



D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies

Document Identification			
Status	Final	Due Date	31/12/2021
Version	1.0	Submission Date	04/03/2022

Related WP	WP3	Document Reference	D3.5
Related Deliverable(s)	D3.1, D3.2, D3.3, D3.4, D4.4, D5.8, D6.6	Dissemination Level (*)	PU
Lead Participant	PSNC	Lead Author	Marcin Lawenda
Contributors	ICCS, BUL, KNOW, USTUTT, PLUS, SZE ECMWF	Reviewers	Tamás Tomaszek (MK)
			Mark Kröll (KNOW)

Keywords:
High Performance Computing (HPC), Big Data, High Performance Data Analytics (HPDA) Benchmarking, Profiling, Scalability, Co-design, Optimization, Visualization, Coupling technologies

This document is issued within the frame and for the purpose of the HiDALGO project. This project has received funding from the European Union's Horizon2020 Framework Programme under Grant Agreement No. 824115. The opinions expressed and arguments employed herein do not necessarily reflect the official views of the European Commission.

This document and its content are the property of the HiDALGO Consortium. All rights relevant to this document are determined by the applicable laws. Access to this document does not grant any right or license on the document or its contents. This document or its contents are not to be used or treated in any manner inconsistent with the rights or interests of the HiDALGO Consortium or the Partners detriment and are not to be disclosed externally without prior written consent from the HiDALGO Partners.

Each HiDALGO Partner may use this document in conformity with the HiDALGO Consortium Grant Agreement provisions.

(*) Dissemination level: **PU**: Public, fully open, e.g. web; **CO**: Confidential, restricted under conditions set out in Model Grant Agreement; **CI**: Classified, **Int** = Internal Working Document, information as referred to in Commission Decision 2001/844/EC.

Document Information

List of Contributors	
Name	Partner
Nikela Papadopoulou, Nikolaos Chalvantzis, Dimitrios Tsoumakos	ICCS
Krzyszimir Samborski, Łukasz Szustak, Piotr Dzierżak	PSNC
László Környei, Ákos Kovács, Mátyás Constans	SZE
Derek Groen, Alireza Jahani	BUL
Leyla Kern, Sergiy Gogolenko	USTUTT
Manuela Rauch	KNOW
Gregor Bankhamer	PLUS
Jesús Ramos Rivas, Francisco Javier Nieto	ATOS

Document History			
Ver.	Date	Change editors	Changes
0.1	15/06/2021	PSNC	Created initial document, table of contents.
0.24	27/07/2021	PSNC	Updated ToC and assigned responsibilities
0.3	19/10/2021	BUL, ICCS, KNOW, PSNC, SZE, USTUTT	First contributions
0.5	06/12/2021	PSNC, PLUS, SZE, KNOW, MOON	Updated content
0.65	14/12/2021	PLUS, PSNC, SZE	Modified document structure and new content in ensemble scenarios, optimization
0.7	22/12/2021	USTUTT, KNOW, PSNC, SZE, BUL, ICCS	Visualization, optimization, coupling, HPDA
0.75	24/12/2021	PLUS, PSNC	SN coupling, data management, data security, optimization (EigHist)
0.79	27/12/2021	ICCS, PLUS	Co-design, SN coupling
0.84	12/01/2022	PSNC, PLUS	Scalability, Optimization
0.94	03/02/2022	SZE, ICCS, PLUS, PSNC	Co-design, scalability, data analytics, coupling technologies

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies			Page:	2 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0
				Status:	Final

Document History			
Ver.	Date	Change editors	Changes
0.95	07/02/2022	PSNC	Version for review
0.98	02/03/2022	ALL	Version with remarks implemented
1.0	04/03/2022	ATOS	FINAL VERSION TO BE SUBMITTED

Quality Control		
Role	Who (Partner short name)	Approval Date
Deliverable leader	Marcin Lawenda (PSNC)	02/03/2022
Quality manager	Marcin Lawenda (PSNC)	02/03/2022
Project Coordinator	Francisco Javier Nieto (ATOS)	04/03/2022

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	3 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

Table of Contents

Document Information.....	2
Table of Contents.....	4
List of Tables.....	9
List of Figures.....	10
List of Acronyms.....	15
Executive Summary.....	17
1 Introduction.....	19
1.1 Purpose of the document.....	19
1.2 Relation to other project work.....	19
1.3 Structure of the document.....	20
2 Co-design.....	21
2.1 Goal.....	21
2.1.1 Systems architecture and characterization.....	22
2.1.2 Co-design analysis and metrics.....	23
2.2 Migration Pilot.....	24
2.2.1 Introduction, goals, and background.....	24
2.2.2 Experimental results.....	25
2.2.3 Overall analysis.....	30
2.3 Urban Air Pollution Pilot.....	30
2.3.1 Introduction, goals and background.....	30
2.3.2 Experimental results.....	31
2.3.3 Overall analysis.....	35
2.4 Social Networks Pilot.....	36
2.4.1 Introduction, goals, and background.....	36
2.4.2 Experimental results.....	37
2.4.3 Overall analysis.....	46
3 HPC benchmarking.....	47

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	4 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

3.1	Migration Pilot	47
3.1.1	Introduction and Goal.....	47
3.1.2	Experimental results.....	48
3.1.3	Overall analysis.....	49
3.2	Urban Air Pollution Pilot.....	49
3.2.1	Introduction and Goal.....	49
3.2.2	Results.....	51
3.2.3	Analysis.....	55
3.3	Social Networks Pilot.....	56
3.3.1	Introduction and Goal.....	56
3.3.2	Experiment 1 (SN-Simulator).....	58
3.3.3	Experiment 2 (SN-Simulator).....	60
3.3.4	Experiment 3 (KPM).....	62
3.3.5	Experiment 4 (KPM).....	65
4	Ensemble scenarios.....	67
4.1	Migration pilot.....	67
4.1.1	Scenario description.....	67
4.1.2	Tests.....	69
4.1.3	Analysis.....	70
4.2	Urban Air Pollution.....	71
4.2.1	Scenario description.....	71
4.2.2	Tests.....	72
4.2.3	Analysis.....	73
4.3	Social Network.....	73
4.3.1	Scenario description.....	74
4.3.2	Tests.....	75
4.3.3	Analysis.....	77
5	Analysis and optimization.....	78
5.1	Migration Pilot.....	78

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	5 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

5.1.1	Goal.....	78
5.1.2	Performance analysis and the Numba-based optimization.....	79
5.2	Urban Air Pollution Pilot.....	82
5.2.1	Goal.....	82
5.2.2	Analysis and conclusions.....	82
5.3	Social Networks Pilot.....	89
5.3.1	Goal.....	89
5.3.2	Social Network Simulator.....	90
5.3.3	The KPM application.....	97
6	HPDA analysis and developments.....	101
6.1	Framework and Execution Environment.....	101
6.2	Migration Pilot.....	101
6.2.1	Simulation Output Statistical Analytics.....	102
6.3	Urban Air Pollution Pilot.....	106
6.3.1	Snapshot Matrix SVD.....	106
6.3.2	Air Quality Index.....	107
6.4	Social Network.....	110
6.4.1	Social Network Analyser.....	110
7	HPDA benchmarking.....	114
7.1	Infrastructure description.....	114
7.2	Migration Pilot.....	115
7.2.1	Simulation Output Statistical Analytics tests.....	115
7.3	Urban Air Pollution Pilot.....	116
7.3.1	Snapshot Matrix SVD tests.....	117
7.3.2	Air Quality Indices tests.....	118
7.4	Social Networks Pilot.....	120
7.4.1	Social Network analyser tests.....	120
7.4.2	Results and findings.....	121
8	GPGPU benchmarking and optimization.....	123

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	6 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

8.1	Infrastructure description.....	123
8.2	Benchmarking of GPGPU cards.....	123
8.2.1	Goal and methodology.....	123
8.2.2	Scalability tests of machine learning models.....	124
8.2.3	Results and findings.....	130
8.3	GPU ML model training optimization.....	131
8.3.1	Goal.....	131
8.3.2	Scenario description.....	131
8.3.3	Tests.....	132
8.3.4	Analysis.....	136
9	Data management.....	138
9.1	CKAN enhancement.....	138
9.2	Benchmarks.....	139
10	Data security.....	148
10.1	Vault Integration.....	148
11	Visualization.....	152
11.1	COVISE.....	152
11.1.1	Extension implementation.....	152
11.1.2	Application & Results.....	153
11.1.3	Benchmarking.....	155
11.1.4	Integration with the Portal.....	157
11.2	Visualizer.....	158
11.2.1	AI method.....	158
11.2.2	Optimization.....	160
11.2.3	Benchmarking.....	161
11.2.4	Dashboard results.....	170
12	Coupling technologies.....	172
12.1	Migration Pilot.....	172
12.2	Urban Air Pollution Pilot.....	173

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	7 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

12.3	Social Networks Pilot.....	175
12.4	Weather data.....	176
12.4.1	Weather data notification system.....	178
12.5	Sensor data.....	180
12.5.1	AIRQ sensory.....	180
12.5.2	Camera Data.....	182
12.6	Telecommunication data.....	183
13	Conclusions.....	187
	References.....	189

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	8 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

List of Tables

Table 1. HiDALGO systems overview	22
Table 2. HiDALGO systems node characterization.....	23
Table 3. Software environment for Flee on the HiDALGO systems.....	25
Table 4. POP co-design metrics for Flee on different systems on a synthetic 10-10-4 graph using 2M initial agents and 10K new agents per time step, for 10 time steps, on 8 nodes. The five metrics are Load Balance Efficiency (LB), Communication Efficiency (CE), Parallel Efficiency (PE), Computational Scaling (CS) and Global Efficiency (GE).....	29
Table 5. Software environment for simpleFoam and pimpleFoam on the HiDALGO systems.....	31
Table 6. POP co-design metrics for pimpleFoam on a mesh of 3.4M points, for 900s in the transient simulation, on 32 nodes. The five metrics are Load Balance Efficiency (LB), Communication Efficiency (CE), Parallel Efficiency (PE), Computational Scaling (CS) and Global Efficiency (GE).....	35
Table 7. Software environment for KPM and SN-simulator on the HiDALGO systems.....	36
Table 8. POP co-design metrics for KPM on different systems on the pokec-5000 graph using 96 intervals, 512 samples and a degree of 62, for 8 nodes. The five metrics are Load Balance Efficiency (LB), Communication Efficiency (CE), Parallel Efficiency (PE), Computational Scaling (CS) and Global Efficiency (GE).....	45
Table 9. POP co-design metrics for SN-simulator on different systems on the NEOS dataset using 400 sources, 1000 samples and 42 as the seed, for 8 nodes. The five metrics are Load Balance Efficiency (LB), Communication Efficiency (CE), Parallel Efficiency (PE), Computational Scaling (CS) and Global Efficiency (GE).....	45
Table 10. Software environment for Flee on the HiDALGO and PRACE systems.....	48
Table 11. Software specification for UAP pilot benchmarks.....	50
Table 12. The number of iterations in simpleFoam and the simulated time in pimpleFoam.....	50
Table 13. The number of iterations in simpleFoam and the simulated time in pimpleFoam.....	51
Table 14. Software stack for Social Networks benchmarks.....	58
Table 15. Software stack for Experiment 4 on Hawk.....	66
Table 16. The average relative difference of the Simulation Approaches for different ensemble sizes.....	70
Table 17: The total execution time of the Simulation Approaches for different ensemble sizes.....	70
Table 18. Runtime used cores and core hours for all ensemble and normal runs for various mesh sizes.....	73
Table 19. Specification of testing platforms.....	78
Table 20. Software stack for SN Simulator and KPM benchmarks.....	79
Table 21. Runtime for Antwerp model of 434k cells for different models on various architectures.....	87
Table 22. Speedup w.r.t. real-time (real-time divided by runtime) for Antwerp model of 434k cells for different models on various architectures.....	88
Table 23. Specification of testing platforms.....	89
Table 24. Software setups for SN Simulator and KPM benchmarks.....	90
Table 25. Agent data file structure and sample values.....	103
Table 26. Location data file structure and sample values.....	103
Table 27. Sample output for Q2 in tabular format.....	106
Table 28. Sample output for Q3 in tabular format.....	106
Table 29. HPDA Spark cluster infrastructure details.....	114
Table 30. HPDA Spark cluster node details.....	115
Table 31. Impact of selectivity on Query Execution.....	120
Table 32. Characteristics of selected dataset graphs.....	120

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	9 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

Table 33. Details of tested machine learning models.....	124
Table 34. Software stack for CKAN benchmark.....	140
Table 35. Route from PSNC to the CKAN.....	140
Table 36. Route from PCz to the CKAN.....	141
Table 37. Route from HLRS to the CKAN.....	141
Table 38 . Endpoints of the Call Detail Record (CDR) API.....	184
Table 39. Parameters Endpoints.....	185
Table 40. Description of the plain CDRs.....	186
Table 41. An example of aggregated CDRs for the social network pilot.....	186

List of Figures

Figure 1. Comparing execution time of Flee on different systems on a synthetic 10-10-4 (left) and 50-50-4 graph (right) using 2M initial agents and 10K new agents per time step, for 10 time steps, for different numbers of nodes.....	25
Figure 2. Comparing speedup w.r.t. 1 node of Flee on different systems on a synthetic 10-10-4 (left) and 50-50-4 graph (right) using 2M initial agents and 10K new agents per time step, for 10 time steps, for different numbers of nodes.....	26
Figure 3. Examining the percentage of execution time spent in MPI functions of Flee on different systems on a synthetic 10-10-4 (left) and 50-50-4 graph (right) using 2M initial agents and 10K new agents per time step, for 10 time steps, for different numbers of nodes.....	27
Figure 4. Examining the percentage of MPI time spent on the various MPI functions of Flee on different systems on a synthetic 10-10-4 (left) and 50-50-4 graph (right) using 2M initial agents and 10K new agents per time step, for 10 time steps, on 8 nodes.....	27
Figure 5. Examining the percentage of total execution time spent on the various functions of Flee on different systems on a synthetic 10-10-4 (up) and 50-50-4 graph (down) using 2M initial agents and 10K new agents per time step, for 10 time steps, on 8 nodes.....	29
Figure 6. Comparing execution times of simpleFoam (left) and pimpleFoam (right) on different systems on a mesh of 3.4M points, for 600 timesteps in the steady simulation and 900s in the transient simulation, for different numbers of nodes.....	32
Figure 7. Comparing speedup w.r.t. 1 node of simpleFoam (left) and pimpleFoam (right) on different systems on a mesh of 3.4M points, for 600 timesteps in the steady simulation and 900s in the transient simulation, for different numbers of nodes.....	32
Figure 8. Examining the percentage of execution time spent in MPI functions of simpleFoam (left) and pimpleFoam (right) on different systems on a mesh of 3.4M points, for 600 timesteps in the steady simulation and 900s in the transient simulation, for different numbers of nodes.....	33
Figure 9. Examining the percentage of MPI time spent on the various MPI functions of simpleFoam (left) and pimpleFoam (right) on different systems on a mesh of 3.4M points, for 600 timesteps in the steady simulation and 900s in the transient simulation, on 32 nodes.....	33

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	10 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

Figure 10. Examining the percentage of total execution time spent on various blocks of code of simpleFoam (left) and pimpleFoam (right) on different systems on a mesh of 3.4M points, for 600 timesteps in the steady simulation and 900s in the transient simulation, on 32 nodes. _____ 34

Figure 11. Comparing execution time of KPM on different systems on the pokec-5000 graph using 96 intervals, 512 samples and a degree of 62, for different numbers of nodes. _____ 37

Figure 12. Comparing execution time of SN-simulator on different systems on the NEOS dataset using 400 sources, 1000 samples and 42 as the seed, for different numbers of nodes. _____ 38

Figure 13. Comparing speedup w.r.t. 1 node of KPM on different systems on the pokec-5000 graph using 96 intervals, 512 samples and a degree of 62, for different numbers of nodes. _____ 39

Figure 14. Comparing speedup w.r.t. 1 node of SN-simulator on different systems on the NEOS dataset using 400 sources, 1000 samples and 42 as the seed, for different numbers of nodes. _____ 39

Figure 15. Examining the percentage of execution time spent in MPI functions of KPM on different systems on the pokec-5000 graph using 96 intervals, 512 samples and a degree of 62, for different numbers of nodes. ___ 40

Figure 16. Examining the percentage of execution time spent in MPI functions of SN-simulator on different systems on the NEOS dataset using 400 sources, 1000 samples and 42 as the seed, for different numbers of nodes. _____ 41

Figure 17. Examining the percentage of MPI time spent on the various MPI functions of KPM on different systems on the pokec-5000 graph using 96 intervals, 512 samples and a degree of 62, for 8 nodes. _____ 42

Figure 18. Examining the percentage of MPI time spent on the various MPI functions of SN-simulator on different systems on the NEOS dataset using 400 sources, 1000 samples and 42 as the seed, for 8 nodes. ____ 42

Figure 19. Examining the percentage of total execution time spent on the various functions of KPM on different systems on the pokec-5000 graph using 96 intervals, 512 samples and a degree of 62, for 8 nodes. _____ 44

Figure 20. Examining the percentage of total execution time spent on the various functions of SN-simulator on different systems on the NEOS dataset using 400 sources, 1000 samples and 42 as the seed, for 8 nodes. ____ 44

Figure 21. Comparing execution time of Flee on different systems on a synthetic 100-100-8 graph using 100M initial agents, for 10 time steps, for different numbers of nodes. Both axes are in logarithmic scale. _____ 49

Figure 22. Execution time vs. number of nodes for steady (left) and transient (right) simulation of the OpenFOAM Air Quality Dispersion Model for small mesh size on different architectures. Both axes use logarithmic scale. _____ 52

Figure 23. Per node Speedup (left) and Parallel efficiency (right) vs. number of nodes for transient simulation of the OpenFOAM Air Quality Dispersion Model for small mesh size on different architectures. Both axes use logarithmic scale. _____ 52

Figure 24. Execution time vs. number of nodes for steady (left) and transient (right) simulation of the OpenFOAM Air Quality Dispersion Model for middle mesh size on different architectures. Both axes use logarithmic scale. _____ 53

Figure 25. Per node Speedup (left) and Parallel efficiency (right) vs. number of nodes for transient simulation of the OpenFOAM Air Quality Dispersion Model for middle mesh size on different architectures. Both axes use logarithmic scale. _____ 53

Figure 26. Execution time vs. number of nodes for steady (left) and transient (right) simulation of the OpenFOAM Air Quality Dispersion Model for large mesh size on different architectures. Both axes use logarithmic scale. _____ 54

Figure 27. Per node Speedup (left) and Parallel efficiency (right) vs. number of nodes for transient simulation of the OpenFOAM Air Quality Dispersion Model for large mesh size on different architectures. Both axes use logarithmic scale. _____ 54

Figure 28. Per core cell count on various architectures and mesh sizes. _____ 56

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	11 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

Figure 29. Comparison of the running times of our new (v0.3) and old (v0.2) SN-Simulator on Hawk and SuperMUC-NG. On the left, we considered the neos data set. On the right, the fpoe data set. 59

Figure 30. Time required by v0.3 of our SN-Simulator for the simulation of the covid19 and fpoe data sets on Hawk, SuperMUC-NG and Altair for an increasing number of compute nodes. 60

Figure 31. Speed-up of v0.3 of our SN-Simulator in comparison with the linear speed-up denoted by “linear” _ 61

Figure 32. Core hour budget required to complete the simulation when supplied with a certain amount of cores. On the x-axis we plot the number of CPU nodes involved in the simulation. On the y-axis the overall time required times the number of cores used. 61

Figure 33. Time required to compute the eigenvalue histogram for an increasing amount of compute nodes on Hawk, SuperMUC-NG and Altair. Comparison of our new version (v0.2) to the old version (v0.1) of our KPM application. 63

Figure 34. Speed-Up of v0.2 of the KPM application. The linear speed-up is denoted by “linear” and stated for comparison. 63

Figure 35. Core hours required to perform complete the eigenvalue histogram computation when supplied a certain amount of cores. Comparison of the new (v0.2) and old (v0.1) version of our KPM application. 64

Figure 36. Weak scaling experiment. Time to compute the eigenvalue histogram with an increasing number of intervals for an increasing amount of compute nodes. Experiments performed on Hawk with KPM v0.3. 66

Figure 37. Multiscale Migration Simulation Workflow Diagram. . Multiscale simulation (yellow boxes) and coupling with Flare (red box) are new models for generating realistic progressions and forecasting how conflicts evolve. 68

Figure 38. Wind profile components: west-east (left) and north-south (right) depending on height for various ensemble scenarios (grey and black) and the normal scenario (red). The minimum and maximum covering curves for ensemble scenarios are also plotted (green). Also, the top of the simulation domain is also shown at 500 meters (purple). 72

Figure 39. Grid search example. The search space is partitioned into a 9x9 grid and at each point a pair of parameters is evaluated. 74

Figure 40. Scalability of the grid search approach for the fpoe dataset on SuperMUC-NG. 76

Figure 41. MAPE of the fpoe data set for each of the 3 experiments. 77

Figure 42. Evaluation of the Numba-based optimization applied for the flee application: a) performance comparison for studied application with enabled and disabled Numba optimizations, b) Partial performance gain measured for a given Kernel separately, and c) the percentage of total execution time for selected measured for the basic version of the application. 81

Figure 43. Performance comparison between RapidCDF and OpenFOAM obtained for different numbers of devices and a variety of domain sizes: a) Computation time [s] (left) and b) Strong scaling speedup (right). ... 84

Figure 44. Runtime for Antwerp model of 434k cells for different models on various architectures. 88

Figure 45. General execution schema for SN Simulator. 91

Figure 46. Performance analysis of different versions of SN Simulator. 92

Figure 47. Workload distributions between MPI workers obtained for different versions of SN Simulator. 93

Figure 48. Performance gain obtained for SN Simulator with a proposed order of tasks execution in comparison to the basic version. 94

Figure 49. Impact of workload (task) size on performance. 95

Figure 50. Performance gain (left) and optimal sample split parameter (right). 96

Figure 51. Performance gain obtained for SN Simulator with enabled NUMBA-based optimizations. 97

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	12 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

Figure 52. Memory trace of the KPM application execution with a) one, b) two, and c) four subgroups of MPI processes per node obtained for the system with 2x Intel Xeon Platinum 8268 configured as four NUMA domains (every sample is averaged over 1~second interval). _____ 98

Figure 53. Performance results obtained for the NUMA aware version of the KPM application in comparison to the basic version. _____ 100

Figure 54. The criteria used for checking validity when aggregating data and calculating statistical parameters according to Directive 2008/50/EC of the European Parliament. _____ 108

Figure 55. Visualization of retweet tree with distances to particular followers _____ 111

Figure 56. Example of a retweet tree _____ 112

Figure 57. Migration Pilot Analytics performance on Spark using varying numbers of executors. _____ 116

Figure 58. SVD performance using varying the input data sizes. _____ 117

Figure 59. SVD performance using varying numbers of spark executors. _____ 118

Figure 60. Air Quality Analytics performance with varying numbers of Spark executors. _____ 119

Figure 61. Performance of social networks analyser on selected datasets. _____ 121

Figure 62. Performance speedup of social networks analyser on selected datasets. _____ 122

Figure 63. Training performance for ResNext neural model on selected architectures. _____ 125

Figure 64. Training performance speedup for ResNext neural model on selected architectures. _____ 126

Figure 65. Training performance for VGG19 neural model on selected architectures. _____ 127

Figure 66. Training performance speedup for VGG19 neural model on selected architectures. _____ 128

Figure 67. Training performance for Mobilenet neural model on selected architectures. _____ 129

Figure 68. Training performance speedup for Mobilenet neural model on selected architectures. _____ 130

Figure 69. Training run without parallelism seen on GPU profiler tool. _____ 133

Figure 70. Training run with data parallelism seen on GPU profiler tool. _____ 134

Figure 71. Training run with distributed data parallelism seen on GPU profiler tool. _____ 134

Figure 72. Training run with DDP and round-robin workload distribution seen on GPU profiler tool. _____ 135

Figure 73. Training run with DDP and distributed sampler seen on GPU profiler tool. _____ 136

Figure 74. Performance of different training strategies. _____ 137

Figure 75. The resource file100M.bin uploaded by GridFTP. _____ 139

Figure 76. Europe map with location of data centres taking a part in testing procedure: Poznań (PSNC), Częstochowa (PCz) and Stuttgart (HLRS). _____ 142

Figure 77. Upload time from PSNC to PSNC. _____ 143

Figure 78. Upload time from PCz to PSNC. _____ 144

Figure 79. Upload time from HLRS to PSNC. _____ 144

Figure 80. CKAN datastore creation time. _____ 145

Figure 81. The results of sequential and in parallel upload data using GridFTP. _____ 146

Figure 82. The results of sequential and in parallel upload data using SCP. _____ 146

Figure 83. The results of sequential and in parallel upload data using CKAN API. _____ 147

Figure 84. A sequence diagram of a user's secondary credentials in the Vault. _____ 151

Figure 85. The module map in COVISE for visualizing UAP simulation data. Coloured boxes indicate: cutting surface (green), iso surface (yellow), streamlines (rose), vector field (blue). _____ 153

Figure 86. Screenshot of the interactive visualization. Iso surface for NOx are shown along with a city model (Stuttgart building scans). _____ 154

Figure 87. The same model from a different perspective: LoD 2 building models and streamlines are added (Stuttgart). _____ 154

Figure 88. Execution time of cold-start task for different mesh resolutions. _____ 156

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	13 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

Figure 89. Processing time per module for different mesh resolutions. 157

Figure 90. Access to COVISE from the HiDALGO Portal. 158

Figure 91. AI-Wizard enabling users to select analytical goals and configure workflows 159

Figure 92. Results from a clustering (left chart) and outlier detection (right chart) method using the AI-Wizard in Visualizer for NEOS twitter data. 160

Figure 93. Average time in seconds for loading a CSV file in Visualizer depending on the number of data rows - blue: initial benchmarking for local files, red: new benchmarking for local files, yellow: new benchmarking for remote files. 162

Figure 94. Average time in seconds for loading a pre-configured dashboard in Visualizer depending on the number of data rows - blue: initial benchmarking for local files, red: new benchmarking for local files. 162

Figure 95. Scatter plot showing sum of sentiments for NEOS (x-axis) and FPÖ (y-axis) for different twitter conversations (colour). 166

Figure 96. Parallel coordinates showing connections between tweet types, conversations and sentiments for NEOS and FPÖ 166

Figure 97. Heat maps showing sum of all sentiments (colour) for different tweet types for each twitter conversation (left charts) and number (colour) of sentiments for each tweet type (right charts) for NEOS (upper charts) and FPÖ (lower charts). 167

Figure 98. Task completion in the user study of Visualizer for eight participants. 168

Figure 99. Task 1 user feedback for task load on a seven-point scale. 169

Figure 100. Task 2 user feedback for task load on a seven-point scale. 170

Figure 101. Dashboard example for the migration use case. 171

Figure 102. Dashboard example for COVID-19 simulation data. 171

Figure 103. Scale Separation Map of Weather Data Coupled Multiscale model. 173

Figure 104. Pre-processing workflow of Urban Air Pollution pilot. 174

Figure 105. The workflow of Twitter Monitor, Cloudify blueprints and Social Network Simulator. 176

Figure 106. Events submitted to Aviso from the ECMWF Dataflow. 179

Figure 107. AIRQ sensory data flow. 180

Figure 108. Location of the Bosch AIRQ sensors. 181

Figure 109. Traffic data flow. 182

Figure 110. Camera locations. 183

Figure 111. Workflow of a query. 185

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	14 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

List of Acronyms

Abbreviation / acronym	Description
ACLED	The Armed Conflict Location & Event Data Project
API	Application Programming Interface
ARH	ARH Informatikai Zartkoruen Mukodo Reszvenytarsasag
BUL	Brunel University London
CAVE	Cave Automatic Virtual Environment
CDS	Climate Data Store
CE	Communication Efficiency
CFD	Computational Fluid Dynamics
CFL	Courant-Friedrichs-Lewy
CKAN	Comprehensive Knowledge Archive Network
CS	Computational Scaling
CSV	Comma-Separated Values
DDM	Distributed Data Management
DMS	Data Management System
Dx.y	Deliverable number y belonging to WP x
EC	European Commission
EMI	Emission file used by Urban Air Pollution
GC	Global Challenges
GE	Global Efficiency
GIL	Global Interpreter Lock (Python)
GPGPU	General-purpose Computing on Graphics Processing Units
HDFS	Hadoop Distributed File System
HLRS	High Performance Computing Center Stuttgart
HPC	High Performance Computing
HPCG	High Performance Conjugate Gradient
HPDA	High Performance Data Analytics
HPL	High Performance LINPACK

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies			Page:	15 of 174		
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status:	Final

HRES	High Resolution Weather Forecast
ICCS	Institute of Communications and Computer Systems
IPC	The Integrated Food Security Phase Classification
KNOW	Know-Center GmbH
LBE	Load Balancing Efficiency
MPI	Message Passing Interface
NUMA	Non-uniform memory access
PCA	Principal Components Analysis
PE	Parallel Efficiency
POD	Proper Orthogonal Decomposition
PSNC	Poznan Supercomputing and Networking Center
SSO	Single sign-on
SVD	Singular Value Decomposition
SZE	Széchenyi István Egyetem University
TCP	Transmission Control Protocol
UAP	Urban Air Pollution
UNHCR	United Nations High Commissioner for Refugees
UUID	Universally Unique Identifier
VM	Virtual Machine
WP	Work Package

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies			Page:	16 of 174		
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status:	Final

Executive Summary

This report summarizes the software development and optimisation activities in the work package WP3. Besides the software implementations, also the final concepts are presented and brought into context with the Benchmarking and Co-Design activity.

For the most part, this report focuses on the evaluation and improvement of the performance of both simulation and data analysis applications as well as data management and visualization solutions. This topic appears throughout most of the chapters, presenting this issue from various perspectives. In general, the method of presentation of the analysis and the obtained results from the point of view of the pilot applications was adopted. This assumption was necessary due to the specificity of implementation solutions used within use cases. Another assumption was to use the existing infrastructure for testing, both offered by project partners and thanks to cooperation with the PRACE project. It should be noted, however, that a distinction was made here within the so-called standard (regular) infrastructure and novelty infrastructure. Only the first one is the subject of this report, while the investigation on the second one is included in the D5.8 report.

The analysis starts with a co-design approach given along with chapter 2. It details a set of metrics oriented on software-software and software-hardware adaptations. This examination performed at smaller scale (one or several nodes) was done on the basis of meaningful execution scenarios (input dimensions), and use a number of performance tools to conduct the relevant performance audit. A broader approach to scalability analysis is presented in the next chapter (chapter 3). This time, the Tier-0 infrastructure was included in the benchmark set (thanks to the cooperation with PRACE). Thanks to optimization techniques implemented throughout the project, it was possible to achieve scalability at a level exceeding 100,000 cores. It was the KPI level established in the Grant Agreement for at least one pilot application. Here it was achieved for two applications from Urban Air Pollution and Social Network pilots. A common approach to the assessment of scalability was related to the so-called ensemble scenarios where the running of multiple instances of simulation applications at the same time was analysed. This is of great importance when, at the modelling stage, the sensitivity and validation of the implemented algorithms are assessed.

The entire chapter (5) of this work is devoted to the presentation of the optimization approach and techniques in various cases. The analysis was conducted from different angles (algorithmic, architectural and library) and provided meaningful indications to improve simulations efficiency. The following two sections/chapters are dedicated to data analytics. The first one concentrates on the presentation of algorithmic approach to implementation methods facilitating data processing in pre- or post-simulation phase of the workflow. The

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	17 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

second one provides feedback on efficiency developed methods by conducting a bunch of benchmarks.

In order to increase the efficiency of the data management system, it was proposed to implement the extension for the CKAN system. It assumes the use of two protocols for data transfer (GridFTP and SCP). The performance of this solution compared to the standard approach offered by CKAN API has been confirmed in a series of tests.

The HiDALGO system is composed of many modules servicing various functionalities. Most often they are deployed on different servers using different frameworks. It is connected with the necessity to harmonize local accounts into a system based on Single Sign On. A description of this solution is based on Keycloak, Vault, Cloudify and Croupier can be found in chapter 9.

The overall aim of visualization solutions is to provide necessary functionality to present data from simulation and subsequent analysis in a way that facilitates their analysis by experts. A number of achievements in the implementation of COVISE and Visualiser extensions are presented. This information was supplemented by test data allowing to assess the overall performance of implemented tools.

The last substantive chapter is about coupling technologies enabling seamless integration of workflow modules. The content description embraces aspects mainly related on facilitating of low-level pilot applications coupling and solutions delivering data from external sources like weather, air quality, camera and telecommunication.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	18 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

1 Introduction

1.1 Purpose of the document

The overriding aim of this document is to provide information on the final achievements of the WP3 package. This work is mainly reported from the perspective of evaluating the performance of implemented applications on various computing infrastructures. Performance presentation is approached from three different perspectives: co-design, scalability and ensemble scenarios. Regardless of performance assessments, the work also focused on code optimization aimed at making the best use of existing software and hardware solutions. In addition to improvements in simulation applications, the report also presents solutions for data analytics that facilitate the mining of data from the simulation process or being their source. In order to ensure the highest possible efficiency of the data source for computational processes, an extension for the CKAN system was proposed to increase data transfer. The unification of access to various computing and service resources was ensured using the SSO methodology based on solutions such as Vault, Keycloak, Cloudify and Croupier. In terms of providing functional visualization tools, a number of extensions have been proposed that meet the requirements reported by the pilots' owners. Additionally, the performance of these tools was assessed in a series of benchmarks. Such a complex system as the HiDALGO environment could not function without numerous technological solutions supporting the connection of many modules with each other. These solutions have been developed to support the internal communication process of pilots as well as to provide data from external sources. The results of the test data presented in this document are supplemented by the raw data published on the GitLab website.

1.2 Relation to other project work

This document is a continuation and summarization of the discussion on HPC, HPDA and visualization applications and their capabilities in WP3. It focuses on development, optimisation, benchmarking and coupling from a technical perspective. It mainly advances the knowledge provided in all previous WP3 reports D3.1 [1], D3.2 [2], D3.3 [3], and D3.4 [4], but it also links to some from other work package documents like:

- D4.4 Final implementation report of the pilot and future applications [5]
- D5.8 Final Benchmark Results for Innovative Architectures [6]
- D6.6 Final Report on Requirements, Components and Workflow Integration [7]

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	19 of 174	
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status:	Final

1.3 Structure of the document

The document is structured into 10 major chapters. Chapter 2 performs detailed analysis to generate the knowledge and metrics for a software-software and software-hardware co-design analysis. Each pilot is examined separately by a number of performance tools to evaluate how best to match pilot applications to the runtime environment.

Chapter 3 focuses on benchmarking to verify the correctness and scalability of updated versions of pilot applications and to extend the scalability analysis to other architectures, made available to the HiDALGO project through the PRACE project.

Ensemble scenarios are tackled in chapter 4. This approach is required to assess testing environment against limitations in running multiple instances of pilot applications in parallel.

Chapter 5 thoroughly analyses pilots' applications and provides indications to improve their efficiency. Applications are examined from different angles depending on the solutions used during development. The proposed solutions are of a different algorithmic nature, using architectural and software solutions.

Chapter 6 and chapter 7 are about data analytics; chapter 6 focuses on presenting their implementation and chapter 7 on testing their performance.

Consecutive chapter 8 details using machine learning algorithms on dedicated hardware accelerators. It is done by benchmarking of GPGPUs using neural models and data distribution strategies for training optimization.

The enhancement of the HiDALGO data management system capabilities is elaborated in chapter 9. It mostly concentrates on delivering new data transfer protocols to the existing regular one offered by the CKAN API system. Applicable tests proving efficiency of proposed solution are performed in various scenarios.

In the next chapter 10 aspects related with security are discussed. The HiDALGO is a multimodule system which requires Keycloak IDM - users accounts system, to enable possibility to log in to all services with support of SSO.

Most recent achievements in visualization are presented in chapter 11. They enlighten advancements in the implementation of COVISE and Visualiser extensions, not forgetting about benchmarking them allowing to assess the overall performance.

The last chapter (12) elaborates on the implementation of coupling technologies required to integrate seamlessly modules of different purposes. These technologies are related to the support of pilot applications and solutions delivering data from external sources like weather, air quality, camera and telecommunication.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	20 of 174	
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status:	Final

2 Co-design

In this section, we perform a detailed performance analysis of the HiDALGO pilots on the available HiDALGO systems. The end target of this analysis is to generate the knowledge and metrics for a software-software and software-hardware co-design analysis. Towards this, we examine each pilot separately, for one or more meaningful execution scenarios (input dimensions), and use a number of performance tools to perform the relevant performance audit.

Data repository folder:

https://gitlab.com/eu_hidalgo/benchmarking/-/tree/master/deliverable_3_5/co-design

2.1 Goal

We focus on 1) examining the scalability of the pilots, 2) breaking down the execution time to identify the most consuming operations/functions, 3) examining the behaviour of communication, and 4) calculating the POP Centre of Excellence [8] metrics for co-design, if possible. We perform this analysis on all available HiDALGO systems, using the same tools, unless otherwise noted in the text. In detail, we use:

- Python profilers for the breakdown of execution time, in the case of Python-based applications. In particular, we use `cProfile` [9], a profiler module in Python, which is a C extension and can provide the execution profile of a program.
- Application-specific timers for the breakdown of execution time, in the case of OpenFOAM-based applications.
- `mpiP` [10] to examine the behaviour of communication. `mpiP` is a lightweight library for the profiling of MPI applications, which collects statistical information about MPI functions.
- `Linux perf` [11] to collect system-level performance counters, which are used for the computation of the POP metrics. The `Linux perf` command is a tool that, among its other capabilities, can collect measurements from the hardware performance counters of a system, during the execution of an application.

We note that, both in the case of profilers and in the case of `perf`, we collect profiles and measurements per MPI process, and aggregate results.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	21 of 174	
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status:	Final

2.1.1 Systems architecture and characterization

In this section, we use four of the available HPC systems within HiDALGO: the Hawk supercomputer at HLRS, a part of the Vulcan cluster at HLRS (clx-25 nodes), the Eagle supercomputer at PSNC, and the new, Altair supercomputer at PSNC. A brief specification of the four systems is presented in Table 1.

	Hawk	Vulcan	Eagle	Altair
Host	HLRS	HLRS	PSNC	PSNC
Number of nodes	5632	96	589	1320
Node type	2 x AMD EPYC 7742 64-core processor @ 2.25 GHz	2 x Intel(R) Xeon(R) Gold 6248 20-core processor @ 2.50GHz	2 x Intel(R) Xeon(R) CPU E5-2697 v3 14-core processor @2.60 GHz	2 x Intel(R) Xeon(R) Platinum 8268 24-core processor @ 2.90GHz
Cores per node	2 x 64	2 x 20	2 x 14	2 x 24
Memory per node	256GB	384GB	64GB	192GB
Interconnect	HDR Infiniband	HDR Infiniband	FDR Infiniband	EDR InfiniBand

Table 1. HiDALGO systems overview

To assist the reader towards a comparison of the applications performance on the different systems, we also present the performance characterization of the nodes of the four different systems, in terms of numbers of cores, peak performance and peak memory bandwidth, in Table 2. We note that Eagle, which hosts the lower-end nodes in all assessed metrics, is the oldest of the four systems. Altair and Vulcan have comparable peak performance, memory bandwidth, and numbers of cores, hosting CPUs of similar generations. Hawk, on the other hand, hosts the larger nodes in terms of cores, peak performance and memory bandwidth.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies			Page:	22 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0
				Status:	Final

	Hawk	Vulcan	Eagle	Altair
Cores	128	40	28	48
Peak performance (GFLOPS/s)	4400	2048	1164	2918
Peak memory bandwidth (GB/s)	230	159	83	148

Table 2. HiDALGO systems node characterization

2.1.2 Co-design analysis and metrics

Our co-design analysis starts from the comparison of the *execution time* and *speedup* with respect to a *single node* of a Pilot, on different numbers of nodes on all available systems. This provides a high-level overview of the scalability of a Pilot and offers a quick answer to the question of which system can offer faster execution for which application.

To further analyse the performance and identify potential room for co-designing the software stack or optimizing a Pilot, we identify the *functions or code blocks that dominate execution time*, using profiling tools.

As all HiDALGO Pilots embrace MPI as the parallel programming model, we perform *lightweight MPI profiling*, to assess the impact of communication as a whole and specific communication operations on execution time. This is an imperative step to co-design parallel applications at larger scales.

Finally, for the Pilots and systems where required metrics can be easily collected, without critical overheads to execution time and further limitations, we perform *a performance audit* similar to the one proposed by the POP Center of Excellence [8]. The POP metrics for co-design [12] attempt to capture parallel efficiency by products of scaling metrics and efficiency metrics, ranging between 0 and 1. We outline briefly the standard POP metrics in the following paragraphs.

The overall quality of a parallel application is measured by its *Global Efficiency (GE)*. The Global Efficiency is a product of the *Parallel Efficiency (PE)* and the *Computational Scaling (CS)*. Parallel Efficiency measures the efficiency of the workload distribution among processes and their communication. It is a product of *Load Balancing Efficiency (LBE)* and *Communication Efficiency (CE)*. The former is a ratio of the average to the maximum computation time and indicates load imbalance, if present. The latter is the ratio of the maximum computation time to the total runtime and indicates if the computation-to-communication balance of the application is good enough. Computational scaling is a product of three metrics, *Instruction*

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies			Page:	23 of 174		
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status:	Final

Scaling (IS), *IPC Scaling (IPCS)* and *Frequency Scaling (FS)*. The three metrics serve as a comparison to a reference case. In our analysis, we always use *single-node execution* as the reference. Instruction scaling compares the number of useful instructions to the reference case. Similarly, IPC scaling and Frequency Scaling compare the instructions per cycle and the frequency against the reference case.

In our analysis, we use `mpiP` and `perf` to collect the necessary metrics to compute the POP metrics. The same analysis can be performed automatically using Scalasca, however the underlying tracing tool of Scalasca [13], `Score-P` [14], does not support some of the features implemented in Python in the HiDALGO applications¹.

2.2 Migration Pilot

2.2.1 Introduction, goals, and background

For the case of the Migration Pilot, we focus on Flee, the core simulation module of the pilot, and in particular, we analyse the performance of the recent Flee 2.0 release [15]. This version encompasses a number of optimizations compared to previous versions of Flee, with respect to optimized memory usage/memory footprint, while the main parallelization scheme is what we refer to as the “advanced” parallelization mode, or else, the agent-space-parallel version, in which case both agents and location updates are distributed among processes, for better scalability.

In this section, we perform a co-design analysis on micro-scale scenarios for Flee, on up to 8 nodes, on all HiDALGO systems. These scenarios are representative of the scalability of the application, and can also reveal performance features of the application for macro-scale scenarios, where, with strong scaling on hundreds of nodes, the equivalent per-node problem size would be similar to a micro-scale scenario.

We examine two micro-scale simulation scenarios, where the initial number of agents in the simulation is 2,000,000 ($N=2M$), with 10,000 new agents added per time step of the simulation ($d=10K$), on two different synthetic input graphs for locations, a smaller graph of 100 vertices ($G=10-10-4$) and a larger graph of 2500 vertices ($G=50-50-4$), with the same connectivity. We perform simulations for 10 time steps ($t=10$). In Table 3 we describe the software environment used for Flee on the HiDALGO systems, for purposes of reproducibility.

¹ `mpi4py.futures` is currently not supported by `ScoreP` and subsequently `Scalasca`, as they are unable to trace MPI inter-communicator creation and related operations, causing profiling of Python applications parallelized with MPI, using futures for concurrency, to hang.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	24 of 174	
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status:	Final

Software	Hawk	Vulcan	Eagle	Altair
Python	3.8.3	3.7.10	3.8.12	3.8.12
Pandas	1.0.5	1.2.5	1.1.5	1.1.5
Numpy	1.19.0	1.20.2	1.20.1	1.20.1
Mpi4py	3.0.3	3.0.3	3.0.3	3.0.3
MPI	MPT 2.23	OpenMPI 4.0.5	OpenMPI 4.1.0	OpenMPI 4.1.0

Table 3. Software environment for Flee on the HiDALGO systems.

2.2.2 Experimental results

2.2.2.1 Execution time and speedup

We first examine the execution time of Flee on 1 up to 8 nodes, on each of the four available HiDALGO systems, for the two execution scenarios. Figure 1 presents the execution time comparison for the four systems. As expected, due to the size of the nodes of the different systems, Eagle results in the higher execution times in both scenarios, and Hawk to the lowest execution time, while Altair and Vulcan demonstrate comparable execution times, especially as the number of nodes increases.

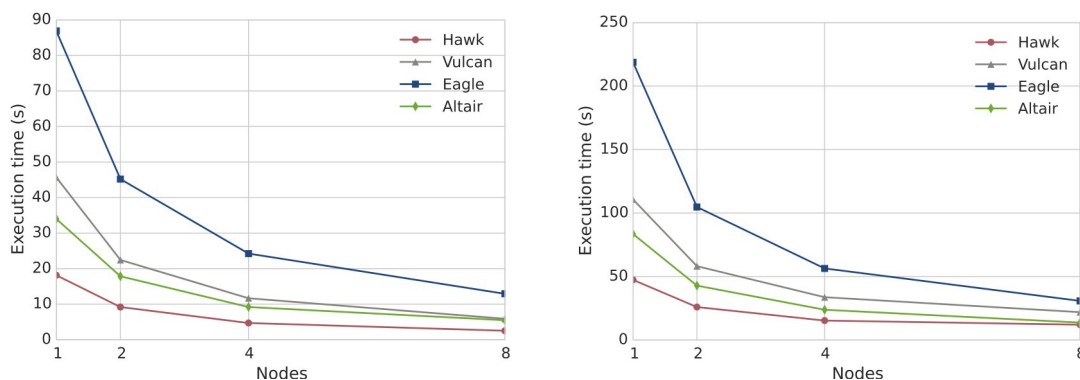


Figure 1. Comparing execution time of Flee on different systems on a synthetic 10-10-4 (left) and 50-50-4 graph (right) using 2M initial agents and 10K new agents per time step, for 10 time steps, for different numbers of nodes.

Figure 2 shows the speedup with respect to a single node, for the same scenarios. We observe that for the smaller 10-10-4 input graph, scalability is very close to linear for all systems as the number of nodes increases, while for the larger 50-50-4 input graph, scalability is lower. Additionally, while Hawk results in the lowest execution time in both scenarios, we observe half the speedup for the larger input graph. This is also true for Vulcan, which we note that it shares the same generation of interconnection network with Hawk.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies	Page:	25 of 174	
Reference:	D3.5	Dissemination:	PU	
	Version:	1.0	Status:	Final

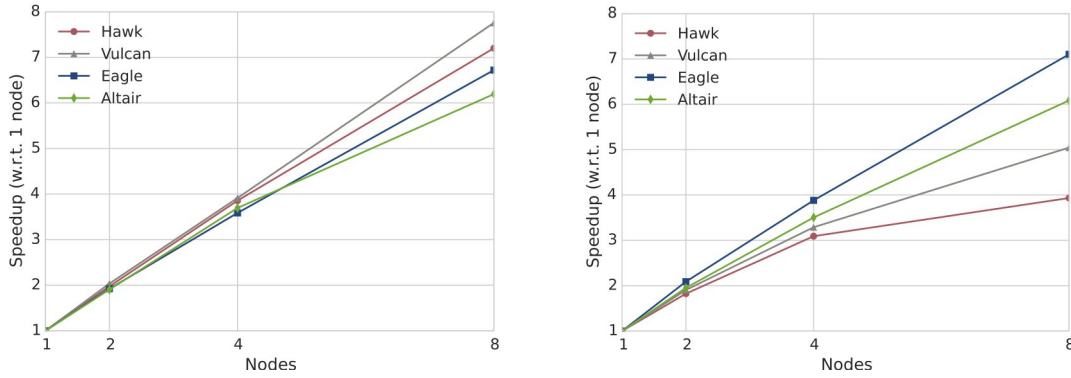


Figure 2. Comparing speedup w.r.t. 1 node of Flee on different systems on a synthetic 10-10-4 (left) and 50-50-4 graph (right) using 2M initial agents and 10K new agents per time step, for 10 time steps, for different numbers of nodes.

2.2.2.2 Communication/computation breakdown

To further examine the scalability behaviour of Flee, we profile the application with `mpiP` and examine the communication behaviour of the application. Figure 3 shows the percentage of execution time spent in MPI functions for the two scenarios, on the four different systems. Overall, with the exception of Hawk, where only a small fraction of total time is spent on MPI functions, for both scenarios, on the remaining three systems, communication time ranges from 10% to 25% of the total execution time, with the single exception of Altair on 4 nodes for the small graph scenario, and of Vulcan on 4 nodes for the large graph scenario, where we observe a peak in communication time. We can therefore assume that the decreased scalability of the application on the larger graph on three out of the four systems does not owe to an increase in communication time, but to other factors.

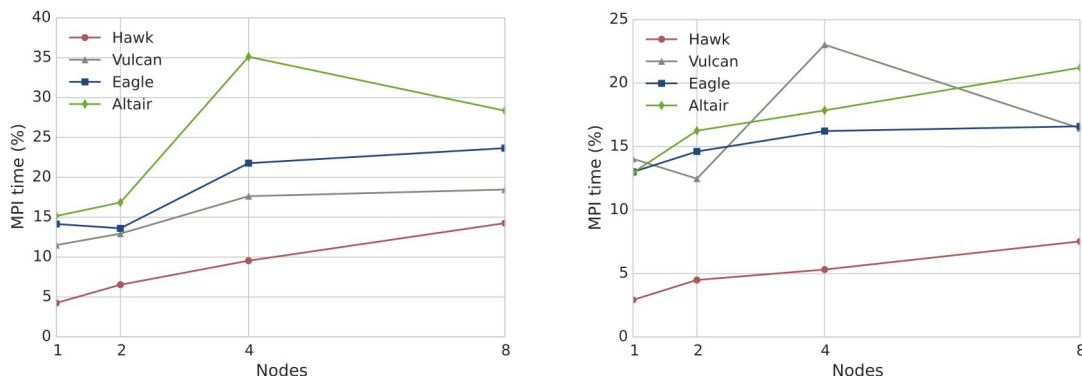


Figure 3. Examining the percentage of execution time spent in MPI functions of Flee on different systems on a synthetic 10-10-4 (left) and 50-50-4 graph (right) using 2M initial agents and 10K new agents per time step, for 10 time steps, for different numbers of nodes.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies			Page:	26 of 174	
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

We additionally break down MPI time in different MPI functions. The results are demonstrated in Figure 4, for 8 nodes, where we observe that only two communication functions appear in Flee, `Allreduce` and `Allgather`. In both scenarios, `Allreduce` dominates communication time. This owes to the function being called more often within the application, compared to the calls to the `Allgather` function. On Hawk and Altair, for the scenario on the largest graph, where the calls to `Allgather` involve larger message sizes, the percentage of time spent on both functions is more comparable, hence the `Allreduce` operation is more optimized on those two systems, therefore they may be preferable for large-scale runs of Flee.

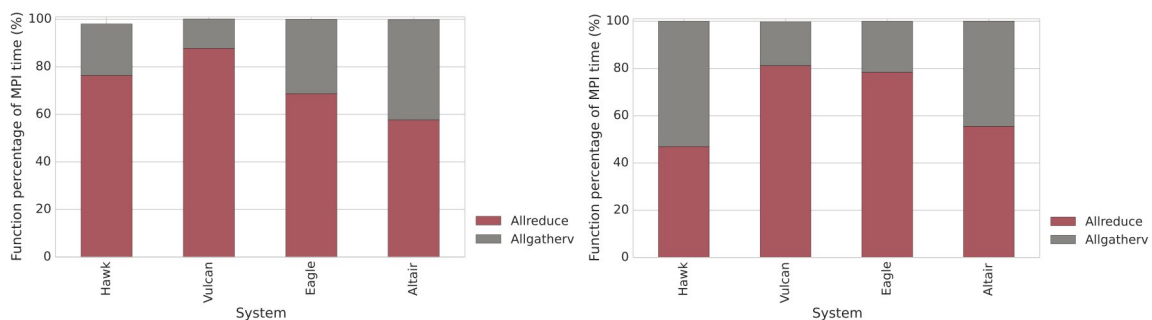


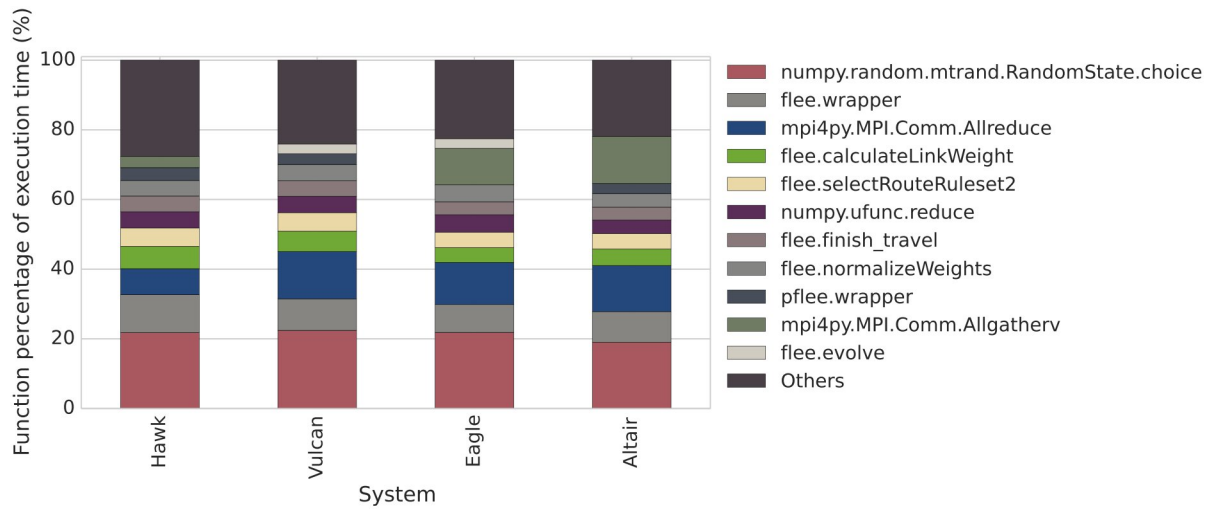
Figure 4. Examining the percentage of MPI time spent on the various MPI functions of Flee on different systems on a synthetic 10-10-4 (left) and 50-50-4 graph (right) using 2M initial agents and 10K new agents per time step, for 10 time steps, on 8 nodes.

2.2.2.3 Execution time breakdown

To understand performance bottlenecks that are not directly related to communication, we profile Flee using the `cProfile` [16] module in Python, and present the results for the 10 more time-consuming functions, on 8 nodes, on all systems, for the two scenarios, in Figure 5. We observe that one of the most time consuming functions is `numpy.random.mtrand.RandomState.choice`, a function that is called multiple times within the simulation and consumes about 20% of the total execution time in the small graph scenario and 10-20% of the total execution time in the large graph scenario, depending on the system. Additionally, some portion of the total execution time is spent on `flee.wrapper` and `pflee.wrapper`, which ensure the correctness of the simulation. However, these overheads can be avoided in large-scale production runs, where the instantiation of the simulation can be debugged in smaller scales, thus enhancing the performance of Flee. A non-negligible fraction of time is spent on `numpy.ufunc.reduce`, which performs a reduction on `numpy` matrices in the simulation. This corresponds to the more computation-intensive parts of the Flee simulation. What is noteworthy is that on Hawk,

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies			Page:	27 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0
				Status:	Final

in the large graph scenario, a significant amount of time is spent on the `flee.linkUp` function, which is a function called more often when the input graph is larger. This is a function of low operational intensity, as it appends elements in lists. Therefore, as more functions benefit from the node architecture of Hawk, the effect of this memory-bound function becomes more evident on this system.



Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies			Page:	28 of 174		
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status:	Final

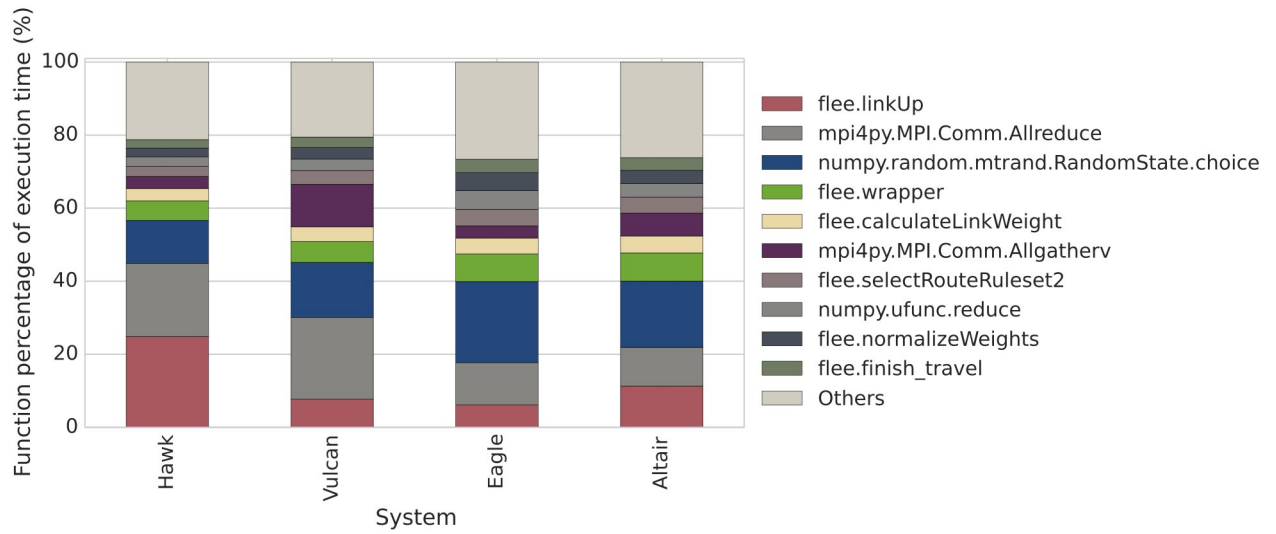


Figure 5. Examining the percentage of total execution time spent on the various functions of Flee on different systems on a synthetic 10-10-4 (up) and 50-50-4 graph (down) using 2M initial agents and 10K new agents per time step, for 10 time steps, on 8 nodes.

2.2.2.4 POP CoE Co-Design Metrics

To conclude our analysis, we compute the POP CoE co-design metrics for the two scenarios of Flee on the four systems. Table 4 presents the POP metrics for Flee on 8 nodes of all systems, using the smaller graph scenario.

System	LB	CE	PE	CS	GE
Hawk	91.44%	78.95%	72.20%	100.00%	72.20%
Vulcan	86.88%	71.47%	62.09%	100.00%	62.09%
Eagle	82.77%	69.84%	57.81%	94.46%	54.60%
Altair	79.08%	65.51%	51.80%	91.66%	47.48%

Table 4. POP co-design metrics for Flee on different systems on a synthetic 10-10-4 graph using 2M initial agents and 10K new agents per time step, for 10 time steps, on 8 nodes. The five metrics are Load Balance Efficiency (LB), Communication Efficiency (CE), Parallel Efficiency (PE), Computational Scaling (CS) and Global Efficiency (GE).

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies			Page:	29 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0
				Status:	Final

We observe that Flee demonstrates good load balancing on all systems, with slightly better scores on Hawk. Load Balance (LB) Efficiency is better when the simulation is run on more cores. Communication Efficiency (CE) ranges from ~65 - 80% on the different systems, to verify that, to amortize the cost of the collective communication functions of Flee, one should increase the useful computation per process. Parallel Efficiency (PE) ranges from 50 to 70% and is dominated by the limited Communication Efficiency. We observe lower values for Computation Scaling (CS), for Eagle and Altair. Note that Computation Scaling is computed with the single-node execution as the reference case. In both cases, the inefficiency in computation stems from an increase in the number of instructions, and not from interference, as indicated by measurements collected with `perf`. What increases is the number of instructions required to perform a simulation. Regarding overall efficiency, as indicated by Global Efficiency (GE), Parallel Efficiency drives Global Efficiency down for Eagle and Altair, however, Global Efficiency also indicates that Hawk and Vulcan, with their current software stacks, are better matches for the Flee application.

2.2.3 Overall analysis

Our co-design analysis demonstrates that Flee performs better on newer-generation systems, like Hawk and Altair. In terms of communication, Hawk appears to be a better match for Flee, due to lower `Allreduce` times. There is room for optimization in the software stack for Flee, mainly in the `numpy` library. Additional optimizations can be performed in memory-bound functions of Flee, which will lead to better scalability on systems like Hawk. Overall, we find that Flee is able to take advantage of high-performance nodes with multiple cores, as our results have not indicated any performance loss from intra-node effects, like cache sharing or limited memory bandwidth.

2.3 Urban Air Pollution Pilot

2.3.1 Introduction, goals and background

For the case of the Urban Air Pollution Pilot, we focus on the M32 (v2107), 2021 July version of the CFD module of the UAP Pilot, and in particular on the optimized version of the steady and transient simulation, using the `simpleFoam` and `pimpleFoam` function of OpenFOAM [17] respectively.

In this section, we perform a co-design analysis on a single scenario for both the steady and the transient simulation with OpenFOAM, on up to 8 nodes, on all HiDALGO systems. The scenario is representative of the two critical most time-consuming part parts of the simulation

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	30 of 174	
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status:	Final

(other functions in the simulation correspond to pre- and post-processing of simulation input/output) and of the scalability of the application, as the distribution of a medium-sized mesh, on fewer nodes, demonstrates similar computational requirements with that of the distribution of a larger mesh on larger node counts, and also allows us to collect performance metrics in reasonable time. Due to the complexity of the OpenFOAM software, we do not use the tools defined in the generic HiDALGO co-design approach, but rely on the reporting APIs offered by the OpenFOAM software, which allow the user to add hooks to measure the execution time of specific code blocks, to profile the time spent on different code blocks. We use `mpiP` and `perf` to analyse communication and calculate the POP CoE metrics and also report the time spent on critical communication functions.

We examine a scenario where the input is a generated 3D mesh of 3,400,00 *points*, with wind profiles obtained from ECMWF weather data on the boundary and emission from traffic simulation based on loop data, for 600 *iterations* for the steady simulation and a *simulated time of 900s* for the transient simulation.

In Table 5, we describe the software environment used for `simpleFoam` and `pimpleFoam` on the HiDALGO systems, for purposes of reproducibility.

Software	Hawk	Vulcan	Eagle	Altair
C compiler	gcc 9.2.0	gcc 10.3.0	gcc 10.2	gcc 10.2
MPI	HPE MPT 2.23	OpenMPI 4.1.1	OpenMPI 4.1.0	OpenMPI 4.1.0
scotch	6.0.9	6.0.9	6.0.9	6.0.9
OpenFOAM	v.20 12	v 20 12	v 20.12	v 20.12

Table 5. Software environment for `simpleFoam` and `pimpleFoam` on the HiDALGO systems.

2.3.2 Experimental results

2.3.2.1 Execution time and speedup

We first examine the execution time of `simpleFoam` and `pimpleFoam` on 8 up to 32 nodes, on each of the four available HiDALGO systems. Figure 6 shows the execution time comparison for the four systems. We observe the lowest execution times on Hawk, and the highest execution time on Eagle, as expected, since this is the oldest among the four systems. At the same time, however, we observe the scalability of both functions breaking on Hawk at 16 nodes. This is also true for `simpleFoam` on Altair, while `pimpleFoam` continues to scale on 32 nodes on both Vulcan and Altair.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	31 of 174	
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status:	Final

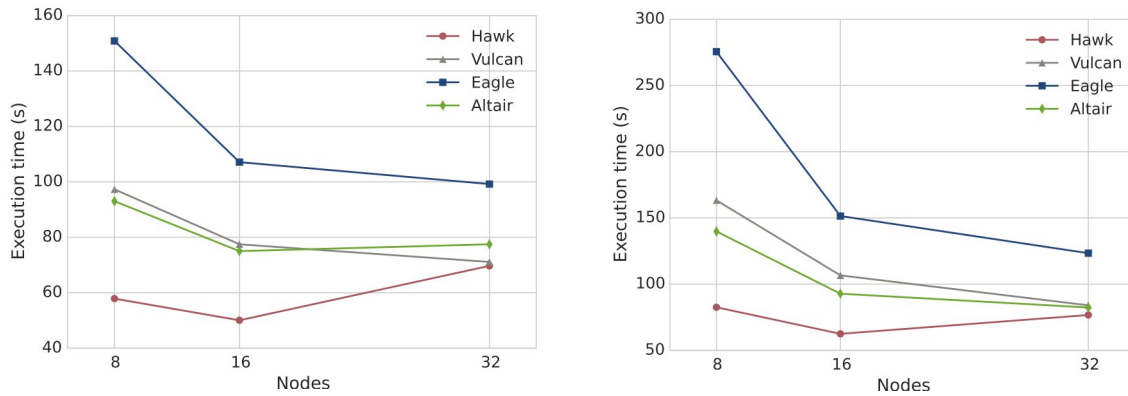


Figure 6. Comparing execution times of simpleFoam (left) and pimpleFoam (right) on different systems on a mesh of 3.4M points, for 600 timesteps in the steady simulation and 900s in the transient simulation, for different numbers of nodes.

We next inspect the speedup of `simpleFoam` and `pimpleFoam`, with respect to a single node, in Figure 7. We indeed observe a very low speedup on Hawk, medium and similar speedups for Vulcan and Altair, and almost linear speedup for Eagle, which is, however, attributed to the high execution time on a single node, of Eagle, which is almost double that of the execution time of other systems.

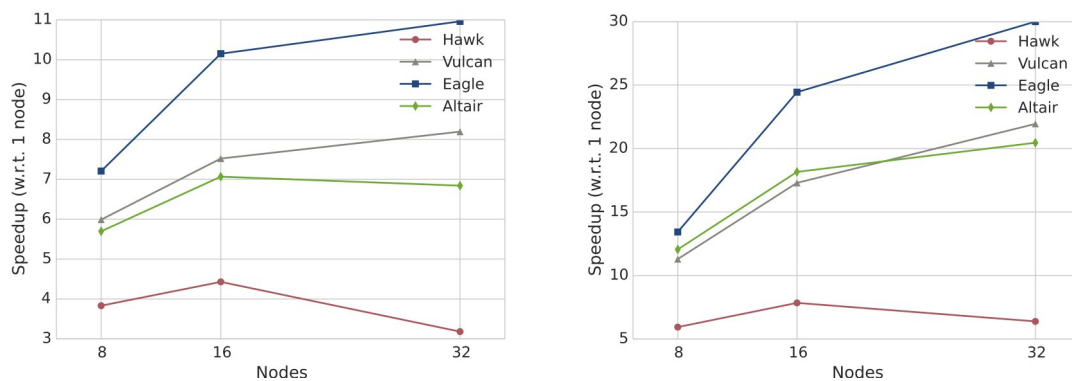


Figure 7. Comparing speedup w.r.t. 1 node of simpleFoam (left) and pimpleFoam (right) on different systems on a mesh of 3.4M points, for 600 timesteps in the steady simulation and 900s in the transient simulation, for different numbers of nodes.

2.3.2.2 Communication/computation breakdown

To examine the reasons behind the scalability breaks, we analyse the communication of the application, using `mpiP`. We omit the analysis on Eagle, due to system storage unavailability, which hindered the collection of profiling results. Figure 8 shows the percentage of execution time spent in MPI functions for the two functions, on the three different systems, Hawk, Vulcan, and Altair. Overall, for both functions in all cases, we observe an extremely high percentage of execution time being spent on MPI, over 80% for the case of `simpleFoam`

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies	Page:	32 of 174
Reference:	D3.5	Dissemination:	PU
	Version:	1.0	Status: Final

2.3.2.3 Execution time breakdown

For a further comparison of function performance between systems, we collect the execution time breakdown into blocks of code, annotated with the help of OpenFOAM APIs for timing/profiling. Figure 10 presents this breakdown for the two functions on 32 nodes, across all four systems. For the case of `simpleFoam`, we observe only slight differences in the percentage of execution time, mainly for *Initialization*, and *the solution of the pressure equation*. Initialization consumes a smaller percentage of the total execution time on Vulcan and Eagle. Contrarily, solution of the pressure equation consumes a smaller percentage of execution time on Hawk and Altair. For the case of `pimpleFoam`, similarly, *Initialization* is the one that consumes a larger percentage of execution time on Hawk and Altair, while *solution of the pressure equation* appears to consume a larger percentage of execution time on Vulcan. Pinpointing the code blocks that are the main contributors to execution time can help tune the respective parts of the software stack, if possible, to optimize their execution.

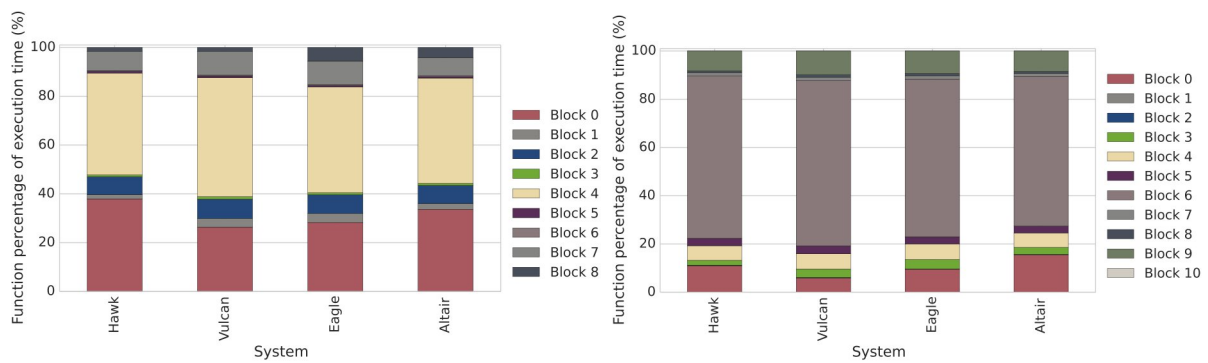


Figure 10. Examining the percentage of total execution time spent on various blocks of code of `simpleFoam` (left) and `pimpleFoam` (right) on different systems on a mesh of 3.4M points, for 600 timesteps in the steady simulation and 900s in the transient simulation, on 32 nodes.

2.3.2.4 POP CoE Co-Design Metrics

To conclude our analysis, we compute the POP CoE co-design metrics for `pimpleFoam` on Hawk, Vulcan and Altair, excluding Eagle, due to storage outages that hindered the collection of profiling information. We focus on this function, as the unsteady simulation performed through it is the most significant part of the Pilot. Table 6 presents the POP metrics for `pimpleFoam` on 32 nodes on the three systems.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	34 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

System	LB	CE	PE	CS	GE
Hawk	30.51%	20.18%	6.16%	84.96%	5.23%
Vulcan	40.54%	37.44%	15.18%	169.93%	25.79%
Altair	40.19%	30.74%	12.36%	197.81%	24.44%

Table 6. POP co-design metrics for pimpleFoam on a mesh of 3.4M points, for 900s in the transient simulation, on 32 nodes. The five metrics are Load Balance Efficiency (LB), Communication Efficiency (CE), Parallel Efficiency (PE), Computational Scaling (CS) and Global Efficiency (GE).

We observe that on all three systems, the function demonstrates very poor Load Balancing (LB) Efficiency and very poor Communication Efficiency (CE), with this being particularly true for Hawk, where we have also observed the worst speedup on 32 nodes. In practice, this is mainly due to the way the original mesh is decomposed, and much of the load imbalance of the application comes from idle time due to communication imbalance, and communication inefficiency, as also indicated by the very low number of the Communication Efficiency metric, and as we have already observed the previous subsections. The two metrics result in a very poor Parallel Efficiency (PE) for the function for the particular mesh, at this node count. Contrarily, Computational Scaling (CS) is very high and exceeds 100%. This is due to Instruction Scaling (IS): we have observed that the number of executed instructions decreases rapidly as one adds more processes, indicating that the useful work per process is probably very low for the particular mesh at this node count. However, this also indicates that the particular function is able to handle more work per process, if one uses a larger problem size. The overall Global Efficiency (GE) is very poor for Hawk, where we also observe the worst speedup. We believe that the mesh we use is too small to be decomposed on the 32 nodes of Hawk, which amounts roughly to 3-4 times more cores than the 32 nodes of the two other systems.

2.3.3 Overall analysis

Our limited co-design analysis for this application shows that, while it can benefit from the fat nodes of Hawk, with the multitude of threads, communication overheads, along with other possible factors, related to the mesh decomposition and communication imbalance, impede its scalability on higher core counts, i.e. on high numbers of nodes, for the mesh size examined. Mid-sized nodes, as are those on Vulcan and Altair, can also offer a good performance both to the `simpleFoam` and to the `pimpleFoam` functions. Further analysis

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies			Page:	35 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0
				Status:	Final

is required, targeting the relation of the mesh decomposition on the performance of the application.

2.4 Social Networks Pilot

2.4.1 Introduction, goals, and background

For the case of the Social Network Pilot, we focus on two applications, 1) eigenvalue calculator (called “KPM”), which computes the approximate spectrum of eigenvalues of a graph, and 2) SN-simulator, which simulates the spread of messages throughout a social networks graph. For KPM, we analyse the performance of KPM v0.2 [18], and for SN-simulator, we analyse the performance of SN-simulator v0.3 [19].

In this section, we perform a co-design analysis on representative inputs for the two applications, on up to 8 nodes of three of the four HiDALGO systems, i.e. Hawk, Vulcan, and Altair. We omit the analysis on Eagle, since the performance measurements have taken place at a time of transition from Eagle to Altair, where Eagle-only node allocations on the system were not available.

For KPM, we examine a scenario where the input graph is a subgraph of the friendship-relationship graph of the `pokec` social network from the SNAP collection [20], of 50,000 nodes from the initial 1,000,000 nodes of the `pokec` graph. The number of samples equals to 512 ($s=512$), the number of intervals equals to 96 ($i=96$) and the degree equals to 62 ($d=62$). For SN-simulator, we examine a scenario where the input data is the *Neos* dataset, of medium size, consisting of users tweeting about the NEOS political party during the Austrian political elections in 2019. The number of sources equals to 400 ($so=400$), the number of samples equals to 1000 ($sa=1000$) and the random seed for the simulation equals to 42 ($se=42$). In Table 7, we describe the software environment used for the two applications on the HiDALGO systems, for purposes of reproducibility.

Software	Hawk	Vulcan	Altair
Python	3.8.3	3.7.10	3.8.12
Pandas	1.0.5	1.2.5	1.1.5
Numpy	1.19.0	1.20.2	1.20.1
SciPy	1.5.0	1.6.2	1.7.1
Numba	0.54.0	0.53.1	0.54.1

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	36 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

Mpi4py	3.0.3	3.0.3	3.0.3
MPI	MPT 2.23	OpenMPI 4.0.5	OpenMPI 4.1.0

Table 7. Software environment for KPM and SN-simulator on the HiDALGO systems.

2.4.2 Experimental results

2.4.2.1 Execution time and speedup

We first examine the execution time of KPM and SN-simulator on 1 up to 8 nodes, on each of the three selected HiDALGO systems. Figure 11 and Figure 12 present the execution time comparison for the two applications on the three systems. We observe that, for both applications, the lowest execution time is observed on Altair. However, for KPM, on 8 nodes, all systems appear to demonstrate comparable execution times, while for SN-simulator, Altair and Vulcan show a similar scaling behaviour, while Hawk remains slower.

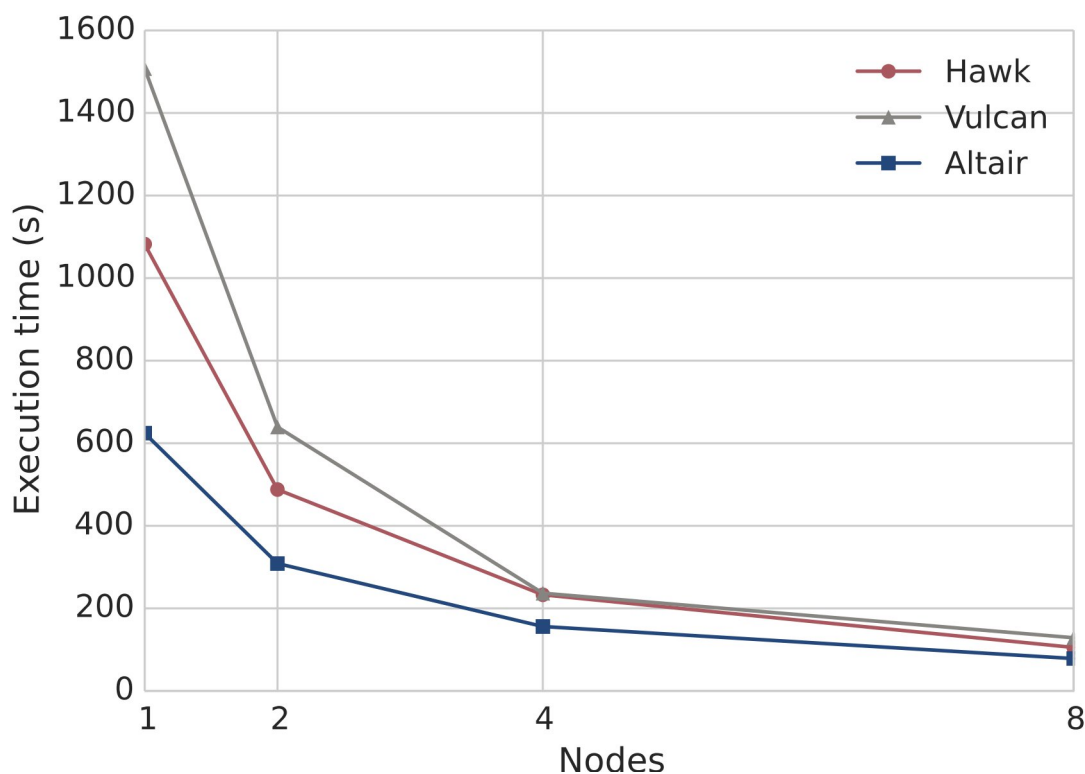


Figure 11. Comparing execution time of KPM on different systems on the pokec-5000 graph using 96 intervals, 512 samples and a degree of 62, for different numbers of nodes.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies	Page:	37 of 174				
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status:	Final

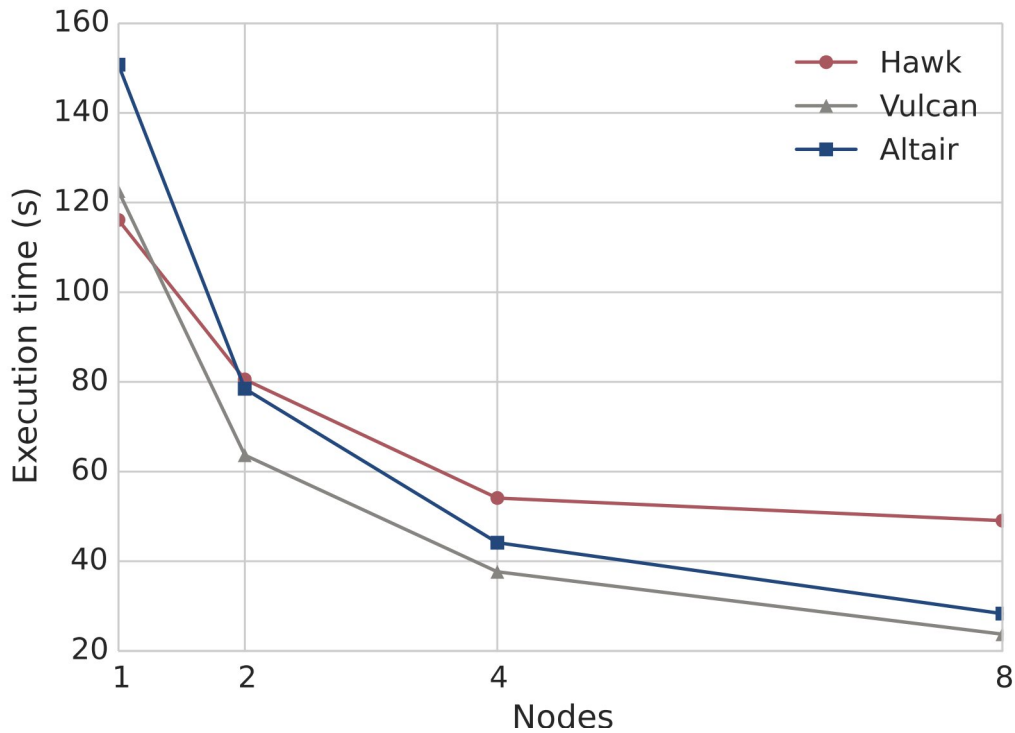


Figure 12. Comparing execution time of SN-simulator on different systems on the NEOS dataset using 400 sources, 1000 samples and 42 as the seed, for different numbers of nodes.

Figure 13 and Figure 14 show the speedup with respect to a single node, for the two applications. We observe that, for KPM, both Vulcan and Hawk, demonstrate a super-linear speedup, compared to Altair where speedup is very close to linear. Contrary, for SN-simulator, the observed speedup is much lower on both systems, with a very low speedup being observed on Hawk, where adding more than 2 nodes does not have any effect on the execution time of the simulation. Therefore, we make the following three key observations: 1) the two applications have very different scalability behaviour, 2) the phenomenon of super-linear speedup for KPM needs to be observed and interpreted with caution, and 3) SN-simulator is less scalable, appearing to be incapable of taking advantage of the multiple available cores of Hawk.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	38 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

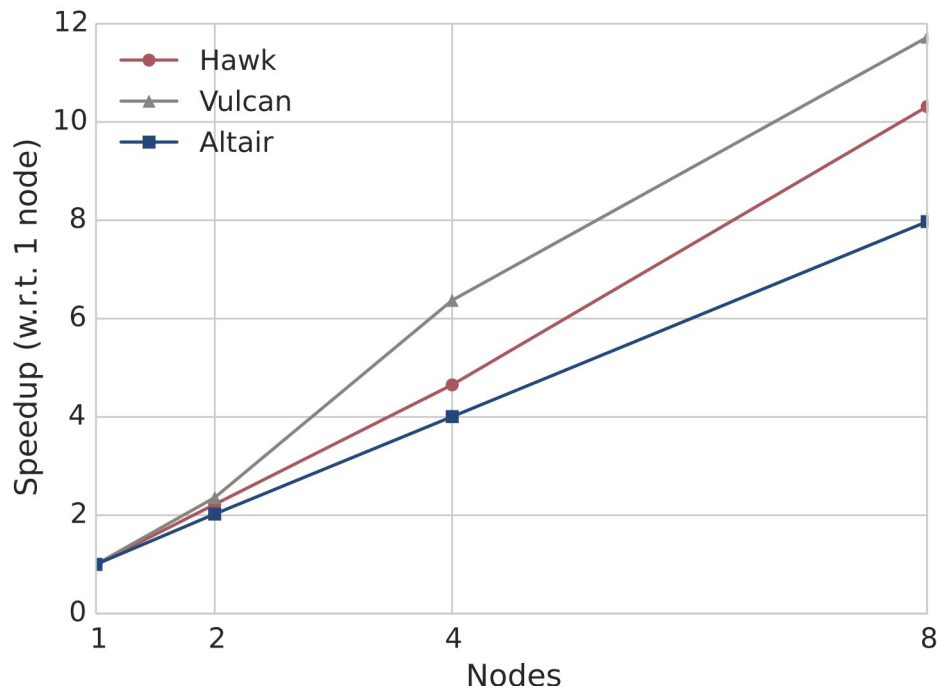


Figure 13. Comparing speedup w.r.t. 1 node of KPM on different systems on the pokec-5000 graph using 96 intervals, 512 samples and a degree of 62, for different numbers of nodes.

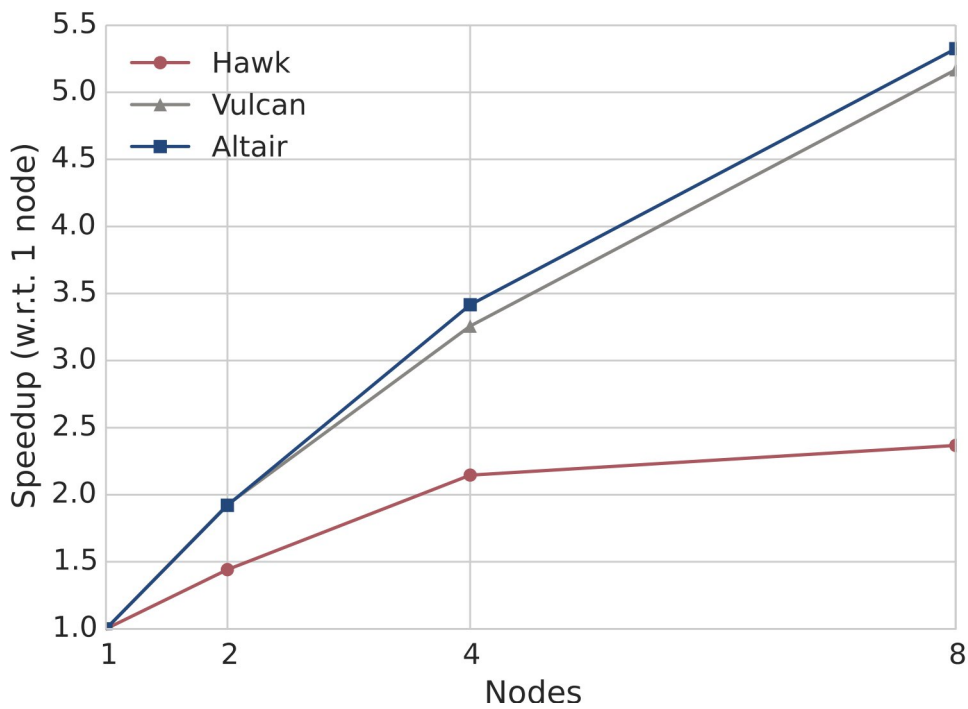


Figure 14. Comparing speedup w.r.t. 1 node of SN-simulator on different systems on the NEOS dataset using 400 sources, 1000 samples and 42 as the seed, for different numbers of nodes.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies	Page:	39 of 174	
Reference:	D3.5	Dissemination:	PU	
	Version:	1.0	Status:	Final

2.4.2.2 Communication/computation breakdown

To examine the scalability of the two applications of the Social Network Pilot, we profile both applications with `mpiP` and examine their communication behaviour. Figure 15 and Figure 16 show the percentage of execution time spent in MPI functions for the two applications, on the three different systems. As with their scalability, we observe very different percentages of MPI time for the two applications. For KPM, in Figure 15, we observe a high MPI time for a single node, both for Hawk and Vulcan, which decreases as the number of nodes increases, while we observe near-zero MPI times on Altair. The decreasing MPI time on Hawk and Vulcan are a potential source of the super-linear speedup we observe. Contrarily, we observe very low percentages of MPI time for the SN-simulator in Figure 16. This percentage only increases with the number of nodes on Vulcan, but only to a small percentage of up to 10%, hence not inhibiting the scalability of the application on this system.

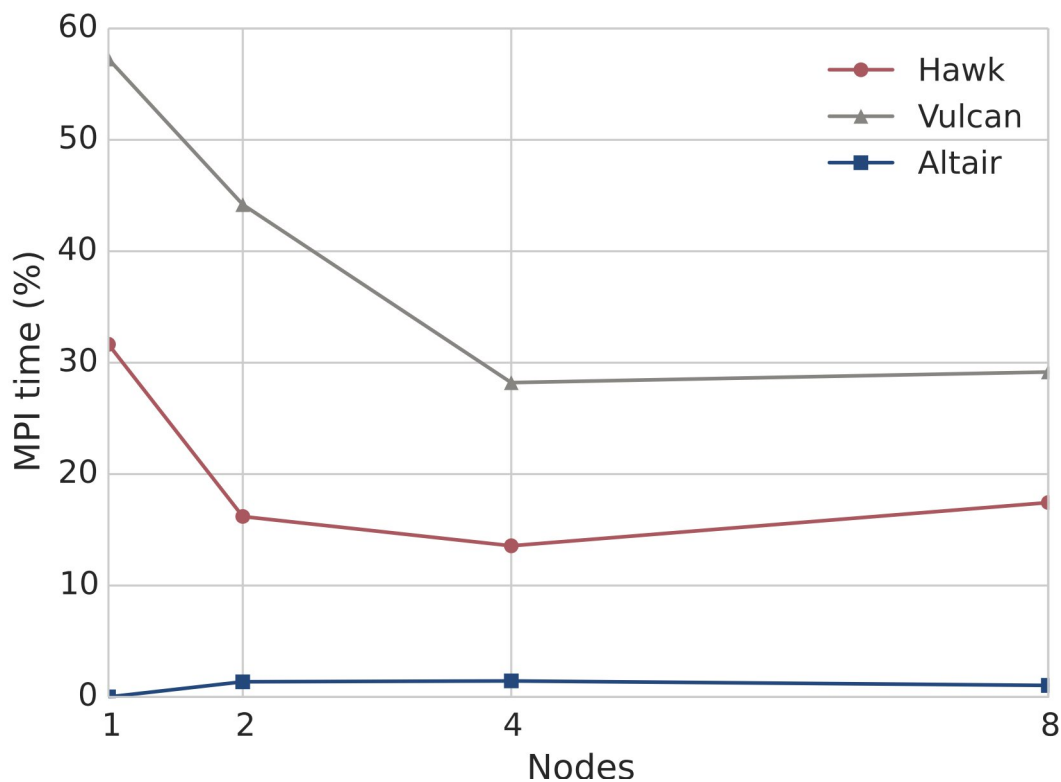


Figure 15. Examining the percentage of execution time spent in MPI functions of KPM on different systems on the pokec-5000 graph using 96 intervals, 512 samples and a degree of 62, for different numbers of nodes.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	40 of 174	
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status:	Final

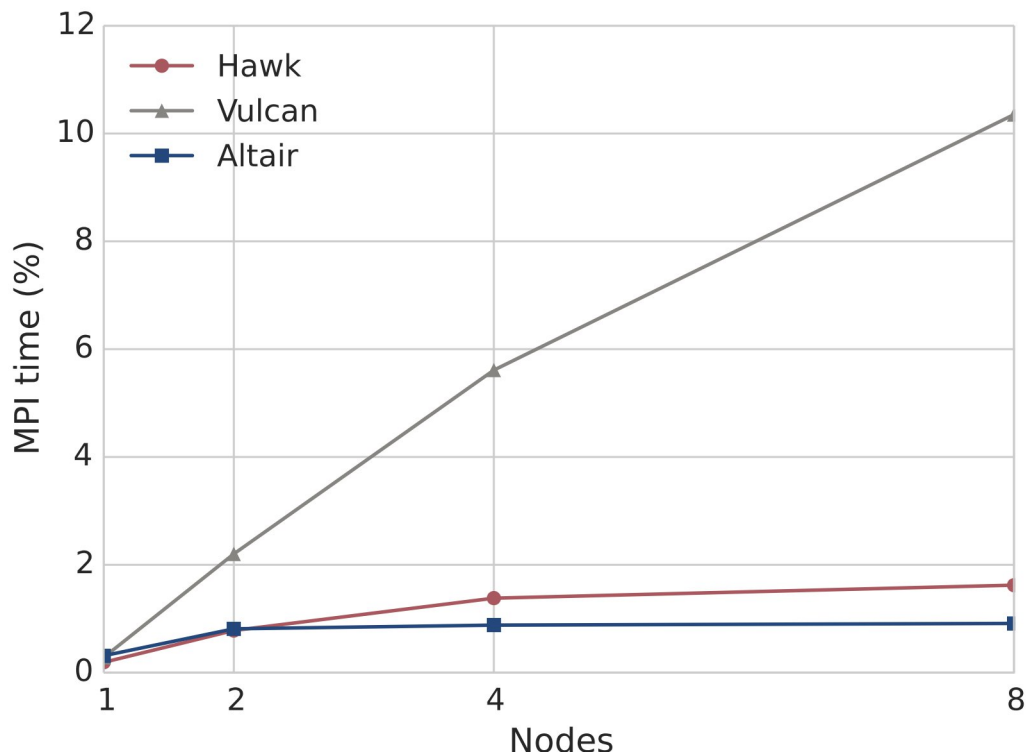


Figure 16. Examining the percentage of execution time spent in MPI functions of SN-simulator on different systems on the NEOS dataset using 400 sources, 1000 samples and 42 as the seed, for different numbers of nodes.

We additionally break down MPI time in different MPI functions, demonstrating the results in Figure 17 and Figure 18, for 8 nodes. For the case of the KPM application in Figure 17, we observe that, on Hawk, and Vulcan, almost all MPI time is spent on the function `MPI_Comm_dup`, which is not called anywhere within the application, and therefore is called internally by the `mpi4py` library. Contrarily, on Altair, the percentage of this particular function is insignificant, although an accountable percentage of time is also spent on the `MPI_Comm_group` and `MPI_Comm_split` functions, also called internally by the `mpi4py` library. All these functions are used to internally optimize collective operations on multi-core systems; therefore, we suspect that, what is identified by the profiler as a call to the `MPI_Comm_dup` function is the outcome of a lazy evaluation in Python of the equivalent broadcast and one-sided operations performed by the application, on Hawk and Vulcan. We can, however, safely assume that the communication of this application is more optimized with the current software stack on Altair, leading to a lower execution time, and the respective parts of the software stack need to be tuned both on Hawk and Vulcan, to see the exact

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	41 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

behaviour of the communication of the application on the two systems, and the room for performance gains.

Looking at the case of the SN-simulator, in Figure 18, we note that, although MPI time appears to be insignificant compared to total execution time, most of it is spend on `Barrier` and `Bcast` functions in MPI, with the `Bcast` function consuming most of the MPI time on Vulcan, which is, most probably, the reason for the increase in MPI time on this system.

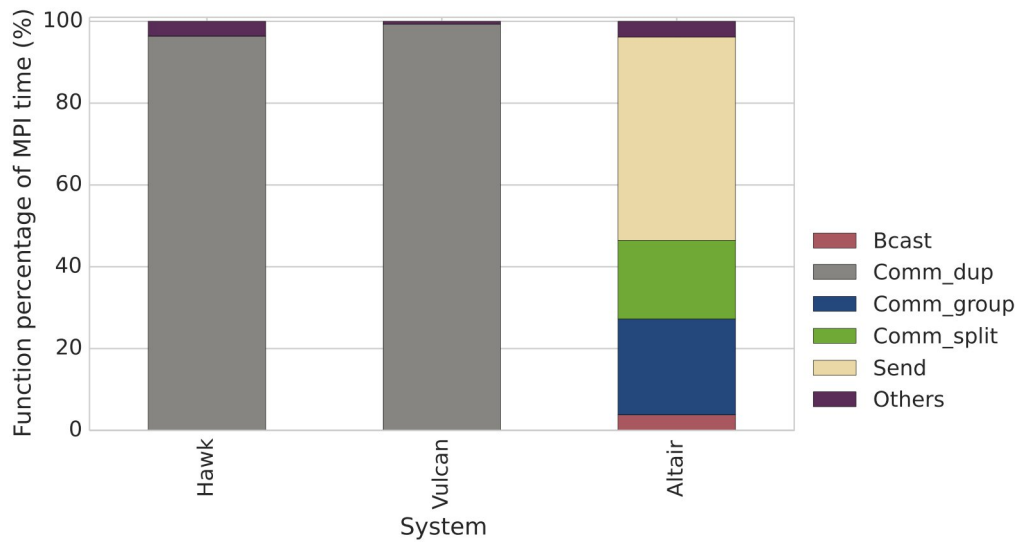


Figure 17. Examining the percentage of MPI time spent on the various MPI functions of KPM on different systems on the pokec-5000 graph using 96 intervals, 512 samples and a degree of 62, for 8 nodes.

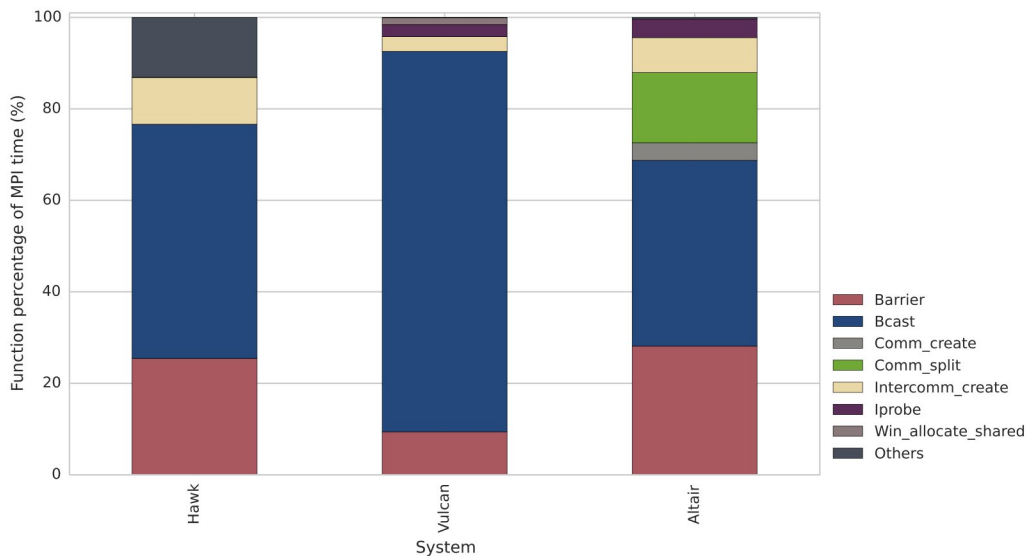


Figure 18. Examining the percentage of MPI time spent on the various MPI functions of SN-simulator on different systems on the NEOS dataset using 400 sources, 1000 samples and 42 as the seed, for 8 nodes.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies			Page:	42 of 174	
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

We note, however, that, for the case of SN-simulator, communication does not contribute significantly on MPI time, and does not explain the reduced performance and scalability of this application on Hawk.

2.4.2.3 Execution time breakdown

To further study performance bottlenecks, we profile the two applications using the cProfile module in Python, and present the results for the most time-consuming functions, on 8 nodes, on all three systems, in Figure 19 and Figure 20. In Figure 19, for the KPM application, we observe that two of the most consuming functions are `data._mul_scalar` and `scipy.sparse._sparsetools.csr_matvec`, which correspond to the most compute-intensive portions of the application on all systems, and refer to scalar matrix multiplication and sparse matrix-vector multiplication in SciPy. We can also observe that the communication time we detected on Hawk and Vulcan corresponds to a `Reduce` operation in MPI, which appears not to consume any significant amount of time on Altair, making this machine more performant for this application. We finally observe that a larger percentage of the execution time is consumed on the `numpy.ndarray.copy` function on Hawk, which is possibly a more memory-bound function.

For SN-simulator, in Figure 20, we observe very different percentages spent on different functions on the three systems. A significant amount of time on all three systems is spent on the `sleep` function, with almost 40% of the execution time of Hawk resulting in idle cores. This consumption is an indicator of load imbalance, which is also the most plausible cause for the reduced performance of the application on Hawk. Additionally, we observe a notable percentage of execution time spend on a `Bcast` function on Hawk and Vulcan, however, given that the percentage of MPI time is higher on Vulcan, we safely assume that this function is actually faster on Hawk for the same number of nodes. We finally observe a significant amount of time spent on the `propagation.edge_sample_Numba` function on Hawk and Altair, and an almost equivalent amount of time spent on the `numpy.random.mtrand.RandomState.choice` function on Altair. The latter is actually called from the `edge_sample_Numba` function, and included in its time, yet on Altair, this appears in our profiles as the most time-consuming part.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	43 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

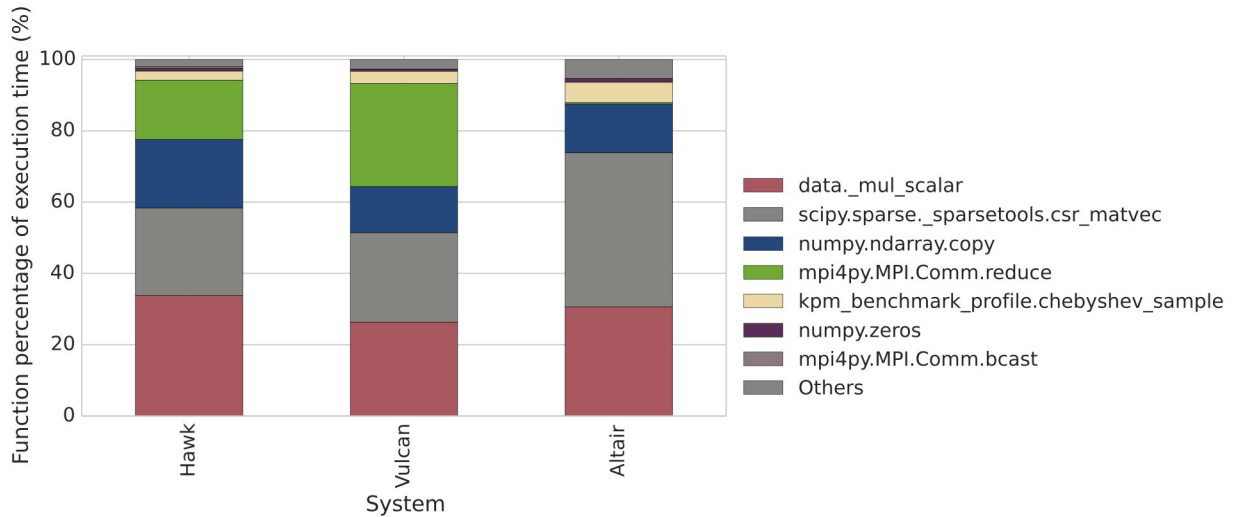


Figure 19. Examining the percentage of total execution time spent on the various functions of KPM on different systems on the pokec-5000 graph using 96 intervals, 512 samples and a degree of 62, for 8 nodes.

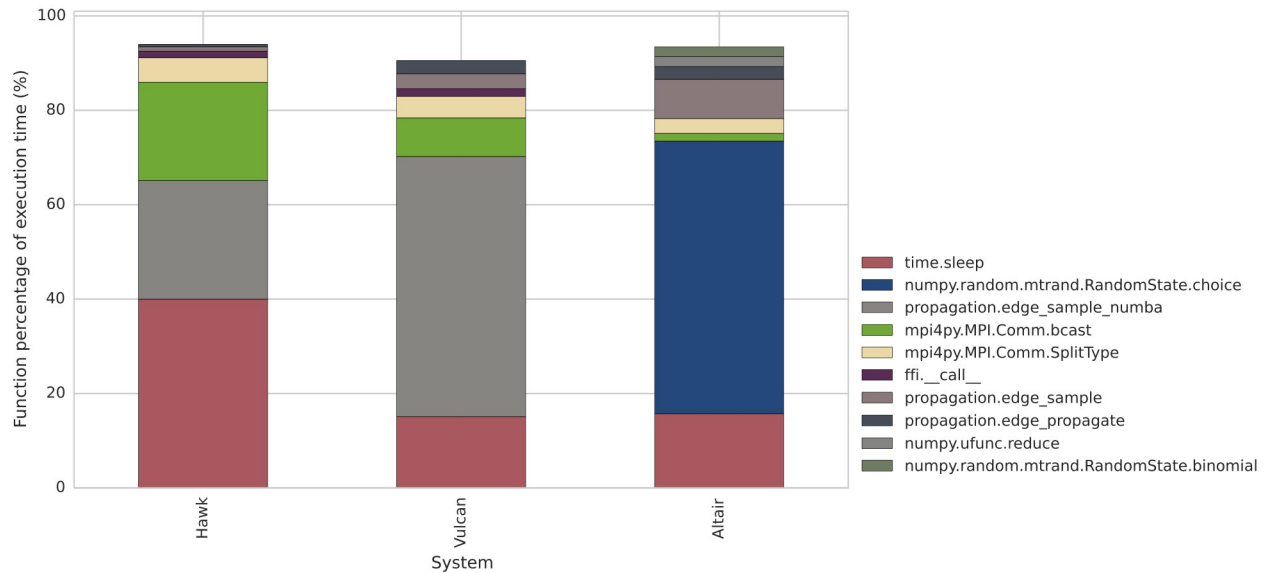


Figure 20. Examining the percentage of total execution time spent on the various functions of SN-simulator on different systems on the NEOS dataset using 400 sources, 1000 samples and 42 as the seed, for 8 nodes.

2.4.2.4 POP CoE Co-Design Metrics

To conclude our analysis, we compute the POP CoE co-design metrics for the application on the three systems. Table 8 presents the POP metrics for KPM on 8 nodes of all three systems, and Table 9 presents the same metrics for SN-Simulator on 8 nodes of all three systems.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	44 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

System	LB	CE	PE	CS	GE
Hawk	71.99%	99.93%	71.95%	91.44%	65.79%
Vulcan	60.44%	99.29%	60.01%	88.42%	53.06%
Altair	94.25%	99.98%	94.24%	99.68%	93.93%

Table 8. POP co-design metrics for KPM on different systems on the pokec-5000 graph using 96 intervals, 512 samples and a degree of 62, for 8 nodes. The five metrics are Load Balance Efficiency (LB), Communication Efficiency (CE), Parallel Efficiency (PE), Computational Scaling (CS) and Global Efficiency (GE).

We observe that KPM demonstrates fair Load Balance (LB) Efficiency on all systems, but with a significant improvement on Altair. The root cause for this is a combination of the amount of work to be distributed per processor, the time to complete this useful work per processor, and the different number of processes on each system. Communication Efficiency (CE) is high on all three systems on 8 nodes, since some of the communication is overlapped with the computation of the process with the maximum computation time, hence mitigating the higher MPI times in the case of Hawk and Vulcan. Parallel Efficiency (PE) is mostly determined by Load Balance Efficiency, with Altair giving the higher score. In terms of Computational Scaling (CS), computed with the case of the single node as the reference point, Vulcan is the system with the lower score, which owes to an increase in the number of instructions, and additionally a drop in the IPC, compared to the single-node case. We note however, that Computational Scaling is high on all three systems, since the main functions for computation in this application refer to widely-used and optimized operations on HPC systems, like scalar multiplication and SpMV. Overall, the Global Efficiency (GE) of the application is very high on Altair, but much lower on Hawk and Vulcan, as also indicated by the equivalent speedups and previous analysis.

System	LB	CE	PE	CS	GE
Hawk	61.45%	98.38%	60.45%	30.02%	18.15%
Vulcan	71.15%	89.65%	63.79%	71.85%	45.83%
Altair	83.37%	99.09%	82.61%	66.97%	55.32%

Table 9. POP co-design metrics for SN-simulator on different systems on the NEOS dataset using 400 sources, 1000 samples and 42 as the seed, for 8 nodes. The five metrics are Load Balance Efficiency (LB), Communication Efficiency (CE), Parallel Efficiency (PE), Computational Scaling (CS) and Global Efficiency (GE).

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies			Page:	45 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0
				Status:	Final

In Table 9, we observe that SN-simulator suffers both from average Load Balancing Efficiency and poor Computational Scaling, which result to an average Parallel Efficiency and a very low Global Efficiency. Vulcan and Altair score better than Hawk in all metrics, resulting to better Global Efficiency. On Hawk, however, as also indicated by our previous analysis of the application profile, very little computational work is performed per process, and our analysis with `perf` additionally indicates an excessive number of instructions being executed compared to the single-node case. We can therefore conclude that the SN-simulator, in its current state, is not able to take advantage of the ample parallelism offered on the 8 nodes of Hawk.

2.4.3 Overall analysis

Our co-design analysis demonstrates that the KPM application is computationally efficient but requires fine tuning of its communication software stack to take advantage of the underlying system. With our current setup, the best system for this application is Altair, however we expect performance degradation due to communication to disappear with the co-design of the application's communication over the relevant software stack on Hawk and Vulcan. For the case of the SN-simulator, we find that this application is heavily unbalanced, however it can be efficiently scaled up to a few hundreds of cores on modern systems like Vulcan and Altair, with the given input. A careful load balancing scheme can release a better scaling potential for the application on systems like Hawk, given that the problem size is increased appropriately to produce meaningful computation for each process/core.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	46 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

3 HPC benchmarking

At this stage of the HiDALGO project, we have focused on benchmarking HiDALGO Pilot applications for two purposes: 1. Verify the correctness and scalability of new versions of applications of the Pilots on the HiDALGO systems, 2. Extend the scalability analysis on other additional HPC system architectures, and in particular, the Mare Nostrum 4 supercomputer at BSC, the Piz Daint supercomputer at CSCS, and the SUPERMUC-NG supercomputer at LRZ, made available to the HiDALGO project through the PRACE project [21]. All benchmarking results are stored in their raw format in the HiDALGO benchmarking repository. HPC benchmarking follows the HiDALGO methodology for benchmarking, as outlined in HiDALGO Deliverable 3.1, unless otherwise stated.

Data repository folder:

https://gitlab.com/eu_hidalgo/benchmarking/-/tree/master/deliverable_3_5/scalability

3.1 Migration Pilot

3.1.1 Introduction and Goal

The core component of the Migration Pilot, parallel Flee, has undergone minor changes in its code base in the latest months of the projects, leading to a code release. In this section, we focus on the recent Flee 2.0 release [15]. As Flee has been thoroughly evaluated with respect to its performance and scalability in previous HiDALGO deliverables, in this deliverable, we consider a single macro-scale scenario, of unprecedented scale, stressing the simulation component to a very large synthetic graph, with a high number of agents, and high numbers of cores. Our experiments are performed on all available large-scale HiDALGO systems, i.e. Hawk and Vulcan at HLRS and Altair and Eagle at PSNC, but also on the Mare Nostrum 4 [22] supercomputer at Barcelona Supercomputing Center (BSC), where access was acquired through the PRACE project.

In this scenario, the initial number of agents in the simulation is 100,000,000 ($N=100M$), with no new agents added at any time step. The synthetic graph consists of 10,000 vertices/locations with a connectivity degree equal to 8 ($G=100-100-8$). We perform simulations for ten time steps ($t=10$), the equivalent of 10 simulated days.

In Table 10 we describe the software environment used for Flee on the five systems, for purposes of reproducibility.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	47 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

Software	Hawk	Vulcan	Eagle	Altair	Mare Nostrum 4
Python	3.8.3	3.7.10	3.8.12	3.8.12	3.8.2
Pandas	1.0.5	1.2.5	1.1.5	1.1.5	1.0.5
Numpy	1.19.0	1.20.2	1.20.1	1.20.1	1.19
Mpi4py	3.0.3	3.0.3	3.0.3	3.0.3	3.0.3
MPI	MPT 2.23	OpenMPI 4.0.5	OpenMPI 4.1.0	OpenMPI 4.1.0	Intel(R) MPI Library for Linux* OS, Version 2017 Update 3 Build 20170405

Table 10. Software environment for Flee on the HiDALGO and PRACE systems.

3.1.2 Experimental results

The experimental results are demonstrated in Figure 21, for the four HiDALGO systems, and Mare Nostrum. First, we observe that execution time is significantly lower on Altair and Vulcan, compared to the remaining three systems, however, we note that the nodes of Altair are similar to the nodes of Mare Nostrum, thus the observed difference can safely be attributed to the software stack. We additionally note that Eagle, although hosting older-generation nodes than the other systems, demonstrates high scalability, up to 64 nodes. Another observation is that the scalability of Flee begins to diminish on more than 64 nodes on Hawk and Vulcan, which share the same interconnection network technology. We therefore believe that, in this case, the application is limited by communication over the particular network.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies			Page:	48 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0
				Status:	Final

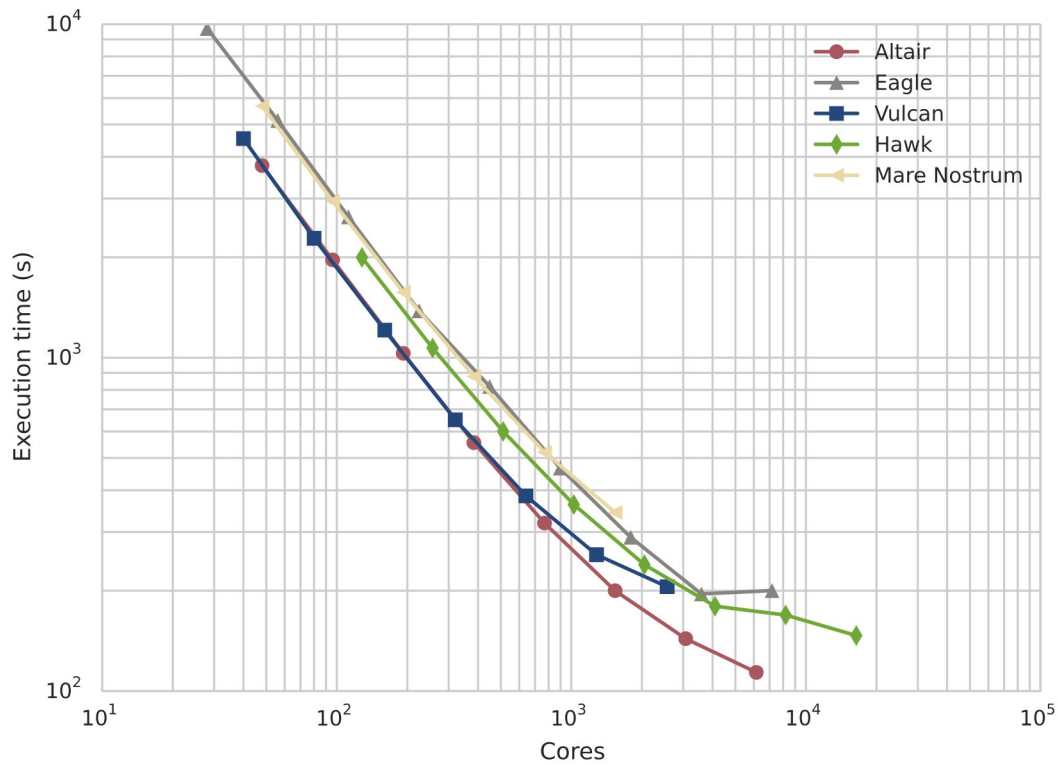


Figure 21. Comparing execution time of Flee on different systems on a synthetic 100-100-8 graph using 100M initial agents, for 10 time steps, for different numbers of nodes. Both axes are in logarithmic scale.

3.1.3 Overall analysis

Overall, we conclude that Flee is scalable on all different systems, without any porting effort, on thousands of cores, even using non-trivial problem sizes, on up to 256 nodes, which translates to 6144 cores on Altair, and 16384 cores on Hawk, with a speedup with respect to a single node of 33x and 13x respectively.

3.2 Urban Air Pollution Pilot

3.2.1 Introduction and Goal

The current benchmarks of the Urban Air Pollution pilot focus on the M32, 2021 July version of the CFD module of the UAP pilot. Current focus is solely on the OpenFOAM implementation. Actual developments include further optimizations of the transient simulation,

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	49 of 174	
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status:	Final

`pimpleFoam`, including new features like multicomponent pollutants and canopy modelling. Performance optimization is discussed in detail in chapter 5.2.2.1. The benchmarked meshes were not changed so performance improvement is easier to demonstrate.

Benchmarks are issued on systems of various architectures, compilers and MPI libraries. Versions of the necessary software are summarized in the following Table 11:

Software	Hawk	Altair	Piz Daint
compiler	GCC 9.2	GCC 10.2	CRAY CLANG 11
MPI	HPE MPT 2.23	OpenMPI 4.1.0	CRAY-MPICH 7.7.16
scotch	6.0.9	6.0.9	6.0.9
OpenFOAM	v20 12	v20 12	v20 06

Software	Joliot-Curie	Vulcan	Eagle	Milan
compiler	ICC 19.1.0.166	GCC 10.3	GCC 10.2	GCC 9.3.0
MPI	OpenMPI 4.0.5	OpenMPI 4.1.1	OpenMPI 4.1.0	OpenMPI 4.0.2
scotch	6.0.9	6.0.9	6.0.9	6.0.9
OpenFOAM	v20 06	v20 12	v20 12	v20 12

Table 11. Software specification for UAP pilot benchmarks.

For details on the instantiation of the simulation, we refer the reader to D3.3. The execution scenarios use generated 3D meshes (octree) of different sizes with wind profiles obtained from ECMWF weather data on the boundary and emission from traffic simulation based on loop data. Various mesh sizes are run for various iterations and different time spans are simulated to fit into time limit for the 1-core simulation. The number of iterations in `simpleFoam` and the simulated time in `pimpleFoam` are reported in the Table 12 below.

Tag	Mesh size	Iterations	Simulated time
728k	728 000	600	3600 s
3.4M	3 400 000	600	900 s
14M	14 000 000	400	100 s

Table 12. The number of iterations in `simpleFoam` and the simulated time in `pimpleFoam`.

All reported times are scaled to a steady simulation of 600 iterations and a simulated time of one hour, the smallest time frame considered in productive runs. For most systems, benchmarks have been repeated only once, due to high load and queue times. We rely on internal OpenFOAM functions to report execution time: the timestamp reported after

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies			Page:	50 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0
				Status:	Final

completion of the first iteration is subtracted from the timestamp reported at completing all calculations, so the initialization part is not considered for scalability.

3.2.2 Results

Results are presented per mesh size separately. Runtime is shown for both steady simulation (simpleFoam) and transient simulation (pimpleFoam). Also, per node speedup and parallel efficiency is shown, although only for the transient simulation, as it takes significantly longer. All seven architectures are presented on one plot, visually grouping AMD based systems with a reddish, and INTEL based systems with blueish colours. Table 13 below summarizes all architectures w.r.t. CPU type, legend, and maximum number of nodes and cores used.








Legend	Architecture	cores per node	# max nodes			# max cores		
	CPU(s)		728k	3.4M	14M	728k	3.4M	14M
	HAWK AMD EPYC 7742	2 x 64	8	64	782*	1 024	8 192	100 096*
	JCR AMD EPYC 7H12	2 x 64	4	32	64	512	4 096	8 192
	MILAN AMD EPYC 7763	2 x 64	8	64	64	1 024	8 192	8 192
	ALTAIR INTEL Xeon Platinum 8268	2 x 24	32	32	32	1 536	1 536	1 536
	VULCAN INTEL Xeon Gold 6248	2 x 20	64	64	64	2 560	2 560	2 560
	EAGLE INTEL E5-2697v3	2 x 14	16	32	32	448	896	896
	PIZ DAINTE INTEL E5-2690v3	1 x 12	32	64	128	384	768	1 536

Table 13. The number of iterations in simpleFoam and the simulated time in pimpleFoam.

Only the steady state simulation is run for 782 cores on HAWK, as runtime already increases at 512 nodes for the transient and 128 nodes for the steady state simulation.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies			Page:	51 of 174	
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

3.2.2.1 Small mesh size (728k cells) analysis

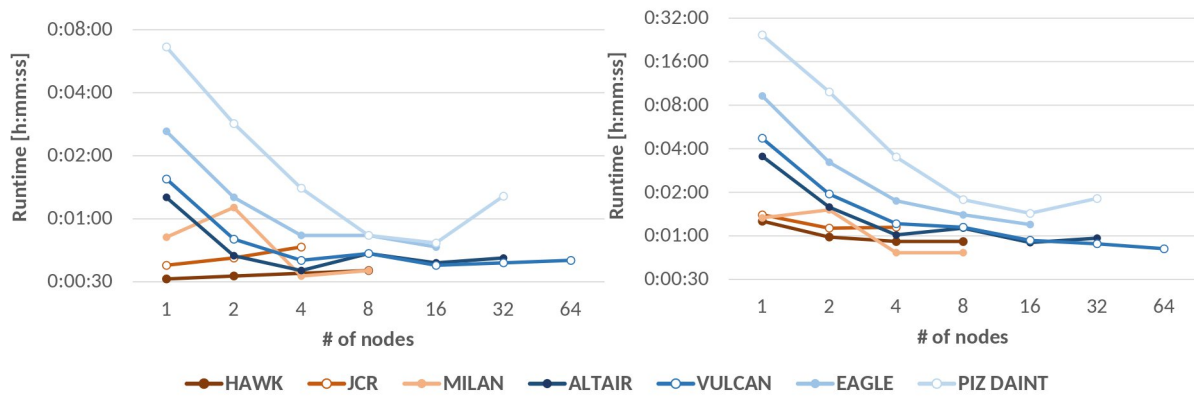


Figure 22. Execution time vs. number of nodes for steady (left) and transient (right) simulation of the OpenFOAM Air Quality Dispersion Model for small mesh size on different architectures. Both axes use logarithmic scale.

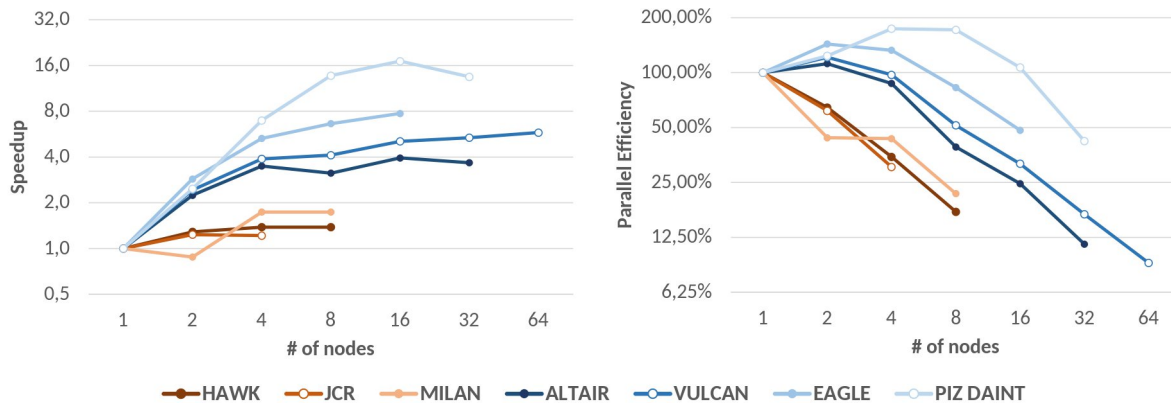


Figure 23. Per node Speedup (left) and Parallel efficiency (right) vs. number of nodes for transient simulation of the OpenFOAM Air Quality Dispersion Model for small mesh size on different architectures. Both axes use logarithmic scale.

Figure 22 and Figure 23 compare different architectures on the investigated metrics with the small mesh size. AMD type architectures do have better single node performance regarding runtime but moving to more nodes is more beneficial on INTEL type architectures. For both types, newer architectures perform better. For steady simulation, HAWK performs best with 31 seconds on 1 node and ALTAIR with 34 seconds on 4 nodes. For transient, MILAN performs best at 46 seconds with 4 nodes and VULCAN with 49 seconds on 64 nodes. Older INTEL type architectures profit significantly more from multi node simulations with efficiencies exceeding 100%.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	52 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

3.2.2.2 Middle mesh size (3.4M cells) analysis

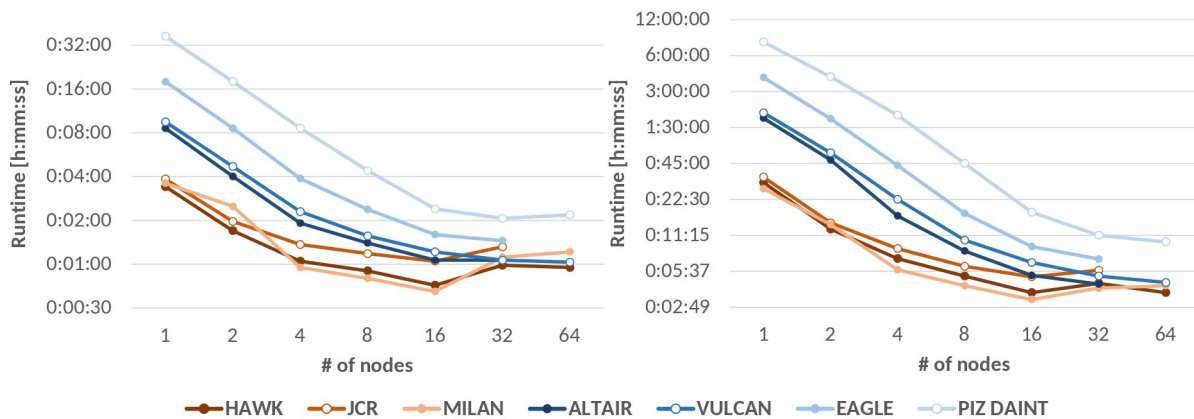


Figure 24. Execution time vs. number of nodes for steady (left) and transient (right) simulation of the OpenFOAM Air Quality Dispersion Model for middle mesh size on different architectures. Both axes use logarithmic scale.

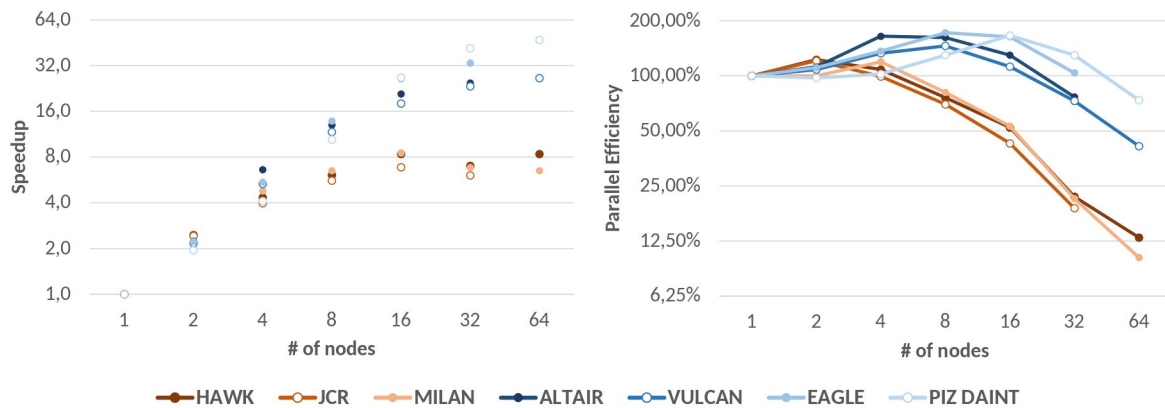


Figure 25. Per node Speedup (left) and Parallel efficiency (right) vs. number of nodes for transient simulation of the OpenFOAM Air Quality Dispersion Model for middle mesh size on different architectures. Both axes use logarithmic scale.

Figure 24 and Figure 25 compare different architectures on the investigated metrics with the middle mesh size. AMD architectures still have better single node performance regarding runtime but, again, moving to more nodes is more beneficial on INTEL architectures up. For both types, newer architectures perform better. For steady simulations, MILAN performs best with 39 seconds on 16 nodes and VULCAN with 62 seconds on 64 nodes. For transient, MILAN performs best at 196 seconds with 16 nodes and ALTAIR with 264 seconds on 32 nodes. All INTEL type architectures profit significantly more from multi node simulations with efficiencies exceeding 100% in many cases. AMD type architectures, however, do show super-linear scaling behaviour only up to 4 nodes.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	53 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

3.2.2.3 Large mesh size (14M cells) analysis

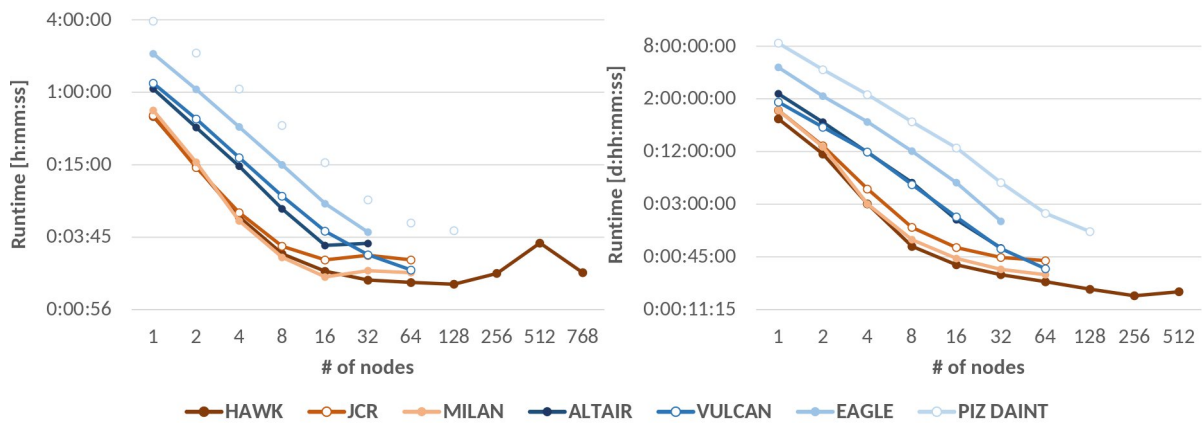


Figure 26. Execution time vs. number of nodes for steady (left) and transient (right) simulation of the OpenFOAM Air Quality Dispersion Model for large mesh size on different architectures. Both axes use logarithmic scale.

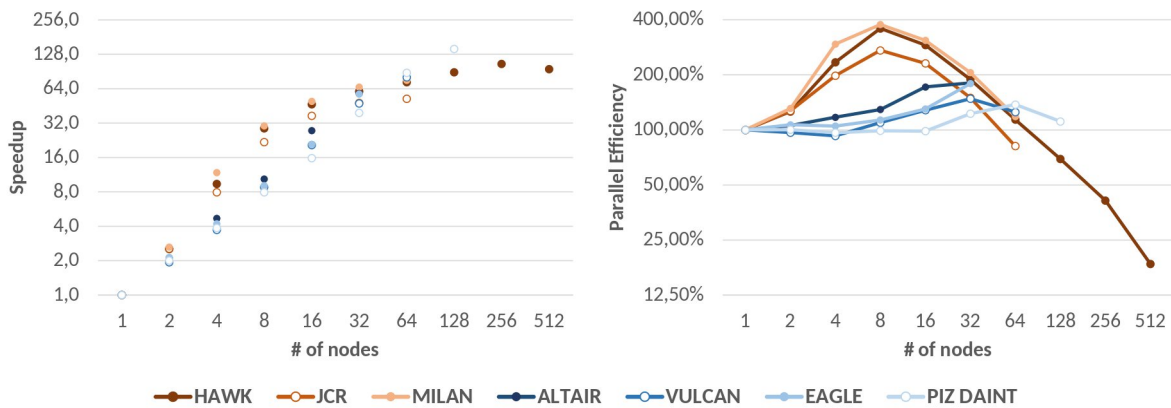


Figure 27. Per node Speedup (left) and Parallel efficiency (right) vs. number of nodes for transient simulation of the OpenFOAM Air Quality Dispersion Model for large mesh size on different architectures. Both axes use logarithmic scale.

Figure 26 and Figure 27 compare different architectures on the investigated metrics with the large mesh size. AMD type architectures now exhibit only a relatively smaller single node performance advantage, and this is increased at 8 nodes. At 64 nodes, modern INTEL and AMD architectures show similar performance. Fastest calculations for steady simulation are done on HAWK at 92 seconds on 128 nodes and VULCAN at 120 seconds on 64 nodes. Best performance for transient simulations are shown on HAWK with 16:12 [mm:ss] with 256 nodes, while VULCAN performs the simulation in 33:00 [mm:ss] although using only 64 nodes. INTEL architectures show stable scaling behaviour. Speedup is almost linear, with slight superlinearity above 8 nodes, which is strongest now on the newest architecture, ALTAIR. On

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	54 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

the other hand, AMD machines show strong super-linear behaviour for a node count up to 8, and strong sublinear behaviour afterwards. Running with 782 nodes on HAWK hits the 100 000 core mark with a reasonable runtime, which suggests, that the code is a good candidate to run on this number of nodes and may even be scalable with a larger cell count mesh.

3.2.3 Analysis

To analyse performance, we can group architectures to AMD and INTEL types. Intel architectures generally do present a lower per node performance, albeit also with lower number of cores per node. They show a more stable parallel efficiency. Super-linear behaviour can be observed for all mesh sizes. AMD architectures exhibit lower per node runtimes with higher per node core count. Super-linear behaviour is only observed with the large mesh size, however, here it is stronger.

We can observe, that save a few exceptions, there is no minimum in runtime with Intel architectures, so adding extra nodes may yield extra performance. All AMD types have a minimum performance at around 4 nodes for the small and at 16 nodes for the middle mesh size. For large mesh size, maximum performance was achieved with 256 nodes on HAWK.

Looking at per node speedup, all AMD architectures had less than 2 for the small mesh size, top value was achieved by PIZ DAINTE at 17 with 16 nodes. For middle mesh size, top AMD was MILAN with 8.5 at 16 nodes, and top Intel again PIZ DAINTE with 47 at 64 nodes. Top per node speedup for high mesh size is 142 on PIZ DAINTE with 128 nodes and 105 for HAWK for 256 nodes.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	55 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

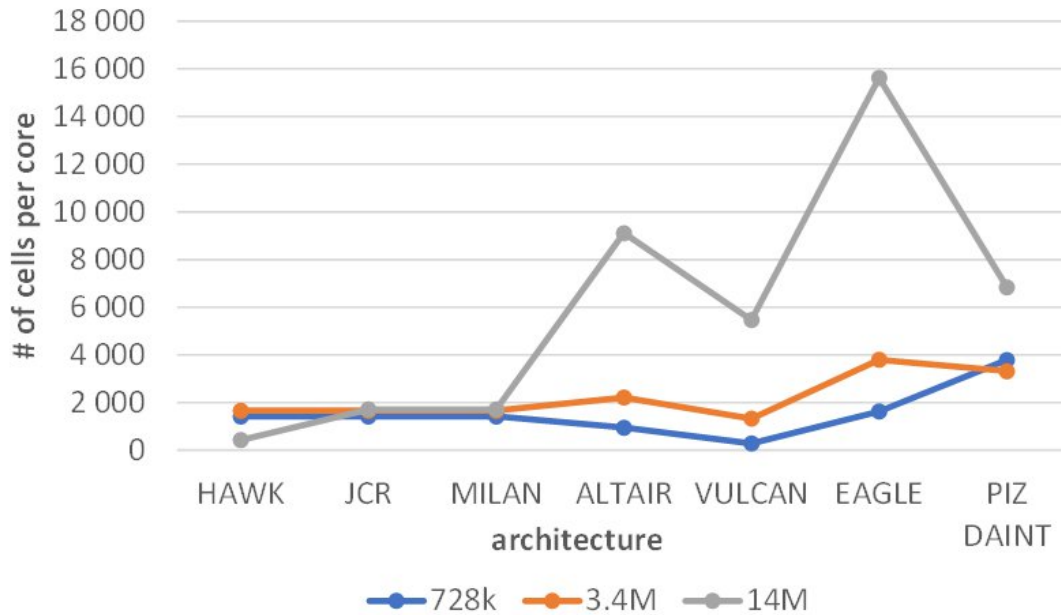


Figure 28. Per core cell count on various architectures and mesh sizes.

To assess performance on larger cell count meshes, it is informative to calculate the number of cells per core for the maximum performance core count. Figure 28 presents cell per core counts on all investigated architectures and mesh sizes. All AMD platforms have sub 2000 cell count per core, that is mostly homogeneous on all mesh sizes. Newer Intel platforms also have this value for small and middle mesh. Large mesh size and older architectures did not reach this number, also do have potential to benefit from more nodes. So, the 2000 is a realistic cell per core limit to estimate core count need for maximum performance.

3.3 Social Networks Pilot

3.3.1 Introduction and Goal

Throughout this Section we focus on the analysis and benchmarking efforts targeted on the two major components of the social network pilot. First, we look at the performance of the simulation framework itself (called “SN-Simulator”), followed by a discussion of the performance of our eigenvalue calculator (called “KPM”).

Since the last set of benchmarks (presented in Deliverable 3.4) we made numerous improvements to both aforementioned applications. A more detailed explanation and analysis of each of these changes can be found in Section 5.3. One of the goals of this section is to

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	56 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

assess the impact these improvements have made on our applications. On the other hand, we also prepared additional benchmarking scenarios and, for the first time, report benchmarks that were performed on the Altair.

In total, we performed three experiments on the Hawk, Altair and SuperMUC-NG supercomputers for this section. SuperMUC-NG is a petascale HPC system operated by the Leibniz Supercomputing Centre in Munich, Germany. It consists of more than 300,000 compute cores that are equipped with Intel Xeon Skylake Platinum 8174 processors [23]. The first two experiments are benchmarks of the SN-Simulator while in the last experiment we considered the KPM application. It is worth to mention that we performed an additional small benchmark of the SN-Simulator in Section 4.3 in form of an ensemble run, where we utilized up to 100K cores on SuperMUC-NG.

3.3.1.1 Description and Input Data

In order to allow the reader to become familiar with our benchmarked applications we quickly summarize the most important aspects as well as input parameters.

SN-Simulator. At the heart of our pilot lies the message simulator itself. Given some network (for example follower-followee relationships on twitter) and some additional parameters it simulates the spread/flow of messages throughout this network.

Throughout the following tests, we will use the following data-sets as input. Each of them consists of a follower-followee network of users that tweeted about a certain topic. They also include some additional metrics that describe flow/spread of messages that are related to that topic. One such metric is for example the average number of retweets.

- **neos:** A smaller data set, which consists of roughly 4,500 tweets about the Austrian neos political party. It was acquired during the 2019 Austrian elections.
- **fpoe:** A larger data set, which was acquired at the same time as the neos data set. In this case, the data set consist of approximately 19,000 tweets about the FPÖ political party
- **covid19:** A large data set, which consists of roughly 375,000 tweets about the covid19 social distancing regulations. It was acquired at the beginning of 2020.

Other notable input parameters include samples and sources which directly control the number of tweets/messages that are simulated. The overall work the simulation needs to perform is roughly correlated to $\text{samples} * \text{sources}$.

KPM. When supplying our simulator with a synthetically generated input graph, we first need to establish whether this graph exhibits the properties of real network social networks. One such property is the so-called eigenvalue spectrum. Computing this spectrum for a given

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	57 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

graph is computationally demanding. Our KPM application computes an approximation of this spectrum in form of a histogram of eigenvalues. The accuracy of the computed histogram can be controlled via three parameters intervals, samples and degree. For our benchmark purposes we considered the friendship-relationship graph of the pokec social network (provided by SNAP [24]). It contains approximately 1 million users.

3.3.1.2 Software Environment

For all of the newly performed benchmarks, we used the following software environments (Table 14).

Software / Package	Hawk and SuperMUC	Altair
Python	3.9.6	3.8.12
Numba	0.53.1	0.53.1
Numpy	1.21.1	1.21.2
Scipy	1.6.3	1.7.1
Pandas	1.3.0	1.3.4
Mpi4py	3.1.1	3.1.1
MPI Library	MPICH 3.4.2	OpenMPI 4.1.0
SN-Simulator Version	0.3	0.3
KPM Version	0.2	0.2

Table 14. Software stack for Social Networks benchmarks.

Note, in some experiments we use previously obtained results for comparison. This concerns results for v0.2 for the SN-Simulator and v0.1 for the KPM application. The exact software environment for these results was stated in Deliverable 3.4. The code of the SN-Simulator as well as the KPM application can be found on [19] and [18], respectively.

3.3.2 Experiment 1 (SN-Simulator)

For our first experiment, we start with a comparison between the new version 0.3 of our SN-Simulator and the previously benchmarked version 0.2 (see Deliverable 3.4 for these benchmarks). We consider the same parameters and data sets as in the benchmarks of version 0.2. Additionally, we perform our simulations on the same supercomputing systems. That is, we considered the neos and fpoe datasets, set sources to 40, samples to 1000, and

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	58 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

performed the benchmarks on Hawk as well as SuperMUC-NG. Due to the lack of benchmarking results of version 0.2 on Altair, we postpone tests on this system until Experiment 2 in Section 3.3.3.

3.3.2.1 Results

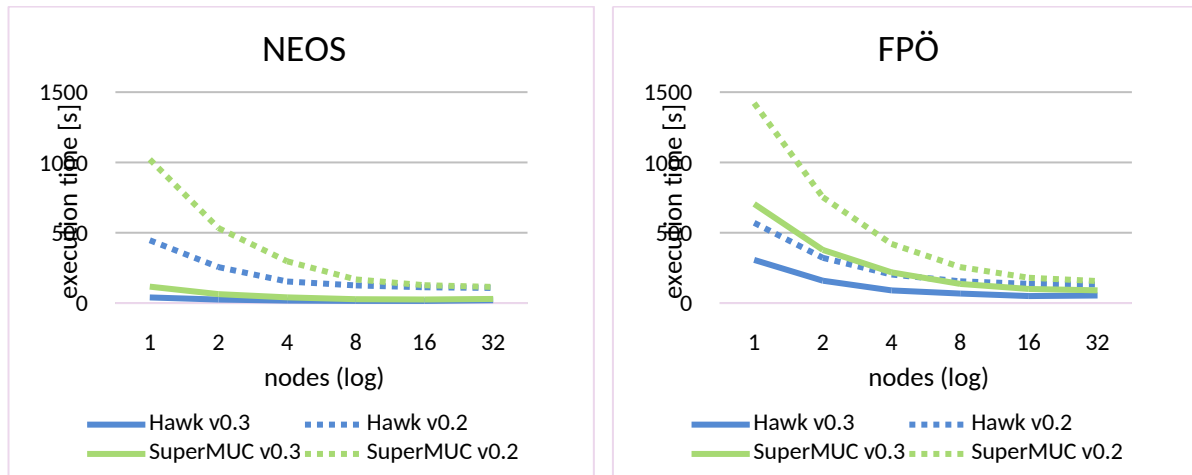


Figure 29. Comparison of the running times of our new (v0.3) and old (v0.2) SN-Simulator on Hawk and SuperMUC-NG. On the left, we considered the neos data set. On the right, the fpoe data set.

3.3.2.2 Analysis

As we can see, the running time required by v0.3 of our simulation decreased significantly for both data sets on both systems. For the smaller NEOS data set, we can observe a speed-up of approximately 10 times on a single node of Hawk and approximately 9 times on SuperMUC-NG. Also, in case of the fpoe dataset we can observe a significant speed of almost factor 2. This speed-up is mostly caused by two optimizations that we implemented in the previous months. First, the re-ordering of tasks when distributing them among the MPI-workers. We applied heuristics to encourage long tasks (consisting of the simulations of tweets which may be retweeted frequently) being scheduled first. The second optimization technique we applied is to translate certain hotspots of the code into efficient machine code via help of the python package Numba. Both of these improvements, together with extensive analysis of their impact on various processor architectures, are described in Section 5.3.

3.3.3 Experiment 2 (SN-Simulator)

The performance improvements of the SN-Simulator caused the running time of our previous experiments to become quite low (<30 seconds starting with 8 nodes on SuperMUC-NG). This

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies	Page:	59 of 174
Reference:	D3.5	Dissemination:	PU
	Version:	1.0	Status: Final

made it hard to estimate the scalability of our application. Therefore, we also performed a larger experiment based on our two bigger data sets fpoe and covid. Additionally, we increased the sources and samples parameters to 5000 and 10000, respectively. This causes the number of simulated tweets to increase by a factor of roughly 1250 compared to the previous experiment. Furthermore, we supplied our application with additional input files (corr and discount-factor) that allow further fine-tuning of the simulation model. This leads to simulations more closely match the behaviour that can be observed in the real world. Additionally, this increases the computational demands of our simulation. Due to the high resource requirements of this experiment, we performed the run for each data point only once.

3.3.3.1 Results

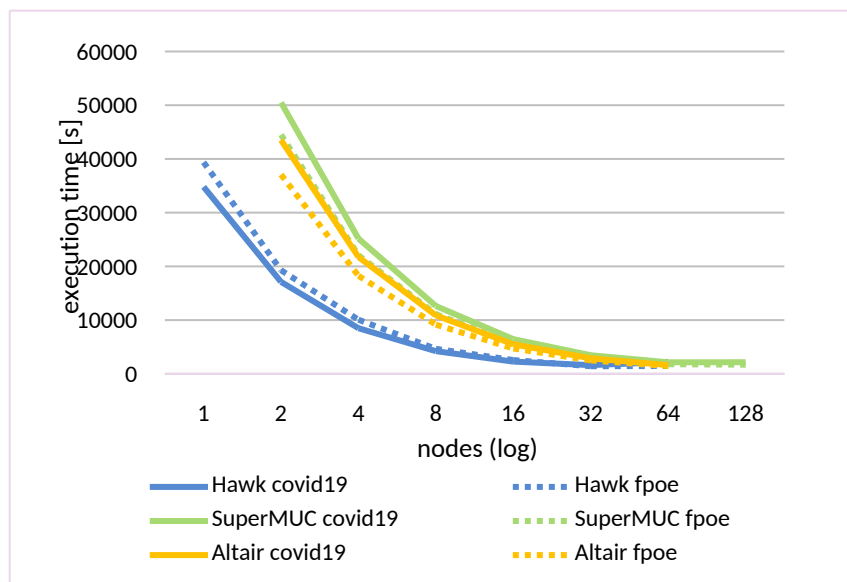


Figure 30. Time required by v0.3 of our SN-Simulator for the simulation of the covid19 and fpoe data sets on Hawk, SuperMUC-NG and Altair for an increasing number of compute nodes.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	60 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

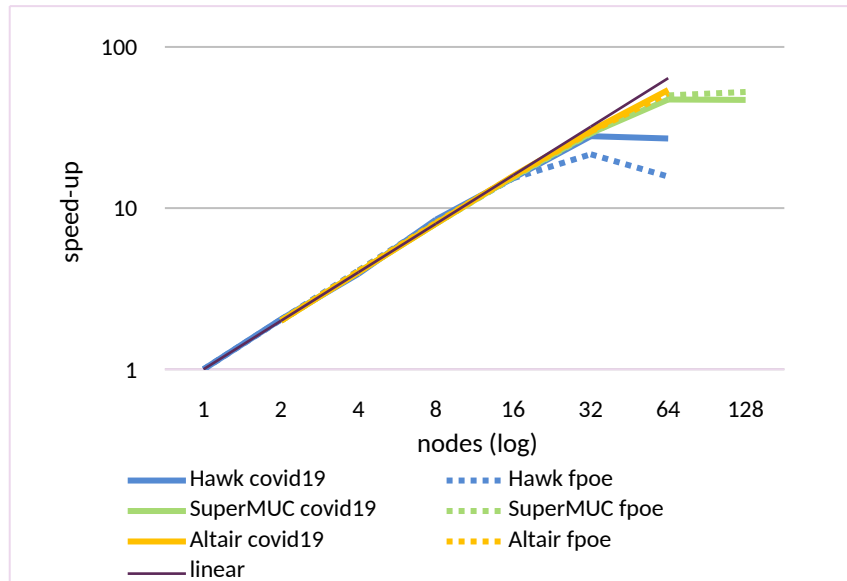


Figure 31. Speed-up of v0.3 of our SN-Simulator in comparison with the linear speed-up denoted by “linear”

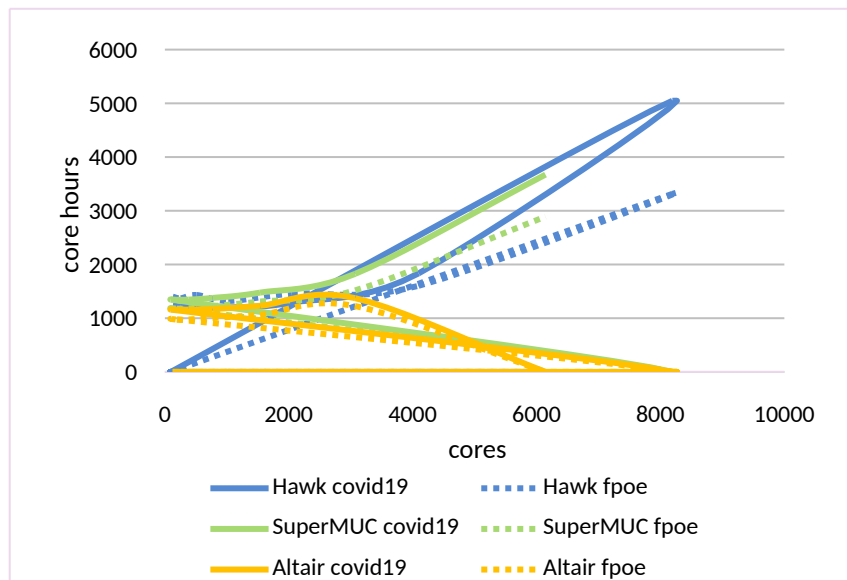


Figure 32. Core hour budget required to complete the simulation when supplied with a certain amount of cores. On the x-axis we plot the number of CPU nodes involved in the simulation. On the y-axis the overall time required times the number of cores used.

3.3.3.2 Analysis

As we can see in the first plot (Figure 30), Hawk performs best for both datasets when executing the simulations on the same number of nodes. We suspect that this is a result of the increased number of processing cores per node compared to SuperMUC and Altair (128

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	61 of 174	
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status:	Final

vs 48). In the Figure 31, which describes the speed-up of our application, we can also see that our application scales very well until roughly 32 nodes. From this point on the benefit of adding additional nodes decreases. After roughly 64 nodes on Hawk the performance of the application starts to degrade. On SuperMUC we only observed this in the run with 128 cores (Figure 31). Again, we suspect that this threshold is reached sooner on hawk due to the larger amount of CPU cores involved in the computation, which increases the overall communication overhead. The third plot (Figure 32) serves as a comparison between the systems on a per-core instead of per-node level. When employing a similar amount of cores, the execution of our application requires roughly the same time on all systems. Altair with its Intel Xeon Platinum 8268 processors takes a slight edge compared to the other systems. Note that an application that follows an ideal linear speed-up would appear as a horizontal line in this plot. For our applications this is roughly the case until starting from approximately 3000 cores the overall core hour requirement increases.

3.3.4 Experiment 3 (KPM)

In this experiment we consider the KPM application. This experiment mostly serves as a comparison of the new version (v0.2) to the previous version (v0.1) of our application. We calculate the eigenvalue histogram of the full pokec friendship network mentioned in Section 3.3.1.1. This graph consists of more than 998k nodes and can be found on [25]. We set intervals to 192, samples to 512 and degree to 62. Note that an experiment with the same input graph was performed in Deliverable 3.4. However, in that experiment we used different values for intervals, samples and degree (101, 200 and 300, respectively). Due to a different approach in distributing the workload among the CPU cores we can no longer support these parameter values in the new KPM version. While this does not allow for an exact comparison, we note that the overall work required for both these selections of parameters is very similar. This is because the overall computational effort is spent to perform $\text{intervals} \times \text{samples} \times \text{degree}$ many matrix vector multiplications. In both, the experiments with our new and old version, this amounts to roughly 6,000,000 such multiplications. Therefore, we feel that a comparison to the results of the old version presented in Deliverable 3.4. is still valid. Finally, we note that we only performed a single run for each data point due to the high resource requirements.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	62 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

3.3.4.1 Results

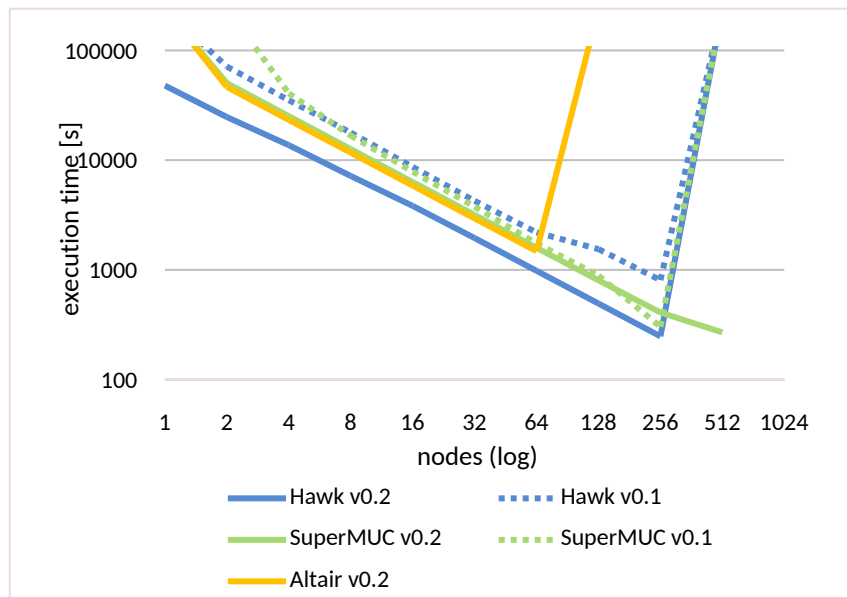


Figure 33. Time required to compute the eigenvalue histogram for an increasing amount of compute nodes on Hawk, SuperMUC-NG and Altair. Comparison of our new version (v0.2) to the old version (v0.1) of our KPM application.

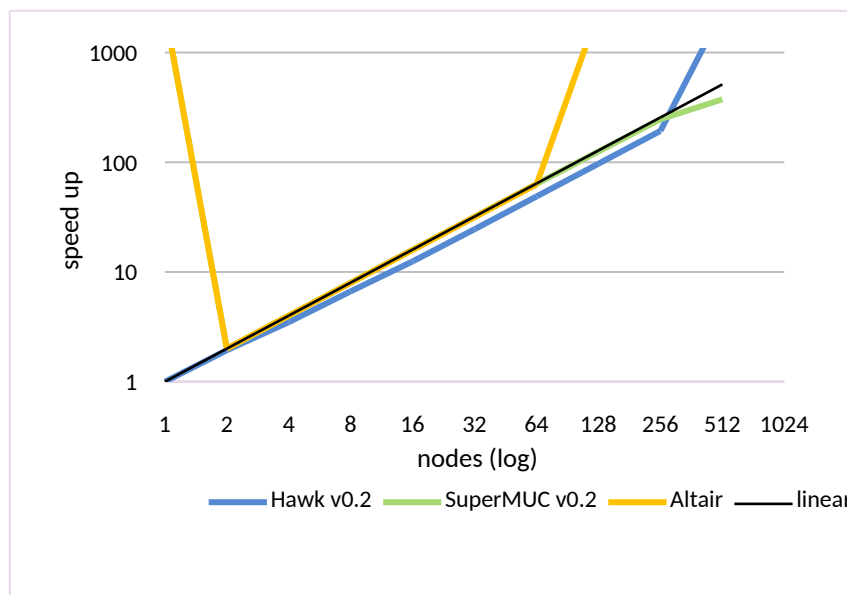


Figure 34. Speed-Up of v0.2 of the KPM application. The linear speed-up is denoted by “linear” and stated for comparison.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies	Page:	63 of 174
Reference:	D3.5	Dissemination:	PU
		Version:	1.0
		Status:	Final

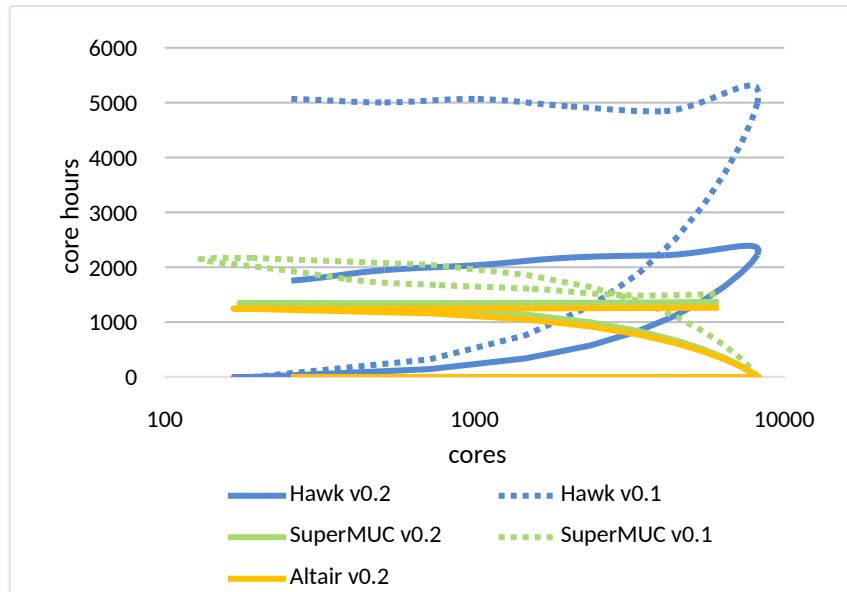


Figure 35. Core hours required to perform complete the eigenvalue histogram computation when supplied a certain amount of cores. Comparison of the new (v0.2) and old (v0.1) version of our KPM application.

3.3.4.2 Analysis

As we can see in the first figure (Figure 33), the performance of our application increased on both Hawk and SuperMUC-NG. For Hawk the performance increased by a factor of 2-3 in all observed runs. For SuperMUC-NG, the only run in which we observed worse performance for our new version is the one with 256 nodes. We suspect that the reason for this is a bug that we discovered in version v0.1 of our KPM application, which caused certain tasks to be dropped in case the amount of compute cores became too high. This error has since been fixed. We can also see in this plot that runs on Altair behaves similar to those on SuperMUC-NG when supplied with a similar amount of cores. This makes sense as both systems have the same amount of cores (48) per node. As observable in the second plot (Figure 34), we can see that the speed-up of our application is close to linear on all systems. Only for 512 nodes on SuperMUC-NG (24576 cores) the speed-up starts to worsen slightly. Another interesting comparison is presented in the third plot (Figure 35), where we look at the total number of core hours required to compute the histogram. All data series follow a strong horizontal trend, which further illustrates the near-linear speed-up of our application. Additionally, we can also observe a phenomena which we already discussed in Deliverable 3.4: v0.1 of our application performed significantly worse on Hawk than on SuperMUC. For the new version of the KPM application this gap between Hawk and SuperMUC is significantly smaller. We suspect that this improvement is caused by our newly employed ccNUMA aware task distribution, which we describe in detail in Section 5.3. The idea is to store multiple copies of the input graph in

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	64 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

the shared memory of each node to avoid contention. As the processing unit on Hawk is equipped with 8 NUMA domains and 128 cores, the performance gain is more significant than in case of SuperMUC. Compared to the cost of roughly 5000 core hours in the previous version of our software on Hawk, the new version now only requires approximately 2000 core hours. All of the above indicates that it is essential to account for the number of NUMA domains in the processor architecture to leverage full performance of our application. In Section 5.3 we give a more detailed analysis of the effect of these ccNUMA improvements when running our application on various different processor architectures. What we can also see in this third image is that, just as in case of the SN-simulator, Altair slightly tops the other systems from an efficiency standpoint (lower amount of core hours per execution).

3.3.5 Experiment 4 (KPM)

In this section we consider our cutting-edge version v0.3 of the KPM application. Compared to version v0.2 this version comes with an additional parallelization axis. This makes it suitable for runs with a huge amount of cores. Before, a single task for MPI worker consisted of one or more matrix-vector multiplications. With v0.3 it is now possible to further split and distribute each of these multiplications among multiple cores, which allowed us to achieve a run with more than 130k cores. As part of the experiment in this section, we performed a weak scaling test on Hawk. More precisely, we fixed the parameters samples to 128 and degree to 300 and consider the same input graph as in Experiment 3 (the full pokec network consisting of more than 998K nodes). The intervals parameter is set to $(\#nodes / 4)$, which causes the amount of work required to increase proportionate to the number of supplied compute nodes. On the side of the output, this has the effect of letting the resolution of the computed eigenvalue histogram (number of bins) depend on the number of nodes. Due to the large amount of resources involved, we performed the experiment once for each of the node amounts in the set $\{4, 16, 64, 256, 1024\}$. We used the following software environment for our experiments (Table 15):

Software / Package	Version on Hawk
Python	3.8.6
Numpy	1.19.4
Numba	0.50.1
Scipy	1.5.4

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	65 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

Mpi4py	3.0.3
MPI Library	MPT 2.23
KPM Version	v0.3

Table 15. Software stack for Experiment 4 on Hawk.

3.3.5.1 Results

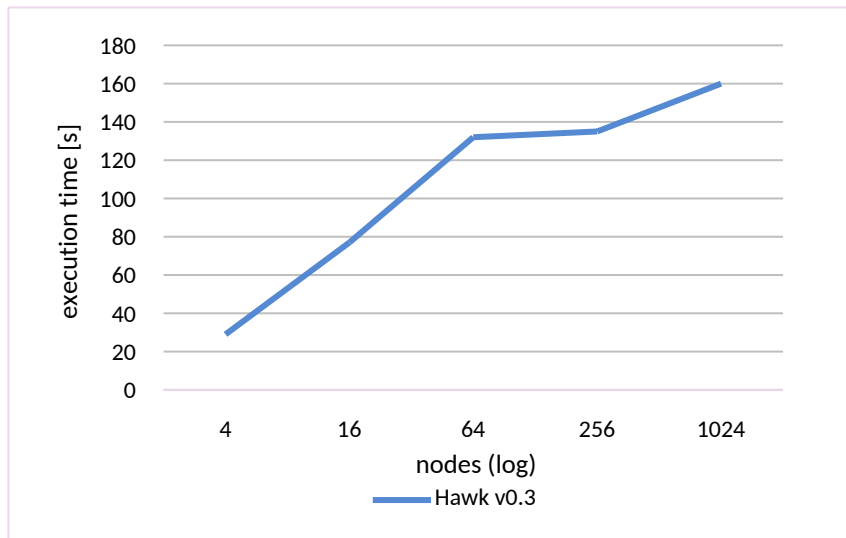


Figure 36. Weak scaling experiment. Time to compute the eigenvalue histogram with an increasing number of intervals for an increasing amount of compute nodes. Experiments performed on Hawk with KPM v0.3.

3.3.5.2 Analysis

The newly introduced parallelization strategy allowed our application to be executed successfully on up to 1024 nodes (**131,072 cores**). For 64, 256 and 1024 the application behaves closely to the ideal behaviour (Figure 36). That is, even though the overall work as well as the node amount is multiplied by 4 in each step, the execution time remains similar (more precisely it remains in the interval [132,160]). It seems that the newly introduced parallelization strategy of distributing the matrix-vector multiplications over multiple cores indeed lends itself to runs with a large amount of cores. We are currently investigating the increase in running time which occurs when increasing the number of supplied compute nodes from 4 to 16 and from 16 to 64.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	66 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

4 Ensemble scenarios

Ensemble scenarios constitute an essential approach to conduct computation composed of series of applications run in parallel. In all HiDALGO pilots it is used to efficiently test simulation facets on data sensitivity and validation. Due to the increased requirements of this approach, it is necessary to check the behaviour of the application in terms of hardware and software.

Data repository folder

https://gitlab.com/eu_hidalgo/benchmarking/-/tree/master/deliverable_3_5/ensemble_scenarios

Specification of the infrastructure used for ensemble scenarios is the same as used for scalability tests in chapter 3.

4.1 Migration pilot

Migration predictions with the Flee code are normally performed using ensemble simulations. We need to use ensembles for instance to: (a) account for the probabilistic nature of the agent-based model, (b) analyse the sensitivity of the assumptions in our model for a given conflict, (c) perform a forecast that takes into account a variety of possible conflict progressions. We provide a detailed example of ensemble runs in the context of sensitivity analysis in [26], while we present an example of ensemble runs in the context of conflict forecasts in [27], and more recently in a more applied example in D4.4. In this deliverable we focus on an ensemble scenario that helps account for the probabilistic nature of the agent-based model (a).

4.1.1 Scenario description

4.1.1.1 Goal

To simulate the forced displacement in South Sudan conflict in 2016-2017 and consider the different climate conditions in this country, we need to address the combination of perceived levels of safety, accessibility or weather conditions in our constructed model. This combination affects refugees' decision to move and their fleeing speed. Besides, simulating this combination realistically requires a coupled approach. Therefore, we implemented a

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	67 of 174	
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status:	Final

multiscale model coupled with weather data. This scenario helps us to investigate the effects of the multiscale simulation and its coupling with weather data on refugees' decisions to move and their speed.

4.1.1.2 Model description

In this model, we defined the whole model with two sub-models, namely, macroscale and microscale models. Each sub-model is executed independently and agents pass between them during the simulation. In this model, each location in the location graph, where agents pass through the coupling interface, should be registered as coupled locations. In addition to coupled locations, all microscale model's conflict locations should be added to the macroscale model as ghost locations. It means that although they are added to the macroscale model, they don't have any link to other macroscale locations and this is why they are named ghost locations. They are a special type of coupling locations where (a) the macroscale model inserts agents into these locations according to the normal FLEE agent insertion algorithm and (b) at each time step, the coupling interface transfers all agents from each ghost location to the microscale model. Regarding required resources, like previous conflict scenarios, we need population and conflict data provided by ACLED [28], validation data extracted from UNHCR portal [29], and some other resources for creating location graph.

At each time step, both microscale and macroscale would run and exchange data to trigger the next run. The workflow diagram of this scenario is shown in Figure 37.

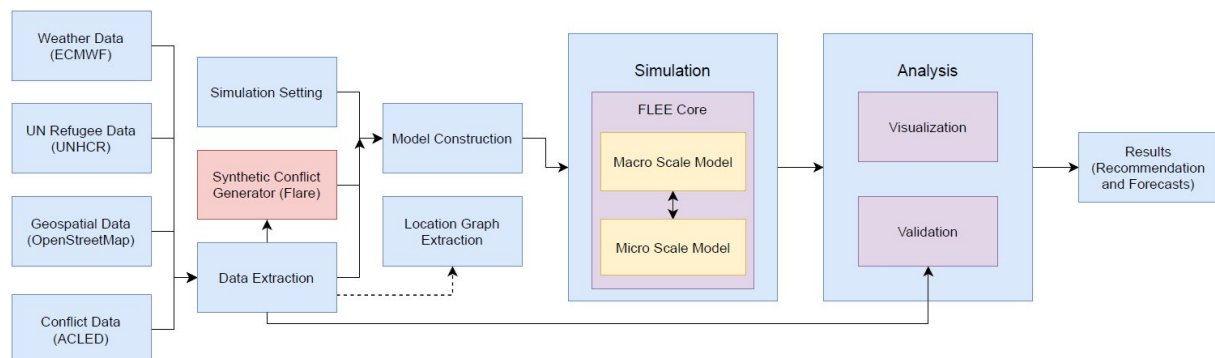


Figure 37. Multiscale Migration Simulation Workflow Diagram. . Multiscale simulation (yellow boxes) and coupling with Flare (red box) are new models for generating realistic progressions and forecasting how conflicts evolve.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies	Page:	68 of 174
Reference:	D3.5	Dissemination:	PU
	Version:	1.0	Status: Final

4.1.1.3 Coupling and dependencies

In this scenario, we coupled these macroscale and microscale models cyclically in two different ways: using file I/O and using the MUSCLE3 [30] coupling environment. For the microscale model, we incorporated weather factors including precipitation and river discharge datasets. File I/O is a coupling approach to exchange data between two sub-models. By establishing this approach, exchanged data, such as the number of new agents added to each location, can be passed between sub-models using a local shared file system. In this work, the number of all agents passing between sub-models through coupled and ghost locations are stored in the format of CSV files. Both sub-models fill their coupled CSV files, in a parallel fashion, to make sure that both sub-models are synchronized in terms of simulation time steps when all necessary coupled inputs files are checked at the start of each iteration.

To implement our coupling strategy with MUSCLE3, we defined two main compute elements, macro and micro, which represent the macro and micro models, and two manager elements, micro_manager and macro_manager, which handle the inputs from multiple instances of each sub-models. By starting the simulation, each launched sub-model will be registered into the coupling system by MUSCLE3 manager. In this example, 10 concurrent macro and micro sub-model will be executed. Each sub-model instance will simulate the agent's movement between locations on each day. To exchange the data, since we have multiple instances, we designed a manager sub-model to (a) gather data from each instance of the sub-models, (b) combine the founded new Agents per location by each instance into one, and (c) pass to the other model, e.g., macro_manager will collect and combine data from all macro instances, and pass to all micro instances.

4.1.2 Tests

To examine the performance of the HiDALGO software, as well as the aleatoric uncertainty of the coupled migration model, we perform a benchmark test of four different ensemble sizes:

- 100 runs, single core per model.
- 100 runs, 4 cores per model.
- 500 runs, single core per model.
- 500 runs, 4 cores per model.

The results of South Sudan Multiscale simulations with file coupling approach, with and without weather data coupling are presented in Table 16 and Table 16. The average relative difference of the Simulation Approaches for different ensemble sizes.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	69 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

. The former presents the average relative difference of the Simulation Approaches for different ensemble sizes and the second table, presents the total execution time of the Simulation Approaches for different ensemble sizes.

Approaches			100 runs 1 core	100 runs 4 core	500 runs 1 core	500 runs 4 core
Multiscale coupling)	Simulation	(file	0.435	0.435	0.435	0.435
Multiscale coupling) + Weather Coupling	Simulation	(file	0.435	0.436	0.437	0.436

Table 16. The average relative difference of the Simulation Approaches for different ensemble sizes.

Approaches			100 runs 1 core	100 runs 4 core	500 runs 1 core	500 runs 4 core
Multiscale coupling)	Simulation	(file	17091.51	13042.84	16376.90	12513.80
Multiscale coupling) + Weather Coupling	Simulation	(file	65227.17	21361.19	65887.81	22431.62

Table 17: The total execution time of the Simulation Approaches for different ensemble sizes.

4.1.3 Analysis

Table 16 and Table 17 illustrate the results of these simulations on the Altair supercomputer, including the total validation error and the total execution time for the aforementioned tests. Also, we deliberately avoid minimizing the validation error by calibrating existing model parameters against data. Because, it might lead to over-fitting which not only reduces the reusability of our simulations in new contexts, but also makes it highly sensitive to the (often incomplete) validation data sources we use. Therefore, we mostly incorporate data sources as model input, and combine them with our general knowledge and qualitative data about human behaviour.

As results show, the total validation error of both multiscale approaches with or without weather coupling, are equal in all ensemble runs. However, the total execution time of multiscale approaches, in 4 core runs are lower than 1 core runs. Furthermore, in case of weather coupled approaches, there is a significant increase in execution time which can be interpreted as coupling overhead of such approaches in comparison to the former

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies			Page:	70 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0
				Status:	Final

approaches. These results show us that weather coupling in South Sudan did not influence the results.

4.2 Urban Air Pollution

The simulation part of the UAP Pilot is mostly done in a sequential way, as current pollution concentrations do constantly depend on not just the current emission, but also concentration at a previous time frame. There are some uses, however, where commissioning simulations in ensembles are useful. Some examples include (a) sensitivity analyses, (b) parameter validation or (c) wind field generation for model reduction. In this deliverable we investigated the computational cost properties of the sensitivity analysis of the boundary condition of air flow.

4.2.1 Scenario description

4.2.1.1 Goal

The goal of the analysis is to assess how fluctuations in the input data, in this case data for boundary conditions influence the results of our simulation, how it affects pollution levels at validation points, or other points of interest. Atmospheric wind conditions do affect pollution spread; however, even minor changes may have drastic effects locally, which makes these analyses essential. Additionally, our goal is also to assess the validity of the ensemble data, which is obtained from ECMWF with the means of Polytope [31].

4.2.1.2 Model description

In this scenario, the evolution of the pollutant spreading in the city of Győr is analysed for a predefined duration for a chosen date (15th November 2021). Weather boundary conditions are set from ECMWF weather data, and pollution emission is derived from the results of the traffic simulation based on camera data from the same day. For these runs, the same meshes are used as for benchmarking, i.e., the small, middle, and large mesh with 728k, 3.4M and 14M cells, respectively. The usual input set is used to analyse pollution spread in the city and sample pollution concentrations at validation points with additional simulations, where wind field boundary conditions are varied with an ensemble set of weather data.

Sensitivity analysis of the input data, in this case using ECMWF provided meteorological ensemble data is done with ensemble data provided by ECMWF itself. For the current investigation a batch of 50 additional boundary condition variations (ensemble scenarios) are used, in addition to the original boundary condition (normal scenario).

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	71 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

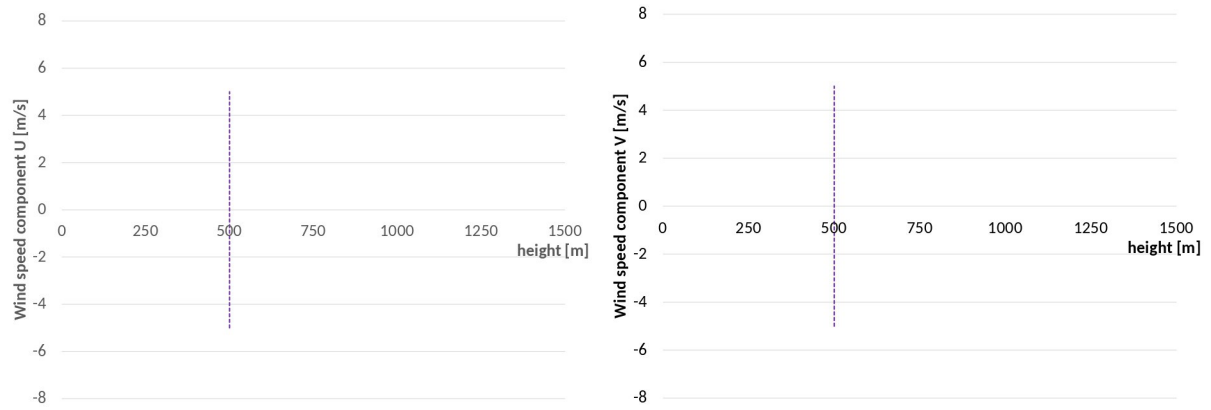


Figure 38. Wind profile components: west-east (left) and north-south (right) depending on height for various ensemble scenarios (grey and black) and the normal scenario (red). The minimum and maximum covering curves for ensemble scenarios are also plotted (green). Also, the top of the simulation domain is also shown at 500 meters (purple).

In Figure 38 we can observe the wind profile behaviour for the normal and ensemble scenarios, including some particular profiles in black from the ensemble for more accurate observation. The height of the model is also plotted at 500 meters, so anything on the right side has no influence on the simulation results.

At the end of the simulation, pollution rates are sampled at data points given by two Bosch sensors (H007 and H0011) and the pollution monitoring station GYOR1.

4.2.1.3 Coupling and dependencies

For data coupling, traffic simulation and weather data acquisition were done separately. Normal and ensemble weather data download via polytope interface and converted to .windxy UAP boundary condition file format for all scenarios separately, yielding 100 files for ensemble and 2 files for normal scenarios. Traffic simulation is done for the 15th November, 2021, and pollution emission is calculated afterwards, which is mapped onto all three 3-dimensional mesh models and stored in the .emi UAP emission rate file format.

For the scenarios investigated, no data exchange between simulations is necessary, data is aggregated after postprocessing.

4.2.2 Tests

To execute the ensemble scenario investigation, a separate code developed to prepare appropriate case directories and separate single core and multi core part of the simulation. The total simulated time in the ensemble scenarios was limited to 1 hour (15th November 2021 from 6AM to 7AM) to limit core hour consumption.

Altogether, the following tests were executed:

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	72 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

- normal run with 3 mesh sizes,
- ensemble runs with low mesh resolution to test the preparatory code,
- ensemble runs with middle mesh resolution for the real investigation.

All simulations were run on the HAWK cluster.

4.2.3 Analysis

Table 18 summarizes runtime and used number of cores used for the simulation. Only the simulation part is shown, data import and domain decomposition, which are single core parts are omitted. The total number of cores for the middle mesh ensemble scenario exceeds 100 000 cores.

Mesh size	# of runs	Runtime Simulation	cores per simulation	total number of cores	core hours
728k	51	172	128	6528	312
3.4M	51	277	2048	104448	8037
14M	1	934	2048	2048	531
			TOTAL:	113024	8880

Table 18. Runtime used cores and core hours for all ensemble and normal runs for various mesh sizes.

The total runtime of the ensemble simulation for middle mesh size remains within 5 minutes, compared to ca. 3.9 hours, if a total of 16 nodes / 2048 cores are used, and 18.1 hours, if only one node is used.

Normal scenario results show good mesh independence for sampling location GYOR1 with lower spread of the ensemble scenario values, too. The sample locations H007 and H011 show poorer mesh independence and higher spread in ensemble scenario values.

4.3 Social Network

The current twitter simulation model relies on multiple parameters, which can heavily influence the accuracy of simulation run. Values for these parameters can either be set manually or learned from a given data set. To learn these parameters, it is beneficial to commission an ensemble of simulation runs to quickly test various different parameter values in parallel. In this section, we investigate the scalability and computational cost of such a learning approach.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies			Page:	73 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0
				Status:	Final

4.3.1 Scenario description

4.3.1.1 Goal

The current twitter simulation model relies on 6 parameters (3 continuous, 2 discrete, 1 binary) that currently have to be chosen independently for each feature vector. Our efforts focus on tuning the 3 continuous parameters (i.e. `propagation_probability`, `discount_factor`, and `correlation_probability`) minimizing the error of the retweet probability and the mean number of retweets. In the currently considered configuration of the binary parameter, the `propagation_probability` can be directly calculated from input data. Therefore, it only remains to find optimal `discount_factor` and `correlation_probability` values that minimize the error of the mean number of retweets.

The goal is thus to optimize a stochastic function over a multidimensional search space. Methods for stochastic optimization have most recently been explored in the context of hyperparameter tuning. The most basic approach here is a grid search, that is an exhaustive search over some chosen (uniform) discretization of the search space. This approach is visualized in Figure 39, where we divide the 2-dimensional search space into a 9x9 grid. Each point corresponds to an assignment of the `discount_factor` and `correlation_probability` parameters to certain values. With each such selection of parameter values, a simulation is run to assess their performance with respect to the error in the mean number of retweets (compared to our ground-truth data).

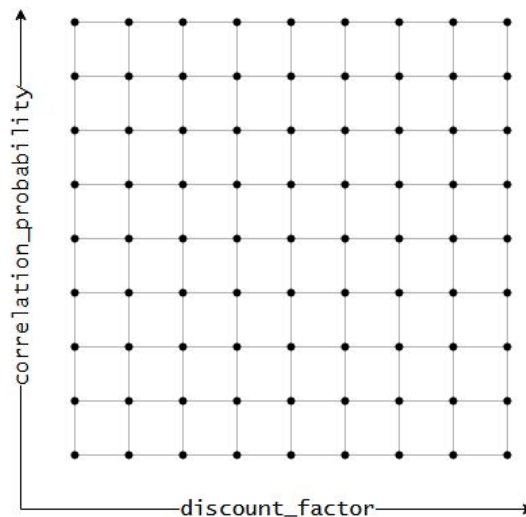


Figure 39. Grid search example. The search space is partitioned into a 9x9 grid and at each point a pair of parameters is evaluated.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	74 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

Choosing this approach has three benefits:

1. We gain valuable insight in the global behaviour of the error function and thus our model.
2. The best points of such grid can be used as starting points of a subsequent much finer grained local search. One can then reasonably expect the resulting points to be very close the global optimum.
3. The algorithm lends itself to massive parallelism as multiple points can be evaluated independently in parallel.

4.3.1.2 Model description

We search a 20x20 grid for each of the 132 feature vectors of the fpoe_20201110 dataset (see Section 3.3.1.1 for a description of this data set). The corresponding follower graph we use as input has 15k nodes and 63k edges. Each point of the grid search is evaluated using 400 sources with 1000 samples each. Therefore, there are $20 \times 20 \times 132 \times 400 \times 1000 = 21.12 \times 10^9$ simulations to be run in total.

4.3.1.3 Coupling and dependencies

As input data we use the follower-followee relationship graph and the ground truth retweet data of the fpoe dataset that were crawled on twitter and can be found on the CKAN. This data is read on a single head node that then also orchestrates the simulation and aggregates the results. The simulations required for each data point can be performed independently from the simulations of other data points.

4.3.2 Tests

We performed the ensemble run on the SuperMUC-NG supercomputer and repeated it 4 times with an increasing amount of cores times to assess the scalability of our approach. Due to the high amount of cores used (more than 98k) each run was performed only once.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	75 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

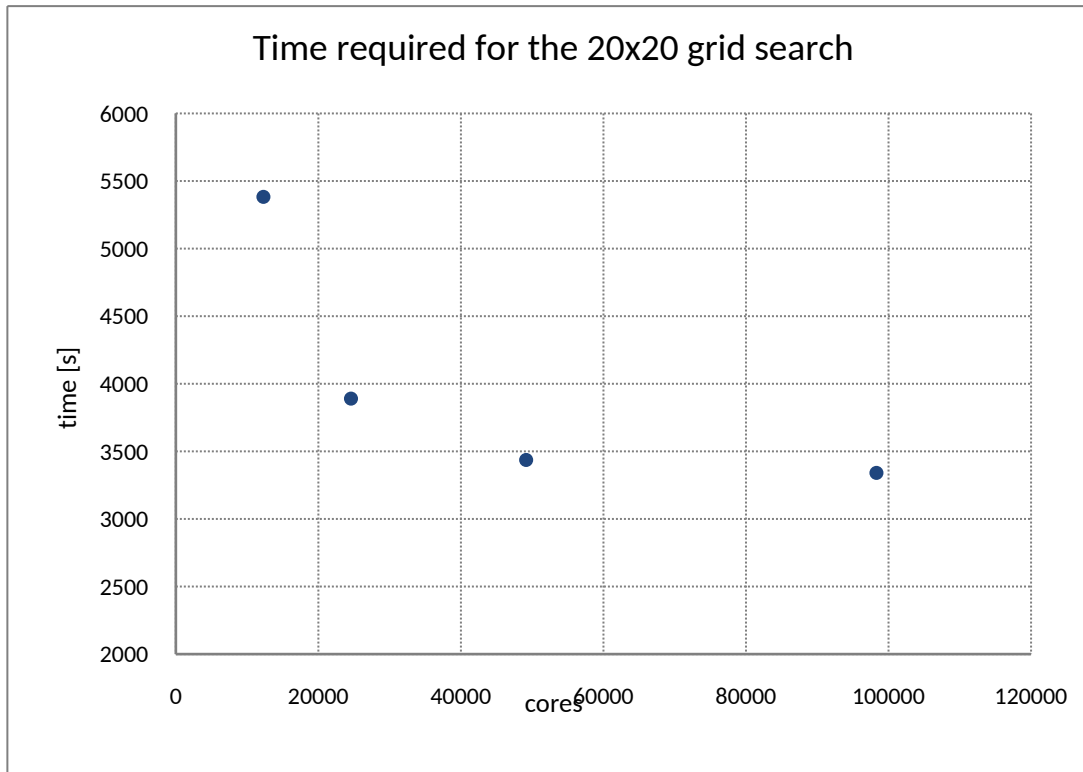


Figure 40. Scalability of the grid search approach for the fpoe dataset on SuperMUC-NG.

In order to quickly assess the quality of the obtained parameters values for `discount_factor` and `correlation_probability`, we compare them to the behaviour of the default parameters (`discount_factor = 1` and `correlation_probability = 0`). For the obtained parameter values of the grid search as well as default experiment, we performed a simulation run with the `fpoe_20201110` data set (400 sources and 1000 samples) to measure the mean absolute percentage error (MAPE) in the number of retweets (compared to the ground truth data). The results are depicted in Figure 41. A more thorough analysis of the obtained parameters, including a comparison with additional methods for learning parameters, is reported in Section 6.2 of Deliverable 4.4.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies			Page:	76 of 174		
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status:	Final

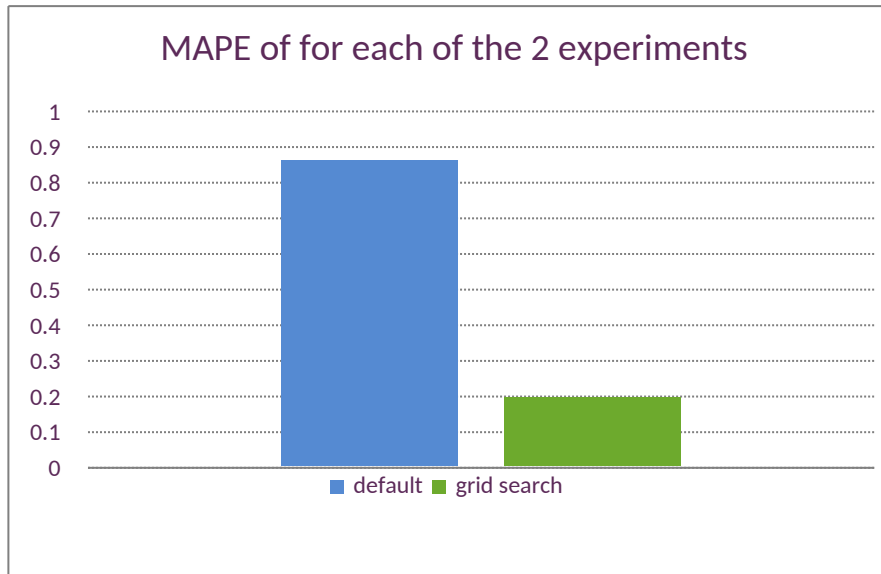


Figure 41. MAPE of the fpoe data set for each of the 3 experiments.

4.3.3 Analysis

We start with a discussion about the scalability of our approach. Going from roughly 12k to 24k cores we can still observe a speed up approximately factor 1.38. However, the benefit of adding additional cores after roughly 40k cores is limited: the running time improves only slightly when increasing the cores from 49152 to 98304. We suspect that this phenomenon could be avoided by running the search on an even larger (i.e. more fine-grained) grid. While originally this was our intention, we encountered a bottle-neck with our current implementation that uses a single root process that dispatches tasks to workers. In particular, scenarios larger than the above exhausted the limited main memory on the root node. In the future, this problem could be circumvented by employing an approach in which multiple root nodes are used to dispatch tasks and collect results from the workers.

When it comes to the quality of the obtained parameters, we can see in Figure 40 that the grid search yields similar results as the binary search approach. Both of these approaches yielded a MAPE in the number of retweets of below 20%. There is also an additional benefit of the grid search is not visible in the chart above: we obtained multiple parameter values close to the maximum. We are currently working on an implementation of a local searching approach that allows us to these parameter values to find an improved global optimum.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	77 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

5 Analysis and optimization

This section thoroughly analyses pilots' applications and provides indications to improve their efficiency.

Data repository folder:

https://gitlab.com/eu_hidalgo/benchmarking/-/tree/master/deliverable_3_5/optimization

5.1 Migration Pilot

5.1.1 Goal

The primary goal of this investigation is to improve the overall performance of the Flee application. To reach this goal, we propose to use the Numba [32] python library to optimize hotspots in the application and overcome the limits of the standard python interpreter. The proposed approach is examined using real data from the South Sudan (ssudan) conflict. This study explores a variety of modern parallel architectures, including a single computing node with two processors based on x86-64 and ARM-based architectures. Table 19 and Table 20 outline explored computing platforms and software environments, respectively.

Type of CPU	Codename	Cores	Freq. [GHz]	Memory
2x AMD Epyc 7763	Milan	2x 64	2.45	2x8x32GiB DDR4-3200
2x AMD Epyc 7742	Rome	2x 64	2.25	2x8x32GiB DDR4-3200
2x Intel Xeon Platinum 8268	Cascade Lake	2x 24	2.9	2x6x16GiB DDR4-2933
2x Huawei Kunpeng 920	TaiShan (ARMv8.2)	2x 48	2.6	2x8x32GiB DDR4-2933

Table 19. Specification of testing platforms.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies			Page:	78 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0
				Status:	Final

Software / Package	All Platforms
Python	3.8.11
Numba	0.53.1
Numpy	1.20.3
Pandas	1.3.1
Mpi4py	3.0.3
MPI Library	OpenMPI 4.0.2

Table 20. Software stack for SN Simulator and KPM benchmarks.

5.1.2 Performance analysis and the Numba-based optimization

To employ the Numba-based modification, we distinguish the main hotspots of the flee code. The performed analysis indicates four computing Kernels that jointly consume more than 70% of the total execution time. These Kernels refer to the most notable functions of the flee application and are mainly associated with generating random values with probability support.

More precisely, Kernels 1, 2, and 3 are part of the `evolve` function responsible for calculating the probability that a moving agent will select a given route and making the probabilistic route selection. In this case, the 1st Kernel determines the weights of each adjacent link, the 2nd Kernel normalizes the weights, while the 3rd Kernel returns the eventual movement decision made. The last 4th Kernel is included in the `addAgentTime` function and selects the conflict zone where a given newly spawned agent emerges, based on a population-weighted probability function.

All of these Kernels are provided with the usage of the NumPy python library, which is eligible for performance improvements by Numba. The selected Kernels are marked as Numba functions and translated into machine codes during application execution (just-in-time compilation). As a result, every Kernel compiled once is further used tens of millions of times in a typical simulation run. However, while the compilation process accomplishes for Kernels 1-3, we observe some compilation issues for the last 4th Kernel. We reveal that the current version of the Numba python library does not support generating a set of random samples from the given array with population-weighted probability, required by the last Kernel. In consequence, we are not able to employ Numba for Kernel 4.

Figure 42 illustrates an evaluation of the proposed approach using different computing platforms and simulating migration for South Sudan (ssudan) conflict. As shown in Figure 42a,

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	79 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

enabling Numba-based modification for the basic version of code reduces the total execution time and improves the overall performance achieving the speedup in a range from 1.4x to 1.8x. The applied analysis also indicates a significant performance improvement for Kernel 2 and Kernel 3 (see Figure 42b). In this case, employing Numba leads to about 10x and up to 23x faster execution of the Kernel 2 and Kernel 3, respectively. In contrast, Kernel 1 with enabled Numba accelerates computations of about 1.1x for all performed tests. At the same time, Figure 42c shows a percentage of the total execution time measured separately for every Kernel and the aggregate percentage obtained for all selected Kernels.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	80 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

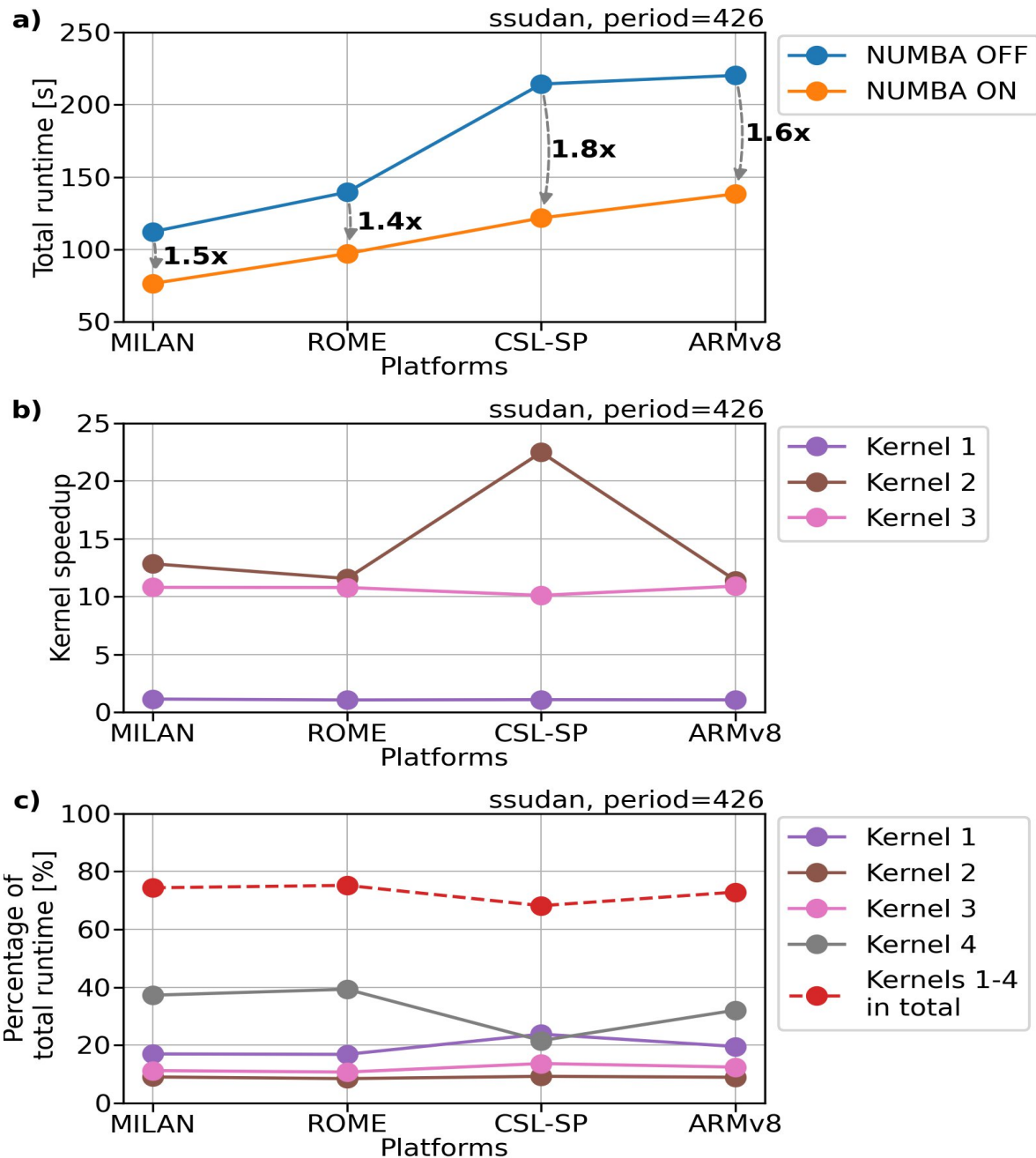


Figure 42. Evaluation of the Numba-based optimization applied for the flee application: a) performance comparison for studied application with enabled and disabled Numba optimizations, b) Partial performance gain measured for a given Kernel separately, and c) the percentage of total execution time for selected measured for the basic version of the application.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies	Page:	81 of 174	
Reference:	D3.5	Dissemination:	PU	
	Version:	1.0	Status:	Final

5.2 Urban Air Pollution Pilot

5.2.1 Goal

The primary goal of this investigation is to improve the overall performance of the UAP solvers by means of optimization the OpenFOAM application and utilization of GPGPU accelerators. The pollution dispersion module is the most computationally intensive part of the UAP pilot. Real world use cases range from simulating 1 day up to 1-year real time with street level resolution from 4 to 1 meter. Estimating runtime using chapter 3 results for simulating a 1-year scenario on HAWK would take 5.6 days for the low mesh size with 4 nodes, 22.7 days for the middle mesh size with 16 nodes and 98.6 days for the high mesh size with 256 nodes. Reducing these simulation times is critical to get results in a reasonable time frame.

5.2.2 Analysis and conclusions

5.2.2.1 Excellerat project collaboration

The EXCELLERAT project [33] is a single point of access for expertise on how data management, data analytics, visualisation, simulation-driven design, and Co-design with high-performance computing (HPC) can benefit engineering, especially in the aeronautics, automotive, energy and manufacturing sectors. The goal of EXCELLERAT is to enable the European engineering industry to advance towards Exascale technologies and to create a single-entry point to services and knowledge for all stakeholders of HPC for engineering.

HiDALGO developed the Urban Air Pollution Pilot (UAP) and was supported by the EXCELLERAT team during the pilot's application improvement phase. The primary goal of the collaboration was to improve UAP's performance on HPC systems, that uses OpenFOAM in its CFD module for pollution dispersion simulation. This included three components: On the first level, the team of both HiDALGO and EXCELLERAT experts aimed to improve the mesh quality with `snappyHexMesh`. The second part of the collaboration dealt with the optimisation of the steady state simulation using `simpleFoam` and then the third, improving transient simulation with `pimpleFoam`.

The team used `snappyHexMesh` for meshing instead of `octree mesher`. Advanced pre-processing enabled finer surface and edge fit. Also, higher quality mesh comes with lower generation time for same resolution.

For the steady state simulation with `simpleFoam` wide range of numerical schemes were investigated for stability, runtime, and parallel performance. Unnecessary IO was turned off,

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	82 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

and mesh cell index renumbering and multilevel decomposition was issued and incorporated into the automatic workflow. Improvements were most significant at lower mesh sizes.

EXCELLERAT proved to be an invaluable partner in helping and consulting HiDALGO for parallel performance optimisation. The issued optimisations proved to have a factor 5 improvement of the speedup on our cluster, from 18 to 102 for a 1 million cell mesh. Larger meshes of almost 10 million cells also had a speedup improvement from 49 to 77. This greatly improved parallel efficiency, from around 8% to more than 47%. This success was efficiently achieved within a few months thanks to the excellent exchange of knowledge and cooperation. Previously we had several performance issues, as development was focused on feature integration and automatization.

5.2.2.2 GPU solvers

The solvers were tested on the following GPUs:

- NVIDIA Tesla V100,
- AMD MI50.

More information on above architectures can be found in deliverables D5.5 [34] and D5.8 [6].

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	83 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

5.2.2.2.1 RapidCFD

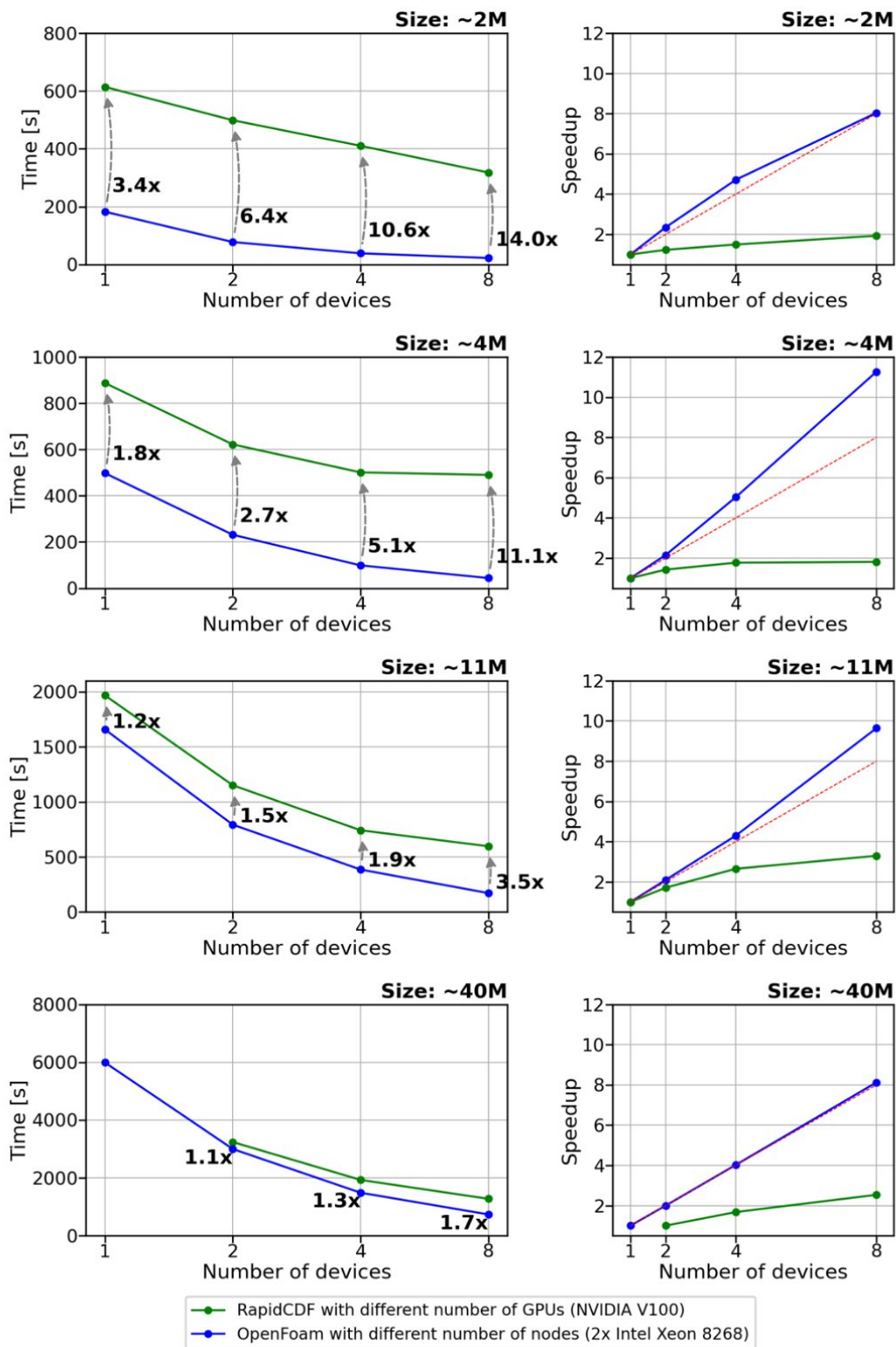


Figure 43. Performance comparison between RapidCFD and OpenFOAM obtained for different numbers of devices and a variety of domain sizes: a) Computation time [s] (left) and b) Strong scaling speedup (right).

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies	Page:	84 of 174
Reference:	D3.5	Dissemination:	PU
	Version:	1.0	Status: Final

To assess the GPU capabilities of OpenFOAM variants, 3 possibilities were considered: coupling OpenFOAM with PETSc using `petsc4Foam`, `foamextend` version 4.1 and RapidCFD [35]. Both `petsc4Foam` and `foamextend` use GPU-s only in the case of solving the discretized problem and with RapidCFD all calculations are done on the GPU. This makes RapidCFD the best candidate to port the UAP pilot to.

Before porting the pilot, preliminary benchmarks were issued to assess the performance of the code. The latest version of RapidCFD <version from May 13th, 2020> was compared to the compatible OpenFOAM version 2.3.1. The tutorial case `motorbike` was chosen for the comparison, as it uses one of the tools also used in UAP, `simpleFoam`.

To assess performance meshes of various sizes were generated from 300k to 40M. Results for all, but the smallest mesh size is presented on Figure 43. For this benchmarks NVIDIA V100 GPUs were used and compared to dual socket Intel Xeon 8268 CPU nodes. The comparison was made for 1 GPU – 1 node, scaled up to 8 GPUs – 8 nodes.

5.2.2.2.2 Fluid solver

Fluid-Solver is a 3-dimensional computational fluid dynamics solver, based on a finite volume method that runs either in full order mode (FOM) or reduced order mode (ROM). Reduced-Order-Modeling is based on the proper orthogonal decomposition method (POD), which has two major steps: an offline phase consisting of a snapshot collection and singular value decomposition (SVD) of the snapshot matrix, and an online phase for the ROM-simulation. Fluid-Solver has been implemented for multicore computers: CPU nodes (using OpenMP) and NVIDIA GPGPUs, of which the implementation framework is CUDA [36]. Linear algebra algorithms are done with cuBLAS. Using CUDA with OpenGL, fluid-solver has a mode where results can be visualized in real time.

For benchmarking purposes, we have chosen the Antwerp May 6, 2016, use case from FAIRMODE's Intercomparison Exercise (for more details, see D4.4). The mesh size was chosen 440.000, which results in 2.2M number of freedom (i.e. the number of floating point numbers to be computed) in each time step. We have tested the performance on different architectures.

Since our solver is `tetrahedral`-based, and uses the **finite-volume** method, we can break down our algorithm into several categories: `vertex`, `face`, and `cell-iterative` algorithms. Since we are using an `explicit-Euler` based approach, the current global algorithm for the solver essentially consists of computing flux values by iterating through all vertices/faces/cells (we need to iterate through vertices as well, since we're using a 2nd order

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	85 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

spatial method). Our flux value is essentially 5 variables that we keep track of: a density (1 component), velocity (3 components), and energy (1 component).

During the very first optimization stages, we had to consider one of the most fundamental aspects of our code; storing flux values. Two options arise: either SOA-s (Structure of Arrays), or AOS-s (Array of Structures). This proved to be an interesting decision, since this would have a huge effect on cache misses (ex. L2); for the solver, we ended up using AOS-s, since when accessing one flux variable (ex. velocity), we most definitely need access to the others (ex. density, energy). However, this isn't always true; in some pre-processing stages, and with visualization scripts especially, we often need to iterate per variable, while ignoring the others. In these cases, we opted for using AOS-s, that is, storing each flux variable as its own array (a separate array for the densities, velocities, energies).

To optimize the CPU version, we ended up using OpenMP. We didn't directly use any SIMD instructions, since the compiler generated `assembly_code` seemed to generate fairly optimal results. It should be noted however, that we do plan on manually implementing some routines in SIMD in the future, for further non-negligible performance improvements.

For the GPU version, we ended up using CUDA, and wrote all vertex/face/cell functions as CUDA kernels. The only library we ended up using was cuBLAS, for matrix-vector multiplication. We did initially use some minimum reduction-related functions from the CUB library, from the CUDA SDK; but we ended up implementing our own code. To optimally evaluate the performance of our CUDA kernels, and further optimize our code, we used NVIDIA's NSight Compute software.

We present in the results below, our benchmark for Antwerp (Figure 44). In this case, a wind-field was computed every iteration, alongside the density and energy fields. For the FOM (Full-Order-Model) results concerning the CPU runs on HAWK and SOLYOM, we only ran 50 seconds and scaled the results, since we couldn't simulate for such a long amount of time. (Since we're using the explicit-Euler method, with mostly constant tau steps, it is fair to assume all our results scale linearly).

There are two parameters of interest in our benchmark; the dimension of the reduced-order model r ; and the CFL (Courant-Friedrichs-Lewy) number. Since we are using the POD, we can raise the CFL value much higher than usual (for our FOM, the maximal value for a stable run is roughly 0.8).

As you can see in the results below, when running in FOM mode (Full-Order-Model), the results on an NVIDIA A100 GPU are, roughly speaking, 27x faster than on an AMD7742 CPU, and 12x faster than an Intel Xeon Gold 4 x 6230 CPU. When considering the reduced order model, results on the NVIDIA A100 GPU are roughly 43000x faster than the FOM on an AMD7742 CPU.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	86 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

We provide the benchmark results below, alongside the following list of hardware we benchmarked on:

- **HAWK:** 1 CPU None on HAWK Cluster
(AMD7742 256GB 128 cores)
- **SOLYOM Compute:** 1 CPU Compute Node on SOLYOM Cluster
(2 X Intel(R) Xeon(R) Gold 6226R CPU @ 2.90GHz, 32 cores, 384GB)
- **SOLYOM Fat:** Fat note on SOLYOM Cluster
(4 X Intel(R) Xeon(R) Gold 6230 CPU @ 2.10GHz, 80 cores, 1,5TB)
- **ALTAIR GPU V100:** NVIDIA TESLA V100
- **SOLYOM GPU V100:** NVIDIA Tesla V100S-PCIE-32GB
- **SOLYOM GPU A100:** NVIDIA A100-PCIE-40GB

Architecture Name	FOM	ROM (r=10, CFL=1000)	ROM (r=10, CFL=2000)	REAL-TIME
HAWK	820109	2842	1500	86400
SOLYOM COMPUTE	617534	758	382	86400
SOLYOM FAT	387205	519	277	86400
ALTAIR GPU V100	86486	82	41	86400
SOLYOM GPU V100	71036	78	39	86400
SOLYOM GPU A100	30221	38	19	86400

Table 21. Runtime for Antwerp model of 434k cells for different models on various architectures.

Architecture Name	FOM	ROM (r=10, CFL=1000)	ROM (r=10, CFL=2000)	REAL-TIME
HAWK	0.1054	30.3974	57.6060	86400
SOLYOM COMPUTE	0.1399	113.9705	226.1215	86400
SOLYOM FAT	0.2231	166.4945	311.4223	86400
ALTAIR GPU V100	0.9990	1054.9291	2108.6370	86400
SOLYOM GPU V100	1.2163	1114.3455	2224.6622	86400
SOLYOM GPU A100	2.8589	2256.3833	4482.3621	86400

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies			Page:	87 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0
				Status:	Final

Table 22. Speedup w.r.t. real-time (real-time divided by runtime) for Antwerp model of 434k cells for different models on various architectures.

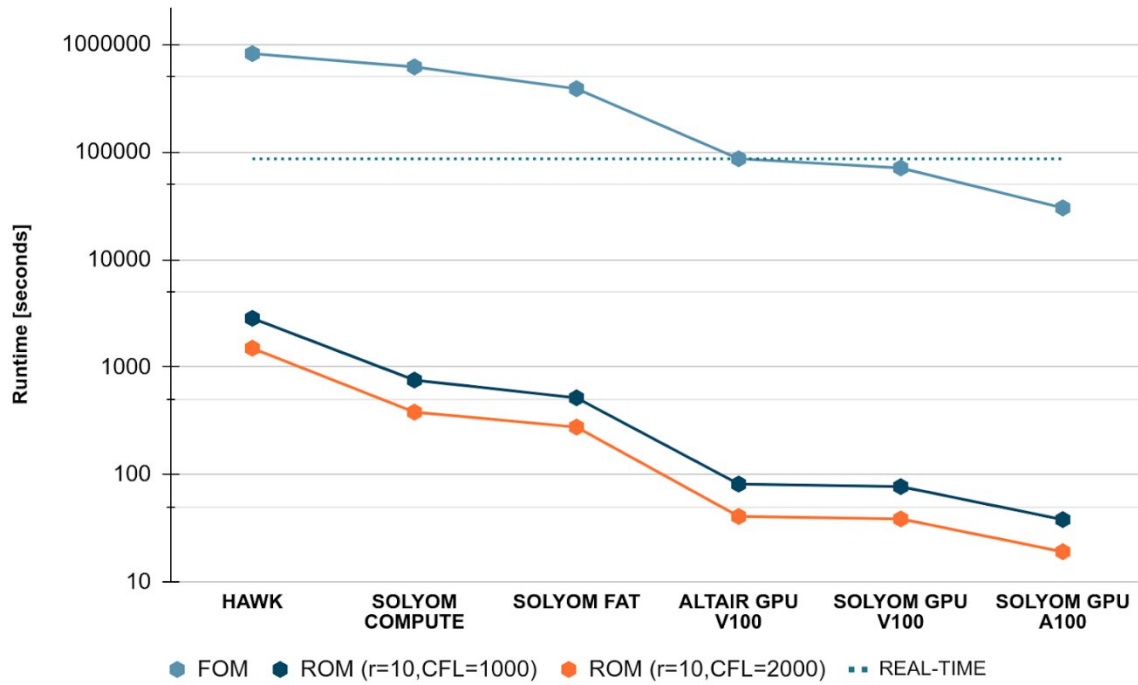


Figure 44. Runtime for Antwerp model of 434k cells for different models on various architectures.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	88 of 174	
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status:	Final

5.3 Social Networks Pilot

5.3.1 Goal

One of the essential goals of our activity is a better utilization of computer HPC resources. We search for the trade-off and correlation between a given application and the computing hardware to achieve this goal. We intend our work to reduce performance and memory bottlenecks to help overcome compute and memory limitations and improve the overall performance. We tackle a wide range of profiling scenarios and a deep code analysis to reach these goals.

In this investigation, the analysis and optimizations are performed for two applications, including the SN simulator and the KPM application (EigHist eigenvalue approach). This activity addresses a variety of modern parallel architectures, including typically a single computing node based on x86-64 and ARM-based architectures. Table 23 and Table 24 outline explored computing platforms and software environments, respectively.

Type of CPU	Codename	Cores	Freq. [GHz]	Memory
2x AMD Epyc 7763	Milan	2x 64	2.45	2x8x32GiB DDR4-3200
2x AMD Epyc 7742	Rome	2x 64	2.25	2x8x32GiB DDR4-3200
2x Intel Xeon Platinum 8360Y	Ice Lake	2x 36	2.4	2x8x32GiB DDR4-3200
2x Intel Xeon Platinum 8268	Cascade Lake	2x 24	2.9	2x6x16GiB DDR4-2933
2x Huawei Kunpeng 920	TaiShan (ARMv8.2)	2x 48	2.6	2x8x32GiB DDR4-2933

Table 23. Specification of testing platforms.

Software / Package	All Platforms
Python	3.9.6
Numba	0.53.1

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies			Page:	89 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0
				Status:	Final

Software / Package	All Platforms
Numpy	1.21.1
Scipy	1.7.1
Pandas	1.3.2
Mpi4py	3.1.1
MPI Library	OpenMPI 4.1.1

Table 24. Software setups for SN Simulator and KPM benchmarks.

5.3.2 Social Network Simulator

The Social Network (SN) Simulator is parallelized using the MPI standard for Python, commonly known as the mpi4py module. This module provides an object-oriented interface resembling the message passing interface (MPI), allowing Python programs to exploit multiple processors on multiple compute nodes.

Generally, the SN Simulator consists of two major stages. The first stage is responsible for application initialization and is mainly focused on propagating input data across all computing resources. In this stage, a single MPI process called master reads all required data from the files and then replicates them between used computing nodes. Simultaneously, a single MPI process assigned to each computing node received the input data necessary to share them between other MPI processes assigned to a given node. Figure 45 illustrates the general execution schema of the SN Simulator application. Even though the first stage executes mainly sequentially, it does not limit the overall performance, and the measured cost is negligible for all performed tests and application scenarios.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	90 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

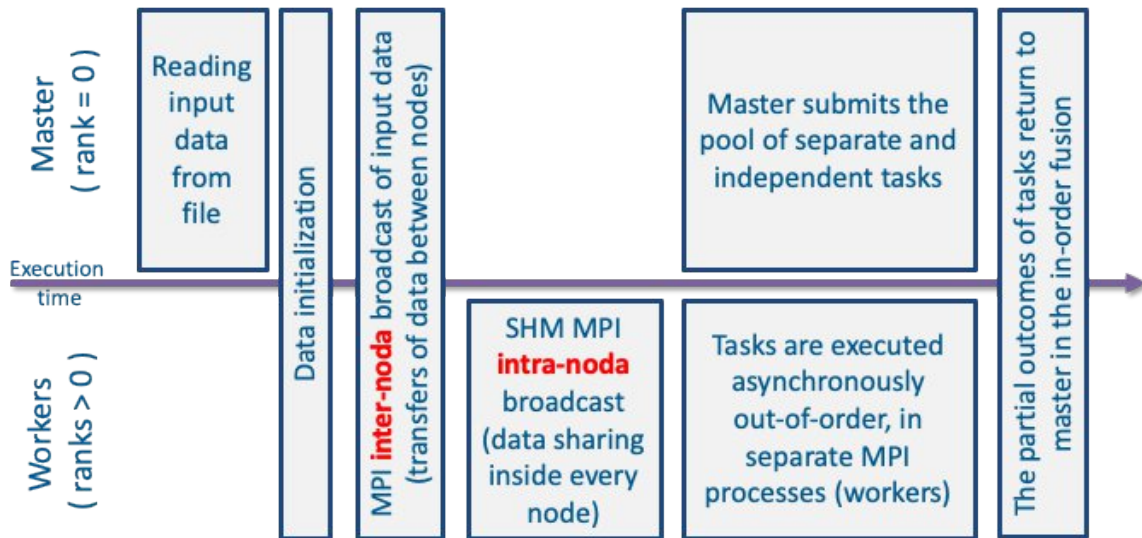


Figure 45. General execution schema for SN Simulator.

The second - most-time consuming - stage of SN Simulator application corresponds to the parallel computation performed according to the task-based master/worker approach offered by the *mpi4py.futures* package. This parallelization strategy enables asynchronously executing tasks on a pool of MPI processes called workers. Generally, the master process (i) generates a queue of independent tasks, (ii) hands out simulations tasks to worker processes, and then (iii) waits for the partial outcomes. In particular, for every feature vector under consideration, multiple users are sampled, and for each user, a task to run numerous simulations is submitted round-robin to the workers. Simultaneously, all workers receive an incoming series of tasks, execute them one by one, and send back the partial outcomes to the master. The master process further aggregates the returned partial outcomes to produce the final results of the simulation. It should be noted here that MPI processes (master and workers) are pinned to cores to avoid migration of tasks between cores.

In the basic version of the code, all workers process a similar number of tasks. However, we observe that some workers finished processing tasks earlier than others. To explain this behaviour, we have to look at the costs of all tasks individually.

The left side of Figure 46 shows an example of performance analysis for workload/tasks distribution between workers, obtained for the basic version of the code. In this case, Figure 46a illustrates (i) the aggregate time of tasks performed on every MPI worker marked as a red line, and (ii) the total execution time of simulation marked as a blue line. Figure 46b outlines the number of tasks tackled by a given worker, while Figure 46c traces a detailed cost analysis for tasks assigned to the worker with the slowest execution time.

The performed analysis reveals large load imbalancing between MPI workers. In the illustrated example, the worker with the shortest execution time finishes simulation of all tasks more

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	91 of 174	
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status:	Final

than 3x quicker than the worker with the slowest execution time even though all workers process a similar number of tasks. During profiling the cost of tasks of every worker, we reveal that during the final part of the simulation, every worker usually processes a small pool of very long-running tasks. This behaviour is illustrated in Figure 47a, which shows a detailed analysis of the costs of tasks performed for all workers. Consequently, we observe that it leads to large load imbalancing between workers and limits the overall performance.

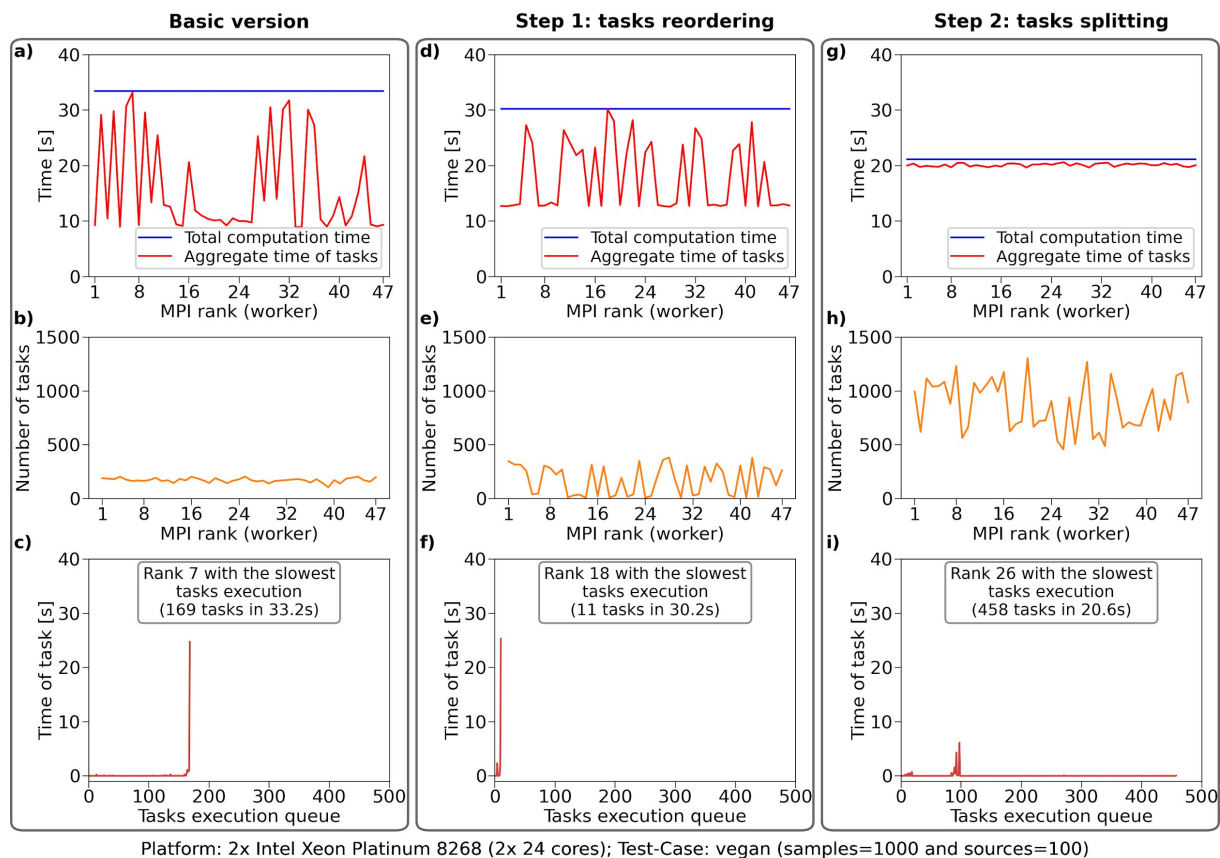


Figure 46. Performance analysis of different versions of SN Simulator.

To alleviate the load imbalancing and improve the overall performance, we propose the 2-step procedure for code optimizations. While we were puzzled as to why the tasks at the end were running longer, at the first step, we propose to modify the order of tasks execution to move the costliest tasks at the beginning of application execution.

More precisely, we add higher execution priority for the tasks that correspond to the simulation of messages which are predicted to be more likely to be retweeted frequently and - in consequence - require more computation. To achieve this goal, we propose to sort the feature vectors heuristically according to their expected runtime. Since the runtime of a simulation is dependent on the number of retweets and the target mean number of retweets

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies	Page:	92 of 174
Reference:	D3.5	Dissemination:	PU
		Version:	1.0
		Status:	Final

is known for each feature vector, this target number serves as a good heuristic assuming our simulation parameters are aligned with this goal.

During the analysis, we then also found the reason for the long-running tasks. The feature vectors were originally sorted lexicographically, and the first feature (i.e. the most significant bit) corresponded to the verification status of the user. Since verified users are generally more connected, this feature is the highest single feature predictor of many retweets. Thus, the default ordering of the feature vectors offered by the basic version is indeed one of the worst possible orderings.

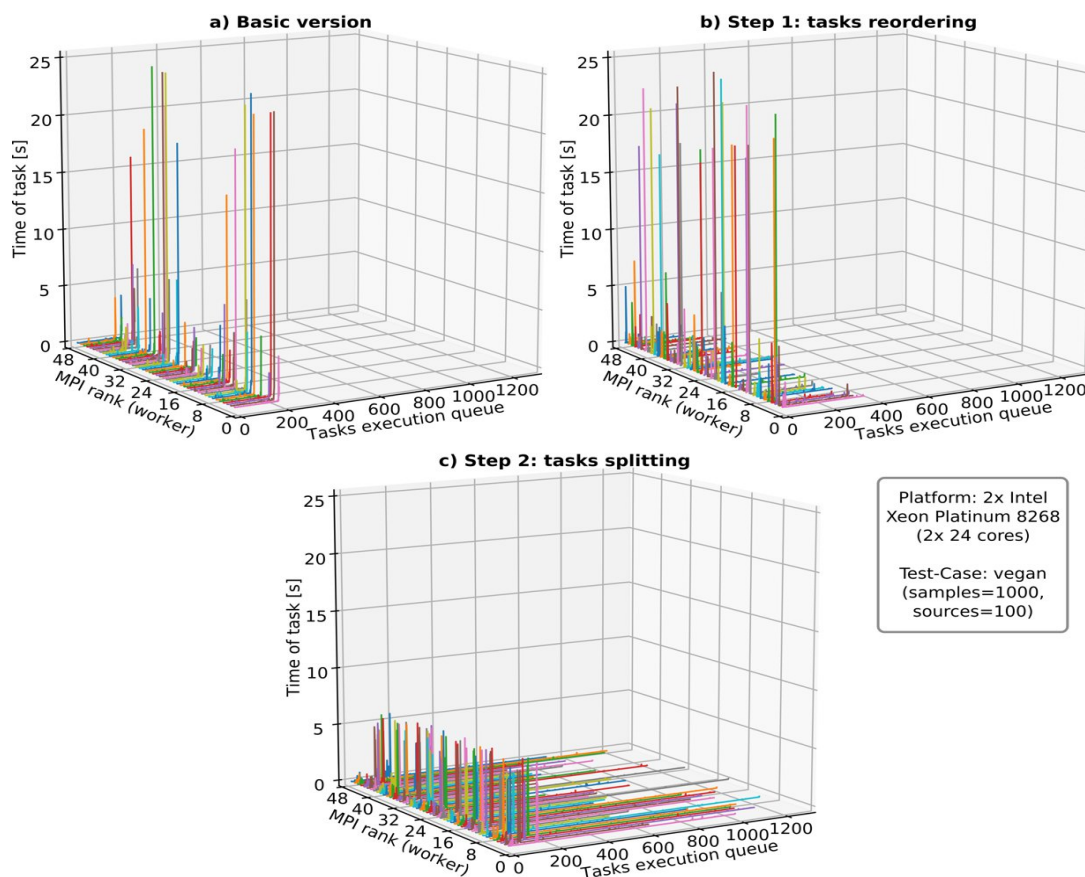


Figure 47. Workload distributions between MPI workers obtained for different versions of SN Simulator.

Figure 46d-f and Figure 47b demonstrate an example of how the proposed modification of the new task execution order affects load balancing, tasks distribution, and overall performance. We investigate the proposed approach using different computing platforms and application scenarios that mimic the flow of messages/tweets on twitter about a certain topic at a certain time, including four datasets *neos*, *covid19_socialdistance*, *vegan*, and *fpoe*.

The performed test reveals that the proposed method reduces the load imbalance and improves the overall performance. The summary of the performed examination for the first

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies	Page:	93 of 174
Reference:	D3.5	Dissemination:	PU
		Version:	1.0
		Status:	Final

step of the proposed modification is delivered in Figure 48. This figure illustrates the achieved performance gain for SN Simulator with the new order of tasks execution. The proposed approach leads to sustained speedup due to a more uniform utilization of the workers, and it improves the overall performance up to 2.07x.

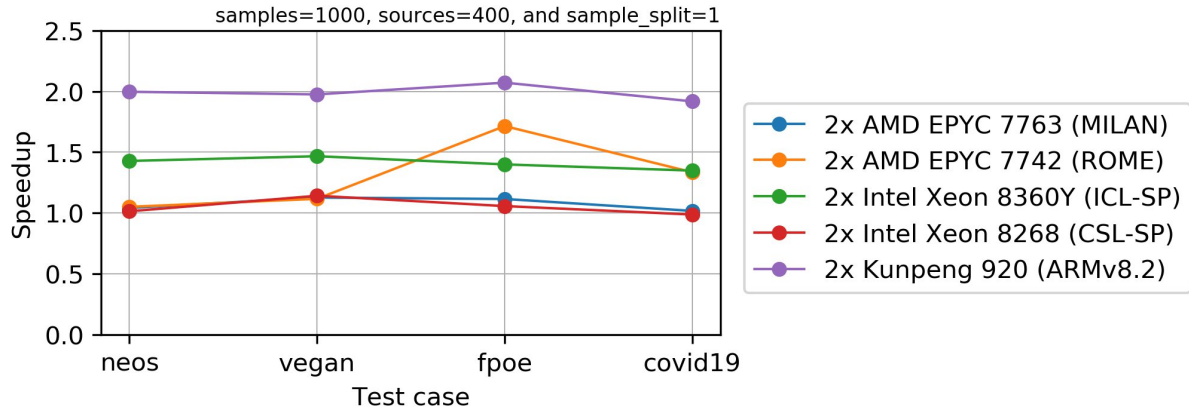


Figure 48. Performance gain obtained for SN Simulator with a proposed order of tasks execution in comparison to the basic version.

The performance analysis also reveals significant differences in the costs of tasks (see Figure 46a and Figure 46b). We observe that even a relatively small number of heavy tasks per worker can cause a significant load imbalance between MPI workers and strongly limit the overall performance. The key to understanding this behaviour is analysing the tasks construction process.

In the basic version of the code, the SN Simulator simulates the propagation of retweets in a Twitter follower graph. In essence, two parameters determine the number of tweets that are simulated: *sources* determines the number of tweet authors of which tweets originate, and *samples* denotes the number of tweets that are initiated from each of these authors. All of the samples of many simulated tweets that are initiated from a fixed author were considered an atomic task. That is, in all of these samples many simulations were executed by the same MPI worker process on a single core. As some authors are prone to generate tweets that are likely to be retweeted multiple times, this caused some of these tasks to take way longer to be processed than others. Therefore, it is harder to balance the load among workers. The performed analysis reveals that reducing the task size can lead to better CPU utilization.

To tackle this problem, we propose to split tasks into smaller subtasks by reducing the number of tweets simulated per task. To achieve this aim, we add a new parameter called `sample_split` that helps us partition each task into `sample_split` many smaller independent sub-tasks. These sub-tasks do not need to be placed on the same processor, and therefore we can now distribute long tasks among multiple cores. However, this approach

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies			Page:	94 of 174	
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

allows us to manipulate the task size at the cost of additional communication overhead when distributing tasks and collecting results.

Experimentally, we validate and examine the proposed approach, testing different sizes of tasks for various application scenarios and a wide range of computing platforms. We perform nine improvements and present the minimum, median, and maximum computation times for every tested application configuration. Figure 49 demonstrates three examples of the correlation between different task sizes and overall performance obtained for parameter *sources* configured as 100, 200, and 400.

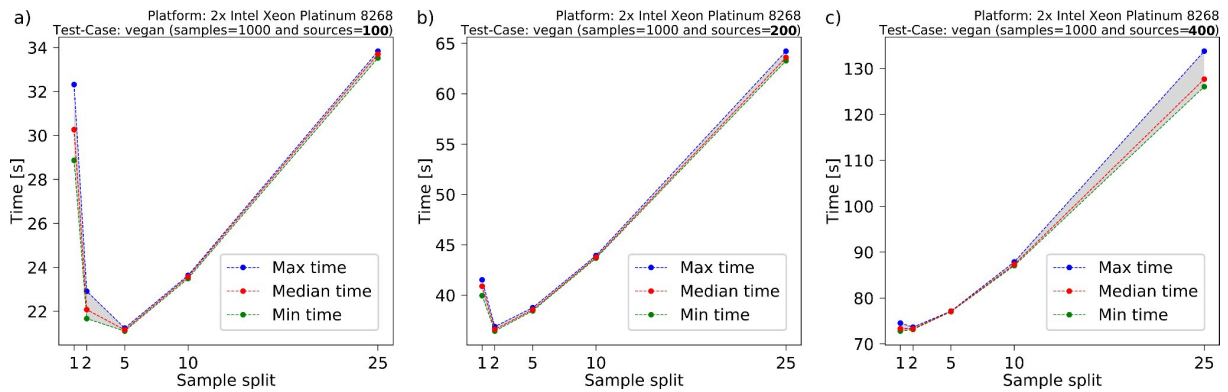


Figure 49. Impact of workload (task) size on performance.

We observe that selecting the optimal value of task size is strongly correlated with the parameter *sources* for all performed tests. The performed analysis reveals that the proposed modification of task splitting brings noticeable performance improvement for relatively small values for source parameter, as tested configurations where source equals 100 or 200 (see Figure 49a and Figure 49b). In contrast, the performance gain is negligible for larger source parameter values, as in the tested case where the sources parameter is set to 400 (see Figure 49c). The whole summary of this investigation is outlined in Figure 50, demonstrating the final gains (left side of figure) and selected optimal sample split parameter (right side of figure) obtained for different platforms and applications scenarios.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies	Page:	95 of 174
Reference:	D3.5	Dissemination:	PU
		Version:	1.0
		Status:	Final

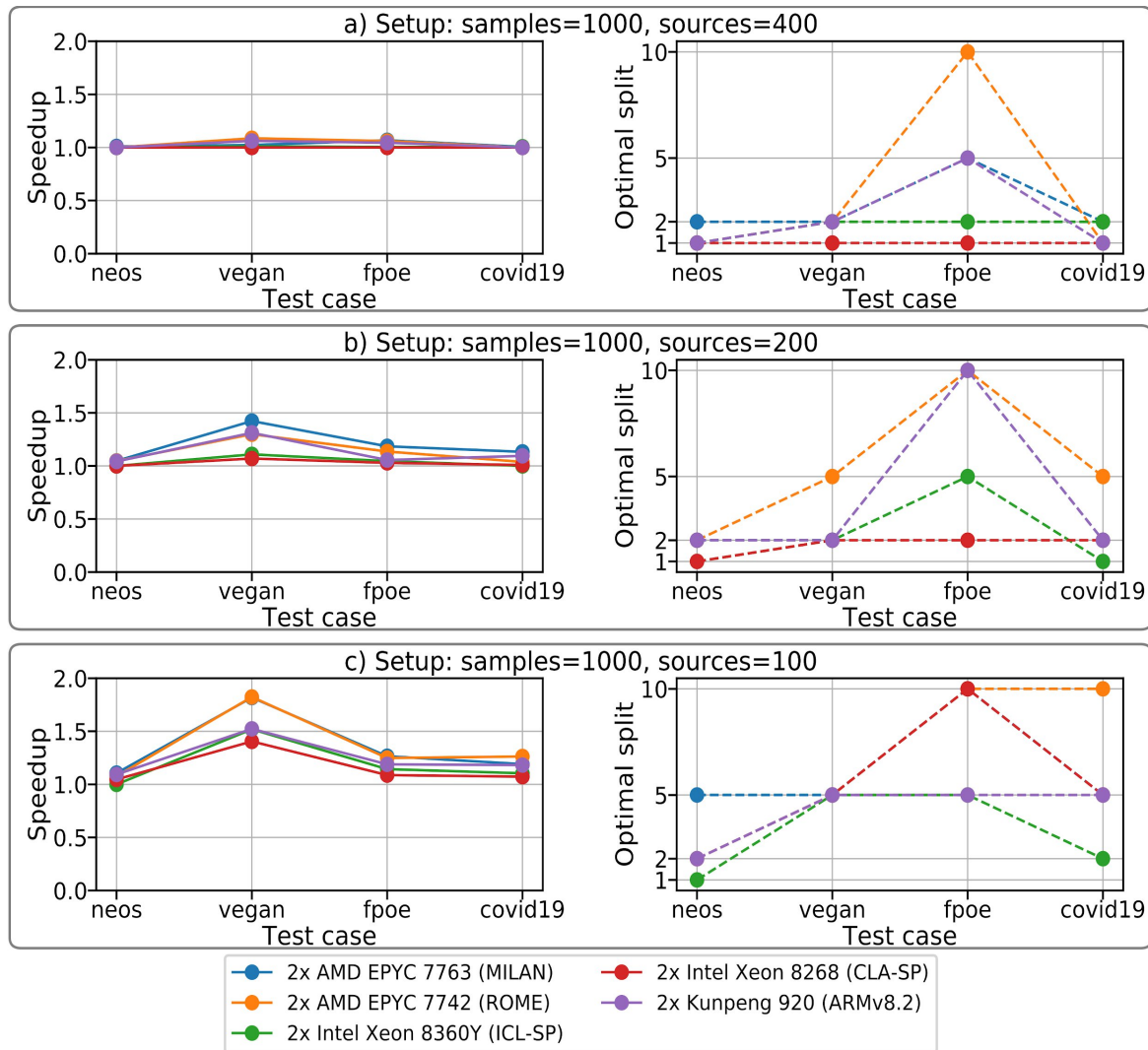


Figure 50. Performance gain (left) and optimal sample split parameter (right).

Additionally, to elevate the performance of our application beyond the limits of the standard Python interpreter we employ the Numba [32] python library to optimize hotspots in our application. We use the just-in-time (JIT) compiler provided by Numba to translate the computing kernels that - in a typical simulation run - are called tens of millions of times and consume more than 95% of computing time.

The function `edge_sample` lies at the core of our simulation. It is the primary hotspot in the simulations of the spread of messages as it is called each time a simulated message arrives at a user in the simulated network. The function is used to determine which followers of this user will adopt (i.e. retweet) this message. For each of these retweeting users, the function `edge_sample` then needs to be called again. This continues until the message is no longer retweeted by any user and the simulation of the current message stops.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies			Page:	96 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0
				Status:	Final

The most notable operations used inside this `edge_sample` function are (i) the generation of random values that follow a binomial distribution and (ii) the random sampling from a list without replacement. Both of these operations are provided by the NumPy python library, which is a library that is eligible for performance improvements by Numba.

The selected functions are marked as Numba functions to be translated into machine codes during application execution (just-in-time compilation). However, to improve the efficiency of our code, we successfully overlap the time-consuming compilation process of selected kernels with reading input data. As a result, all workers hide the compilation process during data loading from files tackled by the master. Figure 51 demonstrates the performance gains of the new version of code with enabled and disabled num JIT compilation process obtained for different computing platforms and application scenarios.

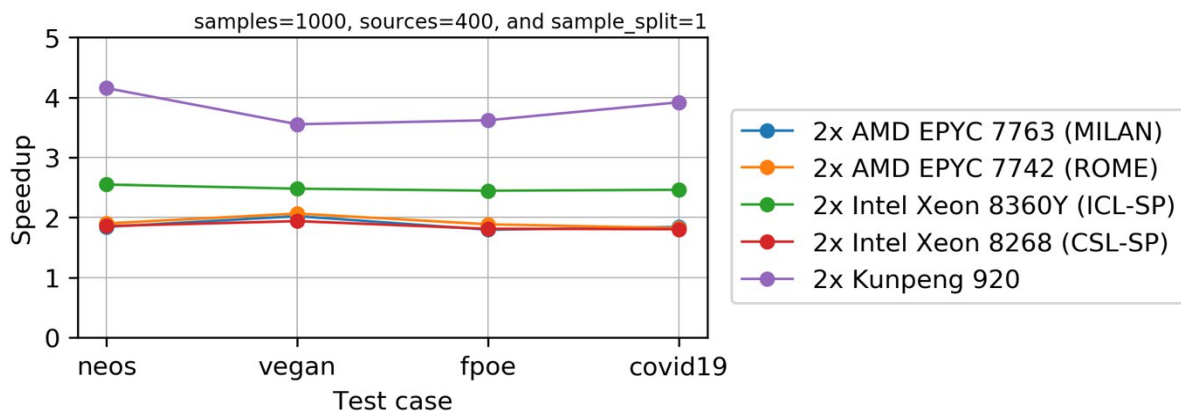


Figure 51. Performance gain obtained for SN Simulator with enabled NUMBA-based optimizations.

5.3.3 The KPM application

In contrast to the computed oriented code of SN Simulator, the KPM application (EigHist eigenvalue) is a memory-sensitive and demands a higher memory bandwidth. This application is also parallelized using the MPI standard for Python. During the analysis, we reveal some communication constraints for inter- and intra-CPU data traffic between the NUMA domains. We use the `numastat` tool to track memory statistics, including allocation usage, hits and misses on a per-NUMA-node basis for the application execution. Figure 52a illustrates the trace memory usage for the basic version of KPM code measured on the system equipped with two Intel Xeon Platinum 8268 CPUs and configured as four NUMA domains.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies			Page:	97 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0
				Status:	Final

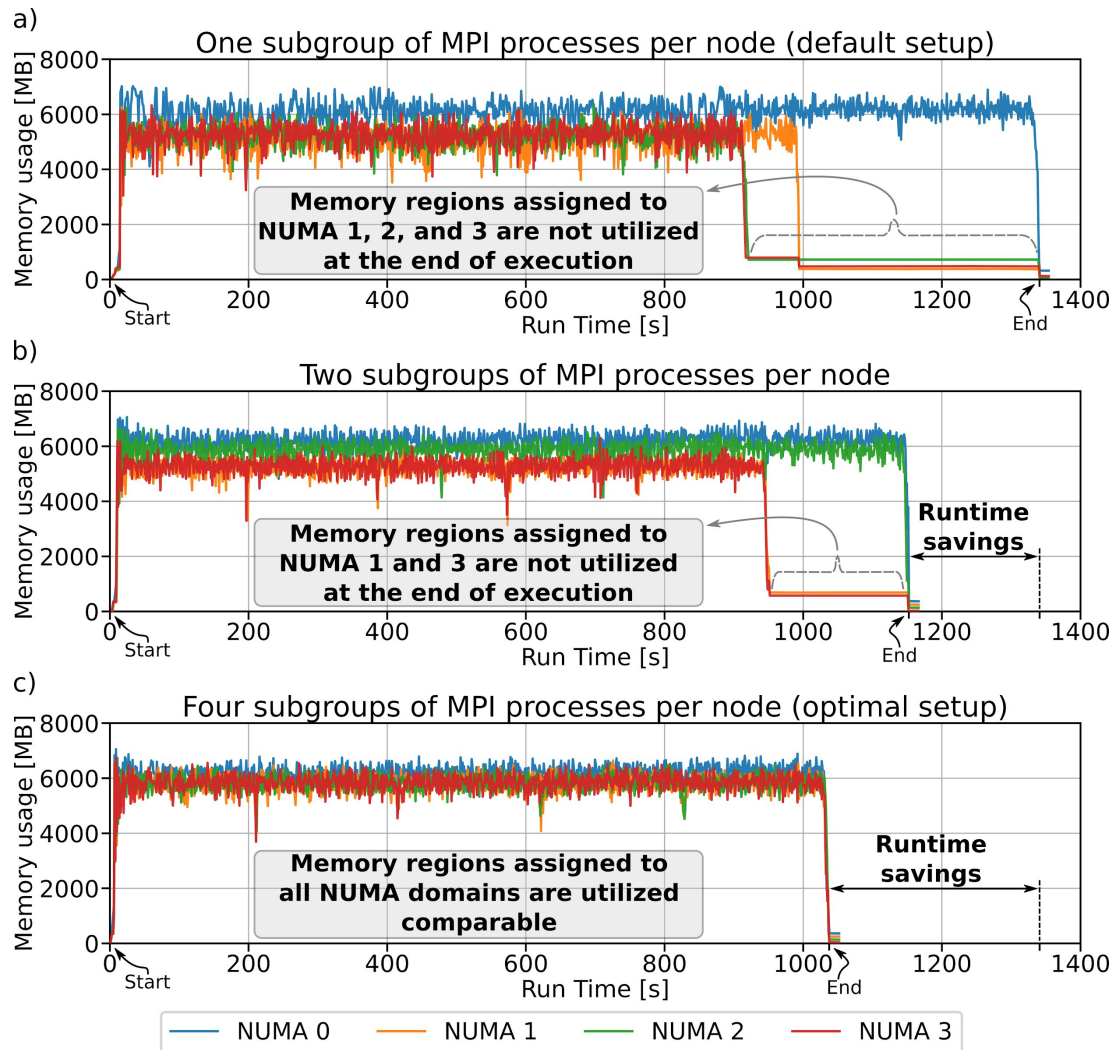


Figure 52. Memory trace of the KPM application execution with a) one, b) two, and c) four subgroups of MPI processes per node obtained for the system with 2x Intel Xeon Platinum 8268 configured as four NUMA domains (every sample is averaged over 1-second interval).

The conducted analysis reveals that the basic version of the code cannot fully take advantage of the memory subsystem of modern ccNUMA-based multicore shared memory systems. Notably, as shown in Figure 52a) the memory regions connected through the NUMA domains 1, 2, and 3 are not explored fully at the end of application execution. All MPI processes read and write data through a single NUMA domain and utilize only about a quarter of the whole I/O memory subsystem. For explaining and solving such memory constraints, we have to look at the data allocation process of the KPM code.

In the basic version of the code, a single MPI process reads all required data from the files and then replicates them between used computing nodes. At the same time, a single MPI process per every node receives required input data and shares them between other MPI processes

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies	Page:	98 of 174
Reference:	D3.5	Dissemination:	PU
		Version:	1.0
		Status:	Final

assigned to a given node through MPI-based shared memory functionality. Assuming that the computing systems typically utilize the first touch policy for memory allocation, a single MPI process per node keeps all input data in its NUMA domain. Consequently, the MPI processes assigned to cores of other NUMA domains have to load and save data using the foreign NUMA domain.

The current top-of-the-line HPC systems offer up to eight NUMA domains for AMD-based dual-socket servers and typically two NUMA domains for Intel-based solutions. Generally, every NUMA domain consists of a team of cores, cache hierarchy, and memory channels that ensure direct access to the main memory of the domain' teams of cores. To get the optimal performance out of current machines requires the NUMA-aware movement of data around the whole memory system. NUMA-aware computing can improve an application's bandwidth and latency requirements on a given NUMA compute node.

This study introduces the ccNUMA aware approach to allocating all required input data within every NUMA domain. The MPI for python provides a communicator split operation that allows one to create subgroups of MPI processes such that the ranks in each subgroup can share memory in systems with shared memory. In the basic code version, a single subgroup of the MPI process is formed and mapped on a single node. In contrast, we propose to create more subgroups per a given dual-socket system. As a result, the master rank broadcasts all input data between subgroups. Simultaneously, every sub-master from each subgroup (i) receives required data, (ii) saves them in its NUMA domain, and (iii) shares them with other MPI processes from a given subgroup by using MPI-based shared memory functionality.

However, the essential point is mapping subsequent sub-masters with cores assigned to different NUMA domains. As a result, every MPI process is mapped and pinned to a single core by adding the following MPI running parameters `--map-by core --bind-to core` available from used OpenMPI implementation of MPI. Afterward, the sub-masters are selected from the pool of MPI processes in a round-robin fashion taking into account the desired number of subgroups.

The optimal number of subgroups is expected to be related to the number of physical NUMA domains. However, we compare the overall performance by examining different application setups, including 1, 2, 4, and 8 subgroups per node. An example of the performed comparison is outlined in Figure 52a-c. This figure illustrates the trace memory usage for 1, 2, and 4 subgroups on the system equipped with two Intel Xeon Platinum 8268 CPUs and configured as 4 NUMA domains. The presented example reveals the optimal utilization of all NUMA domains for application configuration with four subgroups. In contrast, the other setups with 1 and 4 subgroups limit exploration of three and two NUMA domains, respectively, at the end of application execution.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	99 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

Figure 53 shows the performance gain achieved for different numbers of subgroups in comparison to the basic version of code with a single subgroup per node. The proposed approach leads to a better exploration of the whole memory subsystem, improving the overall performance up to 2.5x.

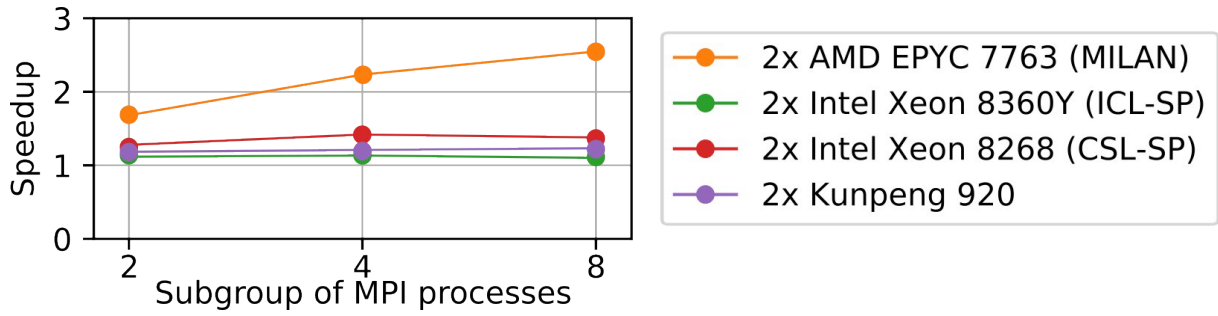


Figure 53. Performance results obtained for the NUMA aware version of the KPM application in comparison to the basic version.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	100 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

6 HPDA analysis and developments

Global Challenges applications by nature produce massive data. In this section we undertake the challenge of utilizing highly scalable analytics runtimes and algorithms to process data in time to be handled or provisioned by HPC applications.

6.1 Framework and Execution Environment

Apache Spark [37] has been chosen as the framework and execution environment for the HPDA methods in HiDALGO. In the next few paragraphs, we will give a brief presentation of Apache Spark that will help the reader understand the intricacies of its programming paradigm and allow us to justify our findings in the benchmarking section that follows.

Apache Spark is a general-purpose framework for quickly processing large scale data that is developed in Scala programming language. It is an open-source project highly adopted by both industry and academia, designed to implement the distributed computing paradigm by leveraging in-memory, fault tolerant data structures, known as RDDs. The versatility and state-of-the-art performance that Apache Spark offers have been the main reasons behind its adoption by HiDALGO as our choice for the implementation of HPDA methods.

Specifically engineered to support the development of big data applications, Spark comes with a wide range of special-purpose libraries (SparkSQL, MLlib, Structured Streaming, GraphX) that can be employed to perform tasks such as data loading, SQL querying, data transformations, machine learning and graph analytics workloads, streaming computations, etc. Additionally, Spark supports programming interfaces with a variety of languages (Scala, Python, Java), a feature which has been exploited in HiDALGO.

The processing engine of Spark is built on top of the MapReduce paradigm [38], initially implemented by Spark's predecessor, Apache Hadoop [39]. MapReduce is a programming model for processing large data sets with a parallel, distributed algorithm on a cluster.

In the following sections we will present the HPDA methods developed to address the needs of the HiDALGO project, per pilot.

6.2 Migration Pilot

The contribution of HPDA in the Migration Pilot has been focused on providing methods that can support the development of the HPC simulator. More specifically, we developed an analytical toolset that can be used on the output of large scale HPC simulations to provide

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	101 of 174	
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status:	Final

insights useful for the verification of correctness of the simulated results. Therefore, we refer to post-processing HPDA to HPC coupling.

6.2.1 Simulation Output Statistical Analytics

6.2.1.1 Purpose

In this sub-section we will present three representative queries that have been developed to assist in the development of the Migration Pilot use case. These queries use the output of large scale HPC simulations as input. Using Apache Spark, the data is processed and manipulated in order to provide insights useful for the verification of correctness of the simulated results.

6.2.1.2 Prerequisites

This application assumes that a working environment of Apache Spark and HDFS is in place. The input data is read from .csv file and the output of the analytics written to HDFS.

6.2.1.3 Input Dataset description

The input consists of a large collection of .csv files, produced by a single simulation, containing data describing the movement of agents in our simulated scenario. In the simulation, agents representing people caught in conflict areas, move through a network of locations such as towns, conflict zones and camps. Additionally, we have access to the data that was used to produce the simulation, containing the location data.

Agent Data Files

These files are by default given a unique identifier by FLEE in the following form: `agents.out.x` where `x` is each file's identifier (e.g. `agents.out.1`, `agents.out.58` etc.). The agent data files follow the structure displayed in Table 25.

Column Name	Sample Value	Description
<code>time</code>	1	Current time step.
<code>rank-agentid</code>	102-1	A unique identifier for distinct agents. It always comes in an "x-y" format, where x

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies			Page:	102 of 174		
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status:	Final

Column Name	Sample Value	Description
		is the unique output file identifier and y is the agent identifier within the file.
agent_location	Kajo-Keji	Current agent location.
distance_travelled	414.0	The total distance the agent has travelled since spawn.
places_travelled	7	The total number of locations the agent has visited.
distance_moved_this_timestep	414.0	The distance the agent travelled during the last time step.

Table 25. Agent data file structure and sample values

Location Data File

A single location data file is available in CSV format. This file is used as input to produce the simulated dataset and contains information about the locations in the scenario. The location data file follows the structure displayed in Table 26.

Column Name	Sample Value	Description
name	Juba	The location name.
region	Central_Equatoria	The location region.
country	South_Sudan	The location country.
lat	4.86086	Latitude.
lon	31.61782	Longitude.
location_type	conflict_zone	Location type (can hold the following values: camp, conflict_zone, town).
conflict_date	0	The time step that conflict reached the location.
population	368436	The location population.

Table 26. Location data file structure and sample values

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies			Page:	103 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0
				Status:	Final

6.2.1.4 Implemented Queries

Using the data described above, we have implemented the following 3 queries:

- **Q1: *Which agents are spawn in camps?***

Q1, is rather simple: the task is to identify the agents which first appear in the simulation in a camp location.

- **Q2: *How many days do the agents take to reach the first camp?***

Q2 requires us to identify the first camp each agent reaches in the simulations and then calculate the number of days it has taken for them to encounter the first camp, since the agent's first appearance in the dataset. Obviously, for agents included in the result set for Q1, the answer would be 0 days.

- **Q3: *How much distance do the agents cover until they reach the first camp?***

Q3 is similar to Q2. However, in this case instead of days gone by, we calculate the distance covered by each agent since its first appearance in the dataset until they reach the first camp. Again, for agents included in the result set for Q1, the answer would be 0 kilometres.

All three queries focus on instances where agents are found in camps. Therefore, having appended all agent data files into one large table, we subsequently perform a join operation against the location data in order to access the information regarding location type. We group these data points by agent and store them in chronological order. Next, we perform another data transformation to filter out every datapoint that indicates the presence of an agent in a location tagged as camp, excluding its initial appearance. Finally, we apply a number of transformations to present the results in a readable format (csv) and write the corresponding files. For queries Q2 and Q3, we prepare the results to be ready for visualization purposes in the form of a frequency histogram - in the case of Q3 using a bucketed approach. The pseudo-code of how the queries' implementation is given below:

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	104 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

1. **Read** all agent data files into a single table ("agent_data").
2. **Read** the location data files into a second table ("location_data").
3. Perform a **left outer join** between the two tables on ("agent_data.agent_location" == "location_data.name").
4. **Group** the result **by** "rank_agentid", thus creating a list of locations visited by each agent in chronological order.
5. To reduce the size of unnecessary data, **create a summary** of the agent's journey including the "start" and "end" records (the first and last time an agent appears in the dataset).
For this summary we only retain records for time steps where an agent's location changes, indicated by a positive distance travelled.
6. **Filter out** all summary entries where ("location_type" != "camp").
7. **Store** in column "first_camp" the chronologically first record of a camp that the agent encounters.
8. **Store** in column "dst_first_camp" the "dst_trvld" element of the "first_camp" record.
9. **Store** in column "days_first_camp" the difference between the time steps the agent is spawn("start.time") and the first time they enter a camp ("first_camp.time").

For Q1:

10. **Select** column ("rank_agentid") **where** first_camp.time == start.time.
11. **Write** results into csv.

For Q2:

12. **Select** columns ("rank_agentid", "days_first_camp").
13. **Group by** "days_first_camp" and count apply count() to aggregate.
14. **Sort by** "days_first_camp".
15. **Write** results into csv.

For Q3:

16. **Select** columns ("rank_agentid", "dst_first_camp").
17. **Use "bins"** of configurable width to categorize agents according to distance travelled (e.g. 0-50 kms, 51-100 kms etc.)
18. **Group by** "bin" and apply count() to aggregate.
19. **Sort by** "bin".
20. **Write** results into csv.

This code segment consolidates the three queries into a single application. However, in the corresponding benchmarking section (Chapter 7) we study the behaviour of the three both separately and as a consolidated application.

6.2.1.5 Output description

The output of the three queries is stored in CSV format. For Q1, we have a simple list of agent-ids that satisfy the query condition. For Q2, we have a tabular format with two columns, the first corresponding to the days that passed between an agent first appeared in the dataset until they reach the first camp and the second to number of agents in the simulation output that fall into that category. Similarly, for Q3, we have a tabular format with two columns, the first corresponding to the distance travelled by the agents until the first camp was reached. The distance values are measured in kilometres and bucketed in bins of 50 so bin 1 translates

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	105 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

to 0-50 kilometres. The second column, again, corresponds to the number of agents in the simulation output that falls into the specific category.

days_first_camp	agents
0	105472
1	40887
2	23517

Table 27. Sample output for Q2 in tabular format.

bin	agents
1	5
2	5983
3	42049

Table 28. Sample output for Q3 in tabular format.

6.3 Urban Air Pollution Pilot

The contribution of HPDA in the Urban Air Pollution Pilot has been twofold. First, we implemented a post-processing method for calculating the Single Value Decomposition (SVD) of snapshot matrices. Second, we worked with the simulation output data in order to provide an analytical toolset that can answer queries regarding air pollution levels and give out notifications for threshold violations consistent with the Directive 2008/50/EC of the European Parliament and of the Council of 21 May 2008 [40] on ambient air quality and cleaner air for Europe.

6.3.1 Snapshot Matrix SVD

6.3.1.1 Purpose

The singular value decomposition of a matrix A is the factorization of A into the product of three matrices $A = U\Sigma V^T$ where the columns of U and V are orthonormal and the matrix Σ is diagonal with positive real entries. The SVD is a common processing task, useful in many applications such as dimensionality reduction. For the implementation we adopted in

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies			Page:	106 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0
				Status:	Final

HiDALGO, we leveraged the power of Spark’s MLib library [41] for linear algebra, which includes an optimized function that calculates the SVD of a given matrix. Our work in this case was focused on finding the optimal format and processing pattern for the data to be brought in the shape that MLib’s SVD implementation can consume.

6.3.1.2 Prerequisites

This application assumes that a working environment of Apache Spark and HDFS is in place. It makes use of the Spark MLib library for the implementation of linear algebra operators. The input data is read from .csv or binary files and the output of the analytics written to HDFS.

6.3.1.3 Input Dataset description

The input consists of .csv or binary files, each of which includes the output of simulations for air quality measurements corresponding to a specific snapshot, taken at regular intervals. Each datapoint in the distinct snapshot files contains data to a single point of a three-dimensional grid covering an urban area. We also have access to information on the location of each point, which is not significant for the SVD workload.

6.3.1.4 Implementation Details

We read the data from the input files located in HDFS in parallel to optimize performance. Our initial implementation only utilized CSV files but we also support binary input. After the input is read into the Spark cluster, a series of data transformations follow, constructing the data structures required to utilize the Spark MLib version of SVD, which is thoroughly documented [42].

Using the MLib SVD function, we calculate the matrices U , Σ , and V and store them for reference in .csv file format.

6.3.2 Air Quality Index

6.3.2.1 Purpose

The second part of the HPDA contribution to the Urban Air Pollution Pilot focuses on implementing the methods described in the Directive 2008/50/EC of the European Parliament and of the Council of 21 May 2008 on ambient air quality and cleaner air for Europe. That document describes multiple methods and processes for the assessment of air quality. Taking those into consideration, we set out to create a toolset that can be used to answer related queries using output data from the simulations.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	107 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

More specifically, we aim at answering queries that focus on specific locations in an urban environment, that cover specific windows of time, as dictated by the user. We implement the distinct parameters specified in the Directive document, as displayed in Figure 54.

Parameter	Required proportion of valid data
One hour values	75 % (i.e. 45 minutes)
Eight hours values	75 % of values (i.e. 6 hours)
Maximum daily 8-hour mean	75 % of the hourly running eight hour averages (i.e. 18 eight hour averages per day)
24-hour values	75 % of the hourly averages (i.e. at least 18 hour values)
Annual mean	90 % ⁽¹⁾ of the one hour values or (if not available) 24-hour values over the year

(¹) The requirements for the calculation of annual mean do not include losses of data due to the regular calibration or the normal maintenance of the instrumentation.

Figure 54. The criteria used for checking validity when aggregating data and calculating statistical parameters according to Directive 2008/50/EC of the European Parliament.

6.3.2.2 Prerequisites

This application assumes that a working environment of Apache Spark and HDFS is in place. The input data is read from .csv files and the output of the analytics written to HDFS.

6.3.2.3 Input Dataset description

As with the previous case of the Snapshot Matrix SVD, the input consists of binary or CSV files, each of which includes the output of simulations for air quality measurements corresponding to a specific snapshot in time, taken at regular intervals. Each datapoint in the distinct snapshot files contains data to a single point of a three-dimensional grid covering an urban area.

In this case we also expect an additional input from the user, indicating the aggregation criteria as listed in Figure 54, the aggregation function (min, max, mean), the time frame for the analytics, the specific point of interest that will be used for the query execution along with

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	108 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

a small radius of effect, and, finally, the selected pollutants the concentration of which will be calculated and information regarding the accepted thresholds of the latter.

6.3.2.4 Implementation Details

We start by reading in the input in CSV format. We proceed to filter out the snapshots that are out of range in terms of the time frame given by the user. Subsequently, we calculate the distance between all points in the grid and the given point of interest and filter out all points outside the given radius.

We then perform a grouping function according to the aggregation criteria given to us by the user and apply the aggregating function they have indicated. For instance, in the simple case of the one-hour values, we group all the data points by hour and then calculate the aggregation function on the data values for each group. We factor in distance when calculating mean values in a sphere by using a distance-weighted mean.

Finally, we perform aggregations according to the windowing parameters that have been defined and compare the results against the accepted threshold numbers given by the user and report whether there has been a violation in the time window of interest.

For reference, three sample queries that can be executed using our developed applications are:

- Calculate possible threshold violations for a specified pollutant concentration (or wind velocity) over 1-hour windows (a limited number of violations are acceptable) in a specified time range. Report whether the limit of acceptable violations has been surpassed or not.
- Calculate possible threshold violations for a specified pollutant's concentration (or wind velocity) over the 16 tumbling 8-hour windows with a step of an hour that cover the duration of a day (24 hours) for a specified time range. We use a time frame of 24 hours to aggregate our findings (a limited number of violations within the frame are acceptable). For each day in the initial time range, we can register at most 16 violations. Report whether the limit of acceptable violations has been surpassed or not.
- Calculate a specified pollutant's concentration or wind velocity over 1-hour windows in the frame of a day (no thresholds are applied).

The implementation of these queries is based on the same common steps described in the previous paragraphs with the differentiation between them being the number and parameterization of the (last) aggregation step that takes place in the algorithm. This method has been implemented as a parameterized application; therefore, the same code will be executed for every submitted query.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	109 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

6.4 Social Network

The contribution of HPDA in the Social Network Pilot has been aimed on providing a way to verify correctness and measure precision of Social Network Simulator. In effect an Analyser has been developed that can be used on real-world data on which the Simulator is trained on, in order to provide in-depth statistics. These statistics are lacking from original data but can be compared with statistics extracted from the Simulator itself.

6.4.1 Social Network Analyser

6.4.1.1 Purpose

The Analyser is used to determine statistics from real-world Twitter data that can be then compared with the output of the Twitter Simulator, in order to verify Simulator's accuracy in predicting Twitter user behaviour.

Presently available real-world data only informs which users retweeted and at what time, but through HPDA methods the retweet order can also be extracted. Of special interest is the number of users that have retweeted an original message and are n-hops away from the original poster.

The following figure (Figure 55) presents the method of determining the distance in retweet tree.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	110 of 174	
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status:	Final

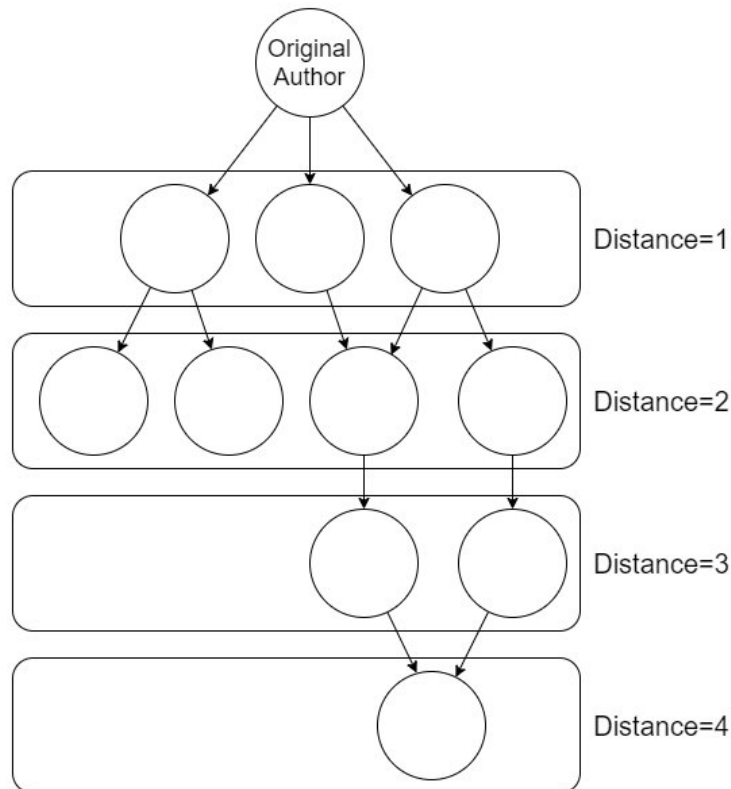


Figure 55. Visualization of retweet tree with distances to particular followers

After this step the total number of nodes at each particular distance can be calculated, producing the desired statistic. In this particular example there are 3 users at distance 1, 4 users at level 2, 2 users at level 3 and finally 1 user at level 4.

6.4.1.2 Prerequisites

This application assumes that a working environment of Apache Spark and HDFS is in place. It requires a Python virtual environment with Graphframes library [43], which is used for graph-related calculations. The input data is read from .csv file and the output of the analytics written to HDFS.

6.4.1.3 Input and output data

There are two kinds input files required – a .metis file containing follower graph and a .csv file for actual tweet data.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	111 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

The `.metis` file holds an adjacency list of the follower graph. It allows to build a network of connections between users: first line contains metadata, then every line number signifies followee and its contents follower1, follower2, follower3, etc.

The `.csv` contains retweet data for each of analysed tweets and is composed of one line per tweet in following structure:

`author, author_feature, tweet_feature, list of retweeters, e.g.:`

author	author_features	Retweeters
23	0,1,0,1	„[34, 45, 56]“

The Analyser outputs a list of n-level followers who retweeted any given tweet:

author	retweets	level1	level2	level3	level4	...
23	„[34, 45, 56]“	2	1	0	0	
65	„[34, 56, 67]“	1	1	1	0	

In this particular case 34 and 45 are the followers of 23 and 65, 56 is follower of 34 and finally 67 is a follower of 56.

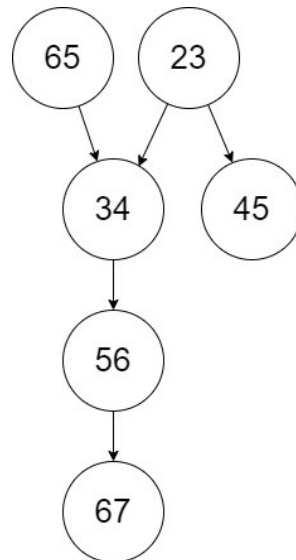


Figure 56. Example of a retweet tree

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies	Page:	112 of 174				
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status:	Final

6.4.1.4 Implementation details

Statistics are extracted from real-world data. The statistics extraction is performed in several steps:

1. First, the datasets (follower graph and (re)tweet list) are loaded into Spark dataframes format and pre-processed. This includes renaming column names, so they match, filtering unused or unnecessary rows/columns, adding IDs, etc. To give an example, if a tweet has not been retweeted or a user does not follow anyone, they are excluded from further analysis.
2. Then, the node dataframe and edges dataframe are combined into a graph representation.
3. Next, for every tweet, the following operations are made:
 - a. distances are calculated for all retweeters
 - b. resulting distance list gets sorted, grouped and differing distance instances are counted
 - c. the resulting row of data is then attached (unioned) to the already existing distance matrix; empty columns are filled in with 0.
4. Finally, after all the tweets have been processed, the distance matrix is recombined (joined) with authors list, creating the results dataset.
5. The results dataset is then turned into a .csv format and saved to disk.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	113 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

7 HPDA benchmarking

In this section we will present the results of the HPDA methods benchmarking process. The goal of the benchmarking has been to prove our applications' ability to scale and gracefully process large data-sets while retaining competitive performance. The presentation of the benchmarking results follows the content of Section 6.

Data repository folder:

https://gitlab.com/eu_hidalgo/benchmarking/-/tree/master/deliverable_3_5/hpda_benchmarking

7.1 Infrastructure description

All HPDA benchmarks have been conducted on a dedicated Spark cluster. It consists of 24 compute nodes with 2.4GHz Intel Haswell processors and 32GB of RAM. For HPDA purposes a 2.5TB storage space is mounted.

The software stack consists of Spark, Hadoop and supporting libraries such as Graphframes. More details about the cluster and nodes are presented in Table 29 and Table 30.

HPDA Spark cluster infrastructure details	
System	PSNC HPDA testing infrastructure
Total Compute Nodes	24
Total Cores	345
Total Memory	690GB
Python	3.6.9
Scala	2.11.12
Spark version	2.4.5
Hadoop version	2.7
Graphframes library version	0.8.1

Table 29. HPDA Spark cluster infrastructure details.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	114 of 174	
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status:	Final

Node details	
Vendor	Intel
Model	Haswell
CPU Name	Model 60
Cores per chip	16
Threads per core	1
CPU MHz	2400
Primary Cache	32 KB I + 32 KB D on chip per core
Secondary Cache	4096K
L3 Cache	16384K
Memory	32GB
OS	Ubuntu 18.04.4 LTS
Disk drive	HDD 7200rpm

Table 30. HPDA Spark cluster node details.

7.2 Migration Pilot

7.2.1 Simulation Output Statistical Analytics tests

We benchmark the statistical analytics methods described in Section 6.2 and report our results here. Our goal is to document our methods' capacity to scale with hardware. We conduct three separate experiments in order to document the performance of each of the queries implemented.

Test runs are conducted on a dataset with the following characteristics:

- A simulation output data set from the South Sudan scenario has been created, which comprises 128 csv files, each containing roughly 2.6 million rows,
- Time variable takes values ranging from 1 to 425, indicating days elapsed,
- The whole dataset measures up to 14.4GB.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	115 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

We scale the resources reserved for the workload execution from 2 to 7 Spark executors and notice that the performance follows a near-linear speedup. As we can see in Figure 57, the three queries Q1, Q2 and Q3 do not display much variance in their behaviour between them, which is expected, since the algorithmic steps followed to reach the final calculations are very close as described in Section 6.2.

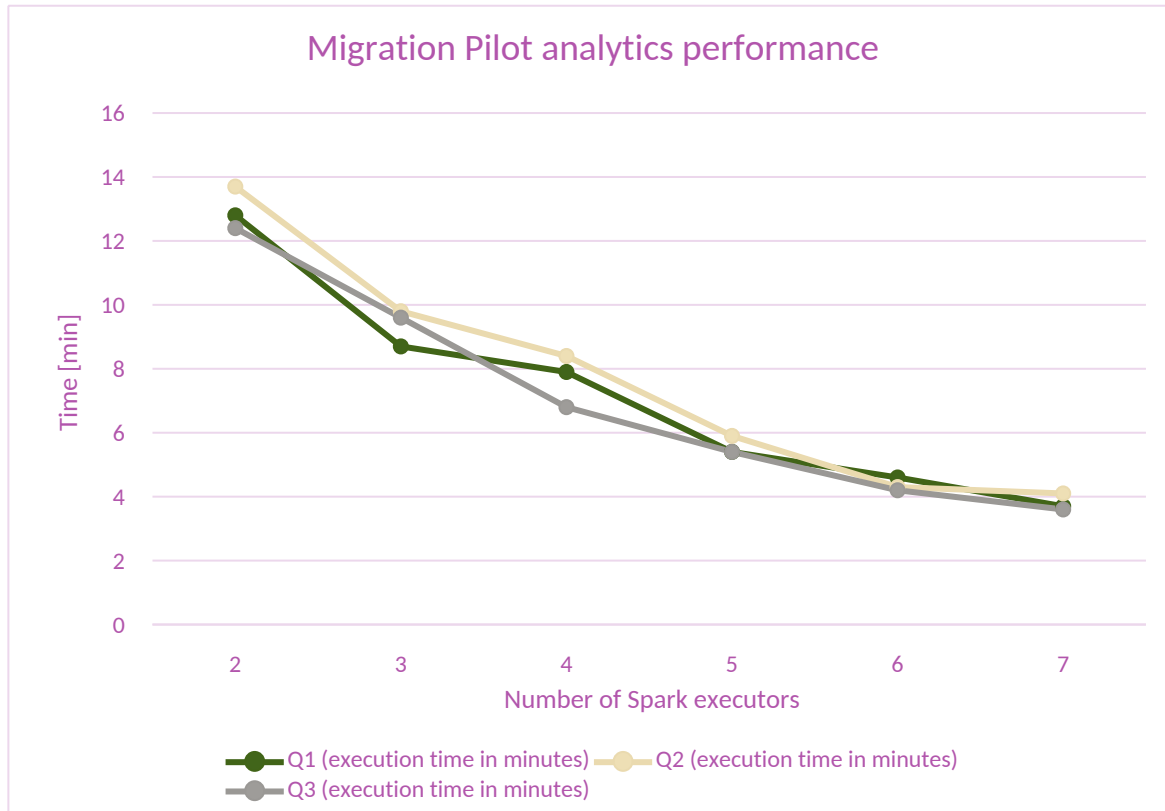


Figure 57. Migration Pilot Analytics performance on Spark using varying numbers of executors.

7.3 Urban Air Pollution Pilot

We benchmark the performance of the analytics methods described in Section 6.3 and report our findings in the present section. We conduct two separate sets of experiments, the first one on the SVD method and the second one on the Air Quality analysis methods. We focus on the scalability of the methods and examine their behaviour when assigned resources of various sizes.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	116 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

7.3.1 Snapshot Matrix SVD tests

The original data-set, we experiment with, consists of 361 snapshots. We conduct experiments varying the data-set size and the number of used spark executors to benchmark the application in terms of scalability. Our first experiment displays the impact of the input size. Using the same number of resources, we execute the workload on the whole dataset, then using half of it, and finally only a quarter of the snapshots. We notice that the execution time does indeed scale almost linearly.

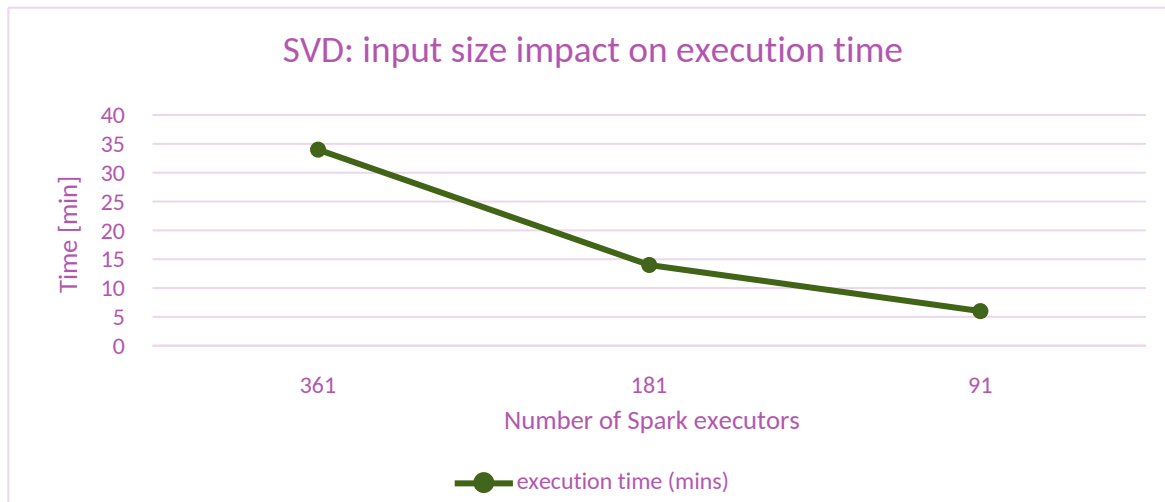


Figure 58. SVD performance using varying the input data sizes.

Next, we document the execution time of our application that calculates the SVD using the full data-set with varying numbers of spark executors allocated. We notice that the scaling is not quite linear. This happens due to some of the linear algebra operations involved, the parallelization of which cannot exploit the whole of the cluster resources.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	117 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

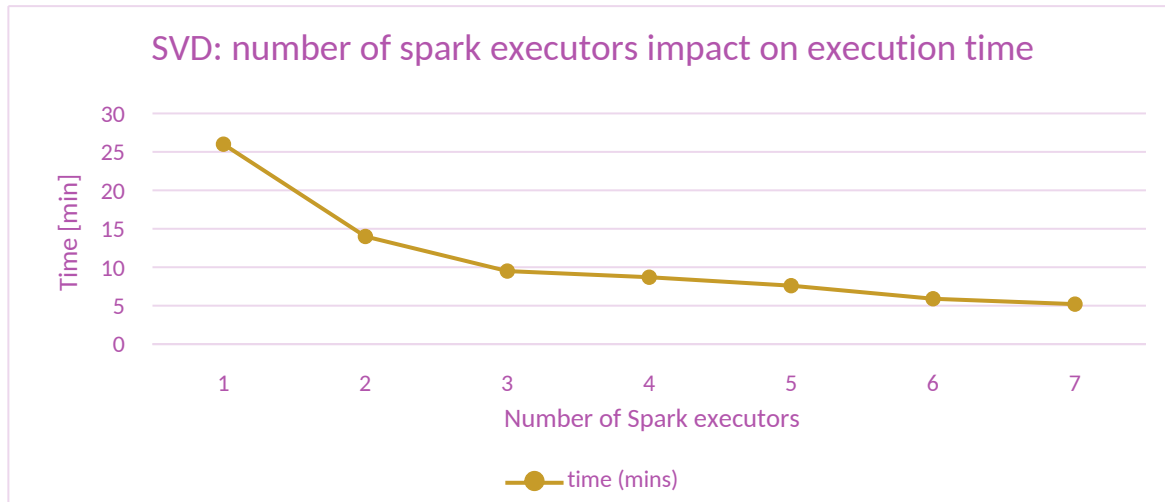


Figure 59. SVD performance using varying numbers of spark executors.

7.3.2 Air Quality Indices tests

The data-set we experiment with consists of 361 snapshots – each marked with a timestamp. They contain simulation information at a 10-minute interval in an urban environment in Bologna. The total size of the dataset measures up to 3.6GB.

To benchmark our methods, we use the 3 distinct queries we introduced in section 6.3.2.4.

- **Query 1:** Calculate possible threshold violations for a specified pollutant concentration (or wind velocity) over 1-hour windows (a limited number of violations are acceptable) in a specified time range. Report whether the limit of acceptable violations has been surpassed or not.
- **Query 2:** Calculate possible threshold violations for a specified pollutant concentration (or wind velocity) over 1-hour windows (a limited number of violations are acceptable) in a specified time range. Report whether the limit of acceptable violations has been surpassed or not.
- **Query 3:** Calculate a specified pollutant's concentration or wind velocity over 1-hour windows in the frame of a day (no thresholds are applied).

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	118 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

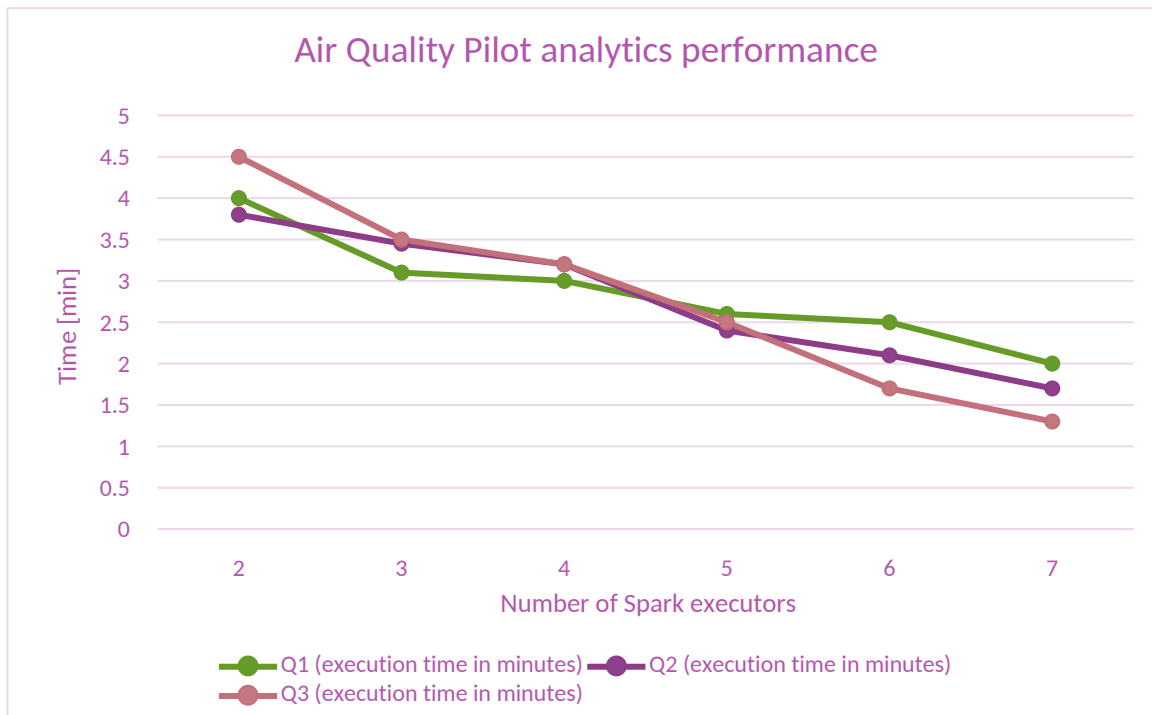


Figure 60. Air Quality Analytics performance with varying numbers of Spark executors.

We execute the queries multiple times using the same point in the urban environment as reference and the same radius (10 m) to ensure fairness, and report the mean execution times in Figure 60. We can observe that the different queries cause minimal fluctuation in the performance, as they follow a similar execution pattern, with minor changes due to the aggregations that take place when there are time frames and windowing involved – which adds an extra execution layer. We additionally notice that the performance indeed improves with more resources, as we gain a speed-up of a few seconds in each case.

Next, to assess the impact selectivity (i.e., the number of mesh points which are within radius distance with regards to the specified point of interest) has on the processing time, we change the selected point and radius combination. Selectivity can be affected by the mesh geometry (some parts can be denser than others), or radius selection (a large radius would result in more points selected). We use Query 2 from the experiment above and fix the configuration to 5 executors (the baseline for this experiment is an execution time of just 129 seconds, as depicted in Figure 60). We then modify our selected point of interest and radius to increase selectivity by x10, x100, and x1000 in comparison to the reference and report the impact in execution times (Table 31). We can see that the query selectivity barely impacts performance, as the filters applied due to the temporal and special constraints set by the user already limit the size of the working data set significantly.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	119 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

Query Selectivity	Execution time (secs)
x1	129
x10	134
x100	169
x1000	190

Table 31. Impact of selectivity on Query Execution.

7.4 Social Networks Pilot

7.4.1 Social Network analyser tests

The Twitter analyser is used to derive statistics from real-world Twitter data. It has been described in more details in Section 6.3.

There are two purposes of this benchmark: one is to determine the ability of Twitter analyser to scale with hardware and the other is to find the impact of graph size on execution time. To this end, the analyser has been executed on various number of nodes, each featuring 8 cores.

Three dataset graphs of differing size in total were tested. They are presented in Table 32. In order to keep runs between different graphs comparable, the list of tweets for each dataset has a constant length of 1274 and features similar complexity level.

Dataset	Nodes	Edges
FPOE	21291	2527541
VEGAN	11015	381627
NEOS	8277	799792

Table 32. Characteristics of selected dataset graphs.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	120 of 174	
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status:	Final

7.4.2 Results and findings

The benchmarking results of performance and scaling speedup are presented in Figure 61 and Figure 62.

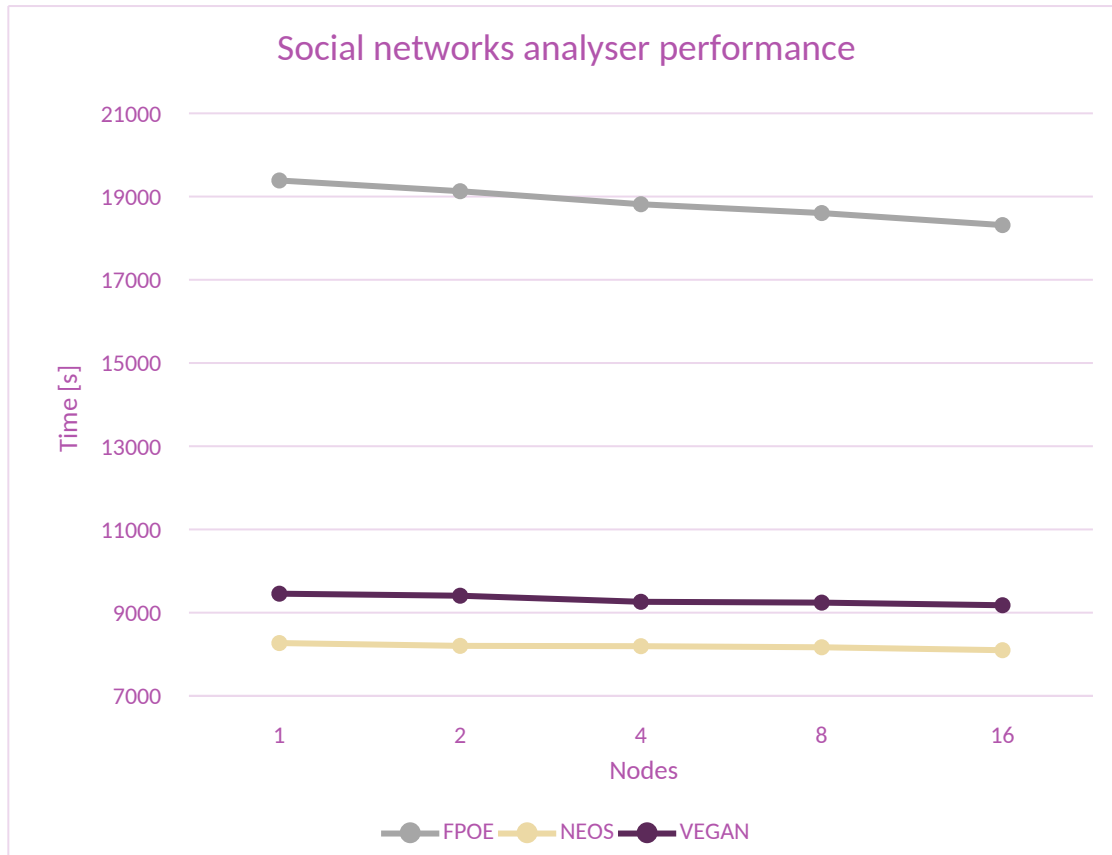


Figure 61. Performance of social networks analyser on selected datasets.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	121 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

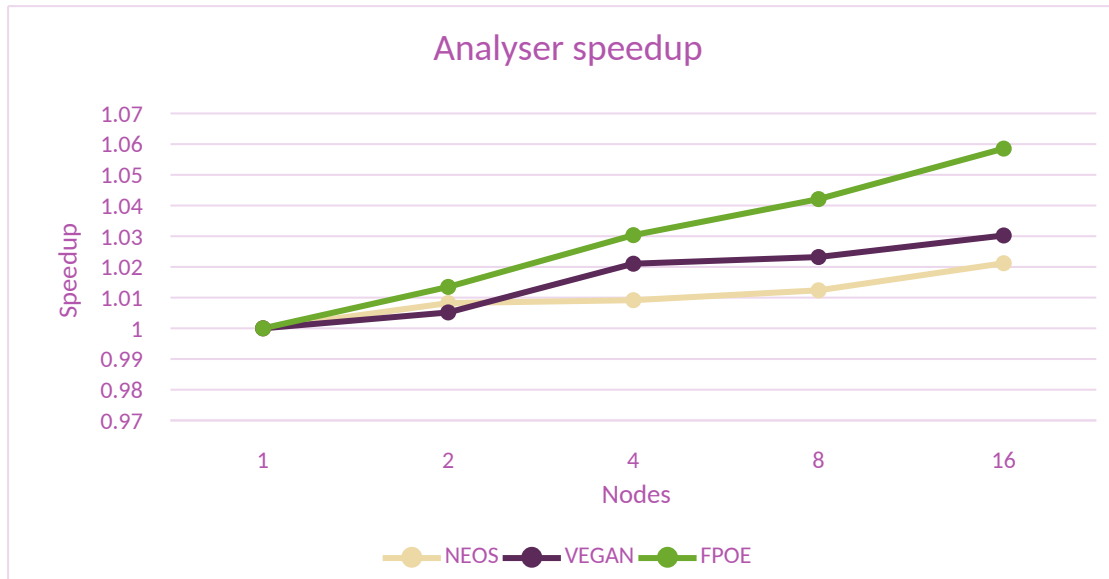


Figure 62. Performance speedup of social networks analyser on selected datasets.

The results show a significant time increase for bigger datasets, suggesting calculating distances in large graphs is much more demanding than smaller ones.

Apart from this, a minor performance improvement is observed with increased number of nodes dedicated to the task. The effect is most pronounced with bigger datasets. It is because the graph searching, which is the most time-consuming part of processing needs to be executed sequentially for each tweet with frequent synchronisation barriers, which limits the effects of scalability of the whole application. This is necessary, as the computations depend on data from multiple datasets and therefore cannot be parallelized any further. Both pre- and postprocessing are impacted by scaling hardware resources as they fully utilize Spark transformations.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	122 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

8 GPGPU benchmarking and optimization

An emergent way to deal with complex tasks is using machine learning algorithms run on dedicated hardware accelerators. In this section we present the results of the work done in these areas. It is split into benchmarking of GPGPUs using neural models and data distribution strategies for training optimization. The focus of these tests is on scaling in multiple-GPU environment.

The purpose of the experiments has been to provide an overview of available techniques and determine their ability to scale and perform.

8.1 Infrastructure description

GPGPU tests have been conducted on several infrastructures: Vulcan, Altair and AMD cloud. Each of them provides nodes with multiple GPUs.

Both Vulcan and Altair are equipped with Nvidia Tesla V100 accelerators with SXM2 interface. Where they differ is in CPUs (Intel Xeon 8268 vs Intel Xeon 6240) and job scheduling system (PBS Torque vs SLURM). On AMD cloud there are nodes with two different GPU models – AMD MI50 and AMD MI100 but they all use the same CPU - AMD EPYC 7742. The software stack consists of essential GPU tooling and environment: CUDA for Nvidia and ROCm for AMD, as well as PyTorch 1.7.0 with profiling libraries for machine learning.

More details about graphical accelerators can be found in section 3.2.2 of deliverable 5.5.

8.2 Benchmarking of GPGPU cards

This work measured performance of available GPGPU architectures in a variety of machine learning tasks. They were compared by running training of image classification algorithms differing in complexity.

8.2.1 Goal and methodology

The goal of these tests is to compare the performance of various GPGPU architectures when running diversely demanding machine learning tasks. Additionally, impact of hardware scaling (1 up to 8 GPUs) and other infrastructure elements have also been investigated.

Each neural model has been run with the same setup:

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies			Page:	123 of 174	
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

- Input images have dimensions of 224x224x3, are made of randomized, double-precision data,
- Batch size is 48,
- First, 5 warmup forward and backward propagation rounds are performed, then results of 50 rounds of training are averaged to get the final result.

8.2.2 Scalability tests of machine learning models

Tests have been performed with 3 readily available neural models used for image classification: ResNext101_32x8d, VGG19 and Mobilenet_v2. They differ in scale and complexity, therefore necessitating a varying amount of GPU processing power.

8.2.2.1 Model comparison

The details of all three models are presented in Table 33.

Model	Layers	FLOPs	Number of parameters
ResNext101_32x8d	101	16B	88M
VGG19	19	19.6B	144M
Mobilenet_v2	53	300 M	3.47 M

Table 33. Details of tested machine learning models.

The first series of charts present the training time for steady, 48 batch size training of a particular neural model on specified number of GPUs.

The second series of charts present relative increase in speed when scaling the training job for multiple GPUs.

8.2.2.2 ResNext101_32x8d

ResNext is machine learning architecture that is 101 layers deep. It is a homogeneous neural network, which reduces the number of hyperparameters required by conventional ResNext. This is achieved by their use of "cardinality", an additional dimension on top of the width and depth of ResNext. Cardinality defines the size of the set of transformations. It has 88 million

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	124 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

parameters and a single forward pass requires 16 billion FLOPs. This model has been described in [44].

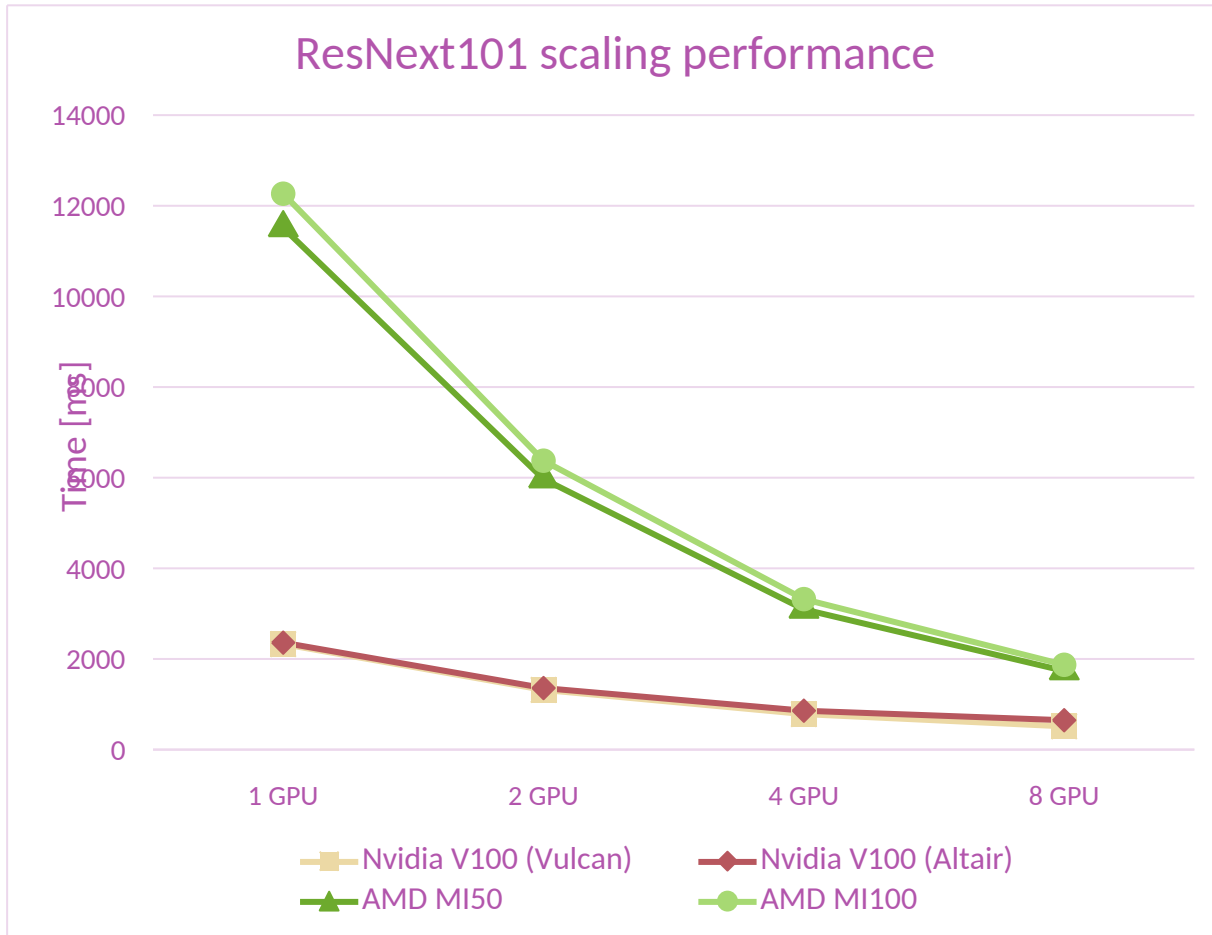


Figure 63. Training performance for ResNext neural model on selected architectures.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies			Page:	125 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0
				Status:	Final

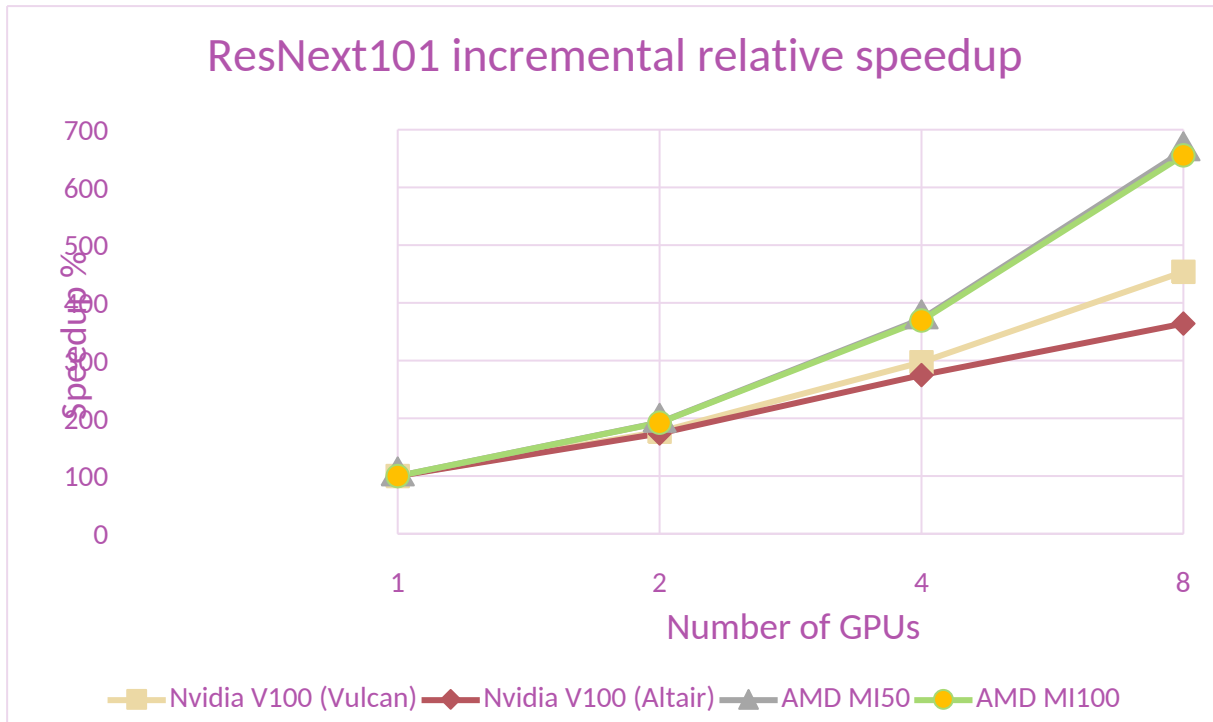


Figure 64. Training performance speedup for ResNext neural model on selected architectures.

ResNext tests display a linear speedup (Figure 64). Nvidia cards offer better performance in this scenario, but AMD offerings scale better (Figure 63).

8.2.2.3 VGG19

Used in large-scale image recognition setting, VGG19 is a variant of VGG model which in consists of only 19 layers (16 convolution layers, 3 fully connected layers). VGG19 uses 19.6 billion FLOPs per forward pass. The number of parameters is 144 million making it a big and complicated neural model. This model has been described in [45].

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies			Page:	126 of 174		
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status:	Final

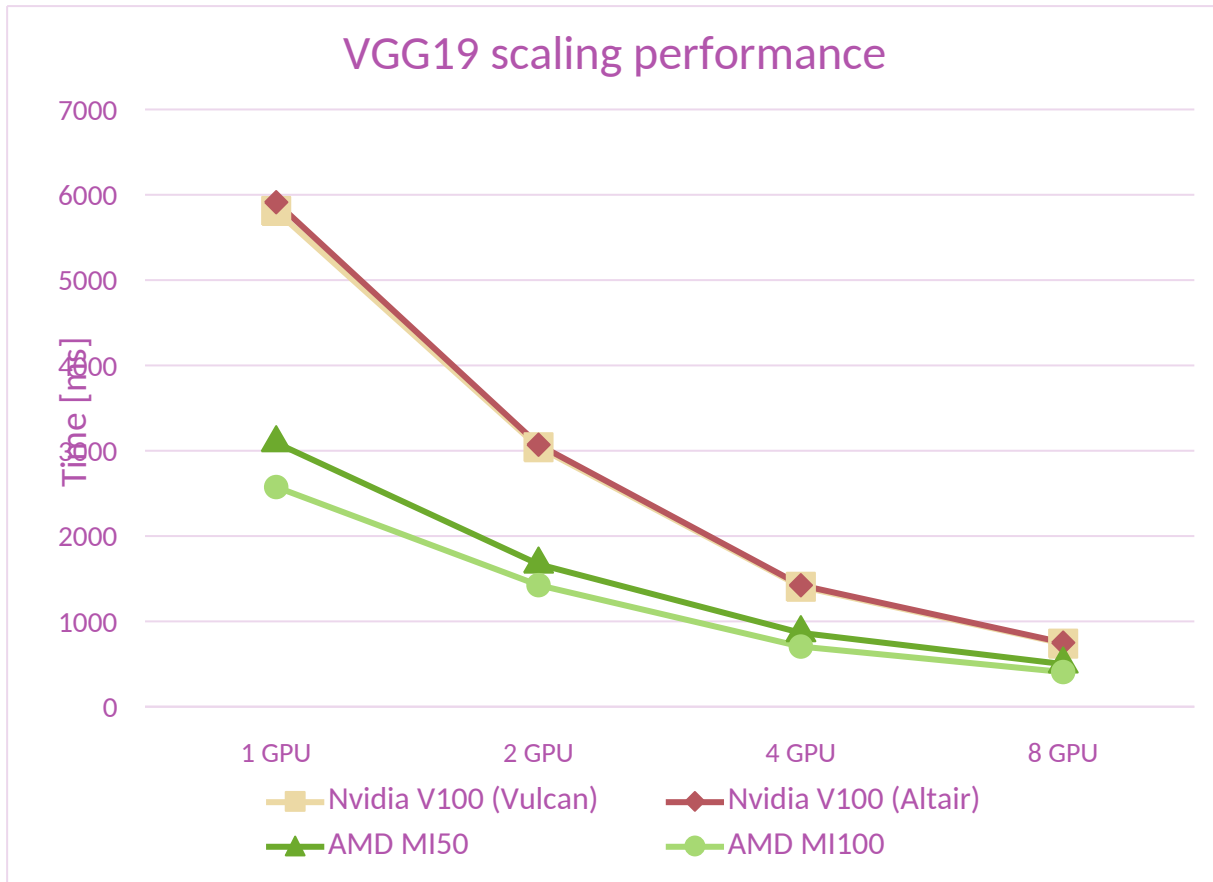


Figure 65. Training performance for VGG19 neural model on selected architectures.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies			Page:	127 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0
				Status:	Final

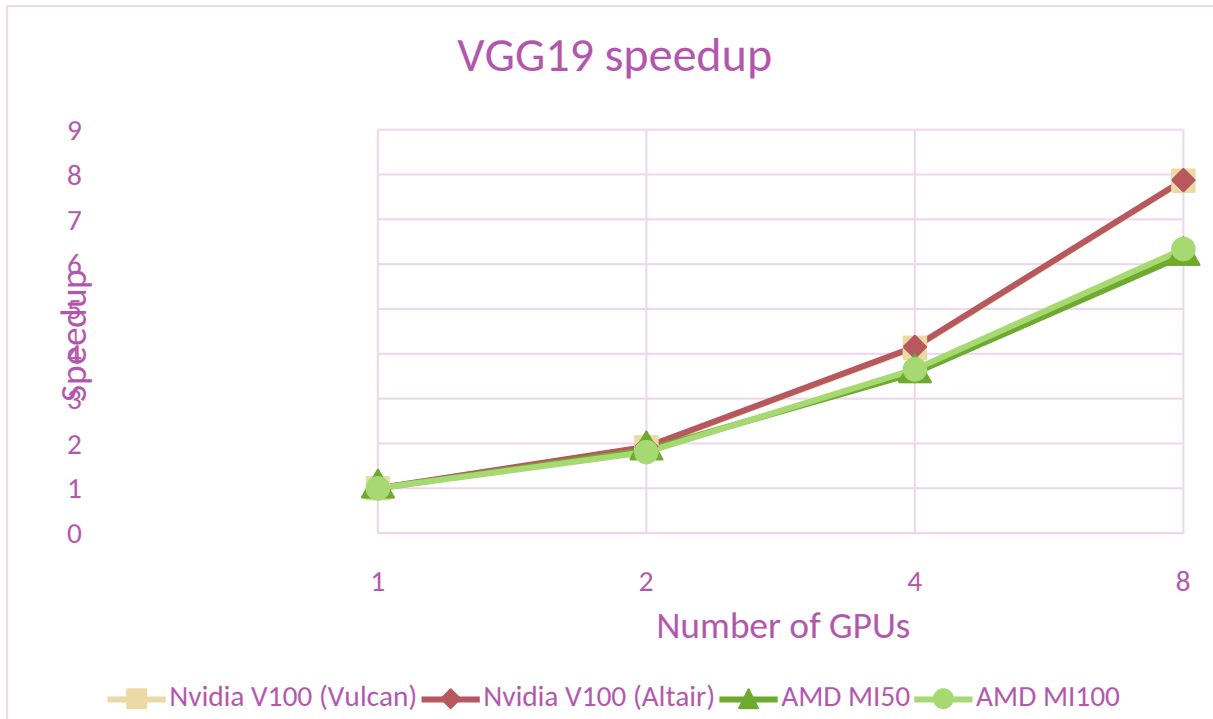


Figure 66. Training performance speedup for VGG19 neural model on selected architectures.

VGG model also scales linearly (Figure 65). This time it is AMD cards that perform better, while Nvidia GPUs scale better (Figure 66).

8.2.2.4 Mobilenet_v2

MobileNet-v2 is a convolutional neural network that is 53 layers deep. The MobileNet v2 architecture is based on an inverted residual structure where the input and output of the residual block are thin bottleneck layers opposite to traditional residual models which use expanded representations in the input. MobileNet v2 uses lightweight depthwise convolutions to filter features in the intermediate expansion layer. Additionally, non-linearities in the narrow layers were removed in order to maintain representational power. This model has been described in [45].

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies			Page:	128 of 174	
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

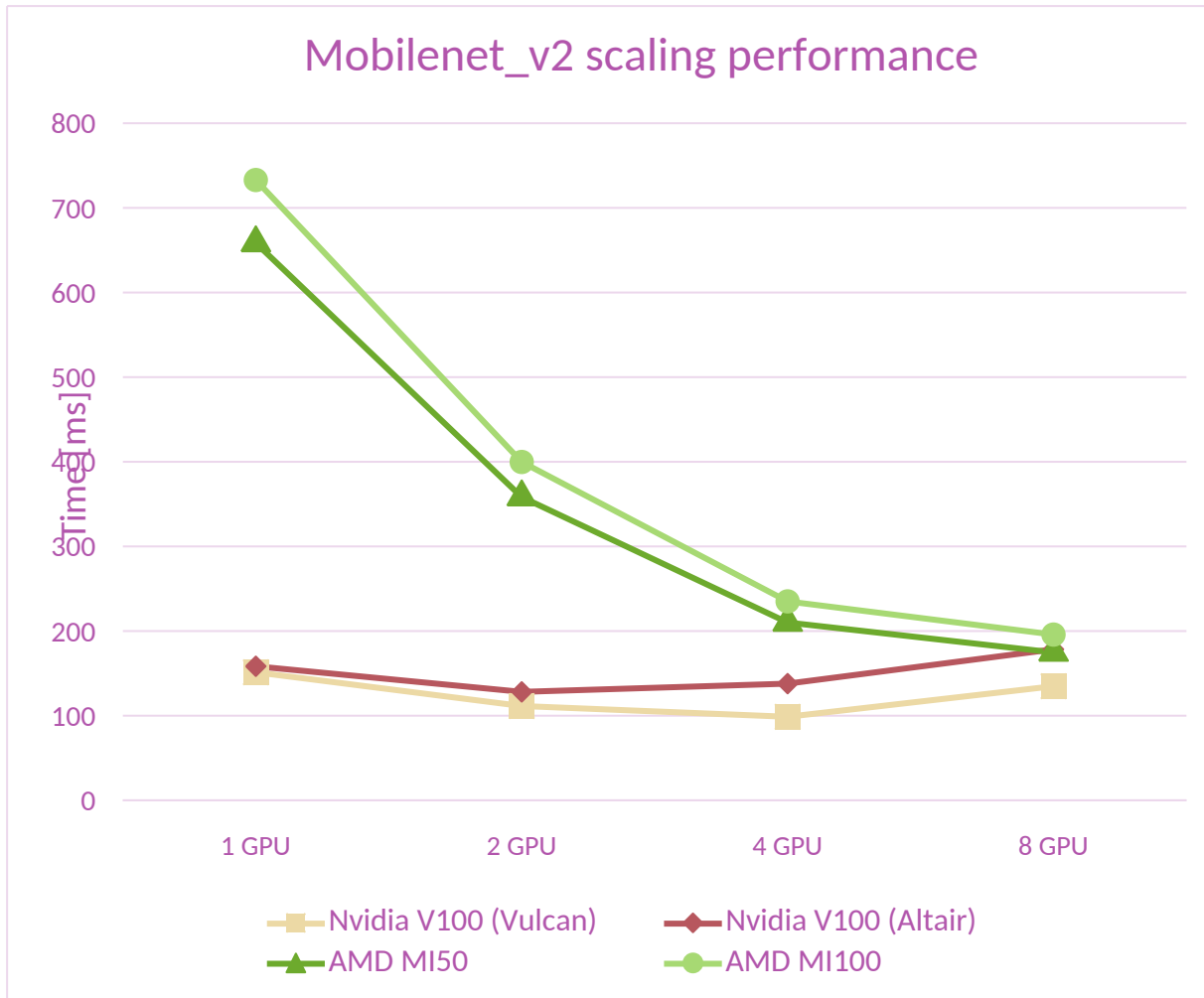


Figure 67. Training performance for Mobilenet neural model on selected architectures.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies			Page:	129 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0
				Status:	Final

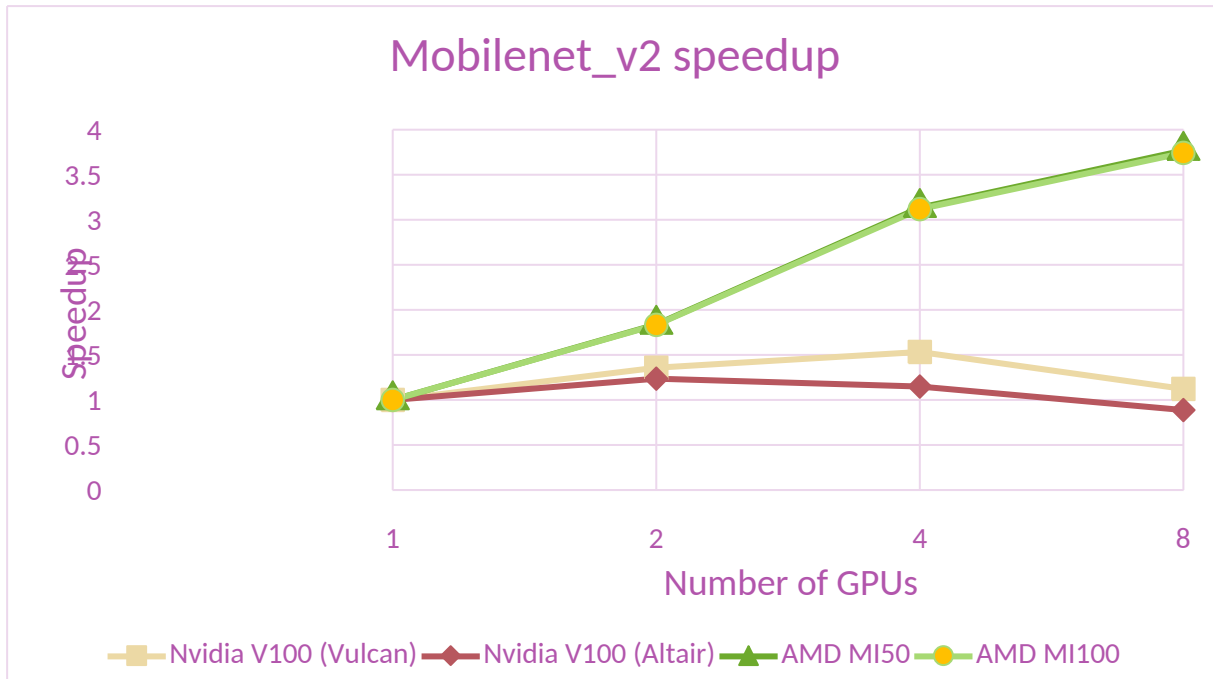


Figure 68. Training performance speedup for Mobilenet neural model on selected architectures.

Mobilenet, being the least complicated model, presents a different dynamic. Up to a point a linear speedup can be observed, but further scaling slows down the speedup. Again Nvidia GPUs achieve better times and this effect is more pronounced there.

8.2.3 Results and findings

The results show that model complexity correlates with the effect of GPU scaling, meaning that more complicated models gain more speedup from additional hardware than simpler ones.

What was also observed is that adding more GPUs comes with high data distribution penalty, which is only offset, when task is sufficiently large. Again, smaller models might be adversely affected by adding more processing units than is necessary.

Moreover, GPUs from the same manufacturer appear to follow the same trends when it comes to performance and scaling. It should be noted however, that even though GPUs are the same in case of Vulcan and Altair, other factors of the infrastructure (CPU, memory, hard drive) impact the results, explaining the differences in performance.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	130 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

8.3 GPU ML model training optimization

Here we present an overview of ways to increase neural model training performance when scaling out hardware. A variety of data distribution strategies were examined to show their implementation differences and efficiency.

8.3.1 Goal

The aim of this analysis is to determine the best possible utilization of GPU resources in large machine learning training tasks. To that end, several methods of distributing training job among multiple GPUs were analysed and compared.

8.3.2 Scenario description

There exist a number of training approaches varying in complexity and efficiency. They all use the same image classification model - ResNext101, but they differ in the way they distribute data among GPUs.

The setup for each test is similar to GPGPU benchmark:

- Input images have dimensions of 224x224x3, are made of randomized, double-precision data
- Batch size is 48
- 55 steps of both forward and backward propagation are performed

All data is being randomly generated and its size remains constant to ensure all cases are comparable.

8.3.2.1 No parallelism (single GPU)

In this simplest case training is being done using only one GPU on a single node. No additional measures are implemented to parallelize the execution of the code.

8.3.2.2 Data parallelism

This approach uses PyTorch's DataParallel [46] mechanism. It is a single-process multi-thread parallelism that allows executing the same model on multiple GPUs simultaneously. Because implementing it does not require additional code to set up process groups it is easy to use even for existing non-parallel projects.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	131 of 174	
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status:	Final

There are however some drawbacks. `DataParallel` cannot scale beyond one machine. It is also slower than distributed approaches even in a single machine with multiple GPUs due to global interpreter lock contention across multiple threads, the extra overhead introduced by scatter and gather operations and per-iteration model replication.

8.3.2.3 Distributed Data Parallelism (DDP)

This more advanced mechanism introduces multi-process parallelism, in which each GPU performs computations on its own dedicated process. Because of it, it requires process spawning and communications to be already set up, which often requires an extensive code rewrite. On the other hand, it works for both single- and multi-node training, making it suitable for particularly large tasks in supercomputing infrastructures.

It is important to point out that on its own, DDP comes with no workload distribution - each GPU is processing the entire dataset it is provided with. In order to address this issue we have tested 3 approaches.

8.3.2.3.1 DDP with no workload distribution

In this case all data sent to GPU is being processed. It is not being divided it into smaller chunks nor are there any other restrictions on processing applied. Since we are sending entire dataset to 8 accelerators, GPUs perform effectively 8 times the work.

8.3.2.3.2 DDP with Round-robin workload distribution

This approach introduces a check inside parallelized process function that tests whether a part of dataset is assigned to a particular GPU. All data is still being sent to all GPUs, but in this case only a chunk of it is being processed by each of accelerators.

8.3.2.3.3 DDP with Distributed Sampler

Here we introduce Distributed Data Sampler mechanism, which manages dividing the dataset into samples and parallelized workload distribution. This requires moving Data Loader logic into parallelized process function. However, thanks to it, only needed chunks of data are being sent to GPUs.

8.3.3 Tests

Tests were conducted on a GPU equipped nodes on Vulcan cluster at HLRS. More details have been described in section 3.2.2 of deliverable 5.5.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	132 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

All training runs were performed on ResNext101_32x8d neural model (for more details see section 8.3.2). Also, unless specified otherwise, tests were run with 8 GPUs on a single node. In each case two distinct stages can be observed: preparation phase and computation phase. Preparation phase consists mainly of lengthy data transfers to GPU and during computation phase the actual forward and backward passes of neural model take place. These phases have been marked on figures.

8.3.3.1 No parallelism (single GPU)

Training with no parallelism produced the following results:

- Total duration - 154s
- Preparation phase - 10s

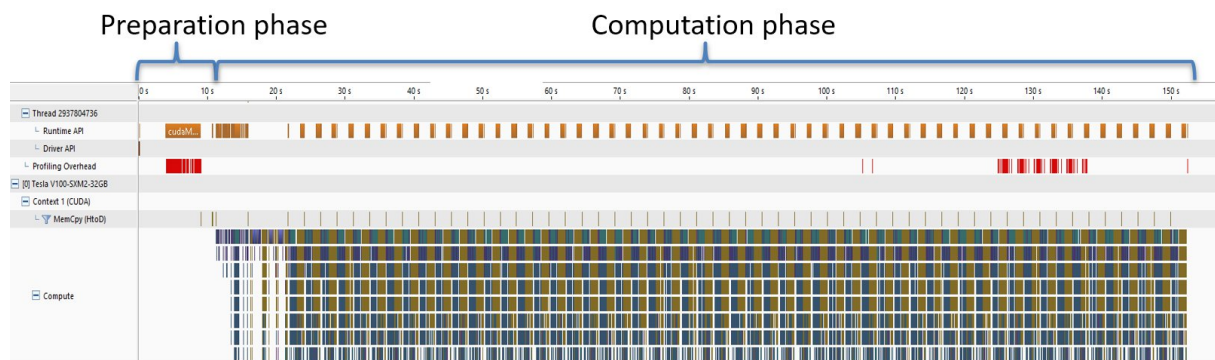


Figure 69. Training run without parallelism seen on GPU profiler tool.

In the simplest, sequential scenario a majority of the time is spent on computation tasks. Preparation phase comprises only a small fraction (around 10s) of total runtime duration (154s).

8.3.3.2 Data parallelism

Training with data parallelism generated the following times:

- Total duration - 105s
- Preparation phase - 46s

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies	Page:	133 of 174
Reference:	D3.5	Dissemination:	PU
	Version:	1.0	Status: Final

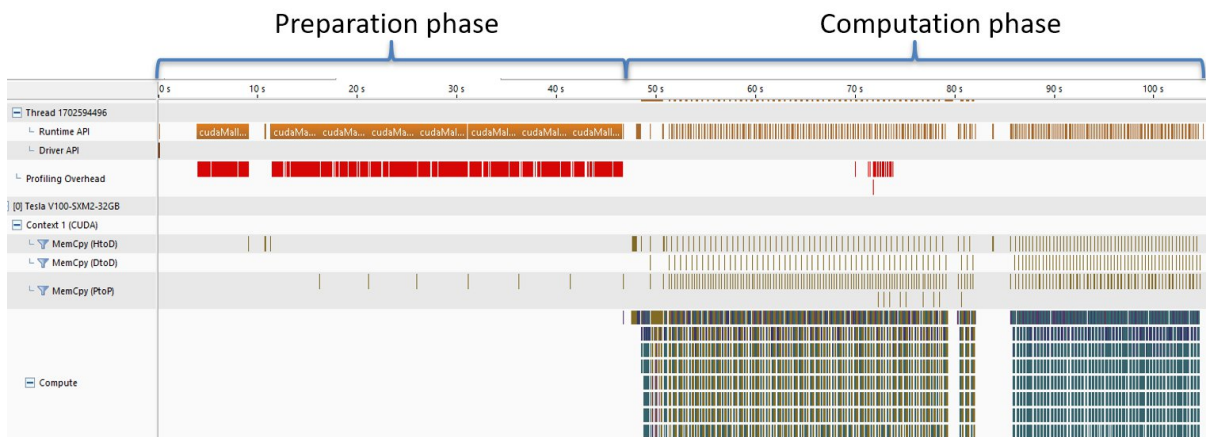


Figure 70. Training run with data parallelism seen on GPU profiler tool.

With data parallelism a significant speedup in total execution time can be observed. However, preparation phase is now comparatively longer, making up 46s out of total 105s run time. This is a result of lengthy memory transfer operations to GPUs being sequential, which can be explained because of Python's Global Interpreter Lock limitation.

8.3.3.3 Distributed data parallelism with no workload distribution

Training with distributed data parallelism without workload distribution produced the following results:

- Total duration - 178s
- Preparation phase - 28s

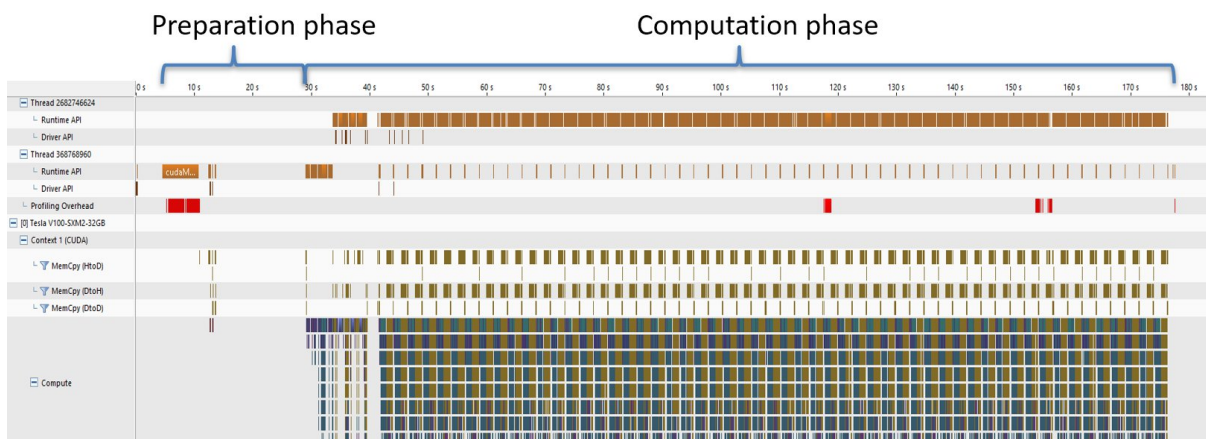


Figure 71. Training run with distributed data parallelism seen on GPU profiler tool.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies	Page:	134 of 174
Reference:	D3.5	Dissemination:	PU
	Version:	1.0	Status: Final

With DDP a major reduction in preparation phase occurs (28s from 46s), as now all the data transfers to GPUs happen simultaneously. On the other hand, overall execution time is extended (178s). This is explained by no workload distribution - each GPU is effectively performing 8 times the work by processing all the dataset instead of its chunks.

8.3.3.4 Distributed data parallelism with Round-robin workload distribution

Training with distributed data parallelism with round-robin data distribution resulted in the following times:

- Preparation phase - 29s
- Total execution time - 57s

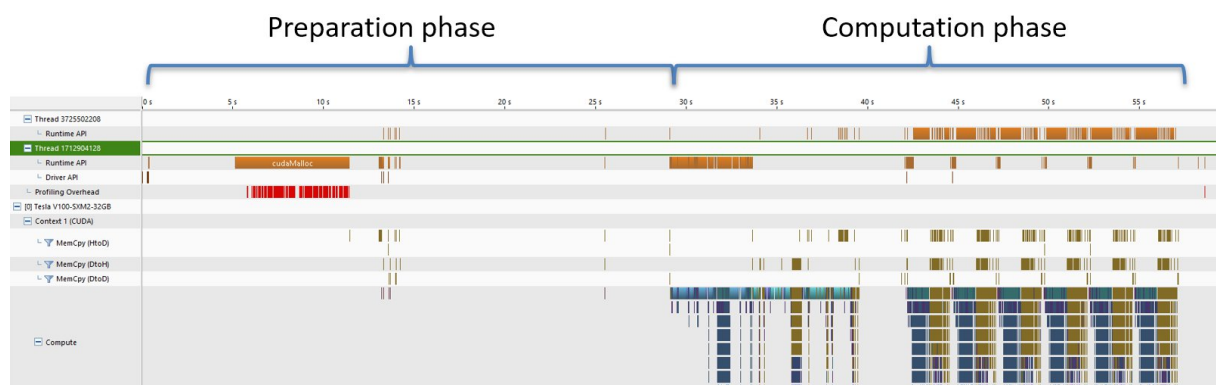


Figure 72. Training run with DDP and round-robin workload distribution seen on GPU profiler tool.

In this scenario, all data is still being sent to all GPUs, but not all of it is being processed. As a result, there is a significant speedup of execution time. Nonetheless, extraneous data is still being transferred.

8.3.3.5 Distributed data parallelism with DistributedSampler mechanism

Training with distributed data parallelism with DistributedSampler returned with the following results:

- Preparation phase - 21s
- Total execution time - 47s

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies			Page:	135 of 174		
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status:	Final

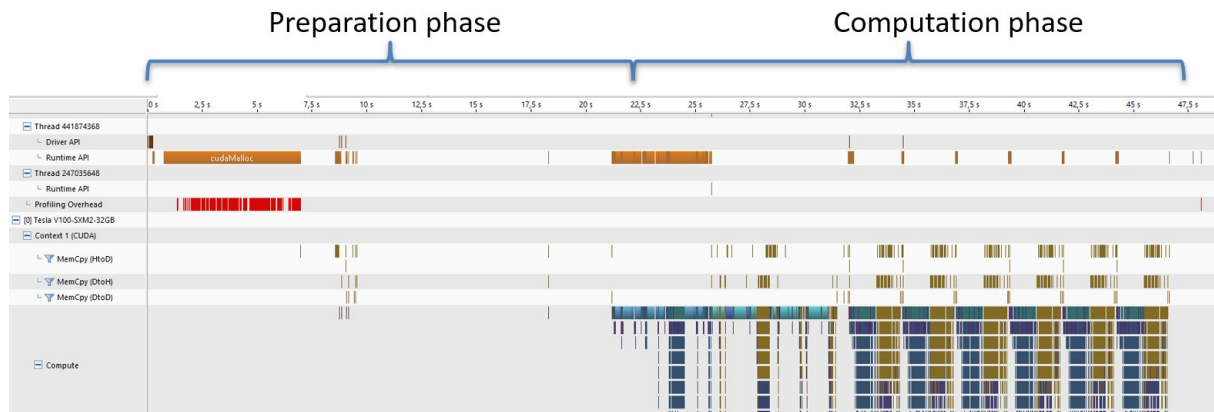


Figure 73. Training run with DDP and distributed sampler seen on GPU profiler tool.

In a final approach unnecessary data is no longer sent to all the GPUs. Owing to that, preparation phase can be further reduced (21s). This approach provides best performance for training neural models out of all strategies tested.

8.3.4 Analysis

The combined results for all the scenarios are shown in Figure 74.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies			Page:	136 of 174	
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

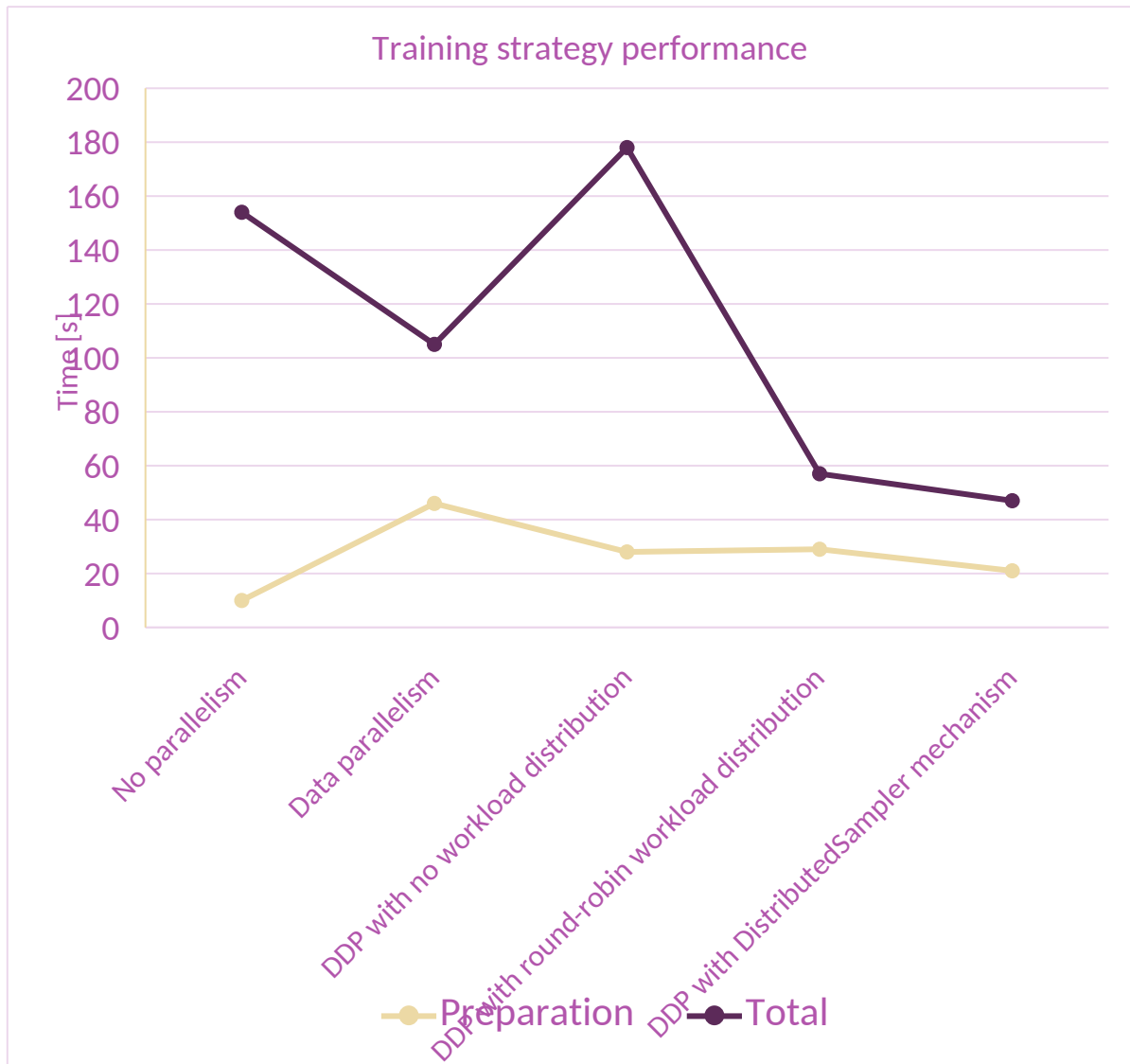


Figure 74. Performance of different training strategies.

The analysis of the results allows us to draw several conclusions. Easy to implement Data Parallelism already offers a sizeable run time reduction of model training (32%). In contrast, more advanced approaches offer even better results (70% run time reduction) at the cost of more complicated implementation.

Finally, distributing data to multiple GPUs requires some overhead, but it is overcome for larger jobs by gains in computing performance (16% run time increase for 800% work increase in case of DDP).

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies			Page:	137 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0
				Status:	Final

9 Data management

Data Management System (DMS) is an indispensable part of the workflow for Global Challenges (GC) systems. In the chapter we analyse influence of other protocols on CKAN [47] data transfer performance.

Data repository folder:

```
https://gitlab.com/eu_hidalgo/benchmarking/-/tree/master/deliverable_3_5/  
data_management
```

9.1 CKAN enhancement

The purpose of CKAN is to register datasets, to facilitate the search of datasets, and finally to provide access to these datasets. Besides providing these core functionalities related with data storing, CKAN also allows grouping of datasets, creating organizations, metadata management, and relationship management of data, it can handle different data formats, can harvest external data sources and provides a shared pool of data where all can benefit from the publicly available collection of data of other users which covers most if not all of the pilots' requirements. CKAN is integrated with the HiDALGO SSO Keycloak instance.

We have noticed that the processes of data uploading with the CKAN API takes a long time and needs more server resources (CPU, RAM, CACHE on disk) than to transfer the data with file transfer protocols like FTP, SCP. It is especially visible for single files larger than 2GB.

We have installed and configured the GridFTP server on the CKAN instance. We have also configured a web-server to serve the data and created dedicated scripts to upload data to the CKAN.

The data is uploaded to the GridFTP server with unique file name. The data is saved in the dedicated "ckan" directory in the users' home dirs. After the transfer is completed, the scripts create a new resource in the given dataset in the CKAN.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	138 of 174	
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status:	Final

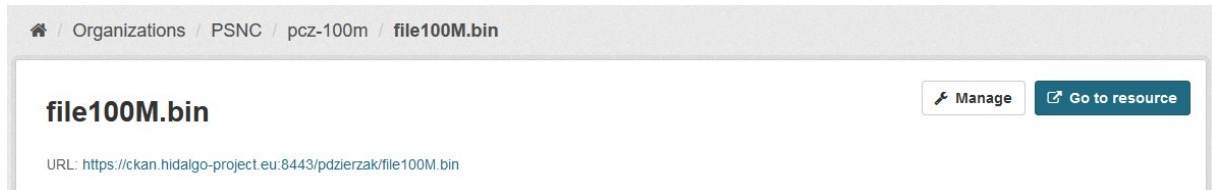


Figure 75. The resource file100M.bin uploaded by GridFTP.

To download the data just click on the URL or use ckan-client, wget, curl etc. If the data is located in the user's home directory, GridFTP or SCP client can be used as well.

Uploading data via GridFTP and SCP protocols requires additional authorization, which can be done as described in chapter 9.2.

9.2 Benchmarks

The purpose of the tests is to check the efficiency of the data management system by sending files of various sizes with a total size of 100GB from three different locations: PSNC, PCz (Czestochowa University of Technology) and HLRS. Performance tests were performed sequentially and in parallel mode. In order to determine the efficiency of the proposed solution, the following procedure has been proposed:

1. make a traceroute to the CKAN/GridFTP server,
2. test the upload and download speed with the iperf3 tool,
3. upload 100GB of 100MB, 1GB, 2GB, 5GB and 10GB files (repeat 10 times for each files) using GridFTP, SCP and CKAN API (curl tool),
4. upload data in sequential and parallel mode (from 3 locations for each protocol).

Software	CKAN [PSNC]	Client [PSNC]	Client [PCz]	Client [HLRS]
OS	Ubuntu 20.04.3 LTS	Ubuntu 18.04.6 LTS	Debian 10	Ubuntu 18.04.6 LTS
vCPU	8	8	2	2
RAM	16GB	16GB	2GB	7GB
Disk	300GB and 2TB	40GB and 1TB	32GB	30GB

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies			Page:	139 of 174	
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

CURL	7.68.0	7.58.0	7.64.0	7.58.0
Globus-GridFTP toolkit	6.21	6.21	6.21	6.21
SSH/SCP	OpenSSH_8.2p1	OpenSSH_7.6p1	OpenSSH_7.4p1	OpenSSH_7.6p1
CKAN	2.9.4 / Python 3.8	N/A	N/A	N/A
PostgreSQL	12.8	N/A	N/A	N/A

Table 34. Software stack for CKAN benchmark.

The table below presents the route of packets from PSNC to the CKAN virtual machine located in PSNC:

HOST	PSNC	Loss%	Snt	Last	Avg	Best	Wrst	StDev
1.	62.3.171.150	0.0%	4	0.7	0.8	0.5	1.1	0.2

Table 35. Route from PSNC to the CKAN.

The table below presents the route of packets from PCz to the CKAN virtual machine located in PSNC.

HOST	PCz	Loss%	Snt	Last	Avg	Best	Wrst	StDev
1.	212.87.229.129	0.0%	4	19.4	10.1	0.3	20.2	11.2
2.	10.2.12.1	0.0%	4	0.3	0.3	0.2	0.6	0.2
3.	150.254.255.177	0.0%	4	6.4	6.4	6.3	6.4	0.0
4.	150.254.255.76	0.0%	4	6.4	6.4	6.4	6.4	0.0
5.	150.254.166.89	0.0%	4	6.7	36.0	6.5	124.3	58.9
6.	10.0.20.3	0.0%	4	7.0	43.2	7.0	151.7	72.3
7.	62.3.171.150	0.0%	4	8.1	8.1	7.9	8.2	0.1

Table 36. Route from PCz to the CKAN

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	140 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

The table below presents the route of packets from HLRS to the CKAN virtual machine located in PSNC.

HOST	HLRS	Loss%	Snt	Last	Avg	Best	Wrst	StDev
1.	141.58.0.254	0.0%	4	0.5	0.6	0.5	0.7	0.1
2.	141.58.252.110	0.0%	4	1.1	0.9	0.7	1.1	0.2
3.	141.58.252.2	0.0%	4	0.7	0.8	0.7	1.0	0.1
4.	193.197.63.12	0.0%	4	2.3	2.2	2.1	2.4	0.1
5.	129.143.60.76	0.0%	4	2.9	3.1	2.9	3.3	0.2
6.	129.143.60.113	0.0%	4	6.3	6.5	6.3	6.7	0.2
7.	62.69.146.103	0.0%	4	5.9	6.3	5.9	6.7	0.3
8.	109.105.98.124	0.0%	4	19.6	16.0	14.7	19.6	2.4
9.	109.105.98.125	0.0%	4	23.3	23.2	23.0	23.3	0.2
10.	212.191.224.18	0.0%	4	26.4	25.4	25.0	26.4	0.7
11.	150.254.255.76	0.0%	4	25.0	25.0	25.0	25.2	0.1
12.	150.254.166.89	0.0%	4	25.1	25.2	25.1	25.4	0.2
13.	???	100%	4	0.0	0.0	0.0	0.0	0.0
14.	62.3.171.150	0.0%	4	26.7	26.8	26.7	26.9	0.1

Table 37. Route from HLRS to the CKAN.

The route from HLRS is the longest one, while in PSNC the CKAN and the client are in the same data centre.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	141 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

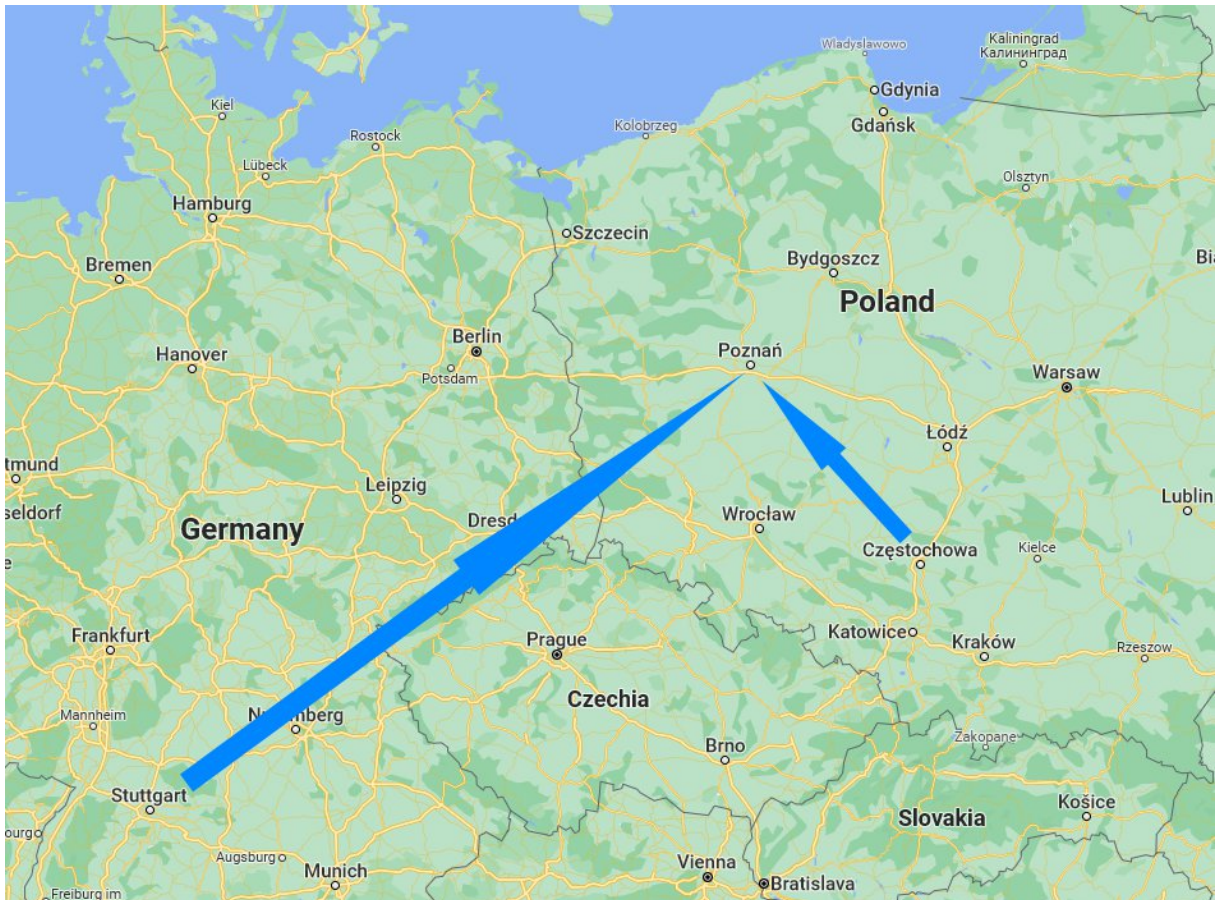


Figure 76. Europe map with location of data centres taking a part in testing procedure: Poznań (PSNC), Częstochowa (PCz) and Stuttgart (HLRS).

In order to determine connection bandwidth iperf3 tests were determined for upload and download transfer. Achieved results are:

- PSNC <-> PSNC: AVG **621 Mbits/sec**
- PSNC <-> PCz: AVG **148 Mbits/sec**
- PSNC <-> HLRS: AVG **72.9 Mbits/sec**

As expected, both the transmission speed and the latency were the best for PSNC local transfer. By default, files with a size of 10MB can be uploaded via the CKAN API. For the purposes of the HiDALGO project, we have introduced modifications to the CKAN and Nginx configuration to improve servers performance. As a consequence of set of tests we increased the following buffers: `post-buffering`, `ckan.max_resource_size`,

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies			Page:	142 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0
				Status:	Final

client_max_body_size and timeout limits: proxy_connect_timeout, proxy_send_timeout, proxy_read_timeout. We have also set the enable-threads option in CKAN to true and increased our storage capacity for cache and storage.

After iperf3 and traceroute benchmarks, we have performed a sequential tests of upload data with GridFTP, SCP and CURL / CKAN API. We have uploaded 100GB of files 100MB, 1GB, 2GB, 5GB and 10GB.

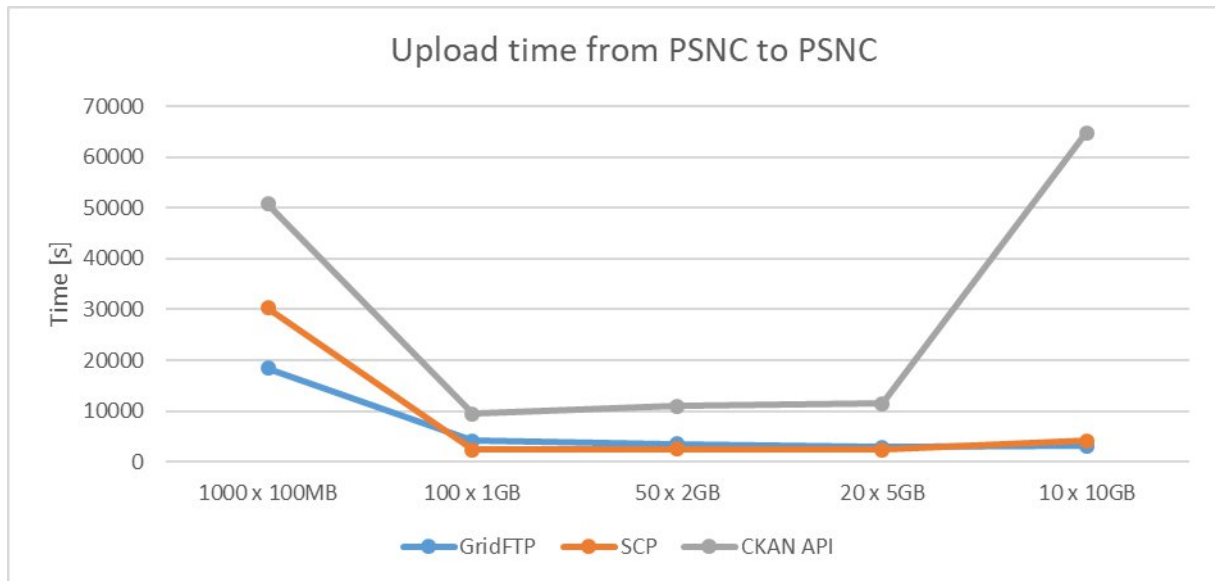


Figure 77. Upload time from PSNC to PSNC.

The results for GridFTP and SCP for files 1GB and larger are very close. For files of size 10GB, SCP is slightly faster than GridFTP. The results for CURL / CKAN API are 3 times slower compared to GridFTP and SCP.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies			Page:	143 of 174		
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status:	Final

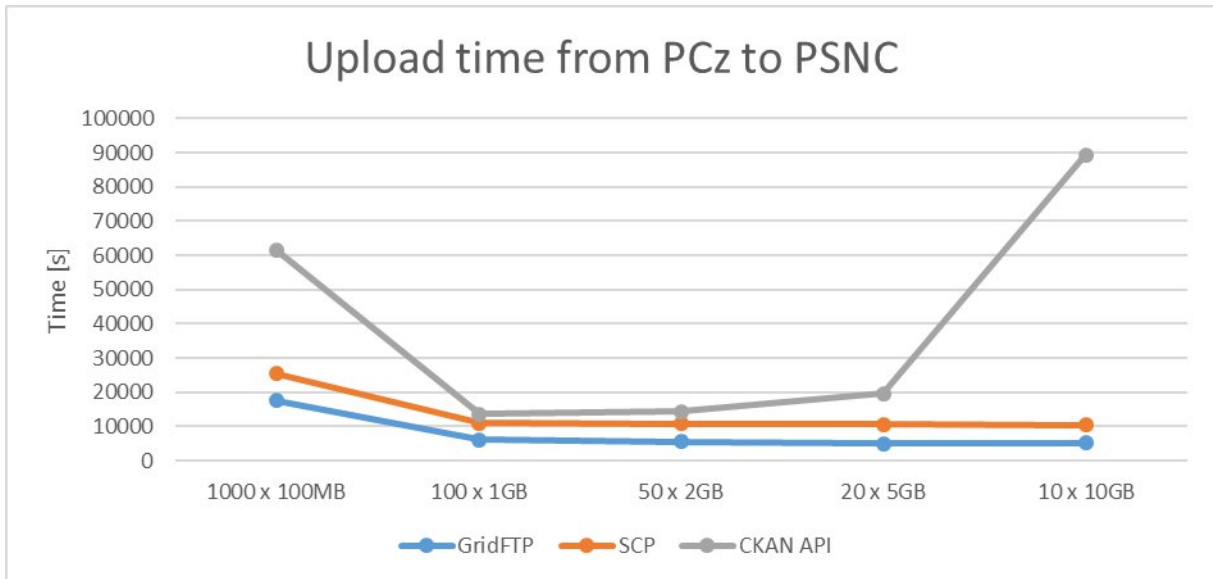


Figure 78. Upload time from PCz to PSNC.

The results for GridFTP and SCP are close. The CURL / CKAN API is significantly slower than with PSNC. The worst results are obtained for the CKAN API method and a large number of small files or large files (larger than 5GB).

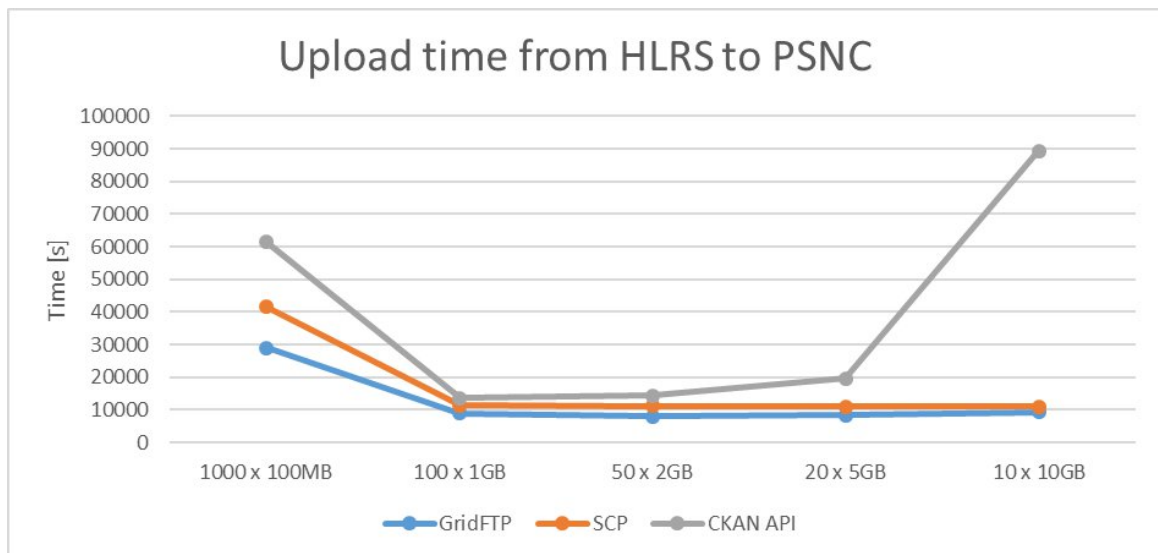


Figure 79. Upload time from HLRS to PSNC.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies			Page:	144 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0
				Status:	Final

The results for GridFTP, SCP and CURL / CKAN API are close for files 1GB and 2GB. The CURL / CKAN API method is significantly slower than with PCz.

For all locations the GridFTP is the fastest method. The slowest method is CURL / CKAN API, because of caching data in the web server. The best results are obtained with 1 - 10 GB files, using the GridFTP and SCP protocols. In the case of 100MB files, there is a large overhead on communication with the CKAN API and creating data stores. We noticed that CKAN slows down with the number of objects in datasets.

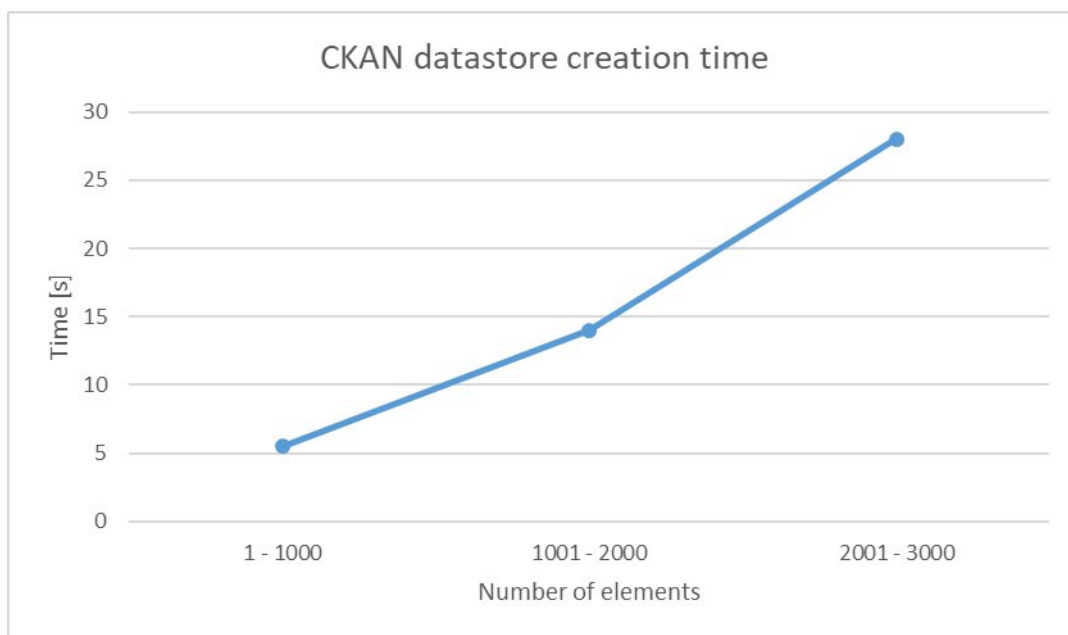


Figure 80. CKAN datastore creation time.

For the first 1000 elements in the dataset the creation datastore time is about 1 - 11 s. For the next elements, this time increases linearly.

After performing the sequential tests, we selected the top 3 results to compare them with the parallel mode transfer. We collected times for sequential and parallel data transfer, a total of 300 GB (100 GB for each of the 3 locations).

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies			Page:	145 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0
				Status:	Final

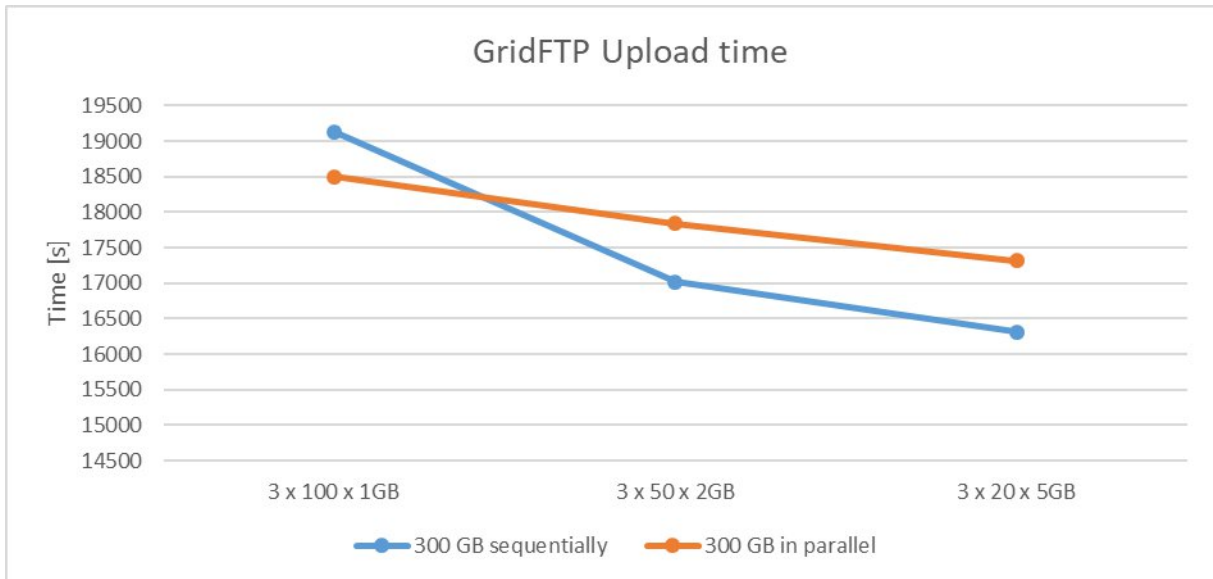


Figure 81. The results of sequential and in parallel upload data using GridFTP.

For the GridFTP the parallel method is faster for files 2 GB and 5 GB. The transfer time is faster than SCP method. As the file size increases, the transfer time decreases faster than for SCP.

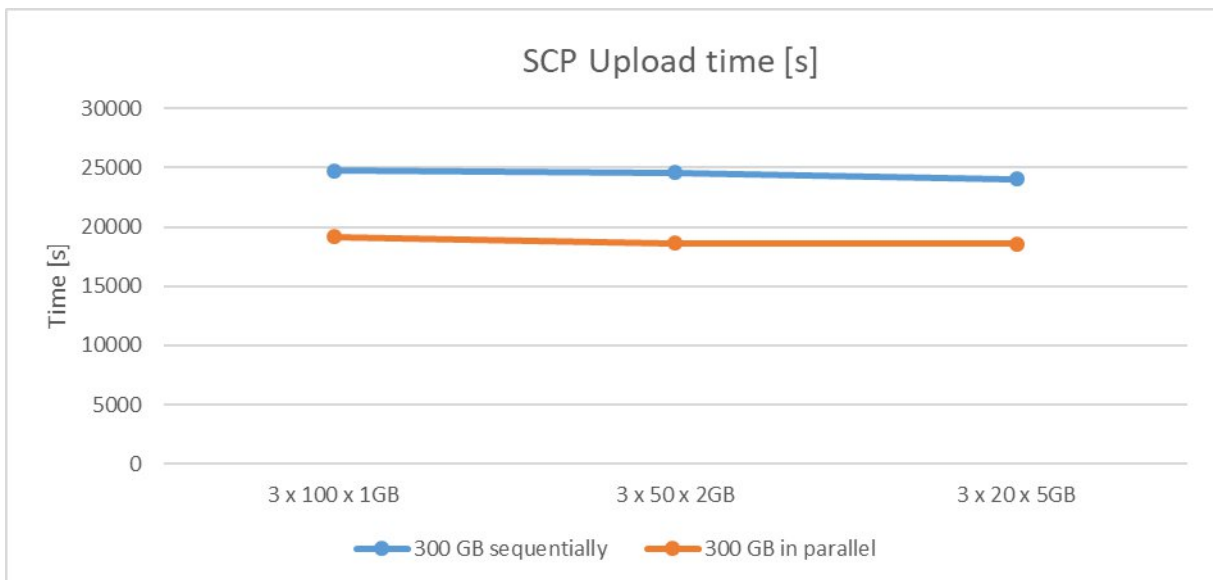


Figure 82. The results of sequential and in parallel upload data using SCP.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	146 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

For the SCP the parallel method is faster than sequential for all files. The transfer with SCP protocol is very stable and reproducible. In the most Linux systems We don't need to install additional software like GridFTP client.

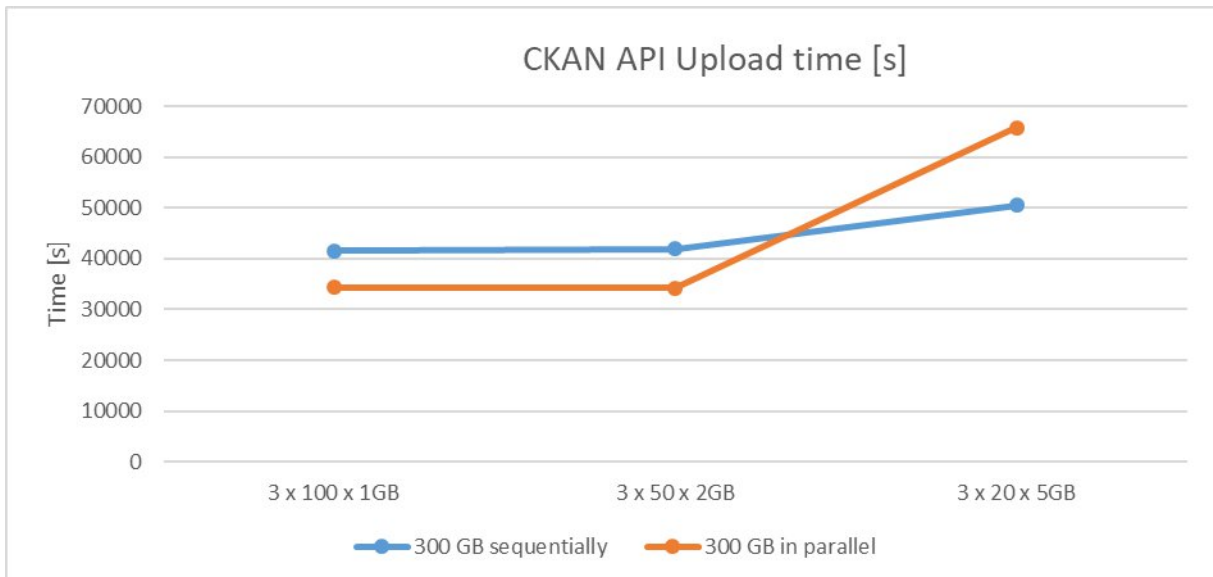


Figure 83. The results of sequential and in parallel upload data using CKAN API.

For the CKAN API the parallel method is faster than sequential for files 1GB and 2GB. Unlike SCP and GridFTP methods, the transfer time using the CKAN API increases with the size of the transferred file.

Generally, the best method for uploading large amounts of data is the GridFTP and SCP protocol. All methods can be successfully used for small amounts of data (up to 10GB). In the case of CKAN API, the data is additionally cached by Nginx proxy and Supervisor. As the data size increases, the performance of this solution decreases.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	147 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

10 Data security

The HiDALGO user accounts are defined at Keycloak IDM [48]. Using SSO (single sign on) mechanism there is the possibility to log in to the CKAN portal [49] and other services that support the SSO.

In HiDALGO, the orchestrator is tasked to deploy jobs on HPC or VMs. It requires either SSH login (user/password combinations) or credentials like API tokens.

The used orchestrator is Croupier, a Cloudify plugin. Since Cloudify doesn't offer multi-tenancy in their community version, any user that logs in must do it with the admin account, which can see everything in Cloudify, including deployment inputs, Cloudify secrets, etc. A method of hiding users' secrets from other users, and ideally from administrators as well is required.

Another requirement is that the method used must not require anything be installed or configured on the remote machines or services that are protected by these credentials, since a general method needs to be developed and most infrastructures don't allow a modification to their authentication methods. Therefore, any solution must allow the orchestrator to use the infrastructures with the normal access methods (private-key, password, API token, etc.).

10.1 Vault Integration

A solution is proposed to manage user's credentials using Hashicorp's Vault [50] as the main method to store them. Every functionality in Vault is defined by a URL: any service, any secret engine, has a URL through which it is accessed. For example, policy management is accessed through /sys/policies, therefore if a user wants to create a policy named ssh-username, they would send a POST request to `http://<vault_address>/v1/sys/policy/ssh-username`.

Secrets are also identified by their URLs, for example, a secret stored in the /ssh secret engine might be located at /ssh/john/hawk.hhrs.de. To access said secret a user would have to direct a GET request to `http://<vault_address>/v1/ssh/john/hawk.hhrs.de`.

In order to control who can access which URL, Vault uses policies and tokens. Every call made to Vault's API must contain an access token in its header. Every token has a number of policies attached to it. The policies attached to a token determine which URLs the token can access.

The Vault instance in HiDALGO has been configured to allow authentication via jwt, by making use of Keycloak. When user wants to interact with Vault, they need to send their active JWT

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	148 of 174	
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status:	Final

(received from Keycloak when they log-in) and Vault will return a token with the policies defined in the user's role in Vault. The user's role in Vault is just a configuration that maps information on the JWT to the policies that need to be granted to the generated Vault token. Therefore, to allow users authenticated through Keycloak to access private access to their secrets in Vault permanently, a system needs to be setup so that a new role and a new policy is generated for each secret that gets uploaded to Vault, to allow access to that secret to the user that uploaded it. For the Horizon2020 project SODALITE a component was developed that does just that, it is called vault-secret-uploader and is a Flask API that allows user to upload secrets to Vault, creating the necessary policies and roles so that the user can access them when they authenticated using the JWT method.

Vault provides another method so that certain secrets are only accessible by one user, called cubbyhole. Every Vault token has their own cubbyhole secret engine such that every secret uploaded there using a token can only be accessed by that same token. This cannot be used in conjunction with JWT authentication since every time a user is authenticated into Vault using JWT a new Vault token is generated and therefore the cubbyhole is reset. However, the cubbyhole method can be used in HiDALGO for temporary secret storage.

There are 3 different ways Hidalgo allows users to manage their HPC credentials, each with its own benefits and disadvantages. These 3 methods are listed as follows, from least secure to most secure (and from most convenient to least convenient):

- Permanent storage of the users main credentials - using this method, a user's credentials are stored permanently on Vault. Therefore, they would only need to upload their credentials once to HiDALGO's platform and they can be used by the orchestrator until the user decides to delete them. The users would upload their credentials through the portal, which would use vault-secret-uploader to register the user in vault (create role/policy) and upload the credentials to Vault. Once the users have their credentials safely stored in Vault, they are used in Croupier in the following way.
- Permanent storage of a user's secondary credentials - This method is very similar to the first one, it can only be used to manage SSH credentials (HPC or VM credentials, for example). The main difference with this method is that the user doesn't provide the Portal with their full HPC or VM credentials, they only give the Portal the username and host of each HPC or VM they need to save the credentials for. The Portal generates a new key pair, it saves the private key in Vault using the same method as described previously (through the vault-secret-uploader). It then gives the user the public key. The user must then connect to the HPC or VM by themselves to upload this public key

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	149 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

to allow croupier to connect to the HPC in the name of the user. The main benefit of this method is that if Vault or HiDALGO is ever compromised, the user can simply remove the public key from the HPC and doesn't need to get in contact with the HPC to change their primary keypair. It also allows users that have hardware methods of authentication or other methods that cannot be easily uploaded to Vault to use the HiDALGO platform. The drawback is that the user must do some configuration actions outside the control of the portal, so it is reserved for experienced users.

- Temporary storage of the user's credentials - In this method the vault-secret-uploader is not used. Every time the user wants to run an application, they have to provide their credentials in the portal. The portal generates a Vault token with access to only that token's cubbyhole and stores the credentials in that cubbyhole. The portal then runs the application, passing to Cloudify the token where the secrets are stored. Croupier downloads the credentials for all the infrastructures it needs to connect to before the token expires and the secrets are deleted. This is the most secure option of all, since the secrets are only stored on the HiDALGO platform for a maximum of 10 minutes. The drawback is that the user needs to upload their main credentials every time. It also cannot be used by users who have hardware private keys. Even if a system could be made that combined this method and the previous one, where a private key generated by the portal is saved on a cubbyhole, we can't rely on the user to upload their public keys to the HPCs in time before the token expires.

In our opinion the “permanent storage of a user's secondary credentials” method is an optimal solution for HiDALGO users and HPC / VM environments. All the operations related with the portal and Cloudify will use the secondary credentials (user will not know the private key). The public key will only be transferred to the HPC / VM environment once. Users will also be able to use their primary credentials to log into HPC / VM environments for development and testing. To use the portal, users will always have to log in using HiDALGO IDM. However, since all 3 methods have their use cases, all 3 will be implemented in the Portal.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	150 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

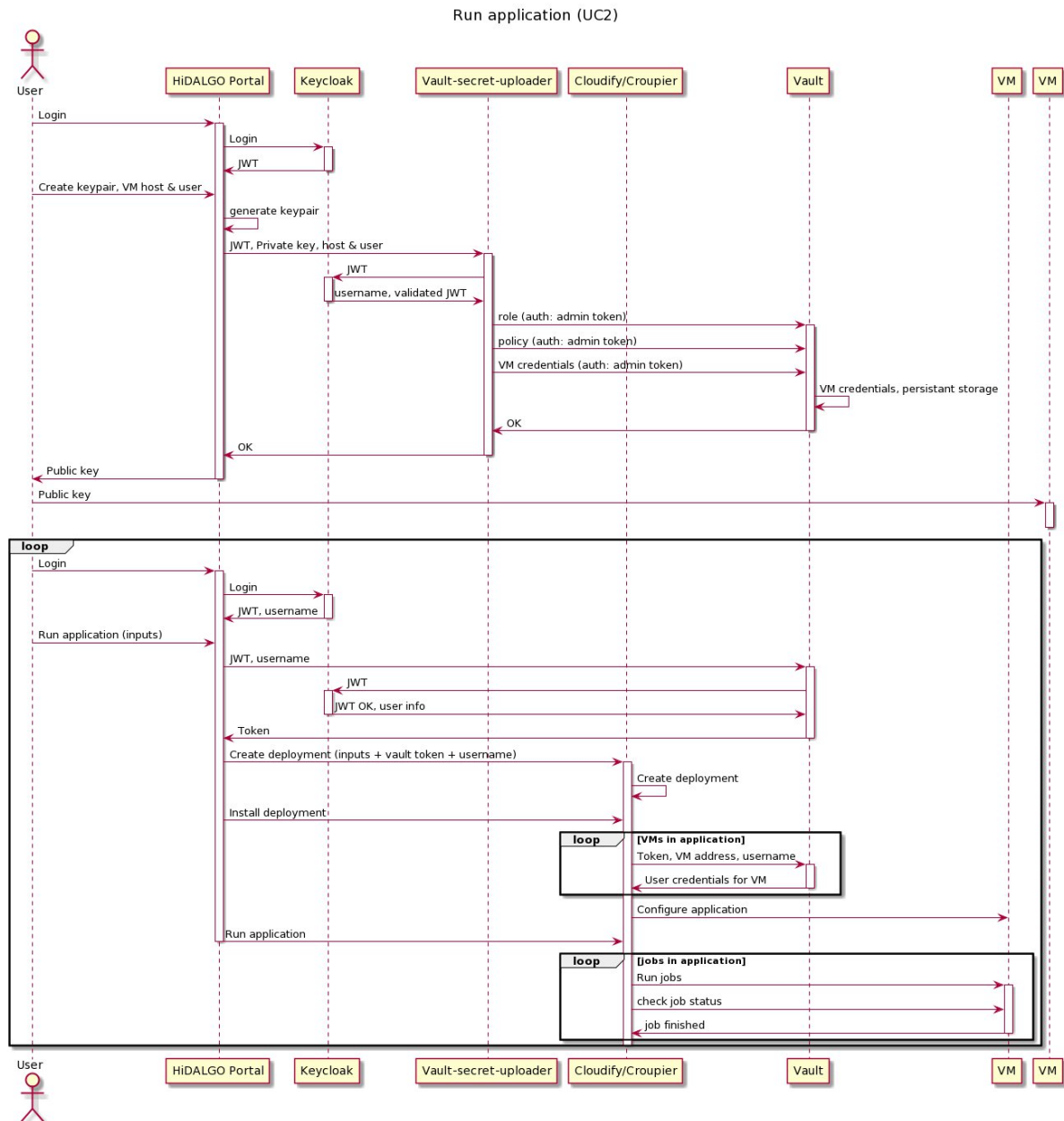


Figure 84. A sequence diagram of a user's secondary credentials in the Vault.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies	Page:	151 of 174
Reference:	D3.5 Dissemination:	PU	Version: 1.0
		Status:	Final

11 Visualization

The chapter describes most recent achievements in the implementation of HiDALGO visualization tools (COVISE and Visualiser) extensions. As part of the completed tasks, a number of additional visualization extensions were prepared, from presenting specific properties to AI solutions. Moreover, the set of performance tests were conducted to assess the benchmarking results.

Data repository folder:

https://gitlab.com/eu_hidalgo/benchmarking/-/tree/master/deliverable_3_5/visualization

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	152 of 174	
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status:	Final

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9

11.1 COVISE

11.1.1 Extension implementation

Several extensions were implemented to improve COVISE for the visualization of Urban Air Pollution simulation results. Besides the provision of a web-accessible visualization, also a more extensive model for presentations in the CAVE was developed. This enables the combination of the simulation data with further data sets, such as geographic imagery and building models. The model was implemented using Vistle, the highly parallel successor of COVISE. By its extended parallelism, Vistle also enables the processing of larger data sets than COVISE. The resulting visualization can be best explored in a CAVE.

The visualization is aimed at simulation results obtained from OpenFOAM. Results are typically stored in either native OpenFOAM format or converted into Enight format. Thus, the import of files in these formats had to be implemented. This was realized as COVISE/Vistle processing modules (“ReadEnight” and “ReadFoam”). As the handling of COVISE and Vistle is similar, only one tool, COVISE, will be described in the following. However, it is also applicable to Vistle.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	153 of 174	
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status:	Final

COVISE already provides plenty of processing modules to extract inter alia iso surfaces, cutting surfaces or streamlines from a data set. The module pipeline in Figure 85 shows an example of a workflow of processing modules in COVISE. Further modules were added to provide additional functionalities, such as the computation of cutting surfaces parallel to a given ground terrain.

As mentioned above, there is also the option to extract HTML files from COVISE which allows an interactive model exploration that can be integrated on websites.

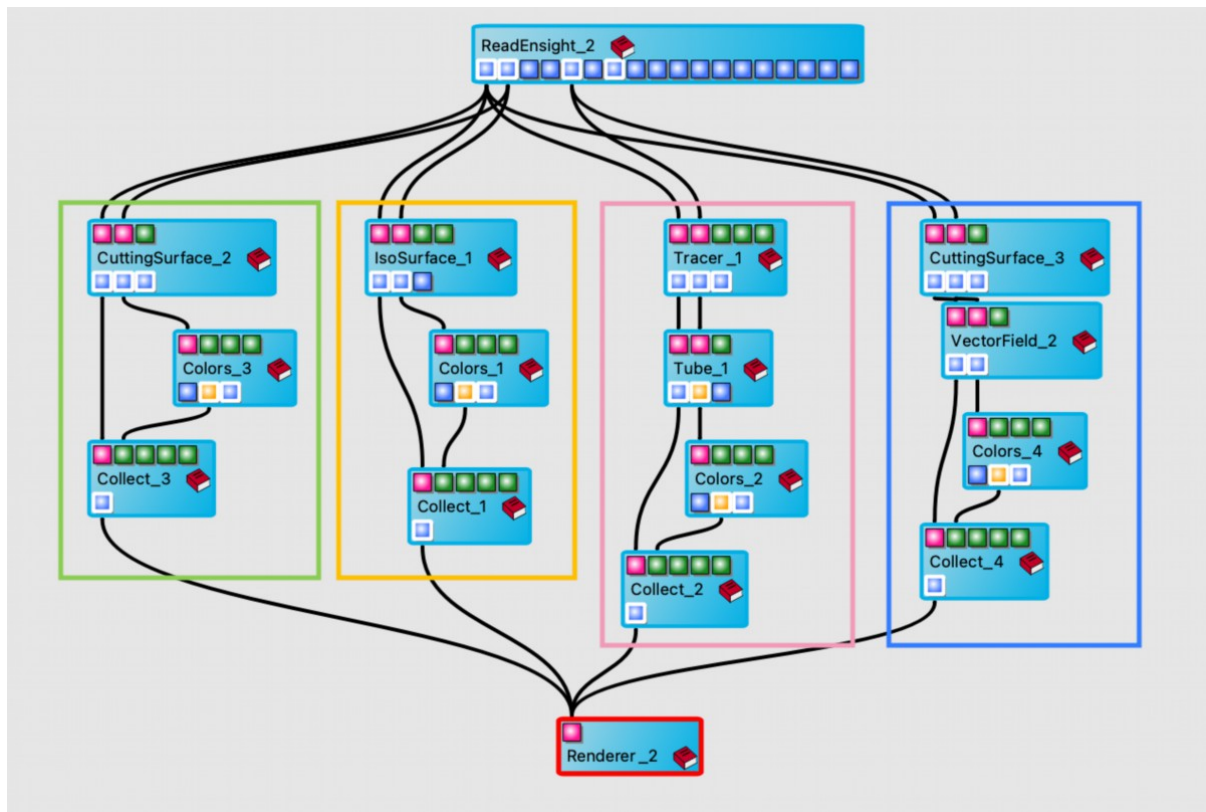


Figure 85. The module map in COVISE for visualizing UAP simulation data. Coloured boxes indicate: cutting surface (green), iso surface (yellow), streamlines (rose), vector field (blue).

11.1.2 Application & Results

The COVISE visualization workflow was applied to the data derived from the Urban Air Pollution use case. The two application domains considered in the simulations were the city centre of Stuttgart, Germany and the city of Győr, Hungary. The latter was prepared as a stand-alone model using COVISE, while the former was combined with other data for the region of Stuttgart using Vistle. However, both data sets can be handled with the respective other tool as well.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies			Page:	154 of 174		
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status:	Final



Figure 86. Screenshot of the interactive visualization. Iso surface for NO_x are shown along with a city model (Stuttgart building scans).

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies		Page:	155 of 174			
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status:	Final

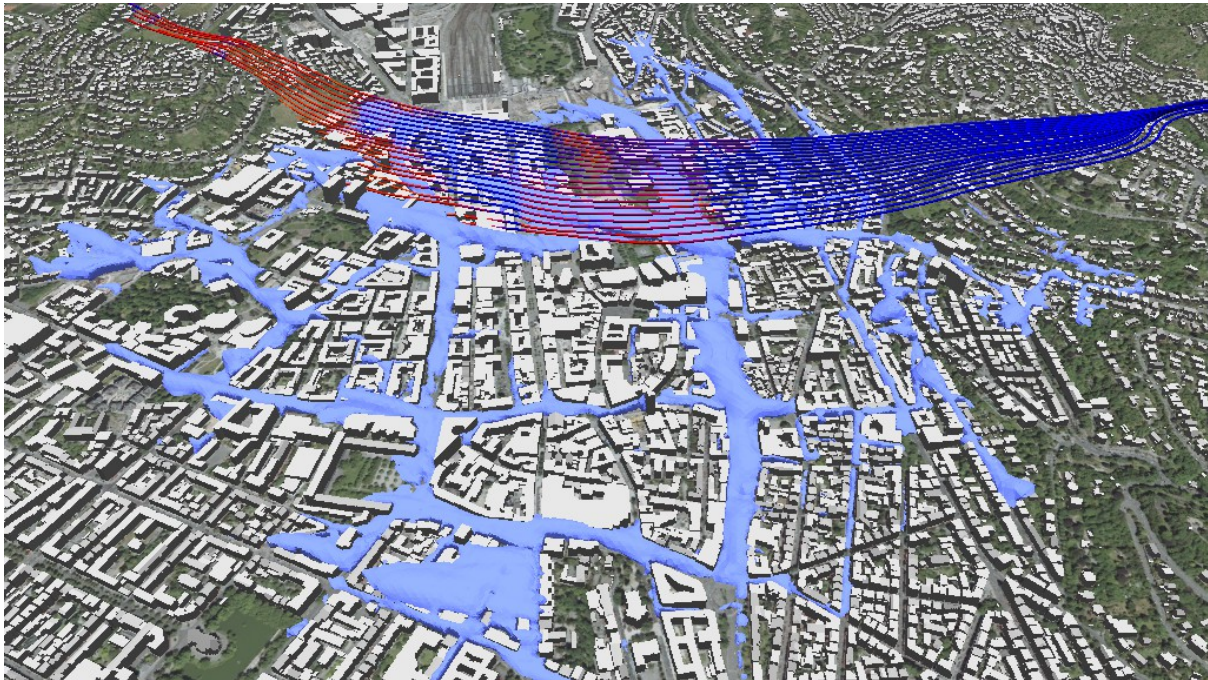


Figure 87. The same model from a different perspective: LoD 2 building models and streamlines are added (Stuttgart).

Extractions from the Stuttgart model can be seen in Figure 86 and Figure 87. A coarser building model is used for Figure 87 to show to location and approximate extends of the city’s buildings. For Figure 86, more detailed building scans are used, presenting textured geometries. Both figures include the visualization of NOx iso surfaces to present the simulation predictions for NOx distribution. Moreover, Figure 87 also shows streamlines to illustrate the wind speed and direction.

11.1.3 Benchmarking

Benchmarking of COVISE was conducted. Therefore, sample data sets of different size were generated. The data sets were created using OpenFOAM to enable benchmarking on a sample case comparable to the actual data as obtained from the UAP use case. The benchmarking was performed on one node of the HLRS visualization cluster, a 64-bit system running Centos 8 equipped with Intel Xeon Gold 6134 CPU at 3.2GH and K6000 GPU. A COVISE workflow as described above in Figure 88 was used containing processing modules to read the OpenFOAM data set and compute iso surfaces, cutting surfaces and domain surfaces.

Execution time with different grid resolutions

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies			Page:	156 of 174	
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

The execution time of the COVISE workflow was compared for different mesh resolutions. The data set sizes were between 4 thousand and 2.6 million grid vertices. COVISE was executed on one rank of the HLRs visualization cluster. The results for the cold start task, i.e. including the start-up of the application itself, are shown in Figure 88. It can be observed that for smaller data sizes, the time is almost the same while it is increasing for more than ~150k points. This is due to the overhead where most time is spent for smaller resolutions while significantly more time is needed for the actual computations when the number of vertices is increased.

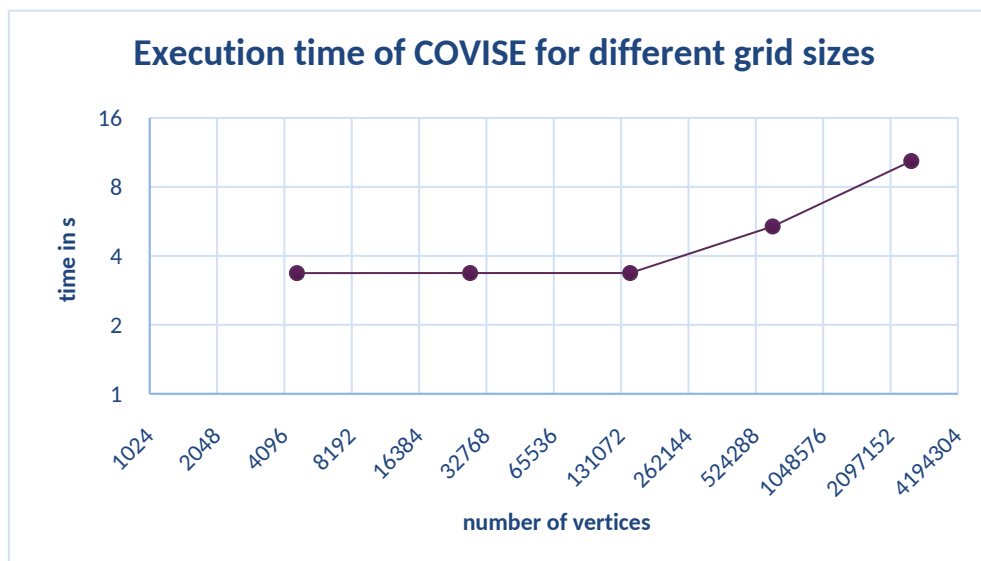


Figure 88. Execution time of cold-start task for different mesh resolutions.

Execution time per module

Besides the overall execution time, the time spent in each module was measured. Two different mesh resolutions were compared. As shown in Figure 89, in both cases, most time is spent in the *DomainSurface* and *IsoSurface* module, although the time is significantly longer for the larger data set. Further, the figure shows that for the smaller data set, the execution time of the different modules is almost the same. In contrast, the deviations in time for the larger data set are much stronger, confirming the impressions of the previous figure.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	157 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

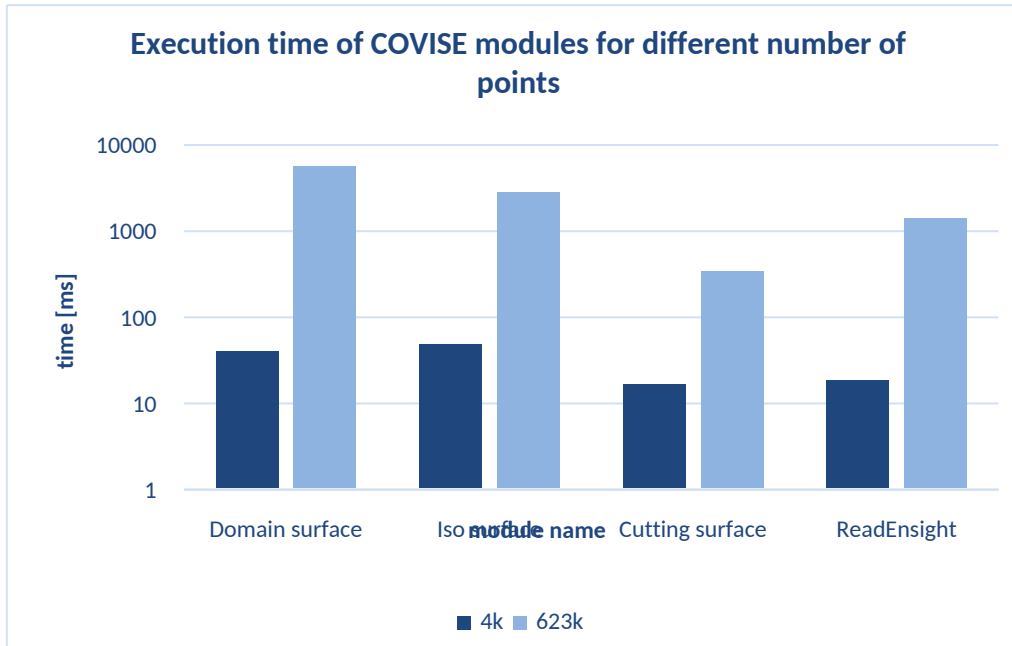


Figure 89. Processing time per module for different mesh resolutions.

11.1.4 Integration with the Portal

The COVISE tool has been integrated in the Portal in order to facilitate the access to visualization (Figure 90). First of all, a new option was included under the ‘Visualization’ category for accessing information about COVISE. Since COVISE is a desktop application, the new page includes information about how to install and use the COVISE tool.

Additionally, the backend of the Portal has included a new filter for instances, so it can identify those instances with file formats which are compatible with COVISE (like .vtk, .openfoam, .gmv and more). Thanks to this filter, the Portal can show a list of those instances that could use COVISE, so the user can see the list and select one of those instances.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	158 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

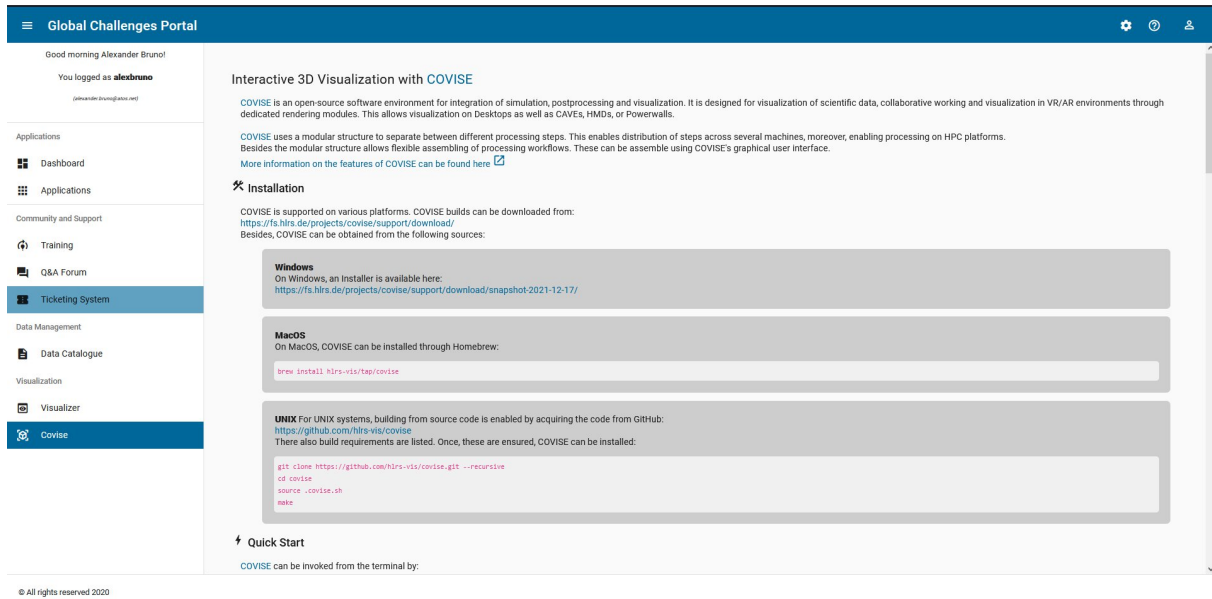


Figure 90. Access to COVISE from the HiDALGO Portal.

If the output file is accessible (i.e. openly published in the CKAN), the user has the option to download it, together with an example configuration file configuration for COVISE, so it will be easier for the user to generate the needed visualization.

11.2 Visualizer

Visualizer is a web-based visualization tool for CSV data using coordinated multiple views [51]. It has been introduced and initially benchmarked in Deliverable 3.2, additional information has been provided in Deliverable 3.3, and first dashboards for the use cases have been described in Deliverable 3.4. Additionally, details on integrating Visualizer to other web-based applications are provided in Deliverable 5.6 and 5.7.

11.2.1 AI method

Besides creating visualizations, Visualizer enables users to apply analytical workflows depending on their analytical goals. So far, two analytical goals have been integrated in Visualizer. These are "Gain Overview" and "Analyse Outliers".

For gaining an overview of the data, users need to specify the feature selection method, the dimension reduction method, the pattern recognition method (so far, only clustering is

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	159 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

available), the machine learning algorithm for clustering and finally, the visualization for displaying the results.

For outlier detection, users need to specify the outlier detection method, the dimension reduction method and again, the visualization for displaying the results, see Figure 91.

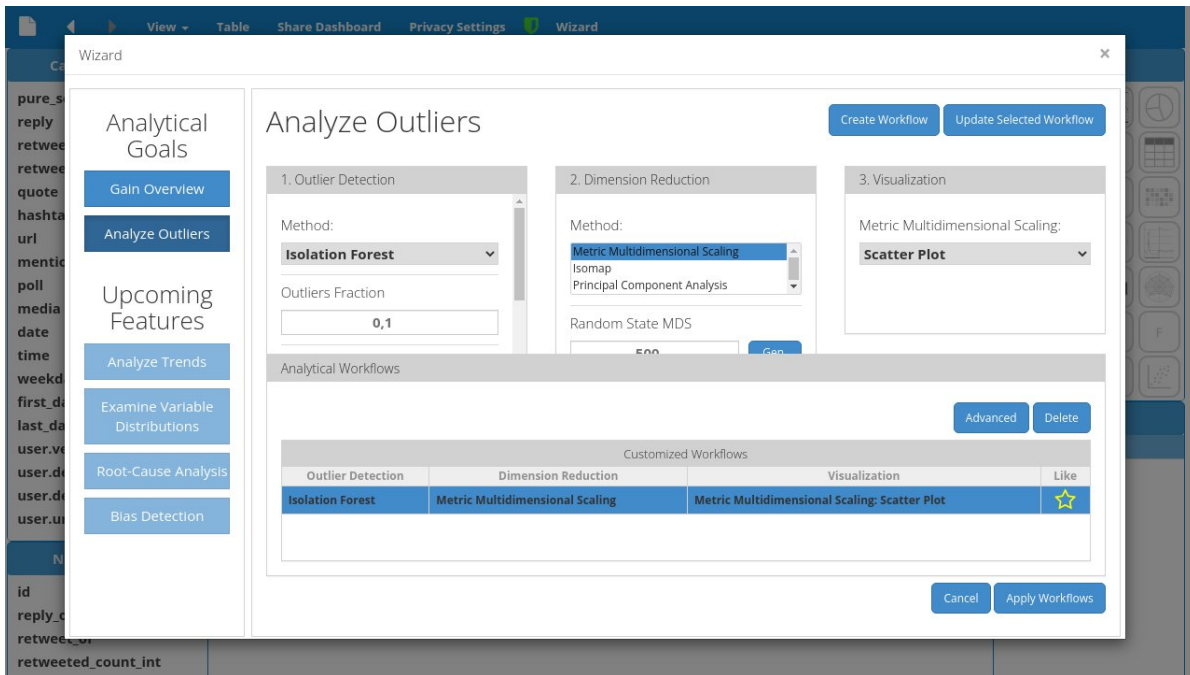


Figure 91. AI-Wizard enabling users to select analytical goals and configure workflows

After configuration, the workflows are applied. Figure 92 shows the results for the clustering and outlier detection method using a subset of the NEOS Twitter data set from the social network use case. By selecting outliers in the right chart, corresponding elements are highlighted in the left chart. The distance between data points in both charts show their similarity, close data points indicate a higher similarity, whereas data points which are far away from each other have a lower similarity.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	160 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final



Figure 92. Results from a clustering (left chart) and outlier detection (right chart) method using the AI-Wizard in Visualizer for NEOS twitter data.

11.2.2 Optimization

The following features have been implemented in Visualizer to improve it in terms of supported data sizes, code quality and UI features:

- Loading large data files
 - Local files: The initial benchmarking revealed that loading large data files (CSV file with more than 4.5 Mio rows and 12 columns, 516.7MB) was not possible due to limitations of string sizes in JavaScript. In order to support reading larger files, they are split up into chunks and merged later. Benchmarking results are provided in the next section.
 - Remote files: The same limitations concerning string sizes have been identified and fixed for loading remote files in Visualizer. Again, detailed results for benchmarking are provided in the benchmarking section below.
- Outsourcing data aggregation methods to web workers: To improve the code quality and optimize data processing, an API has been defined enabling us to outsource data aggregation methods to web workers. This ensures that calculations are executed in a background thread not blocking the main thread.
- Rule-based visualization recommender: The rule-based visualization recommender implemented in Visualizer (enable/disable visualization depending on the selected

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies			Page:	161 of 174	
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

data fields) has been extended to support multiple occurrences and optional visual channels at the same time.

- The coordination framework in Visualizer has been extended and improved by:
 - supporting multiple brushes within different visualizations, where brushes can be AND or OR connected,
 - implementing an id-based coordination of visualizations for those visualizations using the same data sets and keeping the value-based coordination of visualizations for those with different data sets,
 - allowing users to specify for each visualization how incoming events from other visualizations are handled (highlight data, filter data or ignore events from other visualizations).

11.2.3 Benchmarking

For evaluating Visualizer, different methods are used. These include measuring the time for loading data sets and dashboard on the one side, and the usability of Visualizer on the other side.

First, the time required for loading local and remote data sets and dashboards has been benchmarked for different data set sizes. Results are compared with initial benchmarking performed in Deliverable 3.2.

For benchmarking, the following notebook configuration has been used:

Notebook: Lenovo ThinkPad T490S

CPU: 8 Intel ® Core™ i7-8565U CPU @ 1,80 GHz 1.992 GHz

RAM: 16GB

System: 64 Bit operating system, x64-based processor

Operating System: Windows 10

Version: Windows 10 Pro

Browser: Chrome, version 96.0.4664.45

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	162 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

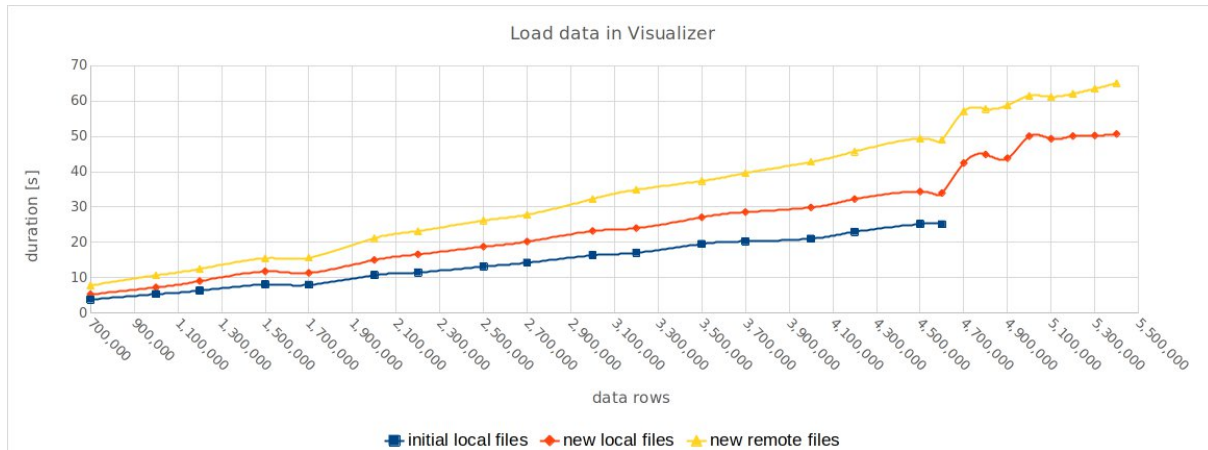


Figure 93. Average time in seconds for loading a CSV file in Visualizer depending on the number of data rows - blue: initial benchmarking for local files, red: new benchmarking for local files, yellow: new benchmarking for remote files.

Figure 93 shows the average time consumption in seconds for loading a data set in Visualizer for different file sizes. The blue line shows the original benchmarks for local files, which was limited to about 500MB file size. To support files larger than 500MB, files are split up into chunks for reading, files smaller than 500MB are not split up to avoid a performance loss during file reading. Therefore, it is called hybrid approach. The new, hybrid approach (red) has slightly increased values for local datasets until 500MB and significantly higher values for larger files due to splitting up the file content and reading in chunks. The reasons for higher values for files less than 500MB size needs further investigation. The yellow line shows the benchmarks for the new hybrid approach for remote file loading. It shows a similar pattern as the benchmarks for local files, but higher values due to loading remote files.

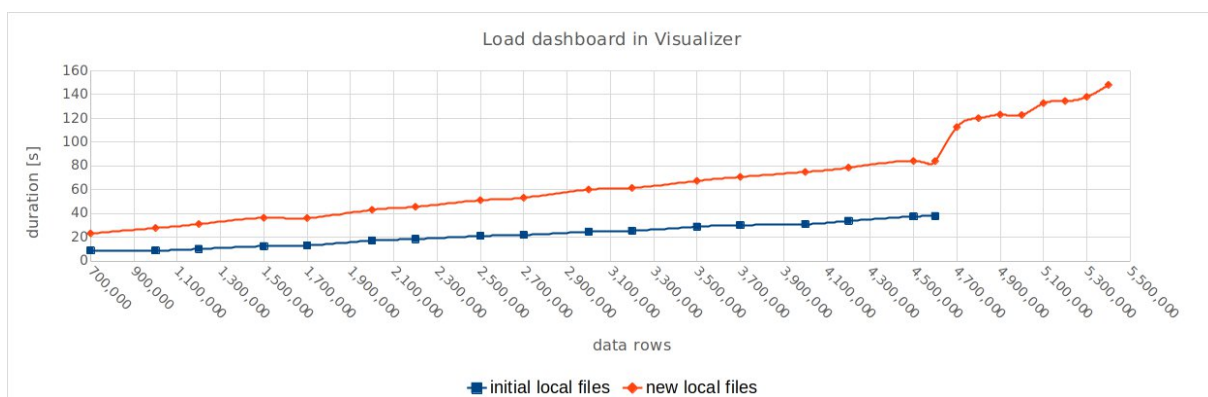


Figure 94. Average time in seconds for loading a pre-configured dashboard in Visualizer depending on the number of data rows - blue: initial benchmarking for local files, red: new benchmarking for local files.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies	Page:	163 of 174
Reference:	D3.5	Dissemination:	PU
	Version:	1.0	Status: Final

Figure 94 shows the benchmarks for loading a dashboard in Visualizer. The blue line, again, shows the results for the initial benchmarking while the red line shows the results for the hybrid approach. Again, a similar pattern can be observed with increasing average durations. Furthermore, memory consumption for loading local data sets has been investigated. Benchmarking revealed varying results for memory consumptions. First investigations allow assumptions that this happens due to Chrome’s garbage collector which cannot be influenced from outside and cannot be disabled for benchmarking.

Furthermore, the usability of Visualizer has been evaluated in a user study. Evaluation procedure:

- Participant information and consent form,
- Demographic and experience questionnaire,
- Visualizer introduction video and try out Visualizer,
- Tasks execution and feedback on task load,
- Post evaluation questionnaire.

The results allow to evaluate:

- whether Visualizer enables users to analyse their data,
- whether available interaction possibilities in Visualizer are intuitive and easy to use,
- which further visualizations and interaction possibilities are required to analyse data.

The user study was executed with eight participants, four of them female and four male. Most of them are associated to the HiDALGO project. Most of them are researchers (5), one stated to be a data scientist, one student and one support staff.

One participant uses data analysis tools at least once a month, others use them several times a month or week (each three) and one participant every workday. For data analysis, seven participants stated to use Python, five Excel, four MATLAB, three R and finally, KNIME, RapidMiner, SAS and MAX QDA social science software are used by one. They mainly use these tools for getting an overview of the data (seven), identification of correlations (four), anomaly detection and cause-effect analysis (both two) and trend analysis (one). Other than that, data analysis tools are also used for simulations by one participant.

Considering the usage of visualization tools, one participant stated to never use visualization tools, two participants use them once a month or less, one participant several times a month, three participants several times a week and one participant uses visualization tools every

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	164 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

workday. For this, users mainly use Excel (four), Power BI, Shiny R, Matplotlib (each one) or their own visualization tools including COVISE and Vistle. Visualization tools are also mainly used for getting an overview of the data (six), trend analysis (two), identification of correlations (two), cause-effect analysis (two), anomaly detection (one), visualization of simulation results (one), comparison (one), visualization of geo-spatial information (one) and holistic perspectives (one).

In the first tasks, users were requested to investigate data from one twitter conversation related to the social network use case and answer questions by selecting valid data fields and creating visualizations. Additional hints were provided to support users in solving the tasks.

Task 1:

- T1.1: Which type of tweets are available?
- T1.2: Which tweet type is the most frequent one?
- T1.3: How often does it occur?
- T1.4: What is the number of all neutral sentiments for NEOS (sentiment_neos = 0) for type "quotes" only?
- T1.5: What is the number of quote-retweets for NEOS if you brush the data for sentiment_fpo = positive (1) in a parallel coordinates visualization?

In the second task, a dashboard containing multiple visualizations for seven twitter conversations, again from the social network use case, has been shared with the participants to respond to a set of questions.

Task 2:

- T2.1: There is at least one conversation with overall (> 20) positive sentiments for both political parties, NEOS and FPÖ (yes/no).
- T2.2: There is at least one conversation with overall (> 20) positive sentiments for FPÖ only (yes/no).
- T2.3: There is at least one conversation with overall (> 20) positive sentiments for NEOS only (yes/no).
- T2.4: There is at least one conversation with overall (< -20) negative sentiments for FPÖ only (yes/no)
- T2.5: There is at least one conversation with overall (< -20) negative sentiments for NEOS only (yes/no)

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	165 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

- T2.6: There is at least one conversation with a similar (difference < 10) sum of sentiments for NEOS and FPÖ (yes/no).
- T2.7: The conversation with most positive sentiments for NEOS is?
- T2.8: The conversation with most positive sentiments for FPÖ is?
- T2.9: Investigate whether, and if so, which conversations have either a positive (1) or a negative (-1) initial tweet (type=tweet) for NEOS which turns into the opposite for any of its responses (retweet, quote, quote-retweet).

The visualizations provided in the dashboard for task 2 are shown in Figure 95, Figure 96 and Figure 97. Figure 95 shows the sum of all sentiments for each conversation (colour) for the two political parties on the x- (NEOS) and y-axis (FPÖ). High values indicate a high number of positive sentiments for the corresponding party, negative sentiments decrease these values. Figure 96 shows a parallel coordinates visualization with four axes. The first one, which always specifies the colour, shows the different tweet types, which are in this case tweet, retweet, quote and quote-retweet. The second axis shows the corresponding conversations, and the last two axes show the sentiments for the two political parties. In contrast to Figure 95, the data in the parallel coordinates visualization is not aggregated. Figure 97 shows the sum of all sentiments (colour) for each conversation and tweet type on the left side and the number (colour) of tweets for each tweet type and sentiment on the right side. In the heat map, low values are encoded yellow, while high values are encoded red. The upper heat maps belong to NEOS, the lower ones to FPÖ.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	166 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

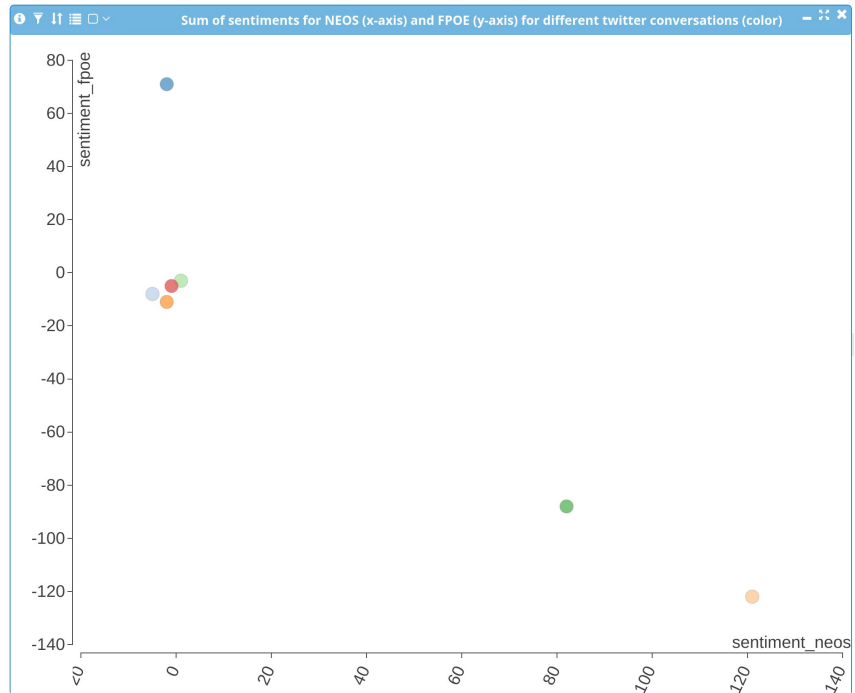


Figure 95. Scatter plot showing sum of sentiments for NEOS (x-axis) and FPÖ (y-axis) for different twitter conversations (colour).

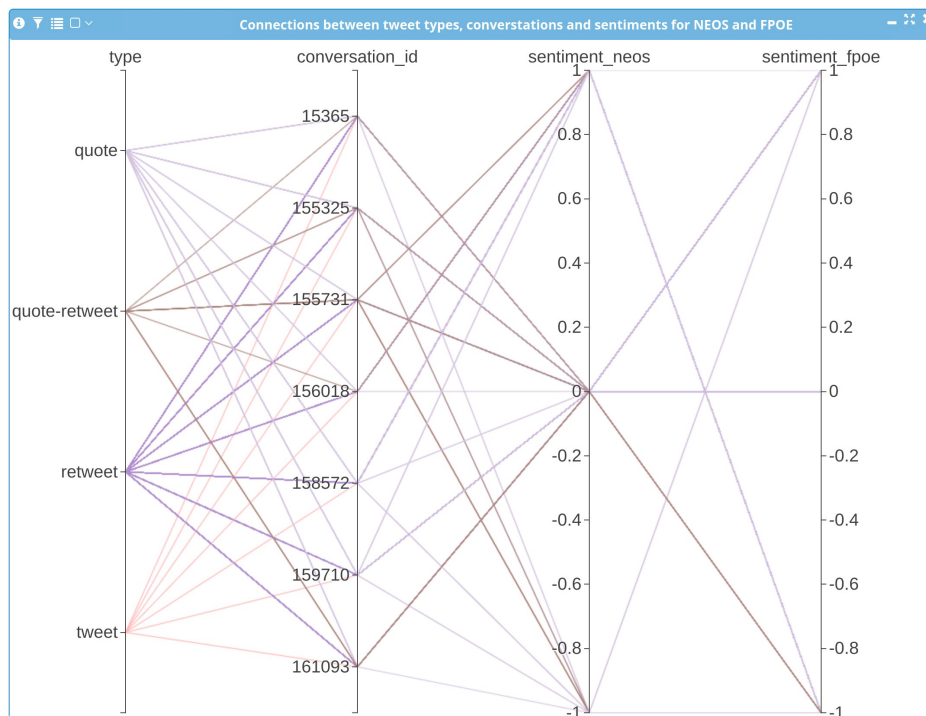


Figure 96. Parallel coordinates showing connections between tweet types, conversations and sentiments for NEOS and FPÖ

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	167 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

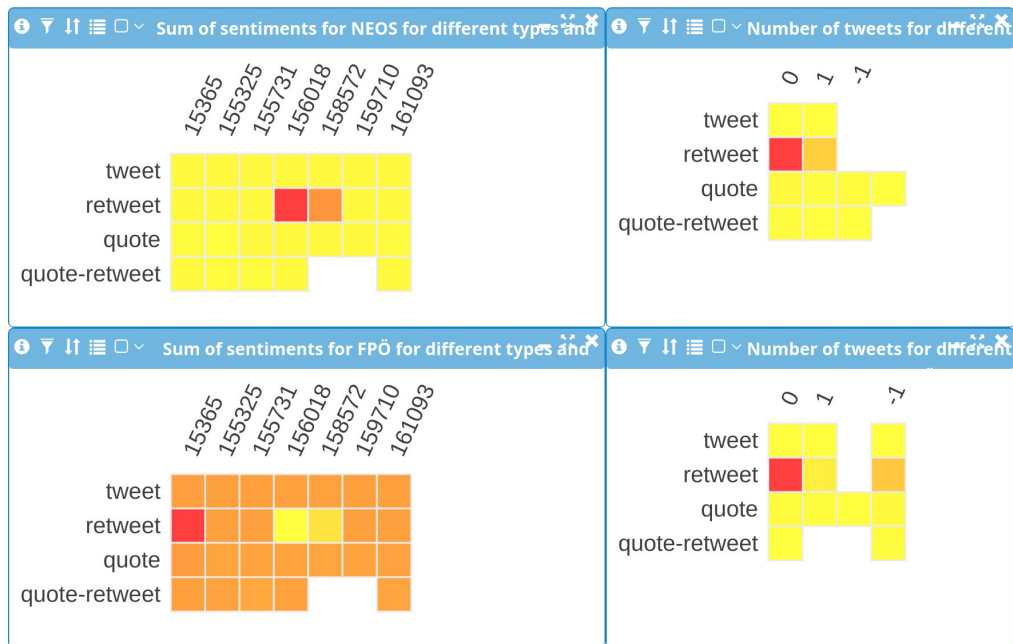


Figure 97. Heat maps showing sum of all sentiments (colour) for different tweet types for each twitter conversation (left charts) and number (colour) of sentiments for each tweet type (right charts) for NEOS (upper charts) and FPÖ (lower charts).

Figure 98 shows the task completion for each sub-task. Task 2.7 has been successfully completed by all participants, whereas two tasks, T1.4 and T2.9 have been completed successfully by only three participants each. Besides creating and/or investigating visualizations, users were requested to perform brushes within the visualizations and interpret the results in the same or other visualizations. The partially lower task completion rate for T2.1 to T2.6 was due to investigating less suitable visualizations which did not reveal the requested information easily. Furthermore, anonymous user interactions have been recorded during the task execution to gain additional insights.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	168 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

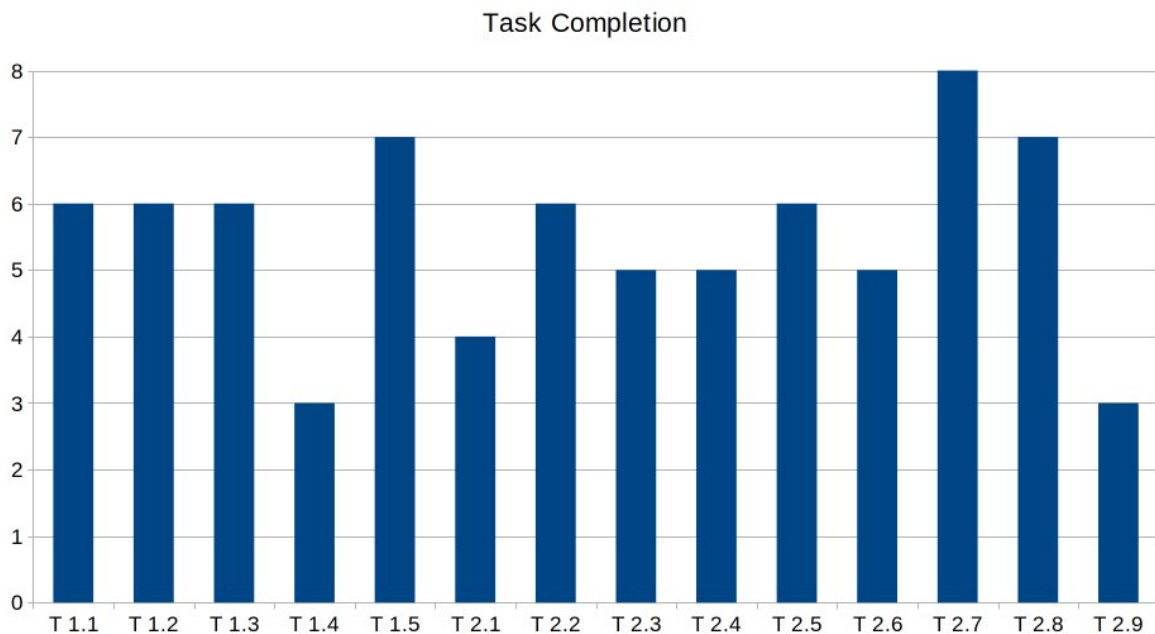


Figure 98. Task completion in the user study of Visualizer for eight participants.

After both main tasks, users were requested to provide feedback about their task load using the NASA Task Load Index [52].

Results for both tasks are shown in Figure 99 and Figure 100. This corresponds also with the feedback provided by the participants, stating that it was easier for them to investigate the provided visualizations in task two compared to selecting required data fields and creating the visualizations by themselves in the first task. Other than that, users mainly had difficulties in investigating the different brushing methods in the visualizations and identify whether there are active brushes. However, the provided functionalities in Visualizer have been mainly perceived as useful (rating between 4-7 on a 7-point Likert scale) and users did not have difficulties in using them. These are creating visualizations, re-positioning and resizing visualizations, data aggregation for visualizations, filtering and specifying incoming event handling as well as coordinated brushing over multiple visualizations. All participants stated that Visualizer helped them in gaining insights in the data (average rating of 5.875 on a seven-point Likert scale). However, most of them did not feel confident in using Visualizer and stated to need assistance. The overall feedback provided by the users was very positive. They highlighted that Visualizer was easy to use, they appreciated the UI design, the coordination of visualizations using multiple brushes and the number of features available in Visualizer.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	169 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

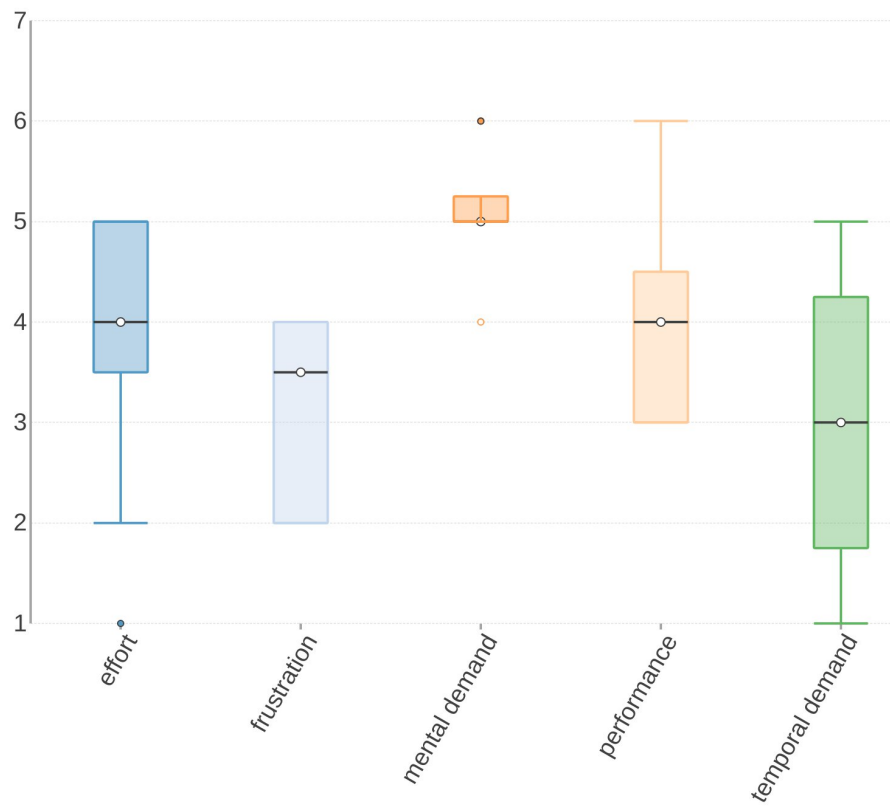


Figure 99. Task 1 user feedback for task load on a seven-point scale.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies			Page:	170 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0
				Status:	Final

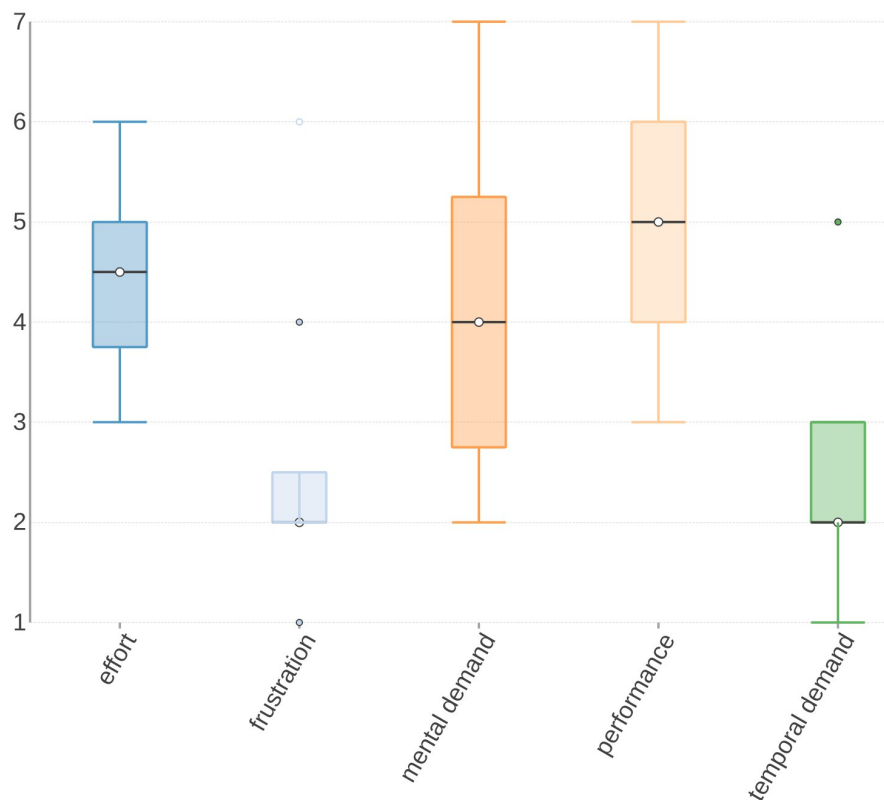


Figure 100. Task 2 user feedback for task load on a seven-point scale.

11.2.4 Dashboard results

For the social network use case, different dashboards have been created and described in previous sections (AI Methods and Benchmarking) and will not be described here in detail. Until the end of the project, new visualizations, e.g., stacked bar chart, will be included and existing visualizations and interaction possibilities will be extended.

For the migration use case, data has been transformed, visualizations have been extended and improved, e.g. geo-visualization or line chart, to support the requirements of the use case.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	171 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

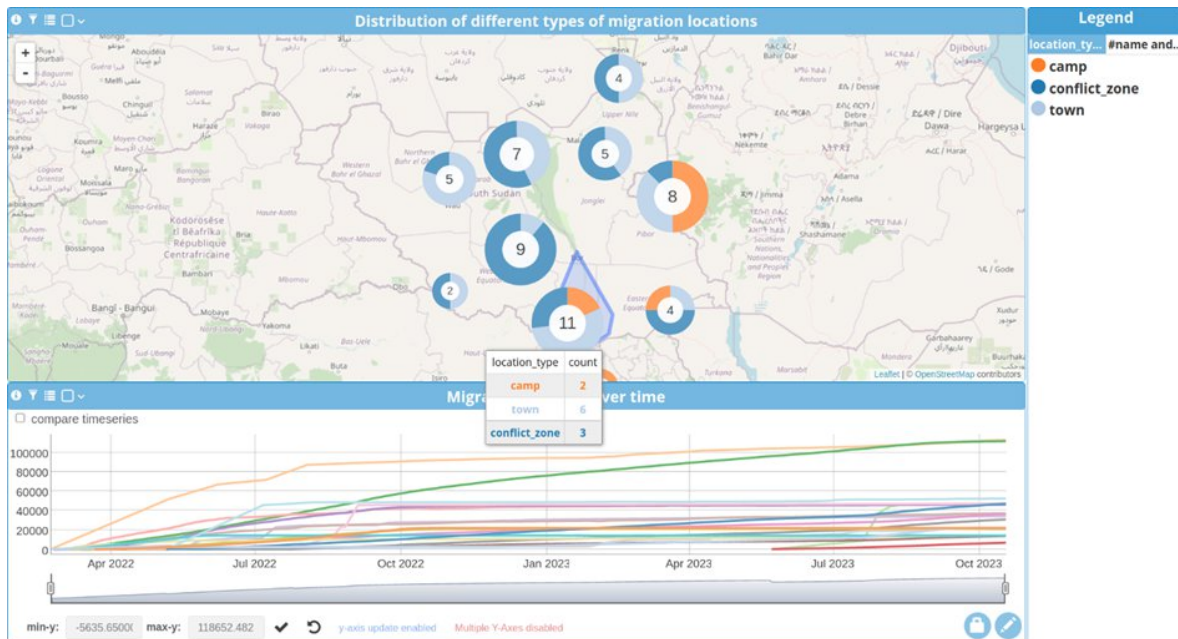


Figure 101. Dashboard example for the migration use case.

Figure 101 shows a dashboard example for the migration use case. Depending on the selection in the geo-visualization, data is shown in the line chart or can be highlighted on demand.



Figure 102. Dashboard example for COVID-19 simulation data.

Figure 102 shows a dashboard example for the COVID-19 simulation data.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies	Page:	172 of 174
Reference:	D3.5	Dissemination:	PU
Version:	1.0	Status:	Final

12 Coupling technologies

Coupling technologies are used to combine different (existing) applications to make them work together for an overarching purpose. Coupling technologies are a staple in the multiscale and hybrid simulation approaches, and a range of generic technologies have emerged in recent years, each with their unique added values.

12.1 Migration Pilot

Within the Migration pilot we have established two cyclic (two-way) coupling approaches for multiscale simulations and couple the macro and micro scale models. The first approach is file I/O coupling and the second approach is MUSCLE3 [30]. Besides, by using an acyclic (one-way) coupling, we have incorporated weather data provided by ECMWF including river discharge level and precipitation level datasets, to examine the influence of different levels of the mentioned climate factors on the movement of forced displaced people. With this, we aimed to answer the following questions: How weather conditions like precipitation level affect refugees' decision to move from a location? How do they affect refugees' speed? How river discharge levels influence refugees' movement? And more importantly, how to reflect all these assumptions in a rule set for future models? Therefore, we examined our proposed approach for the South Sudan conflict with real data. We described the application of these approaches in some detail in D4.3 and D3.4. Figure 103 illustrates the schematic scale Separation Map for data coupling between macroscale and microscale models and with weather forecast data. The macro- and micro-scale model have identical time scales and overlapping spatial scales, and are coupled cyclically. In addition, the micro-scale model receives data from the weather forecast data source (or from ECMWF Climate Data Store in the case of historical data).

We planned to perform a performance test of the different coupling approaches and aimed to provide first coupling performance results in this report. However, due to the MUSCLE3 difficulties in the execution of high number of multiscale simulation ensembles on the Eagle supercomputer, we could only perform these tests based on file I/O coupling on the Eagle. Hopefully, the results of file I/O coupling show the coupling rules are indeed scientifically robust. The results are presented in the D4.4.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	173 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

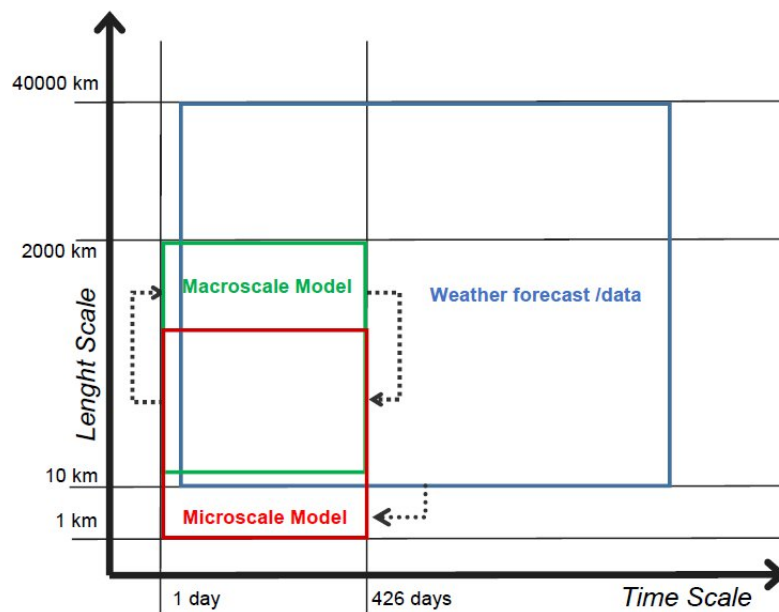


Figure 103. Scale Separation Map of Weather Data Coupled Multiscale model.

12.2 Urban Air Pollution Pilot

The pre-processing workflow of the Urban Air Pollution Pilot has integrated many different tools and applications (Figure 104). Firstly, the predefined input files like meshes and geo files. The second step is to load all necessary modules and tools to run the workflow, it checks the user parameters, the missing input files and the simulation settings.

The missing files will be generated according to user parameters. If random traffic is chosen it will generate traffic to the geometry and run it, if not then precompiled traffic output or predefined simulation files are loaded and generating the traffic output EMI file.

After the traffic we have to provide weather data to the pilot application. This can be done in several ways. The most common option is that the dataset already has weather data, if not then it can be directly download from ECMWF using ECMWF provided EWcloud workflow or custom made downloader developed by us based on ECMWF polytope python library. Last but not least, synthetic wind data can be generated based on user inputs to test special scenarios. If all the necessary inputs are present, then the Computational Fluid Dynamics (CFD) will be prepared based on user parameters, and the simulation starts running.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies			Page:	174 of 174	
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

After the CFD simulation is done our postprocessing tools start if user checked this option, and final results are generated.

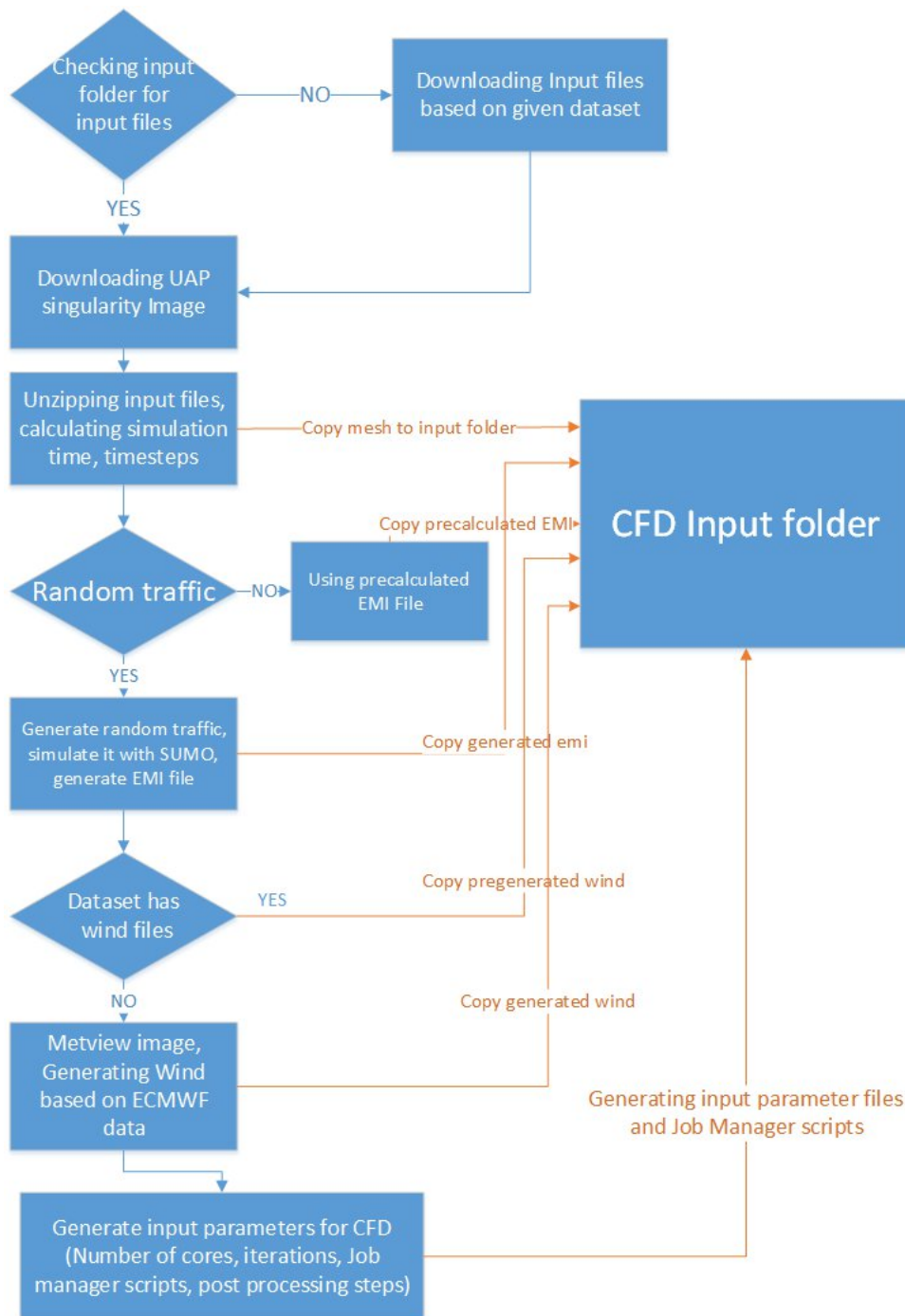


Figure 104. Pre-processing workflow of Urban Air Pollution pilot.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies			Page:	175 of 174	
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

12.3 Social Networks Pilot

The social networks pilot focusses on models and simulation of the spread of messages in social media. Apart from deriving proper random graph models for real-world social networks, we are also interested in understanding the dynamic processes within a social network. Our simulator uses several characteristics of the author of a message and the message itself, and simulates the spread of corresponding messages through the network. However, depending on the underlying events, these characteristics may change or they may change the spreading behaviour of a message. To account for these changes, a close coupling with live Twitter data is indispensable.

The coupling task builds on a Twitter monitoring system, which focusses on certain topics (defined by a user w.r.t. keywords or hashtags), and acquires data about the parameters of the messages regarding these topics. These parameters are compared to the parameters used by the simulator, and if a predefined difference between the two is observed, then the simulator is notified and its parameters are updated. Subsequently, the simulator can be used for forecasts about the message flow with the updated parameters. Here, the coupling is crucial in order to run the simulator continuously with realistic parameters, and to provide accurate forecasts about the dynamic processes behind the spread of messages.

The coupling has been realized through two Cloudify blueprints. The first blueprint invokes the Twitter monitor. From the monitor, the simulator can be launched through the second blueprint. The social networks simulator could be launched asynchronously from the Twitter monitor (Figure 105). In this case, we are dealing with a two-step processing.

The Twitter Monitor consists of the Twitter listener and the Twitter analyser. The listener is connected to the Twitter Streaming API and it collects Tweets of a topic that is given as a user input. The Twitter analyser summarizes the collected Tweets and generates statistics w.r.t. certain features of the Tweets and authors of the Tweets. The same features are also used by the simulator. The Twitter analyser will upload the statistics CSV file to the project CKAN platform, if any of the pre-defined thresholds is exceeded. After uploading the statistics, the social networks simulator is deployed through its blueprint. It means, a simulation run will be created, installed and executed on one of the available HPC clusters. The results are uploaded again to the CKAN.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	176 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

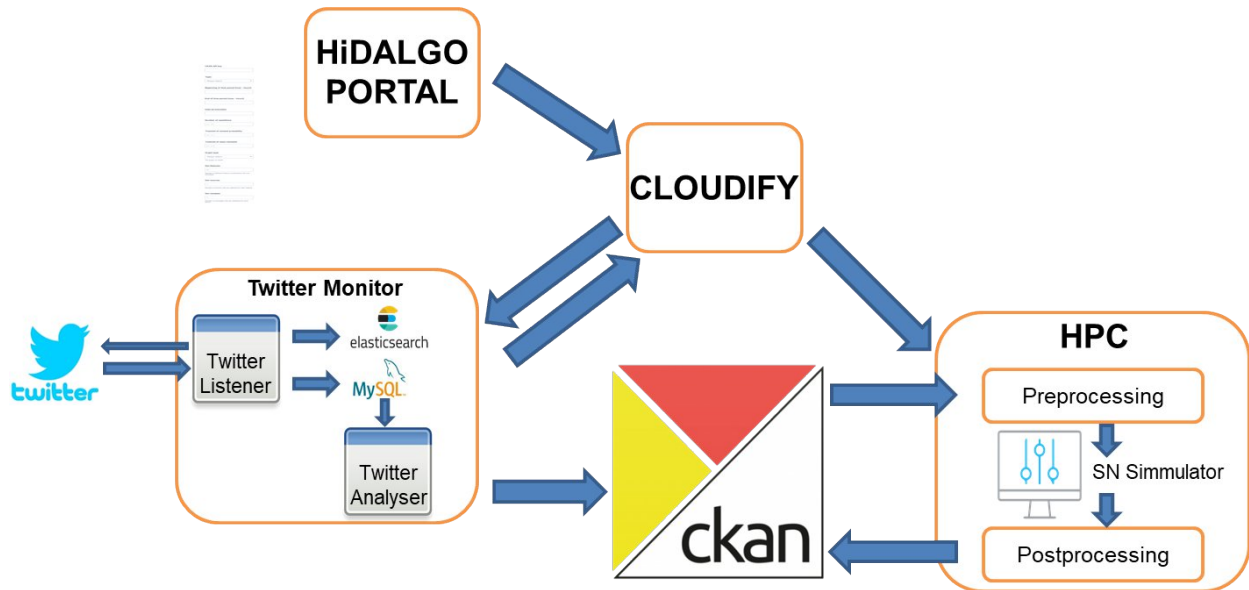


Figure 105. The workflow of Twitter Monitor, Cloudify blueprints and Social Network Simulator.

12.4 Weather data

ECMWF's weather forecasts, climate reanalysis, global hydrological, and Copernicus Atmosphere Monitoring Service (CAMS) data play a key role in coupling with the simulated models of migration and UAP pilots. The overall strategy for the coupling of weather and climate data consists of two stages: (a) static coupling of climate reanalysis data, (b) dynamic coupling via a RESTful API. This section provides an update of its progress.

The development of the last year has aimed at increasing the security and the performances of the system. Here is a detailed list of the changes to the WCDA:

- WCDA is now able to authenticate the requests against the HiDALGO Keycloak system. Users can request data directly with the HiDALGO accounts. This integration allows requests to be submitted from HiDALGO orchestrator using the users' credentials common to HiDALGO portal.
- Users can request and retrieve real-time forecast data. Previously, a limit was in place to allow only data older than 30 days. This was due to licences issues that have now been resolved in the consortium.
- Secure server connection through HTTPS has been enabled.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies	Page:	177 of 174
Reference:	D3.5	Dissemination:	PU
	Version:	1.0	Status: Final

- A quality of service (QoS) mechanism has been implemented to guarantee a certain level of performance to all users. In the case of HiDALGO the following rules have been set:
 - Max of 6 simultaneous requests per user
 - Max of 15 simultaneous requests among all HiDALGO users
- The following performance improvements have been achieved:
 - Requests larger than 2GB are now supported
 - Improvement in processing and interpolation speed
 - Fixed failing of requests involving interpolations of large fields

As part of the effort to increase the security of the system, a penetration testing for WCDA was procured. This to ensure that the service provided over openly-accessible web ports is safe and secure, both from the perspective of service resilience and internal data security, but also to ensure that communication between WCDA servers and clients is secure. Penetration testing was performed by Bulletproof [53]. The first test ran in March 2021 and raised a number of issues:

1. Incorrect cache control headers,
2. Ability to see the names of collections which you could not access,
3. Ability to downgrade from HTTPS to HTTP,
4. Limited input validation on requests,
5. Access keys had no expiration date,
6. Our service was using outdated cryptographic ciphers (TLSv1.0 and TLSv1.1),
7. Ability to access the download endpoint (when a data access request is complete) without authentication, the UUID of the request is known.

1, 2, 3, and 5 have been fixed: HTTP access has been disabled, cache control headers have been implemented, and authentication/authorization has been improved. 6 has been mostly resolved by disabling TLSv1.0 and TLSv1.1. [4] is by design, as WCDA is not intending to inspect request payloads but merely pass them on to the back-end data stores (for example, ECMWF's MARS archive) – the backend must do input validation. WCDA is secure against injection attacks in the request payload and malformed requests as they are simply rejected by the backend.

7 is also by design, as it is intended that a request can be made by a user/orchestrator for later download on compute-oriented machines (such as HPC, HPDA or elastic cloud resources)

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	178 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

where it can be complicated to store user credentials safely. It is not possible to access other user's downloads without knowing the version 4 UUID of the request, which it is deemed safe.

A second penetration testing was performed in September 2021 and confirmed that only issues 4 and 7 remain (as intended), although there is still a minor point related to [6] which should be addressed, concerning disabling of CBC ciphers.

This penetration testing gives confidence that access to WCDA is secure; and that it is reasonably safe against malicious intrusion.

12.4.1 Weather data notification system

The development of the last year has also aimed at designing and integrating a weather data notification system into HiDALGO infrastructure.

Recent adoption of Open Data policies and investments towards Cloud-based platforms have attracted a growing number of consumers of ECMWF data. An example of these initiatives is the European Weather Cloud (EWCloud), where users wish to run automated, real-time tasks or workflows closer to the latest data produced by the model in the ECMWF HPC facility, thus avoiding costly data transfers out of the data centre. This trend is likely to increase together with the exponential growth of weather forecast data. From an operational perspective, this convergence of HPC and cloud is dependent on timely synchronisation with the forecast schedule. A mechanism is needed to notify the consumers of specific data availability in a scalable manner and provide the capability to automatically trigger their workflows based on this data.

To accomplish this, ECMWF is developing a system, named Aviso, designed to notify of availability of real-time forecast data or derived products, and to trigger user-defined workflows in an automated fashion. End-users can build their workflow based on events, using a When <this>... Do <that> logic directly linked to ECMWF metadata semantics. The system is composed of a server application based on a persistent key-value store, leveraging modern technologies such as `etcd` [54], to provide consistency, transactionality, reliability and scalability to the end-users. The client side is a lightweight Python application providing a CLI interface as well as a Python API for easy integration in the users' workflows. Finally, the notifications can be exchanged using CloudEvents messages; this allows workflows that span across multiple data centres and cloud-based infrastructures.

ECMWF has deployed Aviso as a notification service for the availability of the data produced by the Centre. Figure 106 shows ECMWF dataflow; it starts from the data assimilation of

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	179 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

observations, it then follows to the generation of the model output, the real-time global forecast. This is a time-critical step relevant for users' workflows and therefore Aviso is notified when this data is stored in the archive. The dataflow continues with the generation of derived products that are then disseminated via ECMWF dissemination system. The delivery of these products is also notified to Aviso as users depend on custom products for their downstream applications. Once notified, Aviso will forward the relevant notifications to all the clients registered. These are processes running in the EWCloud infrastructure, set by the end users to triggering further data processing.

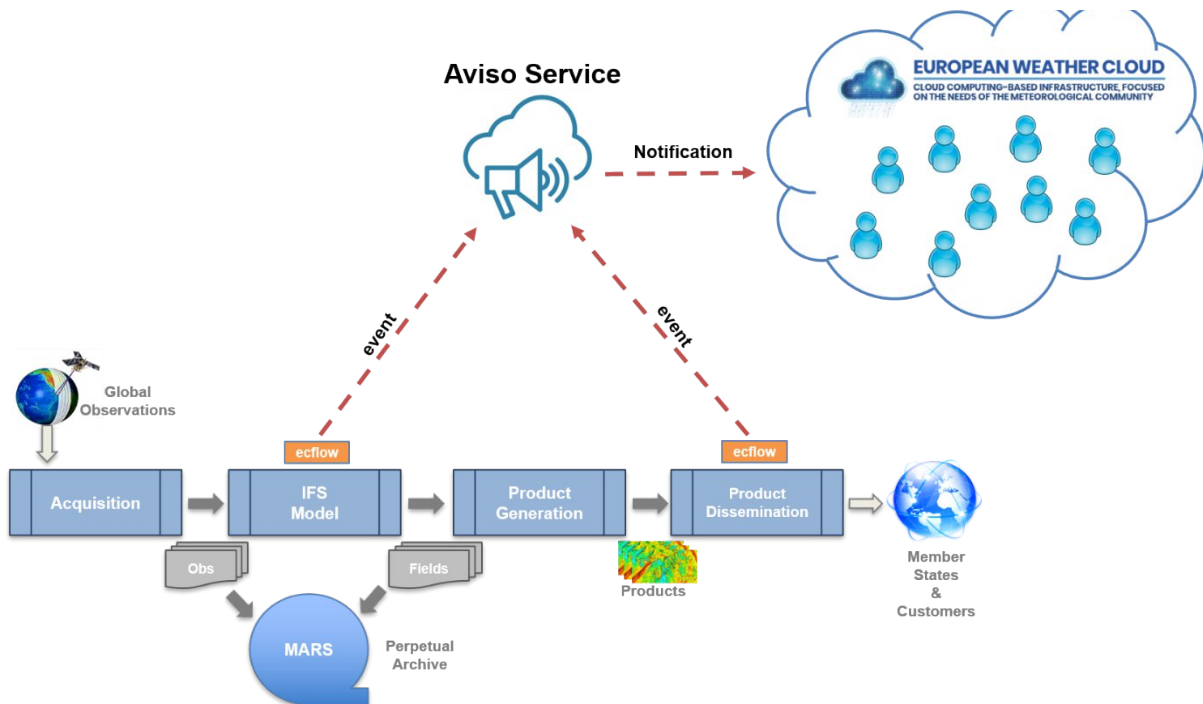


Figure 106. Events submitted to Aviso from the ECMWF Dataflow.

This system is opensource and available on GitHub [55]. The project is funded also by other European projects. In HiDALGO the aim is to adapt it to the Urban Air Pollution pilot workflow. As described in D5.7 and D4.4, ECMWF has provided an EWCloud VM for running custom postprocessing for the Urban Air Pollution workflow. This workflow is triggered by the HiDALGO portal via the Cloudify orchestrator. Cloudify interfaces with EWCloud VM that requests forecast data, processes it and then copies the output to an external CKAN store. Once in CKAN, the data are used by the UAP simulation to compute high resolution urban air maps. The workflow here described runs on request by the end users interacting with the

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies	Page:	180 of 174
Reference:	D3.5	Dissemination:	PU
	Version:	1.0	Status: Final

HiDALGO portal where they can select for which day and location they wish to run the simulation. By doing this, end users can create maps related to days in the past.

If this workflow was to become an operational service, time would be critical and running this simulation with the latest forecast data would be essential. Aviso allows to run this workflow only when the latest forecast data is available. This enables end users to define a data responsive processing pipeline. In terms of usability, Aviso is completely transparent to end users, which, in the definition of their simulations, have to select the option “real-time” rather than a day in the past.

12.5 Sensor data

Sensory Data are used by Urban Air Pollution Pilot. To have an accurate simulation, we have to collect data from the real life traffic, to create a specific traffic simulation and to validate it. To validate the whole simulation, we deployed Bosch AIRQ sensory with 5 sensors to collect emission data.

12.5.1 AIRQ sensory

Bosch Air Quality Sensor data is collected every day at midnight from 5 sensors across Győr through Bosch IoT platform. We have developed a downloader tool for that, which can download data for the given date from the IoT platform. After it is downloaded, it's formatted and uploaded to CKAN (Figure 107).

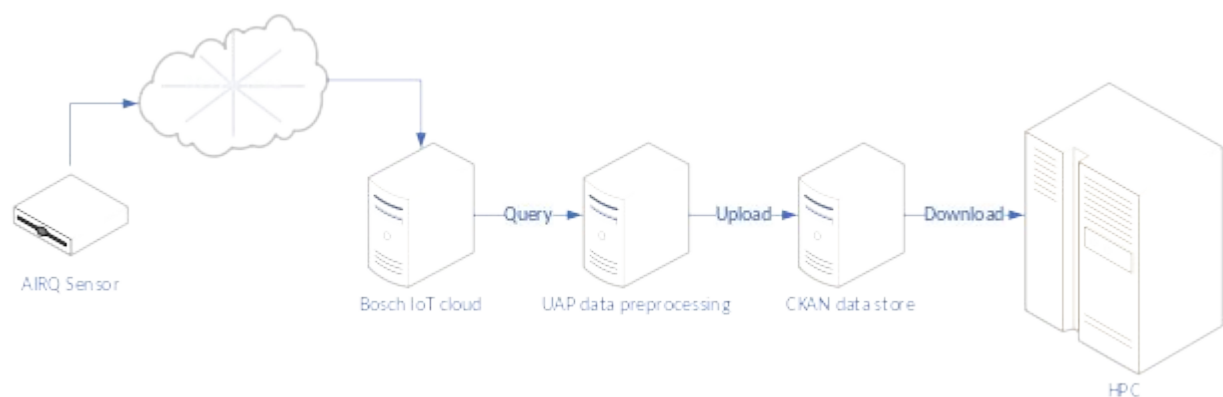


Figure 107. AIRQ sensory data flow.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies			Page:	181 of 174	
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

The data structure is the following:

- ESP_0_RH_AVG: Relative Humidity
- ESP_0_TEMP_AVG: Temperature average
- ES_0_PRESS: Air Pressure
- NO2_1_CORR: NO2 Concentration
- O3_0_CORR: O3 Concentration
- PS_0_PM10_CORR: PM10 Concentration
- PS_0_PM2P5_CORR: PM2.5 Concentration
- Type: DATA
- UTC: Timestamp
- deviceId: Device ID

The uploaded data has a 1-minute resolution and can be retrieved using the CKAN API. We have developed a simple downloader tool which can download data with the given dates. These daily merged data have typically around 25M size. Thanks to the simple CSV format most tools can directly work with it; only a minor programming is needed to use these data sources. There are five sensors deployed around the city, each in a frequented place, where most of the traffic goes through each day in the morning and in the afternoon (Figure 108).

Location Map

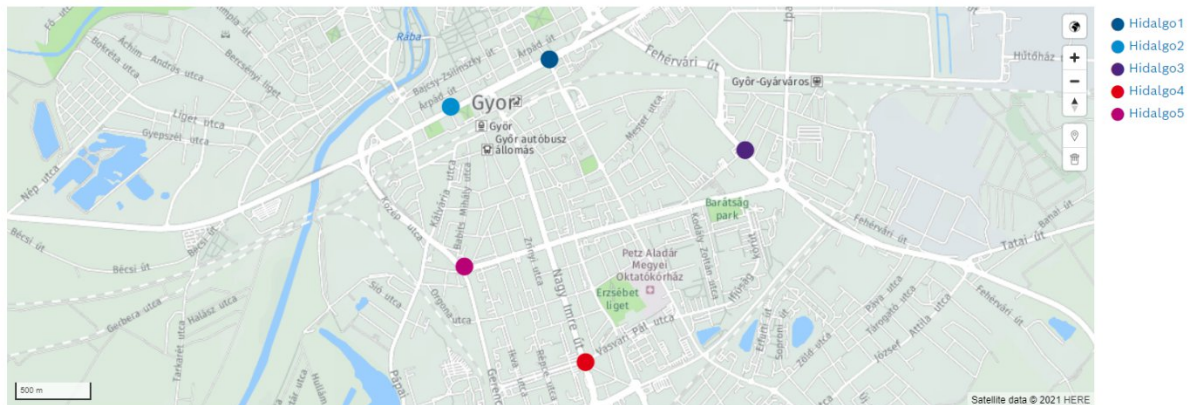


Figure 108. Location of the Bosch AIRQ sensors.

12.5.2 Camera Data

The camera data flow is very similar to AIRQ sensory data. When a camera sees a car, it classifies the category of the car, registers its speed and encrypts the plate number with one

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	182 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

way encryption (so decryption is not possible) and sends the data to the GDS server deployed in the University by ARH. After that we create 2 different datasets, with and without plate numbers, and upload it to CKAN (Figure 109).

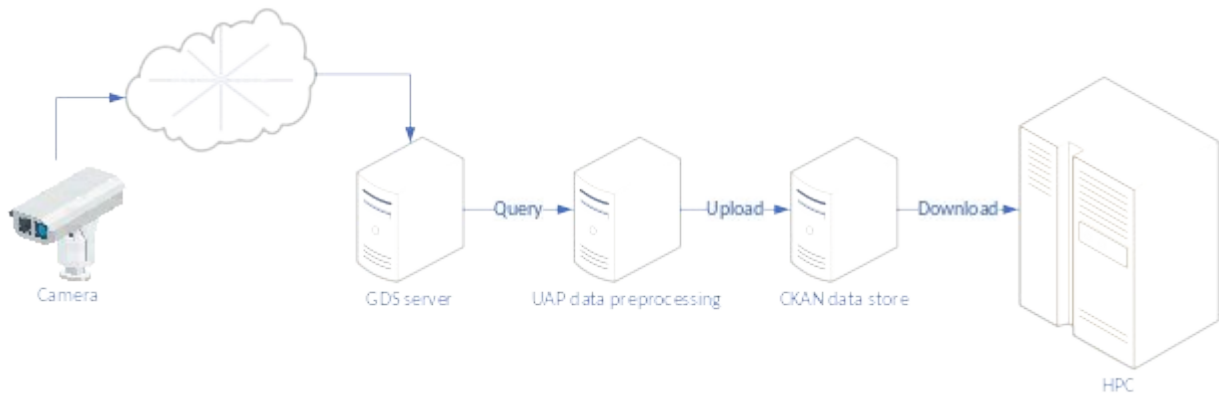


Figure 109. Traffic data flow.

ARH deployed more than 80 cameras in Győr to monitor and collect data from the whole city (Figure 110).

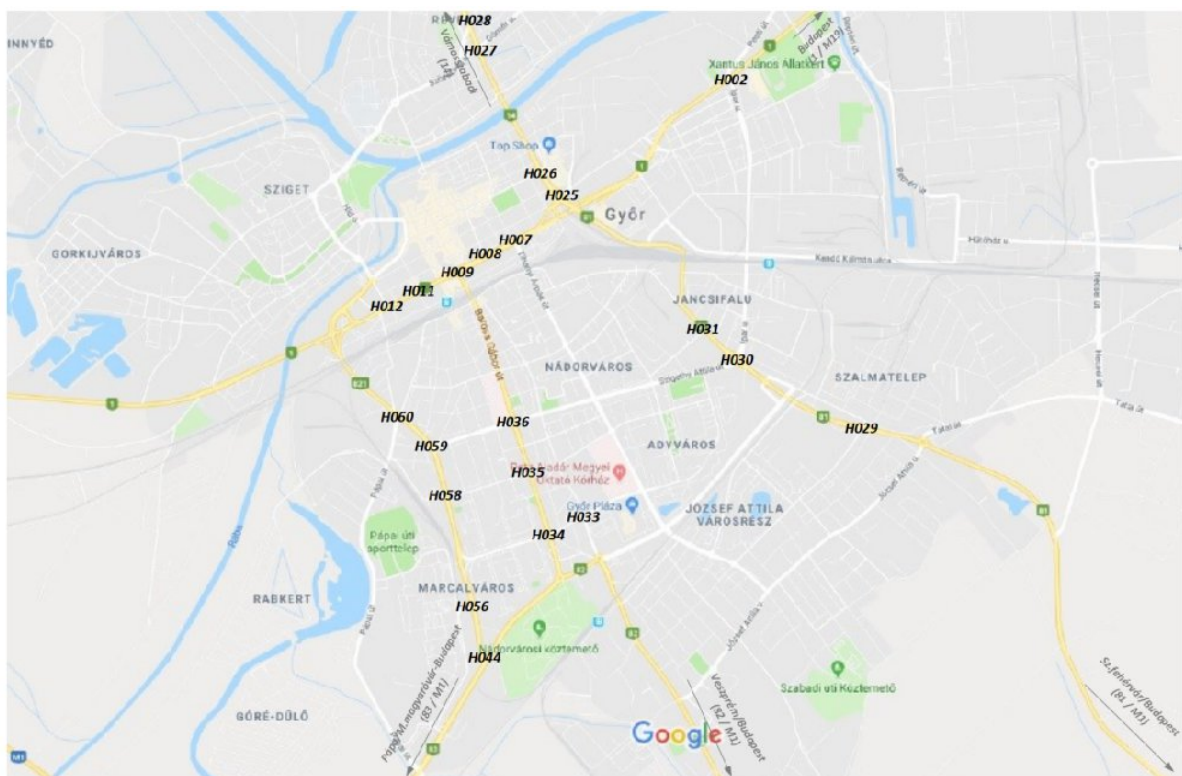


Figure 110. Camera locations.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies	Page:	183 of 174
Reference:	D3.5	Dissemination:	PU
		Version:	1.0
		Status:	Final

The exact locations of the cameras are shown in Figure 110, each location has more than one camera installed. The numbering of the locations is based on the Hungarian Road Maintainer intersection numbering, so camera H561 is the first camera in the H056 section.

12.6 Telecommunication data

Telecommunication data are coupled with the migration and social network pilots through a REST API. The API is a flexible HTTP based web technology, that could be used by different client applications and programming languages.

The API v1.0 endpoints, see Table 38, can be accessed using the following HTTPS URL [56]. Every member in the consortium can request an account and a key to use the API.

Endpoint	Endpoint Path	Method	Description
Query	/api/v1/query/	GET	Requests the CDRs of a specific time period and origin and destination prefix. Depending on the use case/pilot (migration or social networks) the aggregation type of the CDRs can be set.
Check	/api/v1/check/	GET	Checks if a specific query with a given request_id is completed and ready for download or not yet.
Download	/api/v1/download/	GET	Download the data of a specific query by a given request_id. The returned data can be in plain or zipped csv.

Table 38 . Endpoints of the Call Detail Record (CDR) API.

In the following, the workflow of a query is described.

1. Figure 111 shows that the client sends a POST query to retrieve Call Detail Records (CDRs) of a specific time period, origin and destination prefixes using this URL:

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies			Page:	184 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0
				Status:	Final

https://hidalgo.msc-services.net/api/v1/query/{aggregation_type}/{from_date}/{to_date}/{origin_prefix}/{dest_prefix}/. Table 39 provides a description of the different parameters of a query. The server returns a JSON object containing the parameters sent in the query plus a server-side generated request-id that can be used in the other endpoints.

- The client can check if a specific query with a given request_id is completed and ready for download or not yet using this URL: https://hidalgo-cdrs.msc-services.net/api/v1/check/{request_id}. The server returns a JSON file containing the request_id and a status attribute (IN_PROGRESS, COMPLETED, ERROR).

The data can be downloaded in plain or zipped CSV using this URL: https://hidalgo-cdrs.msc-services.net/api/v1/download/{request_id}/{format}{request_id}/{format}. The CDRs are aggregated depending on the migration or social network use cases. [Table 39](#) and [Table 39. Parameters Endpoints](#).

- describe the data when the user requested plain (used by the migration pilot) or social (used by the social network pilot), respectively.

The server accepts only query requests with a time period not longer than a month in order to speed up the processing of the requests.

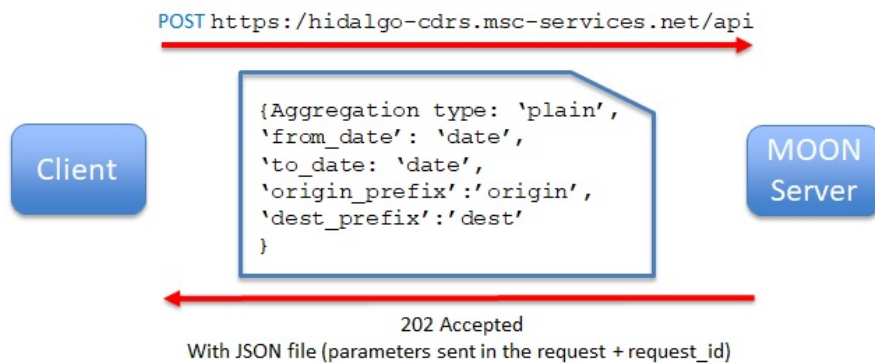


Figure 111. Workflow of a query.

Name	Type	Parameter Type	Mandatory or optional	Description
aggregation_type	String	Path	mandatory	The CDRs are aggregated according to the migration (plain)

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	185 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

				or social network (social) pilots
from_date	Date	Path	mandatory	Starting date in UTC (e.g. 2017-02-02-1600)
to_date	Date	Path	mandatory	End date in UTC (e.g. 2018-06-1600)
origin_prefix	String	Path	mandatory	Prefix of the origin country
dest_prefix	String	Path	mandatory	Prefix of the destination country
api_key	String	GET	mandatory	API key

Table 39. Parameters Endpoints.

Field	Description
TIMESTAMP	Timestamp when call is initiated
A_NUMBER	Hashed number of calling party
B_NUMBER	Hashed number of called party
A_NETWORK	Network of calling party
B_NETWORK	Network of called party
CALL_DURATION	Call duration, if call is established
A_ROAMING	Roaming network of the calling party
A_GPS_LOCATION	GPS location of the city (A_NUMBER) if a home call is initiated
B_GPS_LOCATION	GPS location of the city (B_NUMBER) if a home call is initiated

Table 40. Description of the plain CDRs.

B_numbers	Description
(B_number_1, B_network)	[(A_number_1, country, A_network, number of calls),

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies	Page:	186 of 174				
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status:	Final

B_numbers	Description
	(A_number_2, country, A_network, number of calls), ...]
(B_number_2, B_network)	[(A_number_1, country, A_network, number of calls), (A_number_2, country, A_network, number of calls), ...]
...	...

Table 41. An example of aggregated CDRs for the social network pilot.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	187 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

13 Conclusions

This document delivered the information about the final status of the work in work package 3. The leitmotif of this publication was to provide high-quality information on the evaluation of application performance and techniques to achieve it. The achievements in the areas were highly stressed: co-design, optimization and profiling of HPC pilot's simulations, infrastructure benchmarking and development of applications for HPDA, the data management system improvements, unified secure access to resources, visualization enhancements and delivering coupling technologies facilitating seamless module integration.

First of all, it should be mentioned that all applications developed under the HiDALGO project have undergone a process of significant development from single-threaded applications to highly parallelized and optimized applications that consciously use the infrastructure on which they are run. One of the most visible sign of the progress achieved in the application development is exceeding the KPI scalability indicator not for one but two pilot applications: OpenFoam from Urban Air Pollution (100 096 cores) and KPM from Social Network (more than 130k cores). This is substantial progress comparing the previous report (D3.4): Urban Air Pollution (4k cores) and KPM from Social Network (32k cores).

Thanks to the numerous tests and analyses carried out, we know how applications behave in various infrastructure environments and what to expect when running in various configurations. We learned that Flee performs better on newer-generation systems (Hawk, Altair), OpenFOAM benefits from the fat nodes (Hawk), which at the same way impede its scalability on higher core counts, KPM application is computationally efficient but requires tuning of its communication, SN-simulator is heavily unbalanced but can be efficiently scaled on modern systems (Vulcan, Altair).

Much effort has also been invested in applying various optimization techniques. It required an in-depth knowledge of the application, fragments generating the highest load, and only then choosing a solution, whether in the hardware or software range, to increase performance. In many cases, however, it was possible to do it, by a dozen or so percent in the scale of the entire application. Flee overall performance was improved in a range from 1.4x to 1.8x, OpenFOAM improved parallel efficiency from around 8% to more than 47%, speedup of SN Simulator and KPM by means of various methods improved from 2 to 4 times.

As part of the activities carried out in the WP3 package, several data analytics applications were prepared. In the case of the Migration pilot, there was a statistical analysis of the simulation results. For the Urban Air Pollution pilot, two applications were developed in the

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	188 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

field of Snapshot Matrix SVD (post-processing of snapshot matrices) and Air quality Index (indicating air pollution levels). The SN pilot was equipped with analytics that allow to determine statistics from real-world Twitter data that can be then compared with the output of the Twitter simulator, in order to verify simulator's accuracy in predicting Twitter user behaviour.

Facilitation of the CKAN by new extension enabling new transfer protocols improved overall data management performance in this respect. This is especially true for transporting files larger than 10GB.

Ensuring transparency in access to many services constituting the elements of the HiDALGO landscape required the preparation of a solution ensuring SSO. It was developed in collaboration with WP5 based on solutions provided by Hashicorp's Vault, Keycloak IDM and Cloudify with its Croupier extension. This allowed to meet the overriding condition of not installing or configuring anything on the remote server side, which is often prohibited due to security policy.

The implementation of coupling technologies was based on solutions specific to individual pilots. For the Migration pilot two cyclic (two-way) coupling approaches for multiscale simulations and couple the macro and micro scale models was established. In case of Urban Air Pollution a pre-processing workflow that integrates many different tools and applications was designed. The coupling task on Twitter monitoring system for Social Network, is built upon flagging certain topics and acquiring data on message parameters regarding these topics. As a complementary functionality for coupling technologies, a set of operations has been implemented that allow to download data from external sources, e.g. services for weather, air quality, street camera and telecommunication.

In conclusion, it should be noted that the WP3 package significantly contributed to the development of tools in the HiDALGO project, both in terms of the functionality provided as part of the designed workflow (data analytics, data management, security, visualizations, coupling technologies) and the efficiency achieved by these applications (co-design, scalability, ensemble scenarios, accelerators utilization).

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	189 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

References

- [1] P. N., "HiDALGO D3.1 Report on Benchmarking and Optimisation v1.8 - Revised version," 2021.
- [2] L. M., "Initial Specifications for HPC Scalability Optimisation, HPDA Model Implementation, Data Management, Visualisation and Coupling Technologies - Revised version," 2021.
- [3] L. M., "Intermediate Report on Implementation and Optimisation Strategies," 2021.
- [4] G. D., "Intermediate Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies," 2021.
- [5] G. D., "Final Implementation Report of the Pilot and Future Applications," 2022.
- [6] G. S., "Final Benchmark Results for Innovative Architectures," 2022.
- [7] K. M., "Final Report on Requirements, Components and Workflow Integration," 2021.
- [8] "Performance Optimisation and Productivity - A Centre of Excellence in HPC," [Online]. Available: <https://pop-coe.eu/>.
- [9] "The Python Profilers," 2022. [Online]. Available: <https://docs.python.org/3/library/profile.html>.
- [10] "A light-weight MPI profiler," 2022. [Online]. Available: <https://software.llnl.gov/mpiP/>.
- [11] "Linux profiling with performance counters," 2022. [Online]. Available: https://perf.wiki.kernel.org/index.php/Main_Page.
- [12] "POP - list of metrics," [Online]. Available: <https://co-design.pop-coe.eu/metrics/index.html>.
- [13] "Scalasca," [Online]. Available: <https://www.scalasca.org/>.
- [14] "Score-P - Scalable Performance Measurement Infrastructure for Parallel Codes," [Online]. Available: <https://www.vi-hps.org/projects/score-p/>.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	190 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

- [15] "Flee 2.0," 2021. [Online]. Available: <https://github.com/djgroen/flee/releases/tag/v2.0>.
- [16] "cProfile - the Python Profilers," [Online]. Available: <https://docs.python.org/3/library/profile.html>.
- [17] "OpenFOAM," [Online]. Available: <https://www.openfoam.com/>.
- [18] KPM, "KPM application," 2022. [Online]. Available: <https://github.com/sarming/kpm>.
- [19] SNSIM, "Social Network Simulator," 2022. [Online]. Available: <https://github.com/sarming/propagation>.
- [20] "Stanford Network Analysis Platform," 2022. [Online]. Available: <https://snap.stanford.edu>.
- [21] "PRACE project," [Online]. Available: <https://prace-ri.eu/>.
- [22] "MareNostrum," 2022. [Online]. Available: <https://www.bsc.es/marenostrom/marenostrom>.
- [23] "SuperMUC," 2022. [Online]. Available: <https://doku.lrz.de/display/PUBLIC/SuperMUC-NG>.
- [24] "SNAP," 2022. [Online]. Available: <https://snap.stanford.edu/data/>.
- [25] "Pokec at CKAN," 2022. [Online]. Available: <https://ckan.hidalgoproject.eu/dataset/pokec-relationship-graphs>.
- [26] D. Suleimenova, H. Arabnejad, W. N. Edeling and D. Groen, "Sensitivity-driven simulation development: a case study in forced migration," *Philosophical Transactions of the Royal Society A*, 29 March 2021.
- [27] D. Groen, D. Bell, H. Arabnejad, D. Suleimenova, S. J. E. Taylor and A. Anagnostou, "Towards Modelling the Effect of Evolving Violence on Forced Migration," *Computational Science - ICCS 2021*, vol. 12746, p. 502, 2021.
- [28] "The Armed Conflict Location & Event Data Project," [Online]. Available: <https://acleddata.com/>.
- [29] "UNHCR data portal," 2022. [Online]. Available: <https://data2.unhcr.org/en/search>.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	191 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

- [30] "Muscle 3," [Online]. Available: <http://muscle3.readthedocs.io>.
- [31] "Polytope," [Online]. Available: <https://github.com/ecmwf-projects/polytope-server>.
- [32] "NUMBA," 2022. [Online]. Available: <https://numba.pydata.org/>.
- [33] "Excellerat project," [Online]. Available: <https://www.excellerat.eu/>.
- [34] L. M., "Innovative HPC Trends and the HiDALGO Benchmarks," 2022. [Online].
- [35] "RapidCFD - OpenFOAM running on GPU," [Online]. Available: <https://simflow.com/rapid-cfd-gpu/>.
- [36] "CUDA from NVIDIA," [Online]. Available: <https://developer.nvidia.com/cuda-zone>.
- [37] "Apache Spark," [Online]. Available: <https://spark.apache.org/>.
- [38] "The MapReduce paradigm," [Online]. Available: <https://www.ibm.com/docs/en/netezza?topic=guide-mapreduce-paradigm>.
- [39] "Apache Hadoop," [Online]. Available: <https://hadoop.apache.org/>.
- [40] "Directive 2008/50/EC of the European Parliament and of the Council of 21 May 2008," [Online]. Available: <https://eur-lex.europa.eu/legal-content/EN/LSU/?uri=celex:32008L0050>.
- [41] "Apache Spark Machine Learning Library (MLlib)," [Online]. Available: <https://spark.apache.org/docs/latest/ml-guide.html>.
- [42] "Dimensionality Reduction - RDD-based API," 2022. [Online]. Available: <https://spark.apache.org/docs/latest/ml-lib-dimensionality-reduction.html>.
- [43] "Python Graphframes library," [Online]. Available: <https://pypi.org/project/graphframes/>.
- [44] S. Xie, R. Girshick, P. Dollar, Z. Tu and K. He, "Aggregated Residual Transformations for Deep Neural Networks".
- [45] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks For Large-Scale Image Recognition," no. <https://arxiv.org/pdf/1409.1556.pdf>, 2015.
- [46] "PyTorch Dataparallel mechanism," [Online]. Available:

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	192 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final

<https://pytorch.org/docs/stable/generated/torch.nn.DataParallel.html>.

[47] "CKAN," [Online]. Available: <https://ckan.org>.

[48] "HiDALGO Keycloak IDM," no. <https://hidalgo-idm.hlrs.de/>, 2022.

[49] "HiDALGO CKAN Portal," 2022. [Online]. Available: <https://ckan.hidalgo-project.eu/>.

[50] "Vault by HashiCorp," 2022. [Online]. Available: <https://www.vaultproject.io/>.

[51] "Visualizer," 2022. [Online]. Available: <https://visualization.hidalgo-project.eu/>.

[52] "NASA Task Load Index," [Online]. Available: <https://humansystems.arc.nasa.gov/groups/tlx/downloads/TLXScale.pdf>.

[53] "Comprehensive penetration testing from certified experts," [Online]. Available: <https://www.bulletproof.co.uk/penetration-testing>.

[54] "A distributed, reliable key-value store for the most critical data of a distributed system," [Online]. Available: <https://etcd.io/>.

[55] "GitHub - Aviso," [Online]. Available: <https://github.com/ecmwf/aviso>.

[56] "Telecommunication data API," [Online]. Available: <https://hidalgo-cdrs.msc-services.net/api/>.

Document name:	D3.5 Final Report on Benchmarking, Implementation, Optimisation Strategies and Coupling Technologies				Page:	193 of 174
Reference:	D3.5	Dissemination:	PU	Version:	1.0	Status: Final