

HP-SEE

High-Performance Computing Infrastructure for South East Europe's
Research Communities



Deliverable 8.4

Assessment of interoperability and scalability solutions

Author(s): Gabor Roczei (editor)

Status –Version: final – k

Date: February 28, 2013

Distribution - Type: Public

Code: HPSEE-WP8-HU-23-D8.4-j-2013-02-25

Abstract: This deliverable is a detailed report on how the line of actions and recommendation proposed in D8.1 improved the scalability of HP-SEE applications. The deliverable also presents the level of the internal harmonization (inside the SEE infrastructure) and of the external harmonization (related to pan-European level), that was reached following a number of interoperability actions that were implemented within this work-package.

© Copyright by the HP-SEE Consortium

The HP-SEE Consortium consists of:

GRNET
ICCT-BAS
IFIN-HH
TUBITAK ULAKBIM
NIIFI
IPB
UPT
UOBL ETF
UKIM
UOM
RENAM
IIAP NAS RA
GRENA
AZRENA

Coordinating Contractor
Contractor
Contractor
Contractor
Contractor
Contractor
Contractor
Contractor
Contractor
Contractor
Contractor
Contractor
Contractor
Contractor

Greece
Bulgaria
Romania
Turkey
Hungary
Serbia
Albania
Bosnia-Herzegovina
FYR of Macedonia
Montenegro
Moldova (Republic of)
Armenia
Georgia
Azerbaijan

This document contains material, which is the copyright of certain HP-SEE beneficiaries and the EC, may not be reproduced or copied without permission.

The commercial use of any information contained in this document may require a license from the proprietor of that information.

The beneficiaries do not warrant that the information contained in the report is capable of use, or that use of the information is free from risk, and accept no liability for loss or damage suffered by any person using this information.

Document Revision History

Date	Issue	Author/Editor/Contributor	Summary of main changes
May 15 th , 2012	a	Gabor Roczei	Initial version of ToC
May 16 th , 2012	b	Gabor Roczei, Ioannis Liabotis, Barbara Toth	ToC comments implemented
May 31 th , 2012	c	Boro Jakimovski, Dusan Vudragovic, Emil Slusanschi, Silviu Panica, Ionut Vasile, Dobromir Georgiev, Ljupco Pejov, Anastas Mishev, Dragan Jakimovski, Neki Frasheri, Gabor Roczei, Ioannis Liabotis, George Kastellakis, Manthos G. Papadopoulos, Heribert Reis, Neki Frasheri, Betim Cico	First contributions from partners (application scalability analysis, HPC software stack information)
June 12 th , 2012	d	Yiota Poirazi, Anastasis Oulas, Dragan Jakimovski, Neki Frasheri, Betim Cico, Ramaz Kvatadze, Jumber Kereselidze, George Mikuchadze, Todor Gurov, Emanouil Atanassov, Aneta Karaivanova, Dobromir Georgiev	Second contributions from partners (application scalability analysis)
January 24 th , 2013	e	Branko J. Drakulić, Ivan Juranic, Ramaz Kvatadze, George Mikuchadze, Ljupčo Pejov, Anastas Mishev, Jane Jovanovski, Boro Jakimovski, Dragan Jakimovski, Neki Frasheri, Dusan Vudragovic, Antun Balaz, Mihajlo Savic, Luka Filipovic, Nicolai Iliuha, Boris Rybakin, Peter P. Bogatencov, Secrieru Renam, Gergely Windisch, Akos Balasko, Miklos Kozlovsky, Gabor Roczei	Third contributions from partners (application scalability analysis)
February 15 th , 2013	f	Ioannis Liabotis, Gergely Windisch, Akos Balasko, Miklos Kozlovsky, Gabor Roczei	Helpdesk contribution; gUSE portal contribution; software stacks added; HP-SEE modules added; software monitoring added
February 20 th , 2013	g	Gabor Roczei, Vladimir Slavnic, Dragos Zabet, Ionut Vasile, Todor Gurov, Silviu Panica, Anastas Mishev, Emil Slusanschi, Alexandru Herisanu, Anastas Misev	Contributions from HPC centres
February 22 th , 2013	h	Gabor Roczei	Prefinal version
February 24 th , 2013	i	Gabor Roczei, Ioannis Liabotis	Scalability methodology table added; conclusion and recommendation has been extended
February 25 th , 2013	j	Gabor Roczei, Milosavljevic Lidija	Quality control check
February 28 th	k	Ioannis Liabotis, Ognjen Prnjat	Final editing

Preface

The core European eInfrastructure for large-scale eScience research consists of the backbone GÉANT network; distributed storage & computing infrastructure - European Grid Initiative (EGI); and the PRACE initiative providing tier-0 High Performance Computing (HPC) infrastructure. South-East European eInfrastructure initiatives aim for equal participation of the less-resourced countries of the region in the European trends. SEEREN initiative established a regional network and the SEE-GRID initiative the regional Grid, with majority of countries now equal partners in GÉANT and EGI. BSI project established the GÉANT link to the Caucasus, active until mid-2010. However, HPC involvement of the region is limited. Only few HPC installations are available, not open to cross-border research; while the less-resourced countries have no mechanism established for interfacing to the pan-European HPC initiatives.

HP-SEE focuses on a number of strategic actions. First, it will link the existing and upcoming HPC facilities in the region into a common infrastructure, and provide operational solutions for it. As a complementary action, the project will establish and maintain the GÉANT link for Caucasus. Moreover, it will open this HPC infrastructure to a wide range of new user communities, including those of less-resourced countries, fostering collaboration and providing advanced capabilities to researchers, with an emphasis on strategic groups in computational physics, chemistry and life sciences. Finally, it will ensure establishment of national HPC initiatives, and act as a SEE bridge for PRACE. In this context, HP-SEE will aim to attract the local political & financial support for a long-term sustainable eInfrastructure.

HP-SEE aspires to contribute to the stabilisation and development of South-East Europe, by overcoming fragmentation in Europe and stimulating eInfrastructure development and adoption by new virtual research communities, thus enabling collaborative high-quality research across a various spectrum of scientific fields.

The main objectives of the HP-SEE project are:

1. Empowering multi-disciplinary virtual research communities. HP-SEE will involve and address specific needs of a number of new multi-disciplinary international scientific communities (computational physics, computational chemistry, life sciences, etc.) and thus stimulate the use and expansion of the emerging new regional HPC infrastructure and its services.
2. Deploying integrated infrastructure for virtual research communities. HP-SEE will provide and operate the integrated South-East European eInfrastructure and specifically the HPC eInfrastructure for the region. In the context of the project, this focuses on operating the HPC infrastructure and specific end-user services for the benefit of new user communities, and establishing the continuity of the GÉANT link to Caucasus.
3. Policy development and stimulating regional inclusion in pan-European HPC trends. The inclusion of the new Virtual Research Communities and the inauguration of the infrastructure, together with a set of coordinated actions aimed at setting up HPC initiatives in the region, aim to contribute to regional development and ensure that countries in this outermost European region will join the pan-European HPC trends.
4. Strengthening the regional and national human network. The project will capitalize on the existing human network and underlying research infrastructure

to strengthen scientific collaboration and boost more effective high-quality research and cooperation among participating SEE communities.

The expected results of the project are:

1. Project management information system established
2. Promotional package available
3. National HPC initiatives in core countries established
4. HPC related Memorandum of Understanding on the regional level
5. Set of inter-disciplinary applications running on regional infrastructure
6. Regional HPC resources available to target virtual research communities
7. Realization of Network Connections and deployment of relevant management and monitoring tools
8. Application software environment deployed
9. Establishment of a relevant for the region HPC technology watch

The HP-SEE project kicked-off in September 2010 and is planned to be completed by May 2013. It is coordinated by GRNET with 13 contractors participating in the project: major lead institutes in the region for computing aspects of eInfrastructures in Bulgaria, Romania, Turkey, Hungary, Serbia, Albania, Bosnia-Herzegovina, FYROM, Montenegro, Moldova (Republic of), Armenia, Georgia, Azerbaijan. The total budget is 3.885.196 €. The project is funded by the European Commission's Seventh Framework Programme for Capacities-Research Infrastructures.

The project plans to issue the following deliverables:

Del. no.	Deliverable name	Nature	Security	Planned Delivery
D1.1	Project management information system and "grant agreement" relationships	R	CO	M01
D2.1	Procurement guidelines analysis	R	PU	M04
D2.2	National HPC task-force modelling and organizational guidelines	R	PU	M10
D2.3	HPC centre setup cookbook	R	PU	M14
D2.4	Regional collaboration modalities and European integration feasibility	R	PU	M16
D2.5	Final report on international collaboration	R	PU	M36
D3.1	Internal and external web site, docs repository and mailing lists	R	PU	M02
D3.2	Promotional package	R	PU	M03
D3.3	HPC training and dissemination plan	R	PU	M03
D3.4	Regional & national training and dissemination events report	R	PU	M12
D3.5	Promotional package	R	PU	M13
D3.6	Regional & national training and dissemination	R	PU	M36

	events report			
D3.7	Final plan for the use and dissemination of foreground	R	PU	M36
D4.1	Target applications analysis	R	PU	M05
D4.2	Report on application deployment and support	R	PU	M12
D4.3	HPC programming techniques guidelines	R	PU	M20
D4.4	User community engagement and applications assessment	R	PU	M31
D4.5	Pilot Call Report	R	PU	M35
D5.1	Infrastructure, Network and Management Deployment Plan	R	PU	M05
D5.2	Infrastructure overview and assessment	R	PU	M12
D5.3	Infrastructure deployment plan	R	PU	M14
D5.4	Infrastructure overview and assessment	R	PU	M34
D6.1	Tender evaluation results 1	R	PU	M06
D6.2	Tender evaluation results 2	R	PU	M06
D6.3	Final Tender Results	R	PU	M11
D7.1	Network Implementation and equipments configuration	R	PU	M13
D7.2	Deployment of essential network services and management tools	R	PU	M17
D7.3	CSIRT/NOC Cooperation Report and Harmonization of Efforts among South Caucasus NREs	R	PU	M23
D7.4	Analysis of the connectivity requirements of the HPC users in the beneficiary regions	R	PU	M34
D8.1	Software scalability analysis and interoperability issues assessment	R	PU	M06
D8.2	Design of interoperability and scalability solutions	R	PU	M12
D8.3	Permanent technology watch report	R	PU	M35
D8.4	Assessment of interoperability and scalability solutions	R	PU	M30

Legend: R = Report, O = Other, PU = Public, CO = Confidential (only for members of the consortium incl. EC).

Table of contents

1. Introduction.....	21
2. Analysis of applications scalability.....	22
2.1. HC-MD-QM-CS	24
2.1.1. Summary	24
2.1.2. Implemented scalability actions	24
2.1.3. Benchmark dataset	25
2.1.4. Hardware platforms	25
2.1.5. Execution times	25
2.1.6. Memory Usage	27
2.1.7. Profiling	27
2.1.8. Communication	28
2.1.9. I/O	28
2.1.10. CPU and cache	28
2.1.11. Analysis	28
2.2. GENETATOMICS	29
2.2.1. Summary	29
2.2.2. Parallelization	29
2.2.3. New way to measure relative scalability of algorithm	29
2.2.4. Benchmark dataset	30
2.2.5. Hardware platforms	30
2.2.6. Execution times	30
2.2.7. Memory Usage	31
2.2.8. Profiling	32
2.2.9. Communication	32
2.2.10. I/O	32
2.2.11. CPU and cache	32
2.2.12. Analysis	32
2.3. GIM	33
2.3.1. Summary	33
2.3.2. Implemented scalability actions	33
2.3.3. Benchmark dataset	33
2.3.4. Hardware platforms	34
2.3.5. Execution times	34
2.3.6. Memory Usage	36
2.3.7. Profiling	36
2.3.8. Communication	36
2.3.9. I/O	37
2.3.10. CPU and cache	37
2.3.11. Derived metrics	37
2.3.12. Analysis	37
2.4. MSBP	38
2.4.1. Summary	38
2.4.2. Implemented scalability actions	38
2.4.3. Benchmark dataset	38
2.4.4. Hardware platforms	38
2.4.5. Execution times	38
2.4.6. Memory Usage	39
2.4.7. Communication	39
2.4.8. I/O	39
2.4.9. CPU and cache	39
2.4.10. Analysis	39

2.5. SET	40
2.5.1. Summary	40
2.5.2. Implemented scalability actions	40
2.5.3. Benchmark dataset	41
2.5.4. Hardware platforms	41
2.5.5. Execution times	41
2.5.6. Memory Usage	42
2.5.7. Profiling	42
2.5.8. Communication	43
2.5.9. I/O	43
2.5.10. CPU and cache	43
2.5.11. Analysis	43
2.6. NUQG	44
2.6.1. Summary	44
2.6.2. Implemented scalability actions	44
2.6.3. Benchmark dataset	44
2.6.4. Hardware platforms	44
2.6.5. Execution times	44
2.6.6. Memory Usage	46
2.6.7. Profiling	47
2.6.8. Communication	47
2.6.9. I/O	48
2.6.10. Analysis	48
2.7. SFHG	49
2.7.1. Summary	49
2.7.2. Implemented scalability actions	49
2.7.3. Benchmark dataset	50
2.7.4. Hardware platforms	50
2.7.5. Execution times	50
2.7.6. Memory Usage, CPU and cache	52
2.7.7. Profiling	52
2.7.8. Communication and I/O	52
2.7.9. Analysis	52
2.8. CFD OF	53
2.8.1. Summary	53
2.8.2. Implemented scalability actions	53
2.8.3. Benchmark dataset	53
2.8.4. Hardware platforms	53
2.8.5. Execution times	53
2.8.6. Memory Usage, CPU and cache	54
2.8.7. Profiling	54
2.8.8. Communication and I/O	54
2.8.9. Analysis	54
2.9. DNAMA	56
2.9.1. Summary	56
2.9.2. Implemented scalability actions	56
2.9.3. Benchmark dataset	56
2.9.4. Hardware platforms	56
2.9.5. Execution times	56
2.9.6. Memory Usage	58
2.9.7. Profiling	58
2.9.8. Communication	58
2.9.9. I/O	58
2.9.10. CPU and cache	58
2.9.11. Analysis	58
2.10. AMR_PAR: PORTING FROM WINDOWS TO LINUX	60
2.10.1. Summary	60

2.10.2. Implemented scalability actions	60
2.10.2.1 The sources of some problems when porting applications' code from Windows to Linux platform	60
2.10.2.2 Problems when porting interactive applications from Windows to Linux platform	61
2.10.2.3 The development of portable code with a graphical interface.....	61
2.10.2.4 Packages for helping to solve the problem of porting applications	61
2.10.2.5 Specificity of porting of AMR_PAR application (64 bit, Fortran).....	62
2.10.2.6 Intel Parallel Studio XE 2011 with VS2010.....	62
2.10.3. Hardware platforms	63
2.10.4. Execution times	63
2.10.5. Memory Usage	63
2.10.6. Analysis	64
2.11. DEEPALIGNER AND DISEASEGENEMAPPER	65
2.11.1. Summary	65
2.11.2. Application description	65
2.11.3. Implemented scalability actions	65
2.11.4. Benchmark dataset.....	65
2.11.5. Hardware platforms	65
2.11.6. Software platforms	66
2.11.7. Execution times	67
2.11.8. Further optimization	68
2.11.9. Memory Usage	69
2.11.10. Profiling.....	69
2.11.11. Communication	70
2.11.12. I/O.....	70
2.11.13. Analysis.....	70
2.12. FMD-PA	71
2.12.1. Summary	71
2.12.2. Implemented scalability actions	71
2.12.3. Benchmark dataset.....	72
2.12.4. Hardware platforms	72
2.12.5. Execution times	72
2.12.6. Memory Usage	72
2.12.7. Analysis	72
2.13. COMPCHEM	73
2.13.1. Summary	73
2.13.2. Implemented scalability actions	73
2.13.3. Benchmark dataset.....	74
2.13.4. Hardware platforms	74
2.13.5. Execution times	74
2.13.6. Memory Usage	76
2.13.7. Profiling.....	76
2.13.8. Communication	76
2.13.9. I/O	76
3. Software harmonization.....	77
3.1. HARMONIZATION STATUS	77
3.2. HP-SEE SOFTWARE STACK.....	79
3.2.1. Minimal software stack (mandatory for all HPC centre)	79
3.2.2. Recommended software stack.....	81
4. Interoperability.....	83
4.1. RESOURCE MANAGEMENT SYSTEM AND USER AUTHENTICATION	83
4.1.1. Resource management system	83
4.1.1.1 Registration of new users	83
4.1.2. LDAP.....	85
4.2. HP-SEE COMMON ENVIRONMENT	86

4.3. SOFTWARE MONITORING WITH NAGIOS	87
4.4. USAGE OF GUSE PORTAL	88
4.4.1. Availability	88
4.4.1.1 DiseaseGeneMapper	88
4.4.1.2 Deep Aligner	88
4.4.2. Requirements	88
4.4.3. Installation (for portal administrators)	88
4.4.4. Using the portlets.....	90
4.4.5. Creating a new DiseaseGeneMapper query	91
4.4.6. DiseaseGeneMapper job lifecycle.....	91
4.4.7. Submitting jobs	91
4.4.8. Downloading results	92
4.4.9. Parameters	93
4.4.10. DeepAligner.....	93
4.4.10.1 Parameters	94
4.4.11. Appendix	94
4.5. HP-SEE HELPDESK SYSTEM.....	96
5. Conclusions.....	97

References

- [1] Project HP-SEE – 261499 – Annex I – Description of Work
- [2] HP-SEE Deliverable 4.1 - Target Applications Analysis
- [3] HP-SEE Deliverable 4.2 - Report on application deployment and support
- [4] HP-SEE Deliverable 4.3 - HPC programming techniques guidelines
- [5] HP-SEE Deliverable 5.3 – Infrastructure overview and assessment
- [6] HP-SEE Deliverable 8.1 – Software Scalability Analysis and Interoperability Issues Assessment
- [7] HP-SEE Deliverable 8.2 – Design of interoperability and scalability solutions
- [8] 1st HP-SEE Technical Review Report, 2011
- [9] HP-SEE Wiki, http://wiki.hp-see.eu/index.php/HP-SEE_Wiki
- [10] <http://www.prace-ri.eu/IMG/pdf/D6-2-2.pdf>
- [11] http://hpseewiki.ipb.ac.rs/index.php/Module_framework
- [12] <http://nagios.sourceforge.net/docs/nrpe/NRPE.pdf>
- [13] <http://www.deisa.eu/usersupport/user-documentation/deisa-common-production-environment/components-of-the-software-stacks>
- [14] GPU supported applications: <http://www.nvidia.com/object/gpu-applications.html>
- [15] Grid middleware access: <http://survey.ipb.ac.rs/index.php?sid=21188>
- [16] Titan supercomputer: <http://top500.org/system/177975>
- [17] HP-SEE common environment: <https://github.com/HP-SEE/hce>
- [18] Bioportal: <http://ls-hpsee.nik.uni-obuda.hu:8080/liferay-portal-6.0.5>

List of Figures

Figure 1 - HC-MD-QM-CS, demonstrate the scalability	26
Figure 2 - HC-MD-QM-CS, demonstrate the scalability, (log-log plot)	26
Figure 3 - Required computational time 1, HC-MD-QM-CS	27
Figure 4 - Required computational time 2, HC-MD-QM-CS	27
Figure 5 - Calculate speed up of algorithm	31
Figure 6 - 3D cuboid underground geosection	33
Figure 7 - Tests with OpenMP in HPCG	34
Figure 8 - Tests with OpenMP in Pecs SC	35
Figure 9 - Tests with MPI in HPCG	35
Figure 10 - Tests with MPI in Pecs SC	35
Figure 11 - CPU time on Blue Gene/P	42
Figure 12 - Execution time and speedup as a function of the number of CPU cores at PARADOX and HPCG clusters	45
Figure 13 - Ratio of the CPU execution time of the original MC and the improved quasi-MC code as a function of the precision of the amplitude.	45
Figure 14 - Speedup in the execution time of the NUQG GP module as a function of the number of CPU cores.	46
Figure 15 - Effective speedup of the NUQG GP application.	46
Figure 16 - NUQG GP total memory allocation.	46
Figure 17 - CPU / Wall time ratio as function of number of CPU cores (BA-01-ETFBL)	51
Figure 18 - CPU / Wall time as function of number of CPU cores (Pecs SC)	52
Figure 19 - Simulation and total times and speedup as function of number of CPU cores	54
Figure 20 - Cross section of burner after T=0,1 s and T=1,3 s (illustration)	55
Figure 21 - DNAMA MPI result	57
Figure 22 - DNAMA Pthreads result	58
Figure 23 - AMR_PAR, HPCG cluster, CPU and wall times	63
Figure 24 - Execution times measured by mpiBlast in seconds	67
Figure 25 - mpiBlast speedup using multiple CPU cores compared to running it on just one CPU core	68
Figure 26 - Efficiency of using multiple CPU cores	68
Figure 27 - Execution times in seconds using different number of Database Fragments	69
Figure 28 - HPCG cluster, FMD-PA	71
Figure 29 - Blue Gene cluster, FMD-PA	71
Figure 30 - Scalability of the NAMD 2.9, CPU's / Nodes	75
Figure 31 - Scalability of NAMD 2.9, CPU's	75
Figure 32 - Registration	83
Figure 33 - HPC Centres	84
Figure 34 - Statistics for the used CPU time	84
Figure 35 - LDAP topology	85
Figure 36 - HP-SEE module system	86
Figure 37 - Nagios Remote Plugin Executor	87
Figure 38 - Software monitoring with Nagios	87
Figure 39 - Concept of ASM	90
Figure 40 - Upload certificate to use the portlets	90
Figure 41 - DiseaseGeneMapper main window	91
Figure 42 - DeepAligner main window	91
Figure 43 - Executing DeepAligner query	92
Figure 44 - Downloading results from the applications	92
Figure 45 - Setting the parameters for Disease Gene Mapper	93
Figure 46 - Setting the parameters for DeepAligner	94

List of Tables

<i>Table 1 – Methodologies used by the applications</i>	23
<i>Table 2 - Calculate speed up of algorithm</i>	31
<i>Table 3 - Calculate speed up of algorithm, HT</i>	31
<i>Table 4 - The volume of calculations for each iteration</i>	34
<i>Table 5 – Memory usage of GIM</i>	36
<i>Table 6 – Execution times of SET</i>	41
<i>Table 7 Comparison of the execution time and parallel efficiency of SET</i>	42
<i>Table 8 - Memory usage of NUQG GP codes.</i>	47
<i>Table 9 - Scalability test results for BA-01-ETFBL with up to 16 CPU cores</i>	50
<i>Table 10 - Scalability test results for Pecs SC with up to 48 CPU cores</i>	51
<i>Table 11 - Scalability test results for PARADOX with up to 32 CPU cores</i>	53
<i>Table 12 – DNAMA MPI result</i>	57
<i>Table 13 – DNAMA Pthreads result</i>	57
<i>Table 14 – Memory usage of AMR_PAR</i>	64
<i>Table 15 - Memory usage while executing the application. The results come from the maxvmem parameter of qacct</i>	69
<i>Table 16 - Execution time ratio of the jobs in the whole DGM and DA portlets</i>	69
<i>Table 17 - Execution time ratio inside Job2</i>	70
<i>Table 18 - I/O as measured using the IO parameter of qacct</i>	70
<i>Table 19 - Scalability of NAMD 2.9 on PARADOX/IPB</i>	74
<i>Table 20 - Scalability of NAMD 2.9 on HPCG/BG</i>	75
<i>Table 21 – Software stack status, input: D8.2</i>	78
<i>Table 22 – Software stack status, 22th February 2013</i>	79
<i>Table 23 – Minimal and recommended software stack status, 22th February 2013</i>	82

List of Equations

<i>Equation 1 - Measure relative scalability</i>	30
<i>Equation 2 - Measuring scalability of algorithm</i>	30
<i>Equation 3 - Harmonization metric</i>	78

Glossary

Acronym	Description
ACML	AMD Core Math Library
AMD	Advanced Micro Devices
AMR_PAR	Parallel algorithm and program for the solving of continuum mechanics equations using Adaptive Mesh Refinement
API	Application Programming Interface
ARC	Advanced Resource Connector
AREX	ARC Remote EXecution service
ASM	Application Specific Module
ATLAS	Automatically Tuned Linear Algebra Software
BES	Basic Execution Services
BLACS	Basic Linear Algebra Communication Subprograms
BLAS	Basic Linear Algebra Subprograms
BLAST	Basic Local Alignment Search Tool
CAD	Computer Aided Design
ccNUMA	Cache coherent Non-Uniform Memory Access
CFD	Computational Fluid Dynamics
CFDOF	CFD Analysis of Combustion
CHARMM	Chemistry at HARvard MacromolecularMechanics
COM	Component Object Model
CompChem	Quantum Mechanical, Molecular Mechanics, and Molecular Dynamics computation in chemistry
CPMD	Car-Parrinello Molecular Dynamics
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
DCOM	Distributed Component Object Model
DeepAligner	Deep sequencing for short fragment alignment
DFT	Discrete Fourier Transform
DiseaseGeneMapper	In-silico Disease Gene Mapper
DNA	Deoxyribonucleic acid
DNAMA	DNA Multicore Analysis
DRD	Data-Race Detector
EMI	European Middleware Initiative
ESSL	Engineering And Scientific Subroutine Library

FFTW	Fastest Fourier Transform in the West
FMD-PA	Design of fullerene and metal-diothiolene-based materials for photonic applications
GAMESS	General Atomic and Molecular Electronic Structure System
GCC	GNU Compiler Collection
GENETATOMICS	Genetic algorithms in atomic collisions
GIM	Geophysical Inversion Modeling
GPGPU	General-Purpose computing on Graphics Processing Units
GPU	Graphical Processing Unit
GROMACS	GRoningen MACHine for Chemical Simulations
GUI	Graphical user interface
gUSE	grid User Support Environment
HC-MD-QM-CS	Hybrid Classical/Quantum Molecular Dynamics – Quantum Mechanical Computer Simulation of Condensed Phases
HCE	HP-SEE common environment
HPC	High Performance Computing
Intel MKL	Intel Math Kernel Library
JSDL	Job Submission Description Language
LAPACK	Linear Algebra Package
LDAP	Lightweight Directory Access Protocol
LDIF	LDAP Data Interchange Format
MD	Molecular Dynamics
MESI	Modified, Exclusive, Shared, Invalid
MFC	Microsoft Foundation Classes
MIDL	Microsoft Interface Definition Language
MKL	Math Kernel Library
MPI	Message Passing Interface
MSBP	Modeling of some biochemical processes with the purpose of realization of their thin and purposeful synthesis
NAMD	NANoscale Molecular Dynamics
NCBI	National Center for Biotechnology Information
NRPE	Nagios Remote Plugin Executor
NUMA	Non-Uniform Memory Access
NUQG	Numerical study of ultra-cold quantum gases
OLE	Object Linking and Embedding
OpenCL	Open Computing Language
OpenCV	Open Source Computer Vision

OpenFOAM	Open Field Operation And Manipulation
OpenMP	Open Multiprocessing
OS	Operating System
PBS	Portable Batch System
PME	Particle Mesh Ewald
pNFS	Parallel Network File System
POSIX	Portable Operating System Interface for uniX
QDR	Quad Data Rate
RAM	Random Access Memory
RAxML	Randomized Axelerated Maximum Likelihood
RFC	Request for Comments
RT	Request Tracker
SDK	Software Development Toolkit
SEE-GRID	South Eastern European GRid-enabled eInfrastructure Development
SEEREN	South Eastern European Research & Education Network
SET	Simulation of electron transport
SFHG	Self Avoiding Hamiltonian Walk on Gaskets
SGE	Sun Grid Engine
SIMT	Single Instruction Multiple Threads
SMP	Symmetric multiprocessing
SPMD	Single Program Multiple Data
SPRNG	Scalable Parallel Random Number Generators
TORQUE	Terascale Open-Source and QUEue Manager
UNICORE	Uniform Interface to Computing Resources
UNIX	Dynamic-link library
VMD	Visual Molecular Dynamics
WS-PGRADE	WebService-Parallel Grid Run-time and Application Development Environment

Executive summary

What is the focus of this Deliverable?

This deliverable presents the key outcomes of WP8 activities WP8.1 and WP8.2 - namely "Software environment scalability analysis and interoperability issues" and "Software environment adjustment". The main aim is to describe in detail the achievements of the WP8 in terms of improving the scalability of the HP-SEE selected applications while also providing transparent job execution and usage experience across different HPC centers of the region. The deliverable makes use of the recommendations and guidelines presented in D8.2 [7] "Design of interoperability and scalability solutions" to demonstrate their effectiveness in the duration of the HP-SEE project.

What is next in the process to deliver the HP-SEE results?

The conclusions and recommendations from this deliverable will be used in the following HP-SEE activities:

- WP3.3: Plan, organize and participate in technical workshops and training events
- WP3.4: Develop and maintain training infrastructure and training community
- WP4.2: Port and optimize regional applications of interest
- WP4.3: Application deployment and support
- WP5.2: Implementation of the regional HPC infrastructure
- WP5.3: Resource Management
- WP5.4: Application Support
- WP5.5: Infrastructure Monitoring
- WP8.2: Software environment adjustment
- WP8.3: Permanent technology watch

Additionally, this deliverable will be used as an important input in preparation and formulation of the following future deliverables:

- D4.4: User community engagement and applications assessment
- D4.5: Pilot Call Report
- D5.4: Infrastructure overview and assessment
- D8.3: Permanent technology watch report

What are the deliverable contents?

The deliverable covers all scalability and interoperability actions which have been defined in the D8.2. Main deliverable contents are:

- Analysis of applications' scalability
- Harmonization status between the HPC centres
- HP-SEE software stacks
- HP-SEE common environment

- Resource management system and user authentication
- Software monitoring
- Bioinformatic portal

Conclusions and recommendations

Based on the work related to scalability and interoperability, performed within WP8, we can draw the following conclusions and recommendations for achieving better utilization of the infrastructure based on the types and variety of applications deployed in the HP-SEE infrastructure:

- MPI is the most widely used parallelization method used among the HP-SEE applications. This trend is seen also world-wide as most HPC codes have MPI implementations.
- The increase in the number of cores per node in the recent HPC systems has led to an increase of applications that are using the OpenMP parallelization paradigm as well as the hybrid model (MPI + OpenMP) to achieve higher scalability.
- Enabling hyper-threading (although it is documented not to benefit all applications) in some cases has been proven to provide some better performance as shown by at least one HP-SEE application.
- The usage of different compilers in some platforms and some applications can provide better performance. The same effect is less commonly observed by the usage of different MPI implementations.
- Applications' performance and scalability can vary based on the type of HPC systems that are being used for its deployment.
- HP-SEE common environment has been installed on the HPC centres which helps to improve the transparent access of the users.
- A second set of recommended software environment has been defined (mainly composed of user level libraries or complete application codes), to facilitate a more uniform interoperable infrastructure.
- By defining software stacks that are both designed for the needs of the HP-SEE applications, as well as adhering to the international and mainly European stacks, the HP-SEE infrastructure is highly harmonised at the regional as well as the European level.
- The modules framework is used by HP-SEE as well as PRACE for providing a mechanism that hides the underlying software configuration complexity.
- The HP-SEE Module's git repository [17] is publicly available via the project.
- Several operational and user level tools, such as the helpdesk, the access portal and the grid middleware, among others, provide interoperability solutions to the infrastructure.
- The Biportal [18] gives a good opportunity for the users to use the HPC sites in uniform and user friendly way.

- The scalability of the participating to the study applications varies depending on the application. Specific applications have demonstrated scalability of 4096 or 1024 cores.

1. Introduction

Two major topics are discussed in this document: scalability and interoperability assessment. A set of actions for achieving higher application scalability has been defined in D8.2; this set of actions has been implemented regarding several applications in the duration of the project. This deliverable presents the assessment of the results of the above process.

The main actions used by the HP-SEE applications to improve their scalability and therefore their performance are:

- Usage of different programming models
- Efficient usage of parallel libraries
- Efficient usage of compilers or compiler flags
- Usage of different compute technologies i.e. CPU vs GPU

Tools such as profilers and debuggers have assisted in the implementation of the above actions.

The second part of the document is about software harmonisation. WP8 has defined a metric within deliverable D8.2 which describes the harmonisation status of the HPC centres. The metric values have been calculated again in this document. The result has been improved since missing software components have been installed in the centres.

WP8 has defined two software stacks, which helped to improve the harmonisation level of the sites:

- Minimal software stack
- Recommended software stack

The minimal software stack should be installed on all sites (with some exceptions in case of non-suitability) while the recommended software stack contains optional software components that improve the interoperability of the infrastructure if used. HP-SEE Common Environment (HCE) has been created for these software stacks, facilitating the transparent access to the HP-SEE infrastructure based also in the European standards set by PRACE. This common environment is using the Module framework. The HCE modules are used in the software monitoring scripts too. The monitoring architecture is based on Nagios system, which is free available open source software.

Finally a Bioinformatics portal has been deployed in the HP-SEE infrastructure. The portal is mainly used by the Life Science users providing transparent access to the infrastructure being available to regional scientists. The usage of this portal is described at the end of this document.

2. Analysis of applications scalability

Table 1 summarizes the scalability methodologies, which are used by the applications that took part in the WP8 scalability studies. 14 applications in total have been analysed by the WP.

Apppplication	Parallelization technologies				Compilers	Technologies	Hardware platforms	Comments
	OpenMP	POSIX threads	MPI	GPU				
HC-MD-QM-CS	*		*		GCC	Intel XEON	HPCG cluster, FINKI SC, local cluster	Different parallelization technologies
GENETATOMICS			*		GCC	Intel XEON	HPGCG cluster	Different type of MPI has been used; hyper threading should be used when available
GIM	*		*		GCC	Intel XEON, ccNUMA architecture	HPCG cluster, Pecs SC	Different parallelization technologies
MSBP			*		GCC (4.4.x, 4.7.X), Open64	AMD Opteron	NCIT cluster, Szeged SC	Different compilers have been tested
SET			*	*	GCC, Intel compiler, IBM XL compiler	PowerPC based, Intel Xeon, M2090 GPU card, GTX 295-based GPU	HPCG Cluster, BlueGene BG	Different version of MPIs, Intel compiler provides the best result, 30% improvement when hyperthreading is turned on, M2090 card give better result then GTX295
NUQG	*		*		GCC	Intel XEON, ccNUMA architecture	HPCG cluster, Pecs SC, PARADOX	PARADOX, HPCG cluster (MPI parallelization used), Pecs SC (OpenMP

								parallelization)
SFHG	*		*		Open64, PGI, Intel, GCC (4.3, 4.1, 4.5, 4.6)	Intel XEON, AMD Opteron, ccNUMA architecture	Pecs SC, Szeged SC, PARADOX, BA-01- ETFBL	Different compiler flags have been tested
CFDOF			*		GCC	Intel XEON, AMD Opteron	PARADOX, BA-01- ETFBL	Intel and AMD platform has been tested too
DNAMA	*	*	*		Intel compiler, GCC	Intel XEON, ccNUMA architecture	HPCG cluster, Pecs, Debrecen SC	Intel compiler gave better result than GCC for larger number of cores
AMR_PAR	*				Visual Studio, Intel compiler	Intel XEON, ccNUMA	IMI ASM RENAM cluster, Pecs SC	Porting from Windows to Linux experiences
DeepAligner			*		Intel compiler, GCC, Open64	AMD Opteron, ccNUMA architecture, Intel XEON, Power PC based	BlueGene/ P, Szeged SC, Budapest SC, Debrecen SC, Debrecen SC	Several MPI implementa tions (SGI- MPT, OpenMPI)
DiseaseGeneMap per								
FMD-PA	*		*		GCC, IBM XL compiler	PowerPC CPU, Intel Xeon, M2090 GPU card, GTX 295- based GPU	HPCG Cluster, BlueGene BG	Scales on both architectures
CompChem			*		GCC	Intel XEON	PARADOX, HPCG	NAMD testing with several configurations

Table 1 – Methodologies used by the applications

The table depicts that several of those actions have been implemented by the applications while no common action is suitable for all applications. A more detailed description of the implemented actions and detailed results is being given in the following section.

2.1. HC-MD-QM-CS

2.1.1. Summary

Code author(s): Ljupčo Pejov, Anastas Mishev	
Application areas: Computational Chemistry	
Language: FORTRAN, C/C++	Estimated lines of code: 10000
URL: http://wiki.hp-see.eu/index.php/HC-MD-QM-CS	

2.1.2. Implemented scalability actions

- The computational methodology that we develop and use is a hybrid one, consisting of several steps, each of which demands computational resources to a various extent. It is therefore expected that each step would scale rather differently with the number of processors/cores. The overall hybrid computational procedure could not be fully automated, first of all due to its complexity and the need to check certain results manually. The best possibility to judge on its overall scalability is, therefore, to test the scalability of each of the component phases. Our focus in this application was, therefore, to achieve the optimal output from the available hardware architectures by optimizing the overall complex hybrid computational approach. To achieve an overall good scalability is possible only if one avoids the main bottlenecks in the overall procedure, which appear to be the statistical physics simulations as well as the subsequent supermolecular quantum mechanical calculations of the electronic structure.
- Both MPI and OpenMP paradigms were implemented to achieve the parallelization (depending mostly on the codes for MC/MD simulations in the overall algorithm), testing several particular implementations thereof. The scalability results are, however, affected to a minor degree by a particular choice of MPI paradigm/version. Of course, this finding implies easy porting to various different cluster architectures.
- Benchmark calculations were carried out with different codes enabling statistical physics simulations of the condensed phases that we have studied. Careful comparisons have been carried out between different codes and different segment calculations thereof. For statistical physics simulations, we have used CPMD, NAMD, NWCHEM, and other codes, while for supermolecular quantum chemical electronic structure calculations we have mostly used Gaussian, Gamess and ORCA. For subsequent nuclear quantum mechanical vibrational calculations, as well as for analysis of the results from the statistical physics computational phase, we have used our own (home-made) codes, mostly written in FORTRAN.
- Though testing of compilation with various compilers was also done, due to certain specificities in the codes used for statistical physics simulations, we could not perform direct comparisons between scalability and performance thereof for each particular compiler. We are also currently testing the performances on GPU systems and implementing the overall methodology on these new computer architectures. In certain case, however, wherever that was possible, we performed tests to choose the best possible compiler and linker options, in particular the locally optimized versions of specific libraries (such as variants of BLAS, LAPACK, FFT etc.). On one of our local clusters, based on Intel XEON

processors, we are currently also testing the performances of the parallel codes with and without hyper threading.

2.1.3. Benchmark dataset

One of the bottlenecks in the overall computational methodology that we are developing is its first phase, which involves classical or quantum molecular dynamics or Monte-Carlo simulation of the system in question. In this report, we will focus on the scalability of some quantum molecular dynamics approaches. We will discuss first the scalability of the Car-Parrinello molecular dynamics (CPMD). Each CPMD simulation consists of two phases: wave function optimization and molecular dynamics simulation. As the optimization runs involve an iterative procedure that needs to converge, the number of iterations required to achieve final convergence being strongly dependent on the particular architecture, compilers and compilation parameters etc., this phase is strongly platform – dependent and not so suitable for benchmarking. Though in the future we aim to make careful comparisons of the optimization results as well, the main accent in the present report will be put on the molecular dynamics phase.

Computations involved in the phase of MD trajectory analyses, in the sense of checking the mutual statistic dependence of the snapshots are generally not much time- and resource-consuming, and therefore not much would be gained by their parallelization. Of course, other trajectory analyses, such as, e.g. analyses of the hydrogen bonding networks within a molecular liquid could benefit from parallelization.

The next “bottleneck”-phase of the methodology involves quantum mechanical computations either of the points on a grid of points to obtain the vibrational potential energy curve or surface, or single-point computations of other type (such as, e.g. time-dependent HF or DFT calculations of electronic spectra). We have also paid particular attention to this phase in the course of scalability studies.

Further computation of e.g. the vibrational frequencies (*i.e.* the energies of vibrational transitions) involves either standard diagonalization procedures or fourier-transform – based techniques. In general, in the case of one-dimensional problems, diagonalization and FFT computations are quick, and do not benefit much from parallelization. However, in the case of multi-dimensional vibrational problems, other techniques could be implemented, for which the scalability is significant.

2.1.4. Hardware platforms

HPCG cluster and FINKI SC

The following distinct hardware platforms were used:

- the HPCG cluster with Intel Xeon X5560 CPU @2.8 GHz,
- the FINKI SC with Intel Xeon L5640 CPU @2.26GHz
- our local cluster at the Institute of Chemistry (with Intel XEON 12-core processors and fiber-channel interconnection)

2.1.5. Execution times

To demonstrate the scalability of the approach, in Figure 1 variation of the wall-clock computational time required to carry out an MD simulation of a modest-size water cluster (consisting of 32 water molecules) is plotted against the number of computing

processes (processors/cores). Figure 2 shows the same data, where both axes are logarithmic (log-log plot). The parallelization has been achieved by the MPI paradigm.

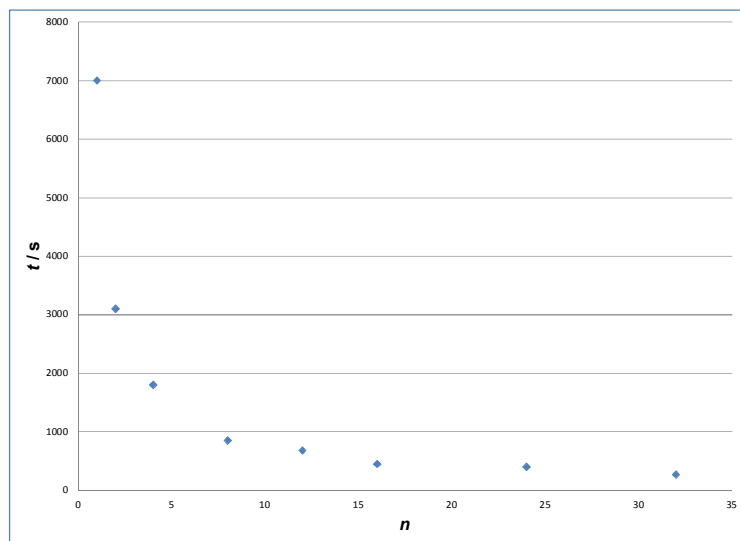


Figure 1 - HC-MD-QM-CS, demonstrate the scalability

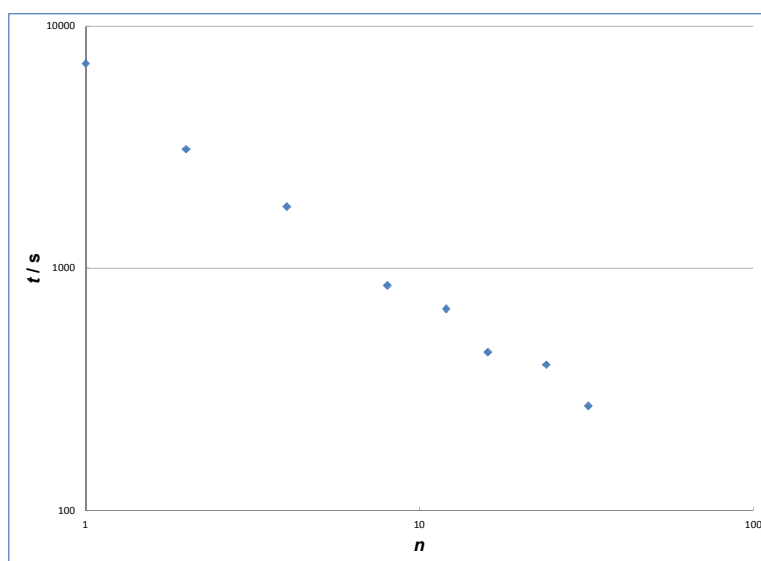


Figure 2 - HC-MD-QM-CS, demonstrate the scalability, (log-log plot)

The effectiveness of the quantum mechanical electronic structure calculation phase heavily depends on the parallelization. To illustrate this point, as a typical example, we consider single-point energy calculations by Gaussian, required to be performed in order to obtain the vibrational potential energy curve or surface. Figure 3 and Figure 4 show the computational time required to carry out single-point Gaussian calculations at HF level of theory, for a system containing 10 non-hydrogen atoms, using a modest-size basis set, plotted vs. the number of cores, in linear and logarithmic scales.

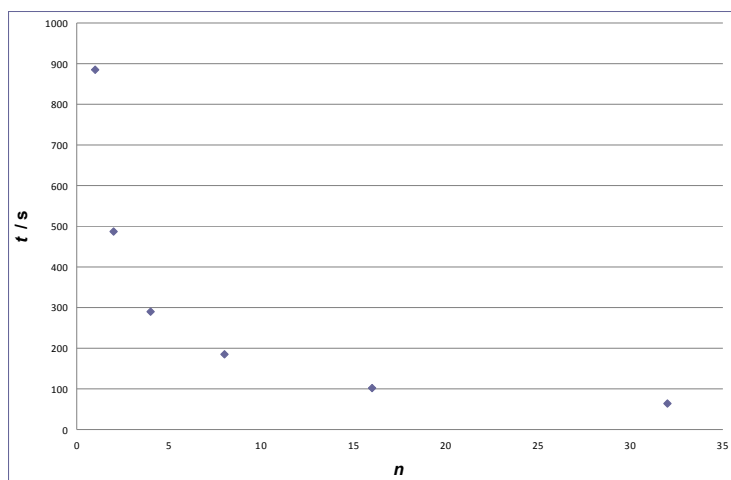


Figure 3 – Required computational time 1, HC-MD-QM-CS

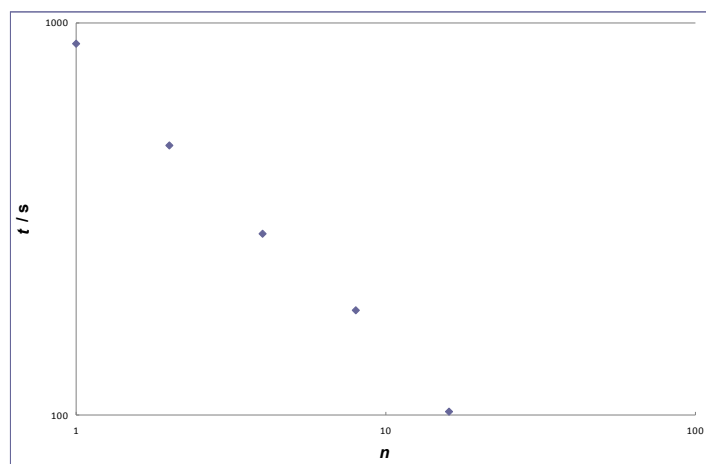


Figure 4 – Required computational time 2, HC-MD-QM-CS

2.1.6. Memory Usage

The maximum memory usage of a single computational thread is heavily dependent on the complexity and size of the studied system. It could vary from a relatively small value, of the order of several hundreds of MBs, up to several GBs.

2.1.7. Profiling

The usefulness of profiling analysis is that it can shed some light on the computationally most demanding sub-phases of the overall computational algorithm. In our study, we have found out that, during e.g. a typical CPMD simulation, most of the computational time is spent in Fourier transformation-related computations, as well as in many body perturbation theory calculations in the phase of electronic structure computations.

2.1.8. Communication

The overall communication time was shown to be somewhat less than 10% of the overall execution time.

2.1.9. I/O

Though the user-defined inputs for the computations in all phases are rather small, as they contain the basic parameters for particular computation, the statistical physics as well as quantum mechanical electronic structure codes contain their internal databases concerning the atomic parameters, basis set data etc. The size of the output is heavily dependent on the frequency with which the data from MC/MD trajectories are saved during the computations. Numerous scratch files are also heavily generated, which enable restart of the computations (if required), and also they allow intermediary checks of the computations.

2.1.10. CPU and cache

Most of our computations, when using the CPU-based version fit in the cache for the Intel-based version. For other computational architectures, we still haven't carried out such testing (e.g. PowerPC processors of the Blue Gene/P and GPUs). The overall significance of these operations is, however, expected to be small (of the order of several percents).

2.1.11. Analysis

The summary of the results from our testing is the following:

- There are two bottlenecks in our complex hybrid MC/MD-QM methodology, which heavily depend on parallelization: the statistical physics simulations (especially the QM MD) and the QM electronic structure calculations. The overall scalability of the two phases, however, seems to be rather good.
- While the computations involved in the phase of MD trajectory analyses are generally not much time- and resource-consuming, other trajectory analyses, such as, e.g. analyses of the hydrogen bonding networks within a molecular liquid could benefit from parallelization.
- Further computation of e.g. the vibrational frequencies (*i.e.* the energies of vibrational transitions) involves either standard diagonalization procedures or fourier-transform – based techniques. While in the case of one-dimensional problems, diagonalization and FFT computations are quick, and do not benefit much from parallelization, in the case of multi-dimensional vibrational problems, other techniques could be implemented, for which the scalability is significant.

2.2. GENETATOMICS

2.2.1. Summary

Code author(s): Jane Jovanovski, Boro Jakimovski and Dragan Jakimovski	
Application areas: Computational Physics	
Language: Fortran	Estimated lines of code: 2000
URL: http://wiki.hp-see.eu/index.php/GENETATOMICS	

2.2.2. Parallelization

We use genetic algorithms and genetic programming for developing algorithm for solving order differential equation (single or system of ODE). One of the main characteristics of genetic algorithms and genetic programming as techniques for implementation of evolutionary paradigms is their exceptional ability to be parallelized. This comes from the fact that the individuals can be evaluated in parallel as their performance rarely, if ever, affects that of other individuals. There are numerous ways for parallelization of genetic algorithms, but here we will consider the following two techniques:

- Island GA: The population is divided on several subpopulations - islands, each subpopulation is a population on its own and is developed on a separate processor. After a certain number of generations, all subpopulations are gathered together into a single population to get mixed, after which they are resent to the processors.
- Parallelization of the fitness function: The most used operation in the evolutionary algorithms is an evaluation of the fitness value of each of the chromosomes. The fitness value is evaluated during selection, after crossover, after mutation. Therefore, this operation takes most of the processor time. There are different techniques for parallelization of the fitness function depending on its shape.

We used the Message Passing Interface (MPI) standard for building the parallel evolutionary algorithm. We tested efficient of parallelism with different type of MPI but we found that version of MPI is not related with scalability results. For different version of MPI we got similar results. The current population consists of chromosomes which contain as many stacks as there are equations in the system that is being solved, and each of the stacks that denotes a postfix representation of the function is represented by a stack with a variable size. For example if we solve single ODE we have a chromosome with one stack, but if we solve system of three ODEs we have chromosome with 3 stacks. As a result of the dynamic size of the arrays and due to the fact that MPI cannot deal with arrays with variable size, it is impossible to divide the population on smaller islands to be sent to the corresponding processors. This reason led us to the implementation of the second technique for parallelization of the fitness function.

2.2.3. New way to measure relative scalability of algorithm

The innovative approach was adopted for measuring the speed-up of parallel implementation of the algorithm due to its stochastic nature. With different runs of the code one gets different functions with different evaluation times. To smooth out this

inherent stochasticity of the genetic algorithm we modified the expression for speed-up relative nonparallel case to

$$s = \frac{t_1 \sum_{i=1}^k \langle r_{i,1} \rangle}{t_j \sum_{i=1}^k \langle r_{i,j} \rangle}$$

Equation 1 - Measure relative scalability

Where t_j equals time to develop the j th generation with j processors, t_1 equals time to develop 1 th generation with one processor, k equals the number of equations in the system, $\langle r_{i,1} \rangle$ equals the mean value of the stack for whole population for 1 th generation as represents the 1 th equation of the system when the algorithm execution falls on one processor, $\langle r_{i,j} \rangle$ is the mean value of the stack, for whole population for j th generation, as represents the i th equation of the system when the algorithm runs on j processors.

2.2.4. Benchmark dataset

For measuring scalability of algorithm we chose to solve one ODE. We choose next equation:

$$\frac{dy}{dx} = (2x - 1)y - 0, y(1) = 2, x \in [1,3]$$

Equation 2 - Measuring scalability of algorithm

We choose step of 0,002, so we must develop fitness function in 1001 points. Our algorithm is based on two genetic algorithms, the parameter of algorithm are follows:

1. Main genetic algorithm
 - a. Number of generations: 12
 - b. Population size: 50
 - c. Mutation rate: 5%
 - d. Cross over rate: 85%
2. Sub genetic algorithm:
 - a. Number of generations: 10
 - b. Population size: 75
 - c. Mutation rate: 5%
 - d. Cross over rate: 50%

2.2.5. Hardware platforms

The application was tested only on HPGCC cluster with Intel Xeon L5640 CPU @2.26 Ghz

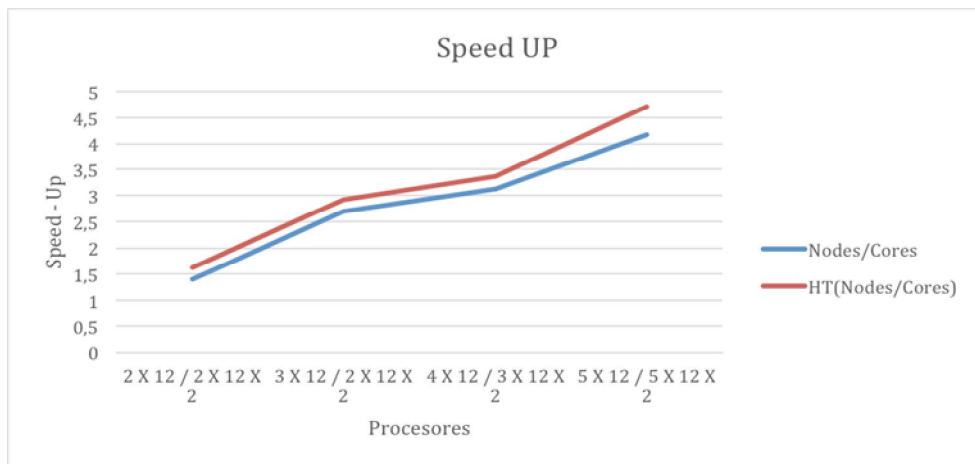
2.2.6. Execution times

For measuring speed – up of algorithm we measure time needed for develop 15th generation. After that we use our equation for calculate speed up of algorithm. The results is shown in next tables.

Nodes/Cores	t	Average size of stack (I)	t/I	Speed - up	Parallel Efficiency
1 X 12	0,5277	32,9200000	0,016028391		
2 X 12	0,5708	50,2000000	0,011370745	1,409616609	0,704808305
3 X 12	0,2278	38,2200000	0,005960888	2,688926793	0,896308931
4 X 12	0,1603	31,2400000	0,005131462	3,123552625	0,780888156
5 X 12	0,1360	35,4400000	0,003836756	4,17758944	0,835517888

Table 2 - Calculate speed up of algorithm

HT(Nodes/Cores)	t	Average size of stack (I)	t/I	Speed - up	Parallel Efficiency
1 X 12 X 2	0,3021	22,7000000	0,013309498		
2 X 12 X 2	0,4289	52,5200000	0,008166195	1,629828659	0,814914329
3 X 12 X 2	0,2094	45,7600000	0,004576607	2,908158372	0,969386124
4 X 12 X 2	0,1831	46,3600000	0,003950551	3,36902307	0,842255767
5 X 12 X 2	0,1609	57,0000000	0,00282246	4,715566694	0,943113339

Table 3 - Calculate speed up of algorithm, HT**Figure 5 - Calculate speed up of algorithm**

2.2.7. Memory Usage

Maximum memory usage per node is 2.5GB. Active memory is only 30MB per process, but no matter the number of processes per node are started, total memory usage for the processes on one node is 2.5GB.

2.2.8. Profiling

Profiling was performed on the sequential algorithm in order to detect the initial CPU utilization code. The MPI application profiling will be done in the following months.

2.2.9. Communication

The MPI communication totals to 200MB/s over QDR Infiniband for 48 processes.

2.2.10. I/O

The input and output of algorithm is small. Input contains: system of ODE or single ODE, initial condition of ODE, interval in which we solve the equation, parameters related with two genetic algorithm. The output is not depend of input parameter. The output is txt file with the best results for each generation. Also for each generation we measure same specific characteristic of population, like: time needed for developed generation, average size of stack size, constant for the best solution etc., therefore the output file is less than 1MB.

2.2.11. CPU and cache

No CPU level analysis was done.

2.2.12. Analysis

From our testing we concluded that hyper threading should be used when available. For future work it remains to make MPI profiling and if needed to find an efficient strategy of reordering of the computations in order to achieve more stable computational times.

2.3. GIM

2.3.1. Summary

Code author(s): Neki Frasheri, Betim Cico	
Application areas: Computational Physics	
Language: C (gcc)	Estimated lines of code: 3000
URL: http://wiki.hp-see.eu/index.php/GIM	

2.3.2. Implemented scalability actions

Actions:

- Usage of different parallel paradigm: both OpenMP and MPI are used
- Profiling: runtime measured using /usr/bin/time to execute the program
- Usage of parallel libraries: no use of external parallel libraries
- Usage of compilers: GCC with standard flags for OpenMP and MPI

2.3.3. Benchmark dataset

The model consists of a 3D cuboid underground geosection (with depth half of horizontal extension) represented with a 3D array of nodes.

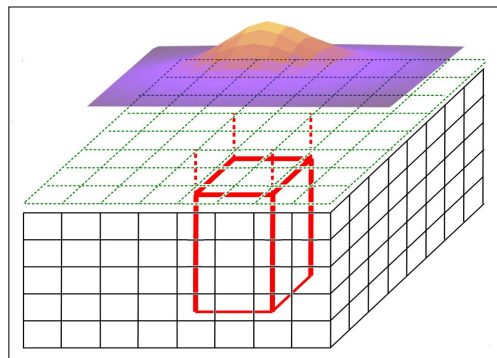


Figure 6 - 3D cuboid underground geosection

The ground surface over the geosection where the gravity anomaly is surveyed is represented by a 2D array of points.

The volume of data depends on the size of one edge of the model:

Points: surface points of 2D array = size^2

Nodes: underground nodes of 3D array = $(\text{size}^3) / 2$

The experimented model size was with linear size varying 11 – 21 – 41 – 81 – 161 nodes and respective step between nodes 400m – 200m – 100m – 50m – 25m.

Number of array nodes defines the volume of calculations for each iteration as in the table:

Model size			2D array points	3D array nodes	Elementary calculations
Ny	Nz				
11	11	6	121	726	87,846
21	21	11	441	4,851	2,139,291
41	41	21	1,681	35,301	59,340,981
81	81	41	6,561	269,001	1,764,915,561
161	161	81	25,921	2,099,601	54,423,757,521

Table 4 - The volume of calculations for each iteration

The case 161x161x81 was not experimented because the huge runtime expected at the range of 1 year.

2.3.4. Hardware platforms

Both OpenMP and MPI solutions were tested in two platforms:

Platform 1: HPCG-IICT,

- OpenMP scaled up to 16 cores due to hardware limitations.
- MPI scaled between 1 up to 256 cores.

Platform 2: Pecs Supercomputing Centre, Sun Grid Engine

- OpenMP scaled between 1 up to 1024 cores.
- MPI scaled between 1 up to 256 cores.

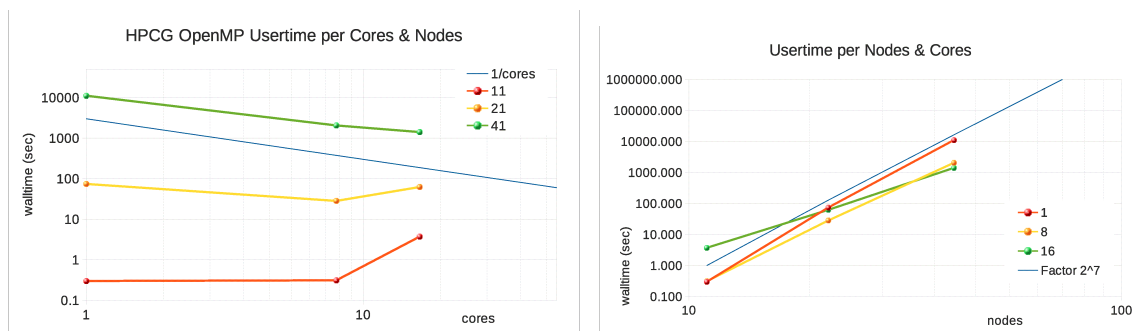
Number of cores varied with a factor of 2:

1 – 2 – 4 – 8 – 16 – 32 – 64 – 128 – 256 – 512 – 1024

2.3.5. Execution times

Tests with OpenMP in HPCG are with 1, 8 and 16 cores:

Figure 7 - Tests with OpenMP in HPCG



Full scale tests with OpenMP were done in Pecs SC with up to 1024 cores:

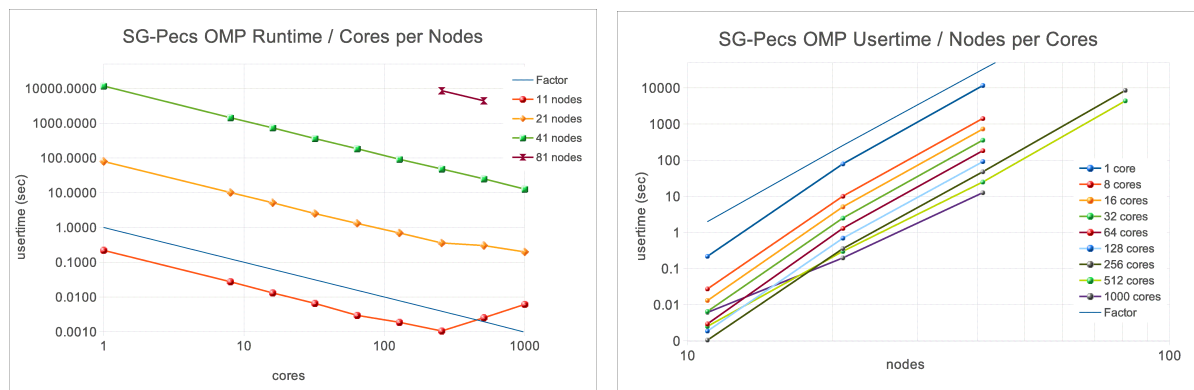


Figure 8 - Tests with OpenMP in Pecs SC

Tests with MPI were carried out in HPCG with up to 256 cores:

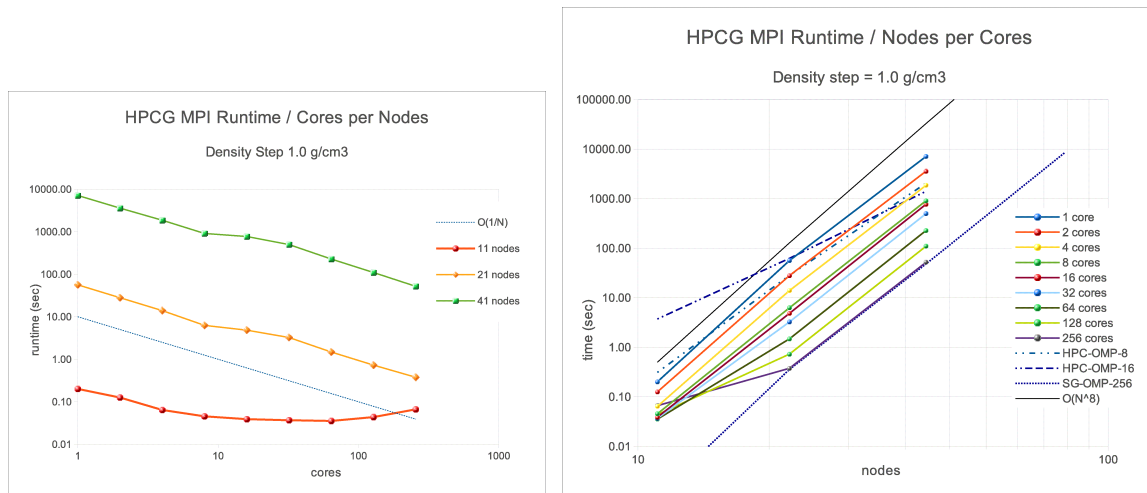
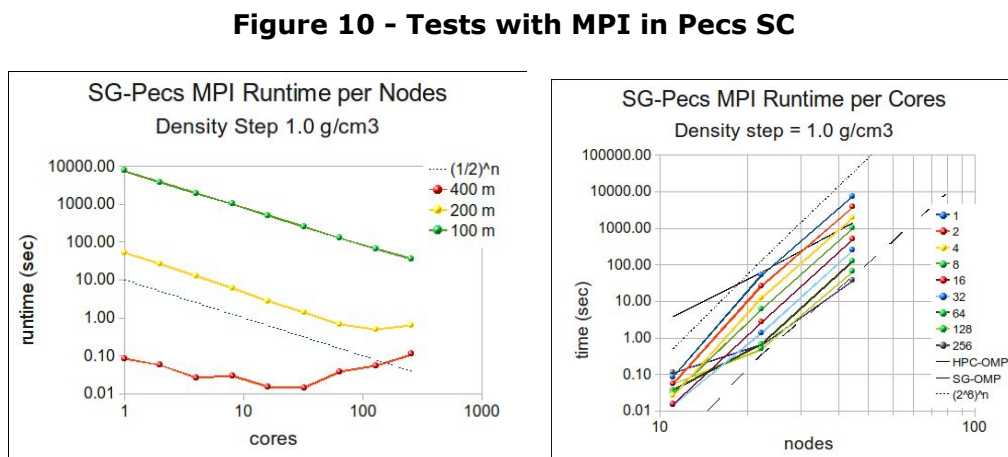


Figure 9 - Tests with MPI in HPCG

Tests with MPI were carried out in Pecs SC with up to 256 cores:



Evaluated runtime for the model size 161x161x81 (spatial discretization step of 25 meters) is one year.

2.3.6. Memory Usage

The size of the model is defined by the 2D array of ground points and the 3D array of geosection nodes. The application uses 6 arrays 2D and 1 array 3D. For considered model sizes the use of main memory in Bytes is given in the table:

Model size			6 x 2D array elements	3D array elements	Used Memory (Bytes)
Ny	Nz				
11	11	6	726	726	11,616
21	21	11	2,646	4,851	59,976
41	41	21	10,086	35,301	363,096
81	81	41	39,366	269,001	2,466,936
161	161	81	155,526	2,099,601	18,041,016

Table 5 – Memory usage of GIM

The model 161x161x81 was not experimented because of the huge runtime expected (approximated one year)

The same range of storage capacity is used in hard-disk units for the results file.

The evaluation of memory requirements for the same geosection model with spatial discretization step of 1 meter is 260 GB.

2.3.7. Profiling

Statistics related with the time statistics were obtained in two ways:

- a) using the time function within the program
 - a) for MPI: `MPI_Wtime()`
 - b) for OpenMP: `omp_get_wtime()`
- b) running it through the `/usr/bin/time` command:
 - a) `(/usr/bin/time gmj4-v600omp ...) ...`
 - b) `(/usr/bin/time gmj4-v600mpi ...) ...`

The time command was used for statistics:

- `summary_per_processes` user time
- elapsed time
- `summary_per_cores` CPU%

2.3.8. Communication

Network communication is not measured. Experimented models had memory sizes up to 2.5 MB, requesting a negligible transfer time compared with the runtime.

2.3.9. I/O

Number of I/O operations with the external storage was not measured. The software uses only central memory. Communication with the external storage are only at the beginning (reading of input data) and at the end of execution (writing results data).

2.3.10. CPU and cache

Summary percentage of exploitation of cores was evaluated using the `/usr/bin/time` command (see Profiling section). Usage of the cache was not measured.

2.3.11. Derived metrics

The scalability of algorithm was analyzed using the average net runtime per process / thread.

2.3.12. Analysis

Algorithm works through scanning in each iteration of the 3D array of geosection nodes to define the optimal one. Parallelism is achieved splitting the geosection in fragments and scanning each of them in a separate core.

Runtime resulted comparable with the complexity of the algorithm. Considering N the mean number of nodes in one edge of the 3D array representing the geomodel (the linear size), the complexity of algorithm gives magnitude orders for the volume of calculations (proportional with the runtime):

- magnitude of runtime per cores: $O(1/\text{cores})$
- magnitude of runtime per linear_size per iteration: $O(N^5)$
- magnitude of runtime per linear_size: $O(N^8)$

Experiments were done for small and medium model discretization sizes. Scalability degenerates for small models run in many parallel nodes. Maximal runtime obtained was at the range of one day using 1024 cores for models with spatial discretization step of 50m, which is large for many geophysical engineering problems. Prognosis for models with spatial discretization step of 25m was at the range of one year in 1024 cores – practically impossible, while for engineering works smaller spatial steps to one meter may be required.

Modification of the algorithm for the reduction of the volume of calculations leading to a reduction of the runtime is considered. The request for many cores makes difficult the widespread of similar algorithms for engineering works, because of difficulties to access easily traditional parallel systems anytime, and the use of GPU for parallel processing is considered as a way to bring necessary parallel capacities in desktop platforms.

2.4. MSBP

2.4.1. Summary

Code author(s): Jumber Kereselidze, George Mikuchadze	
Application areas: Life Science	
Language: C/C++	Estimated lines of code: 337000
URL: http://wiki.hp-see.eu/index.php/MSBP	

2.4.2. Implemented scalability actions

- As a first step application was ported on SEE-GRID infrastructure using MPICH1.
- At the beginning we were using GNU C/C++ compilers version 4.4.x which was not optimized for current processors and hence run time was very large.
- At a second step the parallelization has been performed with OpenMPI implementation of MPI (v. 1.6.x) installed on the hardware platforms that were available to us.
- We tested various compilers and concluded that the within free compiler suites the GNU Compiler v4.7.x and Open64 Compiler Suite with ACML provided the best results for our AMD Opteron clusters.
- We used the both C compilers through mpicc wrapper with default flags which were optimal for the working nodes

2.4.3. Benchmark dataset

For the benchmarking we chose a particular molecular structure with 53 atoms and 216 electrons for stable output.

2.4.4. Hardware platforms

NCIT-Cluster and Szeged supercomputer.

Two distinct hardware platforms were used:

- NCIT cluster with AMD Opteron 2435 CPU @2.6 Ghz in Romania;
- Szeged supercomputer with AMD Opteron 6174 (12-core Magny-Cours) CPU @2.2 Ghz in Hungary.

2.4.5. Execution times

NCIT-Cluster

working nodes: 4 X 6core (24 cores) AMD Opteron 2435 (2.6Ghz)

mpi: openmpi v1.6

C compiler: gcc v4.7.0

# cores	Time (HH:MI:SS)	# nodes
16 cores	06:25:26	2

24 cores	05:33:34	2
32 cores	08:46:45	2

Szeged SC

working nodes: 4 X 12core (48 cores) AMD Opteron 6174 (2.2Ghz)

mpi: openmpi v1.6

C compiler: Open64 Compiler Suite: Version 4.2.4 with AMD Core Math Library (ACML)

# cores	Time (HH:MI:SS)	# nodes
08 cores	10:17:25	2
16 cores	07:54:30	6
24 cores	07:01:17	2
32 cores	08:45:08	7
64 cores	09:03:20	13

2.4.6. Memory Usage

The maximum memory usage of MPI implementation was quite small 100Mb per cpu/core.

2.4.7. Communication

The communication for this application is critical (Changing of the Infiniband QDR with 1Gb Ethernet results to 2-3 times increasing of execution time). Therefore we used only clusters with the fast Infiniband network for the internal high-performance communication.

2.4.8. I/O

The input for the application is small less than tens of MBs and output file size is in the same size.

2.4.9. CPU and cache

We believe that most of the computations fit in the cache for the AMD Opteron CPUs.

2.4.10. Analysis

From our testing we concluded that most effective are CPUs with high brutal fpv performance. Inter-process communication intensity reduces scalability efficiency and the most effective are 16-24 CPU/cores for MPI parallelism.

2.5. SET

2.5.1. Summary

Code author(s): Emanouil Atanassov	
Application areas: Computational Physics	
Language: C/C++	Estimated lines of code: 6000
URL: http://wiki.hp-see.eu/index.php/SET	

2.5.2. Implemented scalability actions

Actions:

- Our focus in this application was to achieve the optimal output from the hardware platforms that were available to us. Achieving good scalability depends mostly on avoiding bottlenecks and using good parallel pseudorandom number generators and generators for low-discrepancy sequences. Because of the high requirements for computing time we took several actions in order to achieve the optimal output.
- The parallelization has been performed with MPI. Different version of MPI were tested and we found that the particular choice of MPI does not change much the scalability results. This was fortunate outcome as it allowed porting to the Blue Gene/P architecture without substantial changes.
- Once we ensured that the MPI parallelization model we implemented achieves good parallel efficiency, we concentrated on achieving the best possible results from using single CPU core.
- We performed profiling and benchmarking, also tested different generators and compared different pseudo-random number generators and low-discrepancy sequences.
- We tested various compilers and we concluded that the Intel compiler currently provides the best results for the CPU version running at our Intel Xeon cluster. For the IBM Blue Gene/P architecture the obvious choice was the IBM XL compiler suite since it has advantage versus the GNU Compiler Collection in that it supports the double-hammer mode of the CPUs, achieving twice the floating point calculation speeds. For the GPU-based version that we developed recently we rely on the C++ compiler supplied by NVIDIA.
- For all the choosen compilers we performed tests to choose the best possible compiler and linker options. For the Intel-based cluster one important source of ideas for the options was the website of the SPEC tests, where one can see what options were used for each particular sub-test of the SPEC suite. From there we also took the idea to perform two-pass compilation, where the results from profiling on the first pass were fed to the second pass of the compilation to optimise further.
- For the HPCG cluster we also measured the performance of the parallel code with and without hyperthreading. It is well known that hyperthreading does not always improve the overall speed of calculations, because the floating point units of the processor are shared between the threads and thus if the code is highly intensive in such computations, there is no gain to be made from

hyperthreading. Our experience with other application of the HP-SEE project yields such examples. But for the SET application we found about 30% improvement when hyperthreading is turned on, which should be considered a good results and also shows that our overall code is efficient in the sense that most of it is now floating point computations, unlike some earlier version where the gain from hyperthreading was larger.

- For the NVIDIA-based version we found that we have much better performance using the newer M2090 cards versus the old GTX295, which was to be expected because the integer performance of the GTX 295 is comparable to that of M2090, but the floating performance of the GTX is many times smaller.

2.5.3. Benchmark dataset

For the benchmarking we fixed a particular division of the domain into

800 by 260 points, electric field of 15 and 180 femto-seconds evolution time. The computational time in such case becomes proportional to the number of Markov Chain Monte Carlo trajectories. In most tests we used 1 billion (10^9) trajectories, but for some tests we decreased that in order to shorten the overall testing time.

2.5.4. Hardware platforms

HPCG cluster and Blue Gene/P supercomputer.

Four distinct hardware platforms were used:

- the HPCG cluster with Intel Xeon X5560 CPU @2.8 Ghz,
- Blue Gene/P with PowerPC CPUs,
- our GTX 295-based GPU cluster (with processors Intel Core i7 920)
- our new M2090-based resource with processors Intel Xeon X5650.

2.5.5. Execution times

Table 6 – Execution times of SET

Table I
THE CPU TIME (SECONDS) FOR ALL 800×260 POINTS, THE SPEED-UP,
AND THE PARALLEL EFFICIENCY.

Blades/Cores	CPU Time (s)	Speed-up	Parallel Efficiency
1 x 8 = 8	202300	-	-
4 x 8 = 32	50659	3.9937	0.99834
8 x 8 = 64	25423	7.9574	0.99467
16 x 8 = 128	12735	15.8853	0.99283
Blades/Cores/ Hyperthreading	CPU Time (s)	Speed-up	Parallel Efficiency
1 x 8 x 2 = 16	148602	-	-
4 x 8 x 2 = 64	37660	3.94588	0.98647
8 x 8 x 2 = 128	18957	7.83889	0.97986
16 x 8 x 2 = 256	9552	15.55716	0.97232

Comparison of the execution time and parallel efficiency of SET application are shown on HPCG (Table below) and BlueGene/P (Table above).

Table II
THE CPU TIME (SECONDS) FOR ALL 800×260 POINTS, THE SPEED-UP,
AND THE PARALLEL EFFICIENCY.

Cores	CPU Time (s)	Speed-up	Parallel Efficiency
1024	23498	-	-
2048	12082	1.9449	0.97245
4096	6091	3.8769	0.96923

Table 7 Comparison of the execution time and parallel efficiency of SET

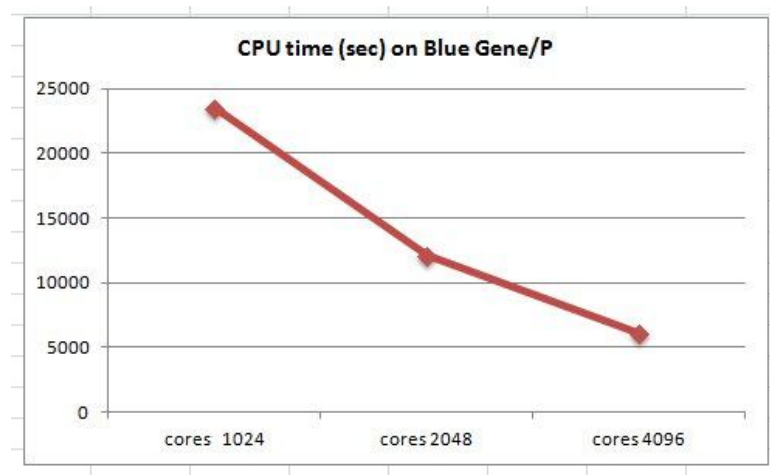


Figure 11 – CPU time on Blue Gene/P

2.5.6. Memory Usage

The maximum memory usage of a single computational thread is relatively small, in the order of 100 MB.

On the GPUs there are several different kinds of memory, some of them rather limited. The available registers and the shared memory are especially problematic, since there is a risk if the available registers are all used some local variables to be spilled to global memory, encountering high latency and other issues. Still we found reasonable performance using 256 GPU threads, which is an acceptable number.

2.5.7. Profiling

Profiling was performed in order to improve the compiler optimisation during the second pass and also in order to understand what kind of issues we may be having in the application. We found as expected that most of the computational time is spent in computing of transcendental function like sin, cos, exp, and also in the generation of pseudorandom numbers. We attempted in the GPU version to replace the regular sin,

cos, etc., with the less-accurate versions that are more efficient, but we found that the gain from that is relatively small and is not worth the loss of accuracy. For the GPU-based version we obtained relatively high percentage of divergence within warps, which means that some logical statements are resolved differently within threads of the same warp and there is substantial loss of performance. So far we have not been able to reorder the computation so as to avoid it.

2.5.8. Communication

The communication for this application is not critical in the sense that the communication takes less than 10% of the execution time.

2.5.9. I/O

The input for the application is small, containing the parameters of the problem at hand. The output is written out at the end of the computation and its size depends on the parameters. For a reasonable size of the domain the output is in the order of several megabytes. More accurate mesh is reasonable only for smaller evolution times and the output size will be proportional to the size of the mesh

2.5.10. CPU and cache

We believe that most of the computations of the CPU-based version fit in the cache for the Intel-based version. For the PowerPC processors of the Blue Gene/P some lookup operations when sampling the random variables use the main memory and thus entice higher latency. For the GPU-based version the situation is similar, since some of the tables are larger than the size of the so-called shared-memory. In both cases, the overall significance of these operations is less than 5%.

2.5.11. Analysis

From our testing we concluded that hyperthreading should be used when available, production Tesla cards have much higher performance than essentially gaming cards like GTX 295, two passes of compilation should be used for the Intel compiler targeting Intel CPUs and that the application is scalable to the maximum number of available cores/threads at our disposal. For future work it remains to find an efficient strategy of reordering of the computations on the GPUs in order to avoid warp divergence. For the CPU-based version we have also developed an MPI meta-program that measures the variation and uses genetic algorithm (from galib library) to optimise the transition density. This step will be added as a pre-processing stage of the program in order to provide some speedup in order of 20% to the overall computations, but to do so we need to find the right balance between this stage and the main computational stage.

2.6. NUQG

2.6.1. Summary

Code author(s): Scientific Computing Laboratory, Institute of Physics Belgrade, Serbia	
Application areas: Computational Physics	
Language: C/C++	Estimated lines of code: 260000
URL: http://wiki.hp-see.eu/index.php/NUQG	

2.6.2. Implemented scalability actions

The NUQG application modules are developed at IPB's PARADOX cluster, and within the framework of HP-SEE project ported to the HPCG and PECS SC. NUQG SPEEDUP module is MPI-parallelized at HPCG cluster, while the NUQG GP module is OpenMP-parallelized at PECS SC. The applications are compiled using the Intel C/C++ compiler that gives much better performance compared to the GCC compiler. In addition, NUQG SPEEDUP module is improved using the algorithm that uses Sobol's set of quasi-random numbers.

2.6.3. Benchmark dataset

Performance of the NUQG SPEEDUP module is assessed for the case of a quantum anharmonic potential, with the level $p = 4$ effective action, and using the target bisection level $s = 8$ (corresponding to the calculation of 255-dimensional integrals). Each quantum amplitude is calculated using the Monte Carlo sample of $N_{MC} = 10^9$ trajectories.

Performance of the NUQG SPEEDUP quasi-MC algorithm is also assessed for the anharmonic potential, with the effective action level $p = 4$, bisection level $s = 5$, and using the Monte Carlo sample of $N_{MC} = 10^8$.

The NUQG GP module performance at single multi-core machine is calculated for a 3D-algorithm for space discretization with the grid mesh $N_x=N_y=1200$ and $N_z=600$.

2.6.4. Hardware platforms

The NUQG application is developed at PARADOX cluster (84 worker nodes with 2 x quad core Xeon E5345 @ 2.33 GHz with 8 GB of RAM per node), and ported to HPCG (36 blades BL 280c with 2 x Intel Xeon CPU X5560 @ 2.8 GHz) and Pesc SC resource centres (SGI 1000 Ultraviolet with SMP ccNUMA architecture - Intel Xeon X7542 6-core processors).

2.6.5. Execution times

The NUQG SPEEDUP module is MPI parallelized. Its execution time and speedup as a function of the number of CPU cores at PARADOX and HPCG clusters is illustrated in Figure 12. While the speedup remains practically perfect on both of these two clusters,

better absolute execution times at HPCG cluster is a consequence of a higher CPU frequency.

Further improvement in the performance of the NUQG SPEEDUP module is achieved by using quasi-random numbers instead of pseudo-random numbers generated by the SPRNG algorithm. The improved algorithm uses Sobol's set of quasi-random numbers for generation of trajectories relevant for calculation of transition amplitudes in the path integral formalism. The ratio of the CPU execution time of the original MC and the improved quasi-MC code in order to achieve the same precision of the amplitude is illustrated in Figure 13. This result is obtained at HPGC cluster.

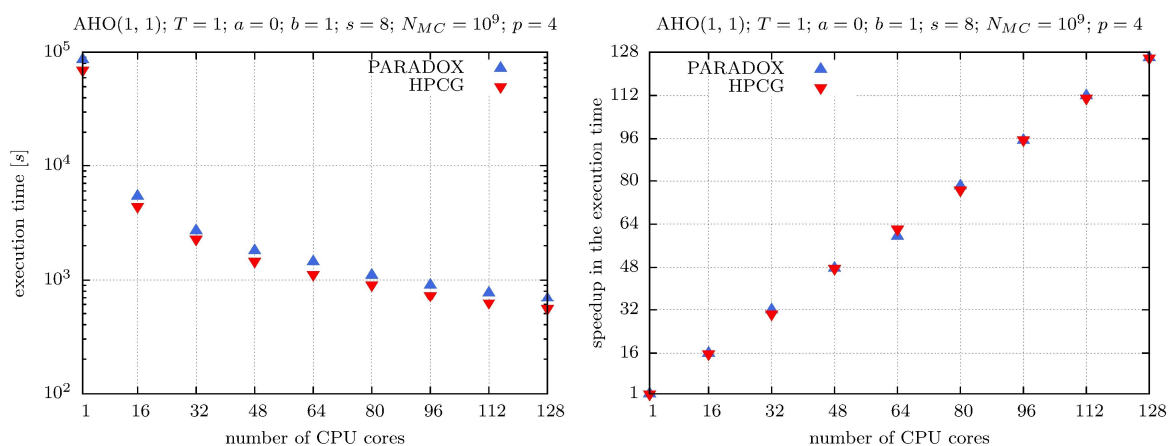


Figure 12 - Execution time and speedup as a function of the number of CPU cores at PARADOX and HPCG clusters

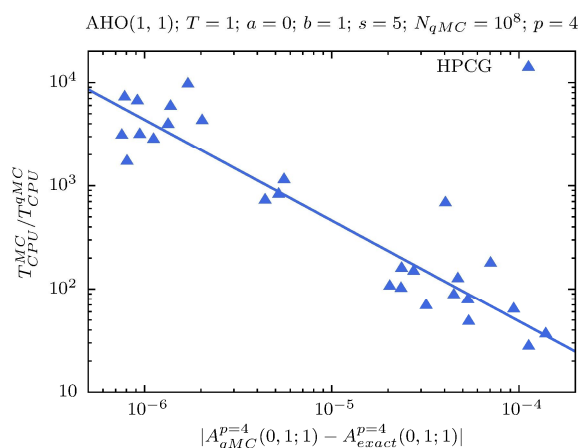


Figure 13 - Ratio of the CPU execution time of the original MC and the improved quasi-MC code as a function of the precision of the amplitude.

The NUQG GP module is OpenMP parallelized. The speedup of the NUQG GP 3D module for real- and imaginary-time propagation at a single large multi-core machine is given in Figure 14.

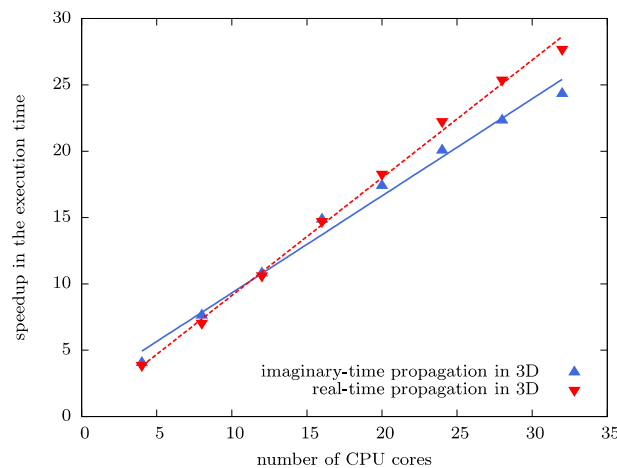


Figure 14 - Speedup in the execution time of the NUQG GP module as a function of the number of CPU cores.

The NUQG GP is ported to PECS shared memory cluster - SGI 1000 Ultraviolet with SMP (ccNUMA) architecture. Since NUQG GP module is highly memory-intensive application, and due to the fact that NUMA architecture memory access time depends on the memory location relative to a processor, initial performance of the code was very fluctuating. The stable performance is achieved when all available memory is utilized. Effective speedup of the application for such cases is given in Figure 15, while Figure 16 shows total occupied memory at the cluster.

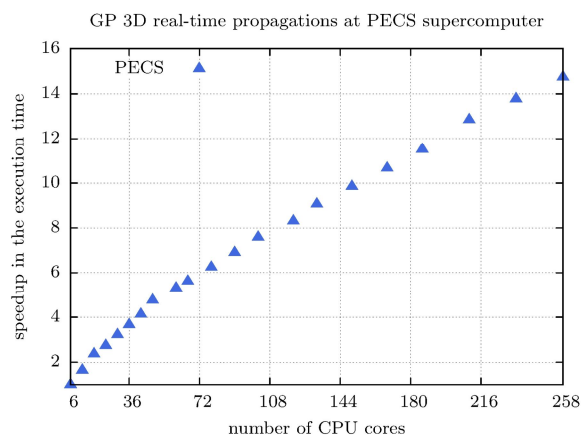


Figure 15 - Effective speedup of the NUQG GP application.

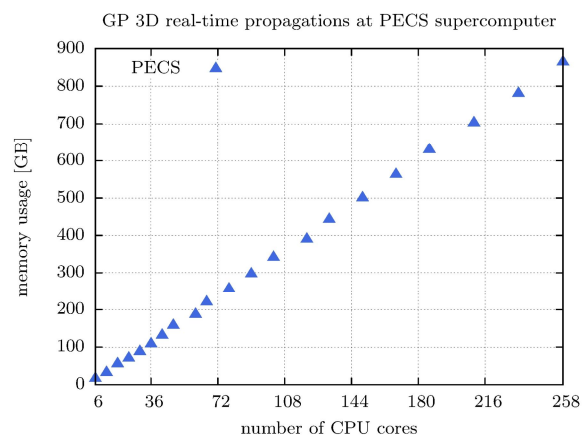


Figure 16 - NUQG GP total memory allocation.

2.6.6. Memory Usage

Memory usage of NUQG SPEEDUP and GP modules depends on the physical characteristics of the physical system of interest. For typical configurations, NUQG SPEEDUP module requires less than 1 GB of RAM, while the execution of NUQG GP 3D module typically requires more than 32 GB of RAM.

In the NUQG SPEEDUP module, maximal accessible level p is limited by the amount of memory required for symbolic derivation of the effective potential. This symbolic calculation is implemented in Mathematica for general 1D, 2D, and 3D potentials, as well as for a general many-body theory in arbitrary number of spatial dimensions. Execution of these codes for level $p = 10$ requires 10-15 MB of RAM in 1D, 60 MB in 2D, 860 MB in 3D, while the execution of the many-body SPEEDUP Mathematica code requires approximately 1.6 GB. Beside of this, minor additional memory (less than 100 MB) is required by the SPEEDUP C code.

Memory consumption of the NUQG GP module depends on a number of spatial dimensions (1D, 2D, 3D), symmetries of the trapping potential of the Bose-Einstein condensate (axially-symmetric, spherically-symmetric), and the type of time propagation studied (real-time, imaginary-time). The following table illustrates typical memory usage of 12 NUQG GP module codes.

NUQG GP code	Memory usage
imagtime1d	> 1 GB
imagtimecir	> 1 GB
imagtimesph	> 1 GB
realtime1d	> 2 GB
realtimecir	> 2 GB
realtimesph	> 2 GB
imagtimeaxial	> 8 GB
imagtime2d	> 8 GB
realtimeaxial	> 16 GB
realtime2d	> 16 GB
imagtime3d	> 16 GB
realtime3d	> 32 GB

Table 8 - Memory usage of NUQG GP codes.

2.6.7. Profiling

Profiling of the NUQG SPEEDUP module shows that most of the computational time is spent in computing of transcendental function (exp), as well as in the calculation of the effective potential. Based on this, using the Intel compiler optimization, calculation of transcendental functions is reused. The NUQG GP module profiling shows that most of the time is spent in time propagation functions and in the normalization. We have managed to optimize the normalization function using several analytical insights, while the time propagation functions will require further cache optimization.

2.6.8. Communication

Communication for the NUQG SPEEDUP module is not critical, but, on the contrary, it becomes one of the main issues for the NUQG GP module. Further optimizations of the GP module will focus on this.

2.6.9. I/O

Input parameters of the NUQG SPEEDUP module are specified from the command line, while the output is a small text file (up to 1KB) that contains numerical values of transition amplitudes for different levels p . The NUQG GP module has a small input file that describes physical system of interest. In addition to this, it is possible to specify the initial state of the condensate. This initial state is provided as a single file, whose size depends of the spatial grid mesh, and can be from a few MBs to several GBs. The output of the NUQG GP module is also configurable, and may require from several GBs to several TBs of disk space.

2.6.10. Analysis

The NQG SPEEDUP module gives excellent performance results. The application is in a mature stage, and further optimization may give very small improvements in the performance. On the other hand, the NUQG GP module invites further optimizations. The further development of this application will be focused on MPI parallelization, which will provide better scalability on shared memory systems, but also at e-Infrastructure with the InfiniBand interconnect. Also, cache optimization within time propagation functions may give better performance.

2.7. SFHG

2.7.1. Summary

Code author(s): Sreten Lekic, Igor Sevo, Mihajlo Savic	
Application areas: Computational Physics	
Language: C/C++, Fortran	Estimated lines of code: 2900
URL: http://wiki.hp-see.eu/index.php/SFHG	

2.7.2. Implemented scalability actions

- We have first refactored the serial Fortran code we had and after that created a new C++ code as existing Fortran code was unsuitable for parallelization. New approach yielded much better performance, even in serial mode, with execution time decrease to approximately 1/15 of original. This was tested only on a small scale problem as anything over the level of 7 was unusably slow on original code.
- Parallelization was attempted with MPI and OpenMP. Due to the nature of the problem being solved, MPI yielded improvements only up to a small number of CPUs and as such was dropped for current time frame. Version of code with OpenMP was fully developed and benchmarked. Hybrid approach is planned for future versions.
- We have tested three different compilers: Open64, Portland Group, Intel and GNU C/C++ compilers. Open64 and PGCC had an incomplete support for OpenMP features required due to either somewhat older version or incomplete standard support and as such provided limited benefit as compared to serial code. ICC compiled with no problems even at -O3 level but the performance was found to be inferior to code produced by GCC (execution time for both serial and parallel version was up to two times longer for all types of walks and tested levels).
- We have tested performance and correctness of different optimization levels. We found significant performance increase when using -O3 with default settings for GCC 4.5 and GCC 4.6. Execution time was 1.8 times shorter with -O3 as compared to no optimization for serial and 1.6-1.8 times shorter for parallel version.
- Not all versions of GCC support all needed OpenMP functions so we have to emulate desired behavior on older versions (GCC 4.3 at Pecs SC and GCC 4.1 at Szeged SC).
- We have tested different approaches to new thread creation as default OpenMP settings were proving to be overly optimistic at higher levels. The issue arises from the huge number of threads created by nested parallel code. If we severely limit the number of threads or remove nesting and expand manually first few levels we achieve very limited speedup which decreases with the level of the problem as there are lingering long running threads that severely decrease parallel performance. On default settings we achieved excellent speedup up to level 8 but had issues with huge number of threads created by OpenMP at level 9. After performing benchmarking with different parameters we settled at limiting the nesting at level 21 and using maximum 2 threads per level. Testing has shown that we are running manageable number of active threads (under 40 per CPU core) with no significant loss of performance.

2.7.3. Benchmark dataset

We have used Sierpinski gaskets of levels 7, 8 and 9 for testing. Level 7 was used for initial very small scale tests while levels 8 and 9 were used for proper performance testing.

2.7.4. Hardware platforms

We have performed performance testing on:

- PARADOX with up to 8 CPU cores
 - Intel(R) Xeon(R) CPU E5345 @ 2.33GHz
 - Small scale test not suitable for true scalability tests
- BA-01-ETFBL with up to 16 CPU cores
 - AMD Opteron 6128 @ 2.00 GHz – total 16 CPU cores
- Szeged SC with up to 48 CPU cores
 - AMD Opteron 6174 @ 2.00 GHz – total 48 CPU cores
- Pecs SC with up to 48 CPU cores – SGI 1000 Ultraviolet
 - Intel Xeon X7542 – 1152 total CPU cores

2.7.5. Execution times

It can be seen from the execution times that for small scale tests with levels of up to 8 there is little benefit of scaling past 16 or at most 24 CPU cores. While the ratio of wall time and CPU time does significantly increase, the number of found walks per second is entering saturation. One also has to take into consideration an architecture and organization of a specific SMP implementation especially in NUMA cases.

For level 9 we can see that there are clear benefits of scaling past 16 CPU cores and this level was chosen for scalability testing with larger number of cores. Walks per second values are not comparable among different levels.

Cores	Walk type	Level	Nest limit	Wall time [s]	CPU time [s]	Wall/CPU time	Eff.	Walks/s
1	A (2)	8	14	4041	4040	1.00	1.00	16388.71
2	A (2)	8	14	2050	4098	2.00	1.00	32305.74
4	A (2)	8	14	1044	4155	3.98	0.99	63435.61
8	A (2)	8	14	571	4354	7.63	0.95	115983.84
12	A (2)	8	14	455	4664	10.25	0.85	145553.35
16	A (2)	8	14	405	5023	12.40	0.78	163522.89
8	A (2)	9	21	39227	300334	7.66	0.96	62026.69
16	A (2)	9	21	21134	336201	15.91	0.99	115128.27

Table 9 - Scalability test results for BA-01-ETFBL with up to 16 CPU cores

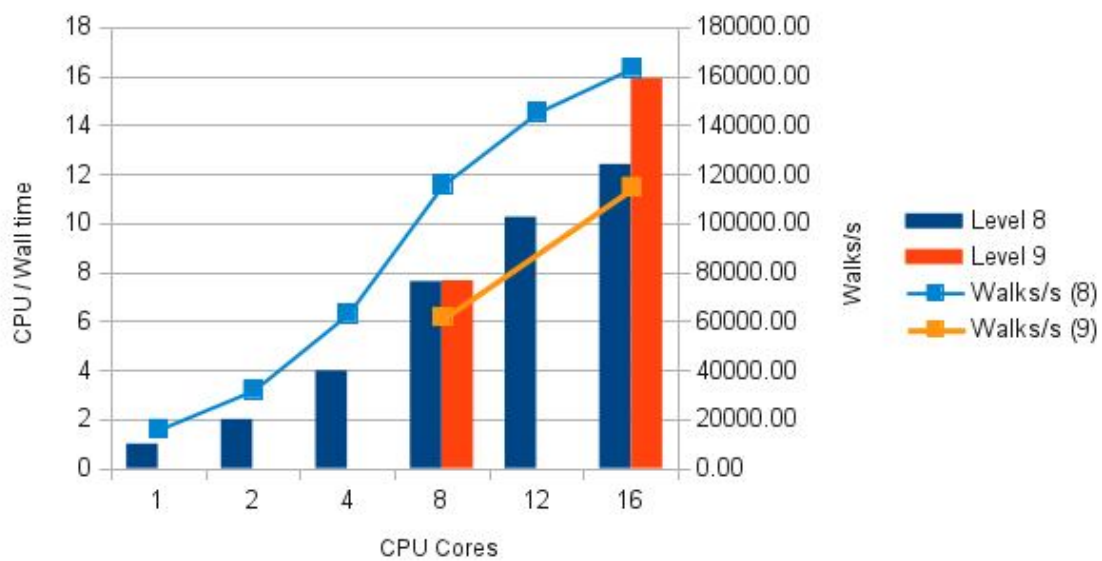


Figure 17 - CPU / Wall time ratio as function of number of CPU cores (BA-01-ETFBL)

Cores	Walk type	Level	Nest limit	Wall time [s]	CPU time [s]	Wall/CPU time	Eff.	Walks/s
6	A (2)	8	emul.	486	2803	5.77	0.96	136269.08
12	A (2)	8	emul.	406	3487	8.59	0.72	163120.13
18	A (2)	8	emul.	375	4244	11.32	0.63	176604.73
24	A (2)	8	emul.	319	5089	15.95	0.66	207607.44
48	A (2)	8	emul.	413	8216	19.89	0.41	160355.38
48	A (2)	9	emul.	18415	561336	30.48	0.64	132127.11

Table 10 - Scalability test results for Pecs SC with up to 48 CPU cores

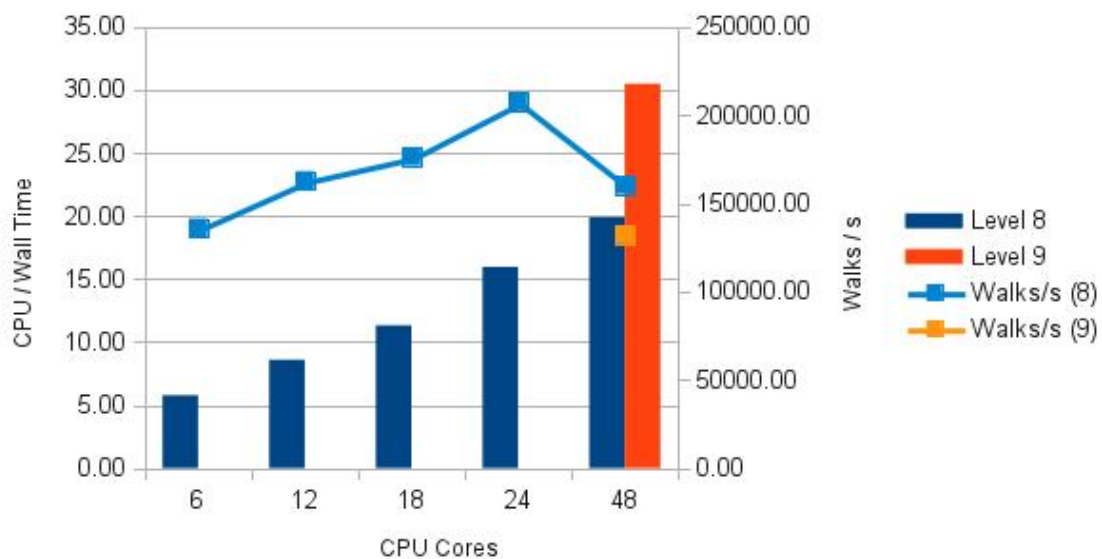


Figure 18 - CPU / Wall time as function of number of CPU cores (Pecs SC)

2.7.6. Memory Usage, CPU and cache

Memory requirements for this application are very modest and comfortably fit inside operating RAM with significant percent also fitting inside cache of CPU. Total memory requirements depend on level of recursion, nesting level and walk type but are under 128 MB total for tested levels.

2.7.7. Profiling

We have used gprof to find bottlenecks in execution and measure impact of fork placement at various steps in the algorithm.

2.7.8. Communication and I/O

Communication and I/O expenses are not critical for this application. We do however have a significant influence of thread forking to overall performance, which was explained in previous sections. This is also a reason that MPI version provided very limited performance increase.

2.7.9. Analysis

We have concluded that for this application significant performance increase was a result of rearchitecturing the application by producing new C++ code. MPI currently provides negligible benefits while use of OpenMP on SMP machines produces significant performance gains and scales very well up until tested 48 CPU cores. In order to achieve better scalability it is necessary to use higher levels of recursion for Sierpinski gaskets as level 8 stops producing meaningful performance increase after 24 CPU cores.

2.8. CFDOF

2.8.1. Summary

Code author(s): Sreten Lekic, Mihajlo Savic	
Application areas: Computational Chemistry	
Language: C, OpenFOAM	Estimated lines of code: 300
URL: http://wiki.hp-see.eu/index.php/CFDOF	

2.8.2. Implemented scalability actions

- In this application we are using OpenFOAM CFD toolbox and as such we did not alter the source code of the application.
- We have experimented with different mesh generation approaches and mesh partitioning algorithms in order to obtain better scalability.
- We created a proof-of-concept parallel post-processing application but we are currently facing stability issues with it.

2.8.3. Benchmark dataset

We used simplified smaller scale methane burner and rhoReactingFoam solver for benchmarking (gor_fine_paradox4). Total mesh size was 811 MB and contained 1893571 points, 18913870 faces and 9132732 tetrahedral cells.

2.8.4. Hardware platforms

We have performed performance testing on:

- PARADOX with up to 32 CPU cores
 - Intel(R) Xeon(R) CPU E5345 @ 2.33GHz
- BA-01-ETFBL with up to 16 CPU cores
 - AMD Opteron 6128 @ 2.00 GHz – total 16 CPU cores

2.8.5. Execution times

It can be seen from table with execution times that there are two dominant factors: pre/post-processing and simulation. Pre-processing and especially post-processing time increases with number of CPU cores used for simulation but, fortunately, not dramatically.

Table 11 - Scalability test results for PARADOX with up to 32 CPU cores

CPU cores	Pre-/Post-processing [s]	Simulation [s]	Total [s]	Simulation Speedup	Total Speedup
1	0.00	127137.60	127137.60	1.00	1.00
4	2560.00	33922.80	36482.80	3.75	3.48
8	2650.72	17496.00	20146.72	7.27	6.31
16	2817.06	8359.20	11176.26	15.21	11.38
32	3140.02	4135.05	7275.07	30.75	17.48

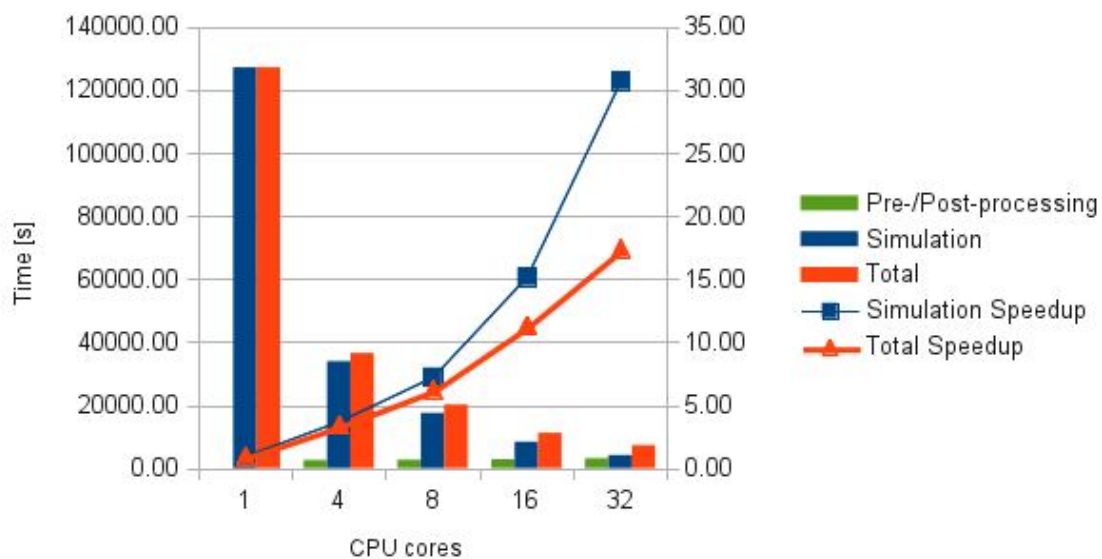


Figure 19 - Simulation and total times and speedup as function of number of CPU cores

2.8.6. Memory Usage, CPU and cache

Memory requirements for this application are heavily dependent on the size of the problem being simulated. In our testing, we concluded that one should not use over 1 GB per CPU core (which fits with most of the problems simulated as well as available project infrastructure).

2.8.7. Profiling

As we did not alter the source code of the CFD and chemistry solver we did not have the need for profiler.

2.8.8. Communication and I/O

I/O operations were affecting mostly pre- and post-processing parts of the workflow but not in the significant amount. Communication between processes can be minimized by using adequate mesh partitioning approach. For generic use case we would suggest using scotch which is designed to minimize number of processor boundaries. If there is sufficient knowledge of concrete simulation and model behavior better results can be achieved by using hierarchical or, better yet, manual decomposition.

2.8.9. Analysis

Since this application was based on tried and true solvers that have already been thoroughly benchmarked and analyzed, we focused our attention to issues specific to this concrete case.

When discussing scalability of simulation, one has to take into consideration several factors. If we are dealing with cold flow simulations, with no chemistry involved, then it is fairly easy to achieve good scalability of simulation. But, when we include chemistry in simulation things start to get complicated as chemistry is more time consuming to simulate especially when dealing with complex reactions and more realistic models. As can be seen on Figure 20 close to the beginning of the simulation, computationally intensive part of the simulated model is fairly small and using huge number of CPUs would bring limited benefits. If increasing performance is a must even at this stage, one must take care of properly partitioning the mesh so that we do not end up with many processes waiting for few with more complex simulation to handle. After a while, we can see that most of the model volume is now involved in time consuming chemistry calculations and as such it becomes easier to decompose the mesh in such a way to achieve good scalability.

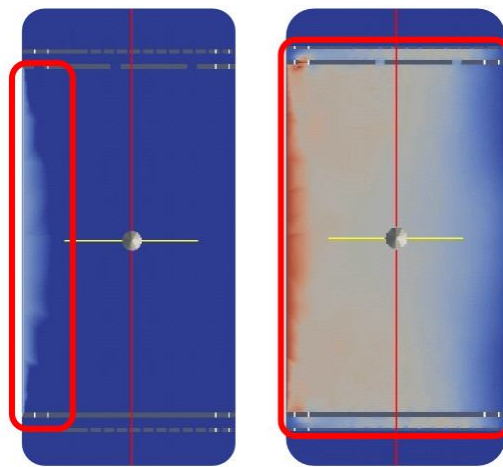


Figure 20 - Cross section of burner after $T=0,1$ s and $T=1,3$ s (illustration)

Above analysis applies foremost to solvers that work in time domain. When we are dealing with steady-state solvers, according to our results, they tend to reach an analogous point in fairly short amount of time and are as such better, or at least easier, choices for good scalability.

2.9. DNAMA

2.9.1. Summary

Code author(s): The Exelixis Lab	
Application areas: Life Sciences	
Language: C	Estimated lines of code: 50000
URL: http://wiki.hp-see.eu/index.php/DNAMA	

2.9.2. Implemented scalability actions

- RAxML versions can be divided by parallelization methods as:
- Coarse grained parallelization – using MPI
- Fine grained parallelization – using OpenMP and later Pthreads
- Hybrid version, which combines coarse and fine grained parallelization.
- Application was tested without and with SSE3 support for Intel compilers
- Raxml was compiled with GCC and Intel compiler (ICC)
- Work was performed for smaller and larger datasets for single gene and up to 5 genes in multigame mode
- Since RAxML doesn't offer some tree visualization tools for randomly generated and best trees we used Dendroscope and ugene for this action.
- Application was launched on platform with PBS scheduler (HPCG) and Sun Grid Engine (Pecs SC and Debrecen SC)

2.9.3. Benchmark dataset

Benchmark was completed on sample of 213 DNA sequences of *Salmo trutta* from different geographical region of Europe - 552 base pairs; 20 DNA sequences with 5 genes for multigene analysis

2.9.4. Hardware platforms

Three supercomputers are used for RAxML tests:

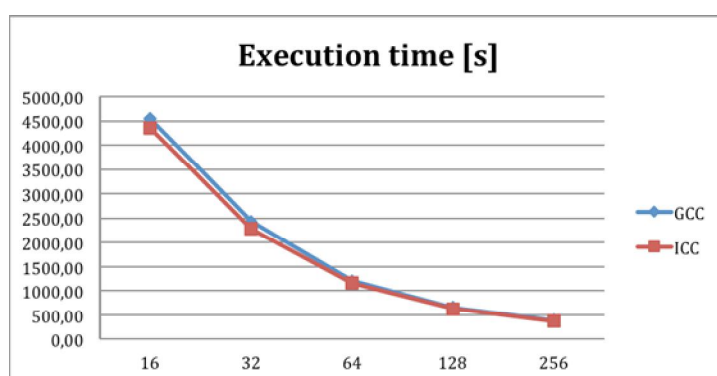
- HPCG cluster with Intel Xeon X5560 CPU @2.8 Ghz,
- Debrecen SC with Intel Xeon X5680
- Pecs SC with Intel Xeon X7542 (Nehalem EX)

2.9.5. Execution times

Executing time is measured on HPCG cluster using MPI and OpenMP and GCC and Intel compiler. MPI test performed with 3200 bootstraps (randomized trees) while Pthreads was done with 100 bootstraps.

Table 12 – DNAMA MPI result

Cores	GCC			ICC		
	CPU time [s]	Speedup	Efficiency	CPU time [s]	Speedup	Efficiency
16	4544,49			4348,24		
32	2426,11	1,87	0,94	2278,67	1,91	0,95
64	1203,55	3,78	0,94	1155,79	3,76	0,94
128	646,42	7,03	0,88	619,96	7,01	0,88
256	382,62	11,88	0,74	367,97	11,82	0,74

**Figure 21 – DNAMA MPI result**

Cores	GCC			ICC		
	CPU time [s]	Speedup	Efficiency	CPU time [s]	Speedup	Efficiency
1	1185,35			1195,83		
2	815,34	1,45	0,73	823,37	1,45	0,73
3	697,62	1,70	0,57	621,74	1,92	0,64
4	601,46	1,97	0,49	552,54	2,16	0,54
6	557,04	2,13	0,35	504,29	2,37	0,40
8	549,03	2,16	0,27	492,02	2,43	0,30
12	608,27	1,95	0,16	517,01	2,31	0,19
16	656,04	1,81	0,11	529,77	2,26	0,14

Table 13 – DNAMA Pthreads result

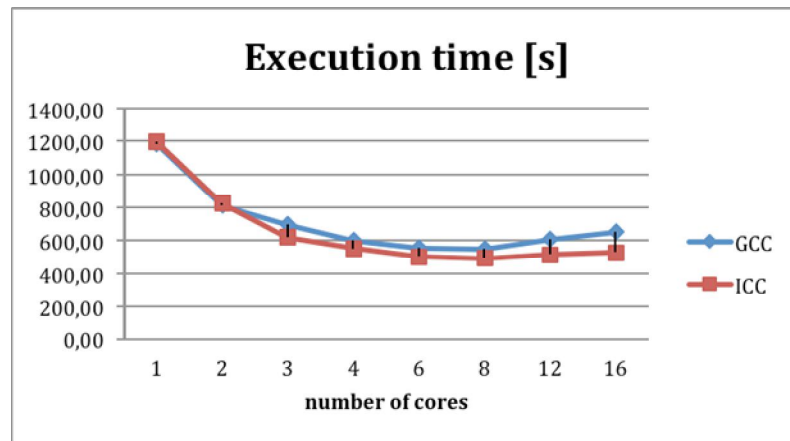


Figure 22 - DNAMA Pthreads result

2.9.6. Memory Usage

RAXML uses up to 10MB per working bootstrap (or per core) or up to 160 MB per server on HPCG, so it can be classified as low consumer of RAM.

2.9.7. Profiling

Profiling was done by code developers.

2.9.8. Communication

Communication takes less than 10% of time and therefore DNAMA can be classified as not communication intensive application.

2.9.9. I/O

Input file is small or medium sized (up to few MB) as well as main output files. Application makes few files for every bootstrap which summary size can go up to few GB, but they are downloaded only in case of analysis of every tree in bootstrap, not only best tree.

2.9.10. CPU and cache

We consider that most of the computations of the RAXML work in the cache for the tested Intel-based version.

2.9.11. Analysis

MPI version of RAXML shows relatively good results and we used them most of time, mostly up to 128 cores. Number of used cores should be commensurate with number of executed bootstraps and number of DNA sequences. GCC and ICC showed similar results with MPI.

Since Pthreads version gave us bad results for larger number of cores we decided to use application with 2 and 4 threads in regular use. Pthreads parallelization is per length of DNA sequence, so this version can give better performance for different dataset with larger number of base pairs per DNA sequence, especially with more than 10.000 base-pairs. Intel compiler gave us better result than GCC for larger number of cores.

Hybrid version, which combines MPI and Pthreads, was tested, but their scalability results were weaker since Pthreads can't give enough speedup with dataset used in benchmark. RAxML light and Examl was also tested, which do computations over predefined tree. RAxML was more comfortable for our work, because it makes computation over different, randomly generated trees and gives best tree as a result.

2.10. AMR_PAR: porting from Windows to Linux

2.10.1. Summary

Code author(s): Boris Rybakin, Nicolai Iliuha	
Application areas: Computational Physics	
Language: Fortran	Estimated lines of code: 700 (9 modules)
URL: http://wiki.hp-see.eu/index.php/AMR_PAR	

2.10.2. Implemented scalability actions

Initial version of the AMR_PAR application elaborated for running in IMI ASM-RENAM Cluster environment that supported virtualization platforms and allowed to use various operating systems. The application was elaborated in OpenMP mode and prepared for execution on Microsoft Windows Compute Cluster 2003.

For transforming AMR_PAR application to run in the regional HP-SEE infrastructure the application developers made preparation works for the application porting to the assign remote sites. Preparation was being carrying out on virtual machine with Scientific Linux 5.5 and Intel® Parallel Studio XE 2011 for Linux.

2.10.2.1 *The sources of some problems when porting applications' code from Windows to Linux platform*

- Using in program code blocks in other languages;
- Using the default declarations of variables and constants. They can be signed or not, long or short integer;
- The absence of variables initialization when allocating memory for the structure or class. Do not rely on the compiler options to automatically clear the memory with zeros;
- Not set the initial values for local variables. Failure can take many forms when initial values of the variables are erroneous. When the program runs under the debugger, memory is usually cleared, making it difficult to localize the error;
- Lack of control when using pointers in C language or C++. When passing pointers as parameters to functions it is appropriate to pass the size of the area referenced by a pointer. If a pointer is passed to a structure it is expedient to provide a field with size of the structure, which must be filled before the function call. After receiving of the pointer function first step is to check that the pointer is not equal to zero and the size of the corresponding field has a valid value;
- Function returns a pointer to a local variable - this code may work correctly, but porting to another platform or using of a different compiler generates an error because of differences in the organization of the stack. These errors should be excluded at the stage of analyzing of the source code - or the function must return a local variable, or to fill the data area which is given by a pointer;
- The lack of control of the results of memory allocation. Failure to allocate memory can lead to an exception or return a null value. It depends on the compiler, its parameters and the platform used;
- Lack of control over the release of repeatedly allocated memory area. Analysis of the pointer at the end of the program and after the redistribution of memory will help to avoid waste of resources.

2.10.2.2 Problems when porting interactive applications from Windows to Linux platform

As the results of use of visual development environments - interface mixed with "computing" part.

Use of system-dependent functions for the implementation of the processing events associated with I/O devices: mouse, keyboard, timer, etc. Dominated calls to API (application programming interfaces) and MFC (Microsoft Foundation Classes) instead of the standard library functions.

2.10.2.3 The development of portable code with a graphical interface.

First of all, it is necessary to design the program or make changes to the ready-one to realize graphical user interface in separate routines. Interactive and computing parts should be separated in source code.

The problem of porting of the source can be solved in two ways:

- The first - to use the POSIX (Portable Operating System Interface for Unix) standard functions;
- The second - to create the macro for the required functions in order the main text looks the same on different platforms.

Most often used encoding for storing text data are OEM, ANSI, KOI-8 and UNICODE. In different operating systems functions which work with character strings require different encoding. For permanent storage is advisable to use ONE encoding of all text data. Before the output of the text on the screen it can be recoded in accordance with the required current encoding.

To simplify the maintenance and upgrading of software it is need to design reentrant subroutines. One of the conditions for this is to minimize the use of global variables - use them to store constants or include in routines critical sections and semaphores. In most cases local variables and parameters passing can be used. But in this case it may be a bug related to stack overflow.

2.10.2.4 Packages for helping to solve the problem of porting applications

1. Mainsoft for UNIX and Linux, formerly called Visual MainWin for Unix and Linux. It is a cross-platform development solution that enables software developers to write Visual C++ applications in the productive Microsoft Visual Studio development environment and deploy them natively to a variety of UNIX and Linux platforms.

The package consists of several parts:

- Inspector of code that allows to detect system-dependent areas;
- Preprocessor that prepares the source code for compiling with GCC (or any other UNIX-compiler);
- An extensive library of functions, implementing:
 - Windows-primitives (SEH, DLL, processes / threads, tools for their synchronization, registry, clipboard and national languages support);
 - a graphical and user interface (GDI32, USER32);
 - COM-model (ActiveX, OLE, MIDL, DCOM);
 - runtime library (ALT, MFC, C Runtime library).

<http://dev.mainsoft.com/Default.aspx?tabid=53>

2. wxWidgets (formerly wxWindows) is a widget toolkit and tools library for creating graphical user interfaces (GUIs) for cross-platform applications. wxWidgets enables a program's GUI code to compile and run on several computer platforms with minimal or no code changes.

It is free and open source software, distributed under the terms of the wxWidgets License, which satisfies those who wish to produce for GPL and proprietary software.

<http://www.wxwidgets.org/about/feature2.htm>

Porting MFC applications to Linux. *A step-by-step guide to using wxWindows:*

<https://www.ibm.com/developerworks/library/l-mfc/>

2.10.2.5 Specificity of porting of AMR_PAR application (64 bit, Fortran)

Applications, performed on the computing resources of the project do not have interactive graphic interfaces. In this case one of the ways to avoid errors when porting - to use in Windows and Linux compilers, produced by one developer, for example Intel Parallel Studio XE for Windows and Linux.

The absence of a graphical interface, the use of standard functions and libraries, taking into account the previously described problems of portability allows porting of application to reduce to a simple recompilation of the source code.

For AMR_PAR application porting to Linux next main steps were done:

- Removed interactive interface;
- Removed block visualization of calculation results;
- Dynamic memory was used for large arrays;
- Intel Parallel Studio XE for Windows and Linux were used. Next keys were used when compiling the application:
- «-heap-arrays» - all local arrays are moved to the heap, greatly reducing the load on the stack.
- «-mcmmodel large -shared-intel» - to use static arrays larger than 2 Gb.

2.10.2.6 Intel Parallel Studio XE 2011 with VS2010.

<http://software.intel.com/en-us/articles/intel-parallel-studio-xe/>

Intel® Parallel Studio XE parallel software development suite combines Intel's C/C++ and Fortran compilers, performance and parallel libraries, error checking, code robustness and performance profiling tools into a single suite offering.

This helps boost application performance and increase the code quality, security, and reliability needed by high-performance computing.

At the same time, the suite eases the procurement of all the necessary tools for high performance, and simplifies the transition from multicore to manycore processors in the future.

Intel® Parallel Studio XE includes the following industry-leading components:

- [Intel® Composer XE](#) Optimizing compilers and high-performance libraries
- [Intel® Inspector XE](#) Powerful thread and memory error checker

- [Static Security Analysis](#) Close security vulnerabilities and weed out a range of bugs
- [Intel® VTune™ Amplifier XE](#) Advanced performance profiler
- Intel® Parallel Advisor Threading assistant tool for C/C++ Microsoft Visual Studio developers - available with Windows versions of Intel® Parallel Studio XE or Intel® C++ Studio XE.

2.10.3. Hardware platforms

Testing procedure for the application version that was transferred to Linux platform is based on using access to SGI UltraViolet 1000 supercomputer, located in Pecs, Hungary at NIIFI branch. Access to Pecs supercomputer was provided within special Agreement signed by RENAM and NIIFI;

2.10.4. Execution times

Acceleration and Run Time dependences from CPU cores is presented on the figure below. For 128x128x128 dimension best number of cores — 4.

4 cores: walltime - 3,3 min, CPU time -13,2 min.

16 cores: walltime - 3,7 min, CPU time - 59,1 min

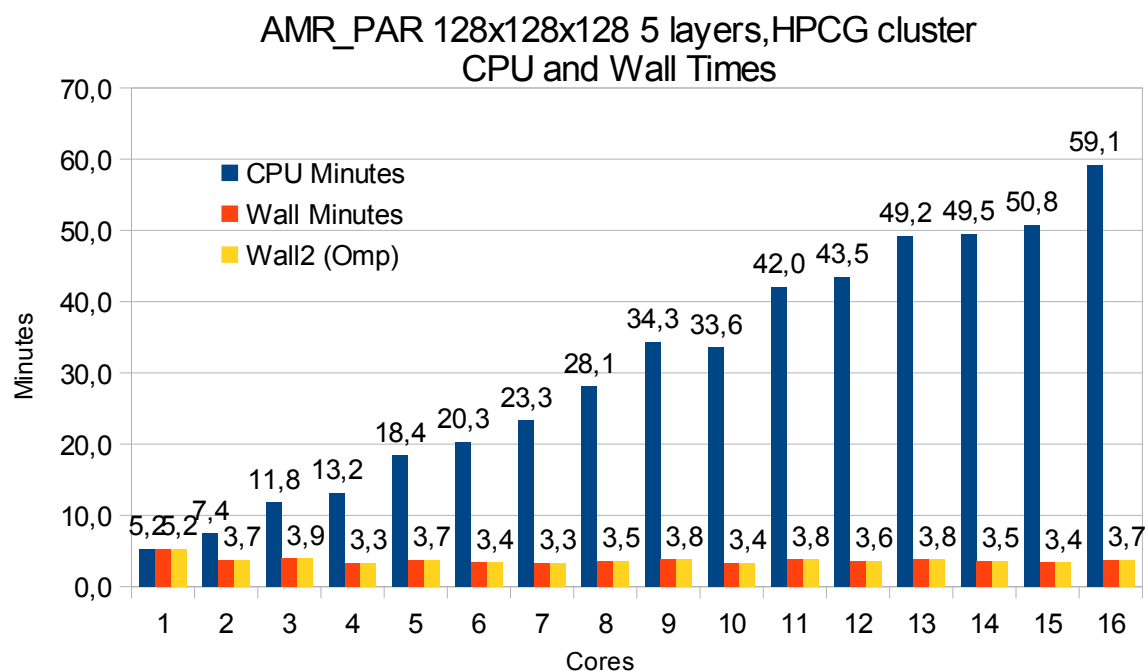


Figure 23 – AMR_PAR, HPCG cluster, CPU and wall times

2.10.5. Memory Usage

Calculated requirements of computational resources for the current OpenMP version of AMR_PAR application

Dimension	Layers	Max Iteration per level	Cores	RAM Gb	CPU minutes	WallTime minutes
128x128x128	5	200000	4	0,789	28	3,5
256x256x256	5	200000	4	5,972	273	68
256x256x256	5	200000	8	6,062	527	66
256x256x256	5	200000	12	6,068	807	68
384x384x384	5	200000	8	19,2	2110	270
448x448x448	5	200000	8 — 16	37,7	~ 4500	~ 500
512x512x512	5	200000	8 — 16	~ 55,6	~ 130 hours	~ 17 hours
1024x1024x1024	5	200000	16 — 32	~ 415	~ 2000 hours	~ 248 hours
2048x2048x2048	5	200000	32 — 64	~ 3250	~ 1200 days	~ 154 days

Table 14 – Memory usage of AMR_PAR

2.10.6. Analysis

Some application scalable problems were found during tests. Information was begun analysing by the application developers. After negotiation of approaches for creation of a new version of scalable OpenMP AMR_PAR application the refine code was proposed for further application development.

2.11. DeepAligner and DiseaseGeneMapper

2.11.1. Summary

Code author(s): Gergely Windisch, Akos Balasko, Miklos Kozlovsky	
Application areas: Life sciences	
Language: C++, BASH	Estimated lines of code: 2000
URL:	
<ul style="list-style-type: none"> ▪ http://wiki.hp-see.eu/index.php/DeepAligner ▪ http://wiki.hp-see.eu/index.php/DiseaseGene 	

2.11.2. Application description

The two portlets developed at Obuda University (Deep Aligner and Disease Gene Mapper) share a common algorithm which takes up about 95% of the total execution time so the scalability studies were done together for both applications.

The Disease Gene Mapper service allows researchers to utilize the HPC infrastructure to find gene sequences in an organism which have already been connected to a disease in a different organism. The users of DGM have to provide the "source" disease name and an organism, and a second organism against which the gene sequence search will be executed. Deep Aligner portlet allows researchers to search for a multitude of short gene sequences in a given organism. The users can upload multiple sequences in a compressed file (.rar, .zip or .tar.gz), the portlet searches for all of them in the selected database.

2.11.3. Implemented scalability actions

Our aim was achieving high performance for our two portlets. The portlets execute a number of different applications, but the most computationally challenging is mpiBlast which takes most of the execution time so we focused our benchmarks on that. Amongst other things we have tried

- running the program under different implementations of MPI
- executing on different hardware environments
- experimenting with different compilers and compiler options
- experimenting with mpiBlast options like database fragmenting, enabling parallel write etc.

2.11.4. Benchmark dataset

The blast database size was 5.1 GB, and the input sequence size was 29.13 kB. Each measurement was executed 10 times, the average of the 10 executions was taken as the final result

2.11.5. Hardware platforms

A number of hardware platforms have been used for the testing of the applications. The portlet we have developed is connected to all these different HPC infrastructures and it

is the job of the middleware to choose the appropriate for each execution. For our benchmarks we specified the infrastructure the application was supposed to use.

The benchmarks were executed on five different HPC infrastructures:

- Debrecen
 - Intel Xeon X5680 (Westmere EP) 6 core nodes, SGI Altix ICE8400EX
 - 1536 CPU cores
 - 6 TB memory
 - 0.5 PB storage
 - Total capacity: ~18 TFlops
- Budapest (NIIF)
 - fat-node cluster using CP4000BL blade
 - AMD Opteron 6174 CPUs, 12 cores (Magny Cours)
 - ~700 cores
 - Total Capacity ~5 TFlops
- Pecs
 - SGI UltraViolet 1000 - SMP (ccNUMA)
 - CPU: Intel Xeon X7542 (Nehalem EX) - 6 cores
 - 1152 cores
 - 6 TB memory
 - 0.5 PB memory
 - Total capacity: ~10 TFlops
- Szeged
 - fat-node cluster using CP4000BL blade
 - AMD Opteron 6174 CPUs, 12 cores (Magny Cours)
 - 2112 cores
 - 5.6 TB memory
 - 0.25 PB storage
 - Total Capacity ~14 TFlops
- Bulgaria
 - Blue Gene/P with PowerPC CPUs
 - 2048 PowerPC 450 based compute nodes
 - 8192 cores
 - 4 TB memory

2.11.6. Software platforms

The applications were tested using multiple software stack

- Different MPI implementations
 - openmpi_gcc-1.4.3
 - openmpi_open64-1.6
 - mpt-2.04
 - openmpi-1.4.2
 - openmpi-1.3.2
- Different compilers
 - openc
 - icc
 - openmpi-gcc

Each of the different hardware platforms have multiple MPI environments. We have tested our applications with multiple versions. There usually is one specific preferred at each of the HPC centers which we preferred using.

2.11.7. Execution times

The following graphs show the results of the executions. The execution times varied a little depending on the HPC centre used, but they were more or less stable so we only include the results from the Budapest server. The following graphs show the result of multiple executions of mpiBlast on the same database with the same input sequence on the same computer. The only difference being the number of CPU cores allocated to the MPI job¹. Figure 24 shows the execution times measured by mpiBlast. If executed on just one CPU it takes 3376 seconds for the job to finish (about 53 minutes). As we can see the applications scales well, the execution times drop when we add more and more CPUs.

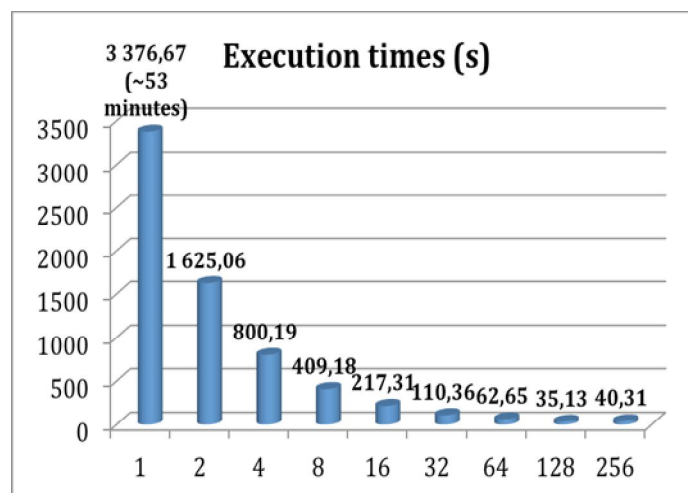


Figure 24 - Execution times measured by mpiBlast in seconds

Figure 25 shows the speedup in percentage compared to the original measurement on one CPU core. The results show that the application loses momentum around 32 cores but the performance increases until around 128 cores. Figure 26 shows the same results but from a different angle: that of the efficiency – the speedup / number of cores.

¹ The actual number of CPU cores was two more than what is shown in the graphs – 2 additional cores are used by mpiBlast for execution maintenance and management

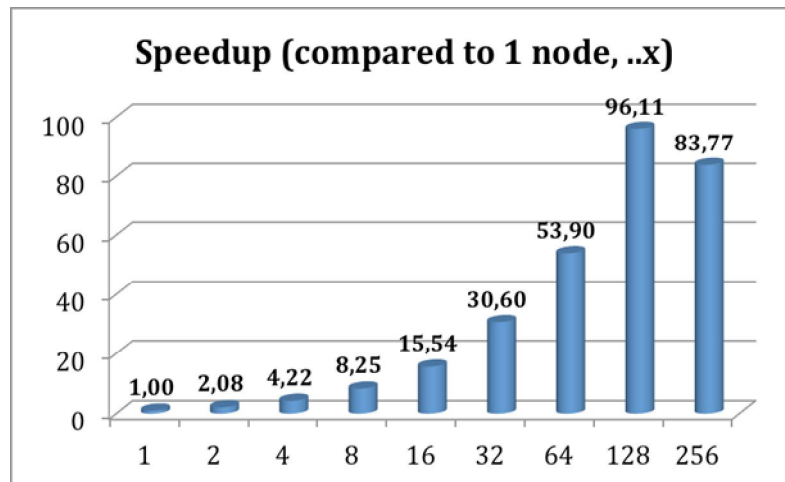


Figure 25 - mpiBlast speedup using multiple CPU cores compared to running it on just one CPU core

Ideally in a perfectly scaling application the numbers should be around one. As we can see from the graph the efficiency is quite high (>75%) until the number of cores reaches 128 where it starts to drop.

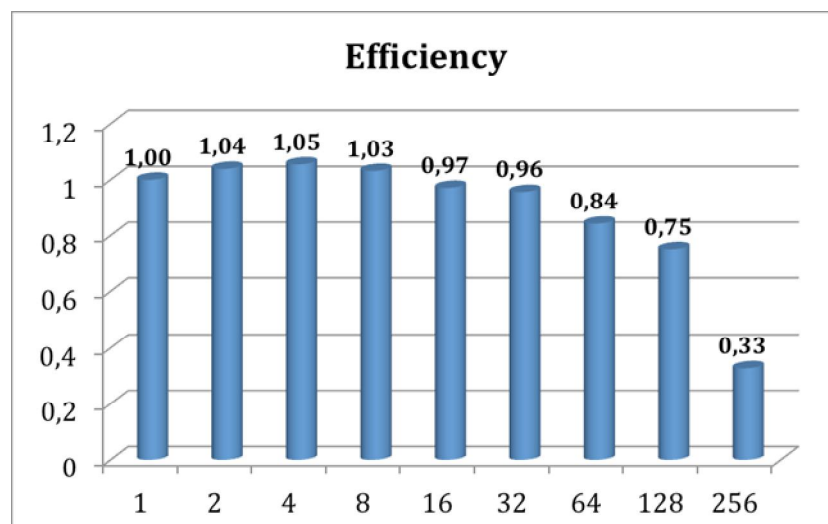


Figure 26 - Efficiency of using multiple CPU cores

2.11.8. Further optimization

The first task when using mpiBlast is to split the blast database into multiple fragments. According to previous research, the number of database fragments have a direct impact on the performance of the application. Finding an optimal number was essential, so our database was split into different sizes. Figure 27 shows the measured execution times. The measurements were executed on 64 cores.

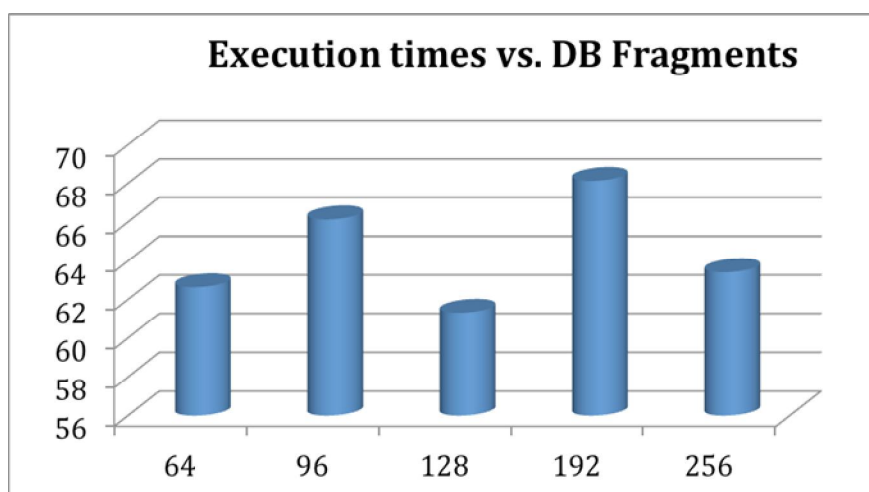


Figure 27 - Execution times in seconds using different number of Database Fragments

As it is apparent from the graph, the application performs best when the number of DB segments are integer multiples of the number of CPU cores. The reason is straightforward: this is the only way an even data distribution can be achieved amongst the cores.

2.11.9. Memory Usage

1	2	4	8	16	32	48	64	96	128
1,257	2,112	3,345	4,131	5,434	6,012	4,153	8,745	9,897	12,465

Table 15 - Memory usage while executing the application. The results come from the maxvmem parameter of qacct

As we can see the memory consumption (measured by qacct) increases as the number of cores is increased.

2.11.10. Profiling

The two applications we have created share some of the code base which results in a similar behavior. Both applications consist of three jobs in a WS-PGRADE workflow with job 1 being the preprocessor, job 2 doing the calculations and job 3 collecting the results and providing it to the user. The current implementation for the preprocessing is serial, we have investigated parallelizing but according to our profiling approximately 0.02 % of the total execution time is spent on Job 1 in DeepAligner, so yields no real performance gain but can cause problems so we voted against it. Job3 is 0.01% - most of the work is done in Job2. Job2 consists mainly of mpiBlast, the profiling shows the following results.

Job1	Job2	Job3
0,02%	99,97%	0,01%

Table 16 - Execution time ratio of the jobs in the whole DGM and DA portlets

Init	BLAST	Write	Other
1,79%	97,18%	0,19%	0,84%

Table 17 - Execution time ratio inside Job2

2.11.11. Communication

mpiBlast uses a pre-segmented database and each node have their own part where it searches for the input sequence so the communication overhead is very small.

2.11.12. I/O

1	2	4	8	16	32	48	64	96	128
0,001	0,001	0,002	0,003	0,004	0,011	0,016	0,019	0,027	0,029

Table 18 - I/O as measured using the IO parameter of qacct

As we can see on the previous table the I/O use increases as we increase the number of CPU cores in the job.

2.11.13. Analysis

From our tests, we conclude that our application scales reasonably well up until about 128 cores. When the appropriate MPI implementation is used on the HPC infrastructure the performance figures are quite similar – the scalability results are within the same region as expected. The number of database fragments play a significant role in the whole application and the best result can be obtained when that number is equal to or is an integer multiple of the number of cores. We have also noted that because of the high utilization of the supercomputing centers real life performance – wall clock time measured from the initialization of the job until the results are provided – could be better when using a smaller number of cores because small jobs tend to get scheduled easier and earlier.

2.12. FMD-PA

2.12.1. Summary

Code author(s): Manthos G. Papadopoulos, Heribert Reis	
Application areas: Computational Chemistry	
Language: Fortran	Estimated lines of code: 10000
URL: http://wiki.hp-see.eu/index.php/FMD-PA	

2.12.2. Implemented scalability actions

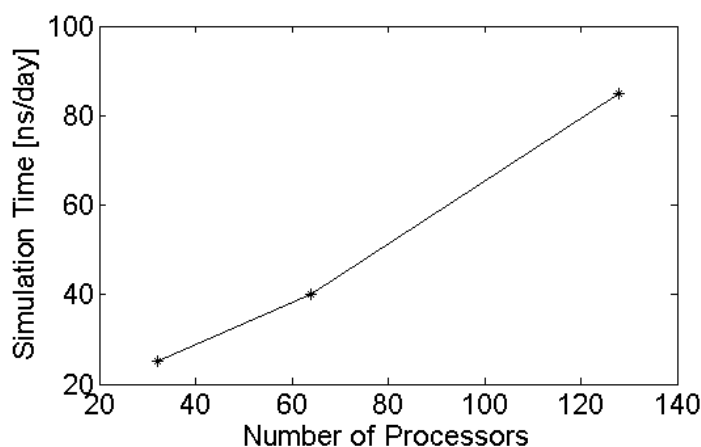


Figure 28 - HPCG cluster, FMD-PA

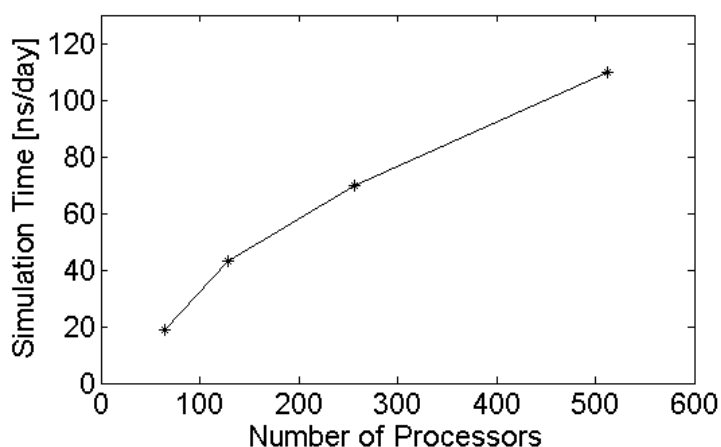


Figure 29 - Blue Gene cluster, FMD-PA

The above diagrams depict the efficiency of the two clusters as the number of processors increases. The efficiency is measured in nanoseconds of a Molecular Dynamics (MD) simulation per day using GROMACS package in double precision. The system under study composes of water molecules described by the Simple Point Charge

(SPC) model. As far as Blue Gene cluster, it is recommended to run with many processors (>128).

2.12.3. Benchmark dataset

A series of Molecular Dynamics (MD) simulations were conducted in order to test the efficiency of the two clusters. As a model system we chose an aqueous phase of 17131 molecules described by the well-known Simple Point Charge model. Besides, the treatment of electrostatics was done through the Particle Mesh Ewald method which is regarded quite time-consuming. The duration of all MD simulations was 20ns producing output files of approximately 3GB while all runs were ended successfully within 20h.

2.12.4. Hardware platforms

HPCG cluster and Blue Gene cluster.

2.12.5. Execution times

From one hour to several days.

2.12.6. Memory Usage

Several GBs (e.g. for a Gaussian job needs 4 GBs).

2.12.7. Analysis

The team tried different architectures: BlueGene, HPCG cluster (it scales on both)

2.13. CompChem

2.13.1. Summary

Code author(s): NAMD 2.9; http://www.ks.uiuc.edu/Research/namd/	
Application areas: Computational Chemistry	
Language: Charm++	Estimated lines of code: -
URL: http://wiki.hp-see.eu/index.php/CompChem	

2.13.2. Implemented scalability actions

- CompChem/RS application is oriented toward usage of existing source codes installed on our home cluster PARADOX/IPB and HPCG/BG. Programs for molecular dynamics simulations, *Ab initio*/DFT/Semiempirical QM calculation, docking calculations and cheminformatic tools were installed, covering diverse need of the users, which are mainly directed to design and development of novel molecules with potential therapeutic value.
- Some programs installed on PARDOX were offered by developers exclusively as executables (for example OpenEye applications, see <http://www.eyesopen.com/>), with their own MPI implementation for multi-CPU usage.
- Part of other programs are precompiled by developers, for example ORCA and NAMD, and executable most suitable for particular architecture can be found on the developers download area.
- The majority of CPU times used so far by CompChem application were spent by NAMD.
- In the next lines we intent to respond to request of referees – to find better architecture for our application, because of obvious poor scalability of the NAMD as the major application on PARADOX/IPB, which failed to provide good scalability with NAMD for bigger systems (usually medium sized proteins with ligands, counterions and explicit solvent – size in total ~ 90 000 atoms in majority of simulations). Problem of scalability is clearly due to the architecture of our home cluster – with slow interconnection between the nodes.
- In order to overcome overestimation of achieved speed/scalability, common for using predefined benchmarks, we challenged NAMD 2.9 scalability by benchmark made by the system taken from our every-day practice. The care was taken that whole system be comparable with NAMD native benchmark (<http://www.ks.uiuc.edu/Research/namd/performance.html>), but we add more demanding criteria.
- The scalability of NAMD 2.9 is examined on two clusters PARDOX/IPB and HPCG/BG.
- Very good scalability was obtained on HPCG/BG clusters, as is shown in following lines.

2.13.3. Benchmark dataset

As we mentioned above, our benchmark include the system that comprise all elements as native NAMD benchmark (size of the system, frequency of electrostatic evaluations, periodic boundary conditions). Our system comprises of protein, ligand, counter-ions and explicit solvent, and larger cut-offs; requesting more, and more demanding non-bonded and electrostatic interactions evaluation comparing to native NAMD benchmark.

Additionally, in our scalability results we include the output trajectory writing – time demanding, but crucial for every MD simulations. Also we applied external biasing forces, reference and constraints. All listed usually are challenge for the both speed of calculation and the scalability (for any MD simulation).

In this scalability study we duplicated number of patches in one dimension to use maximum of 64 or of 128 CPU's applied (same for all calculations).

System is minimized during 30 000 steps, than heated to 310 K during 10 000 steps. The 5 ns of unconstrained MD simulation retain ligand close to initial position; this means that system is very stable. Few pulling calculations (2 ns) give reliable results.

Benchmark was as follows: CHARMM FF, ~ 87 000 atoms, 14 Å cutoff, PME, PBC, SMD (reference file, constraints applied to stabilize system, pulling force applied), DCD (trajectory) writing every 1000 steps / 50 000 steps (1 ns = 1000000 steps)

2.13.4. Hardware platforms

PARADOX/IPB and HPCG/BG clusters

The following distinct hardware platforms were used:

- the PARADOX/IPB cluster with Intel 2 x quad core Xeon E5345 @ 2.33 GHz
- the HPCG/BG cluster with Intel Xeon X5560 CPU @2.8 GHz

2.13.5. Execution times

Scalability of the NAMD 2.9 - 86_64-ibverbs (installed on both clusters and used for scalability study) were shown in Table 19 and Table 20, and Figure 30 and Figure 31.

Table 19 - Scalability of NAMD 2.9 on PARADOX/IPB

CPU's	Nodes	Wall Clock (s)	CPUTime(s)	Memory (MB)	Speed-up Wall Clock	Speed-up CPU Time
8	1	7921.667	7620.740	86.401	-	-
16	2	5302.639	4177.521	69.138	1.494	1.824
32	4	3410.354	2268.378	62.297	2.323	3.360
64	8	2729.427	1377.746	57.380	2.902	5.531
128	16	2700.188	1399.444	58.531	2.934	5.446

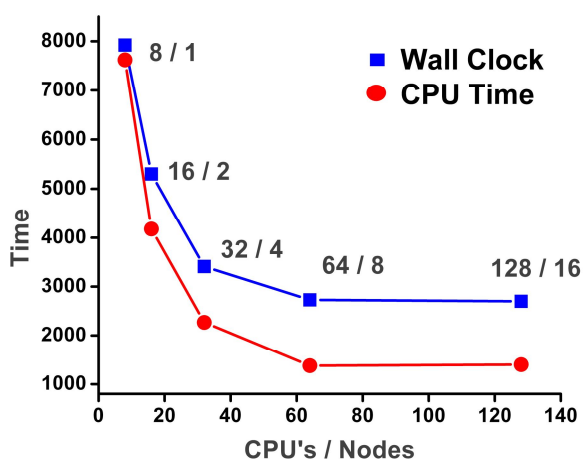


Figure 30 - Scalability of the NAMD 2.9, CPU's / Nodes

CPU 's	Nodes	Wall Clock (s)	CPU Time (s)	Memory (Mb)	Speed-up Wall Clock	Speed-up CPU Time
8	1	5526.474	5513.867	210.199	-	-
16	1	4702.783	4443.861	189.453	1.175	1.241
32	2	2173.008	2166.701	187.320	2.543	2.545
64	4	1289.415	1281.039	199.539	4.286	4.276
128	8	853.092	849.037	228.988	6.479	6.488

Table 20 - Scalability of NAMD 2.9 on HPCG/BG

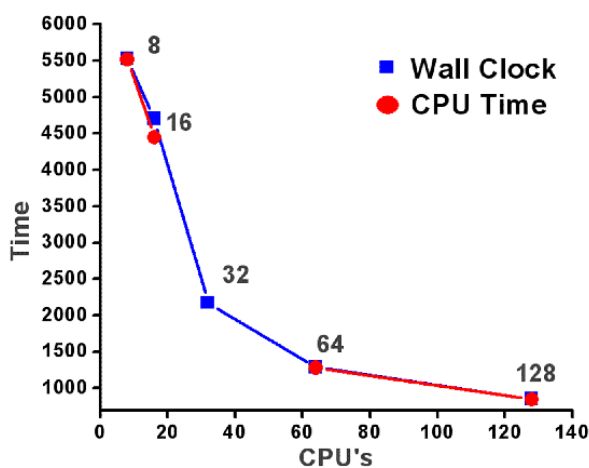


Figure 31 - Scalability of NAMD 2.9, CPU's

2.13.6. Memory Usage

NAMD always use low memory resources (except if large steps in trajectory writing is requested – which is *very* uncommon), but parallel efficiency is an issue.

2.13.7. Profiling

During NAMD MD simulations, after start-up phase which is usually exceptionally fast, most CPU time were used on evaluation inter-particle interactions, as well as trajectory file writing. Please be aware that we used (real) all-atom force field (means many more such evaluations than with united atom force field used).

2.13.8. Communication

The drastic difference in time needed for communication, comparing HPCG/BG and PARADOX/IPB can be seen from Table 19 and Table 20 (graphical representations on Figure 30 and Figure 31); and this is main and well-known issue of scalability of NAMD on PARADOX.

2.13.9. I/O

All pre- and post-processing are typically done on user terminals, therefore those phases were not included in our scalability study, different of some other application reports in this document.

It should be mentioned that commonly NAMD save two backup copies of all files needed for restarting the simulation (if not otherwise requested) and we used standard configuration in this part. Size of output files heavily depends of the time of simulation requested, trajectory frequency and external procedures applied. We applied external procedure (steered molecular dynamics) in our benchmark set, which did not slow-down NAMD on HPCG/BG.

3. Software harmonization

3.1. Harmonization status

	NIIF Szeged, HUN	NIIF Pécs, HUN	NIIF Debrecen, HUN	BG,BG	HPCG,BG	IFIN_BC, RO	IFIN_Bio, RO	InfraGRID, RO	ISS_GPU, RO	NCIT-Cluster, RO	PARADOX, RS	Metric value
Atlas		3,83		3,83	3,83					3,911		3,622
BLAS	3,037	3,22	3,22	3,037	3,037						3,037	2,988
BLACS		1,1	1,1	1,1	1,1							3,500
charm++				6,21	6,21		6					4,280
CPMD				3,151	3,151						3,132	4,025
FFTW	3,22	3,12		3,12	3,12	2,15	3,11	3,22			3,12	3,245
GotoBLAS				2			1,26					5,240
GROMACS	4,53	4,54		4,54	4,53							3,520
LAPACK	3	3,22	3,22	3,2	3,037	3,11					3,037	2,574
NAMD				2,7	2,7		2,6				2,8	3,700
OpenMPI	1,42	1,32				1,32				1,53	1,25	3,428
ROOT									5,26	5,28		4,520
ScaLAPACK				1,9	1,8						1,8	4,133
SPRNG				2	2						4	6,667
VMD				1,9	1,9		1,86					4,053
											Sum:	59,495

Table 21 shows the most typical libraries that are needed for the HP-SEE applications to be ported from one site to other. This table was defined in the D8.2 deliverable [7]. It shows the actual software stack of the HPC sites when D8.2 has been written more than a year ago. We created an equation, which help to analyze the harmonization status between the sites. If several versions of the same software are installed on the same site, then only the higher version number is used.

The metric is defined as follows:

$$k*0.5 + \sum_{i=1}^n |x_i - x_{avg}|$$

Equation 3 – Harmonization metric

The result of the formula is the distance value for a particular library. The target is to reduce it. Index "i" denotes the count of resource centers that have installed this library and index "k" represents those who have not. Value 0.5 is the weight of all not-installed software. Value of X_{avg} represents the average distance of the installed library versions. If the metric value is reduced, then the concerned library versions are coming closer, and thus the software components are considered to be more harmonized.

	NIIF Szeged, HUN	NIIF Pécs, HUN	NIIF Debrecen, HUN	BG,BG	HPCG,BG	IFIN_BC, RO	IFIN_Bio, RO	InfraGRID, RO	ISS_GPU, RO	NCIT-Cluster, RO	PARADOX, RS	Metric value
Atlas		3,83		3,83	3,83					3,911		3,622
BLAS	3,037	3,22	3,22	3,037	3,037						3,037	2,988
BLACS		1,1	1,1	1,1	1,1							3,500
charm++				6,21	6,21		6					4,280
CPMD				3,151	3,151						3,132	4,025
FFTW	3,22	3,12		3,12	3,12	2,15	3,11	3,22			3,12	3,245
GotoBLAS				2			1,26					5,240
GROMACS	4,53	4,54		4,54	4,53							3,520
LAPACK	3	3,22	3,22	3,2	3,037	3,11					3,037	2,574
NAMD				2,7	2,7		2,6				2,8	3,700
OpenMPI	1,42	1,32				1,32				1,53	1,25	3,428
ROOT									5,26	5,28		4,520
ScaLAPACK				1,9	1,8						1,8	4,133
SPRNG				2	2						4	6,667
VMD				1,9	1,9		1,86					4,053
											Sum:	59,495

Table 21 – Software stack status, input: D8.2

	NIIF Szeged, HUN	NIIF Pécs, HUN	NIIF Debrecen, HUN	BG,BG	HPCG,BG	IFIN_BC, RO	IFIN_Bio, RO	InfraGRID, RO	ISS_GPU, RO	NCIT-Cluster, RO	PARADOX, RS	Metric value
Atlas	3,83	3,83	3,83		3,83	3,8	3,8			3,911	3,8	1,673
BLAS	3,037	3,22	3,22	3,037	3,037	3,037	3,037	3,037		3,037	3,21	1,255

BLACS	1,1	1,1	1,1	1,1	1,1	1,1	1,1	1,1		1,1	1,1	0,500
charm++					6,21							5,000
CPMD				3,151	3,151						3,132	4,025
FFTW	3,22	3,12	3,12	3,12	3,12	2,15	3,11	3,33		3,31	3,12	2,344
GotoBLAS	1,13	1,13	1,13	2								4,805
GROMACS	4,53	4,54	4,54	4,54	4,53						4,55	2,533
LAPACK	3	3,22	3,22	3,2	3,037	3,42	3,42	3,42		3,31	3,037	1,813
NAMD	2,9	2,9	2,9	2,7	2,7						2,9	3,033
OpenMPI	1,43	1,32	1,42		1,43	1,43	1,43	1,54		1,6	1,25	1,587
ROOT	5,34	5,34	5,34						5,26			3,620
ScaLAPACK	1,75	1,75	1,75	1,9	1,8	2,02	2,02	2,02		2,02	1,8	1,630
SPRNG				2	2						4	6,667
VMD	1,9	1,9	1,9	1,86	1,9							3,064
											Sum:	43,549

Table 22 – Software stack status, 22th February 2013

The harmonization level has been improved because the metric value has been decreased from **59,495** to **43,549**.

3.2. HP-SEE software stack

WP8 has defined a metric within deliverable D8.2 which describes the harmonisation status of the HPC centres. The metric values have been calculated again in this document. The result has been improved since missing software components have been installed in the centres.

WP8 has defined two software stacks, which helped to improve the harmonisation level of the sites based on the needs of the regional HPC user communities and also taking into account the relevant work that has been done in the pan European HPC infrastructure PRACE:

- Minimal software stack
- Recommended software stack

The minimal software stack should be installed on all sites (with some exceptions in case of non suitability) while the recommended software stack contains optional software components that improve the interoperability of the infrastructure if used.

3.2.1. Minimal software stack (mandatory for all HPC centre)

Shells:

- bash
- tcsh

Compilers:

- C
- C++
- Fortran
- Java

Comments:

- GNU Compilers should be available in all sites (unless specific reasons prevent it)
- Vendor specific compilers where appropriate (Intel, IBM, PGI, AMD compiler)
- Java of any version or vendor (if architecture justifies it)

Libraries/Communication

- MPI (At least one of MPICH1, MPICH2, OpenMPI, MVAPICH1, MVAPICH2), OpenMPI is preferred if possible.
- OpenMP
- CUDA (For sites with NVIDIA GPUs)

Libraries/Numerical and I/O:

- BLACS
- BLAS
- FFTW
- ScaLAPACK
- MPIBLAST
- LAPACK

Tools:

- gprof or any other profiler
- gdb or any other debugger
- Perl
- Python
- Tcl

Grid Middleware

- We are recommending EMI middleware (ARC, UNICORE or gLite)

3.2.2. Recommended software stack

Minimal software stack + optional applications and libraries

Libraries:

- SPRNG
- Atlas
- charm++
- GotoBLAS

Applications:

- Octave
- GROMACS
- AMBER
- GAMESS
- NAMD
- VMD
- CPMD
- ROOT

Table 23 notations:

- IP: Installation is in progress
- N/A: It is not installed (reason: HPC centre policy does not allow to install it or it is not available for that platform)
- *: Installed

Table 23 – Minimal and recommended software stack status, 22th February 2013

	NIIF Szeged, HUN	NIIF Pécs, HUN	NIIF Debrecen, HUN	NIIF Budapest, HUN	BG Blue Gene/P, BG	HPCG, BG	IFIN_BC, RO	IFIN_Bio, RO	InfraGRID, RO	UVT Blugene, RO	NCIT-Cluster, RO	PARADOX, RS	FINKI SC, Macedonia
Minimal software stack													
bash	*	*	*	*	*	*	*	*	*	*	*	*	*
tcsh	*	*	*	*	*	*	*	*	N/A	N/A	*	*	N/A
GNU compiler (C, C++, Fortran)	*	*	*	*	*	*	*	*	*	N/A	*	*	*
Java compiler	*	*	*	*	*	*	*	*	N/A	N/A	*	*	*
Vendor compiler (C, C++, Fortran)	*	*	*	*	*	*	N/A	N/A	*	*	*	*	N/A
OpenMP	*	*	*	*	*	*	*	*	*	*	*	*	*
MPI	*	*	*	*	*	*	*	*	*	*	*	*	*
CUDA	*	N/A	N/A	N/A	N/A	*	N/A	N/A	N/A	N/A	N/A	N/A	N/A
BLACS	*	*	*	*	*	*	*	*	*	*	*	*	*
BLAS	*	*	*	*	*	*	*	*	*	*	*	*	*
FFTW	*	*	*	*	*	*	*	*	*	*	*	*	*
Scalapack	*	*	*	*	*	*	*	*	*	*	*	*	*
MPIBLAST	*	*	*	*	N/A	N/A	*	*	*	*	*	*	*
LAPACK	*	*	*	*	*	*	*	*	*	*	*	*	*
Profiler	*	*	*	*	*	*	*	*	*	*	*	*	*
Debugger	*	*	*	*	*	*	*	*	*	*	*	*	*
Perl	*	*	*	*	*	*	*	*	*	*	*	*	*
Python	*	*	*	*	*	*	*	*	*	*	*	*	*
Tcl	*	*	*	*	*	*	*	*	N/A	N/A	*	*	*
Middleware	*	*	*	*	*	*	IP	IP	*	N/A	N/A	*	*
Recommended software stack													
SPRNG						*						*	
Atlas	*	*	*	*		*	*	*			*	*	*
charm++						*							
GotoBLAS	*	*	*	*	*								
Octave	*	*	*	*	*								*
GROMACS	*	*	*	*	*	*						*	*
AMBER					*	*							
GAMESS					*	*							*
NAMD	*	*	*	*	*	*						*	*
VMD	*	*	*	*	*	*							
ROOT	*	*	*	*									
CPMD					*	*						*	*

4. Interoperability

4.1. Resource management system and user authentication

4.1.1. Resource management system

The HP-SEE Resource Management System has two main goals:

- Providing a centralized way for requesting access to the HP-SEE infrastructure and requesting computing resources
- Providing an easy way to monitor the resources used by the project

The system is responsible for managing both requests for access to the infrastructure and requests for local access to the HPC centers. It also provides an easy way for monitoring resource requests and remaining resources. New HP-SEE users can register here: <https://portal.ipp.acad.bg:8443/hpseeportal/>

Requirements for system usage:

- X509 certificate
- Browser and Internet

4.1.1.1 Registration of new users

The first step in using the Resource management system is to register and submit a request for account creation to the HP-SEE Application Review Committee.

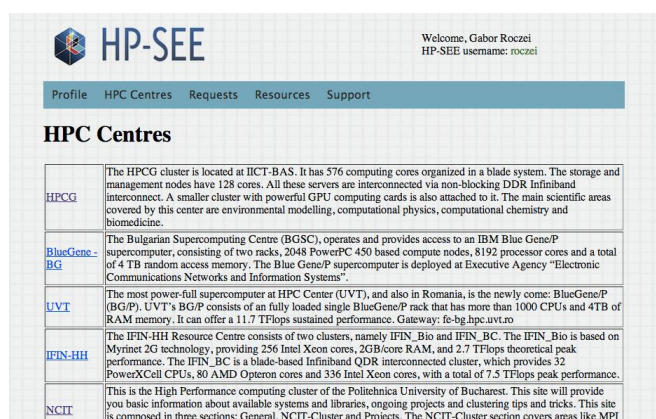
The screenshot shows the HP-SEE registration page. At the top, it says 'HP-SEE' and 'Welcome, Dobreva M. Georgiya'. Below this, it states 'You are not registered within the HP-SEE Portal. Please fill this form to register.' The form includes fields for 'Name', 'Surname', 'Country', 'City', 'Telephone', 'Email', and 'Organization'. Under the 'Applications' section, there are several checkboxes for different projects: 'CompPlex VBC', 'LabPlex VBC', and 'CompPlex VBC'. At the bottom, there is a 'Register' button.

Figure 32 - Registration

The user can browse the available HPC centres while waiting for approval from the HP-SEE ARC. If the user request has been approved by the HP-SEE ARC then he will be notified by e-mail.

User need to do these steps to request HPC computing resources:

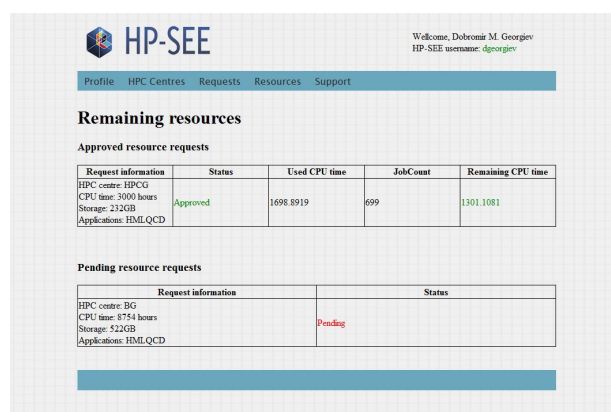
- Choose an HPC Center
- Download it's request form
- Fill in the requested information
- Scan a signed copy of it
- Upload it via the Resource Management System
- Fill in the upload form
- Upload it



Centre	Description
HPCG	The HPCG cluster is located at ICT-BAS. It has 576 computing cores organized in a blade system. The storage and management nodes have 128 cores. All these servers are interconnected via non-blocking DDR Infiniband interconnect. A smaller cluster with powerful GPU computing cards is also attached to it. The main scientific areas covered by this center are environmental modelling, computational physics, computational chemistry and biomedicine.
BlueGene-BG	The Bulgarian Supercomputing Centre (BGSC), operates and provides access to an IBM Blue Gene/P supercomputer, consisting of two racks, 2048 PowerPC 450 based compute nodes, 8192 processor cores and a total of 4 TB random access memory. The Blue Gene/P supercomputer is deployed at Executive Agency "Electronic Communications Networks and Information Systems".
UVT	The most power-full supercomputer at HPC Center (UVT), and also in Romania, is the newly come: BlueGene/P (BG/P). UVT's BG/P consists of an fully loaded single BlueGene/P rack that has more than 1000 CPUs and 4TB of RAM memory. It can offer a 11.7 TFlops sustained performance. Gateway: fe-bg.hpc.uvt.ro
IFIN-HH	The IFIN-HH Resource Centre consists of two clusters, namely IFIN_Bio and IFIN_BC. The IFIN_Bio is based on Myrinet 2G technology, providing 256 Intel Xeon cores, 2GB/core RAM, and 2.7 TFlops theoretical peak performance. The IFIN_BC is a blade-based Infiniband QDR interconnected cluster, which provides 32 PowerXCell CPUs, 80 AMD Opteron cores and 336 Intel Xeon cores, with a total of 7.5 TFlops peak performance.
NCIT	This is the High Performance computing cluster of the Politehnica University of Bucharest. This site will provide you basic information about available systems and libraries, ongoing projects and clustering tips and tricks. This site is composed in three sections: General, NCIT-Cluster and Projects. The NCIT-Cluster section covers areas like MPI

Figure 33 – HPC Centres

The Resource management system provides information about requests as well as statistics for the used CPU time and job count on different HPC centers.



Request information	Status	Used CPU time	JobCount	Remaining CPU time
HPC centre: HPCG CPU time: 3000 hours Storage: 232GB Applications: HML,QCD	Approved	1698.8919	699	1301.1081

Request information	Status
HPC centre: BG CPU time: 8754 hours Storage: 522GB Applications: HML,QCD	Pending

Figure 34 – Statistics for the used CPU time

4.1.2. LDAP

The authentication and authorization of supercomputer users is significantly different from similar solutions applied in grids. While the former often requires local personalized user credentials as well as local user access to at least the HPC front-end nodes, the later uses temporary local credentials assigned to individual user jobs rather than to individual users. Furthermore in the grid case the users are authenticated and authorized by the grid middleware (i.e. in the application level), rather than the lower, say operating system level, software layers.

When creating a structured HPC network an appropriate convergence between the two are needed combining the advantages of both. The classic way of having OS-level user credentials on HPC facilities is to use centralized, yet fail-safe LDAP databases to store user and group parameters and expose them on all HPC elements: compute, front-end and storage elements uniformly. Even though it works quite well in any local configuration, in the everyday practice it is difficult to share multiple LDAP servers administered over multiple organizations. One fallback strategy might be to build up a directory service, a hierarchically built LDAP database that has multiple subtrees and each subtree belongs to different organizations.

Figure 35 shows the LDAP topology of HP-SEE. There is a central LDAP, which content each HPC center and user. The HPC centres also have local LDAP server. The data synchronization between the central LDAP and local LDAP is done by a script. This script do the mapping, it is essential for example if the uidNumber need to be changed. The users are authenticated from this local LDAP service. It is important that the userid need to be extended with a "see-" prefix because it must be unique in each HPC center's LDAP.

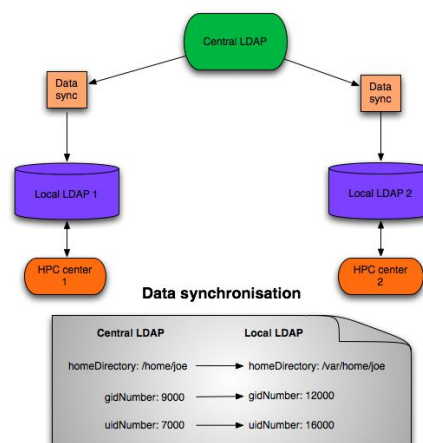


Figure 35 – LDAP topology

Central LDAP server has been installed on this machine: *see-ldap.grid.niif.hu*. It can be accessed over LDAPS. The server's certificate issued by TERENA CA.

Authorization to a HPC centre:

```
dn: cn=fep.grid.pub.ro,ou=hpc-groups,o=romania,dc=hp-see,dc=eu
objectClass: groupOfUniqueNames
cn: fep.grid.pub.ro
uniqueMember: uid=see-martin,ou=users,dc=hp-see,dc=eu
uniqueMember: uid=see-joe,ou=users,dc=hp-see,dc=eu
```

If we would like to authorize a user then we need to add the user's DN to the required HPC centre's tree.

4.2. HP-SEE common environment

The Environment Modules package [11] provides for the dynamic modification of a user's environment via modulefiles. Each modulefile contains the information needed to configure the shell for an application. Once the Modules package is initialized, the environment can be modified on a per-module basis using the module command which interprets modulefiles. Typically modulefiles instruct the module command to alter or set shell environment variables such as PATH, MANPATH, etc. modulefiles may be shared by many users on a system and users may have their own collection to supplement or replace the shared modulefiles.

HP-SEE common environment (HCE) has been created for the HP-SEE centres which is using this module framework. The HCE can be downloaded from here:

<https://github.com/HP-SEE/hce>

These modules have been created for the HP-SEE minimal software stack and recommended software stack components which has been installed on the HP-SEE centres. Figure 36 shows an example for HCE usage.

Figure 36 – HP-SEE module system

```
[login.budapest:~]$ module load hce
[login.budapest:~]$ module list
Currently Loaded Modulefiles:
  1) jdk/1.7.0_13
  2) open64/4.2.4
  3) openmpi/1.4.3
  4) blacs/1.1-gnu-openmpi
  5) gotoblas2/1.13-gnu
  6) fftw/3.3.3-gnu
  7) scalapack/2.0.2-openmpi-1.6.3
  8) atlas/3.10.1-gnu
  9) acml/5.1.0
 10) mpiblast/1.6.0-openmpi-gnu
 11) octave/3.0.5
 12) gromacs/4.5.5
 13) namd/2.9-smp
 14) vmd/1.9.1
 15) root/5.34-gnu
 16) arc/1.1.1
 17) hce
[login.budapest:~]$ module help arc

----- Module Specific Help for 'arc/1.1.1' -----

Application : ARC
Description : ARC middleware 1.1.1
License      : Apache License Version 2.0
URL          : http://www.nordugrid.org/arc
[login.budapest:~]$
```

4.3. Software monitoring with Nagios

The installed softwares are monitored via Nagios, which is a free available, open source monitoring system. This is perfectly suitable for this purpose because we can easily write a test in any kind of script languages (Perl, shell script, Python, etc.). The return value of the test script will be used to check the test result. We created such tests for the minimal and recommended software stacks. These test scripts can be downloaded from here:

<https://github.com/HP-SEE/hce/tree/master/nagios>

We are recommending the Nagios Remote Plugin Executor (NRPE) [12] because this addon is designed to allow you to execute Nagios plugins on remote Linux/Unix machines. The main reason for doing this is to allow Nagios to monitor "local" resources (like CPU load, memory usage, software version, etc.) on remote machines. Since these public resources are not usually exposed to external machines, an agent like NRPE must be installed on the remote Linux/Unix machines.

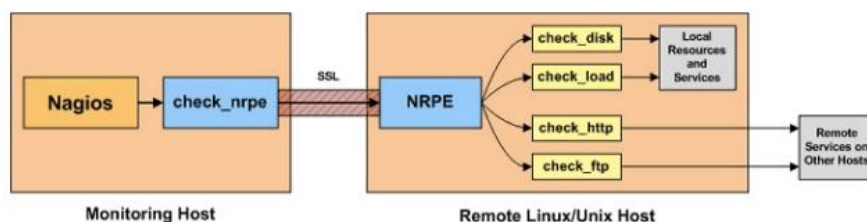


Figure 37 - Nagios Remote Plugin Executor

HP-SEE NRPE command names are described in this file:

<https://github.com/HP-SEE/hce/blob/master/nagios/etc/nagios.cfg>

Figure 38 - Software monitoring with Nagios

Service	Status	Last Update	Response Time	Details
perl	OK	02-21-2013 22:49:12	3d 4h 0m 0s	1/3 OK - This is perl, v5.8.8 built for x86_64-linux-thread-multi Copyright 1987-2006, Larry Wall Perl may be copied only under the terms of either the Artistic License or the GNU General Public License, which may be found in the Perl 5 source kit. Complete documentation for Perl, including FAQ lists, should be found on this system using "man perl" or "perldoc perl". If you have access to the Internet, point your browser at http://www.perl.org/, the Perl Home Page.
python	OK	02-21-2013 22:50:13	3d 3h 58m 29s	1/3 OK - GNU GPREF version: 2.17
root	OK	02-21-2013 22:50:44	3d 3h 58m 28s	1/3 OK - ROOT version: root5.34-gnu
scalapack	OK	02-21-2013 22:51:15	3d 3h 57m 57s	1/3 OK - SCALAPACK version: scalapack2.0.2-gnu
lsash	OK	02-21-2013 22:51:48	3d 3h 57m 25s	1/3 OK - lsash 8.14.00 (Astron) 2005-03-25 (x86_64-unknown-linux) options wide.nis.d.a.kan.am.rh.color.flec
vmd	OK	02-21-2013 22:42:16	3d 3h 56m 56s	1/3 OK - VMD version: vmd1.9.1
nslookup	LOOKUP	02-21-2013 22:51:47	20d 21h 20m 32s	1/3 DNS OK: 0.014 seconds response time: nslookup returns 194.102.58.173
ping	OK	02-21-2013 22:51:18	0d 12h 40m 59s	1/3 PING OK - Packet loss = 0%, RTA = 21.10 ms
ssh	OK	02-21-2013 22:51:49	1d 5h 34m 28s	1/3 SSH OK - xxxxxxxx (protocol 2.0)
ldap	OK	02-21-2013 22:51:19	1d 11h 17m 58s	1/3 LDAP OK - 0.133 seconds response time
blat	OK	02-21-2013 22:51:43	14d 9h 51m 56s	1/3 BLAT OK
fftw3	OK	02-21-2013 22:51:13	14d 9h 48m 58s	1/3 FFTW3 OK
gcc	WARNING	02-21-2013 22:51:43	3d 4h 17m 56s	3/3 GCC WARNING - gcc version detected (4.4.6) is different than the expected (4.1.2)
gdb	WARNING	02-21-2013 22:51:14	3d 4h 17m 56s	3/3 GDB WARNING - gdb version detected (7.2.56.e6) is different than the expected (7.0.1.23.e6.5.1)
gprof	WARNING	02-21-2013 22:51:45	3d 4h 17m 56s	3/3 GPROF WARNING - gprof version detected (version) is different than the expected (2.17.50.0.6-14.e6)
icc	WARNING	02-21-2013 22:51:18	3d 4h 18m 55s	3/3 ICC WARNING - icc version detected (13.0.1) is different than the expected (13.0.0)
lapack	OK	02-21-2013 22:51:47	13d 7h 14m 55s	1/3 LAPACK OK
nslookup	OK	02-21-2013 22:51:18	20d 21h 16m 32s	1/3 DNS OK: 0.045 seconds response time: ui.ipb.ac.rs returns 147.91.84.247
namd2	OK	02-21-2013 22:51:48	13d 21h 35m 56s	1/3 NAMD2 OK - info: NAMD 2.8b1 for Linuxx86_64
oprofile	WARNING	02-21-2013 22:51:19	3d 4h 17m 56s	3/3 OPROFILE WARNING - oprofile version detected (0.9.7) is different than the expected (0.9.4)
ping	OK	02-21-2013 22:51:50	5d 8h 39m 53s	1/3 PING OK - Packet loss = 0%, RTA = 0.28 ms
ssh	OK	02-21-2013 22:51:20	16d 8h 37m 57s	1/3 SSH OK - OpenSSH_4.3 (protocol 2.0)
vina	OK	02-21-2013 22:51:51	16d 8h 37m 56s	1/3 VINA OK - AutoDock Vina 1.1.1 (Apr 20, 2010)
vtune	CRITICAL	02-21-2013 22:51:22	3d 4h 17m 56s	3/3 VTUNE CRITICAL - amplexo-c command not found
vserv	LOOKUP	02-21-2013 22:51:44	3d 4h 4m 5s	1/3 DNS OK: 0.010 seconds response time: vserv.szegep.hpc.nif.hu returns 193.224.66.132
ping	OK	02-21-2013 22:51:17	1d 5h 45m 0s	1/3 PING OK - Packet loss = 0%, RTA = 7.57 ms
ssh	OK	02-21-2013 22:51:48	1d 5h 34m 31s	1/3 SSH OK - OpenSSH_4.3 (protocol 2.0)
cuda	OK	02-21-2013 22:45:10	1d 11h 7m 7s	1/3 OK - nvcc: NVIDIA (R) Cuda compiler driver Copyright (c) 2005-2012 NVIDIA Corporation Built on Fri_Sep_21_17:28:58_PDT_2012 Cuda compilation tools, release 5.0, V0.2.1221

Here is the monitoring link which can be opened in the browser:

<https://hpseemon.ipb.ac.rs/nagios/cgi-bin/status.cgi?host=all>

4.4. Usage of gUSE portal

Bioinformatics portal developed in HP-SEE project allows using 2 scientific applications, an application for Deep sequencing for short fragment alignment (called DeepAligner) and application called In-silico Disease Gene Mapper. Both are represented by a workflow structure shown in Figure 1

4.4.1. Availability

4.4.1.1 *DiseaseGeneMapper*

http://ls-hpsee.nik.uni-obuda.hu:8080/liferay-portal-6.0.5/en_GB/web/guest/diseasegenemapper

4.4.1.2 *Deep Aligner*

http://ls-hpsee.nik.uni-obuda.hu:8080/liferay-portal-6.0.5/en_GB/web/guest/deepaligner

4.4.2. Requirements

There are two requirements the users have to comply in order to use the DGM and DA portlets

- appropriate group membership on the ls-hpsee portal – to require the membership, please contact the portal administrator: balasko at sztaki dot hu
- valid, downloaded and associated certificate for the NIIF supercomputers. To apply for such a certificate please contact the NIIF administrator: roczei at niif dot hu

4.4.3. Installation (for portal administrators)

- Import the workflow, export it as an application
- Install the portlet (for details on how to install the portlet, please refer to the ASM User Manual).

The nodes represent jobs are preconfigured to be submitted to computational resources of ARC middleware. The real computational resource on where the job will be executed is selected by ARC's client-side brokering mechanism. The concrete job submission and managing its life cycle fall within the core system's cognizance. This part of the life-cycle is being modified by allowing the users to specify their requirements for the computational resource by modifying the JSDL description of a job. Then a planned development aims to be able to change the strategy of the brokering according to the claims of the portal.

One interface was developed per each application in order to import, parameterize and execute them while the details of the configuration and the execution are totally hidden from the users.

These portlets are based on Application Specific Module(ASM) provided by core WS-PGRADE system which hides all internal communication among gUSE components and offers an easy-to-use Java API that enables the communities to develop a clear and focused interface exploiting the features of the core portal.

Users of WS-PGRADE define their applications as workflows. They can share their applications among each other by exporting them to the repository. Following this way, other users can import such applications and execute or modify them in their user space. Concept of ASM that solves problem of customization is based on this scenario. In this case two different user roles can be defined: Application Developer, who created and shared the application, and the End Users, who import and execute it. Or analogously, Application Developers are administrators, or scientists with developer skills; and the End Users are those, who just use their product.

Using this solution development of science gateways technically means development of web applications that produce a transparent interface, handle the interaction coming from the users, and, according to them, call the internal components. This calling mechanism is simplified by ASM as it hides complex algorithms and web-service calls and provides these functionalities as simple Java methods covering the whole life-cycle of the workflow in aspect of End Users. ASM contains method to get the list of Application Developers, the applications shared by a particular Developer, and to import a particular workflow. To guarantee that the workflow will do the same that the Developer wanted originally, End Users have restricted possibilities to manipulate the workflow. Especially they cannot modify the workflow structure by adding or removing a node, or they cannot replace the program placed in a node, but they can upload and attach a new input file, set or modify command line arguments, etc.

Finally End Users can manage the workflows; they can submit them to a distributed resource, check the execution, download their outputs or delete them. gUSE can be extended with an interface that hides the complexity of the inner abstraction levels, and inner callings of different core services. Without this component, one or more difficult web-service callings should be constructed each time when a customized portlet should get or pass information from/to the portal. In order to avoid this complexity Application Specific Module(ASM) API covers all of these internal information accesses by a simple call of well-parameterized JAVA methods.

Figure 39 shows the main concept of a science gateway based on ASM, where ASM-based user interface represents the interface developed especially for an workflow constructed via the traditional WS-PGRADE interface.

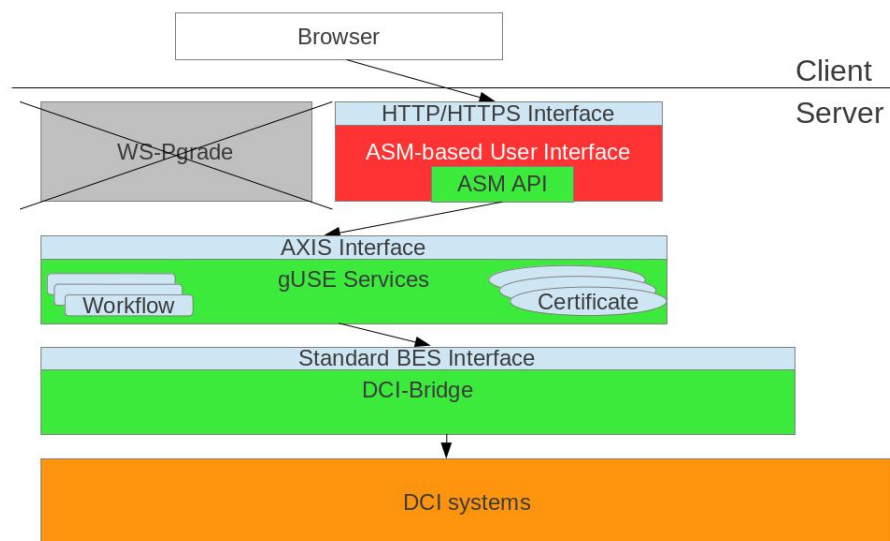


Figure 39 - Concept of ASM

4.4.4. Using the portlets

The first step of using the applications is to download a valid certificate and associate it to the NIIF supercomputers

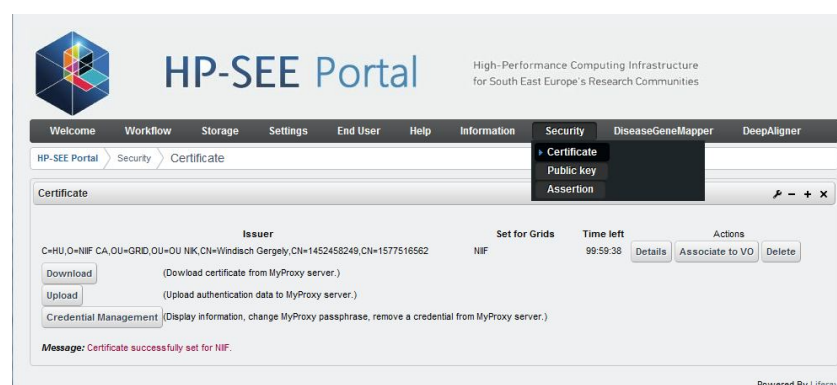


Figure 40 - Upload certificate to use the portlets

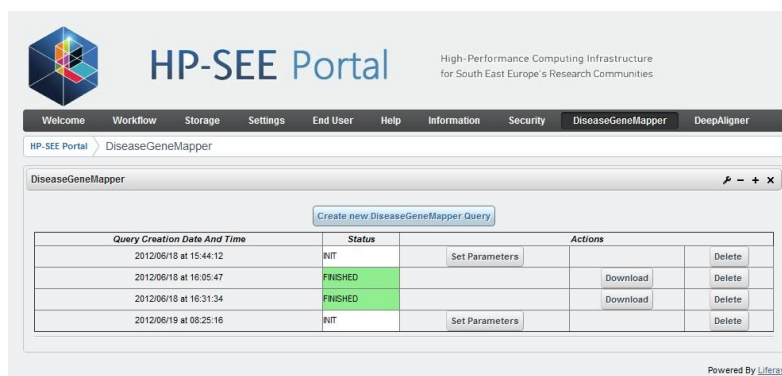


Figure 41 - DiseaseGeneMapper main window

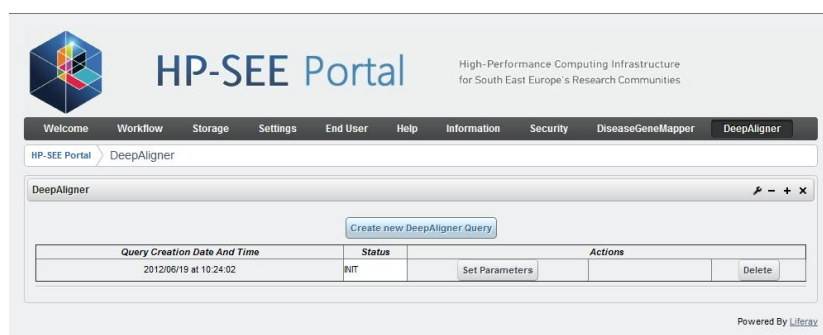


Figure 42 - DeepAligner main window

4.4.5. Creating a new DiseaseGeneMapper query

The first thing that has to be done is clicking on the button **Create new DiseaseGeneMapper Query**. It will import the latest version of the DiseaseGeneMapper workflow (see Appendix, Naming convention). A job is created from the workflow, and it shows up in the main table.

4.4.6. DiseaseGeneMapper job lifecycle

- **INIT**: first state – no parameters are set
- **RUNNING**: parameters are set and the job has been submitted
- **FINISHED**: job finished running – results can be downloaded
- **ERROR**: job finished running – there has been an error

At first the job will be in the **INIT** state. It means that the workflow has been imported, but the parameters have not been set and the job has not been started yet. At this point the user has to press the **Set Parameters** button, and specify all the parameters. All the fields are filled out with example values.

4.4.7. Submitting jobs

Clicking on the submit button submits the job to the ARC middleware which forwards it to the NIIF supercomputers.

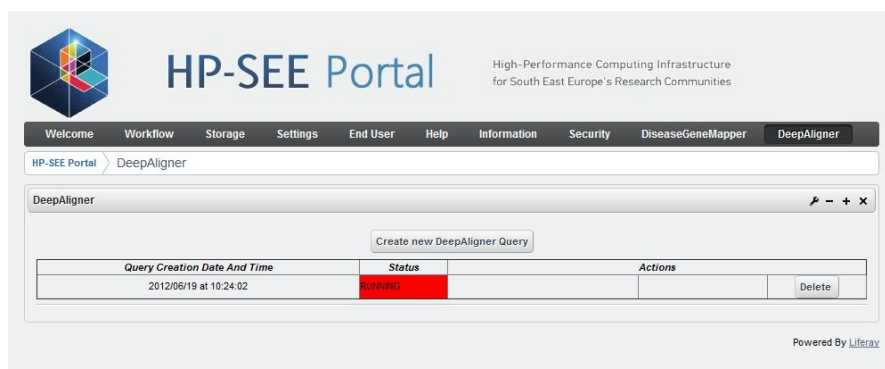


Figure 43 - Executing DeepAligner query

When the job is submitted, the Status becomes RUNNING. The running time depends greatly on the number of input sequences, the size of the blast database and the load of the supercomputer the job is executed on. If the supercomputers are occupied it is not uncommon for the jobs to be on the queue for several hours. Once the jobs are executed they should finish in a matter of minutes.

After the jobs are completed, the status becomes either FINISHED or ERROR. FINISHED means that the workflow ran without any problems – but it does not necessarily mean that the results are calculated as expected. The program handles unexpected situations (like blast database missing, unexpected input file format etc.) quite well, so even if there was an error, the status will be FINISHED, and the error message will be inside the resulting file. If the status is ERROR, it usually means that there was some problem outside the scope of the workflow (like mpiBlast running out of memory or a synchronization problem between the nodes). In rare cases it is even possible that even though the status is ERROR, it did not affect the correct results.

4.4.8. Downloading results

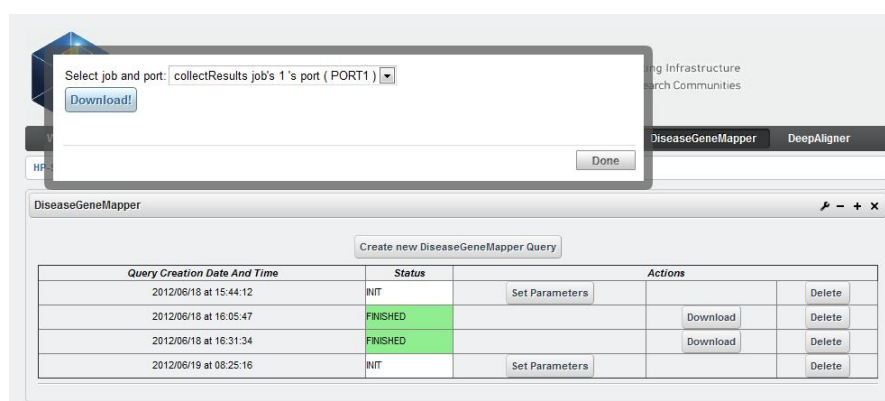


Figure 44 - Downloading results from the applications

When the job finished running (the status is either FINISHED or ERROR), the results can be downloaded by clicking on the Download button. The resulting file is a tar.gz called blastOutput.tar.gz, and it holds either the blast results for all the input sequences or the error message.

4.4.9. Parameters

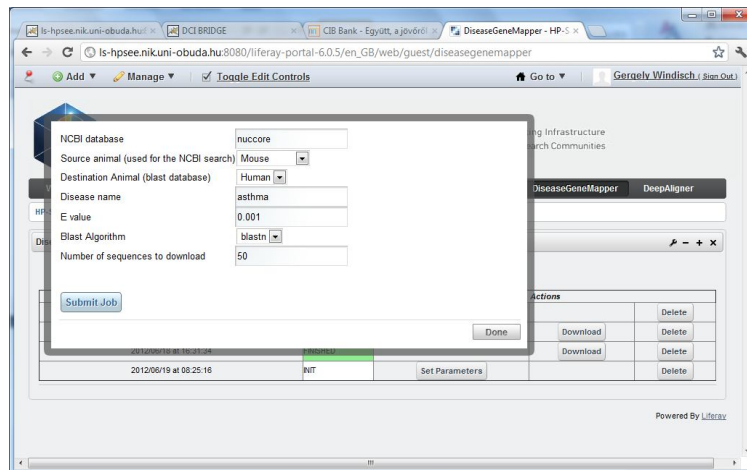


Figure 45 - Setting the parameters for Disease Gene Mapper

- NCBI database: the name of the NCBI database from which the sequences are going to be downloaded.
- Source animal: The downloaded sequences are filtered, only the ones belonging to the given source animal are stored and processed
- Destination animal: Database for the blast search
- Disease name: the name of the disease that the user is looking for in the NCBI database
- E value: Expectation value for the BLAST search. The larger the number the more inaccurate the results are
- Blast algorithm: the algorithm blast uses
- Number of sequences to download: the NCBI database usually holds lots of sequences associated to a disease. The number the user sets here will tell the program how many of these sequences to process. There is no maximum number here, the program will automatically adjust it to number of NCBI results.

4.4.10. DeepAligner

For using the portlet, see DiseaseGeneMapper section.

4.4.10.1 Parameters

Compressed file containing the sequences (accepted types: tar.gz, rar, zip) inputfiles.tar.gz

Blast database

E value

Query Creation Date And Time	Status	Actions
2012/06/19 at 10:24:02	INIT	<input type="button" value="Set Parameters"/> <input type="button" value="Delete"/>

Powered By [Liferav](#)

Figure 46 - Setting the parameters for DeepAligner

There are only three parameters for the DeepAligner portlet

- Compressed file: the most important parameter, this file holds the sequences that are to be searched in against the selected database. The compressed file can either be a tar.gz, zip or rar. The sequences inside the compressed file are store as one sequence per file. The names of the sequence files can be anything, but they will be renamed to output_x, where x is an integer starting from 0
- Blast database: the name of the database against which the sequences will be BLASTed.

The databases currently supported are

- est_human
 - drosoph.nt
 - est_mouse
 - est_others
 - env_nt
 - gss
 - htgs
 - human_genomic
 - igSeqNt
 - nt
 - other_genomic
 - patnt
 - sts
- E value: Expectation value for the BLAST search. The larger the number the more inaccurate the results are

4.4.11. Appendix

Special considerations for the workflows

Naming convention:

Both Disease Gene Mapper and Deep Aligner portals require a special naming convention when importing workflows. DGM loads workflows that are called DiseaseGeneMapper_vX, Deep Aligner expects workflows that are called DeepAligner_vX. The X in both cases mean an integer, which is the version number of

the workflow. The portlets open the version with the highest version number. The names are not case sensitive.

Port values:

Both portlets assign values to the ports using the API call `ASMService.setInputText`. This call only has an effect if the given port in the workflow's configuration is set to be a value, but the value field is left empty. If a value is specified for the port, `setInputText` will not overwrite it.

4.5. HP-SEE helpdesk system

HP-SEE operates a Request Tracker (RT) based [<http://www.bestpractical.com/rt/>] ticketing systems (Helpdesk) that is used for requesting some of the HP-SEE services, reporting problems and tracking requests and reports status. The Helpdesk is available to users either via mail (support@helpdesk.hp-see.eu being the main contact point for the helpdesk) or via a web interface (<https://helpdesk.hp-see.eu>) for users with a valid X509 certificate. The HP-SEE helpdesk is used mainly for two purposes. User/application porting support, as well as for operational purposes. More details in the structure of the support units and its usage have been given in the relevant WP5 deliverables.

In terms of interoperability and mainly interoperation the helpdesk has been designed and implemented in a way that information flow can be transferred to operators or helpdesk systems of each individual HPC centre that participates in the HP-SEE infrastructure. This has been implemented by creating specific queues for each HPC centre. Each queue is staffed with members of the support teams of the HPC centres as well as mail aliases for their "local" Ticketing Systems/Helpdesks. Therefore both users and infrastructure operators are provided with a single access point for requesting feature or report on problems, that facilitates information exchange with the relevant support teams. In cases where users or operators cannot identify the specific support unit via the HP-SEE helpdesk they can submit their request/report to the generic support queue which is monitored by the project's operators that ensure timely forwarding of any request to the appropriate support unit.

5. Conclusions

The previous deliverable (D8.2) has defined a number of actions regarding interoperability and scalability. Table 1 gives a summary of the HP-SEE applications, which have implemented some of the defined scalability improvement actions of D8.2. Most of them (HC-MD-QM-CS, GIM, NUQG, SFHG, FMD-PA) are using MPI and OpenMP parallelization technologies. GENETATOMICS, MSBP, CFDOF, DeepAligner, DiseaseGeneMapper, CompChem is using only MPI and AMR_PAR is using only OpenMP. The AMR_PAR application is unique because it has been ported from Windows to Linux and this gives a very good overview of porting steps. DNAMA application has been tested in hybrid mode too, which combines MPI and Pthreads. Their scalability results were weaker since Pthreads can't give enough speedup with dataset used in the benchmark. Several MPI implementations have been tested by the GENETATOMICS application. The conclusion was that the version of MPI does not effect the scalability. Another interesting result by GENETATOMICS application is that hyper threading should be used where available. Various compilers have been tested by the SET application and the result was that Intel compiler provides the best results on Intel Xeon cluster. For the IBM Blue Gene/P architecture the obvious choice was the IBM XL compiler suite since it has advantage versus the GNU Compiler Collection in that it supports the double-hammer mode of the CPUs. Other interesting result is that SET application has 30% improvement when hyper threading is turned on. SET application has been tested on NVIDIA GPU based systems too and it gives better performance using the newer M2090 cards versus the old GTX295, which was to be expected because the integer performance of the GTX 295 is comparable to that of M2090, but the floating performance of the GTX is many times smaller.

Another aim of this deliverable was to improve the interoperability between the HPC centres and improve transparent access to this integrated infrastructure. HP-SEE common environment has been defined which includes HP-SEE minimal software stack, recommended software stack, and the module system. We have taken into account the user needs and the PRACE [10], DEISA [13] software stacks also. The defined software stacks helped us to improve the harmonization level between the HP-SEE centres: Table 22 and Table 23 give a summary of this. DeepAligner and DiseaseGeneMapper are used to demonstrate the Bioinformatics Portal, which has been integrated with the Hungarian HPC centres. This can be easily extended with other HPC centres too if needed. Finally, the resource management portal helps the user to request new HPC account in a common way; and users can also request support on the HP-SEE helpdesk portal.