SEVENTH FRAMEWORK PROGRAMME
Research Infrastructures

INFRA-2010-2.3.1 – First Implementation Phase of the European High Performance Computing (HPC) service PRACE

PRACE-1IP

PRACE First Implementation Project

Grant Agreement Number: RI-261557

D7.6
Efficient Handling of Petascale Data

Final

Version:	0.5	
Author(s):	Mohammad Jowkar,	Barcelona Supercomputing Center
	Vit Vondrak,	Technical University of Ostrava
	Bjørn Lindi,	Norwegian University of Science and Technology
	Olivier Rouchon,	Centre Informatique National de l'Enseignement Supérieur
	Marzia Rivi,	CINECA Interuniversitary Consortium
Date:	23.12.2011	

Project and Deliverable Information Sheet

PRACE Project	**Project Ref. №: RI-261557**	
	Project Title: PRACE First Implementation Project	
	Project Web Site: http://www.prace-project.eu	
	Deliverable ID: < **D7.6**>	
	Deliverable Nature: <DOC_TYPE: Report / Other>	
	Deliverable Level: PU *	**Contractual Date of Delivery:** 31 / December / 2011
		Actual Date of Delivery: 31 / December / 2011
	EC Project Officer: Bernhard Fabianek	

* - The dissemination level are indicated as follows: **PU** – Public, **PP** – Restricted to other participants (including the Commission Services), **RE** – Restricted to a group specified by the consortium (including the Commission Services). **CO** – Confidential, only for members of the consortium (including the Commission Services).

Document Control Sheet

Document	**Title:** <**Efficient Handling of Petascale Data**>	
	ID: <**D7.6**>	
	Version: <0.5 >	**Status:** Final
	Available at: http://www.prace-project.eu	
	Software Tool: Microsoft Word 2007	
	File(s): D7.6.doc	
Authorship	**Written by:**	Mohammad Jowkar (BSC, Spain) Vit Vondrak (VSB, Czech Republic) Bjørn Lindi (NTNU, Norway) Olivier Rouchon (CINES, France) Marzia Rivi (CINECA, Italy)

	Contributors:	Guillaume Houzeaux (BSC, Spain), Hadrien Calmet (BSC, Spain), Jose M. Cela (BSC, Spain), Mariano Vázquez (BSC, Spain), Raul de la Cruz (BSC, Spain), Xavier Saez (BSC, Spain), Ali Haydar Özer (Bogazici, Turkey), Can Ozturan (Bogazici, Turkey), Oğuz Tosun (Bogazici, Turkey), Seren Soner (Bogazici, Turkey), Yusuf Yılmaz (Bogazici, Turkey), Peter Raback (CSC, Finland), Juha Ruokolainen (CSC, Finland), Ales Ronovsky (VSB, Czech Republic), Pavla Kabelikova (VSB, Czech Republic), Tomas Kozubek (VSB, Czech Republic), Dalibor Lukas (VSB, Czech Republic), Jan Zapletal (VSB, Czech Republic), Charles Moulinec (STFC, England), Andrew G. Sunderland (STFC, England), Yvan Fournier (EDF, France), Ata Turk (Bilkent, Turkey), Cevdet Aykanat (Bilkent, Turkey), Vehbi Gunduz Demirc (Bilkent, Turkey), Sebastian von Alfthan (FMI, Finland), Ilja Honkonen (FMI, Finland), Peicho Petkov (NCSA, Bulgaria), Petko Petkov (NCSA, Bulgaria), Georgi Vayssilov (NCSA, Bulgaria), Stoyan Markov (NCSA, Bulgaria), Valentin Pavlov (NCSA, Bulgaria), Anton Tomov (NCSA, Bulgaria), Vesselin Slavchev (NCSA, Bulgaria), Nina Ilieva (NCSA, Bulgaria), Dimitar Dimitrov (NCSA, Bulgaria), Kiril Alexiev (NCSA, Bulgaria), Huub Stoffers (SARA, Netherlands), Paul Melis (SARA, Netherlands), Mark van de Sanden (SARA, Netherlands), John Donners (SARA, Netherlands), Jan Christian Meyer (NTNU, Norway), Jørn Amundsen (NTNU, Norway), Henrik Nagel (NTNU, Norway), Jörg Herzer (HLRS, Gemany), Orlando Rivera (LRZ, Germany), Peter Nash (ICHEC, Ireland), Philippe Prat (CINES, France), Mathieu Cloirec (CINES, France), Florent Marceteau (CINES, France), Philippe Wautelet (IDRIS-CNRS, France), Pierre Kestener (CEA Saclay, France), Giusy Muscianisi (CINECA, Italy), Luigi Calori (CINECA, Italy), Vladimir Slavnic (IPB, Serbia), George Tsouloupas (CaSToRC, Cyprus), Nick Sinanis (CaSToRC, Cyprus), Johan Raber (SNIC, Sweden), Per Lundqvist (SNIC, Sweden), Tom Lanborg (SNIC, Sweden), Bengt Persson (SNIC, Sweden), Torben Rasmussen (SNIC, Sweden)

	Reviewed by:	Paulo Abreu, IST; D. Erwin, Jülich Organisation
	Approved by:	MB/TB

Document Status Sheet

Version	Date	Status	Comments
0.1	22/07/2011	Draft	Initial skeleton draft
0.2	01/11/2011	Draft	First rough draft
0.3	05/12/2011	Draft	First complete draft version
0.4	09/12/2011	Draft	Final draft ready for review
0.5	23/12/2011	Draft	Updated according to comments from reviewers

Document Keywords

Keywords:	PRACE, HPC, Research Infrastructure, Input/Output, Parallel I/O, Hierarchical I/O, Parallel Pre-Processing, Long-Term Preservation of Data, Post-Processing, Visualisation

Table of Contents

List of Figures

List of Tables

References and Applicable Documents

[1] Z. Dostal, T. Kozubek, A. Markopoulos, M. Mensik, "Cholesky decdomposition and a generalized inverse of the stiffness matrix of a floating structure with known null space", Applied Mathematics and Computation 2011; 217:6067–6077.
[2] T. Brzobohaty, Z. Dostal, P. Kovar, T. Kozubek, A. Markopoulos, "Cholesky decomposition with fixing nodes to stable evaluation of a generalized inverse of the stiffness matrix of a floating structure", IJNME, DOI: 10.1002/nme.3187.
[3] T. Kozubek, V. Vondrak, M. Mensik, D. Horak, Z. Dostal, V. Hapla, P. Kabelikova, M. Cermak, "Total FETI domain decomposition method and its massively parallel implementation", submitted.
[4] http://www.prace-project.eu
[5] PRACE-1IP deliverable D7.1.2 "Report on applications enabling for capability science"
[6] PRACE-1IP deliverable D7.2.2 "Report on collaboration with communities"
[7] C. Farhat, F. X. Roux, "A method of finite element tearing and interconnecting and its parallel solution algorithm", IJNME 1991; 32:1205-1227
[8] Z. Dostal, D. Horak, R. Kucera, „Total FETI - an easier implementable variant of the FETI method for numerical solution of elliptic PDE", Commun. Numer. Methods Eng. 2006; 22:1155–1162

List of Acronyms and Abbreviations

AAA	Authorization, Authentication, Accounting.
ACF	Advanced Computing Facility
ADP	Average Dissipated Power
AMD	Advanced Micro Devices
APGAS	Asynchronous PGAS (language)
API	Application Programming Interface
APML	Advanced Platform Management Link (AMD)
ASIC	Application-Specific Integrated Circuit
ATI	Array Technologies Incorporated (AMD)
BAdW	Bayerischen Akademie der Wissenschaften (Germany)
BCO	Benchmark Code Owner
BLAS	Basic Linear Algebra Subprograms
BSC	Barcelona Supercomputing Center (Spain)
CAF	Co-Array Fortran
CAL	Compute Abstraction Layer
CCE	Cray Compiler Environment
ccNUMA	cache coherent NUMA
CEA	Commissariat à l'Energie Atomique (represented in PRACE by GENCI, France)
CGS	Classical Gram-Schmidt
CGSr	Classical Gram-Schmidt with re-orthogonalisation
CINECA	Consorzio Interuniversitario, the largest Italian computing centre (Italy)
CINES	Centre Informatique National de l'Enseignement Supérieur (represented in PRACE by GENCI, France)
CLE	Cray Linux Environment
CPU	Central Processing Unit
CSC	Finnish IT Centre for Science (Finland)
CSCS	The Swiss National Supercomputing Centre (represented in PRACE by ETHZ, Switzerland)
CSR	Compressed Sparse Row (for a sparse matrix)
CUDA	Compute Unified Device Architecture (NVIDIA)
DARPA	Defense Advanced Research Projects Agency
DDN	DataDirect Networks
DDR	Double Data Rate
DEISA	Distributed European Infrastructure for Supercomputing Applications. EU project by leading national HPC centres.
DGEMM	Double precision General Matrix Multiply
DIMM	Dual Inline Memory Module
DMA	Direct Memory Access
DNA	DeoxyriboNucleic Acid
DP	Double Precision, usually 64-bit floating point numbers
DRAM	Dynamic Random Access memory
EC	European Community
EESI	European Exascale Software Initiative
EoI	Expression of Interest
EP	Efficient Performance, e.g., Nehalem-EP (Intel)

EPCC	Edinburg Parallel Computing Centre (represented in PRACE by EPSRC, United Kingdom)
EPSRC	The Engineering and Physical Sciences Research Council (United Kingdom)
eQPACE	extended QPACE, name of the FZJ WP8 prototype
ETHZ	Eidgenössische Technische Hochschule Zuerich, ETH Zurich (Switzerland)
ESFRI	European Strategy Forum on Research Infrastructures; created roadmap for pan-European Research Infrastructure.
EX	Expandable, e.g., Nehalem-EX (Intel)
FC	Fiber Channel
FFT	Fast Fourier Transform
FHPCA	FPGA HPC Alliance
FP	Floating-Point
FPGA	Field Programmable Gate Array
FPU	Floating-Point Unit
FZJ	Forschungszentrum Jülich (Germany)
GASNet	Global Address Space Networking
GB	Giga (= 2^{30} ~ 10^9) Bytes (= 8 bits), also GByte
Gb/s	Giga (= 10^9) bits per second, also Gbit/s
GB/s	Giga (= 10^9) Bytes (= 8 bits) per second, also GByte/s
GCS	Gauss Centre for Supercomputing (Germany)
GDDR	Graphic Double Data Rate memory
GÉANT	Collaboration between National Research and Education Networks to build a multi-gigabit pan-European network, managed by DANTE. GÉANT2 is the follow-up as of 2004.
GENCI	Grand Equipement National de Calcul Intensif (France)
GFlop/s	Giga (= 10^9) Floating point operations (usually in 64-bit, i.e. DP) per second, also GF/s
GHz	Giga (= 10^9) Hertz, frequency =10^9 periods or clock cycles per second
GigE	Gigabit Ethernet, also GbE
GLSL	OpenGL Shading Language
GNU	GNU's not Unix, a free OS
GPGPU	General Purpose GPU
GPU	Graphic Processing Unit
GS	Gram-Schmidt
GWU	George Washington University, Washington, D.C. (USA)
HBA	Host Bus Adapter
HCA	Host Channel Adapter
HCE	Harwest Compiling Environment (Ylichron)
HDD	Hard Disk Drive
HE	High Efficiency
HET	High Performance Computing in Europe Taskforce. Taskforce by representatives from European HPC community to shape the European HPC Research Infrastructure. Produced the scientific case and valuable groundwork for the PRACE project.
HMM	Hidden Markov Model
HMPP	Hybrid Multi-core Parallel Programming (CAPS enterprise)
HP	Hewlett-Packard
HPC	High Performance Computing; Computing at a high performance level at any given time; often used synonym with Supercomputing

HPCC	HPC Challenge benchmark, http://icl.cs.utk.edu/hpcc/
HPCS	High Productivity Computing System (a DARPA program)
HPL	High Performance LINPACK
HT	HyperTransport channel (AMD)
HWA	HardWare accelerator
IB	InfiniBand
IBA	IB Architecture
IBM	Formerly known as International Business Machines
ICE	(SGI)
IDRIS	Institut du Développement et des Ressources en Informatique Scientifique (represented in PRACE by GENCI, France)
IEEE	Institute of Electrical and Electronic Engineers
IESP	International Exascale Project
IL	Intermediate Language
IMB	Intel MPI Benchmark
I/O	Input/Output
IOR	Interleaved Or Random
IPMI	Intelligent Platform Management Interface
ISC	International Supercomputing Conference; European equivalent to the US based SC0x conference. Held annually in Germany.
IWC	Inbound Write Controller
JSC	Jülich Supercomputing Centre (FZJ, Germany)
KB	Kilo (= 2^{10} ~10^3) Bytes (= 8 bits), also KByte
KTH	Kungliga Tekniska Högskolan (represented in PRACE by SNIC, Sweden)
LBE	Lattice Boltzmann Equation
LINPACK	Software library for Linear Algebra
LLNL	Laurence Livermore National Laboratory, Livermore, California (USA)
LQCD	Lattice QCD
LRZ	Leibniz Supercomputing Centre (Garching, Germany)
LS	Local Store memory (in a Cell processor)
MB	Mega (= 2^{20} ~ 10^6) Bytes (= 8 bits), also MByte
MB/s	Mega (= 10^6) Bytes (= 8 bits) per second, also MByte/s
MDT	MetaData Target
MFC	Memory Flow Controller
MFlop/s	Mega (= 10^6) Floating point operations (usually in 64-bit, i.e. DP) per second, also MF/s
MGS	Modified Gram-Schmidt
MHz	Mega (= 10^6) Hertz, frequency =10^6 periods or clock cycles per second
MIPS	Originally Microprocessor without Interlocked Pipeline Stages; a RISC processor architecture developed by MIPS Technology
MKL	Math Kernel Library (Intel)
ML	Maximum Likelihood
Mop/s	Mega (= 10^6) operations per second (usually integer or logic operations)
MoU	Memorandum of Understanding.
MPI	Message Passing Interface
MPP	Massively Parallel Processing (or Processor)
MPT	Message Passing Toolkit
MRAM	Magnetoresistive RAM
MTAP	Multi-Threaded Array Processor (ClearSpead-Petapath)
mxm	DP matrix-by-matrix multiplication mod2am of the EuroBen kernels

NAS	Network-Attached Storage
NCF	Netherlands Computing Facilities (Netherlands)
NDA	Non-Disclosure Agreement. Typically signed between vendors and customers working together on products prior to their general availability or announcement.
NoC	Network-on-a-Chip
NFS	Network File System
NIC	Network Interface Controller
NUMA	Non-Uniform Memory Access or Architecture
OpenCL	Open Computing Language
OpenGL	Open Graphic Library
Open MP	Open Multi-Processing
OS	Operating System
OSS	Object Storage Server
OST	Object Storage Target
PCIe	Peripheral Component Interconnect express, also PCI-Express
PCI-X	Peripheral Component Interconnect eXtended
PGAS	Partitioned Global Address Space
PGI	Portland Group, Inc.
pNFS	Parallel Network File System
POSIX	Portable OS Interface for Unix
PPE	PowerPC Processor Element (in a Cell processor)
PRACE	Partnership for Advanced Computing in Europe; Project Acronym
PSNC	Poznan Supercomputing and Networking Centre (Poland)
QCD	Quantum Chromodynamics
QCDOC	Quantum Chromodynamics On a Chip
QDR	Quad Data Rate
QPACE	QCD Parallel Computing on the Cell
QR	QR method or algorithm: a procedure in linear algebra to compute the eigenvalues and eigenvectors of a matrix
RAM	Random Access Memory
RDMA	Remote Data Memory Access
RISC	Reduce Instruction Set Computer
RNG	Random Number Generator
RPM	Revolution per Minute
SAN	Storage Area Network
SARA	Stichting Academisch Rekencentrum Amsterdam (Netherlands)
SAS	Serial Attached SCSI
SATA	Serial Advanced Technology Attachment (bus)
SDK	Software Development Kit
SGEMM	Single precision General Matrix Multiply, subroutine in the BLAS
SGI	Silicon Graphics, Inc.
SHMEM	Share Memory access library (Cray)
SIMD	Single Instruction Multiple Data
SM	Streaming Multiprocessor, also Subnet Manager
SMP	Symmetric MultiProcessing
SNIC	Swedish National Infrastructure for Computing (Sweden)
SP	Single Precision, usually 32-bit floating point numbers
SPE	Synergistic Processing Element (core of Cell processor)
SPH	Smoothed Particle Hydrodynamics
SPU	Synergistic Processor Unit (in each SPE)

SSD	Solid State Disk or Drive
STFC	Science and Technology Facilities Council (represented in PRACE by EPSRC, United Kingdom)
STRATOS	PRACE advisory group for STRAtegic TechnOlogieS
STT	Spin-Torque-Transfer
TARA	Traffic Aware Routing Algorithm
TB	Tera (= 240 ~ 1012) Bytes (= 8 bits), also TByte
TCO	Total Cost of Ownership. Includes the costs (personnel, power, cooling, maintenance, ...) in addition to the purchase cost of a system.
TDP	Thermal Design Power
TFlop/s	Tera (= 1012) Floating-point operations (usually in 64-bit, i.e. DP) per second, also TF/s
Tier-0	Denotes the apex of a conceptual pyramid of HPC systems. In this context the Supercomputing Research Infrastructure would host the Tier-0 systems; national or topical HPC centres would constitute Tier-1
UFM	Unified Fabric Manager (Voltaire)
UNICORE	Uniform Interface to Computing Resources. Grid software for seamless access to distributed resources.
UPC	Unified Parallel C
UV	Ultra Violet (SGI)
VHDL	VHSIC (Very-High Speed Integrated Circuit) Hardware Description Language

Executive Summary

This document summarizes the work done in task 7.6 of work package 7 of the PRACE-1IP project [4]. The main objective of task 7.6 has been to support users, from tasks 7.1 and 7.2, to overcome their data challenges at Petascale. A Petascale supercomputer is composed of a very large number of processors and an even larger number of threads. It is common that many of these processors and threads in a Petascale class simulation need to process a large amount of data, which brings with it a set of challenges. T7.6 has been successful in helping users overcome such challenges, by identifying data experts and gathering them in groups, based on their expertise. The experts have been assigned to supporting users with their data needs, in order for them to be able to scale their applications further. Typically, these groups have been composed of experts from different European HPC centers. This approach has thus led to a closer collaboration among European researchers, as well as to the sharing and transfer of knowledge between them.

The support provided by the experts in T7.6, to overcome the challenges of petascale-class applications data, has been documented in detail in whitepapers and summarized in this document. In total 12 application have been supported, which has resulted in 23 whitepapers, detailing the work which has been done.

Introduction

As HPC applications approach petascaling, the efficient handling of data and I/O becomes increasingly more important. The amount of data that applications must process grows substantially at petascale and can pose a serious bottleneck. Tier-0 petascale systems today have impressive amounts of computational power, which means that data handling can consume more time than the actual computations themselves, if not done in an efficient manner. At a lower scale, not much effort has been put in the efficient handling of data and it is often done sequentially since it does not pose a big bottleneck. However, this is becoming less of an option as we move towards petascale and beyond, where the amount of data to be processed exceeds the memory of a shared memory machines.

Data typically goes through several stages during the execution of an HPC application. In the first stage input data is read and some pre-processing is performed, such as domain decomposition, mesh refinement/partitioning, etc. Afterwards during computations, data has to be exchanged among nodes and results have to be output. When the application terminates the output data then has to be post-processed and visualized for analysis. Finally, if the results have importance, they might have to be stored for long term preservation. For a petascale run, each of the above stages can involve the processing of significant amounts of data.

Overcoming the *data challenge* in each of the above stages has been the objective of task 7.6 and the main effort has been to provide support on *efficient handling of petascale-class applications data* to developers wanting to petascale their applications. The request for support has come from tasks 7.1 [5] and 7.2 [6], which deal with application enabling.

In order to be able to provide users with support on overcoming their data challenges, experts were identified from several major European HPC centers, to provide their expertise on helping and working alongside users of T7.1 and T7.2, with the goal of scaling their applications. The experts were divided, based on their expertise, into the following four distinct subtasks: *'Parallel Pre-Processing'*, *'Parallel and Hierarchical I/O'*, *'Post-Processing and Visualization'*, and *'Long-Term Preservation of Applications Data'*. Each of the subtasks has been led by a subtask leader and the work coordinated via telcons and face to face meetings. In turn, each subtask has delt with several data related projects, each led by a team of experts.

The resulting work done in T7.6 has been documented in a total of 23 whitepapers, ranging from best practice guides to optimizations done for applications and the performance improvements achieved. This deliverable summarizes the whitepapers. The title of the whitepaper is identical to the subsection title in chapters 1 to 4. For details the whitepapers should be consulted, which can be found on the PRACE website:

http://www.prace-ri.eu/White-Papers

This deliverable is structured in four main subsections, each covering the work which has been done in the above four subtasks of T7.6, followed by a section on conclusions.

1 Parallel Pre-Processing

Many numerical simulations require nontrivial pre-processing of data before actual computations can start. For Terascale problems, pre-processing is often done on shared memory machines, after which the set of generated files are moved to a parallel computer for the actual simulation or for further processing. For Petascale problems, this approach is very limited, due to the data volume of Petascale problems. At such large scale, the input data might not even fit in the memory of a single machine or it might be too time consuming to process sequentially. Therefore, for certain applications one is forced to parallelize the pre-processing step, in order to take advantage of the distributed memory and processing power of the whole machine. This section summarizes the support requested from T7.6 and the corresponding support provided to users with optimizing and parallelizing various pre-processing steps in their applications.

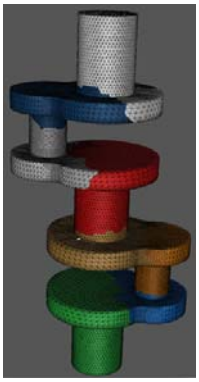
1.1 Parallel Mesh Generation, Migration and Partitioning for the Elmer Application

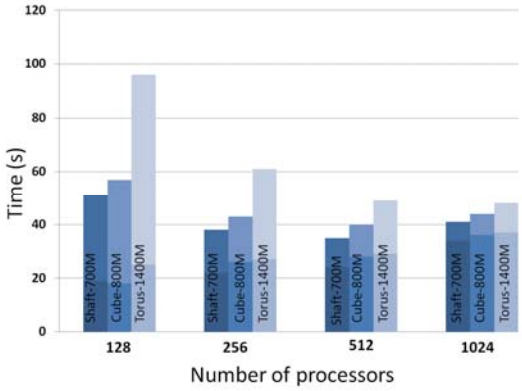
Supported by: Yusuf Yılmaz, Can Ozturan, Oğuz Tosun, Ali Haydar Özer and Seren Soner (Bogazici, Turkey)

Collaborators: Peter Raback (CSC, Finland)

Elmer is an open source multi-physics simulation software developed by CSC - IT Center for Science (CSC). Elmer employs the finite element method to solve partial differential equations associated with, for example, physical models of fluid dynamics, structural mechanics, electromagnetics, heat transfer and acoustics. Elmer has a simple built-in unstructured mesh generator, which can do *sequential* mesh generation on *simple* geometries. In order to scale further, however, support for parallel mesh generation and mesh partitioning was requested by the Elmer developers from T7.6. The main objective of this project was to develop software to "*generate large unstructured meshes with sizes in the hundreds of millions range, on complex geometry*". With sequential mesh generators, memory on a single node becomes a serious bottleneck, allowing generation of only a few tens of millions of elements. When generating huge meshes, another objective is to reduce the mesh generation time drastically.

Since development of a robust mesh generator may take many years, the following approach was followed: An existing sequential mesh generator was taken and the geometry was decomposed for parallel mesh generation, using the existing mesh generator. As a sequential mesh generation software, the Netgen mesh generator was used, due to its availability as LGPL open source software and a wide user base. The parallel mesh generation software was implemented using the MPI libraries and the C++ language.





(a) (b)

Figure 1: Example of generated mesh and performance of parallel generation of various meshes with 700, 800 million and 1.4 billion elements.

The parallel mesh generation algorithm developed in this project starts by decomposing sequentially the whole geometry on a master node into a number of sub-geometries and then meshes each sub-geometry in parallel using multiple processors. When a mesh is generated for each sub-geometry by each processor, the partition boundary mesh faces of adjacent sub-geometries need to be conforming (i.e. match). This is achieved during the initial phase on the master node, by sequentially generating a coarse mesh for the whole geometry, including the sub-geometry faces after the geometry partitioning. Each mesh is then sent to the parallel processors for volume (tetrahedral) mesh generation in parallel. Different methods were implemented based on whether coarse surface or volume meshes were generated on the master node. After the parallel volume mesh generation is performed, the result is a distributed mesh, which may be imbalanced. The imbalanced mesh can be partitioned by ParMetis. Given a distributed mesh, with final destination processors returned from the partitioner for each element, a scalable mesh migration algorithm is needed to do the migration. Therefore a migration algorithm was implemented, which utilises owner updates rule for updating the new destinations of the partition boundary mesh entities.

Figure 1 (a) shows an example of an unstructured mesh, generated on a complex shaft geometry, using the Netgen based parallel mesh generator developed in this project. Figure 1 (b) shows various timings obtained on generating 700 million, 800 million and 1.4 billion mesh for the shaft, cube and torus geometries respectively.

In conclusion, this project has achieved its objective of generating large unstructured meshes, with sizes in the hundreds of millions range, on complex geometry. In particular using refinement based methods, a 1.4 billion element mesh was generated in under a minute. Sequential generation of such a huge mesh would not be possible due to memory limitations. Using the original Netgen code, a mesh with 10 million elements can be generated sequentiall in about a minute, using a simple refinement based method. These results also imply that a user does not need to save the mesh, which would require costly I/O. Instead, the mesh can be generated on the fly, whenever a solver like Elmer needs it.

1.2 Parallel Mesh Multiplication for Code_Saturne

Supported by: Pavla Kabelikova, Ales Ronovsky and Vit Vondrak (VSB-TU, Ostrava)

Collaborators: Charles Moulinec (STFC, Daresbury) and Yvan Fournier (EDF, France)

Code_Saturne® is a multi-purpose Computational Fluid Dynamics (CFD) software, which has been developed by Electricité de France Recherche et Développement EDF-R&D since 1997. The code was originally designed for industrial applications and research activities in several fields related to energy production; typical examples include nuclear power thermal-hydraulics, gas and coal combustion, turbo-machinery, heating, ventilation, and air conditioning. Code_Saturne® was released as open source in 2007 and is distributed under a GPL licence. Code_Saturne® has been selected as an engineering community code in PRACE-1IP and supported by developing teams from several PRACE project partners, to enable its petascale capabilities.

The main challenge to enable petascaling of Code_Saturne® is the time required to generate large meshes. This is the case even for relatively modest sizes, e.g. 10 million cells, where the geometry is very complex and boundary layers have to be meshed. Therefore it is obvious that the generation of billions of cell meshes has to be parallel. Unfortunately no industrial strength open-source parallel mesh generators are available yet. Therefore other routes must be followed and the one proposed by Code_Saturne® developers deals with parallel global mesh refinement (or mesh multiplication), i.e. an initial mesh of about 100 million cells would be read by Code_Saturne®, then each of its cells would be split uniformly. This process could be repeated several times in order for the Navier-Stokes solver to run on a several billion cell mesh, while post-processing would be carried out on the initial 100 million cell mesh.

According to the Code_Saturne® developers' proposal, the main aim of the project was to develop a parallel mesh multiplication package and integrate it into Code_Saturne® to extend its capability of generating very large cell meshes. Since Code_Saturne® primarily uses 3D cells, the work in this project was focused only on structured meshes with regular cells, such as hexahedras, tetrahedras or prisms. The final developed parallel mesh multiplication package implements global mesh refinement of hexahedral meshes.

The main part of the work on developing the mesh multiplication package, was dedicated to the adaptation of the Code_Saturne® data structures into the global mesh refinement data structure. For efficient mesh refinement, it was necessary to extend the Code_Saturne mesh_t mesh storage format with vertex-edge based information, since the original mesh_t structure only handles cell structures of a mesh. This enables the mesh multiplication to be executed in parallel on distributed parts of the mesh, i.e. each parallel process takes care of its own part of the mesh, although the faces on the subdomain interface are shared with more processors. However, it is guaranteed that the resulting refined mesh matches on the subdomain interfaces for hexahedral elements (support for other regular cell types is in progress).

Application of the parallel mesh multiplication on a distributed mesh, duplicates some surface subdomain entities (vertices, edges and faces) in multiple processes. This implies reconfiguration of their global indices, which requires intensive parallel communication between the neighbouring subdomains. This inter-subdomain communication was optimized using "adjacency graph of subdomains", which has to be set-up during the partitioning phase, to extract all necessary information.

The result of this work has enabled Code_Saturne® to obtain very fine meshes by applying the implemented mesh multiplication algorithm recursively. The performance and scalability

of the implemented package have been tested on a local cluster at VSB-TU Ostrava (ComSio) and on the Curie Tier-0 system.

The following table shows the scalability of an example on the ComSio cluster. As input, an initial mesh with 700 cells and 1562 vertices was used. Applying six levels of refinement, a mesh with more than 180 million cells was obtained in less than 7 seconds (using 28 cores). The measured time contains the time needed for subdomains refinement only. The sixth level of refinement cannot be computed on 1 core because of lack of memory per 1 core.

Parameters of given mesh			Number of used cores						
level of refinement	no. of cells	no. of vertices	1 core	2 cores	4 cores	8 cores	16 cores	24 cores	28 cores
			time [s]	time [s]	time [s]	time [s]	time [s]	time [s]	time [s]
0	700	1562	<0.1	<0.1	<0.1	<0.1	<0.1	<0.1	<0.1
1	5600	8883	<0.1	<0.1	<0.1	<0.1	<0.1	<0.1	<0.1
2	44800	57605	<0.1	<0.1	<0.1	<0.1	<0.1	<0.1	<0.1
3	358400	408969	0.36	<0.5	<0.5	<0.5	<0.5	<0.1	<0.1
4	~2.87M	~3M	2.88	<0.5	<0.5	<0.5	<0.5	<0.5	<0.5
5	~22.9M	~23,7M	22.75	12.5	6.3	3.16	1.49	1.11	0.85
6	~183,5M	~186,6M	---	157.3	46.8	22.5	12.7	8.23	6.81

Table 1: Performance of Code_Saturne on the ComSio cluster

The following table shows the scalability test of the same example on the Curie Tier-0 cluster. The times in the table measure the pure time needed for the mesh refinement.

Parameters of given mesh			Number of used cores					
level of refinement	no. of cells	no. of vertices	32 cores		64 cores		128 cores	
			time [s]	no. of vertices	time [s]	no. of vertices	time [s]	no. of vertices
0	700	1562	<0.1	72	<0.1	42	<0.1	24
6	183.5M	186.6M	4.12	6.0M	2.5	3.2M	1.2	1.6M
7	1.5B	1.5B	---	---	---	---	8.05	12M

Table 2: Performance of Code_Saturne on Curie Tier-0 cluster

1.3 Code_Saturne – Optimizations in the Pre-processing Step

Supported by: Ata Turk (Bilkent, Turkey), Charles Moulinec (STFC, England), Andrew G. Sunderland (STFC, England) and Cevdet Aykanat (Bilkent, Turkey)

In this project the developers of Code_Saturne requested support from T7.6 in the investigation of pre-processing schemes, which will enable the partitioning of the domain into 100K or more processors, and analysis and improvement of parallel graph-partitioning schemes, which will enable partitioning of 2 billion cells or more.

This has resulted in the study of the performance of different mesh partitioning software packages, which can be used in Code_Saturne. Previous studies have shown that MeTis, a sequential graph partitioner, provides the best results in terms of reducing the average time spent on a single timestep of Code_Saturne. However, MeTiS has problems with partitioning

into 64K parts or more. Furthermore, for sufficiently large problems, the memory requirements of MeTiS may surpass the memory available even in fat nodes of large-scale clusters.

Analysis conducted reveal that, for medium sized meshes, if the time spent for partitioning is not important, the usage of sequential SCOTCH can be considered, since it provides reasonably good partitions with relatively low memory requirements and can easily scale to 128K cells and beyond on a single fat node. However, as the mesh sizes that are to be partitioned reach billions of cells, sequential partitioning of such meshes becomes infeasible, since they require large amounts of time and memory. In order to adjust to the memory constraints and to avoid unnecessary folding and migration of data, the partitioning has to be done in parallel and to fit such large meshes in the memory of cluster nodes, the mesh has to be partitioned into very large number of cores.

Unfortunately the partitioning performance of parallel graph partitioning packages such as Par-MeTiS and PT-SCOTCH decline with increasing number of cores involved in the partitioning process and are inferior to their sequential counterparts in terms of partitioning quality. To address this problem, a two-level hierarchical partitioning scheme has been proposed, which enables first to partition into a smaller number of parts, compared to the desired final number of parts, after which it partitions the obtained subgraph in a second level, to obtain the final number of parts. This scheme also allows the usage of different partitioning schemes at each level.

Two different sets of experiments on two different datasets (see Table 3 for properties) were conducted, to compare the partitioning schemes and tools. In the first set of experiments the partitioning results of the tools have been analyzed and in the second set of experiments, the execution time of Code_Saturne has been compared when using these partitions.

	# of vertices	# of edges
SMALL	5.780.335	23.064.296
SUBMARINE	107.673.905	427.107.558

Table 3: Dataset properties

Table 4 present the partitioning properties of MeTiS, SCOTCH and PAR-MeTiS. As seen in the table, sequential MeTiS produces the best cut values and runs much faster than sequential SCOTCH. However, in terms of memory utilization, SCOTCH is more efficient and can partition into much larger number of parts. Finally, in terms of execution time, PAR-MeTiS is the fastest. It runs more than 10 times faster than MeTiS and around 100 times faster than SCOTCH. It also produces better cuts than SCOTCH. However, when the number of parts to be partitioned increases, PAR-MeTiS has problems, just like MeTiS.

	MeTiS			SCOTCH			PAR-MeTiS	
Number of parts	cut	time (sec)	memory (GB)	cut	time (sec)	memory (GB)	cut	time (sec)
4096	5.899.387	397,30	20,84	6.365.021	2748,80	14,48	6.550.107	28,27
8192	7.569.441	430,51	21,02	8.134.073	3249,69	14,48	8.338.153	32,26
16384	9.703.853	595,28	21,85	10.332.241	3973,43	14,48	10.610.241	26,28
32768	12.405.125	792,76	24,95	13.242.850	4995,55	14,48	X	X
65536	X	X	X	16.820.681	4990,01	14,48	X	X
131072	X	X	X	21.300.171	4983,38	14,48	X	X

Table 4: Properties of partitions obtained by partitioning a SUBMARINE dataset with MeTiS, SCOTCH and PAR-MeTiS

In order to be able to utilize the fast and relatively nice partitioning capability of PAR-MeTiS, namely partitioning into much larger number of processors, the hierarchical scheme was utilized. The tests for hierarchical schemes were conducted on the BlueGene/P system at STFC, due to the limited quota on the Jugene Tier-0 system. In the runs for HIER (hierarchical scheme), shown in Table 5, PAR-MeTiS is used in the first level to divide the graph in Number_of_parts/4 and then in the second level, Number_of_parts/4 separate calls are made to PAR-MeTiS to divide each subgraph into 4.

Number of parts	HIER	MeTiS	SCOTCH	PAR-MeTiS	PT-SCOTCH
256	13,45	12,37	12,10	12,76	12,51
512	7,64	5,92	7,04	7,23	7,14
1024	4,96	3,87	4,64	4,91	5,01
2048	3,59	3,21	3,26		
4096	3,07	2,52	2,66	2,71	

Table 5: Runtime per timestep (seconds) of Code_Saturne by using partitions obtained by partitioning the SMALL dataset with MeTiS, SCOTCH, PAR-MeTiS and PT-SCOTCH

Table 5 shows that the hierarchical scheme has slightly higher runtime per timestep values, when compared to other schemes, but it has the potential to be able to scale to much larger number of cores then PAR-MeTiS and PT-SCOTCH. It is therefore interesting to investigate the hierarchical scheme further.

1.4 Improving the Load Balancing Performance in Vlasiator

Supported by: Ata Turk, Vehbi Gunduz Demirc and Cevdet Aykanat (Bilkent, Turkey)

Collaborators: Sebastian von Alfthan and Ilja Honkonen (Finnish Meteorological Institute)

Vlasiator is a hybrid-Vlasov simulation code developed at the Finnish Meteorological Institute (FMI). It can be used for modeling the electromagnetic plasma system within the near Earth space including the ionosphere, magnetosphere, and beyond. A large-scale Vlasov-hybrid simulation is highly challenging, since realistic simulations need to be executed in $\sim 10^6$ spatial grid cells for $\sim 10^6$ time-steps, indicating peta-scale computing.

The support requested by the Vlasiator team from T7.6 includes the following: Profiling of the performance of the Vlasiator code up to 10^4 cores (previous tests were performed for less than 10^3 cores due to limited resources) and analysis and improvement of the load-balancing scheme in Vlasiator. Currently Vlasiator uses the parallel hypergraph partitioning option of the Zoltan toolkit for domain partitioning.

In order to support Vlasiator on a large number of cores, it was first ported to the PRACE Tier-0 system Jugene. Also, the ZOLTAN partitioning framework in Vlasiator was modified, such that it can use the parallel graph partitioning tools ParMeTiS, and PT-SCOTCH for domain partitioning. Unfortunately, graph partitioning cannot exactly model the communication overheads associated with the Vlasiator communication model. However, if the problem domain is regular enough, the error made by graph partitioning methods for estimating the communication overhead of a partition is more or less the same for all possible partitions in the solution space. This property enables the graph partitioning schemes to improve their solutions over regular computational domains successfully, since the error made while moving through different partitions in the solution space cancel each other. Since the

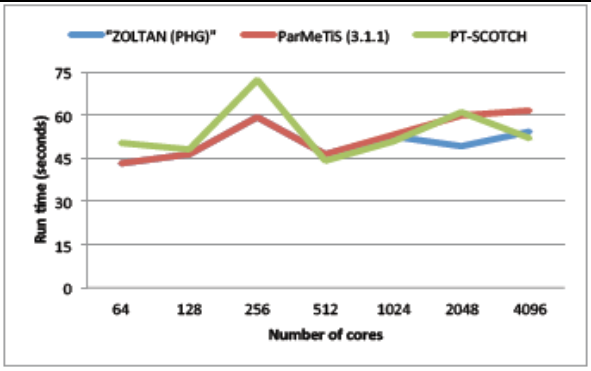
subject problem domain exhibits such features, we believe that the usage of graph partitioning tools for Vlasiator will yield good results as well.

Furthermore, the performance of three parallel partitioning tools were compared, namely ZOLTAN parallel hypergraph partitioner (PHG), ParMeTiS, and PT-SCOTCH, in terms of pre-processing (partitioning) overhead, communication overhead of the obtained partitions and the load-balancing quality up to 4K cores on Jugene. ParMeTiS and PT-SCOTCH were called from within the ZOLTAN framework, so as not to change the interface of the code. Currently, the number of processors/cores used in the experiments is not sufficient, but it is planned to extend the analysis to a larger number of processors, to be able to assess the scalability of the partitioning tools further. The metrics considered for balancing quality involves computational load-balance, total communication volume and communication load-balance.

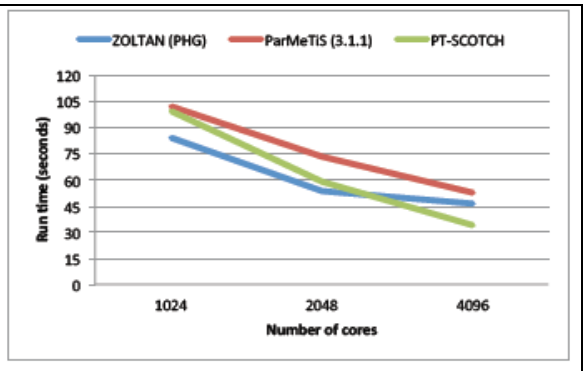
Figure 2 shows the weak and strong scaling performances of Vlasiator while using Zoltan's parallel hypergraph partitioning scheme (PHG), ParMeTiS parallel graph partitioning, and PT-SCOTCH parallel graph partitioning. In these runs, for weak scaling, a 3D grid size is arranged such that under perfect load balance, each process would have to process eight spatial cells, and for strong scaling, the total number of spatial cells is set to 32x32x16. As seen in the figure, all three tools scale reasonably well, however, PT-SCOTCH performs slightly better.

Figure 3 shows the time spent on pre-processing while using the domain partitioning tools. Note that the time spent on partitioning increases dramatically with increasing number of processors, constituting a significant portion of the overall runtime. It can be observed that all three tools perform in a similar way. This may seem awkward at first, since graph-partitioning tools are known to perform much faster than hypergraph-partitioning tools. However, as mentioned before, this equality in performance is due to the fact that all partitioning tools are called from within the ZOLTAN framework. This means that for running the GP tools, first all data structures are converted to ZOLTAN specific data structures, then ZOLTAN converts them back to data structures supported by the GP tools, partitions, and converts back the results to ZOLTAN specific data structures. This extra overhead can be removed by embedding GP support within the Vlasiator code, which would make the gains obtained via PT-SCOTCH more prominent.

Figure 4 shows the computational imbalance values obtained by running the Vlasiator code and computing the time spent on each processor. Among the partitioning tools, PT-SCOTCH seems to perform the best, both for weak and strong scaling experiments, which strengthens the expectation that PT-SCOTCH will perform better in larger number of experiments.

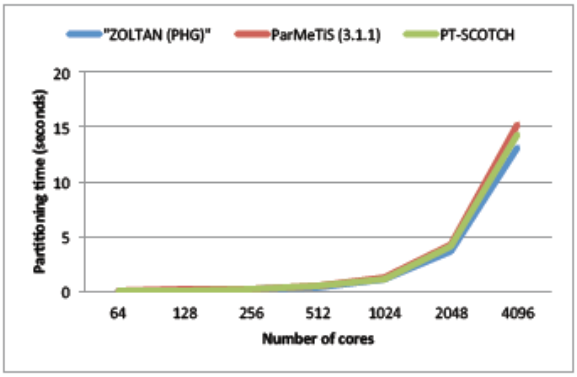


(a) Weak Scaling

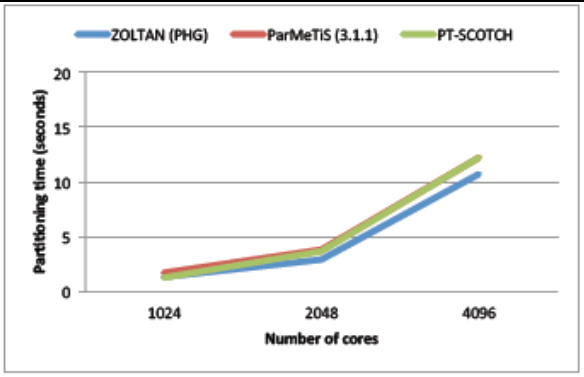


(b) Strong Scaling

Figure 2: Run time performance of Vlasiator utilizing different domain partitioning tools.

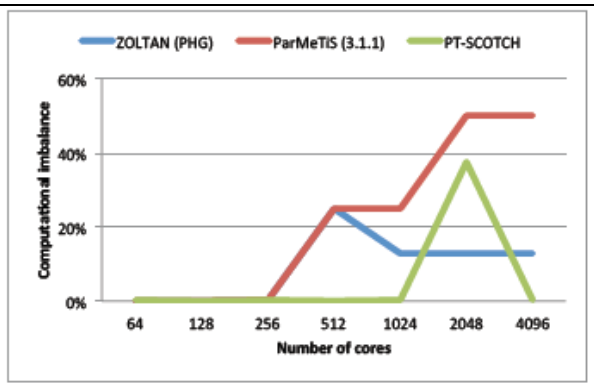


(a) Weak Scaling

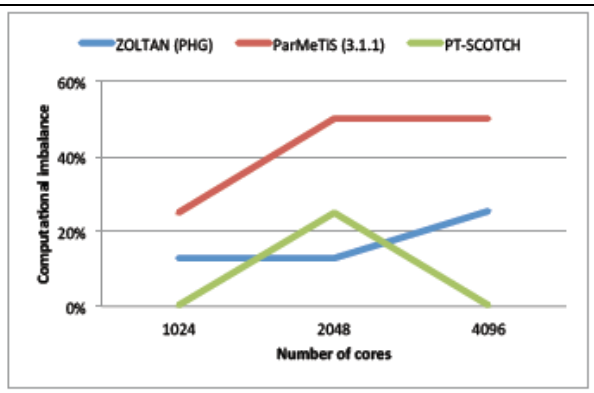


(b) Strong Scaling

Figure 3: Pre-processing time overheads of domain partitioning tools



(a) Weak Scaling



(b) Strong Scaling

Figure 4: Computational imbalance values

1.5 Fixing Node Strategies for the Effective Regularization of the Subdomain Stiffness Matrices Arising in Total FETI

Supported by: Tomas Kozubek and Vit Vondrak (VSB-TU, Ostrava), Peter Råback and Juha Ruokolainen (CSC, Helsinki)

The bottlenecks related to the numerical solution of many engineering problems are very dependent on the techniques used to solve the systems of linear equations, that result from their linearizations and finite element discretizations. Large linearized problems can be solved efficiently using the so-called scalable algorithms, based on multigrid or domain decomposition methods. In cooperation with the Elmer team two variants of the domain decomposition method have been implemented in Elmer: (i) FETI-1 (Finite Element Tearing and Interconnecting) introduced by Farhat and Roux [7] (ii) Total FETI introduced by Dostal, Horak, and Kucera [8]. In the latter, the Dirichlet boundary conditions are torn off to have all subdomains floating, which makes the method very flexible. In this project, the focus has been on the stable and flexible implementation of the FETI pre-processing method.

Due to the rounding errors, effective elimination of the primal variables (e.g. displacements in linear elasticity) of “floating” subdomains is a bottleneck of the implementation of FETI

methods, as it can be difficult to recognize the positions of zero pivots when the nonsingular diagonal block of the subdomain system matrix A, is ill-conditioned. Moreover, even if the zero pivots are recognized properly, it turns out that the ill-conditioning of the nonsingular submatrix, defined by the nonzero pivots, can have a devastating effect on the precision of the solution. Ill-conditioning often appears due to the jumps in coefficients and local mesh refinement.

In this project the results related to the solution of symmetric positive semidefinite systems, arising in FETI methods when they are applied on elliptic boundary value problems, have been reviewed and documented in the corresponding whitepaper. Furthermore, three different strategies are shown to find fixing nodes (or DOFs – degrees of freedom), which enable an effective regularization of the corresponding subdomain system matrices:

- *Kernel strategy* is based on transforming matrix R into echelon form using the Gauss elimination with full pivoting, where the columns of R span the kernel of A. The last nonzero entries of echelon form determine zero pivots, i.e., DOFs to be fixed (see [1]).
- *Geometrical strategy* is based on finding fixing nodes using simple geometrical and combinatorial arguments: choose *M* mesh nodes that are mutually as far as possible and that are not placed near any line.
- *Almost uniform distribution strategy* is based on finding fixing nodes using the following algorithm: Decompose subdomain mesh into submeshes (e.g. using METIS), and then choose centers of each submesh using the so-called Perron vector (see [2]).

Finally, a regularization technique is introduced based on the suitably of the chosen fixing nodes (DOFs), using the above strategies and adding the regularization term to the entries of A, corresponding to these nodes (DOFs), see [3] for more details. This regularization with all mentioned strategies improves stability and flexibility and eliminates the bottleneck of the FETI implementation. Particularly, it improves conditioning and enables factorization using any standard Cholesky type decomposition method for nonsingular matrices. This completely removes the work with singular matrices.

The regularization with the kernel and geometrical strategies for finding fixing nodes have been implemented in Elmer and tested on benchmarks. In Figure 5 and Table 6, the improvement of conditioning of the nonsingular part of A, corresponding to the complementary nodes (DOFs) are shown. The first configuration (NO STRATEGY) may occur if the nodes with arrows (denoting fixing DOFs) are the last three nodes in the used node numbering and the top side is curved. The best conditioning gives an almost uniform distribution strategy described in [2] but the other two strategies also give a practically sufficient improvement of the condition number.

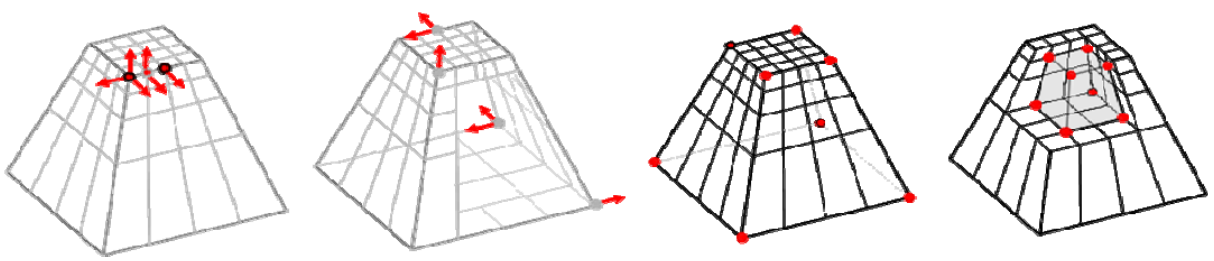


Figure 5: Fixing nodes (DOFs) strategies

NO STRATEGY	KERNEL STRATEGY	GEOMETRICAL STRATEGY	ALMOST UNIFORM DISTRIBUTION
9.4e18	9371	900	518

Table 6: Conditioning of the nonsingular part of A

To show the scalability behaviour of the ELMER FETI implementation, a 3D elastic cube was decomposed into identical boxes. Then each box was discretized by 8000 bricks. Table 7 shows the numbers of unknowns, cores and iterations, and the computational time achieved by ELMER on the French TIER-0 system CURIE. For large decompositions, the so-called coarse problem solution starts to dominate. It is therefore planned to replace the standard FETI method by its hybrid version, which eliminates this drawback.

UNKNOWNS	CORES (SUBDOMAINS)	TIME	CG ITERATIONS
192000	8	5.57	17
648000	27	10.52	26
3000000	125	9.27	31
8232000	343	10.26	32
24000000	1000	19.88	33
81000000	3375	31.52	35

Table 7: Scalability results

1.6 A Parallel Fast BEM for the Helmholtz Equation as an Extension of SPECFEM3D

Supported by: Dalibor Lukas and Jan Zapletal (VSB-TU Ostrava)

In this project, the post-processing step of a parallel acoustic simulation package has been improved, using the boundary element method (BEM). The package is built on top of SPECFEM3D_GLOBE, which is a parallel software for doing seismic simulations, such as earthquake simulations of the globe. The acoustical simulation relies on a Fourier transform of the seismic elastodynamic data, resulting from SPECFEM3D_GLOBE, which are then post-processed by a sequence of solutions to the Helmholtz equations, in the exterior of the globe. For the acoustic simulations, BEM has been employed, which reduces computation to the sphere; however, its naive implementation suffers from quadratic time and memory complexity with respect to the number of unknowns. To overcome the latter limitation, the method was accelerated by using hierarchical matrices and adaptive cross approximation techniques, which is referred to as Fast BEM. First, a hierarchical clustering of the globe surface triangulation is performed. The arising cluster pairs decompose the fully populated BEM matrices into a hierarchy of blocks, which are classified as far-field or near-field. While the near-field blocks are kept as full matrices, the far-field blocks are approximated by low-rank matrices. This reduces the quadratic complexity of the serial code to almost linear complexity, i.e. O(n*log(n)), where n denotes the number of triangles. Furthermore, a parallel implementation was done, so that the blocks are assigned to concurrent MPI processes with an optimal load balance. The processes share the triangulation data. The parallel code reduces the computational complexity to O(n*log(n)/N), where N denotes the number of processes. This is a novel implementation of BEM, which overcomes computational times of traditional volume discretization methods, e.g. finite elements (FEM), by an order of magnitude.

To demonstrate the efficiency of the method, a table with numerical results is given below. A Dirichlet boundary value problem for the Helmholtz equation was considered with a known

analytical solution, in order to document the linear decay of the approximation error. The rows correspond to the levels of discretization. In the first column, numbers of triangles are given. In the second column, the linear error decay in terms of the L2-norm of the computed Neumann data is shown. The third column shows the logarithmic compression rate of the full matrix. Finally, columns 4-8 document the overall serial (top-to-bottom), as well as parallel (left-to-right) scalability of the implementation. A comparable, i.e. producing similar error, FEM discretization of the largest problem would lead to 10^8 volume unknowns.

no. of triangles	approximation error	matrix compression	time [s] 2 cores	time [s] 4 cores	time [s] 8 cores	time [s] 16 cores	time [s] 32 cores
2,560	9.9e-3	100%	142	72	38	20	9
10,240	2.8e-3	65%	1,388	673	335	168	88
40,960	9.0e-4	26%			3,600	1,823	929
163,840	3.3e-4	8%					19,892

Table 8: Efficiency of BEM

1.7 Parallel Uniform Mesh Subdivision in Alya

Supported by: Guillaume Houzeaux, Raul de la Cruz and Mariano Vázquez (BSC, Spain)

The objective of this project has been to implement parallel uniform Mesh Multiplication (MM) in Alya, which is a Computational Mechanics (CM) HPC code, and to apply it to an incompressible Navier-Stokes solver. This has enabled Alya to run very big simulations, otherwise not possible. The basic concept of the MM implemented in Alya is the following:

- An initial mesh (referred to as the 0-level mesh) is obtained from a standard mesh generator. This mesh is generated with a a-priori knowledge of the problem: for example with boundary layer elements near the wall (for CFD). For typical engineering applications, this mesh has between 10M to 100M of elements.
- Since Alya is based on a master-slave strategy the master reads the 0-level mesh, partitions it, and sends the subdomain meshes to the slaves. After this task, the original mesh no longer exists, neither any array defined on it. Each slave automatically subdivides its mesh and reconstructs its communication arrays on the boundaries with its neighbouring subdomains.

Figure 6 below shows speedups obtained for the MM algorithm. The CPU times required to perform the MM makes it a very efficient tool for obtaining a fine mesh on the fly, without having to store it on disk. For example, only 1.4s is required to multiply a 2.9M mesh three times, so as to obtain a 1.5B mesh, on 16384 CPU's.

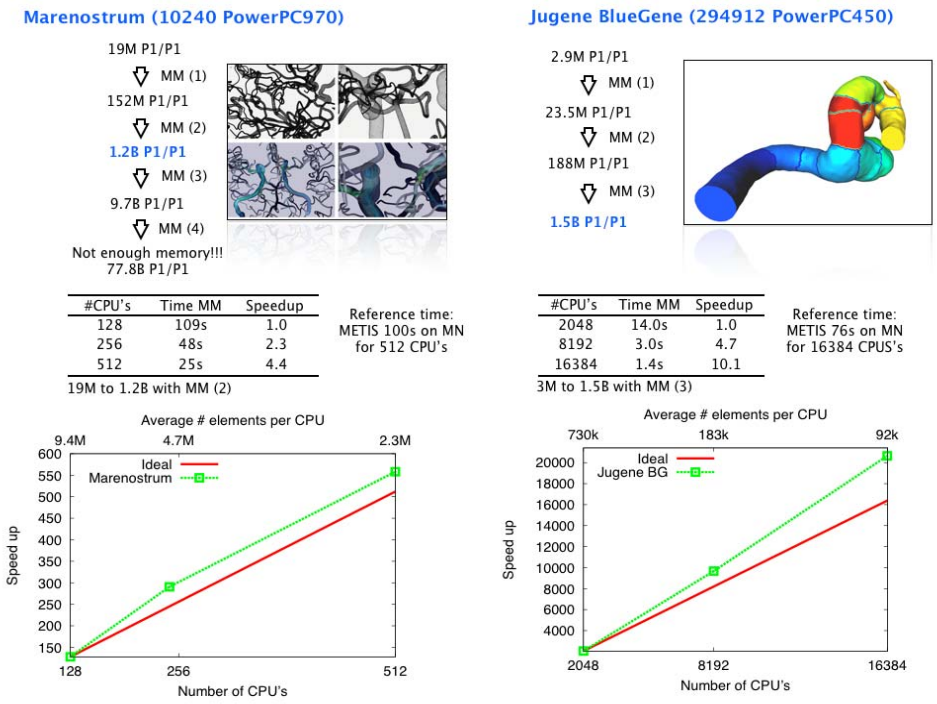


Figure 6: Time and speedup of the implemented Alya mesh multiplication

Figure 7 below shows the resulting (almost linear) speedup obtained for solving the Navier-Stokes equations on Blue Gene.

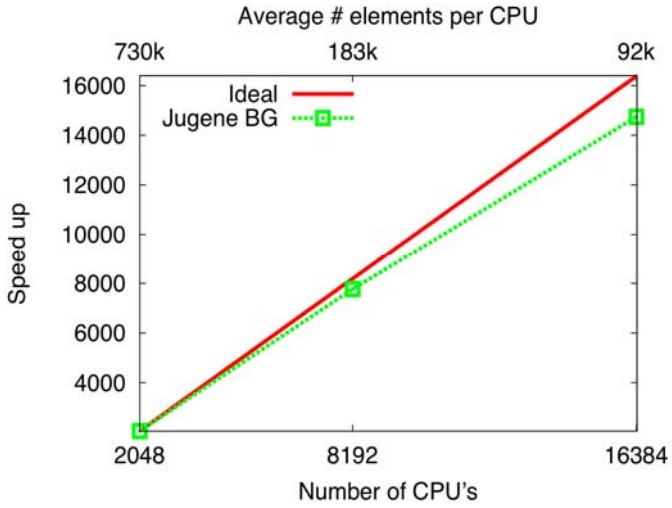


Figure 7: Speedup of the Alya Navier-Stokes equations

Finally, the implemented Alya MM algorithm has been tested to carry out a mesh convergence for the simulation of air flow in large nasal airways. **Figure 8** below shows comparisons of results obtained on meshes, together with the coarse mesh, using one level of MM.

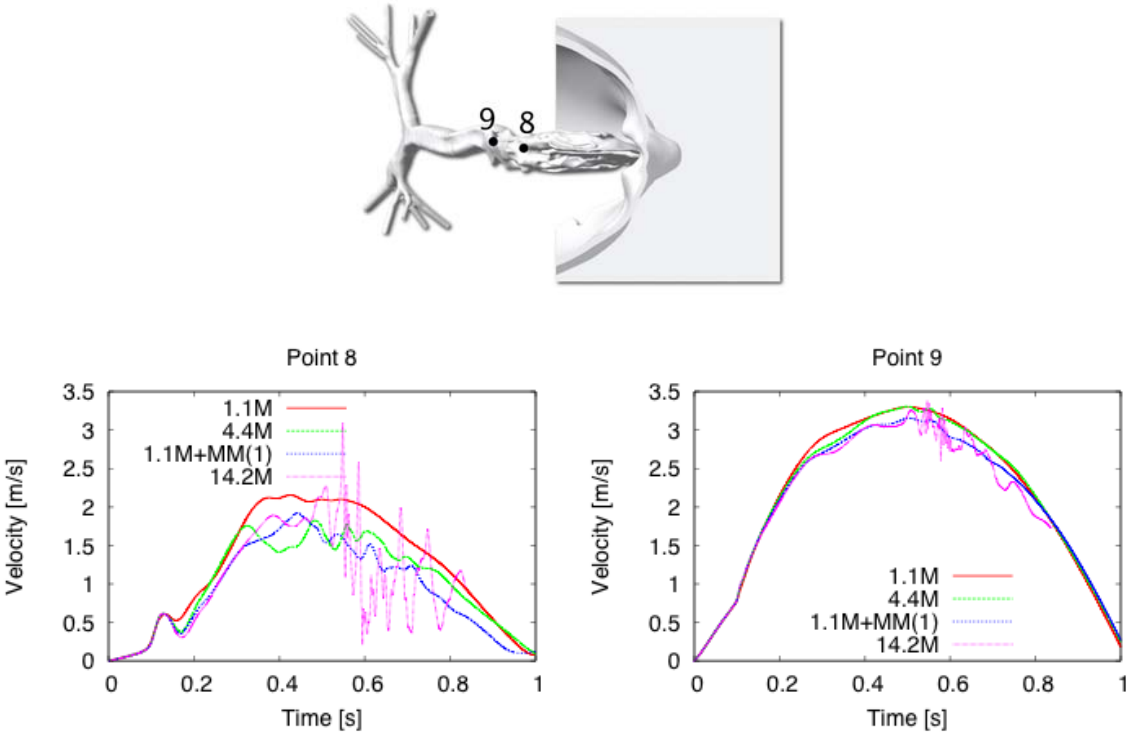


Figure 8: Evolution of velocity modules at points 8 and 9 located in the computational domain using 1.1M, 4.4M, 14.2M meshes compared to 1.1M using 1 Mesh Multiplication level: 1.1M+MM(1)

1.8 Implementation of Fragment Orbital Method (FMO) for Highly Parallelized Quantum Chemical Calculations with CP2K

Supported by: Peicho Petkov, Petko Petkov, Georgi Vayssilov and Stoyan Markov (NCSA, Bulgaria)

The realistic simulation of various biochemical systems requires application of a reliable first-principle quantum chemical method for extra-large systems (of 10^6 to 10^7 atoms), for long simulation times, from nano- to microsecond. Despite the high performance that parallel computing systems make available, their use for such simulations is not trivial, mainly for two reasons:

- Even the most appropriate first-principle methods based on density functional theory (DFT) scale with N^3 (N is the number of electrons in the system).Therefore calculation of the electronic structure of bio-molecules is not possible.
- The possibilities for efficient parallelization of a quantum chemical system, if it is considered as one system, are limited due to mutual interactions between the electron density in all parts of the system (including Coulomb and exchange-correlation interactions).

An approach for division of the large system into many fragments (*monomers*), to be calculated simultaneously on different nodes, has been implemented in CP2K. It provides essentially linear scaling of the computational power, with the size of the system and efficient parallelization of the computational work is based on division of the whole system into fragments (*monomers*). The change in the scaling is achieved not by modification of the quantum chemical calculations, but from this division of the system. In this case the computational time is proportional to the number of monomers n_f and the time necessary for

calculation of the individual monomers, which still scales as N_e^3 for DFT (N_e is the number of electrons in the monomer):

$$\mathbf{t_{mono} \sim n_f \, x \, N_e^3} \quad \textit{or} \quad \mathbf{t_{mono} \sim n_f \, x \, NLogN} \ \textit{(in the case of CP2K)}$$

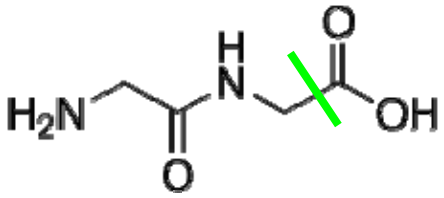
Thus, if the size of the studied system is extended, not by increasing the size of the individual fragments, but by adding new fragments, then the computational time will increase linearly with n_f, i.e. with the size of the system. Moreover, if the next fragments are calculated at additional nodes, than the real time of the calculation does not increase. Such perfect behaviour can be accomplished, however, only for non-interacting fragments, which is not of particular interest to chemistry. Different strategies to take into account the interactions between fragments have been developed. They are based on separation of the calculation for the whole system into three components:

- Global Coulomb interactions of each fragment with the rest of the system, which are less time consuming than the quantum part of the calculations;
- First principles calculation for the quantum (exchange-correlation) interactions within the isolated fragment (in the field of the rest of the system);
- Estimation of the quantum interaction of the fragment with other (neighbouring) fragments.

Since the Coulomb interactions are long-range, it is calculated completely in all methods. The main difference between different approaches comes from the division of the system into fragments and accounting of the quantum interactions between them.

As part of this work so far, the following has been developed:

- A scheme for the implementation of the above approach was created, including a workflow diagram.
- In order to do calculations in the framework of the FMO approach, a concept for division of the whole system into fragments was developed (for proteins and nucleic acids). The system is divided into fragments, as it is shown below. The dangling bonds are saturated by H atoms placed on the C-C bond.



- A pre-processing module for fragment creation was developed
- A pre-processing module for automatic generation of an input file for CP2K for each fragment was developed.
- A pre-processing module for calculation of the electrostatic field of the whole system into the space of each fragment was developed.
- A new module was added in the CP2K code for loading the pre-generated electrostatic potential; this potential will be used as an external field in the quantum chemical calculation for the fragment. DFT calculations with a predefined external electrostatic field was originally supported in CP2K, however only by supplying a function to calculate the external field. If the field map can not be described by a simple function, which is the case with FMO protein calculations, one needs to be able to load the electrostatic field from an external source. This is now possible due to this project.

- A post-processing module was developed, which can calculate the energy and electron density of the whole system from separately calculated fragments.

As a result of this project, pre-processing and post-processing modules which allow calculation of the electronic structure of bio-molecules, were developed. Although the modules were developed for CP2K, they can be easily modified, to support other software which calculate the electronic structure of molecules.

2 Parallel and Hierarchical I/O

An I/O strategy suitable at Terascale can easily swamp Petascale I/O subsystems. Due to the large number of processors and threads in Petascale systems, applications require massive use of I/O during simulations and therefore application I/O strategies have to be adapted accordingly. Furthermore, the lack of global file systems with enough performance for a Petascale machine degrades scalability, thus, hierarchical mechanisms are also often necessary to achieve good scalability. This section summarizes the work done on studying and optimizing the I/O strategies of T7.1 and T7.2 applications and documenting the I/O subsystems of the Jugene and Curie PRACE Tier-0 systems.

2.1 Data I/O Optimization in GROMACS Using the Global Arrays Toolkit

Supported by: Valentin Pavlov and Peicho Petkov (NCSA, Bulgaria)

GROMACS uses a mechanism for data I/O that does not scale well on IBM Blue Gene/P (and other machines with distributed memory architectures). All input data is read by the master MPI node and then scattered to the computing nodes during the domain decomposition step. Later it is gathered from the computing nodes in order to write it down to the output files. This method is fine for clusters and shared memory architectures like CRAY, where the amount of RAM on the master node can be extended, but introduces a bottleneck for distributed memory systems with hard memory limits like the IBM Blue Gene/P. The effect is that even though a Tier-0 Blue Gene/P machine has enough overall computing power and RAM, it cannot process large systems because the master node simply does not have enough RAM to hold all necessary input data. Thus, the support requested in this project was to enable GROMACS to handle large systems (> 5,000,000 atoms) on IBM Blue Gene/P, by introducing changes in its data I/O handling routines.

The objective of the project was to research the means and provide a solution to the memory bottleneck problem in GROMACS in IBM Blue Gene/P and for distributed memory architectures in general.

The approach taken was to define and implement a Virtual Array structure, which from the point of view of the application is just a regular array, but its physical storage is distributed across all compute nodes. The Virtual Array structure content is accessed only through a well-defined interface that hides the details of its implementation. This allows the implementation to change. As part of this project, an implementation based on the Global Arrays Toolkit (http://www.emsl.pnl.gov/docs/global/) from PNNL was created. The following improvements and optimizations had to be done in GROMACS in order to achieve the objective of the project:

- Change the build system to allow the user to choose whether GROMACS is compiled with virtual arrays or not, and if yes, which implementation to use.
- Change the molecular dynamics simulator set of command line arguments to include a new argument, which instructs the runner to use virtual arrays where applicable.
- Specify the virtual arrays interface in the form of structure type definition and function signatures.
- Implement the virtual arrays interface using Global Arrays Toolkit.

- Replace all references to modified structures (atom coordinates and velocities for now), when used in a global context with corresponding calls to the virtual arrays interface.

The correctness of the modified package was verified by running many simulations of different systems, with the original and with the modified package. In all cases, when running experiments on the same system under same conditions, the original and modified packages produced identical trajectories.

The scalability of the modified package was verified by running a series of simulations on exponentially increasing partitions, by both the original package (in DUAL mode) and the modified package (in VN mode). Table 9 shows the raw results. A brief analysis shows that the introduction of GA lead to decreased execution scalability due to the introduction of additional communications. This is expected and it might be argued that the decreased execution scalability is the price to be paid for the increased data scalability. Careful analysis however, shows that the delay introduced with GA, grows linearly with exponential growth of # of CPUs, which means that for larger and longer simulations, this might not be a problem.

	Original package, DUAL mode			Modified package, VN mode		
# Nodes	# CPUs	Time, s	GFLOP/s	# CPUs	Time, s	GFLOP/s
128	256	18020.930	10.786	512	10289.501	18.892
256	512	9389.011	20.706	1024	5986.217	32.475
512	1024	5003.518	38.856	2048	3817.619	50.931
1024	2048	2847.314	68.288	4096	2733.136	71.142

Table 9: Performance data of a system with 5,504,000 atoms.
Box 55x34x27 nm; dt=5fs, 1000 steps

2.2 The JUGENE I/O Subsystem, its Architecture, Guidelines and Tools for Using it Efficiently

Contributor: Huub Stoffers (SARA, Netherlands)

The I/O subsystems of high performance computing installations tend to be very system specific. The PRACE Tier-0 systems are no exception in this respect. Many applications that need handling of peta-scale data cannot afford to use the I/O subsystem inefficiently and hence need to be acquainted with its characteristics. The first part of this paper dissects the I/O subsystem of JUGENE as a layered system and describes the components, the interconnections, and the bandwidths of these. Bottlenecks, or oversubscribed channels on which contention for bandwidth can occur, in the paths from storage to compute environment are pointed out. While the project was underway, a partial overlap of goals, to provide users with this information, with task PRACE-1IP 7.3 – which produces “Best Practice Guides” - was noted. Substantial parts of a preliminary version of this document were contributed to the PRACE best practice guide for JUGENE.

The second section explores what organization of I/O among multiple tasks works best, or is at least fairly efficient on JUGENE. Rather than adopting any particular library for parallel I/O, options open to any program that uses a combination of standard I/O and MPI communication – rather than parallel MPI I/O - are explored. Simply letting each task concurrently do its own standard I/O, using its own task-specific file(s), is adopted as a first baseline model. Routing the I/O of all tasks to a single task is taken as a second baseline

model. Both models are known *not* to work well at all for JUGENE, but are nonetheless realistic points of departure as many existing codes really use one of these two ways of I/O organization. Hierarchical variants of I/O organization, in which a limited number of tasks do I/O on behalf of others, are explored. They are made quantitatively comparable amongst each other, and comparable with the base line models, by implementing them in small I/O-only test programs that are all given essentially the same job to do: viz. to deliver a fixed I/O volume of task specific data, produced per task, to the file system. The programs time aspects of their own I/O organization: opening or closing of file, writing data, and also gathering data from other tasks by means of MPI communication collectives.

The tests have been run in the JUGENE production environment. Results are compared and explained – sometimes with reference to the first part of the paper. The test program for the base line model that uses a file for each task is shown to achieve an average per task throughput in the 20 – 30 KB/s range, for up to 8192 cores. It drops far below that, when scaling to 16384 cores or more. Splitting the tasks into a number of equally sized groups, in which one member does I/O on behalf of others, clearly improves performance - but only moderately, when group membership is essentially determined arbitrarily. Average per task throughput diminishes with higher number of cores, but at a much slower rate. For 16384 cores it ranges between 40 and 85 KB/s, depending on the size for groups.

Per task throughput can be substantially improved by grouping non-arbitrarily, i.e. by bringing topological information about the machine into the program, about which tasks are served by the same I/O node, and use this for grouping tasks. BlueGene specific MPI extensions have been used to create a division into groups that not only is balanced in the I/O volume they produce, but also in the underlying resources to handle the load. The test program that uses this information achieves average per task throughput in the 200 – 230 KB/s range, even when scaled up to 65536 tasks.

Source code listings of tests programs that implement a particular improvement are included as appendices of the paper as programming examples.

2.3 Evaluating Application I/O Optimization by I/O Forwarding Layers

Supported by: Jan Christian Meyer and Jørn Amundsen (NTNU, Norway)

Collaborators: Xavier Saez (BSC, Spain)

This project has assessed the I/O Forwarding and Scalability Layer (IOFSL, http://www.iofsl.org), which provides a software layer between user applications and the file system. It intercepts MPI-IO and POSIX I/O calls and forwards them to a designated I/O server, offloading the native file system. The objective of the project has been to evaluate it for PRACE application use, as an alternative to the cost of rewriting application I/O routines.

IOFSL can be leveraged through preparing a customized MPICH-2 installation, which adds an additional virtual filesystem ‘zoidfs’ into the ROMIO framework. To intercept POSIX I/O calls, a FUSE client program is provided, allowing a zoidfs file system to be mounted at a user-specified directory. Both mechanisms are transparent to the application code.

The EUTERPE plasma physics code is part of the PRACE benchmark suite, and was selected as a test case. Its I/O is entirely written in Fortran, making system calls only as reflected by the compiler’s implementation of Fortran I/O intrinsics. It admits hybrid thread/process parallelism, through the PetSc library and threaded BLAS/LAPACK.

The tested version of IOFSL imposes structural criteria on the application code, particularly restricting concurrent access through the POSIX subsystem, and compiler compatibility.

The interactions of EUTERPE and IOFSL were examined for cases ranging from 128 through 4096 threads, executing on an SGI Altix ICE 8200 system, featuring a Lustre filesystem, and 2048 hyperthreaded CPU cores in 8-way SMP nodes connected by InfiniBand. The topology of this interconnect revealed that hybrid parallelism was required, as the client which intercepts POSIX system calls failed under simultaneous access by multiple processes.

Hybrid runs of EUTERPE use a PetSc library built with Intel math kernel libraries, and expects the Intel Fortran compiler. The POSIX system calls, which translate I/O intrinsics, caused the IOFSL client to fail. Small test cases of Fortran I/O were proven to work with the gfortran compiler from the toolchain, which built the I/O forwarding server, but using this to build EUTERPE would require additional porting work.

As the framework did not admit the application, EUTERPE I/O characteristics were emulated using the IOR filesystem benchmark (http://sourceforge.net/projects/ior-sio/), finding I/O load induced by a given time step from static analysis of the source code. Coupling I/O performance figures from IOR with computation time estimates from tracing EUTERPE on the Lustre filesystem, proved consistent with the overall traces of EUTERPE runs, suggesting that performance estimates with IOR and IOFSL can reflect its characteristics. The estimated accuracy was found to be within the variability of compute time traces.

The dominant I/O requirement of EUTERPE is a periodic checkpoint, with a frequency parameter in the input data. In a strong scaling scenario with a user-provided test case, the cost reduction obtained by the shrinking size of per-process checkpoint files is smaller than the reduction in per-process computational work, accounting for 15% of the cost of its computational step with 8 cpus/128 threads, growing to 44% in the 256 cpu/4096 thread case (and showing signs of stagnation, as evident in Table 10).

cpu#/thread#	8/128	16/256	32/512	64/1024	128/2048	256/4096
I/O cost	3.97s	2.32s	2.18s	2.01s	1.99s	2.04s
Computation	21.28s	11.31s	6.22s	3.77s	2.65s	2.59s
IO / total	15%	17%	26%	34%	42%	44%

Table 10: I/O cost relative to computation step cost

Table 10 implies that the overhead of creating files ultimately dominates the bandwidth requirement of writing them, suggesting that further scaling may benefit from offloading parallel file system, which typically suffer some performance degradation facing growing numbers of smaller files.

The bandwidth penalty of redirecting the I/O traffic, however, overshadows any potential benefit on the test system. IOR benchmarks with per-process file sizes ranging from 1 to 64MB consistently produce mean write bandwidths in excess of 4GB/s running over Lustre, while figures with IOFSL range from 600 though 900 MB/s using MPI-IO, and from 12 to 42 MB/s with the POSIX client, making it intolerable for application use.

The conclusion of this study is that *while the file creation bottleneck which can be ameliorated by user-space I/O forwarding is observable, use of the tested forwarding layer comes at prohibitive cost.* This conclusion is founded on the cost balance using nodes with small counts of powerful processors, and may not be appropriate for massively parallel, simplified core architectures.

2.4 I/O-profiling with Darshan

Supported by: Bjørn Lindi and Henrik Nagel (NTNU, Norway)

Collaborators: Jörg Herzer (HLRS, Gemany), Orlando Rivera (LRZ, Germany) and Peter Nash (ICHEC, Ireland)

Darshan (Sanskrit for "sight" or "vision") is a tool for I/O-profiling without imposing too much overhead. It is in essence a library that either is dynamically or statically linked to the application that gets profiled. The library is implemented with Peta- and Exascale applications in mind and works on different platforms. Darshan has been used on both Jugene and Curie, to profile OpenFOAM (Open Source Field Operation and Manipulation), in order to get a better understanding of its I/O bottlenecks.

OpenFOAM is a C++-library for building applications in continuum mechanics. OpenFOAM has a huge interest in the different parts of the CFD-community. Accordingly it has been selected as the preferred community code by T7.2. The use cases of OpenFOAM showed poor scalability, as the overall compute time increases with increasing number of processes. To understand how I/O played a part in the execution of the use cases, five cases with 64 to 1024 processes were executed with Darshan enabled, the results are shown in Table 11 below:

Number of Processes	64	128	256	512	1024
Compute time [s]	686	801	890	1161	2248
Cumuluate metadata [s]	64	202	274	389	892
The share of Meta data handling of the overall compute time	9.3%	25%	31%	34%	39%

Table 11: Darshan profile of OpenFoam

As can be seen, when increasing the number of processes from 64 to 128, the increase in time spent on metadata handling is larger than the overall increase in compute time. When further increasing the number of cores, metadata handling takes a larger and larger share of the overall compute time. The increase in time spent on metadata handling is partly due to an increasing volume of small files. The number of files created and read doubles with each doubling of number of processes. The average file size is at the same time approximately halved, as shown in the Table 12 below:

Number of processes	64	128	256	512	1024
Number of files created	512	1024	2048	4096	8192
Number of files read	1089	2177	4353	9729	17409
Average file size	597K	317 K	163K	84K	47K
Number of stat() calls	500 000	1000 000	2000 000	4000 000	8500 000

Table 12: Darshan meta data profiling of OpenFoam

OpenFOAMs I/O characteristics are contradictory to features needed to achieve good I/O-performance during scaling. Good I/O-performance is achieved by using the services a global parallel file systems provides, which are reading and writing parallel streams of data in large chunks. The volume of metadata handling must be low or comparable to the volume in data

provided to the computing process. Typical chunk or block sizes are in the range of 1MB – 4MB. OpenFOAM on the other hand, creates files where the average size is 10-20% of the block size of the file system (in the 1024 processes case). Furthermore, each process creates eight files. Accordingly, the number of files is proportional to the number of processes, leading to substantial metadata traffic.

Further increased metadata traffic, is caused by the large volume of stat()-calls. Inspection of the code shows that OpenFOAM uses files as a way of communication between the processes. MPI is used for process creation, but data is scattered and gathered through updates of files created by each process. OpenFOAM uses an instance of an object named FileMonitor to issue stat()-calls, which check the timestamp of the different files during the path of simulation. A newer timestamp indicates that the parameter/result is updated. Increasing the number of processes effectively kills scaling since this increases the number of files to stat(). Accordingly, metadata handling is seen as the bottleneck.

In conclusion the profiling of OpenFOAM with Darshan clearly indicates an application with scaling properties that are constrained by poor I/O-design. Currently, OpenFOAM uses a IOStream object to read and write updated parameters. IOstream is used throughout the application code of more than a million lines of C++-code. It is not easy to introduce a new scheme of communication and I/O under such conditions, although it is worth investigating whether a I/O-library like parallel-NetCDF can be used by OpenFOAM. By encapsulating parallel-NetCDF and overloading the IOStream operator, another way of communication could be implemented.

Finally, it is worth mentioning that a single OpenFOAM user may easily deplete the metadata service of a global parallel file system. There are examples where the number of files created in addition to being proportional to the number of cores, also are proportional to the number of time steps, actually creating a subdirectory for every time step. Under such conditions a user may create a billion files during a simulation of 100 000 time steps with 1000 processes creating 10 files per time step.

2.5 Parallel I/O Performance and Scalability Study of the PRACE Curie Tier-0 Supercomputer

Contributors: Philippe Wautelet (IDRIS-CNRS, France) and Pierre Kestener (CEA Saclay, France)

Data access on large-scale supercomputers tends to be a very big challenge. Understanding and tuning parallel I/O is necessary to leverage aggregate communication and I/O bandwidth of client machines. Finding usable, efficient and portable programming interfaces and portable file formats are also of crucial importance. Ideally, the use of high-level I/O libraries (HDF5, PnetCDF…) or I/O middleware (MPI-IO) should reduce the need for optimizations in application codes. However, to obtain good I/O performance, it is absolutely essential to investigate how to set MPI-IO hints in an application code in order to make an appropriate use of the underlying I/O software stack (MPI-IO, parallel file system) on top of the actual hardware. More precisely, MPI-IO hints are parameters that help tune the MPI-IO middleware for facilitating, for example, concurrent access by a group of processes (collective I/O, atomicity rules …) and efficiently mapping high-level operations into a dedicated parallel file system flavour (GPFS, Lustre, …).

In this whitepaper the results of two kinds of parallel IO performance measurements on the CURIE supercomputer is reported. In a first series of tests, the open source IOR benchmark is used to make a comparative study of the parallel reading and writing performances on the

CURIE Lustre filesystem, using different IO paradigms (POSIX, MPI-IO, HDF5 and Parallel-NetCDF). The impact of the parallel communication mode (collective or independent) and of the MPI-IO hints on the performance is also studied. Using the IOR micro-benchmark, it has been possible to study different combinations of MPI-IO hint values regarding data-sieving techniques and collective buffering. It was discovered that the best performance in aggregate reading bandwidth, with collective IO enabled, is obtained for the following set of parameters: *romio_cb_read/write = disable* and *romio_ds_read/write = disable.* This results in Parallel-NetCDF achieving the best performance by a factor of almost 3, compared to MPI-IO. However, the best performance in aggregate writing bandwidth with collective IO enabled is obtained when all the above mentioned hints are set to value *disable,* and in that case Parallel-NetCDF provides a 20% increased bandwidth compared to MPI-IO, whether in collective mode or not.

Although micro-benchmarks such as IOR are useful, they can give results that are far from what can be seen in a real application. Therefore, in a second series of tests, a well known scientific code from the HPC astrophysics community was used, namely RAMSES, which is a grid-based hydrodynamics solver, with adaptive mesh refinement (AMR). IDRIS added support for the 3 following parallel IO approaches: MPI-IO, HDF5 and Parallel-netCDF. They have been compared to the traditional one file per MPI process approach. In a first step, the MPI-IO hints were selected (they impact also the HDF5 and Parallel-netCDF approaches). For this application, the default values were the most appropriate ones for the writing phase. For reading, disabling the collective buffering optimisations was necessary to get the best performance. After that, scalability tests were made with 2 different problem sizes. The POSIX approach is still the most efficient one despite its drawbacks. However, with a high number of cores, the MPI-IO paradigm begins to compete with the POSIX one. The HDF5 and Parallel-netCDF approaches give reasonable results but in some cases can be problematic.

2.6 Implementing a XDMF/HDF5 Parallel File System in Alya

Supported by: Raúl de la Cruz and Hadrien Calmet (BSC, Spain)

Collaborators: Guillaume Houzeaux (BSC, Spain)

Alya is a Computational Mechanics (CM) code, which solves Partial Differential Equations (PDE's) in non-structured meshes, using Finite Element (FE) methods. Being a large-scale scientific code, Alya demands substantial I/O processing, which can consume considerable time and can therefore potentially reduce speed-up at petascale.

The current Alya I/O model is based on a master-slave approach, which limits scaling and I/O parallelization. However, efficient parallel I/O can be achieved using freely available middleware libraries, which provide parallel access to storage. In order to avoid an I/O bottleneck and allow petascaling, support was requested from T7.6 with implementing parallel I/O in Alya.

To achieve an open-standard format in Alya, the XDMF approach (eXtensible Data Model Format) has been used as a metadata container (for light data), along with HDF5 (for heavy data). To overcome the I/O barrier at petascale, XDMF & HDF5 have been introduced in Alya and compared to the original master-slave strategy.

Two interesting strategies to arrange data in XDMF format for CFD codes have been proposed. Ordered from lower to higher complexity of implementation, the proposals are: one dataset space per unknown variable and multiple dataset spaces per problem domain. In

addition, depending on the simplicity of the implementation, the former strategy supports two ways of writing data, namely repeating and not repeating unknown variables, at boundary nodes. The advantages and disadvantages of each strategy are discussed in the whitepaper.

Finally, the parallel I/O implementation of Alya is also described and assessed on two Tier-0 supercomputers, namely Curie (Intel Nehalem-EX cluster) and Jugene (BG/P cluster). The I/O gain for the post-processing task is quite significant when compared with the original strategy; achieving speedups of 1.57 and 1.67 on Curie and Jugene respectively (see Table 13). It is also shown that XDMF and HDF5 formats are well suited to speedup I/O tasks in CFD codes in general.

		Curie		Jugene	
	Tasks	Master	Slaves	Master	Slaves
Master writes I/O	Running	103.6	2.2	112.1	1469.8
	MPI_Recv	69.7	–	5149.2	–
	MPI_Send	–	551.8	–	18160.5
	I/O (Posix I/O)	547.9	–	15735.1	–
	MPI_Barrier	0.07	164.4	1.21	187.8
	Total time (ms)	719.3	718.4	22824.1	19818.3
XDMF HDF5	I/O (MPI-IO)	–	438.1	–	6591.4
	MPI_Barrier	454.8	18.6	6641.8	48.6
	Total time (ms)	454.8	456.7	6641.8	6640.0
	Speed-up	1.57		3.43	

Table 13: Comparison of post-processing times for the human respiratory model in a single snapshot using master-centric versus XDMF/HDF5 model. Note that only the best case is shown for each architecture.

2.7 Optimizing the I/O of Pluto

Supported by: Giusy Muscianisi and Marzia Rivi (CINECA, Italy)

The goal of this project was to optimize the I/O performance of the PLUTO code on the JUGENE cluster, with the help of T7.6, in order to eliminate its I/O bottlenecks. PLUTO is a modular Godunov-type code intended mainly for astrophysical applications and high Mach number flows in multiple spatial dimensions. The code embeds different hydrodynamic modules and multiple algorithms to solve the equations describing Newtonian relativistic *Magneto Hydro Dynamics* (MHD), or relativistic MHD fluids in Cartesian or curvilinear coordinates. Computations may be carried on either static or adaptive (structured) grids, the latter functionality being provided through the Chombo adaptive mesh refinement library.

The main output formats of PLUTO are raw binary, VTK for the static grid version and HDF5 for the adaptive grid. The main output bottleneck was related to the raw binary part, handled by ArrayLib, which is no longer maintained and limited to a subset of MPI functionalities, which do not allow the proper exploitation of available features of current HPC platforms. Initially, the HDF5 format was extended to the static grid, after which more effective solutions for the binary format have been investigated, exploiting the features provided by using non-blocking collective MPI2 I/O functions.

HDF5 provides two possible parallel file drivers: MPI-IO and MPI-POSIX. While the first one can be set either '*independent*' (i.e. each processor can access the file independently and any conflicting accesses are handled by the underlying parallel file system) or '*collective*' (i.e. collective buffering is enabled), the MPI-POSIX can perform only independent access to the file. After testing and tuning these file drivers, the MPI-IO 'collective' was shown to be the

most efficient. With these settings, a benchmark comparison between the original binary I/O and the HDF5 format has been performed on the Jugene Tier-0 system.

The test case used for the benchmarking is a numerical simulation of hypersonic hydrodynamical jets in a 3D Cartesian domain with size 14x70x14, in units of the jet radius. At the maximum resolution of 20 zones/beam radius, the solution of the Euler equations requires 280 x 1400 x 280 grid cells. Since five variables (including density, three components of velocity and pressure) are solved for and written to disk, each output file is approximately 4 Gb in size for double precision datasets and 2 Gb for single precision. Single precision data is the main output format and, to closely track the evolution, about 50-60 output files are produced during the whole computation. Figure 9 shows that HDF5 I/O performs worse than the binary one. This is most likely due to the overhead related to the HDF5 structure and the corresponding information provided. The compatibility between the internal parameters of this format and the configuration of the underlying file system (GPFS) must also be considered. Finally, certain tests and performance analysis of the optimized binary version have been performed and documented.

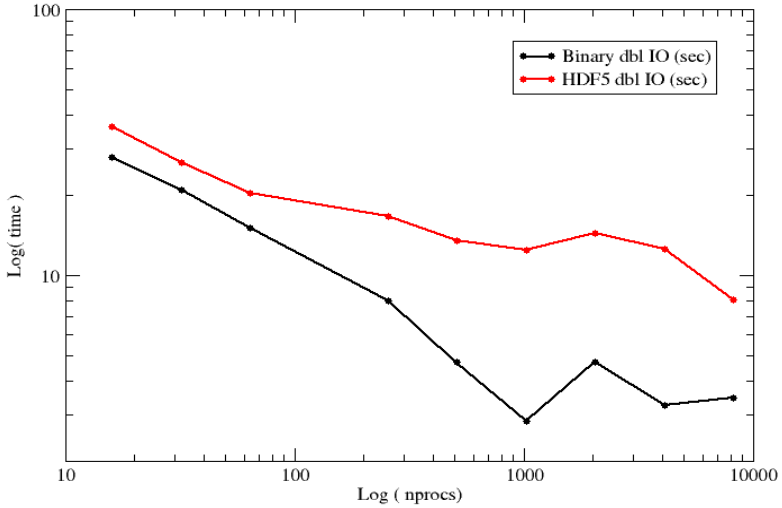


Figure 9: Comparison of HDF5 I/O performance

3 Post-Processing and Visualisation

Due to the large size of petascale simulations, extensive amounts of data are produced as final output. Post-processing and visualizing petascale output data brings with it the same challenges as for pre-processing, namely that the output data produced can be too big to fit in a single machine. Therefore one needs to extract and reduce the relevant information in parallel. This is especially true for present visualization tools, where petascale data size problems cannot be visualized without some data filtering, which reduces the data volume managed by the tool. This section summarizes the work that has been done on introducing techniques such as parallel remote visualization and in-situ simulation in user codes from T7.1 and T7.2.

3.1 Parallel Visualization of Petascale Simulation Results from GROMACS, NAMD and CP2K on IBM Blue Gene/P using Visit Visualization Toolkit

Supported by: Valentin Pavlov, Anton Tomov, Vesselin Slavchev, Nina Ilieva and Dimitar Dimitrov (NCSA, Bulgaria)

Post-processing of petascale simulation results – and in particular visualization – is in itself a hard task that requires a lot of resources. In petascale simulations, life science software packages like GROMACS and NAMD deal with millions of atoms whose trajectory footprint may require hundreds of MB *per frame*. Computational chemistry packages like CP2K will produce similarly sized output for even smaller systems. Just the movement of this data from the computing center to the scientists' workstation can become prohibiting. Moreover, workstation hardware may not be powerful enough to render the millions of atoms and bonds produced in a single frame. Below is an outline of the support requested in this project, namely a post-processing code that satisfies the following:

- Provides parallel visualization post-processing that supports at least GROMACS, NAMD and CP2K output formats;
- Uses a distributed architecture that allows the post-processing engine to run on the remote computing center, using its resources to process/render the scenes, while the actual images are viewed on a local workstation close to the scientist;

The objective of the project was to research the means and provide methodology and tools for users of the above mentioned packages to be able to visualize petascale simulation results, using the computing power of Tier-0 and Tier-1 machines, and more specifically IBM Blue Gene/P, without the need to physically transfer the whole output to their workstations.

The VisIt open source package (http://visit.llnl.gov) from LLNL supports to some extent the above requirements. Still, some improvements and optimizations had to be done in order to achieve the objectives of the project:

- Port the parallel processing engine to IBM Blue Gene/P. The main problem here is that this architecture can only do off-screen rendering and VisIt does not support this a priori. The second difficulty is that VisIt is a very complex, loosely coupled, dynamically loaded and componentized software, parts of which should run on the front-end node of the BG/P, while other parts must run on the compute nodes. This component mix introduces additional complexity and it is very time consuming to produce the correct combination.

- Implement input plug-ins that support the GROMACS, CP2K and NAMD output formats, when needed.
- Provide a methodology for the users of these packages that includes instructions on how to structure their output and best practices for how to use the package.
- Provide a methodology and reference binary package for other centers to integrate the same package in their software stack.

The objectives of the project have been achieved. A Blue Gene/P port of the parallel processing engine has been made available; it supports the required input formats. Methodologies for users and porters are also produced, along with a reference binary package.

3.2 In-Situ Visualization: State-of-the-Art and Some Use Cases

Supported by: Giusy Muscianisi, Luigi Calori and Marzia Rivi (CINECA, Italy), Vladimir Slavnic (IPB, Serbia)

In this project, tools implementing in-situ visualization have been investigated and their use in HPC applications studied. This has resulted in a whitepaper detailing the available in-situ tools and how to integrate such tools in user codes.

The term in-situ visualization means running a solver in tandem with visualization. By coupling these together, one can use the high performance computing for post-processing, while circumventing bottlenecks associated with storing and retrieving data from disk storage. Moreover, it allows monitoring simulations in-situ, performing not only visualization, but also analysis of the incoming data as it is generated, so that the simulation may be stopped or modified, thereby conserving CPU resources.

As a starting point, CINECA organized a workshop on "Visualization of Large Dataset" on the 14th and 15th June 2011 (see http://www.cineca.it/page/workshop-visualization-large-scientific-data), with the objective of bringing together researchers, developers and computational scientists for cross-training and to discuss recent developments and future advancements in remote and in-situ visualization. From this meeting it was concluded that there are two complementary approaches:

1. *Co-processing / in-situ* (*Computation and Visualization on the same nodes*). Parallel simulations are instrumented to communicate to an outside visualization tool; images are generated in parallel by the visualization code, linked to the simulation. General-purpose visualization tools supporting this approach are VisIt (*libsim*) and ParaView (*CoProcessing*).
2. *Parallel Data transfer to a remote Distributed Shared Memory* (*Computation and Post-processing physically separated*). It is aimed mostly at computational steering. It minimizes modification of existing simulation codes but requires data written using HDF5 format and usage of a ParaView plug-in (*ICARUS*) developed by CSCS. Data sent to HDF5 is automatically redirected to the CSCS driver, H5FDdsm, which interfaces ParaView.

From the workshop it was decided to investigate two specific visualization tools, namely ParaView (by CINECA) and VisIt (by IPB), which has resulted in the following:

- A survey of the visualization tools
- Tool deployment on the CINECA Linux cluster PLX

- Testing of the graphical features
- Analysis and testing of the performance with the client-server modality
- Benchmarking of features enabling MPI with and without the usage of GPUs

The above have been reported in a whitepaper, however, results are also available here: https://hpc-forge.cineca.it/trac/viz-simula/

Regarding the in-situ visualization, CINECA has tested the *ICARUS* approach by replicating an example prepared by CSCS with the Gadget2 code. Then, as use case, this technique has been applied to the Pluto code, in order to write data when required and to modify boundary conditions on the fly during simulation. IPB, in turn, has tested the *libsim* library by first instrumenting relatively simple simulations, after which, this library was used to instrument the BrainCore code as a use case for the in-situ approach supported by VisIt (see 3.3 Visualization of Output from Large-Scale Brain Simulation).

3.3 Visualization of Output from Large-Scale Brain Simulation

Supported by: Paul Melis (SARA, Holland), Vladimir Slavnic (IPB, Serbia), Kiril Alexiev (NCSA, Bulgaria)

This project was proposed by members from KTH, specifically the group of Prof. Anders Lanser at the Department of Computational Biology. The project proposal asked for PRACE support in visualizing of large-scale neuron simulations, from 50,000 neurons, up to several 100,000s. Both offline rendered animations as well as real-time interactive visualizations were targeted. For attaining real-time imagery, *in-situ visualization* of a running simulation was chosen, which is described below in more detail.

The project basically has two parts: 1) creating visualizations using large-scale simulation output from existing neural simulation codes (BrainCor, Neuron) and 2) making extensions to some of the existing codes to make in-situ visualization possible.

The first part was done by members from SARA/NCF and NCSA. As a first step, changes have been made to the file formats used, because the simulation data was written to text files and in ways that didn't allow for easy or efficient visualization with the chosen visualization application, ParaView. Simulation data was converted to XDMF (basically HDF5 for the actual data and a high-level XML file for describing that data). Furthermore, the data was split over multiple HDF5 files, allowing efficient parallel visualization. The latter is needed for interactive visualization of large-scale simulation output on the order of hundreds of thousands of neurons. In addition, simulation data was used to visualize relevant aspects of the model dynamics.

Visualization pipelines were created for different types of visualizations. Goals for the visualizations were: 1) provide insight into the model behaviour together with a view on the underlying scientific data, 2) generate synthetic brain signals based on simulation data, to compare with real-life measurements. Figure 10 shows an example image from a preliminary animation of a simulation with 50,000 neurons.

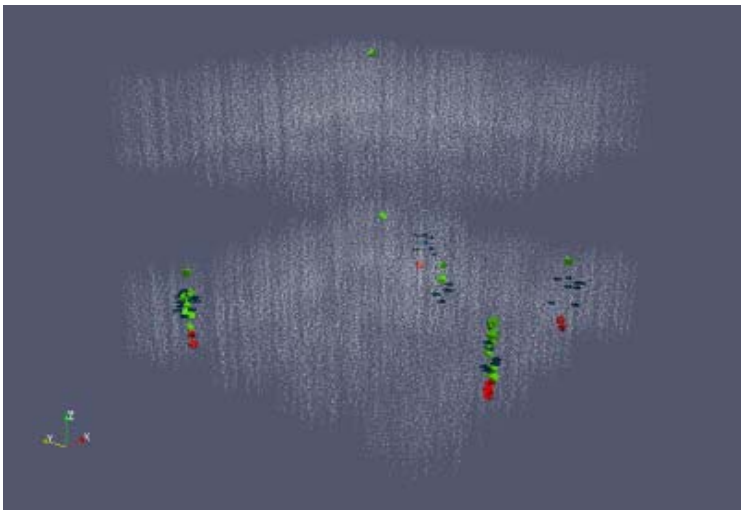


Figure 10: Visualization of simulation with 50,000 neurons

The second part of the project, dealing with in-situ visualization, presents an approach where, while a simulation is running, the data it produces is accessed and visualized (without affecting the simulation itself). By using the VisIt visualization application and its *libsim* library, simulation code can be instrumented so that VisIt can connect to a running simulation and access simulation data directly. This eliminates large amounts of disk I/O, as the simulation data is never written out to files on disk (at least not for visualization purposes). Furthermore, it provides insight into a running simulation, so progress and correctness can be checked on the fly.

IPB performed testing and analyzing of the simplified BrainCore simulation code provided by the developer (Simon Benjaminsson, KTH). In cooperation with the developers, relevant parts of the code for efficient implementation of in-situ visualization using VisIt were identified. Convenient types of data representation were chosen so as to visualize activity of simulated neuron units in a more natural, comprehensive way. The code was instrumented and tested on different clusters, where control of simulation was demonstrated and in-situ visualization of neural unit data was performed.

4 Long-Term Preservation of Applications Data

Applications on Tier-0 systems are able to produce large amounts of output data and preserving such data reliably for a long time can be important. This section summarizes whitepapers written for users of tasks 7.1 and 7.2, wanting to preserve application results. The whitepapers cover different mediums and tools suitable for storing data, use-case analysis, best practices and strategies for long-term preservation of data.

4.1 The Vagn-Ekman Case Study at SNIC-NSC

Contributors: Johan Raber and Per Lundqvist (SNIC-NSC, Sweden)

Vagn-Ekman is a dual cluster setup with specialized functionality in the two parts. Ekman is a large compute cluster located in Stockholm, Sweden, consisting of 1268 compute nodes sporting 8 CPU cores per node and a high bandwidth, low latency Infiniband interconnect. Vagn is a storage and post-processing cluster located at NSC, Linköping, with nodes tailored for analysis of the raw data produced at Ekman.

The Vagn-Ekman cluster duo resembles to an extent future PRACE operations, in the sense that from a large data production facility there will be a need to conveniently transfer the produced data to the researchers, potentially scattered across Europe, in a safe manner with respect to data integrity. The experiences and tools developed during the Vagn-Ekman project can serve as an example for how this data flow can be carried out.

The very large volumes of data produced at Ekman needs to be safely transferred to the Vagn cluster, since Ekman does not have the capability to harbour large quantities of data in the long term. To provide data integrity checks and convenience to the end user, a special purpose tool was built, FFV the File Transfer Tool.

The trivial synchronous procedure for moving data between hosts goes like this: i) log in and ii) copy local source files to remote destination or vice-versa; iii) on a successful copy operation, delete the source and iv) log out.

An application that automatically handles the above steps needs to handle at least the following issues:

- *Interrupts*: There is a risk of interrupts due to system failures or network failures which can be very high over long periods of time. A normal synchronously initiated move requires the user to be logged in while moving and is as such also vulnerable to anything that interrupts the initiating process.
- *Errors*: When moving data, users must themselves make sure that the data was transferred correctly to the other host before deleting the original. It is a seemingly trivial operation, but in a scripted environment, with many users and over a long time span, there is a significant risk for human mistakes.
- *Restarts*: Although restarting of file transfers is in a sense trivial, the information on how to restart, e.g. search paths and authentication, has to be provided by the users themselves. It is also the responsibility of the user not to forget to restart and restart the correct way with respect to search paths. Potentially, this could be a task of some complexity, depending on the error causing the restart and the amount of restarts.
- *Resuming*: The larger the file sets, the higher the risk of being interrupted when

moving it (see *Interrupts*). If the transfer tool does not support resuming, moving a large file set risks being starved out, taking a very long time to finish or even never finishing at all,. Resuming from the last file not already transferred guarantees that very large file sets eventually will complete, even when being frequently interrupted.

- *Isolation*: For performance reasons, no more than one process should move the same data to a remote host. Ideally, the original data should be immutable or the user should be able to detect if it has been modified.

FFV (http://www.nsc.liu.se/~perl/ffv/) was developed for the purpose of moving data between Ekman and Vagn and was designed to handle the problems mentioned above. The basic design idea is simple: 1) Move the data to be transferred into a hidden subdirectory of its parent directory; 2) make a hidden transfer directory on the target side, and 3) repeatedly sync this directory to its remote (hidden) location leveraging the rsync application from a self contained job script run via cron until considered finished; then 4) move the hidden target directory to its final, visible location, and 5) remove the local transfer directory.

Using FFV, the typical workflow can be summarized as:

1. A user submits an FFV job. The minimal information supplied to FFV is the source directory, target directory and references to necessary credentials.
2. FFV does minimal verification that the user request may succeed, e.g. establish a connection to any remote system, validate paths, etc. When using public key authentication through an SSH agent, it establishes an SSH session with an SSH control socket to be used, such that the agent does not need to be contacted again, unless something causes the job to pause. Lastly, it moves the original data into a transfer directory within the parent directory.
3. A self-contained transfer job script is automatically created on permanent storage, containing all information needed to move the requested data to the target location.
4. The user is supplied with a handle that acts as a FFV job identification token, which can be used to query FFV about the transfer status, yielding a brief report on progress or possibly problems and solutions.
5. The self-contained job script is then executed asynchronously via cron, the prompt is returned to the user, and FFV requires no further input unless an exceptional event occurs, such as expired credentials, or other non-transient errors requiring user intervention. When necessary, the user is notified either by mail or to a file in the users home directory, on the system where the job was created.

At its core, FFV is an application automating best practices in data transfer, all the while providing convenience and error checking to the user. Anyone can replicate this functionality, provided they have the scripting or programming skills necessary. However, learning best data transfer practises and automating them should not be required of the user, since it is usually not their core competence and this provided the impetus to make the file transfer tool.

4.2 Best Practices on Standards, Policies and Quality Assurance in Digital Repositories for Long Term Preservation

Contributors: Olivier Rouchon, Philippe Prat and Mathieu Cloirec (CINES, France)

During the past twenty years, the long-term preservation of digital information has only been

a matter under consideration for a few scientific or patrimonial institutions. These have played a key role in the understanding of the subsequent risks and the definition of standards in this domain. The best practices rely on four technological risks which are now commonly agreed: the loss of the knowledge of the content, file format obsolescence, aging media causing data loss, and sudden software or technology changes. They have been put in place in institutions dealing with text data, images, sounds or video. How does that translate into raw, primary data produced by Tier-0 systems, has been evaluated in this project.

The data created, handled, processed, stored, exchanged and distributed in our society is mainly in digital form. This form of representing data is incredibly powerful and the storage costs are going lower and lower; it is now possible to preserve them without any alteration whatsoever, and tools exist for creating complex documents and finding useful information in them. And yet behind all these huge advantages, lies a major risk; that of severe vulnerability to time, explicable and identified. This vulnerability lies in the inevitability of one or more risks linked to the nature of the data itself, if no preventive measures are taken.. The main methods to be implemented to mitigate and manage these risks are now proven:

- The use of persistent metadata and identifiers to preserve knowledge of content.
- The choice of sustainable file formats to keep control and the ability to migrate to new formats (when conversion is the preservation strategy).
- Proactive management of the ageing of storage media for the ability to properly preserve the bit stream making up the files and migrating them to new supports.
- Permanent technological watch and anticipation of technological change.

The quality approach via best practices in this field can be seen from two aspects, technical and organisational. The technical quality approach to preserving digital documents covers all the procedures aiming at ensuring a high level of quality of the digital object itself. It can be divided into three levels: metadata, file formats and storage. There are some specifics in the high-performance computing domain, with those three aspects, due to the nature of the data produced by researchers, the volumes generated, etc.

From an organisational point of view, activity management based on risks should be advocated. The preservation of digital documents is a project like any other, and like any other activity, generates risks. The aim is not to eliminate the risks, but to determine an acceptable level of risk. This is a well-defined and proven method. Such a methodology originated in industry and is now widespread in the management and service sectors.

Certification is the culmination of consolidating an organisation and/or service. It signals recognition of quality and professionalism, and is therefore a means of instilling trust with communities of users, and may also be a way of leveraging budgets from governing bodies. Although, this is a very weighty project, both in the human investment that is required to manage the project or the changes required, and in the financial investment, as regular external audits needs to be planned. As a result, it is important to identify the type of certification that will have the greatest impact, both on the community of users and on the governing bodies.

Awareness of the need to implement standards, and respect for certain best practices, in particular regarding a quality approach to long-term preservation, is not neutral. It represents a considerable and immediate investment, the effects of which are only perceptible in the long-term; moreover, such an initiative, and its culmination in certification, requires the commitment of everyone involved in the preservation process. However, quality indicators and measurements are not yet clearly defined, with little hindsight available to the institutions implementing it to measure the positive effects.

4.3 Storage and Long Term Preservation Strategies in PRACE Tier-1 Datacentres

Contributors: Huub Stoffers, Mark van de Sanden (SARA), Johan Raber, Tom Lanborg (SNIC), George Tsouloupas (CaSToRC), Olivier Rouchon, Florent Marceteau (CINES)

SARA, SNIC-NSC, CaSToRC and CINES are Tier-1 datacenters involved in HPC activities, and particularly active on EU-funded projects such as DEISA, PRACE and EUDAT. In this project, people from across these centers have worked together to document common long term preservation strategies at their centers.

SARA's HPC Central Archive (CA) is a mass storage facility for long-term preservation of large data sets produced in academic research projects that are of particular relevance to (Dutch) academic communities. To be more specific about the groups that can benefit from the service: the CA currently stores data that were produced on several incarnations of the Dutch supercomputing facility and various other academic HPC facilities at SARA.

Swestore is a Swedish national storage service on the infrastructure level, making storage space available over the internet for academics. The aim of the Swestore is to build a robust, flexible and expandable storage system distributed across all six SNIC centers and nationally accessible, which can be used in most cases where access to large scale storage is needed.

CINES operates state of the art computer services in high-performance computing, long term data and digital document preservation, as well as the hosting of computer equipments. The center provides researchers from universities and public research institutes with high performance parallel computing and storage platforms. The staff of the HPC department at CINES makes hardware and software available to the users and provides specific assistance for research modeling.

The Cyprus Institute is developing a Computation-based Science and Technology Research Center (CaSToRC) that will include a Tier-1 HPC and storage facility. CaSToRC aspires to encourage the use of high performance computing in Cyprus and the Eastern Mediterranean region and to serve the needs for HPC and data intensive computing in research, by providing adequate computing and storage resources to enable Cyprus and the Eastern Mediterranean research community to pursue forefront computing-related research.

These four actors have all put in place strategies to store and preserve academic data, produced as part of research projects. Even if they are fairly similar, the policies and technologies that have been deployed have a few differences which have been detailed. The way they address the increasing need for long time archival storage, in combination with the ever-increasing size and rate of research data produced has also been described, as well as data sharing problems in joint research efforts where large data sets need sharing.

4.4 Media and Technology Appraisal for Long-Term Preservation

Contributers: Huub Stoffers, Mark van de Sander (SARA), Johan Raber, Tom Langborg, Per Lundqvist, Bengt Persson, Torben Rasmussen (SNIC), Nick Sinanis (CastorC), Olivier Rouchon, Florent Marceteau (CINES)

Reliability, performance, costs and return on investment are key factors in the long-term preservation of digital data. They differ from one technology to another. Since there are different media and technologies used for storage and transfer, this whitepaper has done a comparison, with a particular focus on disks and tapes. Long-term preservation technologies

are constantly evolving, making it necessary to anticipate and adapt to the various technological advances, thus enabling quality data preservation services. Loss of data is not an option, even after ten or twenty years. Unfortunately, no support is 100% reliable and thus technologies to anticipate failures are useful, as well as a suitable environment. Reliable equipment and the associated technologies imply a cost, which depends on the chosen technology.

The environment also has an important impact on the long-term equipment reliability. Different factors can affect disks and tapes. Humidity, dust, electromagnetism and temperature are the most common ones. Of course, the age of the equipment affects the reliability. But its impact is not linear and depends on the chosen equipment. Disks can present problems with early failures during the first months. But the reliability problems are more present after a use period of five years, where the disks start to become too old. Between one year and five years, the failure rate appears to be relatively constant. Then, the risk of failures increases. This is the reason why five years can be considered as the lifetime of a disk.

The intensive use of disks can also be considered as a risk factor for reliability, but its impact is less important than expected and only plays a role in the early failure and during the fifth year of a disk lifetime. Tapes on the other hand appear to be less time sensitive, with a much higher reliability when the environment is adequate.

Disk are complex equipment, with numerous components that can cause failure. In order to anticipate them, the technology SMART (Self-Monitoring Analysis and Reporting Technology) allows the study of different parameters relevant for reliability. These parameters (e.g. scan errors and reallocation counts) are indicators of the reliability. The risk of failure increases drastically when they are present.

All equipment and technologies have a cost. The comparison between disk and tape shows that tape is the less expensive equipment for large amounts of data. Because tapes require drivers to be used, and because these are fairly expensive, disks can be cost-effective for small amount of data. Similarly, the equipment cost for an important amount of data is more important for disks than tapes. From an energy point of view, disks produce heat, and some components require constant power supply. Tapes, on the other side, only require an appropriate environment. For these reasons, the costs of energy for disks are higher than for tape.

Considering reliability and costs, tapes can be considered as the most interesting equipment for long-term preservation of data. But since no equipment is 100% reliable, it is important to have several copies of each data. In general, it is preferable to make those copies on different media.

The above topics are covered in more detail in the whitepaper.

4.5 The “Jonker Case” / After Care: Handling the “ENTRAIN” Dataset after its Production on Jugene

Supported by: Huub Stoffers (SARA, Holland)

Collaborators: John Donners and Mark van de Sanden (SARA, Holland)

A project of the Dutch investigator Harm Jonker, of the Delft Technical University (NL), titled “Providing fundamental laws for weather and climate models”, was granted 35,000,000

core hours on PRACE Tier-0 resource Jugene, as part of the *'PRACE Project Access'* program. The core hours have been used to run a massively parallel simulation program which produced a raw dataset of approximately 13 TB. The project proposal explicitly states that the produced data should be made publicly available to serve as a benchmark for atmospheric models. However, no provisions were made, except to produce the data. After the data was produced on the Tier-0 system, the project was in principle finished as far as PRACE was concerned — the data had to be moved out from the Jugene file system as soon as possible.

The raw data had to be further analysed and post-processed, including a conversion to a more suitable standardized format for making the data publicly available. Programs for this are generally not massively parallel, so a BlueGene/P would not be the platform of choice for this type of work. SARA committed to "adopting" the data set and taking care of the trajectory that comes *after* the data production phase:

- Via the Dutch PRACE tier-1 system "Huygens" the data has been moved to the SARA central archive facility.
- "Huygens" is available to the investigators to do their post-processing.

The whitepaper reports on the work that has been done to accommodate the project. It illustrates that a considerable amount of improvisation was needed to accommodate rather natural needs of a user using PRACE computational facilities to produce quasi-empirical data that have to be explored further. The paper notes that at present, PRACE lacks well-defined support and guidance for users, which goes beyond the stage that allocated compute cycles on the Tier-0 resource that was used. It also includes a short discussion on how to improve PRACE procedures or arrangements and "logistics" for handling similar cases.

The whitepaper was written in February 2011, when the choices on how to arrange the public availability of the dataset still had to be made. Work in this "after care" trajectory is still on-going. Meanwhile, options have been discussed with Dr. Jonker. A node, publicly accessible by https, equipped to access the SARA archive, has been setup. It will be used for this and other projects. The data are being converted to NetCDF format. A pilot will be setup, using the OpenDAP protocol, for read-only disclosure of the dataset of this project to the public.

5 Summary

During its relatively short 18-month period, task 7.6 has successfully provided support to a wide range of HPC codes and user communities. Experts from task 7.6 have supported in total 12 applications with solving their data challenges. The applications which were supported were all selected in tasks 7.1 and 7.2 and are listed below:

Alya, BrainCore, Code_Saturne, CP2K, Elmer, EUTERPE, Gromacs, NEMO, OpenFoam, Pluto, SPECFEM3D and Vlasiator.

For each of these applications there has typically been several data optimization subprojects lead by teams collaborating from several different PRACE member HPC centers around Europe. This work has resulted in 23 whitepapers, which have been summarized in this deliverable. The whitepaper subjects range from detailed descriptions of implemented parallel algorithms to user guides for Tier-0 I/O subsystems and tools useful for handling of Petascale data. All whitepapers are available online and are useful for users facing similar data challenges.

In summary, task 7.6 has had a positive impact on a wide variety of applications important to European users. Furthermore, it has fostered close collaboration between researchers from across European HPC centers, which might otherwise not have happened was it not for PRACE.