SEVENTH FRAMEWORK PROGRAMME
Research Infrastructures

INFRA-2011-2.3.5 – Second Implementation Phase of the European High Performance Computing (HPC) service PRACE

PRACE-2IP

PRACE Second Implementation Project

Grant Agreement Number: RI-283493

D12.2
Exploration of Scalable Numerical Algorithms

Final

Version:	1.0
Author(s):	Cevdet Aykanat, Ata Turk (Bilkent University)
Date:	27.08.2012

Project and Deliverable Information Sheet

PRACE Project	**Project Ref. №: RI-283493**	
	Project Title: PRACE Second Implementation Project	
	Project Web Site:http://www.prace-project.eu	
	Deliverable ID: <**D12.2**>	
	Deliverable Nature: < Report>	
	Deliverable Level: PU *	**Contractual Date of Delivery:** 31 / August / 2012
		Actual Date of Delivery: 31 / August / 2012
	EC Project Officer: Leonardo Flores Añover	

* - 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: Exploration of Scalable Numerical Algorithms**	
	ID:D12.2	
	Version:<1.0>	**Status:** Final
	Available at: http://www.prace-project.eu	
	Software Tool: Microsoft Word 2007	
	File(s): D12.2.docx	
Authorship	**Written by:**	Cevdet Aykanat, Ata Turk (Bilkent University), Ata Turk
	Contributors:	
	Reviewed by:	Micael Oliveira, Coimbra; Dietmar Erwin, JUELICH
	Approved by:	MB/TB

Document Status Sheet

Version	**Date**	**Status**	**Comments**
0.1	25/07/2012	Draft	First draft
0.2	24/08/2012	Draft	Second Draft
1.0	27/08/2012	Final version	

Document Keywords

Keywords:	PRACE, HPC, Research Infrastructure

Disclaimer

This deliverable has been prepared by the responsible Work Package of the Project in accordance with the Consortium Agreement and the Grant Agreement n°RI-283493. It solely reflects the opinion of the parties to such agreements on a collective basis in the context of the Project and to the extent foreseen in such agreements. Please note that even though all participants to the Project are members of PRACE AISBL, this deliverable has not been approved by the Council of PRACE AISBL and therefore does not emanate from it nor should it be considered to reflect PRACE AISBL's individual opinion.

Copyright notices

© 2012 PRACE Consortium Partners. All rights reserved. This document is a project document of the PRACE project. All contents are reserved by default and may not be disclosed to third parties without the written consent of the PRACE partners, except as mandated by the European Commission contract RI-283493 for reviewing and dissemination purposes.

All trademarks and other rights on third party products mentioned in this document are acknowledged as own by the respective holders.

Table of Contents

List of Figures

List of Tables

References and Applicable Documents

[1] U. V. Catalyurek and C. Aykanat, “Hypergraph-Partitioning-Based Decomposition for Parallel Sparse-Matrix Vector Multiplication”, IEEE Transactions on Parallel and Distributed Systems, vol. 10, no. 7, pp. 673-693, 1999
[2] T. A. Davis, “University of Florida sparse matrix collection”, NA Digest, 1997.
[3] U. V. Çatalyürek, C. Aykanat, and B. Uçar, “On two-dimensional sparse matrix partitioning: Models, methods, and a recipe”, SIAM Journal on Scientific Computing, Vol. 32, pp. 656--683, 2010.
[4] L. N. Trefethen and D. Bau, Numerical Linear Algebra, SIAM, 1997.
[5] M. Benzi, “Preconditioning Techniques for Large Linear Systems: A Survey”, Journal of Computational Physics, 182 (2002), pp. 418 – 477.
[6] R. Barret, J. Demmel, J. Dongarra, “Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods”, SIAM, 1994.
[7] S. Balay, J. Brown, K. Buschelman, “PETSc Users Manual”, ANL-95/11 – Revision 3.3, 2012.
[8] M. A. Heroux, R. A. Bartlett, V. E. Howle, “An Overview of the Trilinos Project”, ACM Transaction on Mathematical Software, vol. 31, pp. 397-423, 2005.
[9] TOP500 PROJECT, www.top500.org
[10] M. Hoemmen, “Communication Avoiding Krylov Subspace Methods”, Ph.D. thesis, 2010.
[11] J. Demmel, M. Hoemmen, M. Mohiyuddin and A. Yelick, “Avoiding Communication In Computing Krylov Subspace”, Technical Report No. UCB/EECS-2007-123, University of California, Berkeley, 2007.
[12] IBM PLX, www.hpc.cineca.it/hardware/ibm-plx
[13] IBM BG/Q FERMI, www.hpc.cineca.it/hardware/ibm-bgq-fermi
[14] G. A. A. Kahou, E. Kamgnia, and B. Philippe, “Parallel Implementation of an Explicit Formulation of the Multiplicative Schwarz Preconditioner”, in CdROM Proceedings of IMACS05, 2005.
[15] M. Naumov, M. Manguoglu, and A. H. Sameh, “A Tearing-Based Hybrid Parallel Sparse Linear System Solver”, Journal of Computational and Applied Mathematics, 234 (2010), pp. 3025 – 3038.
[16] G. A. A. Kahou, E. Kamgnia, and B. Philippe, “An Explicit Formulation Of The Multiplicative Schwarz Preconditioner”, Applied Numerical Mathematics, 57 (2007), pp. 1197 – 1213.
[17] G. A. A. Kahou, L. Grigori, and M. Sosonkına, “A Partitioning Algorithm for Block-Diagonal Matrices With Overlap”, Parallel Computing, 34 (2008), pp. 332 – 344.
[18] M. Manguoglu, “A Domain-Decomposing Parallel Sparse Linear System Solver”, Journal of Computational and Applied Mathematics, 236(3), pp.319-325, 2011.
[19] T. A. Davis, “University Of Florida Sparse Matrix Collection,” NA Digest, 1997.
[20] A. Bhatele, “Automating Topology Aware Mapping for Supercomputers”, PhD Thesis,
[21] The TopoManager API: http://charm.cs.uiuc.edu/research/topology
[22] R.W. Hockney, “A Fast Direct Solution Of Poisson’s Equation Using Fourier Analysis”, Journal of Asso. Comput. Mach, v 8, 1965.
[23] B.L. Buzbee, G.H Golub, C.W. Nielson, “On Direct Methods for Solving Poisson’s Equation”, SIAM J. Numerical Analysis, v 7, 1970.
[24] P. Giannozzi, S. Baroni, N. Bonini, et al., “Quantum Espresso: A Modular and Open-Source Software Project for Quantum Simulations Of Materials”, J. Phys.: Condens. Matter 21, 2009
[25] FFTE: A Fast Fourier Transform Package official web site: http://www.ffte.jp/

[26] M. Frigo, S. G. Johnson, "The Design and Implementation of FFTW3", Proceedings of the IEEE **93** (2005) 216
[27] A. Sunderland, S. Pickles, M. Nikolic, et al., “An Analysis of FFT Performance in PRACE Application Codes”, PRACE-1IP T7.5 Whitepaper (2012)
[28] Developer’s Manual for Quantum ESPRESSO (v. 5.0) http://www.quantum-espresso.org/wp-content/uploads/Doc/developer_man.pdf
[29] QE-SCL, Quantum Espresso extended with FFTE library and FFTW3 threading support http://www.scl.rs/QE-SCL/
[30] Quantum Espresso official web site http://www.quantum-espresso.org/
[31] NIIFI SC, http://www.niif.hu/en/niif_institute/supercomputing_service/
[32] F. Spiga, Implementing and Testing Mixed Parallel Programming Model into Quantum ESPRESSO, Science and Supercomputing in Europe – research highlights 2009, ISBN 978-88-86037-23-5, pp. 6
[33] P. Kent, “Computational Challenges Of Large-Scale, Long-Time, First Principles Molecular Dynamics”, Journal of Physics: Conference Series, vol. 125, no. 1, p. 012058, 2008. [Online]. Available: http://stacks.iop.org/1742-6596/125/i=1/a=012058
[34] T. Auckenthaler, V. Blum, H. J. Bungartz, T. Huckle, R. Johanni, L. Krämer, B. Lang, H. Lederer, and P. R. Willems, “Parallel Solution Of Partial Symmetric Eigenvalue Problems From Electronic Structure Calculations”, Parallel Comput., vol. 37, no. 12, pp. 783–794, Dec. 2011. [Online]. Available: http://dx.doi.org/10.1016/j.parco.2011.05.002
[35] A. Haidar, H. Ltaief, and J. Dongarra, “Parallel Reduction To Condensed Forms For Symmetric Eigenvalue Problems Using Aggregated Fine-Grained And Memory-Aware Kernels”, in SC11: International Conference for High Performance Computing, Networking, Storage and Analysis, Seattle, WA, USA, November 12-18 2011.
[36] http://user.cscs.ch/hardware/castor_ibm_idataplex/index.html
[37] X. S. Li, J. W. Demmel, J. R. Gilbert, L. Grigori, M. Shao, and I. Yamazaki, “SuperLU Users' Guide”, Tech. Report UCB, Computer Science Division, University of California, Berkeley, CA, 1999, update: 2011.
[38] P. R. Amestoy, I. S. Duff, J.-Y. L’Excellent, and J. Koster, “A Fully Asynchronous Multifrontal Solver Using Distributed Dynamic Scheduling”, SIAM J. Matrix Anal. Appl., 23 (2001), pp. 15–41.
[39] O. Schenk and K. Gartner, “Solving Unsymmetric Sparse Systems of Linear Equations with Pardiso”, Future Generation Computer Systems, 20 (2004), pp. 475-487.
[40] O. Schenk and K. Gartner, “On Fast Factorization Pivoting Methods For Sparse Symmetric Indefinite Systems”, Electronic Transactions on Numerical Analysis, 23 (2006), pp. 158 – 179.
[41] A. Duran and B.D. Saunders, Gen_SuperLU package (version 1.0, August 2002).
[42] A. Duran, B. D. Saunders and Z. Wan, “Hybrid Algorithms For Rank Of Sparse Matrices”, Proceedings of the SIAM International Conference on Applied Linear Algebra (SIAM-LA), VA, July 15-19, 2003.
[43] L. Li, L. Li and Y. Guangwen, “A Highly Efficient Gpu-Cpu Hybrid Parallel Implementation Of Sparse Lu Factorization”, Chinese J. of Electronics, 21:7-12, 2012.
[44] J.W. Demmel, J.R. Gilbert, and X.S. Li, “An Asynchronous Parallel Supernodal Algorithm For Sparse Gaussian Elimination”, SIAM J. Matrix Analysis and Applications, 20(4):915-952, 1999.
[45] X.S. Li, “Evaluation Of Sparse Lu Factorization And Triangular Solution On Multicore Platforms”, VECPAR 2008, Springer.
[46] X. S. Li and J. W. Demmel, “Superlu-Dist: A Scalable Distributed-Memory Sparse Direct Solver For Unsymmetric Linear Systems”, ACM Trans. Math. Softw., 29 (2003), pp. 110–140.
[47] L. Grigori, J.W. Demmel, and X.S. Li. “Parallel Symbolic Factorization For Sparse Lu With Static Pivoting”, SIAM J. Scientific Computing, 29(3):1289-1314, 2007.
[48] M.S. Celebi, A. Duran, M.Tuncel, B. Akaydin, “Scalability of SuperLU Solvers for Large Scale Complex Reservoir Simulations”, SPE and SIAM Conference on Mathematical Methods in Fluid Dynamics and Simulation of Giant Oil and Gas Reservoirs, Istanbul, Turkey, September 3-5, 2012.
[49] I.S. Duff and J. Koster, “The Design and Use of Algorithms for Permuting Large Entries to the Diagonal of Sparse Matrices”, SIAM J. Matrix Anal. Appl. 20 (4) (1999) 889–901.
[50] G. Karypis, K. Schloegel, and V. Kumar, “ParMeTiS, version 3.1”, http://www-users.cs.umn.edu/~karypis/metis/parmetis/.
[51] Karypis and V. Kumar, “MeTiS, version 4.0”, http://www-users.cs.umn.edu/~karypis/metis/.
[52] http://www.uybhm.itu.edu.tr/eng/inner/duyurular.html#karadeniz
[53] nPartition Administrator's Guide, HP part number: 5991-1247B, 1st Edition, February 2007, Hewlett-Packard Development Company.
[54] DL_POLY Molecular Simulation Package home page:
http://www.stfc.ac.uk/CSE/randd/ccg/software/DL_POLY/25526.aspx
[55] The DL_POLY_4 User Manual:
ftp://ftp.dl.ac.uk/ccp5/DL_POLY/DL_POLY_4.0/DOCUMENTS/USRMAN4.01.pdf

[56] The DL_POLY test data: ftp://ftp.dl.ac.uk/ccp5/DL_POLY/DL_POLY_4.0/DATA/

[57] GeForce 400 series: http://en.wikipedia.org/wiki/GeForce_400_Series

[58] AMD Radeon HD 6900: http://www.amd.com/us/products/notebook/graphics/amd-radeon-6000m/amd-radeon-6900m/Pages/amd-radeon-6900m.aspx

[59] PRACE-1IP Public deliverable - HPC Programming Techniques: http://www.prace-ri.eu/IMG/pdf/d7.5_1ip.pdf

[60] U. Frish, et al., "Lattice-Gas Automata for the Navier-Stokes Equation", Phys. Rev. Lett. Vol 56: 1505-1508 (1986).

[61] O. Bandman, "Using Cellular Automata for Porous Media Simulation", The Journal of Supercomputing 57(2): 121-131 (2011)

[62] Y. Arai, et al., "A Latice Gas Cellular Automata Simulator on the Cell Broadband Engine", Parallel Computing: Architectures, Algorithms and Applications Vol. 38: 459-466 (2007).

[63] N. Kosturski, S. Margenov, and Y. Vutov, "Balancing the Communications and Computations in Parallel FEM Simulations on Unstructured Grids", Parallel Processing and Applied Mathematics, LNCS, vol. 7204, pp 211-220, 2012

List of Acronyms and Abbreviations

AMD	Advanced Micro Devices
AMG	Algebraic MultiGrid
API	Application Programming Interface
BDO	Block-Diagonal with Overlap
BiCGStab	Bi-Conjugate Gradient Stabilized
BLAS	Basic Linear Algebra Subprograms
BSC	Barcelona Supercomputing Center (Spain)
CEA	Commissariat à l'EnergieAtomique (represented in PRACE by GENCI, France)
CFD	Computational Fluid Dynamics
CINECA	ConsorzioInteruniversitario, the largest Italian computing centre (Italy)
CN	Column-Net
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)
CT	Computer Tomography
CUDA	Compute Unified Device Architecture (NVIDIA)
DEISA	Distributed European Infrastructure for Supercomputing Applications.EU project by leading national HPC centres.
DMA	Direct Memory Access
DNA	DeoxyriboNucleic Acid
DRAM	Dynamic Random Access memory
DSA	Digital Subtraction Angiography
EC	European Community
EESI	European Exascale Software Initiative
EPCC	Edinburg Parallel Computing Centre (represented in PRACE by EPSRC, United Kingdom)
ETHZ	Eidgenössische Technische Hochschule Zuerich, ETH Zurich (Switzerland)
ESFRI	European Strategy Forum on Research Infrastructures; created roadmap for pan-European Research Infrastructure.
FEM	Finite Element Method

FETI	Finite Element Tearing and Interconnect
FFT	Fast Fourier Transform
FP	Floating-Point
FPU	Floating-Point Unit
FZJ	Forschungszentrum Jülich (Germany)
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)
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 CalculIntensif (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
GP	Graph Partitioning
GPGPU	General Purpose GPU
GPU	Graphic Processing Unit
GS	Gram-Schmidt
GWU	George Washington University, Washington, D.C. (USA)
HDD	Hard Disk Drive
HE	High Efficiency
HECToR	High-End Computing Terascale Resource
HMPP	Hybrid Multi-core Parallel Programming (CAPS enterprise)
HP	Hypergraph Partitioning
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
IBM	Formerly known as International Business Machines
IDRIS	Institut du Développementet des Ressources en Informatique Scientifique (represented in PRACE by GENCI, France)
IEEE	Institute of Electrical and Electronic Engineers
IESP	International Exascale Project
IPB	Institute of Physics Belgrade
IMB	Intel MPI Benchmark
I/O	Input/Output
ITU-UHeM	Istanbul Technical University – Center of High Performance Computing
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 (= $2^{20} \sim 10^6$) Bytes (= 8 bits), also MByte
MB/s	Mega (= 10^6) Bytes (= 8 bits) per second, also MByte/s
MDT	Meta Data 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)
Mop/s	Mega (= 10^6) operations per second (usually integer or logic operations)
MoU	Memorandum of Understanding
Mups	Million lattice site updates per second
MPI	Message Passing Interface
NCSA	The National Centre for Supercomputing Applications (Sofia, Bulgaria)
oGPVS	ordered Graph Partitioning by Vertex Separators
OpenCL	Open Computing Language
OpenGL	Open Graphic Library
Open MP	Open Multi-Processing
OS	Operating System
OSS	Object Storage Server
OST	Object Storage Target
oVS	ordered Vertex Seperator
PSNC	Poznan Supercomputing and Networking Center
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)
QE	Quantum Espresso
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
RN	Row-Net
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 Multi Processing
SNIC	Swedish National Infrastructure for Computing (Sweden)
SP	Single Precision, usually 32-bit floating point numbers
SpMxV	Sparse Matrix Vector multiplication
SPU	Synergistic Processor Unit (in each SPE)
SSD	Solid State Disk or Drive
SSE	Streaming SIMD Extensions
STFC	Science and Technology Facilities Council (represented in PRACE by EPSRC, United Kingdom)
STRATOS	PRACE advisory group for STRAtegic TechnOlogieS
TB	Tera (= 240 ~ 1012) Bytes (= 8 bits), also TByte
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
UFL	University of Florida
UNICORE	Uniform Interface to Computing Resources. Grid software for seamless access to distributed resources.
VSB	Technical University of Ostrava
WCSS	Wrocławskie Centrum Sieciowo-Superkomputerowe (Wrocław Centre for Networking and Supercomputing)
WP	Work Package

Executive Summary

Work Package 12 (WP12) "Novel Programming Techniques" performs research and development in four key areas for future multi-petascale and exascale systems. The work in WP12 focuses on auto tuned and automatic techniques to be applied in parallel programming model runtimes (Task 12.1: "Auto-tuned runtime Environments"), performance tools (Task 12.3: "Development environments and tools") and file systems (Task 12.4: "File system optimization"). Furthermore, as it is widely accepted that the key to exploiting future high-end systems will be based on research on new numerical algorithms as well as advancing the parallel processing technology used for higher scalability in numerical applications; consequently WP12 also focuses on research studies exposing more scalability for numerical algorithms (Task 12.2: "Scalable numerical algorithms").

The key research topics investigated in Task 12.2 are:

- Reducing Synchronization Overhead in Iterative Solvers
- Enhancing Parallel Hybrid Sparse Solvers for Scalability
- Topology-Awareness
- Enabling Hybridization in Heterogeneous Architectures
- Application Scalability

Task 12.2 evaluated different algorithms, methods, and approaches and demonstrated the scalability of the algorithms using simple ad-hoc programs. In total 17 complementary areas organized as individual *projects* were covered. This document is a summary of the projects' results. It contains high-level summaries of all projects as well as brief description of covered topics. In most of the summaries, links to PRACE white papers or scientific publications are given for those readers that are interested in more detailed information.

1 Introduction

It is widely accepted that new numerical algorithms will be key to exploit future high-end systems and the focus of this deliverable is on exploring new algorithms and methods for enhancing the scalability of parallel numerical algorithms to meet the demands of petascale computing. The aim of Task 12.2 "Exploration of Scalable Numerical Algorithms" is to address the scalability of parallel numerical algorithms by focusing research on the following key topics:

- *Reducing Synchronization Overhead in Iterative Solvers*: Exploring new numerical algorithms and parallel computing methodologies to reduce the synchronization overhead in parallel iterative solvers,
- *Enhancing Parallel Hybrid Sparse Solvers for Scalability:* Exploiting algorithms and solvers that contain both direct and iterative components to reduce the global synchronization overhead by efficient and effective preconditioning that decrease the number of iterations while increasing the amount of computation per iteration,
- *Topology-Awareness*: Achieving topology-awareness in task-to-processor mapping to minimize the adverse effect of large processor-to-processor distances on communication overhead in large-scale parallel systems,
- *Enabling Hybridization in Heterogeneous Architectures*: Enabling hybridization of numerical operations and kernels via combining MPI programming paradigm with OpenMP-, threading-, CUDA- and OpenCL-based approaches,
- *Application Scalability*: Facilitating the usage and the embedding of state-of-the-art parallel processing technology to some target applications for enhancing scalability.

The focus of this deliverable is on the algorithms and methods but not on their integration on specific production applications. However, along with simple ad-hoc programs demonstrating the scalability of the algorithms, whenever possible, efforts towards initial embeddings of the proposed schemes to existing and widely used scientific applications are performed as well.

The deliverable itself is quite concise in order to allow people to easily identify the projects that are of particular interest for them and to encourage further reading in the accompanying white papers or the referred publications.

The rest of the deliverable is organized as follows. Chapter 2 provides a description of the organization of Task 12.2. Brief discussions on the main research topics as well as brief overviews of the Task 12.2 projects categorized according to these key topics are given in Chapter 3. Chapter 4 provides detailed descriptions of the Task 12.2 projects along with obtained results and discussions. Chapter 5 presents the conclusions.

2 Task Organization

Table 1 displays a rough per partner PM distribution of Task 12.2. Around 130 PMs were allocated for this work and experts from nine countries are participating in Task 12.2. In total 17 projects have been completed. Most of the projects have been carried out with 4-7 PMs.

Country	Partner	PMs
France	IDRIS	6.0
UK	EPCC	7.0
Switzerland	ETHZ	14.0
Poland	WCSS	7.0
Turkey	UHeM-ITU	12.0
	Bilkent	36.0
Bulgaria	NCSA	12.0
Czech Republic	VSB	25.0
Serbia	IPB	4.0
Italy	CINECA	6.0
	Sum	129

Table 1: Overview of efforts per partner

List of projects in Task 12.2 and their distribution among the 3rd parties is given in Figure 1.

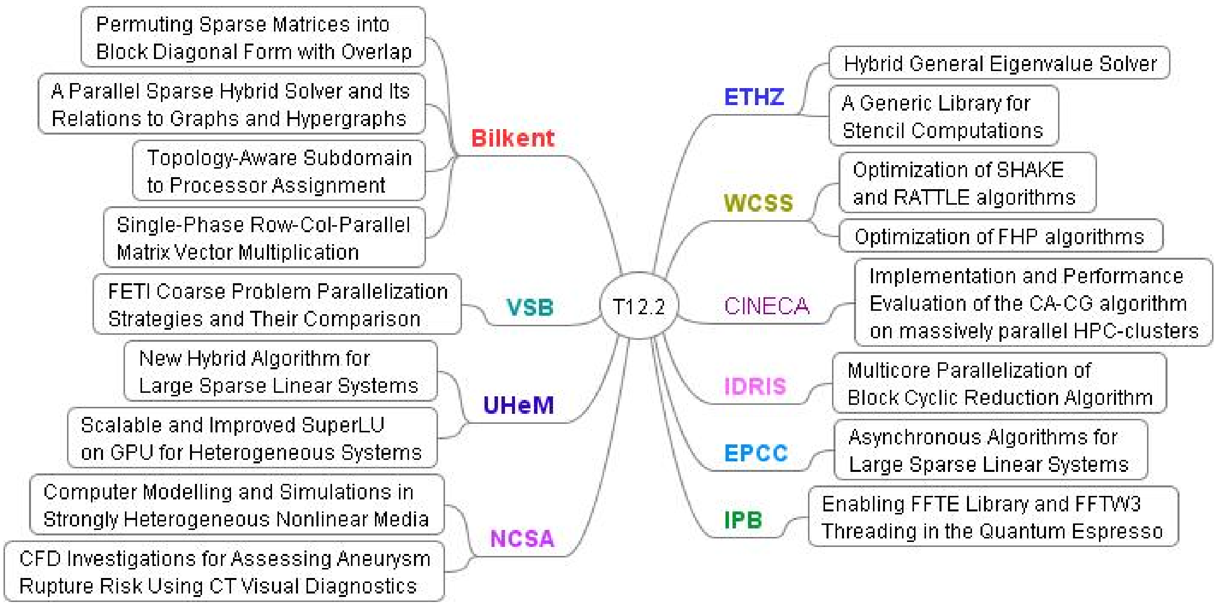
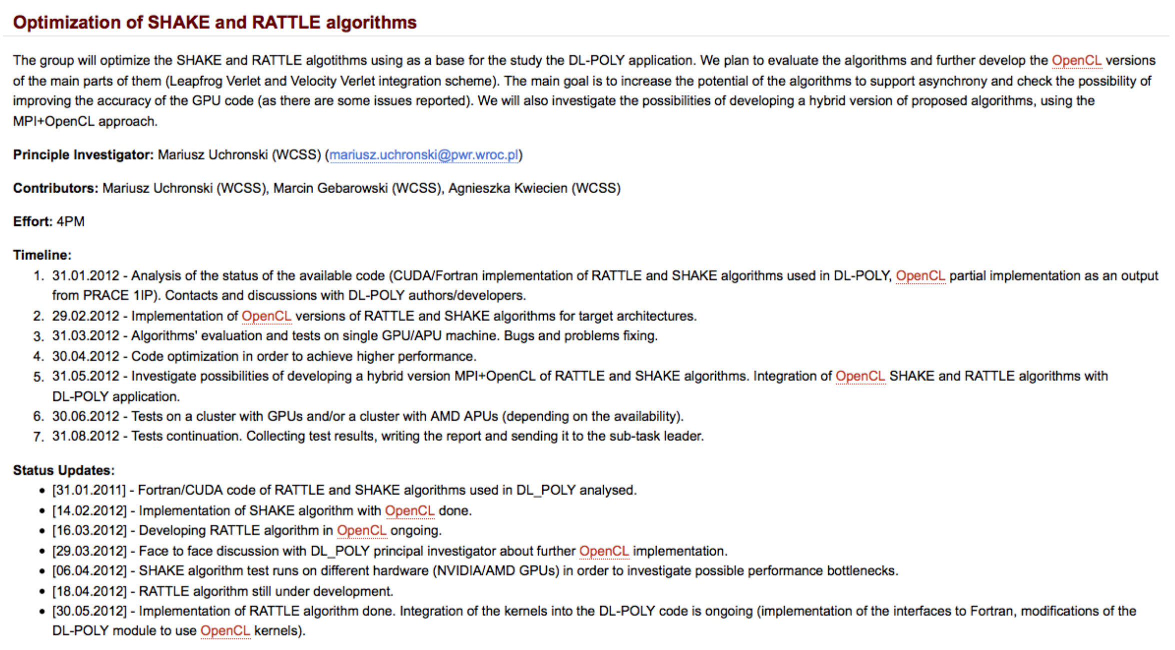

T12.2
Bilkent
Permuting Sparse Matrices into Block Diagonal Form with Overlap
A Parallel Sparse Hybrid Solver and Its Relations to Graphs and Hypergraphs
Topology-Aware Subdomain to Processor Assignment
Single-Phase Row-Col-Parallel Matrix Vector Multiplication
VSB
FETI Coarse Problem Parallelization Strategies and Their Comparison
UHeM
New Hybrid Algorithm for Large Sparse Linear Systems
Scalable and Improved SuperLU on GPU for Heterogeneous Systems
NCSA
Computer Modelling and Simulations in Strongly Heterogeneous Nonlinear Media
CFD Investigations for Assessing Aneurysm Rupture Risk Using CT Visual Diagnostics
ETHZ
Hybrid General Eigenvalue Solver
A Generic Library for Stencil Computations
WCSS
Optimization of SHAKE and RATTLE algorithms
Optimization of FHP algorithms
CINECA
Implementation and Performance Evaluation of the CA-CG algorithm on massively parallel HPC-clusters
IDRIS
Multicore Parallelization of Block Cyclic Reduction Algorithm
EPCC
Asynchronous Algorithms for Large Sparse Linear Systems
IPB
Enabling FFTE Library and FFTW3 Threading in the Quantum Espresso

Figure 1: Projects in Task 12.2

In order to structure work in Task 12.2, firstly, all projects defined the extent and purpose of their works in short statement of works (SOWs). Following these SOWs, to allow easy overview of all projects and to ensure that progress of all projects could be easily monitored, dedicated wiki pages have been set up and augmented with regular status reports. Figure 2 shows an example screen shot of one of the project wiki pages.



Optimization of SHAKE and RATTLE algorithms

The group will optimize the SHAKE and RATTLE algotithms using as a base for the study the DL-POLY application. We plan to evaluate the algorithms and further develop the OpenCL versions of the main parts of them (Leapfrog Verlet and Velocity Verlet integration scheme). The main goal is to increase the potential of the algorithms to support asynchrony and check the possibility of improving the accuracy of the GPU code (as there are some issues reported). We will also investigate the possibilities of developing a hybrid version of proposed algorithms, using the MPI+OpenCL approach.

Principle Investigator: Mariusz Uchronski (WCSS) (mariusz.uchronski@pwr.wroc.pl)

Contributors: Mariusz Uchronski (WCSS), Marcin Gebarowski (WCSS), Agnieszka Kwiecien (WCSS)

Effort: 4PM

Timeline:

1. 31.01.2012 - Analysis of the status of the available code (CUDA/Fortran implementation of RATTLE and SHAKE algorithms used in DL-POLY, OpenCL partial implementation as an output from PRACE 1IP). Contacts and discussions with DL-POLY authors/developers.
2. 29.02.2012 - Implementation of OpenCL versions of RATTLE and SHAKE algorithms for target architectures.
3. 31.03.2012 - Algorithms' evaluation and tests on single GPU/APU machine. Bugs and problems fixing.
4. 30.04.2012 - Code optimization in order to achieve higher performance.
5. 31.05.2012 - Investigate possibilities of developing a hybrid version MPI+OpenCL of RATTLE and SHAKE algorithms. Integration of OpenCL SHAKE and RATTLE algorithms with DL-POLY application.
6. 30.06.2012 - Tests on a cluster with GPUs and/or a cluster with AMD APUs (depending on the availability).
7. 31.08.2012 - Tests continuation. Collecting test results, writing the report and sending it to the sub-task leader.

Status Updates:

- [31.01.2011] - Fortran/CUDA code of RATTLE and SHAKE algorithms used in DL_POLY analysed.
- [14.02.2012] - Implementation of SHAKE algorithm with OpenCL done.
- [16.03.2012] - Developing RATTLE algorithm in OpenCL ongoing.
- [29.03.2012] - Face to face discussion with DL_POLY principal investigator about further OpenCL implementation.
- [06.04.2012] - SHAKE algorithm test runs on different hardware (NVIDIA/AMD GPUs) in order to investigate possible performance bottlenecks.
- [18.04.2012] - RATTLE algorithm still under development.
- [30.05.2012] - Implementation of RATTLE algorithm done. Integration of the kernels into the DL-POLY code is ongoing (implementation of the interfaces to Fortran, modifications of the DL-POLY module to use OpenCL kernels).

Figure 2: An example project wiki page.

As stated before, Task 12.2 covers topics such as topology-awareness, reducing synchrony, and new numerical algorithms. Figure 3 gives a schematic overview of the projects in Task 12.2; it also illustrates how the covered major topics are distributed across projects.

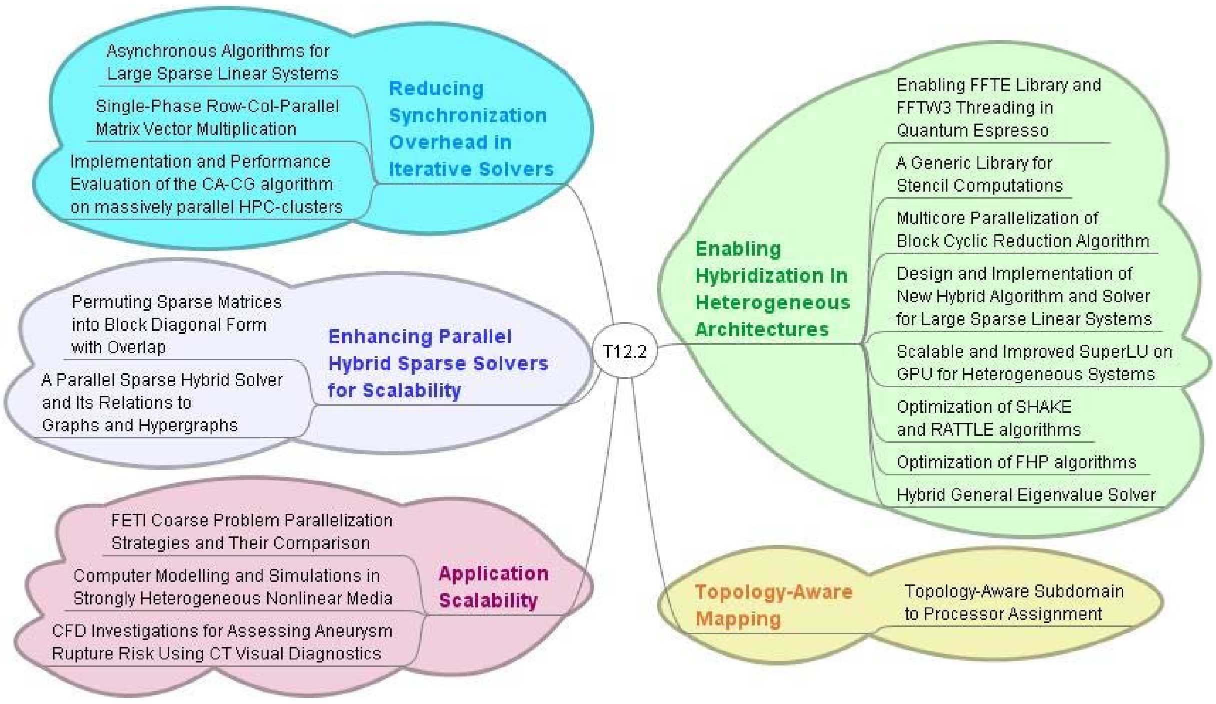

T12.2
Reducing Synchronization Overhead in Iterative Solvers
Asynchronous Algorithms for Large Sparse Linear Systems
Single-Phase Row-Col-Parallel Matrix Vector Multiplication
Implementation and Performance Evaluation of the CA-CG algorithm on massively parallel HPC-clusters
Enhancing Parallel Hybrid Sparse Solvers for Scalability
Permuting Sparse Matrices into Block Diagonal Form with Overlap
A Parallel Sparse Hybrid Solver and Its Relations to Graphs and Hypergraphs
Application Scalability
FETI Coarse Problem Parallelization Strategies and Their Comparison
Computer Modelling and Simulations in Strongly Heterogeneous Nonlinear Media
CFD Investigations for Assessing Aneurysm Rupture Risk Using CT Visual Diagnostics
Enabling Hybridization In Heterogeneous Architectures
Enabling FFTE Library and FFTW3 Threading in Quantum Espresso
A Generic Library for Stencil Computations
Multicore Parallelization of Block Cyclic Reduction Algorithm
Design and Implementation of New Hybrid Algorithm and Solver for Large Sparse Linear Systems
Scalable and Improved SuperLU on GPU for Heterogeneous Systems
Optimization of SHAKE and RATTLE algorithms
Optimization of FHP algorithms
Hybrid General Eigenvalue Solver
Topology-Aware Mapping
Topology-Aware Subdomain to Processor Assignment

Figure 3: Thematic overview of projects in Task 12.2.

3 Task 12.2 Key Research Topics and Project Overviews

In this section, we present an overview of the 17 projects in Task 12.2 categorized according to the above-mentioned key research topics. For each topic, we also provide a brief introduction stating the importance of each topic.

3.1 Reducing Synchronization Overhead in Iterative Solvers

Iterative algorithms have become the de-facto approach for the solution of sparse linear systems of equations on large-scale parallel systems due to their amenability to parallelization. Iterative algorithms have been successfully used for the solution of such linear systems on small-to-medium scale parallel systems. However, especially under peta-scale computing setups, bottlenecks in the parallelization of these algorithms have been observed. All iterative algorithms used for the solution of linear systems require a number of global synchronization operations (e.g., all_reduce) for computing global scalars as well as a number of local synchronization operations due to the point-to-point communications incurred by the sparse matrix-vector operations. These local and global synchronization operations create barriers beyond which computation cannot proceed until all participating processors have reached that point. Projects in Sections 4.1, 4.2, and 4.3 can be considered as research efforts towards reducing synchronization overheads.

Section 4.1, entitled “Asynchronous Algorithms for Large Sparse Linear Systems” exploits asynchronous techniques that avoid the blocking behavior of synchronization operations by permitting processors to operate on whatever data they have, even if new data has not yet arrived from other processors. This project provides an experimental evaluation of the asynchronous approach on the Jacobi method, which is one of the simplest iterative algorithms.

Section 4.2, entitled “Designing and Implementing a Single-Phase Row-Column-Parallel Sparse Matrix Vector Multiplication Algorithm based on 2D Matrix Partitioning” proposes to address the two-phase communication bottleneck of the row-column-parallel SpMxV operation so that 2D nonzero-based matrix partitioning models and methods can be successfully utilized. This project also proposes and implements a two-stage approach to produce a *good* partition for the proposed single-phase row-column-parallel SpMxV algorithm.

Section 4.3, entitled “Implementation and Performance Evaluation of the CA-CG Algorithm on Massively Parallel HPC-Clusters”, tries to deepen the understanding of an emerging class of methods (Communication Avoiding Krylov subspace methods) devised to overcome limitations due to synchronization issues in parallel sparse linear system solvers. The project evaluates the feasibility of implementation and testing the overall performance of the CA-CG algorithm on a set of benchmark platforms.

3.2 Enhancing Parallel Hybrid Sparse Solvers for Scalability

Recently a number of hybrid algorithms and solvers that contain both direct and iterative components are proposed. These algorithms are promising in terms of robustness and scalability on parallel computing platforms. These algorithms are also utilized for reducing the above-mentioned global synchronization overhead by efficient and effective preconditioning that decreases the number of iterations while increasing the amount of computation per iteration. Here, effectiveness refers to decrease in the number of iterations required to convergence, whereas efficiency refers to the computational overhead introduced by the preconditioning steps and the amenability of the preconditioning operations to parallelization. Projects in Sections 4.4 and 4.5 can be considered as research efforts towards reducing synchronization overheads through the use of hybrid linear system solvers.

Section 4.4, entitled "Permuting Sparse Matrices into Block Diagonal Form with Overlap", proposes and implements a matrix reordering scheme for permuting a square sparse matrix into block-diagonal form with overlap for efficient parallelization of the multiplicative Schwarz and the DDKrylov preconditioners. The permutation objective of minimizing the overlap size also has a positive impact on the effectiveness of the Shwarz preconditioner via minimizing an upper bound on the iteration count.

Section 4.5, entitled "A Parallel Sparse Hybrid Solver and Its Relations to Graphs and Hypergraphs", proposes and implements two different (graph- and hypergraph-based) matrix reordering schemes that enables the extraction of a small reduced system from a square sparse matrix for a successful *DS* factorization preconditioning scheme.

3.3 Topology-Aware Mapping

The task-to-processor mapping in parallel systems can be vital for the performance of the executed programs. Especially for architectures that consist of thousands of processors (such as IBM BlueGene/P), the poor mapping of tasks to processors can lead to message contention since the network resources are usually shared by multiple communication tasks. This causes increase in message latencies and degrades the overall system performance. In such architectures, the message latencies are not independent of the number of links between two processors, which is counter to the assumption made by the cut-through and wormhole routing. Hence, the number of links (or hops) between processors becomes important when mapping tasks to processors. Taking this observation into account, careful mapping of tasks to processors which exploits the topology of the parallel architecture and the interaction between tasks can be very beneficial for the overall performance of the parallel system.

Section 4.6, entitled "Topology-aware subdomain-to-processor assignment" proposes and implements a two-phase framework for topology-aware task-to-processor mapping by considering both the task interaction and processor organization graphs. The first phase groups highly interacting tasks into *K* clusters, where *K* is equal to the number of processors, and the second phase performs one-to-one task-cluster-to-processor mapping with the objective of minimizing a metric defined on the volumes and distances of communication operations.

3.4 Enabling Hybridization in Heterogeneous Architectures

The hybrid approach, combining MPI programming paradigm across computing nodes and OpenMP-, threading-, CUDA- and OpenCL-based approach within individual nodes, have been widely adopted by the computational science community for developing programs that execute on heterogeneous platforms (CPU multi-core processors combined with multi-core and/or accelerator technology). Sections 4.7, 4.8, 4.9, 4.10, 4.11, 4.12, 4.13 and 4.14 are all efforts towards exploiting the benefits of hybridization in large-scale computing performance.

Section 4.7, entitled "Multicore Parallelization of Block Cyclic Reduction Algorithm" investigates efficient parallelization of the Buneman's variant of block cyclic reduction algorithm for the solution of linear systems with coefficient matrices of block-tridiagonal form. Performance comparisons are given for a small-scale system using OpenMP and MPI.

Section 4.8, entitled "Enabling FFTE Library and FFTW3 Threading in the Quantum Espresso" investigates the effects of replacing the FFT routines in Quantum Espresso with better performing FFTE and FFTW3 libraries that support hybrid OpenMP + MPI approach.

Section 4.9, entitled “A Hybrid Hermitian General Eigenvalue Solver” proposes and develops general dense eigenvalue solvers based on single-node CPUs + GPU and CPUs + GPUs hybridization. The project provides an experimental comparison of the developed hybrid eigenvalue solvers against both a shared memory and a distributed memory library.

Section 4.10, entitled “A Generic Library for Stencil Computations” investigates the programmability issues, such as productivity and portable efficiency in parallel algorithm design and software development for HPC clusters and machines with accelerators. To this end, the project develops a domain specific C++ generic library for stencil computations, like PDE solvers. The library features high level constructs such as *do_all* and *do_reduce* and allows the development of parallel stencil computations with very limited effort.

Section 4.11, entitled “Design and Implementation of New Hybrid Algorithm and Solver for Large Sparse Linear Systems” describes the efforts towards improving the scalability of the direct linear system solver SuperLU via utilizing the MPI+OpenMP hybrid programming approach that combines the advantages of the two SuperLU libraries that are available for distributed and shared memory architectures.

Section 4.12, entitled “Scalable and Improved SuperLU on GPU for Heterogeneous Systems” describes the efforts towards improving the scalability of the direct linear system solver SuperLU via utilizing the MPI+CUDA hybrid programming approach.

Section 4.13, entitled “Optimization of Shake and Rattle Algorithms”, investigates optimization of SHAKE and RATTLE algorithms, which are widely used in molecular dynamics simulations, through embedding CUDA and OpenCL implementations of these algorithms into the DL_POLY molecular simulation package.

Section 4.14, entitled “Optimization of FHP Algorithms” investigates the possibility of accelerating FHP algorithms, which are used to solve Navier-Stokes equations derived from Newtonian Mechanics to describe the motion of fluid substances, using POSIX Threads, MPI, SSE, AVX and NVIDIA CUDA.

3.5 Application Scalability

In the parallelization of applications, both due to the nature of the underlying application together with the adopted parallelization scheme certain parallelization overheads such as sequential components, synchronization overheads, and load imbalance may be tolerable up to a certain number of processors. However, when the level of parallelization reaches towards petascale, each of these parallelization overheads should be investigated and minimized in order to achieve a decent speedup on such large-scale systems. Sections 4.15, 4.16, and 0 are all efforts towards exploiting various algorithms and methods for increasing the scalability of the parallelization of their target applications.

Section 4.15, entitled “FETI Coarse Problem Parallelization Strategies and Their Comparison” investigates the parallelization of the Finite Element Tearing and Interconnecting (FETI) methods. For small-to-medium scale parallelization, the coarse-grain problem, which is obtained from subdomain interfaces, is solved sequentially on the master core. However, this is infeasible for petascale computing settings, because of the increase in memory requirements and decrease in scalability, which is due to the substantial increase in number of subdomains. The project investigates a number of methods for improving the scalability via parallel solution of the coarse system at different parallelization and redundant computation levels.

Section 4.16, entitled “Computer Modeling and Simulations in Strongly Heterogeneous Nonlinear Media” investigates incorporating selection of efficient parallel preconditioners, parallel implementation of aggressive coarsening algorithms and adaptive time-stepping in the parallelization of the Finite Element Method (FEM) simulation of thermal and electrical fields in strongly heterogeneous nonlinear media on structured and unstructured meshes.

Section 0, entitled "CFD-Investigations for Assessing Aneurysm Rupture Risk for Individual Patient Using CT Visual Diagnostics" perform numerical experiments using the preconditioned *BiCGStab* (Bi-conjugate gradient stabilized) algorithm with incomplete factorization for the parallel solution of 3D Navier-Stokes equations for incompressible fluids, which is used in the modeling of the blood flow in cerebral aneurisms.

4 Detailed Project Descriptions

In this chapter, detailed descriptions of the Task 12.2 projects along with obtained results and discussions are provided. All projects contain links to accompanying white papers or publications. All PRACE technical white papers are available on the PRACE-RI web site [https://bscw.zam.kfa-juelich.de/bscw/bscw.cgi/787675].

4.1 Asynchronous Algorithms for Large Sparse Linear Systems

Supported by: Mark Bull and Iain Bethune (EPCC, University of Edinburgh)

Whitepaper: Mark Bull and Iain Bethune, "Asynchronous Algorithms for Large Sparse Linear Systems ", http://eprints.ma.man.ac.uk/1838/.

Modern high-performance computing systems are typically composed of many thousands of cores linked together by high bandwidth and low latency interconnects. Over the coming decade core counts will continue to grow as efforts are made to reach Exaflop performance. In order to continue to exploit these resources efficiently, new software algorithms and implementations will be required that avoid tightly-coupled synchronization between participating cores and that are resilient in the event of failure.

This project investigates one such class of algorithms. The solution of systems of linear equations of the form A**x**=**b**, where A is a large, sparse n by n matrix and **x** and **b** are column vectors of size n, lies at the heart of a large number of scientific computing kernels, and so efficient solution implementations are crucial. Existing iterative techniques for solving such systems in parallel are typically synchronous, in that all processors must exchange updated vector information at the end of every iteration, and scalar reductions may be required by the algorithm. This creates barriers beyond which computation cannot proceed until all participating processors have reached that point, i.e. the computation is globally synchronized at each iteration. Such approaches are unlikely to scale to millions of cores.

This project is focused on developing asynchronous techniques that avoid this blocking behavior by permitting processors to operate on whatever data they have, even if new data has not yet arrived from other processors. To date there has been work on both the theoretical and the practical aspects of such algorithms. To reason about these algorithms one needs to understand what drives the speed of their convergence, but existing results merely provide sufficient conditions for the algorithms to converge, and do not help in answering some of the questions arising in the use of asynchronous techniques in large, tightly coupled parallel systems of relevance to exascale computing. This project tries to obtain insights by investigating the performance of the algorithms experimentally.

Taking Jacobi's method, one of the simplest iterative algorithms, one traditional synchronous and two asynchronous variants are implemented, using three parallel programming models - MPI, SHMEM and OpenMP. The performance of these implementations is investigated in detail at scale on a Cray XE6, and some counter-intuitive properties which are of great interest when implementing such methods are discussed.

The performance of these algorithms depends on two key factors - the efficiency and scalability of the implementation, and the effect of asynchrony on the number of iterations taken to converge - both of which vary with the number of cores used. Table 2 and Table 3 show the number of iterations and execution time required to converge a heat diffusion problem on HECToR, a Cray XE6 system with 32 AMD Interlagos cores per node. *sync* is the synchronous version, *async* is a race-free asynchronous version and *racy* is an easier-to-program asynchronous version that includes deliberate race conditions: all three version are implemented in both MPI and SHMEM.

Results presented in Table 2 and Table 3 show that (except on the very largest core counts) SHMEM can provide a more efficient implementation of asynchronous message-passing than MPI, and that for problems that require on the order of thousands of cores, asynchronous algorithms can outperform their synchronous counterparts by around 10%. OpenMP (results not shown here) was found to give good performance for asynchronous algorithms, and was also very easy to program compared to either MPI or SHMEM. Although it has limited scalability due to the number of cores in a shared memory node, we suggest that OpenMP might be applicable in a hybrid model with MPI, for example, particularly since we found asynchronous Jacobi in OpenMP to be 33% faster than the synchronous equivalent even on a relatively modest 32 cores. In addition, asynchronous algorithms are expected to be tolerant to hardware performance defects, which could be an advantage on systems with millions of cores.

Processes	Iterations			Execution
Version	Min.	Mean	Max.	Time (s)
32				
sync		9100		28.4
async	10737	11245	12231	33.8
racy	8952	9307	10140	27.7
512				
sync		37500		132.1
async	36861	44776	51198	146.9
racy	32377	38763	44623	126.4
8912				
sync		68500		247.1
async	60612	82381	96700	272.5
racy	61453	77952	90099	264.9
32768				
sync		112000		405.2
async	101053	136337	165366	454.4
racy	86814	118967	144902	419.4

Table 2: HECToR MPI results summary

Processes	Iterations			Execution
Version	Min.	Mean	Max.	Time (s)
32				
sync		9100		27.6
async	8724	9136	10784	26.7
racy	8496	9088	10725	26.3
512				
sync		37500		130.1
async	32356	38285	43178	117.9
racy	32377	38416	46291	117.9
8912				
sync		68500		251.5
async	32612	67417	77382	215.5
racy	47231	70087	80620	219.3
32768				
sync		112000		491.1
async	11265	229266	331664	858.5
racy	10173	231919	314208	817.6

Table 3: HECToR SHMEM result summary

4.2 Designing and Implementing a Single-Phase Row-Column-Parallel Sparse Matrix Vector Multiplication Algorithm based on 2D Matrix Partitioning

Supported by: B. Ucar (CNRS), E. Kayaaslan, O. Ozturk, C. Aykanat (Bilkent University)

Whitepaper: E. Kayaaslan, B. Ucar, O. Ozturk, C. Aykanat, "Designing and Implementing a Single-Phase Row-Column-Parallel Sparse Matrix Vector Multiplication Algorithm based on 2D Matrix Partitioning", PRACE technical white paper.

Sparse matrix vector multiplication (SpMxV) is a kernel operation repeatedly performed in iterative linear system solvers. There are mainly three types of parallel SpMxV algorithms used in the scientific community: row-parallel, column-parallel and row-column-parallel. The row-parallel algorithm involves expand-type point-to-point communication operations on the local input vector entries before the local SpMxV operations; whereas column-parallel algorithm involves fold-type point-to-point communication operations on the local output vector results after the local SpMxV operations. The row-column-parallel algorithm necessitates two-phase communication: expand operation before local SpMxVs and fold operation after the local SpMxVs. 1D rowwise and columnwise partitioning of the coefficient matrix are used for row-parallel and column-parallel SpMxV algorithms, respectively, whereas 2D-nonzero partitioning of the coefficiant matrix is used for row-column-parallel SpMxV algorithms. Several hypergraph partitioning models and methods have been successfully used for sparse matrix partitioning for efficient row-parallel, column-parallel and row-column-parallel SpMxV operations. In all these models the partitioning objective is to minimize the total volume of communication whereas the partitioning constraint is to minimize the computational load balance. 2D nonzero based partitioning models are both more scalable and perform considerably better than the 1D partitioning models in terms of communication volume metric. However, 1D models perform considerably better than 2D models in terms of speedup values due to the increased number of messages in the row-column-parallel SpMxV algorithm. In this project, a one-phase row-column-parallel SpMxV algorithm is proposed to address this bottleneck of the row-column-parallel SpMxV operation so that 2D nonzero-based matrix partitioning models and methods can be successfully utilized.

In this project, a two-stage approach is proposed and adopted to produce a *good* partition for the above-mentioned one-phase row-column-parallel SpMxV algorithm. In the first stage, a *K*-way row/column partition and initial nonzero partition, either using one-dimensional (coarse-grain) [1] or two-dimensional (fine-grain) [3] partitioning approaches, is obtained for a *K* processor system. At the end of this stage, for each ordered pair of parts a submatrix whose nonzeros will be determined to be hold by either receiver or sender processor is obtained. The second stage refines the nonzero partition obtained in the first stage by using Dulmage-Mendhelson decomposition of these submatrices. In case of one-dimensional partitioning in the first stage, the off-diagonal submatrices are used, whereas in case of two-dimensional partitioning, in order to keep load balance, for each ordered pair of parts, the subset of the off-diagonal submatrix that is comprised of only nonzeros assigned to neither receiver nor sender processor is used. In the refinement step, the nonzeros that lie inside the horizontal blocks are assigned to sender, and the remaining nonzeros of the submatrices are assigned to receiver processor.

Parallel algorithm	row-parallel	column-parallel	row-column-parallel		
# of communication phases	1 phase	1 phase	2 phase	1 phase	
Part. scheme in first stage	1D	1D	2D	1D	2D
bundle1	8.326	8.003	5.739	5.918	**8.342**
cbuckle	9.545	**10.622**	8.612	9.819	10.605
finan512	13.575	13.504	11.640	13.677	**13.682**
poisson3Da	6.892	6.907	4.154	**7.179**	6.718
rgg_n_2_17_s0	12.701	**13.240**	11.137	13.180	12.868
shuttle_eddy	6.079	5.991	3.743	6.177	**6.180**
tube1	**11.263**	11.233	9.505	11.149	8.668
vibrobox	5.181	6.017	3.415	**6.123**	4.933
ASIC_320ks	11.504	11.578	10.422	12.104	**12.293**
msc10848	**11.789**	11.786	9.862	11.539	11.466

Table 4: Speedup values obtained on a 16-processor system.

The validity of the proposed one-phase row-column-parallel SpMxV and the associated two-stage partitioning schemes is experimentally evaluated in terms of speedup values obtained on a 16-node PC-cluster located at Bilkent University. The test matrices are obtained from the

Florida Matrix Collection [2]. Table 4 presents these speedup values. In the table, a bold value in a row indicates the best speedup value obtained for the parallelization of the SpMxV associated with the respective test matrix. As seen in the table, these initial results are promising and extensive experimentation on a 512-node system is underway.

4.3 Implementation and Performance Evaluation of the CA-CG Algorithm on Massively Parallel HPC-Clusters

Supported by: G. Erbacci, M. Culpo, M. Guarrasi (CINECA)

Sparse linear systems lie at the core of many scientific computing applications. They can be solved by direct or iterative methods; the former preferred for their robustness when the problem size is small enough to make them affordable, the latter preferred due to their better asymptotic complexity and the high degree of parallelism they generally expose [4] when the systems to be solved are large enough.

In particular, fostered by the need to solve engineering problems of ever-growing complexity, modern iterative methods based on Krylov subspaces received a lot of attention in the last two decades [5]. This led to remarkable improvements in the underlying mathematical algorithms and to careful studies on the optimal way to implement them [6], producing as a result a number of widely known software packages that are proven to scale up to roughly a few thousands cores [7], [8].

While this state-of-the-art fits nicely into the framework of Tier-1 supercomputers, it becomes soon inadequate if Tier-0 resources are instead to be efficiently exploited. The main cause of this inadequacy is to be found in the current trends driving the HPC world towards architectures composed of millions of relatively slow cores that can reach, nowadays, performances up to almost 20 PetaFlop/s [9]. A rough comparison with the architecture of previous generation machines makes evident that algorithms need to increase their scalability by almost two orders of magnitude to effectively exploit the latest technologies. A request of such a kind cannot be satisfied by the standard Krylov subspace algorithms, as the amount and type of communications they require makes them too much synchronous for this task.

This project tries therefore to deepen the understanding of an emerging class of methods (Communication Avoiding Krylov subspace methods) [10], [11] precisely devised to overcome these limitations. This is done by evaluating the feasibility of implementation and testing the overall performance of the CA-CG algorithm on a set of benchmark platforms.

CA methods are based on the assumption that the cost of an algorithm includes both computations and communications[1]: rather than minimizing the computational cost only, they try therefore to minimize the overall cost. As for technological reasons communication costs are much higher than computational costs, this directly leads to the key idea that the best performance is obtained trying to avoid communication as much as possible, even if this may require some redundant arithmetic operations.

This strategy is realized in the CA-CG algorithm through the use of the Matrix Power Kernel and Block Inner Product algorithms. Though for a mathematically sound treatment the interested reader is referred to [10], it is anticipated that the CA-CG algorithms will require in the end a factor of $\Theta(s)$ fewer messages if compared to standard CG, where s is the number of basis vectors generated by the matrix power kernels.

The framework used for the parallel implementation of the CA-CG algorithm is the one provided by PETSc [7], as this choice allows for a fair comparison with the standard CG

[1] The term "communications" includes in this framework both the bandwidth terms (communication among different memory levels) and the latency terms (communication among different nodes)

algorithm available in the library. The two algorithms have been tested on a selected set of Sparse Positive Definite (SPD) matrices taken from the UFL database [2]. Finally, the tests have been conducted on both PLX [12] and FERMI [13] supercomputers.

While for the detailed results of the benchmarks we refer to the associated white paperwe can say that in many cases non-negligible increases of performances were found. This strengthened our conviction on the fact the CA algorithms may be considered the most promising line of research among the ones trying to improve parallelism and performance of Krylov subspace linear solvers.

In the near future CINECA plans to continue the testing activity on CA algorithms even outside of the PRACE framework, probing in particular the effectiveness of the currently available preconditioners for CA-CG and the efficiency of others CA algorithms (e.g. CA-GMRES) on a wider set of benchmarks. The long term aim in case of sufficiently good results is to incorporate these algorithms in production codes that are commonly used on FERMI.

4.4 Permuting Sparse Square Matrices into Block Diagonal Form with Overlap for Parallel Hybrid Sparse Solvers

Supported by: S. Acer, E. Kayaaslan, C. Aykanat, T. Dayar (Bilkent University)

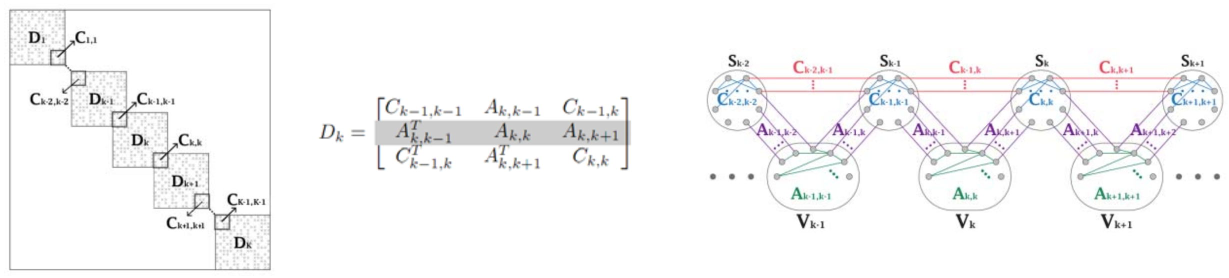
Whitepaper: S. Acer, E. Kayaaslan, T. Dayar, C. Aykanat, "Permuting Sparse Matrices into Block Diagonal Form with Overlap for Parallel Hybrid Sparse Solvers", PRACE technical white paper.

In this project, the problem of symmetrically permuting a sparse square matrix into block diagonal form with overlap (BDO form) is defined and a graph partitioning algorithm for solving this problem is proposed. Results show the validity of the proposed algorithm.

A K-way BDO form contains K diagonal blocks such that each two consecutive diagonal block may overlap (see Figure 4(a)). The problem of permuting a matrix into BDO form has the objective of minimizing total overlap size, i.e., sum of the number of rows/columns in overlaps, while having the constraint of maintaining balance on the number of nonzeros of the diagonal blocks.

This permutation problem arises in the parallelization of an explicit formulation of the multiplicative Schwarz preconditioner [14] and the DDKrylov scheme [15]. In these parallelizations, each diagonal block and its associated computations are assigned to a distinct processor. The permutation objective of minimizing total overlap size corresponds to minimizing communication volume [14], [15] and minimizing the size of the balance system [15], and the permutation constraint of maintaining balance on the number of nonzeros of diagonal blocks relates to maintaining balance on the computational loads of processors during the iterations [14]. In addition to these relations, minimizing total overlap size corresponds to minimizing an upper bound on the number of iterations for convergence [16].

As the ring/chain topology can be easily embedded in almost all of the interconnection topologies utilized in large scale systems, the BDO form ensures nearest neighbour (one-hop) communication in which each processor communicates only with its left and right neighbor. The objective of minimizing the number of off-diagonal-block entries may incur network congestion in the point-to-point communications despite worm-hole routing. The adverse effect of multihop distant messages on the performance of the large scale parallel system has been recently reported in the literature and the need for topology-aware mapping is addressed. The nearest neighbor communication pattern achieved due to the BDO form leads to the contention-free communication on the fly.



(a) BDO form of matrix A (b) Decomposition of a diagonal block (c) oVS form

Figure 4: BDO form of a matrix and its oVS representation.

A graph partitioning problem, namely K-way ordered Graph Partitioning by Vertex Separators problem (oGPVS), which is equivalent to the above permutation problem is defined as follows: For the standard graph representation of a given matrix, find a partition of its vertices into K ordered vertex parts and K-1 ordered vertex separators such that each two consecutive vertex parts can only be connected through a distinct vertex separator and each vertex separator can only connect those consecutive vertex parts and the vertex separators neighbouring those vertex parts. This form of a given graph is referred as the Ordered Vertex Seperator (oVS) form (see Figure 4(c)). The partitioning objective is to minimize the sum of the number of vertices in the vertex separators, whereas the partitioning objective is to maintain balance on the vertex part weights, where the weight of each vertex is assigned as the number of nonzeros in its corresponding row/column of the given matrix.

In this project, a new oGPVS algorithm that utilizes the existing graph partitioning tools is proposed to solve this partitioning problem, hence the permutation problem. This algorithm recursively bisects the graph with an existing graph partitioning tool such that the resulting two vertex parts are connected through a vertex separator at each recursion, and vertex parts and the vertex separators are ordered in the final partition.

This permutation problem is addressed only in a recent work by Kahou et al [17], and a bottom-up graph-partitioning algorithm is proposed. This algorithm and the proposed algorithm are tested over 237 matrices collected from UFL matrix database for K = 4, 8, 16, 32, 64, 128, and 256-way permutations. Since the objective is to minimize the total overlap size in the permuted matrix while maintaining balance on the number of non-zeros of diagonal blocks, the results of both algorithms are compared in terms of the total overlap size and the imbalance ratio on the number of nonzeros and these results are presented in Table 5. On average, 23-35% improvement is achieved in terms of total overlap size for different K values as seen in the last column of Table 5.

K	Number of Matrices	Baseline Algorithm		oGPVS Algorithm		oGPVS/Baseline (Overlap)
		Imbalance	Overlap	Imbalance	Overlap	
4	205	3.62%	2.65%	2.36%	4.08%	0.65
8	183	6.55%	5.13%	4.60%	7.42%	0.70
16	155	9.42%	9.07%	6.62%	9.57%	0.70
32	106	9.91%	9.45%	7.63%	9.38%	0.77
64	63	9.63%	10.26%	7.39%	9.34%	0.77
128	38	10.89%	8.27%	22.22%	6.11%	0.74
256	24	12.64%	9.13%	17.86%	5.46%	0.60

Table 5: Performance comparison of the baseline and oGPVS algorithms in terms of total overlap size and load imbalance for different number of processors (K).

4.5 A Parallel Sparse Hybrid Solver and Its Relation to Graphs and Hypergraphs

Supported by: M. Gundogan (Bilkent University), M. Manguoglu (METU), C. Aykanat (Bilkent University)

Whitepaper: M. Gundogan, M. Manguoglu, C. Aykanat, “A Parallel Sparse Hybrid Solver and Its Relation to Graphs and Hypergraphs”, PRACE technical white paper.

Given a system of equations of the form $\boldsymbol{Ax} = \boldsymbol{f}$, where $\boldsymbol{A}$ is large and sparse, it is known that hybrid solvers that contain both direct and iterative components are promising in terms of robustness and scalability on parallel computing platforms. A state-of-the-art hybrid solver, which uses the generalized form parallel $\boldsymbol{DS}$ factorization, is the focus of this work [18]. In the $\boldsymbol{DS}$ factorization scheme, $\boldsymbol{D}$ is the block diagonal of $\boldsymbol{A}$, and the factor $\boldsymbol{S}$, given by $\boldsymbol{D}^{-1}\boldsymbol{A}$ (assuming $\boldsymbol{D}$ is nonsingular), consists of the block diagonal identity matrix modified by “spikes” to the right and left of each partition. The generalized $\boldsymbol{DS}$ factorization of the system involves reordering $\boldsymbol{A}$ to extract the block diagonal $\boldsymbol{D}$, then, multiplying both sides of the system with $\boldsymbol{D}^{-1}$, from the left side. The resulting multiplied system contains a smaller reduced system of equations based on the nonzero entries in off-diagonal blocks of the reordered system. The solution of the original system, $\boldsymbol{Ax} = \boldsymbol{f}$, can be obtained from the solution of this reduced system, which can be solved independently. After $\boldsymbol{DS}$ factorization, the process of solving $\boldsymbol{Ax} = \boldsymbol{f}$ reduces to a sequence of steps that are ideally suited for parallel execution and performance. In general, the scalability of this factorization scheme depends on decreasing the solution time of the reduced system. Among other factors, the solution time of the reduced system depends on the size and the number of nonzeros of the reduced system.

In this project two different reordering strategies are investigated for a successful DS factorization preconditioning scheme: Reordering via graph partitioning (GP) and reordering via hypergraph partitioning (HP). In the GP scheme, the standard graph representation $G(\boldsymbol{A})$ on matrix $\boldsymbol{A}$ is used. In the HP scheme, the column-net hypergraph model [1] $H_{cn}(\boldsymbol{A})$ of matrix $\boldsymbol{A}$ is used.

For a K processor system, in the GP and HP schemes, a K-way partitioning is performed on $G(\boldsymbol{A})$ and $H(\boldsymbol{A})$, respectively, and the resulting partition is decoded as inducing a K-way symmetric permutation on the rows and columns of [1]. In both schemes, the partitioning constraint of maintaining balance on the part weights corresponds to maintaining balance on the nonzero counts of the diagonal blocks of the reordered matrix. In the GP scheme, the partitioning objective of minimizing the edge cutsize corresponds to minimizing the total number of nonzeros in the off-diagonal blocks of the reordered matrix. In the HP scheme, the partitioning objective of minimizing the cutsize according to the cut-net metric corresponds to minimizing the total number of nonzero columns in the off-diagonal blocks of the reordered matrix. The partitioning objective of the GP scheme relates to minimizing the number of nonzeros in the reduced system, whereas the partitioning objective of the HP scheme exactly models minimizing the size of the reduced system. Hence, the HP scheme can be expected to achieve better preconditioning compared to the GP scheme.

In this project, the experimental performance comparison of the proposed GP and HP schemes for preconditioning with DS factorization are investigated by using the successful multi-level graph and hypergraph partitioning tools MeTiS and PaToH [1] on 71 matrices selected from UFL sparse matrix collection [19]. The biconjugate gradient stabilized (BiCGStab) solver is used as an iterative solver for both inner and outer systems, whereas PARDISO is used as a direct solver for the diagonal blocks. The target parallel architecture is an Intel cluster of 46 nodes located at Middle East Technical University, where each node contains 2 Intel Xeon E5430 Quad-Core CPUs. Table 6 displays the performance improvement of the HP and GP schemes over the unordered scheme in terms of solution times on a 64-processor system averaged over different problem categories.

Problem Category	# of matrices	GP	HP	HP/GP
Chemistry	14	20.8%	26.7%	0.93
Circuit Simulation	9	22.3%	30.6%	0.89
Computational Fluid Dynamics	9	15.6%	27.8%	0.86
Modeling	6	26.1%	17.8%	1.11
Structural	7	18.9%	20.4%	0.98
Other	26	21.5%	32.9%	0.85

Table 6: Performance improvement of HP over GP in terms of solution times on a 64-processor system averaged over different problem categories.

As seen in Table 6, HP and GP schemes achieve 10-30% improvement in the solution times for different problem categories on average. The last column of Table 6 displays the ratio of the solution times of HP and GP schemes averaged over problem categories. Values smaller than one indicate the categories where HP scheme performs better than the GP scheme on average. As seen in the last column of Table 6, the HP scheme performs considerably better than the GP scheme in all problem categories except the "Modeling" category.

Figure 5 displays speedup curves for the solution of four different linear systems selected from the list of matrices used in the experiments. As seen in the figure, HP achieves considerably better speedup than GP and with increasing number of processors the performance gap slightly increases in favor of HP.

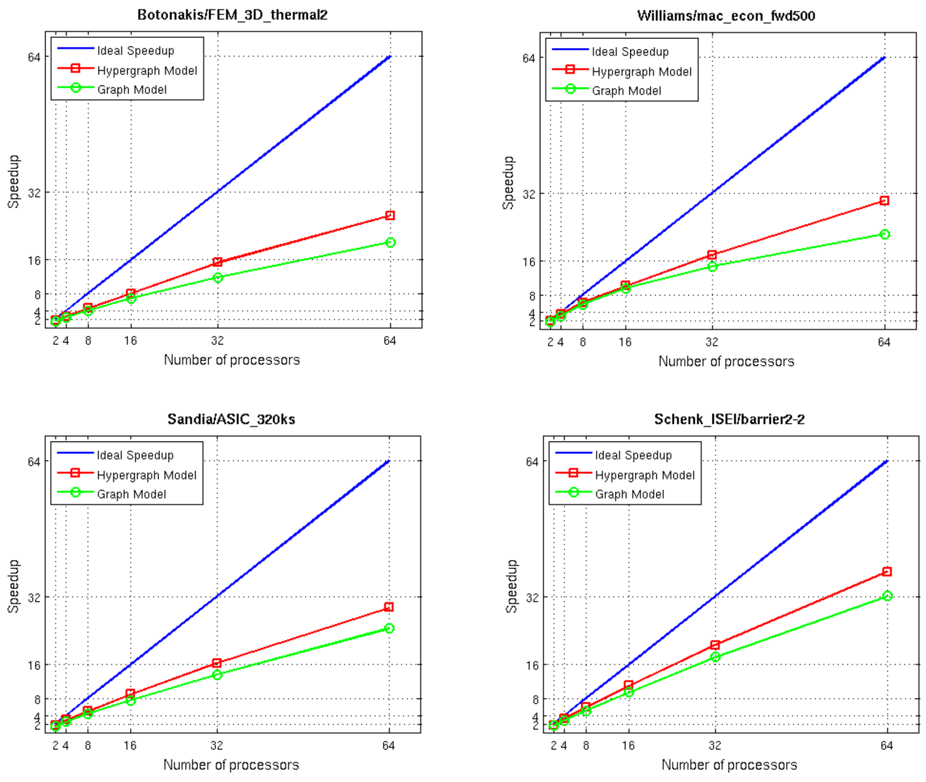


Figure 5: Speedup curves for the solution of four different linear systems on a 64-processor system.

4.6 Topology-Aware Subdomain-to-Processor Assignment

Supported by: R. O. Selvitopi, A. Turk, A. Guvenir, C. Aykanat, (Bilkent University)

Whitepaper: R. O. Selvitopi, A. Turk, A. Guvenir, C. Aykanat, “Topology-Aware Subdomain-to-Processor Assignment”, PRACE technical white paper.

The task/process to processor mapping in parallel systems can be vital for the performance of the executed programs. Especially for architectures that consist of thousands of processors (such as IBM BlueGene/P), the poor mapping of tasks to processors can lead to contention since the network resources are usually shared by multiple tasks. This causes increase in message latencies and degrades the overall system performance. In such architectures, the message latencies are not independent of the number of links between two processors, which is counter to the assumption made by the cut-through and wormhole routing. Hence, the number of links (or hops) between processors becomes important while mapping tasks to processors. Taking this observation into account, careful mapping of tasks to processors which exploits the topology of the parallel architecture and the interaction between tasks can be very beneficial for the overall performance of the parallel system.

In this project, an iterative-improvement-based algorithm is proposed and implemented for mapping tasks to processors. The algorithm utilizes the topology of the parallel architecture and the interaction between tasks. The algorithm starts from a random initial mapping and improves this mapping by a number swaps between tasks. Given n processors with topology information and n tasks with interaction information between the tasks, the proposed algorithm finds a one-to-one mapping so as to minimize the communicated volume between tasks using hop-bytes metric [20]. The hop-bytes for a total of N messages is computed as $\sum_{i=1}^{N} d_i \times b_i$, where b_i is the size of message in bytes and d_i is the number of links message uses.

Using the interaction graphs for standard parallel SpMxV application, the proposed algorithm is tested for various test matrices collected from the UFL matrix database. The parallelization of the application is done via utilizing the row-net and column-net hypergraph partitioning models [1]. The number of tasks/processors is varied from 64 to 2048. Table 7 presents the maximum task interaction degrees for the tested matrices under both row-net (RN) and column-net (CN) models.

	Matrices													
	2cubessphere		fullb		G3_circuit		language		stokes128		ted_A		tmt_sym	
ntasks nprocs	RN	CN	RN	CN	RN	CN	RN	CN	RN	CN	RN	CN	RN	CN
64	12,3	12,3	7,3	7,2	5,1	5,3	63,0	62,8	5,0	5,1	4,6	4,4	5,3	5,2
128	13,5	13,3	7,9	7,7	5,3	5,2	123,4	122,2	5,6	5,5	6,2	6,0	5,5	5,4
256	13,8	13,8	8,3	8,1	5,7	5,6	201,7	197,7	5,7	5,7	8,6	9,7	5,7	5,7
512	14,2	14,0	9,3	9,3	5,9	6,0	224,7	221,6	6,0	6,0	12,1	13,5	5,8	5,7
1024	14,4	14,6	9,7	9,6	6,6	6,6	176,6	184,2	6,2	6,3	15,9	18,9	5,8	5,9
2048	14,5	14,6	9,7	9,6	6,6	7,7	115,3	124,3	6,6	6,6	21,0	28,6	5,9	5,9

Table 7: Average task degrees for the task interaction graphs (RN = row-net, CN = column-net)

Table 8 presents the maximum hops of the tested topologies. These topologies are extracted from JUGENE Blue Gene/P system using the TopoManager API [21]. The maximum hops is defined as the maximum shortest path between two processors.

Number of processors in the topology	Maximum hops
64	9
128	13
256	17
512	12
1024	16
2048	24

Table 8: Maximum hops for each topology

As mentioned, our algorithm improves an initial random assignment. The improvement percentage values compared to a random mapping of tasks to processors are given in Table 9. As seen in the table, improvements ranging from 14% to 63% are observed in terms of the hop bytes metric.

	Matrices													
	2cubessphere		**fullb**		**G3_circuit**		**language**		**stokes128**		**ted_A**		**tmt_sym**	
ntasks nprocs	**RN**	**CN**	**RN**	**CN**	**RN**	**CN**	**RN**	**CN**	**RN**	**CN**	**RN**	**CN**	**RN**	**CN**
64	37.5	38.7	48.4	48.7	45.4	41.4	14.0	16.2	55.9	46.3	51.3	53.2	47.8	40.2
128	48.1	43.2	52.4	50.0	48.3	51.7	17.6	18.9	57.2	53.8	55.9	58.6	51.8	57.4
256	51.6	52.9	51.3	54.2	53.2	56.9	19.3	23.0	57.2	55.1	57.4	55.8	56.0	56.9
512	51.8	51.8	54.0	54.4	59.5	58.2	18.7	20.2	56.0	56.7	55.7	55.8	57.7	56.7
1024	55.3	53.5	59.8	59.5	62.1	62.0	20.2	23.8	61.5	61.6	58.6	58.3	60.2	60.3
2048	59.1	57.2	59.8	58.7	61.5	62.5	21.9	25.6	63.0	63.1	58.1	57.2	62.5	62.9

Table 9: Percentage improvement (%) for tested matrices

4.7 Multicore Parallelization of Block Cyclic Reduction Algorithm

Supported by: D. Lecas (IDRIS), R. Chevalier (IDRIS), P. Joly (LJLL)

Whitepaper: D. Lecas, R. Chevalier, P. Joly, “Multicore Parallelization of Block Cyclic Reduction Algorithm”, PRACE technical white paper.

The goal of this project is to evaluate how to parallelize the block cyclic reduction using MPI and OpenMP. This algorithm is used to solve elliptic problems much faster than the traditional iterative methods. Suppose the linear system can be written as follows, where B and T are two square matrices, U is the unknown, and F is the right hand side.

$$(P) \Leftrightarrow AX = Y \Leftrightarrow \begin{pmatrix} B & T & & & \\ T & B & T & & \\ & T & \ddots & \ddots & \\ & & \ddots & B & T \\ & & & T & B \end{pmatrix} \begin{pmatrix} U_1 \\ U_2 \\ U_3 \\ \vdots \\ U_{m_y} \end{pmatrix} = \begin{pmatrix} F_1 \\ F_2 \\ F_3 \\ \vdots \\ F_{m_y} \end{pmatrix}$$

The concept of block cyclic reduction is to iteratively eliminate half of the unknowns until there is an only single block system which can be solved directly. After the k-th elimination we have this kind of system:

$$\begin{pmatrix} B^{(k)} & T^{(k)} & & & \\ T^{(k)} & B^{(k)} & T^{(k)} & & \\ & T^{(k)} & \ddots & \ddots & \\ & & \ddots & B^{(k)} & T^{(k)} \\ & & & T^{(k)} & B^{(k)} \end{pmatrix} \begin{pmatrix} U_{2^k} \\ U_{2^k*2} \\ U_{2^k*3} \\ \vdots \\ U_{2^k*(2^{j_q-k}-1)} \end{pmatrix} = \begin{pmatrix} F_2^{(k)} \\ F_4^{(k)} \\ F_6^{(k)} \\ \vdots \\ F_{m_y-1}^{(k)} \end{pmatrix}$$

After solving the only block equation, the « odd » values are computed using the « even » values that were computed in the previous step. Buneman's variant is selected due to it numerically stable results [22].

In summary, the Buneman's variant of block cyclic reduction takes the following form:

1. Computation of Buneman's series (P and Q)

```
For $j = 1, \dots, 2^{j_q} - 1$
    $P_j^{(0)} = 0$    and    $Q_j^{(0)} = F_j^{(0)} = F_j$
    For $k = 1, \dots, j_q - 1$
        For $j = 1, \dots, 2^{j_q-k} - 1$
            $\boldsymbol{P}_{2^k*j}^{(k)} = \boldsymbol{P}_{2^k*j}^{(k-1)} - [\boldsymbol{B}^{(k-1)}]^{-1}\left[\boldsymbol{T}^{(k-1)}\left(\boldsymbol{P}_{2^k*j-2^{k-1}}^{(k-1)} + \boldsymbol{P}_{2^k*j+2^{k-1}}^{(k-1)}\right) - \boldsymbol{Q}_{2^k*j}^{(k-1)}\right]$
            $\boldsymbol{Q}_{2^k*j}^{(k)} = \boldsymbol{T}^{(k-1)}\left(\boldsymbol{Q}_{2^k*j-2^{k-1}}^{(k-1)} + \boldsymbol{Q}_{2^k*j+2^{k-1}}^{(k-1)}\right) - 2\boldsymbol{T}^{(k-1)}\boldsymbol{P}_{2^k*j}^{(k)}$
```

2. Solve the single block equation

```
For $k = j_q - 1$
    $\boldsymbol{U}_{2^{j_q}-2^{j_q-1}}^{(j_q-1)} = [\boldsymbol{B}^{(j_q-1)}]^{-1}\boldsymbol{Q}_{2^{j_q}-2^{j_q-1}}^{(j_q-1)} + \boldsymbol{P}_{2^{j_q}-2^{j_q-1}}^{(j_q-1)}$
```

3. Backward substitution

```
For $k = j_q - 1, \dots, 1$
    For $j = 2, \dots, 2^{j_q-k} - 1$
        $U_{2^k*j-2^{k-1}}^{(k-1)} = [B^{(k-1)}]^{-1}\left(Q_{2^k*j-2^{k-1}}^{(k-1)} - T^{(k-1)}\left(U_{2^k*j}^{(k)} + U_{2^k*j-2^k}^{(k)}\right)\right) - P_{2^k*j-2^{k-1}}^{(k-1)}$
    For $j = 1$
        $\boldsymbol{U}_{2^k-2^{k-1}}^{(k-1)} = [\boldsymbol{B}^{(k-1)}]^{-1}\left(\boldsymbol{Q}_{2^k-2^{k-1}}^{(k-1)} - \boldsymbol{T}^{(k-1)}\boldsymbol{U}_{2^k}^{(k)}\right) - \boldsymbol{P}_{2^k-2^{k-1}}^{(k-1)}$
    For $j = 2^{j_q-k}$
        $\boldsymbol{U}_{2^{j_q}-2^{k-1}}^{(k-1)} = [\boldsymbol{B}^{(k-1)}]^{-1}\left(\boldsymbol{Q}_{2^{j_q}-2^{k-1}}^{(k-1)} - \boldsymbol{T}^{(k-1)}\boldsymbol{U}_{2^{j_q}-2^k}^{(k)}\right) - \boldsymbol{P}_{2^{j_q}-2^{k-1}}^{(k-1)}$
```

In this algorithm, there are two levels of parallelization; the j-loop can be distributed and the $\boldsymbol{B}^{(r-1)}\boldsymbol{X} = \boldsymbol{Y}$ result in computing 2^r resolution of linear system with Cholesky decomposition. Two parallel versions of this algorithm are implemented, one with OpenMP and the other with MPI. The two-level parallelization forces the use of group of processors. With MPI, this is easy using communicators, but there is no such feature in OpenMP. So the OpenMP version cannot use the work-sharing directives (DO, SINGLE, ...), all the distribution is made explicitly using rank of threads.

Figure 6 shows the speedup curves of the two parallel algorithms on a single SMP node of Vargas, which is an IBM Power 6 composed of 112 SMP nodes p575 IH with 32 cores Power 6 per node. As seen in the figure, the OpenMP version has a lower scalability because of the global barrier used when a group barrier is needed. Managing the load balance is not easy in Buneman's series computation and this affect the scalability of the algorithm.

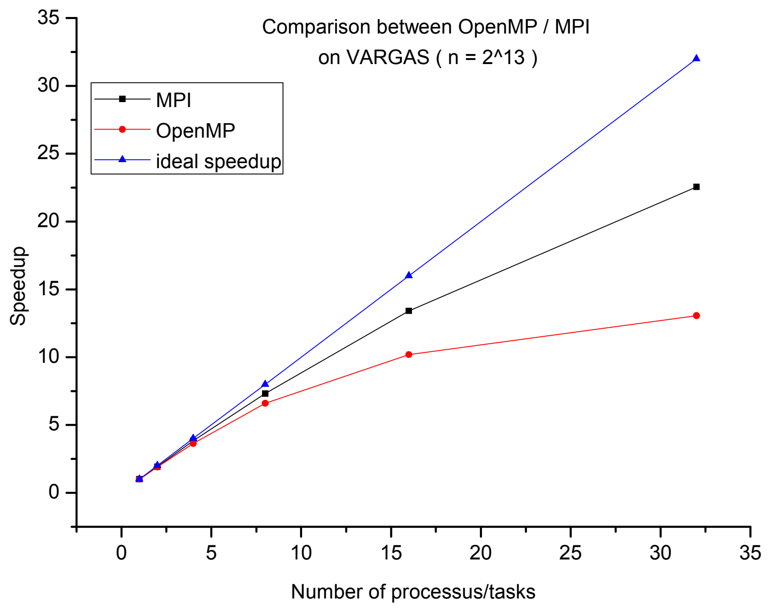


Figure 6: Speedup curves of the two parallel algorithms on a single SMP node of Vargas.

In conclusion, it's possible to parallelize the Buneman's variant of the block cyclic reduction, but there are some load balancing issues. A possible improvement is to use the task feature of OpenMP for better load balancing. Also there is a Fourier variation that exhibits more parallelism [23].

4.8 Enabling FFTE library and FFTW3 Threading in the Quantum Espresso

Supported by: D. Stankovic, D. Vudragovic, V. Slavnic, A. Jovic, P. Jovanovic (Institute of Physics Belgrade, Serbia)

Whitepaper: D. Stankovic, D. Vudragovic, V. Slavnic, A. Jovic, P. Jovanovic, "Enabling FFTE library and FFTW3 Threading in the Quantum Espresso", PRACE technical white paper.

In this project, Quantum Espresso [24] computing codes are extended to use the FFTE [25] numerical library, as well as a threaded version of FFTW3 [26] numerical library. The work has been motivated by excellent performance results of FFTE described in [27], and by the expectation that the Quantum Espresso FFTW3 hybrid approach will achieve better performance compared to the existing MPI implementation. The development is done according to [28], which defines guidelines regarding programming style (variable naming and capitalization, indentation style, use of automatic variables, use of pointers, etc.). The latest version of the modified code is available at the URL given in [29].

The development used Quantum Espresso (QE) version 5.0 (currently the latest version) as the baseline, and it focused on the parts of the code responsible for FFT. Detailed analysis of the QE code shows that all FFT routines are located in Modules/fft_scalar.f90 file of the distribution. Routines for 1D, 2D, and 3D FFT, defined in this file, serve as wrappers and invoke corresponding FFT routines of the supported numerical libraries (FFTW, FFTW3, ESSL, SCSL, MathKeisan and Sunperf). Specification of the particular FFT numerical library is performed by conditional compilation. Using the pre-processor directives (#ifdef, #elif, etc.), individual sections of the Modules/fft_scalar.f90 file are compiled, depending on which parameter macros are defined. In the case when the configuration process is successfully performed and any of the QE supported FFT numerical libraries is available in the environment, matching macros will be listed in the parameters section of the Makefile. For example, __FFTW3 macro parameter will be generated in the Makefile parameters section if the FFTW3 numerical library is available.

The project extends the QE code to use the FFTE numerical library following the described conventions. The new FFTE-specific macro parameter is introduced and named __FFTE. Further, FFTE-specific invoke routines are placed in Modules/fft_scalar.f90 file, accompanied by the appropriate pre-processor directives. Variables required to initialize and execute FFTE are introduced so as to be easily distinguishable by their prefix (ffte_). The new code is compatible with the FFTE version 5.0 (currently the latest version).

In the second part of the project, the threading support for the FFTW3 numerical library is implemented together with the existing QE MPI parallelization, creating a hybrid mode. Two realizations of threading are provided:

- implicit mode - FFTW3 library with OpenMP support is used instead of a serial FFTW3 library,
- explicit mode - inside of an OpenMP parallel region, a serial FFTW3 library is called.

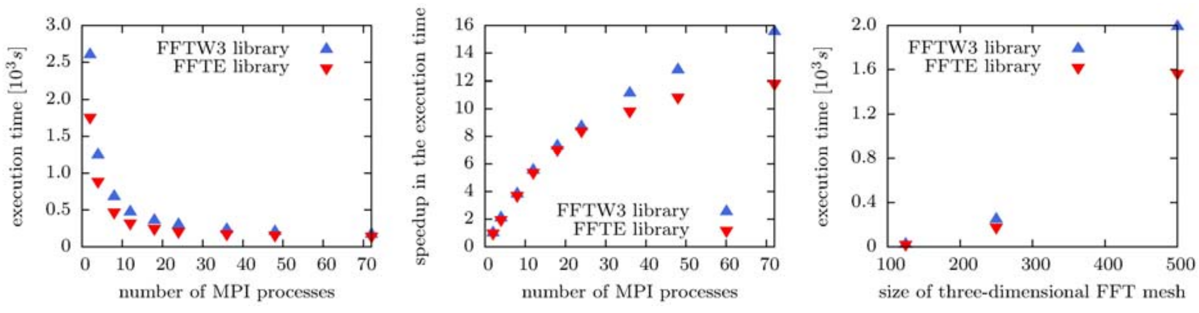
Implementation of the explicit mode is facilitated by QE FFT computation implementation. Since it is divided in many FFT 1D and 2D calls, it is possible to efficiently distribute workload among the threads. Furthermore, FFTW3 routines are thread-safe, and therefore they can be called from the multiple threads at the same time.

In order to enable threading support for the FFTW3 numerical library in QE, it is necessary to initiate configuration with --enable-openmp flag, and to edit make.sys file after the configuration process. Implicit mode is enabled by defining -D__FFTW3 (instead of -D__FFTW) and -D__FFTW3_OMP_IMPL macro parameters in make.sys DFLAGS variable. In addition, it is necessary to specify FFTW3 library location within FFT_FLAGS variable (-L/path/to/FFTW3), as well as FFTW3 linker flags in the same line (-lfftw3_omp -lfftw3). Explicit mode can be enabled using -D__FFTW3_OMP_EXPL macro parameters (instead of -D__FFTW3_OMP_IMPL), and -L/path/to/FFTW3-lfftw3 as linker flags.

Performance of the extended QE using its PW and CP components are measured. Initial configuration input files for these components are obtained from the download page of the QE web site [30], subsection Benchmarks for PW and CP components.

Execution times and scalability of the QE FFTE extension is compared with QE FFTW3 implementation. Results are obtained using NIIFI SC in Hungary [31], a 2x12-cores AMD Magny-Cours Opteron 6174 cluster with Infiniband interconnection.

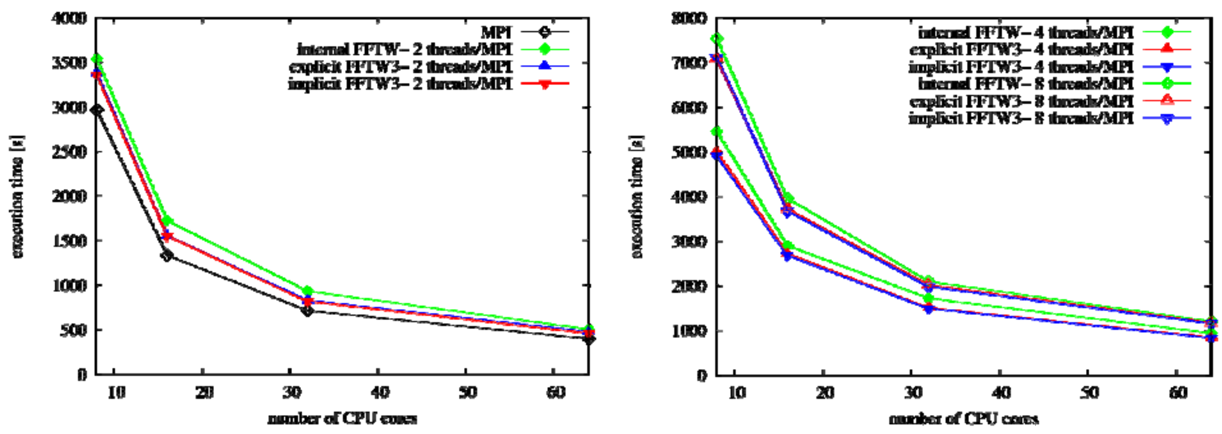
Figure 7(a) shows that the FFTE extension has better overall execution times for different numbers of MPI processes. Since the results are obtained using the fixed three-dimensional FFT mesh (125x125x125), the difference between execution times slowly decreases with an increasing number of MPI processes. This is more obvious from Figure 7(b), which shows speedup in the execution time over the same number of MPI processes. Although the speedup of FFTW3 library is better, execution times are smaller for FFTE library. On the other hand, larger FFT mesh will increase the time each process spends in FFT routine, and thus performance of QE FFTE part of the code will be more significant. This is illustrated in Figure 7(c), where execution time is given as a function of the FFT mesh size. Again, it is possible to observe that FFTE library outperforms FFTW3.



(a) (b) (c)

Figure 7: Performance of QE FFTE extension compared with the QE FFTW3 implementation: (a) Execution times of QE FFTW3/FFTE codes for different number of MPI processes; (b) Speedup in the execution time of QE FFTW3/FFTE codes as functions of a number of MPI processes; (c) QE FFTW3/FFTE execution times as functions of 3D FFT mesh size.

The project also compares performance of the implicit and explicit QE FFTW3 hybrid extensions with the internal QE FFTW hybrid support at PARADOX Cluster at IPB, a 2x4-cores Intel Xeon E5345 cluster with Gigabit Ethernet interconnection. Figure 8(a) shows execution times obtained for pure MPI version and hybrid ones when 2 threads are used per MPI process, while Figure 8(b) illustrates the case when 4 and 8 threads are used per MPI process. Both QE FFTW3 hybrid codes produced in this project give shorter execution times compared to the default QE FFTW hybrid code and no significant difference in performance between the two versions of FFTW3 hybrid implementations was noticed when using this particular QE input dataset and computing infrastructure.



(a) (b)

Figure 8: Performance of the pure MPI, implicit and explicit QE FFTW3 hybrid extensions, and QE FFTW hybrid code for various numbers of CPU cores used: (a)Execution times when pure MPI and 2 threads per MPI process hybrid versions are used; (b) Execution times when 4 and 8 threads per MPI process hybrid versions are used.

QE FFT extension produced in this project shows better performance compared to the default QE FFT. In the case of FFTE extension, this performance improvement could be significant when large charge density FFT mesh is required by the configuration of physical system. QE FFTW3 hybrid implicit and explicit extensions illustrate better performance compared to QE FFTW internal hybrid approach, but still worse than the pure MPI one. As evidenced in [32], this is probably because the overhead related to thread management outweighs the benefits of reduced MPI communication, up to a certain number of MPI processes.

4.9 A Hybrid Hermitian General Eigenvalue Solver

Supported by: R. Solcà (ETH Zürich)

Whitepaper: R. Solcà, T. C. Schulthess, A. Haidar, S. Tomov, I. Yamazaki, J. Dongarra, "A hybrid Hermitian general eigenvalue solver", http://arxiv.org/abs/1207.1773

In this project the focus is on dense eigensolvers, and in particular, generalized Hermitian-definite problems of the form Ax=λBx, where A is a Hermitian dense matrix and B is Hermitian positive definite.

Many scientific computing applications, ranging from computing frequencies of waves that will propagate through a medium, to earthquake response of a bridge, or energy levels of electrons in nanostructure materials, require the solution of eigenvalue problems. These solvers are also needed for solving electronic structure problems in material science and chemistry [33]. In particular we will focus on algorithms that can compute either the complete eigenspace, or a fraction (typically 10-20%) of it.

Particularly problems with modest matrix dimensions of a few thousand to ten or twenty thousand seem to pose a challenge for most practical purposes. Typically these problems must be solved many times in the context of a parallel code. The known power limits in current processors, that would prevent the clock frequency to keep increasing with time, necessitates the solution of the eigenvalue problem on nodes with larger number of threads that are executed on slower cores.

In this project, in collaboration with the Innovative Computing Laboratory of University of Tennessee, in the context of the MAGMA project, single node hybrid (CPUs + GPU, and CPUs + GPUs) general eigenvalue solvers are being developed.

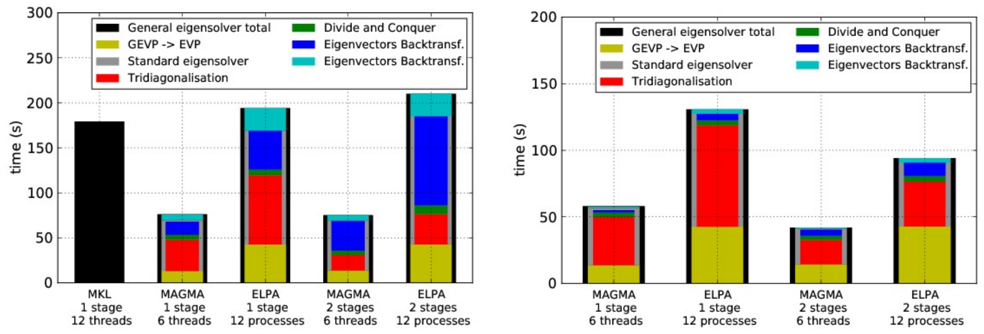


Figure 9: Time to solution of double complex general eigensolver with matrix size 8000. On the left the whole eigenspace is computed, on the right only 10% of the eigenvectors.

Figure 9 shows the results of the hybrid algorithms compared with a shared memory library (MKL) and a distributed memory library (ELPA) [34] with two different algorithms. To have a fair comparison on a node of the Castor cluster located at CSCS (Each node is a dual 6-cores Intel Xeon 5650 with two Nvidia M2090 system) we compare our hybrid routines, tested with six threads on one CPU socket and one GPU, against non-GPU routines tested using both the CPU's sockets – 12 threads for shared-memory routines and 12 processes for distributed-memory processes [36]The two algorithms differ in the tridiagonalisation step. The one-stage approach reduces the matrix to tridiagonal in one step using Householder transformations. This approach spends most of the time in the tridiagonalisation that is inefficient since half of the operation are level 2 BLAS (memory bounded) operations. The two-stage approach [35] first reduces the matrix to a band matrix and then, using a bulge chasing technique, reduces the band matrix to be tridiagonal. Unfortunately this increases the time needed for the transformation of the eigenvectors.

Both the one-stage and two-stage approaches are comparable while computing the whole eigenspace, whereas the two-stage approach is faster when only a fraction of the eigenvectors is computed.

The maximum matrix size that can be used in the solver is limited by the memory available on the GPU. One possibility to get around this problem is to implement the algorithm on a multiGPU system.

Figure 10 shows the time spent in each step of the multiGPU one-stage algorithm on a 2 6-cores Intel Xeon 5660 and an 8 Nvidia M2090 system.

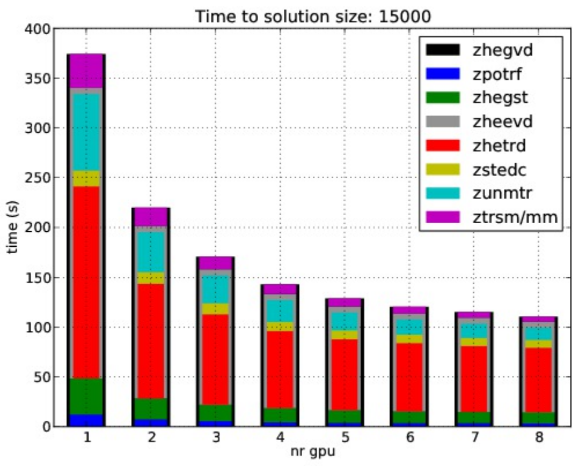


Figure 10: Time to solution for different number of GPUs of double complex general eigensolver with matrix size 15000 and whole eigenspace.

4.10 A Generic Library for Stencil Computations

Supported by: M. Bianco (ETH Zurich/CSCS)

Whitepaper:M.Bianco, U.Varetto,"A Generic Library for Stencil Computations", http://arxiv.org/abs/1207.1746

This project develops the GSCL (*Generic Stencil Computing Library*) to let application programmers specify quite general and widely used algorithms in a very synthetic way, namely the solution of finite difference equations on regular grids and lattice methods, for instance for fluidodynamics. The resulting pattern is more general and allows the use of GSCL in other contexts, for instance many typical dynamic programming algorithms such as longest common subsequence.

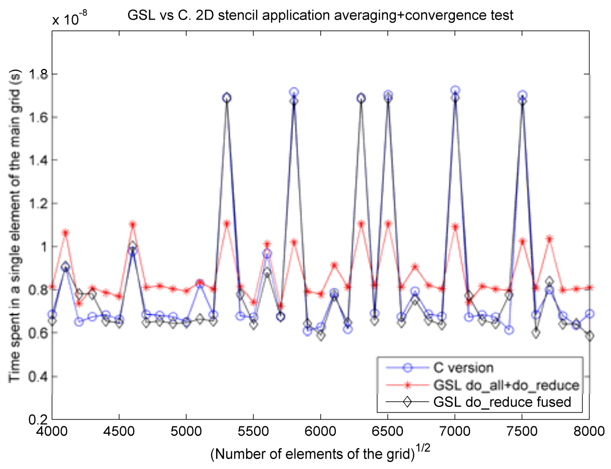
Given the heterogeneity of current computer architectures, raising the level of abstraction can be a useful solution to exploit such diversity. Stating *for all elements of the grids apply the function operator* is a very general way of expressing the need of the computation that can be then implemented specifically for each particular platform. In general, raising the level of abstraction is suitable for specific classes of computations, as in the case of this project, to limit the complexity of the resulting language, in our case at a level of a C++ library.

In the development, the approach of *generic programming* is followed, in which there always exists a generic (possibly) slow implementation for any construct. If some other implementation is found that is better suited for the platform or the input, then it is selected and used in the implementation. Ideally, the efficient implementation is developed by some library developer that may or may not be acquainted with the application domain of the final application. This may sound not satisfactory to the application developer, since the library developer's response may not be fast enough. By keeping GSCL architecture as simple as possible, the possibility for the user to implement, and possibly contribute, to the GSCL itself, with their own specific implementations is left open. From the productivity point of view, this has the advantage that, if the proper specialization already exists, the user can remain at the high level of abstraction and obtain the best sustained performance; while if this is not the case, the development is naturally split in two phases. Initially there is a quick deployment of the program at high-level, and then increasingly more efficient versions can be produced. Those versions do not impact the high level code which can still be read and understood by the application programmers in a much better way than the low-level dirty-but-fast detailed implementation targeted to the architecture.

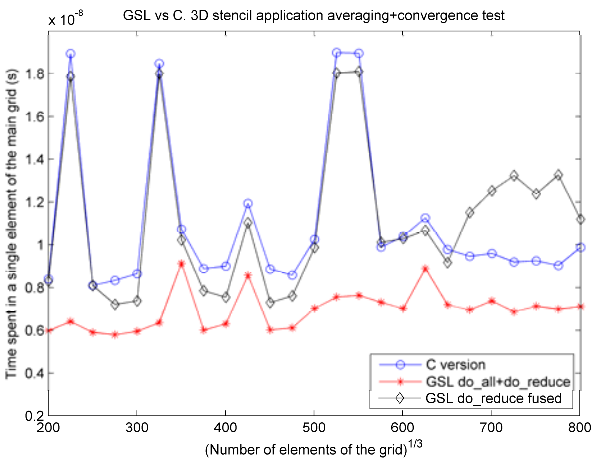
The diversity of the available platforms imposes certain restrictions in the high-level constructs that the GSCL supply and their relations. The consequences of this are tried to be kept at a minimum, but they have impact on the guaranteed semantics of the operations performed by a program. Many restrictions are due to the limitations of the C++ language, others are dictated by the need of exploiting *accelerators* like GPUs, nominally capable of high throughput that is the main characteristic needed by a stencil computation. Also these problems could be removed if GSCL was a stand-alone language. A library development instead of a new language is preferred in order to utilize all the features of an existing language, like abstraction mechanisms, and the availability of a highly sophisticated, reliable, and widely adopted compiler technology.

Figure 11 compares GSCL and C on a sample application that runs an averaging followed by a reduction to check convergence for different input sizes. Figure 11 (a) and Figure 11 (b) show the results for 2D and 3D cases, respectively. The given times are the average time spent on an element of the grid.

As seen in Figure 11(a), in the 2D case, the GSCL version with a single fused reduction is on par with the C version. The peaks at certain given sizes are due to memory architecture issues. It is interesting to note that the implementation with two distinct iteration spaces has a more stable behavior than the other two, even though it tends to be slower in the best case. The reason is due to the cache of the system is less stressed by splitting the operations, thus behaving more smoothly.



(a)

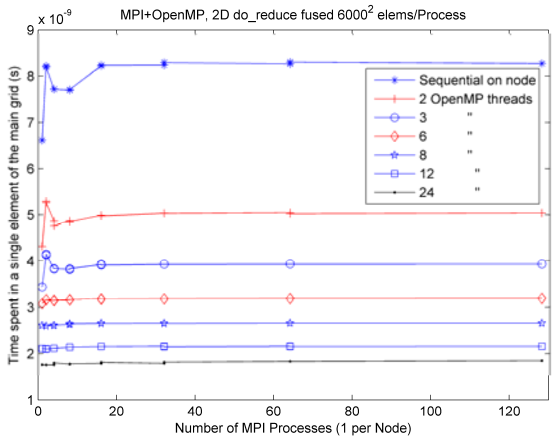


(b)

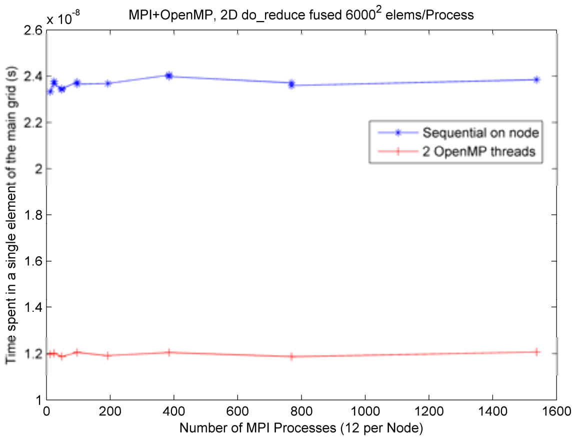
Figure 11: Comparison of GSCL and C99 code on (a) 2D and (b) 3D grids. The fused version (GSL_do_reduce fussed) is algorithmically equivalent to the C version, while the other (GSL_do+all+do_reduce) does an additional scan of the memmory since the loops are not fused.

In Figure 11(b) the results show that GSCL is still the fastest, but the two-iteration spaces implementation is actually faster than the others. This is again due to the memory pressure since the stencil operator accesses 6 elements at different strides instead of 4 in the 2D case.

The two algorithmic equivalent versions are still on par up to a 6400x6400 input size. After that the C++ version exhibits slightly higher execution times. Note that, all the optimizations available for the C version could be applied to the GSCL versions, with the advantage that main application code is not affected. Also, the difference between the two iteration spaces and the fused operators versions differ by only a couple of lines of code.



(a)



(b)

Figure 12: Running GSCL code on a Cray XT5 machine. Each node has 24 cores divided in two sockets. (a) One MPI process per node is used and then a varying number of OpenMP threads for process are used. (b) The node is half filled with MPI processes and none or 2 OpenMP threads are utilized.

To prove that GSCL can run efficiently on traditional parallel machines, the same application used in Figure 11 is implemented in C, and compiled using MPI and OpenMP. A weak scaling experiment, in which each MPI process has a tile of 6000x6000 elements, is performed. Results shown in Figure 12(a) place a single MPI process in a node of the machine. The process then employs from 1 to 24 openMP threads to work on the tile. As can be seen, the time with one OpenMP thread is the same as the sequential time shown in Figure 11(a). With increasing number of OpenMP threads the execution time improves as expected.

Figure 12 (b) shows the same experiment by placing 12 MPI processes on each node. As expected, the execution time increases due to contention of the many processes. The weak scalability of the application is, however, very good.

4.11 Design and Implementation of New Hybrid Algorithm and Solver for Large Sparse Linear Systems

Supported by: A. Duran (ITU), M. S. Celebi (ITU-UHeM), M. Tuncel (ITU-UHeM)

Whitepaper: A. Duran (ITU), M. S. Celebi (ITU-UHeM), M. Tuncel, "Design and Implementation of New Hybrid Algorithm and Solver for Large Sparse Linear Systems", PRACE technical white paper. Related academic publication "Scalability of SuperLU solvers for large sparse linear systems" has been accepted for oral presentation at the SPE & SIAM Conference on "Mathematical Methods in Fluid Dynamics and Simulation of Giant Oil and Gas Reservoirs", 3-5 September 2012, Istanbul, Turkey.

It is important to have a fast, robust and scalable algorithm to solve a sparse linear system $AX=B$ in many science and engineering applications. In this project, a new hybrid algorithm and solver for large sparse linear systems is designed and implemented. Scalable direct solvers are considered for various reasons. First, the effectiveness of the SuperLU_DIST 3.0 for distributed memory and SuperLU_MT 2.0 for shared memory parallel machines among several sparse direct solvers is examined (see [37], [38], [39], [40], [41], [42]).

SuperLU_MT (see [44]) has three major steps including sparsity ordering, factorization that arranges partial pivoting, symbolic factorization and numerical factorization steps to perform in an alternating fashion, and triangular solution. While SuperLU_DIST uses BLAS 3 for factorization, SuperLU_MT has only BLAS 2.5 with multiple matrix vector multiplication. Therefore, SuperLU_DIST outperforms SuperLU_MT (see [45]). SuperLU_DIST (see [46]) uses static pivoting [47] instead of partial pivoting because the implementation of numerical pivoting is complicated on distributed memory architecture. It is advantageous that symbolic and numerical factorization steps can be separated due to the static pivoting. On the other hand, the backward error of a matrix cannot be decreased to machine precision and SuperLU_DIST may be considered for a certain types of matrices. Therefore, it is important to determine and classify those matrices where SuperLU_DIST works well. The maximum matching algorithm (see Duff and Koster [49]) is utilized to maximize the product of the magnitudes of the diagonal entries for a matrix. SuperLU_DIST can use ParMeTiS [50] or MeTiS [51] ordering on the structure of $A+A^T$ in addition to the multiple minimum degree ordering on the structure of $A+A^T$ or A^TA for fill-in reducing preordering. Unlike sequential SuperLU, SuperLU_DIST does not have a COLAMD option that works well for many unsymmetric sparse matrices to reduce fill-ins.

In this work, advantages and limitations of the SuperLU solvers are discussed. Although the existing versions of SuperLU are scalable and tuned for many matrices, they are sensitive to tuning and needs further customization for various large sparse matrices. Therefore, a collection of large patterned and random sparse matrices, which are larger than most of the real matrices from the University of Florida sparse matrix collection [19], are generated. Sensitivity analyses for several parameters including number of nonzero (NNZ) and sparsity level for randomly located sparse matrices are performed. SuperLU_DIST shows scalable speed-up between 256 and 512 cores for many test matrices, for example in Figure 13, on the Linux Nehalem Cluster available at UHeM [52]. On the other hand, for randomly located large sparse matrices, numerical factorization, symbolic factorization, and consequently wall clock time spike up around the sparsity level of 7 related to the ability to find supernodes, for example see Figure 14 and Table 10. The wall clock time decreases gradually as sparsity level decreases from 9 to 75 with a slow rise at 100 number of nonzeros per row. Moreover, the memory overhead coming from ParMeTiS becomes one of the dominating factors in the overall runtime on n-diagonal sparse matrices. Furthermore, new unsymmetric matrices, which consist of the lower triangular part of a symmetric matrix and an upper subdiagonal with d distance from the main diagonal are also generated. While SuperLU_DIST performs properly for symmetric matrices, it produces segmentation fault for the corresponding new unsymmetric matrices.

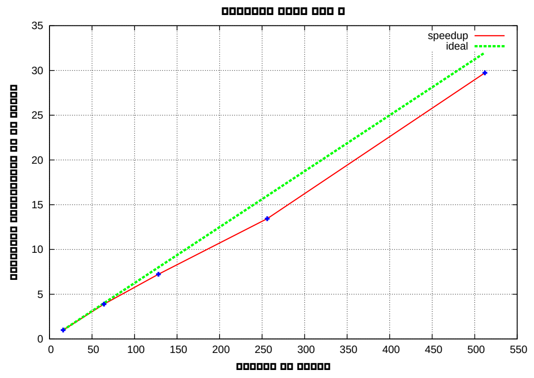


Figure 13: Speed up for matrix RAND_40K_3.

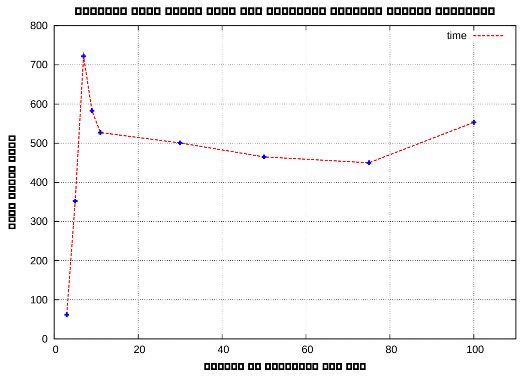


Figure 14: Average wall clock time as a function of various sparsity levels for randomly located sparse matrices.

NNZ per row	3	5	7	9	11	30	50	75	100
Wall clock time	6187	352.10	721.95	583.15	527.20	500.66	465.00	450.08	553.23

Table 10: Wall clock time for randomly located sparse matrices as sparsity level decreases with 64 core (8x8)

The code of SuperLU_MT has been tested up to 64 threads for all sparse matrices in the list on the HP Integrity Superdome SD32B (see[53]) computing server available at UHeM. Speedup between 4 and 32 is achieved depending on the sparsity level, NNZ and structural symmetry, as shown in [44] with different machines. Finally, very large sparse matrices with less sparsity for which SuperLU_DIST works well while SuperLU_MT gives segmentation fault related to memory usage are also generated.

In this project the scalability of the SuperLU solver is improved via several ways. A new hybrid algorithm utilizing the MPI+OpenMP hybrid programming approach that combines the advantages of SuperLU_DIST and SuperLU_MT and diminishes some of their limitations is proposed so that it is possible to avoid extra communication overhead with MPI within nodes.

4.12 Scalable and Improved SuperLU on GPU for Heterogeneous Systems

Supported by: A. Duran (ITU), M. S. Celebi (ITU-UHeM), M. Tuncel (ITU-UHeM)

Whitepaper: A. Duran (ITU), M. S. Celebi (ITU-UHeM), M. Tuncel, "Scalable and improved SuperLU on GPU for heterogeneous systems", PRACE technical white paper.

It is important to use graphic processing units (GPU) as accelerators when we consider a fast, robust and scalable solver for a sparse linear system $AX=B$ in many science and engineering applications. In this project, a new parallel hybrid direct solver is designed and implemented on GPU for large sparse linear systems. In particular, GPU programming using directive based Open ACC is used in order to obtain a scalable and improved SuperLU on CPU+GPU heterogeneous systems.

As a first step, the effectiveness of the SuperLU_DIST 3.0 for distributed memory and SuperLU_MT 2.0 for shared memory parallel machines among several sparse direct solvers (see [37],[38],[39],[40],[41],[42]) on CPU and (see [43] for small matrices) on CPU-GPU are studied. SuperLU_MT (see [44]) has three major steps including sparsity ordering, factorization that arranges partial pivoting, symbolic factorization and numerical factorization steps to perform in an alternating fashion, and triangular solution. While SuperLU_DIST uses BLAS 3 for factorization, SuperLU_MT has only BLAS 2.5 with multiple matrix vector multiplication. SuperLU_DIST (see [46]) uses static pivoting [47] instead of partial pivoting because the implementation of numerical pivoting is complicated on distributed memory architecture. It is advantageous that symbolic and numerical factorization steps can be separated due to the static pivoting. Therefore, SuperLU_DIST outperforms SuperLU_MT (see [45] and [48]) for many sparse matrices.

Second, SuperLU is a complex algorithm and it is important to choose right combination for better intra-node communications and inter-node communications within CPU+GPU heterogeneous systems, given current technology limitations and developments. While SuperLU_MT is a good starting reference for intra-node communications, SuperLU_DIST is more appealing for GPU clusters having inter-node communications using infiniband (IB) network among its several advantages. In this project, SuperLU_DISTs features such as the usage of the extract parallelism reducing communication by avoiding and defining dependencies of data in addition to the usage of static pivoting and BLAS 3 for factorization are utilized. Moreover, utilization of multicore approach inside node analogous to SuperLU_MT is also performed. The first goal is to complete intra-node multi-GPU programming. Next step would be inter-node multi-GPU programming.

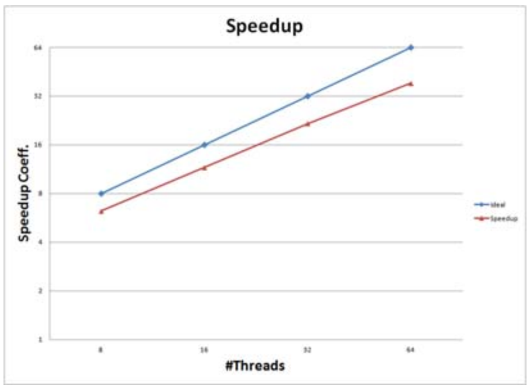


Figure 15: Speed up for matrix RAND_40K_3.

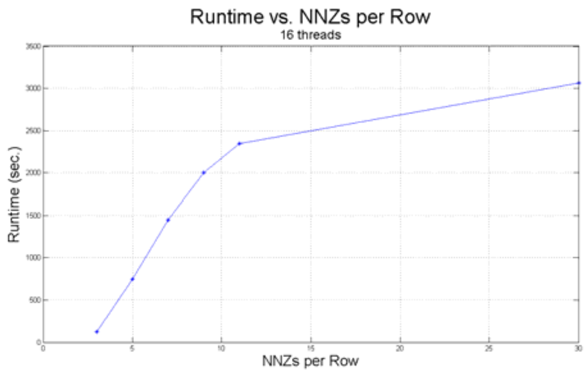


Figure 16: Average wall clock time as a function of various sparsity levels for randomly located sparse matrices.

NNZ per row	3	5	7	9	11	30
Wall clock time	124.46	750.10	1448.22	2003.09	2346.67	3062.57

Table 11: Wall clock time using SuperLU_MT for randomly located sparse matrices as sparsity level decreases with 16 cores

The code of SuperLU_MT has been tested up to 64 threads for randomly located sparse matrices on HP Integrity Superdome SD32B (see [53]) computing server available at UHeM. Almost linear speedup is achieved (see for example Figure 15 for RAND_40K_3). Moreover, the scalability of SuperLU_MT depending on the sparsity level in terms of NNZ per row is also tested. Figure 16 and Table 11 show that average wall clock time increases slowly as sparsity levels decrease for randomly located sparse matrices of order 30000.

In sum, after obtaining a robust version of scalable SuperLU, a new hybrid algorithm for CPU+GPU heterogeneous systems by taking SuperLU_DIST as a starting reference is designed in this project. Directive based parallelization approach using OpenACC for CPU+GPU heterogeneous systems is implemented.

4.13 Optimization of SHAKE and RATTLE Algorithms

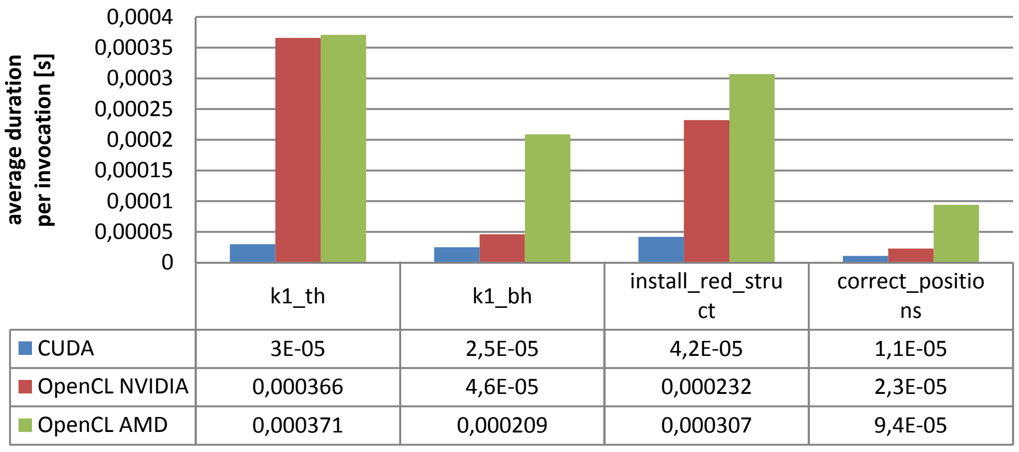
Supported by: M. Uchroński (WCSS), M. Gębarowski (WCSS), A. Kwiecień (WCSS)

Whitepaper: M. Uchroński, M. Gębarowski, A. Kwiecień, “Optimization of SHAKE and RATTLE Algorithms”, PRACE technical white paper.

This project is focused on optimization of SHAKE and RATTLE algorithms using the DL_POLY Molecular Simulation Package [54]. The project evaluates these algorithms and develops the OpenCL versions of the main parts of them (Leapfrog Verlet and Velocity Verlet integration schemes). The main goal is to increase the potential of the algorithms to support asynchrony and check the possibility of improving the accuracy of the GPU code.

The SHAKE is a two stage algorithm based on the Leapfrog Verlet integration scheme. The RATTLE algorithm fits within the concept of the Velocity Verlet integration scheme [55]. These algorithms are widely used in molecular dynamics simulations and for this reason are relevant for a broad range of scientific applications. DL_POLY application already contains implementations of the SHAKE and RATTLE algorithms (on CPU and some parts on GPU using CUDA), and an OpenCL partial implementation, developed by WCSS (within Work Package 7 of PRACE-1IP [59]).

In this project, implementation of the SHAKE algorithm for DL_POLY application has been continued and the code has been analyzed for further optimizations. Tests have been performed on local WCSS GPU machines (2x GTX480 [57], 2x AMD Radeon HD 6900 Series [58]). Performance results for H2O benchmark show that the OpenCL implementation runs slower than the CUDA version of the same algorithms (Figure 17). The biggest performance difference between the NVIDIA-CUDA and the NVIDIA-OpenCL implementations occurs for kernels: *k1_th* (OpenCL code is 10 times slower than CUDA code), and *install_red_struct* (OpenCL code is 5.5 times slower than CUDA code). For other kernels OpenCL calls are 2 times slower than particular CUDA calls.

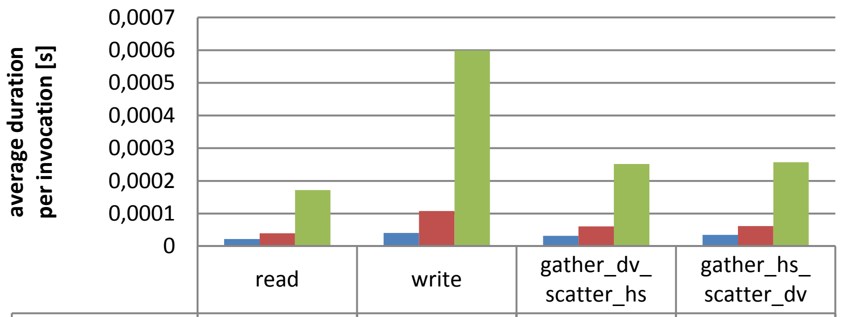


	k1_th	k1_bh	install_red_struct	correct_positions
🟦 CUDA	3E-05	2,5E-05	4,2E-05	1,1E-05
🟥 OpenCL NVIDIA	0,000366	4,6E-05	0,000232	2,3E-05
🟩 OpenCL AMD	0,000371	0,000209	0,000307	9,4E-05

Figure 17: CUDA vs OpenCL kernels for DL_POLY constraints shake component.

The results presented in Figure 18 reveal some of the performance bottlenecks of SHAKE OpenCL implementation. In this table, the kernels gather_dv_scatter_hs and gather_hs_scatter_dv are both used for improving efficiency of data transfers between host (_hs) and GPU device (_dv). The data transfers are required for a synchronization of MPI processes and they are large in volume. The kernels are packing or unpacking data in parallel on a device, depending on a direction of communication:

- Initialization of the OpenCL environment – a possible solution for this issue is to divide the OpenCL routines to separate contexts, for example create one OpenCL context for SHAKE algorithm and another OpenCL context for RATTLE algorithm;
- GPU I/O operations – this is a well known issue for GPU computations;
- Synchronization between MPI processes in a multi-GPU environment – this kind of synchronization requires copying data from GPU memory to local memory, synchronizing the data and then copying synchronized data back to the GPU memory. So far, the only solution for this issue in MPI+OpenCL code is to minimize the number of synchronization operations.



	read	write	gather_dv_ scatter_hs	gather_hs_ scatter_dv
🟦 CUDA	2,2E-05	0,000041	3,2E-05	3,5E-05
🟥 OpenCL NVIDIA	4E-05	0,000108	6,1E-05	6,2E-05
🟩 OpenCL AMD	0,000172	0,000599	0,000252	0,000257

Figure 18: CUDA vs OpenCL I/O and communication for DL_POLY constraints shake component.

The RATTLE algorithm has been analyzed and parts of it identified as good candidates for optimization by moving computations to GPU. These parts have been successfully ported to OpenCL and partially integrated with the DL_POLY application. Results for a subset of test cases [56] are presented in Table 12. Tests have been performed at WCSS (2x GTX480). The results are promising, but as for the SHAKE algorithm the OpenCL environment initialization and I/O operations constitute performance bottlenecks. Performance results in general show that selected parts of Fortran code can be executed faster on GPU using OpenCL (kernel computation time) but the bottlenecks have a great influence on overall computation time. Further optimization work should be focused on decreasing this influence.

Benchmark	Fortran	OpenCL			Speedup
		Initialization, I/O	Kernels	Overall	
H2O	0.001101	0,001792	0,000294	0,002086	0,5278
TEST3	0,152225	0,011471	0,008079	0,01955	7,7864
TEST4	1,247404	0,048104	0,062032	0,110136	11,3260
TEST7	0,134984	0,022904	0,017279	0,040183	3,3592
TEST8	1,09763	0,075339	0,136363	0,211702	5,1847
TEST13	0,162882	0,02114	0,018883	0,040023	4,0697
TEST14	1,280869	3,031746	0,088209	3,119955	0,4105

Table 12: Execution times for the RATTLE algorithm integrated with the DL_POLY code (in seconds).

4.14 Optimization of FHP Algorithms

Supported by: S. Szkoda (WCSS), A. Kwiecień (WCSS)

Whitepaper: S. Szkoda, Z. Koza, M. Tykierko "Accelerating cellular automata simulations using AVX and CUDA", http://arxiv.org/abs/1208.2428.

The FHP model [60] was introduced by Frish, Hasslacher and Pomeau in 1986 as a Cellular Automaton algorithm which is designed to solve the Navier-Stokes equation derived from Newtonian Mechanics to describe the motion of fluid substances. Throughout the years many researchers have examined FHP usability and compared it to methods that have an established position in science, like Finite Element Method or Lattice Boltzmann Method. FHP methods are used for Computational Fluid Dynamics (CFD) simulations and the area in which they are most useful is Physics of Porous Media [61]. For simulating even very small porous specimen a very large cellular automata lattice is needed, which is why a well optimized parallel implementation usage is essential [62]. Every step of a system evolution in the FHP algorithm is split in two sub-steps: motion and collision. In the first sub-step particles move to the nearest neighbor node according to their velocity, which for every particle is given by:

$$\vec{v_i} = \left(\cos\left(\frac{\pi}{3}i\right), \sin\left(\frac{\pi}{3}i\right)\right), \qquad i = 1, \dots, 6$$

In the second sub-step the state of the system is changed through collisions with respect to the rules shown on Figure 19.

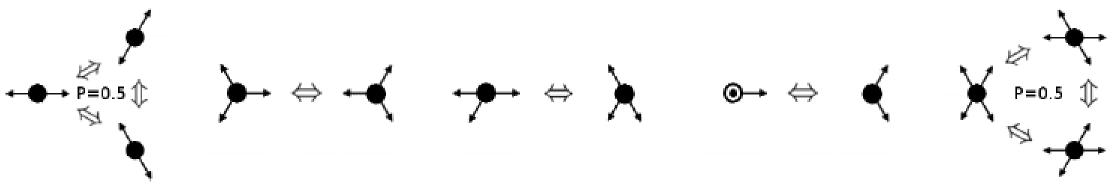


Figure 19: FHP III collision rules

This project investigates the possibility of accelerating FHP algorithms using the Single Instruction, Multiple Data (SIMD) approach. The overall goal of the optimization is to increase the asynchrony and performance of the algorithms by making better use of today's multi-core architectures. The work focuses on modifications of the FHP algorithms and addresses the memory access and domain decomposition schemes.

The FHP algorithm was implemented with the use of main parallel programming technologies: POSIX Threads, MPI, SSE, AVX and NVIDIA CUDA. Each of the implementations was compared to a single-core code, optimized only with the "-O3" GNU GCC compiler flag. Hardware used for testing was a variety of common desktop and server processors. Correctness of each implementation has been tested by measuring tortuosity and mass conservation law.

The general idea for the domain decomposition is to divide a grid of nodes into pieces from which each one is calculated on a separate processing unit. For implementations using different technologies some optimizations to the decomposition scheme are introduced, taking into account memory access and communication patterns. The scheme adopted for the single GPGPU implementation assumes that sub-domains occurring in collision simulation step are independent so a simple division is sufficient. In the motion step each sub-domain has to exchange the information about its outermost nodes with its neighbors. The developed solution expands all sub-domains so that they overlap their outermost parts, what reduces the communication and memory copies. The size of a sub-domain is set up so that it fits in a shared memory of a graphics processing unit. In the case of implementations done with the use of Pthreads and SSE/AVX the domain decomposition for the motion step is simply done by splitting it between the CPU cores. Each core collectively moves particles from the grid nodes (16 in SSE, 32 in AVX) in each direction. A synchronization of the edges of sub-domains has to be done. The collision step is implemented by look-up table so the vector extensions are not used. The MPI implementation extends previous ones by running multiple MPI-processes which may use CPU cores or GPGPUs for calculations. In the motion step MPI processes have to exchange with their neighbors the boundary columns of sub-domains assigned to them.

The computational efficiency of each implementation has been measured in Million lattice site updates per second (Mups). The cost of processing units and electric power consumption during computations was estimated and the cost and energy efficiency of each implementation calculated. Example results are shown in Table 13 (full table is available in the whitepaper).

	single-core	multi-core Pth+SSE	multi-core MPI	multi-core Pth+AVX	single-GPGPU	multi-GPGPU
Processing unit type	Intel i7 3.2GHz	Intel i7 3.2GHz	2x Intel Xeon 2.67GHz	Intel E3-1270 3.4GHz	Nvidia GTX480	2x Nvidia GTX480
Total CPU cores	1	4	12	4	1	2
Total CPU threads	1	4	12	8	1	2
Mups	21	340	230	657	1746	3493
USD/Mups	14,29	0,88	8,70	0,52	0,46	0,37
W/Mups	6,19	0,38	0,83	0,12	0,30	0,27

Table 13: Computational, cost and power efficiency of FHP implementations on different processing unit types.

The cost of hardware components (in USD) is taken as a market prize in July 2012. For the GPGPUs the cost incorporates the corresponding CPUs. The power consumption (in watts) is estimated based on nominal values delivered by producers. The analysis of the power and cost efficiency reveals that, in case of the FHP algorithms runs, modern multi-core CPUs are able to compete with the GPGPUs, if the economic factors are taken into account.

Main results of the work are:

1. 160 times acceleration of single-core calculations via CUDA, MPI and two NVIDIA GTX480 graphics cards.
2. Possibility of reducing this enormous disproportion in calculation speed from 160 times to 10 times using a full functionality of a common desktop processor (i7 960, Pthreads+ SSE) and to 5 times with the use of a modern powerful server computing unit, like Intel Xeon E3-1270 (Pthreads+AVX).

4.15 FETI Coarse Problem Parallelization Strategies and Their Comparison

Supported by: T. Kozubek (VSB), D. Horak (VSB), V. Hapla (VSB)

Whitepaper: T. Kozubek, D. Horak, V. Hapla, "FETI Coarse Problem Parallelization Strategies and Their Comparison" and T. Kozubek,_, M. Jarosova, M. Mensik, A. Markopoulos, "Hybrid Total FETI Method", PRACE technical white papers.

Parallelization of FETI/TFETI (Finite Element Tearing and Interconnecting / Total FETI) can be implemented mostly using data-parallel technique – distributing matrix portions among processing units. This allows algorithms to be almost the same for the sequential and parallel case; only data structure implementation differs. Most of computations (subdomain problems) appearing are purely local and therefore parallelizable without any data transfers. However, if we want to accelerate also dual actions, some communication is needed due to the primal-dual transition. Distribution of primal matrices is quite straightforward as every subblock reflects a subdomain. They can be implemented using general distributed column-block or row-block matrix type with nonzeros only in diagonal blocks. However, some of the primal data possess nice block-diagonal layout and can be implemented more sophisticatedly using block-diagonal composite type, where subblocks are ordinary sequential matrices and every node holds an array of them. Nevertheless this is not directly implemented in most of parallelization libraries.

A natural effort using massively parallel computers is to maximize the number of subdomains so that sizes of subdomain system matrices (stiffness matrices in mechanics) are reduced which accelerates their factorization and subsequent pseudoinverse application, which belong to the most time consuming actions. On the other hand, the negative effect of that is an increase of the null space dimension and the number of Lagrange multipliers on subdomain interfaces, i.e. dual dimension, so that the bottleneck of the TFETI method becomes in this case the action of the projector $Q=G^T(GG^T)^{-1}G$ on a vector which includes the solution of the coarse problem $GG^T x = b$ for given vector b and the rectangular matrix $G = R^TB^T$. Here B denotes the constraint matrix ensuring the gluing and Dirichlet conditions and columns of R span the kernel of the system matrix. We verified that for given vectors v and w the actions Gv and G^Tw take approximately the same time for different G matrix distributions (assembled distributed into horizontal blocks, assembled distributed into vertical blocks, unassembled kept in the form R^TB^T). Consequently the Q action time and level of communication depend first of all on the solution strategy used to solve the coarse problem itself. This can be hardly implemented sequentially on the master core for large scale problems because of increasing memory requirements and losing parallel scalability.

In this project, the set of parallelization strategies tested within PRACE-1IP [7.5]are extended by new strategies based on orthonormalization of G and exploiting MUMPS library. These strategies are tested and compared on both academic and complex engineering benchmarks and their implementation details are discussed in the context of the FLLOP (Feti Light Layer on Petsc) library regarding to computational and programming effectiveness. The machine used for benchmarking is the Hector system at the EPCC site.

The tested strategies are as follows: (1) iteratively using PCG, (2) directly using Cholesky factorization, (3) applying explicit inverse of GG^T, (4) eliminating the coarse problem - provided that the rows of G are orthonormalized.

The groups of cores used for parallel solution of coarse problem - so called subcommunicators - arise from splitting all cores in the global “world” communicator using PETSc built-in pseudopreconditioner PCREDUNDANT specifying by *Nred* the number of these subcommunicators (number of cores doing redundant work), i.e., the number of cores in each subcommunicator is equal to the number of cores / *Nred*.

Here only the results of the most successful method, which is strategy (2) with Cholesky factorization implemented using MUMPS, are reported. Initially, the whole G matrix is transferred to all subcommunicators, which compute GG^T using matrix-matrix multiplication. Then the coarse problem is solved directly using the Cholesky factorization implemented in parallel using MUMPS on subcommunicators. This strategy has a big advantage consisting in the reduction of memory requirements comparing to the factorization on the master core. Practically, there are no limits because of possible attachments of more cores into the subcommunicators.

Numerical experiments were run on matrices and vectors obtained from the decomposition and the discretization of real world elastostatic problem of the car engine block (Figure 20).

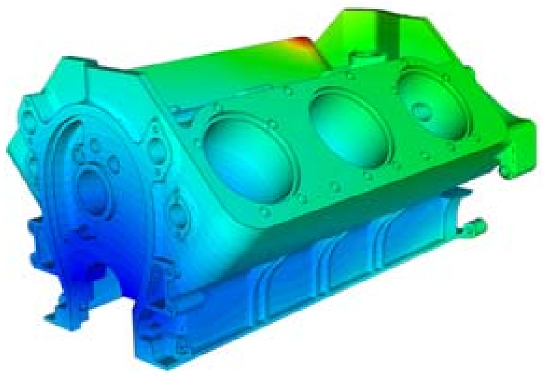


Figure 20: Car engine block – total displacement distribution

The sensitivity of the solution strategy (2) to the number of cores *Nred* doing redundant work is shown in Figure 21 for a problem of size 2,839,734 and the number of cores (subdomains) 1,014. We see that the optimal *Nred* is about 75.

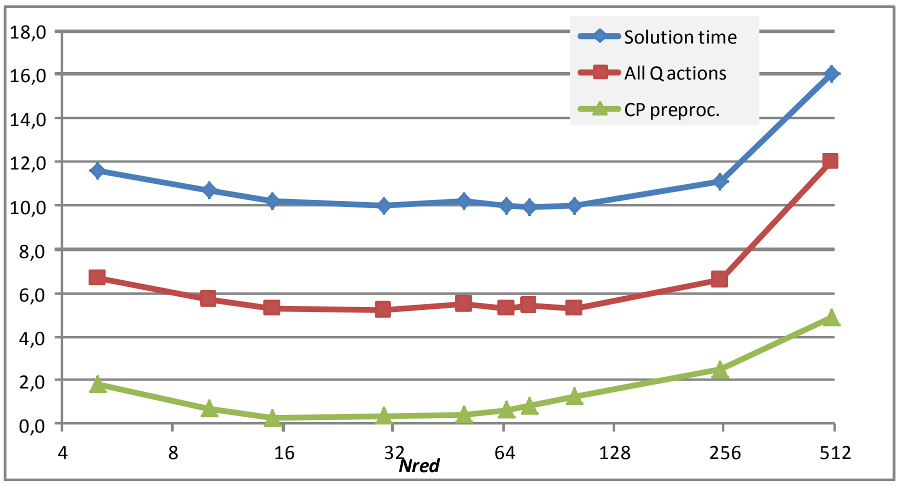


Figure 21: Sensitivity of the solution strategy (2) to Nred

The performance of strategy (2) with *Nred*=75 for a problem size of 98,214,55 and the number of cores (subdomains) 5,012 is highlighted in Table 14.

Parallelization strategy	Coarse problem preproc. [sec]	All coarse problem solutions [sec]	All Q actions [sec]	Total solution time [sec]
(2) MUMPS, *Nred*=75	3.2	77.9	110	220

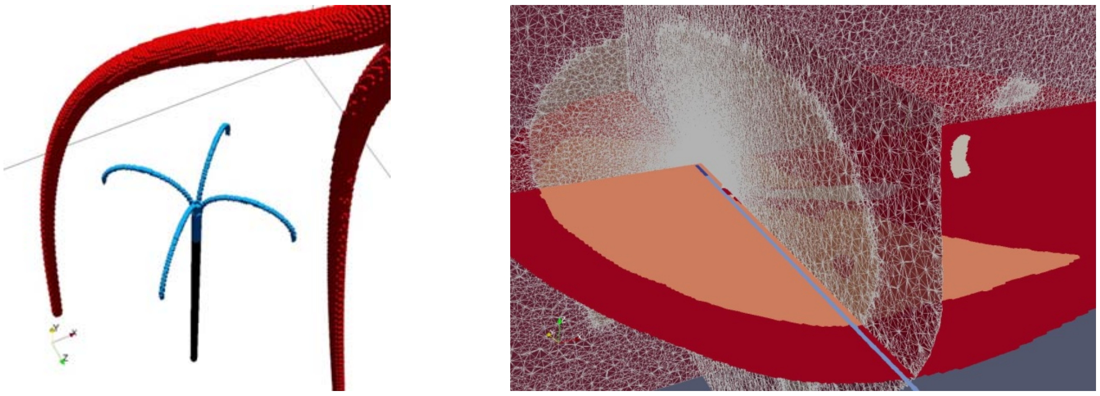
Table 14: Results of the best strategy (2) for the problem size 98,214,55 and the number of cores (subdomains) 5,012

4.16 Computer Modeling and Simulations In Strongly Heterogeneous Nonlinear Media

Supported by: S. Margenov, Y. Vutov, N. Kosturski, K. Georgiev (NCSA Bulgaria)

Whitepaper: S. Margenov, Y. Vutov, N. Kosturski, K. Georgiev, Academic publications [63] and "Computer Simulation of RF Liver Ablation" to be published in Proceedings of the American Institute of Physics (http://2012.eac4amitans.eu/resources/amitansabsbook1.pdf)

The focus of this project is the Finite Element Method (FEM) simulation of thermal and electrical fields in strongly heterogeneous nonlinear media on structured and unstructured meshes. New developed and tuned algorithms and codes for massively parallel platform like IBM BlueGene/P computer are integrated and tested. Mass and heat transfer and coupled electrical processes involved in the radio–frequency (RF) hepatic tumor ablation are considered. Note that RF ablation is a modern low invasive technique for efficient treatment of metastatic tumors, destroying the tumor cells by heating avoiding open surgery. The RF ablation procedure starts by placing the straight RF probe inside the tumor. The surgeon performs this under computer tomography (CT) or ultrasound guidance. Once the probe is in place, the electrodes are deployed and RF current is initiated. Both the surfaces areas of the uninsulated part of the trocar and the electrodes conduct RF current. An important part of the work is related to the construction/selection of efficient parallel preconditioners. Among others, the parallel implementation of aggressive coarsening *Algebraic Multigrid* (AMG) algorithms and adaptive time stepping are studied. MPI and MPI+OpenMP programming models are used in the developed algorithms. The work includes both, developing new algorithms and modifying some existing ones. The parallel algebraic multigrid implementation *BoomerAMG* is successfully used at the framework of the developed new composite iterative solvers in space.



a) RF probe **b) FEM mesh**

Figure 22: Inserted RF Probe and the corresponding FEM mesh

Tests of the basic FEM modules on IBM BlueGene/P supercomputer in Sofia (Bulgaria) are performed on both structured (voxel) and unstructured meshes. The results obtained during the developments of the computer models and their parallel implementations include: (*a*) creation of patient specific benchmark data based on a properly tuned segmentation of a high resolution 3D CT (Computer Tomography) medical image of the human body including the liver (see Figure 22 to see the inserted RF probe and the FEM mesh); (*b*) improved mathematical model of the heat transfer due to the capillary flow of the blood in the hepatic porous media.

Pad position	N	P	Timestep [sec.]	N_{it}	CPU in sec.
front	2 183 424	128	5.00	420	1545
front	17 467 392	1024	1.25	1128	5711
back	2 183 424	128	5.00	420	1584
back	17 467 392	1024	1.25	1136	5720
everywhere	2 183 424	128	5.00	420	1582
everywhere	17 467 392	1024	1.25	1153	5723

Table 15: Comparison on two meshes

Scalability analysis results are presented in Table 15 by increasing both the problem size (N) and the number of processors (P) by a factor of eight. It is well seen that when the time given for a timestep (see the fourth column in Table 15) decreases four times and therefore the number of timesteps increases four times, the run time increase is less than four (see the last column of Table 15). The total number of inner PCG iterations (N_{it}) is less than four times bigger for the larger problem. As a result, we solve 32 times bigger problem on eight times more CPUs less than four times slower, which shows good scalability of the method.

4.17 Cfd-Investigations For Assessing Aneurysm Rupture Risk For Individual Patient Using Ct Visual Diagnostics

Supported by: S. Markov, S. Radev, S. Margenov, G. Bencheva (NCSA Bulgaria)

3D Navier-Stokes equations for incompressible fluids are used for computer modeling of the blood flow in cerebral aneurisms. Realistic blood properties are incorporated in the model. Appropriate boundary conditions are imposed on the inflow, outflow and walls of the vessels. The computational domain is a patient specific blood vessel with two aneurisms close to each other. It is extracted using a proper setting of the software GIMIAS (Graphical Interface for Medical Image Analysis and Simulation) from a set of 225 images in DICOM format obtained after digital subtraction angiography (DSA) with application of contrast media. At the first step, appropriate transfer function is applied to the medical data saved in vtk format. The Visualization ToolKit (vtk) is an open source, freely available software system for 3D computer graphics, image processing, and visualization) in order to visualize the blood vessels and a parallelogram containing the aneurysm is cropped. Next, binary threshold segmentation is performed with respect to the voxel intensity. The threshold interval in the gray scale corresponds to blood presence is taken to be 195-280 (determined by the trial-error approach) and can be additionally tuned to fit more precisely the form of the blood vessels. Afterwards, the marching cubes algorithm and Taubin smoothing are applied to obtain the 3D geometry. At the third stage, the main surface is extracted and after additional smoothing of the domain, the data is stored in STL format. This file is then used in *Netgen* mesh generator to create a tetrahedral mesh. The output of *Netgen* is converted to the file format of the FEM (finite element method) software package *Elmer*. The initial mesh consists of 3500 nodes and 13000 elements (4500 elements being on the domain boundary). After using a procedure for uniform compression the three meshes presented in Table 16 are obtained:

Mesh	Nodes	Elements	Elements on the boundary
1	22 710	106 992	18 168
2	161 495	855 936	72 672
3	1 215 261	6 847 488	290 688

Table 16: The number of the nodes and the different type of elements use in the discretizations.

In the performed numerical tests, the preconditioned *BiCGStab* (Bi-conjugate gradient stabilized) algorithm with incomplete factorization is used. The currently obtained results for the three meshes show a good scalability for the stationary problem. The development of this subtask is based on the recent installation of *Elmer* on IBM BlueGene/P supercomputer in Sofia (Bulgaria). The output results about the CPU time in seconds for solving the stationary Naiver-Stockes equation form the runs using 1, 2, 4, 8, 16 and 32 processors are presented in Table 17.

Mesh	Number of processors					
	1	2	4	8	16	32
1	492.39	316.16	195.98	124.96	85.45	80.03
2		2 820.79	1 568.68	1 128.66	703.79	696.58
3					6 518.13	6 487.84

Table 17: CPU time for solving the Navier-Stokes equation on the IBM Blue Gene/P computer in Sofia

The reason for the empty boxes in Table 17 is that the memory of 2 GB per node of the IBM BlueGene/P computer is not enough for solving the problem on one processor over mesh 2 and on less than 16 processors over mesh 3. The results for runs on 16 processors show that when the size of the problem increases 7.1 times the CPU time for execution increase 8.7 times (mesh 1 to mesh 2) and that when the size of the problem increases 7.5 times the CPU time for execution increase 9.3 times (mesh 2 to mesh 3). The conclusion which can be made when 32 processors are used is that when the size of the problem increases 7.1 times the CPU time for execution increase 8.2 times (mesh 1 to mesh 2) and that when the size of the problem increases 7.5 times the CPU time for execution increase again 9.3 times (mesh 2 to mesh 3).

5 Summary and Conclusions

The main focus of WP12 is to perform research and development on four key areas for future multipetascale and exascale systems: Auto tuned and automatic techniques to be applied in parallel programming model runtimes, performance tools, file systems, and scalable numerical algorithms.

Task 12.2 (Scalable Numerical Algorithms), which covers many different algorithms, methods, and approaches along with simple ad-hoc programs demonstrating the scalability of the algorithms, is composed of 17 projects that evaluate different directions for improving scalability in algorithms often encountered in relevant numerical problems. The findings and the approaches proposed and implemented in some of these projects have the potential to be integrated into production level numerical applications.

The five key research areas that were under investigation in this task are:

- Reducing Synchronization Overhead in Iterative Solvers
- Enhancing Parallel Hybrid Sparse Solvers for Scalability
- Topology-Awareness
- Enabling Hybridization in Heterogeneous Architectures
- Application Scalability

The projects within the deliverable are grouped according to these research areas. The following paragraphs summarize the experiences gained from the different projects.

Two projects on "Reducing Synchronization Overhead in Iterative Solvers" try to reduce synchronization overheads of parallel sparse iterative solvers in petascale computing settings via two different approaches. One of them exploits asynchronous techniques that avoid blocking behavior of synchronization operations and by permitting processors to operate on whatever data they have, even if new data has not yet arrived from other processors. The other one proposes and implements a single-phase row-column parallel SpMxV scheme in order to address the two-phase communication bottleneck of the conventional row-column-parallel algorithm that utilize successful two-dimensional sparse matrix partitioning schemes.

Two projects on "Enhancing Parallel Hybrid Sparse Solvers for Scalability" investigate parallel hybrid sparse solvers that contain both direct and iterative components, which are promising in terms of robustness and scalability. Both projects propose, implement and investigate reordering sparse matrices into specific forms that enhances the scalability of the DDKrylov and *DS* factorization preconditioning schemes, respectively.

One project on "Topology-Awareness" tries to avoid message contention that can be observed in architectures consisting of thousands of processors (such as IBM Blue/Gene P) due to poor mapping of tasks to processors. For this purpose the project proposes and implements a two-phase framework for topology-aware task-to-processor mapping by considering both the task interaction and processor organization graphs.

Seven projects on "Enabling Hybridization in Heterogeneous Architectures" investigate the hybrid programming approach where MPI programming across computing nodes is combined with OpenMP-, threading-, CUDA- and OpenCL-based programming within individual nodes. The proposed schemes utilize the hybrid approach on enhancing the performance of several numerical applications such as block-cyclic reduction, FFT, general eigenvalue solver, stencil computations, direct linear system solver, molecular dynamic simulation, and Navier-Stokes solver.

Three projects on "Application Scalability" address the elimination of parallelization overheads encountered in specific numerical applications due to the desire for petascale-level parallelism. The first project investigates parallelizing sequential component that arises in the FETI method. The second and third projects mainly investigate utilization of appropriate parallel preconditioners for the solution of FEM simulation of thermal and electrical fields in strongly heterogeneous nonlinear media and parallel solution of 3D Navier-Stokes equations for incompressible fluids.