Supercomputing Frontiers and Innovations

2016, Vol. 3, No. 3

Scope

- Enabling technologies for high performance computing
- Future generation supercomputer architectures
- Extreme-scale concepts beyond conventional practices including exascale
- Parallel programming models, interfaces, languages, libraries, and tools
- Supercomputer applications and algorithms
- Distributed operating systems, kernels, supervisors, and virtualization for highly scalable computing
- Scalable runtime systems software
- Methods and means of supercomputer system management, administration, and monitoring
- Mass storage systems, protocols, and allocation
- Energy and power minimization for very large deployed computers
- Resilience, reliability, and fault tolerance for future generation highly parallel computing systems
- Parallel performance and correctness debugging
- Scientific visualization for massive data and computing both external and in situ
- Education in high performance computing and computational science

Editorial Board

Contents

Special Issue on Best ISC'16 Posters
J. Dongarra, Vl. Voevodin 4

Hybrid CPU + Xeon Phi implementation of the Particle-in-Cell method for plasma simulation
I. Meyerov, S. Bastrakov, I. Surmin, A. Bashinov, E. Efimenko, A. Korzhimanov, A. Muraviev, A. Gonoskov 5

Easy Access to HPC Resources through the Application GUI
M. van Waveren, A.S. El Nawasany, N. Hassanein, D. Moon, N. O'Byrnes, A. Cl'o, K. Murugan, A. Arena 11

Predicting I/O Performance in HPC Using Artificial Neural Networks
J.F. Schmid, J.M. Kunkel 19

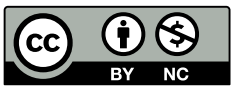
Analyzing Data Properties using Statistical Sampling - Illustrated on Scientific File Formats
J.M. Kunkel 34

Spectral Domain Decomposition Using Local Fourier Basis: Application to Ultrasound Simulation on a Cluster of GPUs
J. Jaros, F. Vaverka, B.E. Treeby 40

HCA aware Parallel Communication Library: A feasibility study for offloading MPI requirements
K. Kulkarni, Sh. Badhe, G. Gadre 56

Parallel Processing Model for Cholesky Decomposition Algorithm in AlgoWiki Project
A. Antonov, A. Frolov, H. Kobayashi, I. Konshin, A. Teplov, Vad. Voevodin, Vl. Voevodin 61

Data merging for the cultural heritage imaging based on Chebfun approach
M. El Afrit, Y. Le Du, R. Del Pino, G. Zhang 71



Special Issue on Best ISC'16 Posters

The International Journal Supercomputing Frontiers and Innovations announces the third issue of 2016 as a special issue devoted to the best posters presented at the ISC Research Poster and ISC HPC in Asia Poster sessions at the International Supercomputing Conference16. ISC is a premier European event for high performance computing, networking and storage which attracts leading researchers, engineers and scientists from universities, national laboratories, companies, scientific institutes and world-class research centers from all over the world.

Posters are an excellent opportunity to present latest research results, projects and innovations to a global HPC audience. Live discussion is a key point of the poster session which gives a lucky chance to find news partners, collaborators and coauthors, and at the same time this is an excellent opportunity to get immediate, informal and honest feedback on presented materials and results.

The both poster sessions were strong with a large number of very interesting presentations and discussions. This was the main reason why we decided to make an experiment and devote the entire issue of the Journal to the best posters presented at ISC this year. Summing up we can say that the experiment was successful and we hope that these special issues of the Journal will appear in the future.

Editors-in-Chief
Dongarra J.
Voevodin Vl.

DOI: 10.14529/jsfi160301

Hybrid CPU + Xeon Phi implementation of the Particle-in-Cell method for plasma simulation

Iosif B. Meyerov[1], *Sergey I. Bastrakov*[1], *Igor A. Surmin*[1], *Alexey V. Bashinov*[2], *Evgeny S. Efimenko*[12], *Artem V. Korzhimanov*[12], *Alexander A. Muraviev*[12], *Arkady A. Gonoskov*[123]

This paper presents experimental results of Particle-in-Cell plasma simulation on a hybrid system with CPUs and Intel Xeon Phi coprocessors. We consider simulation of two relevant laser-driven particle acceleration regimes using the Particle-in-Cell code PICADOR. On a node of a cluster with 2 CPUs and 2 Xeon Phi coprocessors the hybrid CPU + Xeon Phi configuration allows to fully utilize the computational resources of the node. It outperforms both CPU-only and Xeon Phi-only configurations with the speedups between 1.36 x and 1.68 x.

Keywords: hybrid computing, Xeon Phi, Particle-in-Cell.

Introduction

Numerical simulation of plasmas is an important area of computational physics with numerous applications. The Particle-in-Cell method [1–3] is widely used for plasma simulation in ultrahigh fields. Large-scale 3D simulation requires the use of supercomputers. Currently, there are a number of Particle-in-Cell implementations capable of that, including OSIRIS [4], PIConGPU [5], VPIC [6], VLPL [7], WARP [8].

During the recent years there has been a continuous progress in utilization of accelerators, including GPUs [5, 9, 10] and Intel Xeon Phi coprocessors [11, 12]. Our code PICADOR has been previously ported and optimized for Xeon Phi coprocessors that led to 1.6–1.8 x speedup compared to an 8-core CPU on a benchmark problem [13]. Naturally, it is more challenging to efficiently utilize Xeon Phi on real applications that may have large output, significant imbalance, costly diagnostics, and other performance hindering factors. However, even obtaining the same performance on Xeon Phi as on a multicore CPU is beneficial in terms of hybrid CPU + Xeon Phi computing. It allows to fully utilize computational resources of heterogeneous machines with several CPUs and Xeon Phi coprocessors per node and theoretically may significantly outperform CPU-only and Xeon Phi-only configurations.

This paper presents our experience of solving two problems by Particle-in-Cell simulation using PICADOR on a hybrid CPU + Xeon Phi machine. These two problems have been selected as they are relevant for laser-plasma physics and are part of our current research involving PICADOR as a tool for numerical simulation. Thus, the problems considered present a realistic workload and, secondly, any speedup obtained due to Xeon Phi or hybrid CPU + Xeon Phi computing is beneficial for the current research done using the code. The paper is organized as follows. We briefly describe the method and our code PICADOR in section 1. Sections 2 and 3 are devoted to the physical problems we consider. Summary and conclusions are given in section 3.

[1]Lobachevsky State University of Nizhni Novgorod, Nizhni Novgorod, Russia
[2]Institute of Applied Physics, Russan Academy of Sciences, Nizhni Novgorod, Russia
[3]Chalmers University of Technology, Gteborg, Germany

1. Particle-in-Cell code PICADOR

1.1. Particle-in-Cell method

In this subsection we briefly describe the Particle-in-Cell method, a detailed description is given in [3].

The Particle-in-Cell method operates on field data and particle data. Values of electric and magnetic fields are defined on a spatial grid. A plasma is represented as an ensemble of particles, each with a charge, mass, position and momentum. Each particle used in simulation is in fact a macroparticle that represents a cloud of real particles. A particle form factor defines particle distribution inside the cloud. In this paper we use the cloud-in-cell form factor that corresponds to the uniform distribution in the cloud that has the same size as a cell of the spatial grid. A notable feature of the method is that particles do not interact with each other directly, instead each particle interacts with a set of nearby grid values, depending on the form factor.

The basic computational loop of the Particle-in-Cell method consists of the following stages. For each particle the Lorenz force is computed using interpolated values of the electromagnetic field and the particle momentum and position are updated. Grid values of the current created by particle movement are computed. Field interpolation and current deposition depend on the particle form factor being used. Finally, field values are updated by solving Maxwell's equations.

1.2. PICADOR code

PICADOR [13, 14] is a tool for 3D plasma simulation based on the Particle-in-Cell method. The code supports a rather standard set of numerical schemes for laser-plasma simulation: several widely used particle form factors, Boris particle pusher [15], Yee grid [16] and finite-difference time-domain field solver [17], charge-conserving current deposition [18, 19].

Parallel computing on cluster systems is organized using MPI. On the internode level we use a 3D rectilinear domain decomposition, with each MPI process handling a part of the simulation area. Each MPI process stores particles and grid values in the corresponding sub-area with guard cells. Data exchanges occurs only between neighbour sub-areas. On shared memory OpenMP is used to parallelize loops over particles and cells. Processing of particles in a cell is partially vectorized using a combination of compiler auto-vectorization and manually coded intrinsic functions, details are given in [13]. The basic computational scheme is given in fig. 1.

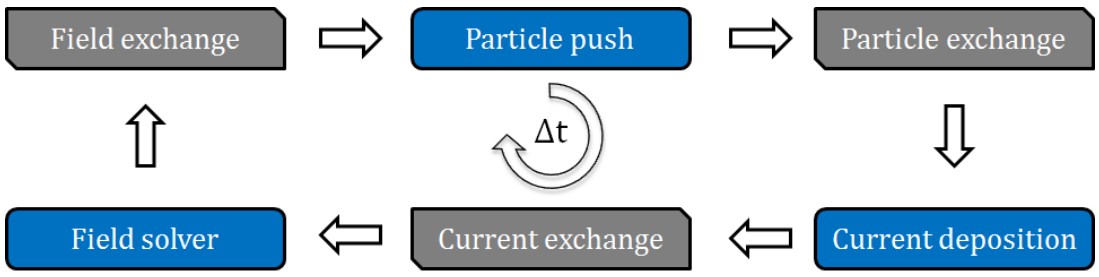


Figure 1. Computational scheme of the Particle-in-Cell method

PICADOR has been previously ported and optimized for Intel Xeon Phi coprocessors [13]. Applying standard optimization techniques such as improving memory locality, enhancing scaling efficiency and vectorization resulted in 1.6–1.8 x speedup on Xeon Phi 5110P relative to an 8-core Intel Xeon E5-2660 CPU on a benchmark problem with a uniform distribution of particles between threads and no MPI communication.

2. Hybrid simulation of wakefield laser pulse self-compression

In this section we consider simulation of a laser pulse self-compression due to wakefield excitation in the relativistic regime. A gas jet is irradiated by a femtosecond high intensity laser pulse. In the result of the interaction a generated wakefield may lead to laser pulse self-compression. This is a promising mechanism for generating ultra-short high-intensity laser pulses [20]. The simulation was performed using the following parameters: $256 \times 128 \times 128$ grid, 78.5 million particles, 7 674 time steps, cloud-in-cell particle form factor, charge-conserving Villasenor-Buneman current deposition scheme [18], double precision floating point arithmetic.

Computational experiments were performed on a node of the MVS-10P supercomputer at the Joint Supercomputer Center of RAS with 2 x Intel Xeon E5-2690 CPU (8 cores, 2.9 GHz, Hyperthreading enabled) and 2 x Intel Xeon Phi 7110 (61 cores, 1.053 GHz), Intel C++ Compiler 14.0.1, Intel MPI 4.1. We used a single MPI process per CPU and Xeon Phi, 2 OpenMP threads per core on CPU and 4 threads per core on Xeon Phi. Our previous results [13] show that this configuration of processes and threads is the best for PICADOR. This is probably due to the fact that during the most computationally intensive stages of the Particle-in-Cell method there are lots of independent subproblems that can be solved in parallel, even with 4 threads per core on Xeon Phi. Meanwhile, running several MPI processes on shared memory requires some additional communication, which could hinder overall performance. Run time on a node of MVS-10P in three configurations: CPU-only, Xeon Phi-only, and hybrid CPU + Xeon Phi is given in tab. 1.

Table 1. Run time of simulation of wakefield laser pulse self-compression on CPU, Xeon Phi, and CPU + Xeon Phi. Time is given in seconds. For the computational core a split into current deposition, particle push, and other operations is given

Stage	2 x CPU	2 x Xeon Phi	2 x CPU + 2 x Xeon Phi
Computational core overall	693.2	681.3	351.2
particle push	408.9	341.6	201.7
current deposition	253.5	314.4	129.6
other	30.8	25.3	19.8
MPI communication	21.1	148.5	144.2
Overall	714.3	829.8	495.3

Run time of the computational core on Xeon Phi is close to that of the CPU. There is some advantage in the particle push stage which is partially vectorized on Xeon Phi, but it is compensated by the non-vectorized current deposition stage. MPI communication on Xeon Phi is slower because some related operations are sequential and single-core performance of the coprosessor is inferior to the CPU. Overall, Xeon Phi is 1.16 x slower compared to the CPU. Nevertheless, utilizing the coprocessors allows to accelerate the simulation compared to the CPU by means of the hybrid CPU + Xeon Phi configuration. It yields 1.44 x and 1.68 x speedup compared to the CPU-only and Xeon Phi-only configurations, respectively.

3. Hybrid simulation of target normal sheath acceleration

In this section we consider a laser ion acceleration in the target normal sheath acceleration regime, which is a widely used approach to laser-driven particle acceleration [21]. A high-intensity

laser pulse heats electrons at the front surface of a target forming a sheath at its rear side. Ions of the target are accelerated from the rear side by the electron sheath. Surface grating is used on the irradiated side of the target to increase efficiency [22]. The hardware and numerical schemes were the same as in the previous section. Run time on a node of MVS-10P in CPU-only, Xeon Phi-only and CPU + Xeon Phi configurations is given in tab. 2.

Table 2. Run time of simulation of target normal sheath acceleration on CPU, Xeon Phi, and CPU + Xeon Phi. Time is given in seconds. For the computational core a split into current deposition, particle push, and other operations is given

Stage	2 x CPU	2 x Xeon Phi	2 x CPU + 2 x Xeon Phi
Computational core overall	3 012.6	2 137.0	1 687.3
particle push	1 888.8	1 214.7	1 004.3
current deposition	974.0	839.7	605.3
other	149.8	82.6	77.7
MPI communication	292.9	1 637.3	748.8
Overall	3 305.5	3 774.3	2 436.2

The simulation area was divided between MPI processes along the direction of the laser pulse to reduce the number of particles going from one process to another. Nevertheless, due to the complex dynamics occurring throughout the simulation, there is a massive migration of particles between processes. Therefore, a large amount of time is spent on MPI communication: 8.8% of the total run time on the CPU and 43.4% on the coprocessor. Due to this fact Xeon Phi is 1.14 x slower compared to the CPU, although the computational core is 1.41 x faster. The issue of the slow particle migration is partially alleviated in the hybrid configuration because of the lower amount of particles per process. The CPU + Xeon Phi combination outperforms the CPU-only and Xeon Phi-only runs by 1.36 x and 1.55 x, respectively.

Conclusions

This paper presents results of simulation of two problems on a node of the MVS-10P cluster using three configurations: 2 x CPUs, 2 x Xeon Phi coprocessors, 2 x CPUs + 2 x Xeon Phi. Both problems considered are not ideally suited for Xeon Phi, with the CPU slightly outperforming Xeon Phi by factors of 1.16 x and 1.14 x because of MPI communication time. Even in this case, it is possible to use the Xeon Phi coprocessors to accelerate simulation by fully utilizing the computational resources of a node in the hybrid CPU + Xeon Phi mode. The speedup of the hybrid configuration compared to the CPU-only and Xeon Phi-only modes is between 1.36 x and 1.68 x. Further progress can be made by reducing MPI communication time, particularly on Xeon Phi. A possible approach is to first process near-boundary particles and then overlap asynchronous particle transfers and processing the remaining particles. Our very first implementation of this idea shows some speedup of data transfers on Xeon Phi for the problem considered in section 2: 1.46 x on 2 x Xeon Phi and 1.35 x in the hybrid mode. However, a tradeoff is some increase in the particle push time, so that the overall speedup is 1.05 x. Further improvement of data transfers is a direction of future work.

The problems considered in this paper have a sufficiently uniform distribution of particles in the simulation area. Thus, they are rather suitable for the manycore architecture of Xeon Phi

with the standard OpenMP thread scheduling policies. However, there are important regimes involving electron-positron cascades [23, 24] with a small number of cells having more particles than the rest of the simulation area. Our first experiments show that in this case the utilization of standard approach results in significant thread workload imbalance on Xeon Phi that causes underwhelming performance. A parallelization scheme for such problems is under development.

The recently introduced Intel Xeon Phi products of the KNL generation look very promising due to a significant increase in the overall performance as well as in the single-core performance. Optimization of the code for KNL is a direction of future work.

This study was supported by the RFBR, project No. 15-37-21015.

References

1. Hockney R.W., Eastwood J.W. Computer simulation using particles. New York: McGraw-Hill. 1981.

2. Dawson J.M. Particle simulation of plasmas, Reviews of Modern Physics. 1983. 55 (2). 403–447. DOI: 10.1103/RevModPhys.55.403.

3. Birdsal C.K., Langdon A.B. Plasma physics via computer simulation. CRC Press. 2004.

4. Fonseca R.A., Vieira J., Fiuza F., Davidson A., Tsung F.S., Mori W.B., Silva L.O. Exploiting multi-scale parallelism for large scale numerical modelling of laser wakefield accelerators. Plasma Phys. Control. Fusion. 2013. 55 (12). 124011.

5. Burau H., Widera R., Honig W., Juckeland G., Debus A., Kluge T., Schramm U., Cowan T.E., Sauerbrey R., Bussmann M. PIConGPU: A Fully Relativistic Particle-in-Cell Code for a GPU Cluster. IEEE Transactions on Plasma Science. 2010. 38 (10). 2831–2839.

6. Bowers K.J., Albright B.J., Yin L., Bergen B., Kwan T.J.T. Ultrahigh performance three-dimensional electromagnetic relativistic kinetic plasma simulation. Physics of Plasmas. 2008. 15 (5). 055703.

7. Pukhov A. Three-dimensional electromagnetic relativistic particle-in-cell code VLPL (Virtual Laser Plasma Lab). Journal of Plasma Physics. 1999. 61 (3). 425–433.

8. Vay J.-L., Bruhwiler D.L., Geddes C.G.R., Fawley W.M., Martins S.F., Cary J.R., Cormier-Michel E., Cowan B., Fonseca R.A., Furman M.A., Lu W., Mori W.B., Silva L.O. Simulating relativistic beam and plasma systems using an optimal boosted frame. Journal of Physics: Conference Series. 2009. 180 (1). 012006.

9. Kong X., Huang M.C., Ren C., Decyk V.K. Particle-in-cell simulations with charge-conserving current deposition on graphic processing units. Journal of Computational Physics. 2011. 230 (4). 1676–1685. DOI: 10.1016/j.jcp.2010.11.032.

10. Decyk V.K., Singh T.V. Particle-in-Cell algorithms for emerging computer architectures. Computer Physics Communications. 2014. 185 (3). 708–719. DOI: 10.1016/j.cpc.2013.10.013.

11. Nakashima H. Manycore challenge in particle-in-cell simulation: how to exploit 1 TFlops peak performance for simulation codes with irregular computation. Computers and Electrical Engineering. 2015. 46. 81–94. DOI: 10.1016/j.compeleceng.2015.03.010.

12. Glinsky B.M., Kulikov I.M., Snytnikov A.V., Romanenko A.A., Chernykh I.G., Vshivkov V.A. Co-design of parallel numerical methods for plasma physics and astrophysics. Supercomputing Frontiers and Innovations. 2014. 1 (3). 88–98.

13. Surmin I.A., Bastrakov S.I., Efimenko E.S., Gonoskov A.A., Korzhimanov A.V., Meyerov I.B. Particle-in-Cell laser-plasma simulation on Xeon Phi coprocessors. Computer Physics Communications. 2016. 202. 204–210. DOI: 10.1016/j.cpc.2016.02.004.

14. Bastrakov S., Donchenko R., Gonoskov A., Efimenko E., Malyshev A., Meyerov I., Surmin I. Particle-in-cell plasma simulation on heterogeneous cluster systems. Journal of Computational Science. 2012. 3. 474–479.

15. Boris J.P. Relativistic plasma simulation-optimization of a hybrid code. Proceedings of the 4th Conference on Numerical Simulation of Plasmas. 1970. 3–67.

16. Yee K. Numerical solution of initial boundary value problems involving Maxwell's equations in isotropic media. IEEE Transactions on Antennas and Propagation. 1966. 14 (3). 302–307. DOI: 10.1109/TAP.1966.1138693.

17. Taflove A. Computational electrodynamics: the finite-difference time-domain method. London: Artech House. 1995.

18. Villasenor J., Buneman O. Rigorous charge conservation for local electromagnetic field solvers. Computer Physics Communications. 1992. 69. 306–316. DOI: 10.1016/0010-4655(92)90169-Y.

19. Esirkepov T.Zh. Exact charge conservation scheme for Particle-in-Cell simulation with an arbitrary form-factor. Computer Physics Communications. 2011. 135. 144–153. DOI: 10.1016/S0010-4655(00)00228-9.

20. Pipahl A., Anashkina E.A., Toncian M., Toncian T., Skobelev S.A., Bashinov A.V., Gonoskov A.A., Willi O., Kim A.V. High-intensity few-cycle laser-pulse generation by the plasma-wakefield self-compression effect. Physical Review E. 2013. 87. 033104.

21. Wilks S.C., Langdon A.B., Cowan T.E., Roth M., Singh M., Hatchett S., Key M.H., Pennington D., MacKinnon A., Snavely R.A. Energetic proton generation in ultra-intense lasersolid interactions. Physics of Plasmas. 2001. 8 (2). 542–549.

22. Bychenkov V.Y., Brantov A.V., Govras E.A., Kovalev V.F. Laser acceleration of ions: recent results and prospects for applications. Physics-Uspekhi. 2015. 58 (1). 71–81.

23. Bell A.R., Kirk J.G. Possibility of Prolific Pair Production with High-Power Lasers. Physical Review Letters. 2008. 101 (20). 200403. DOI: 10.1103/PhysRevLett.101.200403.

24. Gonoskov A., Bastrakov S., Efimenko E., Ilderton A., Marklund M., Meyerov I., Muraviev A., Sergeev A., Surmin I., Wallin E. Extended Particle-in-Cell Schemes for Physics in Ultrastrong Laser Fields: Review and Developments. Physical Review E. 2015. 92 (2). 023305. DOI: 10.1103/PhysRevE.92.039903.

DOI: 10.14529/jsfi160302

Easy Access to HPC Resources through the Application GUI

Matthijs van Waveren[1]***, Ahmed Seif El Nawasany***[1]***, Nasr Hassanein***[1]***, David Moon***[1]***, Niall O'Byrnes***[1]***, Alain Clò***[1]***, Karthikeyan Murugan***[1]***, Antonio Arena***[1]

The computing environment at the King Abdullah University of Science and Technology (KAUST) is growing in size and complexity. KAUST hosts the tenth fastest supercomputer in the world (Shaheen II) and several HPC clusters. Researchers can be inhibited by the complexity, as they need to learn new languages and execute many tasks in order to access the HPC clusters and the supercomputers. In order to simplify the access, we have developed an interface between the applications and the clusters, that automates the transfer of input data and job submission to the clusters and also the retrieval of results to the researchers local workstation. The innovation is that the user now submits his jobs to the cluster from within the application GUI on his workstation, and does not have to directly log into the cluster anymore. This article details the solution and its benefits to the researchers.

Keywords: KAUST, Remote Job Submission, Middleware, MATLAB, VASP, MedeA, ADF.

Introduction

The computing landscape of the King Abdullah University of Science and Technology (KAUST) is increasing in complexity. Researchers have access to the tenth fastest supercomputer in the world (Shaheen II [14]) and several HPC clusters. They work on local Windows, Mac, or Linux workstations. In order to facilitate the access of the HPC systems, we have developed interfaces for several research applications that automate input data transfer, job submission and retrieval of results. The user now submits his jobs to the cluster from within the application GUI on his workstation, and does not have to physically go onto the cluster anymore. This leads to reduced time-to-solution, easier access to the HPC systems, and less time spent on mastering unfamiliar Linux commands.

Remote job submission mechanisms were initially developed in the context of Grid Computing. We cite several frameworks that support remote job submission. A framework covers the user workstations where the jobs are created and the remote resources used to execute the jobs. The Globus Toolkit [4] was one of the first frameworks to support the development of service-oriented distributed computing applications and infrastructures with a remote job submission mechanism. The Uniform Interface to Computing Resources (UNICORE) [1] is a framework originating from Europe for the development of improved and more uniform interfaces to High Performance Computing and data resources. The HPC Gateway, originally called SynfiniWay [5], and before that TAO-W, is a virtualized IT framework developed by Fujitsu that provides a uniform and global view of resources within a department, a company, or a company with its suppliers. It also includes the linking of jobs in a workflow. GRIDBLAST [7] is a framework used to execute Life Science Basic Local Alignment Search Tool (BLAST) searches on a grid consisting of a server and remote worker nodes. eQUEUE [13] is a web-based job submission to run jobs on a cluster. Prajapati and Shah [10] made an experimental study of remote

[1]IT Research Computing, King Abdullah University of Science and Technology, Thuwal 23955-6900, Saudi Arabia; email: Matthijs.vanWaveren, Ahmed.SeifElNawasany, Nasr.Hassanein, David.Moon, Niall.OByrnes, Alain.Clo, Karthikeyan.Murugan, Antonio.Arena@kaust.edu.sa

job submission and execution on local resource managers through grid computing mechanisms. Bright Manager [3], software for deployment, monitoring and managing of HPC clusters, and HPCSpot [9], a solution for HPC on demand, both are related to remote job submission.

More recently, application software vendors have been implementing remote job submission into the application software products themselves. The novelty of the present work is that we configure remote job submission by using the vendor-supplied implementation in the applications. This allows us to configure the access to HPC resources in such a way, that the user can stay in the application GUI.

1. KAUST Computing Ecosystem

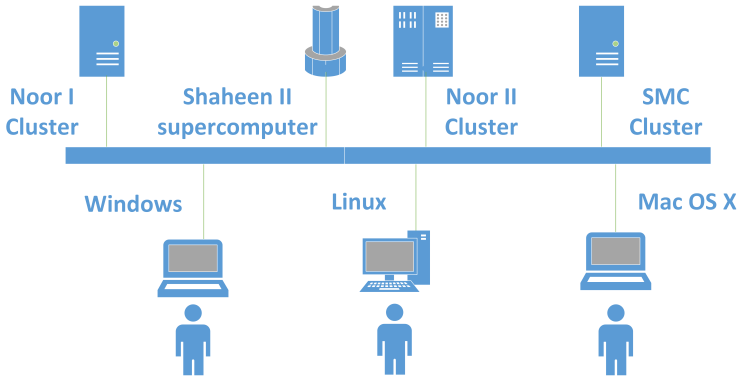


Figure 1. KAUST Computing Ecosystem

Researchers typically start out running their computational jobs on either a Windows, Linux or Mac OS X workstation.

If they need more performance, they have at their disposal the supercomputer Shaheen II and several HPC clusters, called Noor I, Noor II, and SMC, as shown in Fig. 1.

Shaheen II is a 36-cabinet Cray XC40 supercomputer with DataWarp technology for accelerating I/O. It has 192568 processors cores, 5.536 Pflops of sustained LINPACK performance, and 7.2 Pflops of theoretical peak performance. It is number 10 in the Top 500 list of June 2016 [14].

Noor II is a Linux cluster based on the quad-socket AMD Opteron 6376 processor. It provides a total capacity of 160 compute nodes or 10256 cores for a total of 105 TFlops of peak floating point performance. Each node can run 64 threads. It has four sockets of 8-core CPUs, with 2 threads per core, equipped with a high-performance, low latency Infiniband interconnect.

SMC[2] is a Linux cluster based on the Intel Xeon (Sandy Bridge) processor. It provides a capacity of 309 nodes, or 5940 cores for a total of 246 TFlops of peak floating point performance. 108 nodes have two sockets of 8 cores, 177 nodes have two sockets of 10 cores and 24 nodes have two sockets of 14 codes. They are equipped with a high-performance, low latency Infiniband interconnect.

Noor I is a Linux cluster based on the Intel Xeon (Nehalem) processor. Each node has two sockets of 4 cores equipped with a high-performance, low latency Infiniband interconnect.

The workload of all these systems is managed by the Simple Linux Utility for Resource Management (SLURM [2]), which is an open source cluster management and job scheduling system.

[2]SMC = Shared Mini Cluster

Researchers submit jobs to SLURM, and SLURM then allocates resources of the corresponding system to the jobs of the researchers.

2. User Barriers

There are barriers for researchers at KAUST to use the available HPC clusters and a supercomputer. In the course of migration from workstation to cluster or supercomputer, the researchers need to learn a whole new language: Linux commands, SLURM scheduler commands, and data transfer commands. This is especially the case if they come from a Windows world. It slows them down, and may even inhibit them from effectively utilizing the available HPC resources. Researchers need to login to the cluster, transfer input data to the cluster, submit a batch job, wait for it to finish, then transfer results back to the workstation. If they divide their computational work over different clusters, they need to do this sequence of tasks for each cluster separately, and combine the results from each cluster on their workstation, as shown in Fig. 2. This work is tedious, adds no value, and wastes valuable time of the researchers. With this work, we want to give the researchers a way to reduce the time spent on migrating from workstation to HPC resources.

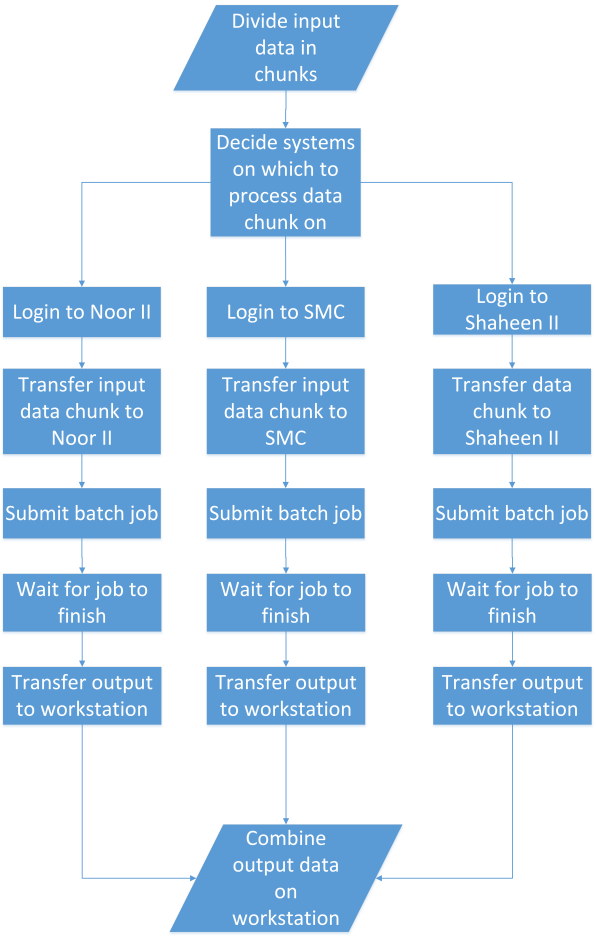


Figure 2. Initial user workflow

3. Solution

We automated the sequences of tasks shown in Fig. 2 in order to reduce the time spent by researchers on executing these tasks. This automation is attained by implementing an interface between the research application running on the workstation (Linux, Windows or Mac) and the clusters. This interface automatically executes the opening of a session on the cluster, the transfer of input data, the job submission, and the retrieval of results. We call these interfaces HPC Add-ons, as they add an HPC capability to the research application running on the workstation. We have implemented these HPC Add-ons for the following research applications - MATLAB, ADF, and VASP. The innovation is that the user now submits his jobs to the cluster from within the application GUI on his workstation, and does not have to directly log into the cluster anymore. The resulting optimized workflow is shown in Fig. 3. The method of implementation of the HPC Add-ons is not standardized and differs for each application.

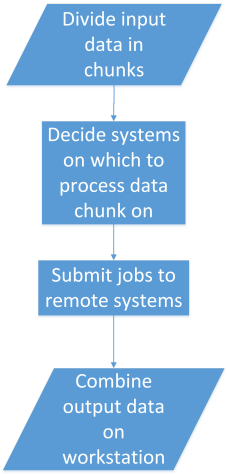


Figure 3. Optimized user workflow

3.1. MATLAB

MATLAB [6] is a high-level language for scientific and engineering computing. The MATLAB HPC Add-on architecture is illustrated in Fig. 4. The HPC Add-on interface is implemented in MATLAB code, and is installed on the client side. This interface code takes care of opening a session to the remote resource, creating and sending a job submission script to the SLURM scheduler [2] on the remote resource, and monitoring the status of the job. The job is executed using the MATLAB Distributed Computing Server, which lets users run MATLAB jobs in parallel on HPC resources. The output directory is mirrored on the local workstation, so the researcher always has access to the latest results.

One of the KAUST researchers developed an algorithm for measuring single-molecule diffusion using MATLAB [12]. With a single run requiring about 200,000 Gaussian fittings, he soon found out that running on a single processor took too long to be practical. To shorten processing times he used the MATLAB Parallel Computing Toolbox to perform the computations on a workstation with multiple cores. Using four cores experiments took about three hours, and with 16 cores, just 45 to 50 minutes.

He often needs to run many simulations and experiments to obtain valid statistical results. To further accelerate the process he began running his jobs on 512 cores at a time on the HPC

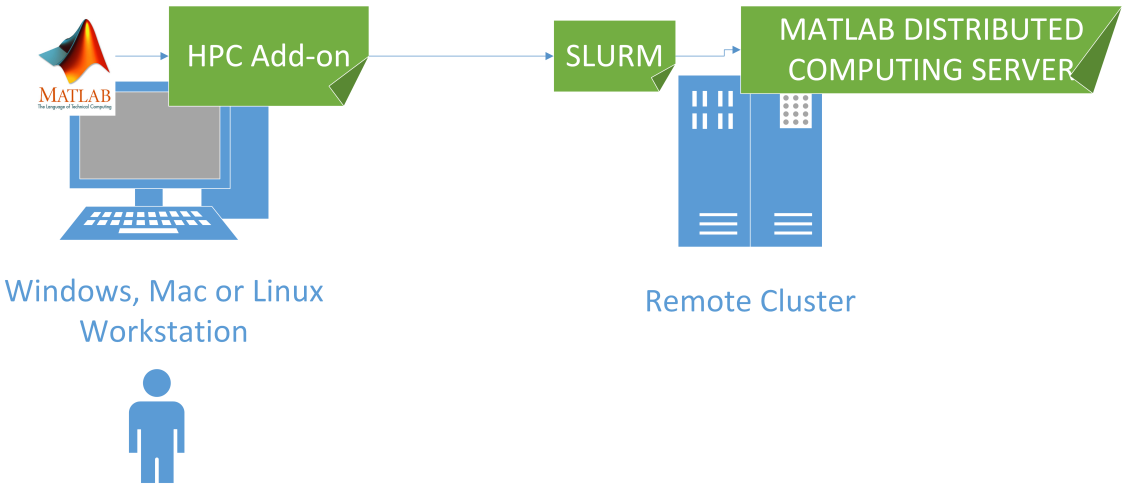


Figure 4. MATLAB HPC Add-on Architecture

clusters with the HPC Add-on and MATLAB Distributed Computing Server. Using this setup he can complete a set of experiments that took 24 hours on a multicore machine in just 15 minutes.

3.2. VASP

VASP [8] is a complex package for performing ab-initio quantum-mechanical Molecular Dynamics simulations using pseudopotentials or using the projector-augmented wave method and a plane wave basis set. The VASP HPC Add-on architecture is illustrated in Fig. 5. The interface between VASP and the remote resources is implemented by a software product called MedeA [11].

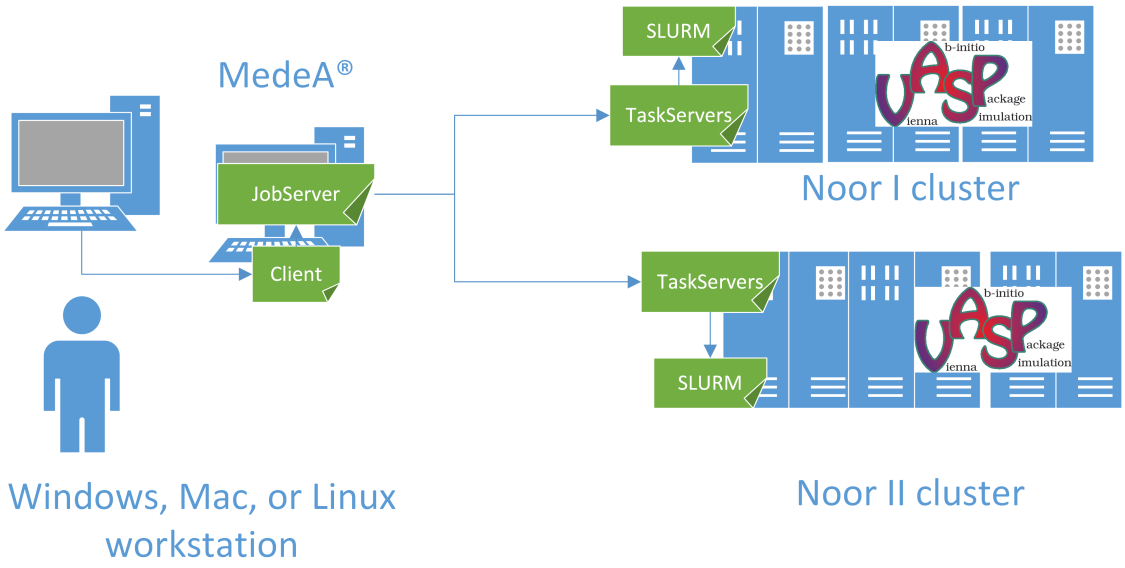


Figure 5. VASP HPC Add-on Architecture

MedeA has a tiered architecture, consisting of the MedeA graphical user interface (main tier), the JobServer (middle tier) and the TaskServer (end tier). The JobServer is the central hub for computational job control, job preprocessing, post processing. The TaskServer takes

care of the communication with the SLURM scheduler on the remote cluster. There are several clusters at KAUST, and we only show the Noor I and the Noor II clusters in the figure.

3.3. ADF

ADF [15] is a molecular density-functional theory code used in many areas of chemistry and materials science. ADF is particularly strong in molecular properties and inorganic chemistry. The ADF HPC Add-on architecture is illustrated in Fig. 6.

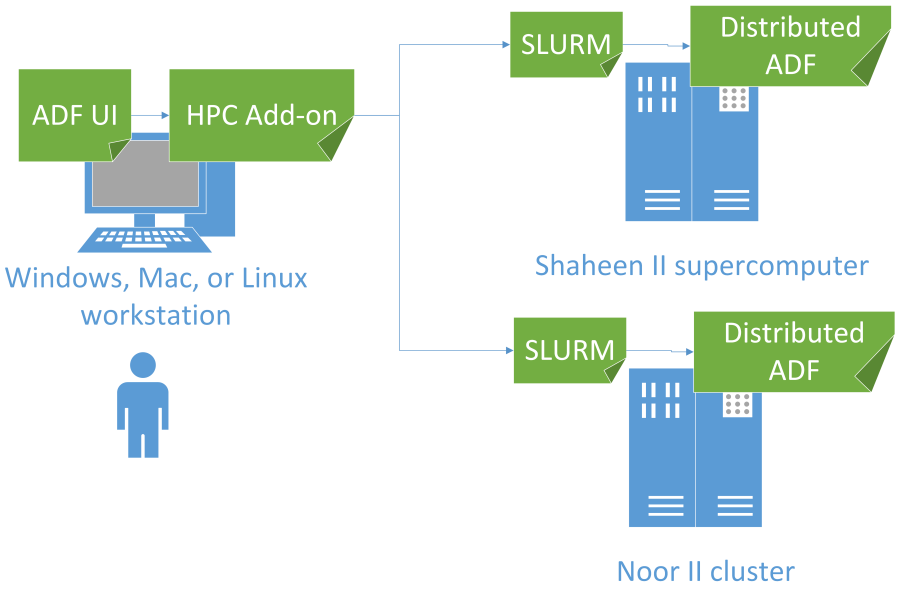


Figure 6. ADF HPC Add-on Architecture

The GUI is written in Tcl/Tk, and it reads a configuration file with details of the remote system. The GUI then opens a session via ssh to the remote system, and submits a job to the SLURM scheduler on the remote system. It can check the status of jobs on the remote system, and can kill the job. Transfers of files to and from the remote system is handled by the GUI automatically (via ssh).

Conclusions

We show that we can simplify the workflow of a user of HPC resources considerably by automating the workflow sequence: logging into the HPC system, job submission and transfer of input and output data. This has been implemented in the user interface of the applications directly, allowing the user to stay in this user interface. The applications we have discussed are MATLAB, VASP and ADF.

Benefits for the researchers include reduced time-to-solution, easier access to the HPC systems, and less time spent on mastering unfamiliar Linux commands. One of the MATLAB researchers used to need 24h wall clock time to execute runs to measure single-molecule diffusion on his workstation, but now has his results in 15 minutes by using the HPC Add-on.

Our future work includes making the code of our HPC Add-ons available as open source, and implementing HPC Add-ons for other applications that have built-in support for remote job submission, for example MaterialsStudio of Dassault Systmes. We plan also to implement an

HPC Gateway for automating the workflow sequence for applications that do not have built-in support for remote job submission.

We acknowledge support given by the KAUST Supercomputer Laboratory (KSL), The MathWorks Inc, Scientific Computing and Modelling NV and Materials Design Inc. We acknowledge the thorough review of Dr Samar Aseeri of KSL.

References

1. Jim Almond and Dave Snelling. UNICORE: uniform access to supercomputing as an element of electronic commerce. *Future Generation Computer Systems*, 15(5):539–548, 1999. http://0-www.sciencedirect.com.libsys.kaust.edu.sa/science/article/pii/S0167739X99000072.
2. D. Auble, B. Christiansen, M. Jette, A. Sanchez, and T. Wickberg. Slurm workload manager. http://slurm.schedmd.com/slurm.html.
3. Martijn de Vries. Bright Cluster Manager for HPC: advanced cluster management made easy. http://www.brightcomputing.com/product-offerings/bright-cluster-manager-for-hpc.
4. Ian Foster. Globus Toolkit Version 4: Software for Service-Oriented Systems. *Journal of Computer Science and Technology*, 21(4):513–520, 2006.
5. Roger Henri, Pierre Lagier, and Didier Plaindoux. FSE grid middleware: Collaborative grid environment for distributed computing. *Fujitsu Scientific and Technical Journal*, 40(2):269–281, 2004.
6. Yuliya V. Karpievitch and Jonas S. Almeida. mGrid: A load-balanced distributed computing environment for the remote execution of the user-defined MATLAB code. *BMC Bioinformatics*, 7(1):139–139, 2006.
7. Fumikazu Konishi and Akihiko Konagaya. The Architectural Design of High-Throughput BLAST Services on OBIGrid. In Akihiko Konagaya and Kenji Satou, editors, *Grid Computing in Life Science: First International Workshop on Life Science Grid, LSGRID 2004, Kanazawa, Japan, May 31-June 1, 2004, Revised Selected and Invited Papers*, pages 32–42, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. http://www.springer.com/us/book/9783540252085.
8. G. Kresse and J. Furthmüller. Efficient iterative schemes for ab initio total-energy calculations using a plane-wave basis set. *PHYSICAL REVIEW B*, 54(16):11169–11186, 1996. DOI: 10.1103/PhysRevB.54.11169.
9. Alexandre Morel. HPCSpot for HPC on demand. http://www.oxalya.com/?p=2173.
10. Harshadkumar B. Prajapati and Vipul A. Shah. Experimental study of remote job submission and execution on LRM through grid computing mechanisms. In IEEE, editor, *Fourth International Conference on Advanced Computing & Communication Technologies*, 2014.
11. X. Rozanska, P. Ungerer, B. Leblanc, P. Saxe, and E. Wimmer. Automatic and systematic atomistic simulations in the MedeA (r) software en-

vironment: Application to EU-REACH. *OIL & GAS SCIENCE AND TECHNOLOGY-REVUE IFP ENERGIES NOUVELLES*, 70(3):405–417, 2015;2014;. http://ogst.ifpenergiesnouvelles.fr/articles/ogst/abs/2015/03/ogst140090/ogst140090.html. DOI: 10.1103/PhysRevB.54.11169

12. Maged F. Serag. Accelerating Development of a New Single-Molecule Localization and Tracking Technique. *MathWorks Newsletter*, 2016. http://www.mathworks.com/company/newsletters/articles/accelerating-development-of-a-new-single-molecule-localization-and-tracking-technique.html.
13. Wade Sisson. eQUEUE Job Submission Tool. http://y2u.be/0IcTq4BMYwQ.
14. E. Strohmaier, J. Dongarra, H. Simon, and M. Meuer. Top 10 sites for June 2016. https://www.top500.org/lists/2016/06/.
15. G. te Velde, F. M. Bickelhaupt, E. J. Baerends, C. Fonseca Guerra, S. J. A van Gisbergen, J. G. Snijders, and T. Ziegler. Chemistry with ADF. *Journal of Computational Chemistry*, 22(9):931–967, 2001.

DOI: 10.14529/jsfi160303

Predicting I/O Performance in HPC Using Artificial Neural Networks

Jan Fabian Schmid[1], *Julian M. Kunkel*[2]

The prediction of file access times is an important part for the modeling of supercomputer's storage systems. These models can be used to develop analysis tools which support the users at integrating efficient I/O behavior.

In this paper, we analyze and predict the access times of a Lustre file system from the client perspective. For this purpose, we measured file access times in various test series and develop different models for predicting access times. The evaluation shows that in models utilizing artificial neural networks the average prediciton error is about 30% smaller than in linear models. One phenomenon in the distribution of file access times is of particular interest: File accesses with identical parameters show several typical access times.

The typical access times usually differ by orders of magnitude and can be explained with a different processing of the file accesses in the storage system – an alternative I/O path.

We investigate a method to automatically determine the alternative I/O path and quantify the significance of knowledge about the internal processing. It is shown that the prediction error is improved significantly with this approach.

Keywords: file system, performance, modeling I/O, artificial neural networks.

Introduction

Tools are demanded that help users of HPC-facilities to implement efficient Input/Output (I/O) in their programs. It is difficult to find the best access parameters and patterns due to the complexity of parallel storage systems. The processing of file accesses in a storage system can be viewed as a task that is sequentially propagated along an I/O path in the storage system. Starting at the invoking processor, the storage system is searching for the data, going further and further through the memory hierarchy, until all data is found, so it can be returned to the processor.

Currently, users have to optimize their programs for each system individually without much assistance. To develop tools which support the implementation of efficient I/O, computational models of the storage system are important. For instance, a tool could estimate the I/O path and classify accesses as outliers if they behave abnormally. For single hard disk systems such a model can be derived analytically [11]; however, for the complex storage system of a supercomputer, these models become too difficult to configure [13].

In this paper, we evaluate predictors of I/O performance using machine learning with artificial neural networks (ANNs). In our analysis, we use ANNs with different input information for the prediction of access times. Additionally, we evaluate strategies to identify the I/O path without a-priori (expert) knowledge of it.

Because of the strong linear correlation between access time and access size, the problem seems to fit linear models. However, our results show that the relation of file access parameters to access time is not sufficiently represented by linear models. ANNs achieve significantly better results than linear models. Our analysis suggests that the I/O path used by the storage system considerably influences the file access time. Therefore, it becomes key for a good model of access

[1]Universität Hamburg, Hamburg, Germany; E-Mail:2schmid@informatik.uni-hamburg.de
[2]German Climate Computing Center (DKRZ), Hamburg, Germany

times to derive knowledge about I/O paths. Unfortunately, I/O paths are difficult to deal with, as it is unknown which path was used for a file access.

This paper is organized as follows: Related work is provided in Section 1. In Section 2, we explain our analysis of file access times. Firstly, we develop a simplified model of the I/O path. Next, a method for approximating I/O paths with derived classes from the error of a prediction model is proposed. At last, we introduce a set of models for access time prediction. Afterwards, in Section 3, we evaluate our analysis of the storage system by measuring access times in various test series, studying the obtained access times and the predictions of our models to them. In the end, the final Section summarizes the paper and suggests future work.

1. Related work

Generally, storage systems are modeled in two different ways for access time prediction; either using white-box modeling or black-box modeling [4].

- **White-box modeling**: The storage system itself is simulated. Details of hardware components like rotation speed of the magnetic disk in a hard drive are considered. The processing of a file access can then be simulated in the model and the resulting access time is then used as prediction for the actual system. Processing and resulting performance can be analyzed in detail on the model.
- **Black-box modeling**: The model abstracts from the real storage system. System performance is approximated without considering the causes. This procedure corresponds to an emulation of a storage system. In contrast to the white-box model, processing of file accesses can't be analyzed on the model itself.

These two ways of modeling are fundamentally different and have to be differentiated.

1.1. White-box modeling versus black-box modeling

For the in-depth analysis of reasoning for behavior of a storage system, a white-box-model is desirable. On the one hand, the modeled system is represented in the model. Thus it, can be examined. On the other hand, these models can be very precise if modeled correctly [11]. There are important limitations of white-box modeling, however. For every system an individual model has to be created and a model becomes quite intricate for single hard drives already [4]. To approach the complexity of white-box modeling, Ruemmler and Wilkes analyzed the relevance of different hard drive components for the model deviation to save effort for insignificant parts [11]. However, white-box modeling is usually used for simple systems like a single hard drive. For these hard drives white-box-modeling is already very demanding, hence for the complex parallel storage system of a supercomputer it's not a feasible approach [13].

The application of black-box-modeling is easier and more flexible as it's independent from the individual system. Stochastic approaches coupled with data mining methods are mostly used for black-box modeling; for example, a combination of regression trees can be used [6], or selective bagging classification and regression trees [13].

1.2. Prediction of I/O performance with ANNs

Computability of ANNs was researched by Rojas [10] and Cybenko [5]; they demonstrated the possibility of modeling non-linear systems. Cybenko also proved the *universal approximation*

theorem, which states that feed-forward networks with sufficient complexity can approximate any continuous functions on compact subsets of $\mathbb{R}^n$. Therefore, ANNs should be sufficient for predicting I/O performance as long as there are no contradictions in behavior. However, no statement can be made about the necessary structure of the network for this task.

Crume et al. developed a method using ANNs which exploits periodic patterns in sequences of file access times [4]. A Fourier analysis is used to determine the most important frequencies, which are then used as input information for the ANNs. In a following publication [3] they move away from Fourier analysis, but instead use additional sine waves as input for the ANNs. This seems to be a promising approach for predicting access times of single hard drives, where the rotational characteristics of the magnetic disk are an essential consideration. It is, however, difficult to extrapolate behavior from a single disk to a distributed storage system consisting of thousands of disks that utilizes several optimization mechanisms.

1.3. I/O performance prediction in HPC

Performance analysis in HPC is an important task to examine and improve system efficiency. For instance Liu et al. simulated scheduling algorithms for research [9]. The simulation used the white-box modeling tool DiskSim for a prediction of occurring file accesses [1].

There are a few simulators for parallel storage systems, for example CODES [2] utilizes a scalable infrastructure to investigate relevant research issues, such as the importance of burst-buffers. PIOSimHD is a simulator that is able to replay (MPI) application traces on a generic I/O system [8].

Analytical and machine learning models for predicting performance are another choice to optimize performance. Kunkel et al. utilized access time prediction with decision trees for varying parameterizations of ROMIO for access of non-contiguous data [7]. Instead of searching for optimal parameters by testing, they were able to find good values through estimating performance.

2. Modeling I/O Access Times

2.1. Characteristics of the Data

We used a synthetic benchmark in which identical measurements for random or sequential I/O are performed and the access time is measured for each operation individually. The resulting timings are stored together with file access parameters in a file that is then used to build and validate the models. This approach allows for an in-depth analysis of system behavior for different use cases from the perspective of a single client. The stored parameters are:

- **Access size**: Number of bytes to read or write.
- **Access type**: Differentiates reading and writing.

Directly measurable or derivable attributes are:

- **Offset**: Distance of file beginning to the starting point of access.
- **Delta-Offset**: Can be calculated for file access i as Offset[i] - (Offset[i-1] + access size[i-1]).
- **Access time**: Time for performing the I/O.

Knowledge of internal aspects of the system about the current system utilization or about the storage media that have to be addressed are not used by the model.

2.2. Model of the I/O path

The internal processing of a file access in the storage system can be viewed using the I/O path. The I/O path of a file access is the interaction between all components that are involved in the transfer of data read or written for its execution. For example, for a node local file system this includes operating system, memory and storage device/medium. Remote systems add network traffic and generally server sided components. Usually, in parallel file systems the I/O of the client is transferred through operating system, client side file system modules, network, file system servers, storage devices and ultimately media. In several of these steps, data may be cached in memory and optimizations can take place.

The resulting access time depends on the depth of this I/O path, because with increasing distance the storage media along the path is slower. While the first levels in memory hierarchy (CPU caches) are the fastest volatile storage, the main memory is already an order of magnitude slower; the same applies for the step into the parallel storage system, that is deployed as servers connected via network to the computer nodes. Reading file accesses are more severely affected by varying depths of I/O paths than writing file accesses, which can be deferred using write-behind and propagated lazily to the disk drives.

2.2.1. I/O path for access time prediction

Due to the exponential decay of processing speed in the hierarchy, file accesses with similar access parameters, but varying I/O path, can be easily differentiated by a step in the magnitude of access time. As the access time is dominated by the slowest component along the I/O path, a step in the measured access time occurs between the I/O paths of diverging length. In Figure 1, measurements of sequential read accesses are shown. The different groups of measurements with equal access parameters (points with the same color) are clearly visible. For a few access sizes, some clusters are observable, this step in the magnitude of access time and can be explained with varying I/O paths (this figure will be discussed in further detail in the following chapters).

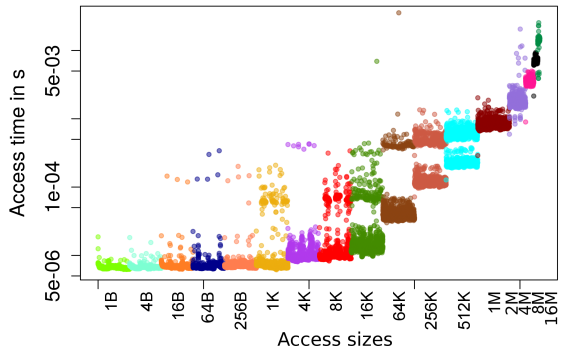


Figure 1. Time series for sequential reads; access sizes increase from left to right and all measurements with equal access parameter values have the same color

2.3. Estimating the I/O path using error classes

We utilize a concept that we call *error classes*. Each class is supposed to approximate an I/O path based on the error (residuum) of the observed access times to a baseline model. The baseline model is constructed as a function of the file access parameters.

At a first consideration, one could come up with the idea of simply clustering the data seen in Fig. 1, because the groups of measurements with varying I/O paths might be correctly differentiated by it. However, it is not a good approach. One would have to cluster for every set of measurements with identical access parameters individually, as access time is strongly dependent on the access size. In a real application scenario, a large number of different access parameters with only very few instances occur, which makes this approach unfeasible.

Another idea might be to use the difference of an arbitrary value to the measured file access time as a parameter for the clustering. Such a value would not be appropriate for all magnitudes of access time occurring in the data. I/O paths with only a small difference in their typical access time might not be found with this approach, because the gap between them doesn't matter at the scale of the chosen value.

Our approach uses the residues of a model like linear regression that can't differentiate between measurements with equal access parameter values. Basically, the linear regression predicts an average access time for the measurements of any given size. The deviation of an individual measurement for these access parameters to the predicted average value is then characteristic for the I/O path. Clustering the residues with a k-Means algorithm allows us then to assign each measurement to a specific cluster, which is the approximation of its I/O path. We call the retrieved clusters **error classes**. We use the absolute error of linear regression as a clustering parameter, so that one cluster can, for example, represent a positive deviation of 1 millisecond. Ideally, it might be the case that file accesses independent of the access parameters that took 1 millisecond longer than usual are processed on the same I/O path. In that case error classes are directly corresponding to I/O paths. However, this has to be investigated.

We use error classes to quantify the importance of knowledge about I/O paths for access time prediction. For the actual prediction of access times, error classes can't be used, because a measured access time is required to determine the error class of measurement. However, the method could be used for a tool, which analyses the execution time of I/O on the client side made during application of a program and assign the observed time to the error class and, thus, the I/O path. It could analyze whether file accesses were slower than expected and therefore unfavorable I/O paths were used during their execution. If a direct correlation of error class to I/O paths were found, such a tool might be able to inform about percentages of used I/O paths.

2.4. Models

Access times of file accesses can be predicted using various models. While we evaluated several models for this paper, only a few relevant models are described; more models are described in [12]. Each model seeks to correlate its known file access parameters to the corresponding access time. Linear regression is used as a baseline model with a simple mapping of access size to access time. Additionally, three models with different input information utilizing ANNs are used:

- The most simple ANN-model only receives information about access sizes, delta-offsets and access types as input.
- The second ANN-model is called **ema-model**. Additionally to the parameters of the previous model, it receives information about the past data throughputs of the system. This can be used to exploit time dependencies of the I/O performance.
- The third ANN-model, called **error-class-model**, receives error classes in addition to the access parameters of the first ANN-model.

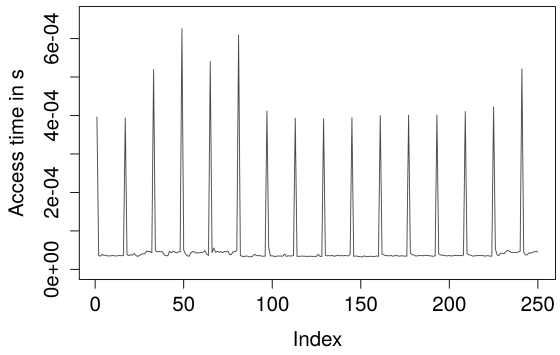


Figure 2. Small sequence of read accesses with periodic peaks in access time

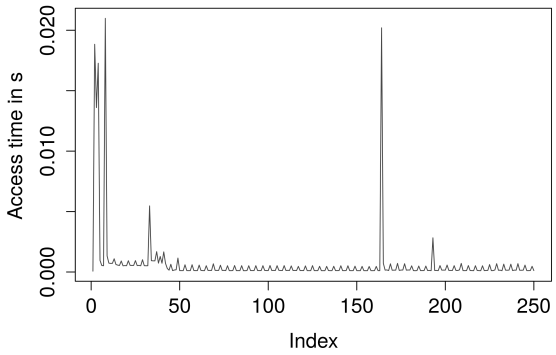


Figure 3. Small sequence of read accesses with sporadic peaks in access time

The error-class-model is used as described in Section 2.3 to examine the potential improvements for access time prediction due to knowledge about I/O paths. Therefore, we evaluate how much the prediction of access times of a model using error classes improves compared to a model without.

The idea of the ema-model is to exploit periodic changes in the performance of the storage system. Our analysis of measurements as well as the work of Crume et al. [4] indicate that periodic phenomena of access times in sequences of files accesses can be exploited for access time prediction. Figure 2 shows a small series of measurements of sequentially read file accesses. The peaks in access time are occurring periodically in this particular sequence. Such behavior appears only partially, often access times fluctuate without observable periodicity as in Figure 3. An occurrence of a periodic pattern could be explained with a processing of the storage system in which a larger amount of data is loaded into the cache at the same time as soon as a cache-miss happens.

With knowledge about the periods of peaks a model could make better access time predictions. The ema-model is supposed to use its input information of previous data throughput for that. The data throughput has peaks in the same period as the access time. We use the data throughput instead of the access times of measurements, because an analysis of the data throughput could find periodic behavior of system performance even in a sequence with different access sizes. Internally the model calculates the exponential moving average (EMA) of data throughput of measurement i as:

$$EMA(i) = 0.5 \cdot t(i) + 0.5 \cdot EMA(i-1) \tag{1}$$

With $t(i)$ the access time of measurement i. The influence of data throughput of a measurement is exponentially decreasing in the series of EMAs. In Figure 4 the EMA-function for a function with periodic peaks can be seen. The ema-model can use $EMA(n-1)$ for the prediction of access time of measurement n. If the model is able to find threshold values of the EMA-function after which another peak follow, it can use this input information for better access time predictions on sequences with periodic peaks in access time. Depending on how relevant periodic behavior of access times is in our measurements, this additional input information could improve the model.

Additional models that were examined in [12], but won't be analyzed in further detail in this paper, are summarized in the following list:

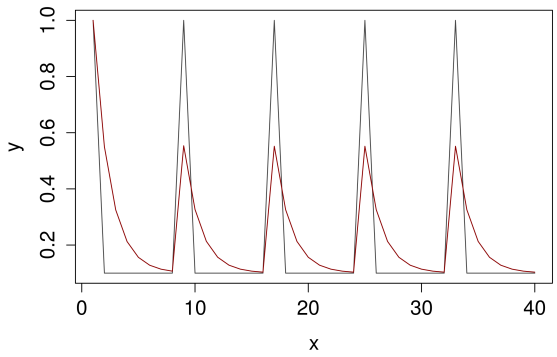


Figure 4. Exponential moving average (in red) for a function with periodic peaks

- Using multiple linear regression, we analyzed the performance of other baseline models. We used access size and delta-offset as tuple and also access size, delta-offset and access type as triple to find a mapping to access time. Despite having more information to work with both models achieve equal or worse deviations to the measured access times than the simple linear regression which only considers access size. If access type and delta-offset have a meaningful correlation to the access time of a file access, they require more sophisticated tools than linear regression to be exploited.
- An ANN-model receiving knowledge about the previous measurement was examined as well. Additionally to the input information of the simple ANN-model the access size, the access type, the delta-offset and also the access time of the previous file access was given as input for this model. Conceptually this model might be able to compare the actual performance on the previous file access with its own prediction for it. If the actual performance deviates from the predicted performance this model could exploit this knowledge to adapt its prediction for the following file access. In case of a prolonged period of high workload this adaptation might have positive effect on the predictions. The results of this model, however, were throughout worse compared to the simple ANN-model which has less information to work with, even though quite complex network structures were used in the learning process (16 hidden layers with 17 neurons each for the sequential access pattern and 14 hidden layers with 17 neurons each for the random access pattern achieved the best results). Therefore we conclude that the performance of the direct predecessor of a file access can't or is too complicated to be exploited for an improved prediction.
- An ANN-model similar to the error-class-model described above, using error classes obtained from the residues of the simple ANN-model instead of residues from the linear regression, was examined as well. This approach lead to very similar prediction performance to the model using error classes from linear regression. On the data set of sequential file accesses this model achieved with $2.4 \cdot 10^{-5}$ to $2.0 \cdot 10^{-5}$ a slightly worse and on measurements with random access pattern with $8.9 \cdot 10^{-4}$ to $10.3 \cdot 10^{-4}$ a slightly better mean average error.

3. Evaluation

The execution and results of analysis are summarized in the following way. First, the test system used for measurements and the strategy for systematic measurement are presented. Next, we explain the computation of error classes. Finally, the quality of predictions of the different models are examined.

3.1. Test system

The measurements were done on the supercomputer Mistral (phase 1 system) of the DKRZ (Deutsches Klimarechenzentrum)[3]. The phase 1 system of Mistral operates with the parallel distributed file system Lustre (Lustre 2.5, Seagate's edition) and consists of over 1500 compute nodes and 30 Petabyte storage. Each compute node consists of two E5-2680v3 with a clock rate of 2.5 GHz and 30 MiB L3 Cache. Measurements were done during daily operation, typical fluctuations in workload may have influenced the execution of benchmarks.

3.2. Benchmarking

We created a simple benchmark (io-model) that used POSIX read/write() and measured time for each I/O individually. In this paper, it is executed on one node with one processor using a single stripe (one OST) to study the quality of the predictions. Each series of measurements follows a certain pattern to study system behavior systematically.

Sequential and random I/O patterns are measured, each with reading and writing file accesses. Our test file has a size of 10 GiB which theoretically fits in the main memory of computer nodes, but measurements and observations from Lustre's /proc statistics suggest that caching is not effective for this configuration (sequential reads are improved by read-ahead, though). Occasional access of hard drives in the parallel storage system is likely. Most series of measurements contain 10 000 file accesses with a certain access size, only series with sequential access and an access size of greater than 2 MiB have fewer measurements as the end of our 10 GiB file is reached. Every series with specific access pattern, access size and access type is repeated three times. Access sizes are varying from 1 B to 16 MiB (in detail: 1 B, 4 B, 16 B, 64 B, 256 B, 1 KiB, 4 KiB, 8 KiB, 16 KiB, 64 KiB, 256 KiB, 512 KiB, 1 MiB, 2 MiB, 4 MiB, 8 MiB and 16 MiB).

3.3. Analysis of measurements

The measured access times for the four different cases are summarized in Table 1.

Table 1. Overview of file access times for the different cases

Case	Min. value	1. Quartile	Median	Arith. mean	3. Quartile	Max. value
Sequential reading	6.2e-06	6.9e-06	9.7e-06	3.8e-04	1.3e-04	5.6e-02
Sequential writing	7.5e-06	8.4e-06	1.6e-05	4.3e-04	2.2e-04	3.0e-02
Random reading	6.5e-06	1.4e-05	1.8e-03	7.4e-03	1.3e-02	5.3e-01
Random writing	1.1e-05	2.6e-04	4.5e-04	3.8e-03	2.2e-03	2.1e+00

As expected, sequential file accesses are on average much faster than random accesses especially for the reading case, also random reads are slower than writes. In Table 2, we consider the correlations between file access parameters and the resulting access time, a strong correlation can be exploited for access time prediction.

Table 2. Correlations between file access parameters and access time

Attribute	Sequential access pattern	Random access pattern
Access size	0.973	0.3291
Delta-offset	NA	0.0069
Access type	0.018	-0.1068

[3] `http://www.vi4io.org/hpsl/2016/de/dkrz/start`

Note that sequential access leads to a constant delta-offset of 0.
Sequential file accesses have with 97.3% a clear correlation to access time, this is with only 32.91% to a lesser degree still true for the random accesses. This is why linear regression can be expected to have acceptable results for access time prediction, especially for the sequential access case.

To study the distribution of measured file access times we display them as follows: All measurements with equal access size have the same color, access sizes of file accesses are increasing in the graphs from left to right. The resulting graphs can be seen in the Figures 5 to 8. Note that they use logarithmic scale.

For readability, the slowest and fastest 1% of measurements are purged and only every 25th measurement is plotted in the figures.
The correlation between access size and access time can be easily identified. As described in chapter 2.2.1 the split of measurements with identical access parameter values, that occurs in particular in the cases of sequential file accesses, can be explained with a processing in the storage system along different I/O paths. It is worth mentioning that the groups of access time for different access sizes are partially at the same range of magnitude. This indicates as well that the splits are caused by varying I/O paths as one I/O path has a typical access time, which only changes slightly for different access sizes.

The phenomenon of these splits clarifies why despite the strong correlation of access size to access time a linear model might produce sub-optimal predictions. Even for identical access parameter, it is difficult to predict which I/O path will be used as the storage system decides the I/O path based on the system state (e.g., is the data available in the client's cache). In Figure 7, the importance of knowledge about I/O paths for the access time prediction can be seen: For example, file accesses of only 1 B can be processed as slow as accesses with 16 MiB if the longest

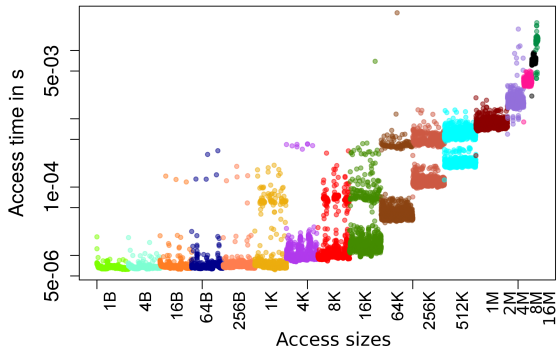


Figure 5. Measurements of sequential reads

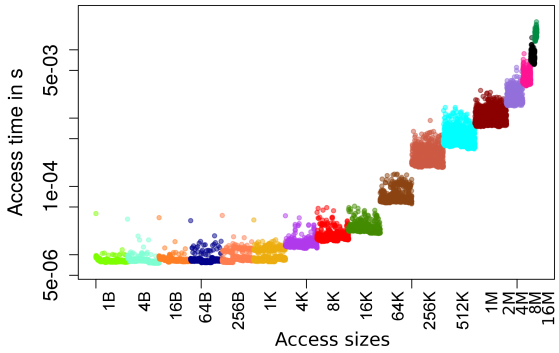


Figure 6. Measurements of sequential writes

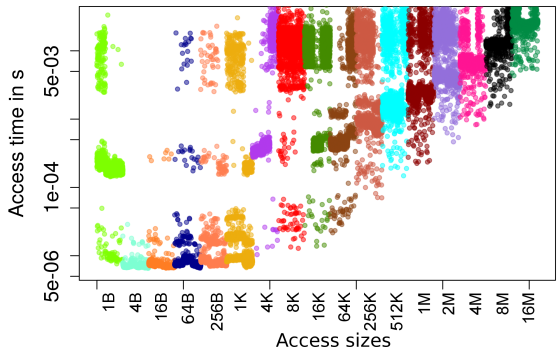


Figure 7. Measurements of random reads

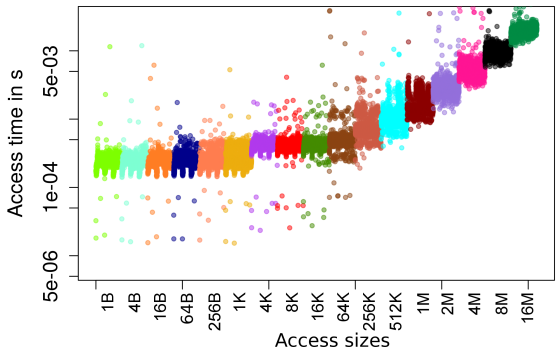


Figure 8. Measurements of random writes

I/O path is used. For writing file accesses the splits only occur occasionally and they are less drastic. These measurements might be well represented with a linear model.

3.4. Analysis of error classes

Error classes are calculated as clusters applying the k-means algorithm on the residues of the linear regression model. As an estimation of possible I/O paths we classify into 10 clusters. In Figure 9, the predictions of linear regression on the random reads can be seen as blue points. Since a linear model predicts the mean value for all measurements with identical arguments, it cannot distinguish between the different I/O paths and may not even predict a valid value (e.g., the median).

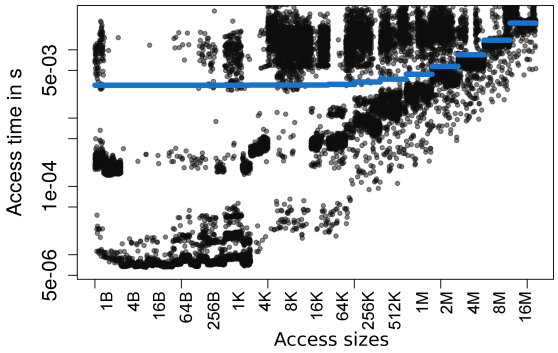


Figure 9. Predicted access times of the linear regression model as blue points

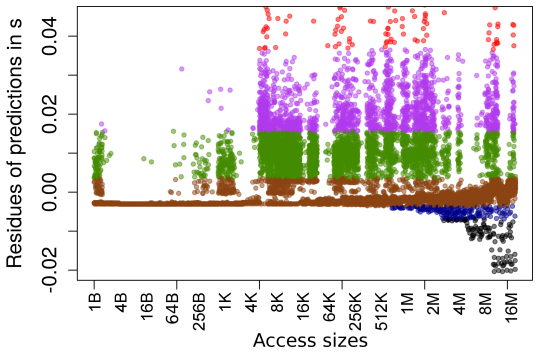


Figure 10. Residues of the linear regression model, colors are corresponding to error classes

For identifying the error classes, the difference between the prediction of linear regression and actual access time is used as input for the clustering algorithm. In Figure 10, the residues for all measurements are shown. The different colors correspond to the computed clusters, which are our error classes. Note that due to the random initialization of k-Means, different runs may result in different clusters [4].

The clusters are differentiated along horizontal lines in the graphs. Without going into further details, one can recognize from the graphs that the error classes are not perfectly approximating the I/O paths. The measurements with the smallest access sizes (on the far left) are only differentiated into two different error classes. However, the two error classes mainly represent measurements with positive errors, and close to 0 or negative error, what seems to be a reasonable approximation of I/O paths.

The computed error classes can be examined in further detail with the data from Table 3. Ideally for a good correlation of I/O paths to error classes, each error class should represent one particular value of data throughput that would be characteristic for the slowest storage medium occurring in the I/O path. This is mostly true for the error classes. The classes 4, 5 and 6, however, have with $5.5 \cdot 10^7$, $6.1 \cdot 10^7$ and $5.8 \cdot 10^7$ very similar average throughput values, which might be a hint for overfitting error classes.

The vast majority of measurements were assigned to error class 3, which is the error class that covers the range around a prediction error of 0. Therefore, most access times can be predicted sufficiently by the linear regression model.

[4] We will consider more robust algorithms in the future.

Table 3. Characteristics of computed error classes for random file accesses in detail

	averaged values			prediction error			
Error class	Throughput (B/s)	Access size (B)	Access time (s)	Min (s)	Avg (s)	Max (s)	# of members
1	1.4e+09	1.5e+07	0.0130	-0.0210	-0.0090	-0.0069	9467
2	9.9e+08	9.1e+06	0.0101	-0.0069	-0.0047	-0.0036	54371
3	2.3e+08	1.4e+06	0.0024	-0.0036	-0.0025	0.0036	825974
4	5.5e+07	1.2e+06	0.0143	0.0036	0.0096	0.0156	85462
5	6.1e+07	2.3e+06	0.0276	0.0156	0.0216	0.0366	37862
6	5.8e+07	4.0e+06	0.0598	0.0366	0.0516	0.0976	4695
7	3.2e+07	4.7e+06	0.1528	0.0977	0.1438	0.2066	1443
8	4.5e+06	1.2e+06	0.2741	0.2067	0.2696	0.4728	567
9	3.0e+05	1.6e+05	0.6956	0.4822	0.6923	1.0063	123
10	9.4e+02	1.0e+03	1.3627	1.0396	1.3597	2.1216	36

One sign for correct approximation of I/O paths are error classes which have equal access sizes but different access times. In such cases a differentiation of measurements with equal access parameters takes place. This occurs for example with the error classes 4 and 8. Both classes have an average access size of $1.2 \cdot 10^6$ Bytes, but with 0.0143 and 0.2741 seconds very different average access times. The error classes of sequential file accesses display similar characteristics as the analyzed random file accesses. However, since sequential accesses are not able to stress all different I/O paths, this result is more interesting.

3.5. Prediction of file accesses

Next, we study the results of our models. To investigate overfitting and the accuracy of the models, we split the available data into a training set and a test set. The test set consists of all measurements that weren't used for training. The parameters for the ANN-models are varied to explore appropriate configurations; they are trained with a wide span of different parameter values for the number of hidden layers and the number of neurons per layer. They had 1000 randomly chosen measurements for each pair of access size and access type as training set. In total, that are 34 000 measurements which should be a significant amount of data to avoid the situation, in which too few data points are available to train the system.

In Tables 4 and 5 characteristics of the ANNs with the smallest average prediction error can be seen. The models for the sequential data set had in general bigger network structures. We do not completely understand the reason, as we naively expect that random I/O is more complex than sequential I/O. Presumably, the system uses additional layers to adjust for rare events or, for example, to differentiate read-ahead and write behind cases.

Table 4. Characteristics of the most successful ANN-models for seq. file accesses

Model	Hidden layers	Neurons per layer	Computing iterations	Training duration (s)
simple ANN-model	12	8	1934	1444
ema-model	11	8	1878	1379
error-class-model	15	27	1551	8655

Table 5. Characteristics of the most succesful ANN-models for random file accesses

Model	Hidden layers	Neurons per layer	Computing iterations	Training duration (s)
simple ANN-model	4	5	1310	222
ema-model	7	5	1106	461
error-class-model	9	22	283	1201

We used the following metrics to quantify the error:

- **MAE**: Mean average error.
- **Avg-MAPE**: The mean absolute percentage error averaged about 12 instances of ANNs with equal parameter values, but different (random) initial network weights.
- **Train-MAPE**: The mean absolute percentage error on the training set.
- **MAPE**: The mean absolute percentage error (on the test set).
- **MSPE**: Mean squared percentage error.
- **RQ3**: Third quartile of prediction errors, which means that three quarters of the test set were predicted with smaller deviation to the true value that this.
- **PMax**: The biggest prediction error in percent.

The prediction errors of our models can be found in the Tables 6 and 7. Listed values refer to the model with the smallest average prediction error. We can observe that for all models values for Avg-MAPE, Train-MAPE and MAPE are close to each other. This means that the training set was representative for the test set with little overfitting, and also the model behavior was stable and did not depend so much on the random initiation of network weights.

Table 6. Results of our models on the sequential file accesses

Model	MAE (s)	Avg-MAPE (%)	Train-MAPE (%)	MAPE (%)	MSPE (%)	RQ3 (%)	PMax (%)
error-class-model	2.0e-05	9	7.7	8.6	14	10	275
ema-model	5.7e-05	14	13.2	13.7	22	18	2336
simple ANN-model	6.0e-05	14	13.6	14.1	22	18	295
Linear regression	7.6e-05	NA	NA	50.8	59	73.7	326

Table 7. Results of our models on the random file accesses

Model	MAE (s)	Avg-MAPE (%)	Train-MAPE (%)	MAPE (%)	MSPE (%)	RQ3 (%)	PMax (%)
error-class-model	0.00103	32	32	31	119	27	4272
simple ANN-model	0.00313	106	104	103	530	70	21786
ema-model	0.00305	421	87	86	619	62	45320
Linear regression	0.00476	NA	NA	5578.4	14185	1158.1	46941

It becomes clear that linear regression is sub-optimal for the prediction of file access times for random file accesses, where its MAPE is about 5578.4 %. For the sequential case the results are better, but still with more than three times higher average percentage error.

The second thing one can take from the results is that the ema-model did not work as intended. Its error values are very close to the simple ANN-model which had less input information to work with. Going a little bit more into detail it is notable that the MAPE is smaller for the ema-models compared to the model without EMA-values. Thus, the additional knowledge was in general useful for better prediction; however, the MSPE and PMax values are higher for the ema-model which means that some predictions were significantly misguided by the EMA-function value. The reason is that the I/O path does not change in a well predictable way (at least from the perspective of a client).

In contrast, the error-class-model worked excellently. The mean average error compared to the model without error classes has been reduced to a third for both use cases. This is a strong confirmation for the thesis that knowledge about I/O paths is crucial for access time prediction. However, it is also clear that the additional parameter of the error class improves the overall performance of the predictor, because it contains knowledge about the measured access times.

To investigate this further, we have to analyze measurements in more detail. To have a further look at the difference that error classes make on the access time prediction, we analyze the predictions of the two models for the case of random reads in Figure 11 and 12.

The model without knowledge about I/O paths is forced to predict some kind of average access time for each set of measurements with equal access parameter values. Therefore, its predictions are in between the two main groups of access time for the higher access sizes. With knowledge about our approximations of I/O paths the error-class-model can discriminate measurements with equal access parameter values and achieve more accurate predictions that way.

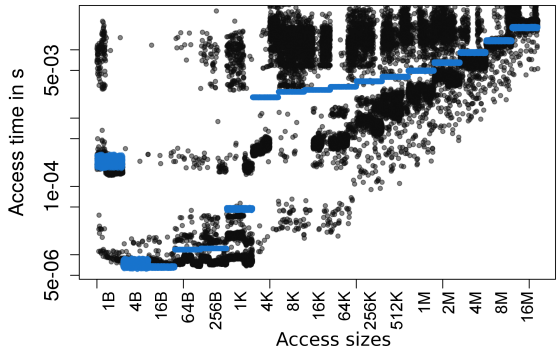


Figure 11. Predicted access times of the simple ANN-model as blue points

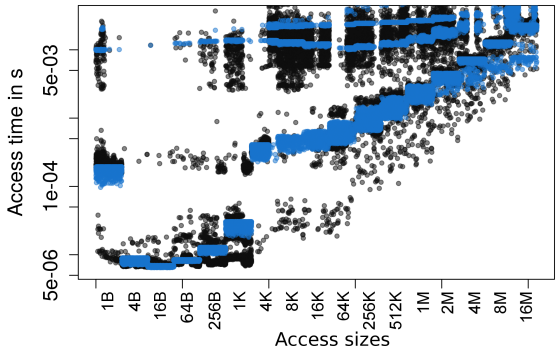


Figure 12. Predicted access times of the error-class-model as blue points

Conclusion and future work

In this paper, we analyzed the performance of a supercomputer's storage system. Using a machine learning approach with artificial neural networks, we developed different models for file access time prediction. Through our study of measured file access times and model results we were able to gain knowledge about the behavior of the storage system. We found out that linear models are not feasible for access time prediction despite the strong correlation of access time to access size. Our models utilizing ANNs achieved much better results than linear regression.

The hypothesis that knowledge about the internal processing of a file access in form of I/O paths is essential for access time prediction was supported by our data. This is because file accesses with equal access parameters can produce strongly deviating access times depending on the I/O path. The model of deriving knowledge about I/O paths by exploiting periodic performance of the storage system was not successful. The additional input information did not lead to a significant improvement of access time predictions.

However, with our method of clustering residues of linear regression for approximations of I/O paths as error classes, we were able to illustrate the importance of I/O paths for access time prediction. The ANN-model with additional input information of error classes was able to reduce its average prediction error to a third compared to the ANN-model with only access parameter values as input information.

For future work a more elaborate approach for exploitation of the periodic storage performance could be used. Crume et al. had success on access time prediction for single hard drives using Fourier analysis [4] or additional sinusoids as input for ANNs [3].

We will research the method to estimate the I/O path based on client measurements further. Residues of other models than linear regression could be used and one could try to assign error classes to I/O paths in a real system – the administrator would have to investigate the error classes and define them according to the storage technology. The method could also be a starting point to develop a tool that provides information on the effectiveness of the used I/O during execution of a program.

References

1. John S. Bucy, Jiri Schindler, Steven W. Schlosser, and Gregory R. Ganger. The DiskSim Simulation Environment Version 4.0 Reference Manual, 2008.

2. Jason Cope, Ning Liu, Sam Lang, Phil Carns, Chris Carothers, and Robert Ross. CODES: Enabling co-design of Multilayer Exascale Storage Architectures. In *Proceedings of the Workshop on Emerging Supercomputing Technologies*, volume 2011, 2011.

3. Adam Crume and Carlos Maltzahn. Latent Frequency Synthesis for Behavioral Hard Disk Drive Access Time Models.

4. Adam Crume, Carlos Maltzahn, Lee Ward, Thomas Kroeger, Matthew Curry, and Ron Oldfield. Fourier-assisted Machine Learning of Hard Disk Drive Access Time Models. In *Proceedings of the 8th Parallel Data Storage Workshop*, PDSW '13, pages 45–51, New York, NY, USA, 2013. ACM. DOI: 10.1145/2538542.2538561.

5. G. Cybenko. Approximation by Superpositions of a Sigmoidal Function. *Mathematics of Control, Signals, and Systems*, 2:303–314, 1989. DOI: 10.1007/BF02551274.

6. Chengjun Dai, Guiquan Liu, Lei Zhang, and Enhong Chen. Storage Device Performance Prediction with Hybrid Regression Models. In *Proceedings of the 2012 13th International Conference on Parallel and Distributed Computing, Applications and Technologies*, PDCAT'12, pages 556–559, Washington, DC, USA, 2012. IEEE Computer Society. DOI: 10.1109/PDCAT.2012.126.

7. Julian Kunkel, Michaela Zimmer, and Eugen Betke. Using Machine Learning to Predict the Performance of Non-Contiguous I/O, 07 2015. DOI: 10.1109/PDCAT.2012.126

8. Julian M Kunkel. Simulating Parallel Programs on Application and System Level. *Computer Science-Research and Development*, 28(2-3):167–174, 2013.

9. Yonggang Liu, Renato Figueiredo, Dulcardo Clavijo, Yiqi Xu, and Ming Zhao. Towards simulation of parallel file system scheduling algorithms with PFSsim. In *Proceedings of the 7th IEEE International Workshop on Storage Network Architectures and Parallel I/O (May 2011)*, 2011.

10. Raúl Rojas. *Neural Networks: A Systematic Introduction*. Springer-Verlag New York, Inc., New York, NY, USA, 1996.

11. Chris Ruemmler and John Wilkes. An introduction to disk drive modeling. *IEEE Computer*, 27:17–28, 1994.

12. Jan Fabian Schmid. Vorhersage von E/A-Leistung im Hochleistungsrechnen unter der Verwendung von neuronalen Netzen. Bachelor's thesis, Universität Hamburg, 12 2015.

13. Lei Zhang, Guiquan Liu, Xuechen Zhang, Song Jiang, and Enhong Chen. Storage Device Performance Prediction with Selective Bagging Classification and Regression Tree. In *Network and Parallel Computing, IFIP International Conference, NPC 2010, Zhengzhou, China, September 13-15, 2010. Proceedings*, pages 121–133, 2010.

DOI: 10.14529/jsfi160304

Analyzing Data Properties using Statistical Sampling – Illustrated on Scientific File Formats

Julian M. Kunkel[1]

Understanding the characteristics of data stored in data centers helps computer scientists in identifying the most suitable storage infrastructure to deal with these workloads. For example, knowing the relevance of file formats allows optimizing the relevant formats. It also helps in a procurement to define benchmarks that cover these formats.

Existing studies that investigate performance improvements and techniques for data reduction such as deduplication and compression operate on a subset of data. Some of those studies claim the selected data is representative and scale their result to the scale of the data center. One hurdle of running novel schemes on the complete data is the vast amount of data stored and, thus, the resources required to analyze the complete data set. Even if this would be feasible, the costs for running many of those experiments must be justified.

This paper investigates stochastic sampling methods to compute and analyze quantities of interest on file numbers, but, also, on the occupied storage space. It will be demonstrated that on our production system, scanning 1% of files and data volume is sufficient to deduct conclusions. This speeds up the analysis process and reduces costs of such studies significantly.

Keywords: Scientific Data, Compression, Analyzing Data Properties.

Introduction

Understanding the characteristics of data stored in the data center helps computer scientists to optimize the storage. The quantities of interest could cover proportions, i.e. the percentage of files with a certain property or means of certain metrics, such as achievable read/write performance, compression speed and ratio. For example, knowing the relevance of file formats may shift the effort towards the most represented formats. When 80% of the capacity is utilized by NetCDF4 files, performance analysis and optimization should target this file format first. Understanding the achievable compression ratio of available compression schemes helps in choosing not only the best one for user-specific compression, but also for file system level compression.

In the literature, one can be found studies that investigate compression ratio, de-duplication factor or improve performance of scientific middleware. Due to the long running time to apply any improvement on large amounts of data, many studies assume the benefit measured on a small data sample can be transferred to the scale on the data center. However, usually in these studies nobody pay attention if the data set is actually representative. In other words, they do not take into account the fraction of the workload that can actually benefit from the advancement. In statistics, the big field of sampling theory addresses this issue. Due to the law of large numbers, there are methods to draw instances appropriately and deduce properties from the sample set to the population with high confidence. However, this process is non-trivial and a research discipline in statistics by itself [3].

This paper investigates statistical sampling to estimate file properties on the scale of data centers using small data sets and statistical simulation. The computation time used for the complete project was 517 core days. With 24 cores per node, a complete system scan of DKRZ's

[1]Deutsches Klimarechenzentrum, Bundesstraße 45a, 20146 Hamburg, Germany

system would have needed about 475 node days which would have cost at least about 4000[2] – while not revealing additional insight. Instead with 1% of scanned files or capacity, similar results are achievable , that means with 1% of scanned files we have confidence that we estimate sufficiently accurate the characteristics of the full system. If scanning all files has no impact on the conclusions we draw (e.g., we can save 51% of storage with compression A), then why should we have to scan all files which is a snap-shot of the system life, anyway? . Note that a significantly extended version of this paper containing results for compression will appear in [5].

1. State of the art

Existing research that analyzes properties of scientific data can be classified into performance analysis, compression ratio and data deduplication. The effort that investigates and optimizes performance usually picks a certain workload to demonstrate that the new approach is superior than existing strategies. A few studies analytically analyze typical patterns and optimize for a large range of access patterns. An example is the study in [8], which analyzes the access pattern for several workloads and discusses general implications. The research on optimization techniques, as far as known to the author, do not check how many people actually benefit from these optimizations and the implications on the system level.

In the field of compression, many studies have been conducted on pre-selected workloads, for example, see [1, 4, 6]. Some of those studies are used to estimate the benefit of the compression on the data center level, for example, Hübbe et al. investigate the cost-benefit for long-term archival. Similarly in [4], the compression ratio is investigated. However, in this case the selected data is a particular volume from a small system.

Modern file systems such as BTRFS and ZFS offer compression on system-side [7]. It is also considered to embed compression into storage devices such as SSDs [10] and evaluate it for OLTP workloads. In [2], Jin et.al investigate the benefit for compressing and de-duplicating data for virtual machines. They created a diverse pool of 52 virtual images and analyzed the impact.

As far as known to the author, statistical sampling techniques have not been used to investigate file characteristics for data centers.

2. Sampling of Test Data

To assess and demonstrate the impact of statistical sampling, firstly, a subset of data of DKRZ's supercomputer Mistral is scanned and relevant data properties about data compression and scientific file types are extracted. Our global Lustre file system hosts about 320 million files and 12 Petabytes of space is occupied; only a subset is scanned: 380k (0.12%) accounting for an (aggregated size) of 53.1 TiB of data (0.44%). To prevent data loss and ensure data privacy, the scanning process is performed using a regular user account and, thus, it cannot access all files. There are still 58 million files and 160 out of 270 project directories are accessible.

The scanning process used as a baseline in the paper works by running find to scan all accessible files, then select 10k files from each project randomly in a candidate list for the scans. Then, the list is permuted and partitioned into different threads. Multiple worker threads are started and each thread processes its file list sequentially running 1) the CDO [9] command

[2]Considering the TCO of purchasing (35M € and operating the system into account (more than 10M € for 5 years).

to identify the scientific file type, 2) the `file` command to guess the file types, and a list of compressors (LZMA, GZIP, BZIP2, ZIP). The `time` command is used to capture runtime information of each step and reported user time is used to assess performance. When 300k files are scanned, the threads are terminated and the results are collected.

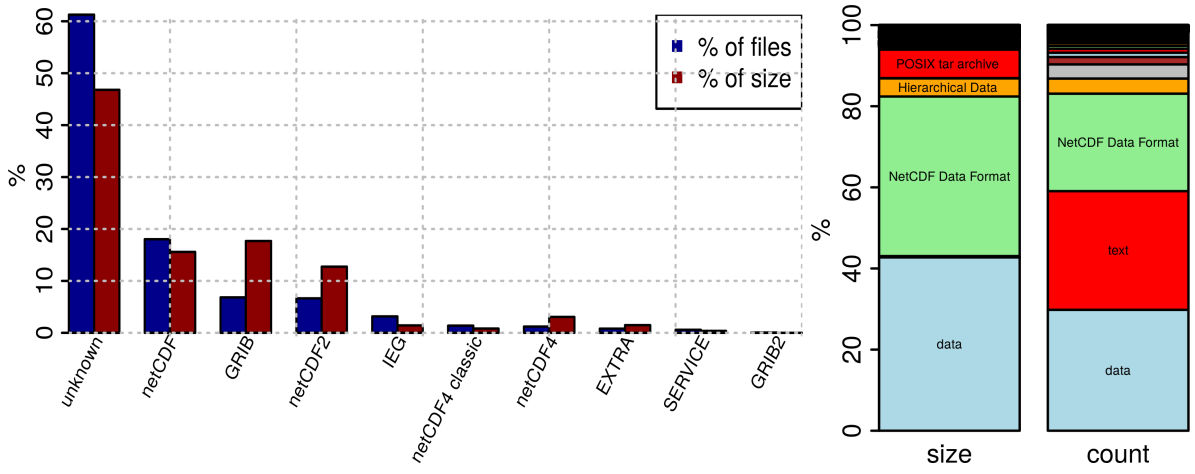
The strategy increases the likelihood that a representative sample of files is chosen from accessible projects. It is expected that projects differ in their file characteristics. The goal of the strategy is not to gain a completely representative sample, since this is to be developed within this paper. The limitations of this sampling strategy to investigate properties based on the occupied file size will be shown later.

Albeit the analysis described in the following sections are achieved with the suboptimal sampling, they correctly simulate the behavior of a system and show that the method delivers the correct results. But it means, that the obtained characteristics computed do not predict DKRZ's full data set correctly. We are currently applying the correct sampling technique on the full data set to identify the true characteristics for DKRZ.

3. Difference in Means Computed by File Count and File Size

This section gives an example to understand that the results vary depending on whether metrics are computed either on file count, i.e. each file is weighted identically, or by weighting each file with its occupied size.

The usage of scientific file formats is shown in Figure 1. The figure shows the relative relevance in terms of occupied space and a number of files of each file format. About 60% of the number of files and 47% of aggregated file size is non-scientific and cannot be resolved with the CDO tool. The dominant scientific formats are NetCDF3, GRIB1 and NetCDF2. The `file` command cannot identify and distinguish scientific formats as reliable as CDO, but can shed light over the distribution of the 60%. Looking at its output, the 60% of capacity seems to be dominated by TAR (7%) and GZIP compressed files (5%) – it classifies 43% of capacity as "data" and 40% as NetCDF (no version information provided). Looking at the proportions in terms of file count, roughly 30% are classified as data, 30% as text (e.g., code), 24% as NetCDF files, 4% as HDF5, and 3.5% as images. Other file types are negligible.



(a) CDO types (b) File types

Figure 1. Relative usage of file formats determined using CDO and `file`

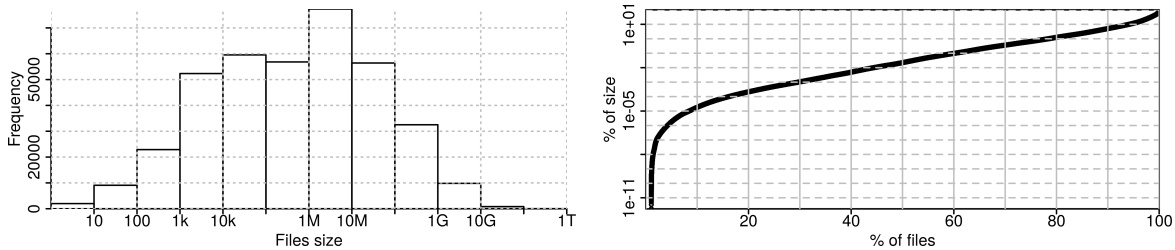
The difference between the proportion computed by the file count and by file size stems from the highly skewed distribution of file sizes and the different mean size across different file formats. Fig. 2 show the distribution of file sizes. Fig 2a) shows a histogram with logarithmic file sizes. In Fig. 2b) the relation between the file size and the file count is illustrated; to construct the figure, files have been sorted by size in ascending order and then the cumulative sum is computed. While the histogram suggests similarities between size distribution and a normal distribution, this is due to the logarithmic x-axis. In the cumulative view, it can be seen that aggregated 20% of files consume one millionth of storage space and 90% still consume less than 10% space. If a study takes small files as representatives, those fail to represent the storage capacity. Similar large files fail to represent the typical (small) file that must be handled by the storage.

4. Stochastic Sampling of Data

The way the quantities of interest are computed are either by file count, i.e. we predict properties of the population based on individual files, or by weighting the individual files with their size. From the perspective of statistics, we analyze variables for quantities that are continuous values or proportions, i.e. the fraction of samples for which a certain property holds. To select the number of observations that allows inference about the population, statistics knows methods for determining a sample size. More information about this topic is provided in the full paper [5].

Sampling method to compute by file count. When computing the proportion or the mean of a variable for files, a strategy is to enumerate all files on the storage system and then create a simple random sample, i.e. choose a number of files for which the property is computed.

Sampling method to compute by file size. Estimating values and weighting them based on the file size requires to enumerate all files and determine their size, then pick a random sample from the file list based on the probability defined by filesize/totalsize. Draws from the list must be done with replacement, i.e. we never remove any picked file. Once all chosen files are determined, the quantities of interest are computed once for each unique file. Then, each time we have chosen a file, we add our quantity of interest without weighting the file size, for example, the arithmetic mean can be computed just across all samples. Thus, large files are more likely to be picked but each time their property is accounted identically as for small files.



(a) Histogram (b) Cumulative file sizes (y-axis in log scale)

Figure 2. Distribution of file sizes

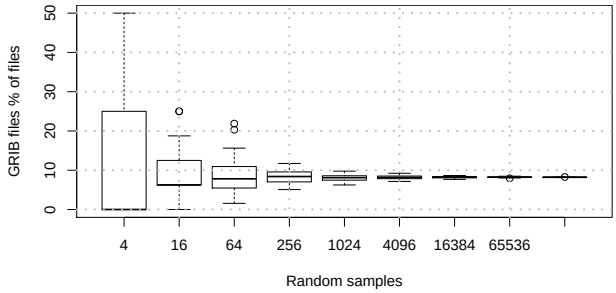
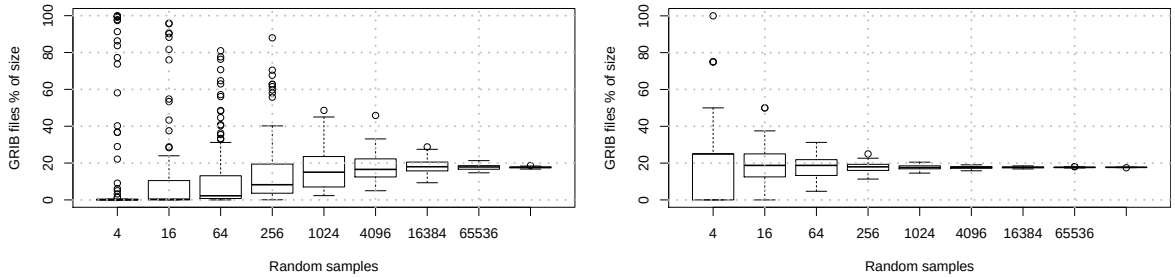


Figure 3. Simulation of sampling by file count to compute compr.% by file count



(a) By file count (this is suboptimal!). (b) Correct sampling proportional to file size.

Figure 4. Simulation of sampling to compute proportions of types by size

Robustness. To illustrate the stability of the approach, a simulation has been done by drawing a variable number of samples from the data. The simulation is repeated 100 times and a boxplot is rendered with the deviations. Naturally, the repeats of a robust method should have little variance and converge towards the correct mean value. The result for the proportion of GRIB files are given as an example, but the results for all variables behave similar. In Figure 3, it can be clearly seen that the error becomes smaller.

The sampling strategy to compute quantities on file size is shown in Figure 4b). Similarly, to the correct method for sampling by file count it converges quickly. However, if we simply use a file scanner to compute the metrics on size, but it would choose files randomly without considering file sizes, we would achieve highly unstable results (Figure 4a). Indeed, the error margin with even one fifth of all files (64k) is comparable to the correct sampling strategy with only 1024 samples. Thus, it is vital to apply the right sampling method. Therefore, the initial approach used to gather the test data as described in Section 3 is suboptimal.

Summary & Conclusions

In this paper, sampling techniques from statistics are applied to estimate data properties. These techniques are demonstrated to be useful approximate the proportions of scientific file types. It has been demonstrated that a random file scanner is not efficient to estimate quantities that are computed on file size. Instead, sampling with replacement and a probability equal to the proportion of file size leads to stable results. The tools which use such techniques can estimate properties of data robust without the need to analyze the huge data volumes of data centers. We will be working on such tools to evaluate the benefit of optimization strategies. More results are found in the full paper [5].

References

1. Nathanel Hübbe and Julian Kunkel. Reducing the HPC-Datastorage Footprint with MAFISC – Multidimensional Adaptive Filtering Improved Scientific data Compression. *Computer Science - Research and Development*, pages 231–239, 05 2013. DOI: 10.1007/s00450-012-0222-4.

2. Keren Jin and Ethan L Miller. The Effectiveness of Deduplication on Virtual Machine Disk Images. In *Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference*, page 7. ACM, 2009.

3. JWKJW Kotrlik and CCHCC Higgins. Organizational Research: Determining Appropriate Sample Size in Survey Research Appropriate Sample Size in Survey Research. *Information technology, learning, and performance journal*, 19(1):43, 2001.

4. Michael Kuhn, Konstantinos Chasapis, Manuel Dolz, and Thomas Ludwig. Compression By Default – Reducing Total Cost of Ownership of Storage Systems, 06 2014.

5. Julian M. Kunkel. Analyzing Data Properties using Statistical Sampling Methods – Illustrated on Scientific File Formats and Compression Features. In *High Performance Computing – ISC HPC 2016 International Workshops, Revised Selected Papers (to appear)*, volume 9945 of *Lecture Notes in Computer Science*. 2016.

6. Sriram Lakshminarasimhan, Neil Shah, Stephane Ethier, Seung-Hoe Ku, Choong-Seock Chang, Scott Klasky, Rob Latham, Rob Ross, and Nagiza F Samatova. ISABELA for Effective in Situ Compression of Scientific Data. *Concurrency and Computation: Practice and Experience*, 25(4):524–540, 2013. DOI: 10.1002/cpe.2887.

7. Solomon Desalegn Legesse. Performance Evaluation of File Systems Compression Features. Master's thesis, University of Oslo, 2014.

8. Jay Lofstead, Milo Polte, Garth Gibson, Scott Klasky, Karsten Schwan, Ron Oldfield, Matthew Wolf, and Qing Liu. Six Degrees of Scientific Data: Reading Patterns for Extreme Scale Science IO. In *Proceedings of the 20th international symposium on High performance distributed computing*, pages 49–60. ACM, 2011. DOI: 10.1145/1996130.1996139.

9. Uwe Schulzweida, Luis Kornblueh, and Ralf Quast. CDO User's guide: Climate Data Operators Version 1.6. 1, 2006.

10. Aviad Zuck, Sivan Toledo, Dmitry Sotnikov, and Danny Harnik. Compression and SSDs: Where and How? In *2nd Workshop on Interactions of NVM/Flash with Operating Systems and Workloads (INFLOW 14)*, Broomfield, CO, October 2014. USENIX Association.

DOI: 10.14529/jsfi160305

Spectral Domain Decomposition Using Local Fourier Basis: Application to Ultrasound Simulation on a Cluster of GPUs

Jiri Jaros[1], *Filip Vaverka*[1], *Bradley E. Treeby*[2]

The simulation of ultrasound wave propagation through biological tissue has a wide range of practical applications. However, large grid sizes are generally needed to capture the phenomena of interest. Here, a novel approach to reduce the computational complexity is presented. The model uses an accelerated k-space pseudospectral method which enables more than one hundred GPUs to be exploited to solve problems with more than 3×10^9 grid points. The classic communication bottleneck of Fourier spectral methods, all-to-all global data exchange, is overcome by the application of domain decomposition using local Fourier basis. Compared to global domain decomposition, for a grid size of $1536 \times 1024 \times 2048$, this reduces the simulation time by a factor of 7.5 and the simulation cost by a factor of 3.8.

Keywords: domain decomposition, ultrasound simulation, spectral methods, GPU, FFT, local Fourier basis.

Introduction

Accurately simulating the propagation of ultrasound waves through biological tissue has a large number of practical applications, including the physics-based simulation of diagnostic ultrasound images [1] and treatment planning for ultrasound therapy [2] (a more comprehensive list is provided in [3]). However, ultrasound simulation for these applications is computationally demanding due to the length scales involved, where the propagation length can be hundreds or thousands of times longer than the acoustic wavelength. In turn, this leads to very large domain sizes, in some cases with more than 100 billion grid points [2]. This puts the simulation times beyond clinically useful time-limits, even when using significant computational resources [4]. The overarching goal of this work is to maximise computational efficiency to minimise the wall-clock time needed to solve such large scale problems.

One of the biggest challenges in performing large-scale ultrasound simulations is the accumulation of numerical dispersion. In general, this can be overcome through the application of spectral methods, which can be considered memory minimising due to their exponential error convergence with grid density [5]. For wave problems, the k-space pseudospectral method is particularly efficient. This combines the spectral calculation of spatial gradients (using the Fourier collocation spectral method) with a dispersion-corrected finite difference scheme to integrate forward in time. This approach was first proposed in [6] and further developed in [7–9]. The parallel implementation of the k-space pseudospectral method has previously been described by several groups [2, 10–13]. Aside from a number of element-wise matrix operations, the most significant operations performed at each time step are multiple real-to-complex and complex-to-real 3D fast Fourier transforms (FFTs). The parallel efficiency of the method therefore depends primarily on the parallel efficiency of the FFT. Note, in the conventional pseudospectral time domain method, the FFTs are 1D. However, for the k-space method, the FFTs are 3D, as the dispersion correction step is applied in the spatial Fourier domain.

The biggest challenge in the calculation of the 3D FFT is a globally synchronising all-to-all data exchange. This is required to transpose the 3D matrix data, as the FFT cannot

[1]Faculty of Information Technology, Brno University of Technology, Brno, CZ
[2]Department of Medical Physics and Biomedical Engineering, University College London, London, UK

stride across data belonging to multiple processes. Despite the large amount of progress on optimizing the implementation of distributed FFTs, the inherent communication bottleneck still limits scaling efficiency. The distributed FFT implementations deployed on CPU clusters usually achieve scaling factors between 1.5 and 1.8 when the number of processing elements is doubled. Pippig [14] reported a comparative study of FFTW [15], PFFT [14], and P3DFFT [16] using an IBM Blue Gene machine. Similar investigations using Intel-Infiniband clusters were reported in [13, 17, 18]. In general, the majority of the execution time is spent in communication. For typical ultrasound simulations with grid sizes ranging from 512^3 to 2048^3 grid points, when distributed over more than 512 CPU cores, 50-90% of the execution time is wasted waiting for data exchanges [2].

The imbalance between communication and computation is even more striking when graphics processing units (GPUs) are used, as the raw performance of GPUs is an order of magnitude above current central processing units (CPUs). In addition, transfers over the peripheral component interconnect express (PCI-E) bus have to be considered as another source of communication overhead. The implementation proposed by Gholami [17], which is currently one of the most efficient, reveals the fundamental communication problem of distributed GPU FFTs. For an 1024^3 FFT calculated using 128 GPUs, the communication overhead accounts for 99% of the total execution time. Although the execution time reduces by 8.6× for a 32× increase in the number of GPUs (giving a parallel efficiency of 27%), this overhead may be acceptable in many applications.

One way to overcome the global communication imposed by the Fourier spectral method is to use a *local* Fourier basis as proposed by Israeli, *et al* [19]. This allows the evaluation of derivatives to be splitted into multiple coupled subdomains, where the Fourier transforms for each subdomain are computed independently, followed by the exchange of data in an overlap or halo region. The spectral accuracy is maintained by forcing the local domains to be periodic through multiplication of the local data by a bell function. The bell function is equal to one within the physical domain, and tapers to zero within the overlap region [20]. Using local, Founer basis rather than global ones, FFTs can have a significant impact on the computational performance of Fourier spectral methods. For example, Ding & Chen implemented a solution to Maxwell's equations using local Fourier basis [21]. For the simulation with 1024^3 grid points running on 32 CPUs, they reported reduction in communication time from 9.49 seconds per time step when using global FFTs, to 1.46 seconds per time step when using local Fourier basis with 32 subdomains. Similarly, Garbey, *et al* reported close to ideal weak scaling when using local Fourier basis to solve a combustion problem using up to 16 processors [22].

In the current work, domain decomposition using local Fourier basis is combined with the k-space pseudospectral method to allow the highly efficient simulation of ultrasound propagation using a cluster of 128 GPUs with grid sizes up to $1024 \times 1536 \times 2048$. The governing equations and their discretisation are discussed in Sec. 2, with the local decomposition introduced in Sec. 3. Details of the parallel implementation are given in Sec. 4, with numerical experiments presented in Sec. 5. Summary and discussion are then given in Sec. 6.

1. Pseudospectral Ultrasound Model

The physical problem considered here is the propagation of small amplitude acoustic waves through a homogeneous and lossless fluid medium. In this case, the governing equations are

given by a set of coupled first-order partial differential equations [23]

$$\frac{\partial \mathbf{u}}{\partial t} = -\frac{1}{\rho_0}\nabla p \ , \quad \frac{\partial \rho}{\partial t} = -\rho_0 \nabla \cdot \mathbf{u} \ , \quad p = c_0^2 \rho \ . \tag{1}$$

Here $\mathbf{u}$ is the acoustic particle velocity, p is the acoustic pressure, c_0 is the sound speed, and ρ_0 and ρ are the ambient and acoustic density, respectively. The governing equations are solved using the k-space pseudospectral method, where spatial gradients are computed using the Fourier collocation spectral method, and time integration is performed using a dispersion-corrected finite difference scheme [8]. Written in discrete form, the governing equations in Eq. (1) become [2, 9]

$$\begin{aligned}
\frac{\partial}{\partial \xi} p^n &= \mathcal{F}^{-1}\left\{ ik_\xi \, \kappa \, e^{ik_\xi \Delta\xi/2} \mathcal{F}\left\{ p^n \right\} \right\} \ , \\
u_\xi^{n+\frac{1}{2}} &= u_\xi^{n-\frac{1}{2}} - \frac{\Delta t}{\rho_0} \frac{\partial}{\partial \xi} p^n \\
\frac{\partial}{\partial \xi} u_\xi^{n+\frac{1}{2}} &= \mathcal{F}^{-1}\left\{ ik_\xi \, \kappa \, e^{-ik_\xi \Delta\xi/2} \mathcal{F}\left\{ u_\xi^{n+\frac{1}{2}} \right\} \right\} \ , \\
\rho_\xi^{n+1} &= \rho_\xi^n - \Delta t \rho_0 \frac{\partial}{\partial \xi} u_\xi^{n+\frac{1}{2}} \ , \\
p^{n+1} &= c_0^2 \sum_\xi \rho_\xi^{n+1} \ .
\end{aligned} \tag{2}$$

The first four equations are repeated for each Cartesian direction, where $\xi = x, y, z$. Here, $\mathcal{F}$ and $\mathcal{F}^{-1}$ denote the forward and inverse FFT over all three spatial dimensions, i is the imaginary unit, k_ξ is the wavenumber vector in the ξ direction, $\Delta\xi$ is the grid spacing in the ξ direction, Δt is the time step, and κ is the so-called k-space operator used to correct for numerical dispersion introduced by the finite difference time step [8]. The acoustic density (which is physically a scalar quantity) is artificially divided into Cartesian components to allow an anisotropic perfectly matched layer (PML) to be applied to model free-field conditions [24]. The exponential terms are spatial shift operators that allow the particle velocity to be evaluated on a staggered grid [8]. The superscripts n and $n+1$ denote the function values at the current and next time points, and $n - \frac{1}{2}$ and $n + \frac{1}{2}$ at time staggered points.

The implementation of the discrete equations requires the storage of thirteen real 3D matrices defined in the spatial domain and three real and one complex 3D matrix defined in the Fourier domain. These are used to store the current values of the acoustic variables, their derivatives, and three temporary matrices. For a single precision shared memory implementation, the memory usage can be estimated as

$$\text{memory usage [GB]} \approx \frac{16.5 \times \text{Nx} \times \text{Ny} \times \text{Nz}}{1024^3/4} \ , \tag{3}$$

(neglecting scalars, 1D arrays, and the code itself). At each time step, the operations performed consist of four forward and six inverse 3D FFTs, and around 100 element-wise matrix operations.

Previously, this type of model has been implemented using C++ and parallelised using either OpenMP, with simulations reported up to 1024 × 1024 × 1024 grid points [9], or MPI, with simulations reported up to 4096 × 2048 × 2048 grid points [2, 10, 11]. In both cases, the 3D FFTs are calculated over the entire 3D domain, which requires an all-to-all global communication for each FFT. Although reasonable scaling is observed, particularly when using hybrid OpenMP/MPI decomposition [13], ten all-to-all communications are still required per time step, which is a major bottleneck in performance [2].

2. Local Fourier Basis decomposition

2.1. Formulation

As described in Sec. , the motivation behind domain decomposition using local Fourier basis is to replace the global FFTs by local FFTs computed independently on a series of subdomains. The general approach as applied to the k-space pseudospectral method can be described as follows. First, the field variables and material parameters are divided across the subdomains, including an overlap region (halo) with a specified width (see fig. 1). Here, the so-called non-overlapped decomposition is used [25], and the subdomains are assumed to always be of an equal size. Second, for each subdomain, an independent version of the complete k-space pseudospectral model is run. The spatial gradients are calculated as normal using local Fourier basis, but before taking the FFT, the halo is exchanged and function values are multiplied by a bell function. This tapers to zero within the overlap region to enforce periodicity. Here, the erf-like bell function defined by Boyd is used [25]. This is equal to 1 within the physical domain and given by $H(x) = \frac{1}{2}(1 + \mathrm{erf}(Lx/\sqrt{1-x^2}))$ within the overlap region, where L is a scaling parameter, which in this case is set to 2. The discrete values for x within the overlap region are given by $x = -1, -1 + 2/(N-1), \ldots, 1$, where N is the size of the overlap in grid points.

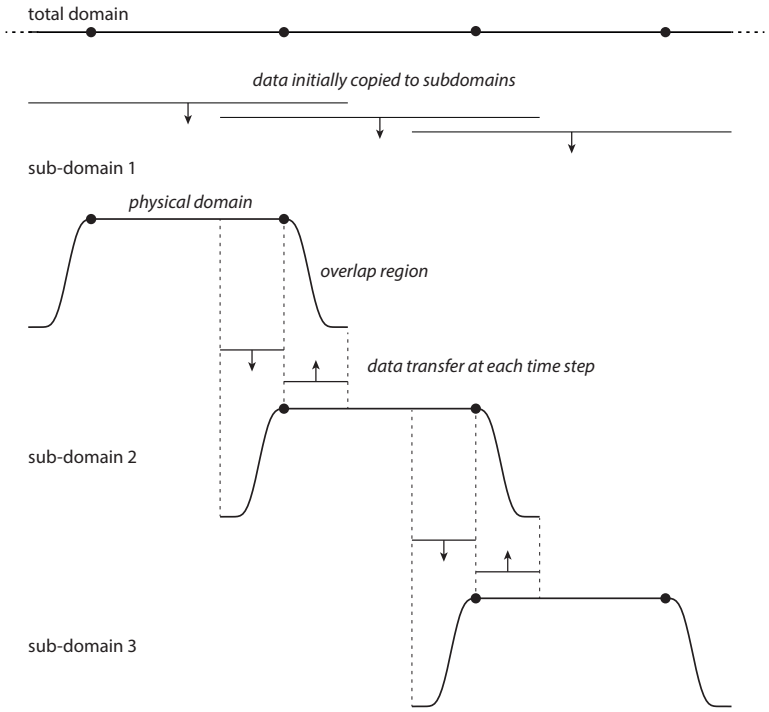


Figure 1. Schematic showing domain decomposition using local Fourier basis

2.2. Multidimensional decomposition

In 3D, there are several approaches to the decomposition of the global domain into subdomains, including 1D slab decomposition, 2D pencil decomposition, and 3D cube decomposition. The main differences are the number of interfaces a wave must travel through when traversing the grid in a given direction (this affects accuracy as discussed in Sec 2.3), the ratio between the halo and local subdomain size, and the number of data transfers that have to be performed [26]. The ratio between the halo and the local subdomain size improves with the dimensionality of

the decomposition. For a fixed number of subdomains, higher dimensionality implies a smaller amount of data that must be exchanged and consequently lower computational overhead. On the other hand, the number of direct neighbours grows with the dimensionality of the decomposition, i.e., there are 2, 8, and 26 direct neighbours for uniform 1D, 2D, and 3D decompositions, respectively. Since the interconnection bandwidth is not constant for all message sizes [27], in some cases it may be better to use 1D decomposition instead of 2D or 3D, even if a larger amount of data must be exchanged among a smaller number of neighbours. The impact of different decompositions on performance is discussed in Sec 4.3.

2.3. Accuracy

To test the accuracy of domain decomposition using local Fourier basis, a series of numerical experiments were conducted using a prototype CPU code. The tests consisted of propagating a plane wave along the grid axis (which reduces to a 1D problem) with the global domain divided into a given number of subdomains with a specified overlap width. The initial particle velocity was set to zero, and the initial pressure was set to be an impulsive pressure source. The spatial distribution of the pressure source was defined as a delta function (filtered by a Blackman window) positioned within the first subdomain. This generates a wave with broadband frequency content that smoothly decays up to the Nyquist limit [28]. For each test, a reference simulation using a global spectral method was also performed.

First, the dependence of the error on the width of the overlap region was examined. The total domain size was fixed at 512 grid points, and the overlap size varied from 8 to 32 grid points. The variation in L_∞ error compared to the reference simulation is shown in fig. 2(a). For an overlap width of 32 grid points, the error has not reached machine precision. However, the equivalent accuracy of the PML is only on the order of 10^{-3} to 10^{-4}, even with optimized parameters [29]. Given accuracy of the global solution is limited by the accuracy of the PML. It is sufficient to maintain a similar level of accuracy for the domain decomposition. Thus an overlap of 16 grid points was chosen, which gives an error less than 10^{-4} when using two subdomains. The change in the error for a fixed overlap size of 16 grid points with the total size of the local subdomain is shown in fig. 2(b). There is almost no change in the error for subdomain sizes from 32 to 1024 grid points, which means the size of the subdomains can be chosen to maximise computational efficiency.

Next, the dependence of the error on the number of domain cuts the wave must traverse was examined (for 1D decomposition, the number of domain cuts is one less than the number of subdomains). The total domain size was fixed at 2048 grid points with an overlap size of 16 grid points, and the number of subdomains was increased from 2 to 32 (in powers of 2). The variation in L_∞ error compared to the reference simulation is shown in fig. 2(c), and the error growth relative to using 2 subdomains is shown in fig. 2(d). The error increases linearly with the number of domain cuts the wave traverses, with a slope of $\sim$0.5. Thus, for typical sized problems (on the order of 2048 grid points in each dimension), up to 31 domain cuts (i.e., 32 subdomains if using 1D decomposition) can be used in each dimension with an overlap size of 16 grid points, and the error is still on the order of 10^{-3}. For 3D decomposition, this corresponds to 32,768 total subdomains (and in this case, GPUs). This means in practice, the level of achievable parallelism is not limited by the reduction in accuracy due to the use of local Fourier basis.

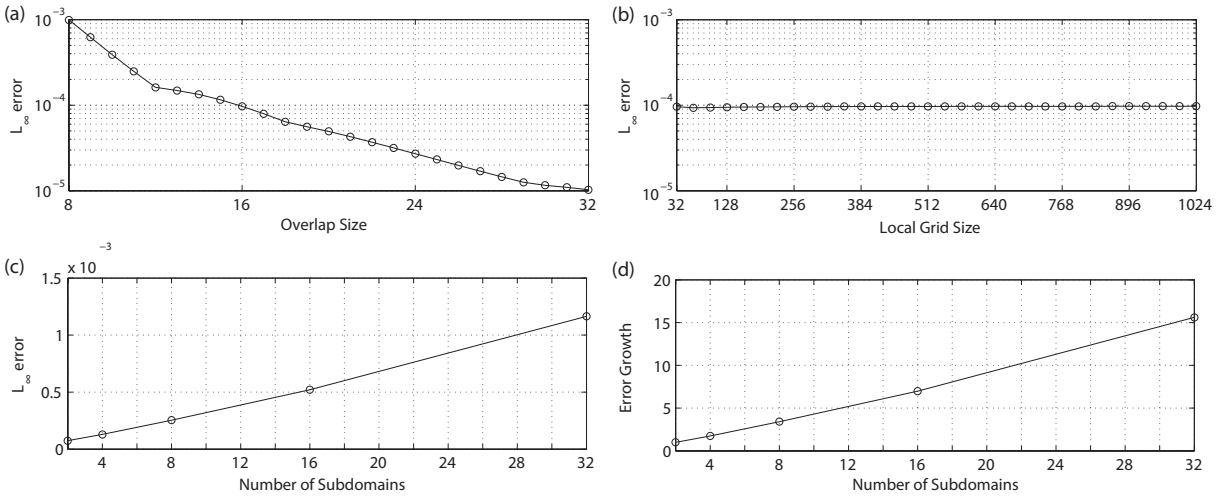


Figure 2. (a) Change in the L_∞ error with the overlap (halo) size for a fixed local domain size of 512 grid points. (b) Change in the L_∞ error with the local domain size for a fixed overlap (halo) size of 16 grid points. (c) Change in the L_∞ error with the number of domain cuts the wave must traverse for a total domain size of 2048 grid points and a fixed overlap size of 16 grid points. For 1D decomposition, the number of domain cuts is one less than the number of subdomains, where 2 subdomains corresponds to a local domain size of 1024 grid points, and 32 subdomains corresponds to a local domain size of 64 grid points. (d) Growth in the error with the number of subdomains relative to 2 subdomains (i.e., 1 domain cut)

3. Implementation

3.1. Communication framework

The numerical model described in Secs. 1 and 2.1 was implemented for multiple NVIDIA GPUs using MPI, C++, and CUDA. The implementation was divided into two components, the first responsible for the initial domain decomposition and periodic halo exchanges, and the second for performing calculations on each local subdomain. Starting with the communication framework, after the simulation is started, the number of subdomains and their organisation in 3D space is determined by parsing command line arguments. The framework supports any 1D, 2D, or 3D partition that fits into on-board GPU memory and meets the minimum size requirements. Each subdomain is assigned to an MPI process. In the case of 1D decomposition, the processes are grouped into an MPI communicator. If 2D or 3D decomposition is chosen, a virtual Cartesian topology is created and MPI is allowed to reorder ranks to preserve spatial locality between neighbouring subdomains. This is particularly useful while working on clusters with multiple GPUs per node.

The next step is GPU acquisition. Every MPI process (rank) inspects the configuration of the node being executed on, and chooses the first free GPU. This allows the framework to run on both slim and fat nodes with multiple GPUs, even in the case of non-uniform clusters (i.e., a mixture of nodes with a different number of integrated GPUs such as the Emerald cluster discussed in Sec. 4.1). The user (cluster batch scheduler) is responsible for assigning the correct number of ranks to individual nodes matching the number of integrated GPUs. The batch scheduler can also assign GPUs directly to ranks. If this feature is not supported by the scheduler and the GPUs are switched into exclusive process compute mode [30], the framework calculates the best assignment automatically and ensures mutual exclusion between ranks.

The execution proceed with the simulation setup. First, the simulation input file is opened and the simulation parameters are loaded. When the simulation domain size is determined, the framework calculates the size and position of the local subdomains, quantifies the size of the halo regions, and allocates all necessary data structures on both the CPU and GPU. The simulation data is then loaded from the input HDF5 file using parallel I/O and transferred into GPU memory.

The simulation time loop is divided into computation and communication phases. In total, there are four data exchanges per time step. These precede the gradient calculation for the acoustic particle velocity $\mathbf{u}$ in each Cartesian direction, and the gradient calculation for the acoustic pressure p (see Eq. (2)). The derivatives of the three spatial components of acoustic particle velocity can be calculated independently, which allows the MPI communications to be partially overlapped with the calculation (this is not possible for the calculation of the pressure gradient). During the data exchange, the halos are extracted from all three 3D matrices of velocity, packed into line-up buffers and downloaded to the CPU. This is done by repeated invocations of simple packing CUDA kernels followed by PCI-E transfers. This way the traffic over PCI-E is minimized. Next, the corresponding `MPI_Isends` and `MPI_Irecvs` are launched. The number of transfers depends on the decomposition chosen and varies between 4 (in the case of 1D decomposition) and 52 (in the case of full 3D decomposition). The execution only waits for the u_x halo to be delivered, uploads the halo back to the GPU, and replaces the appropriate data values. This is done by a PCI-E transfer followed by a CUDA unpack kernel. The calculation of $\frac{\partial}{\partial x}u_x$ is then started while the other four transfers proceed in the background. Thus this implementation partially hides two of three MPI communications.

3.2. Computation framework

The computation framework orchestrates all the necessary calculations in a simulation. It is divided into pre-processing, simulation time loop, data collection, and post-processing phases. The calculations are performed either as calls to the cuFFT library [31], or to custom CUDA kernels. Note, all calculations are performed by the GPU and the CPU only assists with the halo exchange and I/O operations. Since the size of the local subdomains has not been known in advance, several auxiliary variables are calculated during pre-processing, including the local wavenumber vectors, bell function, FFT shifts for staggered grids, etc [2]. The cuFFT library is then initialised and the FFT execution plans are created.

The simulation time loop then follows Eq. (2). The gradient calculations are performed by CUDA kernels which compute the required element-wise operations in both the spatial and Fourier domains. The kernels are organised into 1D CUDA grids composed of 1D CUDA blocks. The grid size is based on the actual number of CUDA multiprocessors (16 blocks per multiprocessor), and the block size is fixed to 256 threads. Every thread is responsible for processing multiple grid elements. The benefit of this solution is high occupancy and memory bandwidth. The same type of CUDA kernel is also used for halo packing and unpacking, as well as for sampled data aggregation and collection. The output data aggregation and post processing steps are described in more detail in [2].

4. Experimental results

4.1. Hardware description

The proposed implementation was deployed and evaluated on two GPU supercomputers, Emerald (e-Infrastructure South, UK) and Anselm (IT4Innovations national supercomputing center, CZ). Emerald is a heterogeneous GPU cluster consisting of several types of nodes equipped with different numbers and models of GPUs. Our allocation was limited to 128 NVIDIA Tesla M2090 cards with 6 GB of on-board memory. As error correction code (ECC) is switched on, the on-board memory capacity is reduced to approximately 5.4 GB. The GPUs are grouped in configurations of 3 or 8 per node, connected by PCI-Express 2.0. In the case of the 8-GPU configuration, pairs of GPUs share 16 PCI-E links. The CPU side is always comprised of two 6-core Westmere processors and 48 or 96 GB of RAM. The interconnection is provided by a 40 Gb/s half-duplex InfiniBand interconnect arranged into a fat-tree topology. The aggregated theoretical GPU performance is 170 TFLOPS in single precision, and the aggregated on-board memory is 768 GB.

Anselm consists of 209 compute nodes with a 40 Gb/s full-duplex InfiniBand interconnect arranged into a fat-tree topology. Each node integrates two 8-core Sandy Bridge CPUs and 64 GB of RAM. Twenty three nodes are equipped with one NVIDIA Kepler K20m GPU card with 5 GB of on-board memory and with ECC switched off. The GPUs are connected by PCI-Express 2.0. Our allocation was limited to 16 GPU cards. The aggregated GPU performance in single precision is 56 TFLOPS, and the aggregated on-board memory is 80 GB. For both systems, the GPU simulation code was compiled with the Intel compiler 2015, Intel MPI 5.0, NVIDIA CUDA 7.5, and HDF5 1.8.16. For comparison, a CPU implementation using global domain decomposition was used [2]. This code was compiled with the same tool chain, in addition to the FFTW 3.3.4 library.

4.2. Strong scaling

Several numerical experiments were performed to assess the performance of the multi-GPU implementation. First, strong scaling was assessed using domain sizes from 256^3 to $1024 \times 1024 \times 2048$ grid points with an overlap size (halo width) of 16 grid points. The problem sizes were limited by the aggregated amount of on-board memory and restrictions imposed by the smallest subdomain size. For Emerald, the scaling was investigated using up to 128 GPUs on the full range of simulation sizes (fig. 3(a)). Since our allocation on Anselm was limited to 16 GPUs, the largest domain size tested was $512 \times 512 \times 1024$ grid points (fig. 3(b)). The domain was partitioned over all three axes into a uniform number of subdomains starting from $1 \times 1 \times 1$ (i.e., running on a single GPU) up to $4 \times 4 \times 8$ and $2 \times 2 \times 4$ for the largest domain sizes on Emerald and Anselm, respectively.

Overall, the code achieves a reasonable scalability. On Emerald, for larger domain sizes the best parallel efficiency is approximately 27% when increasing from 8 to 128 GPUs, 34% from 16 to 128 GPUs, and 55% from 32 to 128 GPUs. In comparison, the strong scaling on Anselm reaches better values of parallel efficiency, reaching approximately 47% when increasing from 2 to 16 GPUs, 56% from 4 to 16 GPUs, and 85% from 8 to 16 GPUs. These levels of parallel efficiency are caused by a combination of multiple factors:

1. As the domain size grows, there is an increase in the number of neighbours the halo must be exchanged with. Since the number of GPUs is initially small, the decomposition is first done

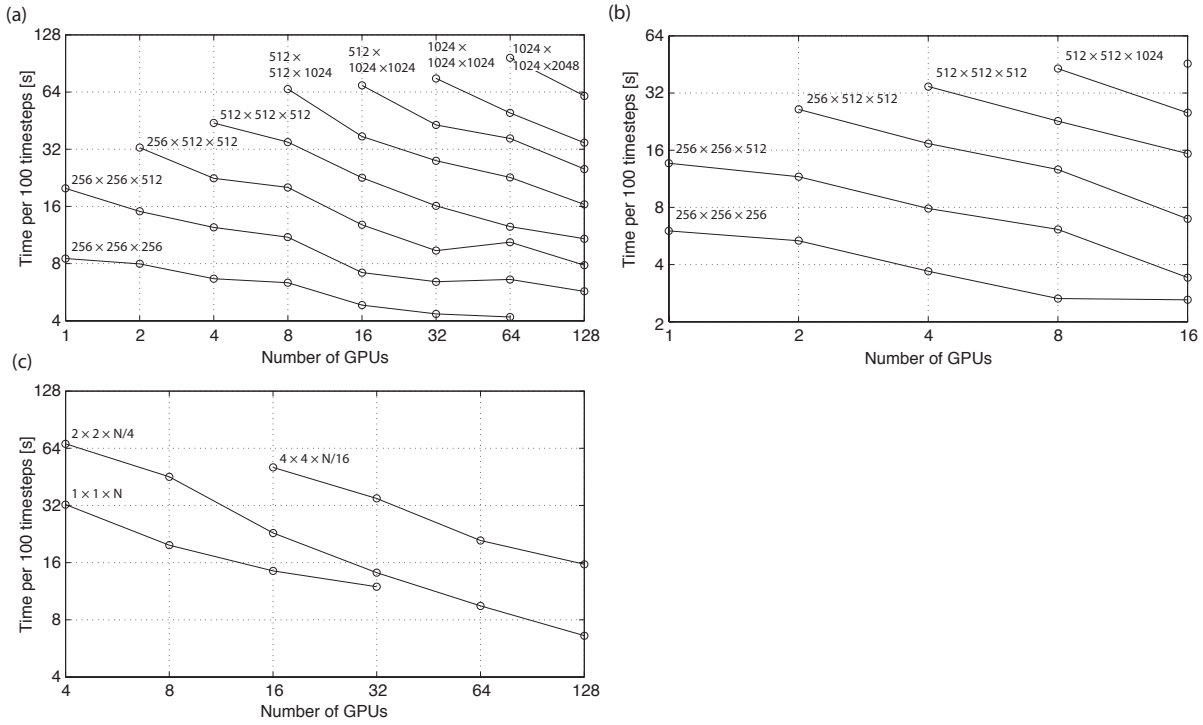


Figure 3. Strong scaling plots for (a) Emerald with 1-128 GPUs, and (b) Anselm with 1-16 GPUs. (c) The influence of different decomposition of the simulation domain of $256 \times 256 \times 2048$ on the strong scaling observed on Emerald. One-dimensional decomposition is compared with half and full 3D decomposition with 2, 11, and 26 neighbours, respectively

in 1D (2 GPUs), followed by 2D (4 GPUs) and 3D (8 GPUs) with only a single neighbour in each dimension. This situation will be referred to as half decomposition. For 16 and more GPUs, the decomposition turns into the full version with neighbours on both sides. This is done first in the x direction (16 GPUs), followed by the y direction (32 GPUs). The first complete full decomposition is employed for 64 GPUs, where the decomposition is $4 \times 4 \times 4$. The growing number of neighbours has a direct impact on the number of extraction/injection CUDA kernels, as well as the number of MPI communications to be performed.

2. On Emerald, the GPUs are packed into fat nodes sharing PCI-E links and the network adapter. The situation is worse in the case of 8-GPU nodes, where pairs of GPU cards share 16 links and the PCI-E bandwidth is halved. Moreover, 4 GPUs are connected to a single CPU socket creating contention in RAM.
3. Since the amount of on-board memory is limited, this also limits the total domain sizes. In the finest decomposition, the local subdomain only contains $64 \times 64 \times 64$ grid points, which makes the calculation time very small compared to the communication time. Moreover, for such a small subdomain, the halo accounts for 70% of the grid points in the local subdomain. The situation improves as the size of the local subdomains is increased. Unfortunately, the biggest subdomain that fits into on-board memory is approximately $256 \times 256 \times 512$. In this case, the halo accounts for around 25% of the local grid points.

Finally, comparing both systems, Anselm reaches almost twice the performance of Emerald. This is due to the combined effects of newer GPUs with higher performance and on-board memory bandwidth, ECC being switched off, and only a single GPU per node.

4.3. Decomposition comparison

Next, the influence of the domain decomposition shape on strong scaling was investigated on Emerald, using a global domain size of $256 \times 256 \times 2048$, a halo width of 16 grid points, and 4 to 128 GPUs. A 1D decomposition was used, where the domain is cut over the longest dimension into N partitions, with N corresponding to the number of GPUs. For comparison, a half 3D decomposition with 11 neighbours cut into $2 \times 2 \times N/4$, and a full 3D decomposition with 26 neighbours cut into $4 \times 4 \times N/16$ were also tested. As shown in fig. 3(c), the impact on performance is significant. The additional communication overhead arising from using a higher dimensionality decomposition with an increased number of neighbours directly translates into a performance drop. An obvious conclusion is that wherever possible, a 1D decomposition is preferred. When this is not possible due to an unacceptable level of numerical accuracy (the numerical error scales with the number of cuts along an axis as discussed in Sec. 2.3) or the subdomains being excessively small, a half 3D decomposition is preferred. Compared to the full 3D decomposition, the half decomposition reduces the simulation time by a factor of at least 2, which correlates with the reduced number of neighbours.

4.4. Simulation time breakdown

Next, the composition of the simulation time was investigated. Only the simulation loop was considered, as the initialisation, pre-processing, and post-processing phases usually take on the order of minutes, while realistic simulations may run for many hours. Figure 4 shows the simulation time breakdown for a domain size of $256 \times 256 \times 1024$, a halo width of 16 grid points, and 1D domain decomposition executed on Emerald and Anselm with 2 to 16 GPUs. Apart from faster execution on Anselm (due to faster GPUs), a difference in the PCI-E and MPI overhead may be observed. Although not clearly visible, the PCI-E latency on Emerald is almost 25% higher due to main memory congestion and shared PCI-E links. In addition, a higher MPI latency on Emerald is clearly noticeable. This is because Emerald only supports half-duplex, which decreases the MPI bandwidth by a factor of two. Furthermore, all inter-node communications are done via the main memory, which becomes the ultimate bottleneck. For the highest number of GPUs, where the local domain size is $256 \times 256 \times 64$, the percentage of time spent performing calculations, communications using PCI-E, and communications using MPI is 32%, 10%, and 58% for Emerald, and 40%, 17%, and 43% for Anselm, respectively. In comparison, previous implementations using global domain decomposition have reported the time spent performing calculations is below 1% on a GPU cluster [17], and 30% on a CPU cluster [13].

4.5. Influence of halo width

The influence of halo width (overlap size) on the communication overhead was investigated on Emerald using a simulation size of 512^3 grid points partitioned over all three dimensions with a halo width of 8, 16, or 32 grid points. The simulations were executed on 4 to 128 GPUs, with the time break down shown in fig. 5. The increase in the PCI-E and MPI overhead when the halo becomes larger is evident. When the halo size is 8 grid points, there is only a small overhead. However, when the overlap is increased to 32 grid points, the communication overhead prevents the code from scaling beyond 16 GPUs. Although the PCI-E latency remains at promising levels and scales with the number of GPUs, the MPI latency is the ultimate bottleneck. The

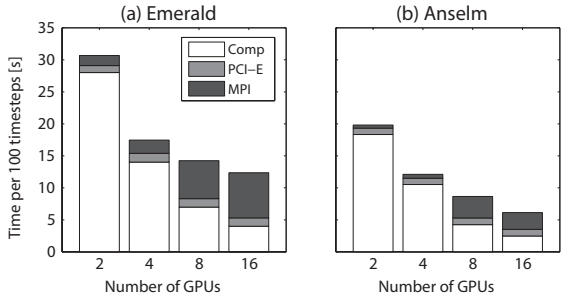


Figure 4. Breakdown of the execution time for a simulation domain size of $256 \times 256 \times 1024$ comprising of the computation part, MPI transfers between nodes, and PCI-E transfers between CPU and GPU

slight increase in the computation time across the three overlap sizes is due to the increase in local subdomain size with the halo size. The rise in MPI overhead as the number of GPUs is increased between 4 and 16 can be attributed to the transition from half 3D decomposition to full 3D decomposition. A halo width of 16 grid points was ultimately chosen as an acceptable compromise between performance and accuracy.

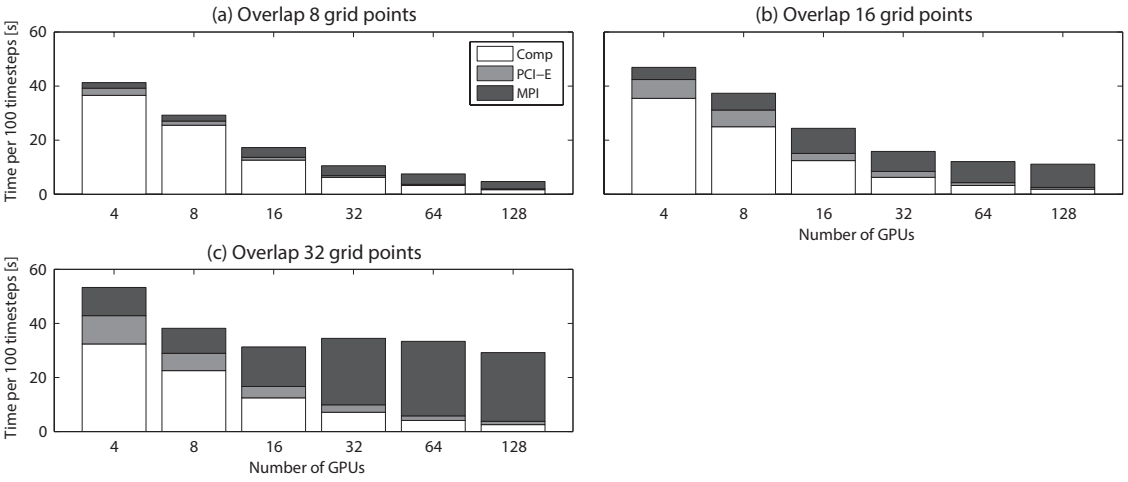


Figure 5. The influence of the overlap (halo) width on the overhead comprising of MPI and PCI-E transfers. The investigation was conducted on Emerald using a simulation size of 512^3, a halo width of 8, 16 and 32, and between 4 and 128 GPUs

4.6. Comparison of GPU and CPU

Finally, the GPU implementation using local Fourier decomposition was compared with an existing CPU implementation using global decomposition [13]. Several benchmark simulations were performed on Emerald and Anselm using three domain sizes (256^3, 512^3 and 1024^3) as shown in fig. 6. Here the horizontal axis corresponds to either the number of GPUs, or the number of CPU nodes (each of which integrates 16 processor cores). This grouping is used to estimate the simulation cost, which is charged per node on Anselm, regardless of whether the GPU is used. In comparison, Emerald has a different charge policy, with higher prices charged per GPU.

Figure 6(a) reveals that for small domain sizes, Anselm's GPUs are much faster than the CPU implementation when the number of nodes employed is small. Here, the simulation cost

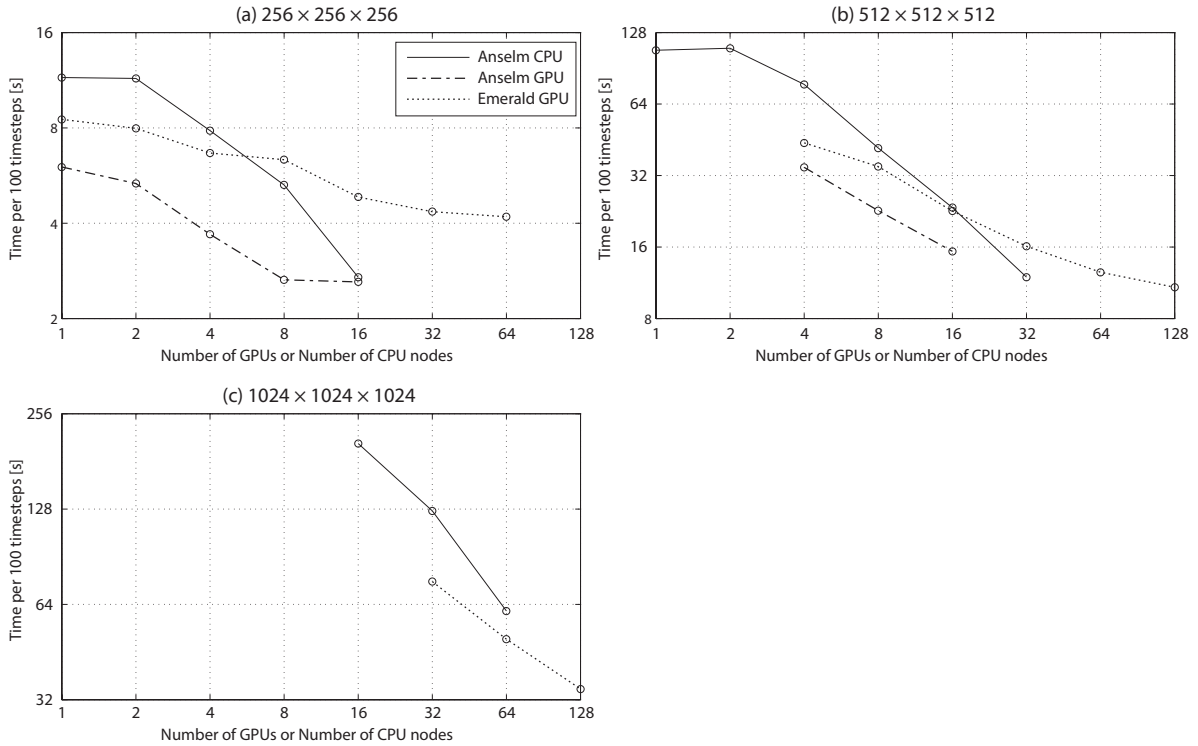


Figure 6. Performance comparison between local Fourier basis decomposition running on GPUs (Emerald and Anselm) and global domain decomposition running on CPUs (Anselm). In the case of the CPU implementation, the number of GPUs translates to the number of nodes, each of which contains 16 CPU cores

can be significantly reduced by utilising GPUs. The compute times for Anselm's GPUs and CPUs meet at 16 GPUs/nodes, where the local subdomains are extremely small. Emerald's GPUs seem to be too slow for such a small domain, where the communication is dominant. Figure 6(b) shows the same results for a larger domain size. Here, Anselm's GPUs outperform the comparable number of CPU cores. When the number of nodes is doubled (32 nodes or 512 CPU cores against 16 GPUs), the CPU cluster is faster by a factor of 1.28, however, for a doubled price. Emerald's GPUs beat Anselm when all 128 GPUs are used in the computation. The last benchmark shown in fig. 6(c) illustrates the benefits of the GPU implementation for larger domain sizes. Here 128 GPUs are faster than a cluster of 1024 CPU cores in 64 nodes by a factor of 1.76. Considering these 128 GPUs could be packed in 16 GPU nodes, this is a significant result. Note, the CPU code is limited by 1D slab decomposition so it is not possible to employ more cores than 256, 512 and 1024 for the domain sizes tested.

4.7. Production simulation

To show the impact of the proposed multi-GPU implementation on the type of ultrasound simulations used for treatment planning in focused ultrasound surgery, a comparison is given of the execution time and the financial aspects of running a single simulation using a grid size of $1536 \times 1024 \times 2048$ with 48,000 time steps performed as a part of the characterisation of a high-intensity focused ultrasound (HIFU) transducer used to treat prostate cancer. The output of such a simulation is given in fig. 7, which shows the maximum steady state acoustic pressure in a 2D plane in front of the transducer. Table 1 illustrates that a cluster of 128 GPUs is able

to deliver the simulation result in 9 hours and 29 minutes for the price of 426 USD (calculated based on the Emerald charge rate of 35.13c per GPU hour). Comparing this with the best price performance on the CPU cluster, the simulation can be completed in 3 days for 1,623 USD, or 2 days and 5 hours for 2,395 USD (calculated based on the Anselm charge rate of 8.8c per core hour). If price is the primary concern, as many tens of simulations are usually performed during any particular study, the GPU cluster can reduce the simulation time by a factor of 7.5 and the simulation cost by a factor of 3.8. The ultimate conclusion is that a GPU cluster is much a better solution for this type of simulation.

Table 1. Simulation time and cost when running a production simulation on Emerald with 96 and 128 GPUs, or Anselm with 128, 256, 512 CPU cores.

	Simulation Time	Simulation Cost
96 GPUs	14h 9m	$475
128 GPUs	9h 29m	$426
128 CPU cores	6d 18h	$1,826
256 CPU cores	3d 0h	$1,623
512 CPU cores	2d 5h	$2,395

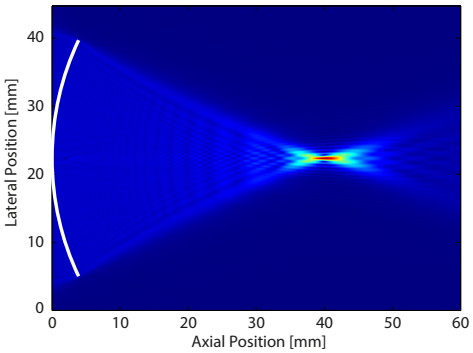


Figure 7. Pressure field from a prostate ultrasound transducer simulated using a domain size of $1536 \times 1024 \times 2048$ grid points ($45 \times 30 \times 60$ mm) with 48,000 time steps (60 μs) calculated in 9 hours and 29 minutes on 128 GPUs

Conclusion

This paper has presented a novel multi-GPU implementation of the Fourier spectral method using domain decomposition based on local Fourier basis [19]. The fundamental idea behind this work is the replacement of the global all-to-all communications introduced by the FFT (used to calculate spatial derivatives) by direct neighbour exchanges. By doing so, the communication burden can be significantly reduced at the expense of a slight reduction in numerical accuracy. The accuracy is shown to be dependent on the overlap (halo) size and independent on the local domain size. And to increase linearly with the number of domain cuts an acoustic wave must traverse. For an overlap (halo) size of 16 grid points, the error is on the order of 10^{-3}, which is comparable to the error introduced by the PML. Consequently, the level of parallelism achievable in practice is not limited by the reduction in accuracy due to the use of local Fourier basis.

Strong scaling results demonstrate that the code scales with reasonable parallel efficiency, reaching 50% for large simulation domain sizes. However, the small amount of on-board memory ultimately limits the global domain size for a given number of GPUs. 1D decomposition is shown to be the most efficient unless the local subdomain becomes too thin. Beyond, it is useful to exploit 2D or half 3D decomposition with only a single neighbour in a given direction to limit the number of MPI transfers. An overlap size of 16 grid points is shown to be a good trade off between speed and accuracy, with larger overlaps becoming impractical due to the overhead imposed by large MPI transfers. Compared to the CPU implementation using global domain decomposition, the GPU version is always faster for an equivalent number of nodes. For production simulations executed as part of ultrasound treatment planning, the GPU implementation reduces the simulation time by a factor of 7.5 and the simulation cost by a factor of 3.8. This is a promising result, given the GPUs utilized are now almost decommissioned.

In future, the code will be extended to model nonlinear wave propagation in heterogeneous media, as considered in [2]. The implementation could also be further improved by exploiting additional opportunities for overlapping communication and computation. First, the PCI-E and MPI communication could be overlapped. Second, the possibility of peer-to-peer communication among GPUs within the same node could be explored. This feature has the potential to eliminate expensive intra-node MPI communications. Third, the CPU could be utilized for additional executions, for example, assigning a subdomain to the idle CPU cores. Finally, multiple subdomains of different sizes could be executed on a single GPU, which might allow the communication on one subdomain to be overlapped while performing calculations on the others.

The project is financed from the SoMoPro II programme. The research leading to this invention has acquired a financial grant from the People Programme (Marie Curie action) of the Seventh Framework Programme of EU according to the REA Grant Agreement No. 291782. The research is further co-financed by the South-Moravian Region. This work reflects only the author's view and the European Union is not liable for any use that may be made of the information contained therein. This work was also supported by the research project "Architecture of parallel and embedded computer systems" Brno University of Technology (FIT-S-14-2297, 2014-2016), the Engineering and Physical Sciences Research Council, United Kingdom (grant numbers EP/L020262/1 and EP/M011119/1), and the Ministry of Education, Youth and Sports, Czech Republic (Large Infrastructures for Research, Experimental Development and Innovations project "IT4Innovations National Supercomputing Center - LM2015070"). The work presented here made use of Emerald, a GPU-accelerated High Performance Computer, made available by the Science & Engineering South Consortium operated in partnership with the STFC Rutherford-Appleton Laboratory. The authors would like to thank Alistair Rendell and Beau Johnston for useful contributions to early versions of the k-Wave GPU kernels.

References

1. G. F. Pinton, J. Dahl, S. Rosenzweig, and G. E. Trahey, "A heterogeneous nonlinear attenuating full-wave model of ultrasound," *IEEE Trans. Ultrason. Ferroelectr. Freq. Control*, vol. 56, no. 3, pp. 474–488, 2009. DOI: 10.1109/TUFFC.2009.1066.

2. J. Jaros, A. P. Rendell, and B. E. Treeby, "Full-wave nonlinear ultrasound simulation on distributed clusters with applications in high-intensity focused ultrasound," *Int. J. High Perf. Comput. Appl.*, vol. 30, no. 2, pp. 137–155, 2016.

3. J. Gu and Y. Jing, "Modeling of wave propagation for medical ultrasound: a review," *IEEE Trans. Ultrason. Ferroelectr. Freq. Control*, vol. 62, no. 11, pp. 1979–1992, 2015.

4. K. Okita, R. Narumi, T. Azuma, S. Takagi, and Y. Matumoto, "The role of numerical simulation for the development of an advanced HIFU system," *Comput. Mech.*, vol. 54, no. 4, pp. 1023–1033, 2014.

5. J. P. Boyd, *Chebyshev and Fourier Spectral Methods.* Mineola, New York: Dover Publications, 2001.

6. N. N. Bojarski, "The k-space formulation of the scattering problem in the time domain," *J. Acoust. Soc. Am.*, vol. 72, no. 2, pp. 570–584, 1982.

7. T. D. Mast, L. P. Souriau, D. L. Liu, M. Tabei, A. I. Nachman, and R. C. Waag, "A k-space method for large-scale models of wave propagation in tissue.," *IEEE Trans. Ultrason. Ferroelectr. Freq. Control*, vol. 48, no. 2, pp. 341–354, 2001.

8. M. Tabei, T. D. Mast, and R. C. Waag, "A k-space method for coupled first-order acoustic propagation equations," *J. Acoust. Soc. Am.*, vol. 111, no. 1, pp. 53–63, 2002.

9. B. E. Treeby, J. Jaros, A. P. Rendell, and B. T. Cox, "Modeling nonlinear ultrasound propagation in heterogeneous media with power law absorption using a k-space pseudospectral method," *J. Acoust. Soc. Am.*, vol. 131, no. 6, pp. 4324–4336, 2012.

10. M. I. Daoud and J. C. Lacefield, "Distributed three-dimensional simulation of B-mode ultrasound imaging using a first-order k-space method.," *Phys. Med. Biol.*, vol. 54, no. 17, pp. 5173–5192, 2009.

11. J. C. Tillett, M. I. Daoud, J. C. Lacefield, and R. C. Waag, "A k-space method for acoustic propagation using coupled first-order equations in three dimensions," *J. Acoust. Soc. Am.*, vol. 126, no. 3, pp. 1231–1244, 2009.

12. J.-L. Vay, I. Haber, and B. B. Godfrey, "A domain decomposition method for pseudo-spectral electromagnetic simulations of plasmas," *J. Comput. Phys.*, vol. 243, pp. 260–268, 2013.

13. J. Jaros, V. Nikl, and B. E. Treeby, "Large-scale Ultrasound Simulations Using the Hybrid OpenMP/MPI Decomposition," in *Proceedings of the 3rd International Conference on Exascale Applications and Software*, pp. 115–119, Association for Computing Machinery, 2015.

14. M. Pippig, "PFFT-An extension of FFTW to massively parallel architectures," *SIAM J. Sci. Comput.*, vol. 35, no. 3, pp. C213–C236, 2013.

15. M. Frigo and S. G. Johnson, "The Design and Implementation of FFTW3," *Proceedings of the IEEE*, vol. 93, no. 2, pp. 216–231, 2005.

16. D. Pekurovsky, "P3DFFT: A Framework for Parallel Computations of Fourier Transforms in Three Dimensions," *SIAM J. Sci. Comput.*, vol. 34, no. 4, pp. C192–C209, 2012.

17. A. Gholami, J. Hill, D. Malhotra, and G. Biros, "AccFFT: A library for distributed-memory FFT on CPU and GPU architectures," *arXiv*, p. arXiv:1506.07933, 2015.

18. K. Czechowski, C. Battaglino, C. McClanahan, and K. Iyer, "On the Communication Complexity of 3D FFTs and its Implications for Exascale," in *Proceedings of International Supercomputing Conference*, ACM, 2012. DOI: 10.1145/2304576.2304604.

19. M. Israeli, L. Vozovoi, and A. Averbuch, "Spectral multidomain technique with local Fourier basis," *J. Sci. Comput.*, vol. 8, no. 2, pp. 135–149, 1993.

20. J. P. Boyd, "Asymptotic fourier coefficients for a C$^\infty$ bell (smoothed-"top-hat") & the Fourier extension problem," *J. Sci. Comput.*, vol. 29, no. 1, pp. 1–24, 2005. DOI: 10.1007/s10915-005-9010-7.

21. M. Ding and K. Chen, "Staggered-grid PSTD on local Fourier basis and its applications to surface tissue modeling.," *Optics Exp.*, vol. 18, no. 9, pp. 9236–9250, 2010.

22. M. Garbey and D. Tromeur-Dervout, "Parallel Algorithms with Local Fourier Basis," *J. Comp. Phys.*, vol. 173, pp. 575–599, 2001.

23. A. D. Pierce, *Acoustics: An Introduction to its Physical Principles and Applications*. New York: Acoustical Society of America, 1989.

24. J.-P. Berenger, "Three-dimensional perfectly matched layer for the absorption of electromagnetic waves," *J. Comput. Phys.*, vol. 127, no. 2, pp. 363–379, 1996.

25. J. P. Boyd, "A Comparison of Numerical Algorithms for Fourier Extension of the First, Second, and Third Kinds," *J. Comput. Phys.*, vol. 178, no. 1, pp. 118–160, 2002.

26. M. J. Quinn, *Parallel Programming in C with MPI and OpenMP*. McGraw-Hill Education Group, 2003.

27. HPC Advisory Council, "Interconnect Analysis: 10GigE and InfiniBand in High Performance Computing," tech. rep., HPC Advisory Council, 2009.

28. B. E. Treeby and B. T. Cox, "A k-space Greens function solution for acoustic initial value problems in homogeneous media with power law absorption," *J. Acoust. Soc. Am.*, vol. 129, no. 6, pp. 3652–3660, 2011.

29. J. L. Robertson, B. T. Cox, and B. E. Treeby, "Quantifying numerical errors in the simulation of transcranial ultrasound using pseudospectral methods," in *IEEE Int. Ultrason. Symp.*, pp. 2000–2003, 2014.

30. NVIDIA, "CUDA Toolkit Documentation v7.5," tech. rep., NVIDIA, 2015.

31. NVIDIA, "cuFFT Library User's Guide," tech. rep., NVIDIA, 2015.

DOI: 10.14529/jsfi160306

HCA aware Parallel Communication Library: A feasibility study for offloading MPI requirements

***Kedar Kulkarni*[1], *Shreeya Badhe*[1], *Geetanjali Gadre*[1]**

Message Passing Interface (*MPI*) is a standardized message passing system, independent of underlying network, and the most widely used parallel programming paradigm. The communication library should make full use of the Host Channel Adapter (*HCA*) characteristics to maximize performance of the HPC cluster. The communication library may not able to take full advantage of the underlying network adapter, if the library is made generalized. This can have a significant impact on the performance.

Our primary goal is to develop a network dependent message passing library called a Parallel Communication Library (*PCL*) that will exploit C-DAC's proprietary PARAMNet HCA [1] features for efficient message transfer. Using PCL, we intend to observe the feasibility of the network and performance enhancement for additional features. The objective is to carry out different trials by implementing additional features and analyze the implications which would give us more insight towards suitability of transport offload/onload mechanism. This experimentation would give us feedbacks for designing the next generation architecture.

Keywords: MPI, HCA, Communication Pattern Offloading, High Performance Networks, Communication Library.

Introduction

Designing a high performance network is a critical and an arduous task. It involves work at multiple layers. The work is carried out at both software and hardware front. The software stack involves development of driver, communication library and message passing library. The hardware development is mainly focused on low latency and high bandwidth data transfer, and providing support for multiple protocols.

Today, typical High Performance Computing (*HPC*) clusters are commodity based clusters that employ OFED [2] software. OFED software provides interface to many applications regardless of underlying Host Channel Adapter (*HCA*). OFED software offers abstraction to the MPI layer and handles all low level activities such as connection management, basic data structures for connections, etc. Similarly, due to the communication stack hierarchy HCA becomes unaware of end application communication pattern. Typically, proprietary network interconnect developer uses OFED software to run MPI based applications. HCA has to provide various features in order to support OFED, such as unreliable datagram service to use OpenSM. Furthermore, due to the standard and generic structure of the communication stack it is possible that the upper layers may not exploit any additional feature provided by HCA that can improve the performance of the entire HPC cluster.

Message Passing Interface (*MPI*) [3] has become the de facto standard for writing parallel applications in HPC domain. To increase the performance of a HPC cluster, the communication library should exploit all HCA features efficiently. It may not be inappropriate to say that, at least in case of proprietary networks that use standard software stack, the upper layers do not comprehensively use of HCA features. This arises as a result of intricacy in modifying the complete software stack.

[1]Centre For Development of Advanced Computing, Bangalore, India

Our primary goal is to develop a network dependent message passing library called a Parallel Communication Library (*PCL*) for PARAMNet HCA. Using PCL, we intend to observe the feasibility of the network with support of additional features. This would help us to evolve the next generation architecture. Furthermore, any performance improvement observed would help us in contributing to the entire HPC community.

1. Scope

Traditional HPC communication stack is a multi-layer architecture. The multi-layer architecture creates a layer of abstraction between an underlying network and upper communication libraries. One peculiar problem with this approach is the communication library will not be able to use additional features provided by an underlying HCA, even though making use of the additional features could improve the performance. To extract maximum performance out of the underlying network, the HCA and the communication library should work in complementary manner. This paper describes a network dependent library, PCL, and its communication architecture for C-DAC's PARAMNet HCA. Our goal is to develop a library that can provide MPI like interface to the application, while exploring how these calls can be effectively implemented using PARAMNet HCA features. The purpose for developing our own library is to study the feasibility of HCA features that can be made available to the upper layers. Modifying standard communication library is an intricate process. It is easy to develop the library and support only necessary features. Using this approach, we can easily incorporate additional features in HCA and make them visible to an application using PCL.

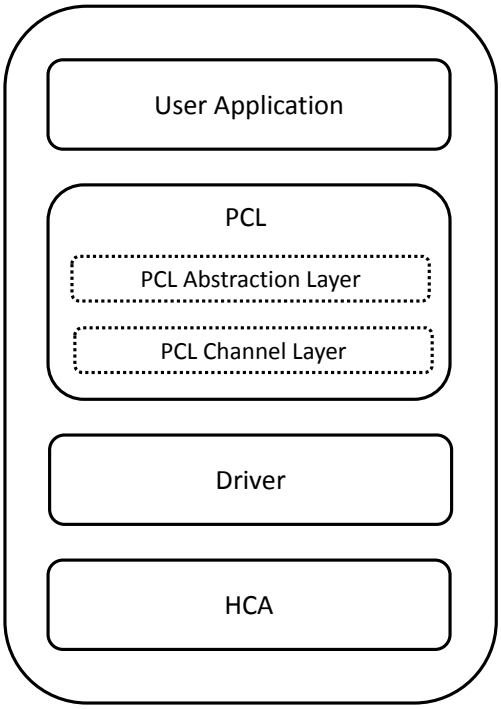


Figure 1. Communication Stack

2. Proposed Framework

The structure of PCL developed for PARAMNet HCA is shown in fig. 1. As shown in the figure PCL is primarily divided into two parts: the PCL abstraction layer and the PCL channel layer. The functionality of main components of PCL is as follows:

2.1. PCL Abstraction Layer

The PCL Abstraction layer provides application programming interface (*API*) for PCL. We have designed this layer to provide an interface similar to that of MPI. This layer is mainly responsible for following set of responsibilities:

- Handling the communication Pattern
  This layer handles the communication pattern that decides whether to use standard communication course or to make use of the additional features. For example, this layer can decide for small message whether to make use a low latency communication path or standard eager protocol.
- Managing connection data base
  In PCL application, each process is assigned a rank. PCL abstraction layer creates a data base for each rank. This data base holds all the information required for further data transfer, such as each rank is mapped to which hardware resources (*End-Points, Completion Queues, etc*).
- Buffer Management for Eager Protocol
  PCL supports eager protocol for message transfer. Allocation of buffers and managing these buffers are handled by PCL abstraction layer. The size of Eager buffer is programmable and controlled by *pcl_eager_threshold*.

Like MPI, the PCL works on reliable connection service. Therefore, PCL abstraction layer created all end-points in reliable connection mode. Along with reliable connection end-points, PCL abstraction layer also reserves an end-point per rank in the unreliable connection mode.

2.2. PCL Channel Layer

The PCL Channel layer handles all the communication with PARAMNet HCA. This layer handles responsibility such as posting work requests, creating an end-point. This layer hides low level details such as depth of work-queues, structure of work requests, end-points, completion queues and completion entry from the upper layer. This layer also translates hardware completion format into simpler format.

Along with these two layers a bash script, *pcl_exec*, is developed to work in background to assist to run an application on PCL. It also assists PCL in its working. This bash script will work as a daemon. It will spawn PCL processes, as user requests, on the nodes that are part of the cluster. Furthermore, it will bind each process to distinct core while spawning PCL processes.

Presently, we have implemented following functionalities:

- *pcl_init*
  This module initializes the setup, establishes connection amongst all the ranks, fill the receive work-queues with work requests in order to make every rank ready to accept data. This module also allocates eager buffers and pin those buffers.

- *pcl_finalise*
  This module is called for destroying all the HCA resources reserved during run time. Also, it frees all the eager buffers.
- *pcl_send*
  This module sends user buffer data to the destined remote rank. The internal implementation decides whether to use Eager or Rendezvous mode. Although, Eager mode is operational presently, and Rendezvous mode is under testing phase.
- *pcl_recv*
  This module polls on the receive data and after successful reception of data it copies intended data from eager buffer to application buffer.
- *pcl_barrier*
  The barrier operation is performed across all processes. It blocks until all processes have reached this routine. The detail implementation is explained in the next section.

3. Barrier Optimization

The Barrier function blocks the caller until all group members have called it. The function does not return on any process until all group processes have called the function. Barrier is been used by the applications quite frequently. The data transferred for barrier is very small as MPI transfers just header information. Using PARAMNet HCA feature, we have optimized barrier functionality for PCL. The barrier payload resides in the host memory. In typical implementation of communication stack, the communication library posts barrier call. For the lower layer this call is another data transfer requests, it simply post send work requests, which specify barrier data transfer request, to the HCA. The HCA has to read this data from host memory and then forward it to the destination. Typical PCI Express [4] latency for smaller host memory read requests is quite high due to round trip latency. This increases barrier execution time. PARAMNet HCA supports data in work request (INLINE DATA) feature. As explained earlier, size of data transfer during barrier is very small, we implemented barrier functionality using data in work request feature. In this support, while posting barrier call the payload for barrier is also given to the HCA. In this implementation HCA avoids host memory read to fetch barrier payload. Using data in work request feature, we have observed performance improvement upto 20%.

4. Conclusion and Future Work

This paper describes development of HCA aware Parallel Communication Library. Also, we presented implementation of barrier functionality using feature supported by HCA and we have observed performance improvement. We also realize that in order to meet higher performance, the communication library must know underlying hardware and implement communication calls accordingly.

In future, we plan to optimize collective calls such as Broadcast. We also plan to explore the advantage of supporting vector data type processing. We also plan to comply with OpenFabrics Interface (OFI) [5] which focuses on the development of software interfaces co-designed with fabric hardware.

References

1. C-DAC, PARAMNet-3 Network Interface Card (NIC) based on Gemini Co-processor. http://www.cdac.in/index.aspx?id=hpc_ss_nic

2. OpenFabrics Enterprise Distribution (OFED) Software Overview. https://openfabrics.org/index.php/openfabrics-software.html

3. Message Passing Interface (MPI) Standards. https://www.mpi-forum.org/. Last accessed: 2016-07.

4. PCI Express Specifications. https://pcisig.com/specifications

5. OpenFabrics Interfaces Working Group (OFIWG). https://www.openfabrics.org/index.php/working-groups-overview.html/

DOI: 10.14529/jsfi160307

Parallel Processing Model for Cholesky Decomposition Algorithm in AlgoWiki Project

Alexander S. Antonov[1]***, Alexey V. Frolov***[2]***, Hiroaki Kobayashi***[3]***,***
Igor N. Konshin[1,2,4]***, Alexey M. Teplov***[1]***, Vadim V. Voevodin***[1]***,***
Vladimir V. Voevodin[1]

The comprehensive analysis of algorithmic properties of well-known Cholesky decomposition is performed on the basis of multifold AlgoWiki technologies. A detailed analysis of information graph, data structure, memory access profile, computation locality, scalability and other algorithm properties is conducted, which allows us to demonstrate a lot of unevident properties split up into machine-independent and machine-dependent subsets. The comprehension of the parallel algorithm structure enable us to implement efficiently the algorithm at hardware platform specified.

Keywords: AlgoWiki, algorithm properties, Cholesky decomposition, memory access locality, dynamic characteristics, scalability.

Introduction

The fundamental problem of high-performance computing consists in the accurate coordination between algorithm and program structure and hardware features that results in high efficiency. Modern computers possess great capability, but if there is a lack of the above mentioned coordination, the final implementation efficiency can be very low. A lot of studies are devoted to the most efficient implementation of some specific algorithm.

This paper presents a comprehensive analysis of Cholesky decomposition algorithm. The analysis is based on AlgoWiki methodology and allows specifying the most important features of the algorithm and its parallel implementation model in order to get the most efficient processing.

AlgoWiki [1, 2] is an open encyclopedia of parallel algorithmic features on the Internet which provides collaboration with the worldwide computing community on algorithm descriptions. The main goal of this project is to develop fundamentals and formalize the mapping of algorithms to the architecture of parallel computers. As a result of the current research, the universal description of the structure of algorithm properties has been developed.

All fundamental algorithmic properties that determine implementation efficiencies on modern computing platforms are divided into machine-dependent and machine-independent subsets. This division is made intentionally in order to separate these features of algorithms, which define their perspective implementations on parallel computational systems from a range of questions associated with consequent stages of programming and execution of the resulting programs on particular computing systems.

The AlgoWiki open encyclopedia is based on wiki technology, which allows for any researcher to add and improve the material. A pilot version of the Internet encyclopedia can be found at [3]. The main part of the AlgoWiki project is the algorithms classification and description of their serial and parallel properties. The algorithm descriptions were performed by groups

[1] Moscow State University, Moscow, Russia
[2] Institute of Numerical Mathematics of the Russian Academy of Sciences, Moscow, Russia
[3] Tohoku University, Sendai, Japan
[4] Dorodnicyn Computing Centre of the Russian Academy of Sciences, Moscow, Russia

corresponding to the types of operations involved. It is assumed that this classification will eventually be expanded while other researchers will give their contribution to this project.

To demonstrate the analysis of the parallel processing model we have chosen one of the most popular algorithms — the Cholesky decomposition. This algorithm is widely used for the direct solution of dense and sparse linear systems. The modification of this algorithm is exploited to construct efficient preconditioners for iterative methods.

On the basis of the Cholesky decomposition algorithm properties we demonstrate a mathematical algorithm description, the analysis of sequential and parallel complexity, the information graph of the algorithm, properties of software implementations, analysis of data locality, scalability, as well as dynamic characteristics and performance efficiency of the algorithm implementation. On the basis of the detailed algorithm analysis we give the conclusions on the most efficient algorithm implementation according to facilities of the supercomputer used.

1. Properties and structure of the algorithm

1.1. General description

The Cholesky decomposition algorithm was first proposed by Andre-Louis Cholesky in 1910 [8, 9]. The idea of this algorithm was published by Benoit in 1924 [7] and later was used by Banachiewicz in 1938 [5, 6]. The Cholesky decomposition is also known as the square-root method [10, 15, 16] due to the square root operations used in this decomposition and rarely exploited in some implementations of Gaussian elimination.

Originally, the Cholesky decomposition was used only for dense real symmetric positive definite matrices. At present, the application of this decomposition is much wider. For example, in order to increase the computational performance, its block versions are often applied.

In case of sparse matrices, the Cholesky decomposition is also broadly used as the main stage of a direct method for solving linear systems. In order to reduce the memory requirements and matrix profile, special reordering strategies (such as RCM) are applied to minimize an algorithm arithmetic complexity. A number of reordering strategies (Metis, Zoltan, etc.) are used to identify the independent matrix blocks for parallel computing systems.

Various versions of the Cholesky decomposition are successfully used in iterative methods to construct preconditioners for sparse symmetric positive definite matrices. In the case of incomplete triangular decomposition, the elements of a preconditioning matrix are specified only in predetermined positions (for example, in the positions of the original matrix nonzero elements; this version is known as an IC0 decomposition). In order to construct a more accurate decomposition, a filtration of small elements is performed using a filtration threshold. The use of such a threshold allows one to obtain an accurate decomposition, but the number of nonzero elements may increase. A decomposition algorithm of the second-order accuracy is proposed in [12]; this algorithm allows one to increase the accuracy of decomposition with about the same number of nonzero elements in the factors. In its parallel implementation, a special version of an additive preconditioner is applied on the basis of the second-order decomposition [13].

Here we consider the original version of the Cholesky decomposition for dense real symmetric positive definite matrices. Let us emphasize some basic properties of the algorithm.

Matrix symmetry. The matrix symmetry allows one to store in computer memory slightly more than half of the number of its elements and to reduce the number of operations by a factor of two compared to Gaussian elimination. Note that the LU-decomposition does not require the

square-root operations when using the property of symmetry and, hence, is somehow faster than the Cholesky decomposition, but it requires storing the entire matrix.

Accumulation mode. The Cholesky decomposition allows one to use the so-called 'accumulation' mode due to the fact that the significant part of computation involves 'dot product' operations. Hence, these dot products can be accumulated in double precision for additional accuracy. In this mode, the Cholesky method has the least 'equivalent perturbation'. During the process of decomposition, no growth of the matrix elements can occur, since the matrix is symmetric and positive definite. Thus, the Cholesky algorithm is unconditionally stable.

1.2. Mathematical description and information graph

The input data of the algorithm is a symmetric positive definite matrix A of size n with elements a_{ij}, while the output is the lower triangular matrix L with elements l_{ij}.

The Cholesky algorithm can be represented in the form:

$$
\begin{aligned}
l_{11} &= \sqrt{a_{11}}, \\
l_{j1} &= \frac{a_{j1}}{l_{11}}, \quad j = 2, ..., n, \\
l_{ii} &= \sqrt{a_{ii} - \sum_{p=1}^{i-1} l_{ip}^2}, \quad i = 2, ..., n, \\
l_{ji} &= \left(a_{ji} - \sum_{p=1}^{i-1} l_{ip} l_{jp} \right) / l_{ii}, \quad i = 2, ..., n-1, \quad j = i+1, ..., n.
\end{aligned}
$$

In the last formula the basic elimination step is performed which is the main computational kernel of the algorithm.

The sequential complexity of the Cholesky decomposition algorithm is as follows: n square roots, $n(n-1)/2$ divisions, $(n^3-n)/6$ multiplications and $(n^3-n)/6$ additions (subtractions). Thus, a sequential version of the Cholesky algorithm is of cubic complexity.

There is a block versions of this algorithm; however, here we consider only its standard 'dot' version for simplicity of its informational graph presentation.

An informational graph [17] is a directed acyclic graph the vertices of which correspond to the operations of the algorithm, and the edges — to the transfer of information between them. The informational graph of the dot version of algorithm consists of three groups of vertices positioned in the three-dimensional spaces with integer coordinates. The first and the second groups of vertices belong to the one-dimensional domains corresponding to square root and division a/b operations, respectively. The third group of vertices belongs to the three-dimensional domain corresponding to elimination operation $a - bc$.

The resulting information graph is illustrated in Fig. 1 for $n = 4$. In this graph, the vertices of the first group are highlighted in yellow and marked as SQ; the vertices of the second group are highlighted in green and marked as Div; the vertices of the third group are highlighted in red and marked as F. The vertices corresponding to the results of operations (output data) are marked by large circles. The arcs doubling one another are depicted as a single one. The additional vertices corresponding to input and output data are marked in blue and pink, respectively.

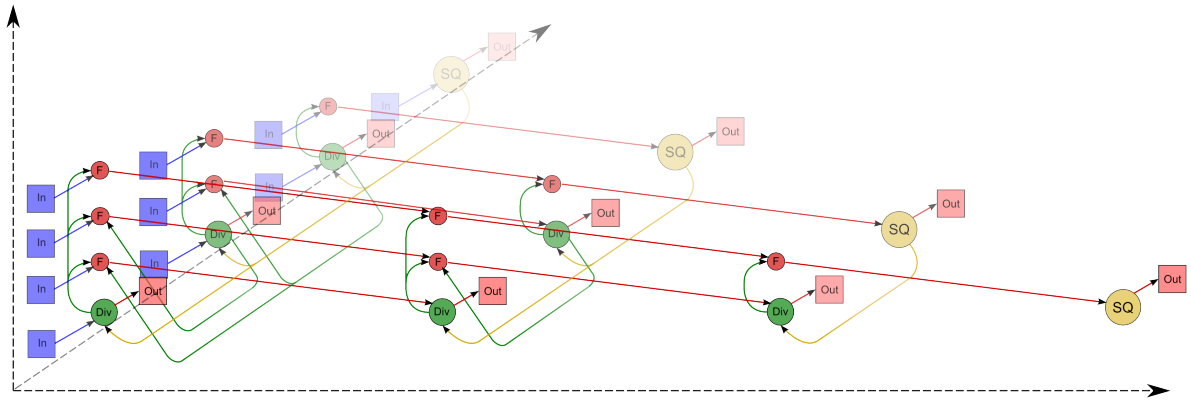


Figure 1. The informational graph of the Cholesky algorithm with input and output data: SQ is the square-root operation, F is the operation $a - bc$, Div is division, In and Out indicate input and output data

1.3. Parallelization resources and other properties of the algorithm

In order to decompose a matrix of order n, a parallel version of the Cholesky algorithm should sequentially perform the following layers of operations: n layers for square roots (a single square root on each layer); $n-1$ layer for divisions (on every layer, the number of divisions is linear and, depending on a particular layer, varies from 1 to $n-1$); $n-1$ layer of multiplications and $n-1$ layer of additions/subtractions (on every layer, the number of these operations is quadratic and varies from 1 to $(n^2-n)/2$).

Contrary to a sequential version, in a parallel version the square-root and division operations require a significant part of overall computational time. The existence of isolated square roots on some layers of the parallel form may cause other difficulties for particular parallel computing architectures. In the case of symmetric linear systems, the Cholesky decomposition is preferable compared to Gaussian elimination because of the reduction in computational time by a factor of two. However, this is not the case for its parallel version.

In addition, we should mention the fact that the accumulation mode requires multiplications and subtractions in double precision. In a parallel version, this means that almost all intermediate computations should be performed with the data given in their double precision format. Contrary to a sequential version, this almost doubles the memory expenditure.

Thus, the Cholesky decomposition belongs to the class of algorithms of linear complexity in the sense of the height of its parallel form, whereas its complexity is quadratic in the sense of the width of its parallel form.

In the case of unlimited computer resources, the ratio of the sequential complexity to the parallel complexity is *quadratic* with respect to the matrix size n. That is, having n^2 processors the Cholesky algorithm can be completed in n steps, while the sequential version can be finished only in $n^3/6$ steps.

The computational power of the Cholesky algorithm considered as the ratio of the number of operations to the amount of input and output data is only *linear* with respect to n.

The Cholesky decomposition is unique for the positive definite matrix, but another order of associative operations may result in the accumulation of round-off errors; however, the effect of this accumulation is not so high as in case when the accumulation mode is not applied while computing dot products.

The information graph arcs from the vertices corresponding to the square-root and division operations can be considered as groups of data such that the function relating the multiplicity of these vertices and the number of these operations is a linear function of the matrix order and the vertex coordinates. These groups may contain long arcs; the remaining arcs are local.

The most known is the compact packing of a graph in the form of its projection onto the matrix triangle, whose elements are recomputed by the packed operations. The long arcs can be eliminated and replaced by a combination of short-range arcs.

The following inequality is valid for the 'equivalent perturbation' M of the Cholesky decomposition (see [16]):

$$||M||_E \leq 2||\delta A||_E,$$

where δA is the perturbation of the matrix A caused by the representation of the matrix elements in the computer memory. Such a slow growth of matrix elements during decomposition is due to the fact that the matrix is symmetric and positive definite.

2. Algorithm implementation

2.1. Sequential implementation of the algorithm

In its simplest version without permuting the summation, the Cholesky decomposition can be represented as follows:

$$\begin{array}{l}
For \quad i = 1, ..., n \quad Do: \\
\quad s := a_{ii} \\
\quad For \quad k = 1, ..., i-1 \quad Do: \\
\quad\quad s := s - a_{ik}a_{ik} \\
\quad EndDo \\
\quad a_{ii} := \sqrt{s} \\
\quad For \quad j = i+1, ..., n \quad Do: \\
\quad\quad s := a_{ij} \\
\quad\quad For \quad k = 1, ..., i-1 \quad Do: \\
\quad\quad\quad s := s - a_{ik}a_{kj} \\
\quad\quad EndDo \\
\quad\quad a_{ij} := s/a_{ii} \\
\quad EndDo \\
EndDo
\end{array}$$

Here s is a double precision variable of the accumulation mode, and the product $a_{ik}a_{kj}$ is assumed to be performed in double precision (as in Fortran intrinsic function 'dprod') provided that original matrix A is stored in a single precision. It means that the entire loop on $k = 1, ..., i-1$ can be implemented as a separate 'inner product' function with the use of accumulation mode.

In addition, the alternative 'outer product' version of the decomposition by Golub and Van Loan [11] can be mentioned. But in this research we focus on the 'inner product' version to make it possible to exploit a higher precision variable s in accumulation mode.

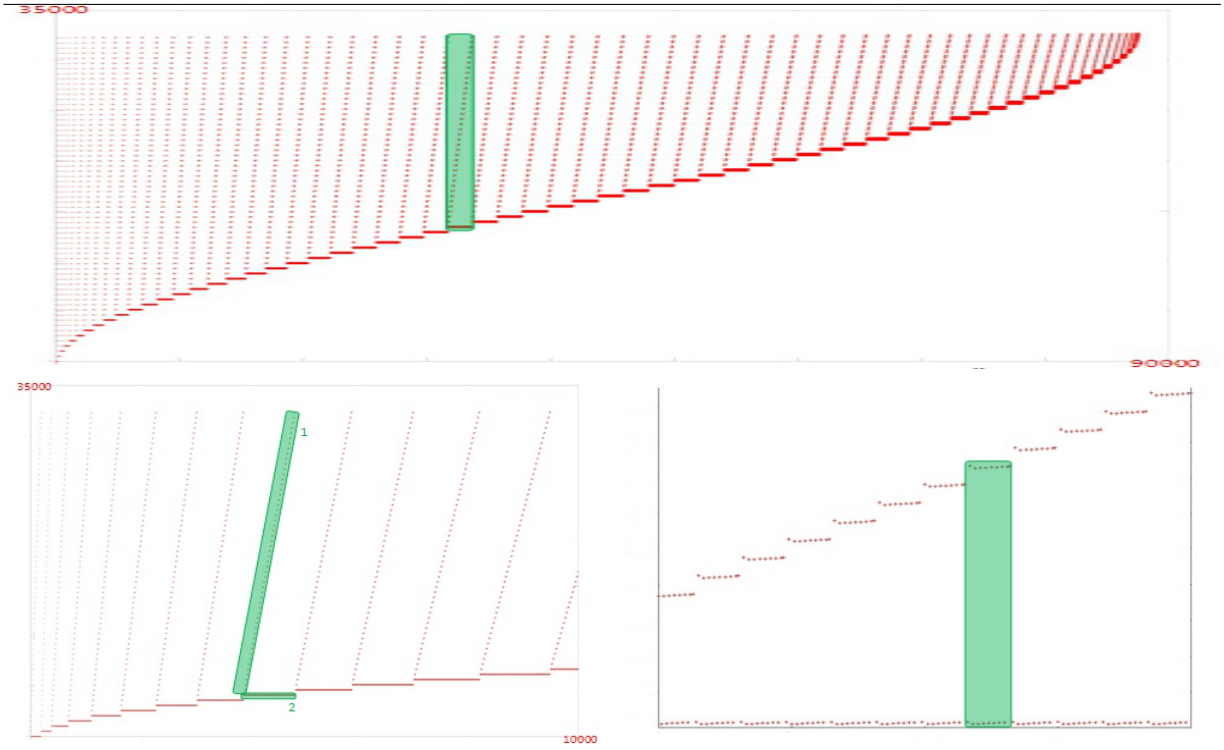


Figure 2. Implementation of the sequential Cholesky decomposition algorithm. A general memory-access profile

A block version of the Cholesky algorithm is usually implemented in such a way that the scalar operations in its sequential versions are replaced by the corresponding block-wise operations instead of using the loop unrolling and reordering techniques.

In order to ensure the locality of memory access in the Cholesky algorithm, the original matrix A and factors L and U should be stored in the lower triangle by rows (or in the upper one by columns) to exploit the sequential access to memory and to improve paging and cache memory usage.

2.2. Structure of memory access and locality estimation

A memory access profile is a sequence of virtual memory addresses (vertical axis) presented in the order they are accessed by the program (horizontal axis). It is illustrated in Fig. 2 for the implementation of the Cholesky algorithm with a single working array. In this profile, only the elements of this array are referenced. The above-illustrated implementation consists of a single main stage; in its turn, this stage consists of a sequence of similar iterations. The example of such iteration is highlighted in green on the top plot. The left-bottom plot gives the fragment of the several first Cholesky decomposition steps. It consists of two parts of different locality with and without of data re-usage. The right-bottom plot gives a more detailed view of a single step.

From Fig. 2 it follows that at each ith iteration, all the addresses starting with a certain one are being used and the address of the first processed element increases with increasing i. We should also note that at each iteration the number of memory accesses increases up to the middle of the algorithm; after that, this number decreases down to the end of the algorithm. This fact allows us to conclude that the data processed by the algorithm are used nonuniformly and that many iterations (especially at the beginning of the process) use a large amount of data,

which decreases the memory access locality. In this case, the structure of iterations is the main factor influencing the memory access locality.

The more detailed study of the locality can be found in [3], as well as 'quantitative' estimation of locality on the basis of daps and cvg values.

2.3. Approaches and features of parallel implementations

The analysis of algorithms structure allows making a conclusion that a large variety of its parallel implementations can be proposed. For example, in the version specified in Section 2.1 only the inner loop over j is parallel. Nevertheless, a simple parallelization technique causes a large number of data transfer between the processors at each step of the outer loop; this number is almost comparable with the number of arithmetic operations. Hence, it is reasonable to partition the computations into blocks with the corresponding partitioning of the data arrays before the allocation of operations and data between the processors of the computing system in use.

A relatively successful decision depends on the features of a particular computer system. If the nodes of a multiprocessor computer are equipped with pipeline accelerators, it is reasonable to compute several dot products at once in parallel. Such a possibility exists in the case of programmable logic devices; in this case, however, the arithmetic performance is limited by a slow sequential square-root operation. In principle, it is possible to apply the so-called 'skew' parallelism; however, this approach is not used in practice because of complexity in the control structure of the algorithms implementation.

2.4. Scalability and dynamic characteristics of the algorithm implementation

The scalability analysis was performed on the Lomonosov supercomputer [14] installed at the Research Computing Center of Moscow State University. For the experiments, the Intel Xeon X5570 processors with 94 Gflops peak performance and the Intel compiler with -O2 option were used. For our experiments we used the ScaLAPACK implementation for the Cholesky decomposition from the MKL library (the pdpotrf method).

Figure 3 illustrates the performance of the parallel implementation of the Cholesky algorithm. The number of processors were taken from 4 to 256, the orders of matrices – from 1024 to 5120. The maximal performance achieved in this experiments was about 2.5 Tflops. We can conclude that the performance of the algorithm increases with the increase of the problem size, and otherwise, for a reasonable problem size, the performance increases with the increase of the number of processors used. Furthermore, the cache effects can be observed by the graph curvature.

Figure 4 illustrates the Cholesky decomposition implementation efficiency (the case of lower triangular matrices) for the matrix order 80000 by using 256 processes. It is shown on the top figure that during the runtime of the program the processor usage level is about 50%. That is a quite good result for programs executed with no use of the Hyper Threading technology. The bottom figure illustrates the variation of FLOPS performance during execution time.

A more detailed analysis of dynamic characteristics can be found in [4]. The diagrams for the number of L1/L3 cache-misses, the number of memory read/write operations, the Infiniband network usage in bytes and packages are presented. The details on collecting the dynamic characteristics of the programm under investigation can be found in [2].

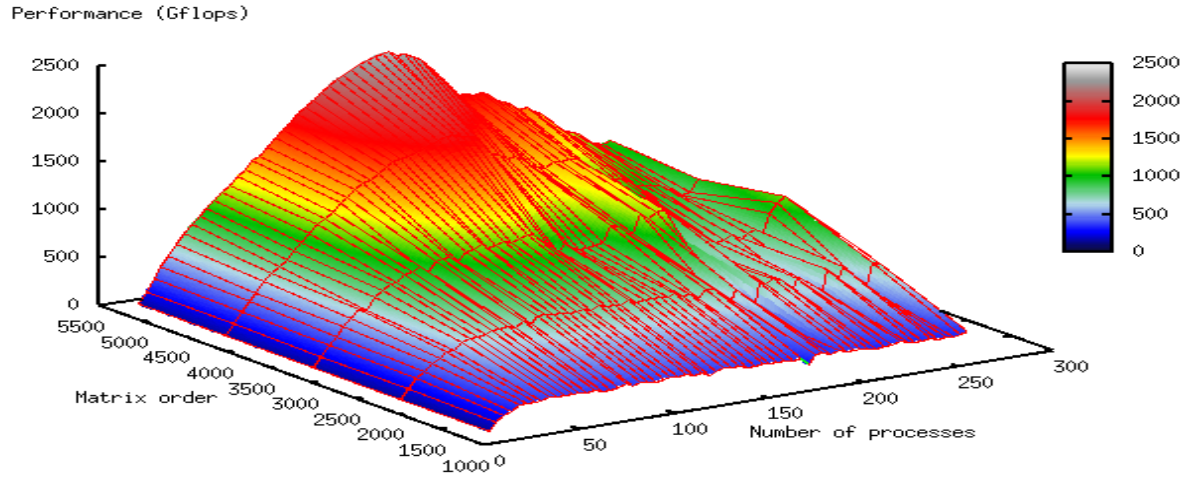


Figure 3. A parallel implementation of the Cholesky algorithm. Performance variation vs. the number of processors and the matrix order

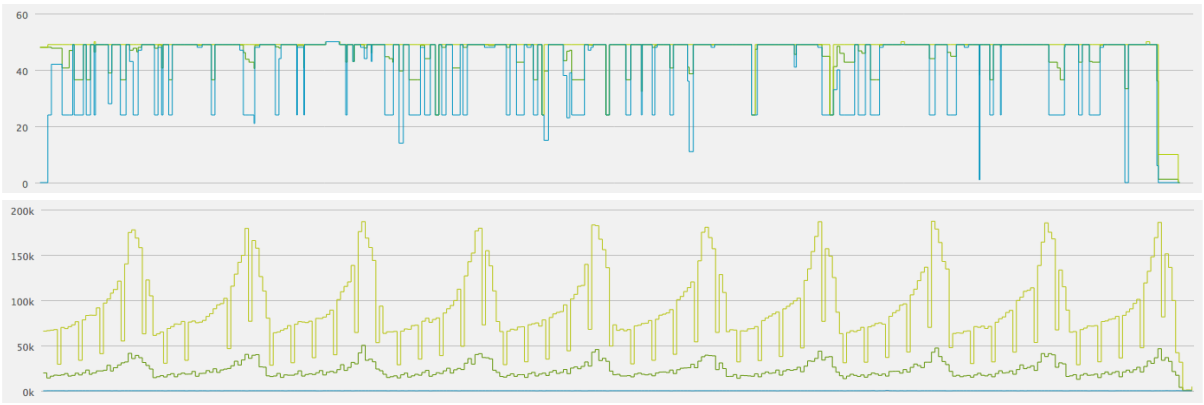


Figure 4. CPU usage diagram (top) and FLOPS performance (bottom) for the Cholesky decomposition execution time

The data obtained from the monitoring system allows one to make a conclusion that the program under the study was operating in an efficient and stable manner. The memory and communication environment usage is intensive, which can lead to an efficiency reduction with the increase of the matrix order or the number of processors in use.

2.5. Computer architectures and existing implementations

Analyzing the performance of the ScaLAPACK library on supercomputers, we can conclude that for large values of the matrix order n the data exchanges reduce the execution efficiency but to a smaller extent than the non-optimality in the organization of computations within a single node. At the first stages, hence, it is necessary to optimize not a block algorithm but the subroutines used on individual processors, such as the dot Cholesky decomposition, matrix multiplications, etc.

Generally speaking, the efficiency of the standard Cholesky algorithm cannot be high for parallel computer architectures. This fact can be explained by the following property of its information structure: the square-root operations are the bottleneck of the Cholesky algorithm in comparison with the division operations or with the $a - bc$ operations, since these operations can easily be pipelined or distributed among the nodes. In order to enhance the performance

efficiency on parallel computers, hence, it is reasonable to use not the original Cholesky algorithm but its well-known modification without square-root operations in the form LDL^T.

The dot version of Cholesky algorithm is implemented in most of linear algebra libraries, such as LINPACK, LAPACK, ScaLAPACK, etc.

Only in few of libraries, the accumulation mode is implemented to reduce the effect of round-off errors. In order to save computer memory, a packed representation of the matrices A and L is also used in the form of one-dimensional arrays.

Unfortunately, in the most of libraries the dot Cholesky implementations are based on its LINPACK implementation utilizing the conventional BLAS library. The dot product is computed in BLAS with no accumulation mode that may substantially reduce the accuracy of computations.

It is interesting to note that the original Cholesky decomposition is available in a majority of libraries, whereas the LDU-decomposition algorithm without square-root operations is used only in specific cases (for example, for tridiagonal matrices), when the number of diagonal elements is comparable with the number of off-diagonal ones.

Conclusion

The detailed analysis of the Cholesky decomposition algorithm properties has been presented. It has been shown that the comprehensive theoretical and practical investigation is important to construct the efficient implementation of the algorithm.

The results described in all sections except 2.4 were obtained in the Moscow State University with the financial support of the Russian Science Foundation (agreement No. 14-11-00190). The research presented in section 2.4 was supported by the Russian Foundation for Basic Research (No. 16-07-01003). Numerical experiments were performed in the Supercomputing Center of Lomonosov Moscow State University [14].

References

1. Voevodin, V., Antonov, A., Dongarra, J.: AlgoWiki: an Open Encyclopedia of Parallel Algorithmic Features. Supercomputing Frontiers and Innovations, Vol. 2, No. 1. Pp. 4–18 (2015). DOI: 10.14529/jsfi150101.

2. Antonov, A., Voevodin, Vl., Voevodin, Vad., Teplov, A.: A Study of the Dynamic Characteristics of Software Implementation as an Essential Part for a Universal Description of Algorithm Properties. 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing Proceedings. Pp. 359–363 (2016).

3. AlgoWiki: an open encyclopedia of algorithms properties. http://algowiki-project.org.

4. AlgoWiki: Cholesky decomposition. http://algowiki-project.org/en/Cholesky_decomposition.

5. Banachiewicz, T.: Principles d'une nouvelle technique de la méthode des moindres carrês. Bull. Intern. Acad. Polon. Sci. A. 134–135 (1938).

6. Banachiewicz, T.: Méthode de résoltution numérique des équations linéaires, du calcul des déterminants et des inverses et de réduction des formes quardatiques. Bull. Intern. Acad. Polon. Sci. A. 393–401 (1938).

7. Benoit, Commandant: Note sur une méthode de résolution des équations normales provenant de l'application de la méthode des moindres carrés à un système d'équations linéaires en nombre inférieur à celui des inconnues (Procédé du Commandant Cholesky). Bulletin Géodésique 2, 67–77 (1924).

8. Brezinski, C.: André-Louis Cholesky. Springer-Verlag GmbH, 311 p. (2014).

9. Cholesky, A.-L.: Sur la résolution numérique des systèmes déquations linéaires La SABIX, Bulletins déja publiés, Sommaire du bulletin n. 39. 81–95 (2005) https://sabix.revues.org/529.

10. Faddeev, D.K., Faddeeva, V.N.: Computational Methods of Linear Algebra. Freeman (1963).

11. Golub, G.H., Van Loan, C.F.: Matrix Computations, 3rd ed. Johns Hopkins Univ. Press, Baltimore, MD, USA (1996).

12. Kaporin, I.E.: High quality preconditioning of a general symmetric positive definite matrix based on its $U^TU + U^TR + R^TU$-decomposition. Numer. Lin. Algebra Appl. 5(6), 483–509 (1998). DOI: 10.1002/(SICI)1099-1506(199811/12)5:6<483::AID-NLA156>3.0.CO;2-7.

13. Kaporin, I.E., Konshin, I.N.: A parallel block overlap preconditioning with inexact submatrix inversion for linear elasticity problems. Numer. Lin. Algebra Appl. 9(2), 141–162 (2002). DOI: 10.1002/nla.260.

14. Sadovnichy, V., Tikhonravov, A., Voevodin, Vl., Opanasenko, V.: "Lomonosov": Supercomputing at Moscow State University. In Contemporary High Performance Computing: From Petascale toward Exascale (Chapman & Hall/CRC Computational Science), Boca Raton, USA, CRC Press. 283–307 (2013).

15. Voevodin, V.V.: Computational Foundations of Linear Algebra. Nauka, Moscow (1977) (in Russian).

16. Voevodin, V.V., Kuznetsov, Yu.A.: Matrices and Computations. Nauka, Moscow (1984) (in Russian).

17. Voevodin, V.V., Voevodin, Vl.V.: Parallel Computing. BHV-Petersburg, St. Petersburg (2002) (in Russian).

DOI: 10.14529/jsfi160308

Data merging for the cultural heritage imaging based on Chebfun approach

Mariem El Afrit[1,2,3], *Yann Le Du*[1,2], *Rafaël Del Pino*[4], *Guolin Zhang*[5]

Cultural heritage imaging has specific needs with regards to the analysis of images that require the manipulation of a single digital object that combines the images obtained from different instruments probing different scales at different wavelengths, with the further possibility of selecting two or three dimensional representations. We propose a unified imaging data processing approach based on the "Chebyshev Technology" using the open source software Chebfun which, by mapping data processing to simple polynomial transformations, brought considerable improvements over already existing procedures. Within that same data processing framework we may further investigate how to merge images originating from different acquisition devices since all images are expressed in the same basis (an approximate Chebfun polynomial basis) before being merged. In the end, we hope to map all imaging data processing to simple polynomial operations. Our massive data-sets required parallelizing some Chebfun functions on GPUs, allowing about 100 times faster polynomial evaluation and up to 12 times faster on CPUs when parallelizing the whole algorithm.

Introduction

The Institut de Recherche de Chimie Paris (IRCP[6]) and Centre de Recherche et de Restauration des Musées de France (C2RMF[7]) have set up a mixed research team, Physico Chimie des Matériaux Témoins de l'Histoire (PCMTH), in order to bring new solutions to the challenges facing the analysis, conservation and restoration of cultural heritage objects, and one of the team's ambitious research projects concerns the latter's digital images : how to best acquire, store, analyze and combine them. The C2RMF tools of the trade include optical and X-ray photography [16], and, thanks to the expertise brought by the team's IRCP component in *Electron Paramagnetic Resonance* (EPR) who pioneered the application of *EPR imaging* (EPR-I) to exobiology [14], has decided to use EPR-I in order to image specific chemical species which X-rays and other traditional techniques are unable to specifically target, like the different layers of carbon-related material found inside paintings.

However, this EPR-I information needs to be merged with the one gathered from other sources, be it X-rays or optical photography in order to provide a single object to which we may attach an interface for subsequent manipulations. This merging would be greatly simplified if we could unify the different pipelines that take raw data and transform them into images. At the moment, there are different pipelines (programs and associated algorithms) for each imaging technique, and even the data formats are different. This actually stands as a major global challenge : many domain specific techniques exist to solve particular problems in imaging, yet the final results obtained after applying each technique are difficult if not impossible to combine. The PCMTH team, in association with the C2RMF imaging team, is thus developing a generic approach which would allow a single pipeline to process all the different kinds of data, with a single unifying data structure and mathematical model founded on *Chebyshev technology* [9], as championed by the Oxford University Numerical Analysis Group through the development of *Chebfun* [13], an open-source software system for numerical computing with functions. The mathematical basis of Chebfun is piecewise polynomial interpolation and in this paper, we first describe the

[1]PSL Research University,Chimie ParisTech-CNRS, IRCP, UMR8247, Paris, France
[2]Centre de Recherche et de Restauration des Musées de France, C2RMF, Paris, France
[3]Université Pierre et Marie Curie, Paris, France
[4]Ecole Normale Supérieure-Paris, Paris, France
[5]Ecole Nationale Supérieure de Chimie de Paris, Paris, France
[6]The IRCP is the research branch of *Chimie ParisTech*, covering many domains of chemistry.
[7]The C2RMF is an institution of the Ministry of Culture devoted to the analysis and conservation of the cultural heritage of the French Museums.

role Chebfun plays in our unifying approach to image data processing[8], and more specifically how we managed to fit the key tomographic process of *backprojection* into the Chebfun paradigm ; we show that this approach provides a promising imaging data processing unification without having to pay a performance cost : it is a zero-cost abstraction[9]. Secondly, we also describe how we managed, by working both on the algorithmic and hardware aspect, to accelerate our Chebfun paradigm application : the execution time speed was scaled down by orders of magnitude compared to our original straightforward implementation on general purpose hardware.

1. The mathematical model of EPR imaging

EPR is a technique that basically finds the value of a magnetic field at which chemical species absorb oncoming microwave radiations, as we can see in figure 1 : this is the signature of the chemical species.

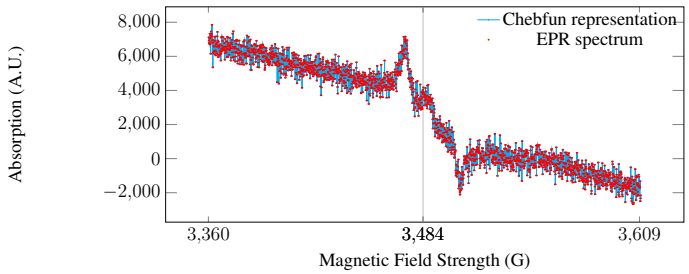


Figure 1. An EPR spectrum with its Chebfun representation. The absorption (arbitrary units) is measured as the magnetic field is varied (abscissa, in Gauss) and the microwave frequency of the oncoming radiation kept constant (around 9.5GHz for X band EPR); EPR traditionally measures the derivative of that signal, explaining the bumps and troughs. The maximum absorption is here around 3500 G

Now, the method of EPR imaging is similar to the well known one of Magnetic Resonance Imaging (MRI) : a magnetic field gradient maps the diffrent position of atoms sensitive to EPR to different positions on a spectrum, so if a species has a resonance spectrum s, then, given a magnetic gradient G and global magnetic field B, the linear density of the species is mapped to a spectrum in the following way :

$$r(B) = \int_{\text{sample}} c(x) \cdot s(B + G \cdot x)\, dx \tag{1}$$

where the integral has the form of convolution, and is applied on the sample. The linear density c is itself related to the volume density ρ by a surface integral : $c(x)$ is the integral of ρ on the plane orthogonal to the direction of the magnetic field gradient at the position x on that direction. In order to simplify the problem, we shall consider our sample as being a 2D flat surface, thus allowing us to represent the surface integral as a line integral, yielding to the Radon transform $\mathcal{R}$ representation of c as being $c = \mathcal{R}\rho$, which allows us to rewrite equation 1 more abstractly as

$$r = s \star \mathcal{R}\rho \tag{2}$$

If we accept that simplification then the process of EPR imaging is depicted in figure (2).

[8]We can find more details on our website HPU4science [2]

[9]We import that concept from the field of computer languages, especially from the abstractions provided by the C++ STL and implemented using Stepanov's generic programming approach [21].

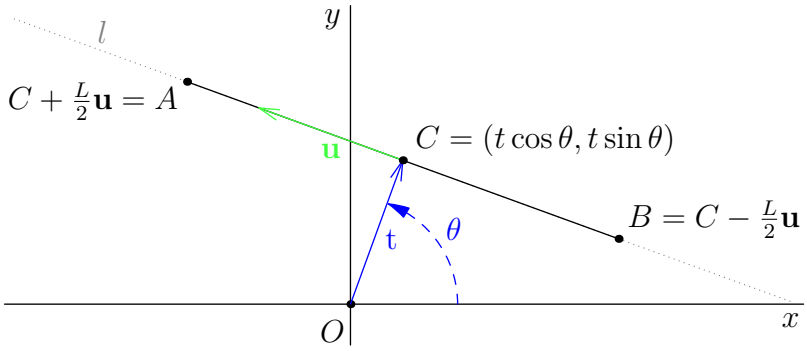


Figure 2. Radon transform geometry : OC is the direction of the magnetic gradient, every spectrum is given by the choice of θ, and on a given spectrum, every data point has an abscissa t and for each t the ordinate on the spectrum is the integrated density along the direction BC. That geometry is used in our backprojection equations starting with 5, and underlies the more qualitative description given in figure 3

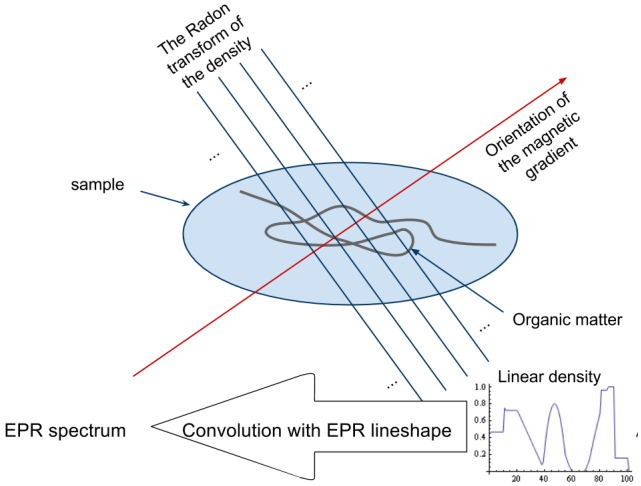


Figure 3. An EPR spectrum is the Radon transform of the density of EPR sensitive chemical species computed for all (sampled) lines orthogonal to the magnetic field gradient. The information is blurred by a convolution with the absorption spectrum of the specific species under study, the *reference lineshape*, to which is added some unavoidable noise

The problem is now to find ρ, a flat surface density which thus depends on the variables (x, y), given r and s : it is a typical inverse problem, involving the well-known Fredholm integral equation of the first kind which the convolution equation 1 is an example of, together with the Radon transform, thus requiring two inversions, the second one being the backprojection operator $\mathcal{B}$. As for the convolutional part, this blurring of the density is traditionally dealt with using Fourier transforms directly on r, but it turns out that the deconvolution can be performed on the final image thanks to the linearity property of the backprojection and convolution operators :

$$\mathcal{B}\left(s \star \mathcal{R}\rho\right)(x, y) = \left(\mathcal{B}s \star \rho\right)(x, y) \tag{3}$$

This commutativity property allows us to postpone the deconvolution and apply it only on the blurred image, instead of applying it on the blurred linear density. But for this paper, we shall even further simplify the problem and suppose that the EPR spectra are a direct mapping of the linear density, a simplification fully justified by the property expressed in equation 3, and we shall thus simply inverse the simplified equation

$$r = \mathcal{R}\rho \tag{4}$$

but with the requirement that we want that inversion to be compatible with the Chebfun approach : we would like to find an inverse transform that maps to simple operations on the chebfun[10], without having to mediate those transformations through sampling.

2. The Chebfun compatible backprojection

In order to inverse the Radon transform, the common practice is to *filter* the backprojection, which we describe by the operator $\mathcal{B}$, and in the end we can retrieve the density ρ by using the well known Fourier form of the *filtered backprojection* [19]

$$\begin{array}{c} \rho\left(x,y\right) \\ = \\ \frac{-1}{2}\mathcal{B}\left\{\mathcal{F}^{-1}\left[i\cdot\operatorname{sgn}\left(S\right)\mathcal{F}\left(\frac{\partial\left(\mathcal{R}\rho\right)\left(t,\theta\right)}{\partial t}\right)\left(S,\theta\right)\right]\left(t,\theta\right)\right\}\left(x,y\right) \end{array} \tag{5}$$

which uses the parametrization described in figure 2. There exists an equivalent formulation that uses the *Hilbert transform* [19]

$$\rho\left(x,y\right)=-\frac{1}{2}\mathcal{B}\left[\mathcal{H}\left(\frac{\partial\left(\mathcal{R}\rho\right)\left(t,\theta\right)}{\partial t}\right)\left(t,\theta\right)\right]\left(x,y\right) \tag{6}$$

which thanks to the commutativity of the Hilbert transform with the derivative becomes

$$\rho\left(x,y\right)=-\frac{1}{2}\mathcal{B}\left[\frac{\partial\mathcal{H}\left(\mathcal{R}\rho\right)\left(t,\theta\right)}{\partial t}\left(t,\theta\right)\right]\left(x,y\right) \tag{7}$$

If we now recall that the $\mathcal{R}\rho$ are the EPR spectra, we can consider each of those to be a chebfun, which we shall call P_θ, and if we define

$$Q_\theta\left(t\right)=P_\theta\left(t\right)\cdot\sqrt{1-t^2} \tag{8}$$

we obtain that in equation (7),

$$\mathcal{H}\left(\mathcal{R}\rho\right)\left(t,\theta\right)=\mathcal{H}\left(P_\theta\right)\left(t\right)=\mathcal{H}\left(\frac{Q_\theta\left(t\right)}{\sqrt{1-t^2}}\right)\left(t\right) \tag{9}$$

we can now express Q_θ in the basis of the Chebyshev polynomials of the first kind

$$Q_\theta\left(t\right)=\sum_n\left(C_n\left(\theta\right)\cdot T_n\left(t\right)\right) \tag{10}$$

and from equations 9, and 10 we obtain

$$\mathcal{H}\left(P_\theta\left(t\right)\right)\left(t,\theta\right)=\sum_n\left(C_n\left(\theta\right)\cdot\mathcal{H}\left(\frac{T_n\left(t\right)}{\sqrt{1-t^2}}\right)\left(t\right)\right) \tag{11}$$

Because we study physical objects which have a finite extension, and under the hypothesis that the corresponding chebfun will also be of finite support, our Hilbert transform becomes a *Tricomi transform* [22] $\mathcal{T}$, which allows us to rewrite [17] equation 11 as

$$\mathcal{H}\left(P_\theta\left(t\right)\right)\left(t,\theta\right)=\sum_n\left(C_n\left(\theta\right)\cdot\mathcal{T}\left(\frac{T_n\left(t\right)}{\sqrt{1-t^2}}\right)\left(t\right)\right) \tag{12}$$

We now use a very useful property of the Tricomi transform, which is crucial in understanding how the filtered backprojection can indeed be mapped to simple operations on chebfuns :

[10] A *chebfun*, with a lower case "c" is a shorthand that means Chebfun object, where the uppercase "C" relates to the concept behind the Chebfun paradigm.

$$\mathcal{T}\left(\frac{T_n(t)}{\sqrt{1-t^2}}\right)(t) = -\frac{1}{n}\frac{\partial T_n(t)}{\partial t} \tag{13}$$

which allows us to rewrite equation 12 as

$$\mathcal{H}\left(P_\theta(t)\right)(t,\theta) = -\sum_n \left(\frac{C_n(\theta)}{n} \cdot \frac{\partial T_n(t)}{n \cdot \partial t}\right) \tag{14}$$

and finally equation 7 becomes

$$\rho(x,y) = \frac{1}{2}\mathcal{B}\left[\frac{\partial^2}{\partial t^2}\left(\sum_n \left(\frac{C_n(\theta)}{n} \cdot T_n(t)\right)\right)\right](x,y) \tag{15}$$

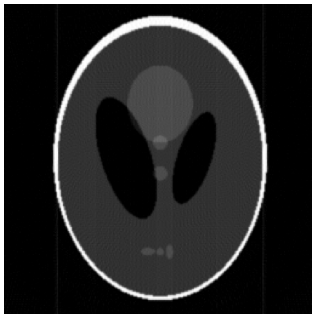
This last expression reveals the simple relationship that exists between the spectrum chebfuns, obtained directly from the raw (sampled) spectra, and the reconstructed image. It is quite straightforward to transform Equation 15 into an algorithm amenable to a Chebfun compatible implementation :

1. Prefactor the raw spectra $r(t)$ by $1/\sqrt{1-t^2}$;
2. transform each prefactored spectrum into a chebfun with the `FUNQUI` function ;
3. apply the square brackets part of equation 15 ;
4. apply the naive backprojection $\mathcal{B}$ that constitutes the remaining part of equation 15.

Part 4 of our algorithmic implementation above requires the computation of the naive backprojection, and here also we took advantage of the Chebfun approach.

Figure 4 shows that our reconstruction is at least as good as the standard procedure using the vanilla Matlab imaging toolbox solution that uses the `IRADON` function.

Original Shepp-Logan phantom.

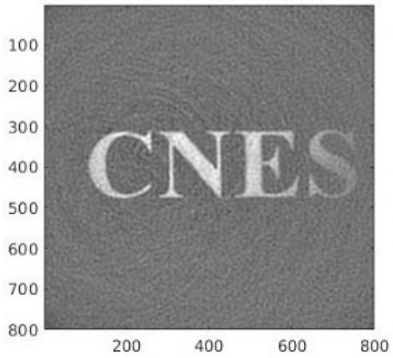


Standard Fourier reconstruction.

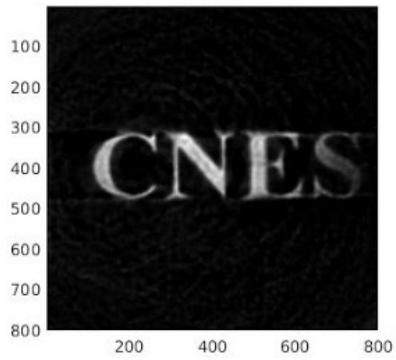
Our Chebfun reconstruction, cf. figure 6.

Figure 4. We tested our approach on the Shepp-Logan phantom (up). The other two figures show the phantom reconstruction using only the information provided by the Radon transform of the phantom sampled on the angles, cf. figure 2. We can see that our approach is at least as good as the standard one

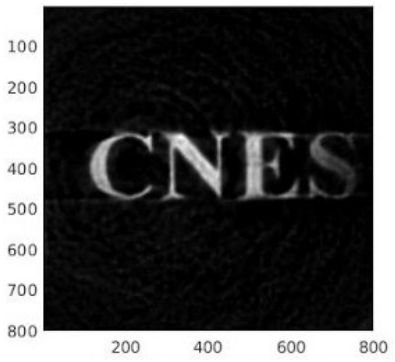
We have also applied our approach to real data [18], as we see in figure 5 : our approach does not yet procede to any noise filtering, yet the result is already an improvement on the traditional approach using a black-box Fourier approach as provided in the software suite that comes with the EPR imaging spectrometer [1].



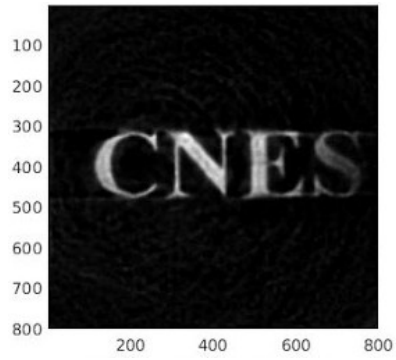
Standard Fourier reconstruction using the BRUKER Xepr software. It took few seconds for the construction but after a heavy manual work that took few hours.



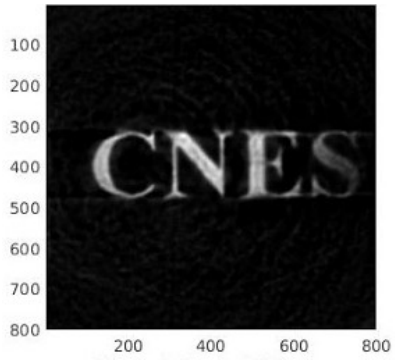
CPU sequential
Time : 6.6 hours



CPU parallel
Time : 33 minutes



GPU simple precision
Time : 2.5 minutes



GPU double precision
Time : 4 minutes

Our Chebfun reconstruction explained in figure 6.

Figure 5. "C(entre) N(ational) d' E(tudes) S(patiales)" (the French space institute) was laser printed (heated toner is very responsive to EPR) on a piece of paper (1cm by 5cm), hidden in an EPR tube filled with sand (EPR neutral) positioned in a Bruker EPR imaging spectrometer. The reconstructions do not (yet) include noise filtering

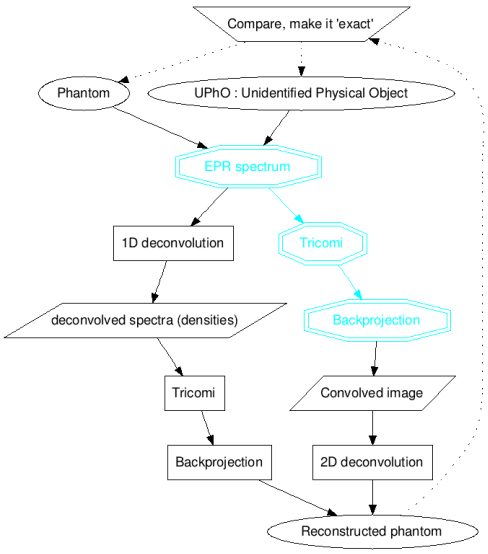


Figure 6. Our approach to tomography uses two mathematically equivalent yet implementation-wise different paths : in this paper, we describe the three nodes which are highlighted in blue and with double-line contours. The key steps are the transformation of spectra into chebfuns, and the formulation of the inverse problem solution using the Tricomi transform

3. Accelerating the code execution time

Because we target real time imaging, we need to decrease the processing time. The accelerations currently only rely on the vectorization of the algorithms, and in order to reap the full benefits we decided to buy Graphical Processing Units (GPU) hardware (Figure 7) and we describe that in sections 3.4 to 3.2. The implementation of the most important part of the code is described in section 3.1 and the whole code will be made available through a Gitlab repository as a literate program [20] written with the in-house developed Vim literate programming plugin[11] called *Kosmogram* [6]. Further improvements are expected with the use of a novel programming language, Pony [10], that puts actor programming to the forefront as we shall explain in section 3.3.

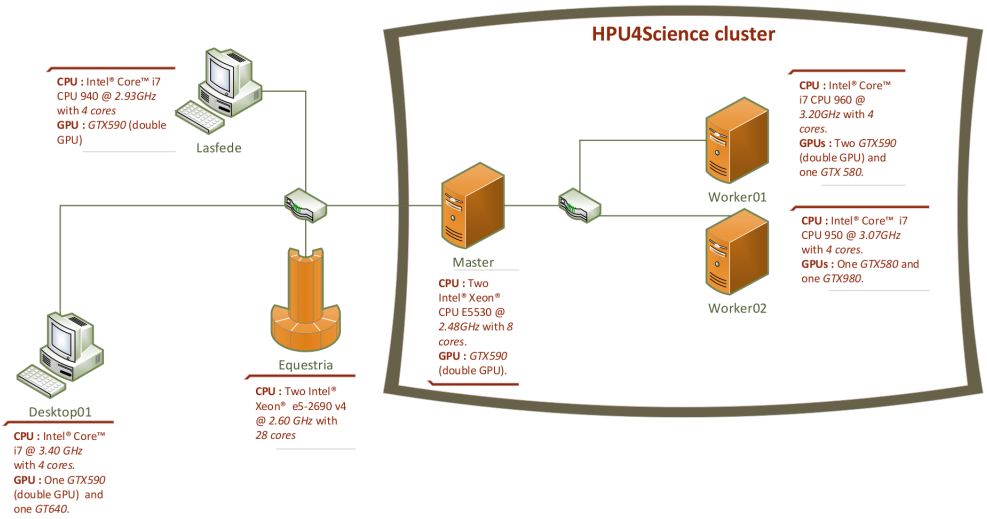


Figure 7. The cluster HPU4Science is composed of fours servers (Equestria, Master, Worker01, Worker02) and two desktops (Lasfede and Desktop01) with a total of 52 CPU cores and nine GPUs

[11] In section 3.3 we describe some more related open source project that we are working on.

3.1. Backprojection implementation

The key step in the imaging pipeline is the backprojection [19], which we express in the following form :

$$\mathcal{B}h\left(x,y\right) := \frac{1}{\pi}\int_{\theta=0}^{\pi} h\left(x\cdot\cos\left(\theta\right)+y\cdot\sin\left(\theta\right),\theta\right)\cdot d\theta \tag{16}$$

So we need to integrate on θ and we must therefore have the function that appears in the integrand, i.e. h as a function of $t = x\cdot\cos\left(\theta\right)+y\cdot\sin(\theta)$ which we have thanks to having transformed each spectrum into a chebfun : this means that for each θ_i we have a chebfun h_i which we compute at the value t, and we therefore need to construct the matrix of values of t. Once this is done, we obtain a matrix in which each line is the computation of a particular polynomial (chebfun) h_i on each t corresponding to a particular θ_i and all values of x and y in the image (Figure 8).

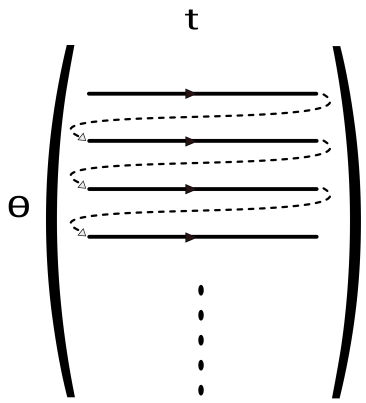


Figure 8. Construction of pixels corresponding to each t and θ

Therefore, if we look at the matrix column-wise (Figure 8), we have a set of values obtained for a particular t this time and all values of θ: we can now compute the chebfun for each column of values (by using `POLYFIT`), thus building a matrix of column chebfuns. Each chebfun is then summed to compute the backprojected value at all x and y: we sum (with the overloaded Chebfun sum) on each chebfun, we obtain a vector of size $x\cdot y$ which is the flattened image (Figure 9), and we can just `RESHAPE` it to an image.

We need to evaluate each θ polynomial on every $t\left(\theta,x,y\right) = x\cdot\cos\left(\theta\right)+y\cdot\sin\left(\theta\right)$ and in order to parallelize this task we construct a N_θxn^2 matrix each row of which contains all the $t\left(\theta,x,y\right)$ for a given θ :

$\theta = \theta_1,\ldots,\theta_n$

$X = [x_1,\ldots,x_1,x_2,\ldots,x_2,\ldots,x_n,\ldots,x_n]$ each value appearing n times.

$Y = [y_1,\ldots,y_n,y_1,\ldots,y_n,\ldots,y_1,\ldots,y_n]$ this succession of values is repeated n times.

$t\left(i,:\right) = X\cdot\cos\left(\theta_i\right)+Y\cdot\sin\left(\theta_i\right)$.

For each line i of the matrix t we apply the polynomial h_i to obtain the matrix $t^{'}$:

Algorithm 1 Construction of the matrix $t^{'}$

```
for i=1:N_theta
        t(i,:) = h_i(t(i,:))
```

We then transform each column of the matrix t' into a chebfun objects and on each column we apply the function `SUM`, which is the integral of the chebfun :

Algorithm 2 Construction of the flattened picture

```
for i=1:n*n
        S(i) = sum(t(:,i))
```

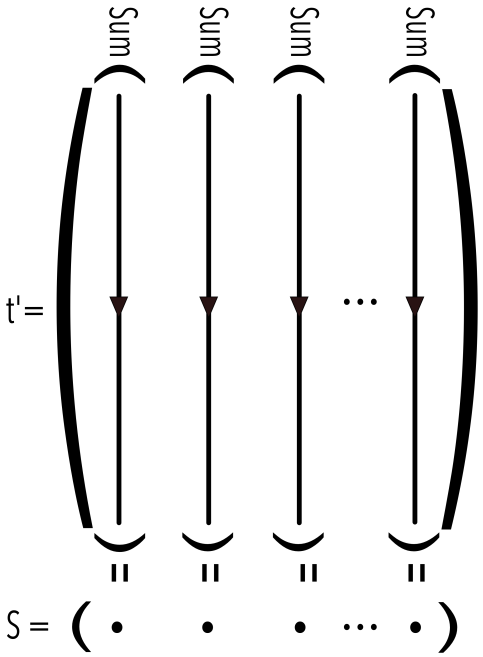


Figure 9. Flattened picture

In our code, wherever it was possible, we replaced the `FOR` loops by the function `IRBSXFUN` in the accelerated version.

3.2. MATLAB Distributed Computing Server

The image construction is done pixel by pixel, which is an embarrassingly parallel process which quiet naturally led us to use the MATLAB Distributed Computing Server [12] on the cluster HPU4Science which we designed accordingly as we shall see in section 3.4. The construction of the pixels is distributed on the appropriate target machines which we selected thanks to the MATLAB Job Scheduler (MJS) – Table 1 explains the MJS configuration for each construction. We then replace each `FOR` loop by a `PARFOR` loop.

Speedup results

In the beginning, the execution time took between 8 to 12 hours (depending on the sample under study) and thanks to the code vectorization (explained in subsection 3.1) we managed to significantlygoing down to 6.6 hours on CPU. Then, using the CPU multi-core power, we further reduced the computation time to 33 minutes (about 14 times faster), and finally reached 4 minutes on GPUs in double precision (more than 100 times faster, or "orders of magnitude" as we claimed at the beginning of the paper). In table 1 we explain the MJS configuration used to get those results.

	MJS configuration	Time
CPU sequential	4 workers on Desktop01 (figure 7)	6.6 hours
CPU parallel	8 workers on the Master (figure 7	33 minutes
GPU simple precision	13 workers using 4 machines : two on Lasfede, two on Master, seven on worker05, two on Desktop01 (figure 7	2.5 minutes
GPU double precision	13 workers using 4 machines : two on Lasfede, two on Master, seven on worker05, two on Desktop01 (figure 7)	4 minutes

Table 1. The optimum configuration of resources used for each construction in Figure 5

3.3. Mixing parallelism and concurrency

Because we envision the application of this pipeline to data generated in real time – notably with the use of mobile imaging EPR probes – we have begun investigating the use of the actor paradigm to efficiently maximize CPU and GPU utilization, with the idea that as data gets collected, then we shall have as many actors constructing the output as there are increments in the data being collected: the first actor will construct a rough image from a small quantity of data, the second actor will improve on it by taking into account the first actor's data plus a newly collected one, and so on until the last actor generates the image from the whole collected data. That will allow users to decide when there is enough data collected for the task at hand, and, perhaps more importantly, decide if some modification to the imaging setup is needed, like changing the imaging probe position because the field of view is not adequate. The programming language Pony [10] [15] has really impressed us because it allows straightforward actor programming while at the same time staying on par with C with regards to speed. Our current focus in on developing a solid scientific library for Pony, by writing efficient wrappers for the GNU Scientific Library (GSL) [8], the ArrayFire [3] GPU optimized scientific library and the Yeppp SIMD library [11]. The development of these wrappers has taken the form of an open-source project called SciPony [7] on the Gitlab [5] platform, and is being developed using the literate programming Vim plugin called Kosmogram [6], itself also an open source project on Gitlab[12].

3.4. The HPU4Science cluster

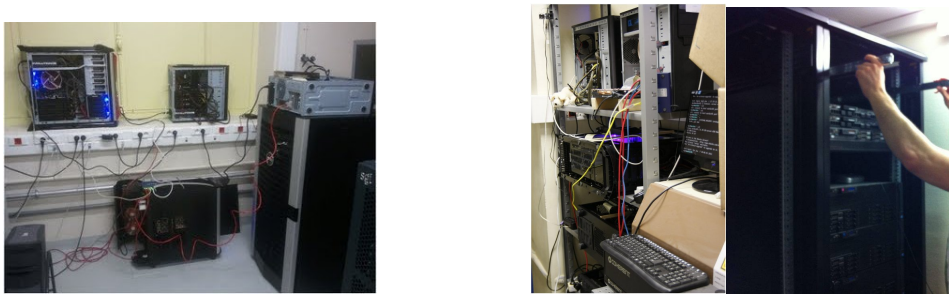
The cluster, known as HPU4Science (Figure 7), began with a $40,000 budget and a goal of building a viable scientific computation system with performance roughly equivalent to that of a $400,000 "turnkey" system made by NVIDIA, Dell, or IBM.

The ideals of the project team required that the system use open source software wherever possible and that it be built only from hardware that is available to the average consumer. The project budget was, of course, an order of magnitude more than the average consumer could afford. In principle, however,

[12]Both SciPony [7] and Kosmogram [6] are open-source software, with the only requirement of having a Gitlab account.

anyone should be able assemble a similar, albeit scaled down, system and freely use the software and code developed by the HPU4Science team [4] to perform high-end scientific computing.

The HPU4Science cluster's construction went through multiple stages. We started by sharing an existing cluster room in Chimie-ParisTech (Figure 10 on the left) but because of electrical power constraints and heat control we had to move to another room that belongs to the PCMTH's team (Figure 10). We have recently (second half 2016) added new nodes in the cluster, with the goal of testing the actor programming framework as championed by the Pony programming language.



HPU4Science in the beginning Servers room

Figure 10. The HPU4Science's current installation as of 2016 with 20 GPUs (4 GPU NVIDIA cards per node, for a total of 20 GPUs thanks to double GPU cards) with another node in the form of a 28-core blade (specifically for developing and testing a conconcurrent imaging pipeline) localized in a dedicated server room at Chimie-ParisTech

For the operating system we use Linux for its stability, the ability to pick from a wide variety of file systems, the large existing code base for high performance computing, and the ease of tailoring the OS to the specific hardware requirements. Availability of open-source projects with code that can be configured for highly parallelized processing was also an important advantage.

The master and all workers run Ubuntu server edition, installed with the minimal OpenSSH server profile. All machines are headless to minimize the OS memory footprint, so all interactions with the cluster happens at the command prompt through ssh.

The only closed source software used on the cluster is Matlab and it is because of the power of the Chebfun toolbox that we are re-coding some of its function using open source languages (3.3), as we explain in subsection 3.3.

4. Authors' contributions

Mariem El Afrit is a PhD candidate working on the development of novel approaches based on Chebfun for (tomographic) imaging and data fusion, and she worked specifically on the implementation of the algorithms and on the details of the mathematical derivations and their algorithmic formulation, together with the subsequent parallelization, while at the same time being responsible for most of the cluster assembling and management.

Yann Le Du is a CNRS research scientist at Chimie-ParisTech specialized in Electron Paramagnetic Resonance Imaging, and he initiated the use of Chebfun for imaging, devised the Tricomi transform approach to backprojection and worked on the whole mathematical framework, while also contributing to the code and launched the work on the re-coding of the pipeline in the Pony language.

Rafaël Del Pino, now a PhD candidate in cryptography, was a summer intern in the team under the guidance of Yann Le Du, and worked on the algorithm implementation, helped with the mathematical

derivations and specifically with the finalization of the mathematical form of the Tricomi expression used in the pipeline.

Guolin Zhang was a summer intern in the team, and worked on the acceleration of the code on the Matlab source code of the imaging pipeline under the guidance of Mariem El Afrit.

As for the paper itself, it was written by Mariem El Afrit and Yann Le Du.

Conclusion

Our goal is to unify imaging processing, especially tomographic imaging, by relying on a powerful data representation based on Chebfun [9]. This requires the adaptation of existing algorithms to this data structure, and we have succeeded in doing so with the most fundamental of the tomographic processes, the backprojection. By reformulating it in terms of the Tricomi transform, and applying the Cbebfun paradigm at each of the underlying algorithmic steps, we have managed to vectorize it and not only obtain results of the same quality as with the traditional approach, but with speeds that makes it possible to use our approach on real time data generation, and further work is thus underway to make the parallel processing concurrent[13].

References

1. *EPR User Manuals & Technical Documentation*.
2. HPU4Science website : http://hpu4science.org.
3. http://arrayfire.com.
4. http://hpu4science.org/overview/the-team.
5. https://gitlab.com/.
6. https://gitlab.com/hpu/codoc.
7. https://gitlab.com/hpu/scipony.
8. https://www.gnu.org/software/gsl/.
9. http://www.chebfun.org/about/.
10. http://www.ponylang.org/.
11. http://www.yeppp.info/.
12. *Mathworks. (2015). Global Optimization Toolbox: User's Guide (r2015b). Retrieved October, 2011 from http://fr.mathworks.com/help/mdce/index.html*.
13. Zachary Battels and Lloyd N Trefethen. An extension of MATLAB to continuous functions and operators. 25(5):1743–1770, 2003.
14. Laurent Binet, Didier Gourier, and Sylvie Derenne. Potential of EPR imaging to detect traces of primitive life in sedimentary rocks. *Earth and Planetary Science Letters*, 273:359–366, 2008.
15. Clebsch et al. Deny capabilities for safe, fast actors. 2015.

[13] We have applied for a funding to develop a mobile EPR spectrometer which requires real time imaging

16. Clotilde Boust et al. L'imagerie scientifique pour la conservation-restauration des oeuvres des musées : quels besoins en traitement d'images ? *GRETSI*, 2015.

17. Le Du et al. Electron Paramagnetic Resonance Imaging (EPR-I) with Chebfun , 2012.

18. Yann Le Du et al. The tricomi approach to chebfun imaging in electron paramagnetic resonance tomography : Towards a unifying imaging process in cultural heritage. *GRETSI*, 2015.

19. Timothy G. Feeman. The Mathematics of Medical Imaging. *The Mathematics of Medical Imaging*, pages 11–19, 2010.

20. Donald E. Knuth. Literate programming. *THE COMPUTER JOURNAL*.

21. Alexander A. Stepanov and Daniel E. Rose. *From Mathematics to Generic Programming*. Addison-Wesley Professional, 2014.

22. F. G. Tricomi. On the finite Hilbert transformation. *Quarterly Journal of Mathematics*, 2(March 1950):199–211, 1951.