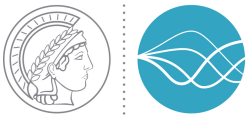

2nd Workshop

on

Power-Aware COmputing (PACO 2017)

July 05-08, 2017

Ringberg Castle
Tegernsee

Supported by:

MEGWARE®

Contents

Program

Wednesday, July 05

15:00 - 15:40 Registration

15:40 - 16:00 Welcome (Garetnzimmer)

16:00 - 17:00 **Jack Dongarra** p. 5
A Look at Energy Saving on the Intel Knights Landing for Linear Algebra Computations

17:00 - 17:40 **Matthias Bolten** p. 5
Increasing efficiency of multigrid methods

17:40 - 18:30 Guided castle tour

18:30 - 20:00 Dinner

Thursday, July 06

09:00 - 10:00 **Laura Grigori** p. 6
Communication avoiding algorithms for linear algebra kernels

10:00 - 10:30 Coffee Break

10:30 - 11:00 **Sue Thorne** p. 11

11:00 - 11:30 **Krzysztof Rojek** p. 17

11:30 - 12:00 **Ernesto Dufrechou** p. 21

12:00 - 12:30 **Julio Ortega** p. 27

13:00 - 18:30 Lunch box pickup & Wallberg trip

18:30 - 20:00 Conference dinner

Friday, July 07

09:00 - 10:00	**Ulrich Rüde** *Algorithmic efficiency and the energy wall*	p. 6
10:00 - 10:30	Coffee Break	
10:30 - 11:00	**Martin Köhler**	p. 33
11:00 - 11:30	**María Barreda**	p. 39
11:30 - 12:00	**José I. Aliaga**	p. 45
12:00 - 12:30	**Carolin Penke**	p. 51
12:30 - 14:00	Lunch break	
14:00 - 15:00	**Axel Klawonn** *Avoiding communication by nonlinear domain decomposition methods*	p. 7
15:00 - 15:40	**Hartwig Anzt** *Compute more, expend less... energy?*	p. 8
15:40 - 16:00	Coffee Break	
16:00 - 16:30	**Anamika Chowdhury**	p. 57
16:30 - 17:00	**Christian Himpe**	p. 63
17:00 - 17:40	**Markus Geveler** *Hardware/algorithm co-design for energy efficient scientific computing*	p. 9
17:40 - 18:30	Open discussion and closing	
18:30 - 20:00	Dinner	

Invited Talks

A Look at Energy Saving on the Intel Knights Landing for Linear Algebra Computations

Jack Dongarra[1]

In this talk we will look at the current state of high performance computing and look to the future toward exascale. In addition, we will examine some issues that can help in reducing the power consumption for linear algebra computations.

Increasing efficiency of multigrid methods

Matthias Bolten[2]

While multigrid methods show an excellent scaling behavior depending on the number of cores only logarithmically. This logarithmic behavior is due to the global nature of the underlying PDE, requiring communication between all parts of the domain and thus on the whole machine. On large-scale supercomputers this logarithmic dependency is visible, moreover, communication costs much more energy than computation and should be avoided as much as possible. To reduce influence of this dependence, the amount of work being carried out on the coarse levels has to be reduced. This can be achieved by applying aggressive coarsening, effectively reducing the number of coarse levels. While this is easily possible in a geometric multigrid setting, the overall performance of the method will deteriorate, as the size of the coarse space is reduced. To retain a good convergence rate, the smoothing procedure has to be improved. We propose to use block smoothers to accomplish this, at the same time this results in a higher locality of the operations performed and thus in a better exploitation of modern computer architectures requiring less energy.

[1]Electrical Engineering and Computer Science Dept., Univ. of Tennessee, Knoxville, TN 37996-3450,
dongarra@cs.utk.edu

[2]Faculty of Mathematics and Natural Sciences, Univ. of Wuppertal, Gaußstraße 20, 42119 Wuppertal, Germany,
bolten@math.uni-wuppertal.de

Communication avoiding algorithms for linear algebra kernels

Laura Grigori[3]

In this talk we discuss one of the challenges we face in high performance computing which is the increased communication cost, the fact that the time needed to communicate a floating-point number between two processors exceeds by huge factors the time required to perform a single floating point operation by one of the processors. Several works have shown that this gap has been increasing exponentially and it is predicted that it will continue to do so in the foreseeable future! Motivated by this trend, we describe novel algorithms for linear algebra computational kernels that drastically reduce the communication cost with respect to classic algorithms.

Algorithmic efficiency and the energy wall

Ulrich Rüde[4]

In the ongoing race to exascale, energy has been identified as a critical resource. While it is important to reduce the energy consumption on the system level, we must be aware that the dissipation of energy is eventually caused by executing digital operations and data transfers and is thus primarily a challenge for designing efficient algorithms. In this light, the energy wall can also be seen as a performance abyss, i.e. it is created by our failure to implement efficient algorithms so that they also execute fast. For example it is known that the fastest algorithm to solve the discrete Poisson's equation in 2D requires 30 N operations. Thus, e.g. on a Petaflop system, we might expect that the solution of a moderately large system with a billion unknowns (N=10^9) should run in 30 microseconds. Current computational practice misses this prediction by several orders of magnitude, creating a performance deficit that cannot be explained by parallel communication overhead or an unfavorable behavior of the memory hierarchy alone. The talk will try to shed some light on this situation. As a test bed, we will use the Hierarchical Hybrid Grids (HHG) multigrid prototyping software that can solve systems with up to 10^{13} degrees of freedom for PDE problems.

[3]Universite Pierre et Marie Curie, 4 Place Jussieu, 75005 Paris, France, laura.grigori@inria.fr

[4]Fiedrich-Alexander Univ. at Erlangen-Nürnberg, Cauerstrasse 11, 91058 Erlangen, Germany, ulrich.ruede@fau.de

Avoiding communication by nonlinear domain decomposition methods

Axel Klawonn[5]

Parallel Newton-Krylov domain decomposition (Newton-Krylov-DD) methods are fast and robust solvers, e.g., for nonlinear implicit problems in solid and fluid mechanics. In these methods, the nonlinear problem is first linearized and then each linearized problem is decomposed, i.e., is solved by a Krylov space method using a domain decomposition preconditioner. In nonlinear domain decomposition methods, by changing the order of these operations, first the nonlinear problem is decomposed and then the nonlinearly decomposed problem is linearized. This allows for the design of new parallel nonlinear FETI-DP and BDDC domain decomposition methods with increased locality and reduced communication. Such nonlinear domain decomposition methods have been successful in reducing the time to solution for different nonlinear problems compared to Newton-Krylov-DD approaches. Since the nonlinear domain decomposition methods are based on a decomposition of the global nonlinear problem into many local nonlinear problems, strongly local nonlinearities can be resolved by a small number of local nonlinear problems or computational cores, respectively. The remaining cores which are not busy with those local, strongly nonlinear effects can wait and save energy. In classical Newton-Krylov-DD approaches, this is not possible since all nonlinear effects, whether local or global, interfere with the global convergence of Newton's method and keep all cores busy all the time.

[5]Mathematisches Institut, Universität zu Köln, Weyertal 86-90, 50931 Köln, Germany, klawonn@math.uni-koeln.de

Computing more for saving energy?

Hartwig Anzt[6]

The cost of moving data between different memory levels and/or processors is typically dominating the energy balance of numerical algorithms. Precisely, the energy needed for communication and data access is much larger than the energy needed for computations. With the shrinking transistor size and no disruptive paradigm change in memory technology in sight, this gap in resource requirements is expected to widen in future. In this talk we evaluate the potential of trading data movement against computations. We specifically focus on iterative linear algebra. Based on the observation that the convergence of iterative methods typically depends on how fast information gets propagated via the system, the idea is to accept some convergence delay and additional computations in favor of reduced communication volume.

[6]Electrical Engineering and Computer Science Depti., Univ. of Tennessee, Knoxville, TN 37996-3450,
anzt@cs.utk.edu

Hardware/algorithm co-design for energy efficient scientific computing

Markus Geveler[7]

The energy crisis is approaching a showdown: Energy consumption due to computing is expected to increase much faster than the entire world's total energy production which will be exceeded in the 2030ies. Even based on extrapolating current technology to a hypothetical future level (mainly based on improved manufacturing processes leading to smaller transistors) this would lead to only an insignificant delay of this point in time.

Performance is a function of hardware and code. Energy to solution is determined by performance. In an ideal scenario scientific software as well as the compute hardware used in scientific workflows should not follow a design paradigm that is oblivious of the respective other.

Today however, scientific computing is still somewhat 'blind on the energy eye': Commodity compute hardware in HPC centers has substantial energy requirements so that the associated expenses over the lifetime of a system may exceed the initial acquisition costs. In addition, the energy supply for supercomputers is not always an integral part of its overall design – consumers (such as the compute cluster, cooling, networking, management hardware) are often developed independently from the key technologies of the energy revolution, e.g. renewable energy sources, battery and power grid techniques. Scientific software on the other hand requires knowledge of the specific target hardware architecture which implies adjustments of numerical methods and their implementation. Otherwise efficiency losses are inevitable and always imply too much energy spent. In the performance engineering studies for decades energy efficiency has been eclipsed by computational performance and only recently power and energy metrics started being included into performance models for numerical software.

This talk explores the possibilities of a system integration approach that brings together an unconventional compute cluster (based on Tegra line mobile GPUs) with a modern photovoltaic energy supply and hardware-oriented algorithms. We exemplify how a combination of market-available hardware that fits to a specific class of numerical workloads and specially tailored, hardware-oriented numerics can be combined to tackle the energy crisis: by lowering energy consumption as well as building the energy supply alongside its demand.

[7]TU Dortmund, Vogelpothsweg 87, 44227 Dortmund, Germany,
markus.geveler@math.tu-dortmund.de

Contributed Talks

Energy consumption of the Jacobi method: shared memory and single/double precision

Sue Thorne[1] Andrew Taylor[2]

Software developers frequently endeavor to ensure that the resulting code is *efficient*, where efficiency is normally measured by considering the wall clock execution time of the code. In the future, it is likely that HPC resources will be charged by the number of energy units consumed. Additionally, the reduction in power usage and energy consumption is of great importance when moving from petascale to exascale computing. Hence, there is now an increasing desire for codes to be *energy efficient*, that is, to minimise the amount of energy consumed whilst the code is run.

In this report, we consider three different architectures (ARMv8, Blue Gene/Q and Intel Xeon IvyBridge-based nodes) and compare how execution time, power usage and energy consumption changes as we increase the number of OpenMP threads being used within our benchmark code, the Jacobi Test Code, which implements the Jacobi method applied to a 3D Poisson problem. We will also compare what happens to the timings, energy consumption and power usage when we run the code in single and double precision. The different architectures all exhibit very different behaviours both in terms of what happens when we switch between single and double precision, and also what happens when we alter the number of threads. For two of the architectures, the energy consumption is not proportional to wall clock execution times and, for one of them, we demonstrate that halving the execution time can have minimal effect on the energy consumption.

1 Background and motivation

In software development, a large amount of work is done to ensure that the resulting code is *efficient*. Efficiency is normally measured by considering the wall clock execution time of the code and software developers often strive for their codes to scale well as the number of processes/threads increases. This is normally driven by the desire to "get a quicker answer" or "solve a larger problem in a given time-frame". Note that the generally used method for charging users of HPC resources is based on the number of node hours or core hours used. In the future, it is likely that HPC resources will be charged by the number of energy units consumed [5]. Additionally, the reduction in power usage and energy consumption is of great importance when moving from petascale to exascale computing.

[1]The Hartree Centre, Science and Technology Facilities Council, Rutherford Appleton Laboratory, Harwell Campus, Didcot, United Kingdom, OX11 0QX ,
sue.thorne@stfc.ac.uk

[2]The Hartree Centre, Science and Technology Facilities Council, Daresbury Laboratory, Sci-Tech Daresbury, Daresbury, Warrington, United Kingdom, WA4 4AD,
andrew.d.taylor@stfc.ac.uk

More specifically, the number of floating point operations per second needs to increase by a factor of 1000 but the power consumption of the required supercomputer is limited to just a factor of 10 increase [3]. Thus, vendors are producing new energy efficient chips but different types of chip can have very different energy consumption and power usage behaviours.

In this report, we will consider three different architectures and compare how execution time, power usage and energy consumption changes as we increase the number of OpenMP threads being used and also alter the problem size within the Jacobi Test Code [2]. We will also compare what happens when we run the code in single and double precision. It is important to note that it is not uncommon for codes to be run with double precision accuracy when the underlying problem is of much lower accuracy and, hence, time and energy may have been wasted by computing a solution to many more significant figures than necessary. Additionally, it may be possible to use mixed-precision within a code, i.e., some components can be run in single precision and techniques are used to recover double precision accuracy.

2 Jacobi Test Code

The Jacobi Test Code (JTC) contains a number of implementations of the Jacobi algorithm for solving a simple 3D partial differential equation (Poisson's equation on a cuboid) [2]. In this report, we will concentrate on the *baseline-opt* implementation and will refer to this as `JTC_CORE`. The baseline implementation of the Jacobi algorithm consists of three nested loops with basic loop optimisations and, in addition, the version used for the tests in this report uses an optimised form of the inner loop to enable the compiler to vectorise the loop.

At the i-th iteration in `JTC_CORE`, each entry in a vector v_i is computed by averaging the values in v_{i-1} at neighbouring grid points. This stencil-type approach is found in many application codes and, hence, the JTC is a good representation of the work performed in the kernel of many codes.

In all of our tests, we set $nruns = 10$ and $niter = 50$, that is, `JTC_CORE` is run 10 times and 50 iterations used in each run. We consider cubic problems with N_x grid points in the x, y and z directions, and, in this paper, we set $N_x = 502$ and 892. The total problem size is $(N_x - 2)^3$.

3 Architectures considered

We ran the JTC on three different architectures, which are summarised in Table 1. These resources were provided by STFC's The Hartree Centre [1]. In all of our results, we use the term *execution time* to refer to the wall clock time for running the test problem. The number of threads, $nthr$, used will be architecture dependent. We always repeat our tests five times for each problem size with each configuration and report the mean values. We cannot do direct comparisons of the energy usage on these different architectures because the domains over which we could measure the power usage varied. On ACE, we were able to obtain a trace of the power consumption for the whole of the node being used; for Bantam, the trace was over the whole of the node board (traces over subdomains of the node board were available but not over an individual node); for Neale, the trace was over the A2 node power domain (CPU execution units and memory) on the individual node being used.

Name	ACE	Bantam	Neale
ISA	ARMv8.1	PowerPC	x86
Processor	Cavium ThunderX	IBM A2 PowerPC	Intel (IvyBridge) Xeon ES-2650v2
Processors/Cores per node	2/96	1/16	2/8 (16 threads)
Clockspeed	2.1GHz	1.6GHz	2.6GHz (3.6 GHz Max)
L1 Cache	78(I) + 32(D) kB	16 kB	32 kB
L2 Cache	16 MB	32 MB	256 kB
L3 Cache	-	-	20 MB
Memory per node	132 GB	16 GB	64 GB
Compiler	GNU gcc (6.1.0)	XL mpicc (12.1)	Intel icc (5.0.3)
Flags	-mcpu=thunderx+lse -03 -std=c99 -static -fopenmp	-03 -std=c99 -fopenmp	-03 -std=c99 -static -fopenmp
Power domain	node	node board	A2 node

Table 1: Summary of the three different architectures.

		$N_x = 502$			$N_x = 892$		
precision	nthr	time(s)	energy(kJ)	power(W)	time(s)	energy(kJ)	power(W)
single	12	140.6	32.8	233.3	881.3	205.6	233.7
	24	86.5	20.9	241.5	469.9	114.9	244.9
	48	50.7	13.0	255.4	266.1	69.8	262.5
	96	31.3	8.6	275.3	169.7	48.3	284.7
double	12	260.6	60.8	233.8	1382.2	324.8	236.9
	24	161.3	39.0	240.5	898.7	221.2	242.7
	48	97.7	25.0	255.7	482.3	128.2	259.4
	96	59.9	16.5	276.3	324.8	93.7	290.1

Table 2: Average execution time, energy consumption and power usage on ACE for the JTC run in single and double precision with different numbers of threads and $N_x = N_y = N_z = 502$ or 892.

4 Numerical Results

In Table 2, we compare the execution time, energy consumption and power usage of a node on ACE when $nthr = 12, 24, 48$ and 96. We also compare the single and double precision versions of the JTC. We start by observing that as we increase the number of threads, the power usage steadily increases. Hence, for $N_x = 502$ and single precision, switching from 12 to 96 threads decreases the execution time by a factor of 4.49 but the energy consumption is only decreased by a factor of 3.81. Thus, on the ARMv8.1, increasing the number of threads has a greater affect on the execution time than the energy consumption. Switching from double precision to single precision has little affect on the power usage but the execution time drops by between 35 and 50%.

We compare the execution time, energy consumption and power usage of a node board on Bantam for $nthr = 1, 2, 4, 8$ and 16 in Table 3. We start by noting that the behaviour is different to that of ACE. In particular, the power usage stays almost static when the number of threads increases and, hence, increasing the number of threads decreases the execution time and energy consumption by similar factors. For the double precision version, increasing the number of threads from 8 to 16 has little affect on the execution time. This is due to increased data traffic causing a large increase in the number of L2 cache misses [4]. The more interesting observations come from comparing the single and double precision versions. For $nthr = 1, 2$ and 4, the single precision version is slower than the

		$N_x = 502$			$N_x = 892$		
precision	nthr	time(s)	energy(MJ)	power(kW)	time(s)	energy(MJ)	power(kW)
single	1	597.8	1.047	1.751	3339	5.846	1.751
	2	267.1	0.455	1.702	1482	2.540	1.714
	4	139.5	0.245	1.753	781.3	1.365	1.747
	8	72.6	0.128	1.766	419.4	0.735	1.752
	16	41.9	0.074	1.761	239.0	0.420	1.758
double	1	480.5	0.825	1.716	2698	4.644	1.721
	2	210.9	0.371	1.759	1146	2.008	1.752
	4	105.8	0.185	1.753	602.9	1.062	1.762
	8	77.0	0.136	1.769	428.5	0.762	1.778
	16	76.2	0.134	1.761	421.2	0.730	1.733

Table 3: Average execution time, energy consumption and power usage on Bantam for the JTC run in single and double precision with different numbers of threads and $N_x = N_y = N_z = 502$ or 902.

double precision version. For these values of *nthr*, the low levels of parallelism mean that the execution time is dominated by the time to perform the float point operations and the time spent doing data movement is a secondary consideration. The Blue Gene/Q uses Quad-Process eXtension, which means that single precision arithmetic is performed by converting the single precision input data to double precision, performing the floating point operation in double precision and converting the result back to single precision. Therefore, each single precision operation takes longer than a double precision operation (and requires more energy), which accounts for the single precision version taking longer for $nthr \leq 4$. However, data movement between threads should be faster for single precision reals compared to double precision reals. For $nthr = 8$ and 16, the extra parallelism means that data movement now dominates. For $nthr = 8$, this domination means that the single precision version is marginally faster than the double precision version. Switching to $nthr = 16$, the data movement becomes even more dominant and the gains in using single precision over double precision for the larger test problems result in the execution time (and energy consumption) of the single precision version being roughly 55% of that of the double precision version.

Finally, in Table 4, we compare the execution time, energy consumption and power usage by the A2 node domain on Neale for $nthr = 1, 2, 4, 8$ and 16. We start by noting that switching from 8 to 16 threads generally increases the execution times. As with ACE, increasing the number of threads increases the power usage but the increase is a lot more dramatic for the Intel Xeon IvyBridge chip. Indeed, the large increase in power usage when moving from 8 to 16 threads means that the increase in energy consumption is more pronounced than the increase in execution time. Also switching from one to two threads almost halves the execution time, the energy consumption only has a small decrease because the power usage almost doubled. The double precision variant takes roughly twice the time of the single precision variant and, since the power usage is the same for both versions, we see a similar drop in the energy consumption.

		$N_x = 502$			$N_x = 892$		
precision	nthr	time(s)	energy(kJ)	power(W)	time(s)	energy(kJ)	power(W)
single	1	72.7	1.875	25.9	431.7	11.14	25.8
	2	36.9	1.621	43.8	219.0	9.63	43.8
	4	20.4	1.044	50.8	162.6	8.19	50.1
	8	16.2	0.967	59.5	110.0	6.62	60.5
	16	17.9	1.344	75.5	97.4	7.36	75.9
double	1	134.9	3.482	25.8	1088	28.13	25.9
	2	68.4	3.069	44.9	549.5	24.55	44.7
	4	41.1	2.128	52.5	354.0	18.09	51.1
	8	35.2	2.191	62.3	233.4	14.19	60.8
	16	41.2	4.036	73.7	235.0	17.26	73.5

Table 4: Average execution time, energy consumption and power usage on Neale for the JTC run in single and double precision with different numbers of threads and $N_x = N_y = N_z = 502$ or 902.

5 Conclusions

We have compared the behaviour of the execution time, energy consumption and power usage of the JTC for varying numbers of threads and precisions on three different architectures. None of the architectures exhibited the same behaviour. On the Blue Gene/Q, the energy consumption was directly related to the execution time but we only saw gains in using single precision over double precision when the data movement dominated the computation. For the ARMv8.1 architecture, increasing the number of threads resulted in the power usage increasing and, hence, reductions in energy consumption were not as big as any reductions in execution times. The popular Intel Xeon IvyBridge architecture had much more dramatic increases in power usage as the number of threads increased and, hence, although the execution times can be large, the energy consumption barely dropped.

We therefore conclude that, for a code that is going to be run on differing architectures, it will be difficult to produce a code that scales well with respect to execution time and energy consumption on all of the architectures. We also observe that, if single precision can be used effectively within a mixed precision code then, on some architectures, there may be benefits in both execution time and energy consumption.

Code Availability

The source code of the implementations used to compute the presented results can be obtained from:

https://ccpforge.cse.rl.ac.uk/gf/project/asearchtest/ and is authored by: A. Taylor, V. Szeremi, L. Anton and M. Mawson

References

[1] See http://www.hartree.stfc.ac.uk/hartree/.

[2] L. Anton, M. Mawson, A. D. Taylor, and V. Szeremi, *Performance profiles with Jacobi Test Code suite*, tech. rep., 2015. http://purl.org/net/epubs/work/12145766.

[3] S. Ashby, P. Beckman, J. Chen, P. Colella, B. Collins, D. Crawford, J. Dongarra, D. Kothe, R. Lusk, P. Messina, and Others, *The opportunities and challenges of exascale computing*, summary report of the advanced scientific computing advisory committee (ASCAC) subcommittee at the US Department of Energy Office of Science, (2010).

[4] T. Byrne, M. Mawson, A. D. Taylor, and H. S. Thorne, *Energy consumption of the Jacobi Test Code on the Blue Gene/Q: does using single precision reduce energy consumption?*, Tech. Rep. RAL-TR-2016-005, 2016. http://purl.org/net/epubs/work/24764929.

[5] A. Langer, H. Dokania, L. V. Kale, and U. S. Palekar, *Analyzing energy-time tradeoff in power overprovisioned HPC data centers*, in 2015 IEEE International Parallel & Distributed Processing Symposium Workshops, Hyderabad, India, May 25-29, 2015, 2015.

Improving energy efficiency of MPDATA on GPU-based supercomputers using mixed precision arithmetic

Krzysztof Rojek[1] Roman Wyrzykowski[2]

In this work, we propose a method that allows us to decrease the energy consumption in supercomputing centers. Our method is based on applying the mixed precision arithmetic, and its use is demonstrated for a real-life scientific application called MPDATA. All the tests are executed on two GPU-based clusters. The first one, the Piz Daint supercomputer ranked 8th at the TOP500 list (Nov. 2016), is equipped with the most powerful GPU accelerators - NVIDIA Tesla P100 that are based on the NVIDIA Pascal GPU architecture. The second one, the MICLAB cluster, is equipped with the most advanced Kepler based GPUs - NVIDIA Tesla K80.
Our research show that using the proposed technique we are able to provide a very high accuracy of computation and significantly decrease the energy consumption. The correctness and accuracy of our implementation are examined in a standard 3D solid body rotation test case.

1 Introduction

The energy consumption is a critical issue due to the significant increase of operation costs in modern computer systems [2]. In consequence, reducing the energy consumption turns into the primary objective for scientific and industrial environments, which are related to large-scale calculations. It is estimated that reducing the energy consumption by one megawatt could save about 1 million $ per year. Given the rapidly climbing power bills, as well as the negative impact of energy production technologies on the environment, achieving power and energy efficiency of parallel systems and applications has become one of the most challenging issues.

In this work, we focus on reducing the energy consumption in big supercomputing centers. We would like to propose to consider this issue from the point of view of average users. Therefore, we need to face with a common problem for users of such centers. A lot of approaches to reducing the energy consumption require some special user authorizations to apply techniques required to modify the hardware configuration (frequency, switching nodes off, powering cores down). However, an average user does not have access to change such characteristics while executing his/her time and energy consuming simulations.

2 Problem Overview

We propose to apply the mixed precision arithmetic [10, 4] under some constraints to reduce the energy consumption. The constraints are related to the accuracy of simulations expressed by three types of errors: diffusion error, phase error, and L2 norm. The

[1]Czestochowa University of Technology, Dabrowskiego 69, 42-201 Czestochowa, Poland, krojek@icis.pcz.pl

[2]Czestochowa University of Technology, roman@icis.pcz.pl

correctness of our technique is examined based on a real-life scientific applications, Multidimensional Positive Definite Advection Transport Algorithm (MPDATA) [9], with a standard 3D solid body rotation test case.

The 3D MPDATA application corresponds to an iterative algorithm that solves the continuity equation describing the advection of a nondiffusive quantity Ψ in a flow field:

$$\frac{\partial \Psi}{\partial t} \; + \; div(V\Psi) = 0, \tag{1}$$

where V is the velocity vector that is stored in three separate arrays $v1$, $v2$, and $v3$; each of them stores the velocity in one direction: i, j, and k, respectively.

MPDATA is the main module of the multiscale fluid model EULAG [5]. The model is an innovative solver in the field of numerical modeling of multiscale atmospheric flows. The algorithm is positive defined and by appropriate flux correction can be also monotonic. This is a desirable feature for advection of positive definite variables such as specific humidity, cloud water, cloud ice, rain, snow, aerosol particles, and gaseous substances. The spatial discretization of MPDATA is based on finite difference approximations. The algorithm is iterative and fast convergent.

In fact, to implement this algorithm we use 11 arrays: $v1, v2, v3, f1, f2, f3$ represents velocities in each direction ($f1, f2, f3$ are required to store intermediate results); x, xP correspond to the scalar quantity Ψ (xP - output of the odd iteration or time step of MPDATA, and input of the even time step; x - vice versa); h represents the vector of density, while cp, cn store intermediate results. Our current GPU implementation [6] consists of 4 GPU kernels which perform a sequence of stencil computations.

The proposed implementation of MPDATA is examined using the standard 3D solid-body rotation test [8], in which a sphere of radius R is advected with a constant angular velocity $\omega = 0.1$ around the domain diagonal axis. The velocity field is stationary and divergent free. The initial distribution of the tracer inside the sphere is defined as $\Psi(r) = 4(1 - r/R)$, where r is the distance from the center of the sphere.

Our numerical simulations have been performed at regular grids ($\Delta x = \Delta y = \Delta z$) with different resolution, namely 41^3, 81^3, 121^3. The size of time step depends on the grid resolution. Assuming the grid spacing equal to 2 ($\Delta x, \Delta y, \Delta z = 2$) and the time increment $\Delta t = 0.2$, we estimate the maximum value of the Courant number as $C_r = 0.29$.

To quantify the accuracy of numerical results we use the following statistical measures:

- phase error defined as the distance between the exact maximum position (at $t = 0$) and that computed after one full rotation:

$$R_{phase} = \sqrt{(i^{start} - i^{end})^2 + (j^{start} - j^{end})^2 + (k^{start} - k^{end})^2} \tag{2}$$

- diffusion error (q represents the quantity Ψ):

$$R_{diff} = max(q^{start}) - max(q^{end}) \tag{3}$$

- L2-norm:

$$R_{L2} = \sqrt{\frac{1}{N} \sum_{i}^{N} (q_i^{start} - q_i^{end})^2} \tag{4}$$

3 Main Contribution

This approach is aiming at reducing the energy consumption beyond using hardware-dependent methods. The advantage of our method is that it does not require any special user authorizations. Thanks to that it can be widely applied to most advanced supercomputing centers by all the users with the standard access. The proposed method minimizes the energy consumption by selecting the precision for each of the MPDATA arrays. Decreasing the precision from double to float decreases the size of an array by the factor of two, as well as memory traffic, while the execution time is reducing by a factor depending on hardware platforms, since the single precision arithmetic is twice faster than the double one using NVIDIA Tesla P100 GPUs and three times faster for Kepler-based GPUs.

Our assumptions are as follows:

- reducing the precision for an array reduces the simulation accuracy;
- reducing the precision for an array decrease the energy consumption;
- reducing the precision for a number of arrays at once has different impact on the simulation accuracy and energy consumption compared to separate reductions.

The constraints are presented below:

- the phase error R_{phase} should not exceed the distance between two nodes in the computational grid;
- the diffusion error R_{diff} should be lower than $R_{diff}^{double} + \Delta R_{diff}$;
- the value R_{L2} of L2-norm should be lower than $R_{L2}^{double} + \Delta R_{L2}$.

As a rule of thumb, we propose to increase the values of errors R_{diff}, R_{L2} by $\Delta R_{diff}, \Delta R_{L2} = 0.15$ in relation to the errors R_{diff}^{double}, R_{L2}^{double} achieved using the double precision format.

To implement the proposed method, the following procedure is executed for MPDATA:

1. Execute the test run using the double precision arithmetic for all the arrays.
2. Execute 11 tests: for each test set one array to float.
3. Run an estimator to select a near optimal solution.
4. If errors are too high, then increase the precision for the last array changed by the estimator; otherwise, decrease the precision for the first array unchanged by the estimator.
5. Repeat steps 3-4 until two consecutive iterations return the same set of precisions.

The estimator collects data from the tests, including the values of energy consumption and errors, and calculates the ratio $r = \delta E / \delta R_{L2}$ for each array, where δE and δR_{L2} are differences between respectively the energy consumption and L2-norm measured for the double and float formats. Then the estimator sorts the arrays by the ratio r, from the highest to the lowest. Finally, it sets the arrays (in the sorted order) to float until the adopted constraints are satisfied, and executes a new test using the new precisions for the selected arrays.

The preliminary tests were performed using the CUDA+MPI environment on the Piz Daint supercomputer [1] equipped with NVIDIA P100 GPUs, and MICLAB cluster

[3] containing NVIDIA K80 GPUs. These tests show that our method allows reducing the energy consumed by the application from 25% to 35% on Piz Daint and from 33% to 43% on MICLAB, considering only the energy consumed by the graphic cards. Furthermore, it allows users to decrease the energy consumption resulted from their simulations without any special access to machines such as the superuser account. Using the mixed precision arithmetic, our approach based permits us to achieve the energy reduction at the similar level as the DVFS technique investigated in paper [7]. What is also important, unlike the DVFS technique, the proposed method reduces the execution time of MPDATA.

Acknowledgements

This work was supported by the National Science Centre, Poland under grant no. UMO-2015/17/D/ST6/04059, and by the grant from the Swiss National Supercomputing Centre (CSCS) under project ID d25. The authors are grateful to the Czestochowa University of Technology for granting access to NVIDIA Tesla K80 GPU providing by the MICLAB project No. POIG.02.03.00.24-093/13.

References

[1] CSCS: Swiss National Supercomputing Centre, *http://www.cscs.ch/*

[2] E. Meneses, O. Sarood, L.V. Kale, *Energy profile of rollback-recovery strategies in high performance computing*, Parallel Computing 40, 2014, pp. 536–547

[3] MICLAB: Pilot Laboratory of Massively Parallel Systems, *http://miclab.pl*

[4] R. Nathan, H. Naeimi, D.J. Sorin, X. Sun, *Profile-Driven Automated Mixed Precision*, https://arxiv.org/pdf/1606.00251

[5] J.M. Prusa, P.K. Smolarkiewicz, A.A. Wyszogrodzki, *EULAG, a computational model for multiscale flows*, Computers & Fluids 37(9), 2008, pp. 1193–1207

[6] K. Rojek, R. Wyrzykowski, *Parallelization of 3D MPDATA algorithm using many graphics processors*, Lect. Notes Comput. Sci. 9251, 2015, pp. 445–457

[7] K. Rojek, A. Illic, R. Wyrzykowski, L. Sousa, *Energy-aware Mechanism for Stencil-Based MPDATA Algorithm with Constraints*, Concurrency and Computation: Practice and Experience, 29(8), 2017

[8] B. Rosa, et al., *Adaptation of multidimensional positive definite advection transport algorithm to modern high-performance computing platforms*, Int. Journal of Modeling and Optimization 5(3), 2015, pp. 171-176

[9] P.K. Smolarkiewicz , L.G. Margolin, *MPDATA: A Finite-Difference Solver for Geophysical Flows*, Journal of Computational Physics 140(2), 1998, pp. 459-480

[10] P. Igounet, E. Dufrechou, M. Pedemonte, P. Ezzatti, *A Study on Mixed Precision Techniques for a GPU-based SIP Solver*, 2012 Third Workshop on Applications for Multi-Core Architectures (WAMCA), 2012, pp. 7–12.

Studying mixed precision techniques for the solution of algebraic Riccati equations

Peter Benner[1] Ernesto Dufrechou[2] Pablo Ezzatti[3] Alfredo Remón[4]

We evaluate different algorithms and the use of a mixed-precision approach for the solution of Algebraic Riccati Equations (AREs). The mixed-precision method obtains an approximation to the solution using single-precision arithmetic and then, this approximation is improved via a cheap iterative refinement. Some numerical results show that the mixed-precision solver reports time and energy savings and also provides similar or even more accurate solutions than well-known methods like the Sign Function or SDA on CPU-GPU platforms.

1 Introduction

We consider the solution of the algebraic Riccati equation (ARE)

$$0 = \mathcal{R}_c(X) \;:=\; Q + A^T X + XA - XGX, \tag{1}$$

where A, Q and $G \in \mathbb{R}^{n\times n}$ are given, and $X \in \mathbb{R}^{n\times n}$ is the sought-after solution. Under certain conditions [8], the ARE (1) has a unique c-stabilizing solution X_c, which is symmetric positive semidefinite. (Here, X_c c-stabilizing means that $A_c := A - GX_c$ is c-stable; i.e., it has all its eigenvalues in the open left half plane.)

The solution of AREs is required in some scientific and engineering applications, e.g., in linear quadratic optimal control (LQOC) and model order reduction problems. It is a computationally intensive operation that involves $\mathcal{O}(n^3)$ floating-point operations (flops) and therefore, the use of high performance computing techniques and hardware is necessary whenever n takes moderate to large values ($n > 1,000$). Software packages as MESS[1], PLiC[2] or the MATLAB Control System Toolbox$^{\text{TM}}$ provide support for the solution of AREs.

2 Solution of AREs

A number of methods have been proposed for the solution of AREs (e.g., see [6]). In this section we briefly review two of the more popular, the Sign Function and the Structure-Preserving Doubling Algorithm (SDA) methods. Additionally, we review an iterative refinement scheme that mixes single precision (SP) and double precision (DP) arithmetic computations to get the maximum performance of the underlying hardware.

[1]Max Planck Institute for Dynamics of Complex Technical Systems, 30.106-Magdeburg, Germany, benner@mpi-magdeburg.mpg.de

[2]Facultad de Ingeniería, Universidad de la República, Montevideo, Uruguay, edufrechou@fing.edu.uy

[3]Facultad de Ingeniería, Universidad de la República, Montevideo, Uruguay, pezzatti@fing.edu.uy

[4]Max Planck Institute for Dynamics of Complex Technical Systems, 30.106-Magdeburg, Germany, remon@mpi-magdeburg.mpg.de

2.1 The Sign Function method

The solution of an ARE (1) can be defined by the invariant subspaces of the pencil $H - \lambda I_{2n}$, where H is the Hamiltonian matrix defined as $H = \begin{bmatrix} A & G \\ -Q & -A^T \end{bmatrix}$. Additionally, it can be shown that from a basis of the H-invariant subspace corresponding to the n eigenvalues in the open left half of the complex plane, the c-stabilizing solution of the associated ARE [4] can be obtained. This solution can be computed by the Sign Function of H, $\text{sign}(H) = Y = \begin{bmatrix} Y_{00} & Y_{01} \\ Y_{10} & Y_{11} \end{bmatrix}$, and then solving an overdetermined system (e.g., via the least squares method). The procedure is summarized in Algorithm `GECRSG`.

Algorithm `GECRSG`

$H_0 := \begin{bmatrix} A & G \\ -Q & -A^T \end{bmatrix}$

`for` $k = 0, 1, 2, \ldots$ *until convergence*

$\quad H_{k+1} := \frac{1}{2}\left(H_k + H_k^{-1}\right)$ $\qquad (16n^3$ flops$)$

`Solve` $\begin{bmatrix} Y_{11} \\ Y_{12} + I_n \end{bmatrix} X = \begin{bmatrix} I_n - Y_{10} \\ -Y_{00} \end{bmatrix}$ $\qquad (13n^3$ flops$)$

Note that the dimension of H doubles that of A and hence, a high performance matrix inversion kernel is mandatory to enable the solution of large problems. However, `GECRSG` exhibits a remarkable convergence rate that makes it very appealing. The variant here evaluated employs a highly tuned CPU-GPU matrix inversion kernel, see [5] for details.

2.2 The Structure-Preserving Doubling Algorithm (SDA)

In the last years, the SDA has received considerable attention as an ARE solver because of its simplicity, efficiency, and convergence properties [7].

Algorithm `GESDA` reflects a basic implementation of the SDA for the solution of an ARE. The major operations (from the computational point of view) are annotated to their right with the cost of a basic implementation. Let us consider only the iterative loop:

- The cost of the algorithm is $(2/3 + 16)n^3$ flops per iteration. Its high cost can be partially compensated by the parallel efficiency of the operations involved in the routine, namely, matrix-matrix products and linear system solves.

- A practical convergence criterion is to check during the iteration for

$$\frac{\|Y_k\|_F}{\|X_{k+1}\|_F} < \tau_S, \tag{2}$$

  with $\tau_S = \sqrt{\varepsilon} \cdot n$, and perform then 2 additional steps. The convergence of the iteration is asymptotically quadratic, which ensures the maximum attainable accuracy.

Algorithm `GESDA`

$\gamma := \max(1, 2\,\|A\|_F)$
$\hat{A} := A - \gamma I_n$
$\hat{Q} := Q\hat{A}^{-1}$ $((2/3+2)n^3$ flops)
$\hat{W} := (\hat{A}^T + \hat{Q}G)^{-1}$ $(4n^3$ flops)
$A_0 := I_n + 2\gamma\hat{W}^T$
$G_0 := 2\gamma(\hat{A}^{-1}G)\hat{W}$ $((2/3+4)n^3$ flops)
$X_0 := 2\gamma\hat{W}\hat{Q}$ $(2n^3$ flops)
`for` $k = 0, 1, 2, \ldots$ *until convergence*
  $\hat{W} := G_k X_k$ $(2n^3$ flops)
  $\hat{A} := (I_n + \hat{W})^{-1} A_k$ $((2/3+2)n^3$ flops)
  $Y_k := \hat{A} X_k A_k$ $(4n^3$ flops)
  $X_{k+1} := X_k + Y_k$
  `if not` convergence
    $G_{k+1} := G_k + A_k G_k (I_n + \hat{W}^T)^{-1} A_k^T$ $(6n^3$ flops)
    $A_{k+1} := A_k \hat{A}$ $(2n^3$ flops)
  `end if`

The implementation evaluated in this work executes the most time consuming operations in the GPU while operations that exhibit a fine-grain parallelism are performed in the CPU. Whenever it is possible, both processors concurrently perform their tasks, reporting significant time savings. Finally, the computation of inverses is replaced by the use of the LU factorization of the related matrix.

3 A mixed-precision ARE solver

In Benner et al. [3], the authors describe a Newton-like method for the solution of an ARE. Given an approximation to the solution of the ARE, X_0, the procedure (Algorithm `GEIR`) performs an iterative refinement that successively approximates the solution X until the desired precision is reached. At every step, `GEIR` solves a Lyapunov equation.

Algorithm `GEIR`**:**

`for` $k = 0, 1, 2, \ldots$ *until convergence*
  $P_k := Q + A^T X_k + X_k A - X_k G X_k$
  `Solve` $\left(A^T - GX_k\right) N_k + N_k \left(A^T - GX_k\right) = P_k$
  $X_{k+1} := X_k + N_k$

In practice, provided a relatively accurate X_0, a few steps of algorithm `GEIR` are enough to get the desired solution as this procedure is a variant of Newton's method for AREs, indicating quadratic convergence. The suitability of `GEIR` requires of a cheap method to compute X_0 and an efficient Lyapunov solver. The initial approximation, X_0, can be efficiently obtained executing some steps of `GESDA`, which can even be performed using SP arithmetic. This way, the solver benefits from the larger performance that the hardware offers in SP arithmetic computations (Intel CPUs are 2× faster and this factor is larger for NVIDIA GPUs). An economic Lyapunov solver was presented in [5]. The solver implements the Sign Function and relies on a tuned CPU-GPU matrix inversion kernel.

		Mojigata	Hetfield
GPU	Processor	NVIDIA K40 "Kepler" GK110B	NVIDIA TitanX "Maxwell" GM220
	# Cores	2,880	3,072
	Memory	12 GB GDDR5	12 GB GDDR5
CPU	Processor	i7-4770	i7-6700
	# Cores	4	4
	Frequency	3.40 GHz	3.40 GHz
	Main memory	16 GB DDR3	64 GB DDR3
SW	Compiler	`icc` 14.0.0	`icc` 14.0.0
	CUDA Version	6.5	8.0

Table 1: Platforms employed in the evaluation

4 Experimental evaluation

The evaluation focuses on two aspects, the time to solution and the energy consumption. The test-cases evaluated were extracted from the Oberwolfach[5] benchmark collection. In particular, two instances of the steel profile (with $n = 1,357$ and $5,177$) and another from the flow meter problem ($n = 9,669$). Although the three cases permit the use of a low-rank solver, only the Sign Function implementation takes advantage of this feature. Table 1 describes the hardware employed in this evaluation. A remarkable difference between both platforms is that the performance of the GPU in Hetfield is 32× larger when using SP arithmetic than using DP, while this factor reduces to 3× in Mojigata. However, when using hybrid (CPU-GPU) variants these ratios can be smoothed.

Power/energy was measured via RAPL to gauge the consumption from the server's package and DRAM, and the NVML library to obtain the dissipation from the GPU.

We first evaluate the computational performance of the Sign Function and the SDA fixing the number of iterations of each solver so that they reach comparable accuracy results. The residual error is computed as

$$\text{RRes} = \|\mathcal{R}_c(X^*)\|_F / (\|Q\|_F + 2\|A\|_F \|X^*\|_F + \|G\|_F \|A\|_F^2). \tag{3}$$

The results summarized in Tables 2 and 3 show that both solvers behave differently in the two platforms. While in Hetfield the Sign Function solver is clearly faster, Mojigata seems to slightly favor the SDA solver. This behavior is explained by noting that the implementation in SDA is more suitable to the GPU architecture, and the GPU in Mojigata is more powerful (when using DP arithmetic). On the other hand, the Sign Function implementation features a better CPU-GPU load-balance.

		Mojigata			Hetfield			
problem	# steps	sign func.	solver	total	sign func.	solver	total	rel. res.
rail_1357	10	3.63	0.37	4.05	4.78	0.30	5.09	1.51E-17
rail_5177	11	71.81	10.45	82.81	154.52	11.76	166.55	4.96E-17
flow_9669	13	411.34	74.90	488.19	1093.64	72.59	1167.12	2.12E-10

Table 2: Run-times (in sec.) of the Sign Function solver.

For the mixed-precision scheme, we execute the proposal modifying the number of SDA and iterative refinement and steps. At every step of the iterative refinement, a Lyapunov equation is solved via the Sign Function method. Once again we fixed the parameters so that a comparable accuracy is reached. The results, summarized in Table 4, demonstrate

[5]Available at https://portal.uni-freiburg.de/imteksimulation/downloads/benchmark

PROBLEM	# STEPS	MOJIGATA	HETFIELD	REL. RES.
RAIL_1357	24	1.85	6.37	4.96E-16
RAIL_5177	27	75.89	316.67	4.61E-16
FLOW_9669	24	401.79	1805.26	7.99E-12

Table 3: Run-times (in sec.) of the SDA solver.

that the mixed-precision strategy is more effective in HETFIELD, where the GPU exhibits a higher performance in SP arithmetic. But it also outperforms the DP solvers (except for the small case of SDA) on MOJIGATA. Regarding the parametrization of the solver, the experiments show that the effect of the number of steps performed by the SP solver on the accuracy reached diminishes as the dimension of the problem grows. As a consequence, the refinement steps have a strong impact on the final accuracy. This is specially relevant in the larger instance.

	#STEPS			MOJIGATA			HETFIELD			
PROBLEM	GESDA	Newton	Lyap.	GESDA	It. ref.	TOTAL	GESDA	It. ref.	TOTAL	Rel. res.
RAIL_1357	10	1	10	0.7	1.5	2.2	0.5	1.9	2.4	1.75E-14
	10	2	8	0.7	2.1	2.9	0.6	2.6	3.2	1.62E-14
	15	1	9	0.9	1.3	2.3	0.6	1.8	2.4	9.10E-15
	15	2	8	0.9	2.2	3.2	0.7	3.0	3.8	1.78E-15
	20	1	9	1.2	1.3	2.5	0.8	2.0	2.8	6.92E-16
	20	2	8	1.1	2.2	3.3	0.9	3.2	4.1	3.70E-16
RAIL_5177	10	1	10	19.8	30.1	50.0	13.0	61.0	74.0	6.53E-15
	10	2	9	19.0	42.4	61.41	12.6	104.3	117.0	4.99E-15
	15	1	10	26.6	23.7	50.08	17.4	61.8	79.1	1.44E-15
	15	2	9	26.4	44.2	70.60	17.3	104.9	122.3	1.00E-15
	20	1	10	33.3	30.2	63.56	21.8	62.0	83.8	7.42E-16
	20	2	9	33.3	39.4	72.66	21.8	105.2	127.0	1.31E-16
FLOW_9669	10	1	7	111.4	113.7	225.12	72.9	291.5	364.4	2.44E-09
	10	2	7	107.6	167.4	275.0	70.7	527.4	598.1	3.94E-13
	15	1	7	149.1	100.4	249.5	99.7	291.7	391.4	2.15E-09
	15	2	7	151.5	164.0	315.5	101.0	529.5	630.6	3.73E-13
	20	1	10	189.1	149.8	339.0	130.5	374.7	505.2	5.39E-10
	20	2	8	178.9	187.0	365.9	129.4	585.3	714.8	7.21E-15

Table 4: Run-times (in sec.) and relative residuals reported by the mixed-precision solver.

In a second experiment, we measure the energy consumption related with the best configuration of each method in MOJIGATA. Tables 5 and 6 show the energy consumption of the three solvers. In the mixed-precision case, the SP and DP stages are distinguished. From the reported results, it can be noticed that the savings in energy consumption of the mixed-precision method is slightly higher than the run-time counterpart. Furthermore, the improvement seems to increase with the dimension of the problem.

SOLVER	PROBLEM	# STEPS	TIME (S)	ENERGY (J)
SIGN FUNC.	RAIL_1357	10	4.0	563.88
	RAIL_5177	11	85.65	14493.7
	FLOW_9669	13	487.29	94516.5
SDA	RAIL_1357	24	2.16	437.2
	RAIL_5177	27	78.12	17730.2
	FLOW_9669	24	467.22	101077.1

Table 5: Energy and run-time evaluation of the double precision solvers.

PROBLEM	#STEPS			GESDA		IT. REFINEMENT		TOTAL
	GESDA	Newton	Lyap.	TIME (S)	ENERGY (J)	TIME (S)	ENERGY (J)	ENERGY (J)
RAIL_1357	10	2	9	0.71	118.3	2.04	309.4	427.7
RAIL_5177	10	2	10	19.55	3342.7	33.72	6554.0	9896.7
FLOW_9669	10	2	7	104.95	16459.0	115.97	24536.0	40995.0

Table 6: Energy and run-time evaluation of the mixed precision solver.

5 Concluding remarks

In this work we presented and evaluated a mixed precision variant for the solution of the Algebraic Riccati Equation. The experimental evaluation showed that the mixed-precision variant offers an interesting reduction on the execution time, compared to traditional methods like the Sign Function and SDA. In addition, the savings in energy consumption reported are even more important than the savings in run-time, what reinforces the convenience of the mixed-precision solver over DP solvers.

As future work, we will study a low rank variant of the mixed precision solver. Specifically, a combination of a low rank variant of the SDA or the Sign Function algorithm to obtain the initial solution and a low rank Lyapunov solver in the iterative refinement.

References

[1] *Matrix Equation Sparse Solver (MESS) library (`www.mpi-magdeburg.mpg.de/projects/mess/`).*

[2] *PLiC library (`www3.uji.es/~quintana/plic/plic/`).*

[3] P. BENNER AND R. BYERS, *An exact line search method for solving generalized continuous-time algebraic Riccati equations*, IEEE Trans. Autom. Control., 43 (1998), pp. 101–107.

[4] P. BENNER, R. BYERS, E. QUINTANA-ORTÍ, AND G. QUINTANA-ORTÍ, *Solving algebraic Riccati equations on parallel computers using Newton's method with exact line search*, Parallel Computing, 26 (2000), pp. 1345 – 1368.

[5] P. BENNER, P. EZZATTI, E. QUINTANA-ORTÍ, AND A. REMÓN, *Matrix inversion on CPU-GPU platforms with applications in control theory*, Concurrency and Computation: Practice and Experience, 25 (2013), pp. 1170–1182.

[6] D. BINI, B. IANNAZZO, AND B. MEINI, *Numerical Solution of Algebraic Riccati Equations*, Society for Industrial and Applied Mathematics, 2011.

[7] E.-W. CHU, H.-Y. FAN, AND W.-W. LIN, *A structure-preserving doubling algorithm for continuous-time algebraic riccati equations*, Linear Algebra and its Applications, 396 (2005), pp. 55–80.

[8] P. LANCASTER AND L. RODMAN, *Algebraic Riccati Equations*, Oxford University Press, 1995.

Energy-aware scheduling for parallel evolutionary algorithms in heterogeneous architectures

Julio Ortega, Juan José Escobar, Antonio Díaz, Jesús González, Miguel Damas[1]

The availability of mechanisms such as dynamic voltage and frequency scaling (DVFS) and heterogeneous architectures including processors with different power consumption profiles allow scheduling algorithms aware of both runtime and energy. In this paper, we propose and evaluate a scheduling strategy that takes into account the relative weights of the workloads and the frequencies and voltages of the different processing cores in a given heterogeneous parallel architecture either to save energy without increasing the running time or to reach a trade-off among time and energy. The parallel algorithms considered to evaluate the proposed scheduling procedure are master-worker evolutionary algorithms whose fitness functions demand high computing times and distribute the fitness evaluation of the individuals among the available cores. As many useful bioinformatics and data mining applications present this profile, the proposed energy-aware scheduling approach could be frequently applied. The experimental results obtained by simulation show relevant energy savings, with values depending on the characteristics of the heterogeneous architecture and on the workload profiles.

1 Background models

Any scheduling procedure locates tasks on the available processors according to predictions about their computational cost and the corresponding energy consumption. Therefore, the procedure also needs information about the characteristics of processors in the system where the tasks are executed. In this section, the models on energy consumption and computing time required by the given tasks are described.

The energy model used by the proposed scheduling procedure is estimated from the power consumption equations corresponding to CMOS circuits that include the terms associated to capacitive, short-circuit, and leakage power. As the most part of previous works, and assuming the capacitive term as the most significant, we will use it to estimate the power consumption in a processor as:

$$P = \beta \times f \times V^2 \tag{1}$$

Where parameter β is related with the product of the number of transistors switching in the processor per clock cycle and the total capacitance load, f is the clock frequency of the processor, and V is the supply voltage. Therefore, the energy E_i consumed by a given task i that requires C_i clock cycles in a processor with a supply voltage V_i can be estimated from (1) by

$$E_i = \beta \times f \times V_i^2 \times \frac{C_i}{f} = \beta \times V_i^2 \times C_i \tag{2}$$

[1]Dept. of Computer Architecture and Technology, CITIC, University of Granada (Spain), jortega@ugr.es, jjescobar@ugr.es, afdiaz@ugr.es, jesusgonzalez@ugr.es, mdamas@ugr.es

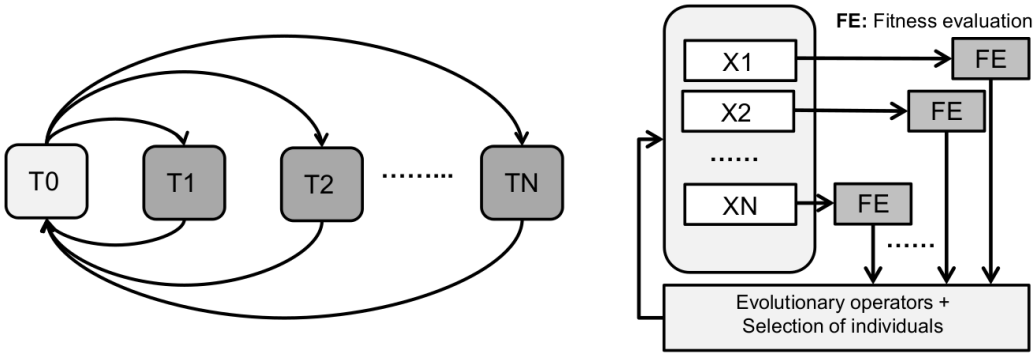

Figure 1: Task dependence graph considered (a), and evolutionary algorithm as example of application with such graph (b)

Whenever a processor is idle, there is also a so called indirect energy consumption that for a given processor k can be estimated by

$$E_k^{\text{idle}} = \beta \times f \times V_{\text{idle}}^2 \times t_k \tag{3}$$

where V_{idle} is the supply voltage of the processor in its idle state, and t_k is the amount of time in which processor k has been in this state. The tasks have to be located on the processors included in a heterogeneous platform with p processors, $P_j (j = 1, ..., p)$. Each processor P_j can operate at different voltage supply levels (VSL), $V_{j,l} (l = 1, ..., \omega(j))$, corresponding to different clock frequencies frequencies $f_{j,l} (l = 1, ..., \omega(j))$.

2 A bi-objective scheduling procedure

This section describes a scheduling procedure that allocates processors and frequencies to tasks trying to minimize both runtime and energy consumption. It can be applied to parallel programs whose tasks dependence graphs are shown in Figure 1.a. In this graph, tasks $T1, T2, ..., TN$ can be executed in parallel after task $T0$, and after synchronizing themselves once they have finished, task $T0$ is executed again and generates another set of parallel tasks $T1, T2, ..., TN$ executed in parallel, and so on. Moreover, the runtime of task $T0$, is negligible with respect to the runtime of each parallel task $T1, ..., TN$. Many useful applications can be parallelized according to the dependence graph of 1.a. Indeed, 1.b schematizes an evolutionary algorithm. Each generation, the fitness of the individuals in the population has to be evaluated according to some performance procedure that could demand a costly computation. For example, in [1] evolutionary multi-objective optimization is applied to solve a feature selection problem in a BCI application. The individuals of the population correspond to different sets of features that define the components of the patterns to be classified. These sets of features have to be evaluated by the accuracy of the classifier once it has been adjusted by using the training patterns characterized by the selected features. The iterations required to train the classifier usually require a high amount of computing time.The fitness evaluation needs between 97.36% (with 30 000 individuals in the population) and 99.93% of the runtime (for 120 individuals).

To define the scheduling strategy, we take into account four parameters, t_{MAX}, t_{max}, t_{min} and t_{MIN}. These parameters can be obtained from the highest and lowest clock cycles

values, C_i, required to complete the estimated workloads of the different tasks ($i = 1, ..., n$) and from the frequencies of the available processors, $f_{j,l}(j = 1, ..., p,\ l = 1, ..., \omega(j))$ as follows:

$$t_{MAX} = max(C_i(i = 1, ..., n))/min(f_{j,l}, (j = 1, ..., p,\ l = 1, ..., \omega(j))) \quad (4)$$
$$t_{max} = max(C_i(i = 1, ..., n))/max(f_{j,l}, (j = 1, ..., p,\ l = 1, ..., \omega(j))) \quad (5)$$
$$t_{min} = min(C_i(i = 1, ..., n))/min(f_{j,l}, (j = 1, ..., p,\ l = 1, ..., \omega(j))) \quad (6)$$
$$t_{MIN} = min(C_i(i = 1, ..., n))/max(f_{j,l}, (j = 1, ..., p,\ l = 1, ..., \omega(j))) \quad (7)$$

The parameter t_{MAX} is the time required by the task with the heaviest workload when it is executed in a processor running at the lowest frequency, the parameter t_{max} is the time required by the task with the highest workload in a processor running at the highest frequency. This way t_{MAX} and t_{max} respectively represent the highest and lowest running times that the heaviest task would require in the present heterogeneous platform. In the same way, t_{min} and t_{MIN} are, respectively, the highest and lowest running times for the lightest task.

It is also possible to define energy consumption parameters, EC_{MAX}, EC_{max}, EC_{min}, and EC_{MIN}, that respectively correspond to the runtimes t_{MAX}, t_{max}, t_{min}, and t_{MIN} as follows:

$$EC_{MAX} = \beta \times max(Ci(i = 1, ..., n)) \times [min(V_{j,l}, (j = 1, ..., p, l = 1, ..., \omega(j)))]^2 \quad (8)$$
$$EC_{max} = \beta \times max(Ci(i = 1, ..., n)) \times [max(V_{j,l}, (j = 1, ..., p, l = 1, ..., \omega(j)))]^2 \quad (9)$$
$$EC_{min} = \beta \times min(Ci(i = 1, ..., n)) \times [min(V_{j,l}, (j = 1, ..., p, l = 1, ..., \omega(j)))]^2 \quad (10)$$
$$EC_{MIN} = \beta \times min(Ci(i = 1, ..., n)) \times [max(V_{j,l}, (j = 1, ..., p, l = 1, ..., \omega(j)))]^2 \quad (11)$$

The parameters t_{MAX}, t_{max}, t_{min}, and t_{MIN} verify that $t_{MIN} < t_{max} < t_{MAX}$ and $t_{MIN} < t_{min} < t_{MAX}$ while EC_{MAX}, EC_{max}, EC_{min}, and EC_{MIN} verify that $EC_{min} < EC_{MAX} < EC_{max}$ and $EC_{min} < EC_{MIN} < EC_{max}$. Therefore, given a task i with C_i clock cycles, that have been allocated to a processor with supply voltage V_j and frequency f_j, it is possible to define two indexes, Δt and ΔE, with values between 0 and 1, and respectively related with the contribution of the task allocation to the runtime and to energy consumption:

$$\Delta t = \frac{\frac{C_i}{f_j} - t_{MIN}}{t_{MAX} - t_{MIN}} \quad (12)$$
$$\Delta E = \frac{K \times C_i \times V_j^2 - EC_{min}}{EC_{max} - EC_{min}} \quad (13)$$

To select a processor and the corresponding frequency for a given task, the scheduling algorithm uses a cost function that takes into account both the energy and runtime objectives through indexes Δt and ΔE. In our procedure we propose the cost function $\Delta(C_i, f_j) = \Delta t^a \times \Delta E^b$ with integers a and b greater than or equal to one. This cost function promotes allocations with low values of Δt and ΔE, and even lower values for one factor whenever the other factor grows. Depending on the relative values of a and b it is possible to give more relevance either to lower runtime or lower energy consumption. The proposed scheduling procedure is shown in Figure 2.

$C(i)(i = 1, ..., p)$ // Cycles of task i to be allocated to a processor
$C_{max} = \max(C(i)(i = 1, ..., p))$;
$C_{min} = \min(C(i)(i = 1, ..., p))$;

// Frequency i-th of processor j-th (FL frequency levels; p processors)
$F(i, j)(i = 1, ..., FL; j = 1, ...p)$
$F_{max} = \max(F(i, j)(i = 1, ..., FL; j = 1, ...p))$;
$F_{min} = \min(F(i, j)(i = 1, ..., FL; j = 1, ...p))$;
Compute t_{MAX}, t_{max}, t_{MIN}, t_{min}, and EC_{MAX}, EC_{max}, EC_{MIN}, EC_{min}

$C(i)(i = 1, ..., p)$ is sorted verifying $C(j) \leq C(j + 1)(j = 1, ..., p - 1)$;
for $i = 1 : p$
    // Select processor to locate $C(i)$ and frequency
    for $j = 1 : p$
        if processor j has not been previously selected
            for $k = 1 : FL$
                $\Delta(C(i), F(j, k)) = \Delta t^a \times \Delta E^b$;
            end;
        end;
    end;
    Select the frequency level of processor, s, not previously selected, for which is obtained the minimum value of $\Delta(C(i), F(j, k))$;
    Mark processor s as selected;
end;

Figure 2: Description of the energy-aware procedure for scheduling

3 Performance evaluation

In this section, we analyze the performance of the proposed scheduling procedure. Table 1 describes the three configurations of eight processors with different clock frequencies we have used in our experiments. As can be seen, each configuration corresponds to a different level of heterogeneity. Indeed, *conf1* is homogeneous as all the processors have the same levels of voltage and frequency. The configuration *conf2* includes two different processors while *conf3* includes four. More specifically, Table 1 provides the relative speeds with respect to the one achieved at the highest frequency, which is 1 GHz in all the configurations.

We have also used three different profiles for the evolutionary algorithm. They are defined by the lowest and the highest values for the computing cost of the fitness evaluation tasks, and the lowest difference between two different computing costs. The cost weights for the individuals in the population are randomly selected. This way, the benchmark $b30{\times}100$ includes tasks with computing costs between 100 and 3 000 cycles with cost differences multiple of 100 cycles. The computing costs of the tasks in $b300{\times}100$ go from 1 000 to 30 000 cycles being 100 cycles the lowest difference between tasks, and finally, the tasks in $b30{\times}1\,000$ go from 1 000 to 30 000 cycles with lowest differences of 1 000 cycles between tasks. Each of these files ($b30{\times}100$, $b300{\times}100$ and $b30{\times}1\,000$) includes 100 configurations of task costs with the aforementioned characteristics. These configurations have been randomly selected by using a standard uniform distribution. This way, the averages for the increments in runtime and energy consumption correspond to 100 different

%	P1	P2	P3	P4	P5	P6	P7	P8
conf1	100	100	100	100	100	100	100	100
	80	80	80	80	80	80	80	80
	50	50	50	50	50	50	50	50
conf2	80	80	80	80	100	100	100	100
	64	64	64	64	80	80	80	80
	40	40	40	40	50	50	50	50
conf3	60	60	80	80	90	90	100	100
	48	48	64	64	72	72	80	80
	30	30	40	40	45	45	50	50

Table 1: Relative speeds (in %) in the processors of the configurations used in the experiments

		(a,b)									
		$(1,1)$		$(1,3)$		$(2,1)$		$(3,1)$		$(3,2)$	
Bench.	Conf.	Δt	ΔE	Δt	ΔE	Δt	ΔE	Δt	ΔE	Δt	ΔE
*b*30×100	*conf1*	100.00	−55.80	100.00	−55.80	**0.00**	**0.00**	**0.00**	**0.00**	100.00	−55.40
	conf2	74.68	−60.96	74.68	−60.96	**6.78**	**−10.84**	**6.78**	**−10.78**	75.19	−60.44
	conf3	58.61	−64.18	58.61	−64.18	148.80	−11.74	**24.41**	**−16.05**	59.71	−63.36
*b*300×100	*conf1*	100.00	−55.80	100.00	−55.80	**0.00**	**0.00**	**0.00**	**0.00**	100.00	−55.40
	conf2	75.23	−61.85	75.23	−61.85	**9.37**	**−10.42**	**8.81**	**−10.27**	76.19	−61.05
	conf3	54.76	−65.06	54.76	−65.06	148.50	−11.96	**24.23**	**−15.78**	57.42	−63.60
*b*30×1 000	*conf1*	100.00	−55.80	100.00	−55.80	**0.00**	**0.00**	**0.00**	**0.00**	100.00	−55.40
	conf2	73.59	−60.95	73.59	−60.95	**8.34**	**−10.21**	**7.62**	**−9.97**	73.71	−60.28
	conf3	53.81	−64.18	53.81	−64.18	143.70	−11.82	**21.85**	**−15.87**	54.97	−63.02

Table 2: Averages of increments in runtime and energy consumptions. The values corresponding to the lowest runtime increments are shown in bold characters

cases.

Table 2 shows the results obtained for the averages of the increments in runtimes and energy consumptions by the proposed scheduling procedure for the different benchmarks. These benchmarks have been executed on the three considered configurations described in Table 1. The increments corresponds to the differences in runtime and energy consumption with respect to a random scheduling of the tasks across the different processors. Several couples of values for the parameters a and b in $\Delta(C_i, f_j) = \Delta t^a \times \Delta E^b$ have been considered.

As can be seen, in the homogeneous configuration *conf1* only it is possible to save energy for the (a, b) couples $(1, 1)$, $(1, 3)$, and $(3, 2)$, that also provide high decrements in the energy consumption in the heterogeneous configurations *conf2* and *conf3*. Nevertheless, in these last two configurations, the couples $(1, 1)$, $(1, 3)$, and $(3, 2)$ also determine increments in the runtime higher than 50%. The (a, b) couples $(2, 1)$ and $(3, 1)$ produce neither increments in the runtimes nor decreases in the energy consumption whenever the homogeneous *conf1* is considered.

In the heterogeneous configuration *conf2*, the couples $(2, 1)$ and $(3, 1)$ allow increments in the runtime much lower than those obtained with $(1, 1)$, $(1, 3)$, and $(3, 2)$ although the energy savings are also lower. In the other heterogeneous configuration, *conf3*, this behavior (i.e. lower increments in the runtime with decrements in the energy consumption) is only observed with couple $(3, 1)$. The couple $(2, 1)$ produces high increments in the

runtime (higher than 140%). From Table 2 it is also apparent that given a parameter couple (a, b), the averages in the increments in runtime and energy consumption are similar for the three different distributions of task costs considered ($b30\times100$, $b300\times100$ and $b30\times1\,000$).

4 Conclusions

This paper proposes a scheduling procedure for evolutionary algorithms, with fitness functions requiring high runtimes to be evaluated, that takes into account not only runtime but also energy consumption. The procedure is based on dynamic voltage and frequency scaling (DVFS) and it is useful in heterogeneous architectures including processors with different power consumption profiles. It uses an approximate estimation of the cost of the fitness evaluation task, to build a cost function, $\Delta(C_i, f_j) = \Delta t^a \times \Delta E^b$, including the effect of energy consumption and speed through two parameters, a and b, that control the trade-off between these two measures. The simulation experiments we have accomplished have shown that by using the adequate combination of those two parameters it is possible to control the strength of each component, runtime and energy consumption. This way, the averages values for the increments of speed and energy consumption obtained across different configurations of task costs show that our procedure is able to reach energy savings of more than 10% with a runtime increment of about 9%. In many configurations of tasks it has been also observed energy savings of about 10% without any increase in the runtime. Given a configuration and a couple of parameters, a and b, similar averages of increments in runtimes and energy consumption have been observed across the considered distribution of task costs.

A lot of researching work still has to be completed. On the one side a more detailed characterization of the heterogeneous configurations of processors in terms of their capabilities to make possible task allocations with the best speed and energy consumption figures would be very useful. The performance evaluation by using profiles of task costs corresponding to real applications should be also completed, along with measuring the real values for energy savings and speeds in the execution of the parallel codes in the available heterogeneous platforms. This way, the usefulness of the consumption models we use in our scheduling procedure would be demonstrated.

Acknowledgements

This work was supported by project TIN2015-67020-P, funded by the Spanish "Ministerio de Economía y Competitividad" and European Regional Development Funds (ERDF).

References

[1] J. Ortega, J. Asensio-Cubero, J. Q. Gan, and A. Ortiz, *Classification of motor imagery tasks for bci with multiresolution analysis and multiobjective feature selection*, BioMedical Engineering OnLine, 15 (2016), p. 73.

Frequency Scaling and Energy Efficiency regarding the Gauss-Jordan Elimination Scheme on OpenPower 8

Martin Köhler[1] Jens Saak[2]

The Gauss-Jordan Elimination scheme is an alternative to the LU decomposition for solving linear systems or computing the inverse of a matrix. We develop a multi-GPU aware implementation of this algorithm on an OpenPOWER 8 system with application to the Matrix Sign Function. Thereby, we analyze the influence to the CPU clock frequency scaling on the overall energy consumption. The results show possible energy saving of 14.2% without a noteworthy increase of the runtime.

1 Introduction

Beside solving a general linear system $Ax = b$ using the LU decomposition there are a few applications, like Newton's Method for computing the Matrix Sign Function [9, 4, 3, 2] or the Polar Decomposition [7], that require the explicit inverse A^{-1}. In this case, either the three step scheme implemented in LAPACK [1] or the Gauss-Jordan Elimination [11] can be used to obtain A^{-1}. The LAPACK approach first computes the LU decomposition of the matrix A, then inverts the upper triangular matrix U ad finally solves $LA^{-1} = U^{-1}$. This procedure takes $2m^3$ flops if m is the order of the matrix. Furthermore, this approach causes that three routines need to be regarded during the optimization of the implementation. Moreover, the two steps working on triangular matrices are complicated to parallelize by their nature. On the other hand, we have the Gauss-Jordan Elimination computing the inverse A^{-1} by rearranging the three step LAPACK scheme [11]. The resulting algorithm is free of any operations dealing with triangular matrices and mainly consists of general matrix-matrix products. This makes the algorithm preferable on massively parallel architectures, like multi-core or accelerator based systems. Furthermore, one can show that the Gauss-Jordan Elimination reduces the number of memory accesses [10] and by using general matrix-matrix multiplies the data locality for the single operations of the algorithm is improved.

In our contribution, we focus on the efficient implementation of the Gauss-Jordan matrix inversion on the OpenPOWER 8 platform. Besides two 10-core IBM POWER8 CPUs the test system is equipped with two Nvidia Tesla P100 accelerators with NVLink interconnect and 256 GB DDR4 memory. The system can be seen as predecessor of the compute nodes in the upcoming super computer "Summit" at Oak Ridge National Laboratory[3] that will use the IBM POWER9 platform together with the next generation of Nvidia's accelerators named "Volta". The most important differences to previous GPU accelerated systems and the named POWER8 system are:

[1]Computational Methods in Systems and Control Theory, Max Planck Institute for Dynamics of Complex Technical Systems, Sandtorstraße 1, D-39106 Magdeburg , koehlerm@mpi-magdeburg.mpg.de

[2]Computational Methods in Systems and Control Theory, Max Planck Institute for Dynamics of Complex Technical Systems, Sandtorstraße 1, D-39106 Magdeburg , saak@mpi-magdeburg.mpg.de

[3]`https://www.olcf.ornl.gov/summit/` – Accessed March 20th, 2017

- The usage of NVLink as interconnect between CPU and GPU. This increases the transfer rate between their memories by a factor of 2 to 3 in comparison to the latest PCI-Express bus. In this way, data transfers between the CPU and the GPU are cheaper (with respect to runtime) than on older systems.
- The ratio of the peak performances between both CPUs and GPUs is a factor of 20. This is a increase by a factor of 5 if we compare it to an older system, like the 16 core Intel Xeon Haswell with two Nvidia K20 accelerators, where we have done our previous work on [10].
- While keeping the energy consumption for the GPUs in the same order of magnitude as for the old K20 GPUs the energy consumption of the POWER8 CPUs is much higher in idle state, as well as in full operation mode, compared to the Intel Haswell Xeon CPUs with a similar peak performance.

The last point makes the difference from the energy point of view. For this reason we want to focus on reducing the power consumption of the CPUs by changing their clock frequency and/or changing the CPU frequency governors that control the automatic adjustment of the CPU clock frequency.

The contribution is organized as follows. First we recall the Gauss-Jordan Elimination approach and its efficient implementation. Later on we use the Matrix Sign Function as an application for our matrix inversion code. Finally, we show the influence of changing the CPU's clock frequency to the time-to-solution, the energy-to-solution and the Energy-Delay-Product (EDP) [5, 8]. Thereby, the EDP can be used to decide whether it is worth to save the energy or to save runtime from an economic point of view.

2 Gauss-Jordan Elimination

The Gauss-Jordan Elimination scheme can be interpreted as an alternative representation of the LU decomposition with reordered operations [11]. Therefore, we will only recall the basics to obtain a blocked algorithm here. First, we consider the Gauss-Transform $G_i \in \mathbb{R}^{m \times m}$:

$$G_i = \begin{bmatrix} 1 & & & -\frac{a_{1i}}{a_{ii}} & & \\ & \ddots & & \vdots & & \\ & & 1 & -\frac{a_{(i-1)i}}{a_{ii}} & & \\ & & & \frac{1}{a_{ii}} & & \\ & & & -\frac{a_{(i+1)i}}{a_{ii}} & 1 & \\ & & & \vdots & & \ddots \\ & & & -\frac{a_{mi}}{a_{ii}} & & 1 \end{bmatrix}, \tag{1}$$

which introduces zeros in the i-th column of a matrix $G_i A$ and sets the i-th row in this column to one. By applying G_i to a permuted matrix $P_i A$, where P_i exchanges row i with row k, $k := \text{argmax}_{k \geq i} |a_{ki}|$, one can use pivoting as in the LU decomposition [6]. We use $\widetilde{G_i} := G_i P_i$ as pivoted Gauss-Transform (PGT). The application of m Gauss-Transforms or pivoted Gauss-Transforms from the left to a matrix $A \in \mathbb{R}^{m \times m}$ yields its inverse:

$$\underbrace{\widetilde{G_m} \cdots \widetilde{G_1}}_{A^{-1}} A = I. \tag{2}$$

Using the fact that $\widetilde{G_i}$ eliminates the i-th column, except of the 1 on the diagonal, one can store the i-th column of $\widetilde{G_i}$ in the i-th column of A, after it is applied, to obtain an in-place algorithm. The block algorithm (with a block size of N_B) is obtained by the following considerations. Without loss of generality, we neglect the pivoting matrix P_i. The application of a Gauss-Transform G_i can be written as a rank-1 update with an additional operation:

$$A \leftarrow A - \frac{1}{a_{ii}}\left(a_{1i}, \cdots, a_{(i-1)i}, 0, a_{(i+1)i}, \ldots, a_{mi}\right)^T A_{i,\cdot}$$
$$A_{i,\cdot} := \frac{1}{a_{ii}} A_{i,\cdot}. \tag{3}$$

By partitioning the matrix A into

$$A \leftarrow \left[\begin{array}{c|c|c} A_{11} & A_{12} & A_{13} \\ \hline A_{21} & A_{22} & A_{23} \\ \hline A_{31} & A_{32} & A_{33} \end{array}\right], \tag{4}$$

where A_{22} is of dimension $N_B \times N_B$ we obtain the block formulation of the rank-1 update (3) as:

$$A \leftarrow \left[\begin{array}{c|c|c} A_{11} & 0 & A_{13} \\ \hline 0 & 0 & 0 \\ \hline A_{31} & 0 & A_{33} \end{array}\right] + \underbrace{\left[\begin{array}{c} -A_{12}A_{22}^{-1} \\ \hline A_{22}^{-1} \\ \hline -A_{32}A_{22}^{-1} \end{array}\right]}_{H} \begin{bmatrix} A_{21} & I_{N_B} & A_{23} \end{bmatrix}. \tag{5}$$

Thereby, the matrix H_k can be regarded as the (partial) inverse of the block column $B := \begin{bmatrix} A_{12}^T & A_{22}^T & A_{32}^T \end{bmatrix}^T$ and can be computed by applying Gauss-Transforms to B as well.

In order to avoid a direct fallback from the rank-N_B updates from (5) to the rank-1 updates from (3) in the computation of H we use the same strategy as in LAPACK since version 3.6.0. There the locality improved LU decomposition was introduced [12] to avoid the direct level-2 BLAS fallback. The key idea is to apply the blocked algorithm again to H but with a block size of $\frac{N_B}{2}$ recursively until the block size is reduced to 1 and the final work consists only in updating a single column. As in the case of the LU decomposition this increases the data locality of the operations and allows to use more level-3 BLAS operations than if one uses the rank-1 formulation immediately.

Taking the GPUs into account, we can easily create a hybrid CPU-GPU version of our algorithm. The rank-N_B update is well suited for the GPU because, on the one hand, the general matrix-matrix product is one of the best optimize routines for the GPUs and, on the other hand, using the block cyclic distribution scheme this can be easily parallelized across multiple GPUs. Asynchronous operations and lookahead are also easy to implement by splitting the update with H and A_{23} into two parts. The first part affects the leading N_B columns of A_{23} and results in the input data for the computation of the next matrix H. Afterwards the GPUs can handle the remaining part of A_{23} while the CPU prepares the next matrix H.

Newton's Method for the Matrix Sign Function The Matrix Sign Function $X := \text{sign}(A)$, e.g. [9], is the generalization of the sign of a scalar number to the matrix case.

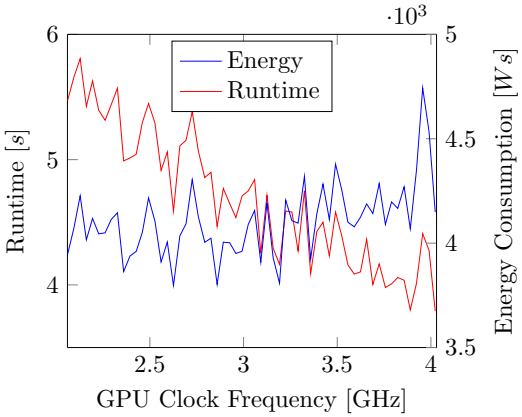

Figure 1: $m = 20\,480$

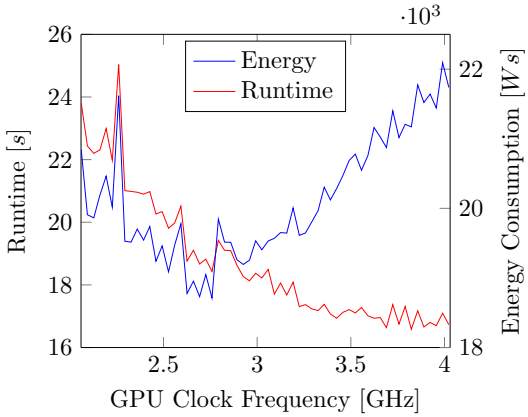

Figure 2: $m = 40\,960$

One way to compute it is to use one of its defining properties, $X^2 = I$, and apply the Newton iteration with the initial value A. This yields the following iteration:

$$X_{k+1} := \frac{1}{2}\left(X_k + X_k^{-1}\right) \quad X_0 = A. \tag{6}$$

On convergence $\operatorname{sign}(A) = X_\infty$ holds. In practical implementations a scaling factor c_k is introduced to accelerate the convergence [3]. For ease of presentation we do not regard this here.

Beside the inversion of the matrix X_k we only need a matrix valued scale and a matrix valued add operation. Having in mind that this operation is bandwidth bound we refer to the high bandwidths of the system here. Reaching a practical GPU–memory bandwidth of 500 GB/s one can still scale a matrix filling up the device memory 31.25 times per second. With a memory bandwidth of 230GB/s bandwidth bound operations can also be performed on the CPUs.

If pivoting is enabled during the inversion of X_k, the Gauss-Jordan-Elimination scheme computes the inverse $\widetilde{X_k}^{-1}$ of PX_k, where P consists of all permutations used during the pivoting. Therefore, we have to add a column permutation to $\widetilde{X_k}^{-1}$ to obtain $X_k^{-1} := \widetilde{X_k}^{-1} P^T$. As long as only one GPU is used this can easily be performed on the GPU. If the matrix is distributed across several GPUs this becomes a communication intensive procedure. Due to the limited bandwidth between two GPUs (40GB/s) the irregular movement of the columns will slow down the whole procedure. In this case we move the whole matrix X_k^{-1} to the host memory again and use a parallel permutation algorithm there. Finally, we distribute the matrix to devices again an create an on device copy of the matrix to have X_{k+1} already available for the next iteration. Changing the devices' data layout to a cyclic block row representation we can easily permute the columns but the problem of the distributed data moves to the row permutation inside the Gauss-Jordan Elimination, which causes the same problems there.

3 Results

We run all experiments on the OpenPOWER 8 system (IBM POWER System 822LC) running CentOS 7.3 (with a custom build Linux 4.8 kernel) mentioned in the Introduction. The software ecosystem consists of Nvidia CUDA 8.0, IBM XLC 13.1.5, IBM XLF 15.1.5,

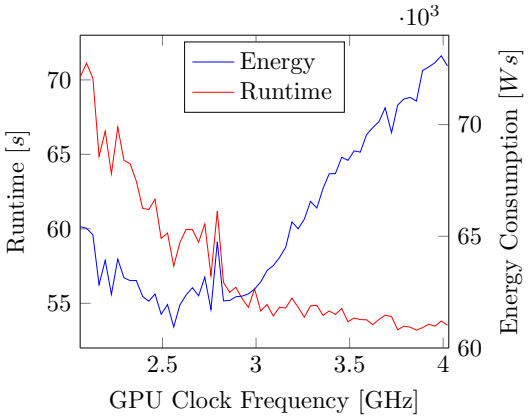

Figure 3: $m = 61\,440$

Dimension	20 480	40 960	61 440
EDP(1)	3.890	3.225	2.959
EDP(2)	3.890	3.225	2.959
EDP(3)	3.890	3.690	3.092

Table 1: CPU Clock Frequency (in $[GHz]$) minimizing the EDP(w) with weights $w = 1, 2, 3$.

and IBM ESSL 5.5 as host BLAS/LAPACK library. The CPUs clock frequency can be adjusted in a range from 2.061GHz to 4.023GHz by steps of $\approx 33 MHz$, where for high clock frequencies above $3.823 GHz$ the frequencies are reduced automatically due to power supply and thermal issues. Nevertheless, we force the CPUs to reach these frequencies by using the *userspace* performance governor of the *cpufreq* mechanism of the Linux kernel.

The main goal of the experiments is to check whether it is worth to spent more energy by enforcing a high CPU clock frequency, or where optimal points between an increase of the runtime and the saved energy are. The optimality is checked with respect to the Energy-Delay-Product (EDP) [5, 8] defined by:

$$\mathrm{EDP}(w) = E \cdot T^w, \tag{7}$$

where E is the energy-to-solution, T is the time-to-solution, and w a weight factor to penalize the time. The optimal block size N_B for the Gauss-Jordan Elimination was determined in previous experiments. Here, we restrict to the matrix inversion since this is the most challenging operation during the computation of the Matrix Sign Function. We use random matrices A of dimension 20 480, 40 960, 61 440.

Figures 1 to 3 show the runtime and the energy consumption of the Gauss-Jordan Elimination. For the smallest case ($m = 20\,480$) we observe that we have an approximately linear decrease of the runtime coupled with a slowly increasing energy consumption. In this case, one can still choose nearly maximum CPU clock frequency and still obtain an economically optimal execution. This coincides with the suggestion of the EDP from Table 1 to chose a frequency next to the maximum. For larger problems we see that beginning with a clock frequency of $\approx 2.9\,GHz$ we have a steep increase of the energy consumption while only obtaining a small speed up in the runtime. On the other hand, regarding the largest example ($m = 61\,440$) the increase of the clock frequency from $2.959\,GHz$ to $4.023\,GHz$ costs 14.2% more energy while only accelerating the process by 2.2%, while switching from the clock frequency from $2.016\,GHz$ to $2.959\,GHz$ we only need 5% more energy and accelerate the algorithm by 26.9%. The EDP from Table 1 suggest exactly this clock frequency for $w = 1$ and $w = 2$. Even if we increase the impact of the runtime to $w = 3$ the EDP only suggests an increase of the clock frequency by 4 steps to $3.092\,GHz$. Finally, we see that for an increasing problem dimension the influence of the pure CPU power decreases and even for higher weights of the runtime the EDP suggests a moderate clock frequency in order to obtain an economically acceptable solution.

4 Conclusions

We have shown that for the Gauss-Jordan Elimination on the OpenPOWER 8 platform one can save a remarkable amount of energy by choosing a proper clock frequency for the CPUs without causing a noteworthy increase of the runtime. This extended abstract only covers the case of fixed CPU clock frequencies but the hardware (supported by the drivers from the Linux kernel) supports automatic adjustment with respect to several policies. These so called *cpufreq* governors will also be taken into account in the final contribution as well as the overall process of the Newton iteration (6).

References

[1] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK Users' Guide*, SIAM, Philadelphia, PA, third ed., 1999.

[2] P. Benner, P. Ezzatti, E. S. Quintana-Ortí, and A. Remón, *Matrix inversion on CPU-GPU platforms with applications in control theory*, Concurrency and Computing: Practice and Experience, 25 (2013), pp. 1170–1182.

[3] R. Byers, C. He, and V. Mehrmann, *The matrix sign function method and the computation of invariant subspaces*, SIAM J. Matrix Anal. Appl., 18 (1997), pp. 615–632.

[4] E. D. Denman and A. N. Beavers, *The matrix sign function and computations in systems*, Appl. Math. Comput., 2 (1976), pp. 63–94.

[5] V. W. Freeh, D. K. Lowenthal, F. Pan, N. Kappiah, R. Springer, B. L. Rountree, and M. E. Femal, *Analyzing the energy-time trade-off in high-performance computing applications*, IEEE Trans. Parallel Distrib. Syst., 18 (2007), pp. 835–848.

[6] G. H. Golub and C. F. Van Loan, *Matrix Computations*, Johns Hopkins University Press, Baltimore, fourth ed., 2013.

[7] N. J. Higham, *Computing the polar decomposition—with applications*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 1160–1174.

[8] M. Horowitz, T. Indermaur, and R. Gonzalez, *Low-power digital design*, Proceedings of 1994 IEEE Symposium on Low Power Electronics, (1994), pp. 8–11.

[9] C. Kenney and A. J. Laub, *The matrix sign function*, IEEE Trans. Automat. Control, 40 (1995), pp. 1330–1348.

[10] M. Köhler, C. Penke, J. Saak, and P. Ezzatti, *Energy-aware solution of linear systems with many right hand sides*, Comput. Sci. Res. Dev., (2016). accepted for publication.

[11] E. S. Quintana-Ortí, G. Quintana-Ortí, X. Sun, and R. van de Geijn, *A note on parallel matrix inversion*, SIAM J. Sci. Comput., 22 (2001), pp. 1762–1771.

[12] S. Toledo, *Locality of reference in LU decomposition with partial pivoting*, SIAM Journal on Matrix Analysis and Applications, 18 (1997), pp. 1065–1081.

Characterization of Multicore Architectures using Task-Parallel ILU-type Preconditioned CG Solvers

José I. Aliaga[1] María Barreda[1] Enrique S. Quintana-Ortí[1]

We investigate the efficiency of state-of-the-art multicore processors using a multi-threaded task-parallel implementation of the Conjugate Gradient (CG) method, accelerated with an incomplete LU (ILU) preconditioner. Concretely, we analyze multicore architectures with distinct designs and market targets to compare their parallel performance and energy efficiency.

1 Introduction

The solution of sparse linear systems via iterative methods has been recently argued to be representative of the actual performance that is experienced by a large fraction of the scientific and engineering codes running on current supercomputers. This has led to the introduction of the HPCG benchmark[2] as a complement to the traditional LINPACK benchmark that ranks supercomputers twice per year in the Top500/Green500 lists.

Following this idea, in this work we investigate the efficiency of state-of-the-art multicore processors using our own task-parallel implementation of the CG method enhanced with an ILU-type preconditioner (hereafter, referred to as ILU-PCG). Our experimental analysis evaluates both the parallel performance and energy consumption of a variety of multicore architectures designed to deliver high performance and/or reduced energy consumption.

The rest of the abstract is organized as follows. In Section 2 we describe the ILU-PCG solver and how to extract the task-parallelism. In Section 3 we present the architectures included in the study. In Sectionand 4 we characterize the multicore architectures using the ILU-PCG solver. Finally, in Section 5 we offer some concluding remarks.

2 Task-parallel ILU-PCG solver

2.1 The iterative solver

In the evaluation we employ an ILU-PCG solver to tackle linear systems with sparse and symmetric positive definite (s.p.d.) matrix A. The solver computes a preconditioner M, via an ILU factorization of the coefficient matrix, and then solves the preconditioned linear system via a convenient variant of the CG method, which hopefully exhibits a fast convergence rate due to the effect of the preconditioner.

The most complex and challenging operations in the ILU-PCG benckhmark are the computation of the preconditioner M and its application. The remaining computations involve basic linear algebra operations such as the sparse matrix-vector product (SPMV),

[1]Dpto. de Ingeniería y Ciencia de Computadores, Universidad Jaume I, 12.071–Castellón, Spain, aliaga@uji.es, mvaya@uji.es, quintana@uji.es

[2]`http://www.hpcg-benchmark.org/`

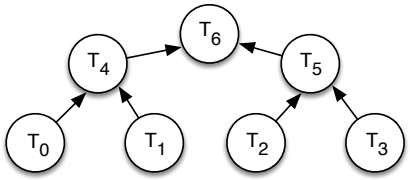

Figure 1: Dependency tree of the diagonal blocks. Task T_j is in charge of processing the diagonal block A_{jj}.

DOT (or inner) product, AXPY-like update, and the calculation of a vector 2-NORM (equivalent to a dot product). We will therefore focus the following analysis in the operations involving the preconditioner.

2.2 Task-Parallel PCG

The concurrency intrinsic to the calculation as well as the application of the preconditioner is considerably involved. Specifically, the parallelism of the process can be unveiled by means of nested dissection applied to the adjacency graph representing the non-zero connectivity of matrix A. By recursively splitting this graph, the result is a hierarchy of independent subgraphs, organized as dependency acyclic graph (DAG) with the shape of a binary tree. In the remaining operations of PCG, concurrency is extracted by dividing the operations into subtasks, and mapping the data into the leaves of the DAG. For example, it is straight-forward to partition the output vector from SPMV (or the AXPYs) into several subvectors, which can be then computed as independent tasks. The DOT and vector 2-NORM are reduction-type operations, also parallelizable, but impose a global synchronization/communication point to the procedure.

Figure 1 illustrates the dependency tree for the factorization of the diagonal blocks. The edges of the DAG define the dependencies between the diagonal blocks (tasks); that is, the order in which these blocks of the matrix have to be processed. Luckily, the leaf tasks of the DAG in general comprise a significant part of the computational cost of the process.

In summary, by recursively applying the same idea, we can explicitly unveil an increasing amount of task-level parallelism during the factorization that computes the preconditioner M as well as the triangular solves involved in its application. The DAGs associated with the first stage (computation of the preconditioner) and the triangular solves with the lower triangular factor present the form of a tree with bottom-up dependencies, from the leaves to the root; on the other hand, the triangular solves with the upper triangular factor share the structure of the DAG but the dependencies are reversed, pointing down from the root to the leaves.

We note that the recursive decomposition of the graph into further levels multiplies the concurrency exponentially. However, there exists a balance between the number of levels, and consequently independent tasks, and the convergence rate of the procedure. Concretely, this recursive process introduces additional numerical levels in the computation of the preconditioner. Thus, different DAGs are associated with distinct preconditioners, which can be expected to feature close numerical properties.

In our particular parallel implementation of the PCG solver, task-parallelism is ex-

Architecture	SANDY	ODROID(A15)	JUNO(A57)	HASWELL	XEON PHI
PROCESOR NUMBER	E5-2620	ARMv7 rev 3 (v7l)	AArch64 rev 0	E5-2603v3	5110P
#SOCKETS	2	1	1	2	1
#CORES	12	4	2	12	60
BASE FREQUENCY	2.0 GHz	2.0 GHz	1.1 GHZ	1.6 GHz	1.053
CACHE	15 MB	2 MB	2 MB	15 MB	30 MB
TDP	95 W	15 W	30 W	85 W	225 W
VOLTAGE RANGE	0.60 V-1.35 V	0.91 V-1.32 V	0.81 V-1.00 V	0.65 V-1.30 V	–
MEMORY	32 GB	2 GB	8 GB	32 GB	8 GB
MAX. MEMORY BANDWIDTH	42.6 GB/s	14.9 GB/s	13.2 GB/s	51 GB/s	320 GB/s

Table 1: Hardware specifications of the platforms.

Architecture	SANDY	ODROID (A15)	JUNO (A57)	HASWELL	XEON PHI
GCC	4.4.6	4.8.2	4.9.1	4.4.7	5.1.0
OMPSS	16.06	16.06.1	16.06.1	16.06.1	16.06
MERCURIUM	2.0.0	2.0.0	2.0.0	2.0.0	2.0.0
NANOX	0.12a	0.10.1	0.10.1	0.10.1	0.12a
METIS	5.0.2	5.0.2	5.0.2	5.0.2	5.0.2
POWER/ENERGY MEASUREMENTS	RAPL	PMLIB	PMLIB	RAPL	PMLIB
FREQUENCY CHANGES	CPUfreq	CPUfreq	CPUfreq	CPUfreq	–

Table 2: Software specifications of the platforms.

ploited using a multi-threaded code that relies on the OmpSs programming model[3] [1].

3 Target Multicore Architectures and General Setup

For the study, we selected three different types of multicore architectures comprising two general-purpose processors from Intel, two low-power systems from ARM, and the Intel Xeon Phi. This collection is representative of todays' multicore technology. Table 1 offers some information about hardware in each platform; and the software employed in the platforms are described in Table 2.

All the experiments in next sections employed IEEE754 real double-precision arithmetic. For the analysis, we employed a large-scale linear system corresponding to the Laplacian equation $-\Delta u = f$ in a $3D$ unit cube $\Omega = [0,1]^3$ with Dirichlet boundary conditions, $u = g$ on $\partial\Omega$, and a discretization that resulted in a SPD system, with instances of different size. The problem size is the largest that fits in the main memory of each platform.

4 Characterizing Multicore Architectures using ILU-PCG

The following experimental evaluation comprises three "dimensions", two of them architectural (number of cores/threads and operation frequency) and one corresponding to software (number of leaves for ILU-type preconditioner task dependency tree). Due to the large number of tests and results, we organize the presentation of the performance analysis of the architecture into the following sequence of steps:

[3] `https://pm.bsc.es/ompss`

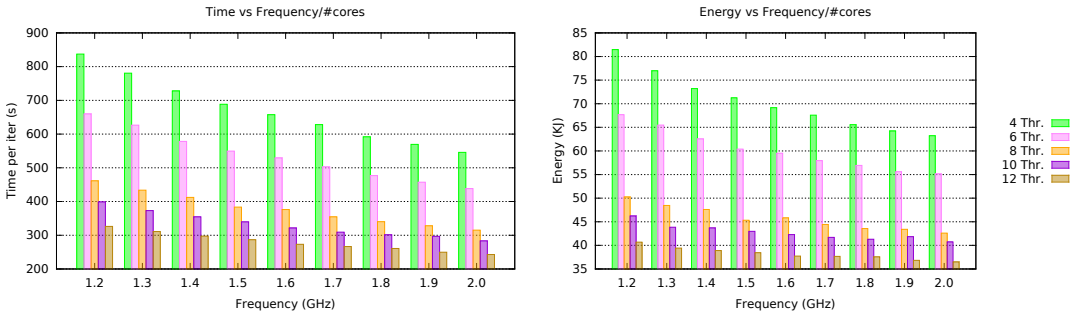

Figure 2: Time and energy consumption for the execution of ILU-PCG in SANDY.

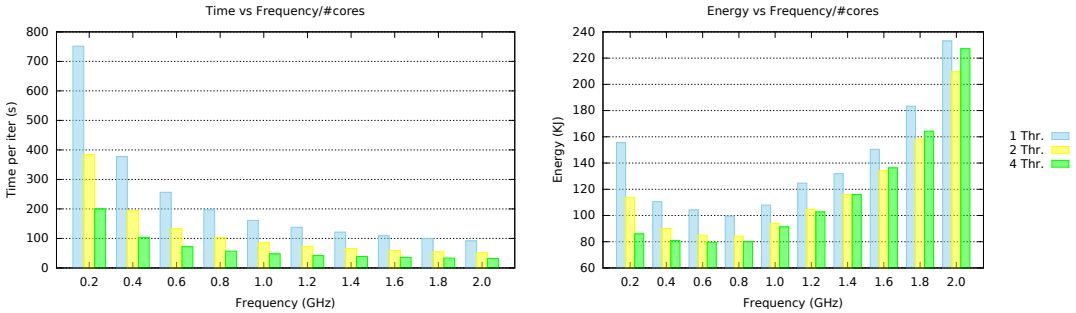

Figure 3: Time and energy consumption for the execution of ILU-PCG in ODROID.

1. Evaluation of the parallel ILU-PCG solver for a range of representative frequencies and number of threads (thread-level parallelism), using 1, 2, 4,..., 64 leaves.

2. Selection of the optimal number of leaves for each level of thread-parallelism.

3. Evaluation of the impact of frequency on the ILU-PCG solver.

4. Selection of the optimal frequency for each number of threads.

5. Evaluation of the impact of the thread-level parallelism on the ILU-PCG solver.

This analysis is then repeated from the perspective of energy efficiency, taking as a basis the performance evaluation to justify some of the results for this second metric.

In order to avoid an exhaustive list of figures and results, we next summarize them using a few plots that illustrate the interplay between frequency/thread concurrency and performance/energy consumption. In particular, Figures 2–6 report absolute values for the last two metrics against processor frequency and number of threads. The third one (number of leaves), depends on the software, and is set for all these experiments to the optimal value. To allow an easier visualization of the differences, for those architectures with a large number of cores, we skip the results obtained with 1 and 2 cores. In any case, these configurations always offered worse performance and energy efficiency than those using a higher level of thread concurrency.

A collection of general remarks can be extracted from this experimental evaluation that emphasize the differences between the performance-oriented architectures (Intel Xeon) and the low-power processors (ARM):

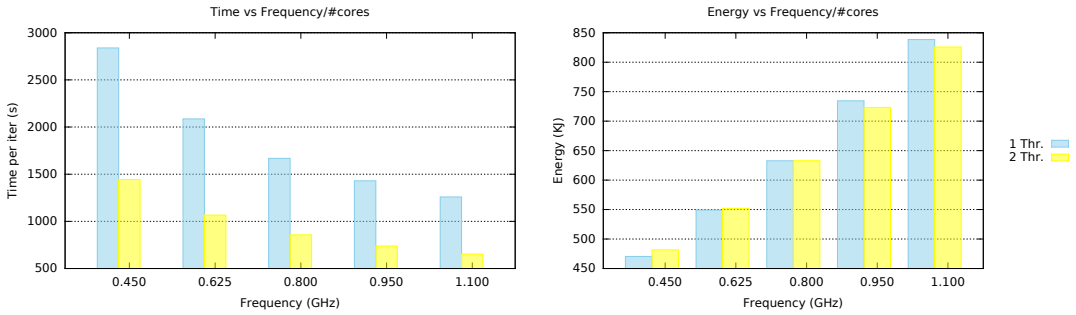

Figure 4: Time and energy consumption for the execution of ILU-PCG in JUNO.

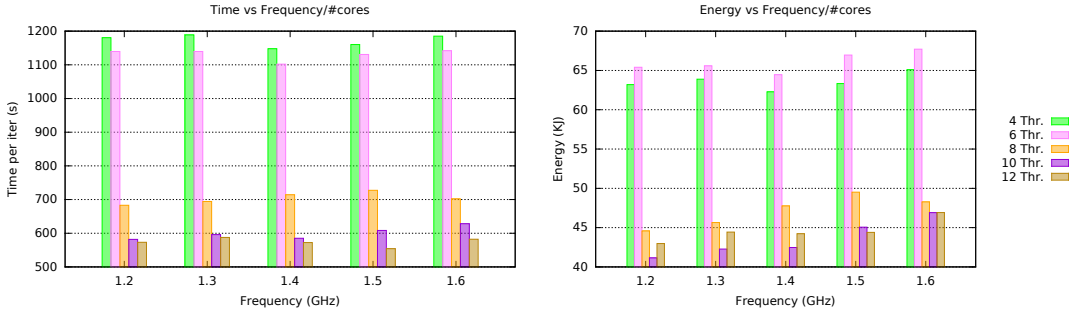

Figure 5: Time and energy consumption for the execution of ILU-PCG in HASWELL.

Performance:

1. The optimal number of leaves is mostly determined by the problem size: a larger dimension can accommodate additional levels of task-parallelism without incurring into a costly overhead. In contrast, the number of leaves is basically independent of the architeture class (performance-oriented versus low-power), frequency, and number of threads. For the small and large problem instances, the optimal number of leaves are, respectively, 8 and 32.
2. The execution time in general benefits from operating at a higher frequency and/or using a larger number of cores. However, the differences may be small when the memory bandwidth is saturated as the results for the low-power architecture demonstrate.

Energy consumption:

1. The optimal numbers of leaves match those obtained when the figure-of-merit is performance. The same remarks apply when the target metric is energy consumption.
2. The optimal frequency is the highest one for the Intel performance-oriented architectures. In contrast, the ARM low-power processors benefit from a more reduced frequency level. The reason for this different behaviour is twofold, and can be used to further distinguish the behaviour of the two types of systems:
   a) The performance-oriented architectures exhibit a considerable static power rate so that increasing the execution time is very costly in terms of energy consumption. The low-power processors do not suffer from this drawback.

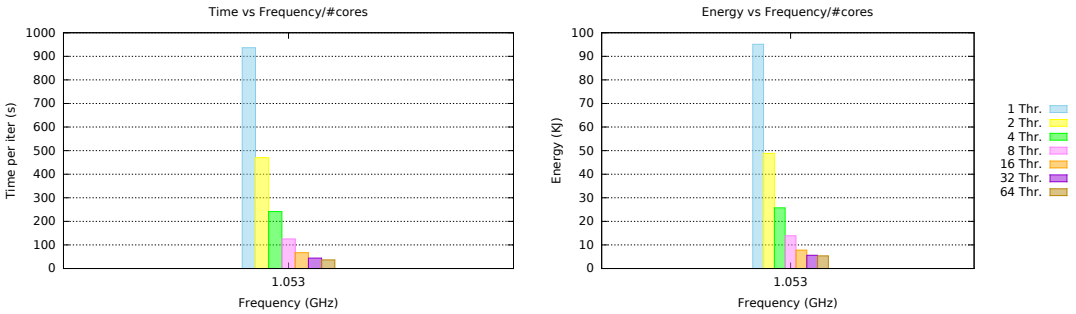

Figure 6: Time and energy consumption for the execution of ILU-PCG in XEON PHI.

b) The low-power processors tend to saturate the memory bandwidth rapidly as the frequency is raised, yielding a negligible improvement of execution time for a linear increase in the power dissipation rate. The consequence is a worse energy efficiency.

3. From the perspective of scalability, adding more cores is beneficial unless the memory bandwidth is saturated. Once that threshold is surpassed, the increase in the dissipation rate directly translates into higher energy costs.

5 Conclusions

We have analyzed the computational performance and energy efficiency of servers equipped with the state-of-the-art general-purpose multicore processors as well as accelerators like the Intel Xeon Phi. Following the introduction of the HPCG benchmark, we adopted ILU-PCG to test performance and energy efficiency of multicore platforms, observing different behaviours for performance-oriented and low-power processors, especially when the figure of merit is energy efficiency.

References

[1] J. I. Aliaga, R. M. Badia, M. Barreda, M. Bollhöfer, and E. S. Quintana-Ortí. Leveraging task-parallelism with OmpSs in ILUPACK's preconditioned CG method. In *26th Int. Symp. on Computer Architecture and High Performance Computing (SBAC-PAD 2014)*, pages 262–269, 2014.

Harvesting Energy in ILUPACK via Slack Elimination

José I. Aliaga[1] María Barreda[1] Asunción Castaño[1]

We develop a new energy-aware methodology to improve the energy consumption of a task-parallel preconditioned Conjugate Gradient iterative solver on a Haswell-EP Intel Xeon. This technique leverages the power-saving modes of the processor and the frequency range of the *userspace* Linux governor, modifying the CPU frequency for some operations. We demonstrate that its application during the main operations of the PCG solver can reduce its energy consumption.

1 Introduction

ILUPACK[2] (Incomplete LU decomposition PACKage) offers an assorted variety of Krylov subspace-based methods, enhanced with a sophisticated ILU-type preconditioner, for the iterative solution of sparse linear systems. The computational cost of computing and applying ILUPACK's preconditioner has sparked several recent efforts to develop parallel versions of this solver, for multicore processors, graphics accelerators, and clusters of computer nodes; see [2, 3] and the references therein.

Task-parallel versions of ILUPACK have also been used as a case study to explore the energy consumption and optimization of iterative solvers. Concretely, the authors of [4] investigated the benefits that an energy-aware implementation of the runtime in charge of the concurrent execution of ILUPACK produces on the time-energy balance of the application. The study in that paper reported energy savings between 7 and 13% (for Intel and AMD multicore processors from 2009-2010), with practically no penalty on performance.

In this paper we explore a new approach to save energy in the task-parallel version of ILUPACK's preconditioned Conjugate Gradient (PCG) method, leveraging the iterative nature of the method to progressively adjust the frequency of the processor cores in order to reduce idle periods and harvest energy. In rough detail, our algorithmic-based energy-saving (ABES) technique is applied to the major operations comprised by ILUPACK, namely the sparse matrix-vector product (SpMV) and the lower/upper triangular solves required for the application of the preconditioner (respectively denoted as LwTrSv and UpTrSv). Each one of these operations is divided into a collection of sub-operations, or *tasks*, to be executed in parallel. Then, by enforcing a deterministic mapping of these tasks to cores, we can detect and quantify idle periods during the first initial iterations, tuning the operating frequency of the cores to reduce these idle times.

2 Brief Overview of ILUPACK

ILUPACK provides C and Fortran routines for the numerical solution of sparse linear systems via Krylov subspace methods [8], combined with multilevel preconditioners that

[1]Dpto. de Ingeniería y Ciencia de Computadores, Universidad Jaume I, 12.071–Castellón, Spain, aliaga@uji.es, mvaya@uji.es, castano@uji.es

[2]`http://ilupack.tu-bs.de`

improve the numerical properties of the linear system, accelerating the convergence of the iterative solver. ILUPACK derives an efficient preconditioner from the ILU factorization of the system matrix, dropping the small entries of the factors, while relying on pivoting to bound the norm of the inverse triangular factors, to compute a numerical multilevel hierarchy of partial inverse-based approximations [5, 6].

Exposing task-parallelism In the remaining of this section, we focus on the parallelization strategy underlying the task-parallel versions of ILU-type iterative solvers in general, and ILUPACK PCG in particular; see, e.g., [2]. Basically, these methods exploit the connection between sparse matrices and adjacency graphs, recursively applying nested dissection to permute the sparse matrix. The goal of this re-organization of the matrix is to obtain a hierarchy of subgraphs and separators that fix the order in which the diagonal blocks have to be factorized. This process renders a task dependency graph (TDG) for the preconditioner calculation with the shape of a balanced binary tree, where the subgraphs occupy the leaves and the separators correspond to the internal nodes.

From the perspective of computational cost and complexity, the major operations in ILUPACK's PCG solve are SpMV, LwTrSv and UpTrSv, each ocurring once per iteration. With a proper organization of the data and distribution of the work, for a TDG with l leaf nodes, the SpMV kernel can be decomposed into an equal number of *independent* tasks. The parallelization of the triangular solves is more complex. These kernels can both be decomposed into the same number of tasks as the TDG identified during the preconditioner calculation, maintaining the same task dependency. Thus, there exist dependencies in the binary-tree, pointing bottom-up for the lower triangular kernel and top-bottom for the upper triangular case. As a result, when these tasks are mapped to the cores, the information flows as in a reduction for LwTrSv or as in a broadcast for UpTrSv.

Mapping tasks to cores In practice, each operation appearing in ILUPACK's PCG is decomposed into a number of tasks that exceeds the number of cores as this produces a more balanced distribution of the workload during the execution. For LwTrSv and UpTrSv, most of the computational work is concentrated into the leaf nodes of the TDG. However, as the number of levels in the TDG is increased, the processing of the separators (non-leaf nodes) introduces some overhead, ultimately constraining the practical number of leaf nodes to a few hundreds. The take-away from this discussion is that, when deciding the number of levels/leaf nodes of the TDG, there is a trade-off between workload balancing and cost of processing the separators of the TDG.

Figure 1 displays an `Extrae` trace for one iteration of the PCG solver, executed on an Intel Xeon 16-core processor using 16 threads. The TDG in this example is composed of 64 leaves (4 leaf tasks per thread) and 7 levels. This trace shows that, even with 4× more leaf nodes than threads, there still appear significant idle times for SpMV, LwTrSv and UpTrSv, motivating the approach to save energy described in the next section.

3 Applying ABES in ILUPACK

In order to describe the principle underlying the ABES technique, let us consider initially a simple TDG consisting of three tasks, T_0, T_1, T_2, organized into two levels, with data dependencies $\mathsf{T}_0 \rightarrow \mathsf{T}_1$, $\mathsf{T}_0 \rightarrow \mathsf{T}_2$; and T_1, T_2 independent of each other. (This reflects the scenario occuring during the lower triangular solve, LwTrSv).

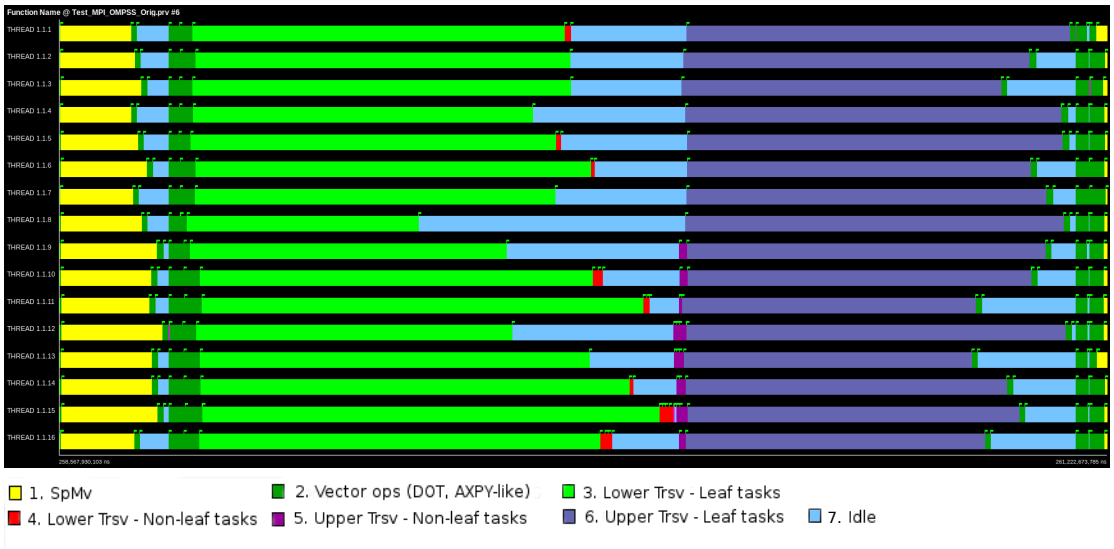

Figure 1: Execution traces of the PCG iterative solve preconditioned with ILUPACK for 16 threads.

In addition, assume a task-parallel execution using two threads, on a platform consisting of two hardware cores, denoted as C_1, C_2; and let us map the execution of T_1 to C_1 and that of T_2 to C_2. Thus, in case the execution time of T_1 does not exactly match that of T_2, due to the data dependencies, the thread in charge of the less expensive task will have to wait for its counterpart to complete its task. It is precisely this "slack" (or idle) period that we aim to eliminate with our ABES technique.

Consider the execution of ILUPACK next. The execution of the main tasks appearing in the iterative solve of ILUPACK's PCG method yield idle periods, due to an unbalanced distribution of the workload (see Figure 1), that our ABES technique targets as follows:

- For the preconditioner computation, we allow a dynamic mapping of tasks to threads, and the same mapping is enforced for the PCG solve. Furthermore, threads are assigned to specific cores (no thread migration is allowed) and all threads/cores initially proceed at the nominal frequency f_n. Because of the strict mapping of threads to cores, we will use "thread" to refer to both terms hereafter.

- During the first five iterations, the ABES mechanism records the termination time for each thread–operation pair, identifying the *slowest thread*. The ABES mechanism then determines the operating frequency of each thread–operation for subsequent iterations. Concretely, the frequency-tuning policy aims to slow-down the last task of all threads that terminate the execution of their tasks earlier than the slowest thread.

- Each four iterations, the ABES mechanism analyze the impact of the frequency-tuning policy. If the defined operating frequencies in a thread yields in longer execution time than the execution time of slowest thread, the last changed task is fixed to the next higher level and the corresponding thread is removed to the ABES policy. Otherwise, the policy aims to slow-down the corresponding task, or the previous task, if its minimum operating frequency has been reached.

- When all the threads are removed to the policy, the operating frequency of each thread–operation has been fixed.

4 Experimental Results

For the experiments in this section, we employ a server equipped with two 8-core Intel Xeon(R) E5-2630 processors (2.4 GHz), with 64 GBytes of DDR3 RAM. The *userspace* Linux governor allows the processor cores to operate at 13 possible frequencies ranging from 1.2 GHz to 2.4 GHz, with a stride of 0.1 GHz. The operating system running in the server is Linux version 2.6.32-642.4.2.el6.centos.plus.x86_64, and the compiler is gcc 4.4.7.

All the experiments employed IEEE754 real double-precision arithmetic. In the first experiment we generated a large-scale linear system for the Laplacian equation $-\Delta u = f$ in a $3D$ unit cube $\Omega = [0,1]^3$ with Dirichlet boundary conditions, $u = g$ on $\partial\Omega$, and a discretization that resulted in a sparse symmetric positive system. This *A200* matrix has $8 \cdot 10^6$ rows/columns. The second matrix in the experimentation corresponds to the sparse symmetric `audikw_1` example from the SuiteSparse Matrix Collection [1], with close to 1,000,000 rows/columns.

Energy was measured using Intel's RAPL (Running Average Power Limit) interface [7], reflecting the estimated consumption of the core-uncore (*package*), DRAM and the total (core, uncore and DRAM) system. For the Haswell-EP, the isolated on-core consumption is not provided by RAPL. The idle power was obtained during the executing the Linux sleep command by all cores during 100 sec. This value was then subtracted to the total power in order to obtain the net energy. The experiments were executed after a warm up period of 150 sec. using a busy-wait loop, and each experiment was repeated 5 times, showing the average values.

In our energy consumption analysis, we consider the following configurations: performance-oriented (PO), energy–aware (EnAw), and three variants of ABES. For PO, the PCG is executed using a power-oblivious runtime; in contrast, for EnAw, the runtime exploits the power-saving modes of the Intel Xeon processors, promoting the idle threads to one of the power-saving C-states [4]. The ABES variants combine EnAw with the ABES technique applied to SpMV(ABES1), SpMV & LwTrSv(ABES2), or SpMV & LwTrSv & UpTrSv(ABES3).

Tables 1 and 2 report the time-power-energy of the five policies applied to the iterative solution of the two sparse examples, using a 32-leaf TDG executed on 8 cores. Two different mappings are considered: balanced and unbalanced. In the first mapping, the operations are issued in decreasing order of computational cost; the second mapping is manually generated to enforce additional ABES steps. The last column in the tables includes the ABES steps which are added to the policy sited in previous row, thus ABES1, ABES2 and ABES3 respectively represent the ABES steps in SpMV, LwTrSv and UpTrSv. Moreover, the remaining columns numbers show the relative improvement of the corresponding variant with respect to the PO policy, therefore, negative values reflect a decrease of performance, power dissipation or energy consumption. Several conclusions can be obtained from the analysis of these tables:

- The number of ABES steps is greater in unbalanced configuration than in their balanced counterparts.

- A higher number of ABES steps is necessary for `audikw_1` than for A200, mainly because the nonzero pattern of the last example is more regular.

- A significant part of the increase in the execution time is due to the introduction of EnAw while the impact of ABES variants are smaller.

Balanced mapping											
	Total energy			Net energy			Time	Total power			Add.
	Package	DRAM	Total	Package	DRAM	Total		Package	DRAM	Total	steps
EnAw	0.62	-0.57	0.49	1.09	-0.55	0.90	-0.65	1.28	0.07	1.14	–
ABES1	0.97	-0.72	0.77	1.56	-0.76	1.29	-0.63	1.61	-0.09	1.41	21
ABES2	1.09	-0.65	0.89	1.73	-0.65	1.46	-0.65	1.75	0.00	1.55	0
ABES3	0.99	-0.74	0.79	1.62	-0.75	1.35	-0.71	1.72	-0.03	1.52	5

Unbalanced mapping											
	Total energy			Net energy			Time	Total power			Add.
	Package	DRAM	Total	Package	DRAM	Total		Package	DRAM	Total	steps
EnAw	0.62	-0.58	0.48	1.14	-0.50	0.95	-0.79	1.42	0.22	1.28	–
ABES1	1.75	-0.82	1.45	2.71	-0.83	2.29	-0.81	2.58	-0.01	2.28	41
ABES2	1.72	-0.86	1.42	2.69	-0.85	2.28	-0.87	2.61	0.01	2.31	10
ABES3	1.57	-0.68	1.31	2.46	-0.63	2.10	-0.84	2.43	0.16	2.16	16

Table 1: Relative variation (in %) of the energy-aware variants with respect to PO, considering the balanced and unbalanced mappings of the A200 matrix when a 32-leaf TDG processed by 8 cores.

Balanced mapping											
	Total energy			Net energy			Time	Total power			add.
	Package	DRAM	Total	Package	DRAM	Total		Package	DRAM	Total	steps
EnAw	1.02	-0.46	0.86	1.59	-0.40	1.39	-0.61	1.63	0.14	1.48	–
ABES1	3.26	-0.96	2.81	4.73	-1.03	4.15	-0.79	4.08	-0.17	3.63	41
ABES2	3.41	-1.00	2.95	4.97	-1.07	4.35	-0.84	4.29	-0.16	3.82	22
ABES3	3.24	-0.96	2.80	4.73	-1.01	4.14	-0.84	4.11	-0.12	3.67	17

Unbalanced mapping											
	Total energy			Net energy			Time	Total power			Add.
	Package	DRAM	Total	Package	DRAM	Total		Package	DRAM	Total	steps
EnAw	2.30	-0.91	1.97	3.46	-0.99	3.01	-0.75	3.08	-0.16	2.74	–
ABES1	5.92	-2.25	5.03	8.67	-2.79	7.45	-0.97	6.95	-1.29	6.06	65
ABES2	6.10	-2.26	5.19	8.96	-2.78	7.70	-1.03	7.21	-1.24	6.29	60
ABES3	5.88	-2.36	4.98	8.68	-2.90	7.44	-1.10	7.06	-1.27	6.16	59

Table 2: Relative variation (in %) of the energy-aware variants with respect to PO, considering the balanced and unbalanced mappings of the `audikw_1` matrix when a 32-leaf TDG processed by 8 cores.

- In general, the improvements in net energy are higher than those observed in the total energy.
- The savings of the ABES variants occur in package energy consumption; in contrast, the DRAM energy is increased. These figures grow with the number of ABES steps.
- The application of ABES to SpMV produces larger savings than the use of the same technique in LwTrSv or UpTrSv.

5 Conclusions

We have introduced the ABES technique to improve the performance of energy-aware variants of a PCG solver. The results demonstrate that the application of this methodology on the main operations of the solver reduces the energy consumption with a negligible impact on the execution time. Furthermore, the technique adapts to the problem, increasing the energy savings for unbalanced mappings.

Acknowledgements

This work was supported by the CICYT project TIN2014-53495-R of the MINECO and FEDER.

References

[1] *The suitesparse matrix collection.* `http://www.cise.ufl.edu/research/sparse/matrices`, 2017.

[2] J. I. Aliaga, R. M. Badia, M. Barreda, M. Bollhöfer, E. Dufrechou, P. Ezzatti, and E. S. Quintana-Ortí, *Exploiting task- and data-parallelism in ILUPACK's preconditioned CG solver on NUMA architectures and many-core accelerators*, Parallel Computing, 54 (2016), pp. 97–107.

[3] J. I. Aliaga, M. Barreda, M. Bollhöfer, and E. S. Quintana-Ortí, *Exploiting task-parallelism in message-passing sparse linear system solvers using OmpSs*, in Euro-Par 2016: Parallel Processing: 22nd Int. Conf. Parallel and Distributed Computing, Springer, 2016, pp. 631–643.

[4] J. I. Aliaga, M. Barreda, M. F. Dolz, A. F. Martín, R. Mayo, and E. S. Quintana-Ortí, *Assessing the impact of the CPU power-saving modes on the task-parallel solution of sparse linear systems*, Cluster Computing, 17 (2014), pp. 1335–1348.

[5] M. Bollhöfer, M. J. Grote, and O. Schenk, *Algebraic multilevel preconditioner for the Helmholtz equation in heterogeneous media*, SIAM J. Sci. Comput., 31 (2009), pp. 3781–3805.

[6] M. Bollhöfer and Y. Saad, *Multilevel preconditioners constructed from inverse–based ILUs*, SIAM J. Sci. Comput., 27 (2006), pp. 1627–1650. special issue on the 8–th Copper Mountain Conference on Iterative Methods.

[7] Intel Corp., *Intel 64 and IA-32 architectures software developer manual. Volume 3B: System programming guide, Part 2*, 2015.

[8] Y. Saad, *Iterative Methods for Sparse Linear Systems*, SIAM, 2003.

GPU-Accelerated Implementation of the Storage-Efficient QR Decomposition

Peter Benner[1] Martin Köhler[2] Carolin Penke[3]

The LAPACK routines `GEQRT2` and `GEQRT3` can be used to compute the QR decomposition of a matrix of size $m \times n$ as well as the storage-efficient representation of the orthogonal factor $Q = I - VTV^T$. A GPU-accelerated algorithm is presented that expands a blocked CPU-GPU hybrid QR decomposition to compute the triangular matrix T. The storage-efficient representation is used in particular to access blocks of the matrix Q without having to generate all of it. The algorithm runs on one GPU and aims to use memory efficiently in order to process matrices as large as possible. Via the reuse of intermediate results the amount of necessary operations can be reduced significantly. As a result the algorithm outperforms the standard LAPACK routine by a factor of 3 for square matrices, which goes hand in hand with a reduced energy consumption.

Along with the LU decomposition, the QR decomposition [5] is one of the basic matrix factorizations used in many numerical linear algebra algorithms. Especially when orthonormal bases come into play, e.g. in least-squares problems, the QR decomposition is commonly involved. During the last decades many different strategies to compute the orthogonal matrix $Q \in \mathbb{R}^{m\times m}$ and the upper triangular matrix $R \in \mathbb{R}^{m\times n}$ from a general matrix $A \in \mathbb{R}^{m\times n}$ fulfilling $QR = A$ were developed. The most common ones are based on Householder reflections [6]. These algorithms represent the matrix Q as a product of orthonormal Householder matrices $H_i \in \mathbb{R}^{m\times m}$:

$$Q = H_1 \cdots H_n, \tag{1}$$

where $H_i = I_m - 2\frac{v_i v_i^T}{v_i^T v_i}$ and v_i is the i-th Householder vector. Typically Q is not stored explicitly but implicitly in factored-form representation, where the scaled Householder vectors v_i and the scalar factors $\tau_i = \frac{2}{v_i^T v_i}$ are stored explicitly [5, 1].

We consider the case where we are interested in a block partitioning of Q like

$$Q = \begin{bmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \end{bmatrix}, \tag{2}$$

where $Q_{ij} \in \mathbb{R}^{\frac{m}{2}\times\frac{m}{2}}$. When Q is stored as the product of Householder matrices (1) in factored-form representation, the explicit setup of the matrix Q is required and the

[1]Computational Methods in Systems and Control Theory, Max Planck Institute for Dynamics of Complex Technical Systems, Sandtor-Str. 1, 39106 Magdeburg, Germany, benner@mpi-magdeburg.mpg.de

[2]Computational Methods in Systems and Control Theory, Max Planck Institute for Dynamics of Complex Technical Systems, Sandtor-Str. 1, 39106 Magdeburg, Germany, koehlerm@mpi-magdeburg.mpg.de

[3]Computational Methods in Systems and Control Theory, Max Planck Institute for Dynamics of Complex Technical Systems, Sandtor-Str. 1, 39106 Magdeburg, Germany, penke@mpi-magdeburg.mpg.de

blocks can only be accessed or applied afterwards. If only one or two blocks are required, e.g. to form a matrix-matrix product, this procedure results in a large overhead. The compact WY or VTV^T variants of the QR decomposition [2, 7, 5] can be used to reduce this overhead. Here, the Householder product (1) changes to

$$Q = I_m - VTV^T \overset{\substack{W=VT \\ Y=V}}{=} I_m - WY^T, \tag{3}$$

where $V = [v_1, \ldots, v_m]$ contains the Householder vectors v_i. The upper triangular matrix T can be computed from the Householder vectors and represents the accumulation of the Householder transformations. When it is additionally stored in explicit form it can be used to apply Q using level-3 BLAS operations, which are the foundation of fast linear algebra algorithms on current computer architectures. Furthermore, regarding the block partitioning of Q in (2), one can extract and apply blocks easily by using T and a partitioned V. The computation of the compact WY representation of the QR decomposition is part of LAPACK [1] in the `GEQRT3` routine. An improved parallel version, employing a Directed-Acyclic-Graph (DAG) for task scheduling, is part of the PLASMA library [3]. Parts of this algorithm have also entered the recent LAPACK 3.7 as the routine `GEQR`. The blocked variant of the QR decomposition typically makes use of the compact representation (3) of decomposed panels to update the trailing submatrix using matrix-matrix products. However, an equivalent of the `GEQRT3` routine, that also exploits the GPU's capabilities to compute the $n \times n$ triangular factor T, does not exist. The MAGMA library [8, 9] only provides QR decompositions which give the scalar factors $\tau_i = \frac{2}{v_i^T v_i}$ of the elementary reflectors or triangular matrices $T_i \in \mathbb{R}^{n_b \times n_b}$ (with n_b as the panel width) that represent the compact QR factorization of the individual panels. To build T from the T_i, a post-processing step would be necessary. Other GPU-accelerated libraries, such as ArrayFire [10], cuSOLVER[4], or CULATools[5], only support the classical representation.

In our contribution, we want to close this gap by presenting a GPU-accelerated approach to not only compute the QR factorization in factored-form representation but also provide T from the compact WY representation of the matrix Q.

We implement the blocked variant of the QR decomposition [5] as a CPU-GPU hybrid with additional operations to compute the T matrix. The matrix A is partitioned into panels. It generally resides in GPU memory during our computations. The current panel A_i is sent to the CPU to be factored:

$$A_i = Q_i R_i. \tag{4}$$

The LAPACK routines `GEQRT2` or `GEQRT3` can be used to compute T_i and V_i defining the compact representation of

$$Q_i = I - V_i T_i V_i^T. \tag{5}$$

The factored panel, consisting of V_i and R_i, and T_i are sent back to GPU memory. Here, T_i is used to update the trailing submatrix.

The following Lemma, which is proved by direct calculation, shows how T describing the compact representation of Q can be computed from the T_i given by the panel factorizations.

[4] http://docs.nvidia.com/cuda/cusolver/

[5] http://www.culatools.com/

Algorithm 1 Block Compact QR Decomposition with Reuse of VT^T

Require: $A \in \mathbb{R}^{m \times n}$

Ensure: V, R, T, S such that $A = QR$ with $Q = I_m - VTV^T$, $S = VT^T$.

A is overwritten by V, R. S, T can be stored together in an $m \times n$ array.

1: **for** $k = 1, \ldots$ **do**

2: $[R_k, V_k, T_{k,k}] \leftarrow QR(A_{k:p,k})$ ▷ by using `GEQRT2` on the host

3: Build new block column of T

$$T_{1:k-1,k} \leftarrow -\underbrace{T_{1:k-1,1:k-1} V_{1:k-1}^T}_{=S_{1:k-1}^T} V_k T_{k,k}$$

4: Update $S_{1:k-1}$ which currently holds $V_{1:k-1} T_{1:k-1,1:k-1}^T$

$$S_{1:k-1} \leftarrow S_{1:k-1} + V_k T_{1:k-1,k}^T$$

5: Build new block column of S

$$S_k \leftarrow V_k T_{k,k}^T$$

6: Update trailing submatrix of A

$$A_{k:p,k+1:q} \leftarrow A_{k:p,k+1:q} - \underbrace{V_k T_{k,k}^T}_{=S_k} V_k^T A_{k:p,k+1:q}$$

7: **end for**

Lemma 1 *Let* $Q_1 = I_m - V_1 T_1 V_1^T \in \mathbb{R}^{m \times m}$ *and* $Q_2 = I_m - V_2 T_2 V_2^T \in \mathbb{R}^{m \times m}$ *with* $V_1 \in \mathbb{R}^{m \times j}$, $V_2 \in \mathbb{R}^{m \times n_b}$ *and* $T_1 \in \mathbb{R}^{j \times j}$, $T_2 \in \mathbb{R}^{n_b \times n_b}$. *Then*

$$Q_1 Q_2 = I_m - V_+ T_+ V_+^T,$$

where

$$V_+ = \begin{bmatrix} V_1 & V_2 \end{bmatrix} \in \mathbb{R}^{m \times j + n_b}, \qquad T_+ = \begin{bmatrix} T_1 & -T_1 V_1^T V_2 T_2 \\ 0 & T_2 \end{bmatrix}.$$

While the QR factorization is being computed, we continuously build up the block columns of

$$T = \begin{bmatrix} T_{1,1} & \cdots & T_{1,q} \\ & \ddots & \vdots \\ 0 & & T_{q,q} \end{bmatrix}. \tag{6}$$

Lemma 1 gives

$$T_{1:k-1,k} = -T_{1:k-1,1:k-1} V_{1:k-1}^T V_k T_{k,k}, \tag{7}$$

where $V_{1:k-1}$ contains the Householder vectors of the previous panel factorizations and V_k contains the Householder vectors of the current panel factorization. We aim to implement this computation efficiently on the GPU employing cuBLAS routines. $T_{1:k-1,1:k-1} V_{1:k-1}^T$, or its transpose $V_{1:k-1} T_{1:k-1,1:k-1}^T$, is necessary to compute the k-th block column of T. It holds

$$\begin{aligned} T_{1:k,1:k} V_{1:k}^T &= (V_{1:k} T_{1:k,1:k}^T)^T \\ &= \begin{bmatrix} V_{1:k-1} T_{1:k-1,1:k-1}^T + V_k T_{1:k-1,k}^T & V_k T_{k,k}^T \end{bmatrix}^T. \end{aligned}$$

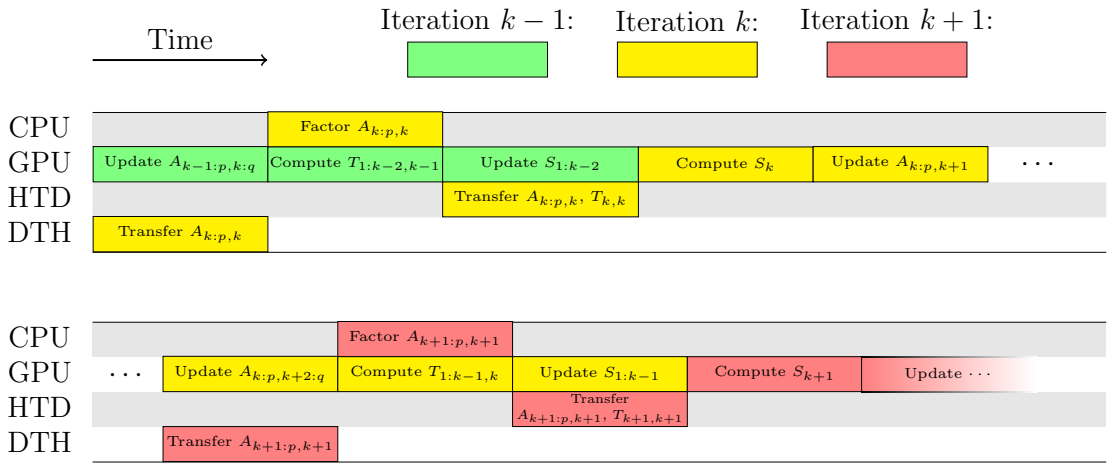

Figure 1: Course of events for the asynchronous version of Algorithm 1.

Table 1: Runtime in [s] for the CPU (LAPACK) and the GPU-accelerated version of the compact WY QR decomposition using a block width of 128.

Dimension	CPU	GPU	Speedup	Dimension	CPU	GPU	Speedup
1 000	0.050	0.048	1.04	9 000	8.532	2.404	3.55
2 000	0.228	0.120	1.90	10 000	11.181	3.202	3.49
3 000	0.583	0.212	2.75	11 000	14.504	4.115	3.52
4 000	1.139	0.335	3.40	12 000	18.067	5.182	3.49
5 000	1.984	0.551	3.60	13 000	22.211	6.566	3.38
6 000	3.183	0.852	3.76	14 000	27.064	8.003	3.38
7 000	4.668	1.260	3.70	15 000	32.319	9.768	3.31
8 000	6.373	1.762	3.62				

We see that, if $V_{1:k-1}T^T_{1:k-1,1:k-1}$ is available from the computation associated to the previous panel, it can be updated and expanded to provide $V_{1:k}T^T_{1:k,1:k}$. This is much cheaper than recomputing $T_{1:k,1:k}V^T_{1:k}$ for every panel, which is why we expand our algorithm to successively compute $V_{1:k}T^T_{1:k,1:k}$. The reason for the transpose is the fact that $V_{1:k-1}T^T_{1:k-1,1:k-1}$ is lower trapezoidal and can therefore be stored together with T in an $m \times n$ array. The new block column $V_kT^T_{k,k}$ can also be used in the update of the trailing submatrix.

These ideas are implemented by Algorithm 1 which realizes a QR factorization that includes the computation of T and VT^T. The panel factorization is performed by the CPU. The following steps are the updates of T, VT^T and the trailing submatrix. They are performed on the GPU by a series of cuBLAS routines, so that leading zeros and triangular structures are exploited to reduce the total amount of operations. Only a minimal further work space of size $n_b \times n$ (n_b denoting panel width) is required.

Furthermore we employ asynchronous communication to overlap CPU and GPU work. Hence, while the GPU is still busy updating the remaining trailing submatrix, the panel is transferred between device and host and can already be factored by the CPU. This is visualized by Figure 1. In the optimal case, that is depicted here, it is possible to have the GPU working to full capacity.

For the performance evaluation of the algorithm we use a dual-socket Intel Xeon E5-

2640v3 system (16 cores, 64 GB RAM) and an Nvidia Tesla K20m accelerator. The results are given in Table 1, where all computations are done in IEEE double precision. Here, the CPU reference result is computed using `GEQRT3` from OpenBLAS 0.2.18. For the panel factorization in the GPU-CPU hybrid implementation we use `GEQRT2` which we evaluated to be the fastest variant in LAPACK for the panel decomposition on the host. The GPU code uses the cuBLAS library of CUDA 8.0 and the whole code is compiled using gcc 4.8. In order to avoid the tuning for a special matrix structure the input square matrices consist of random entries distributed uniformly between 0 and 1. In Table 1 we see that the speedup increases very fast with the growing problem dimension to a factor of up to 3.76 while using only one GPU. For a further increasing problem dimension we obtain a stagnation caused by the limited memory bandwidth.

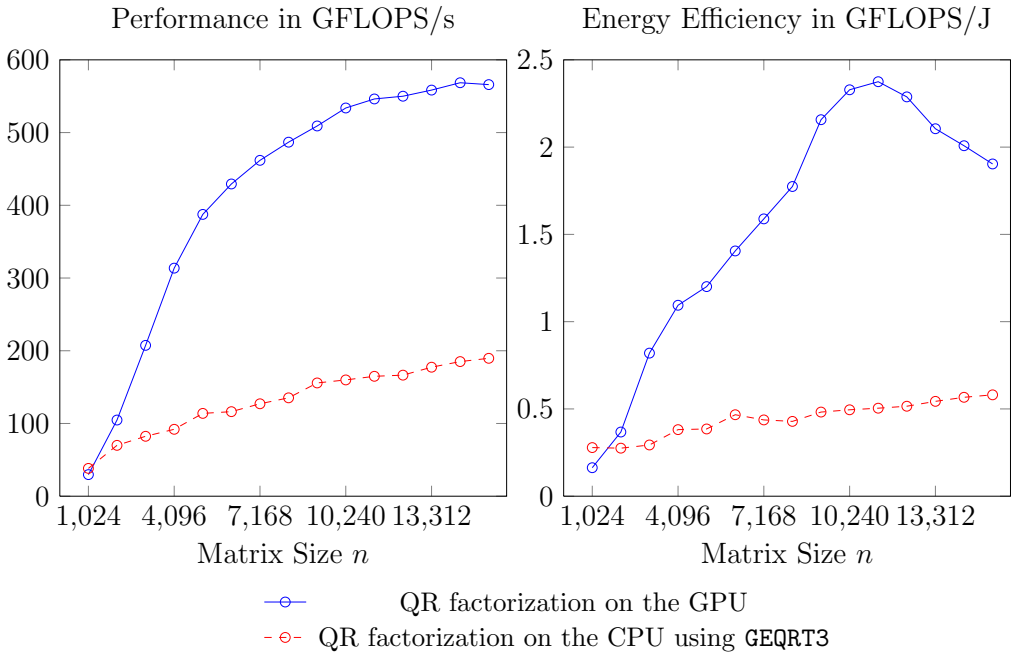

Figure 2: Comparison of approximate performance and energy efficiency of CPU-based and GPU-accelerated algorithm for square matrices.

The performance achieved by the GPU is also highlighted by Figure 2. We approximate the performance by assuming the number of FLOPs to be well represented by $\frac{5}{3}n^3$. This includes $\frac{4}{3}n^3$ from the QR factorizations and $\frac{1}{3}n^3$ from the computation of T ignoring lower order terms. The high performance is due to the GPU's capacities to perform many operations in parallel with little overhead. This also explains its higher energy efficiency which is depicted in the figure as well. Using the same amount of energy the GPU can perform up to 4.75 times as many operations as the CPU. This is achieved for large matrices with sizes of about 10000×10000, because here the GPU cores can be used to full capacity. To explain the following decline for even larger matrices further investigations are necessary.

We showed that the GPU is an excellent tool to compute the storage-efficient QR factorization. This is true with regards to classical FLOP performance as well as energy efficiency. An essential part of our implementation is the simultaneous computation of the VT^T matrix, which to our knowledge is new. To see how this approach compares

to the conventional ones implemented in LAPACK [4], an experimental setup is required that implements both variants on the same computer system, i.e., on the GPU or on the CPU. Future research will investigate this relation and provide further insight into the efficiency of the new approach.

References

[1] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK Users' Guide*, SIAM, Philadelphia, PA, third ed., 1999.

[2] C. Bischof and C. Van Loan, *The WY representation for products of Householder matrices*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. S2–S13.

[3] A. Buttari, J. Langou, J. Kurzak, and J. Dongarra, *Parallel tiled QR factorization for multicore architectures*, Concurr. Comput., 20 (2008), pp. 1573–1590.

[4] E. Elmroth and F. G. Gustavson, *Applying recursion to serial and parallel QR factorization leads to better performance*, IBM J. Res. Dev., 44 (2000), pp. 605–624.

[5] G. H. Golub and C. F. Van Loan, *Matrix Computations*, Johns Hopkins University Press, Baltimore, fourth ed., 2013.

[6] A. S. Householder, *The Theory of Matrices in Numerical Analysis*, Dover Publications (original Blaisdall 1964), New York, 1975.

[7] R. S. Schreiber and C. Van Loan, *A storage-efficient WY representation for products of Householder transformations*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 53–57.

[8] S. Tomov, J. Dongarra, and M. Baboulin, *Towards dense linear algebra for hybrid GPU accelerated manycore systems*, Parallel Comput., 36 (2010), pp. 232–240.

[9] S. Tomov, R. Nath, H. Ltaief, and J. Dongarra, *Dense linear algebra solvers for multicore with GPU accelerators*, in Proc. of the IEEE IPDPS'10, Atlanta, GA, Apr. 2010, IEEE Computer Society, pp. 1–8.

[10] P. Yalamanchili, U. Arshad, Z. Mohammed, P. Garigipati, P. Entschev, B. Kloppenborg, J. Malcolm, and J. Melonakos, *ArrayFire - A high performance software library for parallel computing with an easy-to-use API*, 2015.

Domain Knowledge Specification for Energy Tuning

Anamika Chowdhury[1]
Madhura Kumaraswamy, Michael Gerndt[2]
Zakaria Bendifallah, Othman Bouizi[3]
Lubomír Říha, Ondřej Vysocký, Martin Beseda, Jan Zapletal[4]

The European Horizon 2020 project READEX is developing a tool suite for dynamic energy tuning of HPC applications. While the tool suite supports an automatic approach, domain knowledge can significantly help in the analysis and the runtime tuning phase. This paper presents the means available in READEX for the application expert to provide his expert knowledge to the tool suite.

1 Introduction

Energy efficiency and consumption have become the most important and challenging issues in current HPC systems and in designing future exascale computing systems. Advances in hardware technology, the operating and tuning HPC applications are required to reduce the overall energy consumption. Aspects such as arithmetic intensity and resource utilization can be exploited to benefit energy savings due to their ability to characterize varying application behaviour. In previous works, the energy consumption was optimized by using a model-based approach to predict and statically set the best frequency for the entire application run. However, in READEX, we develop a tool suite that switches tuning parameters dynamically during the application execution based on the dynamic changes over runtime [3].

The READEX methodology is a two-stage approach and consists of Design Time Analysis (DTA) and Runtime Application Tuning (RAT). It uses the Periscope Tuning Framework (PTF) [1] for DTA, and the READEX Runtime Library (RRL) for runtime tuning, with Score-P [2] as the common instrumentation and measurement infrastructure. Pre-analysis steps are performed prior to the DTA in which the application is instrumented and analyzed for dynamism. Coarse granular program regions that constitute most of the execution time are selected for dynamic tuning and are identified as *significant regions* [5].

A novel tuning plugin was developed for PTF to perform DTA and run experiments that currently evaluate three tuning parameters: CPU frequency, uncore frequency and the number of OpenMP threads within a single program run. The experiments are executions of the so-called *phase region*, which is usually the body of the main progress loop and whose individual time steps are called *phases*. The plugin then determines the best *configuration* or settings of the tuning parameters for the *runtime situations* (rts's) of significant regions, i.e., its instances at runtime. Rts's that have similar characteristics are grouped into

[1]Technical University of Munich, Faculty of Informatics, Germany, chowdhua@in.tum.de

[2]Technical University of Munich, Faculty of Informatics, Germany, kumarasw@in.tum.de

[3]Intel ExaScale Labs, Paris, France, zakaria.bendifallah@intel.com

[4]IT4Innovations National Supercomputing Centre, VŠB-TUO, Ostrava, Czech Republic, lubomir.riha@vsb.cz, ondrej.vysocky@vsb.cz, martin.beseda@vsb.cz, jan.zapletal@vsb.cz

scenarios, and best configurations for those scenarios are set. The knowledge obtained during DTA, such as the best-found system configurations for individual scenarios is encapsulated in a *tuning model*. For production runs, this tuning model is forwarded to the READEX Runtime Library (RRL), which performs runtime tuning by dynamically switching to the best configurations for upcoming rts's.

READEX uses the so-called *identifiers* to predict at runtime the characteristics of an upcoming rts by letting the developer specify domain knowledge. Currently, READEX supports region identifiers to distinguish rts's, phase identifiers to distinguish phase characteristics and input identifiers to distinguish executions with different application inputs. Without these identifiers, rts's of a significant region may be merged into the same scenario even if they have different behaviour. Hence, these identifiers will improve the tuning model by distinguishing rts's and assigning them to different scenarios to potentially select a better configuration. The domain knowledge also includes Application-level Tuning Parameters (ATP) that switch the application control flow and expose tuning potential in the target application.

This paper describes how the domain knowledge is used by the READEX tuning plugin during DTA and in brief, the RAT.

Listing 1: Phase specification

```
#include "SCOREP_User.inc"

SCOREP_USER_REGION_DEFINE(R1)


do it=1,max_iter
! phase region begins
  SCOREP_OA_PHASE_BEGIN(R1,...)
    ...
    call VCycle(...)
    ...
  SCOREP_OA_PHASE_END(R1)
! phase region ends
enddo
```

Listing 2: Region identifiers

```
!--- VCycle ---!
...
!--- level k-1 to level k ---!
do  k = min_level+1,max_level
  call interpolate(...,k)
  call resid(...)
  call psinv(...)
enddo

!Interpolate to level k region
subroutine interpolate(...,k)
  SCOREP_USER_PARAMETER("level",k)
  ...
end subroutine
```

2 Domain Knowledge Specification

This section describes how the user can define the Score-P Online Access Phase, provide additional identifiers, and specify application-level tuning parameters. Listings 1 and 2 show a high-level view of the domain knowledge specification for the MG (MultiGrid) benchmark of the NAS parallel benchmark suite. MG uses a V-cycle to solve a discrete Poisson equation on a 3D grid. It is based on a hierarchy of grid levels, where the maximum level is the finest grid with the highest resolution.

During each iteration, an entire V-cycle is executed starting from the highest grid level. The residual on the current grid level k is projected to the next coarser grid level *k-1*. When the coarsest grid is reached, an approximate solution is computed. The result is then interpolated from the coarser to the finer grid, where the residual is calculated and a smoother is applied to correct the result. The result is then propagated further upwards.

2.1 Phase Specification

Before starting DTA, the user must annotate the phase region with Score-P macros. It must first be declared, as shown in line 3 in Listing 1, and then surrounded by begin and end marcros as shown in lines 8 and 12 respectively in Listing 1. The Score-P user manual provides more information on the parameters of the macros.

2.2 Identifier Specification

The READEX tool suite provides support for different types of identifiers for runtime situations.

Region identifiers: The user can specify region identifiers via Score-P user parameters to distinguish rts's of that region if the region has different characteristics in the runtime situations. For example, since the size of the grid processed in *interpolate(...,k)* gets larger when going from the minimum grid level to the maximum, at a certain grid level the computation switches from being compute bound to memory bound. To enable DTA to determine special system configurations for compute and memory bound rts's, a region identifier for the grid level is added to the code (Line 12 of Listing 2). The region name, the call path, and the region identifier are now used as identifiers of the different rts's.

Phase identifiers: DTA also exploits dynamicity in the characteristics of the application across phases. To do so, the application expert can provide phase identifiers as domain knowledge at the start of the phase via region identifiers for the phase region.

Input identifiers: READEX also improves the tuning model by identifying special system configurations for different inputs characteristics. For example, in the multigrid application, the grid level where the computation switches from compute to memory bound depends on the resolution of the finest grid and the number of MPI processes. The finer the grid, the more levels are memory bound. The more processes are used, the fewer levels are memory bound due to an increased amount of cache. Application specific input identifiers are specified in an accompanying input specification file in the form of key-value pairs. These specification files will be used by both PTF and RRL. The number of MPI processes and OMP threads will be known implicitly.

2.3 Application Tuning Parameters

In order to leverage application dynamism, READEX enables to exploit the dynamism available through the use of different code paths such as the use of different preconditioners in the ESPRESO FEM library or different blocking factors in stencil codes.

Part of the READEX tool-suite, the *ATP library* provides an API to annotate the source code in order to identify the control variables responsible for control flow switching. During the first phase of the application execution in DTA, variable types, value ranges and addresses are discovered and agglomerated into an *ATP description file*. The subsequent phases of the application execution are reserved for best configuration discovery. Parameter information collected in the ATP description file is exploited to test different parameter values.

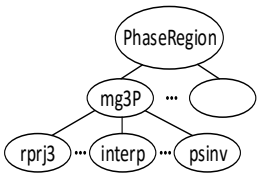

Figure 1: CCT of MG without user parameters

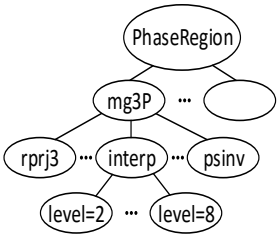

Figure 2: CCT of MG with user parameters

Furthermore, one critical complexity that the exploitation of ATPs exhibits is the presence of dependences between variables, where the values for one variable depend on the values of prior variable(s). In this case, not all value combinations are possible and forcing the values on the program may break its semantics. READEX, through the ATP library, provides the means to handle this by allowing the declaration of parameter dependences in the form of logical constraints. It also relies on a constraints solver called *the omega library*[5] to resolve the dependences. The solver handles affine function based constraints and provides valid combinations of parameter values for use in the DTA phase.

3 Implementation

DTA is carried out by PTF, a distributed framework consisting of a frontend and a hierarchy of analysis agents [1]. First, one phase of the application is executed in which the application regions are gathered and returned to the PTF analysis agents from Score-P via the Online-Access Interface and to generate the ATP specification file. Partial *Calling Context Trees* (CCT) [6] are generated at every analysis agent for those MPI processes controlled by the agent, and are gathered to create the complete tree in the frontend.

Figure 1 presents the CCT of MG if no region identifier is given. Separate nodes are created for the call sites of the projection, interpolation, and the smoother. All runtime situations, i.e. invocations, are represented by one node with its call path. Figure 2 illustrates the situation with the region identifier in the interpolation. For each grid level, a separate node is created and the runtime situations can be distinguished in PTF. Each rts is identified by its region name, the call path, which includes the region identifiers (represented as *parameter_name=value*), and the phase and input identifiers. With the region identifier, a valid rts of the region `interp` is */PhaseRegion/mg3P/interp/level=8*, as shown in Figure 2.

The PTF frontend executes the READEX tuning plugin, which reads from the READEX configuration file the objective(s) (Energy, CPU Energy, Execution Time, Energy Delay Product or Energy Delay Product Squared), the tuning parameters (core frequency, uncore frequency and the number of OpenMP threads), the search strategy (exhaustive, random, individual or genetic) and the significant regions. It also reads the input identifiers from the input specification file and the ATPs from the generated ATP specification file.

[5] http://www.cs.umd.edu/projects/omega/

[6] A context sensitive version of a call graph.

It then assesses selected system configurations from the search space of the tuning parameters and the ATPs generated by the search algorithm. For each configuration, it executes an experiment and measures the objective value for all the rts's of the significant regions. The plugin outputs the best configuration for both the phase and the rts's.

Finally, a *tuning model* is generated from this knowledge. The tuning model generation clusters the rts's into *scenarios* based on their best configuration. It determines a *classifier* that maps each valid rts onto a unique scenario based on the identifiers given at runtime. For each scenario, a *selector* is generated that returns a single or a set of good configurations for that scenario with respect to the chosen objective. The tuning model encapsulates this knowledge, and is stored as a JSON file, which is then read by the RRL to perform dynamic switching at runtime.

4 Example

The ESPRESO [4] library is a combination of Finite Element (FEM) tools and a domain decomposition based Finite Element Tearing and Interconnect (FETI) solvers. The FETI solver contains a projected conjugate gradient (PCG) solver and therefore, its convergence can be improved by several preconditioners. The computational complexity of different preconditioners vary from basic vector scaling (weight function), to sparse-matrix vector multiplication with different number of non-zeros (lumped, light-dirichlet) to dense matrix-vector multiplication (dirichlet). Using a simplified approximation, we can state that from the preconditioners listed above, the more computationally demanding the preconditioner is, the more numerically efficient it is, i.e. the more it reduces the number of iterations to solve the problem. In ESPRESO, we can dynamically switch between any of these during the runtime. If a preconditioner is not used, one iteration contains an action of a FETI operator (cost is 30.9 J and 0.12 s) and an application of a projector (cost is 0.7 J and 0.005 s). If a preconditioner is used, each iteration contains one more projector application in addition to the preconditioner action.

We evaluated the preconditioners on a structural mechanics (linear elasticity) problem with 2.3 million unknowns on a single compute node using 24 MPI processes. The results, see Table 1, show that the solution can be reached in 5.46 s when using Light Dirichlet preconditioner, despite the fact that it needs more iterations than the Dirichlet preconditioner. The Light Dirichlet preconditioner saved 15.9 s and 4 091.5 J in comparison to solving the problem without any preconditioner.

Preconditioner	**# iterations**	**1 iteration**		**Solution**	
none	172	125 ms	31.6 J	21.36 s	5 501.31 J
Weight function	100	130+2 ms	32.3+0.53 J	12.89 s	3 284.07 J
Lumped	45	130+10 ms	32.3+3.86 J	6.32 s	1 636.11 J
Light dirichlet	39	130+10 ms	32.3+3.74 J	5.46 s	1 409.82 J
Dirichlet	30	130+80 ms	32.3+20.62 J	6.34 s	1 594.50 J

Table 1: ESPRESO preconditioners comparison for runtime and energy consumption. The table contains (i) single iteration evaluation including baseline (FETI operator and 2x projector) + resources spent by the preconditioner (ii) overall FETI solver evaluation considering the different number of solver iterations.

5 Conclusion

This paper gave a short overview of the READEX project, which is aiming at improving the energy efficiency of HPC applications by a dynamic tuning approach. At design time, a tuning model that guides the dynamic switching of tuning parameters is determined. The quality of that tuning model can be enhanced by domain knowledge that is provided by the application owner. Part of the domain knowledge are application-level tuning parameters that significantly increase the tuning potential. The tuning potential is demonstrated in an example, where ATPs are used to select different preconditioners for the ESPRESO library.

References

[1] M. Gerndt, E. César, and S. Benkner, eds., *Automatic Tuning of HPC Applications - The Periscope Tuning Framework*, Shaker Verlag, Aachen, 2015.

[2] A. Knüpfer, C. Rössel, D. an Mey, S. Biersdorff, K. Diethelm, D. Eschweiler, M. Geimer, M. Gerndt, D. Lorenz, A. D. Malony, W. E. Nagel, Y. Oleynik, P. Philippen, P. Saviankou, D. Schmidl, S. S. Shende, R. Tschüter, M. Wagner, B. Wesarg, and F. Wolf, *Score-p: A joint performance measurement run-time infrastructure for Periscope, Scalasca, TAU, and Vampir*, in Tools for High Performance Computing 2011, H. Brunst, M. Müller, W. E. Nagel, and M. M. Resch, eds., Springer, Berlin, 2012, pp. 79–91.

[3] Y. Oleynik, M. Gerndt, J. Schuchart, P. G. Kjeldsberg, and W. E. Nagel, *Run-time exploitation of application dynamism for energy-efficient exascale computing (READEX)*, in Computational Science and Engineering (CSE), 2015 IEEE 18th International Conference on, C. Plessl, D. El Baz, G. Cong, J. M. P. Cardoso, L. Veiga, and T. Rauber, eds., Piscataway, Oct 2015, IEEE, pp. 347–350.

[4] L. Riha, T. Brzobohaty, A. Markopoulos, O. Meca, and T. Kozubek, *Massively Parallel Hybrid Total FETI (HTFETI) Solver*, in Proceedings of the Platform for Advanced Scientific Computing Conference, PASC '16, New York, NY, USA, 2016, ACM.

[5] J. Schuchart, M. Gerndt, P. G. Kjeldsberg, M. Lysaght, D. Horák, L. Říha, A. Gocht, M. Sourouri, M. Kumaraswamy, A. Chowdhury, M. Jahre, K. Diethelm, O. Bouizi, U. S. Mian, J. Kružík, R. Sojka, M. Beseda, V. Kannan, Z. Bendifallah, D. Hackenberg, and W. E. Nagel, *The READEX formalism for automatic tuning for energy efficiency*, Computing, (2017), pp. 1–9. DOI: 10.1007/s00607-016-0532-7.

Acknowledgements

The research leading to these results has received funding from the European Union's Horizon 2020 Programme under grant agreement number 671657.

Comprehensive Memory-Bound Simulations on Single Board Computers

Christian Himpe[1] Tobias Leibner[2] Stephan Rave[3]

Numerical simulations of increasingly complex models, demand growing amounts of (main) memory. Typically, large quantities of memory are provided by workstation- and server-type computers, but in turn consume massive amounts of power. Model order reduction can reduce the memory requirements of simulations by constructing reduced order models, yet the assembly of these surrogate models itself often requires memory-rich compute environments. We resolve this deadlock by careful algorithmic design of the model reduction technique. The presented empirical-cross-Gramian-based model reduction comprises two phases; in a first phase the empirical cross Gramian matrix is computed, secondly, a singular value decomposition of this system Gramian matrix reveals a low-rank projection, which can be applied to the original full order model. This model reduction approach can be realized economically memory-wise using the HAPOD algorithm, and we demonstrate its applicability on a low-end single board computer device.

1 Introduction

Numerical simulations of models based on parametric differential equations are an important tool in science and engineering. A common scenario is the repeated (multi-query, many-query) simulation for different parameters. Using high fidelity resolutions or more comprehensive models usually results in large-scale systems, which may even need to be processed on distributed memory systems due to memory or computational constraints. Such multi-node compute clusters consume wast amounts of power.

Model reduction can overcome computational complexity constraints by constructing algorithmically reduced order models. Yet, the assembly of the reduced model may require significant memory resources. Especially, data-driven model reduction techniques for the reduction of the time-domain representation of nonlinear systems need to simulate the full order model multiple times.

This work demonstrates that dense model reduction algorithms can be adapted to memory-constraints environments, not only by using reduced order models for the application simulation, the "online-phase"; but also for the assembly of the reduced order model,

[1]Computational Methods in Systems and Control Theory, Max Planck Institute for Dynamics of Complex Technical Systems, Sandtorstrasse 1, 39106 Magdeburg,
himpe@mpi-magdeburg.mpg.de

[2]Institute for Computational and Applied Mathematics, Westfälische Wilhelms Universtät, Einsteinstrasse 62, 48149 Münster,
tobias.leibner@uni-muenster.de

[3]Institute for Computational and Applied Mathematics, Westfälische Wilhelms Universtät, Einsteinstrasse 62, 48149 Münster,
stephan.rave@uni-muenster.de

the so-called "offline phase". Hence, a full-cycle memory economic simulation environment is provided. Due to the memory-resource independence, low-power or power-aware platforms become applicable for complex simulations.

In the scope of this work, input-output system models of the following form are considered:

$$\begin{aligned} \dot{x}(t) &= f(x(t), u(t)), \\ y(t) &= g(x(t), u(t)), \\ x(0) &= x_0, \end{aligned} \tag{1}$$

which consist of a dynamical system given by an ordinary differential equation (ODE) and an output function. This class of models maps an input function $u : \mathbb{R}_{\geq 0} \to \mathbb{R}^M$ via the state trajectory $x : \mathbb{R}_{\geq 0} \to \mathbb{R}^N$, that is the solution to the ODE with the vector field $f : \mathbb{R}^N \times \mathbb{R}^M \to \mathbb{R}^N$, to the output trajectory $y : \mathbb{R}_{\geq 0} \to \mathbb{R}^Q$ resulting from the output functional $g : \mathbb{R}^N \times \mathbb{R}^M \to \mathbb{R}^Q$.

2 Model Order Reduction (MOR)

Given an input-output system (1), an associated reduced order model has the form:

$$\begin{aligned} \dot{x}_r(t) &= f_r(x_r(t), u(t)), \\ y_r(t) &= g_r(x_r(t), u(t)), \\ x_r(0) &= x_{r,0}, \end{aligned}$$

with a reduced state $x_r : \mathbb{R}_{\geq 0} \to \mathbb{R}^n$, $n \ll N$, a reduced vector field $f_r : \mathbb{R}^n \times \mathbb{R}^M \to \mathbb{R}^n$, a reduced output functional $g_r : \mathbb{R}^n \times \mathbb{R}^M \to \mathbb{R}^Q$ and an approximate output $y_r : \mathbb{R}_{\geq 0} \to \mathbb{R}^Q$, such that $\|y - y_r\| \ll 1$. Various methods exist to obtain such a reduced order model. In the scope of this work we will focus on a data-driven approach from the class of projection-based model order reduction methods.

2.1 Projection-Based MOR

A popular class of model reduction methods uses truncated projections to construct reduced order models. The aim in projection-based model order reduction is to find a set of projections to, and back from, a coordinate system in which its base vectors are ordered by importance in some sense, so that lesser relevant directions can be truncated. Practically, a projection-based reduced order model to (1) is given by:

$$\begin{aligned} \dot{x}_r(t) &= V_1 f(U_1 x_r(t), u(t)), \\ y_r(t) &= \quad g(U_1 x_r(t), u(t)), \\ x_r(0) &= V_1 x_0, \end{aligned}$$

with the truncated reconstructing projection $U_1 \in \mathbb{R}^{N \times n}$, and the truncated reducing projection $V_1 \in \mathbb{R}^{n \times N}$, which are bi-orthogonal: $V_1 U_1 = I_n$.

2.2 Cross-Gramian-Based MOR

To delineate the subsequent nonlinear model reduction approach, the underlying linear model reduction technique is briefly summarized. Given a square ($M = Q$), linear system:

$$\begin{aligned}\dot{x}(t) &= Ax(t) + Bu(t),\\ y(t) &= Cx(t),\\ x(0) &= x_0,\end{aligned}$$

with a linear vector field consisting of $A \in \mathbb{R}^{N\times N}$, $B \in \mathbb{R}^{N\times M}$, and a linear output functional, $C \in \mathbb{R}^{Q\times N}$, the cross Gramian matrix [2] is defined as:

$$W_X := \int_0^\infty \mathrm{e}^{At}\, BC\, \mathrm{e}^{At}\, \mathrm{d}t \in \mathbb{R}^{N\times N}.$$

Following the approximate balancing technique in [9], a singular value decomposition (SVD) of this cross Gramian,

$$W_X \overset{\text{SVD}}{=} UDV$$

yields the projections $U \in \mathbb{R}^{N\times N}$ and $V \in \mathbb{R}^{N\times N}$. Truncating $N - n$ columns of U and rows of V based on the magnitude of singular values $D_{ii} \in \mathbb{R}_{\geq 0}$ gives the truncated reconstructing projection $U_1 \in \mathbb{R}^{N\times n}$ and truncated reducing projection $V_1 \in \mathbb{R}^{n\times N}$. The latter may be obtained by truncating V (Petrov-Galerkin approach) or using $U_1^\intercal$ (Galerkin approach). For further details on balancing methods, see [1].

2.3 Empirical-Cross-Gramian-Based MOR

A cross Gramian matrix can also be computed for nonlinear systems in data-driven manner, which is motivated by the following representation of the linear cross Gramian,

$$W_X = \int_0^\infty (\mathrm{e}^{At}\, B)(\mathrm{e}^{A^\intercal t}\, C^\intercal)^\intercal\, \mathrm{d}t,$$

as a product of the primal and dual (adjoint) impulse responses. For nonlinear systems, an adjoint system is generally not readily available as for linear systems. Yet, a cross Gramian can be computed purely based on state and output trajectory data: the empirical cross Gramian [6], which in a simplified variant is given by:

$$\widehat{W}_X := \frac{1}{M}\sum_{m=1}^{M}\int_0^\infty \Psi^m(t)\,\mathrm{d}t \in \mathbb{R}^{N\times N}, \tag{2}$$

$$\Psi^m_{ij}(t) = (x_i^m(t) - \bar{x}_i^m)(y_m^j(t) - \bar{y}_m^j) \in \mathbb{R}.$$

The $x_i^m(t)$ symbolizes the i-th state component for a simulation with the m-th perturbed input component, while $y_m^j(t)$ symbolizes the m-th output component for a simulation with j-th perturbed initial state component, and $\bar{x}_i^m$, $\bar{y}_m^j$ are averages of the respective trajectory components. for further details see [4].

3 Memory-Economic Computation

A major drawback of the empirical Gramians in general and the empirical cross Gramian in particular, is their dense structure of full order N. In this section we describe a memory-economic algorithm to compute the SVD of the empirical cross Gramian without assembling the full order cross Gramian.

3.1 Memory-Economic Empirical-Cross-Gramian-Based

The definition of the empirical cross Gramian (2) can be computed column-wise [5]:

$$\begin{aligned} W_X &= \begin{pmatrix} \omega_{X,1} & \dots & \omega_{X,n} \end{pmatrix} \in \mathbb{R}^{N\times N}, \\ \omega_{X,j} &= \frac{1}{M}\sum_{m=1}^{M}\int_0^\infty \psi^{mj}(t)\,\mathrm{d}t \in \mathbb{R}^{N\times 1}, \\ \psi_i^{mj}(t) &= (x_i^m(t) - \bar{x}_i^m)(y_m^j(t) - \bar{y}_m^j) \in \mathbb{R}. \end{aligned}$$

Thus the empirical cross Gramian can be assembled in blocks of columns, without any exchange of data between the steps of computing the column blocks. Following, an algorithm is presented to compute the singular vectors from the partitioned empirical cross Gramian incrementally, so only one partition has to kept in memory.

3.2 Memory-Economic SVD

The column-wise partitioning of the empirical cross Gramian is now reused to obtain the full empirical cross Gramian's left singular vectors U_1 associated to the dominant singular values, via a proper orthogonal decomposition (POD). The matrix of singular vectors acts as the truncated reconstructing projection and its transpose as truncated reducing projection. In the scope of this work we restrict ourselves to Galerkin projections $V = U^\intercal$, but Petrov-Galerkin-type projections are computable in a similar manner.

3.2.1 Hierarchical Approximate POD

To obtain the left singular vectors of the empirical cross Gramian, given in a column-wise block partitioning, a method related to the SVD, the hierarchical approximate proper orthogonal decomposition (HAPOD) from [5] is utilized.

The HAPOD algorithms allows to compute the dominant left singular vectors U_1 of a given column-wise partitioned data-set, for example incrementally, such that the mean ℓ_2 projection error is bounded from above by $\|W_X - U_1U_1^\intercal W_X\|_{\ell_2} < \varepsilon$. Given an upper bound ε and a partitioning $W_X = [\omega_1, \dots, \omega_S]$, with ω_s containing K_s columns, this "live HAPOD" computes as:

$$\begin{aligned} &\hat{u}_0 := \{\}, \\ &[\omega_s, \hat{u}_{s-1}] \overset{\text{SVD}}{=} u_s d_s v_s \to \hat{u}_s := u_s \hat{d}_s, \quad \hat{d}_{s,ii} = \begin{cases} d_{s,ii} & d_{s,ii} < \varepsilon^2 K_s \sqrt{\frac{\sum_{j=1}^{s} K_j}{S}} \\ 0 & \text{else} \end{cases} \\ &U_1 := \hat{u}_S \to V_1 = U_1^\intercal. \end{aligned}$$

The combination of the partitioned empirical cross Gramian with the live HAPOD allows a computation of a low-rank reducing Galerkin projection and hence a projection-based reduced order model in a memory economic manner, as the full empirical cross Gramian is never needed and the partitioned only requires communication to forward reduced base components.

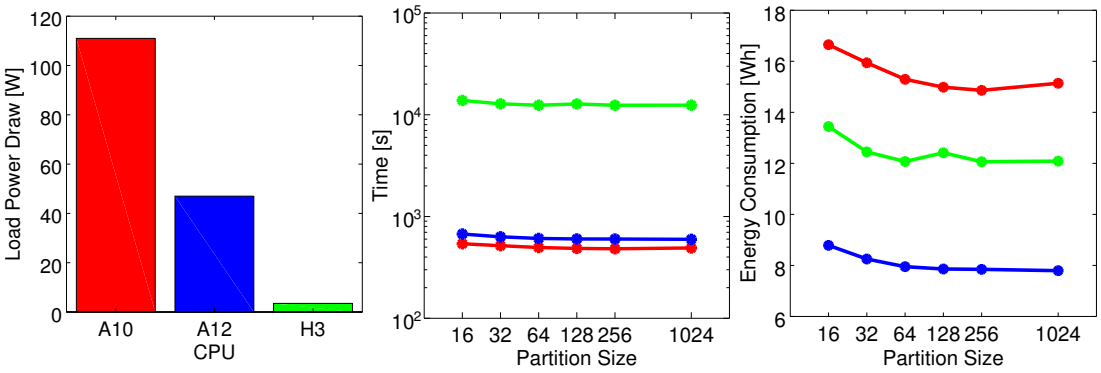

Figure 1: Comparison of power draw under load, computational time (offline phase) and energy consumption on different compute systems and partitionings.

4 Numerical Example

To illustrate this memory-economic model reduction technique combining the empirical cross Gramian with the HAPOD, a nonlinear hyperbolic network model is utilized,

$$\begin{aligned} \dot{x}(t) &= A\tanh(Kx(t)) + Bu(t), \\ y(t) &= Cx(t), \\ x(0) &= 0. \end{aligned}$$

Exemplary, a sparse but stable system matrix $A \in \mathbb{R}^{1024\times1024}$, a sparse random input matrix $B \in \mathbb{R}^{1024\times1}$, a random output matrix $C \in \mathbb{R}^{1\times1024}$, and a diagonal random gain matrix $K \in \mathbb{R}^{1024\times1024}$, $K_{ii} = \mathcal{U}_{[0,1]}$ is selected for this test.

This model is reduced using the conjoined methods of the empirical cross Gramian and the HAPOD on three different compute systems:

- Desktop:

CPU AMD **A10**-7800 (64-bit Quad-Core x86-64) @ 3.9Ghz
SIMD AVX, FMA3 & FMA4
RAM 32GB DDR3-2133 (dual rank & dual channel)

- Thin Client:

CPU AMD **A12**-9800E (64-bit Quad-Core x86-64) @ 3.1Ghz
SIMD AVX2, FMA3 & FMA4
RAM 32GB DDR4-2133 (dual rank & dual channel)

- Single Board Computer:

CPU Allwinner **H3** (32-bit Quad-Core ARM Cortex A7) @ 1.2Ghz
SIMD NEON, VFP4
RAM 0.5GB DDR3-1600 (single rank & single channel)

using `emgr` - empirical Gramian framework [3] in GNU Octave [10] with OpenBLAS [8] via FlexiBLAS [7].

In Figure 1 average power draw under load for each of the systems is depicted, as well as the computational time and consumed energy for the partitioned and unpartitioned (one partition of order 1024) empirical-cross-Gramian-based computation.

The power draw during computational load among the three architectures ranges from the 111 W (A10), over 47 W (A12), to 3.5 W (H3). Even though the A12 requires 23% more time on average than the A10 to obtain the solution, the A12 consumes only about half the energy. The H3 requires expectedly significantly longer computational time but still consumes less energy than the A10. It should be emphasized that the H3 is a 32-bit (ARM) architecture and additional factors, such as cooling have to be considered more carefully than on x86 platforms, hence this comparison is not completely fair.

The partitioning has a minor effect on computational time and energy consumption: Longer times and more energy are required for small partition sizes. For larger partition sizes a slight reduction in time and thus energy consumption can be observed. Overall, the combination of partitioned empirical cross Gramian and hierarchical approximate proper orthogonal decomposition enables model reduction in compute- or memory-limited environments such as single board computers at little to no additional cost.

Acknowledgements

This work is supported by the German Federal Ministry for Economic Affairs and Energy, in the joint project: "**MathEnergy** – Mathematical Key Technologies for Evolving Energy Grids", sub-project: Model Order Reduction (Grant number: 0324019**B**).

References

[1] A. Antoulas, *Approximation of Large-Scale Dynamical Systems*, vol. 6 of Advances in Design and Control, SIAM Publications, Philadelphia, PA, 2005.

[2] K. V. Fernando and H. Nicholson, *On the structure of balanced and other principal representations of siso systems*, IEEE Trans. Autom. Control, 28 (1983), pp. 228–231.

[3] C. Himpe, *emgr – EMpirical GRamian framework (Version 5.0)*. `http://gramian.de`, 2016.

[4] ——, *emgr - the Empirical Gramian Framework*, arXiv e-prints 1611.00675, Cornell University, 2016. cs.MS.

[5] C. Himpe, T. Leibner, and S. Rave, *Hierarchical approximate proper orthogonal decomposition*, arXiv e-prints 1607.05210, Cornell University, 2016. math.NA.

[6] C. Himpe and M. Ohlberger, *Cross-Gramian based combined state and parameter reduction for large-scale control systems*, Mathematical Problems in Engineering, 2014 (2014), pp. 1–13.

[7] M. Köhler and J. Saak, *FlexiBLAS - A flexible BLAS library with runtime exchangeable backends*, Tech. Rep. 284, LAPACK Working Note, Jan. 2014.

[8] *OpenBLAS*. `http://www.openblas.net`.

[9] D. C. Sorensen and A. C. Antoulas, *The Sylvester equation and approximate balanced reduction*, Numer. Lin. Alg. Appl., 351–352 (2002), pp. 671–700.

[10] The Octave Developers, *GNU Octave*. `http://octave.org`.

List of Participants

Name	First Name	Affiliation	Email	Country	Page
Aliaga	José I.	Universitat Jaume I	aliaga@uji.es	Spain	45
Anzt	Hartwig	University of Tennessee	hanzt@icl.utk.edu	USA	8
Barreda Vaya	Maria	Universitat Jaume I	mvaya@uji.es	Spain	39
Benner	Peter	MPI for Dynamics of Complex Technical Systems	benner@mpi-magdeburg.mpg.de	Germany	
Bollhöfer	Matthias	TU Braunschweig	m.bollhoefer@tu-bs.de	Germany	
Bolten	Matthias	University of Kassel	bolten@mathematik.uni-kassel.de	Germany	5
Chowdhury	Anamika	Technical University of Munich (TUM)	chowdhua@in.tum.de	Germany	57
Dongarra	Jack	University of Tennessee	dongarra@cs.utk.edu	USA	5
Dufrechou	Ernesto	Facultad de Ingeniería, UDELAR	edufrechou@fing.edu.uy	Uruguay	
Ezzatti	Pablo	Facultad de Ingeniería, UDELAR	pezzatti@fing.edu.uy	Uruguay	
Faßbender	Heike	TU Braunschweig	h.fassbender@tu-braunschweig.de	Germany	
Geveler	Markus	TU Dortmund University	markus.geveler@math.tu-dortmund.de	Germany	9
González	Jesús	University of Granada (SPAIN)	jesusgonzalez@ugr.es	Spain	27
Grigori	Laura	INRIA Paris	laura.grigori@inria.fr	France	6
Himpe	Christian	MPI for Dynamics of Complex Technical Systems	himpe@mpi-magdeburg.mpg.de	Germany	63
Kang	Kab Seok	MPI for Plasma Physics	kskang@ipp.mpg.de	Germany	
Klawonn	Axel	University of Cologne	klawonn@math.uni-koeln.de	Germany	7
Kumaraswamy	Madhura	Technische Universität München	madhura.kswamy@gmail.com	Germany	
Köhler	Martin	MPI for Dynamics of Complex Technical Systems	koehlerm@mpi-magdeburg.mpg.de	Germany	33
Marek	Andreas	MPCDF	andreas.marek@mpcdf.mpg.de	Germany	
Ortega	Julio	University of Granada (SPAIN)	jortega@ugr.es	Spain	
Penke	Carolin	MPI for Dynamics of Complex Technical Systems	penke@mpi-magdeburg.mpg.de	Germany	51
Pfennig	Tobias	MEGWARE Computer Vertrieb und Service GmbH	tobias.pfennig@megware.com	Germany	
Quintana-Orti	Enrique S.	Universitat Jaume I	quintana@icc.uji.es	Spain	
Rampp	Markus	MPI and Data Facility	mjr@mpcdf.mpg.de	Germany	
Remón	Alfredo	MPI for Dynamics of Complex Technical Systems	remon@mpi-magdeburg.mpg.de	Germany	
Rojek	Krzysztof	Czestochowa University of Technology	krojek@icis.pcz.pl	Poland	17
Rüde	Ulrich	FAU Erlangen-Nürnberg	ulrich.ruede@fau.de	Germany	6
Saak	Jens	MPI for Dynamics of Complex Technical Systems	saak@mpi-magdeburg.mpg.de	Germany	
Thorne	Sue	Science and Technology Facilities Council	sue.thorne@stfc.ac.uk	UK	11
Weber	Michael	MEGWARE Computer Vertrieb und Service GmbH	michael.weber@megware.com	Germany	

Information to Participants

On Site

- Conference room:
  All talks will take place at the *lecture hall*. To reach it, exit the main entrance and keep to the right.

- Coffee breaks:
  Will be served next to the *lecture hall*.

- Meals:
  Breakfasts, lunches and dinners will be served at the dining room, placed at the ground floor at the main building at 8:00-9:00, 12:30-13:30, and 18:30-20:00 o'clock.

- WLAN:
  Internet access is available at most of the facilities (including all rooms and the *lecture hall*). All the necessary information to access the wireless network are provided by the castle's staff and information boards around the castle.

- Wallberg trip:
  All participants and companions are invited to join the Wallberg trip. A lunch box will be provided before departure. There is a funicular that can be used to go up and down to the hill. Participants are free to decide whether to use the funicular in one or both directions. Hiking shoes are recommended. Sturdy shoes are needed for all hike options and one option includes a trail with rocky parts. Details will be announced on site.

- There will be drinks and coffee always available at the *garden room*. If you take any, please write it down in the list you will find next. You can pay at the reception desk on your check-out.

- More info about the Ringberg castle and the surroundings can be found at its webpage and the following APP:

For Speakers

- Please make sure that your presentation is transferred to the computer connected to the beamer before your session starts.
- Ask the local organizers if you have any question.

Local organizers

- Prof. Dr. Peter Benner
- Dr. Jens Saak
- Dr. Alfredo Remón
- Martin Köhler
- Janine Holzmann[1] / Diana Noatsch-Liebke[1]

Important phone numbers

- Emergency number: **112**
- Castle's reception: **+49 (0)802 227 90**

[1] not on site

Max Planck Institute for Dynamics of Complex Technical Systems
Computational Methods in Systems and Control Theory