Karlsruhe Institute of Technology

Analysis and Optimization of Power Consumption in the Iterative Solution of Sparse Linear Systems on Multi-core and Many-core Platforms

H. Anzt, M. Castillo, J. I. Aliaga, J. C. Fernández,
V. Heuveline, R. Mayo, E. S. Quintana-Ortí

No. 2011-05

Preprint Series of the Engineering Mathematics and Computing Lab (EMCL)

KIT – University of the State of Baden-Wuerttemberg and
National Research Center of the Helmholtz Association

www.emcl.kit.edu

Preprint Series of the Engineering Mathematics and Computing Lab (EMCL)
ISSN 2191–0693
No. 2011-05

Impressum

Karlsruhe Institute of Technology (KIT)
Engineering Mathematics and Computing Lab (EMCL)

Fritz-Erler-Str. 23, building 01.86
76133 Karlsruhe
Germany

KIT – University of the State of Baden Wuerttemberg and
National Laboratory of the Helmholtz Association

Analysis and Optimization of Power Consumption in the Iterative Solution of Sparse Linear Systems on Multi-core and Many-core Platforms

Hartwig Anzt, Vincent Heuveline
Institute for Applied and Numerical Mathematics 4
Karlsruhe Institute of Technology
Fritz-Erler-Str. 23, 76133 Karlsruhe, Germany
{*hartwig.anzt,vincent.heuveline*}*@kit.edu*

José I. Aliaga, Maribel Castillo, Juan C. Fernández,
Rafael Mayo, Enrique S. Quintana-Ortí
Depto. de Ingeniería y Ciencia de Computadores
Universidad Jaume I
12.071 - Castellón, Spain
{*aliaga,castillo,jfernand,mayo,quintana*}*@icc.uji.es*

***Abstract*—Energy efficiency is a major concern in modern high-performance-computing. Still, few studies provide a deep insight into the power consumption of scientific applications. Especially for algorithms running on hybrid platforms equipped with hardware accelerators, like graphics processors, a detailed energy analysis is essential to identify the most costly parts, and to evaluate possible improvement strategies. In this paper we analyze the computational and power performance of iterative linear solvers applied to sparse systems arising in several scientific applications. We also study the gains yield by dynamic voltage/frequency scaling (DVFS), and illustrate that this technique alone cannot to reduce the energy cost to a considerable amount for iterative linear solvers. We then apply techniques that set the (multi-core processor in the) host system to a low-consuming state for the time that the GPU is executing. Our experiments conclusively reveal how the combination of these two techniques deliver a notable reduction of energy consumption without a noticeable impact on computational performance.**

***Keywords*-Energy Efficiency, Scientific Computing, Sparse Linear Systems, Iterative Solvers, Power and Performance Analysis and Optimization, DVFS**

I. Introduction

As we approach the Exascale computing era, the focus of the scientific computing community increasingly turns into deriving energy efficient systems, that are able to tackle applications with low power consumption. The reason is, that already today, the running costs for energy often exceed the acquisition cost of a hardware platform [1]. But the economic issue is not the only problem. Appropriate infrastructure able to supply this amount of energy is not always available, and the concerns about an energy crisis and global warming lead to even another level of consideration.

For these reasons, a significant number of researchers working on scientific computing, technical engineering and mathematical modeling have driven their research focus towards power-aware computing [2], [3]. However, while the different experts can improve the factors related to the respective fields they are working in, only the combination of their competences can lead to considerable improvements: the hardware has to be optimized with respect to power consumption, the applications have to be adapted to leverage this hardware, the implementations have to optimize the usage of all available hardware resources and distribute the workload to improve the efficiency, etc.

In this paper, we analyze the iterative solution of sparse symmetric positive definite (SPD) linear systems. This type of problems naturally arises in many applications that require the solution of partial differential equations (PDE) modeling physical, chemical or economical processes. Depending on the specific PDE and the finite element discretization method, the resulting system exhibits the symmetry and positive definiteness properties. A well-known example of an SPD system is the finite difference discretization of a 2D or 3D Laplace problem; see, e.g., [4]. While direct solvers can deal with small to medium-sized sparse linear systems, large-scale systems usually require the use of low-cost iterative solvers based on Krylov Subspace-based methods [5]. If the coefficient matrices are symmetric and positive definite, the Conjugate Gradient (CG) method and its preconditioned variants (PCG), which usually combine a higher robustness with some performance gain, are especially appealing. The main contribution of this paper is a practical demonstration of how the energy consumption of iterative solvers for SPD linear systems can be reduced considerably by using hybrid hardware platforms, adapting the solver to the system, and applying energy-saving techniques like DVFS combined with a low-consuming state. To achieve this purpose, we split the paper into the following parts:

1) We first overview the test framework used in our experiments. This includes a detailed description of the hardware platform and the measurement setup. Additionally, we describe the solver types and the specific linear systems employed in the evaluation.
2) We then perform a detailed analysis of the energy consumption of the different variants of the solver, using the various hardware components, to determine where and how energy saving techniques can be applied.
3) DVFS is known to influence the time/energy trade-off of the solving process [6], [7]. While for CPU-bounded applications there is usually a linear cor-

relation between time and power consumption, this may not be true for cases where the memory bandwidth is the bottleneck [8]. Here we show that, for the solution of sparse linear systems (in general, a memory-bounded application) on hybrid CPU-GPU platforms, DVFS alone is not sufficient to decrease the power consumption. On CPUs, lowering the frequency/voltage increases execution time which blurs energy savings. On the other hand, when the GPU is employed to execute CUDA kernels, the CPU performs a busy-wait with little difference between high/low frequencies from the power consumption perspective.

4) To avoid the negative consequences of the busy-wait, we implement a function that puts the CPU to "sleep" for the time of the GPU kernel call. This is one possibility that actually turns the system into idle mode, and leads to a considerable decrease in the power consumption of the host.
5) To assess the improvements, we apply the different solver implementations to a variety of SPD linear systems, obtained from the finite element discretization of a Laplace problem or available from the University of Florida matrix collection (UFMC; see http://www.cise.ufl.edu/research/sparse/matrices/).
6) In the last section we offer a number of conclusions and a brief overview about open problems that have to be addressed in the future, to enhance the energy efficiency of linear solvers further.

II. Framework for Experiments

A. Hardware Platform and Linear Algebra Libraries

All experiments were performed on a platform consisting of an AMD Opteron 6128 processor with eight cores running at 2.0 GHz, with 12 MB of shared L3 cache and 24 GB of RAM. The system is connected via PCIe (16x) to an NVIDIA Tesla C1060 card (240 processor cores) with 4 GB of GDDR3 global memory. The double-precision peak performance of the multi-core processor is 64 GFLOPS ($64 \cdot 10^9$ floating-point arithmetic operations per second) and that of the GPU is 78 GFLOPS. When possible, we invoked the tuned implementation from Intel MKL (version 11.1) to perform BLAS-1 operations (e.g., for the dot product, the axpy, etc.) on the AMD processor. Although MKL also includes an implementation of the sparse matrix-vector multiplication (necessary in the CG iteration), we decided to employ our own plain implementation of this kernel. Independent experiments showed the superior parallel efficiency of our implementation for the specific matrices involved in the experiments. In order to improve performance, the compilation of the CPU code was done using the GNU `gcc` compiler (version 4.4.3) with the flag `-O3`, which enables aggressive optimizations on the AMD multi-core processor.

Table I: Dimensions of the SPD test matrices.

Matrix name	Size (n)	Nonzeros (nnz)
A318	32,157,432	224,495,280
APACHE2	715,176	4,817,870
AUDIKW_1	943,695	77,651,847
BONES10	914,898	40,878,708
ECOLOGY2	999,999	4,995,991
G3_CIRCUIT	1,585,478	7,660,826
LDOOR	952,203	42,493,817
ND24K	72,000	28,715,634

On the GPU, the BLAS-1 operations were performed using the respective CUBLAS routines from [9] (version 3.0). NVIDIA `nvcc` compiler (version 3.2), with an up-to-date CUDA driver (version 3.2) was employed in the GPU. A specific kernel for the computation of the sparse matrix-vector multiplication on the GPU was implemented following the guidelines suggested in [10].

B. Measurement Setup

Power was measured using an ASIC operating at a frequency of 25 Hz (25 samples per second) and composed of a number of resistors connected in series with the power source. This internal power meter was attached to the lines connecting directly the power supply unit with the GPU and the motherboard (chipset plus processors), to obtain the energy consumption of the computing hardware. Samples from this device were collected in a separate system, so that they do not affect the performance of the tests. Figure 1 captures the connection between the target platform and the energy measurement hardware.

C. Linear Systems

We iteratively compute a solution approximation to the linear system $Ax = b$, where A is one of the matrices listed in Table I and b is a vector with all entries equal 1. The coefficient matrices of the linear system are derived from a finite difference discretization of the 3D Laplace problem (example A318) or taken from the UFMC of freely available matrices arising in scientific applications (all other cases). For simplicity, in the iterative solver we set the initial guess to start the iteration process to $x_0 \equiv 0$, despite there exist sophisticated methods to approximate an optimal initial solution.

We set the relative residual stopping criterion of the solvers to $\varepsilon = 10^{-10}\|r_0\|_2$, where r_0 is the initial residual. As we chose $x_0 \equiv 0$, then $r_0 = b - Ax_0 = b$ and $\varepsilon = 10^{-10}\sqrt{n}$.

D. Iterative Linear Solver

We employ CG [5] to iteratively solve the different linear systems. To increase the performance and the robustness of the solution process, we also implement a preconditioned variant using a Jacobi preconditioner [5]. Note that, as this preconditioner only weights the main diagonal, it will

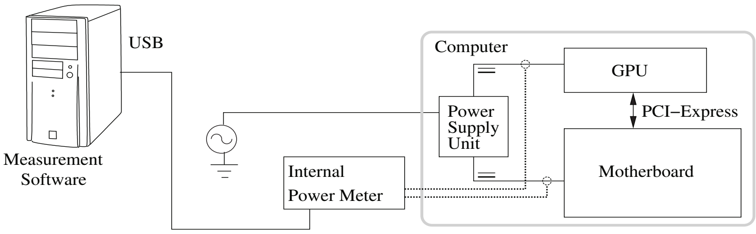

Figure 1: Hardware platform and sampling points.

Table II: Energy Consumption of different implementations of CG solver for G3_CIRCUIT.

Hardware	# iter	Time [s]	Energy consumption [Wh]		
			Chipset	GPU	Total
CPU 1T	21424	1674.45	53.96	-	53.96
CPU 2T	21424	1307.21	45.70	-	45.70
CPU 4T	21424	1076.97	42.18	-	42.18
CPU 8T	21424	1113.34	50.54	-	50.54
GPU 4T	21467	198.43	8.04	3.44	11.48

Table III: Energy Consumption of different implementations of PCG solver for G3_CIRCUIT.

Hardware	# iter	Time [s]	Energy consumption [Wh]		
			Chipset	GPU	Total
CPU 1T	4613	601.97	18.94	-	18.94
CPU 2T	4613	417.33	14.22	-	14.22
CPU 4T	4613	348.79	13.31	-	13.31
CPU 8T	4613	362.44	16.25	-	16.25
GPU 4T	4613	46.28	1.89	0.83	2.72

usually not improve the efficiency of the solution process of linear systems obtained from a Laplace stencil (e.g., A318). Furthermore, as the computation of the preconditioner matrix is sequential, it is always performed by the CPU of the system. Within the iteration process, the difference between the CG and the Jacobi-PCG is one additional matrix-vector multiplication involving the preconditioner matrix.

The different implementations of the solver use either the CPU or the GPU for the sparse matrix-vector product and BLAS-1 operations.

III. Detailed Power Consumption Analysis

In this first test, we monitor the power consumption of the different implementations of CG and PCG on CPU and GPU. To assess the relevance for scientific applications, we choose the linear system G3_CIRCUIT (see Table I) for this test, as this example exhibits a moderate dimension and a complex sparsity pattern.

In Tables II and III we report the results obtained with the CPU solvers using OpenMP with 1, 2, 4 and 8 threads/cores (results labelled as “CPU nT” with “n” equal the number of threads), and compare the execution time (in seconds, s) and the energy consumption (in Watts-hour, Wh) to those of the GPU implementations, performing the BLAS-1 operations and the sparse matrix-vector product on the GPU, and using a total of 4 CPU threads/cores for the remaining operations on the CPU (results labelled as “GPU 4T”).

We observe that, for the CG as well as for the preconditioned variant, the GPU implementation outperforms all CPU solvers by a large factor. This is true for the execution time as well as for the energy cost; for the latter parameter, the improvement is smaller, since the power consumption of both the CPU and GPU have to be taken into account. For the CG solver, the optimal CPU-based configuration (use of 4 threads) results in more than 5× higher execution time and about 4× higher energy consumption. For the preconditioned variant, where the advantage of less iterations is payed off by an additional matrix-vector multiplication in every iteration loop, the speedup and energy saving derived from the use of the GPU as coprocessor are even higher. The optimal CPU implementation can at most reach 1/7 of the GPU performance and consumes more than 480% more energy.

Although most scientific codes target clusters with general-purpose processors, this test validates the assumption that the use of GPUs for elementary kernel operations may improve the overall performance of parallel scientific applications. Despite the additional initialization process and data transfer to the GPU, the high number of computing cores on a graphics processor compensates these overheads, and enables the GPU to perform parallel instructions faster and with higher energy efficiency. The aim of the following analysis is to extract information on how to further reduce the energy consumption of the solvers.

IV. Dynamic Voltage and Frequency Scaling

There is a common belief that, for memory-bounded operations, the power consumption can be reduced by lowering the frequency/voltage operation of the processor (enabled in current architectures by DVFS modules). The next experiment shows that, in general, this is not true for the solution of sparse linear systems using CPU/GPUs. The reason is that often the kernel calls to perform an intensive computation on the GPU set the (CPU) host system in a busy-wait. In this state, the CPU does not perform any computations, but exhibits a power consumption even higher than that of running idle!

Table IV: Energy Consumption of different implementations of CG solver for A318.

Hardware	Freq.	Time	Power/Energy consumption		
			Chipset	GPU	Total
	[MHz]	[s]	[∅ W]	[∅ W]	[Wh]
CPU 1T	2,000	2059.69	116.78	-	66.81
CPU 1T	800	3400.64	103.50	-	97.75
CPU 2T	2,000	1708.31	120.30	-	57.08
CPU 2T	800	2196.63	105.60	-	64.44
CPU 4T	2,000	1441.78	123.99	-	49.66
CPU 4T	800	1674.62	108.11	-	50.29
CPU 8T	2,000	1395.37	129.33	-	50.13
CPU 8T	800	1481.48	110.46	-	45.45
GPU	2,000	253.22	149.04	61.89	14.84
GPU	800	254.25	138.50	61.45	14.12

For the following tests, we move to the linear system A318 (see Table I). This choice is motivated by the measurement device we are using to monitor the power consumption. In particular, to ensure the ASIC is being able to accurately sample the power consumption, we need a large-scale system, so that the time frames of the BLAS-1 operations and the sparse matrix-vector product are in the order of the measurement frequency.

Table IV reports the execution time, average power demanded by the chipset and GPU (in Watts, ∅ W) and total energy consumption of the codes when DVFS is employed to set the operating frequency of the AMD cores to 2.00 GHz and 800 MHz for the time of the CG solver. Since rescaling the CPU frequency is on one side only beneficial when applied to all cores, and introduces some overhead on the other side, additional rescaling inside the algorithm does not improve the overall performance. The results show that, for the CPU-based codes, there is no direct correlation between computation time and the total energy cost. Reducing the CPU-frequency using DVFS lowers the power consumption of the CPU, but since less operations can be conducted per unit of time, it increases the computation period. In the end, lowering the frequency not necessarily yields improvement to the the energy cost. Using more cores reduces the execution time, but increases the energy consumption. Since not only the cores are consuming, also the memory and the chipset demand for some power, whether increasing the number of cores pays off depends on the application and the memory bandwidth. Only if the application is CPU-bounded increasing the number of cores reduces the energy consumption; for memory-bounded operations this may not be true in general. On NUMA architectures, like the one employed in the experiments, using a large number of cores increases also the memory bandwidth, which may lead to a different ratio between the memory- and CPU workload. This effect can be observed when 8 threads are used.

Again, we observe the superiority of the GPU implementations; for those, using DVFS is beneficial, since the operations conducted by the CPU are few and do not demand a high operation frequency.

Furthermore we observe that the graphics card consumes some power over the PCIe. Although we can not give explicit numbers for this energy transfer, the specifications do not allow a higher output than 75 Watts via PCIe [11].

Using the GPU-implementation for this test case, the improvement gained by using DVFS counts up to at most 5.6% of the total energy consumption. The reason for this moderate result is that calling a GPU kernel operation sets the CPU into a busy-wait, a mode where the host system is waiting for feedback from the kernel, sending steadily requests to the device. This results in a energy consumption that equals or is even superior to a system running on full demand.

V. Idle-wait

The previous section showed that lowering the operation frequency of the CPU cores does not yield a significant reduction of power consumption of the host system during the execution of kernels on the device. The reason for this is that the busy-wait status is highly energy inefficient. In this section, we show a possible solution to this problem which sets the host system to sleep for the time of the kernel execution. There may exist other more advanced alternatives, that utilize synchronizing tools provided by the NVIDIA CUDA toolkit or the respective scientific development kit of the device. Here, in order to obtain a good trade-off between the optimization grade and the flexibility of the tool, we use a simple technique, based on the use of the C/C++ `nanosleep()` function (see `sys/time.h`). Specifically, to optimize the idle-wait to the application, we measure the execution time of the first kernel call. Since calling the `nanosleep()` function also triggers some overhead, we subtract this overhead plus some threshold. For subsequent kernel calls, we then set the host system to sleep for this time frame with the `nanosleep()`.

Figure 2 illustrates the power demand of different energy-saving techniques applied to the CG solving process of the linear system A318 during a period of 12.5 secs, chipset-measurement only. Here, using DVFS to lower the operating frequency of the CPU cores from 2.0 GHz to 800 MHz (line labeled as “CG+DVFS”) does not affect the iteration time, as most of the computations are performed on the GPU. The results show that with DVFS alone the improvement is small; the `nanosleep()` function yields a certain drop of the power consumption (line “CG+idle-wait”); and the combination of both techniques improves the performance further (line “CG+DVFS+idle-wait”).

A. Power-Consumption Tests

In the next tests we employ three GPU-implementations of the solver:

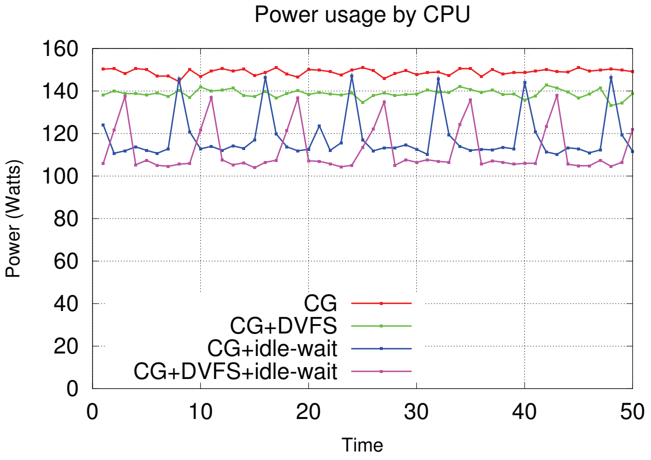

Figure 2: Power consumption of different energy-saving techniques applied to the CG-solver, chipset measurement.

Table V: Energy Consumption of different implementations of the CG solver, chipset + GPU.

matrix	energy consumption [Wh]			improvement [%]	
	(i)	(ii)	(iii)	(i)→(ii)	(i)→(iii)
A318	14.84	14.12	12.18	5.1	21.8
APACHE2	1.98	1.99	1.82	-0.5	8.8
AUDIKW_1	no convergence			-	-
BONES10	no convergence			-	-
ECOLOGY2	2.30	2.27	2.09	-1.3	10.0
G3_CIRCUIT	11.48	11.11	10.10	3.3	13.7
LDOOR	no convergence			-	-
N24K	26.43	25.42	21.17	3.97	24.8

(i) The first implementation is straight-forward, without DVFS or any other power-saving technique. The CPU of the host system runs at full speed (2.0 GHz) during the complete solving process.

(ii) Using DVFS, we scale down the frequency of the host system to 800 MHz during the operation of the GPU-accelerated solver. This has the drawback of slow CPU computations for this part, but since these computations are minor, this choice seems reasonable.

(iii) Additionally to DVFS we set the host system to sleep for the time the GPU performs the sparse matrix-vector multiplication.

Tables V and VI collect the results obtained with the (i)–(iii) solver implementations, applied to all linear systems described in Table I, using, respectively, a plain CG solver and a Jacobi-preconditioned one. We measure the total energy consumption by adding the energy use of chipset and GPU. The last two columns in both tables reflect the improvement in power consumption that can be obtained by using DVFS and the combination of DVFS and idle-wait.

Table VI: Energy Consumption of different implementations of the PCG solver, chipset + GPU.

matrix	energy consumption [Wh]			improvement [%]	
	(i)	(ii)	(iii)	(i)→(ii)	(i)→(iii)
A318	14.84	14.12	12.18	5.1	21.8
APACHE2	1.75	1.76	1.64	-0.6	6.7
AUDIKW_1	47.98	45.61	38.15	5.2	25.8
BONES10	157.32	150.16	125.78	4.8	25.1
ECOLOGY2	2.51	2.45	2.29	2.4	9.6
G3_CIRCUIT	2.71	2.63	2.38	3.0	13.9
LDOOR	43.22	41.18	34.79	5.0	24.2
N24K	34.62	32.97	27.64	5.0	25.3

While for some problems only the preconditioned variant of the CG solver converges, applying a preconditioner is not always reasonable. There exist problems where the plain implementation is superior, but for most systems, adding a preconditioner improves the performance. At this point, it is worth mentioning that we only evaluated example A318 without the preconditioner. The reason for this is twofold. First, applying a Jacobi-preconditioner to this system does not improve the convergence behaviour. Second, the additional memory required for the preconditioner poses a problem for our system equipped with only 3 GB of GPU-memory. Overall, these results demonstrate that DVFS alone renders only small improvement to the power consumption, and in some cases it even triggers a higher energy cost. This happens when the time and related energy overhead triggered by rescaling the CPU frequency exceeds the power savings.

Applying the combination of DVFS and the idle-wait, we observe an improvement in the power consumption for all test-cases. Still we appreciate large differences in the scale of saving. While for some systems the energy saving is in the range of 1/4, it only sums up to a few percent for some others. There exist two main factors determining the energy savings:

1) The time of the matrix-vector operations conducted by the GPU dictates whether it is reasonable to sleep the host system for a considerable time-frame. If the overhead due to calling the `nanosleep()` function exceeds the execution time of the GPU kernel, no improvement can be obtained. Additionally, the sleep function takes some time to scale down the energy consumption of the processor. For the used system, the average time from highest/lowest energy consumption to lowest/highest energy consumption approximates 50/74 microseconds.
2) The sparsity pattern of a matrix determines the ratio between the cost of a sparse matrix-vector product and a vector operation (BLAS-1). Since for the dimensions of the systems that were evaluated the execution of the BLAS-1 kernels on the GPU usually took less time than the overhead for calling the `nanosleep()` function, we set the host system to sleep only for the sparse matrix-vector product. Hence, the improvement

comes from the GPU kernels conducting the matrix-vector operations. If these account for a large part of the overall computation time, we can expect notable energy savings.

The first point leads to the conclusion that the usage of DVFS and idle-wait is only reasonable for systems with expensive matrix-vector products. For other problems, either the computational cost of solving the linear system is low or the condition number is very high, leading to a large number of iterations. In both cases, using the CPU with less computing cores but working at a higher frequency than the GPU leads to the acceleration of the solver. Hence, for the general case where the GPU-implementation of a solver is superior, the dimension of the system allows an efficient use of DVFS and idle-wait.

The second point suggests that larger benefits could be expected from the application of these techniques to the solution of dense linear systems via iterative methods. Since the linear problems occurring in scientific computing are often sparse, and we want to maintain the focus on scientifically relevant problems, we refrain from an analysis of dense problems at this point.

VI. Conclusion

In this paper, we presented and evaluated power-saving techniques for the solution of sparse linear solvers via iterative methods on hybrid hardware platforms. The parallelism of the sparse matrix-vector product enables the efficient usage of GPUs as coprocessor, able to conduct this operation with considerably high performance, both in execution time and energy consumption. Dynamic Voltage and Frequency Scaling can be used to rescale the processor frequency for the time the device is performing computations, and potentially lower the energy consumption. However, since the busy-wait of the host system during the kernel calls sets it into a state where it still consumes about 80% of full-demand power, the effects are close-to-negligible. In response to this, we have shown that using idle-wait for the time of kernel calls gives considerable improvement to the overall energy consumption of an application.

Finally, we have provided a set of experimental results where the power-aware solver framework is applied to different test matrices that are either derived from finite difference discretizations or available in the University of Florida matrix collection. Analyzing the power consumption of the solvers we realized that the use of GPU-accelerated HPC-systems combined with power-saving techniques leads to more reduced energy consumption of all test problems without impacting the performance.

The challenge for the hardware manufacturers is now to enable hardware components like GPU to perform DVFS similar to a CPU, and to develop devices that can be set to sleep or shut down for the time they are not used.

To face the energy problems that come along when approaching the Exascale computing era, the software developers have to continuously optimize their applications to the target hardware systems. This includes not only applying power-aware implementations, but also redesigning the algorithms to enable a higher grade of parallelization, and a better distribution of the workload on the various components forming the HPC-systems.

Acknowledgments

The authors thank M. Dolz, G. Fabregat and V. Roca, for their technical support with the energy measurement framework.

The authors from the Universidad Jaume I were supported by project CICYT TIN2008-06570-C04-01 and FEDER.

References

[1] F. Lampe, *Green-IT, Virtualisierung und Thin Clients : Mit neuen IT-Technologien Energieeffizienz erreichen, die Umwelt schonen und Kosten sparen.* Vieweg + Teubner, 2010.

[2] P. Kogge *et al*, “ExaScale computing study: Technology challenges in achieving ExaScale systems,” 2008.

[3] J. Dongarra *et al*, “The international ExaScale software project roadmap,” *Int. J. of High Performance Computing & Applications*, vol. 25, no. 1, 2011.

[4] J. I. Aliaga, M. Bollhöfer, A. F. Martín, and E. S. Quintana-Ortí, “Exploiting thread-level parallelism in the iterative solution of sparse linear systems,” *Parallel Computing*, vol. 37, pp. 183–202, 2011.

[5] Y. Saad, *Iterative Methods for Sparse Linear Systems*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2003.

[6] H. Anzt, B. Rocker, and V. Heuveline, “Energy efficiency of mixed precision iterative refinement methods using hybrid hardware platforms,” *Computer Science - Research and Development*, vol. 25, Issue 3, pp. 141–149, 2010.

[7] H. Anzt, M. Castillo, J. C. Fernández, V. Heuveline, R. Mayo, E. S. Quintana-Ortí, and B. Rocker, “Power consumption of mixed precision in the iterative solution of sparse linear systems,” EMCL, Karlsruhe Institute of Technology, Tech. Rep. 11-01, 2011, to appear in HPPAC 2011.

[8] 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.*, vol. 18, pp. 835–848, June 2007.

[9] *NVIDIA CUDA CUBLAS Library Programming Guide*, 1st ed., NVIDIA Corporation, June 2007.

[10] N. Bell and M. Garland, “Implementing sparse matrix-vector multiplication on throughput-oriented processors,” in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, ser. SC ’09. New York, NY, USA: ACM, 2009, pp. 18:1–18:11.

[11] PCI-SIG, “PCIe x16 Graphics 150W-ATX Spec 1.0,” 2004.

Preprint Series of the Engineering Mathematics and Computing Lab

recent issues

No. 2011-04 Vincent Heuveline, Michael Schick: A local time–dependent Generalized Polynomial Chaos method for Stochastic Dynamical Systems

No. 2011-03 Vincent Heuveline, Michael Schick: Towards a hybrid numerical method using Generalized Polynomial Chaos for Stochastic Differential Equations

No. 2011-02 Panagiotis Adamidis, Vincent Heuveline, Florian Wilhelm: A High-Efficient Scalable Solver for the Global Ocean/Sea-Ice Model MPIOM

No. 2011-01 Hartwig Anzt, Maribel Castillo, Juan C. Fernández, Vincent Heuveline, Rafael Mayo, Enrique S. Quintana-Ortí, Björn Rocker: Power Consumption of Mixed Precision in the Iterative Solution of Sparse Linear Systems

No. 2010-07 Werner Augustin, Vincent Heuveline, Jan-Philipp Weiss: Convey HC-1 Hybrid Core Computer – The Potential of FPGAs in Numerical Simulation

No. 2010-06 Hartwig Anzt, Werner Augustin, Martin Baumann, Hendryk Bockelmann, Thomas Gengenbach, Tobias Hahn, Vincent Heuveline, Eva Ketelaer, Dimitar Lukarski, Andrea Otzen, Sebastian Ritterbusch, Björn Rocker, Staffan Ronnås, Michael Schick, Chandramowli Subramanian, Jan-Philipp Weiss, Florian Wilhelm: HiFlow^3 – A Flexible and Hardware-Aware Parallel Finite Element Package

No. 2010-05 Martin Baumann, Vincent Heuveline: Evaluation of Different Strategies for Goal Oriented Adaptivity in CFD – Part I: The Stationary Case

No. 2010-04 Hartwig Anzt, Tobias Hahn, Vincent Heuveline, Björn Rocker: GPU Accelerated Scientific Computing: Evaluation of the NVIDIA Fermi Architecture; Elementary Kernels and Linear Solvers

No. 2010-03 Hartwig Anzt, Vincent Heuveline, Björn Rocker: Energy Efficiency of Mixed Precision Iterative Refinement Methods using Hybrid Hardware Platforms: An Evaluation of different Solver and Hardware Configurations

No. 2010-02 Hartwig Anzt, Vincent Heuveline, Björn Rocker: Mixed Precision Error Correction Methods for Linear Systems: Convergence Analysis based on Krylov Subspace Methods

No. 2010-01 Hartwig Anzt, Vincent Heuveline, Björn Rocker: An Error Correction Solver for Linear Systems: Evaluation of Mixed Precision Implementations

No. 2009-02 Rainer Buchty, Vincent Heuveline, Wolfgang Karl, Jan-Philipp Weiß: A Survey on Hardware-aware and Heterogeneous Computing on Multicore Processors and Accelerators

No. 2009-01 Vincent Heuveline, Björn Rocker, Staffan Ronnas: Numerical Simulation on the SiCortex Supercomputer Platform: a Preliminary Evaluation

The responsibility for the contents of the working papers rests with the authors, not the Institute. Since working papers are of a preliminary nature, it may be useful to contact the authors of a particular working paper about results or caveats before referring to, or quoting, a paper. Any comments on working papers should be sent directly to the authors.