# Optimization of Power Consumption in the Iterative Solution of Sparse Linear Systems on Graphics Processors

H. Anzt, M. Castillo, J. C. Fernández, V. Heuveline,
F. D. Igual, R. Mayo, E. S. Quintana-Ortí

No. 2011-06

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

Preprint Series of the Engineering Mathematics and Computing Lab (EMCL)
ISSN 2191–0693
No. 2011-06

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

**EMCL Preprint Series 2011-06**

# Optimization of Power Consumption in the Iterative Solution of Sparse Linear Systems on Graphics Processors

**Hartwig Anzt**[†] · **Maribel Castillo**[‡] · **Juan C. Fernández**[‡] · **Vincent Heuveline**[†] · **Francisco D. Igual**[‡] · **Rafael Mayo**[‡] · **Enrique S. Quintana-Ortí**[‡]

**Abstract** In this paper, we analyze the power consumption of different GPU-accelerated iterative solver implementations enhanced with energy-saving techniques. Specifically, while conducting kernel calls on the graphics accelerator, we manually set the host system to a power-efficient idle-wait status so as to leverage dynamic voltage and frequency control. While the usage of iterative refinement combined with mixed precision arithmetic often improves the execution time of an iterative solver on a graphics processor, this may not necessarily be true for the power consumption as well. To analyze the trade-off between computation time and power consumption we compare a plain GMRES solver and its preconditioned variant to the mixed-precision iterative refinement implementations based on the respective solvers. Benchmark experiments conclusively reveal how the usage of idle-wait during GPU-kernel calls effectively leverages the power-tools provided by hardware, and improves the energy performance of the algorithm.

**Keywords** Sparse Linear Systems · Iterative Solvers · GMRES · Mixed Precision Iterative Refinement · Power-Aware Algorithms · Graphics processors (GPUs) · Idle-Wait

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

‡
Depto. de Ingeniería y Ciencia de Computadores
Universidad Jaume I
12.071 - Castellón, Spain
{castillo,jfernand,figual,mayo,quintana}@icc.uji.es

## 1 Introduction

Solving sparse linear systems is often the most resource-demanding stage when performing scientific simulations. This is true for the computation time as well as for the energy consumption. While the energy prices are rising due to the ever-increasing demand, and the economical impact of the human carbon footprint becomes more apparent, the implementation of energy-saving techniques in modern high performance computing (HPC) becomes indispensable [1]: The power consumption of current HPC and data-processing centers often equals the energy demand of a provincial town and, in the coming Exascale computing Era, the energy factor will play a crucial role [2,3]. Not only will the financial facet of the running energy costs dominate the monetary frame, but also will the available infrastructure determine the feasibility of a project. In other words, it is not only a question of whether the energy cost can be paid, but also whether the energy network is able to bear additional consumers of this scale. For these reasons, an increasing number of scientists of related fields are working on improving the energy efficiency of future HPC. From the technical point of view, hardware developers aim at lowering the energy consumption by, e.g., designing hybrid hardware platforms equipped with graphics processors (GPUs) that can conduct operations with higher efficiency, or introducing techniques like Dynamic Voltage and Frequency Control (DVFS) able to scale down the CPU frequency/voltage and, therewith, the correlated power demand. But the hardware-driven approach to energy-efficient computing is not sufficient.

Despite the fact that many computing centers are nowadays equipped with hardware featuring energy-saving techniques, most scientific applications are still oblivious to power consumption. Therefore, also from the software developers side, the algorithms and simulation structures have to be redesigned, to exploit the possibilities of these new technologies.

The simulation process of many physical and economical processes can often be broken down into the solution process of large sparse linear systems, counting up for a high percentage of the overall resource demand. This is the case, e.g., of many applications that require the solution of partial differential equations (PDE) modeling physical, chemical or economical processes. While direct solvers can deal with small to medium-sized sparse linear systems, large-scale systems frequently require the use of low-cost iterative solvers based on Krylov subspace-based methods [4]. Here, we focus on how these iterative solvers can be improved with respect to power consumption. This demands not only for a thorough analysis of the energy consumption of the algorithms, but also the redesign of the solvers to efficiently exploit the energy-saving techniques offered by the hardware components.

Using hybrid hardware platforms, in particular those equipped with general-purpose multi-core processors and GPUs, often requires a nontrivial adaption of the methods to the heterogeneous computing resources. While the high number of computing cores in GPUs allows the parallel execution of certain tasks and may trigger significant performance gains to data-parallel applications, the architecture often asks for non-negligible modifications to the underlying methods. The limited memory of GPUs and the significantly higher performance when operating in low precision suggests the use of a mixed-precision iterative refinement (MPIR) method with an error correction solver on the accelerator. While applying this variant in general renders a better runtime performance of the solver, this may not necessarily be true for the energy consumption. The reason is that the energy-saving techniques provided by the system and the hardware platform frequently pose some restrictions.

An initial analysis of the computation time and the energy consumption of a plain GMRES solver –see section 3.1, [4,5]– and a MPIR variant was presented in [6]. The results revealed the superiority of solver implementations using hybrid hardware platforms, where the high degree of hardware concurrency of the accelerator was exploited to compute the expensive matrix-vector and vector-vector operations. In this paper, we extend those results showing how the solver can be tuned with power-saving techniques so as to improve the energy efficiency.

Specifically, the main contribution of this paper is a practical demonstration of how energy-saving techniques can be applied with different efficiency to a variety of solver implementations. This reveals that power demand and computation time of scientific applications do not necessarily go hand-in-hand. We also show that, in order to lower the energy consumption of iterative solvers for linear systems, it is not sufficient to optimize with respect to the computing time, but it is also necessary to consider all parameters concerning the linear system, the energy-saving techniques, and the hardware platform. To achieve this goal, we split the paper into the following parts:

1. Following the introductory part, we describe the hardware setup. First, we introduce the hybrid CPU-GPU hardware platform we used to conduct the experiments. Additionally, we provide a description of the power-measurement setup employed to conduct detailed power monitoring.
2. In the next section, we describe the target mathematical problem and introduce the benchmark linear systems. We also provide a brief overview about iterative solvers and review how to use different floating-point formats in an iterative refinement method.
3. In a detailed analysis we then compare the power consumption and the computation time of the various GPU-accelerated GMRES solver implementations. In a first step, we analyze the impact of the Krylov subspace size of the GMRES solver on the runtime and the power consumption. After choosing an adequate restart parameter, we then apply a Jacobi preconditioner, improving the runtime as well as the energy performance the algorithm. Finally, we embed the GMRES solver as well as its preconditioned variant into a MPIR solver framework. For some configurations this gives an additional improvement in the computation time and energy consumption, but in all cases, the gain for the latter one is smaller. Using DVFS and idle-wait is known to decrease the overall power consumption of linear solvers [6–8] We show how this technique works by conducting a detailed energy consumption of chipset and GPU for the time of a kernel call. This shows that optimizing numerical algorithms with respect to energy consumption not only demands the redesign of the code, but also the efficient leverage of the power tools provided by the hardware platforms.
4. In the last section, we offer a number of conclusions and a brief overview about open problems that have to be addressed in future, to enhance the energy efficiency of linear solvers further. This includes hardware components which can be turned on/off depending on the demand.

## 2 Target Setup

### 2.1 Hardware platform and linear algebra libraries

The experiments in this paper were conducted on a system equipped with an AMD Opteron 6128 processor (eight cores) at 2.0 GHz and 24 GB of RAM. The system was connected via PCIe (16x) to an NVIDIA Tesla C1060 board (240 processor cores) with 4 GB of GDDR3 memory. We invoked the tuned implementations from Intel MKL (v11.1) to perform all Level-1 BLAS operations (dot products, "axpys", norm computation, etc.) on the AMD processor. The compilation of the CPU code was done using the GNU `gcc` compiler (v4.4.3) with the flag `-O3`.

On the GPU, the Level-1 BLAS operations were performed using the corresponding CUBLAS routines from [9] (v3.0). NVIDIA `nvcc` compiler (v3.2) with an up-to-date CUDA driver (v3.2) was employed for the accelerator codes. A specific kernel for the computation of the sparse matrix-vector multiplication on the GPU was implemented following the ideas in [10].

### 2.2 Measurement setup

Power was measured using an ASIC built as a number of resistors connected in series with the power source, with a sampling frequency of 25 Hz. This internal power meter obtained the global energy consumption of the chipset, processor and GPU from the lines connecting directly the power supply unit with these components. Samples were collected in a separate system, to avoid interfering the performance of the tests. Figure 1 illustrates the connection of the energy measurement ASIC to the system lines.

## 3 Mathematical Background

### 3.1 GMRES solvers

Large-scale sparse linear systems, $Ax = b$, can usually be solved more efficiently by applying iterative methods instead of direct solvers [4]. Especially the Krylov subspace-based iterative methods have demonstrated remarkable performance for many linear problems, becoming the method of choice for many applications.

GMRES is a projection method that operates on Krylov subspaces generated by the Arnoldi algorithm. It was designed for the solution of linear systems where the coefficient matrix $A$ is neither necessarily symmetric nor positive definite [4,5]. Indeed, GMRES also works for nonsymmetric semi-positive definite systems, and is especially appropriate for large-scale sparse matrices.

In exact arithmetic, after $n$ steps, GMRES computes the exact result to a linear system of dimension $n$. Therefore, GMRES is in fact a direct method, like other Krylov subspace solvers, that computes the analytically exact solution in $n$ steps. In practice, for large linear systems, difficulties appear in the method due to a linear increase in the computational and storage costs, and to the loss of orthogonality of the Krylov subspaces triggered by rounding errors. Because choosing a number of iterations much smaller than $n$ often yields a good approximation of the result, one usually employs GMRES as an iterative solver, with a stopping criterion depending on the residual norm.

In the plain GMRES algorithm, the whole Krylov basis has to be stored until the residual is below a certain threshold. Therefore, for large linear systems, the memory and computational costs of this method become prohibitive. To avoid this, in the Restart-GMRES variant, $m$-GMRES, the Krylov subspace and the approximation are not computed until the residual has reached the demanded threshold, but restarted after a certain number of steps ($m$).

The advantages of the restarted algorithm are that the orthogonality of the computed Krylov subspaces is preserved to a higher degree due to the restart of the Krylov-subspace generator and the computational and memory costs are bounded, as the linear problem stays at a lower dimension, and only $m$ Krylov subspace vectors have to be stored; see Algorithm 1.

### 3.2 Mixed precision iterative refinement

A plain implementation of a Krylov subspace method is usually the best option for a CPU-based system. On the other hand, the superior low precision performance of GPUs suggests the adoption of an iterative refinement method, where the error correction equation is solved in a lower floating point precision format [11–13]. This also reduces the memory demand, which is a substantial advantage, since GPUs are usually equipped with small on-device memory.

Newton's method can be applied to the function $f(x) = b - Ax$ with $\nabla f(x) = A$. By defining the residual $r_i := b - Ax_i$, one obtains

$$\begin{aligned} x_{i+1} &= x_i - (\nabla f(x_i))^{-1} f(x_i) \\ &= x_i + A^{-1}(b - Ax_i) \\ &= x_i + A^{-1} r_i. \end{aligned}$$

Denoting the solution update with $c_i := A^{-1} r_i$ and using an initial guess $x_0$ as starting value, an iterative

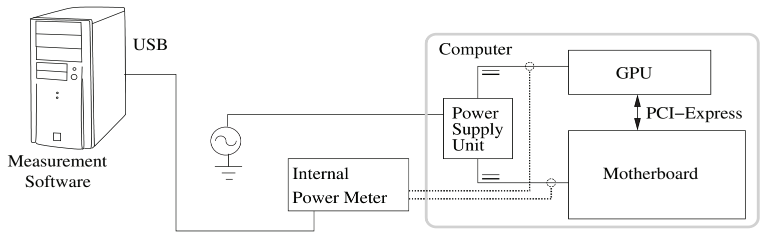

**Fig. 1** Hardware platform and sampling points.

```
1:  for ($l = 1, l++$) do
2:      Compute $r_0 = b - Ax_0$, $d_0 = \beta_0 = \| r_0 \|_2$, $v_1 = \frac{r_0}{\beta_0}$
3:      for ($j = 1, j \leq m, j++$) do
4:              % Iteration process of GMRES
5:          Compute $w_j = Av_j$
6:          for ($i = 1, i \leq j, i++$) do
7:                  % Arnoldi's method
8:              $h_{i,j} = \langle w_j, v_i \rangle$
9:              $w_j = w_j - h_{ij} v_i$
10:         end for
11:         $\omega = \| w_j \|_2$
12:         for ($i = 1, i < j, i++$) do
13:                 % Apply former rotation to $h_k$
14:             $\tilde{h} = c_i h_{i,j} + s_i h_{i+1,j}$
15:             $h_{i+1,j} = -s_i h_{i,j} + c_i h_{i+1,j}$
16:             $h_{i,j} = \tilde{h}$
17:         end for
18:         if ($\omega \leq |h_{j,j}|$) then
19:                 % Compute new rotation
20:             $t_j = \frac{\omega}{|h_{j,j}|}$
21:             $c_j = \frac{h_{j,j}}{|h_{j,j}|\sqrt{1+t_j^2}}$
22:             $s_j = \frac{t_j}{\sqrt{1+t_j^2}}$
23:         else
24:             $t_j = \frac{h_{j,j}}{\omega}$
25:             $c_j = \frac{t_j}{\sqrt{1+t_j^2}}$
26:             $s_j = \frac{1}{\sqrt{1+t_j^2}}$
27:         end if
28:         $h_{j,j} = c_j h_{j,j} + s_j \omega$  % Apply rotation to rest of $\hat{H}$
29:         $d_j = -s_j d_{j-1}$  %Apply the rotation to the RHS
30:         $d_{j-1} = c_j d_{j-1}$
31:     end for
32:     solve $H_l y = d$ with the Gauss-Algorithm
33:     Define the matrix $V_l = [v_1 \dots v_l]$
34:     Compute the approximation $x_l = x_0 + V_l y$
35:     if ($|d_l| \leq \varepsilon$) then
36:         stop
37:     end if
38: end for
```

Algorithm 1: GMRES-($m$) solver.

procedure can be defined as in Algorithm 2. The error correction method makes no demands on the inner linear solver, so that any method can be chosen for this purpose. In particular, in this paper we will use GMRES to solve the general sparse linear systems associated with the CFD application.

```
1: Choose initial guess: $x_0$
2: Compute initial residual: $r_0 = b - Ax_0$
3: $i = 0$
4: repeat
5:     Solve error correction equation $Ac_i = r_i$ for $c_i$
6:     Update solution: $x_{i+1} = x_i + c_i$
7:     Compute new residual: $r_{i+1} = b - Ax_{i+1}$
8:     $i = i + 1$
9: until ($\| r_i \|_2 \leq \varepsilon \| r_0 \|$)
```

Algorithm 2: Error correction method.

## 3.3 Solver parameters

We use a plain GMRES algorithm in restart-variant as reference solver while the MPIR method uses Restart-GMRES solver in low precision as error correction solver. For the different tests, we set the restart parameter to a variety of values. Furthermore, in most of our experiments we fix the relative residual stopping criterion for the final solution approximation to $10^{-10}$. Due to the iterative residual computation in the case of the plain GMRES solvers, the MPIR GMRES variants usually yield a more accurate approximation, since they compute the residual error explicitly. However, as the difference is in general small, the results can be compared.

In the first tests, we vary the relative residual stopping criterion $\varepsilon_{\text{INNER}}$ of the error correction solver inside the MPIR solver. In all other tests, when analyzing the energy consumption of the individual parts of the solver and the comparison to the plain solver implementation, we set the inner stopping criterion to $10^{-1}$, as this choice is optimal for our application from the points of view of execution time and energy consumption.

## 3.4 Benchmark example

We evaluate the performance of the iterative GMRES method to solve the linear system $Ax = b$, where $A$ is derived from a finite difference discretization of a two-dimensional fluid flowing through a Venturi Nozzle, and $b$ is a vector with all entries equal 1. Three linear systems arising from the same application of different

| Example | $n$ | $nnz$ |
|---|---|---|
| CFD1 | 395,009 | 3,544,321 |
| CFD2 | 634,453 | 5,700,633 |
| CFD3 | 1,019,967 | 9,182,401 |

**Table 1** Dimensions of the benchmark examples.

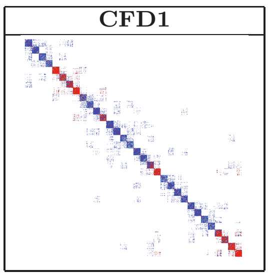

**Fig. 2** Sparsity plot of the CFD1 matrix.

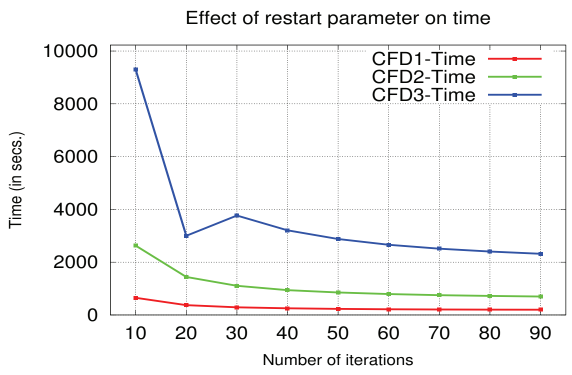

**Fig. 3** Computation time (in secs.) for different values of the restart parameter $m$.

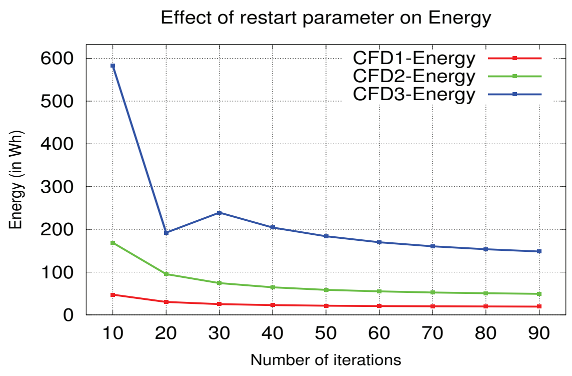

**Fig. 4** Energy consumption (in Wh) for different values of the restart parameter $m$.

granularity were evaluated, CFD1, CFD2 and CFD3, with the dimension/number of nonzero entries ($n/nnz$) given in Table 1. Figure 2 illustrates the sparsity pattern of CFD1. The structure of the other two examples is analogous.

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 $\varepsilon = 10^{-10}\sqrt{n}$.

## 4 Numerical Experiments

### 4.1 Restart parameter tuning

In this first test, we analyze the influence of the restart parameter $m$ (number of iterations between two restarts) on the overall computation time and the energy consumption of the plain GMRES solver for examples CFD1, CFD2 and CFD3.

The results in Figures 3 and 4 reveal that a larger restart parameter improves the solver performance – at least for the problems we analyze. Still, we observe a limitation, and the improvements become negligible for choices larger than 30 for CFD1/CFD2 and 50 for CFD3. Note, however, that a higher value of $m$ increases the dimension of the Krylov subspace putting more pressure on the memory demand, which may become a problem for many hardware platforms, in particular on GPUs, where the memory is scarce. Research analysis has shown, that restart parameters between 10 and 40 usually trigger acceptable performance for many problems [4]. We furthermore observe an almost linear dependency between computation time and energy consumption. This can be expected as long as no energy-saving tools provided by hardware are applied.

In the following experiments we set the restart parameter $m$ to 30 and, for convenience, refer to solver 30-GMRES as (plain) GMRES.

### 4.2 Solver variants

The next experiment enhances the plain GMRES solver implementation with the addition of a Jacobi preconditioner. We will denote this new solver as P-GMRES.

Table 2 collects the computation time and energy consumption for all three benchmark examples. The results there show that, for all considered systems, the improvements of adding a preconditioner are significant. The speedup for the computation time ranges from 1.5× for CFD1 to 2.7× for CFD3. The fact that the energy improvements are in the same range shows the almost linear dependency between energy consumption and computation time for this solver reconfiguration.

**Table 2** Execution time (in secs.) and energy consumption (in Wh) of GMRES and its preconditioned variant P-GMRES for CFD1, CFD2 and CFD3.

| Ex. | Solver | Time | Energy |
|---|---|---|---|
| CFD1 | GMRES | 292.23 | 16.89 |
| | P-GMRES | 194.26 | 11.30 |
| | gain (%) | 33.5 | 33.1 |
| CFD2 | GMRES | 1104.02 | 66.66 |
| | P-GMRES | 601.93 | 36.19 |
| | gain (%) | 45.5 | 45.7 |
| CFD3 | GMRES | 3377.03 | 231.23 |
| | P-GMRES | 1391.16 | 86.12 |
| | gain (%) | 58.5 | 62.8 |

**Table 3** Execution time (in secs.) and energy consumption (in Wh) of the plain GMRES, its preconditioned variant P-GMRES, and their corresponding versions with MPIR for CFD1, CFD2 and CFD3.

| Ex. | Solver | Time | Energy | | |
|---|---|---|---|---|---|
| | | | Chipset | GPU | Total |
| CFD1 | GMRES | 292.23 | 12.00 | 4.89 | 16.89 |
| | MPIR GMRES | 154.30 | 6.34 | 2.71 | 9.05 |
| | gain (%) | 47.2 | 47.2 | 44.6 | 46.4 |
| | P-GMRES | 194.26 | 8.02 | 3.28 | 11.30 |
| | MPIR P-GMRES | 122.43 | 5.05 | 2.30 | 7.35 |
| | gain (%) | 37.0 | 37.0 | 29.9 | 35.0 |
| CFD2 | GMRES | 1104.02 | 46.53 | 20.13 | 66.66 |
| | MPIR GMRES | 640.84 | 26.89 | 11.91 | 38.80 |
| | gain (%) | 42.0 | 42.2 | 40.8 | 41.8 |
| | P-GMRES | 601.93 | 25.19 | 11.00 | 36.19 |
| | MPIR P-GMRES | 416.42 | 17.47 | 8.46 | 25.93 |
| | gain (%) | 30.8 | 30.6 | 23.1 | 28.4 |
| CFD3 | GMRES | 3777.03 | 160.98 | 70.25 | 231.23 |
| | MPIR GMRES | 2459.84 | 104.28 | 47.74 | 152.02 |
| | gain (%) | 34.9 | 35.2 | 32.0 | 34.2 |
| | P-GMRES | 1391.16 | 59.38 | 26.74 | 86.12 |
| | MPIR P-GMRES | 1520.79 | 64.47 | 28.15 | 92.62 |
| | gain (%) | - 9.3 | -8.6 | -5.3 | -7.5 |

In a second step, we embed the GMRES solver and its preconditioned variant into a MPIR solver framework, as this is expected to improve the solver performance for many linear systems [11]. In Table 3 we report the performance gains from this new variant. Since all double precision operations, like the solution update and the residual computation, are handled by the CPU in this approach, we choose to perform a detailed power analysis separate for the chipset and the GPU consumptions.

We observe that the question of whether applying the MPIR framework pays off really depends on the linear system. For CFD1, embedding the solver into it renders superior performance both for the plain GMRES as well as for its preconditioned variant, though for the latter, the performance gain is considerably smaller. Also for CFD2, we benefit from the multi-precision approach, but the improvement for the preconditioned variant is again smaller. For CFD3, MPIR basically yields no gain for the plain GMRES solver, while for the preconditioned variant, it even increases the computation time as well as the energy consumption.

An interesting insight from this study is that, on systems where adding the preconditioner to the solver gave a large factor of improvement, using the MPIR framework produced no gain or even decreased performance. On the other hand, the results illustrated a notable performance increase when MPIR was added to the solver on CFD1, but in this case including the preconditioner made little difference. This confirms that choosing the iterative method to solve a linear system requires exact knowledge about the matrix characteristics.

Overall, we observe that the differences in energy consumption are always smaller than those obtained for the computation time. While the energy consumption of chipset is almost linear to the computation time, the power improvement of the GPU is smaller when switching to the MPIR approach. The main reason for this is that in the MPIR solver a smaller percentage of the overall computational effort is handled by the GPU, since all double precision operations are conducted by the CPU.

## 4.3 Energy-saving techniques

To lower the energy consumption of the GPU-accelerated implementations, DVFS can be applied to lower the frequency of the CPUs when they are not used (e.g., because computations are being performed on the GPU), yielding a more reduced power consumption. Additionally, the solvers may be equipped with the "idle-wait" technique [8] that sets the host system into an energetically very efficient sleeping mode for the time of the kernel calls to the GPU. Applying these techniques usually leads to a considerably decrease in power consumption, without impacting the runtime [8].

In Table 4 we conclusively report how idle-wait can improve the energy performance of all solver implementations. While applying idle-wait improves the energy performance of all solver variants for all targeted linear systems, only negligible increases of the execution time were observed. This reveals how power-saving tools provided by the system can be applied without conducting fundamental changes in the code. Still, the savings for our implementations are considerably smaller than for other solver algorithms (in particular, CG; see [8]). The reason is that the matrix-vector multiplications conducted by the GPU and enhanced with idle-wait only count up to a small percentage of the overall computational cost of our solvers. To enable a more efficient usage of idle-wait, the algorithms have to be redesigned, allowing longer kernel calls on the graphics.

**Table 4** Energy consumption (in Wh) of the plain GMRES, its preconditioned variant P-GMRES, and their corresponding versions with MPIR, with and without idle-wait (columns "Idle-wait" vs. "Plain", respectively), for CFD1, CFD2 and CFD3.

| Ex. | Solver | Energy (Wh) | | |
|---|---|---|---|---|
| | | Plain | Idle-wait | gain (%) |
| CFD1 | GMRES | 16.88 | 15.65 | 7.31 |
| | MPIR GMRES | 12.38 | 11.62 | 8.75 |
| | P-GMRES | 9.01 | 8.22 | 6.09 |
| | MPIR P-GMRES | 7.35 | 6.61 | 9.99 |
| CFD2 | GMRES | 66.66 | 62.03 | 6.95 |
| | MPIR GMRES | 36.19 | 33.83 | 10.22 |
| | P-GMRES | 38.79 | 34.83 | 6.51 |
| | MPIR P-GMRES | 25.94 | 23.39 | 9.80 |
| CFD3 | GMRES | 231.23 | 217.48 | 5.95 |
| | MPIR GMRES | 86.12 | 80.88 | 8.70 |
| | P-GMRES | 152.02 | 138.80 | 6.08 |
| | MPIR P-GMRES | 92.62 | 84.51 | 8.75 |

## 5 Conclusion

In this paper we have presented a energy performance analysis of different variants of a GMRES solver applied to a sparse linear equation system arising in a 2-D two-dimensional fluid flow application. In a first step, we optimized the restart parameter with respect to runtime and power demand of the plain GMRES solver. We then analyzed the energy consumption of different solver variants, with preconditioning and embedding the solver in a MPIR framework. The results revealed, that the choice of the optimal solver depends on the properties of the specific system. Thus, while adding a preconditioner usually improves the runtime as well as the energy performance, the MPIR framework pays off only for some cases. Applying the power-saving technique "idle-wait", we were able to reduce the overall power consumption for all solver implementations and test cases, roughly between 6 and 10%. This shows that optimizing numerical algorithms with respect to energy consumption demands both the redesign of the code and the efficient leverage of the power tools provided by the system. To conclude, only by combining the competences of hardware developers, software engineers and mathematicians, we will able to tackle the energy challenge of an Exascale Computing Era.

## 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.

The authors from the Karlsruhe Institute of Technology (KIT) thank the Landesstiftung Baden Württemberg for its financial support in the framework of the project "Multiscale Ensemble forecasting on HPC-systems", part of the the research program "High Performance Computing".

## 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. Y. Saad, *Iterative Methods for Sparse Linear Systems.* Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2003.
5. Y. Saad and M. H. Schultz, "Gmres: a generalized minimal residual algorithm for solving nonsymmetric linear systems," *SIAM J. Sci. Stat. Comput.*, vol. 7, pp. 856–869, July 1986. [Online]. Available: http://portal.acm.org/citation.cfm?id=14063.14074
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. H. Anzt, J. Aliaga, M. Castillo, J. C. Fernández, R. Mayo, and E. S. Quintana-Ortí, "Analysis and Optimization of Power Consumption in the Iterative Solution of Sparse Linear Systems on Multi-core and Many-core Platforms," EMCL Preprint Series, 2011. [Online]. Available: http://www.emcl.kit.edu/preprints/emcl-preprint-2011-05.pdf
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. J. Palma, M. Dayd, O. Marques, and J. Lopes, Eds., *An Error Correction Solver for Linear Systems: Evaluation of Mixed Precision Implementations*, ser. Lecture Notes in Computer Science, vol. 6449. Springer Berlin / Heidelberg, 2011.
12. J. J. Dongarra, I. S. Duff, D. C. Sorensen, and H. A. van der Vorst, *Numerical Linear Algebra for High-Performance Computers.* Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 1998.
13. D. Göddeke, R. Strzodka, and S. Turek, "Performance and accuracy of hardware–oriented native–, emulated– and mixed–precision solvers in FEM simulations," *Int. J. of Parallel, Emergent and Distributed Systems*, vol. 22, no. 4, pp. 221–256, 2007.