

# Exploring Energy Efficiency for GPU-Accelerated POWER Servers

Thorsten Hater<sup>1</sup>, Benedikt Anlauf<sup>2</sup>, Paul Baumeister<sup>1</sup>, Markus Bühler<sup>2</sup>,  
Jiri Kraus<sup>3</sup>, and Dirk Pleiter<sup>1</sup>(✉)

<sup>1</sup> Forschungszentrum Jülich, JSC, 52425 Jülich, Germany  
{t.hater,p.baumeister,d.pleiter}@fz-juelich.de

<sup>2</sup> IBM Deutschland Research and Development GmbH, 71032 Böblingen, Germany  
{anlaufb,buehler}@de.ibm.com

<sup>3</sup> NVIDIA GmbH, Würselen, Germany  
jkraus@nvidia.com

**Abstract.** Modern servers provide different features for managing the amount of energy that is needed to execute a given work-load. In this article we focus on a new generation of GPU-accelerated servers with POWER8 processors. For different scientific applications, which have in common that they have been written for massively-parallel computers, we measure energy-to-solution for different system configurations. By combining earlier developed performance models and a simple power model, we derive an energy model that can help to optimise for energy efficiency.

**Keywords:** POWER8 · GPU acceleration · Power measurements · Power modelling · Energy efficiency

## 1 Introduction

The power consumption of supercomputers has become a major limiting factor for further increases in performance. For particularly compute intensive work-loads like the high-performance Linpack benchmark, currently a power efficiency of 7 GFlop/s/W can be reached on (smaller) supercomputers [13].

With the goal of keeping the power consumption of an exascale system below 20 MW, an improvement in power efficiency of at least a factor seven is required. Achieving this goal will require combining multiple strategies.

Leveraging existing system features that impact performance and power consumption (and thus energy-to-solution) is one of them. We review some of these features available on GPU-accelerated servers based on POWER8 processors. This includes, e.g., the core clock frequencies of processor and accelerator. Higher frequencies typically lead to a reduction in time-to-solution and (due to the behaviour of CMOS technology) unavoidably to an increase in power consumption. In line with previous observations, we observe a reduction of energy-to-solution for increasing the clock speed, i.e. a “run to idle” clock tuning strategy to be beneficial. For the considered system this effect is particularly large

because of the high idle power consumption. The memory sub-system attached to the POWER processor comprising a larger number of memory buffer chips contributes significantly to that.

To identify the setting for which energy-to-solution becomes minimal performance and power models can be helpful. Performance models enable prediction of time-to-solution for different parts of a work-load, while the power model is needed to predict the power that is consumed on average while executing a particular part of the work-load. Combining both multiplicatively allows to predict energy-to-solution.

For this paper we consider three different applications that come from different research areas. They have in common that they are used to target research questions, for which significant HPC resources are required. Their performance characteristics, however, differ significantly. B-CALM is an application for simulating electro-magnetic fields in dispersive media, which is relevant for developing photonic products. From the area of materials research we selected the application KKRnano, which implements the Density Functional Theory (DFT) method in a particularly scalable way. Finally, we also consider the molecular dynamics package GROMACS.

With this article we make the following contributions:

1. An overview on opportunities for optimising energy-to-solution on GPU-accelerated POWER8 servers is given.
2. Results for energy-to-solution measurements for kernels of several relevant scientific applications and different system configurations are given.
3. A simple power model is derived and combined with available performance models to model energy-to-solution.

After providing background on the relevant technologies in Sect. 2 we discuss the considered applications in Sect. 3 and the options to tune for energy-efficiency in Sect. 4. In Sect. 5 we present results from power measurements for different configurations. Based on these results we derive power and energy models with empirically determined parameters in Sect. 6. Before presenting our conclusions in Sect. 8, we give an overview on related work in Sect. 7.

## 2 Technology Background

All results presented in this paper have been obtained on a single POWER8 8247-42L Server [8]. This server, which comprises 2 POWER8 Dual-Chip Modules (DCM), was the first to support acceleration by NVIDIA GPUs. The server considered here is equipped with 2 Tesla K40m cards.

Each of the POWER8 modules comprises 5 cores, i.e. there are 20 cores per node. Each core offers two sets of the following instruction pipelines: fixed point (FXU), floating-point (VSU), pure load (LU) and a load-store unit (LSU). Instructions are processed out-of-order to increase instruction level parallelism. The cores feature 8-way Simultaneous Multi-threading. The two VSU support

VSX (Vector-Scalar eXtensions) instructions, including two-way SIMD, double-precision fused multiply-add instructions. This results in a theoretical peak performance per node

$$B_{\text{fp}}^{(\text{CPU})} = N_{\text{core}} \cdot f_{\text{CPU}} \cdot 8 \text{ Flop} \leq 590 \text{ GFlop/s}, \quad (1)$$

where  $N_{\text{core}}$  denotes the number of cores (here  $N_{\text{core}} = 20$ ) and  $f_{\text{CPU}}$  is the core clock frequency.

Each POWER8 processor is connected to eight external memory buffer chips (Centaur chips). Each of these 16 chips is connected with one link to one of the POWER8 processors, which has a bandwidth of 16 GByte/s and 8 GByte/s for reading and writing, respectively.

Each Tesla K40m hosts a GK110B GPU of the Kepler generation that comprises  $N_{\text{SM}} = 15$  streaming multi-processors. Each streaming multi-processors has 64 double precision floating point pipelines capable of executing one fused multiply-add instruction every clock. This results in a theoretical peak performance per GPU

$$B_{\text{fp}}^{(\text{GPU})} = N_{\text{SM}} \cdot f_{\text{GPU}} \cdot 128 \text{ Flop} \leq 1622 \text{ GFlop/s}, \quad (2)$$

where  $f_{\text{GPU}}$  is the GPU's clock frequency. The device memory is based on GDDR5 memory technology and is directly attached to GK110B GPU with a theoretical maximum bandwidth of 288 GByte/s.

The POWER8 processor provides an on-chip controller (OCC) to measure a set of sensors in the hardware. The data is available out-of-band via a service processor and can be read out by a tool called Amester [15, 28], which recently has been made available open-source.<sup>1</sup> The power sensors considered in this publication are sampled with an interval of 0.25 ms. The read-out granularity depends on the number of sensors, each requires an additional latency of typically 200 ms. The data is, therefore, gathered in irregular intervals and is re-sampled at evenly spaced intervals of  $\Delta\tau = 0.1$  s. To calculate the overall energy consumption we have to aggregate the power consumption measurements  $P_i$  and multiply this with the measurement interval  $\Delta\tau$ . The sensor does not allow for exactly attributing consumed power to individual server components. For instance, the sensor for the 12 V domain includes different I/O devices, also covering part of the power consumed by the GPUs. We combine the values of these sensors when presenting power traces, thus overestimating the actual power consumed by the GPU.<sup>2</sup>

As can be seen from Fig. 4, the read-out time interval can be large compared to the time scale on which power consumption changes. We verified, however, that energy-to-solution values determined from repeated measurements for the same application kernel are consistent with a reasonable bound<sup>3</sup>, which we take as an indication that the power sensor sampling rates are sufficiently fine.

<sup>1</sup> <https://github.com/open-power/amester>.

<sup>2</sup> Comparison with `nvidia-smi` indicates an overhead of roughly 40 W measured with idle system.

<sup>3</sup> Less than 15 % of the mean value in pathological cases.

### 3 Applications

*KKRnano* is an application from materials research and features a very high level of scalability [39]. It is based on the Density Functional Theory (DFT) method that enables an accurate description of the electronic properties of a material. Instead of solving for the many-electron wave-function, an effective potential is utilised, treating the problem as a single electron system via the Kohn-Sham equation  $\hat{H}\Psi = E\Psi$ . Here  $\hat{H}$  denotes the Hamilton operator,  $E$  the energy and  $\Psi$  the electron wave functions. *KKRnano* finds solutions to this equation in the form of the Green function  $\hat{G}(E) = (E - \hat{H})^{-1}$ , as proposed by Korringa, Kohn and Rostoker (KKR) [4, 24, 25].

In large systems, where the number of atoms  $N_{\text{atom}} \gg 1000$ , the Green function formulation can be approximated by systematically truncating long-ranged interactions between well-separated atoms. This reduces the overall complexity of the method from cubic to linear and, hence, large systems with 100,000 atoms and more become feasible. The high level of exposed parallelism can be exploited using MPI plus OpenMP or MPI plus compute accelerators, like GPUs.

Most of the computational efforts are spent on solving a linear system  $A\gamma = \omega$  for each atom. The matrix  $A$  is block-sparse with a typical block size  $b = 16$ . The inversion is done locally on a single node using the Quasi Minimal Residual (QMR) method, an iterative solver [16], modified to work over matrices and vectors with block structure. For the following discussion, we focus on the application of the operator to a block-structured dense vector as the central part of the QMR solver. *KKRnano* utilises double-precision complex numbers, requiring 16 Byte per element and 8 Flop to perform a complex multiply-accumulate operation.

Our performance modelling approach is based on the information exchange function concept [5, 30], which captures as a function of the problem size the amount of information that has to be exchanged within a computer architecture. On a single node the most important factors are the number of floating-point operations  $I_{\text{fp}}$  as well as the amount of data loaded and stored, namely  $I_{\text{ld}}$  and  $I_{\text{st}}$ . We further need to consider the data exchanged between the accelerator and its host,  $I_{\text{acc}}$ . For the *KKRnano* kernel we previously derived these information exchange functions:

$$I_{\text{fp}} = 2N_{\text{iter}} \cdot \frac{N_{\text{atom}}}{N_{\text{node}}} \cdot N_{\text{tr}} \cdot N_{\text{cl}} \cdot b^3 \cdot 8 \text{ Flop}, \quad (3)$$

$$I_{\text{ld}} = 2N_{\text{iter}} \cdot \frac{N_{\text{atom}}}{N_{\text{node}}} \cdot N_{\text{tr}} \cdot N_{\text{cl}} \cdot b^2 \cdot 16 \text{ Byte}, \quad (4)$$

$$I_{\text{st}} = 2N_{\text{iter}} \cdot \frac{N_{\text{atom}}}{N_{\text{node}}} \cdot N_{\text{cl}} \cdot b^2 \cdot 16 \text{ Byte}, \quad (5)$$

$$I_{\text{acc}} = \frac{N_{\text{atom}}}{N_{\text{node}}} \cdot (2 + N_{\text{cl}}) \cdot N_{\text{tr}} \cdot b^2 \cdot 16 \text{ Byte}. \quad (6)$$

To construct a performance model, we follow a procedure described in [3], where we made the assumption that latency depends linearly on the amount

**Table 1.** Parameters used for all KKRnano benchmark runs.

Implementation	$N_{\text{atom}}/N_{\text{node}}$	$N_{\text{iter}}$	$N_{\text{tr}}$	$N_{\text{cl}}$	$b$
CPU	20	1000	1000	13	16
GPU	20	10000	1000	13	16

of exchanged information. When executing the application on the POWER8 processors, we expect it to be bound by floating-point operation throughput, because of the relatively high memory bandwidth. On the GPU, however, we expect the memory bandwidth to be the limiting factor. Additionally, we have to take the time needed to exchange data between host and device into account. We, therefore, make the following semi-empirical modelling ansatz:

$$\Delta t_{\text{solver}}^{\text{CPU}} = a_0^{\text{CPU}} + a_{1,\text{fp}}^{\text{CPU}} I_{\text{fp}} \quad (7)$$

$$\Delta t_{\text{solver}}^{\text{GPU}} = a_0^{\text{GPU}} + a_{1,\text{mem}}^{\text{GPU}} (I_{\text{ld}} + I_{\text{st}}) + a_{1,\text{acc}}^{\text{GPU}} I_{\text{acc}}, \quad (8)$$

where the coefficients have to be determined by fitting this ansatz to performance measurements. As the constants offsets are sufficiently small, we set  $a_0^{\text{CPU}} = 0$  and  $a_0^{\text{GPU}} = 0$ .

*B-CALM* (Belgium-California Light Machine) is a research application for studying photonics or in general electromagnetic waves in media [41]. It implements the Finite-Difference Time-Domain (FDTD) numerical method for simulating classical electrodynamic interaction, that is to solve Maxwell's equations in a medium [37]. The evolution of the discretised electromagnetic fields is described by a first-order spatial and temporal difference equation, alternating between magnetic and electrostatic components.

B-CALM exploits the inherently high level of parallelism in the FDTD algorithm by making use of GPUs. Significant gains in performance compared to CPU-only-implementations have been reported for FDTD in general [27, 33, 41]. Realistic problems require the use of multiple distributed GPUs due to the memory footprint.

We apply a similar semi-empirical performance modelling approach as for KKRnano [3]. We consider a single node where the simulation domain is parallelised over  $p = 2$  GPUs in  $z$ -direction. The total time taken for an update step  $\Delta t$  can be decomposed into three contributions: boundary update  $\Delta t_{\text{bnd}}$ , update of the interior domain  $\Delta t_{\text{bulk}}$  and exchange of the boundary  $\Delta t_{\text{com}}$  between the direct neighbours of the  $p$  processes, where the last two operations are overlapped. Our semi-empirical performance model ansatz is:

$$\Delta t = \Delta t_{\text{bnd}} + \max(\Delta t_{\text{bulk}}, \Delta t_{\text{com}}), \quad (9)$$

$$\Delta t_{\text{bulk}} = a_{\text{bulk}} + V \cdot b_{\text{bulk}} \text{ 168 Byte}, \quad (10)$$

$$\Delta t_{\text{com}} = a_{\text{com}} + S \cdot b_{\text{com}} \text{ 48 Byte}, \quad (11)$$

$$\Delta t_{\text{bnd}} = a_{\text{bnd}} + S \cdot b_{\text{bnd}} \text{ 336 Byte}, \quad (12)$$

**Table 2.** Parameters used for all B-CALM benchmark runs.

$p = N_{\text{proc}}/N_{\text{node}}$	$L_x$	$L_y$	$L_z$
2	512	512	96

where  $V = (L_x/p - 2) \cdot L_y \cdot L_z$  is the local sub-domain’s volume excluding halo layers and  $S = L_y \cdot L_z$  corresponding halo volume. For our benchmarks, we utilise a one-dimensional domain decomposition with the parameters summarised in Table 2.

*GROMACS* simulates Newtonian equations of motion to perform molecular dynamics. It is primarily designed for biochemical molecules like proteins, lipids and nucleic acids. A very important design goal of *GROMACS* is high performance which is achieved through algorithmic optimisations and by exploiting all available hardware resources. To achieve the latter it uses intrinsic functions for SIMD vectorisation and supports MPI and OpenMP for parallelisation on CPUs. CUDA C is used to accelerate the calculation of the non-bonded force components on NVIDIA accelerators. GPU accelerated runs of *GROMACS* utilise task parallelism between different force components to maximise resource utilisation, i.e. overlap CPU and GPU work. *GROMACS* applies a dynamic CPU/GPU load balancing, however, for optimal efficiency both parts – CPU and GPU – in the system need to be balanced [1].

For all runs of *GROMACS* in this article a water box with 1 million water molecules, i.e. 3 million particles, was simulated for 12000 time steps corresponding to 24 pico seconds of simulation time. We selected the Particle Mesh Ewald method for the long-range electrostatics and applied a domain decomposition with 40 domains. Each domain is processed by one MPI rank with 4 OpenMP threads and respectively 20 domains share a GPU.

## 4 Tuning for Energy Efficiency

The energy  $E$  required to execute a computational task is given by

$$E = \int_{t_0}^{t_0 + \Delta t} d\tau P(\tau), \quad (13)$$

where  $t_0$  is the time when the computational task is started,  $\Delta t$  the time needed to execute the task, and  $P(\tau)$  the power consumed by the system at time  $\tau$ . If we assume the power consumption during execution of the kernel to be constant, i.e.  $P(\tau) = \bar{P}$  then the relation simplifies to  $E = \bar{P} \cdot \Delta t$ .

For the given node architecture, we identify the following options for changing the implementation or the execution environment such that  $\Delta t$  and  $P$  change:

1. Dynamic voltage and frequency scaling (DVFS) capabilities of the POWER8 processor.
2. Exploitation per POWER8 core DVFS settings and low-power states.

3. Modification of the GPU's clock frequency  $f_{\text{GPU}}$ .
4. Change of GPU driver settings.
5. Use of algorithms optimised for energy-to-solution.

The frequency  $f$  a processing element is running at is exerting a significant influence on its power drain as voltage has to be increased when  $f$  is raised. The power consumption effectively scales as  $P \propto f^\gamma$ , with  $\gamma \simeq 3$ . On POWER8 the clock frequency can be managed by restricting the range of frequencies used by the Linux<sup>4</sup> kernel governor. The strategy employed by the governor follows the on-demand setting and results in the highest available frequency being chosen during our kernel's runtime and a power saving state in between runs. The available frequencies are distributed almost uniformly between 2.061 GHz and 3.690 GHz in 50 steps, out of which we chose  $f_{\text{CPU}} \in \{2.061, 2.294, 2.493, 2.693, 2.892, 3.092, 3.291, 3.491, 3.690\}$  GHz.

The POWER8 processor comprises a PowerPC<sup>5</sup>-based On-Chip Controller (OCC), which provides for real-time control of per-core clock frequency and power states [17]. Additionally, cores can be switched off under control of the operating system enabling software to reduce the number of active cores. Instead of user selection of active cores, one may use thread pinning to move all workload to a subset of cores and rely on the OS to detect unused cores and adjust frequency and power settings accordingly.

Recently, also GPUs became able to operate at different clock frequencies. For the K40 GPU we considered 4 different clock states, namely  $f_{\text{GPU}} \in \{666, 745, 810, 845\}$  MHz. Changing the GPU frequency of the K40 GPU is possible via application clock settings which are accessible from the NVIDIA Management Library (NVML) or the `nvidia-smi` command line tool [26].

The architecture considered here is best used from an energy-efficiency point-of-view if significant parts of the application can be off-loaded to the GPUs. The reason is that the GPU typically requires less energy per floating-point operation. In many cases, there is no concurrent execution of tasks both, on CPU and GPU, i.e. the CPU should be largely inactive while a kernel is executed on the GPU. However, by default the GPU driver is polling on an active lock while waiting for a kernel to complete. Alternatively, the driver can yield its thread to the Linux kernel scheduler and thus avoid polling, which is achieved by changing the device flags from `cudaDeviceScheduleSpin` to `cudaDeviceScheduleYield`. This might come with a small cost penalty for the driver as notification through the kernel may be delayed.

Recently, optimisation of algorithms for energy-efficiency attracted increasing attention. In particular, in case of solving linear systems often a choice between different algorithms exists (see, e.g., [22] for a recent investigation). We have not pursued this option within the scope of this paper.

---

<sup>4</sup> Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

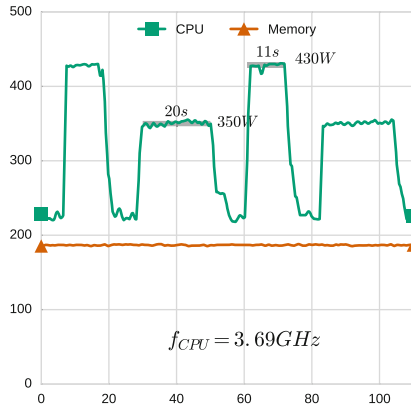
<sup>5</sup> Trademark of IBM in USA and/or other countries.

## 5 Measurements Results

*Exploitation per POWER8 core DVFS settings and low-power states.* In Fig. 1 we show power measurements for the KKRnano solver running either on 20 or 10 cores. We observe that the additional power consumed by the processor after starting the execution of the kernel is reduced by almost a factor 2. At the same time, the execution time  $\Delta t$  almost doubles. Given the high base power load for the processors, memory and GPUs, the overall reduction in power consumption is less than 15 % and thus energy-to-solution increases by more than 70 %.

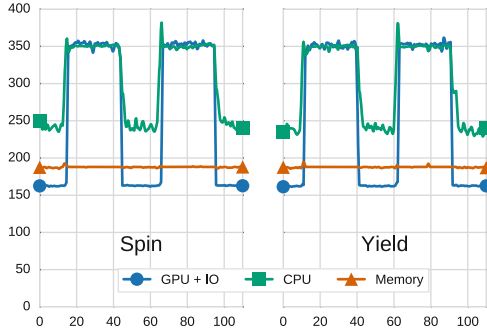
*Change of GPU driver settings.* For the same application kernel, we analysed the power consumption when off-loading this to the GPUs. While a kernel is running on the GPU, there are no application tasks running on the CPU. Nevertheless, a significant increase of the power consumption beyond what is consumed without the user application running, as can be seen in Fig. 2. Between kernel launches the CPU performs tasks for steering the solvers progress. We investigated the effect of advising the CUDA driver to yield its thread to the operating system scheduler when possible. No effect was observed, consistently for both KKRnano and BCALM.

*Modification of clock frequencies.* Next we consider the power consumed by applications running on the POWER8 processor at different clock frequencies  $f_{CPU}$ . In Fig. 3 we show the power consumed by the processors, the memory subsystem as well as the GPU (including the other I/O devices attached to the 12V power rail) during execution of the KKRnano solver on the CPU. As expected, the power consumption increases with  $f_{CPU}$ , while the execution time reduces.



**Fig. 1.** Power consumption for four invocations of the KKRnano solver running on the CPU, only. 40 threads are distributed either over 20 (1st and 3rd invocation) or 10 cores (2nd and 4th).





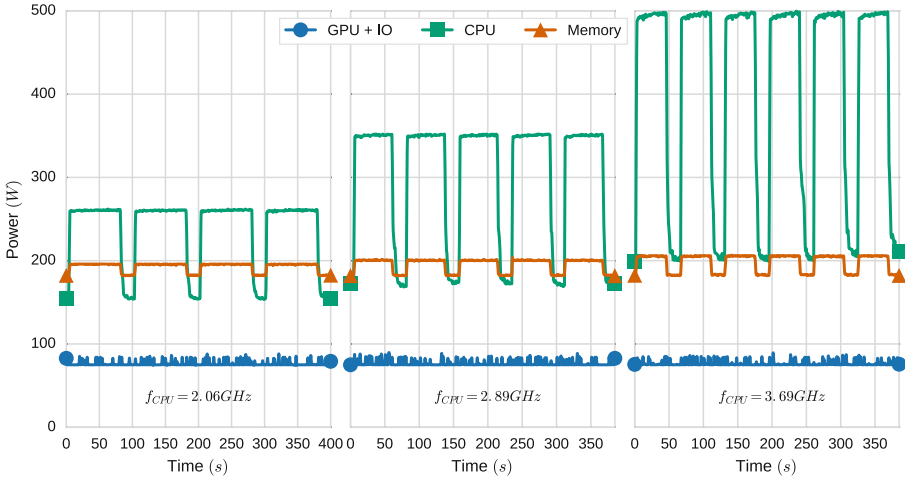
**Fig. 2.** Power consumption for two invocations of the KKRnano solver running on the GPU with the GPU driver polling for completion of the kernel (Spin) or yielding its thread to the operating system (Yield).

**Table 3.** Energy-to-solution for the KKRnano solver running on the CPU in units of kJ.

$f_{\text{CPU}}$ (GHz)	CPU	Disk	Fan	GPU	IO	Memory	Total
2.06	20.6	2.6	5.8	1.9	4.2	15.5	50.5
2.29	20.3	2.4	5.4	1.7	3.8	14.2	47.8
2.49	20.2	2.2	5.0	1.6	3.5	13.3	45.8
2.69	20.2	2.1	4.7	1.5	3.3	12.5	44.4
2.89	20.3	1.9	4.4	1.4	3.1	11.8	42.9
3.09	20.4	1.8	4.1	1.3	2.9	11.1	41.7
3.29	20.5	1.7	4.0	1.2	2.8	10.5	40.7
3.49	21.5	1.7	3.8	1.2	2.7	10.2	41.1
3.69	23.0	1.6	3.5	1.2	2.6	9.8	41.6

Results for energy-to-solution as a function of  $f_{\text{CPU}}$  are documented in Table 3. For small clock frequencies the increase in time-to-solution overcompensates the reduction in power consumption, which causes energy-to-solution to increase. It is thus beneficial to use a higher clock frequency.

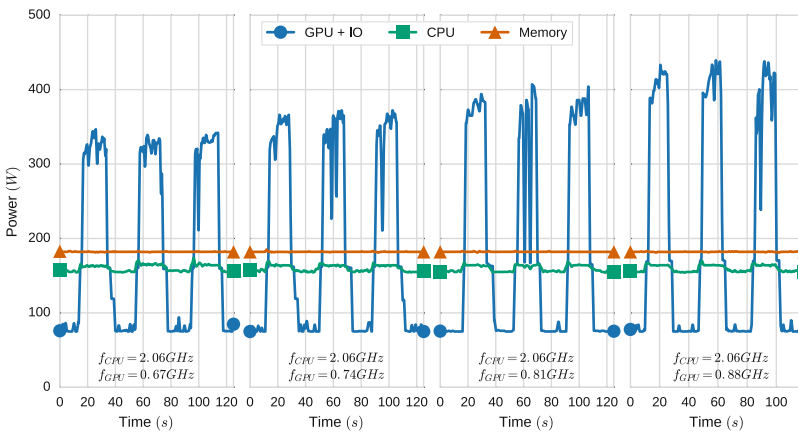
As shown in Fig. 4, a similar effect is observed for B-CALM, where the kernels are running on the GPU. Since the kernels are completely running on the GPU, we consider here only the case where the processor clock is kept fixed at minimal value and  $f_{\text{GPU}}$  is varied. In Tables 4 and 5 we show our results for energy-to-solution for B-CALM as well as the GPU-accelerated version of the KKRnano solver when using different GPU clock settings. In both cases we observe that larger  $f_{\text{GPU}}$  result in smaller energy-to-solution.



**Fig. 3.** Power consumption for multiple invocations of the KKRnano solver running on the CPU for different settings of  $f_{CPU}$ .

**Table 4.** Energy-to-solution for B-CALM in units of kJ.

$f_{CPU}$ (GHz)	$f_{GPU}$ (MHz)	CPU	Disk	Fan	GPU	IO	Memory	Total
2.06	666	4.1	0.8	1.8	3.8	2.9	4.5	17.8
2.06	745	4.0	0.8	1.8	3.8	2.8	4.5	17.7
2.06	810	3.5	0.7	1.7	3.7	2.5	3.9	15.9
2.06	875	3.2	0.6	1.5	4.0	2.5	3.5	15.2



**Fig. 4.** Example of a GPU power trace over multiple invocations of B-CALM. GPU clocks are set to each of the available values.

**Table 5.** Energy-to-solution for the KKRnano solver running on the GPU in units of kJ.

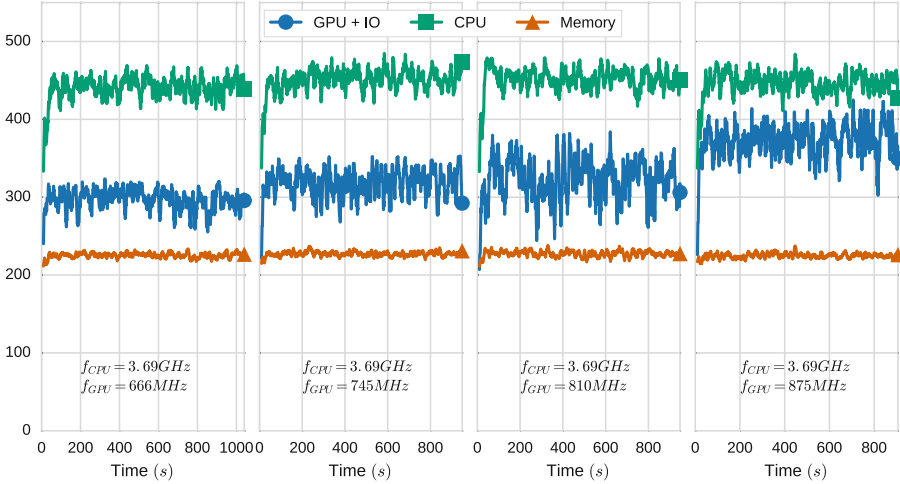
$f_{\text{CPU}}$ (GHz)	$f_{\text{GPU}}$ (MHz)	CPU	Disk	Fan	GPU	IO	Memory	Total
2.06	666	7.1	1.1	2.5	6.3	4.0	5.9	26.8
2.06	745	6.0	1.0	2.2	6.4	3.8	5.4	24.8
2.06	810	5.9	0.9	2.0	6.6	3.6	5.0	24.0
2.06	875	5.6	0.8	1.9	7.1	3.5	4.7	23.7

As opposed to the other two codes, GROMACS has a task parallelisation scheme that allows to execute tasks on CPU and GPU concurrently. Data is exchanged at defined points in the algorithm [29]. As both, the CPU and the GPU consume a significant amount of power even when not being in use, an imperfect overlap of tasks running on any of the 4 processing devices will result in a loss in energy efficiency. Table 6 shows the energy consumption and time to solution for a single run as a function of  $f_{\text{GPU}}$  for different processor frequencies  $f_{\text{CPU}}$ . We observe that changing the processor frequency  $f_{\text{CPU}}$  has a much larger effect on energy-to-solution than changing the frequency of the GPU  $f_{\text{GPU}}$ . The energy to solution is reduced by 8% to 17% when using the higher CPU frequency, while variations of  $f_{\text{GPU}}$  change the energy to solution by at maximum 7%. While the energy consumed by the GPU increases for larger clock frequencies (up to 40% with low  $f_{\text{CPU}}$ , up to 22% for high  $f_{\text{CPU}}$ ), the energy consumed by the remaining parts of the compute node is reduced (by up to 11%) due to the shorter time to solution (see Fig. 5). On the other hand, increasing  $f_{\text{CPU}}$  has a even more significant effect on time-to-solution (up to 32%), which overcompensates the increase in CPU power consumption (up to 20%). The net effect is a reduction of energy-to-solution at higher clock speed, ranging between 8% and 20%.

Finally, we give power measurements for the behaviour of the STREAM benchmark at different clock settings in Table 7 and the corresponding power trace in Fig. 6. Notably, the achieved performance does not depend on the core

**Table 6.** GROMACS energy to solution in kJ.

$f_{\text{CPU}}$ (GHz)	$f_{\text{GPU}}$ (MHz)	CPU	Disk	Fan	GPU	IO	Memory	Total	Time (s)
2.06	666	365.6	44.6	101.3	203.0	181.4	303.2	1199.2	1395
2.06	745	363.6	45.1	103.2	247.2	191.1	304.4	1254.6	1410
2.06	810	352.1	43.4	98.7	259.7	187.1	293.4	1234.4	1359
2.06	875	349.8	43.0	97.6	284.0	188.7	290.9	1253.8	1344
3.69	666	457.2	33.1	75.4	168.5	139.7	234.0	1108.0	1041
3.69	745	426.3	30.2	69.0	170.1	130.1	215.1	1040.8	949
3.69	810	426.7	30.2	68.3	175.0	129.9	215.1	1045.2	949
3.69	875	403.3	28.9	65.3	205.6	130.7	203.5	1037.2	907



**Fig. 5.** GROMACS power trace for different GPU clock settings.

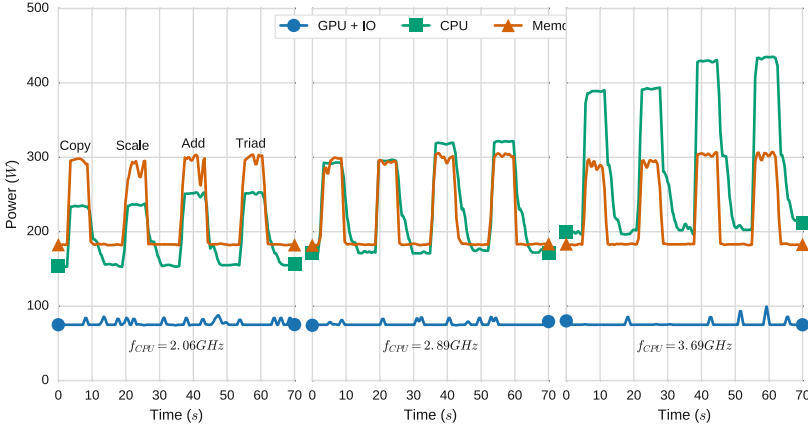
**Table 7.** Performance of STREAM as a function of the CPU frequency.

$f_{CPU}$ (GHz)	Add (GB/s)	Copy (GB/s)	Scale (GB/s)	Triad (GB/s)
2.061	195.1	222.1	338.5	289.7
2.294	197.0	221.8	338.4	293.5
2.493	195.0	221.4	334.5	291.8
2.693	197.4	221.7	337.9	293.9
2.891	196.3	218.6	327.0	293.4
3.291	197.6	222.1	334.3	296.5
3.690	196.0	219.2	330.0	293.1

clock. This is explained by the fact that the serial link between CPU and off-chip memory controller runs at a fixed speed. In line with previous observations [12, 31], one can conclude that for purely memory bandwidth bound operations the clock speed can be reduced without impacting performance.

## 6 Power and Energy Modelling

Our strategy to derive an energy model is as follows. We only consider the case where all options for tuning for energy efficiency except for changing clock speeds of CPU and GPU are applied. Furthermore, we make the assumption that power consumption during application kernel execution is constant. The challenge thus reduces to the design of a performance model and a power model to determine time-to-solution  $\Delta t$  and power consumption  $\bar{P}$ , which we assume to be constant



**Fig. 6.** Example of a CPU power trace over the four tasks in the STREAM micro-benchmark. Three of used values for the CPU clock are shown.

during the time interval  $\Delta t$ , as a function of the CPU and GPU clock frequencies  $f_{\text{CPU}}$  and  $f_{\text{GPU}}$ , respectively.

For deriving a power model we choose for a phenomenological approach, which has the advantage of being sufficient simple to be of practical use. The concerned compute devices are based on CMOS circuits, where power is typically split in two components: A static power consumption due to leakage currents and a dynamic contribution due to charging and de-charging the capacitive elements of the circuit. The latter is assumed to depend linearly on the clock frequency and quadratically on the supply voltage. Since the latter needs to be increased for higher clock frequencies, we can in first approximation assume a cubic dependency on the frequency. We therefore make the following generic ansatz in terms of the frequency  $f$ :

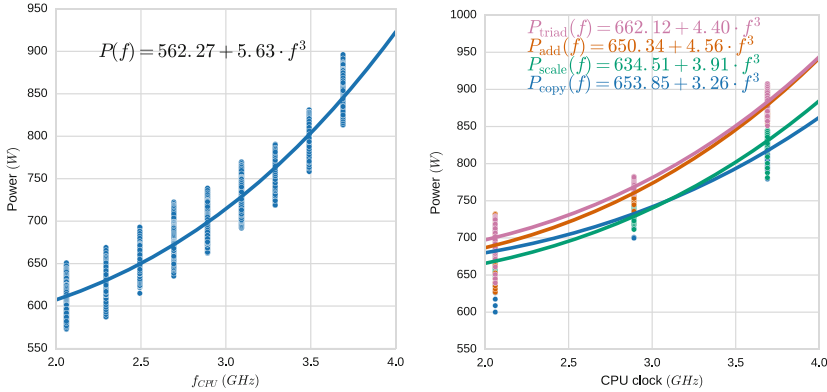
$$\bar{P}(f) = p_0 + p_3 f^\gamma, \quad (14)$$

with  $\gamma = 3$  fixed.<sup>6</sup> The parameters  $p_0$  and  $p_3$  will be determined through least-square fits to experimental data.

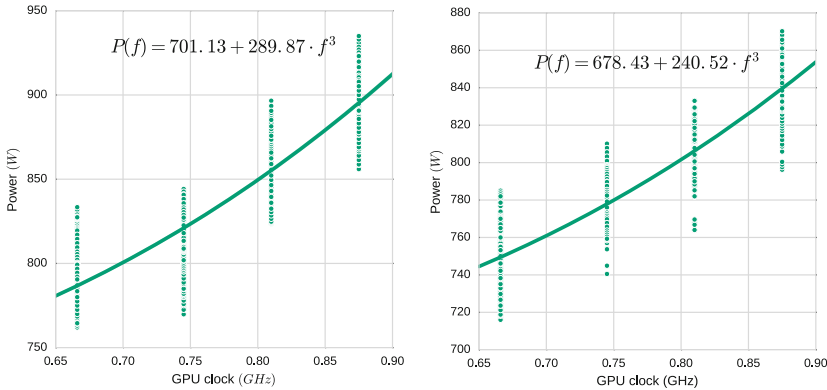
In Fig. 7 we show results for power consumption of the KKRnano solver as well as 4 different STREAM benchmarks, which are all executed on the POWER8 processor, as a function of  $f_{\text{CPU}}$ . Multiple measurements have been obtained by executing the application kernels and benchmarks multiple times in sequence. Active phases were selected based on thresholding the power for the GPU halfway between minimum and maximum values. We observe that Eq. (14) using  $f = f_{\text{CPU}}$  parametrises the measurements very well.

The instruction mix for the KKRnano kernel is similar to the triad micro-benchmark. However, the KKRnano kernel achieves a significantly higher instruction throughput when compared to triad (instruction-per-cycle  $\text{IPC} = 1.44$  versus 0.24 when using 40 threads). The higher throughput is mainly

<sup>6</sup> We also performed fits with  $\gamma$  as a free parameter, where we found  $\gamma \simeq 3$ .



**Fig. 7.** Power measurement results for different settings of  $f_{\text{CPU}}$  for the KKRnano kernel running on the CPU (left) and the STREAM benchmarks (right). The solid lines show results from fits to Eq. (14).

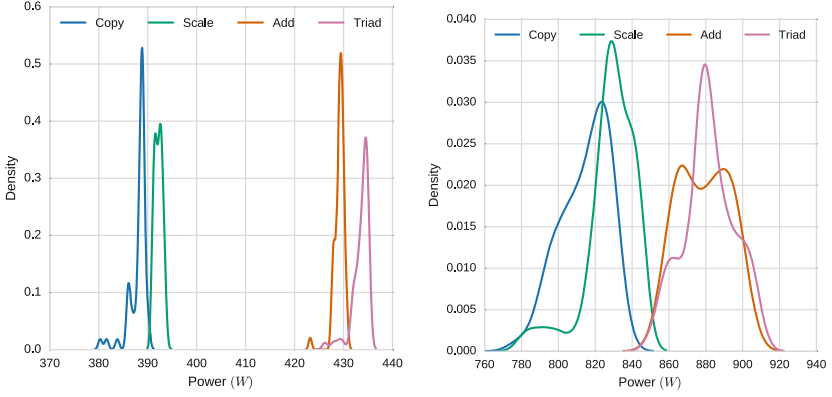


**Fig. 8.** Fitting the model against KKRnano GPU (left) and B-CALM (right) experimental results

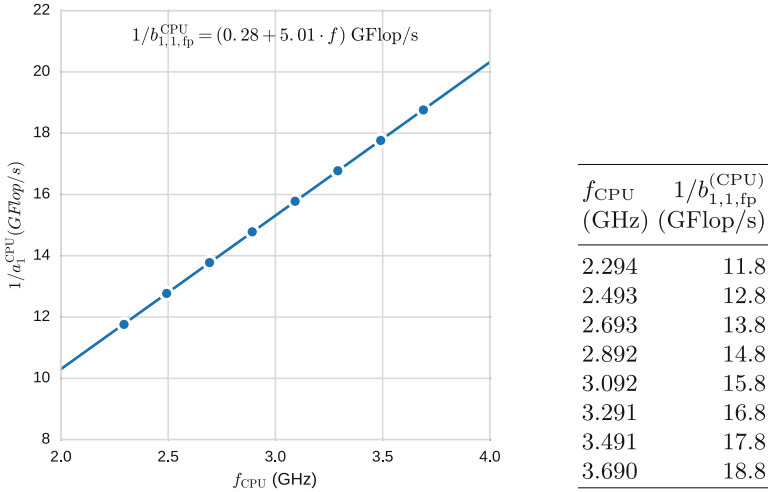
due to the data reuse in the matrix-matrix multiplication and the resulting cache efficiency. This is consistent with the observation that  $p_3$  is larger for KKRnano than for the STREAM benchmarks.

We extended the analysis to the GPU-accelerated application kernels, where we varied  $f_{\text{GPU}}$ . Results for KKRnano and B-CALM are summarised in Fig. 8. Again we observe that Eq. (14) provides a good parametrisation of the measured power.

Next, we extend the performance model for KKRnano introduced in Eq. (7) by considering the parameter  $a_{1,\text{fp}}^{\text{CPU}}$  to be a function of the clock frequency  $f_{\text{CPU}}$ . We observe from Fig. 10 that the results for different  $f_{\text{CPU}}$  can be parametrised as follows:



**Fig. 9.** Estimated distributions of power drain for  $f_{\text{CPU}} = 3.69$  GHz for the four STREAM micro-benchmarks on the CPU level (left) and on the node level (right).



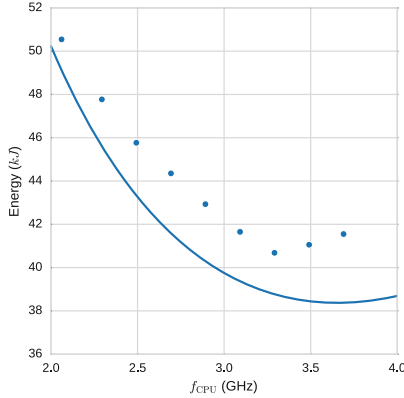
**Fig. 10.** KKRnano performance model results versus processor clock  $f_{\text{CPU}}$ .

$$\frac{1}{a_{1,fp}^{\text{CPU}}(f_{\text{CPU}})} = b_{1,0,fp}^{\text{CPU}} + b_{1,1,fp}^{\text{CPU}} \cdot f_{\text{CPU}} \quad (15)$$

with the empirical factors  $b_{1,0,fp}^{\text{CPU}} = 0.28$  GFlop/s and  $b_{1,1,fp}^{\text{CPU}} = 5.01$  GFlop.

Plugging this result into Eq. (7) gives us  $\Delta t_{\text{solver}}^{\text{CPU}}$  as a function of the processor clock speed  $f_{\text{CPU}}$ . We combine this with the power model of Eq. (14) to derive the following energy model:

$$E_{\text{solv}}^{\text{CPU}} = \overline{P}(f_{\text{CPU}}) \cdot \Delta t(f_{\text{CPU}}) = I_{fp} \frac{p_0 + p_3 \cdot f_{\text{CPU}}^3}{b_{1,0,fp}^{\text{CPU}} + b_{1,1,fp}^{\text{CPU}} \cdot f_{\text{CPU}}}. \quad (16)$$



**Fig. 11.** Modelled energy-to-solution metric for the KKRnano CPU solver as a function of the processor core clock  $f_{\text{CPU}}$ . In comparison the measured data from Table 5 is shown.

In particular, if  $p_0$  is large compared to  $p_3 \cdot f_{\text{CPU}}^3$  and if  $b_{1,0,\text{fp}}^{\text{CPU}}$  is small compared to  $b_{1,1,\text{fp}}^{\text{CPU}} \cdot f_{\text{CPU}}$  the frequency dependence may be dominated by the denominator. In this case we expect  $E_{\text{solv}}^{\text{CPU}} \propto f_{\text{CPU}}^{-1}$ . Once  $f_{\text{CPU}}$  becomes large, the numerator will dominate such that  $E_{\text{solv}}^{\text{CPU}} \propto f_{\text{CPU}}^2$ . In Fig. 11 we plot model results for a wide range of processor clock frequencies. We observe initially energy-to-solution to drop, until for large frequencies the numerator prevails.

Consistent with the results shown in Table 3 we determine an – in terms of energy-efficiency – optimal clock setting, namely  $f_{\text{CPU}} = 3.5$  GHz. We find our model systematically underestimating the required energy by maximally around 7% over the range 2–4 GHz. KKRnano on the CPU achieves energy efficiencies of around 45 J/GFlop or the equivalent of 22.2 MFlop/J.

## 7 Related Work

Given the growing importance of power consumption of modern HPC systems, there is a growing number of papers reporting on work related to power measurements as well as power modelling.

An early framework for accurate and scalable power profiling is PowerPack [18], which relies however on power data acquisition devices to be added to all cluster nodes. Another framework is PowerScope, which was developed for mobile compute platforms [14]. Some node architectures feature fine-grained power measurements capabilities, which are often based on reconfigurable hardware components for high-frequency read-out of power sensors (see, e.g., [20]). Demmel and Gearhart proposed to use on-chip counters, more specifically Intel’s Running Average Power Limit (RAPL) interface, for measuring energy consumption at subroutine level [10]. Investigations about the use of the RAPL interface for measuring memory power can be found in the literature [9]. Such kind of measurements using counters that can be read-out from the processor have become



easier with the integration into performance measurements frameworks like PAPI [42]. An integration of the the node-level power measurement framework Amester into Vampir has been reported by Knobloch and others [23]. Another approach is taken, where power measurement capabilities are integrated into a Linux kernel module [32]. Both, the PowerPack as well as the PAPI framework have been used to explore the energy footprint of linear algebra algorithms [11].

Other work aims on modelling power consumption at processor level. For instance, a power measurement and runtime power modeling framework for the Intel Netburst architecture is presented [21]. Processor level power models have also been proposed and used in the context of processor selection to identify, e.g., processors that yield a certain frequency in a given power budget [45]. At a time, when direct power measurement capabilities were largely non-existing, Van Bui and others advocated using power models that used performance counters as input, for estimating power consumed by applications [6]. Such approaches can be exploited for dynamic profitability estimation of dynamic voltage and frequency scaling (DVFS) [12, 31].

With GPUs becoming more widely used as computational devices, research was started on power models for GPUs. Statistical methods to model power and energy consumption of common high performance kernels have been employed successfully [19]. In a new approach hardware performance counter data is combined with machine learning to model power and performance for modern GPU-based systems [34]. Also Wu and others applied machine learning in their work [44].

Other approaches to power modelling target numerical kernels. Statistical methods like multi-variable regression are used to model the power and performance of the High-Performance Linpack benchmark [36]. Model parameters are application parameters like matrix size or block size. The same numerical kernel is also considered [7]. Power models for another important numerical kernel, namely Cholesky factorisation, are considered using a semi-analytical approach [2]. Machine learning techniques are also employed to create application specific power models [40]. Input parameters include application specific plus system parameters, like clock frequency. For an extensive survey of power and energy efficient techniques for high performance numerical linear algebra operations we refer to the work of Tan and others [38].

Performance models for more complex applications are, e.g., considered in [43]. Here Wittmann and others use models to explore energy efficiency of a computational fluid-dynamics (CFD) application based on the Lattice Boltzmann method on Intel Sandy Bridge multi-core processors. They combined an analytic performance model with a simple phenomenological power model. A semi-analytical performance and power model for another CFD application, namely a mini-application version of the spectral element method based code Nek5000, is presented in [35].

## 8 Summary and Conclusions

In this paper we investigated for several scientific applications, how energy efficiency of currently available, GPU-accelerated POWER8 servers can be optimised. We discussed several options for how to change application and/or system configuration such that energy-to-solution for a given workload could be reduced. We found that, in particular, adjusting processor or GPU clock have major impact on energy-to-solution. Except for the STREAM benchmarks we found that applications typically benefit from higher clock speeds. This behaviour can be explained by the relatively high base power consumption. Within a certain range of clock frequencies it is thus more beneficial to reduce time-to-solution by using higher clock frequencies. For applications running on the processor only, this applies, however, only to applications that are limited by instruction throughput.

Furthermore, we found that the measured power consumption can be described by a simple phenomenological power model, which describes power consumption as a function of the CPU or GPU clock frequency. For one of the application kernels, the KKRnano solver, we extended a performance model to include variations of the processor clock frequency. By combining the performance and power model we derived a simple energy model. The curve shown in Fig. 11 is based on four parameters, only, which are relatively easy to determine. This can be used to tune the clock frequency. For KKRnano the model results and measurements lead to a reasonably consistent results for the CPU clock frequency  $f_{\text{CPU}}$  that minimises energy-to-solution.

A significant fraction of the base power consumption, i.e. the power that is consumed without an application running, is due to the memory subsystem. The base power consumption on servers based on POWER8 processors with a smaller number of memory buffer chips is lower and therefore the clock frequency, for which power consumption is minimised, is likely to be smaller. The picture is expect to change again for the recently announced POWER9-SO processors with directly attached memory.

**Acknowledgements.** This work has been carried out in the context of the *POWER Acceleration and Design Center*, a joined project between IBM, Forschungszentrum Jülich and NVIDIA. We acknowledge generous support from IBM by providing early access to GPU-accelerated POWER8 systems.

## References

1. Abraham, M.J., Murtola, T., Schulz, R., Páll, S., Smith, J.C., Hess, B., Lindahl, E.: GROMACS: high performance molecular simulations through multi-level parallelism from laptops to supercomputers. *SoftwareX* **1**, 19–25 (2015)
2. Alonso, P., Dolz, M.F., Mayo, R., Quintana-Ortí, E.S.: Modeling power and energy of the task-parallel Cholesky factorization on multicore processors. *Comput. Sci. Res. Dev.* **29**(2), 105–112 (2012). doi:[10.1007/s00450-012-0227-z](https://doi.org/10.1007/s00450-012-0227-z)

3. Baumeister, P.F., Hater, T., Kraus, J., Pleiter, D., Wahl, P.: A performance model for GPU-accelerated FDTD applications. In: 2015 IEEE 22nd International Conference on High Performance Computing (HiPC), pp. 185–193, December 2015
4. Beeby, J.: The density of electrons in a perfect or imperfect lattice. *Proc. R. Soc. Lond. A Math. Phys. Eng. Sci.* **302**(1468), 113–136 (1967). The Royal Society
5. Bilardi, G., Pietracaprina, A., Pucci, G., Schifano, F., Tripiccione, R.: The potential of on-chip multiprocessing for QCD machines. In: Bader, D.A., Parashar, M., Sridhar, V., Prasanna, V.K. (eds.) *HiPC 2005*. LNCS, vol. 3769, pp. 386–397. Springer, Heidelberg (2005). doi:[10.1007/11602569\\_41](https://doi.org/10.1007/11602569_41)
6. Bui, V., Norris, B., Huck, K., McInnes, L.C., Li, L., Hernandez, O., Chapman, B.: A component infrastructure for performance and power modeling of parallel scientific applications. In: *Proceedings of the 2008 compFrame/HPC-GECO Workshop on Component Based High Performance, CBHPC 2008*, pp. 6:1–6:11. (2008). <http://doi.acm.org/10.1145/1456190.1456199>
7. Cabrera, A., Almeida, F., Blanco, V., Giménez, D.: Analytical modeling of the energy consumption for the High Performance Linpack. In: 2013 21st Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), pp. 343–350, February 2013
8. Caldeira, A.B., et al.: IBM Power System S824L technical overview and introduction (2014). <http://www.redbooks.ibm.com/Redbooks.nsf/RedbookAbstracts/redp5139.html>
9. David, H., Gorbatoev, E., Hanebutte, U.R., Khanna, R., Le, C.: RAPL: memory power estimation and capping. In: 2010 ACM/IEEE International Symposium on Low-Power Electronics and Design (ISLPED), pp. 189–194, August 2010
10. Demmel, J., Gearhart, A.: Instrumenting linear algebra energy consumption via on-chip energy counters. Technical report, UCB/EECS-2012-168, EECS Department, University of California, Berkeley, June 2012. <http://www.eecs.berkeley.edu/Pubs/TechRpts/2012/EECS-2012-168.html>
11. Dongarra, J., Ltaief, H., Luszczek, P., Weaver, V.M.: Energy footprint of advanced dense numerical linear algebra using tile algorithms on multicore architecture. In: *The 2nd International Conference on Cloud and Green Computing*, November 2012
12. Eyerman, S., Eeckhout, L.: A counter architecture for online DVFS profitability estimation. *IEEE Trans. Comput.* **59**(11), 1576–1583 (2010)
13. Feng, W.C., et al.: Green500 list, November 2015. <http://green500.org>
14. Flinn, J., Satyanarayanan, M.: PowerScope: a tool for profiling the energy usage of mobile applications. In: *Proceedings of the Second IEEE Workshop on Mobile Computer Systems and Applications, WMCSA 1999*, p. 2 (1999). <http://dl.acm.org/citation.cfm?id=520551.837522>
15. Floyd, M., et al.: Introducing the adaptive energy management features of the POWER7 chip. *IEEE Micro* **31**(2), 60–75 (2011)
16. Freund, R.W., Nachtigal, N.: QMR: a quasi-minimal residual method for non-Hermitian linear systems. *Numer. Math.* **60**(1), 315–339 (1991)
17. Friedrich, J., Le, H., Starke, W., Stuechli, J., Sinharoy, B., Fluhr, E., Dreps, D., Zyuban, V., Still, G., Gonzalez, C., Hogenmiller, D., Malgioglio, F., Nett, R., Puri, R., Restle, P., Shan, D., Deniz, Z., Wendel, D., Ziegler, M., Victor, D.: The POWER8™ processor: designed for big data, analytics, and cloud environments. In: 2014 IEEE International Conference on IC Design Technology (ICICDT), pp. 1–4, May 2014
18. Ge, R., Feng, X., Song, S., Chang, H.C., Li, D., Cameron, K.W.: PowerPack: energy profiling and analysis of high-performance systems and applications. *IEEE Trans. Parallel Distrib. Syst.* **21**(5), 658–671 (2010)

19. Ghosh, S., Chandrasekaran, S., Chapman, B.: Statistical modeling of power/energy of scientific kernels on a multi-GPU system. In: 2013 International Green Computing Conference (IGCC), pp. 1–6, June 2013
20. Hackenberg, D., Ilsche, T., Schuchart, J., Schöne, R., Nagel, W.E., Simon, M., Georgiou, Y.: HDEEM: high definition energy efficiency monitoring. In: Energy Efficient Supercomputing Workshop, E2SC 2014, pp. 1–10, November 2014
21. Isci, C., Martonosi, M.: Runtime power monitoring in high-end processors: methodology and empirical data. In: 2003 Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO-36, pp. 93–104, December 2003
22. Klavík, P., Malossi, A.C.I., Bekas, C., Curioni, A.: Changing computing paradigms towards power efficiency. *Philos. Trans. R. Soc. Lond. A: Math. Phys. Eng. Sci.* **372**(2018), 20130278 (2014)
23. Knobloch, M., Foszczynski, M., Homberg, W., Pleiter, D., Böttiger, H.: Mapping fine-grained power measurements to HPC application runtime characteristics on IBM POWER7. *Comput. Sci. Res. Dev.* **29**(3), 211–219 (2013). doi:[10.1007/s00450-013-0245-5](https://doi.org/10.1007/s00450-013-0245-5)
24. Kohn, W., Rostoker, N.: Solution of the Schrödinger equation in periodic lattices with an application to metallic Lithium. *Phys. Rev.* **94**, 1111–1120 (1954)
25. Korringa, J.: On the calculation of the energy of a Bloch wave in a metal. *Physica* **13**(6), 392–400 (1947)
26. Kraus, J.: Increase performance with GPU boost and K80 autoboot (2014). <https://devblogs.nvidia.com/paralleforall/increase-performance-gpu-boost-k80-autoboot/>
27. Lee, K.H., Ahmed, I., Goh, R.S., Khoo, E.H., Li, E.P., Hung, T.G.: Implementation of the FDTD method based on Lorentz-Drude dispersive model on GPU for plasmonics applications. *Prog. Electromagnet. Res.* **116**, 441–456 (2011)
28. Lefurgy, C., Wang, X., Ware, M.: Server-level power control. In: 2007 Fourth International Conference on Autonomic Computing, ICAC 2007, p. 4, June 2007
29. Lindahl, E.: Molecular simulation with GROMACS on CUDA GPUs (2013). <http://on-demand.gputechconf.com/gtc/2013/webinar/gromacs-kepler-gpus-gtc-express-webinar.pdf>
30. Pleiter, D.: Parallel computer architectures. In: 45th IFF Spring School 2014 “Computing Solids Models, ab-initio methods and supercomputing”. *Schriften des Forschungszentrums Jülich, Reihe Schlüsseltechnologien*, vol. 74 (2014)
31. Rountree, B., Lowenthal, D.K., Schulz, M., de Supinski, B.R.: Practical performance prediction under dynamic voltage frequency scaling. In: 2011 International Green Computing Conference and Workshops (IGCC), pp. 1–8, July 2011
32. Ryffel, S.: LEA<sup>2</sup>P: the Linux energy attribution and accounting platform. Master’s thesis, Swiss Federal Institute of Technology (ETH) (2009). <http://ftp.tik.ee.ethz.ch/pub/students/2009-FS/MA-2009-04.pdf>
33. Shahmansouri, A., Rashidian, B.: GPU implementation of split-field finite-difference time-domain method for Drude-Lorentz dispersive media. *Prog. Electromagnet. Res.* **125**, 55–77 (2012)
34. Song, S., Su, C., Rountree, B., Cameron, K.W.: A simplified and accurate model of power-performance efficiency on emergent GPU architectures. In: 2013 IEEE 27th International Symposium on Parallel Distributed Processing (IPDPS), pp. 673–686, May 2013

35. Song, S.L., Barker, K., Kerbyson, D.: Unified performance and power modeling of scientific workloads. In: Proceedings of the 1st International Workshop on Energy Efficient Supercomputing, E2SC 2013, pp. 4:1–4:8. (2013). <http://doi.acm.org/10.1145/2536430.2536435>
36. Subramaniam, B., Feng, W.C.: Statistical power and performance modeling for optimizing the energy efficiency of scientific computing. In: Green Computing and Communications (GreenCom), pp. 139–146, December 2010
37. Taflove, A., Hagness, S.C.: Others: Computational Electrodynamics: The Finite-Difference Time-Domain Method. Artech House, Norwood (1995)
38. Tan, L., Kothapalli, S., Chen, L., Hussaini, O., Bissiri, R., Chen, Z.: A survey of power and energy efficient techniques for high performance numerical linear algebra operations. *Parallel Comput.* **40**(10), 559–573 (2014)
39. Thies, A., et al.: Massively parallel density functional calculations for thousands of atoms: KKRnano. *Phys. Rev. B* **85**, 235103 (2012)
40. Tiwari, A., Laurenzano, M.A., Carrington, L., Snively, A.: Modeling power and energy usage of HPC kernels. In: 2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), pp. 990–998, May 2012
41. Wahl, P., Ly-Gagnon, D., Debaes, C., Miller, D., Thienpont, H.: B-CALM: an open-source GPU-based 3D-FDTD with multi-pole dispersion for plasmonics. In: 2011 11th International Conference on Numerical Simulation of Optoelectronic Devices (NUSOD), pp. 11–12, September 2011
42. Weaver, V.M., Johnson, M., Kasichayanula, K., Ralph, J., Luszczek, P., Terpstra, D., Moore, S.: Measuring energy and power with PAPI. In: 2012 41st International Conference on Parallel Processing Workshops (ICPPW), pp. 262–268, September 2012
43. Wittmann, M., Hager, G., Zeiser, T., Treibig, J., Wellein, G.: Chip-level and multi-node analysis of energy-optimized lattice Boltzmann CFD simulations. *Concur. Comput. Pract. Exper.* **28**, 2295–2315 (2016). doi:10.1002/cpe.3489
44. Wu, G., Greathouse, J.L., Lyashevsky, A., Jayasena, N., Chiou, D.: GPGPU performance and power estimation using machine learning. In: 2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA), pp. 564–576, February 2015
45. Zyuban, V., Taylor, S.A., Christensen, B., Hall, A.R., Gonzalez, C.J., Friedrich, J., Clougherty, F., Tetzloff, J., Rao, R.: IBM POWER7+ design for higher frequency at fixed power. *IBM J. Res. Dev.* **57**(6), 1:1–1:18 (2013)