J Supercomput (2017) 73:4373–4389
DOI 10.1007/s11227-017-2020-z

Modeling power consumption of 3D MPDATA and the CG method on ARM and Intel multicore architectures

Krzysztof Rojek[1] · **Enrique S. Quintana-Ortí**[2] · **Roman Wyrzykowski**[1]

Published online: 28 March 2017

Abstract We propose an approach to estimate the power consumption of algorithms, as a function of the frequency and number of cores, using only a very reduced set of real power measures. In addition, we also provide the formulation of a method to select the voltage–frequency scaling–concurrency throttling configurations that should be tested in order to obtain accurate estimations of the power dissipation. The power models and selection methodology are verified using two real scientific application: the stencil-based 3D MPDATA algorithm and the conjugate gradient (CG) method for sparse linear systems. MPDATA is a crucial component of the EULAG model, which is widely used in weather forecast simulations. The CG algorithm is the keystone for iterative solution of sparse symmetric positive definite linear systems via Krylov subspace methods. The reliability of the method is confirmed for a variety of ARM

The researchers from Czestochowa University of Technology were supported by the National Science Centre, Poland, under Grant No. UMO-2015/17/D/ST6/04059. The researcher from Universidad Jaime I (UJI) was supported by the CICYT Project TIN2014-53495-R of MINECO and FEDER. This work was partially performed during a short-term scientific mission (STSM) of Krzysztof Rojek at UJI supported by EU under the COST Program Action IC1305: Network for Sustainable Ultrascale Computing (NESUS).

✉ Krzysztof Rojek
krojek@icis.pcz.pl

Enrique S. Quintana-Ortí
quintana@uji.es

Roman Wyrzykowski
roman@icis.pcz.pl

1 Institute of Computer and Information Sciences, Czestochowa University of Technology, Czestochowa, Poland

2 Depto. de Ingeniería y Ciencia de Computadores, Univ. Jaume I, Castellón, Spain

and Intel architectures, where the estimated results correspond to the real measured values with the average error being slightly below 5% in all cases.

Keywords Power model · Voltage–frequency scaling (VFS) · Concurrency throttling (CT) · MPDATA · Conjugate gradient · Multicore processors

1 Introduction

Performance analysis has traditionally focused on optimizing the computational throughput of applications (from the perspective of a system's administrator) and/or reducing their execution time (from the point of view of the user). However, the end of Dennard scaling (i.e., the ability to shrink the feature size of integrated circuits while maintaining a constant power density) [8] has promoted energy into a holistic design principle on par with performance [9,18]. As a result, during the past few years we have been witnesses to a considerable amount of works that aimed to analyze the interaction among temperature–power–time–energy for a variety of applications and simple algorithms. In addition, these studies have targeted all sorts of current architectures, including multicore processors, graphics accelerators, many-core processors such as the Intel Xeon Phi or NVIDIA's GPUs, and clusters assembled using these technologies.

One particular aspect that many of these past works address is the use of (dynamic) voltage–frequency scaling (VFS) [10], sometimes combined with (dynamic) concurrency throttling (CT) [7], as a means to reduce power dissipation and/or energy consumption.

In this paper, we contribute toward raising the energy awareness among the scientific community by modeling the impact of VFS and CT on the power dissipation of the *multidimensional positive definite advection transport algorithm* (MPDATA), a key component that strongly dictates the computational performance as well as the energy consumption of the multiscale fluid model EULAG [24,31]. In addition, we also include the conjugate gradient (CG) algorithm for the iterative solution of sparse linear systems via Krylov subspaces methods [30].

In more detail, our paper makes the following contributions:

- We formulate power models that estimate the power consumption as a polynomial function of the frequency or the number of cores. Using these models, we then introduce a methodology for power approximation based on a very reduced set of real power measures.
- In addition, we propose a systematic procedure to select the VFS–CT configurations that should be tested to improve the precision of the power models. Here, we emphasize that both the methodology for power approximation and the procedure to select the samples in principle carry over to any other application characterized by:
  - fair workload balance across all the cores (this model is not designed for algorithms with limited parallelism or dependencies that lead to idle cores); and

  - iterative nature, where all iterations consume similar amounts of time and energy. This makes it easy to predict the power dissipation for the entire simulation based on a few iterations only.
- We perform a detailed power evaluation of MPDATA on a variety of ARM and Intel architectures, representative of both the low-power and high-performance extremes of the spectrum of current multicore processors.
- Finally, we validate the precision of our power models using the same collection of multicore CPUs, for both the MPDATA and the CG algorithms.

At this point, we note that, given the iterative process underlying the simulation performed by MPDATA as well as the algorithm for CG, we can adapt our power estimation approach to adjust the model during a few initial iterations. Furthermore, we recognize that, in most scenarios, energy (which equals the integral of power over a period of time) is the figure of merit, while power only plays a role for embedded systems due to system constraints. However, again due to the iterative nature of MPDATA and CG, it is possible to measure the execution time of the first iterations and combine this information with the power models to obtain accurate predictions for the energy consumption.

The rest of the paper is structured as follows. In Sect. 2, we offer a short review of a number of related works. In Sect. 3, we briefly introduce (the OpenMP version of) MPDATA, emphasizing its role in the framework of EULAG. (For a description of the CG algorithm, we simply refer the reader to [30] or any other basic linear algebra text.) Sections 4 and 5 contain the main contribution of our paper, namely the model to predict the power behavior of MPDATA under different VFS–CT configurations, the procedure for its calibration, and an experimental evaluation that validates the accuracy of the approach. We close the paper with a few concluding remarks in Sect. 6.

2 Related work

An increasing list of recent works have addressed distinct aspects related to energy consumption of scientific applications. Among these, PowerPack [11] and `pmlib` [2] are two frameworks to profile power and track energy consumption of serial and concurrent applications, using wattmeters combined with software microbenchmarks and utilities.

A more reduced number of works are focused on per-component power usage. In [4], the authors present a methodology to produce power models for multicore processors based on performance hardware counters. In [22], a methodology and a collection of microbenchmarks are proposed to analyze the energy efficiency via several operations that feature different access patterns and computation–memory ratios. In [23], the energy performance of both arithmetic operations and memory accesses is characterized, paying special attention to the different levels of the memory subsystem. In [16], the authors evaluate the power usage of arithmetic vs. memory operations and present two analytical models that can be applied to estimate the throughput and power usage of superscalar processors. In [17], the impact of data movement on the total energy consumption is characterized for the NAS parallel benchmarks and several scientific applications. In [6], the authors extend their energy roofline model to cap-

ture arithmetic and basic cache memory energy access costs as well as more elaborate random access patterns.

Some efforts are especially centered in power/energy consumption for sparse linear algebra and stencil computations. In particular, a systematic methodology to derive reliable time and power models for algebraic kernels is introduced in [21], employing a bottom-up approach. In [20], the authors devise a systematic machine learning algorithm to classify and predict the performance and energy costs of the sparse matrix–vector product, and in [5] this work is extended to consider the effect of VFS. Other works related to modeling sparse linear algebra operations can be found in [12,15,19].

Our paper relies on two simple power models and an associated calibration methodology that are easy to adapt, producing accurate estimates for any number of cores and voltage–frequency configuration from a reduced number of samples. Compared with many of the previous approaches, we do not rely on hardware counters or fine-grain measurements, only on the practical execution of a few steps of the target iterative application and the reads from coarse-grain wattmeters. In contrast to other approaches, we do not require samples for the full range of frequency configurations and/or number of threads. Instead, we only count on measurements for a few selected cases to derive the behavior of the application's power consumption. Our methodology for this purpose is formulated in the form of a simple iterative algorithm that, starting from the simple power models, converges rapidly to a stable solution, in the form of a table containing the sought-after estimations.

3 MPDATA overview and parallelization

3.1 Overview

MPDATA is the main module of the multiscale fluid model EULAG [24,31], an innovative solver in the field of numerical modeling of multiscale atmospheric flows. Concretely, MPDATA tackles the continuity equation describing the advection of a non-diffusive quantity Ψ in a flow field, namely:

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

where $\mathbf{V}$ is the velocity vector. The algorithm is positive defined, and by appropriate flux correction [31], it can also be monotonic. This is a desirable feature for advection of positive definite variables such as specific humidity, cloud water, cloud ice, rain, snow, aerosol particles, and gaseous substances.

MPDATA belongs to the group of forward-in-time algorithms [32,33], and the underlying simulation performs a sequence of time steps that call a particular algorithmic template each. The number of time steps depends on the type of simulated physical phenomenon, and it can exceed even a few millions, especially when considering MPDATA as a part of the EULAG model.

Each MPDATA time step consists of 19 parts, including 17 stencils and 2 data copies. A single MPDATA step requires 5 input matrices and returns a single output matrix that is necessary for the next time step.

The technical details of the MPDATA algorithm are widely described in literature. Those readers that are more interested in the specification of MPDATA and its implementation can refer to [29,34,36].

In previous works [26,35], we proved that MPDATA belongs to the group of memory-bound algorithms [13,36]. This makes MPDATA a challenging candidate for a technology such as dynamic VFS [16], which can reduce CPU energy consumption without a significant loss of performance.

3.2 Parallelization of MPDATA

The parallelization of the MPDATA code is designed for a single node containing multicore processors [27,28], employing OpenMP [14] within them [25]. Therefore, in this paper, we focus on investigating the power evaluation on a single node.

Listing 1 Stencil parallelization using OpenMP with thread affinity.

```
#pragma omp parallel
{
//Thread affinity
  cpu_set_t set;
  CPUZERO(&set);
  CPU_SET(omp_get_thread_num(), &set);
  pid_t tid = (pid_t)syscall(SYS_gettid);
  sched_setaffinity(tid, sizeof(set), &set);

//Stencil parallelization
#pragma omp for schedule(dynamic,16) collapse(2)
  for(int i=1; i<n-1; ++i)
    for(int j=1; j<m-1; ++j)
      for(int k=1; k<l-1; ++k) {
        ...
      }
  (...)
}
```

For the single-node case, OpenMP directives are applied to parallelize each stencil following the data-parallel programming model, as illustrated for one of the stencils in Listing 1. In that particular case, a dynamic schedule is applied with a grain size of 16, which yields higher performance than its static counterpart. Concretely, for this application, dynamic scheduling ensures a balanced distribution of the work load balance among the threads/cores. With this type of scheduling strategy, the first thread that completes its job takes the next 16 (in this case) chunks to execute. The grain size is selected empirically. An additional important technique consists in enforcing thread affinity, which prevents ineffective thread migrations between cores by the operating system. This can be applied by setting the environment variable `GOMP_CPU_AFFINITY=0-5` for the *gcc* compiler, or `OMP_PROC_BIND=true`

for OpemMP v.3.1. We can also set the thread affinity directly from the application code (see Listing 1). In our approach, we use the third method as it does not require to set any environment variables outside of the code. In this method, the `CPU_ZERO` macro clears the `set` variable, the `CPU_SET` macro adds the CPU to the set, the `syscall(SYS_gettid)` routine returns the thread id for the current thread, and finally the `sched_setaffinity` routine sets/migrates the thread `tid` to a core specified in the `set`.

To avoid potential overheads due to the re-allocation of threads, the directive `#pragma omp parallel` is used only once, at the beginning of the program, while the directive `#pragma omp for` is applied multiple times, once for each stencil.

4 Formulation of the power models

In this section, we introduce our methodology to estimate the power dissipation of MPDATA. For this purpose, we first describe our models that formulate power dissipation as a function of either the number of threads/cores or the VFS configuration. Next, we propose an iterative procedure to calibrate the proposed models. Our methodology is general and we expect it to carry over to other iterative HPC algorithms such as Jacobi and Krylov subspace solvers for sparse linear systems.

4.1 Power models

Our models estimate the power dissipation P of MPDATA as a function of the number of active cores (c) or the processor frequency (f), see [1,12]. To this end, we will, respectively, start from the following two equations:

$$P(c) = P^Y + P^U + P^C(c) \approx P^Y + P^U + P^C c, \tag{2}$$

and

$$P(f) = CV^2 f + P_S. \tag{3}$$

Here P^Y is the power dissipated by the system components except the CPU (e.g., DDR RAM chips, and motherboard), P^U is the power corresponding to the uncore elements of the processor (e.g., last-level cache, memory controllers, and core interconnect), P^C is the power consumed by the cores (including in-core cache levels, floating-point units, and branch logic), C is the capacitance, P_S is the static power, and V is the voltage.

Considering P as a function of c, we transform Eqn. (2) into

$$P(c) = \alpha_1 c + \alpha_2, \tag{4}$$

where we still assume that the power dissipation depends linearly on the number of active cores.

An additional assumption for Eq. (3) is the relation between the static power and the square of the voltage: $P_S \propto V^2$ [1] and the possibility of accurately modeling power consumption as a (cubic) function of f on recent multicore architectures [3]. Taking these two considerations into account, we evolve Eq. (3) into:

$$P(f) = \beta_1 f^3 + \beta_2 f^2 + \beta_3. \tag{5}$$

In summary, the proposed power models consider a linear function for $P(c)$ and a polynomial function for $P(f)$. Since linear functions are special cases of polynomial functions, the conclusion is that the proposed power model is valid when considering $P(c)$ and $P(f)$ as polynomials.

4.2 Calibration of the power models

The input data for the power dissipation model are a reduced set of actual power data measures collected during the execution of the MPDATA algorithm in the target architecture. The execution configuration includes the number of cores c and the CPU frequency f for those particular executions. The output of the model is the set of power values for MPDATA for any combination of number of cores and frequency level supported by the CPU.

Before we introduce the technical details of the proposed algorithm for power derivation, we outline our approach with a simple example. Let us consider a table PVAL of size $S_f \times S_c$ (7×6 in our example, which means that we have $S_f = 7$ supported CPU frequencies and $S_c = 6$ CPU cores). Each cell (i, j) represents the power dissipation rate observed when using the ith frequency and jth cores for the execution, with $i \in [1, 7]$ and $j \in [1, 6]$. Consider that initially this table contains 6 measured power dissipation values only. This means that we need to estimate the $7 \cdot 6 - 6 = 36$ remaining values to complete the table. This case is shown in Fig. 1a.

In order to fill the table for PVAL, we interpolate the coefficients for a polynomial based on the real measured values. For this purpose, we first determine the columns or rows of the table that contain the largest number of filled cells (excluding those columns and rows that are already complete, as we do not need to estimate any value for them). In our example, columns $j = 2$ and 5 contain the largest number of already filled cells. Based on the values in these columns, we then derive the polynomials $P_2(f)$, $P_5(f)$ (via interpolation) and, using these polynomials, we estimate the values for the entire cells of these two columns, see Fig. 1b. We next repeat this step, but this time we select all the rows (as all of them contain two filled cells). We create the polynomials for each row based on the available values and evaluate the power dissipation for the remaining empty cells, see Fig. 1c. Note that, because of the regular distribution of the real measurements, this case converges to the solution in only two steps. Other patterns may require a larger number of steps.

The calibration procedure that computes the power estimates can be formally defined following the steps exposed in Algorithm 1. The procedure operates on the $S_f \times S_c$ PVAL array. The first step of Algorithm 1 initializes array PVAL which, initially, only contains real measured values of P; during the algorithm's execution, this table is

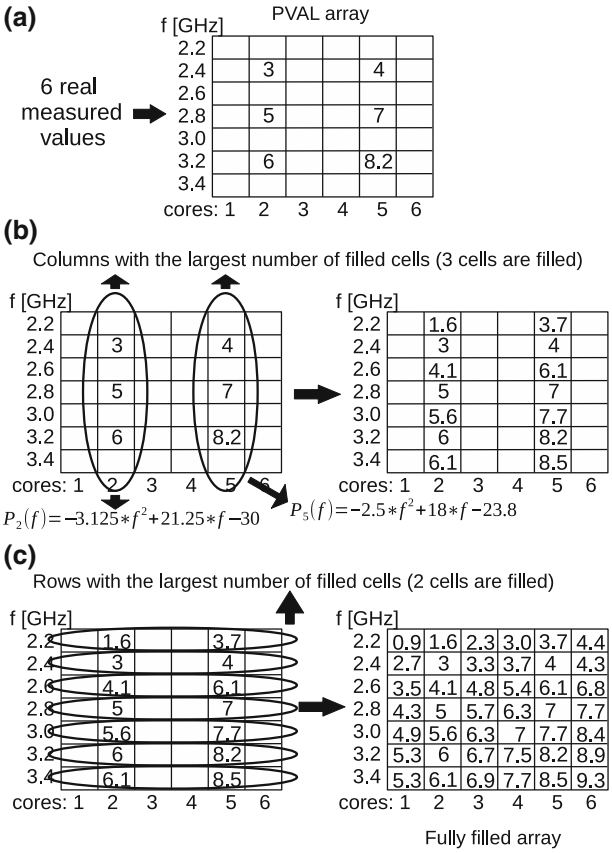

Fig. 1 Power estimation for each configuration of frequency and number of cores

completed with the estimated values for P. Arrays P_j^{col} and P_i^{row} contain the number of already filled values in every column and row of the PVAL array. The columns/rows containing the largest number of filled cells are used to approximate the remaining values.

Steps 3–6 count the elements within each column of PVAL different to -1 (i.e., already initialized). The result of this process determines the global column priority, which is selected to formulate a polynomial to estimate the values of P. Step 5 detects whether all the elements in a column are already filled. This means that the power prediction for this column is completed, and there is no need to approximate any elements within it. Steps 7–10 perform the row-wise process analogous to Steps 3–6.

Step 11 is responsible for selecting the row(s) or column(s) with the highest priority. When only one column/row has the highest priority, that column/row is the only one that is selected. If there is more than one column or row with the same priority, then all these columns or rows are selected. In Step 12, the algorithm leverages the information in the selected columns and rows in order to approximate the power dissipation rates; and in Step 13, it places this result into PVAL (the approximation is described later in

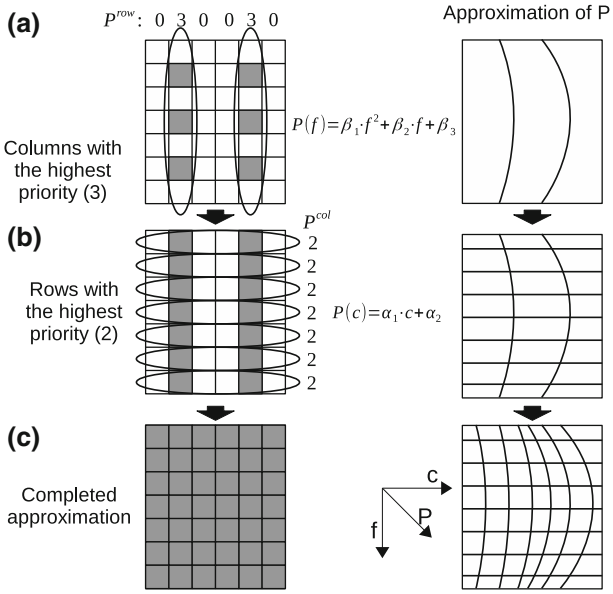

Fig. 2 Power approximation based on the proposed power algorithm

this section). The procedure in Steps 2–14 is repeated until all the elements of PVAL have different values to -1.

In order to gain a better understanding of the algorithm, Fig. 2 presents an example of the process that is carried out to approximate the power values, illustrating the evolution of the PVAL array and the polynomial approximations of P. For this example, we again consider a CPU with $S_c = 6$ cores that supports $S_f = 7$ CPU frequencies. The gray cells in array PVAL denote those values which are measured or already estimated. In this

Algorithm 1 Methodology for power approximation based on the set of measures.

1: Fill array PVAL with the available (real) values of P, the remaining cells set to -1
2: **repeat**
3: **for** $j \in [1..S_c]$ **do**
4: Set P_j^{col} to the number of cells, where $pval(i, j) \neq -1$
5: If $P_j^{col} = S_f$, then set $P_j^{col} = -1$
6: **end for**
7: **for** $i \in [1..S_f]$ **do**
8: Set P_i^{row} to the number of cells, where $pval(i, j) \neq -1$
9: If $P_i^{row} = S_c$, then set $P_i^{row} = -1$
10: **end for**
11: Find the column(s) from P_j^{col} or row(s) from P_i^{row} with the highest value, and store the number of these column(s)/row(s) in M_c and M_r, respectively
12: Use the values in column(s) M_c or row(s) M_r of PVAL to formulate polynomials of degree equal to $P_{M_c}^{col} - 1$ and $P_{M_r}^{row} - 1$
13: Set column(s) M_c or row(s) M_r of PVAL with the estimated values obtained via the formulated polynomials
14: **until** All the cells of PVAL have different values to -1

particular scenario, at the beginning, we start with 6 regularly distributed real measures of P (gray cells). Figure 2a shows that there are 2 columns with 3 (real) measures, and thus the priority of these columns is 3. This is the result of the first execution of Steps 3–6. Step 11 then selects columns 2 and 5 (but no rows) and employs the corresponding values from PVAL to derive polynomials of degree 2 that will be used to approximate the power for the two selected columns.

In the second iteration of Algorithm 1, corresponding to Fig. 2b, all the rows are selected as they have the highest priority (2). Then, the two elements per row that are already filled are employed to derive linear models (polynomials of degree 1) for that row. The completed approximation is shown in Fig. 2c, where all the elements are filled.

The real values of P (which are necessary to derive the estimates) are collected with a fixed pattern following a regular distribution across columns and rows. Let S_{mf} be the maximum number of the real measured values across a single column and S_{mc} be the maximum number of the real measured values across a single row. Taking into account the degree 2 for the polynomial $P(f)$ and the linear (polynomial) model for $P(c)$, we can expect reasonable results with $S_{mf} = 3$ measures across columns and $S_{mc} = 2$ across rows. This requires a total number S_m of real measures equal to $S_m = S_{mf} \cdot S_{mc} = 6$ for P. In order to select the indices I, J of the configurations for which the real power measures are collected, we employ the following equations:

$$I = \lfloor S_f/S_{mf} \rfloor \cdot i + \lfloor S_f/S_{mf}/2 \rfloor, \quad i \in 0, \ldots, S_{mf} - 1, \tag{6}$$

$$J = \lfloor S_c/S_{mc} \rfloor \cdot j + \lfloor S_c/S_{mc}/2 \rfloor, \quad j \in 0, \ldots, S_{mc} - 1. \tag{7}$$

In the example employed in Fig. 2, the row indices I are thus given by $\lfloor 7/3 \rfloor \cdot 0 + \lfloor 7/3/2 \rfloor = 0 + 1 = 1$, $\lfloor 7/3 \rfloor \cdot 1 + \lfloor 7/3/2 \rfloor = 2 + 1 = 3$, and $\lfloor 7/3 \rfloor \cdot 2 + \lfloor 7/3/2 \rfloor = 4 + 1 = 5$, while the column indices J are $\lfloor 6/2 \rfloor \cdot 0 + \lfloor 6/2/2 \rfloor = 0 + 1 = 1$ and $\lfloor 6/2 \rfloor \cdot 1 + \lfloor 6/2/2 \rfloor = 3 + 1 = 4$.

The procedure to define the function that approximates the power dissipation is shown in Fig. 3, where the polynomial for $P(f)$ is presented. The construction of the polynomial for $P(c)$ is straightforward. The approximation is always performed using the filled elements and the number of such elements determines the degree of the polynomial that is generated. When there is only one filled element, the polynomial is of degree 0, and the value is copied across the entire column or row in the PVAL array, see Fig. 3a.

5 Verification of the power estimation methodology

In this section, we validate the proposed power estimates. All tests with MPDATA were executed with double-precision arithmetic, a grid of size $160 \times 160 \times 160$, and 100 time steps. We provide additional evidence of the reliability of our approach using the CG algorithm for sparse linear systems. The loop body of this iterative solver is composed of a few calls to selected kernels for basic linear algebra operations, such as the sparse matrix-vector product, the dot (or inner) product of two vectors, and the "axpy"-vector update [30]. Our OpenMP-based parallel version of this algorithm stores the sparse

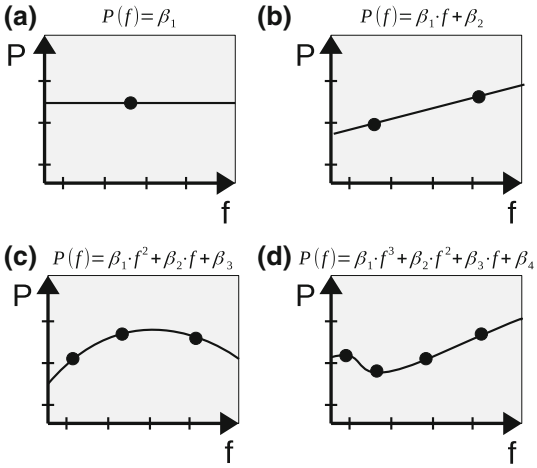

Fig. 3 Definition of the function type depending on the number of already filled elements across a single column of array PVAL

matrix in CSR format [30], employing a simple row-wise parallelization of the sparse matrix-vector product with a static schedule, and sequential versions of the vector kernels.

5.1 Experimental setup

Our goal is to approximate the power dissipation and validate the results of MPDATA (and the CG algorithm) on ARM- and Intel-based multicore architectures. Concretely, our models are calibrated and validated for the following systems:

- ARM.LITTLE: ARMv7 cluster with 4 low-power ARM Cortex-A7 cores (packaged in the Exynos E5422 system-on-chip);
- ARM.BIG: ARMv7 cluster with 4 ARM Cortex-A15 cores (also packaged in the Exynos E5422 system-on-chip);
- INTEL.SANDY1: Intel Xeon E5-2620 (SandyBridge) with 6 cores;
- INTEL.SANDY2: Two Intel Xeon E5-2620 CPUs (SandyBridge) with 6 cores each;
- INTEL.IVY: Intel Core i7-3930K CPU with 6 cores.

For the validation, we employ a representative subset of the frequencies supported by each architecture and any number of cores present in it. The ARMv7 CPUs can operate with a varied range of frequencies. For example, ARM.LITTLE supports $S_f = 8$ frequencies, corresponding to the values $\{0.25, 0.3, 0.35, \ldots, 0.6\}$ GHz; while ARM.BIG supports $S_f = 9$, defined as $\{0.8, 0.9, \ldots, 1.6\}$ GHz. The number of cores in both ARM CPUs is $S_c = 4$. The Intel CPUs support even wider sets of frequencies. For both INTEL.SANDY1 and INTEL.SANDY2, $S_f = 9$, corresponding to $\{1.2, 1.3, \ldots, 2.0\}$ GHz; but $S_c = 6$ for the former while $S_c = 12$ for the latter. Finally, for INTEL.IVY, $S_f = 15$, which include $\{1.2, 1.3, \ldots, 3.0, 3.2\}$ GHz, and $S_c = 6$. The properties of the target platforms are summarized in Table 1.

The power measurements are collected using the `pmlib` library [2]. For the ARM architectures, this tool obtains power data from internal power sensors in the

Table 1 Properties of the target platforms (min. and max. freq. are expressed in [GHz])

Prop.	ARM.LITTLE	ARM.BIG	INTEL.SANDY1	INTEL.SANDY2	INTEL.IVY
S_c	4	4	6	12	6
S_f	8	9	9	9	15
Min. freq.	0.25	0.8	1.2	1.2	1.2
Max. freq.	0.6	1.6	2.0	2.0	3.2
$S_f \times S_c$	32	36	54	108	90

big.LITTLE CPU clusters, Mali GPU, and memory DIMMs. For the Intel servers, `pmlib` collects power samples from a PDU, providing real power measures for the full server.

5.2 Testing the power modeling approach

The measured (real) values of power dissipated by INTEL.SANDY2 are presented in the top half of Table 2. We can observe there that the power dissipation rate increases with the CPU frequency and the number of cores. At this point, we remind that, according to our models, the power dissipation grows linearly with the number of cores and polynomially across the CPU frequency.

A comparison between these real measures and the estimates in the bottom half of the same table, obtained from the power modeling approach using only 6 real measures, shows a fair match between these values. This can be confirmed in the top plot of Fig. 4, which shows the measured values (left-hand side plot) and the relative errors corresponding to the individual estimates (right-hand side plot). Overall, the highest relative error encountered for this platform is around 5.0%, with this metric being much below 2% in many cases.

The remaining two rows of plots in Fig. 4 and the analogous ones in Fig. 5 display the same type of graphical analysis for the remaining architectures targeted in this study. This collection of figures shows a similar behavior for the real power consumption and validates the reliability of the modeling approach. Only for a few frequency configurations using 1 and 2 cores on ARM.BIG, we observe superior relative errors, in the range 6–9% in six of cases. For the rest of configurations of this particular platform, and for all of them in all 5 benchmarking systems, we relative error is bounded above by 5%, being much lower in most cases.

In order to quantify the differences using a single figure (metric), Tables 3 and 4 report the average error E_{AVG} of the estimated values of P for MPDATA, for each computing platform, computed as follows:

$$E_{\text{AVG}} = \frac{1}{S_f \cdot S_c} \sum_{i=1}^{S_f} \sum_{j=1}^{S_c} \left| 1 - \frac{P^p_{i,j}}{P^r_{i,j}} \right| \cdot 100\ [\%]. \tag{8}$$

Here $P^p_{i,j}$ is the predicted value of P power using j cores and the ith CPU frequency, while $P^r_{i,j}$ is the real power measure.

Table 2 Power dissipation (top) and estimates (bottom) [both in W] of MPDATA for a grid of size 160 × 160 × 160, and 100 time steps on INTEL.SANDY2

#Cores	CPU frequency (GHz)								
	1.2	1.3	1.4	1.5	1.6	1.7	1.8	1.9	2.0
1	122.0	123.0	125.0	126.4	128.1	129.7	131.8	133.6	135.1
2	125.7	127.7	130.1	131.4	133.4	137.9	140.8	142.2	144.1
3	129.8	131.8	136.9	138.5	140.7	143.0	145.2	148.7	150.2
4	136.4	138.5	140.0	142.9	145.4	148.6	151.3	154.1	155.9
5	138.8	141.5	144.0	146.4	148.9	152.5	155.4	159.4	162.8
6	141.1	144.5	146.8	149.4	153.4	157.4	160.2	165.2	169.4
7	144.3	146.0	149.3	152.9	155.7	160.1	163.9	167.0	171.4
8	146.3	151.4	153.2	164.7	160.0	166.9	172.3	175.8	180.0
9	148.5	152.7	160.7	162.8	165.2	170.0	175.7	179.9	186.0
10	155.6	**158.8**	163.2	166.3	**171.7**	173.7	180.5	**184.2**	188.4
11	157.1	161.1	164.6	168.3	173.3	177.6	180.6	185.6	190.6
12	158.9	163.6	167.0	170.9	173.9	179.2	184.9	190.3	195.1
1	127.7	128.3	129.3	130.6	132.3	134.2	136.4	139.0	*141.9*
2	130.6	131.7	133.1	134.7	136.6	138.8	141.3	144.0	147.1
3	133.6	135.1	136.8	138.8	141.0	143.5	146.1	149.1	152.2
4	136.6	**138.5**	140.6	142.9	**145.4**	148.1	151.0	**154.1**	157.4
5	139.5	141.9	144.4	147.0	149.8	152.7	155.8	159.1	162.5
6	142.5	145.2	148.1	151.1	154.2	157.4	160.7	164.1	167.7
7	145.5	148.6	151.9	155.2	158.6	162.0	165.5	169.2	172.8
8	148.4	152.0	155.6	159.3	162.9	166.7	170.4	174.2	178.0
9	151.4	155.4	159.4	163.3	167.3	171.3	175.3	179.2	183.2
10	154.4	**158.8**	163.1	167.4	**171.7**	175.9	180.1	**184.2**	188.3
11	157.3	162.1	166.9	171.5	176.1	180.6	185.0	189.3	193.5
12	160.3	165.5	170.6	175.6	180.5	185.2	189.8	194.3	198.6

The values in the bold are those that were selected as the input for the power estimation algorithm. The value in the italic identifies the configuration with the largest relative error in the estimate

Let us focus on the results for MPDATA. Table 3 shows that the highest accuracy is achieved for INTEL.IVY, which attains an average error $E_{\text{AVG}} = 0.36\%$ when employing only 6 measures to calibrate the models. As could be expected, a more reduced number of measures yield less accurate estimates for all platforms. However, the errors are still reasonable for the Intel-based platforms even when only 4 real measures are employed. The lowest accuracy is achieved for the ARM systems as, in these systems-on-chip, the relation between the CPU frequency and voltage is not linear for the lower extreme values of the frequency.

A similar behavior is detected for the CG algorithm, see Table 4. For this particular case, though, the application interleaves sequential and parallel phases, which explains the higher average error rates. However, these errors are still quite satisfactory when six real samples are employed, being below 2% in all cases except INTEL.IVY.

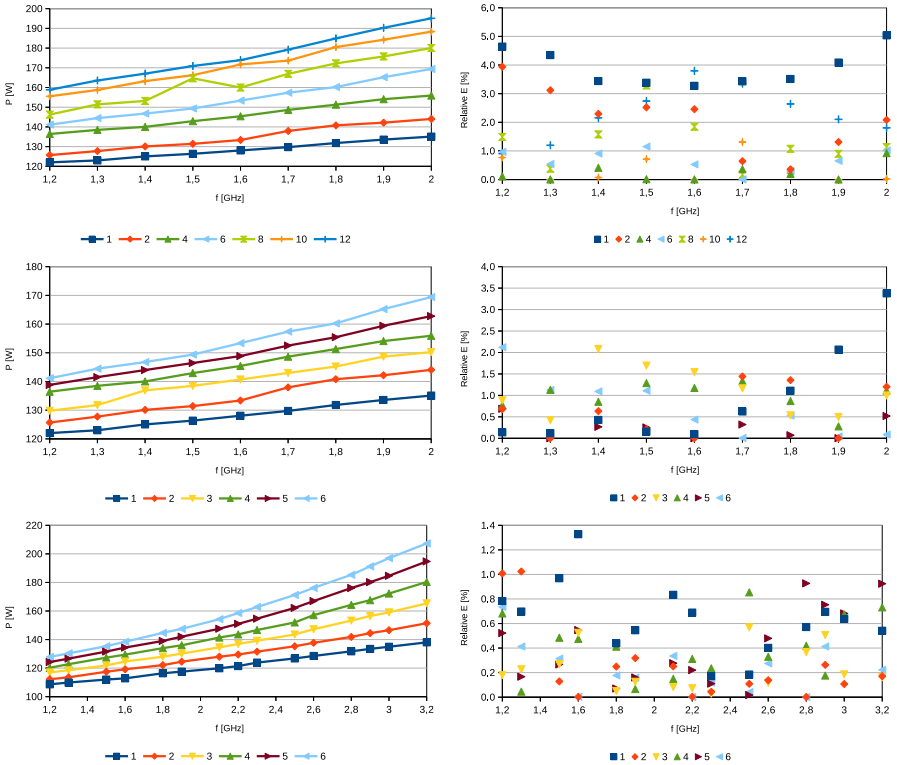

Fig. 4 Real power dissipation (*left*) and relative error for the power estimates (*right*) for INTEL.SANDY2 (*top*), INTEL.SANDY1 (*middle*) and INTEL.IVY (*bottom*)

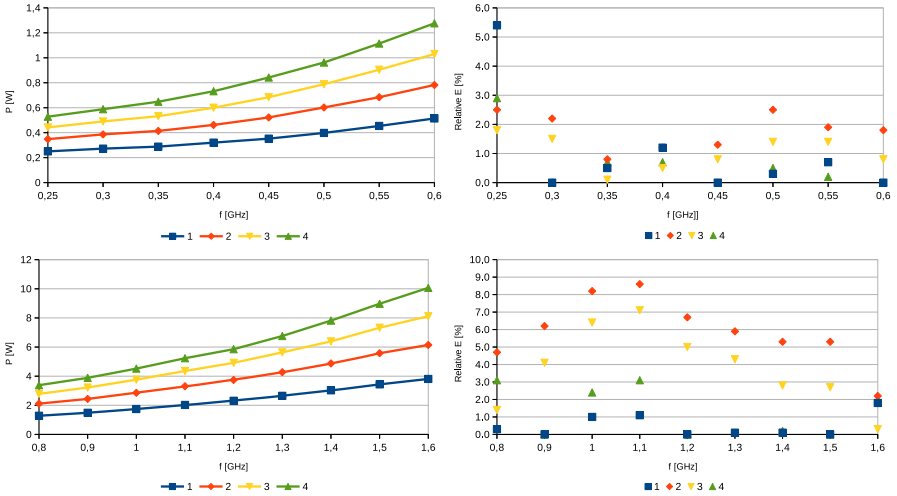

Fig. 5 Real power dissipation (*left*) and relative error for the power estimates (*right*) for ARM.LITTLE and ARM.BIG (*top*, and *bottom*, respectively)

Table 3 Average error [%] for MPDATA as a function of the number S_m of real measures

S_m	ARM.LITTLE	ARM.BIG	INTEL.SANDY1	INTEL.SANDY2	INTEL.IVY
3	29.83	34.74	5.95	8.43	8.68
4	6.43	4.24	0.71	1.42	1.13
5	3.95	3.45	0.70	1.42	0.97
6	1.11	2.84	0.66	1.32	0.36

Table 4 Average error (%) for the CG algorithm as a function of the number S_m of real measures

S_m	ARM.LITTLE	ARM.BIG	INTEL.SANDY1	INTEL.SANDY2	INTEL.IVY
3	29.24	37.73	6.18	9.30	22.88
4	6.59	4.29	1.51	2.12	9.89
5	2.79	3.63	1.30	2.04	9.59
6	1.54	2.17	1.24	1.35	4.90

6 Conclusions and future work

In this paper, we have proposed and validated a power consumption model for the MPDATA simulation and CG algorithm on a variety of ARM and Intel CPUs. We expect the model to carry over to other iterative algorithms where the execution time can be linearly approximated based on a few initial steps of the process. By combining the estimated power values with approximations of the execution time, we can also provide accurate prediction of the energy consumption of the algorithm.

Considering VFS and CT, our results confirm the model hypothesis: for all the target platforms, power dissipation increases linearly with the number of cores and as a polynomial of the CPU frequency.

The power estimates for MPDATA on INTEL.IVY report the average error of just 0.36% when employing only 6 measures to derive 84 values of power. The most challenging CPU is INTEL.SANDY2, which requires power estimations for more than 100 different configurations. Nevertheless, in this case we also obtained accurate estimates, with an average error for MPDATA of 1.32% only. The lowest accuracy for MPDATA is achieved for the ARM CPUs, where the relation between the CPU frequency and voltage is not linear for the lower extreme values of frequency.

All the assumptions and methodology used to develop the proposed model are considered independently on the MPDATA algorithm. Furthermore, they are also validated using an OpenMP version of the CG algorithm for the iterative solution of sparse linear systems. This supports the applicability the power model to any other application that features a good workload balance across the processing cores. Our future work will focus on extending the proposed approach to other architectures, including clusters of nodes. An important direction of future work will also to take into account energy modeling for non-iterative algorithms.

References

1. Aliaga JI, Barreda M, Dolz MF, Martín AF, Mayo R, Quintana-Ortí ES (2014) Assessing the impact of the CPU power-saving modes on the task-parallel solution of sparse linear systems. Clust Comput 17(4):1335–1348
2. Alonso P, Badia RM, Labarta J, Barreda M, Dolz MF, Mayo R, Quintana-Ortí ES, Reyes R (2012) Tools for power-energy modelling and analysis of parallel scientific applications. In: 41st International Conference on Parallel Processing—ICPP, pp 420–429
3. AnandTech Forums (2011) Power-consumption scaling with clockspeed and Vcc for the i7-2600K. http://forums.anandtech.com/showthread.php?t=2195927
4. Bertran R, Gonzalez M, Martorell X, Navarro N, Ayguade E (2013) A systematic methodology to generate decomposable and responsive power models for CMPs. IEEE Trans Comput 62(7):1289–1302
5. Catalán S, Malossi ACI, Bekas C, Quintana-Ortí ES(2016) The impact of voltage–frequency scaling for the matrix-vector product on the "ibm power8". In: Proceedings of 22nd International Euro-Par Conference, Lecture Notes in Computer Science. Springer. To appear
6. Choi J, Dukhan M, Liu X, Vuduc R (2014) Algorithmic time, energy, and power on candidate "hpc" compute building blocks. In: 2014 IEEE 28th international parallel and distributed processing symposium, pp 447–457
7. Curtis-Maury M, Blagojevic F, Antonopoulos C, Nikolopoulos D (2008) Prediction-based power-performance adaptation of multithreaded scientific codes. IEEE Trans Parallel and Distrib Syst 19(10):1396–1410
8. Dennard R, Gaensslen F, Rideout V, Bassous E, LeBlanc A (1974) Design of ion-implanted MOSFET's with very small physical dimensions. IEEE J Solid State Circuits 9(5):256–268
9. Duranton M, De Bosschere K, Cohen A, Maebe J, Munk H (2015) HiPEAC vision 2015. https://www.hipeac.org/publications/vision/
10. Elnozahy E, Kistler M, Rajamony R (2003) Energy-efficient server clusters. In: Power-aware computer systems second international workshop, PACS 2002, Lecture Notes in Computer Science, vol. 2325, pp 179–197
11. Ge R, Feng X, Song S, Chang HC, Li D, Cameron KW (2010) PowerPack: energy profiling and analysis of high-performance systems and applications. IEEE Trans Parallel Distrib Syst 21:658–671
12. Hager G, Treibig J, Habich J, Wellein G (2012) Exploring performance and power properties of modern multicore chips via simple machine models. Concurr Comput Pract Exp 28(2):189–210
13. Hager G, Wellein G (2011) Introduction to high performance computing for scientists and engineers. CRC Press, Boca Raton
14. Home - OpenMP (02/2017). http://www.openmp.org/
15. Karakasis V, Goumas G, Koziris N (2011) Exploring the performance-energy tradeoffs in sparse matrix-vector multiplication. In: Workshop on emerging supercomputing technologies (WEST)—ICS'11
16. Keramidas G, Spiliopoulos V, Kaxiras S (2010) Interval-based models for run-time DVFS orchestration in superscalar processors. In: Proceedings of the 7th ACM International Conference on Computing Frontiers (CF), pp 287–296
17. Kestor G, Gioiosa R, Kerbyson DJ, Hoisie A (2013) Quantifying the energy cost of data movement in scientific applications. In: IEEE international symposium on workload characterization (IISWC), pp 56–65
18. Lucas R (2014) Top ten Exascale research challenges. http://science.energy.gov/~/media/ascr/ascac/pdf/meetings/20140210/Top10reportFEB14.pdf
19. Malkowski K (2008) Co-adapting scientific applications and architectures toward energy-efficient high performance computing. Ph.D. thesis, University Park, PA, USA
20. Malossi ACI, Ineichen Y, Bekas C, Curioni A, Quintana-Ortí ES (2014) Performance and energy-aware characterization of the sparse matrix-vector multiplication on multithreaded architectures. In:

Proceedings of 43rd International Conference on Parallel Processing (ICCP), Minneapolis (MN), USA, pp 139–148
21. Malossi ACI, Ineichen Y, Bekas C, Curioni A, Quintana-Ortí ES (2016) Systematic derivation of time and power models for linear algebra kernels on multicore architectures. Sustain Comput Inform Syst 7:24–40
22. Manousakis I, Nikolopoulos DS (2012) BTL: a framework for measuring and modeling energy in memory hierarchies. In: Proceedings of the 24th IEEE international symposium on computer architecture and high performance computing (SBAC-PAD), pp 139–146
23. Molka D, Hackenberg D, Schone R, Muller MS (2010) Characterizing the energy consumption of data transfers and arithmetic operations on x86-64 processors. In: Proceedings of the 1st International Green Computing Conference (IGCC), pp 123–133
24. Prusa JM, Smolarkiewicz PK, Wyszogrodzki AA (2008) EULAG, a computational model for multiscale flows. Comput Fluids 37(9):1193–1207
25. Rauber T, Runger G (2013) Parallel programming for multicore and cluster systems, 2nd edn. Springer, Berlin
26. Rojek K, Ciznicki M, Rosa B, Kopta P, Kulczewski M, Kurowski K, Piotrowski Z, Szustak L, Wojcik D, Wyrzykowski R (2015) Adaptation of fluid model EULAG to graphics processing unit architecture. Concurr Comput Pract Exp 27(4):937–957
27. Rojek K, Wyrzykowski R (2015) Parallelization of 3D MPDATA algorithm using many graphics processors. Lect Notes Comput Sci 9251:445–457
28. Rojek K, Wyrzykowski R (2017) Performance modeling of 3D MPDATA simulations on GPU cluster. J Supercomput 73(2):664–675
29. Rojek K, Wyrzykowski R, Kuczynski L (2016) Systematic adaptation of stencil-based 3D MPDATA to GPU architectures. Concurr Comput Pract Exp. doi:10.1002/cpe.3970
30. Saad Y (2003) Iterative methods for sparse linear systems. SIAM, New Delhi
31. Smolarkiewicz P (2006) Multidimensional positive definite advection transport algorithm: an overview. Int J Numer Methods Fluids 50:1123–1144
32. Smolarkiewicz P, Prusa J (2002) Forward-in-time differencing for fluids: simulation of geophysical turbulence. In: Drikakis D, Guertz B (eds) Turbulent flow computation. Springer, Dordrecht, pp 207–240
33. Szustak L, Halbiniak K, Kulawik A, Wrobel J, Gepner P (2016) Toward parallel modeling of solidification based on the generalized finite difference method using Intel Xeon Phi. Lect Notes Comput Sci 9573:411–412
34. Szustak L, Rojek K, Olas T, Kuczynski L, Halbiniak K, Gepner P (2015) Adaptation of MPDATA heterogeneous stencil computation to Intel Xeon Phi coprocessor. Sci Program 2015:14. doi:10.1155/2015/642705
35. Wyrzykowski R, Szustak L, Rojek K (2014) Parallelization of 2D MPDATA EULAG algorithm on hybrid architectures with GPU accelerators. Parallel Comput 40(8):425–447
36. Wyrzykowski R, Szustak L, Rojek K, Tomas A (2014) Towards efficient decomposition and parallelization of MPDATA on hybrid CPU-GPU cluster. Lect Notes Comput Sci 8353:434–444