Cluster Comput (2017) 20:2535–2549
DOI 10.1007/s10586-016-0686-2

Energy aware scheduling model and online heuristics for stencil codes on heterogeneous computing architectures

Milosz Ciznicki[1,2] · **Krzysztof Kurowski**[2] · **Jan Weglarz**[1]

Received: 11 March 2016 / Revised: 24 October 2016 / Accepted: 18 November 2016 / Published online: 29 November 2016

Abstract Performance of high-end supercomputers will reach the exascale through the advent of core counts in billions. However, in the upcoming exascale computing era it is important not only to focus on the performance, but also on scalability of fine-grained parallel applications, data locality and energy aware scheduling within the parallel code. In fact, parallel applications need to change even now by redesigning algorithms and data structures respectively to take advantage of the recent improvements in energy efficiency of heterogeneous computing hardware, including multicore processors and GPU accelerators. Over the next few years one of the biggest challenges for exascale will be the ability of parallel applications to fully exploit locality which will, in turn, be required to achieve expected performance and energy efficiency. Future highly parallel applications will have to deal with deep memory hierarchies taking into account energy cost in moving data off-chip. Therefore, they will have to apply new coordinated scheduling approaches to balance energy aware resource utilization and minimize work starvation during runtime. As new constraints and limits on memory bandwidth and energy will play a key role in high performance computing (HPC) in the future, more sophisticated and dynamic scheduling techniques will be needed and applied within the parallel code. In this paper we focus on an energy-aware distribution of the stencil workload on heterogeneous processors. Our analysis of energy and performance models focused on relevant class of stencil computations to explore the relationship between task scheduling algorithms and energy constraints. More precisely, we search for a schedule which minimizes the energy usage within a specified computation's deadline of the stencil workload on heterogeneous architectures. Since the problem is computationally intractable, we present an integer linear programming formulation for finding optimal schedules. As finding optimal schedules is time consuming we have developed four heuristics and tested them experimentally with respect to optimal solutions. In our work we focus on a single node configurations with heterogeneous processors. These configurations represent the state of the art multi- and many-core architectures.

Keywords Power and energy modelling · Performance analysis · Scheduling · Resource management · Stencil computations · GPUs · Many-core systems

1 Introduction

Stencil computations as relevant class of applications occur in many HPC codes on block-structured grids for modelling various physical phenomena, e.g. for computational fluid dynamics, geometric modelling, solving partial differential equations or image and video processing [1–5]. As computing time and memory usage grow linearly with the number of array elements in stencil computations our research targets highly parallel implementations of stencil codes together with task scheduling and optimization techniques taking into consideration energy cost and data locality [6–10]. We have proved during our experimental studies that recent changes introduced in heterogeneous computing hardware resulted in different performance and energy characteristics that are critical for highly efficient and scalable stencil computations [11]. As shown in [12,13], the overall performance of stencil

✉ Milosz Ciznicki
miloszc@man.poznan.pl

1 Institute of Computing Science, Poznan University of Technology, Piotrowo 2, 60-965 Poznan, Poland

2 Poznań Supercomputing and Networking Center, Jana Pawla II 10, 61-139 Poznan, Poland

computations is memory bound. One should note that many existing HPC architectures mainly focus on floating point performance [14]. However, only a partial and limited usage of the floating point units in a given computing architecture is possible today and may reduce energy cost without the performance degradation. Moreover, many latest improvements introduced in dynamic power management policies at the hardware level, e.g. dynamic voltage and frequency scaling (DVFS) or even switching off an entire unit block of a chip (clock gating), can lead to significant reduction in the energy required for memory-bound workloads. Advanced dynamic power management policies give new opportunities for scheduling tasks within the fine-grained parallel code as users are able to control the utilization of various functional units in heterogeneous computing hardware, e.g. turn on and off dynamically individual cores, change on-demand the frequency of a small processing and communication units or even put portions of cache memory at specific sleep states during runtime.

In our previous work [15] we performed an exhaustive evaluation of the key characteristics that have a relevant impact on the performance and energy usage of a stencil computation running on a certain processing unit. Based on these characteristics in this article, we present an energy-aware ILP model that distributes stencil computations to heterogeneous processors and minimizes the schedule energy cost while meeting the computation's deadline. The distribution of stencil computations is done on the blocks obtained from the decomposition of the computational domain. The computational domain is a Cartesian grid on which the stencil computations are defined. The optimization space of the model shows that the best strategy depends not only on load balancing the problem size between the processing units, the processing units specification, and the stencils employed, but also on detailed mapping of the communication dependencies of the blocks to the communication topology of respective processing units. No previous work has attempted to account for the time and energy simultaneously in the context of the distribution of the stencil computations between processing units. We also developed new heuristics that schedule example workloads in real time. The developed heuristics attempt to include the communication overhead in the distribution process. The described algorithms have been tested experimental using the state of the art multi- and many-core architectures. In our work we focus the experiments on a single node configurations with heterogeneous processors.

The paper is organized as follows. In Sect. 2 the related work is discussed. The key properties that have an influence on energy usage are defined in Sect. 3. The scheduling problem is introduced in Sect. 4. Performance and energy models are introduced in Sect. 5. Section 6 describes the integer linear programming (ILP) model. The dynamic scheduling policies are described in Sect. 7. Section 8 presents experiments using a 3D Laplacian stencil defined on different grid topologies using several CPU–GPU configurations. Section 9 concludes our experiments and presents a future work.

2 Related work

In general, considered stencil calculations perform global sweeps through data structures that are typically much larger than the capacity of the available data caches available within processing units. Additionally, accessing data in main memory within the hardware is not fast enough and we often have to deal with the traffic between local cache and main memory. Therefore, many researchers have already tried to exploit data locality in stencil computations by performing operations on cache-sized blocks of data after domain decomposition [16], after time decomposition [17] or proposed cache-aware optimisation algorithms on many-core modern processors [18].

In consequence, there exist frameworks that try to ease the implementation of the stencil calculations. The user writes single stencil code in a framework's specific language which during a compilation is translated to a target architecture. The frameworks distribute the computations to employ multiple processors. The distribution involves the decomposition of the Cartesian domain to overlapping blocks. The overlap, called halo region, is needed to correctly update the decomposed block on borders. Each block is updated by a single processor. The minimal size of the overlap depends on the stencil pattern. The stencil pattern defines which neighbouring points are used during stencil computations. For example, Physis [19] uniformly decomposes the global domain over all the accelerators as instructed by a user-controllable parameter. The user has to experimentally determine which decomposition provides the highest performance. The framework focuses only on the GPU architecture. Similarly, work in [1] utilises a simple decomposition method with uniform partition where each processor and accelerator receives blocks of the same size. On the other hand, authors in [20] provide a method that allows programmers to partition the data contiguously between CPU and GPU within a single node. Unlike our work, their approach does not allow to find an optimal distribution of the domain between heterogeneous architectures in terms of time and energy costs. What is more, there is a lack of careful analyses of stencil optimizations and performance modelling connecting specific properties such as communication and locality with architectural time and energy costs.

Moreover, performance and energy models for modern heterogeneous computing architectures incorporating specialized processing capabilities should be flexible and extendable to explore recent properties of heterogeneous hardware units. A good example is the roofline model which allows a programmer to model, predict, and analyse an individual kernel performance given an architecture communication and computation capabilities [21]. In this approach an application is modelled simply by the ratio of useful operations to memory operations. The roofline model can predict the performance of a simple von Neumann architecture with two levels of memory as well as the more complex design with a multi-level memory hierarchy. It has been successfully used to model the performance of many applications on the multi-core and many-core processors [22]. Recently, it has been extended to model the energy consumption in GPUs [23]. In the new model the authors have assumed that each operation has a fixed energy cost and a fixed data movement cost while the constant energy cost is linear in time. The constant power depends on both a hardware and an algorithm and includes both static and leakage power management. However, the proposed model does not include dynamic power management by charging and discharging gate capacitance. The authors assumed that time per work (arithmetic) operation and time per memory operation are estimated with the hardware peak throughput values, whereas the energy cost is estimated using a linear regression based on real experiments. Another set of extensions to the roofline model have been proposed in [24] to model energy on dual multi-core CPU with three-level cache hierarchy. In this approach the dynamic power management was modelled as a second degree polynomial, based on real benchmark data, that scales linearly with the number of active cores up to the saturation point. The authors assumed that the dynamic power depends quadratically on the frequency. In the saturation point the energy to solution grows with the number of used cores, that is proportional to dynamic power, while the time to solution stays constant. In our article we are providing two examples of architectures CPUs and GPUs. However, the presented model can be utilized with other architectures as well, for instance Intel Xeon Phi or ARM. Antoher example is an energy model presented in [25] to evaluate the cost of parallel algorithms for GPU. Based on the energy model they propose the method for the energy scalability to easy the selection of the optimal number of blocks.

3 Stencil properties

In our previous work [15] we experimentally discovered the key characteristics that have a relevant impact on the performance and energy usage of a stencil computation running on a certain processing unit (PU). We tested the performance and energy usage of an example 3D Laplacian stencil on eight core Intel Xeon E5-2670@2.6GHz CPU and Kepler K20m GPU using multiple of frequency and voltage pairs, called P-states. Firstly, the maximum performance can be reached with a lower number of cores than available. Secondly, to minimize the energy usage it is more important to reduce the frequency than the number of cores used. What is more, in case of CPU, DRAM may use up to 60% of the energy. Thus, the data movement consumes the most of the power. Finally, the lowest energy usage may be reached with not the maximum performance. To summarise the analysis, the stencil computation $u \in \mathcal{T}$, called task, is described by the following parameters:

1. The number of arithmetic operations per grid point $W_{u,p}$ on a processor p,
2. The number of required bytes to update a grid point $Q_{u,p}$ on a processor p,
3. The block dimensions $d_u = [d_u^x, d_u^y, d_u^z]^T$.

The processor $p \in \mathcal{P}$ has following properties:

1. The set of available frequencies $\mathcal{F}=\{f_{p1}, f_{p2}, \ldots, f_{pn}\}$,
2. The set of available cores $\mathcal{C} = \{c_{p1}, c_{p2}, \ldots, c_{pm}\}$,
3. The set of states $\mathcal{L} = \{(f, c) : f \in \mathcal{F} \wedge c \in \mathcal{C}\}$, where $l \in \mathcal{L}$ is a selected state,
4. The sustained bandwidth to the main memory $b_{p,l}$ in bytes per second based on the state l.
5. The performance $h_{p,l}$ in the floating-point operations per second based on the state l.

4 Problem formulation

As showed in the previous section that the data locality has the highest influence on the energy usage, it has encouraged us to focus our research on a stencil workload scheduling using heterogeneous computing architectures to minimize the energy usage while meeting the computation's deadline. The scheduling problem is defined by a set $\mathcal{P}$ of m processors and a workload $\mathcal{T} = T_1, T_2, ..., T_n$ of n dependent tasks.

A considered workload represents a stencil defined on a structural grid. Each point on a grid is updated with a strict pattern, see Fig. 1. The pattern defines which neighbouring points are used during a stencil computation. A single update of the whole grid is called a timestep. In our approach

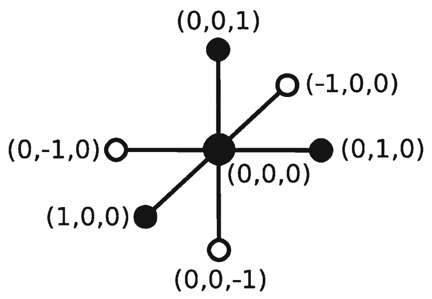


Fig. 1 3D Laplacian stencil

we focus on an explicit method where a current timestep is updated by using values of the grid points from a previous timestep. The considered heterogeneous hardware includes unrelated processing units (PUs) and the same stencil computation takes different execution times on them. Based on our experimental studies we distinguished two different unrelated processing units: central processing units (CPUs) and graphic processing units (GPUs).

The workload contains the set of dependent tasks. The block decomposition of the structural grid updated by the stencil forms the workload of tasks with the communication dependencies. A task represents a single block of the decomposed grid. We assume that the grid is decomposed on equally sized blocks. We assume that a given task may be processed by a single processing unit at a time and each processing unit may execute several tasks.

The tasks are represented by a directed graph defined by a tuple $G = (V, E)$ where V denotes the set of tasks and E represents the set of edges. For simplicity we assume that the task $T_u = u$ and the processor is depicted by p. Each edge $(u, v) \in E$ defines a communication between the tasks $u, v \in V$. The communication load $d_{u,v}$ on the edge (u, v) depicts the number of grid cells exchanged between tasks. The model assumes a fully connected network of heterogeneous processors with heterogeneous communication links. If tasks u and v are executed on different processors $p, k \in \mathcal{P}$, they cause the time $t^e_{p,k}$ and the energy $e^e_{p,k}$ penalty required to exchange a single grid cell between the processors p and k. If both tasks are scheduled on the same processor, then the communication time and the communication energy are equal to zero. The computation load w_u describes the number of grid cells provided by the task u. The computation time and the energy cost to update the single cell on the processor p are represented by $t^c_{u,p,l}$ and $e^c_{u,p,l}$ respectively; see (3), (4). The idle power P^{idle}_p depicts the power used when no computations are executed on processor p. The memory size m_p represents the maximum number of grid cells that can be computed on processor p. The total communication time and the total communication energy to exchange all data are represented by t^e and e^e respectively. Total execution time t^t indicates how much time it takes to finish the whole workload. The execution deadline t^d denotes the time by which all tasks have to be finished. The objective is to determine a schedule such that the total energy cost is minimized and deadline t^d is not exceeded.

5 Performance and energy models

Detailed analysis of the performance and the energy usage of the stencil computations on two unrelated processing units resulted in the following formulation of the performance model. Computation time $t^c_{u,p,l}$ of task u on processor p with state l is estimated as follows:

$$O_{u,p} = W_{u,p} * d^x_u * d^y_u * d^z_u \quad (1)$$

$$B_{u,p} = Q_{u,p} * d^x_u * d^y_u * d^z_u \quad (2)$$

$$t^c_{u,p,l} = max(O_{u,p}/h_{p,l}, B_{u,p}/b_{p,l}) \quad (3)$$

where $O_{u,p}$ is the number of arithmetic operations executed and $B_{u,p}$ is the number of bytes transferred.

The energy model assumes that each arithmetic operation as well as the memory operation consumes some energy:

$$e^c_{u,p,l} = e^{op}_{u,p} * O_{u,p} + e^{byte}_{u,p} * B_{u,p} + P0_{u,p,l} * t^c_{u,p,l} \quad (4)$$

Variables $e^{op}_{u,p}$, $e^{byte}_{u,p}$ approximates the energy usage of stencil operations. For simplicity, it is assumed that arithmetic operations, i.e. additions, multiplications, subtractions and divisions, consume the same amount of energy. Additionally, the energy usage also depends on an instruction set used, thus for the highest performance the CPU implementation of the stencil uses the vector extensions. $P0_{u,p,l}$ is a constant power consumed by the processor P_p based on the state l. The coefficients $e^{op}_{u,p}$, $e^{byte}_{u,p}$ and $P0_{u,p,l}$ are approximated with a linear regression. Table 1 shows estimated values of the energy cost for the double precision floating point operation and the transfer of a single byte of data. For CPU and GPU the cost to transfer a single byte of data is 5.2x and 6x more expensive than the floating point operation, respectively. What is more, both floating point and memory operations are 5x more expensive on CPU than on GPU. Figure 2 shows that the constant power grows linearly with the increasing number of cores using different P-states.

Table 1 Energy coefficients for the CPU and GPU architectures

Platform	CPU Xeon E5-2670@2.60GHz (pJ)	GPU Kepler K20m (pJ)
e^{op}	327	54
e^{byte}	1700	324

6 Optimal model

This section presents a method based on ILP (ILP) to obtain the optimal solution of the energy minimization problem. In particular, this method is developed to have a reference for the heuristics described in Sect. 7. Before going into details let us introduce basic definitions from graph theory [26].

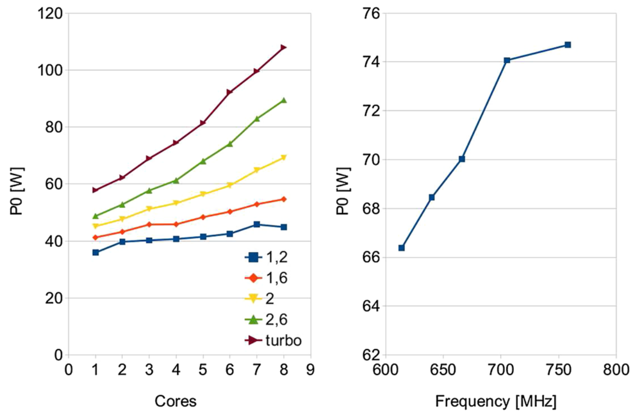


Fig. 2 Constant power P0: *left* CPU, *right* GPU

6.1 Multiplicity

Two edges $uv, st \in E$ are parallel if $\{u, v\} = \{s, t\}$. The multiplicity of an edge $uv \in E$ is the number of edges parallel to uv:

$$\mu_{uv} = |\{st \in E : \{uv\} = \{st\}\}| \tag{5}$$

6.2 Incidence

Two edges $uv, st \in E$ are incident if $\{u, v\} \cap \{s, t\} \neq \emptyset$ and edge $uv \in E$ is also called incident to its both end nodes u and v. The set of edges incident at a node u is denoted by $\delta_G(u)$:

$$\delta_G(u) = |\{e \in E : \{e\} \cap \{u\} \neq \emptyset\}| \tag{6}$$

The number of edges incident to a node u is the degree of this node in G and will be denoted by $deg_G(u)$. For $U \subseteq V$ the set of all edges with exactly one endpoint in U is denoted by $\delta(U)$. In a directed graph the edges in E are assumed to be ordered pairs and are described as $(u, v) \in E$. For a node $u \in V$ in a directed graph $G = (V, E)$ we define $\delta_G^+(u) : \{(v, w) \in E : v = u\}$ as the set of edges leaving the node u and $\delta_G^-(u) : \{(v, w) \in E : w = u\}$ as the set of edges entering the node u.

6.3 Maximum degree and maximum multiplicity

Maximum degree and maximum multiplicity of a graph are defined as

$$\Delta G = \max_{v \in V} deg_G(v) \tag{7}$$
$$\mu G = \max_{e \in E} \mu_G(e) \tag{8}$$

A graph with $\mu(G) = 1$ that contains no parallel edges is called simple. Graphs with maximum multiplicity at least 1 are called multigraphs and denoted by M.

6.4 Chromatic index

An edge colouring of a graph $G = (V, E)$ is a map $c : E \rightarrow C$ which assigns to each edge $e \in E$ a colour $c(e) \in C$ such that no two incident edges receive the same colour. The minimal cardinality of the colour set C for which such a mapping exists is called the chromatic index of the graph and denoted by $\chi'(G)$.

6.5 ILP solution

Our method was inspired by the model proposed in [27]. The idea is to decompose the scheduling problem to two parallel subproblems. At first, the tasks are mapped to processors to minimize the maximum number of grid cells placed on each processor. Secondly, the number of communication rounds is minimized by employing an edge colouring model. The communication is executed in parallel between different pairs of processors in stencil computations. However, each processor can initiate a single communication link with another processor at a time. As a result, we have to employ several communication rounds to exchange all data. The number of communication rounds directly influence the communication time t^e, as each round costs some time. The reason for selecting the ILP solution is that the edge colouring problem is NP-hard [28,29]. For the task scheduling model the set of edges is mapped to processors, where each edge (u, v) may be mapped to a single processor p or might be placed between two different processors p and k. In the first case, both endpoints u and v are mapped to p. In the second case, u is mapped to p and v to k or u is mapped to k and v to p. For each edge the *slots* $(p, k) \in \mathcal{P} \times \mathcal{P}$ are provided and it is required that each edge must be assigned to exactly one slot. If edge e is assigned to slot (p, k) then it starts in p and ends in k. If $p = k$ then e lies completely on p and is intra-processor, in all other cases it is inter-processor. For the minimization of the number of communication rounds the edge colouring model is used. If the graph $G = (V, E)$ of tasks is mapped to the complete graph K_m of m processors to form a new multigraph M_p, then each edge in M_p receives at least as many colours as its multiplicity demands and incident edges do not receive the same colour. What is more, an edge can only receive a colour that is used.

Variables. For the integer programming model we introduce the following variables:

- (*Edge to slot* $x_{e,p,k}$) The binary variable for all $e \in E$ and $(p, k) \in \mathcal{P} \times \mathcal{P}$ equals 1 if and only if edge e is mapped to slot (p, k), and 0 otherwise,
- (*Edge to colour* $y_{e,c}$) For all $e \in K_n$ and $c \in C$, where $C = \{0, \ldots, \Delta(G)+\mu(G)-1\}$, the binary variable equals 1 if edge e receives colour c in M_p, and 0 otherwise. The most simple choice for the number of potential colours

to colour multigraph is $|E|$. However, we can choose a smaller set based on [30,31] that for any multigraph $G = (V, E)$ the chromatic index is $\chi'(G) \leq \Delta(G) + \mu(G)$,

- (*Colour is used* z_c) For all $c \in C$ the binary variable equals to 1 if a colour c is used in the edge colouring of M_p and 0 otherwise,
- (*Number of grid cells* c_p) This integer variable depicts for each processor p the number of allocated grid cells,
- (*Processor idle time* t_p^{idle}) This variable for each processor p with the state l represents the idle time.
- (*Total execution time* t^t) This variable indicates how much time it takes to finish the whole workload,
- (*Energy used for communication* e^e) This variable represents the total energy used for the inter-processor communication.

Constraints The model employs several types of constraints:

- (*Map edge to single slot*) Each edge $e \in E$ must be mapped to exactly one slot,

$$\sum_{(p,k)\in\mathcal{P}\times\mathcal{P}} x_{e,p,k} = 1 \tag{9}$$

- (*Restrict slots*) Mapping edge uv to slot (p, k) restricts the slots to which edges in $\delta(uv)$ can be mapped. Edges in $\delta^+(u)$ must start in p and edges in $\delta^-(u)$ must end there. Likewise, edges in $\delta^+(v)$ must start in k and edges in $\delta^-(v)$ must end there:

$$\sum_{k\in\mathcal{P}} x_{uv,p,k} - \sum_{k\in\mathcal{P}} x_{f,p,k} = 0 \tag{10}$$

$$\sum_{k\in\mathcal{P}} x_{uv,p,k} - \sum_{k\in\mathcal{P}} x_{f,k,p} = 0 \tag{11}$$

$$\sum_{k\in\mathcal{P}} x_{uv,k,p} - \sum_{k\in\mathcal{P}} x_{f,p,k} = 0 \tag{12}$$

$$\sum_{k\in\mathcal{P}} x_{uv,k,p} - \sum_{k\in\mathcal{P}} x_{f,k,p} = 0 \tag{13}$$

These constraints are for all $p \in \mathcal{P}$ and $uv \in E$, where $f \in \delta^+(u)$ is for (10), $f \in \delta^-(u)$ is for (11), $f \in \delta^+(v)$ is for (12), $f \in \delta^-(v)$ is for (13),

- (*Control the number of grid cells*) This constraint controls the number of the grid cells allocated for each $p \in P$. The sum of grid cells mapped to processor p is given by

$$\sum_{uv\in E}\sum_{k\in\mathcal{P}} (w_u/deg_u * x_{uv,p,k} + w_v/deg_v * x_{uv,k,p}) \leq c_p \tag{14}$$

for all $p \in P$,

- (*Number of colours not less than multiplicity*) This constraint requires that each edge in M_p receives at least as many colours as its multiplicity demands. Each edge models time required to exchange single grid cell between processors p and k:

$$\sum_{uv\in E} \left(x_{uv,p,k} * t^e_{p,k} * d_{uv} + x_{uv,k,p} * t^e_{k,p} * d_{uv}\right) \leq y_{p,k,c}\ _{c\in C} \tag{15}$$

- (*Incident edges receive different colours*) This requires that incident edges do not receive the same colour in the edge colouring of M_p and that an edge can only receive a colour that is used:

$$\sum_{k\neq p,k\in\mathcal{P}} y_{p,k,c} \leq z_c \tag{16}$$

- (*Restrict memory capacity for each processor*) This constraint restricts the number of grid cells allocated for each processor p:

$$c_p \leq m_p \tag{17}$$

- (*Control energy used for communication*) The sum of energy used for the inter-processor communication is depicted as

$$\sum_{uv\in E}\ \sum_{p\neq k,(p,k)\in\mathcal{P}\times\mathcal{P}} x_{uv,p,k} * d_{uv} * e^e_{p,k} \leq e^e \tag{18}$$

- (*Control execution time*) These two constraints calculate the total execution time t^t using the maximum value from the computation and the communication time. As described in Sect. 4 the computation and the communication are done in parallel:

$$t^c_{u,p,l} * c_p \leq t^t \tag{19}$$

$$\sum_{c\in C} z_c \leq t^t \tag{20}$$

- (*Control processor's idle time*) This constraint controls the idle time for all $p \in \mathcal{P}$:

$$t^t - t^c_{u,p,l} * c_p \leq t^{idle}_p \tag{21}$$

- (*Deadline*) This inequality restricts the execution time:

$$t^t \leq t^d \tag{22}$$

Table 2 shows the number of variables and constraints that formulate the ILP model.

Optimization objective. Finally, the objective of the model is to minimize the energy cost:

Table 2 Number of variables and constraints that formulate the ILP problem

	ILP
x variables	$\lvert E \times \mathcal{P} \times \mathcal{P}\rvert$
y variables	$\lvert C \times \mathcal{P} \times \mathcal{P}\rvert$
z variables	$\lvert C\rvert$
x constraints	$\lvert E\rvert$
y constraints	$\lvert \mathcal{P} \times \mathcal{P}\rvert$
z constraints	$\lvert \mathcal{P}\rvert$

$$\sum_{p \in P}\left(e^{c}_{u,p,l} * c_p + t^{idle}_{p} * P^{idle}_{p}\right) + e^{e} \tag{23}$$

Algorithm 1 Balancing load

```
1: procedure BALANCINGLOAD
       Input: A set $\mathcal{T}$ of tasks and the processor set $\mathcal{P}$.
       Output: A mapping m.
2:     for $i = 0, ..., |\mathcal{P}| - 1$ do
3:         Set $s = 0$
4:         while $s \leq w_V * r_i / \sum_{p \in \mathcal{P}} r_p$ do
5:             Remove the first task $u$ from $\mathcal{T}$
6:             Set $m(u) = p_i$ and $s = s + w_u$
7:         end while
8:     end for
9: end procedure
```

7 Heuristics

Taking into account the relevance of an energy efficiency issues in the next generation of the high-end supercomputers in this section we introduce new heuristics. In our approach we consider energy aware stencil workload scheduling on heterogeneous architectures with two following objectives:

– minimize the energy usage,
– load balance of the tasks to meet the deadline.

7.1 Simple

This strategy is focused on balancing the load between processors, and does not take into account the communication dependencies. These heuristics are usually quite simple and fast as they act online on the workload.

7.1.1 Balancing load

The algorithm distributes tasks to processors while attempting to keep the maximal load small and not to exceed the deadline. This strategy is called Balancing Load, see Algorithm 1. We start with processor p_0 and assign tasks to this processor until its size is at least $w_V * r_i / \sum_{p \in P} r_p$. Then we move to the next processor and repeat the procedure. The limit $w_V * r_i / \sum_{p \in P} r_p$ stems from the fact that in a prefect balancing of tasks there is one processor that has this many grid cells. This limit is a modification of a limit $w_V / |P|$ for homogeneous processor, as we consider the speed r_p of each processor. The time complexity of the algorithm is $O(|V|)$ to assign all tasks to processors. The algorithm is sensitive to the order in which the tasks and the processors are selected.

Algorithm 2 Minimize degree

```
1: procedure MINIMIZE DEGREE
       Input: A set $\mathcal{T}$ of tasks and the processor set $\mathcal{P}$.
       Output: A mapping m.
2:     for $u \in \mathcal{T}$ do
3:         $deg(u) = deg_G(u)$
4:     end for
5:     for $i = 0, ..., |\mathcal{P}| - 1$ do
6:         Set $s = 0$
7:         while $s \leq w_V * r_i / \sum_{p \in \mathcal{P}} r_p$ do
8:             Find $u' = argmin\{deg(u) : u \in \mathcal{T}\}$.
9:                                 ▷ If there are multiple tasks that attain this
10:                  ▷ minimum pick the one with smallest computational
                     load.
11:            Remove task $u'$ from set $\mathcal{T}$
12:            Set $m(u') = p_i$ and $s = s + w_{u'}$
13:            for $v \in N(u') \cap \mathcal{T}$ do
14:                $deg(v) = deg(v) - \mu_{u'v}$
15:            end for
16:        end while
17:    end for
18: end procedure
```

Algorithm 3 Minimize multicut

```
1: procedure MINIMIZE MULTICUT
       Input: A set $\mathcal{T}$ of tasks and the processor set $\mathcal{P}$.
       Output: A mapping m.
2:     for $u \in \mathcal{T}$ do
3:         $deg(u) = |N(u) \cap \mathcal{T}|$
4:     end for
5:     for $i = 0, ..., |\mathcal{P}| - 1$ do
6:         Set $s = 0$
7:         while $s \leq w_V * r_i / \sum_{p \in P} r_p$ do
8:             Find $u' = argmin\{deg(u) : u \in \mathcal{T}\}$.
9:                                 ▷ If there are multiple tasks that attain this
10:                  ▷ minimum pick the one with smallest computational
                     load.
11:            Remove task $u'$ from set $\mathcal{T}$
12:            Set $m(u') = p_i$ and $s = s + w_{u'}$
13:            for $v \in N(u') \cap \mathcal{T}$ do
14:                $deg(v) = deg(v) - 1$
15:            end for
16:        end while
17:    end for
18: end procedure
```

Algorithm 4 Accumulate neighbours

```
 1: procedure ACCUMULATE NEIGHBOURS
        Input: A set $\mathcal{T}$ of tasks and the processor set $\mathcal{P}$.
        Output: A mapping m.
 2:     Set $i = 0$
 3:     while $\mathcal{T} \neq \emptyset$ and $i \leq |\mathcal{P}|$ do
 4:         for $u \in \mathcal{T}$ do
 5:             $N(u) = 0$
 6:         end for
 7:         Set $s = 0$
 8:         while $s \leq w_V * r_i / \sum_{p \in \mathcal{P}} r_p$ and $\mathcal{T} \neq \emptyset$ do
 9:             if $s \leq f * w_V * r_i / \sum_{p \in \mathcal{P}} r_p$ then
10:                 Find $u' = argmax\{N(u) : u \in \mathcal{T}\}$.
11:             else
12:                 Find $u' = argmin\{deg_G(u) - N(u) : u \in \mathcal{T}, N(u)\}$.
13:             end if
14:                                     ▷ If there are multiple tasks that attain this
15:                 ▷ minimum pick the one with smallest computational load.
16:             Remove task $u'$ from set $\mathcal{T}$
17:             Set $m(u') = p_i$ and $s = s + w_{u'}$
18:             For each $v \in \mathcal{T}$ that is adjacent to $u'$ set $N(v) = N(v)+1$
19:         end while
20:     end while
21: end procedure
```

7.2 Advanced

Algorithms described in this section attempt to include communication overhead in the scheduling process. They try to find such a schedule that the resulting multigraph yields a small chromatic index. Since finding the chromatic index is an NP-complete problem [28], the algorithms employ different approximation methods to minimize it.

7.2.1 Minimize degree

In this algorithm task u with the lowest number of unmapped edges is assigned to the current processor p. Based on the equations $\chi'(G) \leq \Delta(G) + \mu(G)$ and $\chi'(G) \leq \lfloor 3 * \Delta(G)/2 \rfloor$ the chromatic index $\chi'(G)$ of any multigraph G depends on the max degree. Thus, when task u is assigned to processor p, then each incident edge to this task not mapped to p increases the current degree of p by one. The neighbours of the task u that are mapped to another processor $k \neq p$ also increase the degree of p, but they are not considered in this algorithm. Therefore, the array $deg(u)$ is used to keep for each task u the number of unmapped edges. The number by which the degree of processor p would increase if the task u was mapped to it. If two tasks have the same number of unmapped edges, then the task with the smallest computational load is selected. In other words, the number of additional grid points by which computational load on the processor p would exceed the perfect load $w_V * r_i / \sum_{p \in P} r_p$ if the task u was mapped to p. The running time of Algorithm 2 is $O(|V^2|)$. The time needed to find the task with the smallest computational load takes $O(|V|)$, whereas the while loop is executed $O(|V|)$ times.

7.2.2 Minimize multicut

In this algorithm the chromatic index for a multigraph is estimated based on the complete number of edges $|E|$. The previous Algorithm 2 is modified to obtain a minimal multicut. To achieve this task u with the smallest number of the unscheduled neighbours is found to be mapped on the current processor p. In line 3 the $deg(u)$ is initialized with the number of the unscheduled neighbours. Each scheduled task u' decreases the $deg(v)$ for each unscheduled neighbour v of u'. The time complexity of the algorithm is equal to $O(|V^2|)$.

7.2.3 Accumulate neighbours

In this algorithm the unmapped task u with the highest number of neighbours on the currently selected processor p is chosen. This policy tries to yield most of the communication edges of the grid graph intra-processor. The array N records the number of neighbours the task u has on the processor p. In line 10 the task with the most neighbours on the processor p is selected. However, at the end of the inner while loop (line 12) when the processor p is almost full different strategy is employed. The task u connected to the subgraph mapped to p with a minimum number of neighbours not on p is selected. To recognise when the processor is almost full the load factor $f \in [0, 1]$ is introduced. While $s \leq f * w_V * r_i / \sum_{p \in \mathcal{P}} r_p$ the tasks with the maximum number of neighbours on the current processor p are selected, whereas $s \geq f * w_V * r_i / \sum_{p \in \mathcal{P}} r_p$ the tasks with the minimum number of neighbours not on p are picked. When no unmapped task is adjacent to the tasks currently mapped to p, then the task with the maximum degree is preferred. Additionally, for the strategy defined in line 12, the task with the minimum degree among the unmapped ones is selected. To find the task in lines 10 and 12 takes $O(|V|)$ time. The while loops are executed $|V|$ times, thus the whole Algorithm 4 runs in time $O(|V^2|)$.

8 Experimental studies

8.1 Simulation setup

To validate our models a new simulator has been designed and implemented to calculate the total execution time, the energy usage and the number of communication rounds (colours). The simulator is initialized with the following data:

Table 3 Properties of the simulated grids

Name	#Blocks	#Edges	Block size	Grid size
Cuboid	128	608	262144	512x256x256
Sphere	128	704	262144	512x256x256

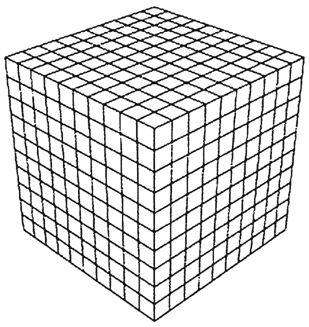


Fig. 3 Cuboid

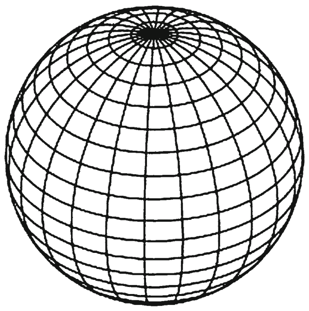


Fig. 4 Sphere

1. a text file with workload dependency graph,
2. a text file with processor topology,
3. the type of scheduling strategy used: ILP or heuristic.

The simulation instances include the two different real world simulation grids. These grids are related to the weather simulations problems. The connection topology of points on each grid is defined by a 3D Laplacian stencil depicted in Figure 1.

Table 3 outlines the properties of the test instances. The first grid called Cuboid (Figure 3) was used to simulate a decaying turbulence of a homogeneous incompressible fluid. Whereas, the second grid called Sphere (Figure 4) was used as a benchmark for the atmospheric circulation models. The connections in the horizontal direction for the Sphere grid are periodical. Figure 5 shows the example of the decomposed Cuboid grid with the connection dependencies. Each number represents the block id that later is mapped to specific processor. To analyse the quality of the ILP model and the heuristics the grids are mapped to single node with the three different configurations of the processors: CPU–CPU, CPU–GPU and 2xCPU-2xGPU. The simulated CPU is Intel Xeon

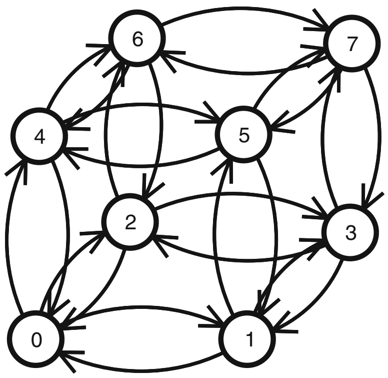


Fig. 5 Graph of stencil tasks with the connection dependencies

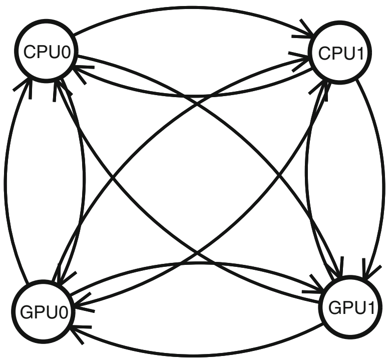


Fig. 6 Graph of processors

Table 4 Parameters setup

Parameter	CPU	GPU
$t^e_{p,k}$	$1 \times 10^{-9} s$	$1 \times 10^{-9} s$
$e^e_{p,k}$	$1.36 \times 10^{-7} J$	$1.36 \times 10^{-7} J$
$t^c_{u,p,l}$	$8.33 \times 10^{-10} s$	$1.06 \times 10^{-10} s$
$e^c_{u,p,l}$	$2.9 \times 10^{-8} J$	$5.5 \times 10^{-9} J$
P^{idle}_p	10 W	30 W
$P0_{u,p,l}$	90 W	74 W

E5-2670 Sandy Bridge 8 core processor and GPU is Nvidia Kepler K20m. Figure 6 presents the node topology with four processors. In all algorithms the CPU and GPU frequencies are set to default values. GPU operates at 705MHz of the core clock and 2600 MHz of the memory clock. Whereas, CPU operates at 2.6 GHz of the core clock. The parameters used in all test runs are shown in Table 4. The values of the parameters are obtained based on methodology described in Sect. 3 and 5.

8.2 Simulation results

First, we show the results for the ILP model, see Tables 5 and 6, where the first column presents the configurations used.

Table 5 ILP on Cuboid with all configurations

Arch.	t^d[ms]	#Col.	E [J]	e^c (%)	e^e (%)	t^t [ms]
CPU–CPU	27.90	0	3.76	100	0	27.90
	26.97	20	4.86	77	23	26.81
	14.00	32	5.27	66	34	13.95
CPU-GPU	3.58	0	0.48	100	0	3.58
	3.46	20	1.70	35	65	3.44
	3.34	28	2.23	30	70	3.33
	3.22	34	2.65	29	71	3.22
2xCPU–2xGPU	3.58	0	0.62	100	0	3.58
	3.46	20	1.73	36	64	3.44
	3.34	28	2.16	28	72	3.33
	3.22	32	2.26	21	79	1.79

Table 6 ILP on sphere with all configurations

Arch.	t^d[ms]	#Col.	E [J]	e^c (%)	e^e (%)	t^t [ms]
CPU–CPU	26.97	30	5.41	69	31	26.81
	25.92	40	5.94	63	37	25.28
	25.02	48	6.37	58	42	24.41
	24.13	56	6.78	54	46	22.67
	14, 00	64	7.05	49	51	13.95
CPU–GPU	3.60	0	0.48	100	0	3.58
	3.46	30	2.26	26	74	3.44
	3.35	38	2.79	24	76	3.33
	3.23	46	3.32	23	77	3.22
2xCPU-2xGPU	3.58	0	0.62	100	0	3.58
	3.46	30	2.28	27	73	3.44
	3.34	38	2.72	22	78	3.33
	3.22	46	3.16	19	81	3.22
	3.11	54	3.60	16	84	3.11
	2.99	56	3.69	16	84	2.91
	2.66	64	4.05	12	88	1.79

Table 7 Heuristics on cuboid with the CPU–CPU configuration

Algorithm	#Edg.	#Col.	E [J]/gap (%)	t^t [ms]/gap (%)
Alg_1-RD	1224	306	20.53/289.49	13.95/0.00
Alg_1-IJK	256	64	**7.05/33.81**	13.95/0.00
Alg_1-JIK	256	64	**7.05/33.81**	13.95/0.00
Alg_1-KIJ	256	64	**7.05/33.81**	13.95/0.00
Alg_2	256	64	**7.05/33.81**	13.95/0.00
Alg_3	256	64	**7.05/33.81**	13.95/0.00
Alg_4-0.1	256	64	**7.05/33.81**	13.95/0.00
Alg_4-0.2	256	64	**7.05/33.81**	13.95/0.00
Alg_4-0.3	256	64	**7.05/33.81**	13.95/0.00
Alg_4-0.4	256	64	**7.05/33.81**	13.95/0.00
Alg_4-0.5	256	64	**7.05/33.81**	13.95/0.00
Alg_4-0.6	256	64	**7.05/33.81**	13.95/0.00
Alg_4-0.7	256	64	**7.05/33.81**	13.95/0.00
Alg_4-0.8	256	64	**7.05/33.81**	13.95/0.00
Alg_4-0.9	256	64	**7.05/33.81**	13.95/0.00
Alg_4-1.0	256	64	**7.05/33.81**	13.95/0.00

The best results in terms of energy efficiency are given in bold

Each configuration is simulated with different deadlines. The deadline is provided as an input parameter. We reduce its value to the point where the ILP model is not able to generate the feasible solution. The next columns provide the number of colours used in a multigraph, the total energy consumed, the energy used for computations, the energy used for communication and the time elapsed. The number of colours used in the graph colouring provide information about the number of the communication rounds employed. As the results show the decreasing deadlines improve the computation times, however they increase the energy usage. This is especially true for the heterogeneous configurations where the energy usage grows up to 7x from the extended deadline to the shortest one. Shorter deadline forces usage of the next processing unit, and as a result, it requires more energy to communicate. For example, for the *2xCPU-2xGPU* configuration 88% of energy is consumed by the communication. Therefore, for this reason it is important to efficiently distribute the stencil tasks to reduce the number of the communication rounds between the processing units. However, as we can see it is beneficial to use the heterogeneous configurations, as we switch from the *CPU-CPU* configuration to the *2xCPU-2xGPU* configuration both the computation time and energy costs decrease by 87 and 57%, respectively, for the Cuboid grid. Similarly, the computation time for the Sphere grid decreases by 87% whereas the energy usage decreases by 42%. Higher energy usage for the Sphere grid is caused by the periodic connections of tasks on the I and J boundaries. For single node configurations we can notice that the maximum computation time $t^c_{u,p,l}$ among the processing units is a bottleneck for the total execution time t^t. Although, we can expect that for the multi-node configurations the limiting factor will be the communication time.

The quality of all the four heuristics described in Sect. 7 is presented. The obtained results are presented in Tables 7, 8, 9, 10, 11 and 12, where Algorithm 1 is tested with four different sorting orders of the tasks: random (RD), IJK indices, JIK indices and KIJ indices. The tasks can be order by the grid indices depending on their location within the grid. This order may have influence on the number of edges mapped between different processors.

For Algorithm 4 the first column in Tables contains the value of the load factor f, that depicts when the processor is almost full. This algorithm is tested with different values of this parameter. The second to last columns show the number

Table 8 Heuristics on cuboid with the CPU–GPU configuration

Algorithm	#Edg.	#Col.	E [J]/gap (%)	t^t [ms]/gap (%)
Alg_1-RD	504	126	7.85/195.85	3.48/8.32
Alg_1-IJK	192	48	3.51/32.29	3.48/8.32
Alg_1-JIK	160	40	**3.06/15.52**	3.48/8.32
Alg_1-KIJ	160	40	**3.06/15.52**	3.48/8.32
Alg_2	192	48	3.51/32.29	3.48/8.32
Alg_3	192	48	3.51/32.29	3.48/8.32
Alg_4-0.1	224	56	3.96/49.07	3.48/8.32
Alg_4-0.2	200	50	3.62/36.49	3.48/8.32
Alg_4-0.3	200	50	3.62/36.49	3.48/8.32
Alg_4-0.4	200	50	3.62/36.49	3.48/8.32
Alg_4-0.5	200	50	3.62/36.49	3.48/8.32
Alg_4-0.6	200	50	3.62/36.49	3.48/8.32
Alg_4-0.7	192	48	3.51/32.29	3.48/8.32
Alg_4-0.8	192	48	3.51/32.29	3.48/8.32
Alg_4-0.9	176	44	3.29/23.90	3.48/8.32
Alg_4-1.0	160	40	**3.06/15.52**	3.48/8.32

The best results in terms of energy efficiency are given in bold

Table 9 Heuristics on cuboid with the 2xCPU–2xGPU configuration

Algorithm	#Edg.	#Col.	E [J]/gap (%)	t^t [ms]/gap (%)
Alg_1-RD	520	318	22.00/870.23	1.74/−2.68
Alg_1-IJK	576	128	8.86/290.69	1.74/−2.68
Alg_1-JIK	480	112	7.52/231.75	1.74/−2.68
Alg_1-KIJ	480	112	7.52/231.75	1.74/−2.68
Alg_2	576	128	8.86/290.69	1.74/−2.68
Alg_3	576	128	8.86/290.69	1.74/−2.68
Alg_4-0.1	568	116	8.75/285.77	1.74/−2.68
Alg_4-0.2	504	92	7.85/246.48	1.74/−2.68
Alg_4-0.3	584	124	8.97/295.60	1.74/−2.68
Alg_4-0.4	480	100	7.52/231.75	1,74/−2.68
Alg_4-0.5	480	100	7.52/231.75	1.74/−2.68
Alg_4-0.6	480	100	7.52/231.75	1.74/−2.68
Alg_4-0.7	472	98	7.41/226.84	1.74/−2.68
Alg_4-0.8	456	92	7.19/217.01	1.74/−2.68
Alg_4-0.9	432	84	**6.85/202.28**	1.74/−2.68
Alg_4-1.0	432	84	**6.85/202.28**	1.74/−2.68

The best results in terms of energy efficiency are given in bold

Table 10 Heuristics on sphere with the CPU–CPU configuration

Algorithm	#Edg.	#Col.	E [J]/gap(%)	t^t [ms]/gap (%)
Alg_1-RD	1408	352	23.09/227.39	13.95/0.00
Alg_1-IJK	256	64	**7.05/0.00**	13.95/0.00
Alg_1-JIK	256	64	**7.05/0.00**	13.95/0.00
Alg_1-KIJ	512	128	10.62/50.53	13.95/0.00
Alg_2	256	64	**7.05/0.00**	13.95/0.00
Alg_3	256	64	**7.05/0.00**	13.95/0.00
Alg_4-0.1	256	64	**7.05/0.00**	13.95/0.00
Alg_4-0.2	256	64	**7.05/0.00**	13.95/0.00
Alg_4-0.3	256	64	**7.05/0.00**	13.95/0.00
Alg_4-0.4	256	64	**7.05/0.00**	13.95/0.00
Alg_4-0.5	256	64	**7.05/0.00**	13.95/0.00
Alg_4-0.6	256	64	**7.05/0.00**	13.95/0.00
Alg_4-0.7	256	64	**7.05/0.00**	13.95/0.00
Alg_4-0.8	256	64	**7.05/0.00**	13.95/0.00
Alg_4-0.9	256	64	**7.05/0.00**	13.95/0.00
Alg_4-1.0	256	64	**7.05/0.00**	13.95/0.00

The best results in terms of energy efficiency are given in bold

Table 11 Heuristics on sphere with the CPU–GPU configuration

Algorithm	#Edg.	#Col.	E [J]/gap (%)	t^t [ms]/gap (%)
Alg_1-RD	576	144	9.53/186.63	3.48/8.32
Alg_1-IJK	256	64	4.40/32.50	3.48/8.32
Alg_1-JIK	192	48	**3.51/5.70**	3.48/8.32
Alg_1-KIJ	320	80	5.29/59.31	3.48/8.32
Alg_2	256	64	4.40/32.50	3.48/8.32
Alg_3	256	64	4.40/32.50	3.48/8.32
Alg_4-0.1	296	74	4.96/49.25	3.48/8.32
Alg_4-0.2	304	76	5.07/52.61	3.48/8.32
Alg_4-0.3	304	76	5.07/52.61	3.48/8.32
Alg_4-0.4	312	78	5.18/55.96	3.48/8.32
Alg_4-0.5	320	80	5.29/59.31	3.48/8.32
Alg_4-0.6	328	82	5.40/62.66	3.48/8.32
Alg_4-0.7	336	84	5.51/66.01	3.48/8.32
Alg_4-0.8	336	84	5.51/66.01	3.48/8.32
Alg_4-0.9	320	80	5.29/59.31	3.48/8.32
Alg_4-1.0	320	80	5.29/59.31	3.48/8.32

The best result in terms of energy efficiency is given in bold

of edges in the returned scheduling, the number of colours used in the obtained multigraph and the objective values for the energy and time. Gap is defined as a difference between the optimal solution o' and the solution o^* returned be the algorithm:

$$gap(o^*, o') = (o^* - o')/o' \qquad (24)$$

The optimal solution with the shortest deadline is selected as a base for the comparison. In other words, the results are compared to the feasible solution with the lowest possible computational time and minimal energy obtained by the ILP model. Tables from 7 to 12 show the time t^t is all the same. All heuristics are based on the idea of the load balancing where the computations of the tasks are well balanced between processors. For each grid configuration the final schedule obtains the same computation time. The communication time between heuristics is different, as the obtained schedules provide different number of the communication

Table 12 Heuristics on sphere with the 2xCPU-2xGPU configuration

Algorithm	#Edg.	#Col.	E [J]/gap (%)	t^t [ms]/gap (%)
Alg_1-RD	1664	350	24.01/492.79	1.74/−2.68
Alg_1-IJK	704	160	10.64/162.77	1.74/−2.68
Alg_1-JIK	704	160	10.64/162.77	1.74/−2.68
Alg_1-KIJ	800	760	11.98/195.77	1.74/−2.68
Alg_2	704	160	10.64/162.77	1.74/−2.68
Alg_3	704	160	10.64/162.77	1.74/−2.68
Alg_4-0.1	744	164	11.20/176.52	1.74/−2.68
Alg_4-0.2	728	158	10.97/171.02	1.74/−2.68
Alg_4-0.3	712	152	10.75/165.52	1.74/−2.68
Alg_4-0.4	712	156	10.75/165.52	1.74/−2.68
Alg_4-0.5	720	158	10.86/168.27	1.74/−2.68
Alg_4-0.6	720	158	10.86/168.27	1.74/−2.68
Alg_4-0.7	704	158	10.64/162.77	1.74/−2.68
Alg_4-0.8	704	158	10.64/162.77	1.74/−2.68
Alg_4-0.9	672	144	**10.19/151.77**	1.74/−2.68
Alg_4-1.0	672	144	**10.19/151.77**	1.74/−2.68

The best results in terms of energy efficiency are given in bold

rounds. The communication time is smaller than the computation time and both are done in parallel. As a result, the communication time do not influence the time t^t. However, the number of communication rounds strongly influence the energy consumption. Tables 7 and 10 show that almost all heuristics except for $Alg\_1 - RD$ are able to schedule stencil tasks with close to the optimal solution for homogeneous hardware configurations with two processors. What is more, the results show that the heuristics that target at the balanced load provide good solutions for simple configurations with two processors. The balancing load algorithm Alg_1 produces an efficient distribution depending on the sorting order of the input tasks. The order based on JIK indices minimizes the number of the communication rounds for both grids with two processors. With four processors it is beneficial to use the heuristics that take into account the communication penalty. The algorithm Alg_4 provides good schedules for the four processors as it tries to yield most of the communication edges of the task graph intra-processor. The quality of schedule depends on the load factor, which determines when to switch the mapping from the task with the most neighbours on the current processor to the task with a minimum number of neighbours not on the current processor. Take as an example the $5x4x5$ gird with 100 blocks distributed on a node with single CPU and two GPUs where the 3D Laplacian stencil is empolyed. Figure 7a shows the schedule from Alg_1 with the best performing JIK order. The blocks are distributed horizontally according to the JIK order. Figure 7b shows the output from Alg_4 with the load factor equal to 0.9. The blocks scheduled to CPU are distributed vertically within the computational grid whereas the blocks scheduled

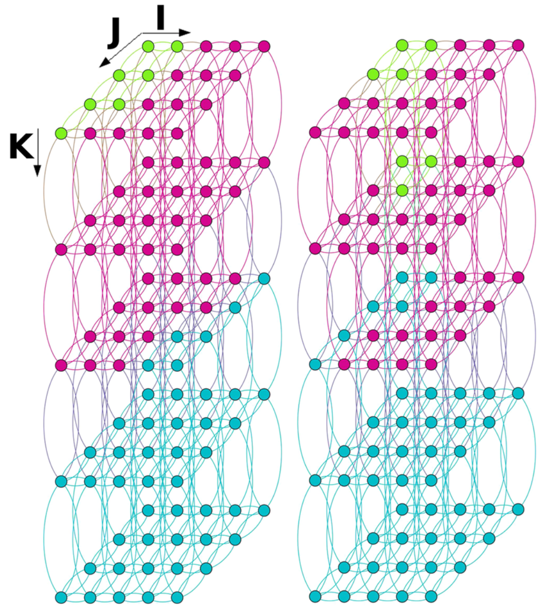


Fig. 7 Comparison of schedule between Alg_1 and Alg_4. The colours represent the scheduling of blocks to the processors: *red* GPU00, *blue* GPU01 and *green* CPU00. *Left* output from Alg_1, *right* output from Alg_4

Table 13 The average execution time (us) of the investigated ILP model and heuristics for the 2xCPU–2xGPU configuration

	ILP	Alg1	Alg2	Alg3	Alg4
Cuboid	3118×0^6	32	78	52	55
Sphere	281836×10^6	33	98	57	65

to GPUs are distributed horizontally. Alg_4 and the rest of the algorithms (Alg_2 and Alg_3) are able to mix the spatial distribution of the blocks. The energy cost is $4.65J$ and $4.43J$ for Alg_1 and Alg_4 respectively. 5% of the energy is saved by reducing the number of communication rounds.

The presented heuristics may be applied to the distribution of the stencil computations between the processing units defined on the Cartesian grids. These grids may be 2D or 3D with or without periodic boundaries.

Table 13 shows the average exeuction time of the investigated ILP model and heuristics for the 2xCPU–2xGPU configuration with previously described grid setups. As one can notice, the time to find the optimal solutions is seven orders of magnitude larger than the time of heuristics.

8.3 Verification of energy model

This section contains the experimental comparison of the energy usage model used in the simulator with the real measurements. Figures 8 and 9 present the comparison of energy usage between the proposed model and the real measurements. The results are obtained for the Intel Xeon processor

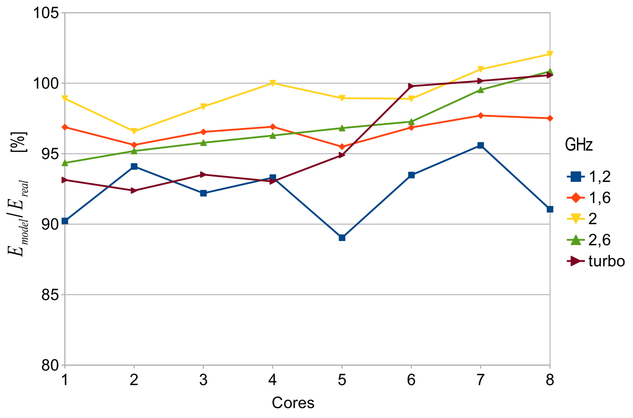


Fig. 8 Comparison of accuracy (%) between the proposed model and the real measurements on the Intel Xeon E5-2670@2.6GHz processor

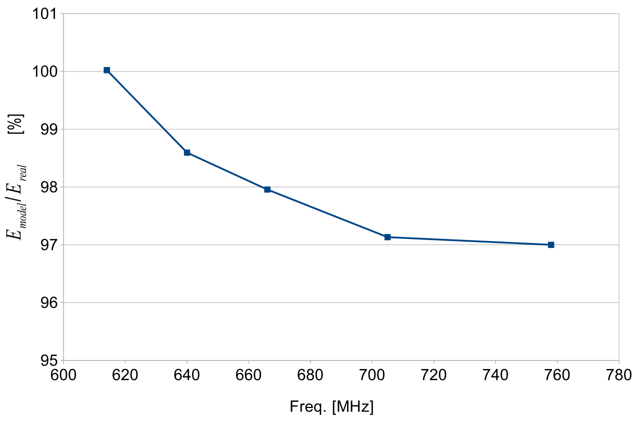


Fig. 9 Comparison of accuracy (%) between the proposed model and the real measurements on the Nvidia K20m accelerator

Table 14 Energy usage (J) of the proposed model and the real measurements for the investigated ILP model and heuristics

	ILP	Alg1	Alg2	Alg3	Alg4
E_{real}	2.29	7.23	8.56	6.56	8.56
E_{model}	2.27	7.53	8.86	6.86	8.86

Table 15 Comparison of accuracy (%) between the proposed model and the real measurements for the investigated ILP model and heuristics

	ILP	Alg1	Alg2	Alg3	Alg4
E_{model}/E_{real}	99	104	103	105	103

and the Nvidia K20m accelartor respectively for the 3D Laplacian stencil defined on the grid with 256^3 points. Table 14 contains the energy usage for all examined heuristics and the ILP model for the 2xCPU–2xGPU configuration of processors defined on the Cube grid presented in Sect. 8.1. Whereas, Table 15 summarizes their accuracy.

As it can be observed, the accuracy of the presented model is high and exceeds visibly 90%. The results suggest that applying the time and energy models, while verifying different scheduling policies, does not lead to deterioration of overall results. This leads to the conclusion that the described environment can be used to simulate the heterogeneous computer system.

9 Conclusions and future work

In this paper new heuristics to distribute efficiently the stencil workload on the heterogeneous architectures and consequently minimize the energy usage within the deadline are presented and evaluated. They are based on our analysis of energy and performance models for a relevant class of stencil computations to explore the relationship between task scheduling algorithms and energy constraints. Additionally, the obtained results during experimental tests of our heuristics are compared to optimal solutions achieved by the ILP formulation of the stencil-scheduling problem. The optimization space of the model shows that the best strategy depends not only on load balancing the problem size between the processing units, the processing units specification, and the stencils employed, but also on detailed mapping of the communication dependencies of the blocks to the communication topology of respective processing units. The careful mapping of the stencil tasks on the heterogeneous architectures can lead to the substantial savings in the execution time and energy costs. We show that even a basic heuristic with load balancing for the configurations of the two processors is sufficient enough with respect to energy efficiency. Moreover, in this paper we demonstrate various improvements which take into account recent achievements in heterogeneous CPU and GPU architectures. Nevertheless, with the increasing number of processors, in our opinion heuristics that take into account the communication penalty are needed. The heuristics are applicable to distribute the stencil computations defined on a Cartesian grid regardless of a domain topology. The domain borders can be both periodic and non-periodic.

In our future work we plan to extend the proposed model and heuristics to take into account the remote communication between nodes to better predict the runtime and the energy usage of stencil computations in large scale. Therefore, we plan to conduct additional experimental tests to model the data movement within the inter-node network. Furthermore, we want to model a workflow of the different stencils to better predict the energy usage of real use cases and applications.

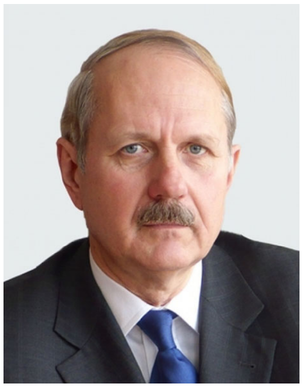
Acknowledgements This work has been supported by the National Science Centre (NCN) in Poland under the MAESTRO Grant No DEC-2013/08/A/ST6/00296.

References

1. Blazewicz, M., Hinder, I., Koppelman, D., Brandt, S., Ciznicki, M., Kierzynka, M., Loffler, F., Schnetter, E., Tao, J.: From physics model to results: An optimizing framework for cross-architecture code generation. Sci. Prog. **21**(1), 1–16 (2013)
2. Ciznicki, M., Kierzynka, M., Kopta, M., Kurowski, K., Gepner, P.: Benchmarking data and compute intensive applications on modern CPU and GPU architectures. Proced. Comput. Sci **9**, 1900–1909 (2012)
3. Ciznicki, M., Kierzynka, M., Kopta, P., Kurowski, K., Gepner, P.: Benchmarking JPEG 2000 implementations on modern CPU and GPU architectures. J. Comput. Sci. **5**(2), 90–98 (2014)
4. Rojek, K.A., Ciznicki, M., Rosa, B., Kopta, P., Kulczewski, M., Kurowski, K., Piotrowski, Z.P., Szustak, L., Wojcik, D.K., Wyrzykowski, R.: Adaptation of fluid model eulag to graphics processing unit architecture. Concurr. Comput. Pract. Exp. **27**(4), 937–957 (2015). doi:10.1002/cpe.3417
5. Ciznicki, M., Kulczewski, M., Kopta, P., Kurowski, K.: Methods to load balance a GCR pressure solver using a stencil framework on multi- and many-core architectures. Sci. Progr. **2015**, 13 (2015). doi:10.1155/2015/648752
6. Kurowski, K., Oleksiak, A., Piatek, W., Weglarz, J.: Hierarchical scheduling strategies for parallel tasks and advance reservations in grids. J. Sched. **16**(4), 349–368 (2013)
7. Mei, J., Li, K., Li, K.: Energy-aware task scheduling in heterogeneous computing environments. Clust. Comput. **17**(2), 537–550 (2014). doi:10.1007/s10586-013-0297-0
8. Chandio, A.A., Bilal, K., Tziritas, N., Yu, Z., Jiang, Q., Khan, S.U., Xu, C.-Z.: A comparative study on resource allocation and energy efficient job scheduling strategies in large-scale parallel computing systems. Clust. Comput. **17**(4), 1349–1367 (2014). doi:10.1007/s10586-014-0384-x
9. Bilal, K., Fayyaz, A., Khan, S.U., Usman, S.: Power-aware resource allocation in computer clusters using dynamic threshold voltage scaling and dynamic voltage scaling: comparison and analysis. Clust. Comput. **18**(2), 865–888 (2015). doi:10.1007/s10586-015-0437-9
10. Terzopoulos, G., Karatza, H.D.: Power-aware bag-of-tasks scheduling on heterogeneous platforms. Clust. Comput. **19**(2), 615–631 (2016). doi:10.1007/s10586-016-0544-2
11. Ciznicki, M., Kurowski, K., Weglarz, J.: Evaluation of selected resource allocation and scheduling methods in heterogeneous many-core processors and graphics processing units. Found. Comput. Decis. Sci. **39**(4), 233–248 (2014)
12. Nguyen, A., Satish, N., Chhugani, J., Kim, C., Dubey, P.: 3.5-D blocking optimization for stencil computations on modern CPUs and GPUs. In: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 1–13. IEEE Computer Society (2010)
13. Treibig, J., Wellein, G., Hager, G.: Efficient multicore-aware parallelization strategies for iterative stencil computations. J. Comput. Sci. **2**(2), 130–137 (2011)
14. Shalf, J., Dosanjh, S., Morrison, J.: Exascale computing technology challenges. In: High Performance Computing for Computational Science–VECPAR 2010, pp. 1–25. Springer, Berlin (2011)
15. Ciznicki, M., Kurowski, K.: Resource management strategies with energy profiles for stencil computing. In: HiStencils 2015, Second International Workshop on High-Performance Stencil Computations (2015)
16. Sellappa, S., Chatterjee, S.: Cache-efficient multigrid algorithms. Int. J. High Perform. Comput. Appl. **18**(1), 115–133 (2004)
17. M. Frigo, V. Strumpen, Cache oblivious stencil computations. In: Proceedings of the 19th annual international conference on Supercomputing, ACM, pp. 361–366
18. Datta, K., Kamil, S., Williams, S., Oliker, L., Shalf, J., Yelick, K.: Optimization and performance modeling of stencil computations on modern microprocessors. SIAM Rev. **51**(1), 129–159 (2009)
19. Maruyama, N., Nomura, T., Sato, K., Matsuoka, S.: Physis: an implicitly parallel programming model for stencil computations on large-scale GPU-accelerated supercomputers. In: High Performance Computing, Networking, Storage and Analysis (SC), 2011 International Conference for IEEE, pp. 1–12 (2011)
20. Pereira, A.D., Ramos, L., Góes, L.F.: Pskel: a stencil programming framework for CPU-GPU systems. Concurr. Comput. Pract. Exp. **27**(17), 4938–4953 (2015)
21. Williams, S., Waterman, A., Patterson, D.: Roofline: an insightful visual performance model for multicore architectures. Commun. ACM **52**(4), 65–76 (2009)
22. Treibig, J., Hager, G.: Introducing a performance model for bandwidth-limited loop kernels. In: Parallel Processing and Applied Mathematics, pp. 615–624. Springer, Berlin (2010)
23. Choi, J.W., Bedard, D., Fowler, R., Vuduc, R.: A roofline model of energy, in: Parallel and Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on, pp. 661–672. IEEE (2013)
24. Hager, G., Treibig, J., Habich, J., Wellein, G.: Exploring performance and power properties of modern multi-core chips via simple machine models, Concurr. Comput. (2014). doi:10.1002/cpe.3180
25. Wang, Z., Xu, X., Xiong, N., Yang, L.T., Zhao, W.: Energy cost evaluation of parallel algorithms for multiprocessor systems. Clust. Comput. **16**(1), 77–90 (2013). doi:10.1007/s10586-011-0188-1
26. Bomze, I.M., Budinich, M., Pardalos, P.M., Pelillo, M.: Modern Graph Theory, vol. 184. Springer, Berlin (1998)
27. Junglas, D.: Optimised grid-partitioning for block structured grids in parallel computing, Ph.D. thesis, TU Darmstadt (2007)
28. Holyer, I.: The np-completeness of edge-coloring. SIAM J. Comput. **10**(4), 718–720 (1981)
29. Garey, M.R., Johnson, D.S., Stockmeyer, L.: Some simplified np-complete problems. In: Proceedings of the sixth annual ACM symposium on Theory of computing, pp. 47–63. ACM 1(974)
30. Shannon, C.E.: A theorem on coloring the lines of a network. J. Math. Phys. **28**(1), 148–152 (1949)
31. Vizing, V.G.: The chromatic class of a multigraph. Cybern. Syst. Anal. **1**(3), 32–41 (1965)

Milosz Ciznicki has graduated Computer Science at Poznan University of Technology, Poland (2011). He works in Applications Department at Poznan Supercomputing and Networking Center. His research is focused on new hardware architectures and designing efficient and energy aware applications.

Krzysztof Kurowski is the Head of the Applications Department at Poznan Supercomputing and Networking Center. He joined in research and development activities conducted at PSNC in 1999. He received his Ph.D. in Computer Science from Poznan University of Technology in 2009. He has taken an active interest in the following research fields: distributed systems, grids, security, scheduling and resource management in distributed computing environments. He has taken also an active part in multidisciplinary EU projects within 5th, 6th and 7th Framework Programme, e.g. GridLab and inteliGrid. Recently, he has been leading technical R&D activities in two EU projects: QosCosGrid and MAPPER. Results of his research efforts have been successfully presented at many international workshops and conferences, including IFORS, CCGrid, HPDC, ICCS, Terena, Supercomputing and OGF. Recently he has been active in the field of ICT in education, e.g. he is a technical coordinator in the national project for e-textbooks "e-podręczniki do kształcenia ogólnego".



Jan Weglarz was born in 1947 in Poznan. He studied at the Faculty of Mathematics, Physics and Chemistry of the Adam Mickiewicz University of Poznan, where in 1969 he received an M.S. degree in Mathematics. He was also a student of the Faculty of Electrical Engineering of the Poznan University of Technology and in 1971 he defended his M.S. thesis in Control Engineering. Since 1971 he has been working at the Poznan University of Technology. First, he worked at the Institute of Control Engineering, then at the Institute of Computing Science, Control Engineering and Robotics, and since 1990 – at the Institute of Computing Science PUT. He received his Ph.D. degree in Control Systems in 1974, and became assistant professor in 1977. He obtained the position of associate professor in 1983, and professor in 1988. In 1993 he became a corresponding member, and in 1998 a full member of the Polish Academy of Sciences (PAS). He is a founding member of the Polish Information Processing Society, and a member of numerous national and international scientific societies including the American Mathematical Society and the Operations Research Society of America.