Resource management strategies with energy profiles for stencil computing

Milosz Ciznicki
Poznan Supercomputing and Networking Center
Dabrowskiego 79a Street
60-529 Poznan, Poland
miloszc@man.poznan.pl

Krzysztof Kurowski
Poznan Supercomputing and Networking Center
Dabrowskiego 79a Street
60-529 Poznan, Poland
krzysztof.kurowski@man.poznan.pl

ABSTRACT

The performance of high-end supercomputers will reach the exascale through the advent of core counts in the 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 analysis of execution and performance models for a relevant class of stencil computations to explore the relationship between task scheduling algorithms and energy constraints.

Keywords

power and energy modeling, performance analysis, scheduling, resource management, stencil computations, GPUs, many-core systems

1. INTRODUCTION

For our research we have selected stencil computations as they occur in many HPC codes on block-structured grids for modeling various physical phenomena, e.g. for computational fluid dynamics, geometric modeling, solving partial differential equations or image and video processing [3, 5, 6, 19, 12]. 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. 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. As shown in [11, 17], the overall performance of stencil 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 approach we propose a new energy-aware performance model for stencil computations that dynamically allocate tasks to various unit blocks and then optimize the schedule according to evaluation criteria. The model has been tested in a real environment using new computing hardware architectures.

The paper is organized as follows. In Section 2 the related work is discussed. The resource allocation problem is defined in Section 3. The energy-aware performance model is introduced in Section 4. The resource allocation policies are described in Section 5. Section 6 shows experiments using a 7-point stencil on a hybrid CPU-GPU node and Section 7 concludes the results.

2. RELATED WORK

In general, considered stencil calculations perform global

HiStencils 2015
Second International Workshop on High-Performance Stencil Computations
January 20, 2015, Amsterdam, The Netherlands
In conjunction with HiPEAC 2015.

http://www.exastencils.org/histencils/2015/

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 a 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 [13], after time decomposition [9] or proposed cache-aware optimisation algorithms on many-core modern processors [8]. Nevertheless, there is a lack of analyses of stencil optimizations and performance modeling connecting specific properties such as concurrency 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 hardware units. A good example is the roofline model which allows a programmer to model, predict, and analyze an individual kernel performance given an architecture communication and computation capabilities [18]. In this approach an application is modeled 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 multilevel memory hierarchy. It has been successfully used to model the performance of many different applications on the multi-core and many-core processors [16]. Recently, it has been extended to model the energy consumption in GPUs [4]. In the new model 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 [10] to model energy on dual multi-core CPU with three-level cache hierarchy. In this approach the dynamic power management was modeled 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.

An interesting multi-objective performance tuning problem formulation for the specific case of time, power and energy metrics has been examined recently by researchers in [2]. They consider decision spaces consisting of loop optimization techniques, clock frequencies and parallelization on multi-core CPU, Xeon Phi and Vesta IBM Blue Gene/Q many-core architectures.

3. PROBLEM DEFINITION

The main focus of our research presented in the paper is on a dynamic stencil workload scheduling using heterogeneous

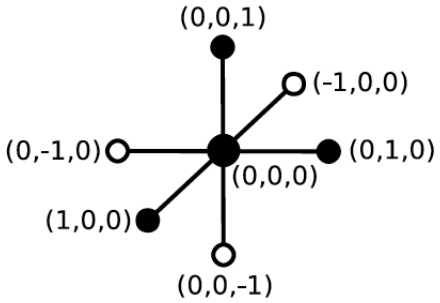

Figure 1: 7 point stencil

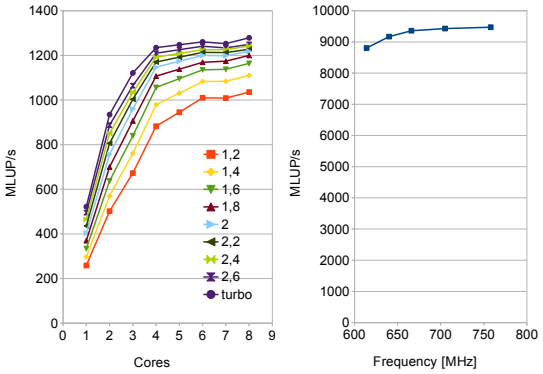

Figure 2: Performance of a 7 point stencil: left - CPU, right - GPU

computing architectures to both minimize the energy usage and improve (or maintain) the overall performance. The scheduling problem is defined by a set $P = P_1, P_2, ..., P_m$ of m processors and a workload $T = T_1, T_2, ..., T_n$ of n independent tasks.

A task is represented by 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 neighboring points are used during a stencil computation. A single update of the whole grid is called a timestep. In our approach 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). We also assumed that a given task can be processed by a single processing unit at a time and each single processing unit can perform only one task at a time.

3.1 Key properties

To discover key characteristics that have a relevant impact on the performance and energy usage of a stencil task running on a certain processing unit we have run several computational experiments. Thanks to the dynamic power management policies introduced on PUs, both the frequency and the number of cores have been controlled during our performance tests. The RAPL (Running Average Power Limit)

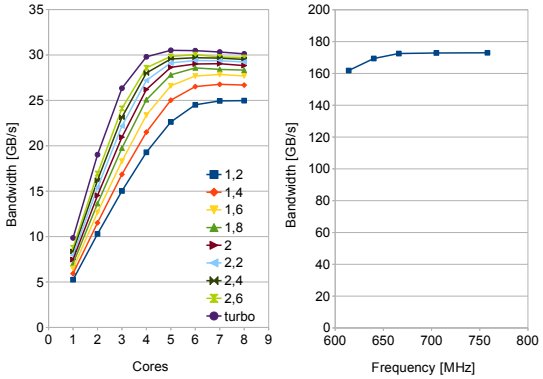

Figure 3: Memory bandwidth benchmark on: left - CPU, right - GPU

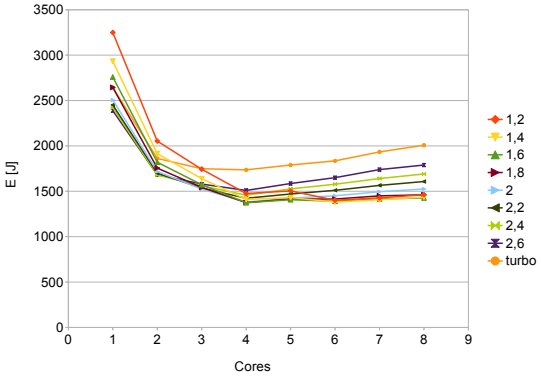

Figure 4: Energy usage of a 7 point stencil on CPU

and NVML (NVIDIA Management Library) interfaces are used during the computational experiments. The RAPL interface provides the ability to monitor and control the power on the CPU socket and DRAM, whereas the NVML interface allows to manage the power states of GPU. On multi-core CPUs the frequency switching implicitly changes an operating voltage. The voltage is optimized by the hardware based on several factors. All processor cores that execute some workload share the same frequency and voltage. The multiple frequency and voltage pairs while executing code are called P-states. The idle cores switch to low-power idle states called C-states. At the higher levels of C-states power save actions are taken such as: flush of the caches, stop of the clocks and reduction of the voltage to zero. Figure 2 shows the performance of a 7 point stencil on 8 core Intel Xeon E5 CPU and Kepler K20m GPU using different P-states. In case of CPU the maximum performance can be reached with 4 to 6 cores depending on the frequency used. For GPU all available cores execute workload and the maximum performance is achieved with the 705 MHz clock. The

Table 1: Properties of the modern architectures. The GPU bandwidth with Error Correting Code (ECC) switched on/off.

Platform	CPU Xeon E5-2670@2.60GHz	GPU Kepler K20m
Peak perf.	173 Gflop/s	1168 Gflops/s
Bandwidth	30 GB/s	143/173 GB/s
Ratio	5.76	8.17/6.75

stencil computations are a class of memory bound problems as the stencil's theoretical flop to byte ratio, which is typically less than 0.5, is significantly lower than that of the current computing architectures, see Table 1. In practice, the flop to byte ratio can be even lower due to a blocking overhead, as each stencil has to be divided to separate blocks to be effectively computed. To minimize the memory bandwidth pressure the sizes of the blocks were carefully selected to fit in a cache hierarchy of the target PU. Due to the bottleneck in the memory, the saturation of the performance is reached with a reduced number of cores. The four cores clocked at the turbo frequency are needed to saturate the memory bandwidth, whereas for the higher number of cores the lower frequency is needed, see Figure 3. The comparison of the Figures 2 and 3 shows that there is strong relationship between the memory bandwidth and the stencil performance. Figures 4 shows an influence of the P-states on an energy usage for a 7 point stencil. The lowest energy usage is reached with 4 cores clocked at 1.6 GHz. From 4 to 8 cores the difference in the energy usage is only 4% for the optimal frequencies, thus to save energy while increasing number of cores the frequency should be minimized. What is more, the lowest energy usage is reached with not the maximum performance of 1278 MLUP/s, but with the lower performance of 1107 MLUP/s. This is contrary to expectations that computations with the highest performance are the most energy efficient. The power consumption of the modern CPU consists two parts the static power and the dynamic power. The static power is referred as a leakage power where the power is lost due to a current finding its way to a ground. This process happens regardless of the operating frequency and it depends entirely on the voltage.

$$P_{static} = m * V_{cc} \tag{1}$$

The m value is a constant coefficient and V_{cc} is the CPU voltage. The dynamic power involves two additional terms one referred as a short-circuit energy and the other one is called a transition energy:

$$P_{transition} = \alpha * C * f/2 * V_{cc}^{B} \tag{2}$$

$$P_{short-circut} = \alpha * E_{short-circut} * f \tag{3}$$

$$P_{dynamic} = P_{shot-circut} + P_{transition} \tag{4}$$

where α is the activity parameter that depends on the instruction mix and overall Instruction per cycle (IPC) utilization, the C parameter is dependent on the layout of the chip, the f parameter is a current frequency, while the $E_{short-circut}$ parameter is the power consumed per short-circuit event. Figure 5 presents the power consumption of a 7 point stencil. Based on the Eq. 4 the theoretical relation of $P \sim V^B$ is not reflected on the figure as the power consumption changes linearly with the increasing voltage. Furthermore, the power consumption of three most important CPU components was checked such as: package (PKG), cores (PP0) and DRAM. PKG component includes power consumption of whole processor die whereas PP0 only contains power consumption of the cores. Figure 6 shows that

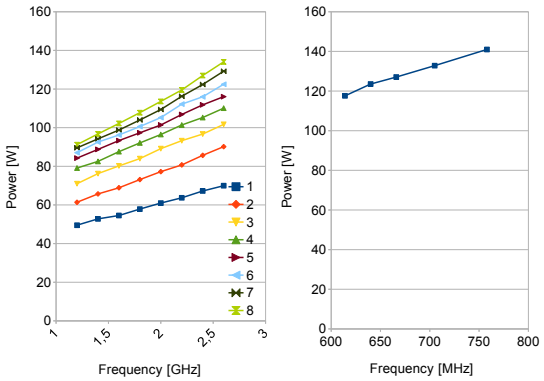

Figure 5: Power consumption of a 7 point stencil: left - CPU, right - GPU

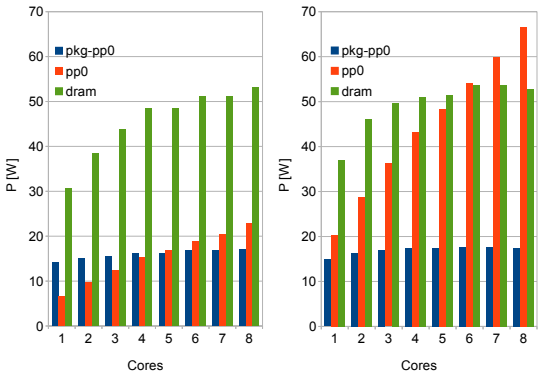

Figure 6: PKG, core and DRAM power consumption of a 7 point stencil on CPU

for the 1.2GHz clock 60% of the power consumes DRAM whereas for the 2.6GHz clock form 50% to 60% of the power consumes PKG. The power consumption of PP0 changes linearly with increasing number of cores.

Figure 7 depicts a power consumption based on lowest energy usage for each P-state. In our case data movement consumes the most of the power, as the PKG part also includes a power consumption of the caches. Figure 8 presents that the computation of a 7 point stencil on GPU is 10x more energy and performance efficient than on CPU.

4. PERFORMANCE AND ENERGY MODELS

After careful analysis of the performance and the energy usage of the stencil computations on two unrelated processing units the task $T_i \in \mathcal{T}$ is described by the following properties:

1. The number of arithmetic operations per grid point o_i.
2. The number of required bytes per grid point b_i.
3. The domain dimensions $d_i = [d_{ix}, d_{iy}, d_{iz}]^T$.
4. The arrival time r_i - the time when task is ready for an execution.

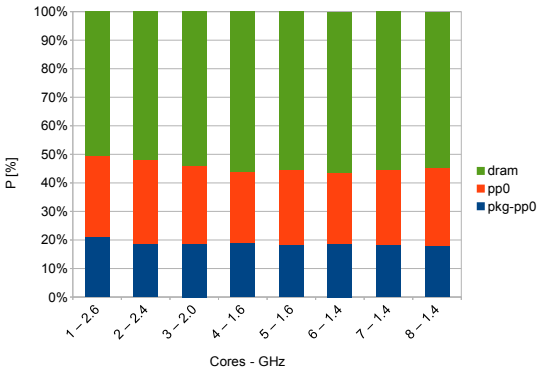

Figure 7: Power consumption of a 7 point stencil on CPU

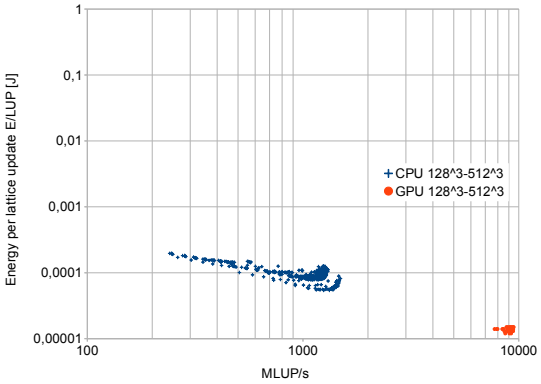

Figure 8: Performance to energy cost per lattice update for a 7 point stencil using different domain sizes on CPU and GPU

Arrival times r_i are not known a priori as tasks appear in the system in randomly manner. The tasks are non-preemptive and possess all required resources whenever they are executed. The processor $P_j \in \mathcal{P}$ has following properties:

1. The set of available frequencies $\mathcal{F} = \{f_{j1}, f_{j2}, \ldots, f_{jn}\}$.
2. The set of available cores $\mathcal{C} = \{c_{j1}, c_{j2}, \ldots, c_{jm}\}$.
3. The set of states $\mathcal{S} = \{(f, c) : f \in \mathcal{F} \wedge c \in \mathcal{C}\}$, where $s_{jl} \in \mathcal{S}$ is a selected state.
4. The sustained bandwidth to the main memory m_{jl} in bytes per second based on the state s_{jl}.
5. The blocking size $rb_j = [rb_{jx}, rb_{jy}, rb_{jz}]$ adjusted to the cache hierarchy.

The execution time τ_{ijl} of the task T_i on the processor P_j with state s_{jl} is estimated with Eq. 5,6,7 whereas the performance is calculated with Eq. 8.

$$O_i = o_i * d_{ix} * d_{iy} * d_{iz} \tag{5}$$

$$B_i = b_i * d_{ix} * d_{iy} * d_{iz} \tag{6}$$

$$\tau_{ijl} = (b_i * d_{ix} * d_{iy} * d_{iz})/m_{jl} \tag{7}$$

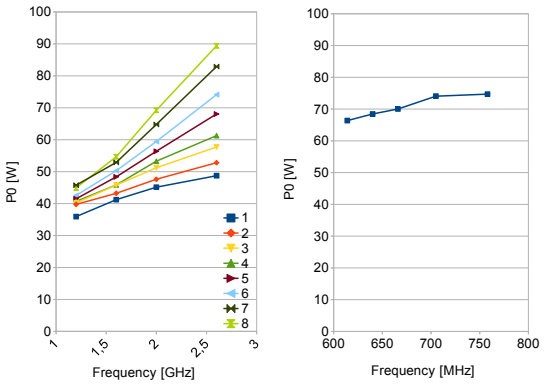

Figure 9: Constant power P0: left - CPU, right - GPU

$$\varphi_{ijl} = (d_{ix} * d_{iy} * d_{iz})/\tau_{ijl} \tag{8}$$

where O_i is the number of arthmetic operations executed and B_i is the number of bytes transfered.

The energy model assumes that each arithmetic operation as well as the memory operation consumes some energy, see Eq. 9:

$$E_{ijl} = e_{op,ij} * O_i + e_{byte,ij} * B_i + P0_{ijl} * \tau_{ijl} \tag{9}$$

The variables $e_{op,ij}$, $e_{byte,ij}$ approximates the energy usage of stencil operations. For simplicity, it is assumed that arithmetic operations, such as additions, multiplications, subtractions and divisions, consume the same amount of the energy. Additionally, the energy usage also depends on an utilized instruction set, thus for the highest performance the CPU implementation of the stencil uses the vector extensions. $P0_{ijl}$ is a constant power consumed by the processor P_j based on the state s_{ijl}. The coefficients $e_{op,ij}, e_{byte,ij}$ and $P0_{ijl}$ are approximated with a linear regression. Table 2 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 9 shows that the constant powers grows linearly with the increasing number of cores using different P-states.

Table 2: Energy coefficients for the CPU and GPU architectures.

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

4.1 Resource allocation strategies

The two types of resource allocation strategies are described in the following paragraphs. First strategy type optimizes the schedule length $C_{max} = max\{C_i\}$, where C_i is a completion time. Second strategy type optimizes the energy usage.

4.1.1 *Schedule length*

The algorithm of the Work-Stealing policy is presented on Figure 1. In Work-Stealing policy each worker is associated with a separate task queue. The tasks are distributed among queues in Round-Robin manner (lines 05-07). If the processor P_j has an empty queue Q_j it tries to lock queue $Q_k != Q_j$ of the processor P_k and steal its task (lines 08-16).

Algorithm 1 Work-stealing policy

```
1: m                                   ▷ the number of queues
2: Q_j                 ▷ the queue associated with processor P_j
3:
4: procedure WorkStealing
5:     for T_i ϵ 𝒯 do
6:         Q_{j%m} ← T_i
7:     end for
8:     if Q_j is empty then
9:         for Q_j ϵ 𝒬 do
10:            if try_lock(Q_j) then
11:                get_task(Q_j)
12:                unlock(Q_j)
13:                break
14:            end if
15:        end for
16:    end if
17: end procedure
```

The next scheduling policy is a Heterogeneous Earliest Finish Time algorithm. The HEFT algorithm schedules each arrived task on the processor that minimizes the finish time of the task. Before describing details of the resource allocation policy it is necessary to define the EST and EFT attributes that are derived from a given partial schedule. $EST(T_i, P_j)$ and $EFT(T_i, P_j)$ are the earliest execution start time and the earliest execution finish time of the task T_i on the processor P_j respectively. The EST value for the first task T_1 is:

$$EST(T_1, P_j) = 0 \tag{10}$$

$$EST(T_i, P_j) = avail[j] \tag{11}$$

$$EFT(T_i, P_j) = \tau_{ij} + EST(T_i, P_j) \tag{12}$$

For the other tasks the EFT and EST values are computed recursively. Starting from the task T_i as shown in Eq. 11 and Eq. 12. The value $avail[j]$ is the earliest time at which processor P_j is ready for the execution of the task. If T_k is the last executed task on processor P_j then $avail[j]$ is the time when processor P_j completed the execution of the task T_k. The algorithm presented in [15] statically schedules tasks on processors where arrival times and computation costs of tasks are known a priori. However, in a dynamical approach this information is not provided. As a result, the HEFT algorithm is adopted to satisfy constraints of the dynamical scheduling strategies. The algorithm is presented on Figure 2. When the task arrives to the system, the processing time τ_{ij} of the task T_i on the processor P_j is calculated based on the execution time model that was introduced in

Section 4. Next, the $EFT(T_i, P_k)$ values between the task T_i and all processors in the set P are calculated (line 03-06). Finally, the task T_i is assigned to the processor P_j that minimizes the EFT value of the task T_i (line 07).

Algorithm 2 Heterogeneous Earliest Finish Time

```
1: procedure HEFT
2:     while T_i ϵ 𝒯 do
3:         for P_j ϵ 𝒫 do
4:             τ_ij ← compute using the time model
5:             EFT(T_i, P_j) = τ_ij + EST(T_i, P_j)
6:         end for
7:         Assign task T_i to the processor P_j
8:         that minimizes EFT of task T_i
9:     end while
10: end procedure
```

Algorithm 3 Modified Heterogeneous Earliest Finish Time

```
1: procedure HEFT
2:     while T_i ∈ 𝒯 do
3:         for P_j ∈ 𝒫 do
4:             for s_jl ∈ 𝒮 do
5:                 find s_jl that minimizes E_ijl
6:             end for
7:             τ_ijl ← compute using selected s_jl
8:             E_ijl ← compute using τ_ijl
9:             EFT(T_i, P_j) = τ_ijl + EST(T_i, P_j)
10:        end for
11:        Assign task T_i to the processor P_j
12:        that minimizes E and EFT of task T_i
13:    end while
14: end procedure
```

4.1.2 Energy usage

The HEFT algorithm is modified to support the runtime control of the processor state, see Figure 3. For each processor P_j the state s_{jl} that minimizes the energy usage E_{ijl} of the task T_i is found. Next, the execution time τ_{ijl}, the energy cost E_{ijl} and the $EFT(T_I, P_j)$ value are computed. Finally, the task T_i is assigned to the processor P_j that minimizes the energy usage and the finish time of the task. During the execution of the task the processor is switched to the selected state s_{ijl}.

5. EXPERIMENTS

The experiments were conducted on single node with two Intel Xeon E5-2670 Sandy Bridge 8 core processors and two Nvidia Kepler K20m GPUs. We used our library to allocate stencils on the heterogeneous resources [7]. The library was modified to treat the whole CPU as a single processor. The workload with an average of 1000 stencils is simulated, each stencil is computed on a cubic domain that range in size from 64^3 to 512^3. The ratio of double precision floating point operations to the number of bytes for each lattice update range in $0.1 - 1.0$. The domain sizes and the flop-byte ratios are generated for each task using the gamma distribution method described in [1]. The 7 point 3D stencil is taken as an example of stencil computation. Each stencil is evaluated for 10000 timesteps. Within the experiment period the arrival times of the tasks are randomly generated using Poisson distribution. To simulate an overloaded system the mean task arrival time is equal to 1 s. The quality of the schedule is at first evaluated with two greedy resource allocation policies: the Work-Stealing (WS) and the Heterogeneous Earliest Finish Time (HEFT). In both policies the CPU frequency is controlled by the operating system with a ondemand governor. The ondemand governor is the default governor in the Linux based operating systems. The ondemand governor sets the operating frequency depending on the current usage of CPU. The GPU frequency is set to default values: 705MHz of the core clock and 2600MHz of the memory clock. The number of cores used in CPU is set to 7 as the the eighth core is utilized for the GPU offloading. The HEFT algorithm selects the task with the highest rank value at each step and assigns the selected task to the processor, which minimizes its earliest finish time. In the WS algorithm each processor has a separate task queue and the processor with an empty queue steals the task from the queue of the other processor. For the WS policy the time need to finish the workload is equal to $35700s$ and the energy cost is equal to $4908Wh$. While for the HEFT policy the evaluation criteria are as follows: workload completion time is equal to $23400s$ and the total energy usage is equal to $2946Wh$. The overall energy saving for the HEFT policy reached 40%.

The modified HEFT policy dynamically adapts the PU state during the execution of each task to minimize the energy usage and the finish time of the task. The CPU frequency range from the 1.2GHz frequency to a turbo frequency. The turbo frequency is a highest P-state called P0 and is controlled by the hardware based on the utilization of the active cores. In our case the observed turbo frequencies were: 3.3GHz for 1-2 cores, 3.2GHz for 3-4 cores, 3.1GHz for 5-6 cores and 3.0GHz for 7-8 cores. The GPU memory frequency is constant whereas the clock frequency is selected from the following values: 758MHz, 705MHz, 666MHz, 640MHz and 614MHz. The number of cores utilized for CPU range from 1 to 7 cores and for GPU the number of cores is not changed. The time need to finish the workload is equal to $25200s$ and the energy cost is equal to $2324Wh$. As we can see, this approach significantly improved the energy cost comparing to the previous policies by 21.6%. Moreover, the proposed allocation strategy only slightly worsen the workload completion time by 7.1%.

6. CONCLUSIONS AND FUTURE WORK

In this paper an energy-aware performance model for stencil computations is described that dynamically allocate tasks to various unit blocks and then optimize the schedule according to evaluation criteria. We showed that a partial usage of the floating point units in a given computing architecture reduces energy cost without the performance degradation. Additionally, the dynamic power management policies such as DVFS lead to significant reduction in the energy required for stencil computations. We shown, with three simple resource allocation policies, that the smart control of the processor state reduces the energy usage by 21.6%.

For future work, we want to extend our model to take into account the remote communication between processors to better predict the runtime and the energy usage of stencil computations. To enable this, we have to model the

data movement within the intra-node memory buses and the inter-node network. Additionally, we are building a dynamic decomposition method that will further minimize the energy usage.

7. 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.

8. REFERENCES

[1] S. Ali, H. Siegel, M. Maheswaran, D. Hensgen, and S. Ali. Representing task and machine heterogeneities for heterogeneous computing systems. *Tamkang Journal of Science and Engineering*, 3(3):195–208, 2000.

[2] P. Balaprakash, A. Tiwari, and S. M. Wild. Multi-objective optimization of hpc kernels for performance, power, and energy. *Proc. 4th Int. Wksp. Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS13)*, 2013.

[3] M. Blazewicz, I. Hinder, D. Koppelman, S. Brandt, M. Ciznicki, M. Kierzynka, F. Loffler, E. Schnetter, and J. Tao. From physics model to results: An optimizing framework for cross-architecture code generation. *Scientific Programming*, 21(1):1–16, 2013.

[4] J. W. Choi, D. Bedard, R. Fowler, and R. Vuduc. A roofline model of energy. In *Parallel & Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*, pages 661–672. IEEE, 2013.

[5] M. Ciznicki, M. Kierzynka, P. Kopta, K. Kurowski, and P. Gepner. Benchmarking data and compute intensive applications on modern CPU and GPU architectures. In *Procedia Computer Science 9*, volume 9, pages 1900–1909, 2012.

[6] M. Ciznicki, M. Kierzynka, K. Kurowski, B. Ludwiczak, K. Napierala, and J. Placzynski. Efficient isosurface extraction using marching tetrahedra and histogram pyramids on multiple GPUs. In *Parallel Processing and Applied Mathematics*, pages 343–352. Springer Berlin Heidelberg, 2012.

[7] M. Ciznicki, K. Kurowski, and J. Weglarz. Evaluation of selected resource allocation and scheduling methods in heterogeneous many-core processors and graphics processing units. *Foundations of Computing and Decision Sciences*, 39(4):233–248, 2014.

[8] K. Datta, S. Kamil, S. Williams, L. Oliker, J. Shalf, and K. Yelick. Optimization and performance modeling of stencil computations on modern microprocessors. *SIAM review*, 51(1):129–159, 2009.

[9] M. Frigo and V. Strumpen. Cache oblivious stencil computations. In *Proceedings of the 19th annual international conference on Supercomputing*, pages 361–366. ACM, 2005.

[10] G. Hager, J. Treibig, J. Habich, and G. Wellein. Exploring performance and power properties of modern multi-core chips via simple machine models. *Concurrency and Computation: Practice and Experience*, 2014.

[11] A. Nguyen, N. Satish, J. Chhugani, C. Kim, and P. Dubey. 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*, pages 1–13. IEEE Computer Society, 2010.

[12] K. A. Rojek, M. Ciznicki, B. Rosa, P. Kopta, M. Kulczewski, K. Kurowski, Z. P. Piotrowski, L. Szustak, D. K. Wojcik, and R. Wyrzykowski. Adaptation of fluid model eulag to graphics processing unit architecture. *Concurrency and Computation: Practice and Experience*, 2014.

[13] S. Sellappa and S. Chatterjee. Cache-efficient multigrid algorithms. *International Journal of High Performance Computing Applications*, 18(1):115–133, 2004.

[14] J. Shalf, S. Dosanjh, and J. Morrison. Exascale computing technology challenges. In *High Performance Computing for Computational Science–VECPAR 2010*, pages 1–25. Springer, 2011.

[15] H. Topcuoglu, S. Hariri, and M. Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE transactions on parallel and distributed systems*, pages 260–274, 2002.

[16] J. Treibig and G. Hager. Introducing a performance model for bandwidth-limited loop kernels. In *Parallel Processing and Applied Mathematics*, pages 615–624. Springer, 2010.

[17] J. Treibig, G. Wellein, and G. Hager. Efficient multicore-aware parallelization strategies for iterative stencil computations. *Journal of Computational Science*, 2(2):130–137, 2011.

[18] S. Williams, A. Waterman, and D. Patterson. Roofline: an insightful visual performance model for multicore architectures. *Communications of the ACM*, 52(4):65–76, 2009.

[19] R. Wyrzykowski, L. Szustak, and K. Rojek. Parallelization of 2d mpdata eulag algorithm on hybrid architectures with gpu accelerators. *Parallel Computing*, 2014.