How Pre-multicore Methods and Algorithms Perform in Multicore Era

Alexey Lastovetsky[1(✉)], Muhammad Fahad[1], Hamidreza Khaleghzadeh[1], Semyon Khokhriakov[1], Ravi Reddy[1], Arsalan Shahid[1], Lukasz Szustak[2], and Roman Wyrzykowski[2]

[1] School of Computer Science, University College Dublin, Belfield, Dublin 4, Ireland
alexey.lastovetsky@ucd.ie
[2] Czestochowa University of Technology, Czestochowa, Poland

Abstract. Many classical methods and algorithms developed when single-core CPUs dominated the parallel computing landscape, are still widely used in the changed multicore world. Two prominent examples are load balancing, which has been one of the main techniques for minimization of the computation time of parallel applications since the beginning of parallel computing, and model-based power/energy measurement techniques using performance events. In this paper, we show that in the multicore era, load balancing is no longer synonymous to optimization and present recent methods and algorithms for optimization of parallel applications for performance and energy on modern HPC platforms, which do not rely on load balancing and often return imbalanced but optimal solutions.

We also show that some fundamental assumptions about performance events, which have to be true for the model-based power/energy measurement tools to be accurate, are increasingly difficult to satisfy as the number of CPU cores increases. Therefore, energy-aware computing methods relying on these tools will be increasingly difficult to verify.

Keywords: Multicore platforms · Load balancing
Power and energy modeling · Performance monitoring counters

1 Introduction

Multicore CPUs and accelerators have become the standard building blocks of computing systems at all levels from supercomputers to mobile and embedded devices. At the same time, fundamentals of the dominant methods and algorithms currently used for performance and energy optimization of these systems were developed in the time when single-core CPUs dominated the computing landscape.

Two prominent examples are load balancing, which has been one of the main techniques for minimization of the computation time of parallel applications since the beginning of parallel computing, and model-based power/energy

R. Yokota et al. (Eds.): ISC 2018 Workshops, LNCS 11203, pp. 527–539, 2018.
https://doi.org/10.1007/978-3-030-02465-9_37

measurement techniques using performance events as model parameters. In this paper, we demonstrate that in the multicore era, load balancing is no longer synonymous to optimization. We also outline recent methods and algorithms for optimization of parallel applications for performance and energy on modern computing platforms, which do not rely on load balancing and often return imbalanced but optimal solutions.

We also show that some fundamental assumptions about performance events, which have to be true for the model-based power/energy measurement tools to be accurate, are increasingly difficult to satisfy as the number of CPU cores increases. Therefore, energy-aware computing methods relying on these tools will be increasingly difficult to verify.

The paper is organized as follows. Section 2 explains limitations of load balancing for performance and energy optimization of applications on multicore-based computing platforms. It also presents recent optimization methods and algorithms, returning optimal but typically unbalanced solutions. Section 3 discusses the accuracy of the popular methods for power and energy measurements of multicore processors in the light of recently discovered irregularities of some fundamental building blocks of these methods. Section 4 concludes the paper.

2 How Much Performance and Energy You Can Lose Through Load Balancing on Multicore Platforms

In this section, we formulate conditions that allow load balancing algorithms to minimize the execution time and the energy consumption of parallel applications. We then show that while these conditions are satisfied for single-core based platforms, they do not hold for multicore-based ones. We outline new workload distribution algorithms that address this problem. We also demonstrate the extent of performance and energy losses due to the use of load-balanced but not optimal application configurations.

2.1 When Does Load Balancing Work?

Load balancing algorithms can be classified as static or dynamic. Static algorithms (for example, those based on data partitioning) [1–6] require a priori information about the parallel application and platform. Dynamic algorithms (such as task scheduling and work stealing) [7–9] balance the load by moving fine-grained tasks between processors during the calculation. Dynamic algorithms do not require a priori information about execution but may incur significant communication overhead due to data migration.

The intuition behind the assumption that balancing the application improves its performance is the following: a balanced application does not waste processor cycles on waiting at points of synchronization and data exchange, maximizing this way the utilization of the processors and minimizing the computation time.

Let us analyze this assumption following [10]. Consider an application, the computational performance of which can be modeled by speed functions. Namely,

let p parallel processors be used to execute the application and $s_i(x)$ be the speed of execution of the workload of size x by processor i. Here the speed can be measured in floating point operations per second or any other fix-sized computation units per unit time. The size of workload can be characterized by the problem size (for example, the number of cells in the computational domain or the matrix size) or just by the number of equal-sized computational units. The speed $s_i(x)$ is calculated as $\frac{x}{t_i(x)}$, where $t_i(x)$ is the execution time of the workload of size x on processor i. Using these definitions, it was proved [10] that in order to guarantee that the balanced configuration of the application will execute the workload of size n faster than any unbalanced configuration, the speed functions $s_i(x)$ should satisfy the condition:

$$\forall \Delta x > 0 \colon \frac{s_i(x)}{x} \geq \frac{s_i(x + \Delta x)}{x + \Delta x} \tag{1}$$

Geometrically, it is illustrated in Fig. 1. The angle $\alpha(x)$ between the straight line, connecting the point $(0, 0)$ and the point $(x, s(x))$ on the speed curve, and the x-axis will be inversely proportional to the execution time of the workload of size x by the processor. Indeed, the cotangent of this angle is directly proportional to the ratio $\frac{x}{s(x)}$ representing the execution time of the workload x. Therefore, larger angles correspond to shorter execution times. Condition 1 means that the increase of the workload, x, will never result in the decrease of the execution time, or equivalently in the increase of the angle $\alpha(x)$: $\forall \Delta x > 0 \colon \alpha(x) \geq \alpha(x + \Delta x)$. Figure 1 illustrates the situation when load balancing will minimize the time of parallel execution of the application. For simplicity, assume that all our p processors are identical, characterized by the speed function in Fig. 1. Equal distribution of the total workload, w, allocating each processor workload $x = \frac{w}{p}$, will result in the execution time characterized by $\alpha(x)$. Any workload redistribution would lead to one of the processors executing larger workload $x + \Delta x$. As in general the parallel execution time is characterized by $\min_i \alpha(x_i)$, where x_i is the workload allocated to i-th processor, then $\min_i \alpha(x_i) \leq \alpha(x + \Delta x) \leq \alpha(x)$. This means that the load-balanced equal distribution will minimize the parallel execution time of any workload.

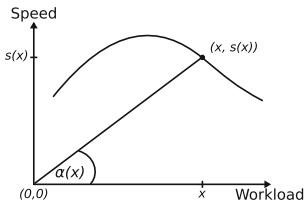


Fig. 1. Speed function suitable for minimization of computation time through load balancing. Angle $\alpha(x)$ represents the computation time: the greater the angle, the shorter the computation time.

The main body of the load balancing algorithms designed for performance optimization explicitly or implicitly assume that the speed of processor does not

depend on the size of workload [1,2,11–14]. In other words, the speed functions $s_i(x)$ are assumed to be positive constants, in which case Condition 1 is trivially satisfied. More advanced algorithms are based on functional performance models (FPMs), which represent the speed of processor by a continuous function of the problem size [15,16]. However, the shape of the function is not arbitrary but has to satisfy the following assumption [4]: Along each of the problem size variables, either the function is monotonically decreasing, or there exists point x such that

- On the interval $[0, x]$, the function is
  - monotonically increasing,
  - concave, and
  - any straight line coming through the origin of the coordinate system intersects the graph of the function in no more than one point.
- On the interval $[x, \infty)$, the function is monotonically decreasing.

These restrictions on the shape of speed functions guarantee that the efficient load balancing algorithms, proposed in [17–22], will always return a unique solution, minimizing the computation time. At the same time, it is easy to show that the restrictions imposed on FPMs will make them comfortably satisfy Condition 1.

Thus, the state-of-the-art load balancing algorithms designed for optimization of the computational performance of parallel applications assume that their performance profiles satisfy Condition 1.

Adding energy to the picture, let $E_i(x)$ be the energy consumed by processor i during the execution of workload of size x. In the case of p identical processors characterized by the same energy function $E(x)$, the equal distribution of the workload will always minimize the energy consumption if $\frac{dE(x)}{dx} \geq 0$ and $\frac{d^2E(x)}{dx^2} \geq 0$. If $E(x)$ is linear, that is, $\frac{d^2E(x)}{dx^2} = 0$, then any distribution of the workload w will result in the same energy consumption, $E(x_1) + \ldots + E(x_p) = (p-1) \times E(0) + E(x_1 + \ldots + x_p) = (p-1) \times E(0) + E(w)$. If $E(x)$ is strictly convex, that is, $\frac{d^2E(x)}{dx^2} > 0$, then any uneven distribution will consume more energy than the equal load-balanced one, which is evident from Fig. 2.

2.2 When Does Load Balancing Not Work?

The conditions on performance and energy profiles formulated in Sect. 2.1 are comfortably satisfied for single-core processors. We illustrate this using the execution of the OpenBLAS DGEMM application on a single core of an Intel Haswell server. Figures 3a and b respectively show the shapes of the experimentally built speed and dynamic energy functions. The application multiplies two square matrices of size $n \times n$ (problem size is equal to n^2). In these experiments, the *numactl* tool is used to bind the application to one core. The dynamic energy consumptions are obtained using Watts Up Pro power meter.

However, if we run the same application on all 24 cores of the multicore CPU of the Haswell server executing 24 threads, the picture will drastically change as shown in Fig. 4. The performance and energy profiles are no longer smooth and

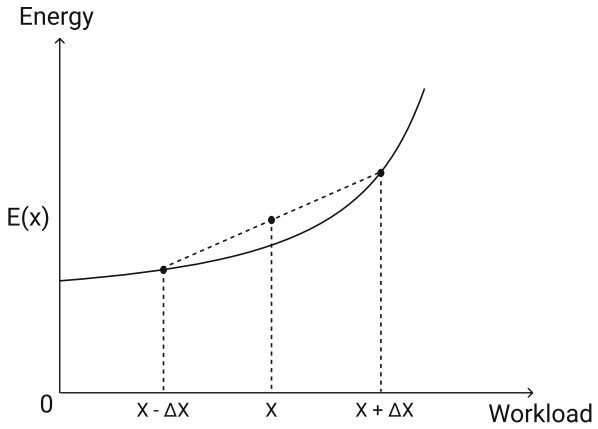


Fig. 2. Convex energy function suitable for minimization of energy consumption through equal distribution of the workload between identical processors. It is evident that for any x, $E(x) < \frac{E(x-\Delta x)+E(x+\Delta x)}{2}$, and hence, $E(x) + E(x) < E(x - \Delta x) + E(x + \Delta x)$.

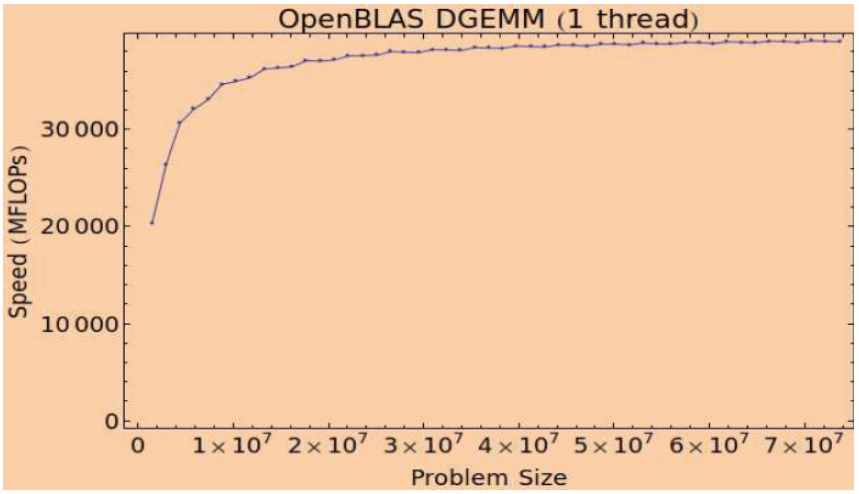
deviate significantly from the shapes observed before. Even more spectacular variations in speed and energy can be seen in Fig. 5 for the FFTW application [23] performing a 2D FFT of size $n \times n$ (the problem size being n^2). The variations in energy reach a maximum of 125%, and the average variation in speed is 60%. It is important to note that these variations are not noise but an inherent trait of applications executing on multicore servers with resource contention and NUMA. It is evident that equal distribution of the workload between identical processors with such performance and energy profiles will no longer guarantee minimization of execution time or energy consumption. More generally, traditional methods and algorithms used for optimization of performance and/or energy of parallel applications will not work for modern multicore-based platforms.

2.3 New Methods and Algorithms for Performance and Energy Optimization on Multicore-Based Platforms

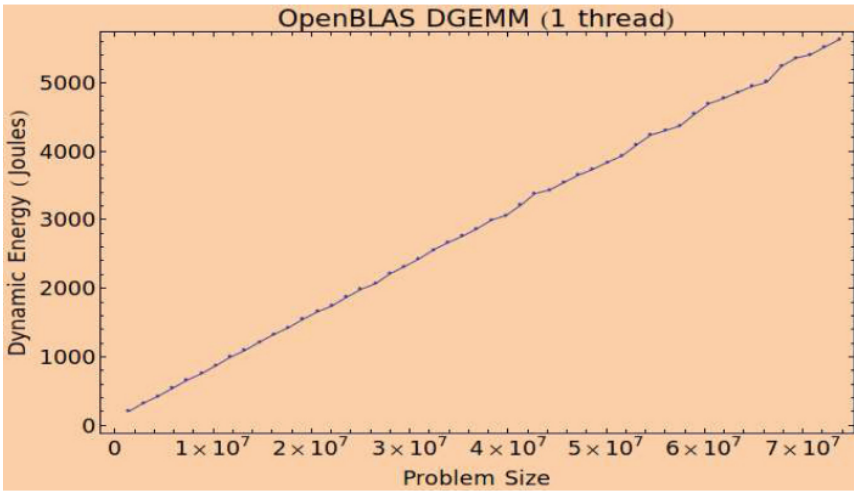
The challenges for performance and energy optimization of parallel applications on multicore-based platforms explained in Sect. 2.2 have been addressed over last 2 years in few publications. In [24], the problems of optimal workload distribution between identical processors for performance and dynamic energy consumptions were formulated and solved. Performance optimization problem (**POPT**) was formulated as follows:

- Given a discrete speed/performance function $s(x)$ of a processor
- Obtain partitioning, $d = \{x_1, \cdots, x_p\}$, of workload of size n using p identical processors so as to: $minimize \max_{i=1}^{p}(\frac{x_i}{s(x_i)}) \quad s.t. \quad \sum_{i=1}^{p} x_i = n.$

An exact algorithm solving this problem, **POPTA**, of complexity $O(m^2 \times p^2)$, where m is the cardinality of the discrete speed function $s(x)$, was proposed. The average and maximum performance improvements of POPTA over the equal



(a)



(b)

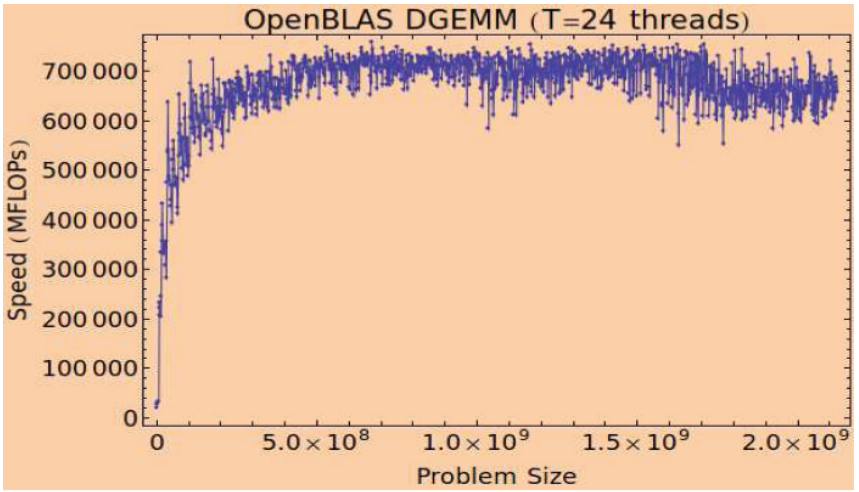
Fig. 3. (a) Speed function of OpenBLAS DGEMM application executed on a single core on the Intel Haswell server. (b) Dynamic energy consumption of OpenBLAS DGEMM application executed on a single core on the Intel Haswell server.

distribution solution were $(13\%, 71\%)$ for DGEMM and $(40\%, 95\%)$ for FFTW on a cluster of Haswell workstations.

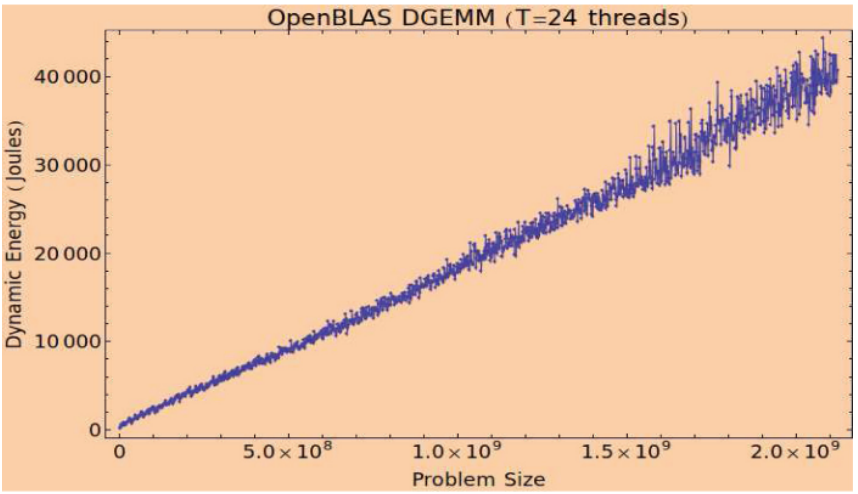
Energy optimization problem (**EOPT**) was formulated as follows:

- Given a discrete dynamic energy function $e(x)$ of a processor
- Obtain partitioning, $d = \{x_1, \cdots, x_p\}$, of workload of size n using p identical processors so as to: $minimize \sum_{i=1}^{p} e(x_i) \quad s.t \quad \sum_{i=1}^{p} x_i = n.$

An exact algorithm solving the energy optimization problem, **EOPTA**, of complexity $O(m^2 \times p^2)$, where m is the cardinality of the discrete energy function $e(x)$, was designed. The average and maximum energy improvements of EOPTA over the equal distribution solution were $(18\%, 71\%)$ for DGEMM and $(22\%, 127\%)$ for FFTW.



(a)

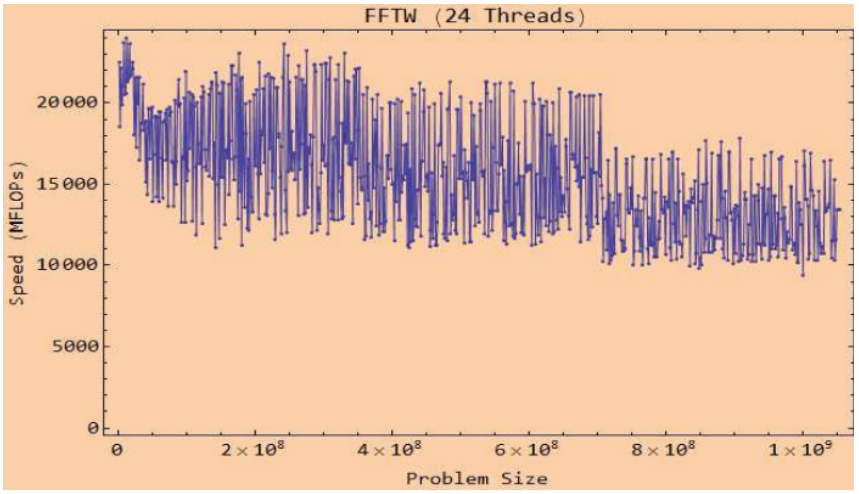


(b)

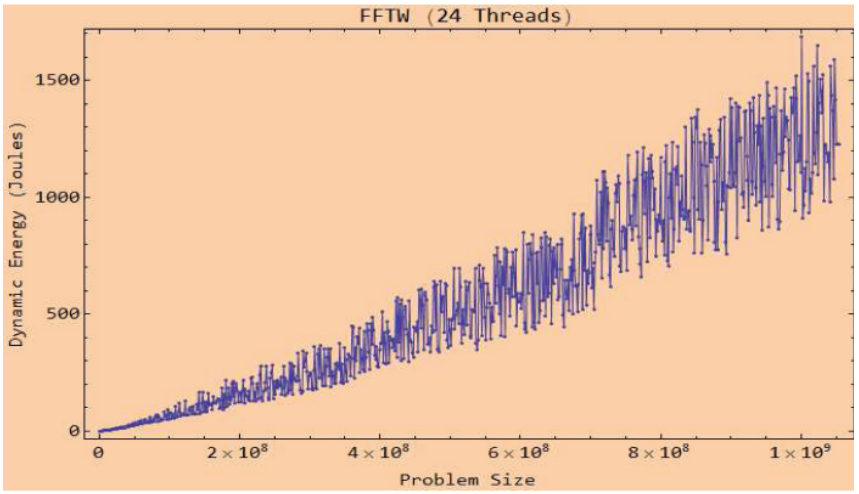
Fig. 4. (a) Speed function of OpenBLAS DGEMM executing 24 threads on the Intel Haswell server. (b) Function of dynamic energy consumption against problem size for OpenBLAS DGEMM executing 24 threads on the Intel Haswell server.

It was observed that optimization for performance only also reduced the energy consumption: $(12\%, 68\%)$ for DGEMM and $(22\%, 55\%)$ for FFTW. At the same time, optimization for energy only significantly degraded the performance: by 95–100% for both DGEMM and FFTW.

In order to better understand the interplay between optimization for performance and energy, a bi-objective optimization problem was studied in [25]. It was mathematically formulated as follows. Consider a workload of size n to be executed using p available identical processors. The performance of a processor executing a problem size x is given by $s(x)$, and the dynamic energy consumption of the execution of a problem size x by a processor is given by $e(x)$. Then the bi-objective optimization problem for minimization of execution time and minimization of total dynamic energy of computations during the execution of the workload is as follows:



(a)



(b)

Fig. 5. (a) Speed function of FFTW executing 24 threads on the Intel Haswell server. (b) Function of dynamic energy consumption against problem size for FFTW executing 24 threads on the Intel Haswell server.

$$
\begin{aligned}
&\textbf{BOPPE}(n, p, s, e, q) : \\
&\qquad \textit{minimize} \quad \{\max_{i=1}^{q} \frac{x_i}{s(x_i)}, \sum_{i=1}^{q} e(x_i)\} \\
&\qquad \text{Subject to} \quad x_1 + x_2 + ... + x_q = n \\
&\qquad\qquad\qquad\quad x_i \geq 0 \qquad i = 1, ..., q \\
&\qquad\qquad\qquad\quad x_i \leq n \qquad i = 1, ..., q \\
&\qquad\qquad\qquad\quad 1 \leq q \leq p \\
&\qquad\qquad \text{where} \quad p, q, n, x_i \in \mathbb{Z}_{>0}, \\
&\qquad\qquad\qquad\quad s(x), e(x) \in \mathbb{R}_{>0}
\end{aligned}
$$

The output of a solution method solving *BOPPE* is a set of Pareto-optimal solutions represented by workload distributions. It is important to note that the optimal number of processors (q) that are selected in a Pareto-optimal solution satisfies the constraint, $1 \leq q \leq p$.

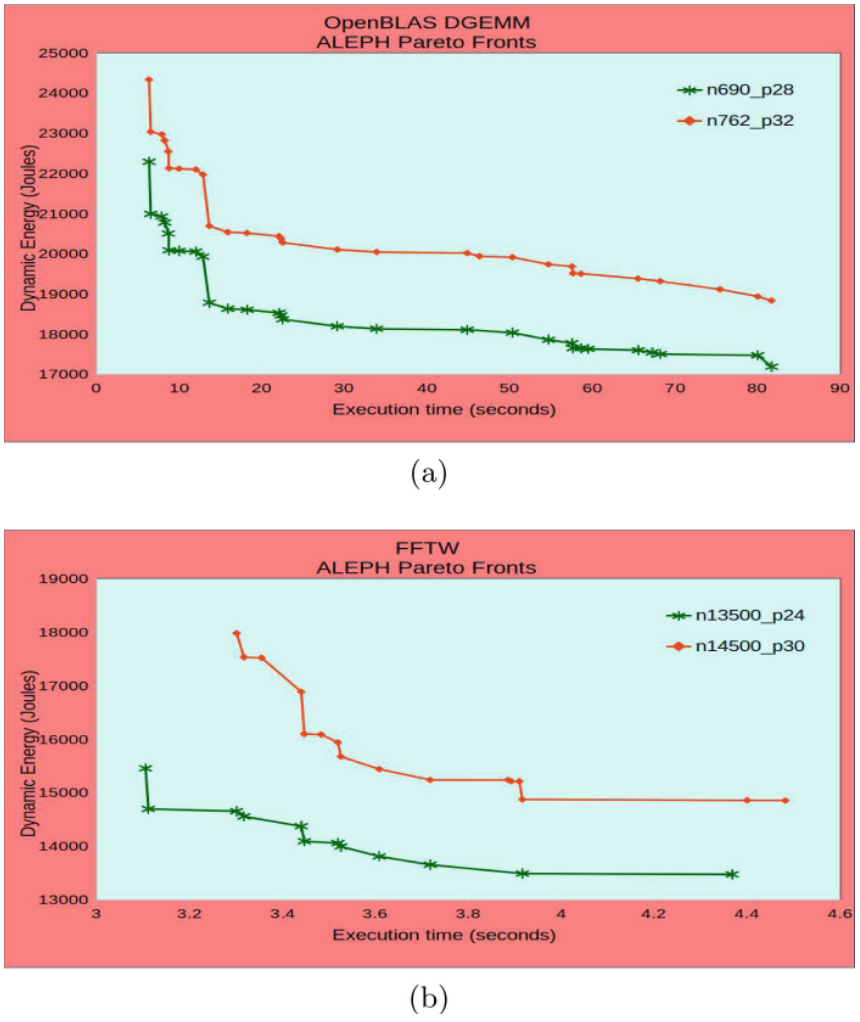


Fig. 6. Globally Pareto-optimal set of solutions with maximum sizes determined by *ALEPH* for OpenBLAS DGEMM and FFTW applications. Each curve shown as nX_pY represents results for data-parallel application workload size given by n (in multiples of granularity) and number of available processors, p.

A global optimization algorithm solving BOPPE (**ALEPH**) of complexity $O(m^2 \times p^2)$, where m is the cardinality of discrete functions $s(x)$ and $e(x)$, was designed. Figure 6 demonstrates solutions returned by ALEPH for DGEMM and FFTW for different problem sizes and numbers of processors. The interplay between single-objective optimizations for performance and energy can be visually studied using POPTA and EOPTA trajectories in the objective space together with the Pareto optimal front as shown in Fig. 7. This visual model can serve as a guidance for the choice of optimization method for performance and energy.

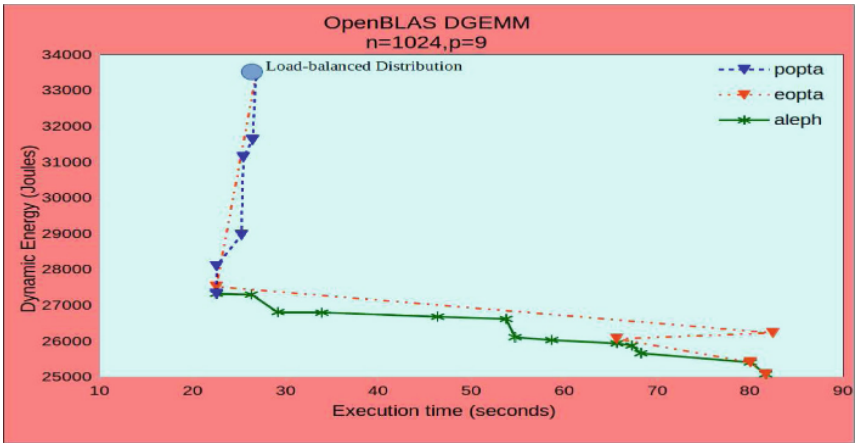


Fig. 7. Blue curve is the path of POPTA. Orange curve is the path of EOPTA. Green curve is the Globally Pareto-optimal front determined by ALEPH. (Color figure online)

The next step was done in [26], where performance optimization on heterogeneous multicore processors was addressed. The following heterogeneous performance optimization problem was studied. Consider a workload of size n executed using p heterogeneous processors, whose speed functions are represented by $S = \{s_1(x), ..., s_p(x)\}$ where $s_i(x)$, $i \in \{1, 2, \cdots, p\}$, is a discrete speed function of cardinality m of processor P_i. Without loss of generality, one can assume $x \in \{1, 2, \cdots, m\}$. The performance optimization problem can be then formulated as follows:

HPOPT$(n,\ p,\ m,\ S,\ d_{opt},\ t_{opt})$: Find a distribution, $d_{opt} = \{w_1, ..., w_p\}$, of the workload of size n using p available heterogeneous processors so as to minimize the computation time of parallel execution of the workload. The parameters $(n,\ p,\ m,\ S)$ are the inputs to the problem. The outputs are d_{opt}, which is the workload distribution, and t_{opt}, which is the optimal execution time. This problem can be formulated as an integer nonlinear programming problem (INLP):

$$\begin{aligned} t_{opt} &= \min_{d} \max_{i=1}^{p} \frac{x_i}{s_i(x_i)} \quad (d = \{x_1, ..., x_p\}) \\ \text{Subject to:} \quad & x_1 + x_2 + ... + x_p = n \\ & 0 \le x_i \le m, \qquad i = 1, ..., p \\ \text{where} \quad & p, m, n \in \mathbb{Z}_{>0} \quad \text{and} \quad x_i \in \mathbb{Z}_{\ge 0} \quad \text{and} \\ & s_i(x) \in \mathbb{R}_{>0} \end{aligned} \tag{2}$$

The objective function in the formulated minimization problem is a function of workload distribution d, $d = \{x_1, ..., x_p\}$, of a given workload n between the p processors. For each given d, it returns the time of its parallel execution, which is calculated as the time taken by the longest running processor to execute its workload. Any distribution that minimizes this function is considered optimal as its execution time of workload n by the p processors cannot be improved. An algorithm, **HPOPTA**, of complexity $O(m^3 \times p^3)$, solving the HPOPT optimization problem was proposed [26]. Its efficiency was demonstrated using DGEMM and 2D–FFT on a heterogeneous cluster of hybrid servers, integrating Intel

multicore Haswell CPUs, Nvidia GPUs, and Intel Xeon Phi co-processors. Significant average and multifold maximum speedups were achieved for both applications against best load-balanced solutions.

3 PMC-Based Power and Energy Modelling in Multicore Era

Efficient methods for performance optimization of parallel applications require accurate measurements of the execution time at all levels of the application, from individual threads to processes to the whole program, from data movements between memory levels to point-to-point and collective communications between processes. Similarly, efficient methods for energy optimization also require accurate measurements of the energy consumption at all levels of the parallel application. The energy consumption of the whole application, running on a server or a data center, can be accurately measured using power meters. Measuring the energy consumption by application components at a thread level is a wide open problem. At the same time, some approaches have been developed for power and energy measurements at the level of processes running on individual processors. The mainstream approach is to employ models predicting power/energy consumption using performance monitoring counters (PMCs). The PMC-based models are predominantly linear [27].

Modern hardware processors provide a large set of PMCs. Determination of the best subset of PMCs for energy predictive modeling is a non-trivial task given the fact that all the PMCs can not be determined using a single application run. Several techniques have been devised to address this challenge. While some techniques are based on statistical methodology, some use expert advice to pick a subset (that may not necessarily be obtained in one application run) that, in experts' opinion, are significant contributors to energy consumption. While a significant number of PMC-based models have been proposed [27], there is still no model, the predictive accuracy of which would be sufficiently high. The best verifiable average prediction error of average dynamic power by such models for a Intel Haswell platform is in the range of 90–100% [27].

One of the causes of this inaccuracy has been recently discovered [28]. It was found that many popular PMCs, widely used in predictive models, are not *additive*. The property of additivity is based on the simple and intuitive observation that the energy consumed by a serial execution of two applications should be equal to the sum of energies consumed by individual applications. This observations is just a manifestation of the fundamental energy conservation law, and it was validated by extensive experimentation [28]. One consequence of this observation is that any PMC parameter in a linear power/energy predictive model should be additive, that is, the value of this parameter for a serial execution of two applications should be equal to the sum of its values for individual applications. The study found many popular PMCs non-additive, some reaching up to 200% deviation from additivity. We conducted a simple experiment and found that just by removing highly non-additive PMCs from the model, we can significantly improve its predictive power. We also found that the number of

non-additive PMCs increases with the increase of the number of cores used to run applications, with very few non-additive PMCs in the case of single core.

4 Conclusion

In this paper, we have demonstrated that transition from single-core to multicore-based platforms makes many traditional methods for performance and energy optimization of parallel applications inefficient, and requires development of new methods and algorithms.

Acknowledgement. This publication has emanated from research conducted with the financial support of Science Foundation Ireland (SFI) under Grant Number 14/IA/2474. This work is partially supported by EU under the COST Program Action IC1305: Network for Sustainable Ultrascale Computing (NESUS).

References

1. Fatica, M.: Accelerating Linpack with CUDA on heterogenous clusters. In: GPGPU-2, pp. 46–51. ACM (2009). https://doi.org/10.1145/1513895.1513901
2. Yang, C., Wang, F., Du, Y., et al.: Adaptive optimization for petascale heterogeneous CPU/GPU computing. In: Cluster 2010, pp. 19–28 (2010)
3. Ogata, Y., Endo, T., Maruyama, N., Matsuoka, S.: An efficient, model-based CPU-GPU heterogeneous FFT library. In: IPDPS 2008, pp. 1–10 (2008)
4. Lastovetsky, A., Reddy, R.: Data partitioning with a functional performance model of heterogeneous processors. Int. J. High Perform. Comput. Appl. **21**(1), 76–90 (2007)
5. Rojek, K., Wyrzykowski, R.: Parallelization of 3D MPDATA algorithm using many graphics processors. In: Malyshkin, V. (ed.) PaCT 2015. LNCS, vol. 9251, pp. 445–457. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21909-7_43
6. Zhong, Z., Rychkov, V., Lastovetsky, A.: Data partitioning on multicore and multi-GPU platforms using functional performance models. IEEE Trans. Comput. **64**(9), 2506–2518 (2015)
7. Linderman, M.D., Collins, J.D., Wang, H., et al.: Merge: a programming model for heterogeneous multi-core systems. SIGPLAN Not. **43**, 287–296 (2008)
8. Augonnet, C., Thibault, S., Namyst, R.: Automatic calibration of performance models on heterogeneous multicore architectures. In: Lin, H.-X., et al. (eds.) Euro-Par 2009. LNCS, vol. 6043, pp. 56–65. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14122-5_9
9. Quintana-Ortí, G., Igual, F.D., Quintana-Ortí, E.S., van de Geijn, R.A.: Solving dense linear systems on platforms with multiple hardware accelerators. SIGPLAN Not. **44**, 121–130 (2009)
10. Lastovetsky, A., Szustak, L., Wyrzykowski, R.: Model-based optimization of EULAG kernel on Intel Xeon Phi through load imbalancing. IEEE Trans. Parallel Distrib. Syst. **28**(3), 787–797 (2017)
11. Luk, C.K., Hong, S., Kim, H.: Qilin: exploiting parallelism on heterogeneous multiprocessors with adaptive mapping. In: MICRO-42, pp. 45–55 (2009)
12. Cierniak, M., Zaki, M., Li, W.: Compile-time scheduling algorithms for heterogeneous network of workstations. Comput. J. **40**, 356–372 (1997)

13. Kalinov, A., Lastovetsky, A.: Heterogeneous distribution of computations solving linear algebra problems on networks of heterogeneous computers. J. Parallel Distrib. Comput. **61**(4), 520–535 (2001)
14. Martínez, J., Garzón, E., Plaza, A., García, I.: Automatic tuning of iterative computation on heterogeneous multiprocessors with ADITHE. J. Supercomput. **58**(2), 151–159 (2011)
15. Lastovetsky, A., Twamley, J.: Towards a realistic performance model for networks of heterogeneous computers. In: Ng, M.K., Doncescu, A., Yang, L.T., Leng, T. (eds.) High Performance Computational Science and Engineering. ITIFIP, vol. 172, pp. 39–57. Springer, Boston, MA (2005). https://doi.org/10.1007/0-387-24049-7_3
16. Lastovetsky, A., Reddy, R.: Data partitioning with a realistic performance model of networks of heterogeneous computers. In: Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS 2004). IEEE Computer Society, Santa Fe (2004)
17. Ilic, A., Pratas, F., Trancoso, P., Sousa, L.: High-performance computing on heterogeneous systems: database queries on CPU and GPU. In: High Performance Scientific Computing with Special Emphasis on Current Capabilities and Future Perspectives. IOS Press, Amsterdam (2011)
18. Colaço, J., Matoga, A., Ilic, A., Roma, N., Tomás, P., Chaves, R.: Transparent application acceleration by intelligent scheduling of shared library calls on heterogeneous systems. In: Wyrzykowski, R., Dongarra, J., Karczewski, K., Waśniewski, J. (eds.) PPAM 2013, part I. LNCS, vol. 8384, pp. 693–703. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-55224-3_65
19. Lastovetsky, A., Reddy, R.: Data distribution for dense factorization on computers with memory heterogeneity. Parallel Comput. **33**(12), 757–779 (2007)
20. Clarke, D., Lastovetsky, A., Rychkov, V.: Dynamic load balancing of parallel computational iterative routines on highly heterogeneous HPC platforms. Parallel Proces. Lett. **21**(02), 195–217 (2011)
21. AlOnazi, A., Keyes, D., Lastovetsky, A., Rychkov, V.: Design and Optimization of OpenFOAM-based CFD Applications for Hybrid and Heterogeneous HPC Platforms. arXiv preprint arXiv:1505.07630 (2015)
22. Clarke, D., Lastovetsky, A., Rychkov, V.: Column-based matrix partitioning for parallel matrix multiplication on heterogeneous processors based on functional performance models. In: Alexander, M., et al. (eds.) Euro-Par 2011. LNCS, vol. 7155, pp. 450–459. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29737-3_50
23. FFTW: Fastest Fourier Transform in the West (2018). http://www.fftw.org/
24. Lastovetsky, A., Reddy, R.: New model-based methods and algorithms for performance and energy optimization of data parallel applications on homogeneous multicore clusters. IEEE Trans. Parallel Distrib. Syst. **28**, 1119–1133 (2017)
25. Reddy, R., Lastovetsky, A.: Bi-objective optimization of data-parallel applications on homogeneous multicore clusters for performance and energy. IEEE Trans. Comput. **67**, 160–177 (2018)
26. Khaleghzadeh, H., Reddy, R., Lastovetsky, A.: A novel data-partitioning algorithm for performance optimization of data-parallel applications on heterogeneous HPC platforms. IEEE Trans. Parallel Distrib. Syst. **29**, 2176–2190 (2018)
27. O'Brien, K., Petri, I., Reddy, R., Lastovetsky, A., Sakellariou, R.: A survey of power and energy predictive models in HPC systems and applications. ACM Comput. Surv. **50**, 37 (2017)
28. Shahid, A., Fahad, M., Manumachu, R.R., Lastovetsky, A.: Additivity: a selection criterion for performance events for reliable energy predictive modeling. Supercomput. Front. Innov. **4**, 50–65 (2017)