Parallelization of Selected Algorithms on Multi-core CPUs, a Cluster and in a Hybrid CPU+Xeon Phi Environment

Adam Krzywaniak(✉) and Paweł Czarnul

Faculty of Electronics, Telecommunications and Informatics,
Gdansk University of Technology, Gdansk, Poland
adam.krzywaniak@pg.gda.pl, pczarnul@eti.pg.gda.pl

Abstract. In the paper we present parallel implementations as well as execution times and speed-ups of three different algorithms run in various environments such as on a workstation with multi-core CPUs and a cluster. The parallel codes, implementing the master-slave model in C+MPI, differ in computation to communication ratios. The considered problems include: a genetic algorithm with various ratios of master processing time to communication and fitness evaluation times, matrix multiplication and numerical integration. We present how the codes scale in the aforementioned systems. For the numerical integration code that scales very well we also show performance in a hybrid CPU+Xeon Phi environment.

Keywords: Parallel programming · Multi-core CPU · Cluster · Intel Xeon Phi · Parallelization

1 Introduction

In the current high performance computing landscape, there are a variety of powerful compute devices that can be exploited with parallel programming. This includes multi-core CPUs such as Intel Xeons with typically 2 CPUs per workstation with up to 48 cores and 96 threads with Xeon E7-8894 v4 CPUs, up to 72 physical cores of the Xeon Phi x200 7290 processor. Apart from such CPUs, typically workstations and cluster nodes are equipped with GPUs. The latest NVIDIA V100 features 5120 CUDA cores and 16 GB HBM2 VRAM.

In terms of parallel programming paradigms, several can be distinguished including: master-slave, geometric parallelism, pipelining and divide-and-conquer. In this paper, we focus on three parallel master-slave applications that differ in compute-communication ratios and demonstrate how this affects speed-ups using various compute devices, including a multi-core CPU, a cluster and a hybrid CPU+Xeon Phi x100 coprocessor environment. This allows readers to predict speed-ups of other parallel applications with similar compute-communication ratios. Furthermore, it is a step towards building a precise model for such applications and systems in the MERPSYS execution time and energy simulation environment [1, 2].

The structure of the paper is as follows. Section 2 contains review of related works on parallelization of master-slave computations. Section 3.1 presents our testbed environments while Sects. 3.2, 3.3 and 3.4 our three representative parallel applications along with results of test runs performed in the aforementioned environments. Section 4 includes conclusions that summarize obtained results and link to the future work based on this paper – simulation of created applications in the MERPSYS environment designed for modeling and simulation of execution time and energy consumption [1, 2].

2 Related Work

There are several works that address parallelization and scheduling of master-slave processing schemes in high performance computing environments. These are possible with several programming APIs such as MPI, OpenMP and CUDA [3].

Paper [4] deals with off-line and on-line scheduling on heterogeneous master-slave platforms, including metrics such as total completion time, makespan, maximum response time. The authors have concluded that heuristics taking into consideration communication links' parameters offer best results. Experiments were conducted with MPI.

Work [5] presents very interesting theoretical modeling of evolutionary master-slave computing in terms of speed-ups for assumed both network and processing parameters. The authors present both theoretical and practical experiments showing almost linear speed-ups for around 100 slaves, however with an assumption that fitness evaluation in a slave requires 0.25 s/individual. The authors have used Distributed BEAGLE - their implementation of the master-slave architecture. The authors considered a Beowulf cluster made of homogeneous computers and a 100 Mbits/s Ethernet switch.

Paper [6] analyzes parallelization of a genetic algorithm for image restoration including various components that contribute to execution time: computations to be parallelized on the slave part T_np, sequential computations on the master T_nq as well as communication time T_nc. It has been concluded that observable parallelization can be seen on condition $T\_np \gg T\_nq + T\_nc$.

Paper [7] presents a Double-Layer Master-Slave Model (DMSM) that is targeted for distribution of independent tasks among cluster nodes (through MPI) and then processed within a node by a number of threads using OpenMP. The HPCVL developed DMSM library realized the proposed processing scheme in Fortran 90 and C. The authors presented that DMSM allows to reduce work imbalance of optimizations of the H2O2 molecule with fixed angles and varying bond lengths to 20% over 16 Sun T5140 nodes and presented benefits of their approach compared to MPI or OpenMP only solutions. It should be noted that recently, apart from the classic cluster systems with multi-core CPUs, more and more accelerators and coprocessors have been engaged for parallel processing. On one hand, such devices such as GPUs or Intel Xeon Phi x100 coprocessors (many-core CPUs in the latest x200 series) offer very good performance/Watt. On the other hand, exploiting full potential of such devices is not easy because it requires highly scalable code due to many more but less powerful cores compared to a standard multi-core CPU.

Paper [8] analyzes both performance and power consumption of several applications on an Intel Xeon Phi coprocessor, a SandyBridge Xeon CPU and Tesla C2050. The tests use the SHOC application benchmark. Tests for FFT, GEMM, MD and reduction show limits up to which the applications scale.

Paper [9] presents KernelHive – a framework for parallelization of computations and data - a set of independent data chunks and OpenCL processing kernels are distributed and scheduled across compute devices such as CPUs and GPUs in a cluster or even among clusters. The system is able to optimize kernel configurations including the numbers of groups and work items and take communication costs into consideration. The paper presents scalability of a parallel MD5 password-breaking application using brute force on a cluster with 16 nodes and a heterogeneous configuration with various CPUs and GPUs. A similar application is presented in [10] for encryption and decryption of large amounts of data using for various CPUs, GPUs and in a cluster environment, demonstrating good scalability for up to 4 nodes.

Paper [11] presents parallel computation of similarity measures between large vectors in a hybrid environment with multi-core CPUs and Intel Xeon Phi coprocessors. It is demonstrated how to best partition input data in such an environment, what the best numbers of threads per CPUs and Xeon Phis are as well as gain from overlapping communication and computations. It is shown that the implementation benefits from adding new compute devices in a heterogeneous environment.

Papers [12, 13] analyzed master-slave parallelization of matrix multiplication and numerical integration for various numbers of nodes and various data sizes.

3 Applications and Experiments

3.1 Testbed Environments

As two default testbed environments we used a workstation with two multi-core CPUs and a cluster of machines with multicore CPUs, both located at the Faculty of Electronics, Telecommunications and Informatics, Gdansk University of Technology, Poland.

The multi-core workstation includes two Intel® Xeon™ CPUs E5-2680 v2 with 10 2.80 GHz physical cores with HyperThreading, 25 MB Cache and 128 GB RAM. It is running Linux kernel version 2.6.32. We used Intel's implementation of MPI and launched 1 master and slaves on physical cores. In the case of one of the three master-slave applications analyzed and benchmarked in this paper we extended the aforementioned multi-core testbed environment with 2 Intel® Xeon Phi™ coprocessors which consist of 60 physical 1.052 GHz cores each with the possibility to use four logical processors per one core. We call this environment hybrid because of various performances of the CPUs and the Xeon Phis.

The cluster environment consists of 3 racks, each with 36 Intel® Xeon™ E5345 CPUs with 4 physical 2.33 GHz cores and HyperThreading, 4 MB cache and 8 GB RAM. Each node is running Linux kernel 2.6.32. For the cluster we used Open MPI. It should be noted that the performance of the workstation CPU is considerably higher than that of the cluster node CPU.

3.2 Parallel Genetic Algorithm

The first representative application is a parallel genetic algorithm in which the master is in charge of execution of successive iterations with successive populations. The master sends chromosomes for evaluation to slave processes. Slaves calculate the value of the fitness function and send the best individual back to the master. The master, based on results received from slaves, generates a new population and redistributes new individuals to slaves to repeat the procedure with a new generation. The application runs for given number of generations and returns the best achieved result.

Scalability of the solution will largely depend on the ratio of the time spent by the master (t_m) compared to the sum of the time of calculations performed by a slave (t_s) with the time of master-slave communication (t_c). In our initial Traveling Salesman Problem implementation using the genetic algorithm this ratio is large because fitness evaluation is just computing the path length. But it is easy to imagine that the length of the route between combinations of nodes (cities) is not the only factor that could be optimized. Other examples of fitness functions for the same set of data prepared by master (combination of nodes) could be evaluating e.g. difficulty level or cost of each path. Optimal conditions for different criteria for the same combination of nodes result in a complex optimization problem that was modeled by us in the paper. Because of that we tested a few ratios of (t_m/(t_c+t_s)) (calculated for 1 master+1 slave configuration). As a representative we have chosen solution of a problem for 50 nodes run for 500 generations of the genetic algorithm. The size of the population was set to 2000. Different ratios of (t_m/(t_c+t_s)) were obtained by multiple execution of fitness function during fitness evaluation in a slave node for one individual. Complexity of the optimization problem was modeled for 1, 100, 500 and 1000 loops of fitness function evaluation. Various ratios were marked respectively in the figures showing the tendency of

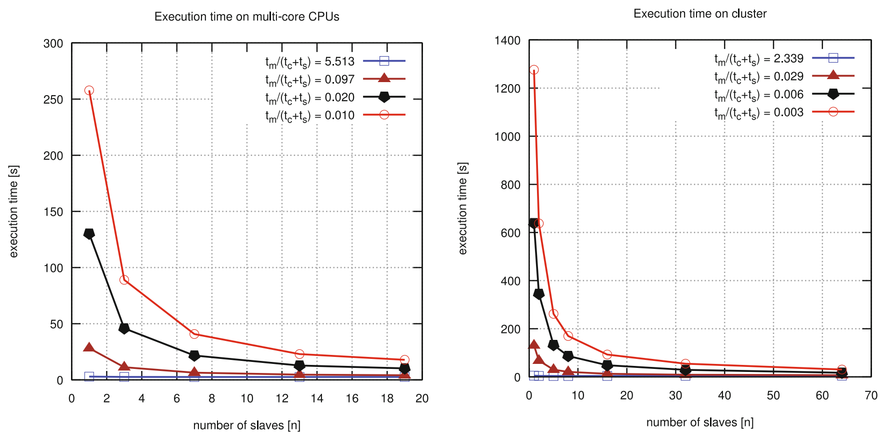

Fig. 1. Execution time of genetic algorithm application for various ratios of (t_m/(t_c+t_s)) for multi-core CPUs (left chart) and cluster (right chart) versus the number of slaves.

improving scalability and speed-up for decreasing (t_m/(t_c+t_s)) ratios. The application was run on both of the two default testbed environments mentioned in Sect. 3.1. Figure 1 presents execution times while Fig. 2 presents speed-ups.

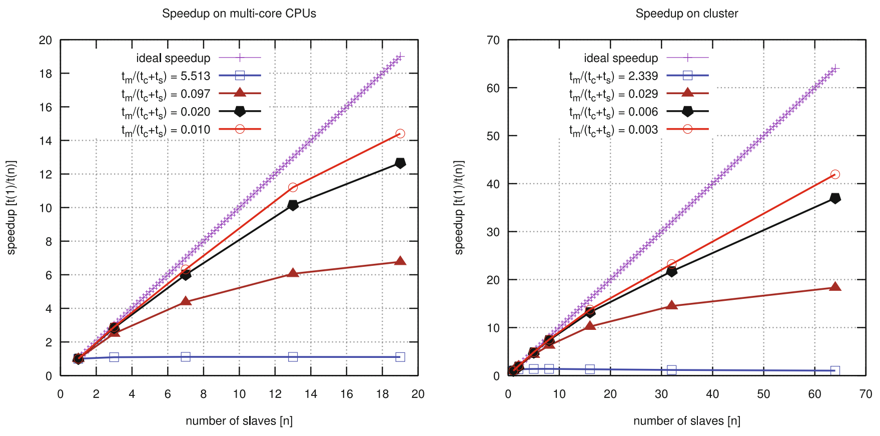

Fig. 2. Speedup of genetic algorithm parallel application for various ratios of (t_m/(t_c+t_s)) for multi-core CPUs (left chart) and on cluster (right chart) versus the number of slaves.

3.3 Parallel Algorithm for Matrix Multiplication

Matrix multiplication is the next problem that was considered by us in this paper in the context of parallelization in order to shorten the time of calculations. For the purpose of resolving this problem we have implemented another master-slave application. The algorithm we used assumes division of an output result matrix into sub-matrices and calculation of each sub-matrix using the standard algorithm. In our solution the master loads two multiplied input matrices and distributes the data required for each sub-matrix calculation among all the slaves. Slaves are designed to calculate sub-matrices (of a given size) of the result matrix. In order to be able to perform the calculations the master must prepare a data package containing proper columns and rows from two input matrices. A slave sends the result sub-matrix back to master and waits for the next calculation task. Therefore, the size of sub-matrix given to slaves to be calculated impacts strictly the size of data exchanged by a pair of master and slave in each iteration. That also – as in the previous genetic algorithm example, results in different communication to calculations ratios which affects the final scalability of the application.

For the purposes of this paper we chose multiplication of two square matrices of size 8000 × 8000. The size of the considered computational problem was determined by the least size of RAM available in considered testbed environments. The sizes of sub-matrices calculated by slaves varied during test runs starting from 800 × 800, through 400 × 400, 200 × 200 down to 100 × 100. Figure 3 presents execution times obtained during test runs for each sub-matrix size while Fig. 4 presents speedups. Both present the results for various numbers of slaves. The application was run in both testbed environments. The best scalability and execution times were obtained with the largest sub-matrices (800 × 800).

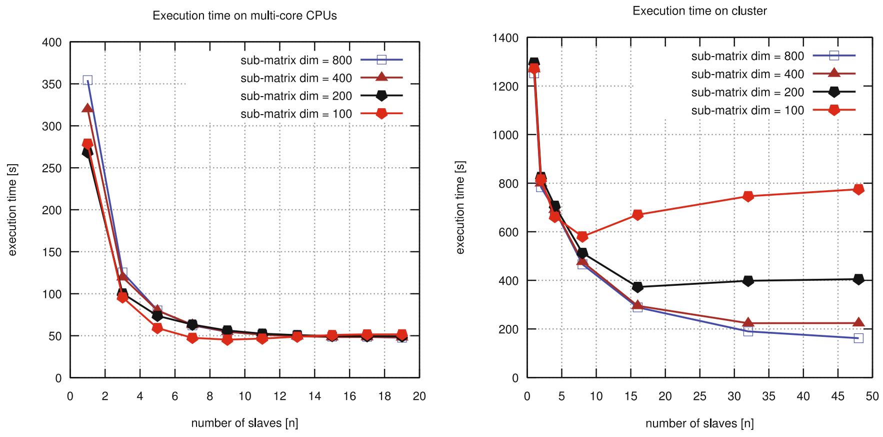

Fig. 3. Execution time for the matrix multiplication application in a multi-core testbed environment (left chart) and in a cluster (right chart) in relation to number of slaves.

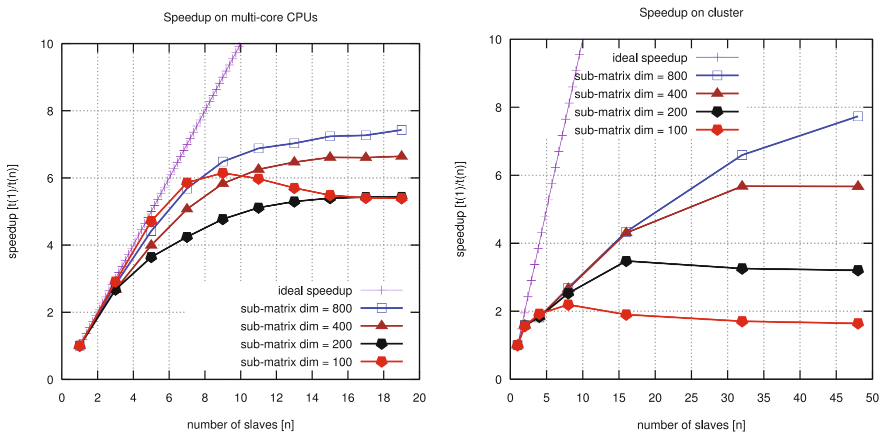

Fig. 4. Speedup for the parallel matrix multiplication application in a multi-core testbed environment (left chart) and in a cluster (right chart) in relation to number of slaves.

3.4 Parallel Algorithm of Integrate Calculation

The third considered problem is numerical integration. We implemented numerical integration in a master-slave paradigm as before, with a trapezoidal rule for computation of subranges. The master is responsible for distribution of subranges to be calculated by slaves as long as there are any left. Slaves perform integration of a given function for each subrange received from the master and send the result back. Complexity of computation blocks in slave nodes depends on the given accuracy of calculation set by PRECISION parameter. The tests were run for several values of the aforementioned parameter.

The results presented below were obtained for integration of the exp(sin(x)) function in the range of [−5,1500]. The PRECISION parameter during test runs was decreased 100 times per each run starting from the value 0.1 down to 0.0000001 and the size of subranges sent to the slaves was fixed and set to 1 in each case. Therefore, complexity of slaves' computation blocks was increased 100 times in each next test run while the communication and size of exchanged data remained the same. The communication to computation ratios were different again in each of the presented test runs.

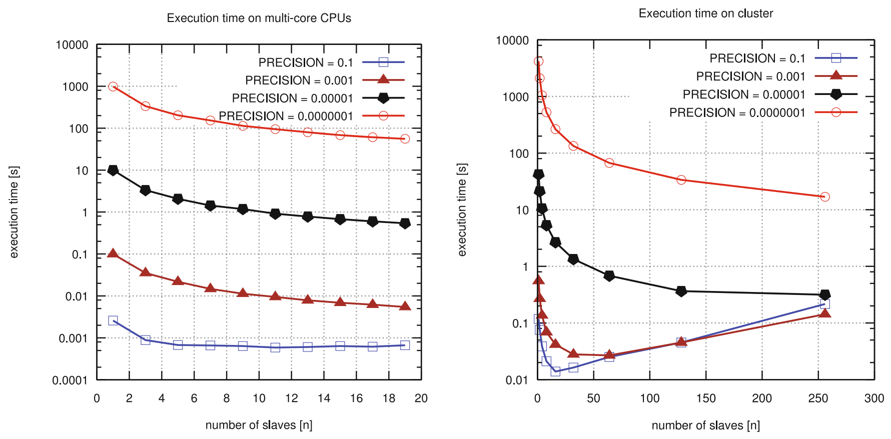

Fig. 5. Execution time for the integration application in a multi-core testbed environment (left chart) and in a cluster (right chart) in relation to number of slaves.

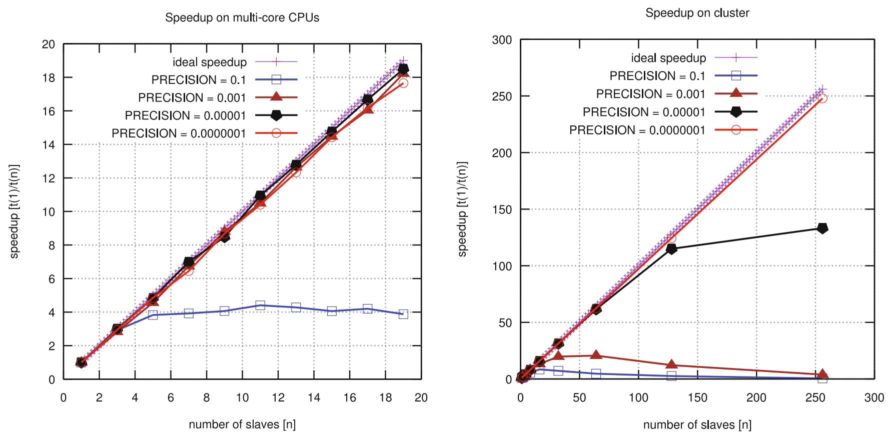

Fig. 6. Speedup for the integration in a multi-core testbed environment (left chart) and in a cluster (right chart) in relation to number of slaves.

The aforementioned application was run not only in both default testbed environments but also in a hybrid environment i.e. with two multi-core Intel Xeon CPUs with one or even two Intel Xeon Phi coprocessors added. The execution times and speedup values for the multi-core testbed environment and the cluster are presented in Figs. 5 and 6 respectively. The results allow to compare execution times as well as scalability between testbed environments also in the computation complexity context.

In the hybrid testbed environment we run the tests only for the best scaling case with the PRECISION parameter set to 0.0000001. The execution times and speedups for aforementioned hybrid testbed environment are presented in Figs. 7 and 8 respectively. It should be noted that the speed-up values were calculated against the performance of a single CPU core which is much more powerful than a single Phi core added for new slaves run on the Phi. Nevertheless we can see decrease in execution times and improvement of speed-ups when adding more slave threads that utilize cores of the first and the second Intel Xeon Phi coprocessor.

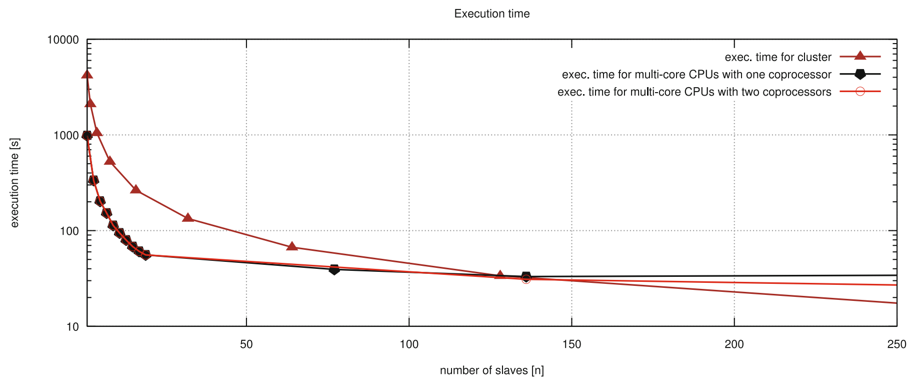

Fig. 7. Execution time for the integration application in the hybrid multi-core CPUs+Xeon Phi coprocessor(s) testbed environment in relation to number of slaves.

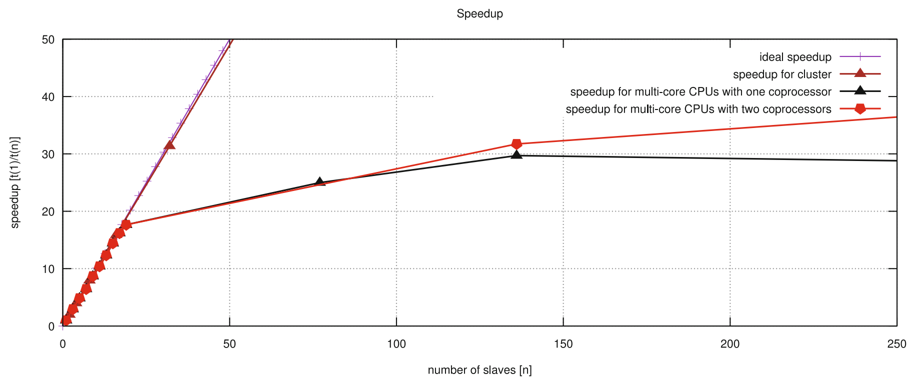

Fig. 8. Speedup for the integration application in the hybrid multi-core CPUs+Xeon Phi coprocessor(s) testbed environment in relation to number of slaves.

4 Summary and Future Work

In the paper we presented parallel implementations of three different C+MPI master-slave applications that differ in computation to communication ratios. The applications include a genetic algorithm, matrix multiplication and numerical integration. We ran experiments in two environments: a workstation with multicore CPUs and a cluster with nodes with multicore CPUs.

For the genetic algorithm as well as parallel matrix multiplication and numerical integration we presented conditions that are needed for obtaining good speed-ups in a parallel environment. Parallel matrix multiplication appeared to be a middle scaling application while the embarrassingly parallel numerical integration was scaling very well not only in a shared memory multicore CPUs machine and in a cluster but also in a hybrid CPUs+Xeon Phis environment.

Obtained speed-ups and growth for the genetic algorithm are in line with observations from [6], which suggests that good values are possible on condition that fitness evaluation is considerable compared to communication and synchronization costs in terms of time required. Speed-ups and their growth obtained for the matrix multiplication and numerical integration are similar to those obtained in [12] and in [13] respectively. In [13] adaptive integration generated subranges recursively based on a particular function. The subrange integration phase, however, is analogous to the one performed in this work. Differences in values stem from various startup times, bandwidths, synchronization costs, specific to the given environment.

We implemented the algorithms and obtained results as representative (in terms of various speed-ups) applications to be used next in our MERPSYS simulator for modeling and simulation of execution time and energy consumption [1, 2] for even greater sizes of input data and sizes of the environment. Another outcome of this work to readers is the possibility to assess potential scalability of such frequently used algorithms in various modern parallel environments.

References

1. Czarnul, P., Kuchta, J., Matuszek, M., Proficz, J., Rościszewski, P., Wójcik, M., Szymański, J.: MERPSYS: an environment for simulation of parallel application execution on large scale HPC systems. Simul. Model. Pract. Theor. **77**, 124–140 (2017). doi:10.1016/j.simpat.2017.05.009. Elsevier
2. Czarnul, P., Kuchta, J., Rościszewski, P., Proficz, J.: Modeling energy consumption of parallel applications. 2016 Federated Conference on Computer Science and Information Systems (FedCSIS), Gdansk, pp. 855–864 (2016)
3. Barlas, G.: Multicore and GPU Programming: An Integrated Approach. Morgan Kaufmann Publishers Inc., San Francisco (2014). ISBN: 9780124171404
4. Pineau, J.F., Robert, Y., Vivien, F.: Off-line and on-line scheduling on heterogeneous master-slave platforms. In: 14th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP 2006) (2006). doi:10.1109/PDP.2006.49
5. Dubreuil, M., Gagne, C., Parizeau, M.: Analysis of a master-slave architecture for distributed evolutionary computations. IEEE Trans. Syst. Man Cybern. Part B (Cybern.) **36**(1), 229–235 (2006). doi:10.1109/TSMCB.2005.856724

6. Chen, Y.-W., Nakao, Z., Fang, X.: Parallelization of a genetic algorithm for image restoration and its performance analysis. In: Proceedings of IEEE International Conference on Evolutionary Computation, Nagoya, pp. 463–468 (1996). doi:10.1109/ICEC.1996.542645
7. Liu, G., Schmider, H., Edgecombe, K.E.: A hybrid double-layer master-slave model for multicore-node clusters. J. Phys. Conf. Ser. **385**(1), 1–7 (2012)
8. Li, B., Chang, H.-C., Song, S., Su, C.-Y., Meyer, T., Mooring, J., Cameron, K.W.: The power-performance tradeoffs of the Intel Xeon Phi on HPC applications. In: Proceedings of the 2014 IEEE International Parallel & Distributed Processing Symposium Workshops (IPDPSW 2014), Washington, DC, USA, pp. 1448–1456. IEEE Computer Society (2014). doi:http://dx.doi.org/10.1109/IPDPSW.2014.162
9. Rościszewski, P., Czarnul, P., Lewandowski, R., Schally-Kacprzak, M.: KernelHive: a new workflow-based framework for multilevel high performance computing using clusters and workstations with CPUs and GPUs. Concurrency Comput. Pract. Exper. **28**, 2586–2607 (2016). doi:10.1002/cpe.3719
10. Niewiadomska Szynkiewicz, E., Marks, M., Jantura, J., Podbielski, M.: A hybrid CPU/GPU cluster for encryption and decryption of large amounts of data. J. Telecommun. Inf. Technol. **3**, 32–39 (2012)
11. Czarnul, P.: Benchmarking performance of a Hybrid Intel Xeon/Xeon Phi system for parallel computation of similarity measures between large vectors. Int. J. Parallel Program. **45**, 1091–1107 (2016). doi:10.1007/s10766-016-0455-0. Springer
12. Datti, A.A., Umar, H.A., Galadanci, J.: A beowulf cluster for teaching and learning. Procedia Comput. Sci. **70**, 62–68 (2015). doi:10.1016/j.procs.2015.10.034. ISSN: 1877-0509
13. Czarnul, P.: Parallelization of compute intensive applications into workflows based on services in BeesyCluster. Scalable Comput. Pract. Experience **12**(2), 227–238 (2011). ISSN: 1895-1767