This is a post-peer-review, pre-copyedit version of an article published in **11th International Conference on Parallel Processing and Applied Mathematics (PPAM 2015).** The final authenticated version is available online at: https://doi.org/10.1007/978-3-319-32152-3_19

Postprint of: Proficz J., Czarnul P., Performance and Power-Aware Modeling of MPI Applications for Cluster Computing. In: Wyrzykowski R., Deelman E., Dongarra J., Karczewski K., Kitowski J., Wiatr K. (eds) Parallel Processing and Applied Mathematics. Lecture Notes in Computer Science, vol 9574 (2016), pp. 199-209, Springer, https://doi.org/10.1007/978-3-319-32152-3_19

Performance and Power-Aware Modeling of MPI Applications for Cluster Computing

Jerzy Proficz(✉) and Paweł Czarnul(✉)

Academic Computer Center TASK, Faculty of Electronics, Telecommunications and Informatics, Gdańsk University of Technology, Gdańsk, Poland
jerp@task.gda.pl, pczarnul@eti.pg.gda.pl

Abstract. The paper presents modeling of performance and power consumption when running parallel applications on modern cluster-based systems. The model includes basic so-called blocks representing either computations or communication. The latter includes both point-to-point and collective communication. Real measurements were performed using MPI applications and routines run on three different clusters with both Infiniband and Gigabit Ethernet interconnects. Regression allowed to obtain specific coefficients for particular systems, all modeled with the same formulas. The model has been incorporated into the MERPSYS environment for modeling parallel applications and simulation of execution on large-scale cluster and volunteer based systems. Using specific application and system models, MERPSYS allows to predict application execution time, reliability and power consumption of resources used during computations. Consequently, the proposed models for computational and communication blocks are of utmost importance for the environment.

Keywords: Performance model · Energy consumption · Cluster computing · MPI

1 Introduction

Modern parallel systems have increased in sizes considerably in recent years. The most powerful cluster on the TOP500 list – Tianhe-2 features over 3 million cores, offers over 33 PFlop/s performance but at over 17 MWatts of power consumption[1]. It should be noted that growth in performance of such parallel systems stems from incorporation of more and more computational cores into the system. At the same time, such large clusters, due to a large number of components, are prone to failures. This may effectively impact execution times of parallel applications due to necessary checkpoints and restarts in order to continue from the last consistent application state. For instance, for Sequoia the reported failure rate reaches 1.25 per day [1]. Consequently, it is of utmost

importance for developers to be able to assess application running times and speed-ups taking into account not only the design and bottlenecks in the application but also potential failures of such large scale cluster systems. On one hand, if we assume a given input data size for an application then speed-up will be dependent on the ratio of computations to communication, synchronization and specific optimization techniques such as overlapping communication and computations, piggybacking etc. On the other hand, hardware parameters such as CPU, GPU performance as well as latency and bandwidth of an interconnect will also impact the speed-up. The aforementioned failure possibilities will further limit performance because if a failure occurs then the application will need to be restarted either from scratch or from the last saved checkpoint. Consequently, a question should be asked what would be the optimal number of CPUs in order to minimize the application running time given this constraints. Furthermore, power-aware metrics are considered nowadays apart from performance only. For instance, one of considered optimization goals is minimization of application running time with a constraint on the total power consumption used by computing devices [2]. Consequently, a good model for parallel applications in a cluster based environment is still crucial for optimization, especially if it addresses power-aware aspects along with performance. Our main contribution is to provide a model related to the cluster performance, power consumption and reliability estimations. We designed the set of formulas working exactly for our simulation tool, we tested them and tuned for specific real environments.

2 Background and Related Work

A cluster model, or more generally, a hardware model needs to be introduced in every simulator of running an application in a parallel environment.

GSSIM [3] provides a configurable solution, where it is possible to use the default mode, where computation times are simply calculated as a linear function of the processor clock, and communication times are analytically solved according to the used network devices, their latencies and bandwidths. The power consumption is based on the three schemes: constant where a device always consumes the same power, the resource based, usually using values for idle and full utilization of the resource and the application related where the exact values need to be provided by the user. The model considered in this work is more focused on the cluster environment, thus it concerns such operations like disk data transfers or HyperThreading out-of-the box, without additional user configuration.

There are two main analytical models of the communication behavior in the network of computation nodes, the one proposed by Hockney in [4] and LogP [5]. The former assumes the time of the message passing between nodes equals $L + m/B$, where L is a latency of the network, B is a network bandwidth and m is a message size. The latter assumes the message delivery time equals $L + 2o$, where L is latency of the network, o is an overhead and an additional parameter describing the modeled system: P – the number of nodes communicating each other. However, it also assumes that the next message cannot be sent during the gap time, denoted by g, thus the network can carry messages $\lfloor L/g \rfloor$ at the time.

While the gap parameter in the LogP model reflects the network contention, it is suited for short messages only. Therefore, many variants were proposed. The LogGP model [6] introduces an additional parameter: G – gap per byte, thus the time of sending m bytes between two nodes can be presented as $o + (m-1)G + o$. The PLogP model [7] introduces additional dependency of the gap and overhead parameters on the message size, and additionally distinguishes overhead for sending and receiving the message: $g(m)$, $o_s(m)$ and $o_r(m)$.

The HLoGP model [8] provides support for heterogeneous environment introducing the parameter matrices instead of scalar model parameters, e.g. $L = \{L_{11}, \ldots, L_{MM}\}$, where M is a number of the nodes (which all can be different from each other) and an additional vector reflecting the differences between computational power of the nodes, i.e. $P = \{P_1, \ldots, P_M\}$. Similarly the computational power of the processor/cores is also considered in MLogP model [9] where multicore processor architecture is taken into account. Finally Log_nP models [10] enable the hierarchical performance analysis for layered systems, including the impact of the memory and middleware on distributed communication.

3 Simulation Environment for Parallel Applications Running on Cluster-Based Systems

Within project "Modeling efficiency, reliability and power consumption of multi-level parallel HPC systems using CPUs and GPUs" sponsored by and covered by funds from the National Science Center in Poland we created an environment for simulation of parallel applications run on large-scale cluster, grid and volunteer based systems. For an application run for a particular input data size on a given system, the environment returns the following:

1. Application execution time.
2. Success/failure of the application – potential hardware failures have become a concern in large scale parallel systems [1] because of the number of components.
3. Energy consumed during execution of the application thanks to considering power consumption of devices such as CPUs, GPUs and network interconnects.

The distributed architecture of the system comprises the following components:

1. A client-side system and application editor as well as a simulation panel:
   (a) System model editor (Fig. 1). A user creates a model of the system by selecting predefined computational components such as CPUs/GPUs which are interconnected using predefined network types such as WANs, LANs or buses within each node. The system model can be defined at multiple levels starting at the top from WAN through LAN up to the node/machine level. For each particular computational or network type

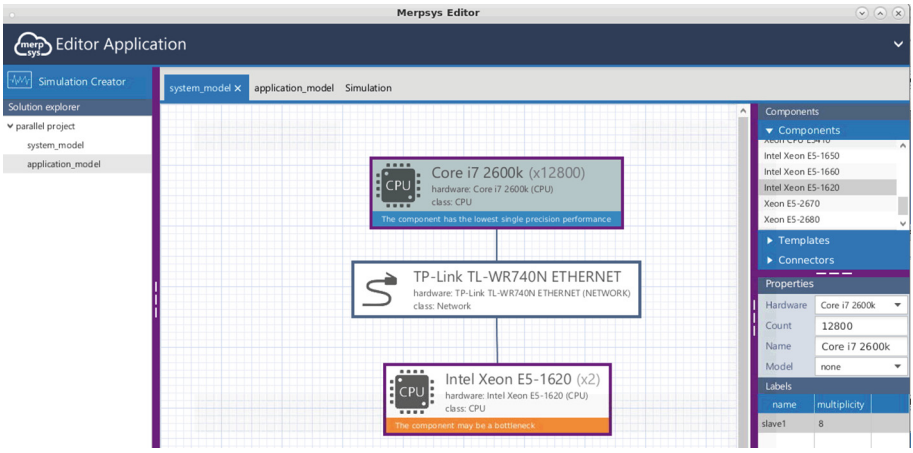

Fig. 1. System model editor – an exemplary system

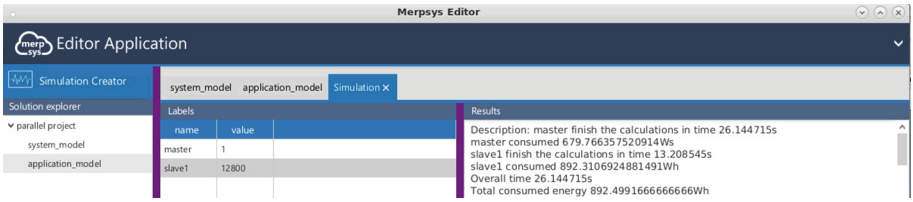

Fig. 2. Sample results

component the user selects one hardware component with specification stored in a database. This includes single, double floating point performance for CPUs/GPUs, power consumption etc. The user assigns labels to particular computational components. Specifically, if a CPU or a GPU is assigned a label e.g. "master" with cardinality 12 this means that up to 12 processes or threads marked "master" in the application can be run there. It is the scheduler described next that will decide how many processes or threads with such a label will be launched on such a computational component. An exemplary system modeling an environment with Intel Xeon E5-1620 CPUs for master and i7 2600k for slave processes is shown in Fig. 1.

(b) Application editor. A user writes code of a parallel application using a special Java type meta-language which uses a generalization of message passing paradigms such as MPI. The application includes codes of various processes/threads each marked with a distinct label such as "master", "slaveX", "slaveY" etc. The code consists of computational or communication blocks and can contain any basic Java constructs such as for, while loops, conditional instructions etc. Computational blocks take as input data size, a function that determines the number of operations vs

the input data size, optional software stack and optimizations. Communication blocks include point-to-point, barrier, broadcast, scatter, gather similar to MPI.

(c) Simulation panel (Fig. 2). The panel allows definition of the number of processes or threads with given labels and optionally a number of variables for which values would be available from within the application code. The panel allows starting a simulation and display of results. Figure 2 shows exemplary results for running a parallel application in the MERPSYS environment.

2. System server that acts as a proxy between users and simulators. It launches and manages several simulations on a cluster in parallel. Upon termination of a simulation, the client application displays results to the user.
3. Scheduler – an application that decides where the required number of processes or threads should be launched considering slots available in the system model.
4. Simulator – an application that simulates execution of the aforementioned application on a large scale system. For each distinct label defined within the application, the simulator starts a separate thread that simulates a given number of processes/threads of this type. Proper scaling is used for both computations and communication using the given number of processes/threads. This allows simulation of thousands of processes/threads running the same code using one simulation thread. The simulator offers two advantages over running a real application: it allows consideration of application and system sizes for which a real application could not be run due to resource limitations (such as the limited RAM size), simulation time for an application can be much shorter than the running time of a real application – this is possible because of encapsulation of computations within computational blocks.
5. Hardware database which stores information about various hardware components such as specific CPUs, GPUs, interconnects within nodes (such as PCI) and among nodes (LANs, WANs etc.).

Consequently, it is important for the simulator to have detailed formulas for CPUs, GPUs and interconnects for correct prediction of execution times and energy consumption of particular blocks of code, either representing computations or communication among processes or threads of a parallel application. Additionally, a rough estimate on successful execution of an application can be derived that uses the number of the nodes involved in computations.

4 Model Formulas

Our model is based on the statistical approach. The measurements realized during the experiments reflecting each simulated block were manually compared to the commonly used formulas (e.g. in [11] for the group communication) and the closest approximation was chosen. In general we used the expert knowledge which reflects the internals of the MPI implementation and the cluster design.

We used the hardware specific implementation (e.g. Infiniband for MVAPICH) thus using of the already proposed models [11] directly could be misleading. For regression we used two different error measurements i.e.: (i) mean percentage error (MPE) in cases where the differences for small absolute value were important (e.g. sending of short messages) in the model, and (ii) a traditional mean squared error for others.

Table 1 presents the formulas of the proposed model. Their parameters are split into two groups:

1. Input parameters that need to be provided for the application model, i.e.: the number of instructions to be executed: h, the number of the active threads executed on a particular node: p_{th}, input data size: d and the number of the nodes involved in computations: P.
2. Parameters related to a specific environment, constant during the execution of an application on the modeled cluster, and dependent on the cluster hardware, software (e.g. the operating system and/or the used message library implementation) and their configuration, e.g. the number of cores in the CPU. Their values were provided directly (like the mentioned number of cores) or by the regression (for less obvious ones). Table 3 contains a complete list of the parameters.

Table 1. Model formulas

Modeled block	Execution time
Computation block (O_x)	$t_{ox} = \begin{cases} T_{min}h & \text{if } p_{th} \leq P_{low} \\ T_{low}h & \text{if } p_{th} \in (P_{low}, P_{hi}\rangle \\ (T_{hi} + K_{hi}p_{th})h & \text{if } p_{th} > P_{hi} \end{cases}$
Communication peer-to-peer (C_{p2p})	$t_{p2p} = T_{p2p} + K_{p2p}\lceil d/D_{tu}\rceil D_{tu}$
Communication broadcast (C_{bcast})	$t_{bcast} = T_{bcast} + K_{bcast}\lceil d/D_{tu}\rceil D_{tu} log(P)$
Communication scatter (C_{scat})	$t_{scat} = T_{scat} + K_{scat}\lceil d/D_{tu}\rceil D_{tu}\frac{log(P)}{P}$
Communication gather (C_{gath})	$t_{gath} = T_{gath} + K_{gath}\lceil d/D_{tu}\rceil D_{tu}\frac{log(P)}{P}$
Communication all-to-all (C_{a2a})	$t_{a2a} = T_{a2a} + K_{a2a}\lceil d/D_{tu}\rceil D_{tu}P$
Communication barrier (C_{bar})	$t_{bar} = T_{bar} + K_{bar}log(P)$
Read block from a network disk (R_{disk})	$t_{rdisk} = T_{rdisk} + K_{rdisk}\lceil d/D_{tu}\rceil D_{tu}$
Write block to a network disk (W_{disk})	$t_{wdisk} = T_{wdisk} + K_{wdisk}\lceil d/D_{tu}\rceil D_{tu}$
Modeled block	Power consumption
Any number of blocks on a single node	$pw = \begin{cases} PW_{low} + KW_{low} \times p_{th} & \text{if } p_{th} \leq P_{low} \\ PW_{hi} + KW_{hi} \times p_{th} & \text{if } p_{th} \in (P_{low}, P_{hi}\rangle \\ PW_{max} & \text{if } p_{th} > P_{hi} \end{cases}$
Modeled block	Probability of the correct execution
Any number of blocks on a set of nodes	$s(\Delta t, P) = e^{-\lambda \Delta t P}$

A computation block O_x is a basic block which represents data processing in the cluster and grid environment. We assume the most typical arithmetical

or comparison operations performed on data. Such a block can be performed sequentially or in parallel using different cores of the nodes processors.

We assumed that the time of parallel computation of a single instruction is constant (T_{min}) as long as the number of threads is lower than the number of cores, thus in this case the total time depends only of the number of executed instructions: $t_{ox} = T_{min}h$. We assume a similar constant time (T_{low}) for HyperThreading however, in this case this time is longer: $T_{low} > T_{min}$, thus $t_{ox} = T_{low}h$. Finally, for the number of threads exceeding the number of (virtual) cores, we assumed the time of the single instruction increases linearly with the number of the threads: $t_{ox} = (T_{hi} + K_{hi}p_{th})h$. Figure 3a presents the regression results with a comparison to the real measurements for an exemplary computation block of 11 million instructions executed by a single thread.

We distinguish a number of communication blocks, including direct peer-to-peer and group operations. The time for the former was assumed to be linear to the size of the message, and for the latter we relied mainly on the analytical models (e.g. provided in [11]). For both types of formulas we introduced an adjustment related to the (maximum) data transfer unit. Figures 4 and 5 present the regression results in comparison to the real measurements for peer-to-peer, barrier and broadcast blocks. Similarly, blocks related to the network disk I/O operations were modeled, and appropriate formulas were proposed. Due to their characteristics, we assumed the same time complexity as for peer-to-peer communication.

During the experiments, we observed the electrical power load being strongly dependent on the number of the active threads rather than on the type of operation (a specific computational block). Thus we proposed a model, in which the power consumed at the moment (expressed in Watts), depends on the number of used processor cores, which are directly involved in processing. For regression, we assumed segmented and linear increase of power consumption. However, we introduced a distinction between the power consumed by active threads assigned to the real and logical (HyperThreading) cores. Obviously, after passing a certain threshold additional, active threads do not introduce additional increase in power consumption. Figure 3b presents the power regression results in comparison to the real measurements.

5 Model Parameter Regression Results

The model parameters were derived for three different cluster environments: Galera+ (all parameters including power), Galera (performance parameters only) clusters, located in the Academic Computer Centre – TASK, and a KASK cluster (performance parameters only) located at the Department of Computer Architecture, Faculty of Electronics, Telecommunications and Informatics. All these machines are located at Gdańsk University of Technology. The clusters work under a Linux operating system and the measurements were performed for the MVAPICH v1.8 MPI implementation.

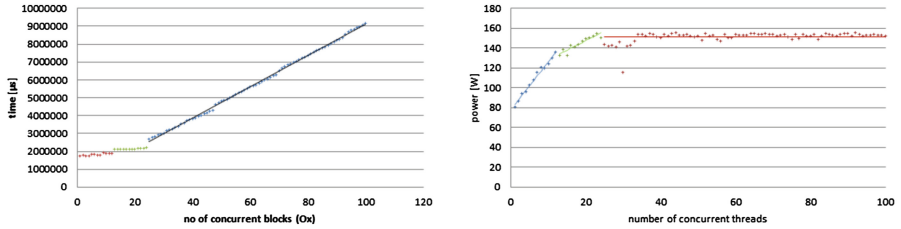

Fig. 3. Computation block regressions vs real measurements for Galera+: (a) execution time, (b) power consumption

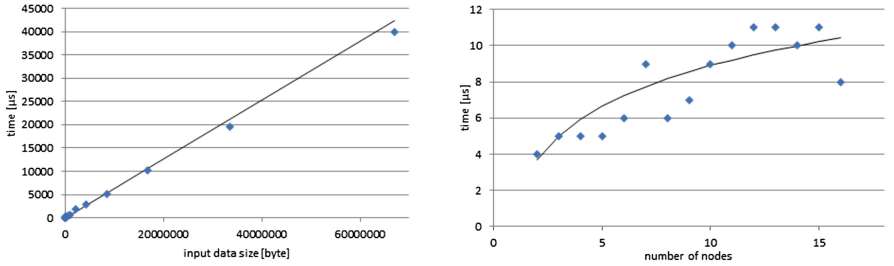

Fig. 4. Execution time regressions versus real measurements for Galera+: (a) peer-to-peer, (b) barrier

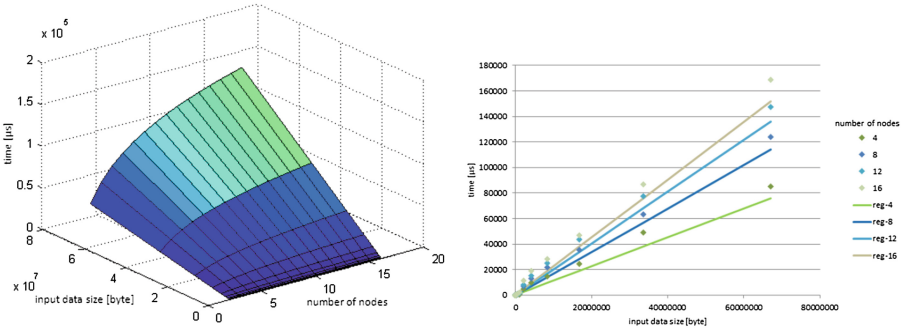

Fig. 5. Broadcast execution time regression for Galera+: (a) 3D view (b) mapping to 2D versus real measurements

The Galera+ cluster consists of 192 computation nodes. Every node is equipped with two Intel Xeon 2.27 GHz multicore processor units, with 6 physical and 12 logical (HyperThreading) cores each, 16 GB RAM, and network interface cards. The cluster uses two interconnection networks: (i) Infiniband QDR (40 Gbps) and (ii) GB Ethernet, supported by respective network switches. Additionally there is a 500 TB disk array exposed to the nodes using a Glustre remote file system. The Galera cluster consists of 672 computation nodes. Every node is equipped with two Intel Xeon 2.33 GHz multicore processor units, with 4 cores

each, 8-32 GB RAM, and network interface cards. The cluster uses two interconnection networks: (i) Infiniband DDR (20 Gbps) and (ii) GB Ethernet, supported by respective network switches. Additionally the 500 TB disk array is exposed to the nodes using a Lustre remote file system.

Table 2. Pearson coefficient squared (R^2) calculated for the regression formulas versus the real measurements

Formula	R^2 Galera+	R^2 Galera	R^2 KASK
t_{ox}	0.9993	0.9999	0.9999
t_{p2p}	0.9996	0.9998	0.9996
t_{bcast}	0.9902	0.9688	0.9102
t_{scat}	0.9643	0.9176	0.9668
t_{gath}	0.9620	0.9374	0.9823
t_{a2a}	0.9296	0.7858	0.8384
t_{bar}	0.7324	0.9223	0.8527
t_{rdisk}	0.9999	0.9971	0.9976
t_{wdisk}	0.9988	0.9999	0.9999
pw	0.9062	—	—

The KASK cluster consists of 10 computation nodes. Every node is equipped with two Intel Xeon 2.8 GHz multicore processor units, with 2 physical and 4 logical (HyperThreading) computation cores each, 4 GB RAM, and network interface cards. The cluster uses two interconnection networks: (i) Infiniband (10 Gbps) and (ii) GB Ethernet, supported by the corresponding network switches. Additionally there is a 4 TB disk array exposed to the nodes using the NFS file system. The regression was performed numerically using the gathered power and time measurements. The final results for three different clusters are presented in Table 3. Figures 3, 4 and 5 present the charts with comparison of some regression results to the real measurements and Table 2 provides the evaluation with the Pearson coefficient squared (R^2) for all estimated formulas.

6 Summary and Future Works

In the paper, we presented modeling of parallel processing in a cluster environment that includes equations representing execution times of computational and communication blocks, power consumption of such blocks as well as a simple estimate on reliability of computations. Furthermore, coefficients for these computational and communication equations were found for three clusters and power consumption and reliability indicated for Galera+ cluster located at Academic Computer Center, Gdańsk, Poland. These equations constitute an integral part of an environment that allows simulation of parallel applications running on

Table 3. Model instance parameters for Galera+, Galera and KASK clusters

Param	Galera+	Galera	KASK
P_{low}	12	8	4
P_{hi}	24	—	8
T_{min} [μs]	0.00165	0.0019	0.0033
T_{low} [μs]	0.00194	—	0.0034
T_{hi} [μs]	3.29e-04	1.16e-04	2.06e-04
K_{hi} [μs]	7.96e-05	2.35e-04	4.14e-04
D_{tu} [B]	2048	2048	2048
T_{p2p} [μs]	3.7	1.3	1.3
K_{p2p} [μs/B]	0.00063	0.00132	0.00181
T_{bcast} [μs]	1.7	-1.1	-1.04
K_{bcast} [μs/B]	0.00188	0.00199	0.0580
T_{scat} [μs]	5.0	1.1	2.9
K_{scat} [μs/B]	0.00340	0.00820	0.00880
T_{gath} [μs]	8.1	16.4	10.0
K_{gath} [μs/B]	0.00346	0.00100	0.00990

Param	Galera+	Galera	KASK
T_{a2a} [μs]	7.5	18.3	10.1
K_{a2a} [μs/B]	0.00012	0.00012	0.00048
T_{bar} [μs]	1.4	-1.2	-1.8
K_{bar} [μs]	7.5	20.6	22.6
T_{rdisk} [μs]	5200	1390	7600
K_{rdisk} [μs/B]	0.00253	0.0324	0.0290
T_{wdisk} [μs]	1200	270	1100
K_{wdisk} [μs/B]	0.00474	0.0066	0.0093
PW_{low} [W]	78	—	—
KW_{low} [W]	4.84	—	—
PW_{hi} [W]	109	—	—
KW_{hi} [W]	1.90	—	—
PW_{max} [W]	151	—	—
λ	5.03372 e-10	—	—

large scale systems and prediction of execution time, potential failures and energy consumed during a run. The environment with its basic components such as a system editor, an application editor and a simulation panel with sample results were presented. The system editor and consequently the simulator use the presented equations for simulation of runs of modeled parallel applications on the three aforementioned clusters. This in turn allows estimation of execution times, power consumption and reliability for various applications and configurations including the number of nodes run on these clusters.

Acknowledgments. The work was performed within grant "Modeling efficiency, reliability and power consumption of multilevel parallel HPC systems using CPUs and GPUs" sponsored by the National Science Center in Poland based on decision no DEC-2012/07/B/ST6/01516.

References

1. Dongarra, J.: Emerging heterogeneous technologies for high performance computing. In: Heterogeneity in Computing Workshop (2013). http://www.netlib.org/utk/people/JackDongarra/SLIDES/hcw-0513.pdf
2. Czarnul, P., Rościszewski, P.: Optimization of execution time under power consumption constraints in a heterogeneous parallel system with GPUs and CPUs. In: Chatterjee, M., Cao, J., Kothapalli, K., Rajsbaum, S. (eds.) ICDCN 2014. LNCS, vol. 8314, pp. 66–80. Springer, Heidelberg (2014)
3. Bak, S., Krystek, M., Kurowski, K., Oleksiak, A., Piatek, W., Weglarz, J.: GSSIM - a tool for distributed computing experiments. sci. program. **19**, 231–251 (2011)
4. Hockney, R.W.: The communication challenge for mpp: Intel paragon and meiko cs-2. Parallel Comput. **20**, 389–398 (1994)

5. Culler, D., Karp, R., Patterson, D., Sahay, A., Schauser, K.E., Santos, E., Subramonian, R., von Eicken, T.: Logp: towards a realistic model of parallel computation. In: Proceedings of the Fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPOPP 1993, pp. 1–12. ACM, New York (1993)
6. Alexandrov, A., Ionescu, M.F., Schauser, K.E., Scheiman, C.: Loggp: Incorporating long messages into the logp model—one step closer towards a realistic model for parallel computation. In: Proceedings of the Seventh Annual ACM Symposium on Parallel Algorithms and Architectures, SPAA 1995, pp. 95–105. ACM, New York (1995)
7. Kielmann, T., Bal, H.E., Verstoep, K.: Fast measurement of LogP parameters for message passing platforms. In: Rolim, J.D.P. (ed.) IPDPS-WS 2000. LNCS, vol. 1800, pp. 1176–1183. Springer, Heidelberg (2000)
8. Bosque, J.L., Perez, L.P.: Hloggp: a new parallel computational model for heterogeneous clusters. In: CCGRID, pp. 403–410. IEEE Computer Society (2004)
9. Chui, C.K.: The logp and mlogp models for parallel image processing with multicore microprocessor. In: Proceedings of the 2010 Symposium on Information and Communication Technology, SoICT 2010, pp. 23–27. ACM, New York (2010)
10. Cameron, K.W., Ge, R., Sun, X.: $\log_n p$ and $\log_3 p$: accurate analytical models of point-to-point communication in distributed systems. IEEE Trans. Comput. **56**, 314–327 (2007)
11. Pjesivac-Grbović, J., Fagg, G.E., Angskun, T., Bosilca, G., Dongarra, J.J.: Mpi collective algorithm selection and quadtree encoding. In: Proceedings of the 13th European PVM/MPI Users' Group Meeting, Bonn, Germany (2006)