Benchmarking of High Performance Computing Clusters with Heterogeneous CPU/GPU Architecture

Pavel V. Sukharev[1], Nikolay P. Vasilyev[2], Mikhail M. Rovnyagin[3], Maxim A. Durnov
Department of Computer Systems and Technologies
National Research Nuclear University MEPhI (Moscow Engineering Physics Institute)
Moscow, Russia
[1]suharev_p@mail.ru, [2]npvasiliev@mephi.ru, [3]m.rovnyagin.2015@ieee.org

Abstract— **The most modern supercomputers are clusters of computing nodes with a heterogeneous architecture; in such a node we can see both classical CPUs and specialized computing coprocessors (GPU- or MIC-based).**

Modern benchmark tests are oriented either for CPU or for specific coprocessor calculations - but not in a whole thing.

This article is dedicated to describe an approach to build a unified performance test for a heterogeneous HPC cluster with CPU/GPU computing nodes.

Keywords— High performance computing cluster, Benchmarking, GPGPU, NVIDIA CUDA, Heterogeneous system

I. Introduction

A computing cluster is a collection of computers united by high-speed links and formed into a single hardware system used for high-performance computing. Computing clusters are characterized by high scalability, reliability and the capability to easily replace a failed node. Computing clusters are currently the most popular solution in the field of high performance computing. A hybrid CPU/GPU architecture, which implies both central and graphics processing units used for calculations, is the most common and effective computing cluster architecture.

The difference between CPU and GPU is that the central processing unit is composed of a small number of cores with a complex architecture operating at a high clock frequency, enabling better handling of separate threads of sequential instructions, while the graphics processing unit is composed of a large number of basic cores, enabling the processor quickly solve highly parallelized problems [1]. As a result, according to numerous studies, for example, in ref. [2], GPU, unlike CPU, provides a significantly better performance when dealing with highly parallelized problems. As shown in [3], behavioral patterns of these two processor types are also different: for example, while CPU requires frequent access to the memory, GPU does it less frequently, but sends considerably larger amounts of information. Hybrid CPU/GPU architecture allows combining the advantages of both processor types, using CPU for sequential computing and GPU – for parallel computing.

To carry out computing on GPU, special hardware and software solutions helping developers implement algorithms for GPUs are required [4]. NVIDIA Corporation, whose graphics processing units are the most popular in the field of high performance computing, provides the developers with the CUDA platform, which allows implementing algorithms for NVIDIA GPUs using a simplified C language dialect [5].

To determine a number of computing cluster characteristics, specially designed software tests are used. Cluster testing solves a number of problems, such as self-diagnostics, system "bottlenecks" determination, and comparison with similar computing systems, optimal computing algorithm discovery and performance computing to solve a particular problem. For example, [6] describes a special software test designed for the system performance assessment while solving Lattice Gauge Theory tasks.

II. Problem Statement

High Performance Linpack test (HPL) [7], which is the Linpack benchmark implementation [8] for multiprocessor distributed memory systems, is currently the most popular parallel computing systems benchmark test. The test involves solving a system of linear algebraic equations using LU-factorization. The test results in the parallel computing system performance assessment in FLOPS – floating operations per second. This test is used to make a TOP500 list of the most productive computing systems [9].

However, according to numerous studies, for example, ref. [10-12], Linpack benchmark calculates results far from the system peak performance. This is due to the lack of test optimization for particular computing system architecture. This problem gets exacerbated when dealing with heterogeneous computing clusters with a hybrid architecture, as they have a complex and heterogeneous set of computing elements causing a non-optimal computing elements load problem, which leads to incorrect assessment of the entire system performance.

In this paper, a new more accurate benchmark test of heterogeneous computing clusters with hybrid CPU/GPU architecture is presented.

III. Algorithm Description

First of all, a mathematical problem to be solved during benchmarking shall be selected. As an example, the problem

978-1-5090-4865-6/17/$31.00 ©2017 IEEE

of double-precision floating-point real numbers square matrix exponentiation was selected. The problem was selected based on its high degree of parallelism. To calculate the problem, we used its implementations in Intel MKL (for CPU) and cuBLAS (for the GPU) libraries. MKL library is used to solve computationally complex mathematical linear algebra problems and fast transformations; it also contains random-number generators for Intel processors and co-processors [13]. cuBLAS Library is used to solve computationally complex mathematical linear algebra problems and fast transformations for CUDA-compatible GPUs [14]. OpenMP library is used for parallel computing [15].

To make the problem potentially redistributable between computing nodes, the total number of matrix exponentiations shall be determined. The total number of matrix exponentiation repetitions is calculated as the most common multiple of the amounts of all available for calculation threads per each cluster computing node multiplied by a whole-number value set by the user when running the test. Besides, not only GPU threads, but also CPU threads not supporting GPU are considered as active threads.

Upon initialization, the file square matrix of a given dimension is read. To perform the operation, the same matrix allowing maintaining the accuracy of the calculated time parameters is used.

Later, the first operation distribution per cluster nodes is carried out. Due to the fact that the number of matrix exponentiation repetitions is a multiple of the number of active cluster threads, the operation can be evenly distributed per all cluster nodes. A calculation is performed using respective functions of Intel MKL (in the case of computing on CPU) or cuBLAS (in the case of computing on GPU) libraries. Task parallelization per computing elements within the node is performed using OpenMP. The matrix power is calculated by the formula:

$$N = 2 * pow + 1 \quad (1)$$

Here *pow* is the value entered by the user when running the test, *N* is the total number of matrix exponentiation repetitions per one iteration.

As a result of the calculation, the time needed per each node to perform its part of operation is measured.

The time parameters obtained are sent to the control node using the MPI standard inter-processor communication functions [16]. Upon that, the performed operation time parameters are analyzed, as well as the optimal problem-performance time, which corresponds to the case, where all nodes are loaded in such a way as to complete the calculation process at the same time, thus saving the cluster from the downtime and increasing its performance metrics, is calculated. The optimal operation execution time shall be determined by the formula:

$$T = \frac{C*I}{\sum_N \sum_{In} \frac{C_i}{T_i}} \quad (2)$$

Where *C* is the total number of matrix exponentiations per cluster, *I* is the total number of computing devices per cluster (both CPU and GPU), *N* is the number of nodes per cluster, I_n is the number of devices per cluster node, C_i is the number of matrix exponentiations per *i*-th device of the *n*-th node, T_i is the time spent by the computing device.

According to the optimal operation execution time value found, the matrix exponentiation repetitions shall be redistributed between cluster nodes, depending on the their performance in the first computation cycle. This is done by sending a new number of matrix exponentiation repetitions from the master to the child node.

Finally, the second computation cycle is carried out, the results of which determine the total computing cluster performance. As a result, the total computing cluster performance is assessed, including hardware features of its nodes, which, unlike the standard HPL benchmark test, should produce more accurate results.

Fig. 1 shows the overall designed software test execution structure.

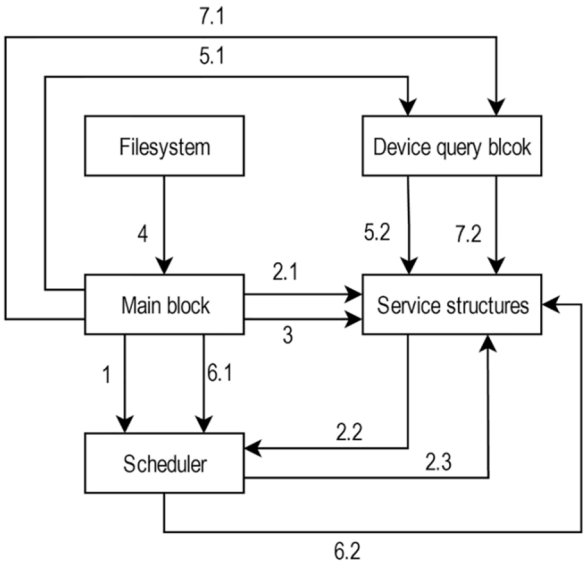

Fig. 1. Software Test Structure

Designations:

1) Scheduler initialization.

2) Cluster structures initialization;

2.1) Cluster devices initialization, performance data collection;

2.2) Task scope determination request;

2.3) Task scope transfer;

3) Post-initialization;

4) File system matrix reading;

5) First computation cycle;

5.1) Matrix exponentiation per device

5.2) Time parameters saving

6) Scope of problem redistribution per cluster device;

7) Second computation cycle;

7.1) Matrix exponentiation per device

7.2) Time parameters saving

IV. Defining Computing Capabilities

Before testing, we need to define which computing devices of nodes shall be involved in the calculations, as well as their parameters.

A. Defining CPU involvement in the calculations

To find the exact value of active threads per device, the number of CPU threads involved in the calculations shall be determined. As is known, in the case of GPU at the computing node, at least one CPU thread shall be allocated to control the computing on GPU. Let's define possible configurations of nodes:

1. The node without GPU;
2. The node with GPU and only one thread to perform computing on CPU;
3. The node with GPU and more than one thread to perform computing on CPU.

In the first case, all CPU nodes can be used for computing. In other cases, the node has GPU and thus part of CPU threads shall be allocated to control computing on GPU. In the second case, only one GPU thread is allocated to control CPU and, therefore, CPU of that node shall not be involved in computing. If the node has CPU with more than one thread available, then one of the available threads shall be allocated to control GPU, and the rest shall be involved in computing.

B. Defining GPU configuration

Within the framework of defining the optimal GPU configuration, the configuration of GPU blocks best meeting the task shall be defined. To do this, the configuration of blocks corresponding to the maximum GPU multiprocessor theoretical load shall be defined.

The maximum GPU multiprocessor theoretical load is considered as the product of the number of active blocks by the number of threads per active block. Therefore, to find the most efficient configuration, the value of the number of threads per block shall be consistently increased, the number of blocks per multiprocessor corresponding to a given number of threads shall be calculated and the best possible value shall be defined. Yet the hardware restrictions of the graphics card shall be considered (e.g., such restrictions as limiting the maximum number of threads per block or limiting the maximum number of blocks depending on the graphics card architecture). For example, the maximum possible number of active blocks per multiprocessor for Kepler architecture is 16 [17].

The resulting GPU configuration is characterized by the available number of threads possible to be involved in computing.

C. Finding unused devices during problem redistribution.

Upon redistribution of the scope of problem on a computing cluster, it may be that the relative performance of some CPU devices is too low and allocating any problems to solve for the latter is inappropriate. This case is calculated in the course of defining the number of matrix exponentiations on cluster computing devices. The value is calculated by the following formula:

$$Cnew_i = \frac{Iter * T * \frac{C_i}{T_i}}{Th_i} \quad (3)$$

Here $Cnew_i$ is the new value of the number of matrix exponentiations per i device, $Iter$ is the integral-valued multiplier set by the user at the program start, linearly increasing the number of iterations and used to reduce the error by increasing the complexity of the problem, T is the optimal test execution time calculated by the formula (2). C_i is the number of matrix exponentiations in the first computation cycle, T_i is the time spent by the computing i-th device in the first computation cycle. Th_i is the number of device threads.

Upon computing, $Cnew_i$ shall be rounded downward. If its value equals to 0, therefore, the test execution using this device shall be inefficient and the latter shall remain unused.

V. Experimental Results

To perform a comparative assessment of the designed computing cluster software test and the popular HPC Linpack benchmark test, a series of test runs on a computing cluster with the different number of active nodes was carried out. The testing was performed on a heterogeneous computing cluster with hybrid architecture and two computing node configuration options:

- CPU: Intel i7-2600 DDR3 4Gb, GPU: NVIDIA GTX550Ti;
- CPU: Intel i7-2600 DDR3 4Gb, GPU: NVIDIA GTX550Ti + NVIDIA GTS450.

The testing involved 7 nodes of the first configuration option and 2 nodes of the second configuration option.

The test based on the generated 512x512 double-precision floating-point real numbers square matrix, the number 150 was used as an integral-valued multiplier for the number of matrix exponentiation repetitions, the calculated matrix power equaled to 3. The table below shows the comparative computing cluster performance results depending on the number of active nodes of the computing cluster tested.

TABLE COMPUTING CLUSTER PERFORMANCE RESULTS

Number of nodes	HPC Linpack, GFLOPS	HPC Dozen, GFLOPS
1	101	53
2	187	110
3	223	166
4	276	227
5	310	283
6	368	343
9	485	650

According to the data received, the graph of performance rating against the number of active nodes was plotted. The graph is presented in Fig. 2.

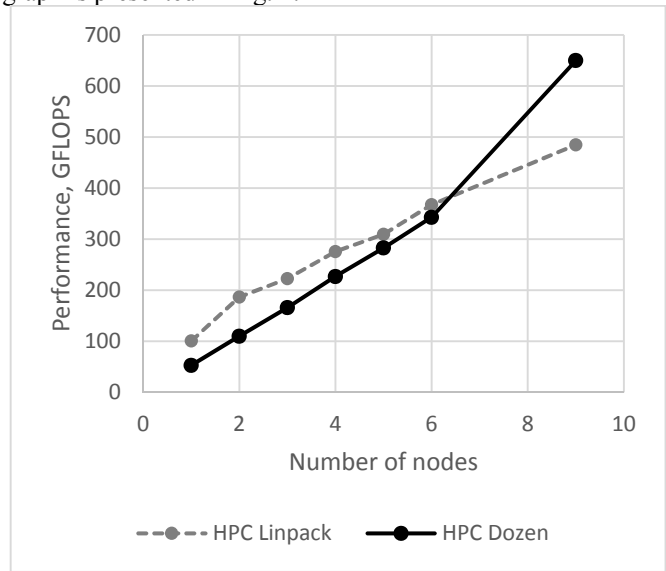

Fig. 2. Cluster performance rating depending on the number of nodes

Based on the results, it can be concluded that the test designed, compared to HPC Linpack benchmark test, is stronger at loading the computing node devices of the cluster consisting of a large number of nodes. Given a small number of nodes, the designed test performance is less than HPC Linpack test performance, which, just like a designed test, was launched on the same number of nodes. This is due to the fact that HPC Linpack test uses computing booster libraries during benchmarking. In the future, the performance obtained using a designed software test shall become greater than the performance obtained using HPC Linpack test.

It can be concluded that the test designed demonstrates a linear relationship between the performance and the number of nodes, which is theoretically true for clusters with a small number of nodes.

The performance value obtained using a designed test lies between a peak cluster performance and the performance value obtained using HPC Linpack test.

Based on the resulting data, it can be concluded that the test designed provides a more accurate cluster performance assessment.

VI. CONCLUSION

Based on the results, the computing cluster software benchmark test focused on defining the performance of heterogeneous computing clusters with hybrid CPU/GPU architecture was designed and implemented. The peculiarity of the test designed lies in its ability to take into account the performance characteristics of the cluster computing devices, and, therefore, distribute the computation load more efficiently, thus leading to more accurate results at defining the computing cluster performance.

REFERENCES

[1] Boreskov A.V., Kharlamov A.A. *Parallel computing on GPU. Architecture and CUDA programming model* (in Russian), pp 17-20, 2012.

[2] S. Gupta, M. R. Babu, "Performance analysis of GPU compared to single-core and multi-core CPU for natural language applications", *International Journal of Advanced Computer Science and Applications*, vol. 2, no. 5, pp. 50–53, 2011.

[3] J. Hestness, S. W. Keckler, and D. A. Wood, "A comparative analysis of microarchitecture effects on CPU and GPU memory system behaviour", *IISWC 2014 - IEEE International Symposium on Workload Characterisation*, pp. 150–160, 2014.

[4] I. V. Chugunkov et al., "Three-dimensional data stochastic transformation algorithms for hybrid supercomputer implementation," MELECON 2014 - 2014 17th IEEE Mediterranean Electrotechnical Conference, Beirut, 2014, pp. 451-457

[5] CUDA Toolkit Documentation [Online]. Available: http://docs.nvidia.com/cuda

[6] E. Bennett, B. Lucini, L. D. Debbio, K. Jordan, A. Patella, C. Pica, A. Rago, "BSMBench: a flexible and scalable supercomputer benchmark from computational particle physics", *2016 International Conference on High Performance Computing & Simulation (HPCS)*, pp. 834-839, 2016

[7] J. Dongarra, P. Luszczek, and A. Petitet, "The LINPACK Benchmark: past, present and future", Concurrency and Computation: Practice and Experience, vol. 15, no. 9, pp. 803–820, 2003

[8] HPL - A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers [Online]: Available: http://www.netlib.org/benchmark/hpl

[9] E. Strohmaier, H. W. Meuer, J. Dongarra, H. D. Simon, "The TOP500 List and Progress in High-Performance Computing", *IEEE Computer,* vol. 48, no. 11, pp. 42–49, 2015.

[10] F. Xing, H. You, C. Lu, "HPC benchmark assessment with statistical analysis", *Procedia Computer. Science*, vol. 29, pp. 210–219, 2014.

[11] E. Phillips and M. Fatica, "Performance analysis of the high-performance conjugate gradient benchmark on GPUs", *International Journal of. High Performance Computing Applications*, vol. 30, no. 1, pp. 28-38, 2015

[12] A. Rajan, B. K. Joshi, A. Rawat, R. Jha, K. Bhachavat. "Analysis of process distribution in HPC cluster using HPL", *2012 2nd IEEE International Conference on Parallel, Distributed and Grid Computing*, 2012, pp. 85-88

[13] Intel Math Kernel Library [Online]. Available: https://software.intel.com/intel-mkl

[14] cuBLAS - mplementation of BLAS (Basic Linear Algebra Subprograms) for CUDA [Online]. Available: http://docs.nvidia.com/cuda/cublas

[15] OpenMP [Online]. Available: http://openmp.org

[16] MPI Standard [Online]. Available: http://mpi-forum.org/docs

[17] Tuning CUDA Applications for Kepler [Online]: Available: http://docs.nvidia.com/cuda/kepler-tuning-guide