Running High Performance Linpack on CPUGPU clusters

Draško Tomić*, Dario Ogrizović**
*Hewlett-Packard Croatia, Zagreb, Croatia
** Center for advanced computing and modeling / Faculty of Maritime Studies, Rijeka, Croatia

Abstract – A trend is developing in High-Performance Computing with cluster nodes built of general purpose CPUs and GPU accelerators. The common name of these systems is CPUGPU clusters. High Performance Linpack (HPL) benchmarking of High Performance Clusters consisting of nodes with both CPUs and GPUs is still a challenging task and deserves a high attention. In order to make HPL on such clusters more efficient, a multi-layered programming model consisting of at least Message Passing Interface (MPI), Multiprocessing (MP) and Streams Programming (Streams) needs to be utilized. Besides multi-layered programming model, it is crucial to deploy a right load-balancing scheme if someone wants to run HPL efficiently on CPUGPU systems. That means, besides the highest possible utilization rate, both fast and slow processors needs to receive appropriate portion of load, in order to avoid faster resources waiting on slower to finish their jobs. Moreover, in HPC clusters on Cloud, one has to take into account not only computing nodes of different processing power, but also a communication links of different speed between nodes as well. For this reasons we propose a load balancing method based on a semidefinite optimization. We hope that this method, coupled with a multi-layered programming, can perform a HPL benchmark on CPUGPU clusters and HPC Cloud systems more efficiently than methods used today.

I. INTRODUCTION

With the advent of GPU cards, HPC landscape has changed dramatically. More FLOPS/watt are available, less space for HPC equipment is needed, and supercomputing on every workstation becomes possible. In addition, GPU concept brought better performance/cost ratio, faster communication within CPUGPU nodes, and smaller footprint, i.e. dense computing. Besides pros, there are several cons when using GPU accelerators on HPC clusters. Some of them are non-uniform programming models, a need for deployment of advanced cooling systems, e.g. water-cooling, and a fact that many HPC applications are still not supporting GPU accelerators. At least but not the last, HPL benchmark has proven not to be so efficient on heterogeneous clusters like on CPU only clusters. As of today, HPL is a standard benchmark for Supercomputing, and a performance measure for ranking supercomputers in the TOP500 list of the world's fastest computers [1].

The package tries to find a solution of the system of ***N*** linear equations with *N* unknowns. In fact, this is a common engineering task, in matrix notation expressed as:

$$Ax = b \tag{1}$$

Where *x i*s a vector of *N* unknowns, *A* is a matrix of *N x N* cofactors in the system, and *b* is a vector of *N* values that upper equation needs to satisfy.
HPL uses Gaussian elimination with partial pivoting as a method of finding a solution for (1). The process of Gaussian elimination has two parts. The first part (Forward Elimination) reduces a given system to triangular form. The second step uses back substitution to find the solution of the system above. In the case of Gaussian elimination with partial pivoting, the algorithm requires pivot element not be zero. Interchanging rows or columns in the case of a zero pivot element is necessary. Choosing pivot element with large absolute value improves the numerical stability. The number of millions of floating point operations per second (MFLOPS) performed with this method is:

$$MFLOPS = 2/3 \cdot N^3 + 2 \cdot N^2 + O(N) \tag{2}$$

HPL runs for different matrix sizes *N* searching for the size N_{max} that delivers the maximal performance R_{max}. The benchmark also reports the problem size $N_{1/2}$ where the system reaches $R_{max}/2$, i.e. the half of its performance. An excellent paper is [2], covering all the details and variants of overview of HPL benchmark .

II. PERFORMANCE MODELLING OF HPL ON CPUGPU CLUSTERS

An Effectiveness Factor (EF) of a certain computer system is a ratio between maximal computation power achieved and computation power installed. A well-designed computer system will have EF near 1, while low EF values are signs for problems coming from architectural, application or both domains. Observing in this way, HPL is the application for measuring EF.

In fact, one can sum installed MFLOPS computational power in a certain HPC system, and then run HPL to see how much MFLOPS this system can actually deliver. Speaking in a top500 jargon, *Rpeak* is computation power high-performance cluster can theoretically deliver, while *Rmax* is a maximum number of MFLOPS achieved during a HPL benchmarking run. A quick comparison of clusters on top500 list without GPU accelerators with those having GPU accelerators shows that EF is remarkably lower on CPUGPU clusters, and that EF does not scale well with overall number of nodes in cluster. Therefore, when designing CPUGPU clusters, it is of the outmost importance to predict their HPL performance with a high level of accuracy. Thus, we need some benchmarking results from smaller clusters, and then a right approach to scale HPL on much larger clusters, even those that will take place on top500 list in forthcoming years. For example, benchmarking results of HPL done on 16- node cluster [3] can serve as a starting point for the performance prediction of larger cluster, if right extrapolation method is applied. In this benchmarking case, each computing node was HP Proliant DL360 G7 with two 5670 Intel 2.67 GHz six-core processors and 48GB of RAM, with RHEL as operating system, and OpenMPI, CUDA and Intel compiler running on each node. From the manufacturer data sheets Tesla NVIDIA 2050 can deliver at its peak 1030 Gflops of floating point arithmetic with double precision, while two Intel 5670 six-core processors on HP Proliant DL360 motherboard are theoretically able to deliver 140 Gflops of the same type of arithmetic. Therefore, a sum of these two values provides Rpeak of 1170 GFLOPS for a single computing node. HPL runs on a various cluster sizes ranging from 1 to 16 delivered following Rmax and EF values:

Nr. nodes	Rmax	Rpeak	EF
1	640	1170	0.55
2	1319	2340	0.56
4	2551	4680	0.55
6	3814	7020	0.54
8	5092	9360	0.54
10	6295	11700	0.54
12	7498	14040	0.54
14	8771	16380	0.53
16	10050	18720	0.53

Table 1: HPL benchmarking results on uniform CPUGPU cluster

Results show us that scaling of HPL in this cluster is good with very few percents of EF decrease starting from a single node to the maximum cluster size. However, EF values in overall are disappointing, although cluster nodes are equal.

III. HOW TO PERFORM EFFICIENT HPL ON HETEROGENEOUS CPUGPU CLUSTERS

In case cluster nodes are not of equal processing power, we are speaking about heterogeneous CPUGPU clusters. There are very few published results about HPL benchmarks in heterogeneous CPUGPU clusters, not only because such clusters are still rare [4]. The main problem found in heterogeneous clusters is still present today. The shining example of this problem is Map Reduce algorithm and its implementations, for example Hadoop. The problem is if one node is slower than the rest of cluster nodes, it can slow down the entire cluster. However, a team of researchers working on Tsubame 1.0 [5] delivered a good work in this area. They were able to report a HPL result with 53% efficiency on the overall Tsubame 1.0 cluster. Load balancing strategy chosen for their HPL implementation on Tsubame 1.0 was fair scheduling. That is, allocation of each parallel task to CPUGPU cluster node was according to its weight. That means, the more instructions a certain process has to execute, a faster resource will handle it. Taking into the consideration that this cluster was a mix of AMD, Intel and various GPU processors, and comparing it with HPL efficiency of the 16-node uniform cluster already discussed before, not so bad result at all.

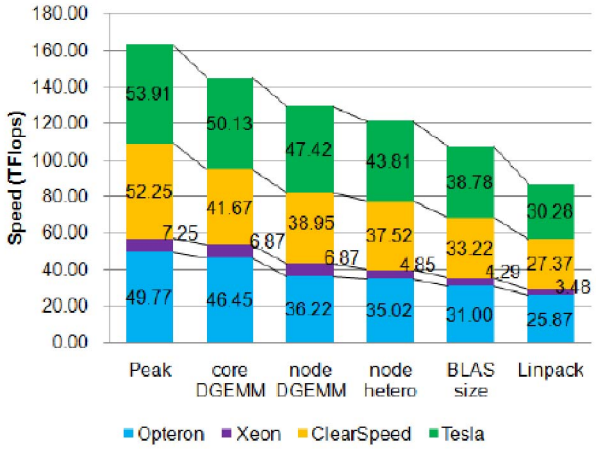

Figure 1: HPL performance on Tsubame 1.0 with 10480 Opteron cores, 640 Xeon cores, 624 NVIDIA and 648 Clearspeed GPUs

The main reason for a relative low EF in both clusters considered before is the core design of HPL, designated for homogeneous clusters. HPL is two-step process. First step is LU factorization of the dense matrix *A* with partial pivoting. Next step is the solution of the resulting triangular system of equations. Let us restate that overall workload of HPL benchmark is $(2/3)N^3 + 2N^2 + O(N)$. LU factorization takes almost all the computation time of the benchmark. Matrix update and the upper (U) matrix factorization dominate the computation time of the LU factorization. Matrix update is a DGEMM (Double Precision General Matrix Multiply) form of the matrix-matrix multiply, and it takes $O(N^3)$ operations.

On the other side, solver kernel takes $O(N^2)$ operations, because of *right*-hand-sides (DTRSM) kernel. Therefore, if one wants to further accelerate HPL on heterogeneous CPUGPU clusters, both load balancing of HPL processes tasks between computing nodes and within computing nodes has to be done. For uniform CPUGPU clusters, it is easy to balance HPL processes between nodes: each node should receive the same portion of load via MPI. Load balancing within nodes requires more attention, because CPU and GPU resources within nodes are not of the same processing power. A good strategy is to put more load on GPU resources, and less load on CPU resources, in a way that both resources finish their jobs at the same time. OpenMP, the Open Source implementation of MP, is a good choice to accomplish this. At last, one needs to program portions of HPL routines within CPUs and GPUs, normally via compilers (CPU) and Streams (GPU). It is clear from the before considerations that effective HPL on CPUGPU clusters requires at least two things: Adequate load balancing and multi-layered programming, i.e. MPI, MP and Streams programming. A team running HPL benchmarking of TianHe-1 [6] has performed a good work involving adequate load balancing of processes and multi-layered programming.

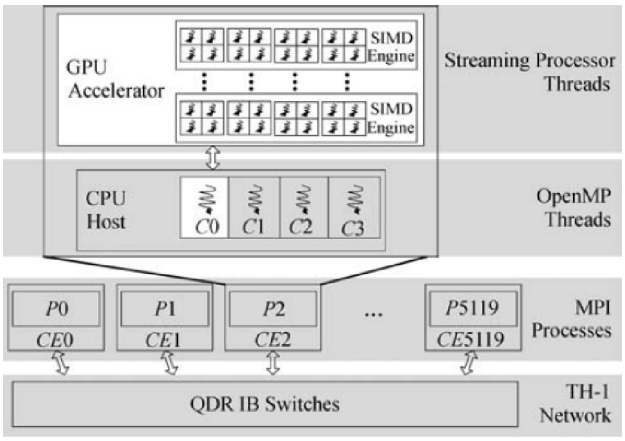

Figure 2: Multi-layered programming model of HPL implementation for the TianHe-1.

The final Linpack result was Rpeak of 563 TFLOPS, which compared to Rmax of TianHe-1 delivers EF of 0.701, considerably more than Tsubame 1.0 and 16-node Proliant cluster.

IV. CLOUD IS CHALLENGING US

While the phenomena of the relatively slow node (compared to the rest of cluster members) able to freeze the whole cluster can be solved in traditional HPC clusters by fair load balancing (each node receives a load proportional to its performance), this is not a right approach when doing HPC on Cloud.

CPUGPU computing nodes in Cloud may have not only various performances, but also communication links between them can have various bandwidth. Only one slow link is able to slow down the entire cluster. Therefore, a challenge of running HPL efficiently is much bigger than on heterogeneous CPUGPU clusters. The main question here is how to distribute load on computational nodes in a balanced way (nodes receive a load with respect to their processing power and utilization) while at the same time minimizing communication times between nodes. Following an idea first described in [7], and with the advent of the algorithm described in [8], a load-balancing algorithm relying of the semidefinite optimization of the data-flow matrix with respect to the second largest eigenvalue in magnitude (SLEM) was developed [9]. This approach is not limited to HPC clusters only, as for example shown in [10]. However, there is no currently known semidefinite optimization with respect to SLEM, which works on general and non-symmetric matrices. Therefore, if one wants to deploy a semidefinite optimization of the certain matrix with respect to the SLEM, he needs to transform it to its doubly stochastic and symmetric form. Because this transformation takes a considerable amount of time, a method is still not appropriate for general cloud applications. In contrary, many HPC workloads, like HPL, tend to execute in a huge number of equal iterations, and it is feasible to consider semidefinite optimization as a first choice in such a cases. We can rewrite this load-balancing problem as:

$$Minimize\ \ \mu(P) \tag{3}$$

With respect to:

$$P \geq 0, P1 = 1,\ P = P^T \tag{4}$$

Where P is doubly stochastic and symmetric matrix, its transpose is P^T, $\mu(P)$ is SLEM, and 1 is unary vector.
In a heterogeneous CPUGPU and Cloud based HPC clusters consisting of many computing nodes with different processing power, each node can have many multi-core processors, cores can have different working frequencies and utilizations, and computing nodes can communicate with other nodes via interconnection communication links of various bandwidths. Let us denote the total number of processing nodes with *n* and a total number of cores within a certain node with m. Additionally, let us denote a frequency and utilization of the core *j* within node *i* with ps_{ij} and u_{ij} respectively. Thus, we can express the effective processing speed of node *i* as:

$$eps_i = \sum_{j=1}^{m} \frac{ps_{ij}}{u_{ij}} \tag{6}$$

Additionally, we can denote with bw_{ij} a bandwidth of the communication link between processing nodes *i* and *j*. Then the following matrix can describe effective processing speed of computing nodes and bandwidth of communication links between them:

$$Y = \begin{bmatrix} eps_1 & bw_{12} & . & bw_{1n} \\ bw_{21} & eps_2 & . & bw_{2n} \\ . & , & . & . \\ bw_{n1} & . & bw_{n,n-1} & eps_n \end{bmatrix} \quad (7)$$

Because optimization algorithm is limited to symmetric matrices only, one needs to transform matrix *Y* to its symmetric form. In general, communication links are bidirectional and not of equal bandwidths in both directions. Therefore, we can write symmetric form of matrix *Y* as:

$$Y_s = \begin{bmatrix} eps_1 & \min(bw_{12}, bw_{21}) & . & \min(bw_{1n}, bw_{n1}) \\ \min(bw_{21}, bw_{12}) & eps_2 & . & \min(bw_{2n}, bw_{n2}) \\ . & , & . & . \\ \min(bw_{n1}, bw_{1n}) & . & \min(bw_{n,n-1}, bw_{n-1,n}) & eps_n \end{bmatrix} \quad (8)$$

If one wants to describe the data-flow within a certain HPC cluster, then besides Y_s, he needs to describe algorithm running within this cluster as well. One of way to do this is to construct weighted and vertex labeled directed graph, with edge weights representing number of messages computing nodes have to exchange with each other, and vertex labels representing number of operations computing nodes have to perform. From this graph, one can construct accompanying matrix *A*, with main diagonal elements representing number of operations computing nodes have to perform, and of-the-main-diagonal elements representing number of messages computing nodes have to exchange with each other.

$$A = \begin{bmatrix} a_{11} & a_{12} & . & a_{1n} \\ a_{21} & a_{22} & . & a_{2n} \\ . & . & . & . \\ a_{n1} & a_{n2} & . & a_{nn} \end{bmatrix} \quad (9)$$

Usually there are many algorithms executing on HPC cluster. Then, for each algorithm matrix *A* needs to be constructed and sum of these matrices will represent all algorithms. The next step is to express a data-flow within HPC system. One of way to do this is to perform Hadamard multiplication between matrices Y_s and A:

$$P_n = \begin{bmatrix} eps_1 \cdot a_{11} & \min(bw_{12}, bw_{21}) \cdot a_{12} & . & \min(bw_{1n}, bw_{n1}) . a_{1n} \\ \min(bw_{21}, bw_{12}) \cdot a_{21} & eps_2 \cdot a_{22} & . & \min(bw_{2n}, bw_{n2}) \cdot a_{2n} \\ . & . & . & . \\ \min(bw_{n1}, bw_{1n}) \cdot a_{n1} & \min(bw_{n2}, bw_{2n}) \cdot a_{n2} & . & eps_n \cdot a_{nn} \end{bmatrix} \quad (10)$$

Matrix P_n is symmetric; however, it is not doubly stochastic. In order to perform semidefinite optimization of P_n with respect to SLEM, one needs to convert it to its doubly stochastic form. A following recursive equation (11) developed in [9] transforms P_n to its doubly stochastic form. Experimental results in [9] show that by doing the following recursion in not more than 100 iterations, an absolute error is less than 0.1%.

$$P_{n+1} = (D_n)^{-1/2} P_n (D_n)^{-1/2} \quad ; n = 1,...,\infty \quad (11)$$

Where:

$$D_n = \begin{bmatrix} d_1 & 0 & 0 & 0 \\ 0 & d_2 & 0 & 0 \\ 0 & 0 & . & 0 \\ 0 & 0 & 0 & d_n \end{bmatrix} \quad (12)$$

Where each element on the main diagonal D_n is a sum of the corresponding row of the P_n:

$$d_i = \sum_{j=1}^{n} P_{ij} \quad ; \; i = 1,...,n \quad (13)$$

After optimization, one needs to obtain the deterministic form of matrix P_n. Let us denote with Q_n optimized version of the matrix P_n. It can be shown that the following transformation can perform this:

$$Q_{n-1} = (D_n)^{1/2} Q_n (D_n)^{1/2} \quad ; n = \infty,...,1 \quad (14)$$

Again, the number of iterations needs to be limited. Experimental results in [9] show that within 100 iterations error drops below 0.1%. At the end of the recursive process in (14), Q matrix will contain an optimal data-flow for a given cloud. In order to obtain an optimal workload distribution on the processing nodes, one has to convert an optimal data-flow in the number of operations computing nodes have to perform. One way to do this is to use trace of the matrix, which gives the sum of the matrix main-diagonal elements. The optimal workload distribution A_o will be then:

$$A_0 = \frac{trace(A)}{trace(Q)} Q \quad (1) \quad (15)$$

This method has been experimentally [9] proven to be effective in a HPC clusters consisting of several dozens of heterogeneous computing nodes.

We hope to continue with experiments when HPC Cloud systems with CPUGPU nodes will be more available to academic community: at this time, we are not aware of any HPC Cloud system with CPUGPU nodes of various performances.

V. CONCLUSION

HPL benchmark is a standard tool for the performance measurement of HPC systems. Applied to homogeneous and heterogeneous CPUGPU clusters, it still shows significantly smaller EF values then on CPU only clusters. There are several reasons for it. The most important is that initial design of HPL was for large SMP and CPU only clusters. Not of the less importance, HPL was born in times when the cloud was meteorological object only, not IT term. Therefore, it is also not suitable for the performance measurement of HPC Cloud systems. We are now at the crossroad and have to choose a right direction: One is leading towards the significant modification of HPL, in order to give it new functionalities needed for the better performance measurement of the heterogeneous and Cloud HPC systems. Another is to rethink the existence of HPL and start searching for new, more appropriate HPC benchmarks. On both roads, there is clear evidence for better, i.e. multi-layered programming habits, and for more effective and wiser load balancing strategies.

REFERENCES

[1] http://www.top500.org

[2] J. Dongarra, P. Luszczek, A. Petitet, "The linpack benchmark: Past, present and future." Concurrency and Computation: Practice and Experience, 2003, 15(9): 803-820.

[3] M. Fatica, "Accelerating Linpack with CUDA on heterogeneous clusters", In Proc. 2nd Workshop on General Purpouse Processing on Graphics Processing Units (GPUGPU-2), Washington DC, USA, 2009, pp. 46-51.

[4] M. Kistler, J. Gunnels, D. Brokenshire, B. Benton, "Petascale computing with acceleratots", in Proceedings of ACM Symposium on Principles and Practice of Parallel Computing, 2009, pp. 241-250.

[5] T. Endo, A. Nukada, S. Matsuoka, N. Maruyama, "Linpack evaluation on a supercomputer with heterogeneous accelerators", 2010 IEEE International Symposium on Parallel & Distributed Processing, 19-23 April 2010., Atlanta GA, pp. 1-8.

[6] F. Wang, C.Q. Yang, Y.F. Du, Y. Chen, H.Z. Yi, W.X. Xu, "Optimizing Linpack Benchmark on GPU-Accelerated Petascale Supercomputer", Journal of Computer Science and Technology, Vol. 26., No. 5. (1 September 2011), pp. 854-865.

[7] D. Tomic, "Spectral performance evaluation of parallel processing systems", J. Chaos, Solitons and Fractals, 13 (1) (2002) 25-38.

[8] S. Boyd, P. Diaconis, L. Xiao, "Fastest Mixing Markov Chains", SIAM Review, 46(4), pp. 667 – 689, December 2004.

[9] D. Tomic, O. Muftic, "Distributing workload in parallel computing environment with SLEM", internal communication, 2006-2010.

[10] D. Tomic, O. Muftic, B. Biocic, "A novel scheduling approach of e-learning content on cloud computing infrastructure", 2011. Proceedings of the 34th International Convention MIPRO, Opatija, pp. 1443 – 1446.