Linpack Evaluation on a Supercomputer with Heterogeneous Accelerators

Toshio Endo
Graduate School of Information Science and Engineering
Tokyo Institute of Technology
Tokyo, Japan
endo@is.titech.ac.jp

Akira Nukada
Global Scientific Information and Computing Center
Tokyo Institute of Technology
Tokyo, Japan
nukada@matsulab.is.titech.ac.jp

Satoshi Matsuoka
Global Scientific Information and Computing Center
Tokyo Institute of Technology/National Institute of Informatics
Tokyo, Japan
matsu@is.titech.ac.jp

Naoya Maruyama
Global Scientific Information and Computing Center
Tokyo Institute of Technology
Tokyo, Japan
naoya@matsulab.is.titech.ac.jp

***Abstract*—We report Linpack benchmark results on the TSUBAME supercomputer, a large scale heterogeneous system equipped with NVIDIA Tesla GPUs and ClearSpeed SIMD accelerators. With all of 10,480 Opteron cores, 640 Xeon cores, 648 ClearSpeed accelerators and 624 NVIDIA Tesla GPUs, we have achieved 87.01TFlops, which is the third record as a heterogeneous system in the world. This paper describes careful tuning and load balancing method required to achieve this performance. On the other hand, since the peak speed is 163 TFlops, the efficiency is 53%, which is lower than other systems. This paper also analyses this gap from the aspect of system architecture.**

I. Introduction

Accelerators attract much attention in the high performance computing community [1], [2], [3], [4], [5], for their excellent performance with limited power consumption and spaces. They include graphics processors (GPU) from NVIDIA or AMD, IBM/Sony/Toshiba Cell processors and ClearSpeed SIMD accelerators. In order to accommodate large scale applications, parallel computation on heterogeneous clusters equipped with accelerators have been investigated by several groups [6], [7], [8]. Currently, the largest heterogeneous system is the LANL Roadrunner [9], which is ranked as No.2 in the Top500 supercomputer ranking [1] , and equipped with 12,960 PowerXCell 8i processors as accelerators. Kistler et al. have implemented a Linpack implementation designed for the Roadrunner architecture, which has achieved more than 1PFlops[10]. Recently, a GPU accelerated supercomputer has been installed in China; the NUDT Tianhe-1, equipped with AMD Radeon HD 4870 X2 GPUs, has achieved 563TFlops in Linpack.

The third largest heterogeneous system is *the TSUBAME supercomputer* installed in Tokyo Institute of Technology. TSUBAME is a 655-node Opteron cluster, equipped with NVIDIA Tesla GPUs and ClearSpeed accelerators; unlike Roadrunner or other systems described above, it includes two types of accelerators. This is due to incremental upgrade of the system, which has been the case in commodity CPU clusters; they may have processors with different speeds as a result of incremental upgrade. In this paper, we present a Linpack implementation and evaluation results on TSUBAME with 10,480 Opteron cores, 624 Tesla GPUs and 648 ClearSpeed accelerators. In the evaluation, we also used a Xeon cluster to get higher performance. Our setting raises several challenging issues that do not appear on Roadrunner: (1) the system has heterogeneous accelerators, (2) nodes are heterogeneously accelerated; some nodes have GPUs while others do not, (3) memory sizes of accelerators are severely limited, (4) omitting CPUs from kernel computation is unreasonable.

In this paper, we discuss the difference in design strategy between Kistler's Linpack implementation on RoadRunner and ours, which is based on one presented in our previous paper[11]. While we observe its basic strategy is still effective with heterogeneous accelerators, we further present several new optimization techniques that are required for improve scalability, such as overlap between communication and computation. As a result, we have achieved 87.01TFlops, which was ranked as No. 56 in the Top500 in November 09. As shown in Figure 1, we have achieved continuous performance improvements for six times, which shows the significance of incremental upgrade of accelerated systems.

We also focus on the gap between the Linpack performance (R_{peak}) and the theoretical peak performance (R_{max}). Since R_{max} is 163.2TFlops in our evaluation, the *efficiency* is $R_{peak}/R_{max} = 87.01/163.2 = 53.3\%$, which is significantly lower than 76% on Roadrunner. The reason of this is also discussed in detail.

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

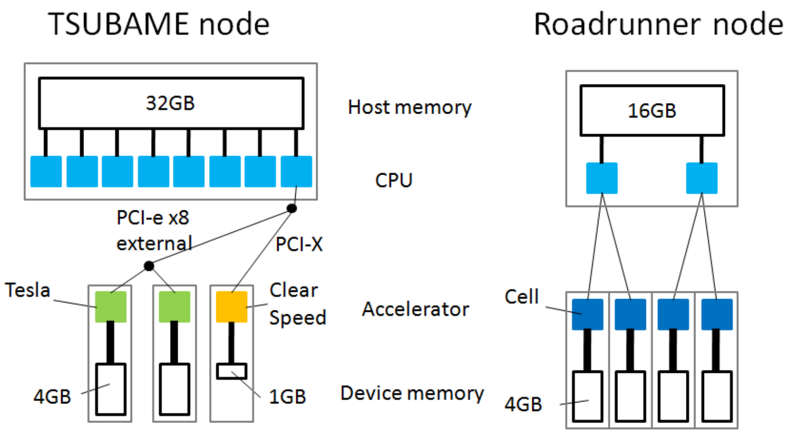
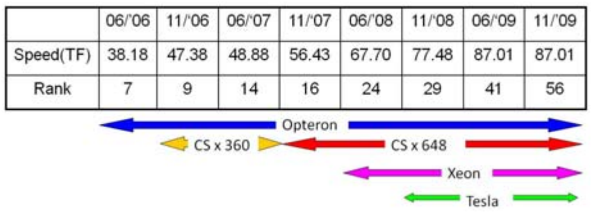


Figure 2. The overview of architecture of a TSUBAME node with Tesla and a Roadrunner node.



	06/'06	11/'06	06/'07	11/'07	06/'08	11/'08	06/'09	11/'09
Speed(TF)	38.18	47.38	48.88	56.43	67.70	77.48	87.01	87.01
Rank	7	9	14	16	24	29	41	56

Opteron
CS x 360
CS x 648
Xeon
Tesla

Figure 1. History of the TSUBAME system in the Top500 ranking. Arrows show the kinds of CPUs/accelerators used in the evaluations ("CS" indicates ClearSpeed accelerators).

II. The TSUBAME Heterogeneous Supercomputer

TSUBAME consists of 655 SunFire X4600 nodes and storage system of 1.6PBytes in raw capacity, and they are connected via InfiniBand. In the following, we describe parts of system architecture that strongly relate to this work.

- **TSUBAME nodes:** Each compute node has 16 2.4 GHz AMD Opteron 880 CPU cores, 32 GBytes of shared memory, and two 4x SDR InifiniBand host channel adapter (HCA). The node has PCI-Express 1.0 x8 slots and PCI-X slots; HCAs and Tesla GPUs (on about the half the nodes) are accessible via PCI-Express bus, while ClearSpeed accelerators are equipped on PCI-X slots as shown in Figure 2. The operating system is 64bit SuSE Linux Enterprise Server.
- **InfiniBand interconnect:** Each node is connected to one of edge switches, which are Voltaire ISR9288 288-port switches. The edge switches are connected to two core switches with 24 links of InfiniBand cables.
- **ClearSpeed accelerators:** Each node has a ClearSpeed X620 accelerator board [2] on its PCI-X slot. The accelerator has two CSX600 SIMD processors and 1GB DRAM, whose memory bandwidth is 6.4GBytes/s. Each CSX600 includes 96 processing entities of 420MFlops (double precision), thus the performance of accelerator board is 80.64GFlops [3]. Since the memory is separated from host memory, communication of input/output data via 1GBytes/s PCI-X bus is necessary. The power consumption of each accelerator is about 25W. The vendor provides linear algebra library named CSXL, which we use in our evaluation.
- **Tesla GPU accelerators:** NVIDIA Tesla S1070 [4] includes four accelerators (identical to graphic boards) in 1U form factor. In the TSUBAME system, 170 S1070 systems with 680 GPU accelerators have been installed in 2008. Among the nodes, 316 nodes are connected with Tesla via PCI-Express external cabling, and each node can access to two GPUs [5]. These two accelerators share a single PCI-Express x8 bus on a node.
  Each accelerator has a Tesla T10GPU, which has 30 SIMD streaming multiprocessors (SM). SMs share 4GB device memory whose memory bandwidth is 102GBytes/s. The peak performance of an accelerator is 86.4GFlops in double precision and 1.04 TFlops in single precision. The power consumption per S1070 is 700W, thus each accelerator consumes about 175W.
  A programming environment named CUDA, which extends C language, is provided. We have made a linear algebra kernel by using CUDA for our evaluation, instead of CUBLAS, the official library from the vendor.
- **Tsubasa Xeon cluster:** We have installed another cluster named "Tsubasa", which is connected to TSUBAME via 20 InfiniBand links (200Gbps in total). In

[2] `http://www.clearspeed.com`

[3] The clock speed is configurable up to 250MHz, but we set the clock as 210MHz for stability.

[4] `http://www.nvidia.com`

[5] Although a few nodes are connected to four accelerators, we use only two on those nodes in our evaluation.

our Linpack evaluation, we also use this cluster for higher performance. This cluster consists of 90 Sun Blade X6250 nodes, each of which has two Quad core Xeon E5440 (2.83GHz) and 8GB memory.

We use almost all of the above computing resources for parallel Linpack run. Here we faced with two types of heterogeneity, *intra-node heterogeneity* and *inter-node heterogeneity*. The former is that TSUBAME nodes have both general purpose CPUs and accelerators. The latter is that we have three types of nodes with different configuration as follows: (a) a TSUBAME node with Tesla has 16 Opteron cores, a ClearSpeed, two Tesla GPUs, (b) a TSUBAME node without Tesla has 16 Opteron cores and a ClearSpeed, and (c) a Tsubasa node has eight Xeon cores.

III. The Linpack Benchmark

Our Linpack implementation is based on High performance Linpack (HPL)[12], which is a well-known parallel implementation of Linpack. HPL is a MPI based parallel program that solves dense linear equations $Ax = b$ of order N. Participating MPI processes conceptually compose a process grid of size $P \times Q$, and the matrix A is distributed among processes according to two-dimensional block cyclic distribution. Hereafter, we let N the matrix size and B the block size.

The algorithm is based on blocked LU decomposition of the matrix A with partial pivoting. All matrix operations are done in double precision. A step of LU decomposition (kth step) proceeds as follows:

- **Panel factorization:** The kth block column is called panel column. LU decomposition is done on the panel column with partial pivoting.
- **Panel broadcast:** The computed panel is broadcast to other processes. This communication is "process row-wise".
- **Row exchange:** According to the result of partial pivoting, rows are exchanged by "process column-wise" communication.
- **Update:** By using computed panel and kth block row after row exchange, all the remaining part of the matrix A is updated by using matrix multiply operation.

The amount of computation of "panel factorization" is $O(N^2B)$ in total of all steps, the communication amount of "panel broadcast" and "row exchange" is $O(N^2P)$ and $O(N^2Q)$ respectively, and the computation amount of "update" is $(2/3)N^3$. We can see that the most time-consuming part is "update", and its dominance gets stronger as N gets larger.

Thus in the Linpack benchmark, generic strategies to obtain better Flops is as follows. First, it is better to configure N as large as possible, while the matrix size should fit the physical memory size of the system. Second, since the kernel computation in "update" is matrix multiply (DGEMM operation in BLAS), one of keys is to use high performance linear algebra library. Third, since the matrix is distributed among processes almost uniformly, thus computation performance of each process should be uniform, since we do not change the basic distribution method of HPL.

IV. Design and Implementation on Heterogeneous Systems

Previously we have presented a Linpack implementation designed for heterogeneous cases with CPUs and ClearSpeed[11]. We have confirmed its basic design is also effective with minor modifications for configuration with heterogeneous accelerators. In this section, we describe its overview and new optimization techniques.

A. Considering System Architecture

First, we present basic design strategies while comparing with Kistler's design for Roadrunner [10]. The Roadrunner is a heterogeneous system that consists of 3240 nodes, each of which has four Opteron cores and four PowerXCell 8i processors as accelerators. Although both TSUBAME and Roadrunner are heterogeneous systems, the Linpack designs are significantly different due to differences in system architecture.

- **Who computes the kernel:** In our design, all of general purposes CPUs, Tesla and ClearSpeed are used for kernel computation (DGEMM: matrix multiply) for "Update", unlike Kistler's that uses only Cell processors for DGEMM. The later is reasonable on Roadrunner, since 96% of the computation performance come from Cell processors. On TSUBAME, on the other hand, general purposes CPUs contribute 35%, Tesla GPUs do 33% and ClearSpeed accelerators do 32%; therefore omitting any types of processors from kernel computation incurs heavy performance degradation.
- **Where matrix data are placed:** In Linpack, the matrix data of size $N \times N$ are distributed among MPI processes. Here each process has two choices to place its own data: host memory and device memory. We have to discuss the physical memory size of system architecture. On a Roadrunner node (the right figure in Figure 2), the size of host memory and that of device memory is balanced; both are 16GB per node. Therefore they decided to place all the matrix data on device memory, which is reasonable as kernel computation is done by Cell processors. On TSUBAME, the host memory size is 32GB per node. Even on GPU-accelerated nodes, the device memory size is 4GB+4GB+1GB with two GPUs and a ClearSpeed, which is much smaller than host memory. Thus we have decided to place the matrix data on host memory basically [6]. In our design,

[6]Strictly speaking, we should compare the case where the matrix size N is configured so that the data fits on device memory, although we have not implemented it. We consider this approach will suffer from heavy performance degradation on TSUBAME since device memory size of ClearSpeed is too small.

PCI-express/PCI-X (hereafter PCI) communication is required when kernel computation is done by accelerators.

- **Who computes non-kernel parts:** Both on Roadrunner and TSUBAME, MPI communication has to be done by CPUs, since accelerators do not have direct interface to interconnect. We also decided the auxiliary computation such partial pivoting in panel functions to be done by CPUs. This is because the computation and communication is interleaved in a fine grained fashion.
- **Coping with inter-node heterogeneity:** On TSUBAME, we have to consider different node configuration described above, while keeping the performance of processors assigned to each MPI process uniform. On Roadrunner or homogeneously accelerated clusters[6], such a consideration is unnecessary.

Note that the above discussion heavily depends on the application characteristics, especially, the Flops/ Byte ratio of kernel computation. Since the kernel in Linpack is DGEMM in the Level 3 BLAS, it is relatively simple to keep the Flops/ Byte ratio higher as follows. In a typical kernel invocation in HPL, each process takes two matrices, whose sizes are $M \times B$ and $B \times M'$, respectively. Then their product matrices are subtracted from the local part of matrix A. Here M, M' denotes the size of local matrices to be updated. In this kernel invocation, the computation amount is $O(MM'B)$, while the PCI communication amount is $O(MM')$ (We assume that B is much smaller than M, M'). Thus when B is large enough, we can hide the effects of PCI communication by computation.

In other applications where improving Flops/ Byte ratio is very hard or impossible, such as stencil computation, placing all the data in host memory is unsuitable, since it causes a large amount of PCI communication. Designing such applications using heterogeneous processors is our future work.

B. Coping with Heterogeneity

As already described, we use all types of CPUs and accelerators for kernel computation, while other computation and communication are done by CPUs as in the original HPL. Thus we do not modify the structure of HPL code largely, except the kernel invocation. Here our goal is to coping with intra-node heterogeneity and inter-node heterogeneity while keeping the performance of MPI processes uniform.

For intra-node heterogeneity, we focus on mapping between MPI processes and physical processors. Since matrix multiply is a divisible workload, it is easy to distribute workload of a single DGEMM invocation to different types of processors according to their respective performance. In this aspect, the type of processors are not important us; we virtualize CPUs and accelerators as providers of computation power with different processing speeds. With this approach, it is straightforward to invoke multiple MPI processes on a single node. The approach taken by the Roadrunner Linpack, which maps an accelerator to a process, does not work well for us since we have accelerators with heterogeneous speeds. We take an approach that maps multiple processors to multiple processes as shown in Figure 3, where each color denotes one of four MPI processes. In (a) in the figure, the first process (colored red) throws DGEMM tasks to three Opteron cores, ClearSpeed and one of Tesla GPUs. On the other hand, each accelerator is shared between two processes. The ratio of task should be configured so that all processes have identical computation speed. Currently this tuning is done by hand.

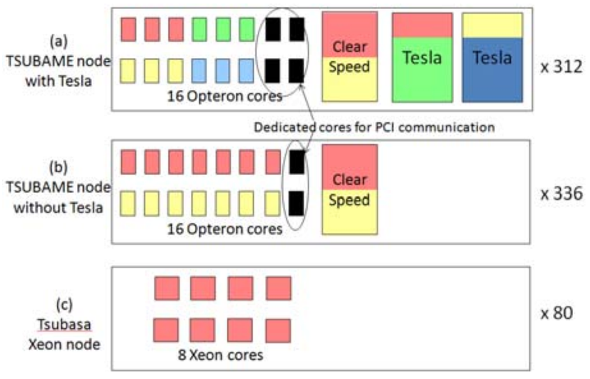


Figure 3. Our mapping between processes and processors on each node type. Each color corresponds to a MPI process.

This virtualization is also the key for inter-node heterogeneity; we have three types of nodes with different configurations. We again regard them as providers of DGEMM computation with different speeds. According to preliminary experiments, their speeds are 262.4GFlops, 121.2GFlops and 85.9GFlops, respectively. Since we like to keep the performance per process uniform, we control the number of processes for each node configuration; we let it four, two, one in our evaluation [7]. To avoid overhead caused by unexpected scheduling by operating system, we bind the process and CPU cores by using sched_setaffinity systemcall.

C. Implementation Techniques

We describe some implementation techniques, mainly due to avoid performance limiting factor we have found in HPL. Some of them have been presented in our previous paper [11].

- In our approach, a single accelerator is shared among multiple processes. However, naive implementation of this would be difficult, since current ClearSpeed accelerators do not support it. To solve this problem, we implement *BLAS servers*, which are daemon processes that have direct access to accelerators. They are invoked

[7]Finer mapping methods, such as 17:8:5, may work, but they invoke too many processes causes overhead.

per accelerator, and arbitrate DGEMM requests from multiple MPI processes, and call DGEMM that works on accelerators. For optimization, BLAS servers and MPI processes share matrix data with `mmap` system call to reduce the copying of data per function call. Actually, unlike ClearSpeed, Tesla GPUs allows sharing by multiple user processes. However, we implemented and invoked BLAS servers for GPUs to make arbitration of computation and communication by multiple DGEMM requests possible.

- We have found that an optimization called *look-ahead* introduced in the original HPL heavily degrades the performance of accelerators in our configuration. With look-ahead, panel broadcast and computation are overlapped to reduce critical path of the algorithm. To do this, the DGEMM calls are fragmented into small pieces to check incoming messages periodically, which is harmful for performance of accelerators. Instead, we have implemented a simple solution by creating a separate thread per process that makes DGEMM function calls for a large granule matrix portions. During that, the main thread calls the communication function. Thus we avoid the fragmentation while keeping the look-ahead optimization alive.
- In preliminary experiments, we have noticed that row-exchange communication consumes a considerable ratio in the total execution time. Its impact is larger than in typical CPU clusters with InifiniBand; this is because computation speed is accelerated in our case while performance of interconnect is similar to typical clusters. To improve this, we have modified the code to support overlap row exchange and kernel computation in update as follows. We conceptually divide the matrix to be updated by each process into two pieces in column-wise. Here we have the four operations to be done: (1) row exchange for left-hand part, (2) update for left-hand part, (3) row exchange for right-hand part and (4) update for right-hand part. Since (2) and (3) do not have dependency, it is possible to overlap them by multi-threading.
  This optimization splits DGEMM calls, which may looks inconsistent to the discussion above. However, we have confirmed that splitting into two is harmless and the advantage caused by overlapping dominates.
- BLAS servers accept DGEMM requests from multiple processes. When the requests are handled in a FIFO method, we have observed performance degradation. When a request for small matrices portion arrives during computation for a preceding request for large matrices portion, computation of the new request will be delayed for a long time, although the caller expects that it finishes soon. Instead of the naive FIFO method, we have introduced a priority so that small computation can interrupt large computations that are already running [8].

D. Tuning Methods

In heterogeneous systems, careful tuning cosidering characteristics of accelerators is necessary for high performance.

- When DGEMM tasks are thrown into CPU cores and accelerators, we found it is harmful to using all CPU cores for DGEMM. This is because PCI-communication to accelerators consumes considerable part of CPU cores (in our implementation, it is consumed by BLAS servers). Thus we prepare dedicated cores for this purpose; two cores for a ClearSpeed accelerator, and one core per Tesla GPUs. In Figure 3, boxes colored black denotes them.
- DGEMM performance heavily depends on the block size B, thus its tuning should be done carefully: (1) As described in Section IV-A, B should be large enough to hide effects of PCI communication. (2) Too large B is raises performance problem, since overhead of some non-kernel parts such as panel factorization, whose computation amount is $O(N^2B)$, gets larger. Moreover, too large B causes load imbalance among MPI processes. (3) Each accelerator type has preferable matrix sizes for DGEMM; B should be a multiple of 288 according to documents of ClearSpeed, while our DGEMM kernel on GPUs works well when it is a multiple of 16.
  Figure 4 shows DGEMM performance of three node types. In each node, all processes are used simultaneously. We see the performance with Tesla heavily depends on B; if it is 864 or less, the performance gets worse. On the other two types, the effect is much smaller. This is because we suffer from larger overhead of PCI communication to three accelerators on the first case. When B is 1152 or larger, the performance is almost constant; thus we choose $B = 1152$ in our evaluation.
  This size is much larger than $B = 128$ on Roadrunner, where small B is feasible since the matrix data are placed on device memory.

V. Linpack Evaluation and Discussion

We have conducted the evaluation on TSUBAME with the following configuration. As the software environments, we have used Voltaire MPI, GCC 4.1.2. We have used GotoBLAS library 1.26 on Opteron and Xeon, CSXL 3.11 on ClearSpeed. For Tesla GPUs, we have implemented DGEMM/DTRSM kernel functions.

[8]Another approaches such as fair scheduling with preemption may also work well, though we have not implemented.

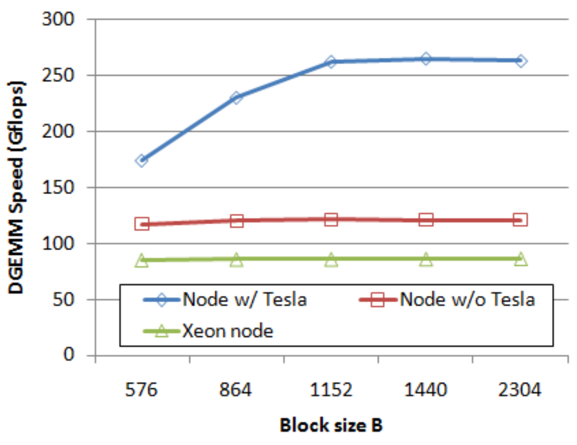


Figure 4. The DGEMM performance on each node. Matrices of size $(23040 \times B)$ and $(B \times 23040)$ are used. Overhead of PCI communication is included.

Table I
EFFECTS OF TWO OPTIMIZATION TECHNIQUES ON 64 NODES. SPEEDS ARE IN TFLOPS. THE MATRIX SIZE N IS 331,775 AND THE BLOCK SIZE B IS 1152.

	no	+overlap
no	7.76	7.71
+priority	8.13	8.74

A. Effects of Optimization Techniques

We evaluate the effects of optimization techniques by medium scale experiments that execute our Linpack implementation on 64 TSUBAME nodes. Among them, 32 nodes have two Tesla GPUs, while others do not. We focus on ”overlap between computation and row exchange” (overlap) and ”priority based kernel scheduling on BLAS server” (priority) described in Section IV-C. The speeds are shown in Table 1; with these techniques, we achieve 12.6% performance improvement. We see ”overlap” alone does not improve the performance, while combining the two causes a good effect. The reason for this is under investigation.

B. Linpack Performance on the Whole System

In the whole system evaluation, we have used the following nodes: (1) 312 TSUBAME nodes with Tesla, (2) 336 TSUBAME nodes without Tesla and (3) 80 Tsubasa Xeon nodes. Here the number of MPI processes are $312 \times 4 + 336 \times 2 + 80 \times 1 = 2000$; we let them compose of a process grid of size 40×50. Considering the host memory sizes, we let the matrix size be $N = 1,059,839$. As already described, the block size B is 1152.

With these settings, we have achieved R_{max} = 87.01TFlops, which is 2.28 times faster than the case only with Opteron (38.18TFlops). This result is ranked as 41th in Top500 ranking in June 2009.

Figure 5 shows the ratio of contribution by processor types in the peak performance and the Linpack performance.

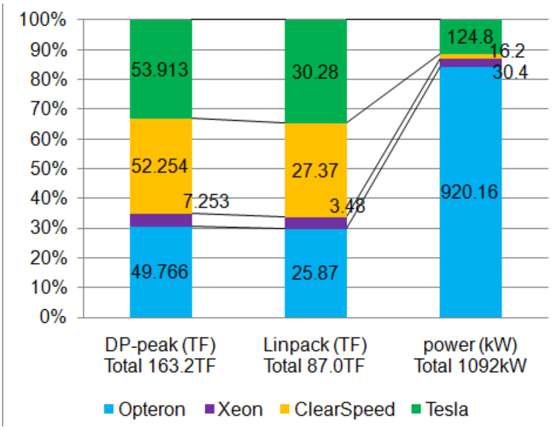


Figure 5. The ratio of contribution by processor types in the peak performance, the Linpack performance and power consumption.

The breakdown in Linpack performance can be obtained by totaling DGEMM task distributions of all MPI processes. We see the contributions of Opteron, Xeon, ClearSpeed and Tesla are 30%, 4%, 31%, 35%, respectively.

The figure also shows the breakdown of electric power consumption, which is obtained as follows. During Linpack execution, we have measured the power consumption of two TSUBAME nodes (one has Tesla, while another does not) and a Tesla S1070 box. By subtracting power of ClearSpeed from that of the second node, we obtain power consumption of Opteron CPUs per node. Currently, power consumption for ClearSpeed and Xeon (Tsubasa node) comes from ”typical power” in vendor’s announcements, since it is hard to measure their power consumption alone. Also note that this graph does not include power consumed by interconnect and cooling.

From the breakdown, we can see superior performance/ watt ratio of accelerators; while 66% of the Linpack performance comes from accelerators, their power consumption is only 15%. Especially, ClearSpeed accelerators consume only 1.5% of the system; however, we should pay attention for fair comparison between ClearSpeed and Tesla GPUs. While power consumption of a ClearSpeed is largely lower than that of a GPU, the former suffers from low bandwidth of device memory, which is almost unrelated to the Linpack performance. We will require further comparison by using other applications, especially ones that require broad memory bandwidth.

Since the total peak performance is R_{peak} =163.2TFlops, the efficiency is $R_{peak}/R_{max} = 87.01/163.2 = 53.3\%$. It is significantly lower than 76% on Roadrunner and close to 47% on Tianhe-1. In the next section, we analyze the gap between R_{peak} and R_{max}.

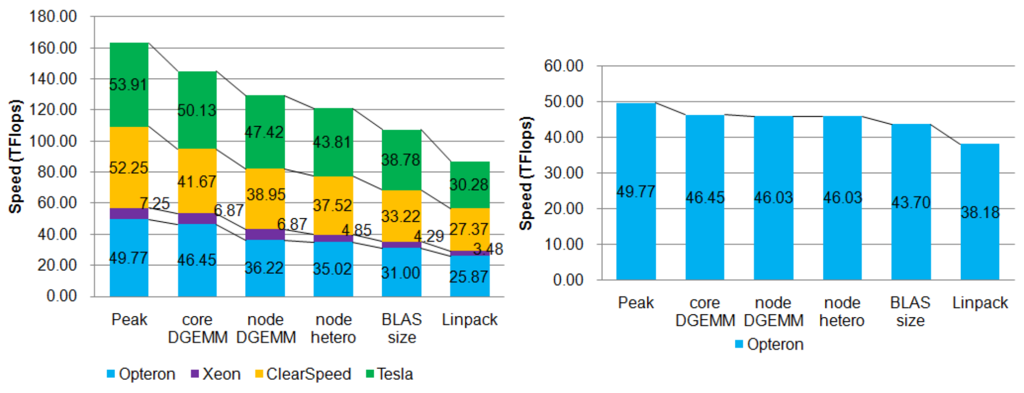


Figure 6. The detailed performance on the TSUBAME. (Left) When all types of CPUs/accelerators are used. Each color indicates the type of processors. Block size B is 1152. (Right) When only Opteron CPUs are used. Block size B is 240.

C. Performance Analysis

We can consider multiple factors as the reason of the gap between the peak performance and the Linpack performance, such as PCI communication overhead and inter-node heterogeneity, some of which may not appear in other heterogeneous systems. Thus we analysis impacts of each factor step by step; the following discussion is based on the left graph in Figure 6. The leftmost bar indicates the peak performance and the rightmost one is the Linpack performance on the whole system. The bars include breakdown of processor types. For comparison, the right graph of the figure shows the case where only Opteron CPUs are used on TSUBAME.

First, ”core DGEMM” in the graph corresponds to DGEMM performance per CPU core/ accelerator. For CPUs, we have measured speed of GotoBLAS with a single thread, and obtained 4.48GFlops on Opteron and 10.74GFlops on Xeon. For Tesla, we measured speed of our DGEMM kernel on a GPU, excluding PCI communication overhead; it is 80.33GFlops. Similarly, we obtained 64.3GFlops on a ClearSpeed. Then these numbers are totaled in the whole system [9] , and we obtain 145.1TFlops, which is 89% of the peak. Among the four types of processors, the performance of ClearSpeed is low, which is 80% of peak. On other processors, the relative performance is 93 to 95%, including Tesla.

Next, ”node DGEMM” corresponds to DGEMM performance that considers contention in PCI bus and memory bus. This is calculated based on DGEMM performance per node, which has shown in Figure 4. From the breakdown of processor types, we see that the performance of Opteron CPUs is largely affected, which is 22% lower than Opteron portion of ”core DGEMM”. On the other hand, the difference of ”node DGEMM” and ”core DGEMM” on the right graph (Opteron-only case) is only 1%; thus we can say impacts of dedicated cores in heterogeneous cases is fairly large. If we conducted the same evaluation on Roadrunner, there would be no difference between ”node DGEMM” and ”core DGEMM”, since PCI communication per DGEMM call is unnecessary there.

The ”node hetero” bar considers inter-node heterogeneity. In our Linpack evaluation, we invoke four processes on a node with Tesla, whose (node) DGEMM performance is 262.4GFlops, while we invoke two on a node without Tesla of 121.2GFlops. In Linpack, the performance is bottlenecked by the slowest process, thus the effective performance of a node with Tesla is $121.2/2 \times 4 = 242.4$GFlops, wasting 20GFlops. From the graph, we see performance degradation of 6.4% in the whole system, which does not appear in Opteron-only case or Roadrunner.

In Linpack execution, the size of matrix to be updated shrinks as computation proceeds. The ”BLAS size” bar includes this effect, which is obtained by a small scale experiment that emulates kernel invocations done in Linpack. It also includes effects of load imbalance among processes and DTRSM kernel invocations. According to the graph, we suffer from overhead of 12% compared with ”node hetero”. It is only 5% in Opteron-only case, and we consider the reason of the gap as follows. First, on accelerators, the DGEMM performance is more sensitive to shrinkage of matrices sizes to be updated. Next, the block size is smaller in Opteron-only case (B=240), the impact of load imbalance is smaller [10].

Finally, the difference between ”BLAS size” and ”Linpack” is due to effects of several factors, such as MPI communication and panel factorization. We see the gap is 19% in the whole system, while it is 13% in Opteron-only

[9]We do not consider the impact of dedicated cores for PCI communication described in Section IV-D; we simply computes 4.48(GFlops) $\times$ 16 (cores) $\times$ 648 (nodes) for Opteron.

[10]We also have measured performance when B=1152 in Opteron-only case, and observed that the difference between ”BLAS size” and ”node hetero” is 8%.

case. Although we would like to analyze this gap in more detail, we have not done it yet since it requires large scale experiments.

Through above discussion, we have demonstrated that the Linpack performance on TSUBAME is large affected by overhead of DGEMM kernel itself, PCI communication, inter-node heterogeneity and load imbalance, and analyzed their impacts quantitatively. Not all of the factors are essential in heterogeneous systems; we would not suffer from inter-node heterogeneity if nodes are homogeneously accelerated. And we would avoid overhead of PCI communication if matrix data are placed on device memory as in Roadrunner.

VI. Conclusion

We have implemented a Linpack implementation for the TSUBAME, a large scale heterogeneous system, evaluated it by using more than 10,000 general purpose CPUs, Tesla and ClearSpeed, and achieved 87.01TFlops. The system architecture we used has many different characteristics than the Roadrunner, which significantly impact the algorithm design. Also we have analyzed the gap between Linpack performance and the peak performance quantitatively. We expect these results and methodologies are useful to design more general applications on heterogeneous supercomputers.

Acknowledgement

This work is partially supported by JST-CREST ”ULP-HPC: Ultra Low-Power, High-Performance Computing via Modeling and Optimization of Next Generation HPC Technologies”, Grant-in-Aid for Scientific Research (No. 18049028) from MEXT, Japan, and Microsoft Technical Computing Initiative project. The Goto BLAS library is made by Kazushige Goto from University of Texas.

References

[1] M. Christen, O. Schenk, E. Neufeld, P. Messmer, and H. Burkhart, “Parallel data-locality aware stencil computations on modern micro-architectures,” in *Proceedings of IEEE International Parallel and Distributed Processing Symposium (IPDPS09)*, 2009.

[2] S. Edelkamp and D. Sulewski, “Parallel state space search on the GPU,” in *Proceedings of Symposium on Combinatorial Search (SoCS 2009)*, 2009.

[3] N. Galoppo, N. K. Govindaraju, M. Henson, and D. Manocha, “LU-GPU: Efficient algorithms for solving dense linear systems on graphics hardware,” in *Proceedings of IEEE/ACM Conference on Supercomputing (SC’05)*, 2005.

[4] J. Meng and K. Skadron, “Performance modeling and automatic ghost zone optimization for iterative stencil loops on GPUs,” in *Proceedings of ACM International Conference on Supercomputing (ICS ’09)*, 2009.

[5] A. Nukada, Y. Ogata, T. Endo, and S. Matsuoka, “Bandwidth intensive 3-D FFT kernel for GPUs using CUDA,” in *Proceedings of IEEE/ACM Conference on Supercomputing (SC’08)*, 2008.

[6] M. Fatica, “Accelerating Linpack with CUDA on heterogeneous clusters,” in *Proceedings of Workshop on General-purpose Computation on Graphics Processing Units (GPGPU ’09)*, 2009.

[7] D. Goddeke, R. Strzodka, J. Mohd-Yusof, P. McCormick, S. H. Buijssen, M. Grajewski, and S. Turek, “Exploring weak scalability for FEM calculations on a GPU-enhanced cluster,” *Parallel Computing, Special issue: High-performance computing using accelerators*, vol. 33, no. 10–11, pp. 685–699, 2007.

[8] J. C. Phillips and J. E. S. andand Klaus Schulten, “Adapting a message-driven parallel application to GPU-accelerated clusters,” in *Proceedings of IEEE/ACM Conference on Supercomputing (SC’08)*, 2008.

[9] S. Swaminarayan, T. C. Germann, K. Kadau, and G. C. Fossum, “369 Tflop/s molecular dynamics simulations on the Roadrunner general-purpose heterogeneous supercomputer,” in *Proceedings of IEEE/ACM Conference on Supercomputing (SC’08)*, 2008.

[10] M. Kistler, J. Gunnels, D. Brokenshire, and B. Benton, “Petascale computing with accelerators,” in *Proceedings of ACM Symposium on Principles and Practice of Paralle Computing (PPoPP09)*, 2009, pp. 241–250.

[11] T. Endo and S. Matsuoka, “Massive supercomputing coping with heterogeneity of modern accelerators,” in *Proceedings of IEEE International Parallel and Distributed Processing Symposium (IPDPS08)*, 2008.

[12] A. Petitet, R. C. Whaley, J. Dongarra, and A. Cleary, “HPL - a portable implementation of the high-performance Linpack benchmark for distributed-memory computers,” `http://www.netlib.org/benchmark/hpl/`.