ISSN 1342-2812

Research Reports on Mathematical and Computing Sciences

Methodology for Coping with Heterogeneity of Modern Accelerators on a Massive Supercomputing Scale

Toshio Endo and Satoshi Matsuoka

November 2007, C–246

Department of
Mathematical and
Computing Sciences
Tokyo Institute of Technology

SERIES C: Computer Science

Methodology for Coping with Heterogeneity of Modern Accelerators on a Massive Supercomputing Scale

Toshio Endo and Satoshi Matsuoka
Tokyo Institute of Technology, Japan

November 2007

Abstract

Heterogeneous supercomputers with combined general-purpose and accelerated CPUs promise to be the future major architecture due to their wide-ranging generality and superior performance / power ratio. However, developing applications that achieve effective scalability is still very difficult, and in fact unproven on large-scale machines in such combined setting. We show that an effective method for such heterogeneous systems so that the porting from applications written with homogeneous assumptions could be achieved. For this goal, we divide porting of applications into several steps, analyze performance of the kernel computation, create processes that virtualize the underlying processors, tune parameters with preferences to accelerators, and balance the load between heterogeneous nodes. We apply our method to the parallel Linpack benchmark on the TSUBAME heterogeneous supercomputer. We efficiently utilize both 10,000 general purpose CPU cores and 648 SIMD accelerators in a combined fashion—the resulting 56.43 TFlops utilized the entire machine, and not only ranked significantly on the Top500 supercomputer list, but also it is the highest Linpack performance on heterogeneous systems in the world.

1 Introduction

Although massively-parallel homogenous supercomputers using low power CPUs and/ or multi-core CPUs such as BlueGene/L [8] are one promising road to petascale, another approach is *heterogeneous* supercomputers. They consist of general purpose CPUs and more specific, dedicated programmable accelerated processors. CPUs offer flexibility and generality over wide-ranging classes of applications, while accelerators provide high performance / power ratio for specific computation patterns, so their combined use would be ideal. Several research or commercial projects have taken diverse approaches for heterogeneous architecture; AMD Torrenza and a vector acceleration project by HP [4] connect heterogeneous resources via front side bus, although their scalability in massive scale is unproved. An example of petascale heterogeneous supercomputers will be the IBM Roadrunner that will combine AMD Opteron CPUs and the Cell Broadband Engines[7] to be built in 2008 and will boast 1.6 PetaFlops.

The largest heterogeneous supercomputer to date is our TSUBAME supercomputer, installed at Tokyo Institute of Technology in April 2006. TSUBAME is also currently the fastest supercomputer in Asia Pacific, based on 5,240 dual-core Opteron CPUs (10,480 CPU cores) and 648 SIMD vector accelerator boards from ClearSpeed[1]. The peak performance of Opterons is approximately 50.4TFlops, and that of 360 accelerator boards is 52.2TFlops, totaling 102.6TFlops of computing resources.

The major issues that arise on heterogeneous supercomputers are how users can develop programs that effectively use the hybrid computing resources. Despite the fairly long history in supercomputers with vector acceleration options such as the CM-5 and the Meiko CS-2, and the recent high interest

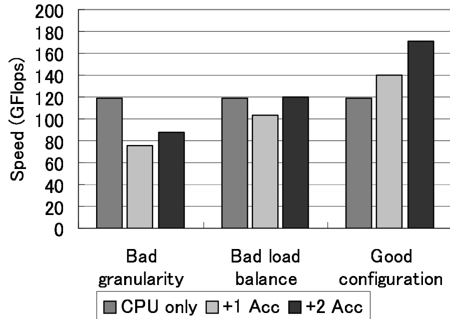

Figure 1: HPL performance on two TSUBAME nodes. In '+n Acc', we use n accelerator boards in addition to CPUs.

on SIMD-vector programming, there have been little results in scalability of tightly-coupled, parallel code on large heterogeneous machines. On the software side, recent research in pursuing easy programming with heterogeneity has focused on small scale (often a single node) systems only, including Sequoia [10], CUDA[2] and Accelerator[18]. We note that such difficulty is rather fundamental; the vast differences in the architecture between CPUs and accelerators not only make programming challenging in terms of coding, but also make proper load balancing difficult. Even a small deviation from the optimally balanced load may have adverse effect on the overall scalability, especially for tightly-coupled parallel codes. As a result, previous results have been for loosely-coupled codes, and/or small scale computation.

Our work, however, demonstrates that combined usage of resources can be achieved with high efficiency and scaling with existing tightly-coupled parallel codes. Our approach consists of 4 steps, namely (1) carefully analyze and model the kernel of the code, (2) create virtual processes to emulate apparent homogeneity with underlying heterogeneous resources, (3) parameter tuning with preferences to accelerators, and (4) load balance taking heterogeneity and overhead for accelerators into account. We demonstrate the effectiveness of our approach by modifying the High Performance Linpack(HPL)[17] for heterogeneous systems consisting of massive numbers of general-purpose and acceleration CPUs, and demonstrate its efficiency and scalability on TSUBAME.

Without careful strategy and tuning, HPL performance even becomes slower due to introduction of accelerators as shown in Figure 1; the graph compares the performance in a 'CPU-only' case and with acceleration on two TSUBAME nodes. When we adopt improper granules of compute size ('Bad granularity' in the graph) or ignore the overhead of introducing accelerators ('Bad load balance'), the overall performance lags behind CPU-only case. With careful parameter configurations ('Good configuration'), HPL with accelerators gets significantly faster than CPU-only. And in fact it scales on TSUBAME with varying number of node configurations, up to the scale of the full machine, achieving 56.43TFlops on more than 10,000 CPU cores and 648 accelerators, a significant improvement from our CPU-only result of 38.18TFlops. As far as we know this is the first time that a heterogeneous supercomputer has been ranked high (9 to 16th) on the Top500 list[3].

2 TSUBAME: The Heterogeneous 100 TeraFlops Supercomputer

NEC/Sun TSUBAME is a 'fat-node' supercomputer cluster that consists of 655 SunFire X4600 compute nodes. Each compute node has 16 2.4 GHz AMD Opteron CPU cores, with 32 GBytes of shared memory, while storage server nodes total 1.6PBytes in raw capacity. Both the computing

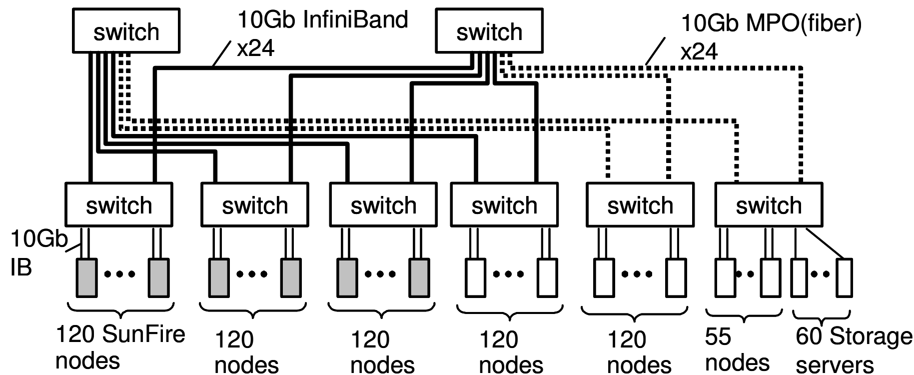

Figure 2: Overview of the TSUBAME Architecture. Currently, shaded SunFire nodes are equipped with ClearSpeed accelerators.

and storage nodes have Voltaire InfiniBand 4x HCAs, interconnected in a restricted fat tree topology with two core switches and six edge switches, 288-port Voltaire ISR9288, as shown in Figure 2. Each node is connected to one of edge switches via two 10Gbps InfiniBand links, and switches are mutually interconnected via 24 trunked InfiniBand links. On each compute node, 64bit SuSE Linux Enterprise Server with kernel 2.6.5 runs. Users invoke their jobs via the Sun N1 Grid Engine scheduler, though our large scale experiments are done in a dedicated situation.

Among the 655 TSUBAME compute nodes, 648 are equipped with the ClearSpeed Advance X620 Accelerator Boards [1] on their PCI-X slots. Each board hosts two ClearSpeed CSX600 SIMD processors, each of which has 96 SIMD processing elements; its theoretical peak performance is 80.6GFlops (double precision) at 210MHz clock speed [1]. A board also hosts 1GB DDR-SDRAM, which CSX600 processors can directly access. Since the memory is separated from host memory, communication of input/output data via PCI-X is necessary. A notable feature of the boards is low power; consumption of a board is only about 25W, or total of 16kW for the entire 648 boards, consisting less than 2% of the power consumption of TSUBAME while offering 50% of the overall compute power. Currently ClearSpeed provides following usages; a SIMD parallel programming language C^n, CSXL library for basic linear algebra (BLAS), mainly used in this paper, and CSFFT library for fast Fourier transformation. Recently molecular dynamics package is also provided.

In heterogeneously accelerated supercomputers such as TSUBAME, we may be are faced with two types of heterogeneity, namely *intra-node heterogeneity* and *inter-node heterogeneity*. The former is that accelerated nodes have both general purpose CPUs and SIMD accelerators that have different characteristics, with promise of generality and low power/space consumption. We may be faced the latter for less technical reasons; if part of nodes are equipped with accelerators and others are not, we have to take difference between accelerated and non-accelerated nodes. This was the case with TSUBAME, since only about 55% of the nodes were equipped with accelerators until October 2007. In general, computer centers with large scale clusters may come across similar situations, for changes of commercial products or reasons of the budget. Therefore techniques to tackle this situation are becoming important for efficient large scale computing. In more general cases, the numbers and/or the type of accelerators may differ among the nodes.

As already discussed, we have several approaches to introduce heterogeneity into supercomputers. TSUBAME's approach to equip accelerators on PCI-X slots has the following advantages. First, it is more portable; accelerators tightly coupled with general purpose CPUs, such as AMD Torrenza, are attractive for their superior bandwidth, but they heavily depend on CPU architecture. Secondly,

[1]The clock speed is configurable up to 250MHz, but faster clock may make computational results unstable

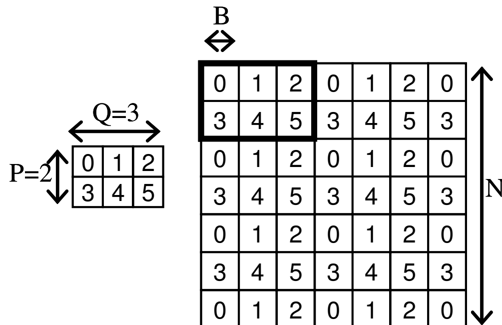

Figure 3: (Left) Process grid with $P \times Q = 2 \times 3$ processes. (Right) Two dimensional block cyclic distribution of $N \times N$ matrix on 6 processes. Block size is B.

it is power and space efficient; if we combine separately designed systems, such as PC cluster and a vector machine, we may suffer from double power consumption and space. On the other hand, ClearSpeed boards consume only 2% of power of the TSUBAME system and no additional racking space.

3 HPL and Preliminary Experiment

3.1 HPL and its Performance on a Homogeneous Environment

Our target program for porting, High performance Linpack (HPL), is a MPI based parallel benchmark that solves dense linear equations $Ax = b$ of order N. HPL computes LU decomposition of the matrix A with partial pivoting. Then it obtains a solution x by backward substitution computation. The amount of computation of the whole algorithm is $(2/3)N^3 + O(N^2)$.

HPL computing processes conceptually compose a process grid of size $P \times Q$, on which the matrix is distributed according to two-dimensional block cyclic distribution as in Figure 3. Here, we let N and B be size of matrix and size of block, respectively. Almost all of the computation time of HPL is occupied by the LU decomposition, where an iteration of the outermost loop corresponds to a block column. On the k-th iteration, HPL performs *panel decomposition, panel broadcast, row exchange communication,* and *update* [17]. The overall computation amount for panel decomposition is $O(N^2B)$, while communication (due to panel broadcast and row exchange) is $O(N^2(P+Q))$. The update phase is usually the most time consuming at $O(N^3)$, being dominated by matrix multiplication.

To discuss porting HPL to heterogeneous architecture, the following properties have to be taken into account. First, HPL performance is largely determined by that of matrix multiply (`dgemm`). Thus the common strategy for high performance is to use the fastest basic linear algebra software (BLAS) optimized for processors, such as GOTO BLAS library[12] or Intel MKL in the case of x86 architecture.

Next, HPL distributes tasks equally to all processes since it is designed for homogeneous architecture. As processes are tightly coupled by data dependencies, the overall performance is degraded if speeds of processes are uneven. HPL introduces an optimization called 'look-ahead', which overlaps computation and panel broadcast communication. Although it makes HPL more tolerant to communication latency or temporary delays of processes, it alone does not solve heterogeneous speeds among processes.

We evaluated the original HPL with 10,368 CPU cores on 648 TSUBAME nodes *without ac-*

Table 1: HPL parameters used in evaluation with 10,368 CPU cores

Matrix size	1334160	Panel broadcast	1ring
Block size	240	Look-ahead depth	1
Process mapping	Row-major	Swap	Mix (threshold=240)
# of processes	36×144	Matrix form	L1 trans, U trans
Panel factorization	Right-looking	Equilibration	yes
NBMIN, NDIV	4, 2	Alignment	8 double words

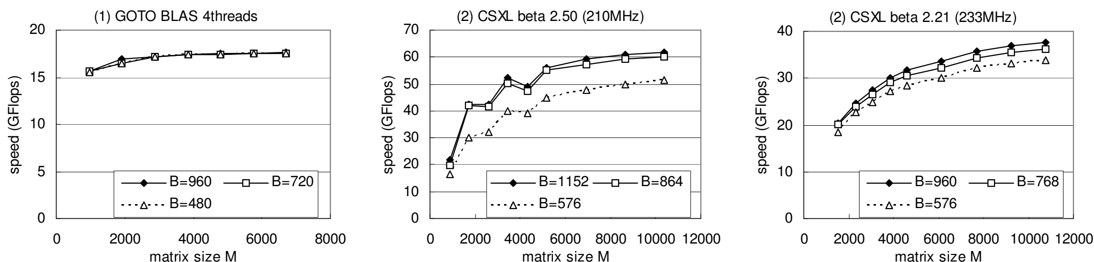

Figure 4: Performance of matrix multiply by GOTO BLAS library and CSXL library (beta 2.50, beta 2.21). Matrix sizes are $(M \times B)$ and $(B \times M)$.

celeration using GOTO BLAS. Its results was 38.18 TFlops with matrix size $N = 1,334,160$ on $P \times Q = 36 \times 144 = 5184$ processes, achieving the efficiency of 76.6% with runtime of 11.5 hours. HPL provides many tuning parameters, such as broadcast topology and block size; the parameters in this evaluation, in which eight processes with two threads each are invoked on each node, are shown in Table 1. The result was ranked No. 7 on the Top500 list in June 2006.

3.2 Kernel Performance on CPUs and accelerators

Performance of HPL is largely determined by kernel BLAS libraries. Here we compare characteristics of GOTO BLAS for Opterons and CSXL for ClearSpeed accelerators. The former is one of the fastest BLAS libraries for Intel/AMD processors developed by K. Goto. Graph (1) in Figure 4 shows its performance with four threads, denoting multiplications of $M \times B$ and $B \times M$ matrices, where M is usually much larger than B. This kind of computation is dominant in HPL. We see the performance ranges from 15.5 to 17.6 GFlops, which are 81 to 92% efficiency with respect to the 19.2 GFlops peak performance of four 2.4GHz Opteron cores. An interesting characteristic is the stability of performance for varying B and M compared to CSXL described next.

CSXL for ClearSpeed accelerators consists of a wrapper (`.so`) running on the host CPU and computing component (`.csx`) running on the accelerator. When a BLAS function is called, the wrapper sends the input matrix data to the accelerator via the PCI-X bus, then the accelerator computes and returns results to the CPU. For efficiency, the wrapper overlaps computation and communication. It also has a facility to dispatch the computation to both the CPUs and the accelerator.

Since CSXL performance has been largely improved by software update, we examined two versions used in Section 5. Figure 4 includes the performance of matrix multiply of CSXL for version beta 2.50 (2). For reference, that of beta 2.21, older and slower version is also shown (3). We see beta 2.50 achieves about 60GFlops with $M = 10,368$ and $B \geq 864$. We also see that despite their

high performance for large matrices, they degrade significantly with smaller matrices unlike GOTO BLAS; they exhibit only about 20GFlops. We consider this degradation is due to communication overhead over the PCI-X bus and the startup cost of SIMD-Vector computation. The sensitiveness to matrix sizes of CSXL suggests that we require finer tuning in heterogeneous setting.

4 Making Tightly-Coupled Applications Scalable on Heterogeneous Supercomputers

4.1 Coping with Heterogeneity

HPL, like many tightly-coupled parallel applications, assumes the underlying computational elements to be homogeneous. However, for heterogeneous supercomputers such as TSUBAME, we are faced with intra-node and inter-node heterogeneity, though the latter has been relieved recently. Both cause load imbalances leading to inefficiency, and overall performance potentially being worse than non-accelerated cases, as we see in Section 1.

Although there have been various proposals for tightly-coupled numerical algorithms for heterogeneous compute nodes, none we have seen to date have dealt with these two types of heterogeneity at the same time at scale. In fact, a typical method of simply assigning data regions of different sizes according to processor performance would not work well, due to the divergent characteristics of general purpose CPUs vs. accelerators. Rather, we propose to cope with the problem with a methodology consisting of the following steps. Although it is currently difficult to automate the entire process, it has proven to be a good guideline for massive scalability on TSUBAME, as well as serving as a basis for future automation we are currently working on:

Step 1 Analyze the performance models of general purpose CPUs and accelerators, in particular their computational kernels. Obtain analytical and/or empirical performance models according to problem sizes and other parameters.

Step 2 Create virtual processes of equal granularity to abstract out the differences between general purpose CPUs and accelerators, so that tightly-coupled programs with homogeneity assumptions could be almost directly executed. Since accelerators would usually be faster than general purpose CPUs but less flexible, a single accelerator would correspond to multiple instances of virtual processes. Here some general purpose CPUs may have to supplement other parts of accelerator-based virtual process where accelerators themselves cannot execute or would be very slow.

Step 3 Since accelerators prefer larger data sizes, we tune the parameters so that the compromise between the general purpose CPUs and accelerators are reached. In the case of HPL, we also remove parts of code where granularity is sacrificed.

Step 4 Balance the load taking inter-node heterogeneity into account, by configuring the number of virtual processes among accelerated and non-accelerated nodes. We also have to take care of overheads caused by introducing accelerators, such as CPU loads for PCI-X communication.

4.2 Applying the Methodology to HPL

As Step 1 for HPL, we consider the performance characteristics of general-purpose CPUs and accelerators, in particular to handle intra-node heterogeneity. One important point in the analysis is the Flops/ Byte ratio of kernel computation, which is `dgemm` (multiplication) of matrices of sizes $M \times B$ and $B \times M'$, where B is the predefined block size and typical smaller than M and M'. The amount of computation is $O(MM'B)$ while data size is $O(MM')$; so the Flops / Byte ratio is $O(B)$. The ratio is reasonably high as long as we adopt sufficiently larger B. Therefore in the following

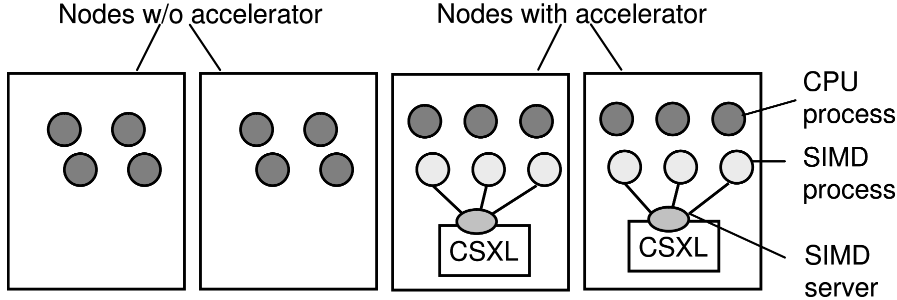

Figure 5: Virtual process configuration to harness CPUs and accelerators

steps, we can expect that a large block size would work well to drive CSXL efficiently, which is also empirically confirmed in Section 3.2.

As already mentioned, CSXL library can dispatch computation to accelerators and CPUs. However, this alone cannot account for inter-node heterogeneity directly. Rather, our approach is to use process virtualization to the control granularity of each process, while using dispatching as a supplemental method.

For Step 2, Figure 5 illustrates our virtual process configuration, albeit for a smaller number of nodes for simplicity. Here, we assume that only two nodes in the right are equipped with accelerators among the four. We introduce two kinds of virtualized processes: *CPU processes* and *SIMD processes*, each of which would be regarded as homogeneous process elements for the overarching homogeneous HPL. CPU processes behave in the same way in original HPL; they are linked with GOTO BLAS library and all computations are done on CPUs. SIMD processes also behave similarly, but only exist on nodes with accelerators. They off-load kernel BLAS computation onto accelerators, while other computations such as panel factorization are done on general-purpose CPUs.

We assign appropriate numbers of virtual processes so that workload *per process* is nearly equivalent, so as to support inter-node heterogeneity. In the figure, an accelerated node contains six virtual processes, while a non-accelerated node have only four.

Such virtualization requires that a single accelerator to represent several processes; however, naive implementation of this would be difficult, since the current ClearSpeed accelerators do not have the multi-processing capabilities of general purpose CPUs. To solve this problem, we implement a *SIMD server*, which is a daemon process linked with the CSXL library and invoked per available accelerator for direct access. It arbitrates matrix multiply requests from multiple SIMD processes, and calls the `dgemm` of CSXL library on its behalf, achieving time-sharing usage of accelerators. For optimization, a SIMD server and SIMD processes share matrix data with `mmap` system call to reduce the copying of data per function call.

4.3 Accelerator-Centric Granularity Tuning

As Step 3 of our methodology, we conduct parameter tuning so that CSXL receives preference from sufficiently large granularity, as GOTO BLAS is much more resilient to changes in the parameters. We basically reuse HPL parameters described in Table 1, but the block size B and process granularity have to be carefully tuned in order to achieve the best compromise to balance the intra- and inter-node heterogeneity. Actually, this step is conducted in parallel with Step 4, since the granularity and the number of processes are related tightly. We describe how we have tuned the parameters in our experiments with CSXL beta 2.50 and beta 2.51 [2].

[2]beta 2.51 is a minor updated version of 2.50, and its performance is similar

Block size: If the block size B is small, CSXL performance degrades significantly as shown in Section 3.2, thus the overall performance suffers considerably. On the other hand, B being too large is generally harmful for HPL and is thus avoided because it increases computational costs other than matrix multiplication such as panel factorization. Thus we have to take a compromise; by conducting empirical performance analysis and modeling, we decided on $B = 864$ with beta 2.50, which is about significantly larger than our CPU-only experiment, where $B = 240$.

Process granularity: We have another 'knob to turn', process granularity, as GOTO BLAS allows multiple threads to exist within a single process. When the number of threads per CPU process (hereafter T) is smaller, granularity of data set per process also gets smaller, which makes CSXL performance worse in SIMD processes. On the other hand, T being too large would make load balancing among nodes more difficult. With extensive preliminary measurements, we decided on $T = 4$, while it was two in our CPU-only experiment.

Additionally in HPL, we found that modification to code was necessary for the following reason. Our preliminary experiments indicated that heterogeneous HPL performance was much lower than expected, as the `dgemm` function calls became unexpectedly 'fragmented'. The fragmentation has been hardwired into HPL for the look-ahead optimization that overlaps between computation and communication; HPL repeatedly checks existence of messages after a small amount of `dgemm` computation. Whereas this works well with GOTO BLAS, it is disastrous for performance of CSXL. 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 of `dgemm` calls while keeping the look-ahead optimization alive.

4.4 Load Balancing Considering Inter-Node Heterogeneity

As Step 4, we determine the number of CPU and SIMD processes to be invoked on both non-accelerated and accelerated nodes. First, since the number of threads per CPU process T is four, four ($= 16/T$) CPU processes are invoked to utilize the 16 CPU cores on a non-accelerated node. At first glance, it might seem reasonable to invoke identical number of CPU processes on an accelerated node as well. Unfortunately, we found this does not work well in practice, as a SIMD server consumes 40-60% of a CPU load for communication with accelerators, severely degrading the overall performance ('Bad load balance' in Figure 1). In order to eliminate this overhead, we assign a semi-dedicated CPU to be the SIMD server, and decrease the number of CPU processes by one, which is three in our case. On the other hand, this introduces idle CPU cores on accelerated nodes due to relatively large process granularity. To exploit the idle cores, we let CSXL on SIMD servers use both accelerators and idle cores.

After the number of CPU processes, we determine SIMD processes by considering the balance between performance of accelerators and CPUs. We found that, thanks to tuning in Step 3, a simple method based on the peak performance of CSXL fairly works well. We also consider the performance of idle cores used by SIMD servers. After all such consideration, the number of SIMD processes on a accelerated node is four with beta 2.50 (it is three with beta 2.21).

5 Experimental Results

We have conducted large scale experiments on the whole TSUBAME four times including the homogeneous (CPU-only) case, as shown in Table 2. With our methodology and tuning to cope with heterogeneity, we have achieved 56.43TFlops on 648 nodes and 648 accelerators with CSXL beta 2.51. It is 47.8% faster than 38.18TFlops in the CPU-only case, and each accelerator contributes for

Table 2: History of our Linpack experiments on the whole TSUBAME system.

	spring 2006	autumn 2006	spring 2007	autumn 2007
# of CPU cores	10368	10368	10368	10368
# of accelerators	0	360	360	648
CSXL version	-	beta 2.21	beta 2.50	beta 2.51
Matrix size	1334160	1148160	1057536	1123200
Speed (TFlops)	38.18	47.38	48.88	56.43
Speedup to CPU-only	0%	+24.1%	+28.0%	+47.8%
Rank in Top500	7th	9th	14th	16th

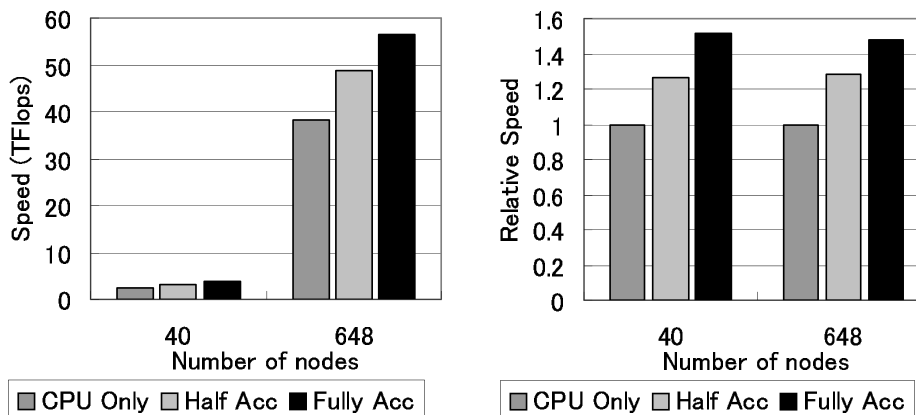

Figure 6: HPL performance on TSUBAME in heterogeneous setting. Half the nodes are equipped with accelerators in 'Half Acc', and all the nodes are accelerated in 'Full Acc'. The left graph shows speeds, and the right one shows relative performance normalized to 'CPU only' case.

speedup of 28.2GFlops. In the thrid experiments in spring 2007, we have shown that our methodology achieves good acceleration even in 'inter-node heterogeneous' case. Our result is currently the highest Linpack performance for the heterogeneous systems in the world.

In the rest of this section, we show detailed results of the third and the fourth experiments, unless explicitly stated. First, we show the retation between the number of accelerators and performance. In Figure 6, 'Full Acc' corresponds to homogeneously accelerated cases. 'Half Acc' corresponds to the inter-node heterogeneous cases, where half of the nodes are accelerated; exceptionally, 360 (55%) are accelerated in the case of 648 nodes. The left graph in the figure shows speeds in TFlops, and the right shows relative performance normalized to 'CPU only'.

When all the nodes are accelerated, the performance improves by a factor of 48 to 52% over the CPU-only case. Interestingly, the right graph shows that the speedup is almost linear with the number of accelerated nodes, though the number of runs are not sufficient yet. As far as we have observed, these results indicate that we are properly handling inter-node heterogeneity, and that an increase in accelerated nodes allow extrapolated acceleration. This observation suggests us that further acceleration that introduces several accelerator boards or several kinds of accelerators per node is promising, though we will have to consider additional communication overhead with accelerators.

Next, Figure 7 shows the relationship between the matrix size and HPL performance on 40 nodes.

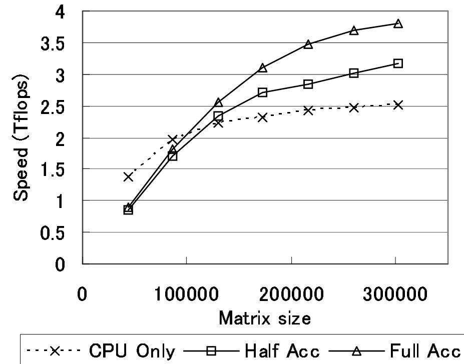

Figure 7: HPL performance on 40 nodes with varying matrix sizes N.

The performance is obviously better with larger N in all cases, which is natural due to characteristics of HPL. We also notice that the tendency is stronger in accelerated cases, because of characteristics of CSXL that favors larger granularity as shown in 3.2. In fact, accelerated cases are slower than 'CPU only' when the matrix size is about 90,000 or less. This indicates that our methodology has room for improvement, so that load balancing is optimized according to data sizes.

6 Related Work

Heterogeneous parallel computing has a longer history, especially for heterogeneity in CPU clock speed. To execute data parallel program on hybrid CPUs, a traditional approach is to assign proper sizes of data to each CPU [5, 6]. However, since many existing codes are designed for homogeneous systems, porting them requires heavy modification.

Our approach, which virtualizes and controls the number of processes per node, is more relevant to AMPI on Charm++[14], although it would entail similar problems as the untuned version of our HPL when applied to environments with accelerators. The Charm++ team has started to accommodate the Cell processor. Kishimoto et al.[15] have constructed a performance model of HPL on hybrid CPUs that allows to determine tuning parameters automatically by using feedback from measurements on each class of CPU. The model takes the matrix size and the number of processors on nodes into account; however, with accelerators, we have observed that tuning space gets much broader. It would be interesting to adapt their model to support accelerators.

Although heterogeneous architectures with accelerators are considered promising, research projects on scalability of heterogeneous supercomputers are still rare. Recently scientific computations on SIMD and/or multi-core processors, including general purpose GPU (GPGPU) and Cell Broadband Engine, have attracted considerable attention. There have been many reports on successful classes of algorithms on GPUs[11, 13, 16] and Cell BE[9], however, most of projects focus on single processor/ node systems, though low degree of parallelism is involved.

Not only design and implementation of algorithms, but some recent projects pursue easy programming. Sequoia by Fatahalian et al.[10] is a programming system that supports processors with different memory hierarchy. Although it supports portable programming for Cell BE and ordinary CPUs, porting existing parallel codes requires rewriting them heavily to conform to Sequoia's divide-conquer style. CUDA[2] allows programmers to write C-like programs in a SIMD style, and Accelerator by Tarditi et al.[18] provides high-level matrix/vector operations, which are automatically executed on a GPU. Currently, the above projects also focus on single node systems, thus scalability of accelerated computation has been unproven. On the other hand, our focus is to prove

scalability of our methodology on large scale machines.

7 Conclusion

Heterogeneous supercomputers with combined general-purpose CPUs and accelerators are expected to be one of the major architectures in the near future for their generality and superior performance / power ratio. We have shown the methodology whereby such supercomputers can be harnessed, and demonstrated its effectiveness with a large scale Linpack evaluation. By using the TSUBAME supercomputer equipped with 10,480 Opteron cores and 648 ClearSpeed SIMD accelerators, we have obtained 56.43TFlops Linpack performance, which was ranked as 16th in Top500 ranking in November 2007. This result is currently the highest Linpack performance on heterogeneous systems in the world.

We have demonstrated scalability of our techniques, however, dispatching of tasks at the library level requires that the application Flops / Byte ratio to be moderately high, which was the case for HPL. The question then is, is our methodology generally applicable to wide-ranging classes of applications? We believe so, as the situation will not be so different for other classes of problems where computational kernels may be different, but still the workload being divisible. In fact, we are currently working on applying our methods to other applications based on multi-dimensional FFT kernel. Although we have shown a step-by-step methodology that would serve as a guideline, for general applications, we need to tune the load in an automated fashion, perhaps at runtime. We also hope that our work will expand the research area of applications to identify their feasibility in heterogeneous architectures.

References

[1] ClearSpeed Technology Inc. `http://www.clearspeed.com/`.

[2] NVIDIA CUDA Homepage. `http://developer.nvidia.com/cuda`.

[3] TOP500 supercomputer sites. `http://www.top500.org/`.

[4] J. Collard N. Jouppi C. Lemuet, J. Sampson. The potential energy efficiency of vector acceleration. In *Proceedings of IEEE/ACM SC06 Conference*, 2006.

[5] P. E. Crandall and M. J. Quinn. Block data decomposition for data-parallel programming on a heterogeneous workstation network. In *Proceedings of IEEE International Symposium on High Performance Distributed Computing (HPDC)*, pages 42–49, 1993.

[6] T. Endo, K. Kaneda, K. Taura, and A. Yonezawa. High performance LU factorization for non-dedicated clusters. In *Proceedings of IEEE International Symposium on Cluster Computing and the Grid (CCGrid)*, pages 678–685, 2004.

[7] D. Pham et al. The design and implementation of a first-generation CELL processor. In *Proceedings of IEEE International Solid-State Circuits Conference (ISSCC)*, pages 184–, 2005.

[8] N. R. Adiga et al. An overview of the BlueGene/L supercomputer. In *Proceedings of IEEE/ACM SC02 Conference*, 2002.

[9] J. Fernandez A. L.Verbanescu M. Kistler F. Petrini, G. Fossum and M. Perrone. Multicore surprises: Lessons learned from optimizing sweep3d on the cell broadband engine. In *Proceedings of IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2007.

[10] K. Fatahalian, T. J. Knight, M. Houston, M. Erez, D. R. Horn, L. Leem, J. Y. Park, M. Ren, A. Aiken, W. J. Dally, and P. Hanrahan. Sequoia: Programming the memory hierarchy. In *Proceedings of IEEE/ACM SC06 Conference*, 2006.

[11] 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 SC05 Conference*, 2005.

[12] Kazushige Goto. Goto BLAS. `http://www.tacc.utexas.edu/resources/software/`.

[13] N. K. Govindaraju, S. Larsen, J. Gray, and D. Manocha. A memory model for scientific algorithms on graphics processors. In *Proceedings of IEEE/ACM SC06 Conference*, 2006.

[14] C. Huang, G. Zheng, S. Kumar, and L. V. Kale. Performance evaluation of adaptive MPI. In *Proceedings of ACM Symposium on Principles and Practice of Parallel Programming (PPoPP)*, pages 12–21, 2006.

[15] Y. Kishimoto and S. Ichikawa. An execution-time estimation model for heterogeneous clusters. In *Proceedings of IEEE International Heterogeneous Computing Workshop (HCW)*, 2004.

[16] S. Ohshima, K. Kise, T. Katagiri, and T. Yuba. Parallel processing of matrix multiplication in a CPU and GPU heterogeneous environment. In *7th International Meeting on High Performance Computing for Computational Science (VECPAR'06)*, pages 41–50, July 2006.

[17] 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/`.

[18] David Tarditi, Sidd Puri, and Jose Oglesby. Accelerator: Using data parallelism to program GPUs for general-purpose uses. In *Proceedings of ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-XII)*, 2006.