Exploring system architectures for next-generation CFD simulations in the postpeta-scale era

Kazuhiko KOMATSU*,**, Ryusuke EGAWA***, Hiroyuki TAKIZAWA***, Takashi SOGAY†, Akihiro MUSA†† and Hiroaki KOBAYASHI*

* Cyberscience Center, Tohoku University
  6-3 Aramaki Aza Aoba, Aoba-ku, Sendai 980-8578, Japan
  E-mail: komatsu@isc.tohoku.ac.jp

** Japan Science and Technology Agency
  Core Research for Evolutional Science and Technology
  7, Gobancho, Chiyoda-ku, Tokyo, 102-0076, Japan

*** Graduate School of Information Sciences, Tohoku University
  6-3 Aramaki Aza Aoba, Aoba-ku, Sendai 980-8578, Japan

† NEC System Technologies, Ltd.
  1-4-24 Shiromi, Cyuuou-ku, Osaka 540 8551, Japan

†† NEC Corporation
  5-7-1 Shiba, Minato-ku, Tokyo 108-8001, Japan

Received 7 March 2014

Abstract

CFD simulations with uniform grids have been paid attention as a next-generation CFD simulation on a large-scale supercomputing system. The Building-Cube Method (BCM) is one of the next-generation CFD methods. The basic idea is to balance loads of calculations among processing elements on a supercomputing system by dividing the whole calculations into many parallel tasks with the same amount of computation. Thus, it is suitable for highly parallel computation on supercomputing systems. This paper firstly implements BCM on five supercomputing systems as an example of a next-generation CFD simulation in the upcoming postpeta-scale era. Then, by theoretical analyses and performance evaluations, this paper clarifies the requirements of future supercomputing systems for a next-generation CFD simulation. The performance evaluations show that as the number of processing elements increases, the imbalance of data exchanges among nodes becomes more serious than that of calculations even in a next-generation CFD simulation. While the calculation time can ideally be reduced according to the number of processing elements, the data transfer time becomes dominant in the total execution time. Different from the massively-parallel system architecture, the number of nodes in a system should be as small as possible to prevent the data transfer. The performance analyses also show that the memory bandwidth limits the performance of BCM and use of an on-chip memory is effective to improve the performance. A memory subsystem that achieves a higher sustained memory bandwidth is required. Therefore, a supercomputing system that consists of a small number of high-performance nodes is essential to achieve high sustained performance of the next-generation CFD in the upcoming postpeta-scale era by reducing the data transfers, which becomes eventually a bottleneck in large-scale simulation.

Key words: Next-generation supercomputing systems, Next-generation CFD simulation, Building-Cube Method, Performance evaluation

1. Introduction

In the postpeta-scale era, due to several constraints such as power, area, cooling capability, and cost limitations, no one can build a universal supercomputing system that can efficiently run any applications. Hence, under the constraints, a future supercomputing system has to be designed so as to maximize the sustained performance of its target applications. That is, a future supercomputing system must be designed by taking into account the characteristics of its target applications, even though the applications would also be adjusted for the future system at a certain level. Co-design of a system
and its application is mandatory for efficient supercomputing in the postpeta-scale era. This paper explores the system architectures for co-design with next-generation CFD simulations.

The conventional CFD simulations with unstructured meshes and/or boundary-fitted meshes have widely been utilized so far. However, these complicated CFD simulations are unable to achieve efficient calculations on supercomputing systems due to load imbalances and irregular memory accesses. Furthermore, data exchanges can easily be bottleneck in the CFD simulations with unstructured meshes. Due to the lower performance capabilities of memory subsystems and network subsystems than that of processing elements, the irregular data exchanges easily dominate the whole computation time on large-scale supercomputing systems. Thus, the scalability of CFD simulations with unstructured meshes and/or boundary-fitted meshes becomes low on a large-scale supercomputing system.

To solve these problems, uniform grids have been paid attention for a next-generation CFD simulation. A CFD simulation using uniform grids is well suited for highly parallel processing on supercomputing systems because equally-spaced meshes can produce many parallel tasks with the same amount of computation. By assigning the tasks into processing elements on a supercomputing system, it prevents load imbalance of calculations among processing elements. Especially, since supercomputing systems are equipped with a large number of processing elements, balancing loads among processing elements is essential. Therefore, uniform grids are candidate to alleviate load imbalance, irregular memory accesses and data exchanges for the large-scale supercomputing system. CFD simulations with uniform grids are expected as a next-generation CFD simulation in the postpeta-scale era.

The purpose of this work is to explore promising system architectures for co-design with next-generation CFD simulations. To this end, this paper clarifies the requirements for efficiently executing the next-generation CFD simulations by taking BCM as an example of next-generation CFD algorithms. The sustained performance of a BCM program is evaluated on supercomputing systems with different features. Through the performance evaluations, this paper discusses the required system architecture for the next-generation CFD simulation.

The rest of this paper is organized as follows. Section 2 introduces the overview of BCM and its incompressible flow solver. Section 3 briefly shows the implementations of BCM on various types of supercomputing systems. Section 4 shows evaluation results to discuss and analyze the performance of BCM on supercomputing systems. Based on the analyses, this paper clarifies the requirements of the next-generation supercomputing systems for a next-generation CFD. Finally, Section 5 gives conclusions of this paper.

2. Overview of the Building Cube Method

The Building-Cube Method (BCM) is a CFD method that employs uniform grids for efficient three-dimensional large-scale flow computations around practical geometries (Nakahashi et al., 2005)(Takahashi et al., 2009). In BCM, a whole flow domain is divided into sub-domains called cubes, and each cube is further divided into equally-spaced Cartesian meshes called cells, which is shown in Figure 1. The size of each cube is determined by complexities of geometries and flow features at its location. One advantage from the viewpoint of engineering is that BCM can easily deal with complex geometries and moving objects. Since its pre-processing, post-processing, and even the flow solver can be simplified by employing uniform grids, the complex geometries and moving objects can be easily handled. Another advantage is that high-order computational schemes can be easily utilized for the solver due to the simplicity of uniform grids. For example, by exploiting the characteristics of the uniform grids, high-resolution finite-difference schemes with low-pass filtering and high-order Lagrange interpolation are implemented (Ishida et al., 2008). The advantage of BCM...
Calculating temporal velocity field

Data exchange of the field among cubes

SOR iterations

Calculating pressure field

Data exchange of the field among cubes

Convergent?

Yes

Calculating real velocity field

Data exchange of the field among cubes

Next time step?

Yes

No

Flow solver start

Flow solver end

Fig. 2 Flowchart of the BCM flow solver.

from the viewpoint of the computer science is that it can easily achieve good load balance of calculations among processing elements. Since the same computational cost and the same data size are necessary for each cube and the calculations of each cube are independent of each other, the calculations of cubes can easily be decomposed into many data parallel tasks of the same size. By assigning the equal numbers of tasks into processing elements, the load of each processing element becomes the same.

The flowchart of the BCM flow solver is shown in Figure 2. The governing equation is the incompressible Navier-Stokes equation. The fractional-step method is used with the finite difference scheme on the staggered arrangement (Kim et al., 1985)(Perot, 1993)(Dukowicz et al., 1992). In the fractional-step method, the solver can be classified into three major stages in one time step; solver stages for calculating a temporal velocity field, for calculating a pressure field, and for calculating a real velocity field. In each stage, calculations of its field and data exchanges between cubes are executed.

In the BCM incompressible flow solver, the temporal velocity field is solved by the momentum equation except the pressure term. The pressure field is solved for satisfying the divergence-free flow-field. The real velocity field is solved by the correction using the pressure gradient to the temporal velocity field. The third-order upwind scheme (Kawamura et al., 1986) and second-order central scheme are used for discretization of the convective term and diffusive term, respectively. The elliptic equation for the pressure is solved by the SOR method.

The most dominant part among these stages is the calculation of the pressure field by solving the Poisson equation using the SOR method. The following equation is expressed for a uniform flow field without any wall boundary.

\[
\frac{P_{i+1} - 2P_i + P_{i-1}}{\Delta x^2} + \frac{P_{j+1} - 2P_j + P_{j-1}}{\Delta y^2} + \frac{P_{k+1} - 2P_k + P_{k-1}}{\Delta z^2} = \frac{1}{\Delta t} \left( \frac{u_{i+1} - u_i}{\Delta x} + \frac{v_{j+1} - v_j}{\Delta y} + \frac{w_{k+1} - w_k}{\Delta z} \right),
\]

where \(P_i, P_j, \) and \(P_k\) indicate pressure values at positions \(i, j, \) and \(k\) in coordinates \(x, y, \) and \(z\). \(u, v, \) and \(w\) indicate temporal velocities in each coordinate, respectively. \(\Delta t\) indicates a time step.

To calculate the pressure of one cell, a seven-point stencil calculation, which requires the pressure data of a cell and its six adjacent cells, is performed. As the stencil calculations for all cells in all cubes are repeated until the difference...
of the calculated field is sufficiently small, the calculations of the pressure field dominates the total execution time. The pressure calculations for cubes can be performed in parallel because they are independent and the computational cost of each cube is completely the same. By assigning the same number of cubes into processing elements, the load of calculations among processing elements is well balanced.

The data exchanges among cubes after the calculations of each field are important to accelerate BCM. As a simple example, assume that only one cube is assigned into a node and the sizes of whole cubes are the same. The cost of calculations of the pressure field is $O(n^3 \times n_{cubes}/PEs)$, where $n$ is the number of cells in a cube, $n_{cubes}$ is the total number of cubes, and $PEs$ is the number of processing elements. When the total number of cubes and the number of processing elements are the same, the cost becomes $O(n^3)$. The cost of data exchange is $O(6 \times n^2 \times PEs) = O(n^2 \times PEs)$ since data of six sides of a cube are exchanged with six adjacent cubes. If the number of processing elements is small, the system performance is limited by the computational capability of the system. However, since it is expected that a future system will have much more processing elements, the system performance will be limited by the data transfer capability rather than the computation capability. Moreover, when the size of a cube is different from the sizes of adjacent cubes, more data exchanges have to be performed. For example, when a side of a cube faces sides of four adjacent cubes, data exchanges are performed four times. Thus, the data transfer capability become more important to achieve a postpeta-scale CFD in a large supercomputing system.

3. Implementation of BCM on Various Systems

This section briefly describes the architecture-aware implementations and optimizations of BCM for scalar supercomputing systems and a vector supercomputing system, whose specifications are shown in Table 1 (Komatsu et al., 2012). In the implementation for scalar systems, the naive SOR method is employed. Although the Red-Black SOR method can eliminate the data dependency of cells, stride memory accesses are required. As the stride memory accesses degrade the spatial locality, the utilization of on-chip caches might decrease, resulting in performance degradation. Thus, to avoid degrading the performance by the inefficient memory accesses, the naive SOR method is employed. As the SOR calculation of each cube is independent, the SOR calculation can be executed in parallel by using only the parallelism of cubes. The number of the cubes becomes the maximum number of nodes that BCM can execute in parallel.

In the implementation for SX-9, the Red-Black SOR method is employed (Stueben et al., 1982). Although there are data dependencies among adjacent cells in a cube, the computational cost per cell is also the same. The Red-Black SOR method can eliminate the data dependency among adjacent cells. In the Red-Black method, all cells are classified into two colors one after another; red cells and black cells. By separately calculating the pressure filed of cells of each color, the pressure calculations for half of all cells in a cube can be performed in parallel. Although parallelizing the Red-Black SOR method generally shortens the length of the loop, use of mask tables can keep the loop long enough to utilize all of the vector units of SX-9. The effective use of an on-chip cache named Assignable Data Buffer (ADB) of SX-9 is also important to efficiently provide data to processing elements. Although the SX compiler can automatically optimize a code so as to use ADB, a programmer can also manually insert directives to direct the use of ADB for a specific array. Once data specified by programmers are accessed, these data are stored in ADB and can be used in the next accesses.

Table 1 Specifications of the supercomputing systems.

<table>
<thead>
<tr>
<th>System</th>
<th>Performance/ System</th>
<th># of Nodes</th>
<th># of Sockets/Node</th>
<th># of cores/ Socket</th>
<th>Memory Bandwidth</th>
<th>Memory Capacity</th>
<th>On-Chip Memory</th>
<th>Network Bandwidth</th>
<th>System Flops/Byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>SX-9</td>
<td>26.2TFlop/s</td>
<td>16</td>
<td>16</td>
<td>1</td>
<td>256GB/s</td>
<td>1TB</td>
<td>256KB ADB</td>
<td>2×128GB/s</td>
<td>0.40</td>
</tr>
<tr>
<td>Nehalem EX</td>
<td>1.74TFlop/s</td>
<td>6</td>
<td>4</td>
<td>8</td>
<td>34.1GB/s</td>
<td>512GB</td>
<td>256KB L2/core</td>
<td>4GB/s</td>
<td>2.13</td>
</tr>
<tr>
<td>Fujitsu FX1</td>
<td>5.16TFlop/s</td>
<td>128</td>
<td>1</td>
<td>4</td>
<td>40GB/s</td>
<td>32GB</td>
<td>6MB shared L2</td>
<td>2GB/s</td>
<td>1.0</td>
</tr>
<tr>
<td>Fujitsu FX10</td>
<td>5.68TFlop/s</td>
<td>24</td>
<td>1</td>
<td>16</td>
<td>85GB/s</td>
<td>32GB</td>
<td>12MB shared L2</td>
<td>5-50GB/s</td>
<td>2.78</td>
</tr>
<tr>
<td>SR16000M1</td>
<td>124TFlop/s</td>
<td>128</td>
<td>4</td>
<td>8</td>
<td>128GB/s</td>
<td>128GB</td>
<td>256KB L2/core</td>
<td>2×24-96GB/s</td>
<td>1.92</td>
</tr>
</tbody>
</table>

Table 2 Three-dimensional test models.

<table>
<thead>
<tr>
<th></th>
<th>F1 sphere</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of cells in a cube</td>
<td>$32^3$</td>
</tr>
<tr>
<td>Total number of cubes</td>
<td>5930</td>
</tr>
<tr>
<td>Total number of cells</td>
<td>194,314,240</td>
</tr>
</tbody>
</table>
without off-chip memory accesses. In the implementation, an ON_ADB directive is manually inserted to the source code to specify reusable data in the seven-point stencil calculations. Thus, the reusable data are kept in ADB across iterations of the loop. As the main memory and ADB can simultaneously provide data to a processing element, the vector processor can access those data at a high sustained bandwidth, and thereby achieve a high performance.

4. Performance Evaluations and Discussions

4.1. Environmental Setup

In order to clarify the basic characteristics of a next-generation CFD, the flow simulations using BCM around 3D test models are performed on the supercomputing systems shown in Table 1. F1 is a large model of 200 million cells, and Sphere is a small model of 5 million cells as shown in Table 2 and Figure 3. The Reynolds numbers for the F1 model and the Sphere model are set to $2.6 \times 10^6$ and $10^2$, respectively. For automatic optimizations by the compiler, the BCM codes are compiled with the highest optimization options in each system. All calculations are performed using double-precision floating-point values. Flat-MPI executions are used for the experiments. In the case of the F1 model, two processes and eight processes are assigned into one node in FX10 and SR16000, respectively. Because 8GB memory capacity for each MPI process is not enough, the maximum number of cores in a node could not use in FX10 and SR16000. FX1 could not execute in the case of one core and two cores in the F1 model due to the restriction of the memory capacity of the system. In the cases of the sphere model and the other systems in the F1 model, one process is assigned to each core in a node.
4.2. Experimental Results and Discussions

Figures 4 and 5 show the performance comparisons of BCM on the supercomputing systems. The horizontal axis indicates the number of processing elements. The vertical axis indicates the execution times. Figure 4 shows that the performance improvements can be observed as the number of processing elements increases. The reason is that the parallel processing can effectively reduce the execution time of the calculations of each field. Figure 5 also shows that the execution time of every supercomputing system becomes short according to the number of processing elements. However, the execution time does not decrease in proportion to the number of cells. As the number of cubes in the Sphere model is smaller than that of the F1 model, the enough number of cubes cannot be assigned to processing elements in a node. As load imbalance among processing elements occurs, the execution time does not decrease. The second reason is that data exchanges are dominant in the execution time. Since the calculations of the Sphere model are smaller than that of the F1 model, the data exchanges easily become bottleneck compared with the F1 model.

In order to further analyze the execution time of BCM, the calculation times and the data exchange times required to solve each field are measured. Figures 6 and 7 show the breakdowns of the execution time of BCM. The horizontal axis indicates the number of processing elements and supercomputing systems. The vertical axis indicates the breakdown of the total execution time which includes the calculations and the data exchanges of a temporal velocity field, a pressure field, and a real velocity field, respectively. These two figures clearly show that the most dominant stage in the BCM solver is the pressure stage. No matter what kind of supercomputing systems are used and/or how many processing elements are used, the pressure stage always dominates the total execution time.

Furthermore, these figures also show that as the number of processing elements increases, the data exchanges generally become a dominant factor even though BCM employs uniform grids. Thus, the scalability is expected to be degraded in most cases when the number of processing elements increases even in the next-generation CFD simulation. This is due to imbalance of data exchange among processing elements incurred by the variety of cube sizes assigned to processing elements. When the sizes of the adjacent cubes are different, more data exchanges are necessary. As the data exchange is not easy to be parallelized, the time for the data exchange becomes dominant in the total execution time. The time for data exchange can be hidden by being overlapped with the time for calculations if the former time is shorter than the latter. However, as the time for data exchange increases with the number of processing elements, the data exchange time is exposed to the total execution time. In some cases such as SX-9 and FX1 in the F1 model and SX-9 and SR16000 M1 in the Sphere model, the percentage of the data exchanges do not increase even if the number of processing elements increases. This is due to the differences in MPI process mapping. Since MPI processes that need data exchanges with each other are mapped in cores in the same node, the data exchanges are performed within the node. Because the execution time for the data exchange within a node is shorter than that among nodes, the percentage of the data exchanges do not
Comparing the two figures, the percentages of data exchanges in the Sphere model are larger than those in the F1 model. The reason is the lack of parallel tasks due to a small number of cubes in the Sphere model. When the number of cubes assigned to processing elements is small, the imbalance of data exchange becomes serious in parallel processing. For example, the data exchange time of a processing element with two assigned cubes is two times longer than that with one cube. Thus, the smaller the number of assigned cubes to processing elements is, the more serious the effect of the imbalance of data exchange times is.

From these results on various supercomputing systems, it is clarified that the performance of the BCM is limited by the inter-node communications for data exchanges even though uniform grids are employed for the next-generation CFD method. As the data exchange consumes a longer time in a larger-scale system, it is one of major potential performance bottlenecks in a massively-parallel computing system. Therefore, the system should be designed so that the number of nodes becomes as small as possible in order to achieve a high sustained performance by minimizing the inter-node communications for data exchanges. In other words, the performance of each compute node should be as high as possible. Differently from the massively-parallel approach, by minimizing non-parallelizable parts of an application that include the data transfers, the execution time can become shorter than that using a larger-scale system.
Figure 4 also shows that the execution time of SX-9 is shorter than those of the other systems. As the most dominant stencil calculations in the pressure calculations are memory-intensive, the sustained memory bandwidth in a single node has a great impact on the node performance. The high requirements for a memory bandwidth can be derived from the theoretical analyses. The ratio of the floating point operations to the memory bandwidth required by Equation (1), which is called the theoretical arithmetic intensity (Flops/Byte ratio), becomes 0.18 since 28 double precision floating point operations and 19 variables are necessary for the pressure calculation. The arithmetic intensity derived from the source code of BCM is also investigated as 0.18 by counting the number of memory operations and floating point operations in the object code. The arithmetic intensity of the stencil calculation considering the on-chip memory of the recent scalar processors is reported from 0.20 to 0.33 (Datta et al., 2011)(Kamil et al., 2010). As the ratio of the floating point operations to the memory bandwidth of the supercomputing systems are from 0.4 to 2.78, the sustained memory bandwidth of the supercomputing systems limits the execution times of BCM.

Figure 8 shows the sustained memory bandwidth of the supercomputing systems. The horizontal axis indicates the supercomputing systems. The vertical axis indicates the sustained memory bandwidth measured by the STREAM benchmark (McCalpin) when the maximum number of threads in a node is utilized. This figure shows that 16 processors of a single SMP node of SX-9 achieve 2.6TB/s, which is the highest sustained memory bandwidth. The sustained memory bandwidth of a single node of SR16000 is higher than those of Nehalem EX, FX10, and FX1. Even though the peak memory bandwidth of a SPARC64VII processor of FX1 is higher than that of a Nehalem EX processor, the sustained memory bandwidth of a SPARC64VII processor is lower than that of a Nehalem EX processor. A SPARC64VII processor achieves only 10.0 GB/s, which is about 25% efficiency to the peak memory bandwidth, while a Nehalem EX processor achieves 20.2 GB/s, which is about 59% efficiency. From Figures 4 and 8, it is clarified that the sustained memory bandwidth greatly affects the execution times in a single-node. Along with the order of the sustained memory bandwidth of the supercomputing systems, the execution time of BCM becomes short.

Figure 9 shows the roofline models of a single node of each supercomputing system (Williams et al. 2009). The horizontal axis indicates arithmetic intensity. The vertical axis indicates attainable floating-point performance. From the peak performance and the stream memory bandwidth of each system, the limit of the attainable performance can be described in the roofline model. The line of each system indicates the upper bound of the attainable performance. As the arithmetic intensity of the pressure calculations ranges from 0.18 to 0.33, which is shown in the pink region in the figure, it is clarified that maximum performances of all supercomputing systems are limited by their sustained memory bandwidth.

In addition to the importance of a high memory bandwidth, the effective use of an on-chip memory is also important to provide data into CPUs. In order to clarify the effects of an on-chip memory, the performance evaluations are conducted. Only SX-9 is used for this evaluation, because the ADB of SX-9 can explicitly be controlled by a programmer and a
The effects of ADB in a single-node of SX-9. This figure compares the performances among automatic use of ADB by the SX compiler, manual use by a programmer, and no use. This figure shows that the sustained performances of automatic use and manual use are almost the same. This is because the SX compiler successfully detects the reusable data in the calculations. Furthermore, the sustained performances of automatic use and manual use are 1.8 times higher than that of no use of ADB at maximum. In the seven-point stencil calculations, as the same data is accessed seven times, ADB is useful for the calculations. Since the main memory and ADB can simultaneously provide data to vector pipelines, the memory bandwidth becomes substantially high due to the efficient use of ADB. Therefore, it is confirmed that use of an on-chip memory is effective for the pressure calculations.

From the experimental results, in order to enhance a single-node performance, a processor that realizes a higher sustained memory bandwidth is strongly required. A vector processor that is essentially equipped with a high performance memory subsystem is the candidate to satisfy the demands for memory performances. Thus, the advancement of the vector architecture that is equipped with both a high single-node performance and high memory bandwidth is the promising to realize a next-generation CFD simulation.

5. Concluding remarks

Due to many constraints to design supercomputing systems, co-design of a supercomputing system and its target application is required for efficient supercomputing in the postpeta-scale era. The purpose of this work is to explore promising system architectures for co-design with next-generation CFD simulations. In order to clarify the basic characteristics of a next-generation CFD method, this paper firstly implements and evaluates BCM on various types of supercomputing systems as an example of a next-generation CFD method. Since BCM employs uniform grids instead of unstructured meshes, it is suitable for large-scale CFD simulation in the postpeta-scale era. From the performance analyses on various supercomputing systems, it is clarified that the data exchanges become a performance bottleneck as the number of processing elements increases even in the next-generation CFD simulation, which causes the low scalability of the CFD. To avoid communicating among nodes, the number of nodes of the next-generation systems to execute a next-generation CFD simulation should be as small as possible. By achieving a high performance in a single-node, the total number of nodes in the next-generation systems can be reduced, which results in a reduction in data exchanges among nodes. Moreover, from the theoretical and experimental analyses of a single-node performance, it is clarified that the sustained memory bandwidth limits the performances. Since the ratio of the floating point operations to the memory bandwidth of the pressure calculations is lower than those of the supercomputing systems, a higher performance memory subsystem is
strongly required to enhance the single-node performance in the next-generation supercomputing systems. Also, it is clarified that the use of an on-chip memory for the reusable data can effectively recover the lack of the memory bandwidth. In conclusion, to achieve efficient execution of a next-generation CFD simulation in the postpeta-scale era, high single-node performance and high memory bandwidth are strongly required. Because the next-generation supercomputing system can reduce the execution time for non-parallelizable parts of CFD simulations such as data transfers, it can alleviate the scalability problem in a next-generation CFD simulation.

Acknowledgments

This research was partially supported by Grant-in-Aid for Scientific Research (S) #21226018 and Core Research of Evolutional Science and Technology of Japan Science and Technology Agency (JST CREST) “An Evolutionary Approach to Construction of a Software Development Environment for Massively-Parallel Heterogeneous Systems”.

References


Williams, S., Waterman, A., Patterson, D., Roofline: an insightful visual performance model for multicore architectures, Commun. ACM 52, 4 (2009), pp.65-76.