Multicores in Cloud Computing: Research Challenges for Applications

Lizhe Wang†, Jie Tao‡, Gregor von Laszewski†, Holger Marten‡

† Institute of Pervasive Technology, Indiana University at Bloomington, USA
‡ Steinbuch Centre for Computing, Karlsruhe Institute of Technology, Germany
Email: Lizhe.Wang@gmail.com, Jie.Tao@iwr.fzk.de, Laszewski@gmail.com

Abstract— **With modern techniques that allow billions of transistors on a chip, microprocessor design is going to a world of multicore. A cluster of multicores will be commonly used as an efficient computational platform for high performance computing in the near future. Correspondingly, the resource providers, who share their computing elements and storage with customers, will also upgrade their machines to the multicore architecture.**

At the same time, the dominating resource sharing platforms, in the form of compute Grids, are facing a grand challenge from Cloud computing. With an easy interface, high-level services, and on-demand provision of compute resources, compute Clouds are attracting more users to join their realm.

Upcoming Cloud environments will be primarily built on top of multicore technologies. However, to fully exploit the computational capacity and the other advantages of multicores, a lot of novel techniques must be proposed and investigated.

This paper serves as an overview for multicore technologies in the context of Cloud computing. This paper firstly introduces multicore technologies and the Cloud computing concept. Then it investigates various open research issues in the context of multicore Cloud computing, for example, how to efficiently employ multicore techniques for Cloud computing, how to achieve the high performance of multicore for high performance applications in the Cloud environments. Several examples and discussion are presented.

Keywords: Multicore architecture, high performance computing, Cloud computing

I. Introduction

The rapid development of multicore processors is a milestone for computing systems. It affects compiler construction and programming language & application models. Cloud computing recently emerges as a new computing paradigm and could render desired computing infrastructure for users on demand. This paper overviews the research fields of multicore Cloud computing. It introduces the background of multicore systems and Cloud computing, identifies research issues and proposes possible solutions. The rest of this paper is organized as follows. Section II discusses the multicore architecture and system. Section III studies the concept of Cloud computing infrastructure. Multicore Cloud computing context, such as research issues, implementation and solutions are investigated in Section IV. Finally the paper concludes in Section V.

II. The Multicore Architecture

According to Moore's law, billions of transistors can be integrated on a single chip in the near future. However, due to the physical limitations of semiconductor-based electronics, the clock frequency of microprocessors will not increase exponentially like it did over the last thirty years. Additionally, heat dissipation is a fatal problem with modern micro architectures. Power consumption also becomes another limitation in regards to enhancing the CPU frequency.

In this case, multicore arises as an adequate solution. Different from the single-processor architecture, which deploys all transistors on the chip to form a single core, this novel design partitions the transistors into groups in which each group builds a separate processor core. As the communication of transistors is within the group, rather than globally across the complete chip, shorter latency and switching time can be maintained. In this case, each core can be granted a certain frequency and the multicore as a whole provides a high performance.

Actually, multicore processor is not only a solution for CPU speed but also reduces the power consumption because several cores in a lower frequency together generate less heat dissipation than one core with their total frequency. Therefore, multicore processor is regarded as the future generation of microprocessor design.

A. Multicore Products

Over the last few years almost every processor manufacturer has put multicores in the market. Active vendors include IBM, Intel, SUN, and AMD.

doi:10.4304/jcp.5.6.958-964

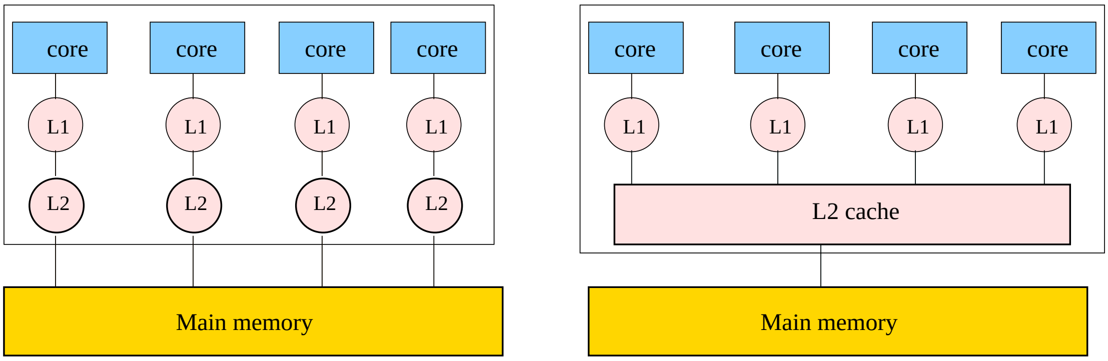

Figure 1. Multicore architecture with private (left) and shared L2 cache (right)

Together with Sony and Toshiba, IBM developed the Cell multiprocessor [15], [23] for the Play Station 3 console. It applied a heterogeneous architecture with a PowerPC as the main processor and additional eight synergistic processing elements for computation. The Cell processor can operate at both a low voltage and low power while maintaining high frequency and high performance. Besides Cell, IBM also developed other multicore architectures like the BladeCenter JS22 with two cores of Power6.

Intel [13] dual-core processors have been successfully integrated in desktop, notebooks, workstations, and servers. An example is Woodcrest with two Xeon processors. Recently, its quad-core technology is delivering outstanding performance and energy efficiency. In addition, Intel has announced a 64-core architecture for the future.

AMD is another active processor developer and most of the HPC platforms are built on top of the Intel and AMD products. For multicore technology, AMD [6] delivered the dual-core Opteron designs and currently the new quad-core Phenom II processor. The well-known AMD HyperTransport Technology provides a high communication bandwidth.

Following UltraSPARC T1, SUN [22] developed UltraSPARC T2 Plus Niagara with 8 cores. In addition to the large number of cores on the chip, this microprocessor has hardware support for 8 threads per core forming a maximal number of 64 logical CPUs.

Overall, multicore has become the trend of microprocessor design and therefore the basic hardware of an HPC platform. According to the statistics published in 2007, more than 60% of the Top500 supercomputers rely on mulicore technology. Hence, multicore technology will be the main components of emerging cloud infrastructures.

B. Multicore Architecture

A multicore processor is typically a single processor which contains several cores on a chip. The cores are fully functional with computation units and caches and hence support multithreading. This is different from the earlier hyper-threading techniques which provide additional sets of registers to allow multiple threads that usually used to prefetch instructions or data [5], [17]. It is true that researchers use idle cores for the same purpose [19], however, multicore is more adequate for parallel processing.

The dominant design of multicores uses a homogeneous configuration with IBM Cell as an exception. In this case, multicores can be observed as SMPs (symmetric multiprocessor) moving their processors from different boards to the same chip on a single main board. This means a shorter inter-processor communication time and a more efficient utilization of the on-chip memory, which are advantages for running multiple threads with data dependency.

Each microprocessor manufacture has its specific implementation of multicore technology. Nevertheless, products on the market mainly distinguish in the organization of the second level cache: one is private and the other is shared. Figure 1 shows both architectures with a four core processor as example.

As shown in the figure, a multicore architecture contains several CPU cores each equipped with a private first level cache. This is common with all multicore designs. Together with the core and the L1 cache, the chip also contains a second level (L2) cache or a few L2 caches, depending on the concrete implementation. The former can be typically found in the Intel multicore processors and the latter is adopted by AMD. Through a memory controller, the second level cache is combined with main memory. Additionally, some products have an off-chip L3 in front of the main memory.

Theoretically, a shared L2 cache performs better than private L2 caches because with a shared design, the working cores can use the caches of idle cores. In addition, cache coherence issues are not problematic with the shared cache.

Actually, we have proven this theory using our cache

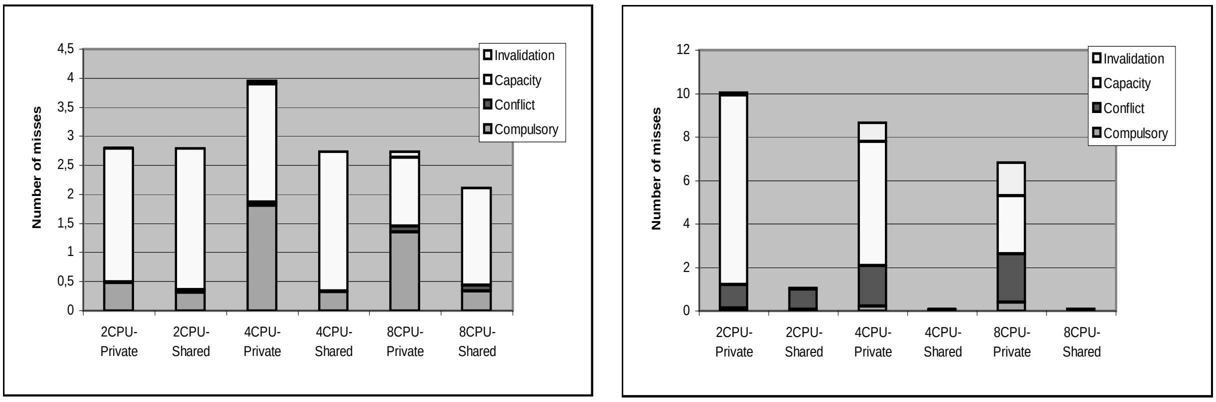

Figure 2. L2 misses of the FT (left) and CG (right) applications

simulator for multicore [10]. Figure 2 depicts sample results we acquired with the simulator which models the cache hierarchy of a multicore architecture.

The figure demonstrates the number of the second level cache with both private and shared organization, with the left side for FT and the right side for CG, two applications from the NAS [9], [14] OpenMP benchmark suite.

For a better understanding of the cache behavior, the cache miss is shown in a break-down of four miss categories: compulsory miss, capacity miss, conflict miss, and invalidation miss. As can be seen with both applications, the shared cache performs generally better than the private caches. The performance gain with FT is mainly contributed by the reduction of compulsory misses while CG shows a significant decrease in the number of capacity misses with a shared L2 cache.

Different applications have their own reason for the better performance, however, several common factors exist. For example, parallel threads executing a program partially share the working set. This shared data can be reused by other processors after being loaded to the cache, thus reducing the compulsory caused by the first reference of a data item. Shared data also saves space because the memory block is only cached once. This helps to reduce the capacity miss. In addition, shared cache has no consistency problem and hence there are no invalidation misses with a shared cache.

C. Programming Models

Intel is planning to integrate 64 cores on a chip. Actually, this is a trend with multicore technology: the number of CPU cores on a chip continues to increase. A question arises: how to parallelize programs in order to fully utilize the computational capacity?

MPI and OpenMP are dominating programming models for parallelizing sequential programs. The former is a communication-based paradigm that requires programmers to use explicit send/receive primitives to transfer data across the processes. This model is typically applied on loosely coupled architectures like clusters of commodity PCs.

Alternatively, OpenMP is a programming model for parallelization on multiprocessor systems with a shared memory. It is gaining broader application in both research and industry. From high performance computing to consumer electronics, OpenMP has been established as an appropriate thread-level programming paradigm. OpenMP provides implicit thread communication over the shared memory. Another feature of OpenMP is that it maintains the semantic similarity between the sequential and parallel versions of a program. This enables an easy development of parallel applications without the requirement of specific knowledge in parallel programming. Due to theses features, OpenMP is regarded as a suitable model for multicores.

However, OpenMP was proposed at the time when SMP was a dominating shared memory architecture. It uses the traditional data and function parallelism to present the concurrency in a program. Hence, it is adequate for coarse-grained parallelization. Even though task parallelism has been introduced in the OpenMP standard, it is still hard to generate sufficient parallel tasks for keeping all cores busy on the future multicores.

A novel programming model has to be designed for multicore, however this research work is in the beginning stages. For example, Ostadzadeh et al. [20] provides an approach that observes parallelization in a global way rather than just parallelizing a code area like the traditional model does. Hence, this approach is capable of detecting more concurrency in a program.

D. Multicore Clusters

Clusters of commodity PCs/workstations are generally adopted to perform scientific computing. With the trends of microprocessor architecture towards multicore, future clusters will be composed of multiple nodes of multiprocessors with shared memory. This kind of cluster is similar to the current SMP cluster, however, the system scale will be much larger, with potentially several thousands and even ten thousands cores in a single cluster.

One question is how to program such clusters with a shared memory on the node and distributed memories on the system. For SMP clusters, most deployed applications use the MPI model, i.e. message passing across all processors including those within one node. The advantage of this approach is that applications need not be rewritten, however, the explicit communication between processors on the same node introduces overhead. This overhead could be small when the node only contains two or four processors, like it is the case with an SMP node, nevertheless, for a multicore node it can significantly influence the overall performance.

Another approach [16] is to program an SMP cluster is using the OpenMP model with a shared virtual memory for the cluster. A well-known commercial product is the Intel compiler [12] which runs an OpenMP program on cluster architectures by establishing a global memory space for shared variables. The advantage of this approach is that OpenMP applications can be executed on a distributed system without the need of any adaptation and the parallel threads running on the same node communicate via the local memory. However, the management of the virtual shared space again introduces overhead which can be large on a cluster with thousands of processor nodes.

A hybrid approach seems to be the best solution because the application is parallelized with OpenMP on the node and with MPI on the system, causing a hybrid communication form of intra-node with shared memory and inter-node with message passing. However, the existing implementations require that the users apply both MPI primitives and OpenMP directives in the program to give the compiler and the runtime operation guides. This is clearly a burden to the programmers. Hence, a better way is to only apply the OpenMP model, which is easier to learn than MPI, and to build a compiler capable of transforming a shared memory communication to MPI send/receive in case that the communication is beyond a processor node.

Another problem with multicore clusters is scalability. Most applications do not initially run efficiently on large clusters and hence need performance tuning. The existing tools for performance analysis, visualization, and steering have helped programmers to improve their codes and can also be further applied on multicore clusters. However, most of the tools must be extended to grant a good behavior with larger systems.

III. Study on Computing Cloud Infrastructure

Cloud computing is becoming one of the next IT industry buzz words: users move data and applications out to the remote "Clouds" and then access them in a simple and pervasive way. This is again a central processing use case. Similar scenario occurred around 50 years ago: a time-sharing computing server served multiple users. Until 20 years ago, when personal computers came to us, data and programs were mostly located in local resources. Certainly the Cloud computing paradigm is not a recurrence of the history. 50 years ago we had to adopt the time-sharing servers due to limited computing resources. Nowadays, Cloud computing comes into fashion due to the need to build complex IT infrastructures. Users have to manage various software installations, configurations, and updates. Computing resources, and other hardware, are prone to be outdated very soon. Therefore, outsourcing computing platforms is a smart solution for users to handle complex IT infrastructures.

At the current stage, Cloud computing is still evolving and there exists no widely accepted definition. Based on our experience, we propose an early definition of Cloud computing as follows [25]:

> *A computing Cloud is a set of network enabled services, providing scalable, QoS guaranteed, normally personalized, inexpensive computing infrastructures on demand, which could be accessed in a simple and pervasive way.*

In the initial step of the study, we focus on how to build infrastructures for Scientific Clouds. The Cumulus project [24] is an on-going Cloud computing project, which intends to provide virtualized computing infrastructures for scientific and engineering applications. The Cumulus project currently is running on distributed IBM blade servers with Oracle Linux and the Xen hypervisors. We design the Cumulus system in a layered architecture (see also Figure 3):

- The Cumulus frontend service resides on the access point of a computing center and accepts users' requirements of virtual machine operations. The Cumulus frontend service is implemented by re-engineering of Globus virtual workspace service and Nimbus frontend [2].
- The OpenNEbula [1] works as Local Virtualization Management System (LVMS) on the Cumulus backends. The OpenNEbula communicates with the Cumulus frontend via SSH or XML-RPC.
- The OpenNEbula contains one frontend and multiple backends. The OpenNEbula frontend communicates to its backends and the Xen hypervisors on the backends via SSH for virtual machine manipulation.

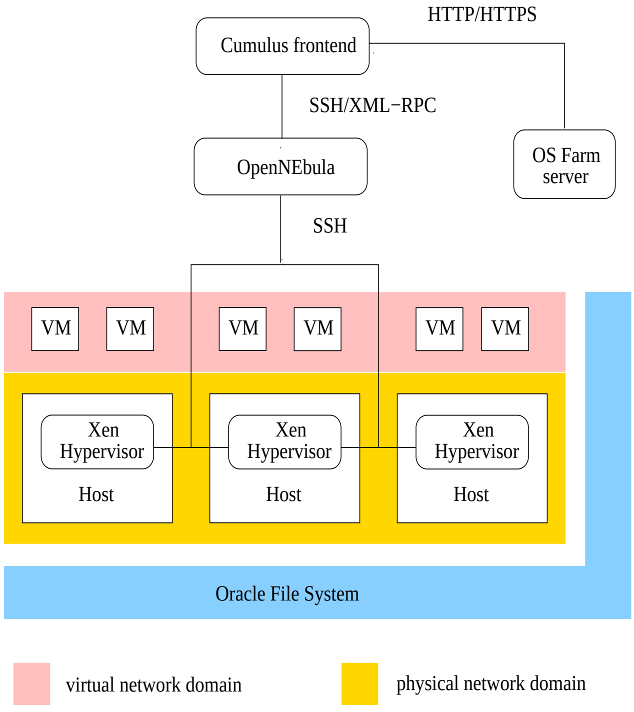

Figure 3. Overview of Cumulus project

- OS Farm
  The OS Farm [3] is a Web server for generating and storing Xen virtual machine images and Virtual Appliances. The OS Farm is used in the Cumulus system as a tool for virtual machine template management.

IV. Multicore in the Cloud Computing Context

A. Multicore meets Clouds

Computing Clouds could definitely benefit from the rapid growth of multicore systems. For example, Google data centers can adopt uniform commodity multicore clusters as computing servers. Computing Grids are normally suitable for "coarse grained" parallel applications. In general, the latencies in Grid applications are measured in macroseconds [11]. Computing Clouds normally contain several homogeneous data centers or computer centers, where commodity high performance multicore clusters are employed. Therefore, "fine grained" parallel applications could be mapped computing Clouds, which could provide low latencies between clusters and high data transfer rate of multicore systems.

B. Computing Infrastructure Provision from Multicore Cloud

We could expect the following Cloud computing context:

- a computing Cloud is composed by several data centers or compute centers, and
- each center contains multiple identical commodity multicore clusters.

Computing Cloud users can then lease computing infrastructures and platforms from the computing Clouds, which normally are virtual machine based system. The consolidation of a set of virtual machines require to efficient and scheduling and management on multicore systems, such as:

- How to schedule a virtual machine across multiple cores, multi-processors, or even multi-clusters?
- How to move a virtual machine when slots of cores are available?

These open questions call some solutions and work on efficient power management on multicore clusters and clouds.

C. Programming requirements of Multicore Cloud computing

From the user side, proper programming models and paradigms should be developed to fulfill the requirements of multicore based computing Clouds:

- Task level atomicity
  Normally, computing resources are provided to users in the form of virtual machines, and one task is allocated one or more virtual machines. The basic unit of resource provision and management is virtual machine. To adapt to this, user programs should be atomic in the unit of task, which is clear defined with its purpose, operations, and control.
  Task level atomicity can bring several advantages, for example, fault tolerance in commodity computing clusters and Clouds, easy management (such as checkpointing and migration), robust control from user level and system level.
- Stateless task
  In the Grid computing community, especial the Globus world, people would like to promote stateful services, which keep the information or context between calls on the services. However, we discover this style in the multicore Clouds:
  - Stateless tasks could be easily managed, e.g., checkpointing, task migration, both in the user level and system level.
  - Stateless tasks therefore can bring scalability and fault tolerance in homogeneous multicore cluster environments, in a Cloud.

D. Where goes the parallelism?

It is noted that multicore based system could not be automatically exploited if users don't apply explicit parallel computing models and language in their applications [7], [18]:

- User or system level software should know that their underlying platform contains multicores and try to map their multiple tasks or threads to multicores.
- The multicore system or Cloud middleware should provide some development environments or enabling compilers to make applications multicore enabled.

As we discussed above, multi-task programming and multi-thread computing are suitable for multicore Clouds. This could be justified by the most popular programming model – MapReduce [8]. MapReduce is a programming model and framework for processing large data sets across distributed data centers. The MapReduce framework inputs a set of input key/value pairs and produces a set of output key/value pairs. There are two functions, Map and Reduce, in the programming model. The Map function accepts an input pair and generates a set of intermediate key/value pairs. The MapReduce library groups together all intermediate values associated with the same intermediate key and passes them to the Reduce function. The Reduce function takes an intermediate key and a set of values for that key, merges together these values to form a smaller set of values.

Phoenix [21] is an implementation of MapReduce from Stanford University for shared memory systems, e.g., multicore processors. It uses threads instead of cluster nodes for parallelism. The communicates through shared memory instead of network messages. Current version of Phoenix provides C/C++ APIs and uses P-threads. The Phoenix orchestrates application execution across multiple threads:

- initiates and terminates threads,
- assigns map & reduce tasks to threads, and
- handles buffer allocation and communication.

V. CONCLUSION

Cloud computing and mulicore technologies are listed in 10 “disruptive technologies” to reshape the IT landscape between 2008 and 2012 [4]. To prepare for the multicore era and Cloud computing stage, various research issues should be identified and studied. This paper overviews the research context of multicore Cloud computing. It introduces the background of multicore system and cloud computing concept, investigates research problems and discusses possible solutions for the study.

REFERENCES

[1] Opennebula project.
[2] Nimbus Project, access on June 2008.
[3] Os farm project, access on July 2008.
[4] Gartner identifies top ten disruptive technologies for 2008 to 2012, access on Jan. 2009.
[5] T.M. Aamodt, P. Chow, P. Hammarlund, H. Wang, and J. P. Shen. Hardware Support for Prescient Instruction Prefetch. In *Proceedings of the 10th International Symposium on High Performance Computer Architecture*, pages 84–95, 2004.
[6] AMD. AMD Multi-Core Technology. http://multicore.amd.com/.
[7] K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams, and K. A. Yelick. The landscape of parallel computing research: a view from berkeley. Technical report, Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, 2006.
[8] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, 2008.
[9] D. Bailey et. al. The NAS Parallel Benchmarks. Technical Report RNR-94-007, Department of Mathematics and Computer Science, Emory University, March 1994.
[10] J. Tao et al. Performance Advantage of Reconfigurable Cache Design on Multicore Processor Systems. *International Journal of Parallel Programming*, 36(3):347–360, June 2008.
[11] Geoffrey Fox and Marlon Pierce. Grids challenged by a web 2.0 and multicore sandwich. *Concurrency and Computation: Practice and Experience*, 21(3):265–280, 2009.
[12] Intel Corporation. Intel C++ Compiler 11.0 Professional Edition for Linux. http://www.intel.com/cd/software/products/asmo-na/eng/compilers/clin/277618.htm.
[13] Intel Corporation. Intel Multi-Core Technology. http://www.intel.com/multi-core/.
[14] H. Jin, M. Frumkin, and J. Yan. The OpenMP Implementation of NAS Parallel Benchmarks and Its Performance. Technical Report NAS-99-011, NASA Ames Research Center, October 1999.
[15] J.A. Kahle, M.N. Day, H.P. Hofstee, C.R. Johns, T.R. Maeurer, and D. Shippy. Introduction to the Cell Multiprocessor. *IBM Research & Development*, 49(4/5):589–605, 2005.
[16] Y. Kee, J. Kim, and S. Ha. ParADE: An OpenMP Programming Environment for SMP Cluster Systems. In *Proceedings of SC 2003*, 2003.
[17] J. Lee, C. Jung, D. Lim, and Y. Solihin. Prefetching with Helper Threads for Loosely-Coupled Multiprocessor Systems. *IEEE Transactions on Parallel and Distributed Systems*, 99(1), 2008.
[18] C. E. Leiserson and I. B. Mirman. How to survive the multicore software revolution. Technical report, Cilk Arts, Inc, 2008.
[19] J. Lu, A. Das, W. Hsu, K. Nguyen, and S.G. Abraham. Dynamic Helper Threaded Prefetching on the Sun UltraSPARC CMP Processor. In *Proceedings of the 38th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 93–104, 2005.
[20] S. A. Ostadzadeh, R. J. Meeuws, K Sigdel, and K.L.M. Bertels. A Multipurpose Clustering Algorithm for Task Partitioning in Multicore Reconfigurable Systems. In *Proceedings of IEEE International Workshop on Multi-Core Computing Systems*, page to appear, March 2009.
[21] C. Ranger, R. Raghuraman, A. Penmetsa, G. R. Bradski, and C. Kozyrakis. Evaluating mapreduce for multi-core

and multiprocessor systems. In *Proceedings of 13st International Conference on High Performance Computer Architecture*, pages 13–24, 2007.

[22] SUN Microsystems. UltraSPARC T2 Processor. http://www.sun.com/processors/UltraSPARC-T2/.

[23] O. Takahashi, S. Cottier, S.H. Dhong, B. Flachs, and J. Silberman. Power-Conscious Design of the Cell Processor's Synergistic Processor Element. *IEEE Micro*, 25(5):10–18, 2005.

[24] Lizhe Wang, Jie Tao, Marcel Kunze, A.C. Castellanos, David Kramer, and Wolfgang Karl. Scientific cloud computing: Early definition and experience. In *Proceedings of the 10th IEEE International Conference on High Performance Computing and Communications*, pages 825–830, September 2008.

[25] Lizhe Wang, Gregor von Laszewski, Jie Tao, and Marcel Kunze. Cloud computing: a perspective study. *accepted by New Generation Computing*, 38(5):63–69, accepted.