Electronic Structure Calculations and Adaptation Scheme in Multi-core Computing Environments

Lakshminarasimhan Seshagiri[1], Masha Sosonkina[1], and Zhao Zhang[2]

[1] Ames Laboratory
Iowa State University
Ames, IA 50011 USA
{sln,masha}@scl.ameslab.gov
[2] Department of Electrical and Computer Engineering
Iowa State University
Ames, IA 50011 USA
zzhang@iastate.edu

Abstract. Multi-core processing environments have become the norm in the generic computing environment and are being considered for adding an extra dimension to the execution of any application. The T2 Niagara processor is a very unique environment where it consists of eight cores having a capability of running eight threads simultaneously in each of the cores. Applications like General Atomic and Molecular Electronic Structure (GAMESS), used for ab-initio molecular quantum chemistry calculations, can be good indicators of the performance of such machines and would be a guideline for both hardware designers and application programmers. In this paper we try to benchmark the GAMESS performance on a T2 Niagara processor for a couple of molecules. We also show the suitability of using a middleware based adaptation algorithm on GAMESS on such a multi-core environment.

Keywords: Multi-Core, GAMESS, Niagara, Adaptation, NICAN.

1 Introduction

Computational chemistry applications like GAMESS [5] are widely used to perform ab-initio molecular quantum chemistry calculations. These calculations include a wide range of Hartree-Fock (HF) wave function (RHF,ROHF and UHF) calculations. Such calculations are not only complex but also have high computational requirements. These calculations are currently run on SMP clusters where each node consists of single or dual core processors. SMPs can be viewed as a form of NUMA (Non Uniform Memory Access) architecture [4]. NUMA is the design used where each processor in a multi processor environment is provided with a separate memory space and data is being shared between different memory banks. This needs to be handled using separate hardware and software since the data can be distributed over different processor memories and coherency

G. Allen et al. (Eds.): ICCS 2009, Part I, LNCS 5544, pp. 3–12, 2009.

between this data needs to be maintained. The communication cost plays a significant role in this design. Each node in a computing cluster can be considered equivalent to a processor in the NUMA architecture, but with a coupling that is not as tight as that in NUMA. The inter-node latency and bandwidth in clusters is much worse than a normal NUMA machine. Hence when we run an IO intensive application like the conventional GAMESS job, it is very likely to bog down the channel thereby resulting in slower execution times. The problem of remote data access on symmetric multiprocessor (SMP) nodes is avoided by the usage of a native communication layer DDI (Distributed Data Interface) that utilizes the shared memory effectively [13].

A multi-core processor capable of running multiple threads in each core can be used as a single execution environment in itself instead of a SMP cluster. The execution semantics change in such an environment. Each of the threads act as a virtual processor (VP) to the outside world. Thus the user application sees itself running on a multi-processor machine with access to each and every one of them. Each of these VPs include all the architecturally required components to execute a task. These components include registers (both general purpose and special), integer and floating point execution units and can handle interrupts. The execution units are present inside each core of the processor and the VPs belonging to each core share these components. Thus each VP contains a separate instance of the user state. Since the multi-core processor is fabricated on a single chip, the resources such as memory bandwidth, L1 and L2 caches are shared among the VPs. This has a significant impact on the performance of the task being executed.

There have been studies using benchmarking tools to show the performance of SMP clusters which use single core or dual core processors such as the Intel Woodcrest processor [1]. There have also been similar studies on the Niagara processor [1] that extol the advantages of using a multi-core and a multi-threaded processor. An understanding on the performance of an application like GAMESS on a SMP and a multi-core environment like the Niagara processor will be of immense help not only for application programmers but also from processor designers. We compare the performance of GAMESS on these two environments in order to understand the relative advantages and disadvantages of the two. Also, of note is the fact that resource sharing in a multi-core processor running multiple VPs would have a great impact on the performance of a computationally intensive application like GAMESS. Various studies [10], [9], [7] have been done to arrive at the best possible combination of GAMESS processes per node (on a SMP) so as to overcome the resource constraints. It has already been shown in [8] that an adaptation algorithm using a generic middleware tool NICAN [2,6] would improve the performance of GAMESS. We show that this adaptation scheme is of relevance and importance in a multi-core and multi-threaded environment as well.

The rest of the paper is organized as follows. Section 2 describes the workload used and the architecture of the execution environment. Section 3 describes the performance of GAMESS on a SMP cluster and a Niagara processor. Section 4 deals with the adaptation algorithm and the results obtained by using this algorithm on a Niagara processor.

2 Methodology

2.1 Application Workload

General Atomic and Molecular Electronic Structure (GAMESS) performs ab-initio molecular quantum chemistry calculations [5] to perform a wide range of Hartee-Fock (HF) wave function (RHF, ROHF and UHF) calculations. Using Self Consistent Field (SCF) method, GAMESS iteratively approximates solution to the Schrödinger equation that describes the basic structure of atoms and molecules. Numerous GAMESS calculations have parallel implementations utilizing distributed resources like memory and disk storage. The scalability of GAMESS is aided by the use of a native communication layer DDI [13] that takes advantage of shared memory on symmetric multiprocessors (SMP) and reduces the remote data access bottleneck. The SCF method is one of the most computationally intensive parts in the GAMESS execution. It has two implementations, *direct* and *conventional*, which differ from each other in the handling of the two-electron (*2-e*) integrals.

In the *direct* SCF method, the 2-e integrals are recalculated for each iteration and it avoids any I/O bottleneck. In the *conventional* SCF method, the 2-e integrals are calculated once at the beginning of the SCF process and stored in a file on disk for subsequent iterations. These two implementations are interchangeable [8] due to the iterative nature of the process. SCF method also gives a good indication of the processor computation power as well the I/O capabilities of the system on which GAMESS is being run. Thus a GAMESS run on a SMP and a Niagara processor can be favorably compared to get some sort of an opinion on the relative merits of using either of these two architectures. We chose *Luciferin* and *Ergosterol* molecules for testing GAMESS on these two platforms. GAMESS converges in 15 iterations for both *Luciferin* and *Ergosterol*. A *conventional* execution of *Luciferin* requires a storage of almost 3.5GB of files while a *direct* execution consumes 5.65MB of main memory. On the other hand, a *conventional* execution of *Ergosterol* molecule stores 22GB of files and requires nearly 16MB of main memory for the *direct* implementation.

2.2 Architectures Used

We used two different architectures to test GAMESS. One was a SMP cluster of 4 nodes, each node having two dual-core 2.0GHZ Xeon "Woodcrest" CPUs and 8GB of RAM [1]. The nodes were interconnected with both Gigabit Ethernet and DDR Infiniband. Each processor has a shared 4MB L2 cache. It also contains a 32KB L1 instruction and data cache per core.

The second architecture used for testing was the Sun T2 Niagara processor (T2) [11,14]. The T2 processor has a unique architecture which consists of 8 SPARC physical processor cores built in a single chip and each core is capable of running 8 threads. Each of these threads can be considered to be a processor in itself and are called as Virtual Processors (VP). Thus a user application sees itself running on a machine of 64 processors rather than on a processor containing 8 cores. The VPs operate at a frequency of 1167 MHz. Each of these cores

contain full hardware support for the eight VPs. There are two integer execution pipelines, one floating point pipeline and one memory pipeline inside a single core that are shared between all the VPs. The eight VPs are divided into two groups of four each with the VPs 0-3 occupying one group and 4-7 occupying the other group. Obviously, the hardware support inside a single core also gets divided accordingly with each group of VP having access to a single integer pipeline and sharing the floating point and memory pipelines. Each SPARC physical core contains a 16 KB, 8-way associative instruction cache (32-byte lines), 8 KB, 4-way associative data cache (16-byte lines), 64-entry fully-associative instruction TLB, and 128-entry fully associative data TLB that are shared by the eight VPs. The eight SPARC physical cores are connected through a crossbar to an on-chip unified 4 MB, 16-way associative L2 cache (64-byte lines) which is banked eight ways to provide sufficient bandwidth for the eight SPARC physical cores.

3 Performance Results

3.1 Benchmark Performance for GAMESS on T2 Niagara Processor

We first benchmark *Luciferin* and *Ergosterol* molecules on a Niagara processor by running single jobs on different sets of VP combinations. We create VP sets such that the processes that are run on these VPs have access to as much hardware as possible for speedier execution. For example, if we need to create a set of 8 VPs, we distribute the VPs among all the 8 cores. We then bind the GAMESS processes to the VP sets that have been created so as to take advantage of the processor affinity property. Processor affinity [15] exploits the fact that some remnants of the process's state may remain in the processor's cache. The benchmarking results have been shown in Figures 1 and 2.

From the results we can deduce three different trends clearly. In case of the *direct* execution of *Luciferin* and *Ergosterol*, increase in the number of VPs used for execution results in better performance. The increase in hardware resources and the thread level parallelism help speed up the computations. This trend is not followed in case of the *conventional* execution of *Luciferin* and *Ergosterol*. The execution time of *conventional Luciferin* reduces initially untill about 32 VPs and then steadily increases as we move from 40VPs to 63VPs. *Conventional Ergosterol* degrades in performance as we increase the number of VPs. A *conventional* GAMESS job can be characterized into two parts. The first part is writing of the integral files and the second is RHF SCF calculation using the integral values that are stored in the disk files. The application fetches the integral values from the files and then performs the RHF SCF calculations iteratively. In case of *Ergosterol*, the integral files (size 22GB) cannot fit in the main memory of the Niagara processor (16GB). This gives rise to a large number of page faults and cache misses thus leading to a drop in performance. Another contributor to the slow execution time for the *conventional Ergosterol* molecule could be the issue with parallel reads and writes at higher thread counts that affect GAMESS [16] performance. This explains the degradation in the performance in a *conventional Luciferin* molecule once the number of VPs are increased beyond 32.

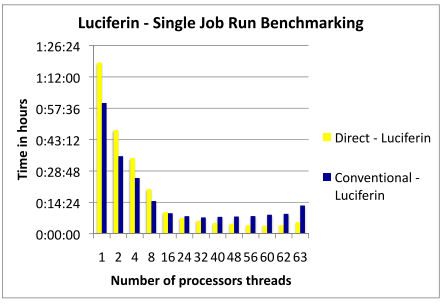

Fig. 1. *Luciferin*: Single job benchmark

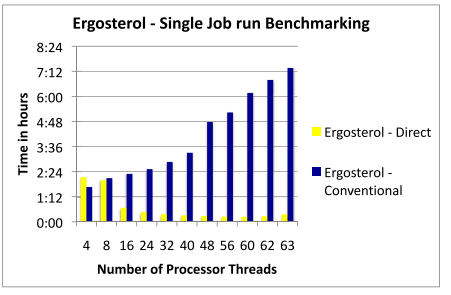

Fig. 2. *Ergosterol*: Single job benchmark

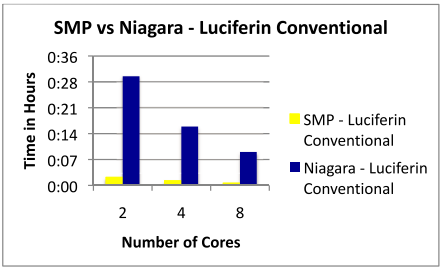

Fig. 3. *Luciferin*: *Conventional* job SMP Vs Niagara comparison

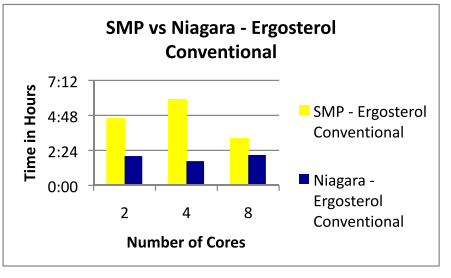

Fig. 4. *Ergosterol*: *Conventional* job SMP Vs Niagara comparison

One more thing to note is that the kernel itself is run on one of the 64 VPs. When we create processor sets, we cannot assign all the 64 threads to different processor sets since at least a single VP is required for running the kernel. In such a scenario, if we assign 63 VPs to execute a single GAMESS job, the job takes more time than when run with 60 VPs. Also, there is a steady increase in the execution time as we move from 60 VPs to 62 and 63 VPs as seen in the *direct* execution of *Luciferin* and *Ergosterol* in Figure 1 and 2. As the hardware resources used by the kernel are shared with the GAMESS threads, we can see a performance degradation. Hence 60 VPs would be an optimal number to be used for application usage and 4 VPs for the system usage.

3.2 Performance Comparison between T2 Niagara Processor and a SMP 8-Core Cluster

We compared the performance of GAMESS on a T2 Niagara processor with its performance on a SMP cluster. The SMP cluster contains 4 nodes, each containing two dual core Intel Xeon "Woodcrest" processors. For the sake of comparison with the Niagara processor, we used only two of the nodes on this cluster to run GAMESS (since this would be equivalent of running GAMESS

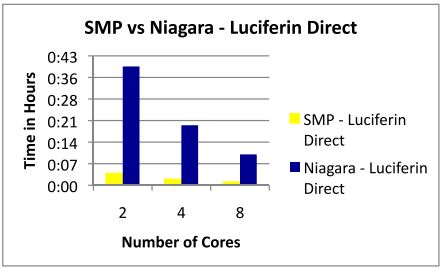

Fig. 5. *Luciferin*: *Direct* job SMP Vs Niagara comparison

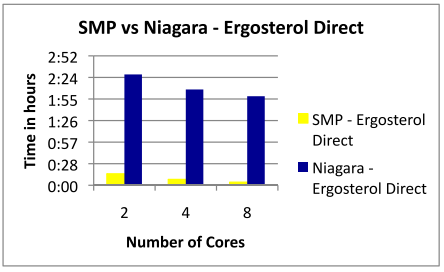

Fig. 6. *Ergosterol*: *Direct* job SMP Vs Niagara comparison

on 8 cores). The performance results for *conventional* and *direct* executions of *Luciferin* and *Ergosterol* are given in Figures 3, 4, 5 and 6.

We can see that the SMP out performs the Niagara Processor for the *Luciferin* molecule while it is worse than the Niagara performance in case of the *conventional* mode execution of *Ergosterol* molecule. We need to note that each core in an Intel processor can run only a single thread while each Niagara core can run up to eight simultaneous threads. The cache available to each core on an Intel processor chip is more than the cache available in a Niagara processor as the cache is distributed amongst all the cores. Each core on an Intel processor runs at a higher clock rate as compared to the clock rate of a VP in the Niagara processor. These clearly help to understand the reasons for the higher performance in case of the *Luciferin* molecule calculations and the *direct* calculations for the *Ergosterol* molecule.

However, the true advantage of using a Niagara processor can be seen when we run a *conventional Ergosterol* calculation. The main reason for this is the usage of a dedicated floating point pipeline in each of the 8 cores in a Niagara processor. The GAMESS calculations are inherently floating point in nature and dedicated floating point pipelines help to improve performance by nearly 50 percent. As indicated by the performance results, execution of a *conventional Ergosterol* calculation on the T2 processor is very time consuming, though it performs better than the SMP. The performance degrades as we increase the number of VPs on which GAMESS is run. This scenario can be readily exploited to improve the performance by utilizing the adaptation algorithm first introduced in [8]. The next section explains the suitability of this algorithm in a multi-core and multi-threaded scenario.

4 Adaptations in GAMESS Using NICAN Middleware

The SCF algorithm is one of the most computationally intensive parts in the GAMESS execution. Selection of the correct electronic structure calculation routine has a very big effect on the overall calculation and calculation time. The iterative nature of the SCF algorithm allows us to switch between the *conventional*

and *direct* implementations in an arbitrary SCF iteration. The switching is carried out using the middleware tool NICAN in order to decouple the application from having to make any adaptation decisions during the application execution. The application is responsible only for the invocation of the adaptation handlers. The adaptations are handled by a control port that is part of the NICAN tool.

The adaptation scheme used in [8] for SMPs is summarized ahead. The adaptation scheme consists of a static and a dynamic part. Every *conventional* GAMESS job gets modified to a *direct* execution mode if there is a "peer" *conventional* GAMESS job already running in the system. It was shown in [9] that while running concurrent scattered GAMESS jobs, a single conventional job helps to achieve better performance. This constitutes the static adaptation method. The dynamic adaptation is used during the iterative SCF calculations. The control port gathers system and application information that allows it to decide on the adaptation at runtime using the algorithm given below.

t_N = Actual time taken for iteration N
t^u = Upper bound for the time per iteration (taken as a arbitrary large value)
m = Average iteration time over N iterations
t_0^e = Estimated ideal run time for running a single iteration (obtained by NICAN after running a GAMESS check run at startup)
$\Delta t_0 = \mid t_0^e$ - $t_0 \mid$
if ($t_i > t^u$ OR $t_i > m + \Delta t_0$) **then**
  if (SCF is *conventional*) **then**
    switch to *direct*
  else if ((no peer *conventional* jobs) AND (enough memory)) **then**
    switch to *conventional*
  end if
end if

The experimental results obtained for this algorithm on a SMP have been given in [8]. It has been shown on a two processor system with I/O congestion, that the performance of dynamically adaptive GAMESS is nearly the same as a "no-congestion" case. If the I/O bandwidth is fully consumed, then the adaptation scheme gives two times improvement in the execution time of GAMESS. Also, on running two simultaneous parallel GAMESS jobs on two and four processors, a gain of 10-15 percent in the cumulative execution time is obtained through a dynamic adaptation scheme. This dynamic adaptation algorithm holds true for a multi-core and multi-threaded environment as well. As seen in the benchmarking results of Section 3, the degradation in the performance of a *conventional Ergosterol* molecule calculation at higher values of VPs is a good starting point to apply the above adaptation algorithm.

4.1 Adaptation Results on T2 Niagara Processor

The adaptation scheme was tested by running simultaneous parallel GAMESS jobs on the T2 Niagara processor. The physical cores were partitioned equally

among the jobs thus ensuring that the hardware resources of a core is used exclusively by the GAMESS job assigned to the VP belonging to that core. The performance was measured by executing two parallel GAMESS jobs that consisted of one *Luciferin* molecule and one *Ergosterol* molecule. Similar results were obtained for three parallel GAMESS job execution which have two *Luciferin* Molecules and one *Ergosterol* molecule. If the SCF method is not defined in the GAMESS input file, then GAMESS selects the SCF mode to be *conventional* by default. Hence both the above tests were performed using *conventional* SCF mode at the start. The performance graphs have been given in Figures 7 and 8. We have not distinguished between static and dynamic adaptation in the results.

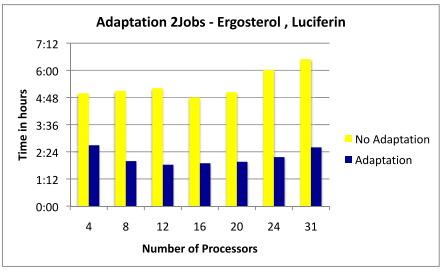

Fig. 7. Two simultaneous parallel jobs execution

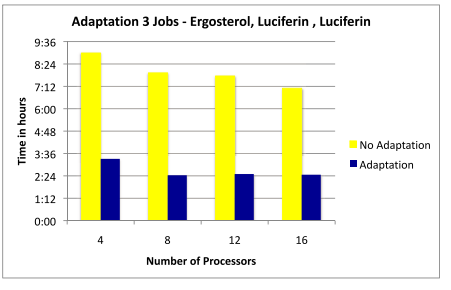

Fig. 8. Three simultaneous parallel jobs execution

We compare the performance of a non-adaptive GAMESS job ($GAMESS_{ORIG}$) and dynamically adaptive GAMESS ($GAMESS_{ADP}$) obtained using the NICAN middleware tool. We can see in the graphs that cumulative running time for $GAMESS_{ADP}$ is about 50 percent faster than $GAMESS_{ORIG}$. For two simultaneous GAMESS job execution, the adaptation gives a steady gain irrespective of the number of VPs used for running the simultaneous jobs. It was observed that at lower VP allocation values, the smaller molecule (*Luciferin*) is transformed into a *direct* method of execution due to the presence of a peer *Ergosterol* molecule but then switches back to *conventional* mode dynamically to ensure a faster run time. For larger VP allocations, both *Luciferin* and *Ergosterol* adapt and complete their execution in the *direct* mode. Similarly, in case of three simultaneous GAMESS jobs, we see that the cumulative run time of $GAMESS_{ORIG}$ is reduced by more than 50 percent by using the adaptation algorithm. All the three jobs complete their execution in the *direct* mode.

5 Conclusions and Future Work

The main focus of this work is to compare the performance of Electronic Structure calculations on a SMP with the performance on a T2 Niagara Processor. We have seen that SCF calculations for small molecules like *Luciferin* perform

much better on a SMP than on the Niagara processor. This trend can be seen for the *direct* SCF calculation for an *Ergosterol* molecule as well. However, the Niagara processor provides much better performance as compared to a SMP when we consider a *conventional* execution of the *Ergosterol* molecule. The T2 Niagara processor looks to be a good processing environment for such an execution scenario.

We also demonstrated that by using the adaptation algorithm introduced in [8], we can obtain performance improvement in GAMESS on a multi-core and multi-threaded environment. We have shown that the execution of adaptive GAMESS can be several magnitudes faster than the non-adaptive GAMESS execution. On a multithreaded processor like the Niagara, the I/O becomes a bottleneck as we start increasing the number of threads for a *conventional* mode of execution. In such cases, it was observed that the *direct* mode is the best way of execution. The performance difference between the *conventional* and *direct* mode at higher allocations of VPs is essential in getting the adaptation to work on such processors.

As a future work, it would be interesting to see how the application adaptability behaves when we use a cluster of such multithreaded processors. We would like to develop multiple adaptation control strategies for usage on such processors. These could include strategies such as changing the thread allocation dynamically from a single core to span multiple cores and to take advantage of processor affinity. This requires further research into the performance of GAMESS at different VP allocation configurations, along with the cache and IO performance of the Niagara processor for GAMESS. It would also be interesting to examine how any other cluster application would behave on a Niagara processor. Since the GAMESS version used for testing uses TCP/IP as an underlying communication framework, suitability of using MPI applications can be explored and documented. Ultimately, the aim is to develop generic adaptation control strategies that can be reused with any parallel application and are scalable for a multi-core and multi-threaded environment.

References

1. Terboven, C., an Mey, D., Sarholz, S.: OpenMP on Multicore architectures. In: Chapman, B., Zheng, W., Gao, G.R., Sato, M., Ayguadé, E., Wang, D. (eds.) IWOMP 2007. LNCS, vol. 4935, pp. 54–64. Springer, Heidelberg (2008)
2. Kulkarni, D., Sosonkina, M.: A framework for integrating network information into distributed iterativesolution of sparse linear systems. In: Palma, J.M.L.M., Sousa, A.A., Dongarra, J., Hernández, V. (eds.) VECPAR 2002. LNCS, vol. 2565, pp. 436–450. Springer, Heidelberg (2003)
3. White, E.H., Capra, F., McElroy, W.D.: The Structure and Synthesis of Firefly Luciferin. J. Am. Chem. Soc. 83(10), 2402–2403 (1961)
4. Hennessy, J.L., Patterson, D.A., Arpaci-Dusseau, A.C., et al.: Computer architecture: A quantitative approach, 4th edn. Morgan Kaufmann, San Francisco (2006)

5. Schmidt, M.W., Baldridge, K.K., Boatz, J.A., Elbert, S.T., Gordon, M.S., Jensen, J.H., Koseki, S., Matsunaga, N., Nguyen, K.A., Su, S., Windus, T.L., Dupuis, M., Montgomery, J.A.: General Atomic and Molecular Electronic Structure System. Journal of Computational Chemistry 14, 1347–1363 (1993)
6. Sosonkina, M.: Adapting Distributed Scientific Applications to Run-time Network Conditions. In: Dongarra, J., Madsen, K., Waśniewski, J. (eds.) PARA 2004. LNCS, vol. 3732, pp. 747–755. Springer, Heidelberg (2006)
7. Sosonkina, M., Storie, S.: Parallel performance of an iterative method in cluster environments: an experimental study. In: Proceedings PMAA 2004 (October 2004)
8. Ustemirov, N., Sosonkina, M., Gordon, M.S., Schmidt, M.W.: Dynamic Algorithm Selection in Parallel GAMESS Calculations. In: International Conference Workshops on Parallel Processing (ICPPW 2006) (2006)
9. Ustemirov, N., Sosonkina, M., Gordon, M.S., Schmidt, M.W.: Concurrent Execution of Electronic Structure Calculations in SMP Environments. In: Proceedings HPC 2005 (April 2005)
10. Ustemirov, N., Sosonkina, M.: Efficient Execution of Parallel Electronic Structure Calculations on SMP Clusters. Minnesota Supercomputing Institute Technical Report umsi-2005-227, University of Minnesota (2005)
11. Kongetira, P.: A 32-way Multithreaded SPARC(R) Processor. In: Proceedings of the 16th Symposium On High Performance Chips, HOTCHIPS (2004)
12. McDougall, R., Mauro, J.: Solaris Internals: Solaris 10 and OpenSolaris Kernel Architecture. Prentice-Hall, Englewood Cliffs (2006)
13. Olson, R.M., Schmidt, M.W., Gordon, M.S., Rendell, A.P.: Enabling the Efficient Use of SMP Clusters: The GAMESS/DDI Model. In: Proceedings of the 2003 ACM/IEEE conference on Supercomputing, November 15-21, 2003, p. 41 (2003)
14. Sun Microsystems Inc., `http://www.sun.com/processors/UltraSPARC-T2/`
15. Kazempour, V., Fedorova, A., Alagheband, P.: Performance implications of cache affinity on multicore processors. In: Luque, E., Margalef, T., Benítez, D. (eds.) Euro-Par 2008. LNCS, vol. 5168, pp. 151–161. Springer, Heidelberg (2008)
16. Wu, M.-S., Bentz, J.L., Peng, F., Sosonkina, M., Gordon, M.S., Kendall, R.A.: Integrating Performance Tools with Large-Scale Scientific Software. In: IEEE International Parallel and Distributed Processing Symposium, 2007. IPDPS 2007 (2007)