Exploring Tuning Strategies for Quantum Chemistry Computations

Lakshminarasimhan Seshagiri[1] , Meng-Shiou Wu[1], Masha Sosonkina[1], and Zhao Zhang[2]

[1] Scalable Computing Laboratory, The Ames Laboratory, US DoE
Ames, IA 50011 USA
sln,mswu,masha@scl.ameslab.gov,
[2] Department of Electrical and Computer Engineering
Iowa State University
Ames, IA 50011 USA
zzhang@iastate.edu

Abstract. Effective execution of applications using a parallel computing environment that share resources such as network bandwidth, I/O and main memory require that some sort of adaptive mechanism be put in place which enables efficient usage of these resources. The adaptation adjusts the most computationally intensive part of the application thus leading to an efficient execution. General Atomic and Molecular Electronic Structure (GAMESS), used for ab-initio molecular quantum chemistry calculations, uses NICAN for dynamically making adaptations so as to improve the application performance in heavy load conditions. The adaptation mechanism uses the iteration time of each SCF iteration to make a decision. The advantage of such a mechanism is the ability to modify the application execution in a very simplistic yet effective manner. In this work, we have explored methods to expand this adaptation mechanism by analyzing the GAMESS performance through the use of fine-grained data.

Key words: Multi-Core, GAMESS, Niagara, Adaptation, NICAN, TAU,Quantum Chemistry

1 Introduction

Computational chemistry applications such as GAMESS [12] 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, UHF, GVB, MCSF) calculations. Such calculations are not only complex but also have high computational requirements. Using the Self Consistent Field (SCF) method,

This work was supported in part by the National Science Foundation Grants NSF/OCI-0749156 and NSF/CHE-0535640; and in part by Iowa State University of Science and Technology under the contract DE-AC02-07CH 11358 with the U.S. Department of Energy.

GAMESS iteratively approximates solution to the Schrödinger equation that describes the basic structure of atoms and molecules. The SCF method is one of the most computationally intensive parts in the GAMESS execution and it gives a good indication of the processor computation power as well the I/O capabilities of the system on which GAMESS is being run. It has two implementations, *direct* and *conventional*, which differ from each other in the handling of the two-electron (*2-e*) integrals. 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. This could prove to be resource intensive in terms of disk space and file systems requirements on certain systems. In the *direct* SCF method, the 2-e integrals are recalculated for each iteration and it's a computationally intensive process. GAMESS calculations utilize distributed resources like memory and disk storage; the HF wave function solution also depends on a lot of other factors like the wave function solution method, the input molecule, basis set and the underlying hardware. The numerous complex computations involved in the calculations of HF wave function solution and the amount of system resources affect application performance.

GAMESS is just one of many computational chemistry applications that are available which offer a range of theoretical methods with varying implementations. As described above, each quantum chemistry application is characterized by different input parameters and the execution performance is also dependent on the underlying hardware. For example, the current computer architectures are moving towards many-core and this will likely affect the application input parameter combination for best possible performance. In a dynamic execution environment, where different applications are running at the same time, the ability of an application to adapt itself to the varying system conditions is very important. Such a tuning capability can be designed for an application, but this requires that the performance data and the application specific metadata is first collected and analyzed. Considering the number of system and application parameters that affect the performance, performance analysis is not easy task. In this work, we have conducted a large number of experiments on several architectures to collect performance data of different granularity as the foundation to develop tuning strategies. By analyzing these data, we have acquired more in-depth understanding of how different architectures affect performance of GAMESS computations, thus helping us in exploring tuning strategies for complex quantum chemistry computations.

The rest of the paper is organized as follows. Section 2 provides an overview of related work in computational quantum chemistry. Section 3 describes the workload used, the architecture of the execution environment and the tools utilized to get the data. Section 4 describes the performance of GAMESS on the different architectures. In the Section 5 we propose the tuning strategy based on the observations from the previous sections. We have not fully implemented the tuning strategy yet but with we have incorporated different parameters affecting

GAMESS performance, thereby improving the dynamic adaptation of GAMESS. Finally we give the conclusions and the future work in Section 6.

2 Related Works

There are many different approaches to tuning high performance chemistry applications like compiler based optimizations, performance modeling and adaptive algorithms. [4–7] have showcased such adaptations in the high performance computing domain. Concurrently, in [3], the authors have described a component based method to provide easy accesses to different scientific computing applications. This work enables CQoS(Computation Quality of Service) through a generic database component that interacts with different chemistry components and a classifier to provide an adaptive mechanism that achieves better performance than any trial and error approach. In [15], a middleware tool NICAN was used to perform this adaptation in GAMESS. It takes advantage of the fact that the SCF process is iterative in nature and the two implementations can be interchanged. NICAN helps to decouple the application from having to make any adaptation decisions during the execution. The application is responsible only for the invocation of the adaptation handlers. The adaptations are handled by a control port that is a part of the NICAN tool. The NICAN adaptation process and its results for SMP clusters is explained in detail in [15]. We propose to extend the NICAN functionality to incorporate the tuning strategy devised in this paper.

3 Methodology

We have observed in [15] and in [3] that the data used to adapt the application is very coarse grained and coarse grained implies that the adaptation algorithm depends only on the wall clock time. In [15], the data regarding the iteration time is collected on-the-fly and utilized by the middleware NICAN in order to make the adaptation decision. This mechanism makes the tuning decision depending on whether the iteration time is above a certain limit. In [3], the data is collected offline and fed into a database which helps the CCA components to make a decision on tuning the quantum chemistry application. While these approaches provide certain adaptive mechanisms, using only coarse-grained performance prevents us from gaining insight into how and why a computation performs differently on different architectures, and why different sets of molecules can show totally different performance characteristics. To design tuning strategies that cover such large parameter sets, a methodology that covers from data acquisition, performance data and metadata management to performance analysis is desired.

The first step is to choose an application workload that is diverse enough to help us to discern between different nuances of the performance output. The molecules need to be useful in the practical sense since that allows us to obtain

data that is as close as possible to a real life scenario. All the molecules that we have chosen are important from the point of view of chemistry and biology. The molecules chosen in our tests include molecules representing fundamental aromatic systems, represent models used for DNA stacking and protein folding and are part of carbon nano materials. More information regarding the input molecules has been provided in the *Application Workload* subsection. The application performance depends a lot on the hardware on which it is run. Each hardware characteristic such as the processor type, the cache design, the memory bandwidth and the inter-nodal connection determine the application performance to a great extent. We require performance data from diverse architectures in order to cover as much of the architectural features as possible. The details regarding the three different architectures that we have chosen in accordance with our requirements have been given in the *Architectures Used* subsection. The data collection is a very laborious process considering that hundreds of data files have to be collected over different architectures and different input parameter settings. The code profiler used to collect the data needs to be available and proven to work over different operating systems and different processor architectures. One such tool is the TAU toolkit [2], and more details regarding its usage in our data collection have been described in the subsection *Tools Used.*

Application Workload

In [1], we had chosen *Luciferin* and *Ergosterol* molecules to test the GAMESS performance on a SMP cluster and a Sun Niagara T2 processor. These two molecules were also used to test the NICAN adaptation strategy on the Sun Niagara T2 processor. These molecules were chosen because of the relative difference in their execution times on the above two architectures. In this work, we have chosen two different sets of molecules. The first set contains 7 molecules having varying molecular structure as shown in Figure 1. These molecules are Benzene(bz) and its dimer, Naphthalene (np) and its dimer, Adenine-Thymine DNA base pair (AT), Guanine-Cytosine DNA base pair(GC) and Buckminsterfullerene (C60). The number of basis functions are shown in parenthesis. This diverse set of molecules allows us to see if there are any similar characteristics in the performance data. The details regarding their practical usage is given in [3]

The second is a set of 6 Benzene molecules that are very similar in their structures. The molecules are Picene, Pentacene, Dibenz Anthracene (J and H) , Benzo naphthacene and Benzo triphenylene. This is advantageous since it gives us a group of molecules having a very similar molecular structure with very little differences. We can study the performance characteristics in very minute detail due to this property. We can also check and confirm if the performance characteristics of one molecule can be applied to the rest of the molecules in the set.

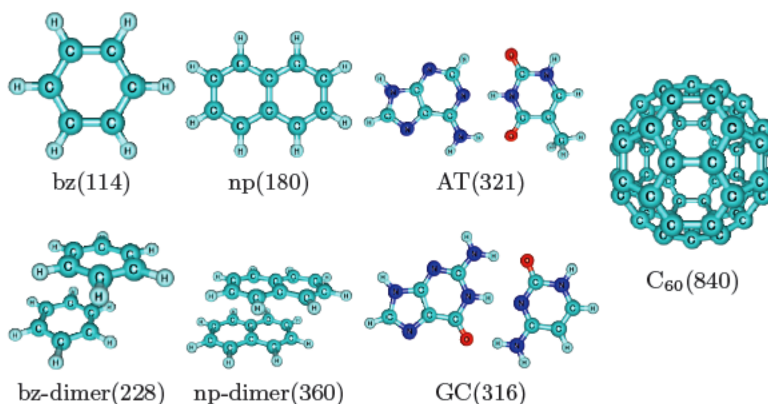


Fig. 1. Hiro Inputs

Architectures Used

We used three different architectures to get the performance data. The first is an Ames Lab SMP cluster “Borges” that consists of 4 nodes, each node having two dual-core 2.0GHZ Xeon “Woodcrest” CPUs and 8GB of RAM [8]. 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) [18, 21]. The T2 processor has a unique architecture that 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 contains 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.

The third machine used is Franklin, which is a massively parallel processing (MPP) CRAY-XT4 system with 9,572 compute nodes provided for scientific use by NERSC(National Energy Research Scientific Computing Center). Each node has quad processor cores, and the entire system has a total of 38,128 processor cores available for scientific applications. Each of Franklin's compute nodes consists of a 2.3 GHz single socket quad-core AMD Opteron processor "Budapest" with a theoretical peak performance of 9.2 GFlop/sec per core (4 flops/cycle if using SSE128 instructions). Each compute node has 8 GB of memory (2 GB of memory per core). Each compute node is connected to a dedicated SeaStar2 router through Hypertransport with a 3D torus topology.

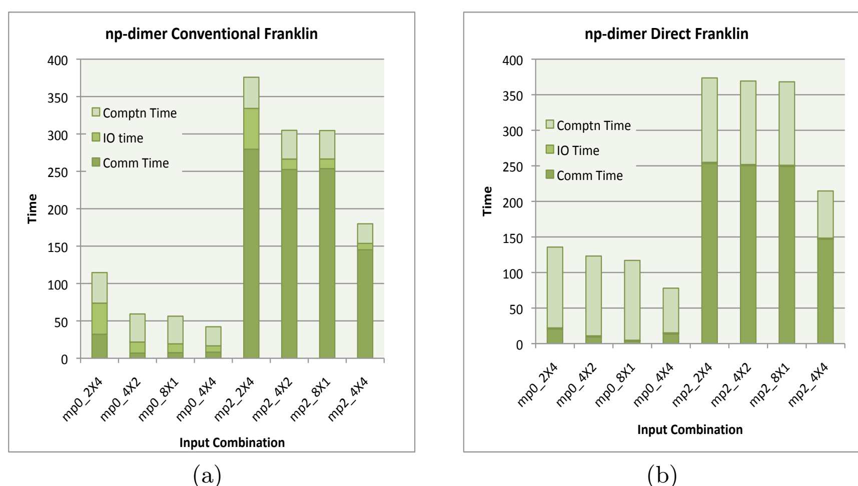
Tools Used

The adaptation in [3] uses wall clock time while the adaptation mechanism described in [15] uses the individual SCF iteration time. These methods have proven to be successful but they do not provide the complete picture on the application behavior on different architectures. Given that we have a large number of molecules and their large number of input parameters, it is possible that there are many conditions where the adaptation mechanism does not provide the desired performance improvement. To explore adaptation mechanisms to handle complex scenarios, we need to profile the application, obtain performance data for a wide range of test parameters and then analyze the obtained data. For any scientific application, the runtime can be represented as a sum of the time spent by the application in computing the required data, the time spent by different threads of the program communicating with each other and the time spent by the program in moving the data back and forth between the disk and the memory (the time spent in I/O). These three components usually provide a good insight as to where the application is getting slowed down during its execution and can be obtained using profiling tools. This led us to use the TAU [2] (Tuning and Analysis Utility) toolkit, which is a popular multi-level and multi-user source code profiler and instrumentor. Since TAU can extract metadata such as the application and system characteristics to the granularity required by us in these experiments, it is extremely useful in our work.

4 Performance Results and Observations

The performance data were collected for different combinations of the processors and GAMESS processes. It is not possible to represent performance data for all the 13 molecules here. Hence the performance data have been shown for the np-dimer and C60 molecules since they allow us to showcase the important observations leading to the tuning mechanism. There were a couple of failures, which we would like to note here. One recurring theme that we found out was

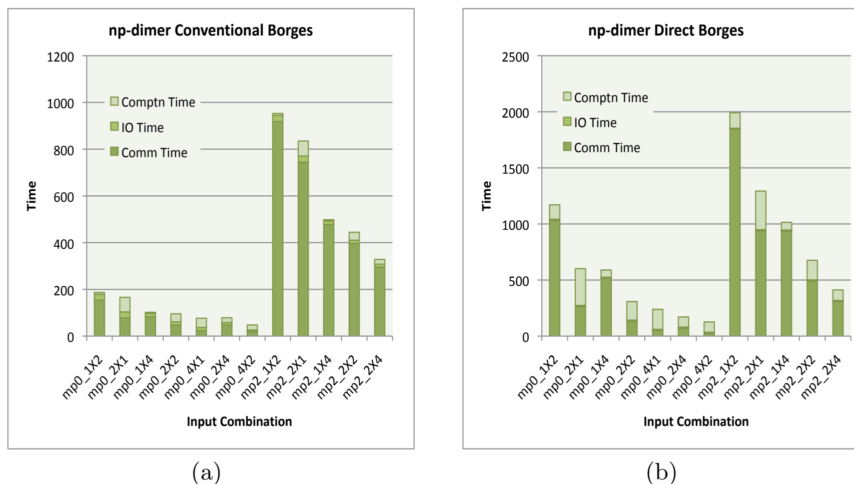
that the C60 molecule failed every time we used the MP2 "electron correlation" calculation due to its high memory requirement. The picene molecule (part of the second set of 6 benzene molecules) failed on the Niagara machine only when running the TAU instrumented code. We have observed that the failure could be due to TAU invariably changing some variable in case of picene. Let us now look at all the observations from the results that were obtained. The results have been shown in Figures 2, 3 and 4. The figures are calibrated with respect to the combination of the number of processors per node and the input type (MP0 or MP2). For example, "2x4" implies that GAMESS application data has been collected with the job executing on 2 Nodes, running 4 GAMESS processes each. *It is important to remember that on Franklin and Borges, we cannot run more than 4 GAMESS processes per node while on the Niagara machine, we can run 8 processes per core.* In case of the Niagara machine, "2X4" implies 2 cores running 4 GAMESS processes each.



(a) (b)

Fig. 2. (a)np-dimer conventional molecule results on Franklin (b)np-dimer direct molecule results on Franklin

From the results shown in Figures 2, 3 and 4, we can clearly see that the execution time for the *conventional* implementation is different than the *direct* implementation on all the three architectures for the np-dimer molecule. This difference exists for all the molecules tested in the course of these experiments. For most molecules, we found that the *conventional* implementation is faster but for larger molecules like C60 and Ergosterol (As shown in [1]), the *direct* implementation is faster. This difference is exploited in the existing adaptive mechanism implemented through NICAN discussed in [15]. It was established in this paper



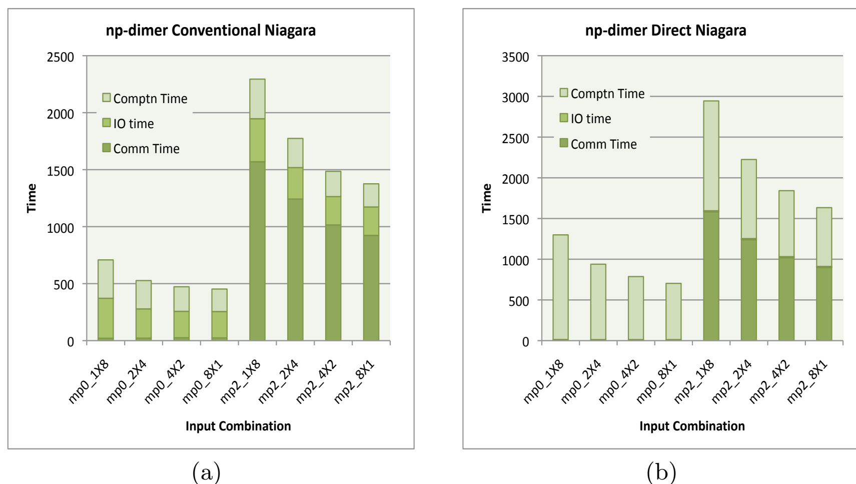
(a) (b)

Fig. 3. (a)np-dimer conventional molecule results on Borges (b)np-dimer direct molecule results on Borges

that given a high I/O usage on a system, the *conventional* method slows down considerably and switching to the *direct* implementation using NICAN middleware ensures better GAMESS performance.

From Figures 2, 3 and 4, it's clear that on all the three architectures, the total time taken for MP2 is at least 3 times higher than the time taken to complete MP0 calculations. In some cases, the MP2 time is nearly 10 times as high as the time taken to complete the MP0 calculations. On comparison of the three performance timings for MP0 and MP2, we can see that the I/O time and the computation time are fairly constant in both these implementations but there is an increase in the communication time. This seems counter-intuitive since we would expect MP2 calculations to increase the computation time. However, this can be explained by how the MP2 calculations are performed. In case of MP2 calculations, four matrices are created to hold the output at each node and then these matrices are aggregated. The cost involved in transmitting this data over the network is huge and results in an increase in the communication time. MP2 calculations give a higher degree of accuracy over MP0. Switching between the two cases does not actually make sense since the user has opted for MP2 in order to obtain this accuracy. Its obvious that for such cases, other switching techniques would need to adopted.

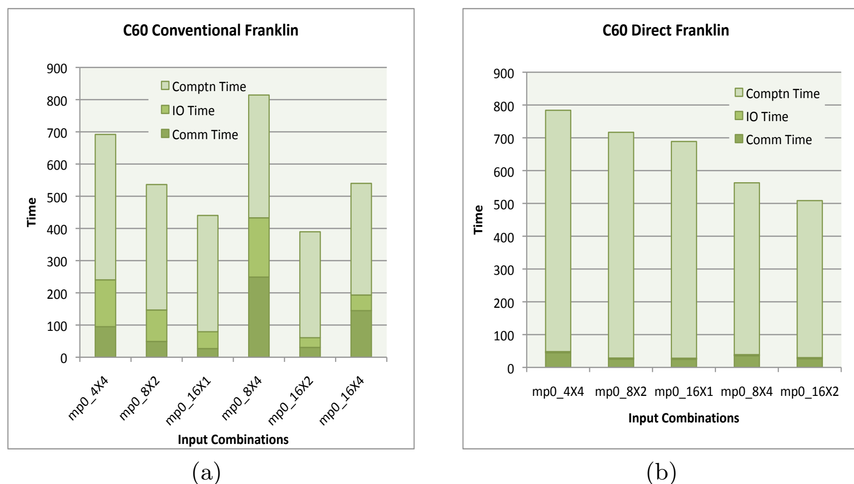
In order to develop other switching techniques for the GAMESS molecules, we need to deduce other application characteristic patterns through Figures 2, 3 and 4. These figures are for the molecule np-dimer for different combinations of nodes and processes per node. On Franklin (figure 2), if we keep the total



(a) (b)

Fig. 4. (a)np-dimer conventional molecule results on Niagara (b)np-dimer direct molecule results on Niagara

number of processes as 8, then we get three different combinations of 2x4, 4x2 and 8x1. For the conventional MP0 method, the cost of I/O varies from 42 seconds for 2x4 to 15 seconds for 4x2 to 12 seconds for 8x1. A similar trend can be seen for MP2 conventional as well. If we increase the number of processes on a single node instead of distributing amongst more nodes, it may be deduced that the I/O contention increases. However, this observation does not work in the case of 4x2 and 4x4. The I/O time falls when we move from 4x2 to 4x4. The I/O contention depends on the bandwidth, the I/O channel and memory system interconnection structure, data size and the operating system. A more thorough investigation will be needed to untangle the complex interactions among the system resources. Consider the Figure 5, which shows the results for the C60 molecule on Franklin. We found that C60 successfully executes only if at least 16 processes are spawned irrespective of the distribution on the nodes. Hence the C60 *conventional* results have been given for 4X4, 8X2, 8X4, 16X1, 16X2 and 16X4 combinations. For the *conventional* molecule, as we keep the number of processes constant and change the number of nodes on which the job is run, we can see a general reduction in the runtimes. For the input combinations of 4X4, 8X2 and 16X1, the computation time is nearly constant while the I/O and communication time reduces. Intuitively, for better performance, we are looking at getting more resources for the application. However, as the number of cores per node increases, the complex interaction among system resources, gives us a more unpredictable nature of results. This can be seen in the results for 16X1, 16X2 and 16X4. The I/O and computation time reduce and the communication time remains fairly constant when the number of processes increases from 1 to 2. But the communication time increases dramatically when the number of pro-



(a) (b)

Fig. 5. (a)C60 conventional molecule results on Franklin (b)C60 direct molecule results on Franklin

cesses on a single node is increased to from 2 to 4. This trend indicates that for larger molecules, at higher distribution of processes among nodes, there is a very good chance that the *conventional* method gives rise to I/O contention or the operating system does not handle well for I/O with large data size requests. We would require more fine-grained performance data for further investigation. The more consistent trend that we can observe here is that of the communication cost increasing when the number of processes on a single node is increased. This trend can possibly be exploited in such a way that the distribution of processes among different nodes is modified to obtain the least communication cost possible.

Referring to Figure 3, which indicates the results for np-dim on Borges, we can see that for the 1x2 and 2x1 combinations, there is a big increase in the computation cost while the communication cost reduces. This is consistent for other combinations like 1x4 and 4x1, 2x4 and 4x2 though in a lesser degree, and for these mentioned combinations in the *direct* implementation. This is surprising since intuitively we expect the communication cost to increase when the processes get distributed over the network. One of the possible reasons for this could be an issue with the shared memory and inter-nodal communications and more fine-grained data is required for investigating this issue. The results also show good scalability on this architecture for both *direct* and *conventional* method. The increase of the number of processors on a single node does not change any of the characteristics associated with the application. The percentage of communication, I/O and computation times remains constant as we move from 1x2 to 1x4 or from 2x1 to 2x2. However, the performance characteristics

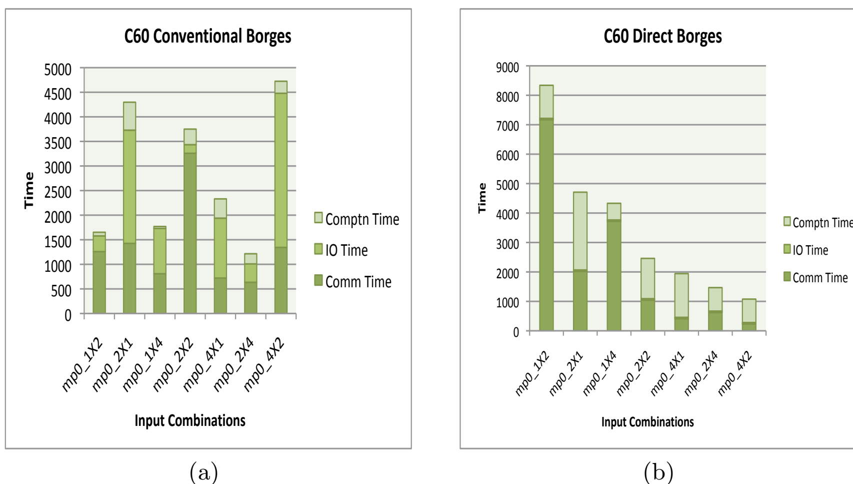


Fig. 6. (a)C60 conventional molecule results on Borges (b)C60 direct molecule results on Borges

change when we move from 2x2 to 2x4 or from 4x1 to 4x2, which shows that increasing the number of processes from 4 to 8 probably caused this. Also, it is important to note that the Borges cluster might not be as fine-tuned as Franklin since Franklin is a cluster being used by the scientific community and this could result in some adverse results being more prominent on Borges.

The observations from the results for np-dimer on the Niagara machine are shown in Figure 4,. We can notice that the ratio of I/O on the Solaris machine is much higher than the ratio of I/O on the other machines for the conventional method. The Niagara machine can be considered as a single node multi-core machine that can run multiple threads on each of its cores. The cache system is shared between the cores using a crossbar architecture, which ensures that the time taken to access the cache is constant for all the cores. The memory bandwidth gets shared among the cores. Hence for larger number of processes, there is a good possibility of contention. As we increase the distribution of the GAMESS threads among the cores, the time taken to complete the execution goes down. This is due to the increase in the amount of hardware available for execution. The scalability of the Niagara processor was shown in [1]. However, we would need more performance data to have a better understanding of this architecture. It would suffice to say that with the data we have, the I/O does seem to be a bottleneck. On such architectures, an adaptation strategy can be designed which would look to distribute the number of processes to the maximum available cores and thus improve application performance. However, we can also see from the results that the scalability is not high enough. Even after increasing the number of cores from 1 to 8, the performance improves by a factor

of 2.

5 Tuning Strategy

The observations in Section 4 helps us in formulating the tuning strategy and in incorporating them into NICAN. The usage of NICAN has been explained in detail in [15] and [1]. The NICAN adaptation mechanism 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 [16] 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 NICAN structure shown in Figure 7 consists of a control port which has a handle to the GAMESS application. After every SCF iteration, the control port checks for different parameters such as "peer" GAMESS jobs and system resources in order to determine the best possible method to perform the next SCF iteration. Depending on this information, it switches between *conventional* and *direct* implementations in order to achieve best possible performance. The static and dynamic adaptations can be controlled by the user through a NICAN input file.

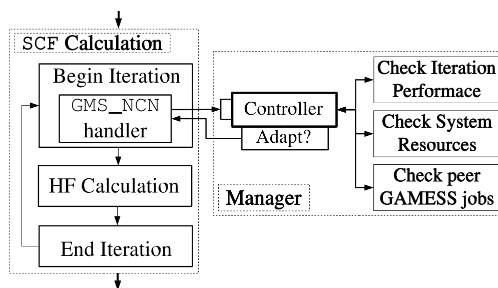


Fig. 7. NICAN control port structure

The tuning strategy that we propose on the basis of the results obtained will augment the existing NICAN adaptation strategy. Considering the amount of information available, it is prudent to utilize a database for storing this information. The NICAN manager spawns a thread that connects to the database and gets the required information. The current implementation of NICAN requires a check run in order to get the memory requirements of the input molecule. We propose that this information can be loaded offline into the database. One important information that can be offloaded into the database is the size of the

files that will be written on the disk in case of a *conventional* implementation. The external disk sizes on clusters are normally huge. However, its possible that in case of small clusters, the disk size might get exceeded due to residual files. This information would allow NICAN to calculate the amount of space available to store integral files and in case that sufficient space is not available, the implementation can be modified to *direct*. Check runs are also useful in cases such as the MP2 calculations for C60 molecules where they provide the user with the exact input memory requirements in order for the job to successfully execute. This information can be stored in the database, which would ensure that the job is stopped immediately after NICAN checks the input parameters and not when the MP2 calculations are being done.

The adaptation between *conventional* to *direct* is an existing feature in NICAN and we don't intend to modify the algorithm. However, we will incorporate database access as a NICAN module due to easy extensibility to the NICAN features. From the results obtained, we have seen the best combination to obtain the most efficient application performance. These combinations can be stored in the database for each particular operating environment. For example, on a Sun T2 Niagara machine, the best method to obtain fastest application run time would be to distribute the number of GAMESS processes to on as many cores as possible. Obviously, this adaptation would be possible only if there are cores or nodes (in case of Franklin and Borges) available so as to distribute the processes. The scalability of GAMESS on the Niagara machine is something, which needs to be studied, but we can get a decent speed up for large molecules by increasing the number of cores used for execution. It is also important to note that the application characteristics on different input processor and node combinations such as the computation time, I/O time and communication time differ from one executing environment to another as observed in Section 4. Hence, NICAN needs to have the ability to recognize the architecture on which it is currently running. Such recognition will help the application to dynamically adapt to varying system conditions.

6 Conclusions and Future Work

In this work, we have conducted a preliminary performance analysis of GAMESS in order to develop an automatic tuning strategy. Quantum chemistry applications such as GAMESS have numerous input parameters, which determine their performance characteristics. Also, the architecture on which the application is executed makes a difference to the application performance. The performance data collected has given us good directions to proceed. There are a few trends that stand out despite the differing architectures. These include, MP2 is taking more time than MP0 to complete due to the increase in communication cost; *direct* taking more time than the *conventional* implementation for all the tests conducted till now. The tuning strategy has been briefly mentioned in the Sec-

tion 5.

Future work includes running more performance evaluation tests on the three architectures mentioned above. There are open issues to resolve such as the I/O contention when the processes get split among different nodes, the effect of architecture and the operating system on the I/O contention and the communication bottleneck which could be due to shared memory communications. The current level of instrumentation and profiling is not enough to give the required information to deduce the reasons for the above issues. Hence we would need to conduct more performance analysis in order get to the bottom of these issues.

References

1. Lakshminarasimhan Seshagiri, Masha Sosonkina and Zhao Zhang Electronic Structure Calculations and Adaptation Scheme in Multi-core Computing Environments In Proceedings of 2009 International Conference on Computational Science (ICCS-2009), Baton Rouge, Louisiana, May 25-27, 2009
2. Shende, S. , Malony A. The TAU parallel performance system Int. J. High-Perf. Computing Appl., ACTS Collection special issue 20 (Summer 2006) 287-331
3. Li Li and Joseph P. Kenny and Meng-Shiou Wu and Kevin Huck and Alexander Gaenko and Mark S. Gordon and Curtis L. Janssen and Lois Curfman McInnes and Hirotoshi Mori and Heather M. Netzloff and Boyana Norris and Theresa L. Windus Adaptive Application Composition in Quantum Chemistry Proceedings of The 5th International Conference on the Quality of Software Architectures (QoSA 2009) February 2009
4. Jack Dongarra and Victor Eijkhout Self-adapting Numerical Software for Next Generation Applications INT. J. HIGH PERF. COMPUT. APPL, 2003
5. Liu, Hua and Parashar, Manish Enabling self-management of component-based high-performance scientific applications HPDC '05: Proceedings of the High Performance Distributed Computing, 2005. HPDC-14. Proceedings. 14th IEEE International Symposium
6. Tapus, Cristian and Chung, I-Hsin and Hollingsworth, Jeffrey K. Supercomputing '02: Proceedings of the 2002 ACM/IEEE conference on Supercomputing 2002
7. Vuduc, Richard and Demmel, James W. and Bilmes, Jeff A. Statistical Models for Empirical Search-Based Performance Tuning Int. J. High Perform. Comput. Appl., 2004, pg 65-94
8. Christian Terboven, Dieter an Mey and Samuel Sarholz. OpenMP on Multicore architectures. A Practical Programming Model for the Multi-Core Era, Lecture Notes in Computer Science, Springer, Berlin Heidelberg, 54-64 (2008).
9. D. Kulkarni and M. Sosonkina. A framework for integrating network information into distributed iterativesolution of sparse linear systems. High Performance Computing for Computational Science - VECPAR 2002, 5th International Conference, Porto, Portugal, June 26-28, 2002, Selected Papers and Invited Talks, volume 2565 of Lecture Notes in Computer Science, pages 436450. Springer, 2003.
10. E.H. White, F. Capra, W.D. McElroy. The Structure and Synthesis of Firefly Luciferin *J. Am. Chem. Soc.*, 83(10), 2402-2403(1961).
11. John L Hennessy, David A. Patterson with contributions by Andrea C. Arpaci-Dusseau [et al.]. Computer architecture: A quantitative approach 4th ed. Morgan Kaufmann, 2006.

12. M.W. Schmidt, K.K. Baldridge, J.A. Boatz, S.T. Elbert, M.S. Gordon, J.H. Jensen, S. Koseki, N. Matsunaga, K.A. Nguyen, S. Su, T.L. Windus, M. Dupuis, J.A. Montgomery. General Atomic and Molecular Electronic Structure System. *Journal of Computational Chemistry*, 14, 1347-1363(1993).
13. M. Sosonkina. Adapting Distributed Scientific Applications to Run-time Network Conditions. In Applied Parallel Computing, State of the Art in Scientific Computing, 7th International Workshop, PARA 2004, Revised Selected Papers, volume 3732 of Lecture Notes in Computer Science, pages 745–755. Springer, 2006.
14. M. Sosonkina, S. Storie. Parallel performance of an iterative method in cluster environments: an experimental study. In *Proceedings PMAA 2004*, October 2004.
15. Nurzhan Ustemirov, Masha Sosonkina, Mark S. Gordon and Michael W. Schmidt. Dynamic Algorithm Selection in Parallel GAMESS Calculations. International Conference Workshops on Parallel Processing, (ICPPW'06).
16. Nurzhan Ustemirov, M. Sosonkina, M.S. Gordon, M.W. Schmidt. Concurrent Execution of Electronic Structure Calculations in SMP Environments. In *Proceedings HPC 2005*, April 2005.
17. Nurzhan Ustemirov, M. Sosonkina. Efficient Execution of Parallel Electronic Structure Calculations on SMP Clusters. Minnesota Supercomputing Institute Technical Report umsi-2005-227, University of Minnesota, 2005.
18. Poonacha Kongetira. A 32-way Multithreaded SPARC(R) Processor. In Proceedings of the 16th Symposium On High Performance Chips (HOTCHIPS), 2004.
19. Richard McDougall and Jim Mauro. Solaris Internals: Solaris 10 and OpenSolaris Kernel Architecture. Prentice Hall, 2006.
20. R. M. Olson, M. W. Schmidt, M. S. Gordon, A. P. Rendell. Enabling the Efficient Use of SMP Clusters: The GAMESS/DDI Model, Proceedings of the 2003 ACM/IEEE conference on Supercomputing, p.41, November 15-21, 2003.
21. Sun Microsystems Inc. http://www.sun.com/processors/UltraSPARC-T2/.
22. Vahid Kazempour, Alexandra Fedorova, Pouya Alagheband Performance Implications of Cache Affinity on Multicore Processors. Euro-Par 2008: 151-161.
23. Wu, M.-S. and Bentz, J.L. and Peng, F. and Sosonkina, M. and Gordon, M.S. and Kendall, R.A. Integrating Performance Tools with Large-Scale Scientific Software. IEEE International Parallel and Distributed Processing Symposium, 2007. IPDPS 2007.