2009

Performance analysis and middleware assisted adaptation for quantum chemistry computations

Lakshminarasimhan Seshagiri
Iowa State University

Performance analysis and middleware assisted adaptation for quantum chemistry computations

by

Lakshminarasimhan Seshagiri

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Engineering

Program of Study Committee:
Masha Sosonkina, Co-major Professor
Zhao Zhang, Co-major Professor
Mark Gordon

Iowa State University

Ames, Iowa

2009

DEDICATION

I would like to dedicate this thesis to my family without whose support I would not have been able to complete this work. I would also like to thank my friends for their help during the writing of this work.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENTS

I would like to take this opportunity to express my thanks to those who helped me with various aspects of conducting research and the writing of this thesis. First and foremost, Dr. Masha Sosonkina for giving me this opportunity to work at Ames Lab and for her guidance, patience and support throughout this research. I would like to thank my committee members Dr. Zhao Zhang and Dr. Mark Gordon for their guidance and contributions to this work. I am thankful to Dr.Zhang's for his guidance throughout my graduate study. Special thanks to Dr.Meng-Shiou Wu, for his immense help in my research and without whose guidance this work would have been incomplete. I would also like to thank Dr.Mike Schimdt for his patience in explaining quantum chemistry and GAMESS computations to me.

ABSTRACT

Quantum chemistry applications such as General Atomic and Molecular Electronic Structure System(GAMESS) that can execute on a complex peta-scale parallel computing environment have a large number of input parameters that affect the overall performance. The application characteristics vary according to the input parameters. This is due to the difference in the usage of resources like network bandwidth, I/O and main memory according to the input parameters. Effective execution of applications in a parallel computing environment that share such resources require some sort of adaptive mechanism to enable efficient usage of these resources. The adaptation adjusts the most computationally intensive part of the application thus leading to sizable gains. General Atomic and Molecular Electronic Structure System (GAMESS), used for ab-initio molecular quantum chemistry calculations, utilizes NICAN (Network Information Conveyer and Application Notification) for dynamically making adaptations so as to improve the application performance in heavy load conditions. The adaptation mechanism has the ability to modify the application execution in a very simplistic yet effective manner. In this work, we have explored methods to expand the structure of NICAN in order to include other input parameters based on which the application performance can be controlled. The application performance has been analyzed on different architectures to otain fine grained performance data and a tuning strategy has been identified. A generic database framework has been incorporated in the existing NICAN mechanism.

CHAPTER 1. OVERVIEW

1.1 Introduction to Quantum Chemistry

Chemistry is the science dealing with construction, transformation and properties of molecules. Theoretical chemistry is the subfield where mathematical methods are combined with fundamental laws of physics to study processes of chemical relevance (6). For a given set of nuclei and electrons, theoretical chemistry can attempt to calculate different molecular properties such as the geometry of the nuclei and their relative energies. Theoretical chemistry or quantum chemistry is based on the simplified time-independent Schrödinger equation:

$$H\Psi = E\Psi$$

where H is the Hamiltonian operator, Ψ is a set of wave-functions and E is the total energy of the system. The methods aimed at solving the electronic Schrödinger equation are called as "Electronic Structure Calculations". These equations can be solved either by using semi-empirical models or by using *ab initio* methods. *Ab initio* methods iteratively generate approximate solutions without reference to experimental data. The *Hartee-Fock* equations can be derived from the above equation. The HF model is a branching point for getting further approximations or for getting more accurate treatments. SCF or Self Consistent Field orbitals are a set of functions that provide a solution to the HF model. In large basis sets (A basis set is a set of functions used to create the molecular orbitals), the SCF method requires the usage of computational resources which increases as the number of basis functions to the fourth power. Thus the scaling is of the order N^4 where N is the number of basis functions describing the atom. This scaling can be smaller in actual calculations. (as given in (6))

GAMESS is only one of the applications used by chemists worldwide to perform *ab initio* calculations. MOLPRO, NWChem and MPQC are other packages that are used for performing *ab initio* calculations. MOLPRO (27) is a package of *ab initio* programs for electronic structure calculations. NWChem (9) is written in Fortran 77 and uses Global Arrays on top of the ARMCI (Aggregate Remote Memory Copy) library (14) for parallel communications. MPQC (7) or the Massively Parallel Quantum Chemistry program is another computational chemistry package that is written in C++ programming language. It uses MPI as its communication mechanism. We have performed this research on GAMESS and hence we have provided an introduction to GAMESS in the subsequent chapters.

1.2 Related Work

There are many different approaches to tuning high performance chemistry applications like compiler based optimizations, performance modeling and adaptive algorithms. Dongarra and Eijkhout (2) talk about *Self-adapting Numerical Software* systems that consist of a framework of a database assisted decision making component, a Network Scheduler and underlying Adaptable libraries in order to automatically pick the best software/hardware combination for High Performance Computing. Liu and Parashar (5) provide self-managing high performance simulations based on the Common Component Architecture (CCA). Tapus et al. (24) talk about a framework called Active Harmony that allows runtime switching of algorithms and tuning of library and application parameters. Vuduc et al. (32) focus on searching the implementation space using heuristic performance modeling and empirical evaluation. Concurrently, Li et al. (12) 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. Ustemirov et al. use (29) a middleware tool NICAN 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 ap-

plication 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 (29). Taylor et al. (25) have created a performance database for distributed scientific applications. This relational database allows for recording the performance data, system features and application details that help in analyzing the performance of scientific applications. However, this database can only be used as a means for creating performance models and predicting the performance of an application on different systems. We have not only created a performance database for performance analysis but also extended the NICAN functionality to incorporate the tuning strategy devised in this research.

1.3 Thesis Layout

The thesis has been organized as follows. Chapter 2 gives background information on GAMESS, NICAN, adaptation using NICAN and the necessity for expanding the reach of the existing adaptation strategy. In Chapter 3, we present the performance analysis of GAMESS on different architectures and the usage of this performance data to develop an adaptation strategy. Chapter 4 presents the adaptation strategy and its results. Finally in Chapter 5, we summarize our research and the future direction that this research can take.

CHAPTER 2. APPLICATION AND ADAPTATION STRATEGY INTRODUCTION

2.1 GAMESS

Computational chemistry applications such as GAMESS (16) are widely used to perform ab-initio molecular quantum chemistry calculations. These calculations include a wide range of Hartree-Fock (HF) wave function calculations such as RHF(Restricted Hartree-Fock), ROHF(Restricted open shell Hartree-Fock) , UHF(Unrestricted Hartree Fock), GVB(Generalized valence bond wavefunction), MCSF(Multiconfigurational SCF wavefunction). The capabilities of GAMESS has been described in detail in the appendix. GAMESS computations can be run in parallel mode on HPC environments to utilize distributed resources such as main memory and disk storage. GAMESS uses the Distributed Data Interface(DDI) server model to utilize the shared memory on Symmetric Multiprocessor (SMP) nodes. Such calculations are not only complex but also have high computational requirements. Using the Self Consistent Field (SCF) method, GAMESS iteratively approximates solution to the Schrödinger equation that describes the basic structure of atoms and molecules. The SCF method 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 and basis set. The numerous complex computations involved in the calculations of HF wave function solution and the amount of system resources affect application performance. 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. Conducting performance analysis is not easy task due to a large set of system and application parameters than can affect the overall performance. Huge amounts of performance data with proper granularity needs to be collected for a large number of molecules on different architectures in order to discern the performance trends. By analyzing these data, we can acquire 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.

2.2 NICAN Middleware Library

Network Information Conveyer and Application Notification or NICAN (22) is a framework that is used to provide methods for applications to adapt their utilization of computational resources based on various conditions. NICAN has been successfully integrated with several applications to aid their execution in a distributed environment. NICAN was used to monitor the network bandwidth, analyze the collected data and notify the network benchmarking tool NetPIPE (1) by Gan Chen in his Master's thesis. Similarly, NICAN was integrated with Parallel Algebraic Multilevel Solver (pARMS) by Sosonkina and Kulkarni (11), where NICAN helps pARMS to manage changes in network conditions. Nurzhan Ustemirov, in his M.S,

thesis has integrated NICAN with GAMESS and created an adaptation infrastructure in order to improve the application performance under heavy system load (29). We have augmented Nurzhan's work with a database framework and this has been described in the Chapter 4.

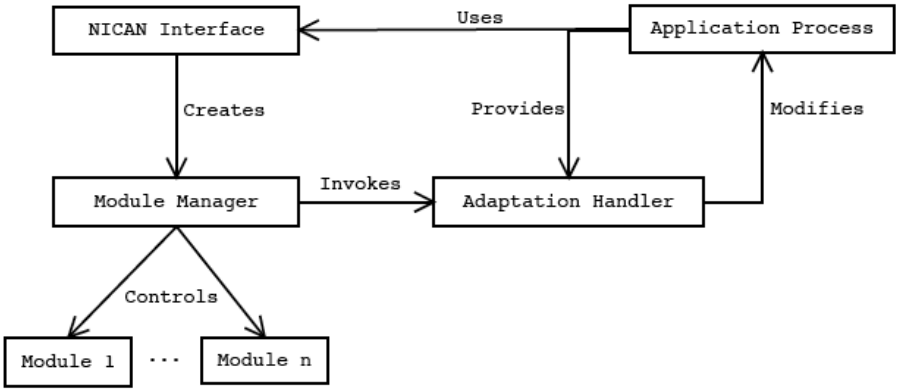


Figure 2.1 NICAN layout

NICAN provides modules to monitor various attributes of the environment in which the application is operating. NICAN is unique in the sense that not only does it provide a method of monitoring specific system and application characteristics; it also provides a mechanism for the computing path of applications to be modified when a specific trigger condition is reached. The application and NICAN communicate with each other regarding the resources that needs to be monitored and adapted. This communication is carried out using a register/notify paradigm. The application is responsible only for starting the NICAN library that invokes the user-defined modules used for application/system monitoring. NICAN uses an application handler provided by the application to enable application adaptations. All the modules are dynamic libraries that can be loaded into NICAN. The trigger conditions for the characteristics being monitored can be modified by the user through an input file.

2.3 Adapatation using NICAN

The adaptation in GAMESS using NICAN was designed for SMP (Symmetric Multi Processor) clusters in order to improve GAMESS performance by Nurzhan Ustemirov (29). The GAMESS package consists of a computational part written in Fortran and a memory management part written in C (handles DDI). Since modifying GAMESS source code could have reduced its usability by application scientists, it was essential to use an adaptation mechanism which would avoid making changes to the computation part of GAMESS. Instead a library based approach with the help of NICAN was designed. The adaptation focussed on the SCF algorithm of the GAMESS computation which is one of the most computationally intensive parts. 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 GAMESS-NICAN integration model (Figure 2.2) includes both specific- (S) and general-use (G) modules. GAMESS-Check specific module calculates required memory for the given job and makes proper memory parameter adjustments to the GAMESS input file. Memory module monitors available physical memory on the node. The disk I/O module checks the available I/O resources by performing light weight quick benchmarking. The Daemon module starts NICAN daemon (NcnD), if there is no other NcnD running on the node, for observing peer application jobs and for communication between distributed NICAN processes. The daemon is self-contained and is independent from the job for which it has been started. The daemon performs standard operations such as: read, write, delete and self-destroys if there are no records. Each record consists of a process ID and a description to the job. Thus, the Manager can record any useful information about the process executed on the node. The design

of NcnD is versatile, so that the daemon module may be used for any application integrated with middleware NICAN.

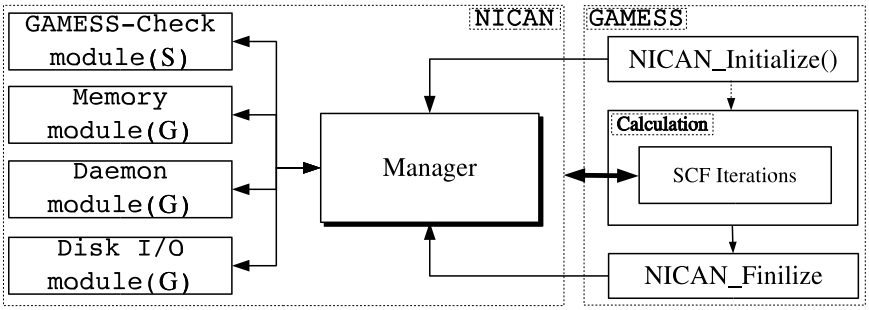


Figure 2.2 GAMESS-NICAN Integration Model

The adaptation scheme used in (29) 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 (30) 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)

$\Delta t_0 = | t_0^e - t_0 |$

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 (29). 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.

CHAPTER 3. INITIAL PERFORMANCE MEASUREMENTS

The NICAN dynamic adaptation discussed in Chapter 2 uses the iteration time information collected on the fly in order to make the adaptation decision. This mechanism makes the tuning decision depending on whether the iteration time is above a certain limit. The data used to adapt the application is very coarse grained and here coarse grained implies that the adaptation algorithm depends only on the wall clock time. Also, currently, the adaptations are possible only between direct and conventional implementation of GAMESS. It has been shown in (29) by Ustemirov et al. that this adaptation is extremely effective in improving GAMESS application performance. However, GAMESS input parameters are very vast and the job of formulating an adaptation algorithm for such applications is very laborious and complex. Thus we would need to include much more fine-grained performance data to the existing adaptation mechanism in order to derive a strategy that reflects this complexity. One of the methods that can be used to provide this fine-grained performance data to the NICAN library for decision-making is to obtain this data apriori and allow NICAN to access it anytime. Thus having a database component to the existing NICAN framework would provide NICAN with the repository of the required fine-grained performance data. This also implies that the management of performance data is of paramount importance. The following sections give a more detailed look at the performance analysis done for GAMESS and the development of the adaptation strategy using the obtained performance data.

3.1 Methodology

We have observed in the works by Ustemirov, Sosonkina et al. ((29)) and Li, Kenny et al(12) that the data used to adapt the application is very coarse grained which implies that

the adaptation algorithm depends only on the wall clock time. In (29), 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 based on whether the iteration time is above a certain limit. In (12), 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. One of the disadvantages of using only a coarse grained performance data is that it 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 for large parameter sets, a methodology that spans data acquisition, performance data, metadata management and performance analysis is desired. We proceeded to design this tuning strategy by following a collection-analysis-implementation method. The performance data for GAMESS was collected on different architectures and using different sets of molecules so as to understand the application performance variation. Using this data, application performance analysis was performed and the performance trends were identified. An adaptation strategy was formulated depending on these trends.

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, 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

acquire performance data from diverse architectures in order to cover as much of the architectural features as possible. The details regarding the three different architectures are 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 (19), and more details regarding its usage in our data collection have been described in the subsection *Tools*.

3.2 Application Workload

In our work (17), we had chosen *Luciferin* and *Ergosterol* molecules to test the GAMESS performance on a SMP cluster and a 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 3.1. 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 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.

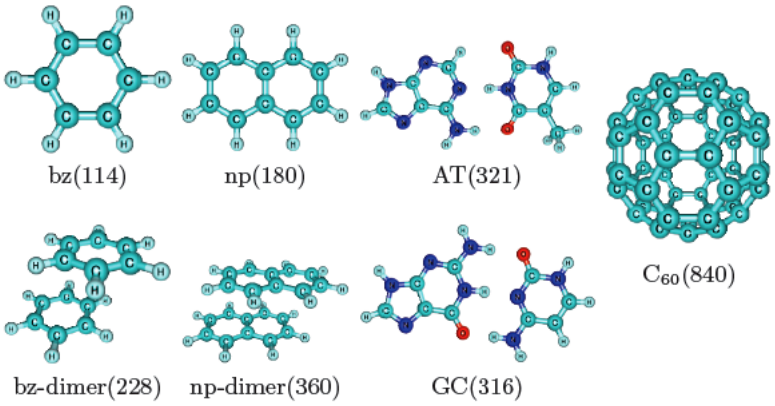


Figure 3.1 Hiro Inputs

3.3 Architectures

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 (26). 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) (10; 23). 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.

3.4 Tools

The adaptation scheme used by the authors Li et al. in (12) uses wall clock time while the adaptation mechanism described in (29) by Ustemirov et al. 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 with a 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 (19) (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.

TAU is a program and performance tool framework that provides a suite of static and dynamic tools for parallel Fortran, C++, C, Java and Python applications.The TAU framework can be divided into a portable profiling package and a code analysis package. The profiling package model maintains performance data for each thread context, and node in use by a parallel, multi-threaded application. TAU's profile analysis procedures generate useful information from the profile data collected. This data includes exclusive and inclusive time spent in each function with nanosecond resolution. Other data such as the number of times each function was called, number of profiled functions invoked by each function, and mean inclusive time per call. Time information can also be displayed relative to nodes, contexts, and threads. Instead of time, hardware performance data can be shown. Also, user-level profiling is possible. The TAU code analysis package contains static analysis tools based on PDT (19) , a Fortran and C/C++ code analysis system. These tools support sophisticated views of program structure, incorporating the latest C++ language features such as templates, namespaces, and exceptions. Currently, the code analysis systems have been used to analyze C++ source to automatically generate TAU profiling instrumentation.

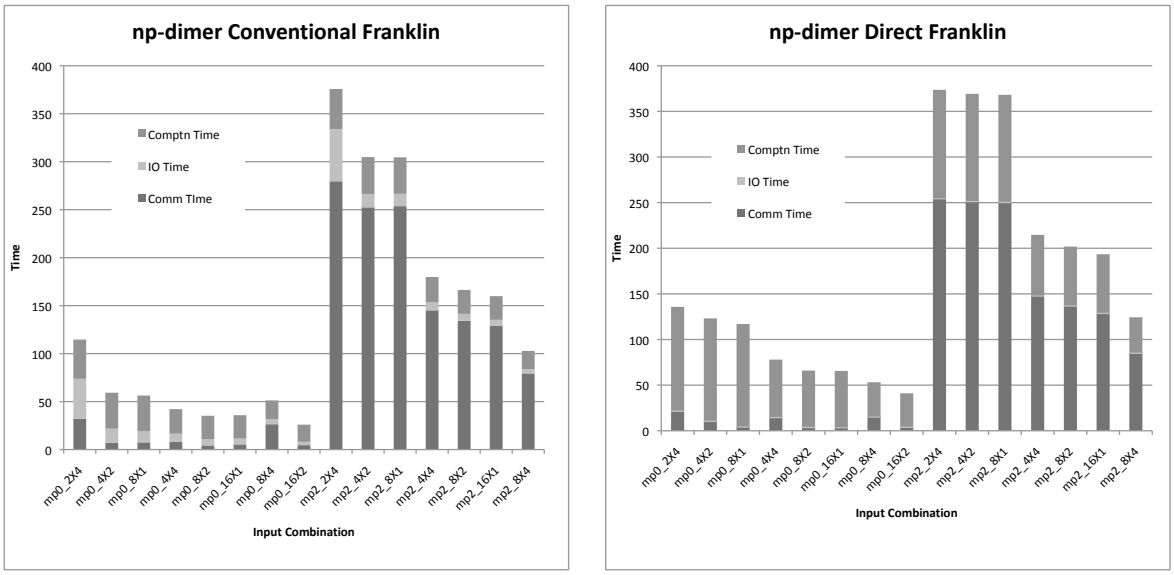
GAMESS was instrumented and profiled on all the three architectures. It is also important

to note that usage of the TAU profiler slows down the application considerably. In order to overcome this issue, the profiler was operated on functions that can be broadly classified as ones which provide communication, IO and computation. The profiler output contained data for each of these functions invoked by GAMESS. We wrote a separate program in C that collated the profiled data for each of these categories and gave a final value for the total time spent on communication, IO or computation.

3.5 Performance Results and Analysis

The performance data was collected for different combinations of the processors and GAMESS processes. We concentrated on 4 different input combinations. These were MP0 (Hessian), MP2 (Many body perturbation theory) computations using *direct* and *conventional* executions. 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 np-dimer results have been shown in Figures 3.2, 3.3, 3.4 and 3.5. The C60 results have been shown in 3.6, 3.7 and 3.8. 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.

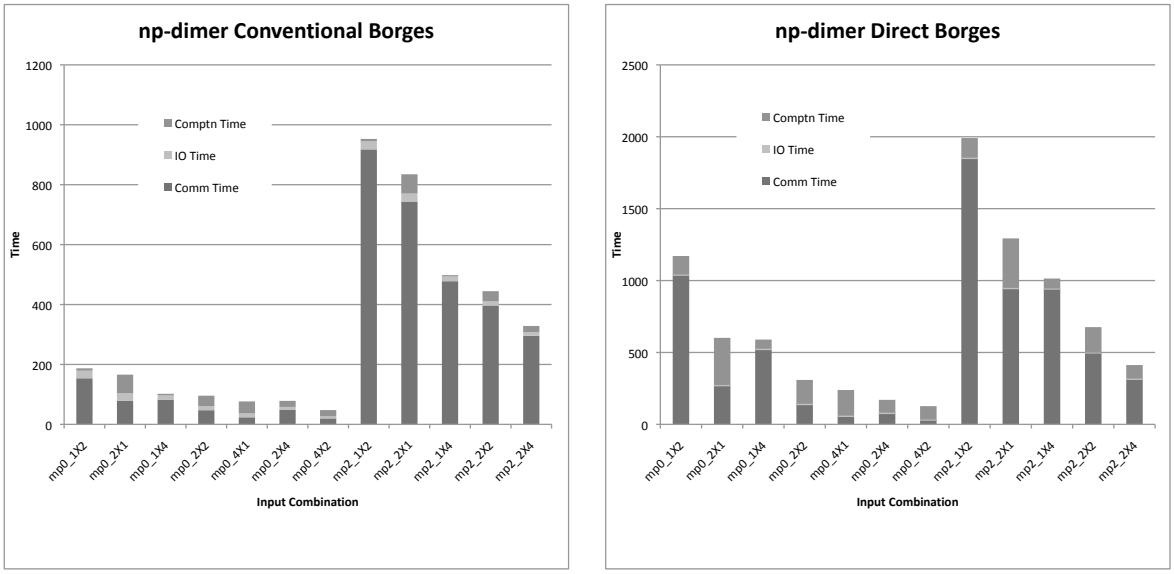


(a) (b)

Figure 3.2 (a)np-dimer conventional molecule results on Franklin (b)np-dimer direct molecule results on Franklin

From the results shown in Figures 3.2, 3.3, 3.4 and 3.5, 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 (17)), the *direct* implementation is faster. This difference is exploited in the existing adaptive mechanism implemented through NICAN discussed by Ustemirov et al. in (29). It was established in this paper 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.

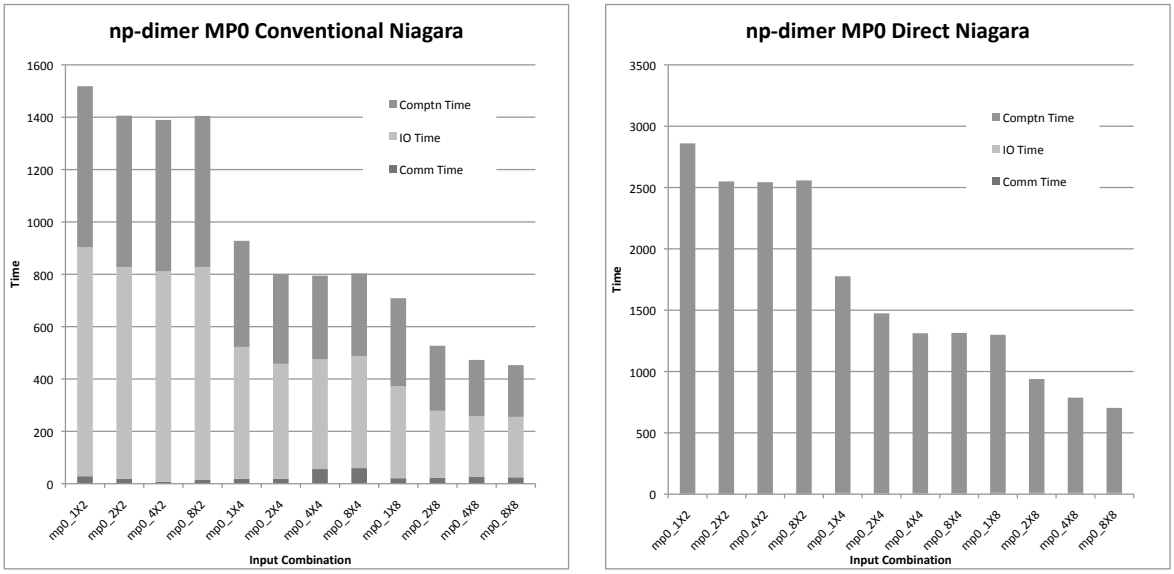
From Figures 3.2, 3.3, 3.4 and 3.5, it's clear that on all the three architectures, the total



(a) (b)

Figure 3.3 (a)np-dimer conventional molecule results on Borges (b)np-dimer direct molecule results on Borges

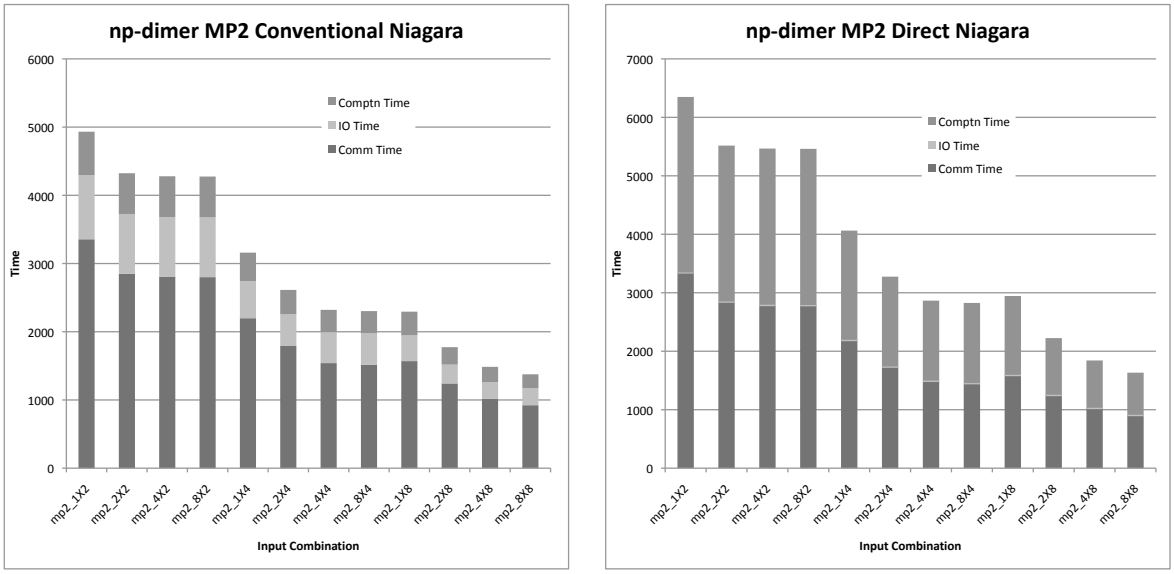
time taken for MP2 is at least 3 times higher than the time taken to complete MP0 calculations. In some cases, the MP2 time is nearly 10 times as high as the time taken to complete the MP0 calculations. On comparison of the three performance timings for MP0 and MP2, we can see that the I/O time and the 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.



(a) (b)

Figure 3.4 (a)np-dimer MP0 conventional molecule results on Niagara (b)np-dimer MP0 direct molecule results on Niagara

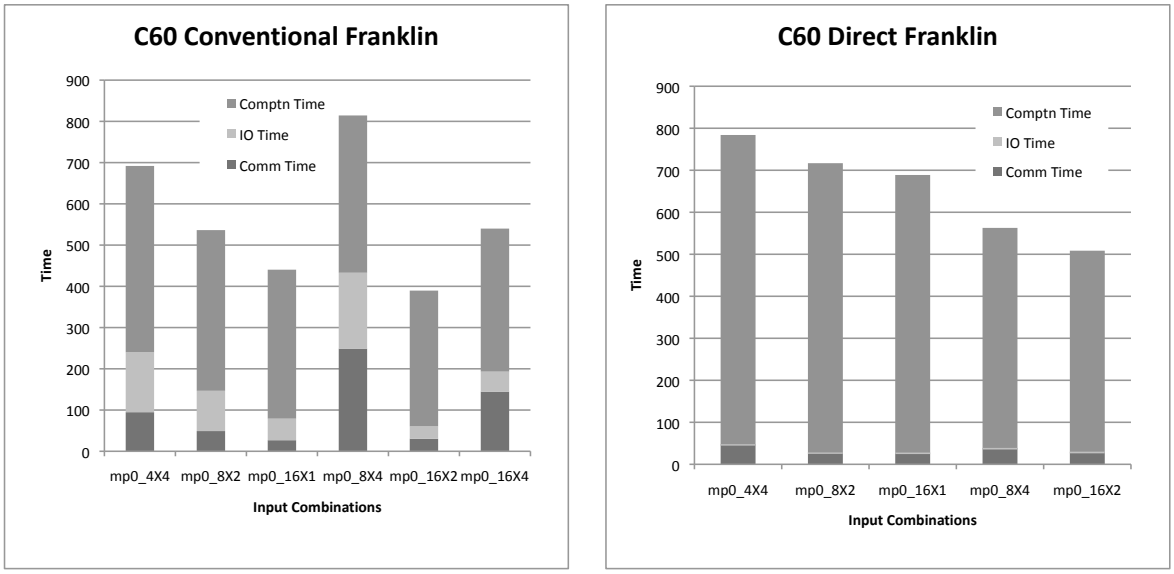
In order to develop other switching techniques for the GAMESS molecules, we need to deduce other application characteristic patterns through Figures 3.2, 3.3, 3.4 and 3.5. These figures are for the molecule np-dimer for different combinations of nodes and processes per node. On Franklin (figure 3.2), if we keep the total 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 3.6, which shows the results for the C60 molecule on Franklin. We found that C60 successfully executes only if at least 16 processes



(a) (b)

Figure 3.5 (a)np-dimer MP2 conventional molecule results on Niagara (b)np-dimer MP2 direct molecule results on Niagara

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 processes on a single node is increased 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

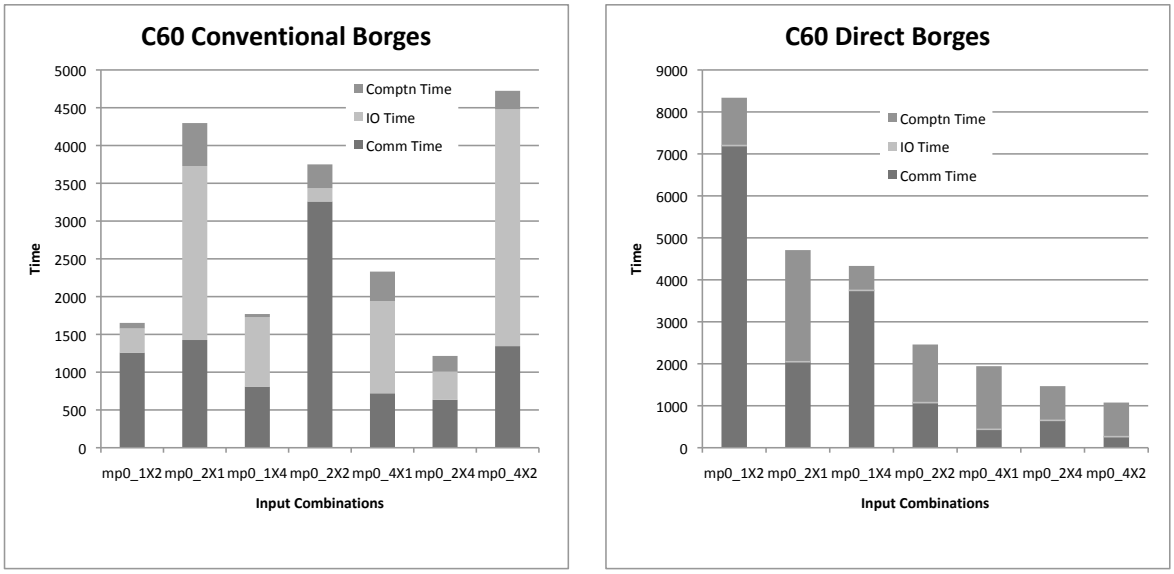


(a) (b)

Figure 3.6 (a)C60 conventional molecule results on Franklin (b)C60 direct molecule results on Franklin

well for I/O with large data size requests. 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 3.3, 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

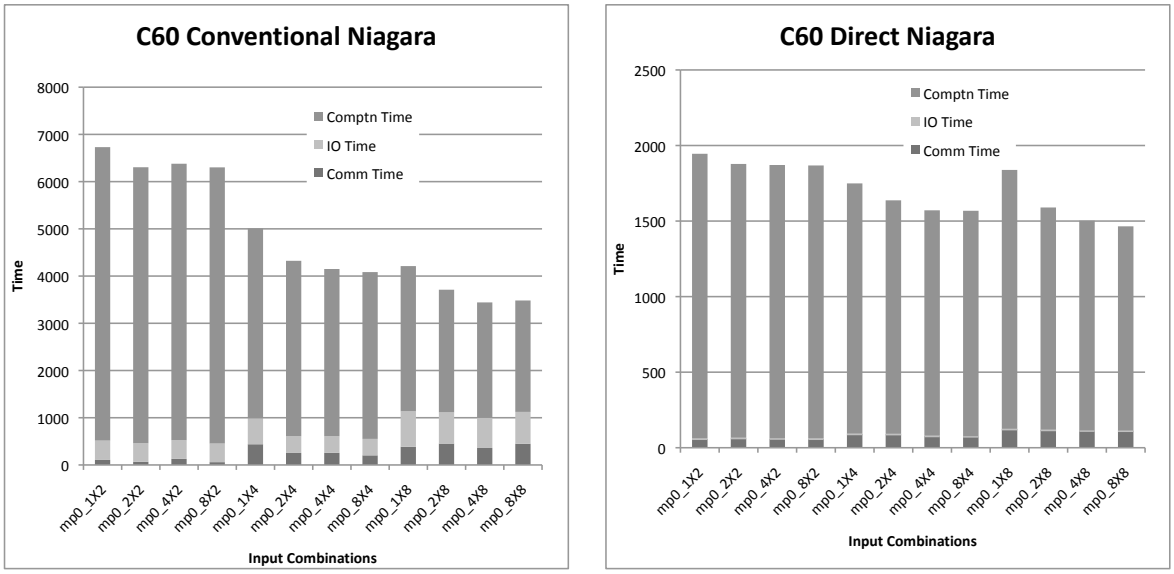


(a) (b)

Figure 3.7 (a)C60 conventional molecule results on Borges (b)C60 direct molecule results on Borges

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 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. Figure 3.7, shows that for larger molecules, the distribution of processes has a very good effect on the performance of GAMESS.

The observations from the results for np-dimer on the Niagara machine are shown in Figures 3.4 and 3.5. 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



(a) (b)

Figure 3.8 (a)C60 conventional molecule results on Niagara (b)C60 direct molecule results on Niagara

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. One more observation is that the best possible performance for a given number of threads is obtained when each GAMESS thread gets a single core to execute. From the results in Figure 3.8, we can see that Niagara gives very good performance when the cores are used for the *direct* computations. 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.

CHAPTER 4. DATABASE ASSISTED ADAPTATION AND RESULTS

4.1 Adaptation Architecture

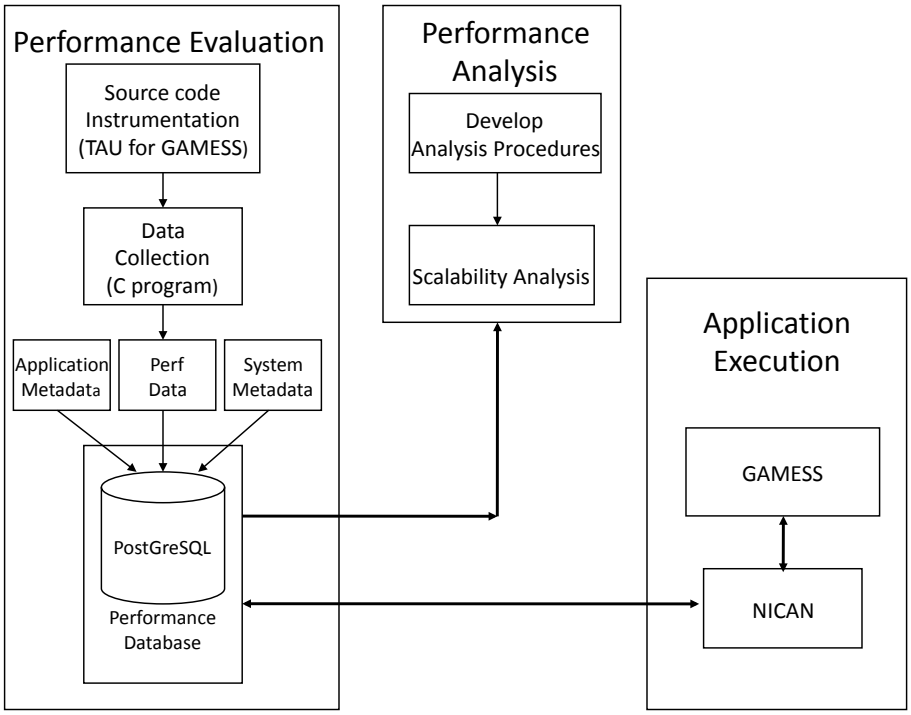


Figure 4.1 Database Adaptation Architecture

Figure 4.1 shows the complete architecture of the database adaptation framework. It is important to note that the data management as well as the adaptation are considered as part of

this framework. The adaptation architecture has been shown below. The architecture can be divided into two distinct sections. One is the offline section which consists of the performance evaluation and analysis of the application. In this section, we instrument the GAMESS code using TAU, collect the performance data for different combinations of the application and system parameters and store the data into a *PostGreSql* (3) performance database. This data is then analyzed by separate programs to obtain such metrics like scalability on a particular machine. The second section of the architecture is the application execution which involves the application adaptation using the NICAN middleware tool.

4.2 Adapatation Strategy

In the tests conducted for different input combinations of the two molecule sets, we concentrated on combining the direct and conventional implementations with MP0 and MP2 computations for all these 13 molecules. The TAU performance tool split the wall clock time as communication time, computation time and I/O time. This allowed us to broadly generalize the performance trends for the selected molecules and the input parameters. However, the amount of data generated for these 13 molecules on the three different architectures for different input combinations clearly showed a need for using full fledged performance scenarios in the adaptation process. Consider an example of a single molecule. We conducted performance tests for four different input combinations (MP0 and MP2, *direct* and *conventional*, for RHF computation type). These tests were run on 3 different architectures. On each architecture, we performed at least 8 different test runs for different combinations of input processors and nodes. This gives rise to 96 different sets of performance data for only a single molecule. The data sets would increase even further if we get the data for the other SCF implementations like UHF and ROHF or if we modify the run type from energy to optimize, gradient, hessian etc. Thus, the amount of data that can be generated is huge. The challenge is to ensure that the collected data is in a format that is easily analyzed to discover the performance issues and decide on strategies for consistent good performance. Also, all the performance data should be stored in such a manner that any developer or user will be able to manage this data easily.

Hence, it was decided that a database containing the relevant data would be integrated with GAMESS-NICAN.

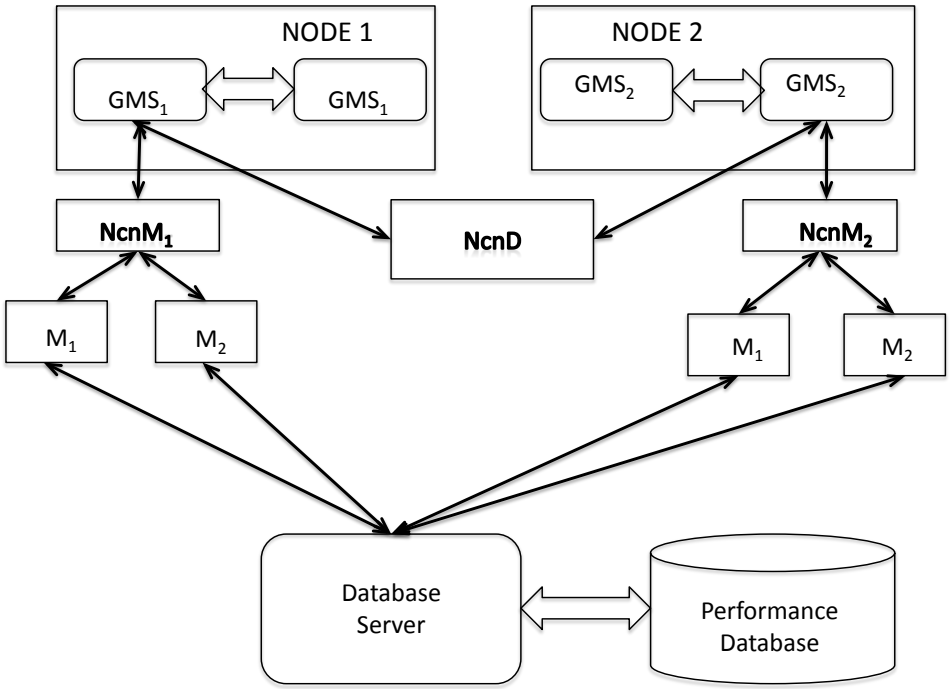


Figure 4.2 Database Connection Architecture

We chose *PostGreSql* database for storing the application metadata and the adaptation related data. The data to be stored in the database can be divided loosely into tables representing the data for performance analysis and data for adaptation. The tables used for adaptation are given in the appendix. The connection mechanism of GAMESS jobs to the database is shown in Figure 4.2. The application connects to the database using a database daemon server (DBd) that is started by the application. The application ensures that only a single DBd is running for that particular machine. This is essential since we only have a single database instance for the

entire machine. The architecture is similar to a client-server architecture wherein the different application jobs running on the machine are the clients connecting to a single DBd server. The user provides the database connection string to the application through the existing NICAN XML file. This enables each user to connect to his own schema though the database daemon server will be the same for all the users. The other important detail provided in the file is the machine name. The database stores machine specific details like the cluster node names, node configurations, available memory and other performance related details such as the best node-processor combination. The data format of the data exchanged between the DBd and the NICAN module is predefined. The database class provided to the NICAN module contains all the functions required to extract the result from the database. One other advantage of using a DBd server is that different modules of the same application job can connect to the database at the same time. Currently, the database management is handled manually. The data regarding each molecule on the particular machine is either inserted by the programmer or gets inserted during runtime by the NICAN module. We intend to integrate PerfExplorer and PerfDMF with the current setup so that we can manage the data more easily. This has been done before by authors Li et al. in (12). Also, we can then input much more detailed TAU data into the database for usage by the NICAN module.

The tuning strategy that we propose on the basis of the results obtained augments the existing NICAN adaptation strategy. 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 have offloaded this information into the database. The amount of data being written by GAMESS into files depends on the input parameters chosen. For example, when the *gbasis* value is taken as CCQ, the amount of data written for the *conventional* method is very large. The external disk sizes on clusters are normally huge. However, it is possible that in case of small clusters, the disk size might get exceeded due to residual files. Hence, NICAN calculates the amount of space available to store integral files and in case sufficient space is not available, the implementation

can be modified to *direct*. For MP2 computations, we have observed that the shared memory availability is the most important factor that determines if the job execution is successful or not. Since, it does not make any sense to switch between MP0 and MP2, it would be useful for the users if they are provided with the exact input memory requirements for the job to execute successfully. NICAN compares the memory requirement and the memory requested by the user. The job is immediately stopped if the memory requested by the user is less than the memory required to execute the job. The correct value of DDI memory is printed in the log file. This ensures that the job is stopped before the start of the execution instead of failure at the MP2 calculation stage.

From the initial results obtained, we have seen the best combination to obtain the most efficient application performance on a given architecture. Such combinations are 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 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 not as good as desired, but we can get a decent speed up for large molecules by increasing the number of cores used for execution. The adaptation between *conventional* to *direct* is an existing feature in NICAN and we have not modified the algorithm. However, the database access has been incorporated as an easy extensibility to the NICAN features.

4.3 Database Framework Adaptation Results

The adaptation was tested on Borges using two representative molecules, AT and C60. The database was created on Borges and the necessary data was inserted into this database. The results have been shown in Figure 4.3. The Y-axis of the graph shows the combined execution time for both these molecules. The X-axis of the graph represents the input node-processor

combination. We can see that the performance improvement varies from around 28% to 54%. This improvement is mainly due to offloading the idealized iteration time to the database instead of computing it every time the adaptation has to run. The database entries in the table *Machine_Combination* were modified in such a way that the host configuration was not modified in these tests.

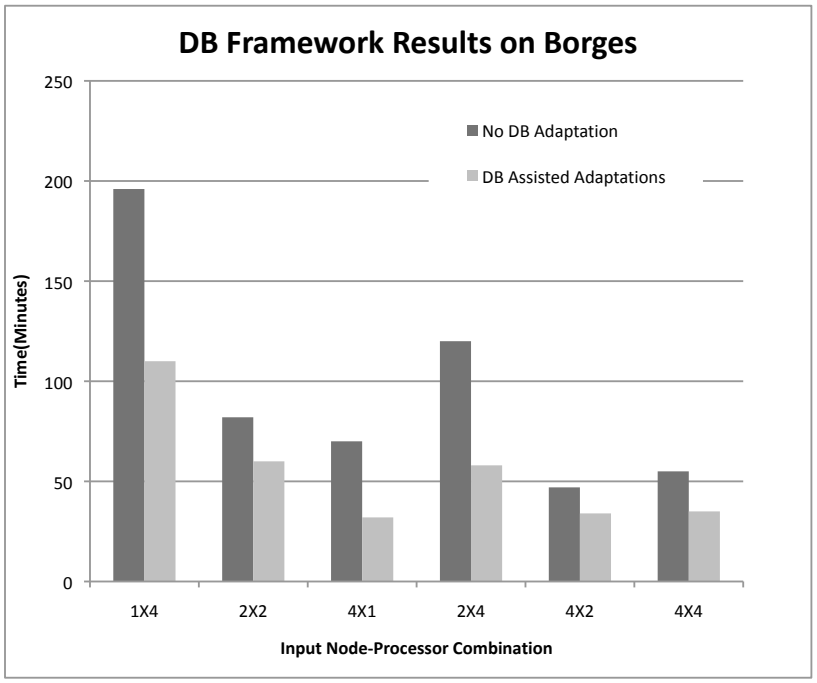


Figure 4.3 Database Adaptation Results

The host configuration modification would ensure that for a specific number of processes, the input node-processor configuration is always changed to a particular value. For example, if the user requests 4 processes, the best performance is obtained when the job is run on 4 nodes with each node running a single process. The results for host configuration modification are shown in Figure 4.4. We have also tested the other capabilities added to NICAN. The GAMESS adaptation was tested when the filesystem limit was reached. The filesystem limit was kept at 95% and the GAMESS execution was modified from *conventional* to *direct* when this filesystem limit was reached. The NICAN library was modified to check the database for

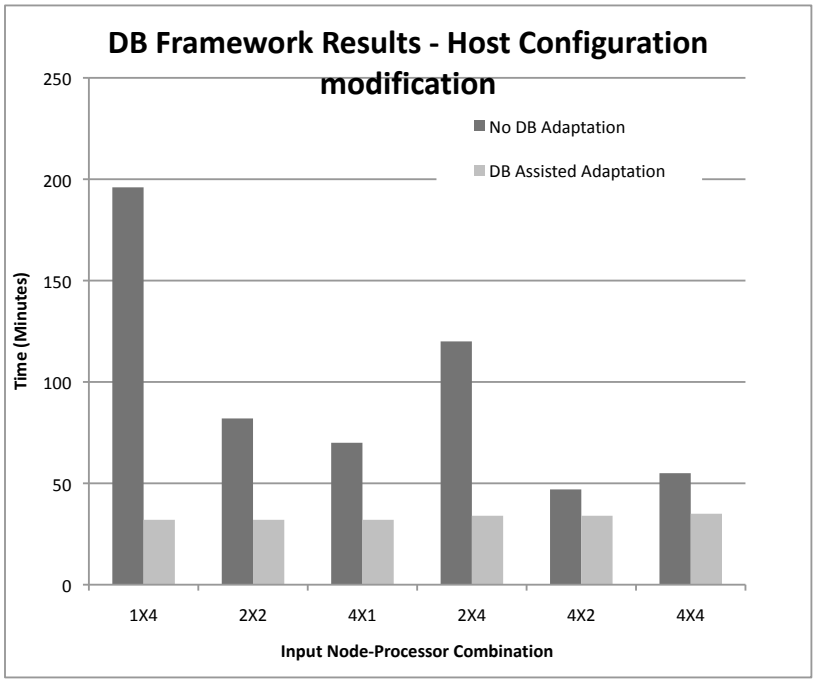


Figure 4.4 Database Adaptation Results with host configuration modification

MP2 memory requirements and then compare it with the memory requested by the user. The job is appropriately terminated if the memory requirements do not match.

4.4 Database Framework Scalability Analysis

The performance data collected can be utilized for different analyses and derive a variety of analysis results. One advantage of having a vast amount of performance data for an application such as GAMESS is to be able to analyze the scalability of the application on different architectures and deduce the inflection point for the performance degradation or for performance improvement. According to the Webster dictionary, Scalability is defined as the property of being easily expanded or upgraded on demand. With respect to any computer application, we can define it as the ability of any program to function correctly given that the application or its context is modified in size or volume. The re-scaling can occur either in terms of the application itself (Ability to create more threads, Ability to handle more users in case of servers)

or the context in which it is operated (Increase in hardware resources, upgrade of operating system etc). The scalability of an application also refers to the ability of the application to take full advantage of the modification that has occurred. If the factors such as the underlying hardware configuration and operating system parameters are the same, the performance of the application would depend solely on the number of threads that the application runs. However, if the number of cores or number of nodes is modified along with the application threads, the results give an indication about the ability of the underlying hardware architecture to improve application performance. With respect to GAMESS, the scalability analysis indicates the improvement in the performance of GAMESS on a particular architecture, when the number of cores or nodes, on which the application is executed, is increased.

The scalability of GAMESS on the three architectures is calculated using a separate C program. This program operates on the TAU results file; takes the molecule name and SCF type as input and outputs the scalability results in tabular format. Scalability is calculated as the *inverse* ratio of the total time taken by GAMESS on a given number of nodes and the total time taken by GAMESS for a single node, when the number of GAMESS processes per node is constant. A node can contain multiple cores and hence is capable of running multiple processes. In case the TAU results have not been obtained for a single node, we consider the denominator to be the minimum of the total time. The tables 4.1 , 4.2 , 4.3 , 4.4, 4.5 and 4.6 show results for the molecule AT on all the three architectures. The zero values indicate that results were not obtained for those particular combinations. Consider the following example to understand the scalability calculation. Table 4.3 shows the scalability of a conventional GAMESS job on Franklin. As per our definition, the scalability while using 16 nodes and a single process on each node, would be the *inverse* ratio of total time obtained by running a single process of GAMESS on 16 nodes each and the total time obtained by running a single process of GAMESS on a single node. Since there are no results available for the denominator, we use the minimum time available which in this case equates to the total time required to complete a GAMESS job on 8 nodes running a single process per node.

Cores Procs	1	2	4	8
2	1	1.06	1.06	1.05
4	1	1.17	1.24	1.25
8	1	1.35	1.59	1.67

Table 4.1 AT MP0 Conventional on Solaris

Cores Procs	1	2	4	8
2	1	1.14	1.14	1.14
4	1	1.20	1.34	1.34
8	1	1.36	1.63	1.82

Table 4.2 AT MP0 Direct on Solaris

t_N=Total run time of GAMESS on a given number of nodes.

t_1=Total run time of GAMESS for a single node.

$$Scalability = \frac{1}{\frac{t_N}{t_1}}$$

Consider the results shown in Tables 4.1 , 4.2 for the Niagara machine. Since the Niagara machine consists of a single node and 8 cores, the scalability is calculated with respect to the increase in the number of cores. We can see that as we increase the number of cores and keep the number of processes constant, the scalability does not increase similarly. The runtimes decrease as we increase the number of threads. However, the runtimes stabilize once the number of threads are equal to the number of cores on which they are run. Even if we increase the number of cores, the performance remains flat. Thus the best possible combination for running the GAMESS application would be to allocate individual threads to single cores.

The Tables 4.3 and 4.4, give the scalability results for Franklin. For the *conventional* run of AT on Franklin, the scalability improves by 60% when we move from 4 nodes to 8 nodes but a quadruple increase in the number of nodes (from 4 to 16) does not improve the performance 4 times. The scalability is only 2.2 times in this case. Also, the scalability actually reduces from 1.83 to 1.64 when we use 8 nodes to run 4 processes each instead of 4 nodes. Since AT is a small molecule, running 32 processes increases the communication delay leading to reduction in scalability. For the *direct* job execution of AT, the scalability looks good and the performance nearly doubles when the number of nodes is doubled. The quadrupling of the nodes does not have the same effect though it does provide a three time increase in the performance.

Nodes Procs	2	4	8	16
1	0	0	1	1.66
2	0	1	1.59	2.22
4	1	1.83	1.64	0

Table 4.3 AT MP0 Conventional on Franklin

Nodes Procs	2	4	8	16
1	0	0	1	1.86
2	0	1	1.95	3.42
4	1	1.77	3.08	0

Table 4.4 AT MP0 Direct on Franklin

Nodes Procs	1	2	4
1	0	1	2.24
2	1	2.26	4.67
4	1	1.95	0

Table 4.5 AT MP0 Conventional on Borges

Nodes Procs	1	2	4
1	0	1	2.54
2	1	3.86	9.63
4	1	3.56	0

Table 4.6 AT MP0 Direct on Borges

The tables 4.5 and 4.6 indicate the performance scalability on Borges, which is an internal cluster of Ames Lab. The doubling of the number of nodes doubles the performance for *conventional* and triples it in case of *direct*. These tables give us a good idea about the scalability on different machines through the usage of the total time taken by GAMESS jobs. We could use communication time, IO time or computation time instead of total time and get the same scalability analysis done. Also, this scalability computation may be integrated with NICAN to provide NICAN with scalability information for either monitoring or adaptation.

CHAPTER 5. CONCLUSIONS AND FUTURE WORK

Conclusions

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 in the application performance. We proposed to enhance the adaptation strategy of GAMESS so as to encompass different parameters that affect the application performance. We first understood the performance characteristics of GAMESS on different architectures. We conducted performance tests for 13 different molecules on three different architectures. MP0 and MP2 computations for *direct* and *conventaional* implementations were tested. By using the TAU tool, we split the total wall clock time as communication time, computation time, and IO time. By analyzing these data, we arrived at a few rules which would aid the adaptation process. The data analysis was a tedious and time consuming process due to the volume of data generated for only 13 molecules and 4 different input combinations. Hence usage of a performance database was mooted. The idea was to create a repository to hold this performance data such that this database can be used for both performance analysis and application adaptation. We have utilized a PostGreSql database to hold this data. The obtained performance data was analyzed and a set of rudimentary and simple rules were created to augment the existing adaptation process. A database framework was created around NICAN so as to access the postgresql database and aid GAMESS adaptation. The performance data is collected offline and then inserted into the database. The existing dynamic adaptation of GAMESS has not been modified. The performance improvement obtained by using the database framework has been shown up to 40%. A scalability analysis module has also been created which calculates the scalability of each architecture.

Future Work

1. It is not possible to store performance data and rules for all the molecules used by the chemists. Hence, as a future work, we envision that NICAN should be able to recognize the similarity of the given input molecule with the data available in the performance database and then approximate the required data.

2. Application related data such as compiler options or system related data such as cache performance data may be extracted from the application execution and stored in the database. This can help us to refine the adaptation strategy.

3. The performance data has been collected on different architectures. Each architecture has its unique features but they also have features which can be compared. Using the performance data, we intend to compare different architectures and also extrapolate the performance of GAMESS on other architectures.

4. Currently the code abstracts the underlying hardware to the level of the machine name. This can be extended to be more generic abstractions such as the underlying hardware architecture.

5. Currently the total runtime is being split into communication time, IO time, and computation time by the TAU tool. This granularity can be further increased so that we can get the data within a particular computational phase or even within an individual SCF iteration.

6. The NICAN dynamic adaptation algorithm can be modified such that the algorithm uses communication time or IO time, instead of just the iteration time for deciding on the switching between *conventional* and *direct*.

7. The dynamic adaptation algorithm is currently in use only for the SCF iterations. MP2 has different implementations that can be switched dynamically. Also, the DFT (Density

Functional Theory) implementation can be modified to use dynamic adaptations.

8. We are also concerned with the usage and management of the performance data that can be collected for a single molecule on different machines and different input combinations. We need to explore the usage of tools such as PerfDMF and PerfExplorer to obtain a dimension reduction of the performance data and for database management.

9. Analysis techniques such as machine learning techniques can be employed.

APPENDIX A. ADDITIONAL MATERIAL

GAMESS Capabilities

1. Calculates RHF, UHF, ROHF, GVB, or MCSCF self-consistent field molecular wavefunctions.

2. Calculates the electron correlation energy correction for these SCF wavefunctions using Density Functional Theory (DFT), Configuration Interaction (CI), Many Body Perturbation Theory (MP2), coupled-cluster (CC) or Equation of Motion CC (EOM-CC) methodologies.

3. Calculates semi-empirical MNDO, AM1, or PM3 models using RHF, UHF, ROHF, or GVB wavefunctions.

4. Calculates analytic energy gradients for any of the SCF or DFT wavefunctions, closed or open shell MP2, or closed shell reference-based CI.

5. Optimizes molecular geometries using the energy gradient, in terms of Cartesian or internal coords.

6. Searches for saddle points (transition states) on the potential energy surface.

7. Computes the energy hessian, and thus normal modes, vibrational frequencies, and IR intensities.

8. Obtains anharmonic vibrational frequencies and intensities (fundamentals or overtones).

9. Traces the intrinsic reaction path from a saddle point to reactants or products.

10. Traces gradient extremal curves, which may lead from one stationary point such as a minimum to another, which might be a saddle point.

11. Follows the dynamic reaction coordinate, a classical mechanics trajectory on the potential energy surface.

12. Computes excited state energies, wavefunctions, and transition dipole moments at various levels

    (a) SCF (e.g. ROHF or MCSCF)

    (b) CIS (RHF plus single excitations)

    (c) much more general CI functions

    (d) time dependent DFT

    (e) Equation of Motion-Coupled Cluster with analytic gradients for SCF, CIS, and GUGA CI.

13. Searches for the minimum energy crossing point between two such potential energy surfaces.

14. Evaluates relativistic effects, including scalar corrections, via 3rd order Douglas-Kroll transformations. Gradients are available. spin-orbit coupling matrix elements and the resulting spin-mixed wavefunctions.

15. Evaluates the molecular linear polarizability and the first and second order hyperpolarizabilities for all wavefunctions, by applying finite electric fields.

16. Evaluates both the static and frequency dependent polarizabilities for various non-linear optical processes, by analytic means, for RHF wavefunctions.Nuclear derivatives of the polarizabilities lead to analytic Raman and hyperRaman spectra, also for RHF.Imaginary frequency dependent polarizabilities can also be obtained, again for RHF only.

17. Obtains localized orbitals by the Foster-Boys, Edmiston-Ruedenberg, or Pipek-Mezey methods, with optional SCF or MP2 energy analysis of the LMOs.

18. Calculates the following molecular properties:

    (a) dipole, quadrupole, and octupole moments, electrostatic potential

    (b) electric field and electric field gradients

    (c) electron density and spin density

    (d) Mulliken and Lowdin population analysis, virial theorem and energy components

    (e) Stones distributed multipole analysis

19. Models solvent effects by

    (a) effective fragment potentials (EFP)

    (b) polarizable continuum model (PCM)

    (c) surface and simulation of volume polarization for electrostatics (SS(V)PE)

    (d) conductor-like screening model (COSMO)

    (e) self-consistent reaction field (SCRF)

20. Performs all-electron calculations based on the Fragment Molecular Orbital (FMO) method.

21. Models the formation of aperiodic polymers with the Elongation Method.

22. When combined with the plug-in TINKER molecular mechanics program, performs Surface IMOMM (SIMOMM) or IMOMM QM/MM type simulations. Download from `http://www.msg.ameslab.gov/GAMESS/GAMESS.html`.

23. When combined with the plug-in NEO program (Nuclear Electron Orbitals), performs quantum mechanics computations of nuclear structure. NEO's code is included with GAMESS source distributions, see the directory `~/gamess/qmnuc`.

24. When combined with the plug-in VB2000 program, performs valence bond calculations. See `http://www.scinetec.com/~vb` for more information.

25. When combined with the plug-in XMVB program, performs valence bond calculations. Please contact Professor Wei Wu of Xiamen University for more information, weiwu@xmu.edu.cn, and see also `http://ctc.xmu.edu.cn/xmvb/index.html`.

26. When combined with the plug-in NBO program, performs Natural Bond Order analyses. This program is available at `http://www.chem.wisc.edu/~nbo5`, for a modest license fee.

Database Tables

1. *Molecule* Contains details regarding the molecule.
2. *Checkrun_Details* Gives the information regarding the idealized iteration time.
3. *Machine_Details* Contains information regarding the nodes in a machine.
4. *Machine_Combination* Contains information regarding the best possible combination on a particular machine for a given number of application processes.
5. *Performance_Data* Contains TAU performance data for a particular molecule on a particular machine and input-processor combination.

BIBLIOGRAPHY

[1] G. Chen Providing Dynamic network information to distributed applications Master's Thesis, University of Minnesota Duluth, May 2001.

[2] J. Dongarra and V. Eijkhout Self-adapting Numerical Software for Next Generation Applications International Journal of High Performance Computing applications (IJHPCA) volume 17, Number 2, Pg 125-131, 2003

[3] K. Douglas and S. Douglas *PostgreSQL* , 2003 New Riders Publishing,Thousand Oaks, CA, USA,

[4] J.L. Hennessy and D.A. Patterson Computer Architecture, Fourth Edition: A Quantitative Approach Morgan Kaufmann Publishers Inc., 2006.

[5] H. Liu and M. Parashar 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 Pg 59–68, 2005

[6] F. Jensen Introduction to Computational Chemistry, Wiley, Chester, UK, 1999.

[7] C.L. Janssen, I.B. Nielsen, M.L. Leininger, E.F. Valeev, J.P. Kenny, E.T. Seidl . The Massively Parallel Quantum Chemistry Program (MPQC), 3.0, Sandia National Laboratories, Livermore, CA, USA, 2008.

[8] V. Kazempour, A. Fedorova and P. Alagheband Performance Implications of Cache Affinity on Multicore Processors. Euro-Par '08: Proceedings of the 14th international Euro-Par conference on Parallel Processing 2008,151–161, Springer-Verlag

[9] R. Kendall, E. Aprà, D. Bernholdt, E. Bylaska, M. Dupuis, G. Fann, R. Harrison, J .Ju, J .Nichols, J. Nieplocha, T.P. Straatsma, T. Windus and A. Wong High performance computational chemistry: An overview of NWChem a distributed parallel application In Journal of Computer Physics Communications, June 2000, No 1-2, Pages 260-283, Volume 128

[10] P. Kongetira, K. Aingaran and K. Olukotun A 32-way Multithreaded SPARC(R) Processor. In IEEE Micro, Volume 25, Number 2, 2005, Pg 21–29.

[11] 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 436–450. Springer, 2003.

[12] L. Li, J. Kenny, M.S. Wu, K. Huck, A. Gaenko, M.S. Gordon, C.L. Janssen, L.C. McInnes, H. Mori, H.M. Netzloff, B. Norris, T.L. Windus. Adaptive Application Composition in Quantum Chemistry Proceedings of The 5th International Conference on the Quality of Software Architectures (QoSA 2009) February 2009

[13] R. McDougall and J. Mauro. Solaris Internals: Solaris 10 and OpenSolaris Kernel Architecture. 2nd Edition, Prentice Hall, 2006.

[14] J. Nieplocha, V. Tipparaju, M. Krishnan, and D. Panda High Performance Remote Memory Access Communications: The ARMCI Approach. International Journal of High Performance Computing and Applications, Vol 20(2), 233-253p, 2006.

[15] R. Olson, M.W. Schmidt, M.S. Gordon and A.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.

[16] M. Schmidt, K. Baldridge, J. Boatz, S. Elbert, M.S. Gordon, J. Jensen, S. Koseki, N. Matsunaga, K.Nguyen, S. Su, T. Windus, M. Dupuis, J. Montgomery,Jr. General Atomic and Molecular Electronic Structure System. *Journal of Computational Chemistry*, 14, 1347-1363(1993).

[17] L. Seshagiri, M. Sosonkina, Z. 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 Pg 3–12, Lecture Notes In Computer Science; Vol. 5544, Springer-Verlag

[18] L. Seshagiri, M.S. Wu, M. Sosonkina, Z. Zhang Exploring Tuning Strategies for Quantum Chemistry Applications In Proceedings of 2009 International Workshop on Automatic Performance Tuning (iWAPT-2009), Tokyo, Japan, Oct 1-2, 2009

[19] S. Shende and A. Malony The TAU parallel performance system Int. J. High-Perf. Computing Appl., ACTS Collection special issue 20 (Summer 2006), Pg 287-331.

[20] 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.

[21] M. Sosonkina, S. Storie. Parallel performance of an iterative method in cluster environments: an experimental study. In *Proceedings of Parallel Matrix Algorithms and Applications (PMAA 2004)*, Marseille, October 2004.

[22] S. Storie. Aspects of Communication Subsystem Analysis for Distributed Scientific Applications. Master's Thesis, University of Minnesota Duluth, May 2004.

[23] Sun Microsystems Inc. http://www.sun.com/processors/UltraSPARC-T2/.

[24] C. Ţăpuş, I. Chung and J. Hollingsworth Active Harmony: Towards Automated Performance Tuning Supercomputing '02: Proceedings of the 2002 ACM/IEEE conference on Supercomputing 2002 , Pg 1–11

[25] V. Taylor, X. Wu, R. Stevens Prophesy: an infrastructure for performance analysis and modeling of parallel and grid applications, SIGMETRICS Perform. Eval. Rev., volume 30, number 4, pages 13–18

[26] C. Terboven, D. Mey, S. Sarholz OpenMP on Multicore architectures. In IWOMP '07: Proceedings of the 3rd international workshop on OpenMP A Practical Programming Model for the Multi-Core Era, Lecture Notes in Computer Science, Springer, Berlin Heidelberg, 54-64 (2008).

[27] H.-J. Werner, P. J. Knowles, R. Lindh, F. R. Manby, M. Schütz and others. MOLPRO, version 2008.3, a package of ab initio programs

[28] E.H. White, F. Capra, W.D. McElroy. The Structure and Synthesis of Firefly Luciferin *J. Am. Chem. Soc.*, 83(10), 2402-2403(1961).

[29] N. Ustemirov, M. Sosonkina, M.S. Gordon and M.W. Schmidt Dynamic Algorithm Selection in Parallel GAMESS Calculations. ICPPW '06: Proceedings of the 2006 International Conference Workshops on Parallel Processing 2006, Pg 489–496

[30] N. Ustemirov, M. Sosonkina, M.S. Gordon, M.W. Schmidt. Concurrent Execution of Electronic Structure Calculations in SMP Environments. In Proceedings 2005 Spring Simulation MultiConf, High Performance Computing Symposium, J.A.Hamilton Jr. et al, Soc. for Modeling and Simulation Internat. , San Diego, CA. 2005

[31] N. 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.

[32] R. Vuduc, J.W. Demmel and J.A. Bilmes Statistical Models for Empirical Search-Based Performance Tuning Int. J. High Perform. Comput. Appl., 2004, pg 65–94

[33] M.S. Wu, J.L. Bentz, F. Peng, M. Sosonkina, M.S. Gordon, R.A. Kendall Integrating Performance Tools with Large-Scale Scientific Software. In Proceedings of IEEE International Parallel and Distributed Processing Symposium, 2007. (IPDPS 2007). Pg 1–8