Dynamic Algorithm Selection in Parallel GAMESS Calculations [*]

Nurzhan Ustemirov
Masha Sosonkina
Department of Computer Science
Ames Laboratory
Iowa State University
Ames, IA 50011 USA
{nurzhan,masha}@scl.ameslab.gov

Mark S. Gordon
Michael W. Schmidt
Department of Chemistry and
Ames Laboratory
Iowa State University
Ames, IA 50011 USA
{mark,mike}@si.fi.ameslab.gov

Abstract

Applications augmented with adaptive capabilities are becoming common in parallel computing environments which share resources such as main memory, network, or disk I/O. For large-scale scientific applications, dynamic adjustments to a computationally-intensive part may lead to a large pay-off in facilitating efficient execution of the entire application while aiming at avoiding resource contention. Application-specific knowledge, often best revealed during the run-time, is required to initiate and time these adjustments. In particular, General Atomic and Molecular Electronic Structure System (GAMESS) used for ab initio *molecular quantum chemistry calculations has two different implementations of Self-Consistency Field (SCF) methods, each of which targets either disk I/O or memory. This paper describes a mechanism enabling switching of algorithms during GAMESS run-time and shows the effect of the adaptations on the performance of GAMESS calculations as well as on a parallel GAMESS execution for different resource availability. The test results indicate that, in the presence of I/O resource contention, parallel GAMESS enhanced with adaptive mechanism may sustain the performance similar to that of full resource availability.*

1. Introduction

The General Atomic and Molecular Electronic Structure System (GAMESS) performs *ab initio* molecular quantum chemistry calculations [6] to perform a wide range of Hartree-Fock (HF) wave function (RHF, ROHF, and UHF) calculations. Using Self-Consistent Field (SCF) method, GAMESS iteratively approximates solution to the Shrödinger equation which describes the basic structure of atoms and molecules. There are two different implementations, *direct* and *conventional*, of the SCF method in GAMESS. The *direct* algorithm recomputes integrals "on-the-fly" for each iteration mainly consuming physical memory and CPU resources. Conversely, the *conventional disk-based* algorithm calculates integrals once, stores them on disk, and reuses during iterations resulting in a heavy disk I/O usage.

1.1 GAMESS in distributed environments

Many GAMESS calculations have parallel implementations which enable the utilization of distributed resources, such as main memory and disk storage. Remote data access, however, may appear as a limiting factor to GAMESS scalability. To avoid this problem, GAMESS native communication layer (called Distributed Data Interface (DDI)) takes advantage of shared memory on symmetric multiprocessor (SMP) nodes [5].

We have observed [12] that a parallel job in which computing processes are *equally distributed (scattered)* over different nodes runs faster than the one using multiple processors within the same SMP node. This could be due to the inability of certain communication layer implementations to efficiently handle shared-memory access [8, 2]. On the other hand, by using a single processor per node (*scattering*), each process utilizes the networking hardware without waiting for CPU or memory. In addition, we have observed that GAMESS *conventional* parallel job on two processors of the same SMP node may be actually *slower* compared to its sequential counterpart. The poor performance could be the result of disk I/O channel bottleneck due to simultaneous access by more than one *conventional* processes constituting a parallel job. On the other hand, running concur-

[*]This work was supported in part by the U.S. Department of Energy under Contract W-7405-ENG-82, in part by National Science Foundation under NSF/CHE-0535640 grant, in part by Minnesota Supercomputing Institute, and in part by the University of Minnesota Duluth.

rently scattered parallel GAMESS jobs with only one designated as *conventional* SCF algorithm on the same set of SMP nodes, has shown rather good performance. Thus, in parallel environments, the choice of an SCF algorithm has significant effect on performance. Similar to the study of concurrent GAMESS processes [11, **?**], a particular SCF algorithm was selected at the *start* of execution such that it does not compete with the peer GAMESS calculations already running in the system. It has been also assumed that the parallel environment is dedicated to running peer GAMESS calculations (i.e., those GAMESS applications which were integrated with a helper-middleware). For realistic situations, this assumption may be too restrictive since computational resource availability often changes in the course of application execution. Thus, additional *run-time* adaptations may be required to maintain good efficiency of execution.

This is especially true in a typical multiuser cluster environment, where users could simultaneously run a variety of high performance applications. Thus, adaptations made only in the preprocessing stage may not always lead to an improvement under such system conditions, and thus dynamic adaptations are desirable.

Modifying GAMESS source code to insert adaptations is not a feasible option. The source codes of quantum chemistry algorithms are already complex, so increasing their complexity may be detrimental to their correctness and usability. The latter is of particular concern since GAMESS is designed to run on a given computational platform by an application scientist without the knowledge of computer science. Therefore, employing some generic middleware tools, which may access the dynamic system conditions and invoke GAMESS adaptations – via a separate function handler – may be beneficial, assuming that such tools are lightweight and may be turned on/off by the user in a simple manner. In this work we present a generic integration model for applications, such as GAMESS, with dynamically interchangeable algorithms. The integrated middleware tool monitors system resources, analyzes job performance and invokes adaptation handlers to improve resource utilization and application performance.

The paper is organized as follows. Section 2 explains the dynamic selection of GAMESS algorithms while Section 3 presents its implementation using the middleware tool NICAN, which has been integrated with GAMESS. In Section 4, we show test results for the integrated software. The work is summarized in Section 5.

2 Possibility of dynamic switching in the SCF method

The SCF algorithm is one of the computationally-intensive parts of an electronic structure calculation that GAMESS performes. Thus, a proper selection of its implementation has a considerable effect on the overall calculation. Figure 1, modified from [3], sketches the SCF algorithm [3, 6]. The main difference between the two implementations is the handling of the two-electron (*2-e*) integrals. In the *direct* SCF method, the *2-e* integrals are recalculated for each iteration. This method has high demand for CPU time but avoids the possible disk I/O bottleneck. In the *conventional* SCF method, the *2-e* integrals are calculated once at the beginning of the SCF process (box with dashed border in Figure 1) and stored in a file on disk for subsequent iterations. This implementation performs best on a system where the available physical memory exceeds the file size for *2-e* integrals, so that the file is cached in physical memory. Thus, re-reading buffered *2-e* integrals is faster than recalculating them as in the *direct* method.

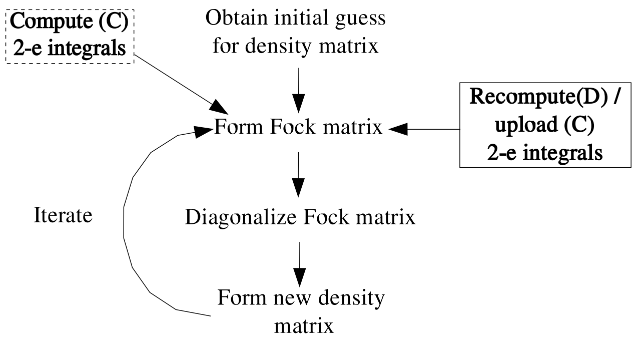

Figure 1. Illustration of the SCF algorithm as *conventional* (C) and *direct* (D) implementations.

Since the SCF algorithm (Figure 1) is of iterative nature, switching between *conventional* and *direct* implementations may be possible in an arbitrary SCF iteration, although the precise implementation details of the switching are rather complex. The following is a brief outline of the procedure. One may pre-compute the *2-e* integrals before commencing the iteration cycle, as for the *conventional* SCF algorithm. Then at each new iteration, the *2-e* integrals are either fetched from a file or (partly) recomputed for the *conventional* or *direct* implementations to be used depending on the resource availability. In a similar fasion, any quantum computation algorithm that may be implemented to recompute certain large quantities, as *2-e* intergals, may be a candidate for the algorithm selection technique presented in this paper.

3 Using middleware with GAMESS

Following our previous work [11, **?**, 12], we have chosen the middleware NICAN [7] to guide the toggling between the two implementations of SCF algorithm in GAMESS.

3.1 NICAN middleware tool

The main idea of integrating NICAN with an application is to decouple the *system-related* decision making from the execution of the application, while timely invoking application adaptation functions (handlers) [9]. The NICAN engine is encapsulated into a separate thread, Manager which controls the functional modules and invokes application adaptations. Due to dynamically loadable modules, NICAN is versatile and may have a wide variety of interactions with the system or application. Each module is designated to perform a separable function, such as to determine a system runtime characteristics or to validate machine-dependent parameters. NICAN has a rather general and flexible interaction mechanism, which permits to "talk" to a variety of application codes [7, 9]. Enhanced with "general-use" modules, such as CPU monitoring or disk I/O checking, NICAN may not require customized integration with an application. However, to explore application-specific trigger conditions, "specific-use" NICAN modules may also be needed.

An attractive feature of NICAN is that it does not require substantial coding modifications to the high-performance application with which it is interfaced. In the case of GAMESS, we have only a few changes to the source code. Specifically, the changes made to GAMESS enables it to dynamically switch between two SCF algorithms. Whereas resource monitoring, analysis and triggering adaptive mechanism is implemented within the NICAN engine.

3.2 Integration model for dynamic adaptations

The GAMESS package consists of two main parts. The first one, written in C, handles the setup of DDI needed for efficient memory management in parallel executions [5]. The second part, the legacy code written in Fortran77, does the actual computations. The initial GAMESS-NICAN integration was designed to encapsulate the entire GAMESS computation code [12]. A high-level approach was considered for adaptation: Adjust GAMESS *input* settings based on the system conditions and on the settings of the "peer" GAMESS-NICAN jobs co-existing in the system. Thus, NICAN Manager made changes to the GAMESS input parameters by selecting appropriate SCF algorithm at the preprocessing stage. The design assumed that the system is dedicated to run only GAMESS jobs. However, the assumption is too restrictive for modern realistic computing environments when e.g., several applications may reside on the same SMP node or when the I/O channel is shared by many applications.

Figure 2 depicts the GAMESS-NICAN integration for

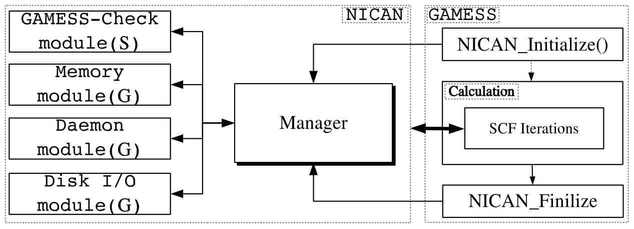

Figure 2. GAMESS-NICAN Integration

the GAMESS adaptations to be performed at runtime (shown as bold-faced double arrow). These adaptations are implemented as changes to iteration start parameters and result in switching from one algorithm to the other between SCF iterations. The GAMESS-NICAN integration model (Figure 2) includes both specific- `(S)` and general-use `(G)` modules. GAMESS-Check specific module, utilizing the special execution mode of GAMESS, calculates required memory for the given job and makes proper memory parameter adjustments to the GAMESS input file [11]. Memory module monitors available physical memory on the node. By performing light weight quick benchmarking, disk I/O module checks the available I/O resources. 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 among distributed NICAN processes. The daemon is self-contained and is independent from the job for which it has been started. The daemon performs standart operations such as: read, write, delete and self-distroys if there are no records. Each record consists of a process ID and a description to the job. Thus, the Manager can record any usefull 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.

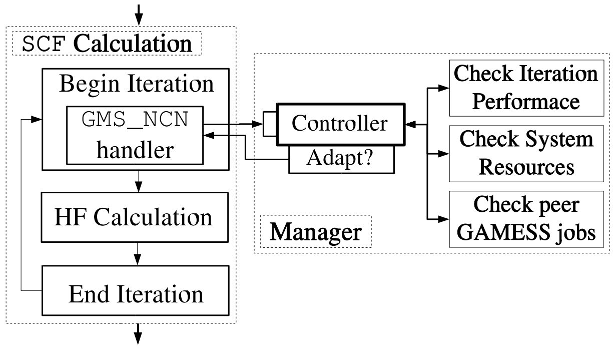

Figure 3. GAMESS adaptation scheme

Figure 3 details how the interaction between the NICAN

Manager and GAMESS SCF calculation is implemented. At the beginning of an SCF iteration, GAMESS calls the adaptation handler *GMS_NCN* the function of which is to communicate with the NICAN manager and to contain the adaptation decisions, if any, as returned by the NICAN manager. The decision is made by Manager's controller, which analyzes job's past performance, system resource information collected my modules and the state of peer GAMESS jobs as obtained from NcnD daemons. While a precise collibration of the decision making is of paramount importance, in this paper, we concentrate mostly on the toggling abilities of the SCF calculations and their implementation using middleware. In Section 4, we describe our experiments with only one possible adaptation control option, leaving its broader investigation as a future work.

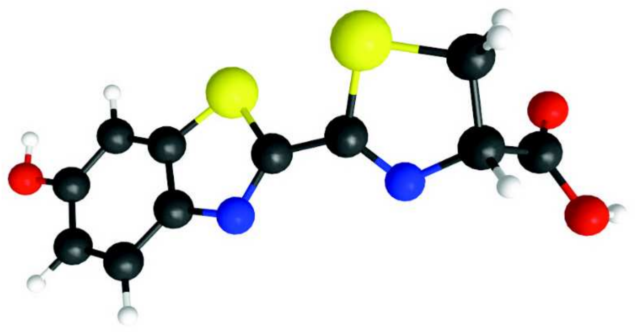

Figure 4. Luciferin molecule structure obtained by GAMESS RHF calculation

4 Experiments

We consider the behavior of parallel GAMESS calculations in a dynamically changing computing environment. In particular, the computing platform is shared by the GAMESS calculations integrated with NICAN and other applications that burden the resources critical for GAMESS execution. To study the adaptive features of the SCF algorithm in GAMESS which, in the *conventional* mode, requires much disk I/O, another application competing for this resource has been introduced during GAMESS runtime. The *iozone* benchmarking tool [4] has been taken as such an application. Although this tool may be used for measuring a variety of operations on files, we are interested in its ability to congest the disk I/O channel by writing to disk the files of a particular size, and thus occupying a certain percentage of the I/O channel.

The tests were executed on the Tools cluster owned by the SCL group at the Ames Laboratory. This cluster has eight SMP nodes, one half of which are 2.2 GHz Intel XEON dual-processor nodes and the other half is 1.7 GHz AMD Athlon MP 2100+ dual-processor nodes. All nodes have 768 MB of physical memory. Each XEON node has its own scratch directory mounted on a 250 GB 7200 RPM hard drive. On the other hand, the hard drives for the AMD nodes have varying characteristics. To correlate the performance of GAMESS with the I/O bandwidth consumed by *iozone* and to exclude the effects of processor differences, all the experiments were performed on the XEON nodes only.

As a test problem, we consider the computation of the luciferin molecule structure using the RHF energy calculation. In Figure 4, the molecule structure is plotted using MacMolPlt program [1, 10]. For luciferin energy, GAMESS converges in 15 iterations with *conventional* SCF algorithm requiring to store files of at most 3.57 GB or with *direct* SCF algorithm consuming 5.65 MB of main memory as estimated by the GAMESS-NICAN Check module (Figure 2). By default, GAMESS is given the conventional mode on input.

In the first set of experiments, we compare the performances of the original GAMESS with conventional mode ($GAMESS_{orig}$), i.e., GAMESS without any kind of adaptation, and of GAMESS in which NICAN triggers a dynamic selection of the SCF algorithm. The latter is denoted as $GAMESS_{dyn}$. Each job was executed on two processors, one per SMP node, such that any GAMESS process may not share the (same SMP) I/O channel with another process of the same parallel GAMESS calculation. At the GAMESS runtime, *iozone* starts to introduce a certain amount of I/O Congestion (denoted here as C_{io}) by writing a large file to disk on one of the SMP nodes to which GAMESS calculation is mapped. Since parallel GAMESS algorithms synchronize at certain times, the C_{io} perceived on an I/O channel may affect the execution of all the GAMESS processes.

Figure 5 depicts the time per SCF iteration. Each curve represents the GAMESS performance when different percentages, C_{io}, of I/O bandwidth are consumed by *iozone*, which starts at the point of the fourth GAMESS iteration. (Note that iteration four was chosen arbitrarily to start I/O congestion at runtime. If the I/O congestion is present at the very start of GAMESS calculation than the *preprocessing* adaptation of GAMESS, described in [12] and denoted in this section as $GAMESS_{prep}$, will detect it and adjust the SCF algorithm accordingly.) Until the fourth iteration, $GAMESS_{orig}$ and $GAMESS_{dyn}$ have the same performances. Starting at the fourth iteration, the execution time for the next SCF iterations has increased, depending on C_{io}. In $GAMESS_{dyn}$ (solid lines in Figure 5), NICAN monitors the time to perform an SCF iteration and compares it with the times to perform the previous two iterations. If the time has first increased by 10% and then either keeps constant or increases further, the adaptation handler is invoked on the SCF algorithm and is toggled from conventional to direct. In other words, this adaptation control strategy is based on the window of three iterations and on the increase of 10%. In the present experiments, these parameters have been dictated mainly by the type of molecule and of GAMESS calculation considered. In addition, the architecture parameters, such as the I/O bandwidth and the main memory size, may need to be considered, which is left outside of the scope of this paper. The dashed lines in Figure 5 show the performance of $GAMESS_{orig}$, while the solid lines reflect the change in the course of GAMESS execution at the sixth iteration when $GAMESS_{dyn}$ is used. For any C_{io}, the performance of $GAMESS_{dyn}$ is almost indistinguishable from the "no-congestion" case (denoted with circles in Figure 5) and it may be more than *two times* better than when $GAMESS_{orig}$ is used with a I/O bandwidth fully consumed.

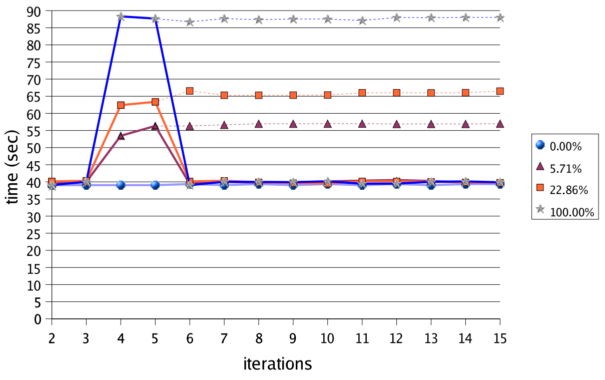

Figure 5. Two-processor execution of parallel GAMESS job for different amounts of I/O congestion, C_{io}

The aim of the second set of experiments is to observe the multiple parallel peer GAMESS calculations executing concurrently in the same computing environment that experiences congestion of I/O channels. Two jobs of the same GAMESS calculation using three adaptivity settings — non-adaptive ($GAMESS_{orig}$), at runtime ($GAMESS_{dyn}$), and only in the preprocessing stage ($GAMESS_{prep}$) — are executed on either two or four processors. Figure 6 shows the total time taken by two jobs for each adaptivity setting. In each pair, the jobs are started one after another with minimal delay. $GAMESS_{orig}$ jobs have finished as conventional resulting in a bad resource utilization caused by mutual competition, as described in [11]. With the I/O congestion, the performance of multiple $GAMESS_{orig}$ jobs deteriorates further (gray bars in Figure 6). Since GAMESS selects by default the conventional mode, this scenario is rather likely to happen in a multiuser cluster environment. When preprocessing is used, in $GAMESS_{prep}$ and $GAMESS_{dyn}$, once the presence of the peer GAMESS conventional calculation is detected for the job starting second, its SCF algorithm is switched to the direct mode. Thus, the $GAMESS_{prep}$ pair finishes with one conventional and one direct SCF algorithm (dark bars in Figure 6). The $GAMESS_{dyn}$ pair, however, is adapted further to the I/O channel congestion $C_{io} = 46\%$ by changing the algorithm of the job started first to the direct mode. (light-colored bars in Figure 6). Similar to the previous set of experiments, this change happens at the sixth iteration since the I/O congestion procedure and the control strategy are taken as in the first set of the experiments. We observe that dynamic adaptations of GAMESS calculations bring 10% to 15% improvement in the cumulative execution time of two jobs on four and two processors, respectively. We expect this improvement to hold for massively parallel GAMESS calculations performed on the computing platforms with few I/O channels, which are requested simultaneously by many GAMESS processes.

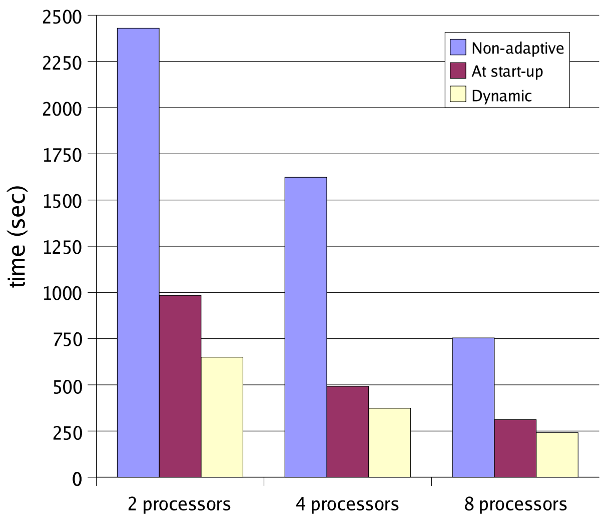

Figure 6. Total time to complete two simultaneous parallel GAMESS jobs for different adaptivity settings

5 Conclusions and future work

The main contribution of this work is to demonstrate the adaptive capabilities of GAMESS, a large-scale parallel legacy code, to dynamic changes in the vital system resources. In particular, we considered the effect of the disk I/O channel congestion on the conventional SCF algorithm when GAMESS was executed with and without adaptive features. As the result of adaptations, the SCF algorithm mode may be switched dynamically based on the current state of the multiuser computing environment. This switching as well as the analysis of system parameters and of the peer GAMESS jobs are performed by the NICAN middleware that is tightly integrated with GAMESS and controls the adaptation process. It has been observed that dynamically adaptive GAMESS calculations may be several hundred percent faster than non-adaptive ones. Dynamic adaptations of GAMESS also proved beneficial in all the test cases when I/O has been used by other applications, and thus switching of the SCF algorithm mode only once, at start-up, was insufficient.

Although our tests do not cover all the possible GAMESS calculations which use the SCF algorithm, the obtained results already indicate that GAMESS benefits considerably from an application-level middleware that facilitates its parallel execution. In the future, we envision to apply similar technique for post Hartree-Fock and non-SCF calculations as well as to develop multiple adaptation control strategies suitable for many GAMESS calculations. The obtained results may also be extended to various types of heterogeneity in computing platforms.

We envision that the proposed GAMESS-NICAN integration model may be easily reused, with only a few application-specific changes, for dynamic adaptations of a wide class of HPC applications. In particular, it may be most suitable for the applications trading heavy I/O for extra computations in an iterative fashion. Such an algorithmic trade-off may be readily exploited within the integration model at the run-time.

References

[1] B.M. Bode and M.S. Gordon. Macmolplt: a graphical user interface for GAMESS Journal of Molecular Graphics and Modeling, 16, 133-138 (1998).

[2] X. Chen, D. Turner. Efficient Message-Passing within SMP Systems. Recent Advances in Parallel Virtual Machine and Message Passing Interface, 10th European PVM/MPI conference, Venice, Italy, pg 286-293 (October 2003).

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

[4] W. D. Norcott and D. Capps. Iozone Filesystem Benchmark. `http://www.iozone.org`

[5] 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

[6] 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).

[7] M. Sosonkina. Adapting Distributed Scientific Applications to Run-time Network Conditions. In J. Dongarra, K. Madsen and Jerzy Wasniewski, editors, 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.

[8] M. Sosonkina, S. Storie. Parallel performance of an iterative method in cluster environments: an experimental study. In *Proceedings PMAA 2004, Marseille*, October 2004.

[9] S. Storie. Aspects of Communication Subsystem Analysis for Distributed Scientific Applications. Masters Thesis, University of Minnesota Duluth, May 2004.

[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] N. Ustemirov, M. Sosonkina, M.S. Gordon, M.W. Schmidt. Concurrent Execution of Electronic Structure Calculations in SMP Environments. In *Proceedings HPC 2005*, April 2005.

[12] 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