View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Elsevier - Publisher Connector

Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 9 (2012) 489 – 497

Procedia Computer Science

International Conference on Computational Science, ICCS 2012

Fragment Molecular Orbital Method Adaptations for Heterogeneous Computing Platforms

Sai Kiran Talamudupula, Masha Sosonkina*, Alexander Gaenko, Michael W. Schmidt

U.S. DOE Ames Laboratory
Iowa State University
Ames, IA 50011, USA

Abstract

Modern electronic structure calculations are characterized by unprecedented complexity and accuracy. They demand the full power of high-performance computing and must be in tune with the given architecture for superior efficiency. Thus, it is desirable to enable their static and dynamic adaptations using some external software (middleware), which may monitor both system availability and application needs, rather than mix science with system-related calls inside the application.

Building on the successful usage of the NICAN middleware with the computational chemistry package GAMESS, the work described in this paper links NICAN with the fragment molecular orbital (FMO) method to augment FMO with adaptive capabilities. Specifically, its fragment scheduling is performed, both statically and dynamically, based on current conditions within a heterogeneous computing environment. Significant execution time and throughput gains have been obtained with static adaptations, while the dynamic ones prevented FMO to abort calculations due to the insufficient memory available at the runtime.

Keywords: Fragment Molecular Orbital method, GAMESS, algorithmic adaptations, middleware, heterogeneous computing platforms

1. Introduction

Reliable quantum chemical (QC) calculations feature computational complexity of $O(N^3)\ldots O(N^4)$ even with the most basic quantum chemical method, and reach $O(N^6)\ldots O(N^8)$ for more accurate methods, where N is the number of basis functions, which is roughly proportional to the number of atoms in the chemical system. Therefore, as the size of a molecular system grows, quantum chemical treatment of large molecules quickly becomes prohibitively expensive. To reduce computational complexity, a fragmentation approach may be used, in which a large molecular system is first divided into fragments, second, a quantum chemical method is applied to each of the fragments, after which the fragment interactions are taken into account. In the Fragment Molecular Orbital (FMO) method [1, 2], the long-range (electrostatic) interactions between fragments are accounted for by the iterative calculations of each fragment in the

*Corresponding author
Email address: masha@scl.ameslab.gov (Masha Sosonkina)

1877-0509 © 2012 Published by Elsevier Ltd. Open access under CC BY-NC-ND license.
doi:10.1016/j.procs.2012.04.052

electrostatic field of all the other fragments (the FMO-1 approximation); the short-range interactions are accounted for by the explicit computations of fragment dimers (i.e., pairs of fragments considered as a single fragment) and optionally trimers (i.e., triples of fragments considered as a single fragment). The calculation of short-range interactions via dimer and trimer computations are referred to as FMO-2 and FMO-3 approximations, respectively.

In addition to fragmentation algorithms, the efficiency of computationally demanding quantum chemistry calculations depends on the availability of the system resources, both at the time of their allocation and during application execution. To optimize parallel performance, an application must be tuned to the system conditions. Several general approaches exist for application auto-tuning. The least intrusive and portable is to empower an application with a middleware that will provide a liaison between the application and system by monitoring the system resources dynamically, by making adaptation decisions based on the application performance, and then by invoking application adaptations, if needed. The NICAN middleware, proposed in [3], has been already used extensively as adaptation invocation tool in quantum chemistry applications, such as GAMESS [4], in which a runtime toggling of different Self-Consistent Field (SCF) method implementations constituted the nature of adaptations [5, 6, 7, 8]. Joint execution protocols of GAMESS and NICAN have been carefully studied in [9] under the system I/O congestion conditions. Based on these previous investigations, it was a natural choice to consider NICAN as a helper tool for adapting the FMO method to heterogeneities in the computing environments.

State-of-the-art and related work. Quantum chemistry calculations in heterogeneous (mostly, GPU-accelerated) environment recently received much attention [10, 11, 12]. Typically, the computational work is scheduled on the accelerator at compile time by invoking the kernels compiled to run on the accelerator. Load balancing is achieved statically, before the computation starts, by a selective assignment of different calculations to either CPU or GPU [11]. Alternatively, the work can be distributed among a pool of threads, some of which run on the CPU and some on a GPU [13]; or the optimal CPU-GPU work sharing ratio can be determined dynamically for an iteration by timing the CPU and GPU executions in the previous iteration [14].

Infrastructure for adaptive algorithms in computational chemistry is an emerging area of research. It draws from successful auto-tuning tools, such as Active Harmony [15, 16], for high-performance computing in general. Active Harmony provides for work migration on different levels of programming abstractions (threads, processes, etc.) with critical parameter prioritizing. Much heterogeneity appears in computational grids [17], so there exist various tools to deal with it. For example, IANOS [18] provides a middleware infrastructure that allows optimal positioning and scheduling of applications.

The paper is organized as follows. Section 2 gives an overview of FMO and describes its interface to the middleware NICAN. The proposed FMO task scheduling is provided in Section 3. Some experimental results and conclusions are given in Sections 4 and 5, respectively.

2. FMO implementation overview

First, a given molecular system has to be divided into fragments. If the system consists of several molecules not connected by chemical bonds (e.g., calculation of cluster of water molecules), each molecule or group of adjacent molecules is assigned to a fragment. If the molecular system represents a polymer, that is, a molecule composed of repeating structural units (e.g., a protein or cellulose molecule), each of the structural units is assigned to be a fragment. Generally, chemical functional groups are assigned to be fragments while taking care for the electron density within each fragment to be as localized as possible [2]. One such a division is shown in Fig. 1. After the molecule is divided into fragments, specific decisions are to be made regarding the theory levels applied to fragments and the choice of various computational parameters.

To support parallel computation both within and between the fragments, a new hierarchical parallelization scheme was proposed in [19], supported by the Generalized Distributed Data Interface (GDDI) library in GAMESS. The GDDI forms groups out of the available nodes (see Fig. 2) and schedules tasks to these groups. Here, a task is denoted as a fragment (monomer, dimer, or trimer) QC calculation in the FMO method. Note that a group may contain one or more nodes and each node may contain one or more cores. This distribution of tasks to nodes may be viewed as parallelization at the outer level. Then, in each group, the assigned task is itself calculated in parallel, which is termed as the inner-level parallelization.

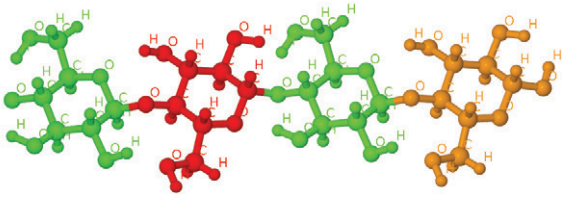

Figure 1: A molecule is divided into fragments (color-coded).

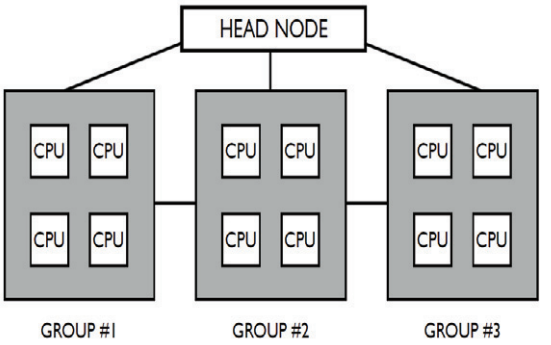

Figure 2: Nodes divided into groups in GDDI.

Once the groups are assigned to fragments, a quantum chemical (e.g., Restricted Hartree-Fock (RHF)) calculation is performed on each fragment separately by one group. After the electron density convergence is reached for all monomers, these densities are used to perform RHF runs on dimers by each group independently. Since the dimer calculation requires monomer density and energy exchanges, there exists a global synchronization at this point. To reduce the amount of wait time at these synchronization points, load balancing is necessary, which is organized as follows in the original FMO implementation. Monomers and dimers are executed in the decreasing order of their computation amount. The groups are ordered in the decreasing order of their numbers of nodes. Furthermore, each node in a group is assumed to be uniform in terms of computational characteristics. Then, the first-in-first-out policy is implemented for the task-to-group scheduling and is denoted here as LTFG-UN policy standing for the "largest task first to first group with uniform nodes". The present work expands this strategy to handle groups with nodes of different resource capacities that, in addition, may change dynamically during the FMO execution. If, for example, variations in the main memory occur at the runtime, the LTFG-UN policy may have to be modified in the course of execution to react to the changed system memory. These modifications lead to dynamic load balancing. In general, the group resource heterogeneity may be described in many ways, some of which include different number of nodes, cores, CPU loads, and varying memory availability. Although, in this paper, only two major heterogeneity causes are considered, namely, different number of cores per node and different memory availability at runtime, their treatment provides an approach to addressing other possible causes.

2.1. FMO with the middleware NICAN

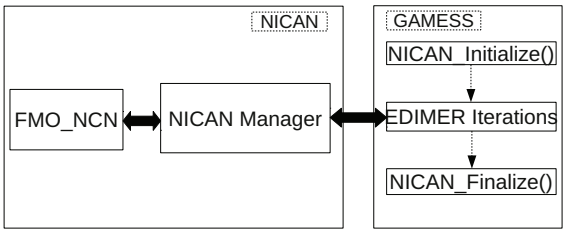

Figure 3: Example FMO dimer calculation connected to NICAN. Only one parallel FMO process is shown in the rectangle on the right. Thick two-sided arrows represent interprocess communications between FMO and NICAN Manager as well as between the Manager and the NICAN module FMO_NCN responsible for system resource monitoring.

Fig. 3 sketches the interlinking between FMO (represented as a large rectangle on the right) and NICAN (on the left). The `FMO_NCN` is the NICAN module used to check the system resources. `NICAN Manager` communicates directly with the FMO task calculation (shown as `EDIMER Iterations` in Fig. 3) via a NICAN daemon (not shown in Fig. 3) servicing a given parallel FMO process. Hence, the FMO process ID is also known to the NICAN Manager. In general, NICAN is charged with scheduling the tasks on the groups for execution. Specifically, its modules check first for the available memory and number of cores on each node. Then, the NICAN Manager takes this information along with the current order of the FMO tasks and makes the decision on where to place the task currently on the top of the task list. To accomplish the decision making, the NICAN manager needs to know the FMO task order in a certain format. The task description is expected to include the group number, node name, iteration number, the previous and current task number.

3. Description of FMO task scheduling adaptations

Although the LTFG-UN policy for task scheduling works well in homogeneous multicore computing systems and when the fragment sizes are relatively uniform, the scheduling of the incoming tasks may hinder the performance considerably when the significant heterogeneities are present either in the system or in the fragment division. The adaptations, which are proposed here for the heterogeneous environments, may be classified into static and dynamic as to their action on the task ordering. In the former, the tasks are reordered only once, before the first task execution while, in the latter, the ordering of yet to be executed tasks is subject to change. This paper considers a variable number of cores among nodes as an example of the system resource heterogeneity that may need only static scheduling adaptation. Conversely, the variability of the available memory serves to test the dynamic rescheduling of tasks in this work.

Even a simple situation when there is an equal number of nodes in each group with each node having different number of cores leads to inefficient scheduling under the LTFG-UN policy. Therefore, LTFG-UN is modified in a straightforward manner (denoted as LTBG-HN, where "B" stands for the "biggest" group having the most cores and "H" — for "heterogeneous") to account for an additional partial ordering of groups based on the numbers of cores in each group. Thus, the change in the LTFG-UN task ordering is observed only for the first k tasks, where k is the number of groups. Fig. 4 provides an example of two heterogeneous groups and how the task assignment either LTFG-UN (left) or LTBG-HN (right) policy affects the overall throughput of FMO tasks.

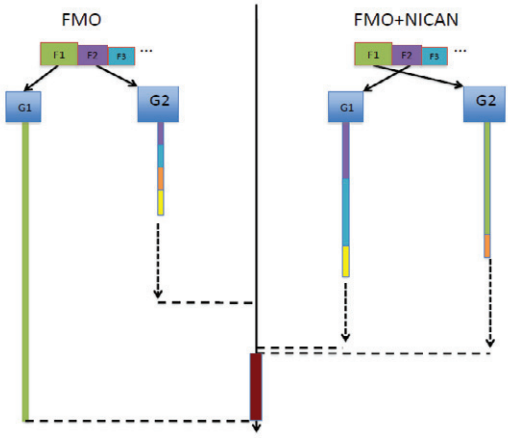

Figure 4: Example of an FMO calculation with two heterogeneous groups G1 and G2, such that G1 has fewer cores than G2 but both groups have the same number of nodes. The LTFG-UN and LTBG-HN policies, the latter being governed by NICAN, are shown on the left and right, respectively, of the timeline (straight solid arrow). Each task, denoted as F1, F2, F3,. . ., is attributed its own color in the execution sequence of each group. For a group, the end of all the computations is marked with a horizontal dashed line, such that the difference in the overall throughputs of the FMO with LTFG-UN and with LTFG-HN is indicated with a thick solid bar on the timeline.

Node heterogeneity implies system difference not only in static characteristics, such as number of cores, but also in dynamic ones, such as memory amount. The available main memory may change drastically at the runtime if, for example, a competing application is present on a node or no memory clean-up occurred since the last node usage. Even a moderate lack of memory, on the other hand, will hinder large-scale quantum chemistry computations. Currently, they check the available memory against the user-specified memory request at the start of the actual calculation while no runtime checks are performed. There are two obvious problems with this approach: firstly, the user easily may err on the requested amount, secondly, this check is done only once. To remedy this situation, a method to estimate the memory request for a given FMO calculation is considered. This memory estimation entails running the calculation as a "dry run", only to get the request size, under the assumption that there is enough memory to do so. In the dry run, all the fragment memory requirements are calculated, which is done by monitoring all the memory allocation calls inside the calculation. (Note that this operation is useful in diverse settings beyond the dynamic FMO scheduling.) Next, this request is checked against the current available memory amount, which is being monitored using an appropriate NICAN module according to the procedure provided in [20].

If a group fails to meet the memory requirements of any task to be executed, the scheduling is adapted dynamically. In particular, a task for which there is not enough memory on the requesting group is marked as *missed* and is scheduled to be executed later yielding to the next task that is a good fit to this group. Fig. 5 gives a scenario with a dynamic task scheduling. As before, tasks F1, Fi, Fj, and Fk are ordered in the decreasing order of their sizes. Each task would be assigned to group G1, Gi, Gj, or Gk, respectively, without runtime memory check. When such a check is performed, however, and a group, say, Gi fails to satisfy the memory requirement, the corresponding task Fi is marked as *missed* and stored in the *missed task queue* (shown on the left of Fig. 5). The next missed task (Fj) is appended to this queue. After the first pass through all the tasks, the queue is iterated over to allow the execution of

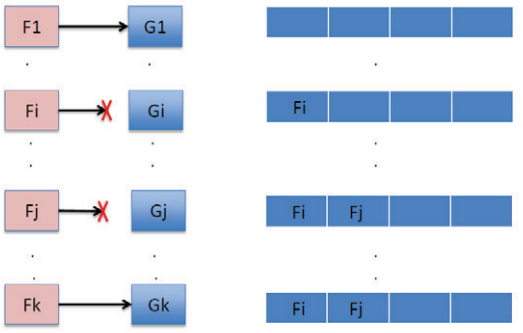

Figure 5: Scenario for a dynamic memory check when certain groups lack memory for a task execution (marked with "X"). The *missed* tasks are placed in the queue depicted as horizontal bar on the right of each group memory check. The queue is empty for the first task F1 assignment; it did not grow after the check of the Fk.

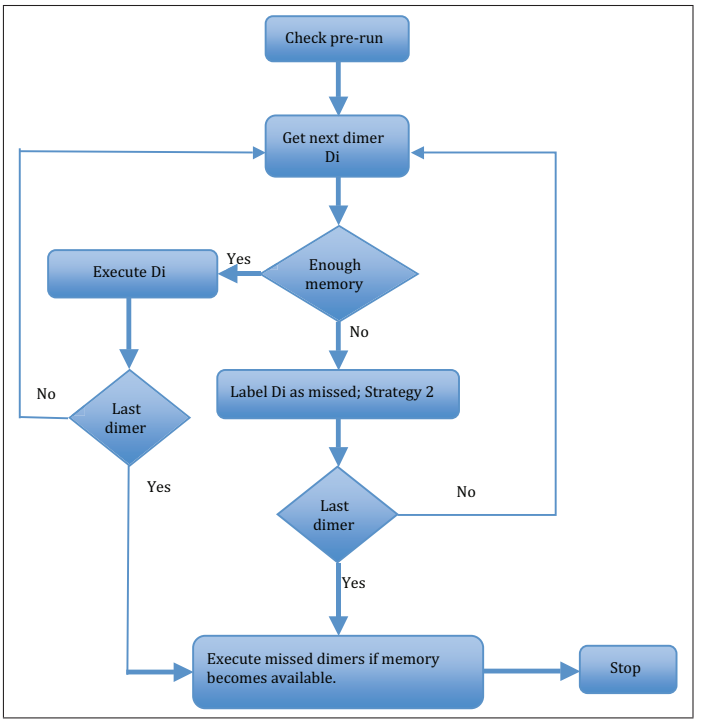

Figure 6: Flowchart for the DSQL policy implementation.

missed tasks. Such a dynamic scheduling policy that handles the queue last will be denoted as the DSQL policy. Fig. 6 shows a flowchart for the DSQL. It is typical that only one group at a time requests a task, so to compare the memory availability of only this group and a given task is reasonable and then, if needed, to delay the task execution according to certain policies, such as DSQL. A more sophisticated queue handling involves attempting to first schedule its items, as soon as the memory is available for their execution, before proceeding to the next task-to-group assignment in the list of tasks. An advantage of this policy (by analogy with DSQL, call it DSQF with "F" standing for "first") is that it adheres as much as possible to the "largest task first" principle whereas DSQL abandons it when the queue starts to fill in.

4. Experimental results

For the implementation and analysis purposes, the integration of the middleware with the dimer calculations only was considered, and all the results correspond to dimer calculations. An MP2 (Møller-Plesset 2nd order perturbation theory) calculation has been performed for each dimer. An FMO-MP2 computation includes the following stages: an iterative RHF computation of monomers, MP2 correction of the electronic density of the monomers, RHF computation of dimers, and an MP2 correction of dimers. A preliminary analysis of a single-node run of the nine-fragment system revealed that about 20% of the total calculation time is spent in the computation of the monomers (with the MP2 correction taking less than 2%), while the remaining 80% are spent in the dimer calculations. Specifically, the RHF computation of dimers took about 46% and the MP2 correction took 34% of the total time; and it was observed that the CPU utilization was much higher during MP2 than RHF.

The experiments have been done on the local cluster named Dynamo with 35 Intel Xeon E5420 nodes with two quad-core processors running at 3 GHz. Each node has a RAM of 16 GB, a 1.5 TB scratch space and runs a 64-bit Redhat Linux. The FMO calculations integrated with NICAN are run on up to five entire compute nodes (5 groups). Generally, FMO runs take advantage of the fact that the "one node per group" configuration works well for smaller fragments. Hence, in the experiments conducted in this work, a single node per group is used as well. However, the proposed static and dynamic adaptations work for any group size.

Figure 7: Cluster of water molecules divided into fragments shown as black circles.

As a test case, a cluster of water molecules has been considered. An eight-molecule cluster may be divided into fragments as shown in Fig. 7, for example. To use as static adaptation test case, a 16-molecule water cluster was taken, such that its 48 atoms were divided into nine fragments, each having the number of atoms as follows: [15, 12, 3, 3, 3, 3, 3, 3, 3]. All the possible dimers are represented as 27×1, 18×7, 15×7, and 6×21 in the notation $A \times B$, where A corresponds to the number of atoms in the dimer and B corresponds to the number of dimers with A atoms. For the dynamic adaptation tests, the same 16-molecule water cluster was divided into fragments somewhat differently: [15, 9, 3, 3, 3, 3, 3, 3, 3, 3]. Therefore, the dimer set is 24×1, 18×8, 12×8, and 6×28, consisting of 45 dimers total, which are composed out of ten fragments

4.1. Static adaptations for heterogeneous core counts

The experiments were organized into several scenarios each having n groups (i.e., n nodes) and $c_1, \ldots, c_n$ cores per group. In Fig. 8–11, these scenarios are labeled as Ci in the x-axis, where i enumerates all the scenarios in a given plot. The core-per-group assignments are shown under each scenario label. In Fig. 8–11, for a scenario Ci, the y-axis represents the maximum among the total execution timings in a group, defined as the the sum of the wallclock timings for all the dimers executed in that group.

In all the scenarios of Fig. 8–10 the LTFG-HN policy outperforms the original LTFG-UN policy (although by a tiny amount in C4 of Fig. 10), which is to be expected for the nonuniform core counts across the groups performing an FMO calculation. In general, the performance gains are more pronounced for (1) the scenarios having the weakest first group (C1–C3 bars in Fig. 8 and C1 bars in Fig. 9 and Fig. 10) and (2) when the variation between the extremal core counts is the largest (C1 bars in Fig. 8–10). Both observations may be explained by the situation that, under the LTFG-UN, the first (largest) task fully saturates the performance of the first one-core group causing the other groups to complete the execution of the remaining tasks quickly and wait for the first group to finish. Such an explanation is supported by the LTFG-UN standard deviation explosions (cf. the white boxes in Fig. 8–10) for all the scenarios.

The more homogeneous are the groups, the fewer performance differences one may observe between LTFG-UN and LTFG-HN. Furthermore, Fig. 11 presents a reversal of the relative performances, which may be surprising at first since there is a good deal of heterogeneity in the group core counts, such as $c_1 = 2$ and $c_3 = 6$ for C1. The standard deviations, shown in Fig. 11, are more mutually compatible than those of the previous experiments, although the same trend of LTFG-HN producing a smaller deviation persists except in the scenario C2. Fig. 12 and Fig. 13 provide some insight into this case. For the scenario C1 in Fig. 11, they present the individual execution timings for the first 15 dimers under the LTFG-UN and LTFG-HN, respectively. Observe that the first (largest) dimer took almost the same time to execute on the two-core group as on the six-core one (cf. the fist bars in Fig. 12 and Fig. 13), which indicates that there was not enough resource utilization on the six-core group to justify the scheduling of the largest dimer there and to expect this largest group to finish as fast as all the other groups calculating smaller dimers.

4.2. Dynamic adaptations to the main memory available at runtime

After executing a "dry run" and computing the memory requirements for each dimer, the dynamic scheduling policies, DSQL and DSQF, ave been simulated. Fig. 14 compares the two policies when executing the 16-molecule water cluster — divided as [15, 9, 3, 3, 3, 3, 3, 3, 3, 3] — on two groups having the same number of cores. The homogeneity in the group sizes was considered here in order to isolate the performance changes due to the dynamic adaptations. In the course of the given FMO run, dimer #1 was marked as *missed* per description in Section 3. At the end of the initial pass over all the dimers, using DSQL, the missed dimer #1 was finally executed by Group 1, because it was Group 1 that became available first for this missed dimer and that had now enough main memory for its execution. Since dimer #1 is the largest, Group 1 took longer to finish all the assigned tasks than Group 2 did so, which may be seen in Fig. 14 for the DSQL bars. Using DSQF, however, a periodic check was performed during the initial pass over the dimer list to determine whether the group requesting a task at that time had enough memory to fit dimer #1, which was missed at the start of the dimer list. After the calculation of dimer #3, when such a check was performed, the requesting Group 2 appeared to have sufficient memory and dimer #1 was scheduled on it, thus

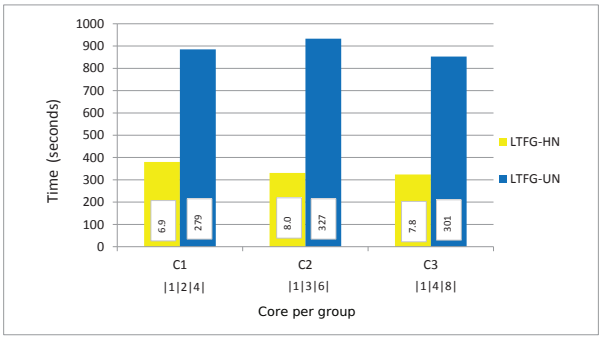

Figure 8: Maximum of the total execution times among three groups, such that the first group has one core, performing an FMO run with LTFG-HN or LTFG-UN static scheduling. The standard deviation of the total times is shown in white boxes in each bar.

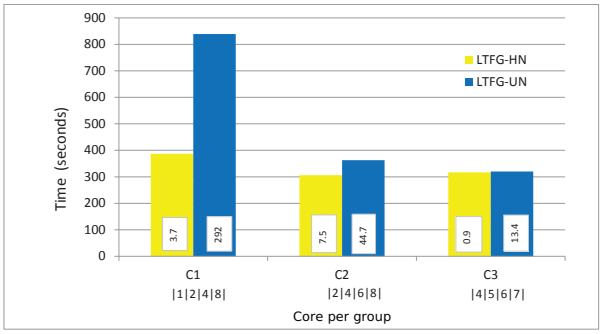

Figure 9: Maximum of the total execution times among four groups, with varying core counts per group, performing an FMO run with LTFG-HN or LTFG-UN static scheduling. The standard deviation of the total times is shown in white boxes in each bar.

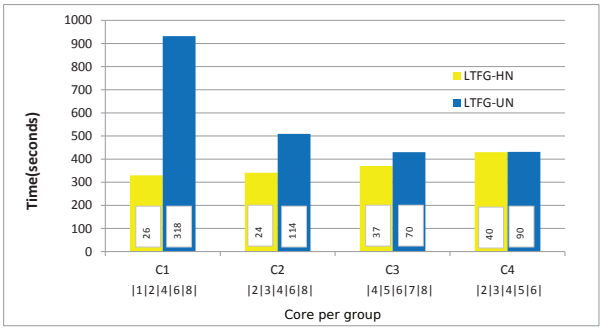

Figure 10: Maximum of the total execution times among five groups, with varying core counts, performing an FMO run with LTFG-HN or LTFG-UN static scheduling. The standard deviation of the total times is shown in white boxes in each bar.

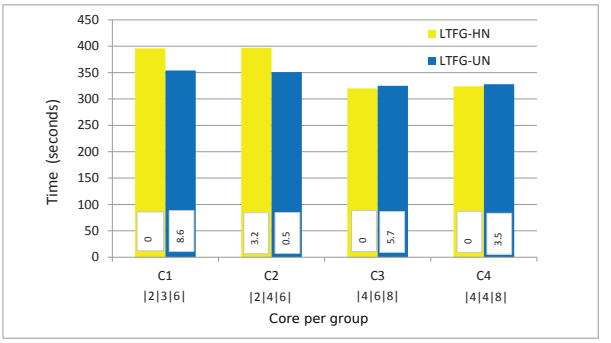

Figure 11: Maximum of the total execution times among three groups, such that differences of the group core counts are small, performing an FMO run with LTFG-HN or LTFG-UN static scheduling. The standard deviation of the total times is shown in white boxes in each bar.

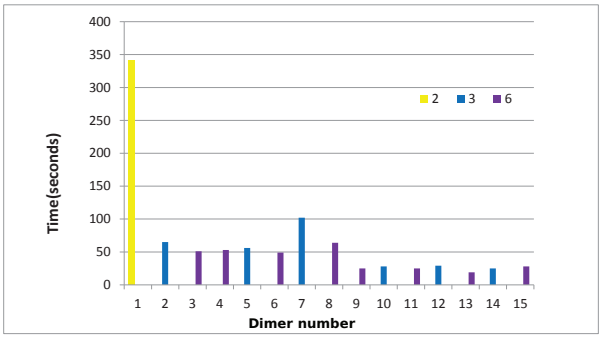

Figure 12: Execution timings of the first 15 largest dimers for the scenario C1 from Fig. 11 under the *LTFG-UN policy*. The groups are color-coded and their sizes are given in the plot legends.

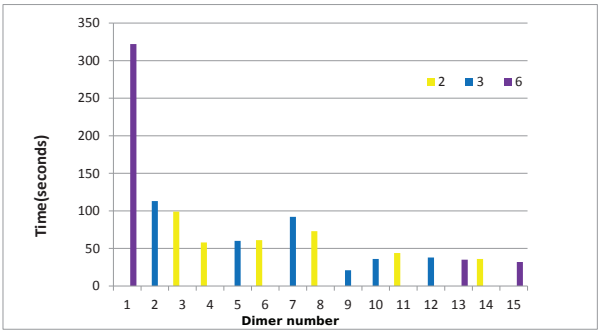

Figure 13: Execution timings of the first 15 largest dimers for the scenario C1 from Fig. 11 under the *LTFG-HN policy*. The groups are color-coded and their sizes are given in the plot legends.

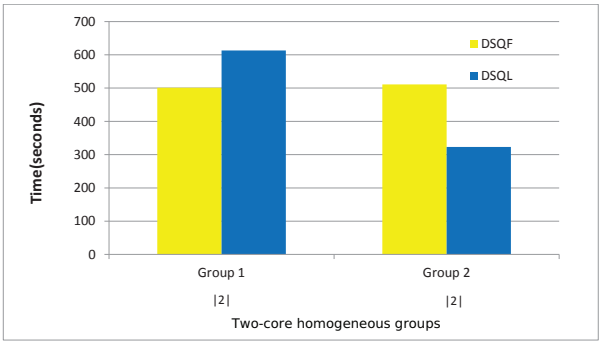

Figure 14: Comparison of two dynamic scheduling policies, DSQF and DSQL, on two homogeneous (two-core, labeled as |2|) groups, when dimer #1 is marked as missed and the rest of the dimers are calculated in the initial order.

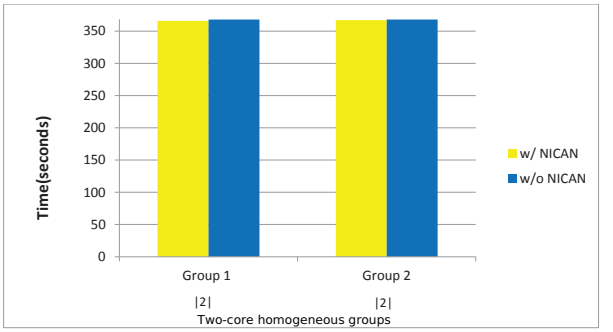

Figure 15: Total execution time on two groups (with two cores each, labeled as |2|) of an FMO run with and without middleware NICAN (`w/ NICAN` and `w/o NICAN`, respectively) when no lack of available memory occurred during the runtime.

changing the initial dimer ordering. Since there were no other missed dimers so far, dimer #4 was to be scheduled for the next requesting group (assuming this group had enough memory). The DSQF policy resulted in a more uniform overall completion for both groups (cf. the DSQF set of bars in Fig. 14), which is a consequence of its better adherence to the "largest task first" strategy, providing more opportunities to calculate the largest dimers, such that the rest of the groups does not remain idle due to the lack of tasks to execute.

On the overhead in dynamic adaptations. A "dry run " is required for adapting the FMO scheduling dynamically to the available memory at the runtime. Although this extra FMO calculation may be viewed as an overhead, its general usefulness puts it squarely into a category of handy utilities, instead, especially since the dry run may be performed only once for a given FMO fragment splitting and input settings. On the other hand, there is no tangible overhead due to dynamic adaptation decision-making by NICAN *per se*, which can be seen in Fig. 15, where the total execution times on both groups with and without NICAN are the same when no adaptation is needed, but the group memory is monitored and checked against the memory requirements anyway by NICAN. When an adaptation is necessary, there is an overhead only due to the change of the initial dimer ordering incurred by the *missed* dimers, which may cause for some groups to remain idle, as represented, e.g., by the DSQL bars in Fig. 14.

5. Summary and future work

This paper presents the first attempt to adapt the scheduling of FMO calculations to heterogeneous environments. Although preliminary, this work already provides some important insights. Firstly, such adaptations are possible and may be implemented seamlessly and efficiently with a help of a middleware, such as the NICAN tool, which has a long history of usage with the GAMESS quantum chemistry package. Secondly, the more heterogeneity is observed in the fragment division or in the computing system, the more performance gains are expected from the adaptations. Hence, these adaptations pave the way towards large chemical and biological system FMO calculations on heterogeneous computer architectures comprising not only multicore nodes but also accelerators, such as GPUs.

In the future, the current work will be expanded to consider a greater variety of the system heterogeneities, both static and dynamic, and to perform more diverse FMO calculations. In addition, other queue handling strategies can be considered, which, for example, involve modifying the described scheduling policies for various memory check conditions. Together with dynamic scheduling, more types of dynamic adaptations are possible, such as group splitting and agglomeration as well as the choice of the implementation for the chemical calculation. Finally, static and dynamic adaptations will be combined in a single FMO calculation at hand.

Acknowledgment

This work was supported in part by Iowa State University under the contract DE-AC02-07CH11358 with the U.S. Department of Energy, by the Director, Office of Science, Division of Mathematical, Information, and Computational

Sciences of the U.S. Department of Energy under contract number DE-AC02-05CH11231, and by the National Science Foundation grants NSF/OCI – 0749156, 0941434, 0904782, 1047772. The authors are thankful to the reviewers for their insights that helped improve the paper.

References

[1] M. Gordon, D. Fedorov, S. Pruitt, L. Slipchenko, Fragmentation methods: A route to accurate calculations on large systems, Chemical Reviews 112 (1) (2012) 632–672. arXiv:http://pubs.acs.org/doi/pdf/10.1021/cr200093j, doi:10.1021/cr200093j.
URL http://pubs.acs.org/doi/abs/10.1021/cr200093j

[2] D. Fedorov, K. Kitaura, Extending the power of quantum chemistry to large systems with the Fragment Molecular Orbital method, The Journal of Physical Chemistry A 111 (30) (2007) 6904–6914. arXiv:http://pubs.acs.org/doi/pdf/10.1021/jp0716740, doi:10.1021/jp0716740.
URL http://pubs.acs.org/doi/abs/10.1021/jp0716740

[3] M. Sosonkina, G. Chen, Design of a tool for providing network information to an application, in: V. Malyshkin (Ed.), Parallel Computing Technologies, PACT2001, Vol. 2127 of Lecture Notes in Computer Science, Springer-Verlag, Berlin, 2001, pp. 350–358.

[4] K. Baldridge, J. Boatz, S. Elbert, M. Gordon, J. Jensen, S. Koseki, N. Matsunaga, K. Nguyen, S. Su, T. Windus, M. Dupuis, J. M. Jr, M. Schmidt, General atomic and molecular electronic structure system, Journal of Computational Chemistry, November 1993 (1993) 1347–1363.

[5] L. Seshagiri, M.-S. Wu, M. Sosonkina, Z. Zhang, Exploring tuning strategies for quantum chemistry computations, in: R. S. *et al.* (Ed.), Software Automatic Tuning, Springer, 2010, pp. 193–208.

[6] L. Seshagiri, M.-S. Wu, M. Sosonkina, Z. Zhang, M. Gordon, M. Schmidt, Enhancing adaptive middleware for quantum chemistry applications with a database framework, in: Int'l Parallel and Distributed Processing Symposium (IPDPS 2010), 11th Workshop on Parallel and Distributed Scientific and Engineering Computing, Atlanta, GA, Apr. 2010, 2010.

[7] L. Seshagiri, M. Sosonkina, Z. Zhang, Electronic structure calculations and adaptation scheme in multi-core computing environments, in: G. Allen, J. Nabrzyski, E. Seidel, G. D. van Albada, J. Dongarra, P. M. A. Sloot (Eds.), Computational Science - ICCS 2009, 9th International Conference, Baton Rouge, LA, USA, May 25-27, 2009, Proceedings, Part I, Vol. 5544 of Lecture Notes in Computer Science, Springer, 2009, pp. 3–12.

[8] N. Ustemirov, M. Sosonkina, M. Gordon, M. Schmidt, Dynamic algorithm selection in parallel GAMESS calculations, in: Proc. 2006 International Conference on Parallel Processing Workshops, Columbus, OH, IEEE Computer Society, 2006, pp. 489–496.

[9] S. Talamudupula, M. Sosonkina, M. Schmidt, Asynchronous invocation of adaptations in electronic structure calculations, in: L. W. *et al.* (Ed.), Spring Simulation Multiconf., High Performance Computing Symp. (HPC 2011), Soc. for Modeling and Simulation Internat., Vista, CA, 2011, pp. 111–117.

[10] A. Asadchev, V. Allada, J. Felder, B. Bode, M. Gordon, T. Windus, Uncontracted Rys quadrature implementation of up to G functions on graphical processing units, Journal of Chemical Theory and Computation 6 (3) (2010) 696–704. arXiv:http://pubs.acs.org/doi/pdf/10.1021/ct9005079, doi:10.1021/ct9005079.
URL http://pubs.acs.org/doi/abs/10.1021/ct9005079

[11] C. Isborn, N. Luehr, I. Ufimtsev, T. Martínez, Excited-state electronic structure with configuration interaction singles and Tamm-Dancoff time-dependent density functional theory on graphical processing units, Journal of Chemical Theory and Computation 7 (6) (2011) 1814–1823. doi:10.1021/ct200030k.
URL http://pubs.acs.org/doi/abs/10.1021/ct200030k

[12] A. Götz, T. Wölfle, R. Walker, Chapter 2 – quantum chemistry on graphics processing units, in: r.A. Wheeler (Ed.), Annual Reports in Computational Chemistry, Vol. 6 of Annual Reports in Computational Chemistry, Elsevier, 2010, pp. 21 – 35. doi:10.1016/S1574-1400(10)06002-0.
URL http://www.sciencedirect.com/science/article/pii/S1574140010060020

[13] B. Bode, V. Allada, A. Asadchev, Taking GAMESS to the petascale, in: Abstracts of Papers, 242nd ACS National Meeting & Exposition, Denver, CO, United States, August 28-September 1, 2011, American Chemical Society, 2011, pp. PHYS–111.

[14] W. Brown, P.Wang, S. Plimpton, A. Tharrington, Implementing molecular dynamics on hybrid high performance computers – short range forces, Computer Physics Communications 182 (4) (2011) 898 – 911. doi:10.1016/j.cpc.2010.12.021.
URL http://www.sciencedirect.com/science/article/pii/S0010465510005102

[15] C. Tapus, I.-H. Chung, J. Hollingsworth, Active Harmony: Towards automated performance tuning, in: In Proceedings from the Conference on High Performance Networking and Computing, 2003, pp. 1–11.
URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.19.2497

[16] I.-H. Chung, J. Hollingsworth, Using information from prior runs to improve automated tuning systems, in: Proceedings of the 2004 ACM/IEEE conference on Supercomputing, SC '04, IEEE Computer Society, Washington, DC, USA, 2004, pp. 30–. doi:http://dx.doi.org/10.1109/SC.2004.65.
URL http://dx.doi.org/10.1109/SC.2004.65

[17] I. Foster, C. Kesselman, The grid, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999, Ch. Computational grids, pp. 15–51.
URL http://dl.acm.org/citation.cfm?id=289914.296088

[18] P. Wieder, W. Ziegler, V. Keller, IANOS – efficient use of HPC grid resources, ERCIM News 2008 (74).

[19] D. Fedorov, R. Olson, K. Kitaura, M. Gordon, S. Koseki, A new hierarchical parallelization scheme: Generalized distributed data interface (GDDI), and an application to the fragment molecular orbital method (FMO), Journal of Computational Chemistry 25, Issue 6 (2004) 872–880. arXiv:http://onlinelibrary.wiley.com/doi/10.1002/jcc.20018/full.
URL http://onlinelibrary.wiley.com/doi/10.1002/jcc.20018/full

[20] L. Xiao, S. Chen, X. Zhang, Dynamic cluster resource allocations for jobs with known and unknown memory demands, IEEE Transactions on Parallel and Distributed Systems 13, Issue:3 (2002) 223 – 240.