MPI/OpenMP parallelization of the Hartree-Fock method for the second generation Intel Xeon Phi

Extended Abstract

Kristopher Keipert
Department of Chemistry and Ames Laboratory, Iowa State University
Ames, Iowa 50011-3111, USA
kris@si.msg.chem.iastate.edu

Vladimir Mironov
Lomonosov Moscow State University
Leninskie Gory 1/3
Moscow 119991, Russian Federation
vmironov@lcc.chem.msu.ru

Yuri Alexeev
Argonne National Laboratory, Leadership Computing Facility
Argonne, Illinois 60439, USA
yuri@alcf.anl.gov

Michael D'mello
Intel Corporation
Schaumburg, Illinois 60173, USA
michael.dmello@intel.com

Alexander Moskovsky
RSC Technologies
Moscow, Russian Federation
moskov@rsc-tech.ru

Mark S. Gordon
Department of Chemistry and Ames Laboratory, Iowa State University
Ames, Iowa 50011-3111, USA
mgordon@iastate.edu

ABSTRACT

Modern OpenMP threading techniques are used to convert the MPI-only Hartree-Fock code in the GAMESS program to a hybrid MPI/OpenMP algorithm. Two separate implementations that differ by the sharing or replication of key data structures among threads are considered, density and Fock matrices. The hybrid MPI/OpenMP implementation reduces the memory footprint by approximately 200 times compared to the legacy code. The MPI/OpenMP code was shown to run up to six times faster than the original for a range of molecular system sizes.

KEYWORDS

Quantum chemistry, parallel Hartree-Fock, parallel Self Consistent Field, OpenMP, MPI, GAMESS

1 INTRODUCTION

The subject of this work is the successful adaptation of the HF method in the General Atomic and Molecular Electronic Structure System (GAMESS) quantum chemistry package to the second-generation Intel Xeon Phi processor platform. GAMESS is a free quantum chemistry software package maintained by the Gordon research group at Iowa State University. Many existing methods in GAMESS are parallelized with MPI. OpenMP is an attractive high-level threading application program interface (API) that is scalable and portable. The OpenMP interface conveniently enables sharing of the objects in memory thereby reducing the memory footprint of the whole program and more easily leverage all of the resources (cores, fast memory etc.) of the Intel Xeon Phi processor. Reducing the memory footprint is also expected to lead to better cache utilization, and, therefore, enhanced performance.

The HF method is a core quantum chemistry method which is used to iteratively solve the electronic Schrödinger equation for a many-body system. The resulting electronic energy and electronic wave function can be used to compute equilibrium geometries and a variety of molecular properties. The wave function is constructed of a finite set of basis functions suitable for algebraic representation of the integro-differential HF equations. The HF equations are solved numerically by self-consistent field (SCF) iterations. Contrary to what one might expect, the most time-consuming part of the calculation is not the solution of the Hartree-Fock equations, but rather the construction of a model Hamiltonian matrix which is also called Fock matrix. In most cases of practical interest, the calculation of the two-electron contribution to the Fock matrix occupies the majority of the overall compute time. On this step N^4 electron repulsion integrals (ERIs) are calculated leading to the theoretical time complexity of $O(N^4)$.

The Fock matrix as well as the so-called density matrix account for the majority of the memory footprint. The density matrix is accessed read-only during Fock matrix construction step and it is very easy to move it into shared memory. On the other hand, sharing the Fock matrix among threads is not an easy task: each ERI contributes to up to six elements scattered across the Fock matrix. Thus, explicit locks or barriers are required to eliminate possible data races. We developed two versions of the hybrid MPI/OpenMP HF algorithm. In one of them only the density matrix is shared and no locks and barriers are used. In the second one both Fock and density matrices are shared but Fock matrix update is protected by an explicit barrier.

2 RESULTS AND DISCUSSION

The benchmarks reported in this paper were performed on the Theta supercomputer at the Argonne Leadership Computing Facility (ALCF), which is a part of the U.S. Department of Energy (DOE) Office of Science (SC) Innovative and Novel Computational Impact on Theory and Experiment (INCITE) program. Theta is a 10-petaflop Cray XC40 supercomputer consisting of 3,624 Intel Xeon Phi 7230 processors. For large problem sizes, different memory and clustering modes were observed to have little impact on the time to solution for the three versions of the GAMESS code. For this reason, we simply chose the default one on Theta: quadrant-cache mode.

SC17, Denver, CO, USA
2017. 978-x-xxxx-xxxx-x/YY/MM…$15.00
DOI: 10.1145/nnnnnnn.nnnnnnn

Table 1: Parallel efficiency of the three different HF algorithms using 2.0 nm dataset

# Nodes	Time-to-solution, s			Parallel efficiency, %		
	MPI[a]	Pr.F.[a]	Sh.F.[a]	MPI[a]	Pr.F.[a]	Sh.F.[a]
4	2661	1128	1318	100	100	100
16	685	288	332	97	98	99
64	195	78	85	85	90	97
128	118	49	43	70	72	96
256	85	44	23	49	40	90
512	82	44	13	25	20	79

[a] MPI-only SCF code [b] Private Fock SCF code
[c] Shared Fock SCF code

For benchmarks, a system consisting of parallel series of graphene sheets was chosen. The graphene-sheet system is ideal for benchmarking, because the size of the system is easily manipulated. Various Fock matrix sizes can be targeted by adjusting the system size. In all, five configurations of the graphene sheets system were studied. The datasets for the systems studied are labeled as follows: 0.5 nm, 1.0 nm, 1.5 nm, 2.0 nm, and 5.0 nm.

If one compares the memory footprint of the original (MPI-only) and new HF codes, one sees that the private Fock code has about a 50 times less footprint compared to the stock MPI code. For the shared Fock code, the difference is even more dramatic with a savings of about 200 times. The ideal difference is 256 times since we compare 256 MPI ranks in the stock MPI code where all data structures are replicated versus 1 MPI rank with 256 threads for the hybrid MPI/OpenMP codes. The actual memory savings are not ideal due to additional replicated structures in the MPI/OpenMP codes.

Table 1 shows the multi-node scalability of the MPI-only version of GAMESS versus the private Fock and the shared Fock hybrid versions. It is important to appreciate at the outset that the multi-node scalability of the original MPI-only version of GAMESS is already reasonable. For example, the code scales linearly to 256 Xeon Phi nodes, and it is really the memory footprint bottleneck that limits how well all the Xeon Phi cores on any given node can be used. This pressure is reduced in the private Fock version of the code, and it is essentially eliminated in the shared Fock version. However, as with any shared object that is updated by multiple entities (threads or processes), the need for locking mechanisms, to ensure correctness, is what presents the limit to scalability of the shared Fock code. Overall, for the 2 nm dataset, the shared Fock code runs about six times faster than stock GAMESS on 512 Xeon Phi processors.

For small numbers of Intel Xeon Phi nodes and threads, the shared Fock version is expected to be on par with the other versions. This is because in these cases, the synchronization overhead is expected to be low. Eventually, as the overhead of the synchronization mechanisms begins to increase, the private Fock version of the code is found to dominate. In the end, the private Fock version outperforms stock GAMESS because of the reduced memory footprint, and outperforms the shared Fock version because of a lower synchronization overhead.

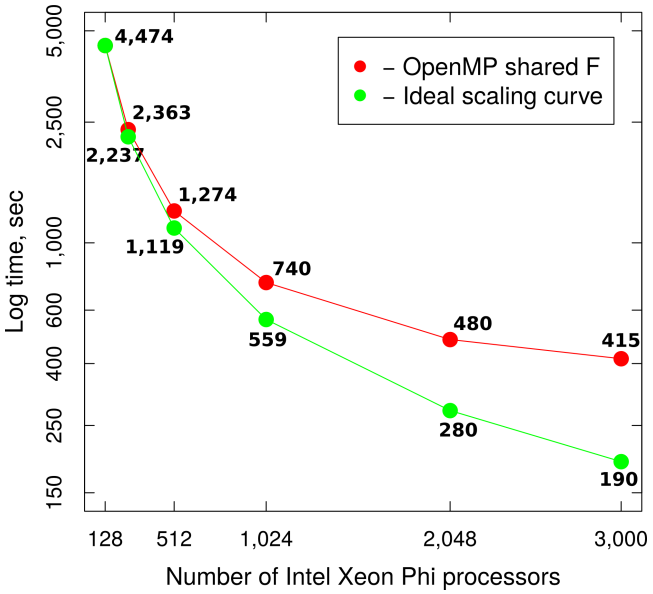

Figure 1: Scalability of the Shared Fock hybrid MPI-OpenMP version of GAMESS on the Theta for the 5.0 nm dataset.

3 CONCLUSION

In this paper, conversion of the MPI-only GAMESS HF code to hybrid MPI-OpenMP versions is described. The resulting hybrid implementations are benchmarked to exhibit improvements in the time-to-solution and memory footprint compared to the original MPI-only version. The code design decisions taken here were justified and implemented in a systematic way. Focus was placed on sharing the two primary (memory consuming) objects, the density and Fock matrices, in the SCF loop among the computation units. To the best of our knowledge, having a shared Fock matrix is an unique feature of our implementation. Indeed, this is absent in all other threaded HF codes known to us. Our new hybrid MPI/OpenMP codes significantly outperform the official stock MPI-only code in GAMESS. Our best case implementation has about 200 times smaller memory footprint and runs up to 6 times faster than the original MPI-only version. Both our hybrid versions also have better scalability with respect to cores and nodes on single node and multi-node Intel Xeon Phi systems respectively.

ACKNOWLEDGMENTS

This research used the resources of the Argonne Leadership Computing Facility, which is a U.S. Department of Energy (DOE) Office of Science User Facility supported under Contract DE-AC02-06CH11357. We gratefully acknowledge the computing resources provided and operated by the Joint Laboratory for System Evaluation (JLSE) at Argonne National Laboratory. Dr. Kristopher Keipert and Prof. Mark S. Gordon acknowledge the support of a Department of Energy Exascale Computing Project grant to the Ames Laboratory. We also thank the Intel® Parallel Computing Centers program for funding. The authors would like to thank the RSC Technologies staff for the discussions and help.