Exploring the performance of electron correlation method implementations on Kove XPDs

Colleen Bertoni
Argonne National Laboratory
Argonne, IL
bertoni@anl.gov

William Allcock
Argonne National Laboratory
Argonne, IL
allcock@anl.gov

Brian Toonen
Argonne National Laboratory
Argonne, IL
toonen@alcf.anl.gov

Spencer R. Pruitt
Worcester Polytechnic Institute
Worcester, MA
srpruitt@wpi.edu

Mark S. Gordon
Iowa State University
Ames, IA
mgordon@iastate.edu

ABSTRACT

In electron correlation methods in quantum chemistry, there are often high memory requirements which can reach terabytes for medium-sized molecular systems. For second-order perturbation theory (MP2), the two-electron integral arrays are the main memory bottleneck. Previously the two-electron integrals were recomputed, stored in distributed memory, or stored on disk. A way of storing the arrays which would remove the dependence on compute node memory and large latency associated with disk is by using an external memory appliance, like Kove®'s XPD ®.

In this work we modified a distributed memory implementation of MP2 to use XPDs instead of distributed memory. We evaluated the performance of our implementation against the distributed memory version for several molecular systems by considering scaling behavior with respect to compute processes and connections to XPDs.

1 MOTIVATION AND INTRODUCTION

A main goal of computational chemistry is to perform highly accurate calculations on large molecular systems. If the molecular systems studied have relatively strong dispersion interactions, like interactions between DNA base pairs, high-level electron correlation methods, which take into account instantaneous electron-electron interactions, are required to achieve accurate results.

However, electron correlation methods are very expensive, both in terms of computational complexity and memory costs. Second-order perturbation theory (MP2), one of the least expensive electron correlation methods, scales as $O(N^5)$, where N is a measure of molecular system size. Coupled Cluster Singles, Doubles, and Perturbative Triples (CCSD(T)), considered one of the most accurate methods, scales as $O(N^7)$.

In addition to high computational complexity, the MP2 method requires the calculation, storage, and manipulation of two-electron integrals. The memory needed for the two-electron integrals can reach terabytes for calculations on medium-sized molecular systems (thousands of basis functions). The main ways of handling storage and calculation of the two-electron integrals have been to recompute the integrals on the fly, store them on disk, or store them in a distributed array spread over nodes. The GAMESS quantum chemistry software package[5] implements two different versions of MP2, one that uses distributed memory, handled in the Distributed Data Interface (DDI) parallelization model [1, 4], and another that uses a disk-based parallelization model[2, 3].

In the DDI model, the two-electron integrals are spread across all the nodes. The distributed approach decreases the amount of memory required per core, but the only way to add additional memory to a calculation is by adding more nodes. In some MP2 and CCSD(T) calculations, additional compute nodes are used because more memory is needed to carry out the calculation, not necessarily because more compute power is needed. In addition, although some parts of the array are local to a node, other parts of the array will be stored on a remote node, and thus there will be overhead from communication across the network.

An alternative to either a distributed memory or disk-based approach is to store the arrays on external appliances which provide only RAM and not compute power. Kove ® XPDs® can be used for this case. Although we are still limited by the amount of memory available on XPDs, using XPDs could be an effective way to add extra memory to a compute cluster when only RAM is needed, instead of buying more compute nodes or more expensive memory for existing nodes. This approach decouples the compute power from the memory pool, so the available resources can be used more efficiently. Additionally, depending on a cluster's size, using XPDs can allow access to a larger memory pool than might be available on the cluster, and so allow larger calculations to be performed.

In this work, we implemented a version of DDI which stores the two-electron integrals on XPDs instead of in distributed memory (DDI/XPD). The CCSD(T) and MP2/DDI implementations in GAMESS can then immediately use DDI/XPD. We performed experiments to explore whether using XPDs is a viable alternative for scientific applications, using the implementation of MP2/DDI in GAMESS as a test case.

2 EXPERIMENTAL SETUP

All experiments were carried out on Cooley, a cluster at Argonne National Laboratory. Cooley is comprised of 126 nodes, each with dual Intel Haswell 6-core processors, a FDR Infiniband network, and one HCA per node. Three XPDs are connected to Cooley, two with 4 Infiniband links and one with 6 Infiniband links.

Kove provides the Kove Direct System Architecture (KDSA) C API to manipulate data on XPDs. Among many functions, it can synchronously and asynchronously read/write data on an XPD.

3 IMPLEMENTATION AND OPTIMIZATION

The DDI contains three blocking calls to manipulate data in a distributed array: ddi_get, ddi_put, and ddi_acc. These functions take in an array handle and the positions of the array to manipulate. The details of transferring data across the nodes is hidden from the caller. For the standard DDI implementation, the data transfer is handled using MPI or TCP/IP and a second process, called a "data server" for each compute process. The time spent in the DDI functions can be a large percentage of the total runtime, as shown in Fig. 2 of the poster.

An important note is that in the MP2 implementation (as well as in many other scientific algorithms) the arrays are not always read/written/accumulated to in a contiguous fashion (for example, accumulating a submatrix to a previously defined matrix). Additionally, there is no overlap of computation and communication in the MP2 implementation.

For the DDI/XPD implementation, ddi_get, ddi_put, and ddi_acc were reimplemented with calls to the KDSA API, and no data servers were used. Initially synchronous calls to read and write were used. However, as mentioned above, some array manipulations are non-contiguous, which means that each column in the array has to have a separate call to the KDSA API. A better alternative implemented ddi_get, ddi_put, and ddi_acc with asynchronous read/write calls from the KDSA API for each column. The overall ddi_get, ddi_put, and ddi_acc calls remained blocking, but using asynchronous calls inside the functions helped hide the overhead from making many calls to the KDSA API.

4 PRELIMINARY PERFORMANCE RESULTS

To evaluate the performance of DDI/XPD in GAMESS, MP2 gradient calculations on several benchmark test molecular systems were run. We present the scaling of the MP2 gradient calculation with standard DDI and DDI/XPD with respect to compute processes and with respect to the number of Infiniband links to the XPD.

The amount of memory needed depends on the molecular system and basis set, which determines the number of basis functions (Nbasis), the number of occupied orbitals (Nocc), and the number of virtual orbitals (Nvir). The test systems are shown in Table 1.

Table 1: Benchmark test systems

Molecular System	Basis set	Nocc	Nvir	Nbasis	Distributed/ XPD Memory (GB)
quinone	6-31G(d)	54	191	245	1.2
19 waters	6-311	95	266	361	8.2
$C_{40}H_{20}$	cc-pVDZ	130	570	700	55.4

4.1 Scaling with respect to XPD links

Using the KDSA API, each process can use a different link to an XPD, and data can be striped across multiple XPDs. Experiments were run using the quinone and water cluster test cases and 6 compute processes per node (ppn). The number of links used was varied from 2 to 14, and the stripe size was varied between 2, 512, and 1024 KiB if multiple XPDs were used. The links were distributed round-robin to the compute processes. For the quinone test case, the best time to solution tended to be with 6 links and 1 XPD, although the results from using 14 links only differed by about a second for larger node counts. For the water cluster, the best time to solution tended to be 14 links with a 512 or 1024 KiB stripe size. Both test cases stopped scaling around 16-32 nodes, although this is likely due to the method, since standard DDI behaves similarly.

4.2 Scaling with respect to compute processes

The DDI/XPD runs were performed with 3 XPDs using 14 links, with a 1024 KiB striping size. Water cluster and quinone results are an average of five runs, while the $C_{40}H_{20}$ results are an average of three runs, except for 1 and 2 node results. The average coefficient of variation (CV) was small ($\leq$ 3.0 %) for all test cases. For the water cluster the maximum CV for runtime was 3.0% and 3.8% for DDI/XPD and standard DDI, respectively, while for the $C_{40}H_{20}$ test case, the maximum CV was 5.7% and 4.6% for DDI/XPD and standard DDI, respectively. For the quinone test case, the maximum CV was larger, 17.4% for DDI/XPD and 24.8% for standard DDI. The maximum CV for quinone for DDI/XPD and standard DDI occurred when the total runtime was 16.3s and 12.7s for DDI/XPD and standard DDI, respectively. The corresponding standard deviations are 2.8 and 3.2 seconds. It is possible that the variation is due to network contention from other jobs sharing the network.

We performed benchmarking runs ranging from 1 to 96 nodes with 2, 4, 6, 8, 10, and 12 ppn. From comparing the performance (Fig. 2 of the poster), we found that DDI/XPD scales similarly to or better than standard DDI for the two smaller test cases, and standard DDI scales better for the largest test case. In the largest test case, it is possible that the striping size needs to be changed to improve scaling. The time to solution varies from DDI/XPD being about 4.4x slower than standard DDI in the worst case (water cluster test case, 1 node with 12 ppn) to slightly faster (2.0x faster for the quinone test case with 96 nodes and 6 ppn). We note that the single-node results are the worse case scenario for the comparison, since DDI/XPD must transfer data over the network, while standard DDI only uses memory local to the single node.

The fastest time to solution in the water cluster and $C_{40}H_{20}$ test cases is from using standard DDI, and the fastest time to solution in the quinone test case is from using DDI/XPD. The fastest DDI/XPD runs were only 2.1x and 3.1x slower than the fastest standard DDI runs for the water cluster and $C_{40}H_{20}$ test cases, respectively, and the fastest standard DDI run was 1.07x slower than the fastest DDI/XPD run for the quinone test case.

5 FUTURE WORK

The MP2 implementation we modified to use XPDs did not overlap communication and computation, so the time spent reading and writing to XPDs could not be hidden. In the future, we plan to modify an implementation of MP2 which does overlap communication and computation.

ACKNOWLEDGMENTS

Thanks to Kove for helpful discussions. This material is based upon work supported by the U.S. Department of Energy, Office of Science, under contract number DE-AC02-06CH11357.

REFERENCES

[1] Graham D. Fletcher, Michael W. Schmidt, Brett M. Bode, and Mark S. Gordon. 2000. The Distributed Data Interface in GAMESS. *Computer Physics Communications* 128, 1 (2000), 190–200. https://doi.org/10.1016/S0010-4655(00)00073-4

[2] Kazuya Ishimura, Peter Pulay, and Shigeru Nagase. 2006. A new parallel algorithm of MP2 energy calculations. *Journal of Computational Chemistry* 27, 4 (2006), 407–413. https://doi.org/10.1002/jcc.20348

[3] Kazuya Ishimura, Peter Pulay, and Shigeru Nagase. 2007. New parallel algorithm for MP2 energy gradient calculations. *Journal of Computational Chemistry* 28, 12 (2007), 2034–2042. https://doi.org/10.1002/jcc.20731

[4] M. Olson Ryan, W. Schmidt Michael, S. Gordon Mark, and P. Rendell Alistair. 2003. Enabling the Efficient Use of SMP Clusters: The GAMESS/DDI Model. *SC Conference* (2003), 41–41. https://doi.org/10.1109/SC.2003.1001

[5] Michael W. Schmidt, Kim K. Baldridge, Jerry A. Boatz, Steven T. Elbert, Mark S. Gordon, Jan H. Jensen, Shiro Koseki, Nikita Matsunaga, Kiet A. Nguyen, Shujun Su, Theresa L. Windus, Michel Dupuis, and John A. Montgomery. 1993. General atomic and molecular electronic structure system. *Journal of Computational Chemistry* 14, 11 (1993), 1347–1363. https://doi.org/10.1002/jcc.540141112