Exploring the performance of electron correlation method implementations on Kove XPDs

Colleen Bertoni[1], William Allcock[1], Brian Toonen[1], Spencer R. Pruitt[2], Mark S. Gordon[3]
[1]Leadership Computing Facility, Argonne National Laboratory (USA); [2]Worcester Polytechnic Institute (USA); [3]Iowa State University (USA)

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. A way of storing the arrays which removes 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:

1) modified a distributed memory implementation of MP2 to use an XPD instead of distributed memory,

2) 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.

Introduction

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 implements two different versions of MP2, one that uses distributed memory, handled in the Distributed Data Interface (DDI) parallelization model, and another that uses a disk-based parallelization model.

In the DDI model, the two-electron integrals (and partially calculated 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. An alternative way of storing the arrays is by using an external memory appliance, like an XPD. XPDs are 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.

GAMESS/DDI API

We implemented a version of the DDI library which stores the two-electron integrals on XPDs instead of in distributed memory (DDI/XPD). The MP2/DDI implementations in GAMESS can then immediately use DDI/XPD.

DDI has three main functions, which are blocking:

- ddi_get: reads a 2D block of an array from shared memory
- ddi_put: writes a 2D block of an array from shared memory
- ddi_acc: accumulates a 2D block of an array from shared memory

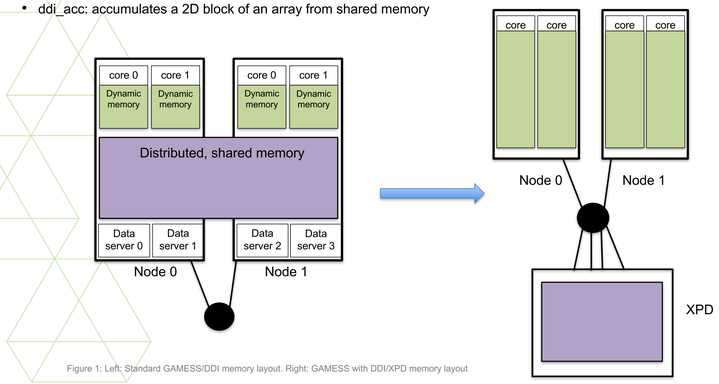


Figure 1: Left: Standard GAMESS/DDI memory layout. Right: GAMESS with DDI/XPD memory layout

Scaling with nodes

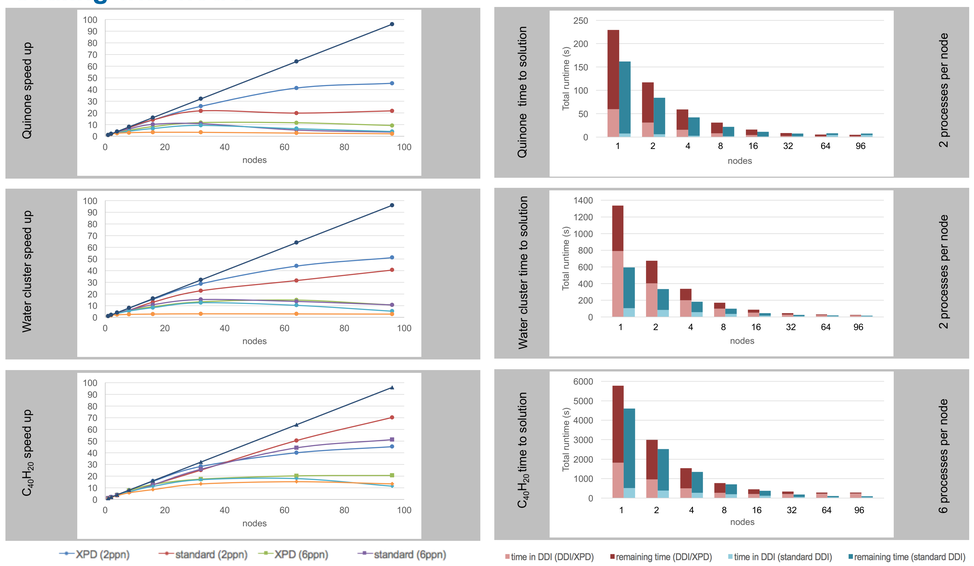


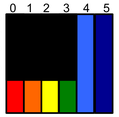
Figure 2: Scalability with respect to nodes for the three test cases. DDI/XPD used 14 XPD links on 3 XPDs with a strip size of 1024 KiB. The runtimes are the average of five runs. ppn denotes compute processes per node. The left hand panels show speed-up curves for various ppn. Runs with 2,4,6,8,10,and 12 ppn were performed, but only 2,6, and 10 are presented here for clarity. The right hand panel shows the time to solution curves which resulted in the fastest time to solution. The time is split into the time spent in DDI routines and the time not spent in DDI calls (the remaining time).

Implementation details

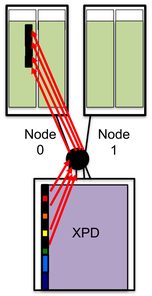
In the MP2 implementation (as well as in many other scientific algorithms) the arrays are not always read/written/accumulated to in a contiguous fashion.

Thus, DDI/XPD 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 library.

```
1.  while( read columns < number of columns )
2.      check how many async calls have finished
3.      if( at least one async call finished )
4.          for each finished async call, send out another async call
5.      else
6.          send out a set number of async calls
7.      end if
```



Algorithm 1: Using async calls to manipulate a 2D array in terms of columns



Basic Equations

The equation for the MP2 energy is

$$E^{MP2} = \sum_{ab}^{AO} \sum_{ij}^{MO} t_{ij}^{ab} \left(ai|bj\right) \quad (1)$$

where $t_{ij}^{ab} = \frac{2(ai|bj) - (bi|aj)}{\varepsilon_i + \varepsilon_j - \varepsilon_a - \varepsilon_b}$, ε_r is an orbital energy, *i,j* denote occupied orbitals, *a,b* denote virtual orbitals, and

$$\left(ai|bj\right) = \sum_{\mu\nu\lambda\sigma}^{AO} c_{\mu a} c_{\nu i} c_{\lambda b} c_{\sigma j} \left(\mu\nu|\lambda\sigma\right) \quad (2)$$

where $\left(\mu\nu|\lambda\sigma\right)$ is a two-electron integral. The Greek letters denote Gaussian basis functions (AOs) and *c* are previously calculated molecular orbital coefficient matrices. An expression for the MP2 gradient is

$$(E^{HF} + E^{MP2})^{x} = \sum_{\mu\nu}^{AO} H_{\mu\nu}^{x} \left(P_{\mu\nu}^{MP2} + \sum_{rs}^{\text{all MO}} c_{\mu r} c_{\nu s} P_{rs}^{(2)} \right) + \sum_{\mu\nu}^{AO} S_{\mu\nu}^{x} \left(W_{\mu\nu}^{MP2} + \sum_{rs}^{\text{all MO}} c_{\mu r} c_{\nu s} W_{rs}^{(2)} \right) + \sum_{\mu\nu\lambda\sigma}^{AO} \left(\mu\nu|\lambda\sigma\right)^{x} \Gamma_{\mu\nu\lambda\sigma}^{MP2} + V^{x} \quad (3)$$

where $P_{rs}^{(2)}$, $\Gamma_{\mu\nu\lambda\sigma}^{MP2}$, $W_{rs}^{(2)}$ are coefficients which also depend on the two-electron integrals. The basic task is to calculate and store (2), and use it to calculate (1) and (3).

Experimental Setup and Test Systems

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 the XPD. Among many functions, it can synchronously and asynchronously read/write data on the XPD.

The benchmark molecular test systems are shown in Table 1 below. To evaluate the performance, we considered 1) the strong scaling of the MP2 gradient calculation on one node and multiple nodes, shown in Figure 2, and 2) the scaling with respect to the number of Infiniband links to the XPD, shown in Figure 3.

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

Table 1: Molecular test systems, where Nocc is the number of occupied orbitals, Nvir is the number of virtual orbitals, Nbasis is the number of basis functions.

Conclusions and Future Work

1) The scalability, as presented in the speed up curves in Fig. 2, is similar or better for DDI/XPD compared to standard DDI for the water cluster and quinone test case, and better for standard DDI than DDI/XPD for the $C_{40}H_{20}$ test case. For time to solution, the XPD version was 2.0x faster and 4.4x slower than the standard DDI version in the best case (quinone, 96 nodes, 6 ppn) and worst case (water cluster, 1 node, 12ppn), respectively.

2) 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.

3) Increasing the number of XPD links improved performance, as shown in Fig. 3. 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 512 or 1024 KiB stripe size.

4) The performance variability when using XPDs was small. The average coefficient of variation for the DDI/XPD version was 3.0%, 0.9%, and 0.6% for the quinone, water cluster, and $C_{40}H_{20}$, respectively. 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.

In the future, we plan to modify an implementation of MP2 which overlaps communication and computation, as well as test the implementation of DDI/CCSD(T) using an XPD.

Scaling with XPD links

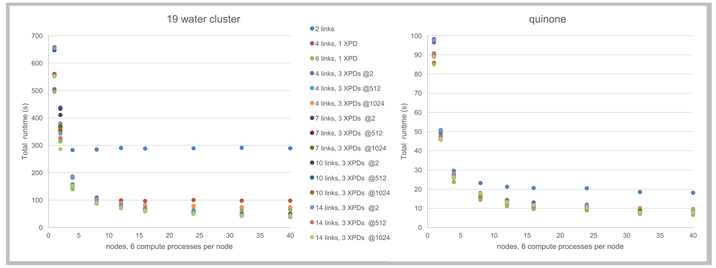


Figure 3: Scalability with respect to connections to XPDs, with 6 ppn. When multiple XPDs are used, the stripe size is denoted by @N, where N is the strip size in KiB. The results are averaged over three runs.

Argonne NATIONAL LABORATORY
U.S. DEPARTMENT OF ENERGY
Argonne National is a U.S. Department of Energy laboratory managed by UChicago Argonne, LLC.

Acknowledgements: This material is based upon work supported by the U.S. Department of Energy, Office of Science, under contract number DE-AC02-06CH11357.