

MPI/OpenMP Parallelization of the Fragment Molecular Orbital Method in GAMESS

Vladimir Mironov¹, Yuri Alexeev², Dmitri G. Fedorov³

¹Department of Chemistry, Lomonosov Moscow State University; ²Leadership Computing Facility, Argonne National Laboratory; ³National Institute of Advanced Industrial Science and Technology

Abstract

In this work, we present a novel parallelization strategy for the Fragment Molecular Orbital (FMO) method in the quantum chemistry package GAMESS. The original FMO code has been parallelized only with MPI, which limits scalability of the code on multi-core massively parallel machines. To address this problem, we parallelized FMO with a new hybrid MPI-OpenMP scheme that shows excellent scaling up to 2,048 Intel Xeon Phi nodes (131,072 cores) on Theta supercomputer. MPI-OpenMP code not only scales better compared to MPI code, but also performs up to two times faster and has significantly smaller memory footprint.

Fragment Molecular Orbital (FMO) Method

□ In FMO, a molecular system is divided into fragments and the total properties, such as the energy or its gradient, is calculated from those fragments and (in FMO2) their pairs, computed in the embedding potential. FMO treats the electrostatics for the whole system while the exchange-repulsion and charge transfer are only accounted for at the local level of fragment pairs.

□ The total energy can be written as:

$$E = \sum_I^{N_{frag}} E_I + \sum_I^{N_{frag}} \sum_{j < I}^{N_{frag}} (E_{IJ} - E_I - E_J)$$

where the monomer (I) [FMO1], dimer (IJ) [FMO2] energies can be obtained using any standard QM method

□ Current applications:

- geometry optimization (~2,000 atoms)
- ab initio level single point energies (~20,000 atoms)
- pair interaction analysis (~20,000 atoms)
- drug design, ligand docking, chemical reactions in solution etc

□ Massively parallel FMO utilizes the Generalized Distributed Data Interface (GDDI):

- With the molecule divided into fragments, each fragment or pair is computed on a distinct subset, or group, of the available processors
- Each fragment/pair is then run in parallel in each group
- This provides two levels of parallelization, greatly increasing the parallel efficiency

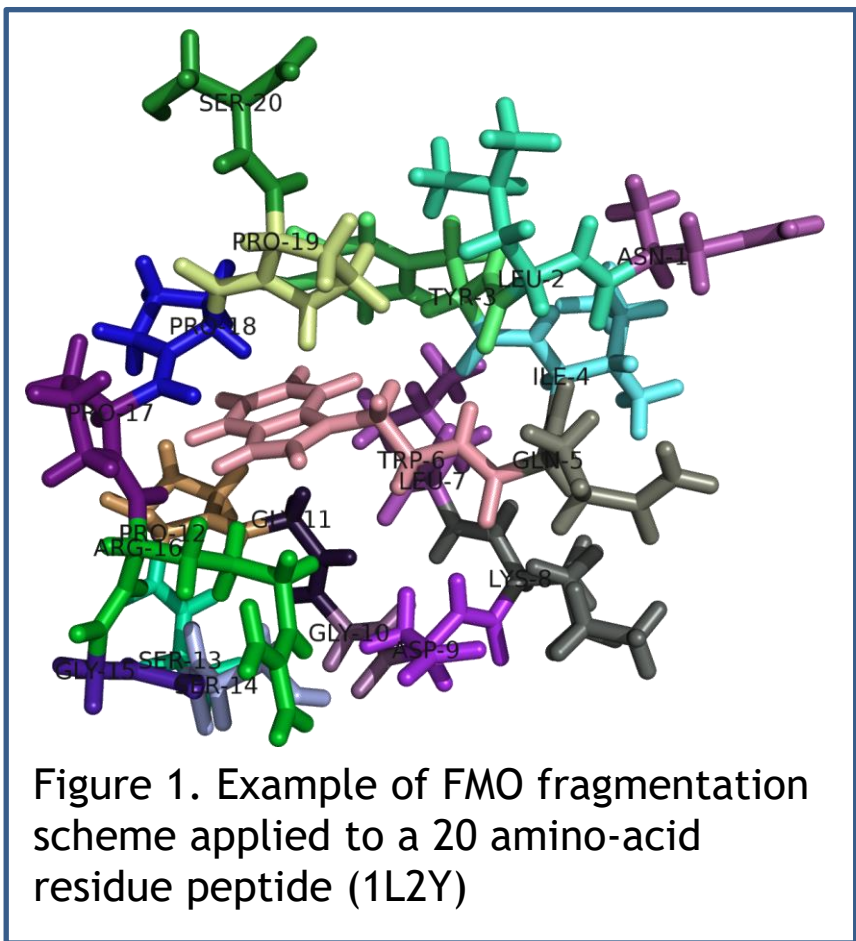


Figure 1. Example of FMO fragmentation scheme applied to a 20 amino-acid residue peptide (1L2Y)

GAMESS (General Atomic and Molecular Electronic Structure System

□ A general purpose comprehensive *ab initio* quantum chemistry package

□ Maintained by the research group of Prof. Mark Gordon at Iowa State University (<http://www.msg.ameslab.gov/gameSS>).

✓ Enables most major quantum mechanical methods (Hartree-Fock, Møller-Plesset perturbation theory, coupled cluster, multiconfiguration self consistent field, configuration interaction, density functional theory).

✓ Ported to many computer architectures.

✓ Free and widely used on everything from laptops to supercomputers.

✓ Highly scalable, including many distributed data algorithms.

✗ More than a million lines of legacy Fortran 77 code, with an associated parallelization library comprising 15,000 lines.

✗ Parallelization library does not support multithreading

Description of Target Platform

This work targets the Intel Xeon Phi platform provided by the Stampede2 in Texas Advanced Computing Center (TACC) and Theta supercomputer at the Argonne Leadership Computing Facility (ALCF).

	Stampede2 (TACC)	Theta (ALCF)
Intel Xeon Phi Models	Intel Xeon Phi 7250, 68 cores, 1.4 GHz	Intel Xeon Phi 7230, 64 cores, 1.3 GHz
Memory per node	16 GB MCDRAM + 96 GB DDR4	16 GB MCDRAM + 192 GB DDR4
Number of KNL nodes	4,200	3,624
Interconnect type	100Gb/sec Intel Omni-Path	Aries interconnect with Dragonfly topology
Compiler	Intel Parallel Studio XE 2018.2	

FMO algorithm in GAMESS

```
1: Calculate one-electron Hamiltonian of each fragment
2: Guess initial density matrix for each fragment Di
  ▶ Monomer calculation:
3: repeat
4:   for i = 1, NumFrag do
5:     GDDI load balancing
6:     Compute embedding ESP quasi-Fock matrix
7:     Compute wavefunction of the fragment i, update
      energy and density matrix of fragment i (εi, Di)
8:     Compute fragment properties
9:     Broadcast results (energy, density, charges)
```

```
10:   end for
11:   until max(|Diold - Di|) < conv
  ▶ Dimer calculation:
12:   for i = 1, NumFrag do
13:     for j = 1, i - 1 do
14:       GDDI load balancing
15:       Compute embedding ESP of the dimer
16:       Compute wavefunction, get energy (εij) and
          density matrix(Dij) of the fragment pair ij
17:     end for
18:   end for
19: Gather results of fragment pairs calculation
20: Calculate whole system properties, print results
```

Figure 2. Algorithm of FMO (FMO2) method

Goals of the study

□ Eliminate or reduce drawbacks of MPI-only parallelization:

- Excessive memory utilization on massively parallel CPU architectures for legacy codes like GAMESS.
- Only half of GAMESS processors are doing computation, other are used as data-servers.

□ Solution:

- Implement multithreaded algorithms for intranode parallelization of FMO-specific parts of code

MPI+OpenMP parallelization of embedding ESP calculation step

```
1: !$omp parallel private(F) shared(D)
2: for i = 1, NumShells1 do
3:   for j = 1, i do
4:     call ddi.dlbnxt(i,j)
5:   screened ← schwartz(i,j,i,j)
6:   if screened cycle j-loop
7:     !$omp do schedule(dynamic)
8:     for k = 1, NumShells2 do
9:       for l = 1, k do
10:        screened ← schwartz(i,j,k,l)
11:        if not screened then
12:          Compute ERI Xijkl = (i,j|k,l)
13:          call eri(i,j,k,l,Xijkl)
14:          Update ESP matrix:
15:            Fij += Xijkl · Dkl
16:          end if
17:        end for
18:      end for
19:    end for
20:  !$omp end parallel
  ▶ Reduce ESP matrix over MPI processes:
21:  call ddi.gsum(F)
```

Figure 3. Hybrid MPI+OpenMP algorithm of embedding ESP calculation

Basic steps of FMO (FMO2) calculation:

1. Compute one-electron (core) Hamiltonian of each fragment
2. Construct initial guess wavefunction of each I fragment
3. Monomer self consistent field calculations. Compute the wavefunction of fragments in a field of other fragments. The latter is described by the embedding electrostatic potential (ESP). This step loops until all wavefunctions converge
4. Dimer calculations. They are done once at the end
5. Aggregate the results and compute properties

FMO parallelization strategy:

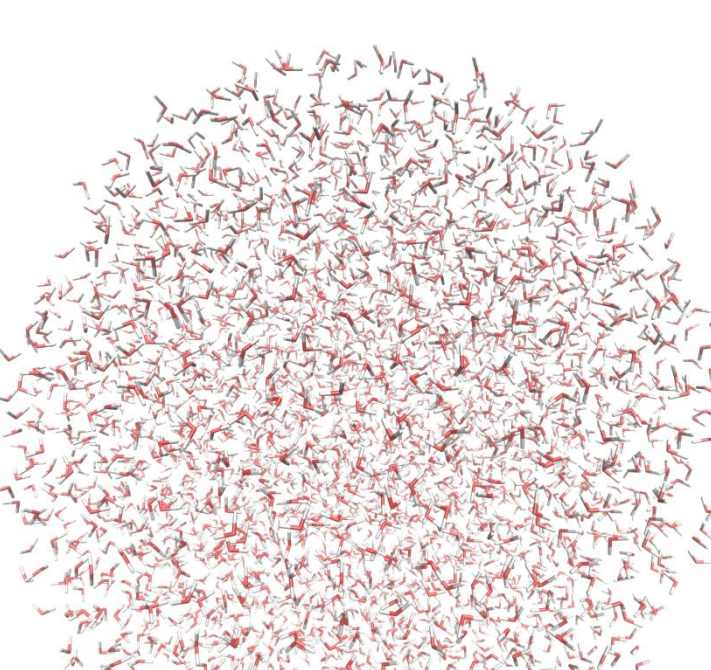
- Each monomer or dimer calculation is a regular quantum chemistry calculation which is independent on other calculations
- Steps 1-4 are parallelized by mapping calculations on subgroups of processors within the whole compute environment
- GDDI library is used for parallelization

MPI+OpenMP parallelization of one-electron integrals calculation

```
1: !$omp parallel shared(S,T,V)
2: for i = 1, NumShells do
3:   call ddi.dlbnxt(i)
4:   !$omp do schedule(dynamic)
5:   for j = 1, i do
6:     ▶ Loops over primitives:
7:     NumPrimi, NumPrimj ← basis
8:     for p = 1, NumPrimi do
9:       for q = 1, NumPrimj do
10:        calculate exppq
11:        if exppq < cutoff cycle q-loop
12:        spq ← overlap(p,q)
13:        tpq ← kinetic(p,q)
14:        ▶ Compute nuclear attraction integrals vpq:
15:        vpq = 0
16:        for nuc = 1, NumNuclei do
17:          vpq += nuclear(p,q,nuc)
18:        end for
19:        ▶ Update corresponding one-electron matrices:
20:        Sij += spq; Tij += tpq; Vij += vpq
21:      end for
22:    end for
23:  !$omp end parallel
  ▶ Reduce S, T and V matrices over MPI processes:
24:  call ddi.gsum(S, T, V)
  ▶ Compute one-electron Hamiltonian matrix:
25:  H = T + V
```

Figure 4. Hybrid MPI+OpenMP algorithm of one-electron integrals calculation

Benchmark description



For benchmarks, water chemical system was chosen:

- Water usually contributes a lot to FMO calculations
- Problem size can be easily tuned
- Spherical water droplet with 30 Å radius was used in all benchmarks
- Gaussian basis set cc-pVTZ

Two setups to test FMO parallelization

- FMO1 single iteration
 - Quite simple calculation – each fragment is computed only once
 - Verify that new algorithm is scaling
 - Used extensively during development and debugging
 - KNL partition on the Stampede 2 supercomputer was used
- FMO2 full calculation
 - Iterative FMO1 runs (until convergence) + single dimer calculation, much more subtasks than in FMO1 calculation
 - Large-scale benchmarking

Conclusions

In this work, conversion of the MPI-only GAMESS FMO code to hybrid MPI+OpenMP versions is described. The resulting hybrid implementations are benchmarked to exhibit improvements in the time-to-solution compared to the original MPI-only version. Our new hybrid MPI+OpenMP code significantly outperform the legacy MPI-only code in GAMESS. It scales fairly good up to 2,048 of KNL nodes and runs up to 2 times faster than MPI-only version.

Benchmark results

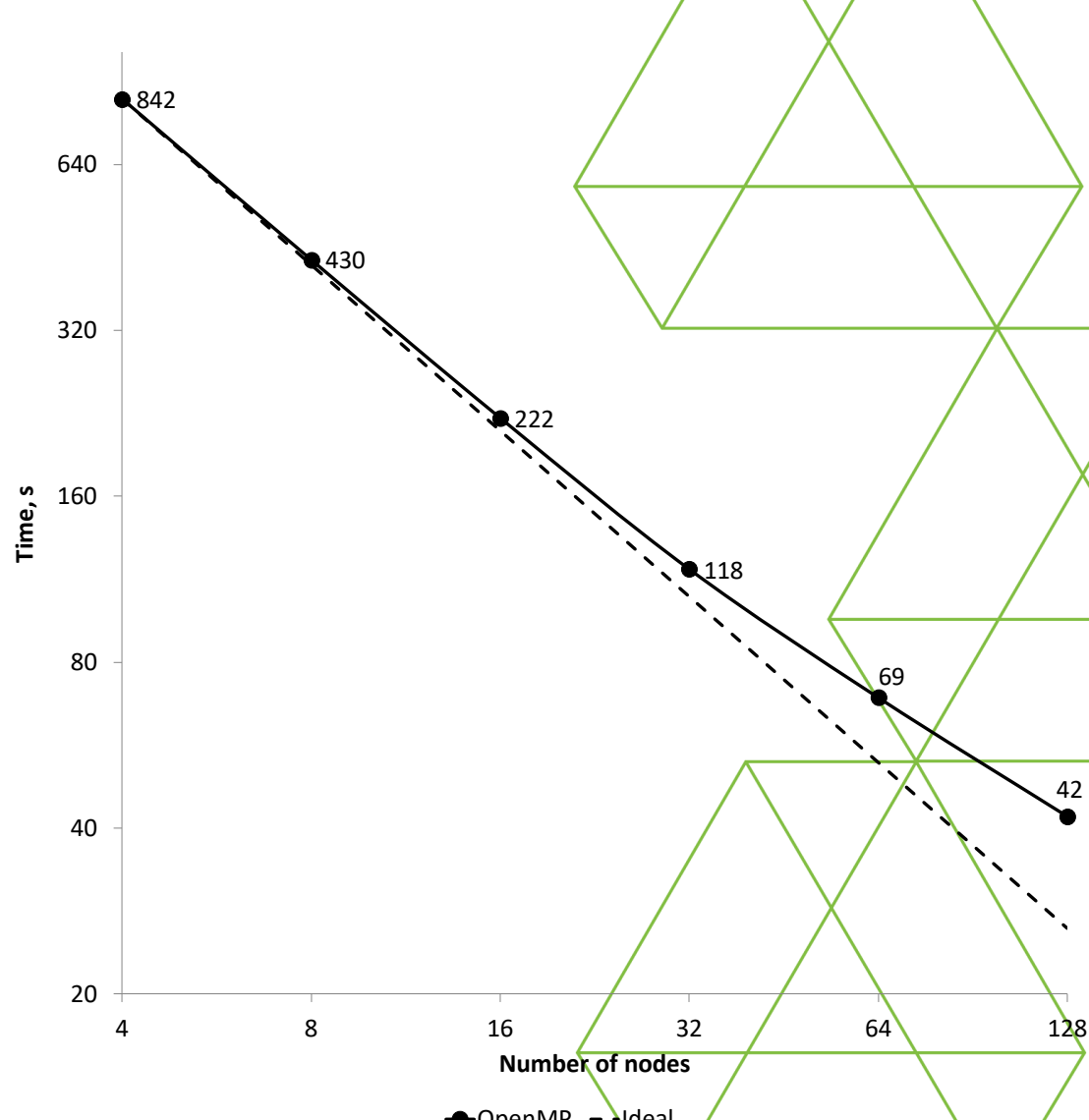


Figure 5. Parallel efficiency of the hybrid MPI+OpenMP algorithm. Times in seconds correspond to the single FMO1 iteration on Stampede2 supercomputer.

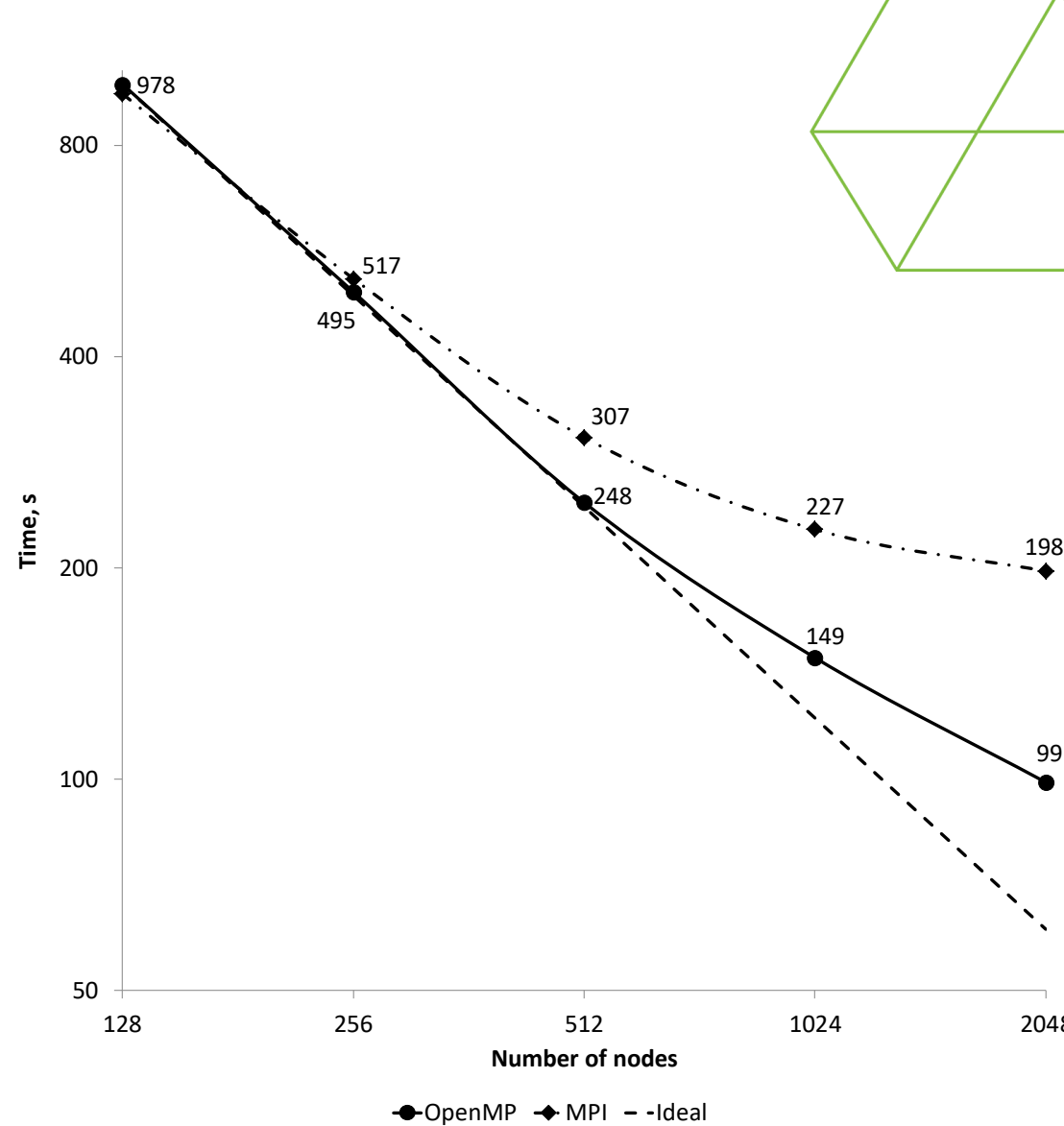


Figure 5. Comparison of the new MPI+OpenMP and legacy implementations of the FMO method in GAMESS. Times in seconds correspond to the complete FMO2 simulation on Theta supercomputer.