Large-Scale MO Calculation with GPU-accelerated FMO Program

Center for Computational Sciences, University of Tsukuba

Hiroaki Umeda[1*], Toshihiro Hanawa[2], Mitsuo Shoji[1], Taisuke Boku[1], and Yasuteru Shigeta[1]

[1]Center for Computational Sciences, University of Tsukuba, Tsukuba, Ibaraki, Japan. [2]Information Technology Center, The University of Tokyo, Kashiwa, Chiba, Japan.

*E-mail address: *umeda@ccs.tsukuba.ac.jp*

Introduction

Ab initio molecular orbital (MO) calculation is a standard tool to obtain molecular properties and to analyze reaction mechanisms in quantum chemistry, and it is wanted to perform MO calculation much faster for large-scale molecule, such as proteins. Fragment molecular orbital (FMO) method is one of such MO calculation technique. Besides this computational technique, it is also required to execute MO calculation on modern computer architecture, such as an accelerator computer, in order to realize large-scale MO calculation.

In this presentation, we describe our implementation of GPU-enable FMO program, and evaluate performance of GPU-accelerated FMO calculation on GPU cluster, including the first large-scale GPU-accelerated FMO calculation for 23,460 and 46,641 atomic molecular systems.

HA-PACS GPU cluster @CCS, Univ. of Tsukuba

HA-PACS Base-cluster (268 nodes, 802TFLOPS)
- CPU: 2 Intel E5-2680 2.6GHz (8core), 20.8GFLOPS/core
- GPU: 4 NVIDIA Tesla M2090 (Fermi), 665GFLOPS/GPU
- Network: 2xQDR IB

HA-PACS/TCA (64 nodes, 364TFLOPS)
- CPU: 2 Intel E5-2680v2 2.8GHz (10core), 22.4GFLOPS/core
- GPU: 4 NVIDIA Tesla K20Xm (Kepler), 1.23TFLOPS/GPU
- Network: 2xQDR IB

HA-PACS GPU cluster system
(Left: HA-PACS/TCA, Right: base-cluser)

Softwares:
Intel C Compiler,
15.0CUDA Toolkit 6.5,
Intel MPI 5.0

OpenFMO

OpenFMO is a compact (simple) FMO program, targeting for massively parallel computer, and have been developed by Y. Inadomi (Kyushu Univ.) et al.

Codes

» C program (~50,000 lines)
» OpenMP+MPI parallelization
» Running on PC cluster, K-computer, and *GPGPU cluster*
» Will be opened. (http://www.openfmo.org)

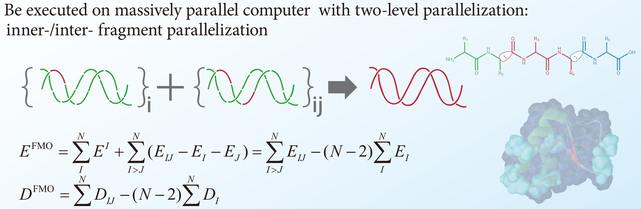
Master-worker execution model

» master process, worker groups, data server
» MPI Dynamic Process Management or Falanx programming middleware

FMO Method

The FMO (fragment molecular orbital) method [1] is one of the computational technique to perform large MO calculation, and is expected to be a quantum chemistry application on the modern parallel computer system. The FMO calculation can

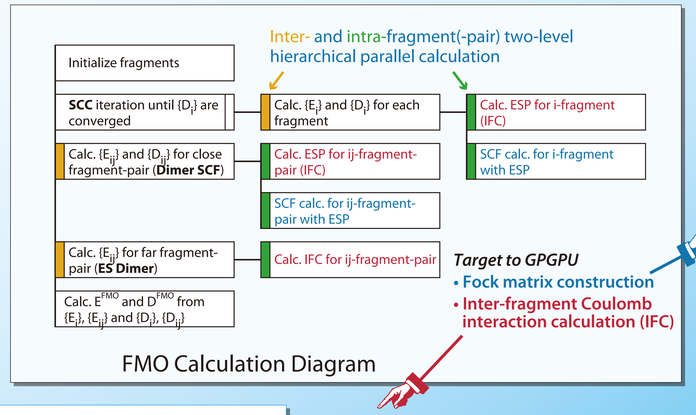
- Avoid costly Fock matrix calculation $O(N^4)$ for a entire molecule by following steps;
  - » Divides molecule into many fragments,
  - » Determine environmental electrostatic potential (ESP) by self-consistent charge (SCC) iteration,
  - » Reconstructs entire properties from MO calculations of fragments and fragment-pairs with ESP
- Evaluate Interaction energy analysis between fragments
- Be executed on massively parallel computer with two-level parallelization: inner-/inter- fragment parallelization



$$E^{FMO} = \sum_{I}^{N} E^{I} + \sum_{I>J}^{N} (E_{IJ} - E_I - E_J) = \sum_{I>J}^{N} E_{IJ} - (N-2)\sum_{I}^{N} E_I$$

$$D^{FMO} = \sum_{I>J}^{N} D_{IJ} - (N-2)\sum_{I}^{N} D_I$$

[1] K. Kitaura et al., *Chem. Phys. Lett.*, **312**, 319 (1999).



FMO Calculation Diagram

4C-IFC Calculation with GPGPU

The electrostatic potential is calculated as a sum of inter-fragment Coulomb interaction (IFC) between the focused fragment (fragment-pair) A and all other fragments. And the hotspot is four-center IFC (4C-IFC) calculation for neighboring fragments. The 4C-IFC calculation is described as similar two-folded loops as that of Fock matrix preparation, except index ijcs and klcs run on different fragments, and only Coulomb interaction term is accumulated. Therefore, the same approach has been applied to 4C-IFC code for CUDA implementation.

5x speedups by CPU&GPU overlapping calculations

```
bidx = blockIdx.x; nblk = gridDim.x;
tidx = threadIdx.x; nth = blockDim.x;
while ((ijcs=get_ijcs()) < Nijcs) {
  double Vij[]=0.0;
  survived_klcs[]=make_sklcs(ijcs);
  for (sklcs=tidx; sklcs< Nsklcs; sklcs+=nth) {
    klcs = survived_klcs[sklcs];
    x[] = calc_2e_psss(ijcs, klcs);
    for (i=0,iao=iao0; i<3; i++,iao++) {
      Vij[i] += 4*x[i]*D[kao][lao];
    }
  } // klcs
  __syncthreads();
  BlockReduce(Vij[]);
  if (tidx==0)
    for (i=0; j<3; i++) {V[iao][jao] += Vij[i];}
  }
} // ijcs
```

ijcs index of fragment A is dynamically allocated to thread blocks

klcs index of fragment B is allocated to threads as round-robin fashion

ESP array V[] can be shared within a GPU board without atomic operation

Pseudo CUDA kernel code of 4C-IFC calculation between fragment A and fragment B

2C-, 3C-IFC calculations have not been implemented with GPGPU, yet.

Performance Benchmark

Speedups from CPU only calculation

Performance evaluation is examined by FMO-HF/6-31G(d) calculations of lysozyme molecule with HA-PACS GPU cluster. The calculations are performed with 15 worker groups of 2 ranks. (1 rank = 4 (5 for TCA) CPU core and 1 GPU).

3.8x speedups for FMO calculation than CPU on-the-fly calculation

Elapsed time [sec.] for FMO-HF/6-31G(d) calculation of lysozyme

HA-PACS	Base cluster					TCA
	CPU		CPU+GPU		Speedups	CPU+GPU
	On-the-fly	Buffered	Buffered	On-the-fly	On-the-Fly	On-the-Fly
#node (#GPU)	8	8	8 (32)	8 (32)		8 (32)
SCC [s]	3,070.5	3,026.0	905.0	827.5	3.7	457.6
Dimer SCF [s]	6,246.2	4,968.8	1,725.6	1,674.8	3.7	960.8
ES Dimer [s]	407.3	407.3	77.8	77.8	5.2	39.8
Total [s]	9,770.5	8,408.5	2,723.8	2,596.8	3.8	1,498.9

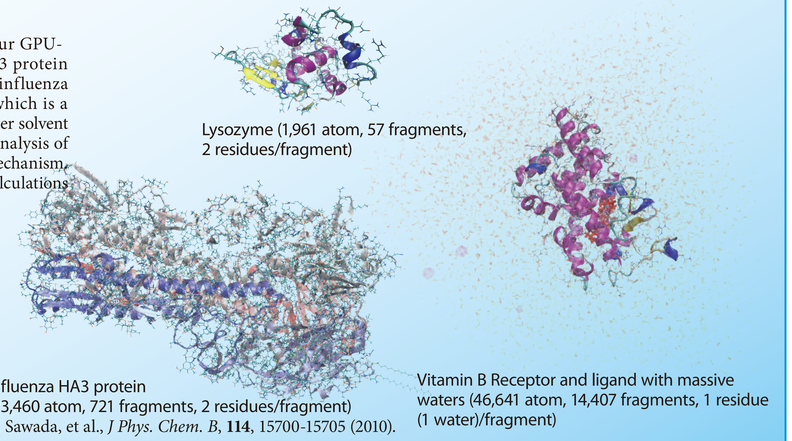
Large-scale FMO applications

Two large-scale FMO calculation are examined with our GPU-enable FMO program. First application is influenza HA3 protein (HA3), which is considered as acting important roles with influenza virus. Second application is vitamin B receptor (VBR), which is a member of nuclear receptor, and a binding ligand within water solvent (46,641 atoms in total). The pair interaction energy (PIE) analysis of FMO calculation could be helpful to analyze their binding mechanism.

Both applications are performed as FMO-HF/6-31G(d) calculations on HA-PACS base cluster.

Elapsed time [hr.] for FMO-HF/6-31G(d) calculation of large molecular systems

HA-PACS	Base cluster	
Application	HA3	VBR
#atoms	23,460	46,641
#fragments	721	14,407
#nodes (#GPU)	64 (256)	128 (512)
#groups	84	510
SCC [hr]	0.52	0.96
Dimer SCF [hr]	0.90	0.27
ES Dimer [hr]	0.45	0.66
Total [hr]	1.97	1.97



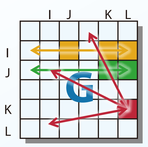
Two hoursfor FMO calculation of 23,460 atomic protein with 256 GPUs

It is difficult to perform efficient GPU-enable MO calculation for small fragment, such as an water molecule.

Fock matrix construction with GPGPU

In order to share Fock (G) matrix within a thread block, Fock matrix preparation algorithm without atomic operation has been developed for the size of FMO fragment-pair HF calculations [2].

- Two-folded loops for ijcs and klcs, which is survived pair-index after overlap screening
- Access pattern classifies three-types in a kl-loop: Gi for i column, Gj for j column, and Gkl
- kl runs only limited area of G matrix, and points different elements of G for a given ij index



Access pattern to G matrix for one 2e-integral (IJ|KL).

```
bidx = blockIdx.x; nblk = gridDim.x;
tidx = threadIdx.x; nth = blockDim.x;
Gkl[]=0.0;
while ((ijcs=get_ijcs()) < Nijcs) {
  Gi[]=0.0; Gj[]=0.0;
  survived_klcs[]=make_sklcs(ijcs);
  for (sklcs=tidx; sklcs< Nsklcs; sklcs+=nth) {
    klcs = survived_klcs[sklcs];
    x[] = calc_2e_psss(ijcs, klcs);
    for (i=0,iao=iao0; i<3; i++,iao++) {
        Gi[i*nao+jao] += 4*x[i]*D[kao][lao];
        Gkl[klcs] += 4*x[i]*D[iao][jao];
        Gi[i*nao+kao] -= x[i]*D[jao][lao];
        Gi[i*nao+lao] -= x[i]*D[jao][kao];
        Gj[kao] -= x[i]*D[iao][lao];
        Gj[lao] -= x[i]*D[iao][kao];
    }
  } // klcs
  __syncthreads();
  BlockReduce(Gi[]); BlockReduce(Gj[]);
  for (j=tidx; j<nao*3; j+=nth) {G[iao0][j] += Gi[j];}
  for (i=tidx; i<nao; i+=nth) {G[jao][i] += Gj[i];}
} // ijcs
__syncthreads();
for (klcs=tidx; klcs<Nklcs; klcs+=nth) {
  G[kao][lao] += Gkl[klcs];
}
```

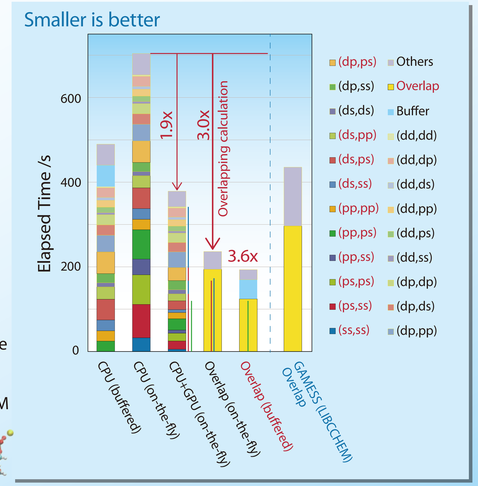
Arrays G[], and Gkl[] are shared within a thread block

Arrays Gi[], and Gj[] are allocated as private for each thread on global memory

Sum up Gi[]s, Gj[]s for all threads after synchronize, following by adding to the block shared G[]

Pseudo CUDA kernel code for Fock matrix preparation without atomicAdd operation.

Elapsed time for HF/6-31G(d) calculation of 126 atomic molecule (1,282AO) with a single HA-PACS base-cluster node w/wo overlap. cf.) GAMESS(2013May) + LIBCCHEM with CPU&GPU.



3.6x speedups by CPU&GPU overlapping calculations

[2] H. Umeda et al., *IPSJ Transactions on Advanced Computing Systems*, **6**,4, 26-37 (2013); H. Umeda et al., *HPC in Asia poster presentation in ISC'13*, 19a (2013).

Summary

We have implemented GPU-accelerated FMO program and perform several benchmark calculations on HA-PACS GPU cluster. The performance of our code shows 3.8 times faster than CPU on-the-fly calculation for middle-sized protein. Large-scale FMO application of 23,460 atomic protein is also performed and successfully calculated in 2 hours with 256 GPUs.

Acknowledgments: This study is partially supported by the project "Research and Development on Unified Environment of Accelerated Computing and Interconnection for Post-Petascale Era" of Basic Research Programs: CREST "Development of System Software Technologies for post-Peta Scale High Performance Computing," Japan Science and Technology Agency, and FMO benchmarks have been carried out under the "Interdisciplinary Computational Science Program" in CCS, University of Tsukuba. The model of Influenza HA3 protein is provided by T. Sawada, PhD. (AIST, Fujifilm Corporation).

筑波大学 University of Tsukuba

Japan Science and Technology Agency