Direct Self-Consistent Field Computations on GPU Clusters

Guochun Shi,
Volodymyr Kindratenko
National Center for Supercomputing Applications
University of Illinois at Urbana-Champaign

Ivan Ufimtsev,
Todd Martinez
Department of Chemistry
Stanford University

National Center for Supercomputing Applications
University of Illinois at Urbana-Champaign
Presentation Outline

GPU computing
NCSA’s Lincoln GPU cluster
SCF theory in Quantum Chemistry
Implementation on a GPU cluster
Kernels for J and K matrices
Parallelization strategy for GPU cluster
Performance
Conclusions and future work
Why GPUs?

GPU Performance Trends

5800 5950 Ultra 6800 Ultra 7800 GTX
NVIDIA Tesla T10 GPU Architecture

**TPC 1**
- Geometry controller
- SM
- Shared memory
- Texture units
- Texture L1

**TPC 10**
- Geometry controller
- SM
- Shared memory
- Texture units
- Texture L1

**T10 architecture**

- 240 streaming processors arranged as 30 streaming multiprocessors

At 1.3 GHz this provides

- 1 TFLOP SP
- 86.4 GFLOP DP

512-bit interface to off-chip GDDR3 memory

- 102 GB/s bandwidth
Intel 64 Tesla Linux Cluster

Dell PowerEdge 1955 server

Intel 64 (Harpertown) 2.33 GHz dual socket quad core

16 GB DDR2

Infiniband SDR

Tesla S1070 1U GPU Computing Server

1.3 GHz Tesla T10 processors

4x4 GB GDDR3 SDRAM

Cluster

Servers: 192

Accelerator Units: 96
HPL Benchmark for Lincoln

We used Massimiliano Fatica (nvidia)'s GPU enabled HPL package.
Quantum Chemistry

Why do we need to deal with…

Energy (\(H = E\):)
  - Quantifies intra/intermolecular interactions
  - Drives chemistry, little interesting happens on flat surface

Geometry optimization (\(RE = 0\))
  - Searches for stable atomic arrangements (molecular shapes)

Molecular dynamics (\(\frac{\partial^2 R}{\partial t^2} = -\frac{1}{M} \cdot RE\))
  - The chemistry itself (at some, sometimes crude, approximation)
  - Studies system at atomistic time, and length scales
Exact energy is a hard problem

\[ \Psi(r_i) = ? \]
\[ E = ? \]
\[ \left\{ -\frac{1}{2} \sum_i \left( \frac{\partial^2}{\partial x_i^2} + \frac{\partial^2}{\partial y_i^2} + \frac{\partial^2}{\partial z_i^2} \right) - \sum_{i,A} \frac{Z_A}{|r_i - R_A|} + \sum_{i,j} \frac{1}{|r_i - r_j|} \right\} \Psi(r_i) = E \Psi(r_i) \]
Hartree-Fock approximation is one of the simplest

is an antisymmetrized product of $N$ 1-electron orbitals

$$\Psi = A \left[ \psi_1(r_1) \psi_2(r_2) ... \psi_N(r_N) \right]$$

Expand over predefined basis set

$$\psi_i(r) = \sum_{j=1}^{K} C_{ij} \varphi_j(r)$$

$$\Psi \leftrightarrow C_{ij} = ?$$
Hartree-Fock Self Consistent Field (SCF) procedure

\[ F(C)C = \text{ESC} \]

\[ F_{k+1}(C) = F(C_k) \]

\[ F_{k+1}C_{k+1} = \text{ESC}_{k+1} \]

Repeat until \( C_{k+1} \) more or less equals \( C_k \)
Hartree-Fock equations

\[ F(C)C = ESC \]

\[ F_{ij}(C) = H_{ij}^{\text{core}} + J_{ij}(C) - \frac{1}{2} K_{ij}(C) \]

\[ J_{ij} = \sum_{k,l} [ij | kl] P_{kl}(C) \]

\[ K_{ij} = \sum_{k,l} [ik | jl] P_{kl}(C) \]

\[ [ij | kl] = \int \int \varphi_i(r_1)\varphi_j(r_1) \frac{1}{|r_1 - r_2|} \varphi_k(r_2)\varphi_l(r_2) d\mathbf{r}_1 d\mathbf{r}_2 \]

\cdot All matrices are of \( N \times N \) size (\( N \sim 1,000 \ldots 10,000 \))
\cdot \( N^3 \) operations to solve HF equations (need to deal with diagonalization)
\cdot \( N^4 \) operations to get \( F \)
2e integral grid

\[
[ij \mid kl] = \int \int \varphi_i(r_1) \varphi_j(r_1) \frac{1}{|r_1 - r_2|} \varphi_k(r_2) \varphi_l(r_2) dr_1 dr_2
\]

\[
[ij \mid kl] \leq \sqrt{[ij \mid ij]} \sqrt{[kl \mid kl]} \geq 10^{-11}
\]

leaves only \( N^2 \) out of \( N^4 \) integrals

|kl|

---

**Original integral grid**

**Presorted integral grid**

**SIMD warp**

Most negligibly small integrals will be calculated

Only significant integrals will be calculated
Kernel in GPU: J-matrix implementation

\[ J_{ij} = \sum_{k,l} [ij \mid kl] P_{kl} \]

\[ [ij \mid kl] \leq \sqrt{[ij \mid ij]} \sqrt{[kl \mid kl]} \]

[p|q] Hermite primitive integral grid
Each row leads to one J[p] matrix element

IPDPS 200
Kernels in GPU: K-matrix implementation

\[ K_{ij} = \sum_{k,l} [ik \mid jl] P_{kl} \]

One block scans the whole data segment leading to one K-matrix element. Blue arrows represent the actual block’s trace as it scans this segment row-by-row. In the end, all integrals calculated by all threads are summed through subsequent row- and column-wise interblock and reduction leading to one K-matrix element.
The J and K matrices computation and Linear Algebra (LA) computation dominate the overall execution time.

- Pair quantity computations can be significant.
GPU cluster parallelization strategy

Each GPU has a global id
	nodeid * num_gpu_per_node + local_gpu_index

J/K matrices work distribution

Computations for elements in J and K matrices are not even.

Sort pre-computed pair quantities and choose every one element in N to compute for each GPU

LA using intel MKL
Parallelization strategy (II)

- Start as MPI program, each node has as many MPI processes as CPU cores
- One MPI process per node is designated as “master”
- The master MPI processes create threads for controlling GPUs as well as CPU work threads
- MPI processes/GPU management threads/CPU work threads are awakened or put to sleep as needed
Computing J and K matrices on GPUs

Reduction of J and K matrices, form the Fock matrix

Pair-quantity computing on CPU
Using density matrices P

Distribute the Fock matrix, do linear algebra, compute matrix C and P, gather P

Generated guess matrix C and compute matrix P

MPI process
CPU work thread
CPU thread for managing GPU kernels

MPI process
CPU work thread
CPU thread for managing GPU kernels

Partial J and K

Fock matrix

Distr-ed fork matrix
Distr-ed P matrix

P matrix

Broadcast P

IPDPS 200
Performance: load balancing

- Sorting for pair quantity computations and work selection strategy makes the computation on GPUs well balanced, reducing performance degradation
Performance

<table>
<thead>
<tr>
<th></th>
<th>Atoms</th>
<th>Electrons</th>
<th>Orbitals</th>
<th>S shells</th>
<th>P shells</th>
</tr>
</thead>
<tbody>
<tr>
<td>Olestra</td>
<td>453</td>
<td>1366</td>
<td>2131</td>
<td>1081</td>
<td>350</td>
</tr>
<tr>
<td>BPTI</td>
<td>875</td>
<td>3400</td>
<td>4893</td>
<td>2202</td>
<td>897</td>
</tr>
<tr>
<td>CspA</td>
<td>1732</td>
<td>6290</td>
<td>8753</td>
<td>4220</td>
<td>1511</td>
</tr>
</tbody>
</table>

- Olestra
- BPTI
- CspA

Using 321g basis set

# of nodes

Runtime (s)

IPDPS 200
J and K matrices computation can scale well to 128 nodes

Linear Algebra scales only up to 16 nodes even for CsPA molecule
Performance: Linear Algebra breakdown

- Diagonization scales the worst, dgemm is also important
- A fast, scalable GPU based SCALAPACK is needed
  - Magma from UTK?
  - Cula?
Results: Olestra molecule

Olestra molecule consisting of 453 atoms (a small example model used of testing the developed software) can be computed by the state-of-the-art quantum chemistry software package GAMESS running on an Intel Pentium D 3 GHz processor in over 12,408 seconds whereas our 8-node GPU cluster implementation performs the same computation in just over 5 seconds, a 2,452× speedup.
Example: CspA molecule

For larger models, one SCF iteration for Cold shock protein A (CspA) molecule consisting of 1,732 atoms can be done in 88 seconds on a 16 node GPU cluster.
Conclusions and future work

GPU computing brings Quantum Chemistry computing to a new level

Parallelization enables computing of large molecules in shorter time

J and K matrices show good scalability

Linear Algebra can only scale up to 16 nodes

Linear Algebra becomes a major bottleneck

A linear algebra package using GPUs with good scalability is needed

- Matrix multiplication and eigenvalue solver

Only S and P orbitals are supported at this moment.
Acknowledgement

This work was supported by the National Science Foundation grant CHE-06-26354.