Fast Quantum Molecular Dynamics on Multi-GPU Architectures in LATTE

S. Mniszewski*, M. Cawkwell, A. Niklasson
GPU Technology Conference
San Jose, California
March 18-21, 2013

*smm@lanl.gov
Background: Quantum Molecular Dynamics

- In molecular dynamics simulation, the relative positions of atoms evolve over a series of time steps according to the force acting on each atom.
- Employed in materials science, chemistry, and biology to study structures, defects, and equilibrium and non-equilibrium phenomena.
- Dependence on an interatomic potential to calculate forces and energy.
- Quantum-based models capture the making and breaking of covalent bonds, charge transfer between species of differing electronegativities, and long-range electrostatic interactions.
Quantum-based Interatomic Potentials

- Electronic structure of atoms and molecules is modeled explicitly
- Most accurate and reliable descriptions of interatomic bonding
- Their prohibitive computational cost has prevented widespread use – better algorithms and GPU architectures are important paths forward

- Hamiltonian matrix $H$
- The density matrix, rho, is computed self-consistently from $H$

Energy

$$E = 2 \text{Tr} \left[ \rho H \right]$$

Force

$$f_i = -2 \text{Tr} \left[ \rho \frac{\partial H}{\partial R_i} \right]$$
The Density Matrix Computation

- Typically, algorithms used in quantum-based models, most notably matrix diagonalization, are not ideally suited to GPUs
  - Due to their complexity
  - Difficulty in extracting thread-level parallelism
  - Difficulty of avoiding branching within warps

- New approach in LATTE
  - Computed directly from the Hamiltonian through a recursive expansion of the Fermi Operator with the second order spectral projection (SP2) algorithm
  - Based on a series of generalized matrix-matrix multiplications
  - Only one matrix-matrix multiplication is required per iteration
  - Maps very well to GPUs
The Second Order Spectral Projection Algorithm (SP2) – Reduced Complexity

Recursive Fermi Operator expansion

\[ \rho = \theta \left[ \mu I - H \right] = \lim_{i \to \infty} f_i[f_{i-1} \ldots f_0[X_0] \ldots] \]

\[ X_0 = \frac{\epsilon_{\text{max}} I - H}{\epsilon_{\text{max}} - \epsilon_{\text{min}}} \]

\[ f_i[X_i] = \begin{cases} X_i^2 & \text{if } 2 \text{Tr}[X_i] \geq N \\ 2X_i - X_i^2 & \text{if } 2 \text{Tr}[X_i] < N_e \end{cases} \]
The GPU Implementation

- **Part of the LATTE codebase**
  - Employs a semi-empirical tight-binding model of interatomic bonding that is based on the formalisms derived from density functional theory
  - Density matrix build is by far the slowest step in the calculation
  - CPU version in Fortran 90

- **Hardware/Software Architecture**
  - Keeneland* cluster at the National Institute for Computational Sciences
  - CPU - 2 Intel hex-core Xeon CPUs per node, Intel Fortran Compiler, MKL
  - 3 Nvidia M2090 GPUs (previously M2070 GPUs)
  - CUDA 4.2, CUBLAS, and a thread block size of 1024

- **Use of CUDA Features on GPUs**
  - Unified Virtual Addressing
  - Peer to peer memory access/copy
  - Streams –sequence of commands
  - Single thread access to all GPUs

---

**Estimate $\varepsilon_{\text{max}}$ and $\varepsilon_{\text{min}}$**

\[ X = (\varepsilon_{\text{max}} I - H)/(\varepsilon_{\text{max}} - \varepsilon_{\text{min}}) \]

**Trace**

\[ \text{Trace}X = \text{Tr}[X] /* \text{Trace kernel on GPU} */ \]

**Until converged do**

\[ X_{\text{tmp}} = X \]

\[ X_{\text{tmp}} = X^2 + X_{\text{tmp}} /* \text{CUBLAS xGEMM} */ \]

\[ \text{Trace}X_{\text{tmp}} = \text{Tr}[X_{\text{tmp}}] /* \text{Trace kernel on GPU} */ \]

**if** $|2\text{Trace}X - 2\text{Trace}X_{\text{tmp}} - N_e| > |2\text{Trace}X + 2\text{Trace}X_{\text{tmp}} - N_e|$

\[ X = X + X_{\text{tmp}} \]

\[ \text{Trace}X = \text{Trace}X + \text{Trace}X_{\text{tmp}} \]

**else**

\[ X = X - X_{\text{tmp}} \]

\[ \text{Trace}X = \text{Trace}X - \text{Trace}X_{\text{tmp}} \]

**end until**

\[ \rho = X \]
SP2 Algorithm Using the Full GPU Approach

Estimate $\varepsilon_{\text{max}}$ and $\varepsilon_{\text{min}}$

$X = (\varepsilon_{\text{max}} I - H) / (\varepsilon_{\text{max}} - \varepsilon_{\text{min}})$

TraceX = Tr[X] /* Trace kernel on GPU */

Until converged do

$X_{\text{tmp}} = X$

$X_{\text{tmp}} = X^2 + X_{\text{tmp}}$ /* CUBLAS xGEMM */

TraceX_{tmp} = Tr[X_{tmp}] /* Trace kernel on GPU */

if $|2 \text{TraceX} - 2 \text{TraceX}_{\text{tmp}} - N_e| > |2 \text{TraceX} + 2 \text{TraceX}_{\text{tmp}} - N_e|$

$X = X + X_{\text{tmp}}$ /* CUBLAS xAXPY */

TraceX = TraceX + TraceX_{tmp} /* CUBLAS xAXPY */

else

$X = X - X_{\text{tmp}}$ /* CUBLAS xAXPY */

TraceX = TraceX - TraceX_{tmp}

end until

$\rho = X$
CUBLAS Matrix Multiplication Performance (Nvidia M2070)

Array Padding for Performance (M \times M)

Average time to execute the CUBLAS xGEMM generalized matrix-matrix multiplication for $M \times M$ matrices. (a) DGEMM, (b) SGEMM. The broken and solid lines correspond computations performed with and without the padding of the arrays, respectively.
Performance Analysis: Density Matrix Calculation (Nvidia M2070) – Liquid Methane (10-1000 molecules)

![Graph showing time per \( \rho \) calculation (s) vs. number of orbitals for different algorithms. The graph includes lines for Diagonalization, SP2: CPU algorithm 1, SP2: CPU/GPU algorithm 2, and SP2: GPU algorithm 3. The y-axis is on a logarithmic scale, and the x-axis is linear. The graph shows the performance for single and double precision calculations.]
The SP2 algorithm has errors that are independent of system size whereas traditional diagonalization yields errors that increase with the number of atoms.
Multi-GPU Generalized Matrix-Matrix Multiplication

- Using multiple streams for sub-block matrix-matrix multiplications, additions, and matrix traces
- Efficient reassembly of blocked matrix via native functionality of CUBLAS DGEMM/SGEMM
Using Streams for SP2 Sub-block Matrix Operations

Matrix blocks assembled in GPU 0 and redistributed
Performance Analysis: Density Matrix Calculation
(Nvidia M2090) – Liquid Methane (10 – 1250 molecules)
Performance Analysis for 1-3 GPUs

The graph shows the time per density matrix build (in seconds) as a function of matrix dimension for 1, 2, and 3 GPUs. The y-axis represents the time in seconds, ranging from 0.001 to 100, while the x-axis represents the matrix dimension ranging from 100 to 10,000. The graph includes three curves: 1 GPU (black), 2 GPUs (red), and 3 GPUs (green). The optimal number of GPUs for each range is indicated:

- 1 GPU optimal
- 2 GPUs optimal
- 3 GPUs optimal
Summary

- GPUs can be effectively used for the density matrix computation in quantum mechanical models
- The recursive SP2 algorithm is well suited to the GPU architecture
- Transfer of arrays between the CPU and GPU are a minor performance contribution
- Array padding is important for performance
- The GPU version of the SP2 algorithm provides comparable or better accuracy over traditional diagonalization
- Massive speed-ups with respect to traditional algorithms have been seen with no loss of accuracy
Related Publications