Fast Binding Site Mapping using GPUs and CUDA

Bharat Sukhwani       Martin C. Herbordt

Computer Architecture and Automated Design Laboratory
Department of Electrical and Computer Engineering
Boston University

http://www.bu.edu/caadlab

* This work supported, in part, by the U.S. NIH/NCRR
Why Bother?

Problem: Combat the bird flu virus

Method: Inhibit its function by “gumming up” Neuraminidase, a surface protein, with an inhibitor
  - Neuraminidase helps release progeny viruses from the cell.

Procedure*:
  - Search protein surface for likely sites
  - Find a molecule that binds there (and only there)

Binding site mapping:
  - Very compute intensive: Usually run on clusters
  - GPU based desktop alternative

#From New Scientist  www.newscientist.com/channel/health/bird-flu
Outline

- Overview of Binding Site Mapping
  - Rigid Docking
  - Energy Minimization

- Overview of NVIDIA GPUs / CUDA

- Rigid Docking on GPU

- Energy Minimization on GPU

- Results
Binding Site Mapping

Purpose: Identification of hot spots

Significance: Very effective for drug-discovery

Rationale:
- Hot spots are major contributors to the binding energy
- They bind a large variety of small molecules

Process: Docking small probes
- Rigid Docking
- Energy Minimization
Mapping: Two Step Process

- Rigid Docking of Probes into Protein
  - Grid-based computation
  - Exhaustive 6D search
  - Find an approximate conformation

- Local refinement – Energy Minimization
  - Model the flexibility in the side-chains
FTMap*

- 16 small molecule probes

- Dock each probes into the protein
  - 500 rotations – $10^6$ translations per rotation
  - 30 minutes on a single CPU

- Energy minimize 2000 conformations per protein-probe complex
  - Up to 30 seconds per conformation
  - 16 hours per probe!

Outline

- **Overview of Binding Site Mapping**
  - Rigid Docking
  - Energy Minimization

- **Overview of NVIDIA GPUs / CUDA**

- Rigid Docking on GPU

- Energy Minimization on GPU

- Results
NVIDIA GPU Architecture

NVIDIA Tesla C1060 Architecture

- 30 Multiprocessors
- 240 Processor cores
- 4 GB Device memory
- 1.3 GHz Clock

* Source: NVIDIA Corporation

8 SPs per SM
Memory Hierarchy

* Source: NVIDIA Corporation
CUDA Programming Model

Threads within a block can be synchronized

Different blocks must be independent

* Source: NVIDIA Corporation
Outline

- Overview of Binding Site Mapping
  - Rigid Docking
  - Energy Minimization
- Overview of NVIDIA GPUs / CUDA
- Rigid Docking on GPU
- Energy Minimization on GPU
- Results
Rigid Docking: Procedure

- **Protein**
- **Probe**
- **Rotation**
- **Grid Assignment**
- **Pose Score: 3D FFT Correlation**
- **Scoring and Filtering**
PIPER Rigid Docking Program

- Structural Bioinformatics lab at BU
- Complex energy functions
- Top scorer in CAPRI* challenge

\[ E = E_{shape} + w_2 E_{elec} + w_3 E_{desol} \]

\[ E_{shape} = E_{attr} + w_1 E_{repul} \]

\[ E_{elec} = E_{born} + E_{coulomb} \]

\[ E_{desol} = \sum_{k=0}^{P-1} E_{pairpot}^{(k)} \]

Up to 22 FFT correlations are required

Rigid Docking on GPUs - Correlation

- Direct Correlation (better than FFT!)
  - For small grid sizes
  - Replaces FFT, voxel-voxel summation, IFFT

- Each multiprocessor accesses both the grids
  - Protein grid on the global memory
  - Probe grid duplicated on shared memories

- Multiple correlations together
  - Voxel represents multiple energy functions
Direct Correlation on GPUs

- Shared memory limits the probe size
  - *With 8 correlations – 8 cubed*
  - *Probe grids are typically 4 cubed*

- Multiple rotations together
  - *8 rotations*
  - *Effectively loop-unrolling*
  - *Multiple computations per global memory fetch*
  - *2.7x additional performance improvement*
Direct Correlation on GPUs

- Distribution of work among threads / blocks
  - Scheme 1: Entire 2D-plane to a thread block
  - Scheme 2: Part of the 2D-plane to a thread block
  - Both yield similar results
Scoring and Filtering on GPUs

- **Score Computation**
  - Divide work among different threads
  - Sync and Serialize to find the best-of-the-best
  - Only one multiprocessor utilized

- **Flagging for exclusion**
  - Serial code – Exclusion bit-vector
  - GPU Solution 1 – Exclusion index array
  - GPU Solution 2 – Exclusion bit-vector on GPU global memory
Outline

- Overview of Binding Site Mapping
  - Rigid Docking
  - Energy Minimization
- Overview of NVIDIA GPUs / CUDA
- Rigid Docking on GPU
- Energy Minimization on GPU
- Results
Energy Minimization

- Minimizing energy between two molecules
  - Iterative process
  - Optimization moves
- Used to model flexible side chains

N-body problem with a cut-off

\[ E_{total} = E_{vdw} + E_{elec} + E_{bond} + E_{angle} + E_{torsion} \]

\[ \begin{align*}
E_{vdw} & \quad \text{bonded} \\
E_{elec} & \quad \text{non-bonded} \\
E_{bond} & \quad \text{bonded} \\
E_{angle} & \quad \text{bonded} \\
E_{torsion} & \quad \text{bonded}
\end{align*} \]
Looks like MD, but it’s not

- Performed on a local region
  - Many fewer atoms, typically few thousand
- Much smaller atom neighborhoods
  - Very small cut-off radius
- Move to the next position
  - Coordinate adjustments - No motion / velocity updates
- No cell-lists / efficient filtering
  - Refinement step; close to dest. - small motions
- Neighbor lists are very sparse, with non-uniform distribution
Energy Minimization step of FTMap

FTMap Minimization Step

Energy evaluation phase

absolute time ~ 10 ms per iteration (on a single core)

\[ E_{\text{total}} = E_{\text{vdw}} + E_{\text{elec}} + E_{\text{bond}} + E_{\text{angle}} + E_{\text{torsion}} \]

- bonded
- non-bonded
FTMap Electrostatics Model

\[ E_{\text{total}} = E_{\text{vdw}} + E_{\text{elec}} + E_{\text{bond}} + E_{\text{angle}} + E_{\text{torsion}} \]

Analytic Continuum Electrostatics (ACE)

Atom Self Energy: Electrostatic energy due to the charge itself

\[ E_{i}^{\text{self}} = \frac{q_{i}^{2}}{2\varepsilon_{s}R_{i}} + \sum_{k\neq i} E_{ik}^{\text{self}} \]

\[ E_{ik}^{\text{self}} = \frac{\tau q_{i}^{2}}{\omega_{ik}} e^{-\left(\frac{r_{ik}^{2}}{\sigma_{ik}}\right)} + \frac{\tau q_{i}^{2} \tilde{V}_{k}}{8\pi} \left(\frac{r_{ik}^{3}}{r_{ik}^{4} + \mu_{ik}^{4}}\right)^{4} \]

Pairwise interaction – Generalized Born eqn.: Electrostatic energy due to the presence of other charges

\[ E_{ij}^{\text{int}} = 332 \sum_{j\neq i} \frac{q_{i} q_{j}}{r_{ij}} - 166 \tau \sum_{j\neq i} \frac{q_{i} q_{j}}{r_{ij}} \sqrt{r_{ij}^{2} + \alpha_{i} \alpha_{j} e^{-\left(\frac{r_{ij}^{2}}{4 \alpha_{i} \alpha_{j}}\right)}} \]

Born Radii – depends on \( E_{\text{self}} \)
FTMap Data Structure - Neighbor Lists

- Random updates for second atoms
  - Can’t distribute the atoms list across multiprocessors
- Write conflicts
  - Second atom might appear in multiple lists
- Not suitable for parallel implementations
Energy Minimization on GPU – Challenges

- Little to no data reuse (within and across iterations)
- Small computation per iteration
- Multiple accumulations – self energy of each atom must be computed
- Large data transfer time
- Random updates – write conflicts
- Accumulation requires serialization

Inherent to the algorithm

$$E_i^{\text{self}} = \frac{q_i^2}{2\epsilon_s R_i} + \sum_{k \neq i} E_{ik}^{\text{self}}$$

$$E_{ij}^{\text{int}} = 332 \sum_{j \neq i} \frac{q_i q_j}{r_{ij}} - 166 \tau \sum_{j \neq i} \left( \frac{r_{ij}}{\alpha \alpha_i} \right)$$

Born Radii – depends on $E_{ij}^{\text{self}}$

Architecture related

First Atoms

Second Atoms

Atoms List

Self Energy

Inherent to the algorithm
Neighbor Lists on GPUs

- Separate energy arrays for first and second atoms
  - *Allows parallel updates by multiple threads*
- Multiple copies of arrays for second atom
  - *One in each thread block*
  - *Parallel updates – no conflicts*
- First arrays reduced to single values
  - *Within the shared memory*
- Second atoms arrays merged by moving to global memory
  - *Large copy and accumulation time*
  - *Slow*
Modified Data Structure - Pairs List

- 2D neighbor $\rightarrow$ 1D pair list
  - Each pair contains atom indices and types

- Distribute pairs across multiple threads
  - More uniform work distribution

- Compute partial energies in parallel

- Perform accumulations serially
Pairs List on GPUs – Initial Attempts

- Pairs distributed on different threads
  - Energy of an atom computed in different multiprocessors
  - Serialization during accumulation

- Accumulation on GPU
  - From global memory: Slow

- Accumulation on host
  - Fast, but requires energy arrays to be transferred every iteration
  - 2x-3x speedup
Pairs List on GPUs – Improved Scheme

Pairs list with two changes

- Conflicts due to random occurrence of second atoms
  - Split forward and reverse pair list
  - Process only the first atom of each list

- Indeterminate distribution requires serialization during accumulation
  - Statically map the pairs onto GPU threads
  - New data structure: Assignment tables
Split Pairs List

- **Forward list**: Same as before
- **Reverse list**: Treat every second atom as a first atom
- **Process only the first atoms of each list**
- **Adds determinism => Better distribution**

<table>
<thead>
<tr>
<th>Pair id</th>
<th>Atom 1</th>
<th>Atom 2</th>
<th>Self energy</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

**Forward List**

**Reverse List**
Static Mapping - Assignment Table

- Pairs can be grouped by first atom
- Groups mapped to different thread blocks
  - Look for next block with enough threads
- One pair per thread (multiple if $N_{pair} > N_{threads}$)
- Reverse Assignment table for the second atoms

<table>
<thead>
<tr>
<th>Thread</th>
<th>Pair Id</th>
<th>Atom 1</th>
<th>Atom 2</th>
<th>Master</th>
<th>Num. Atoms</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>Thread Block 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0</td>
<td>11</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>0</td>
<td>14</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>9</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Thread Block 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>1</td>
<td>5</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td>2</td>
<td>15</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>9</td>
<td>8</td>
<td>2</td>
<td>12</td>
<td>0</td>
<td>3</td>
</tr>
</tbody>
</table>

Unused threads used by next group
Does not fit on TB_0
Computing and Accumulating Energies

- Threads store partial energies in shared memory
  - Address = Local Thread Id

- Master thread performs accumulation
  - ‘N’ locations starting from its thread id

- Multiple parallel accumulations per thread block (from shared memory)
Outline

- Overview of Binding Site Mapping
  - Rigid Docking
  - Energy Minimization
- Overview of NVIDIA GPUs / CUDA
- Rigid Docking on GPUs - PIPER
- Energy Minimization on GPUs - FTMap
- Results
## Results - Speedups

### Speedups for Rigid Docking Step

<table>
<thead>
<tr>
<th>Computation (Per Rotation)</th>
<th>Serial Runtime (ms)</th>
<th>GPU Runtime (ms)</th>
<th>Speedup vs. 1 core</th>
<th>Speedup vs. 4 cores*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rotation + Grid Assignment</td>
<td>80</td>
<td>80</td>
<td>1x</td>
<td>--</td>
</tr>
<tr>
<td>Correlations</td>
<td>3600</td>
<td>13.5</td>
<td>267x</td>
<td>70x</td>
</tr>
<tr>
<td>Accum. Of Desolvation Terms</td>
<td>180</td>
<td>1</td>
<td>180x</td>
<td>--</td>
</tr>
<tr>
<td>Scoring and Filtering</td>
<td>200</td>
<td>30</td>
<td>6.67x</td>
<td>--</td>
</tr>
<tr>
<td><strong>Total time per rotation</strong></td>
<td><strong>4060</strong></td>
<td><strong>125.5</strong></td>
<td><strong>32.6x</strong></td>
<td><strong>11x</strong></td>
</tr>
</tbody>
</table>
Results - Speedups

Speedups for Energy Minimization Step

<table>
<thead>
<tr>
<th>Computation</th>
<th>Serial Runtime (ms)</th>
<th>GPU Runtime* (ms)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Self Energy</td>
<td>6.15</td>
<td>0.23</td>
<td>26.7x</td>
</tr>
<tr>
<td>Pairwise Interaction</td>
<td>2.75</td>
<td>0.19</td>
<td>17x</td>
</tr>
<tr>
<td>van der Waals</td>
<td>0.5</td>
<td>0.19</td>
<td>17x</td>
</tr>
<tr>
<td>Force Updates</td>
<td>0.95</td>
<td>0.14</td>
<td>6.7x</td>
</tr>
<tr>
<td>Optimization move</td>
<td>0.005</td>
<td>0.005</td>
<td>1x</td>
</tr>
</tbody>
</table>

* Overall Speedup on EM computations: 18.5x
* Overall FTMap speedup (including overhead): 15x

* GPU runtimes include data transfer time

2260 atoms
9780 atom-pairs
Results – Precision Analysis

- Single vs Double Precision
  - RMSD error on force values for first iteration: $10^{-6}$
  - Convergence in 50 iterations (as opposed to 600)
  - Error on final energy and force values
    - Energy : $10^{-3}$
    - Forces : $10^{-5}$
  - Error on atom coordinates after minimization
    - 0.5 Å

- Exact match for double precision
  - Atom coordinates within $10^{-5}$ Å
  - More complex mapping on GPU – Similar speedup numbers
Conclusion

- GPUs can deliver high performance
  - Even for double precision computations
- To obtain good performance:
  - Alternate algorithms are often needed
  - Restructuring of data-structures is crucial
  - Efficient use of memory hierarchy is essential
- Getting it right on the GPU is easy …
  - … getting good performance is not so much!
Thank You!