# High Performance Computing using GPUs: Examples from Computational Biology

Bharat Sukhwani Martin 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 Both

© Mike Baldwin / Cornered

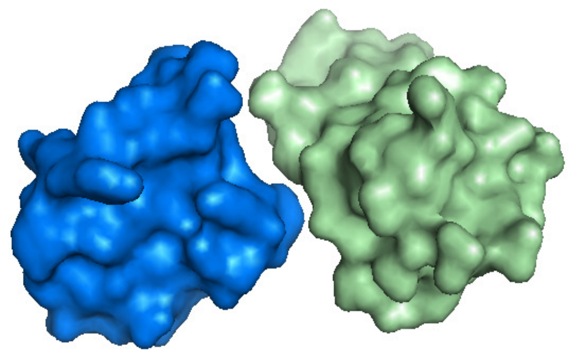
- Drug discovery is an expensive process
- Computational methods play an important role

# Molecular Docking

Molecular Docking ≡ Modeling interactions between two molecules

## Computational Task

- ❑ Finding the least energy 'pose'
  - ❑ Offset and rotation of one relative to the other

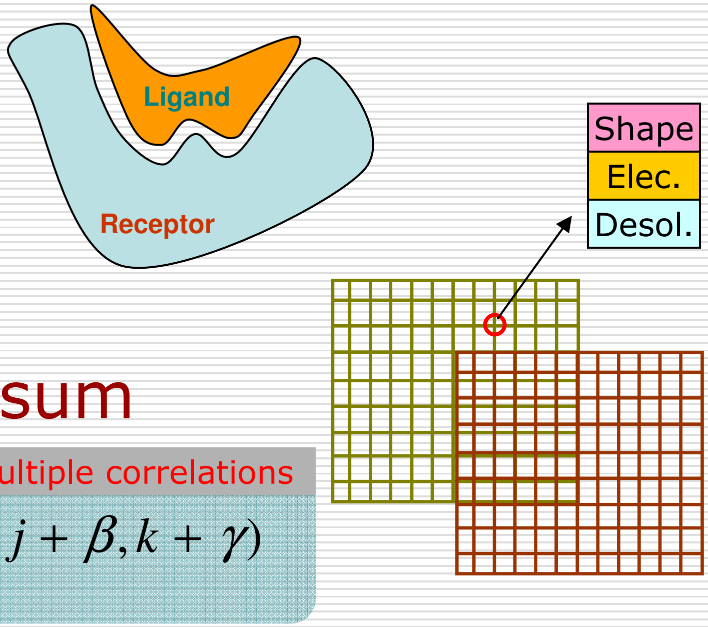


## e.g. – Exhaustive search

- ❑ Usually performed in two steps
  - ❑ Rigid Docking – Exhaustive sampling of 3D space
  - ❑ Energy minimization

# Modeling Rigid Docking

- ❑ Rigid-body approximation
  - ❑ Lock and Key model
- ❑ Grid based computing
- ❑ Exhaustive 6D search
- ❑ Pose score = 3D correlation sum



Multiple correlations

$$E(\alpha,\beta,\gamma)=\sum_{p}\sum_{i,j,k} R_p(i,j,k)\cdot L_p(i+\alpha, j+\beta, k+\gamma)$$

- ❑ FFT to speedup the correlation
  - ❑ Reduces from $O(N^6)$ to $O(N^3 \log N)$

# Computations in Rigid Docking

- ❑ Rotation
  - ❑ Increments of 5 to 15 degrees
- ❑ Grid a
- ❑ Pose s
  - ❑ FFT,
- ❑ Filterin
  - ❑ Selec

Latency Hiding

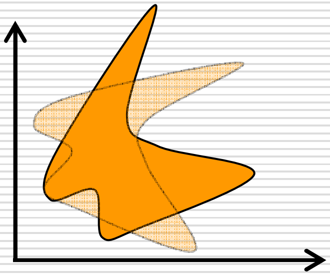
PIPER Docking program

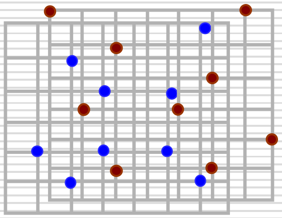
- 8 to 22 correlations

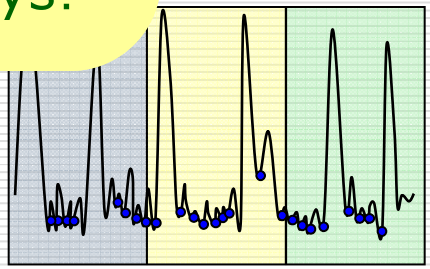
Typical serial runtime:

- Per rotation: 10 sec

- Total: Many hours to days!

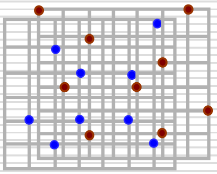




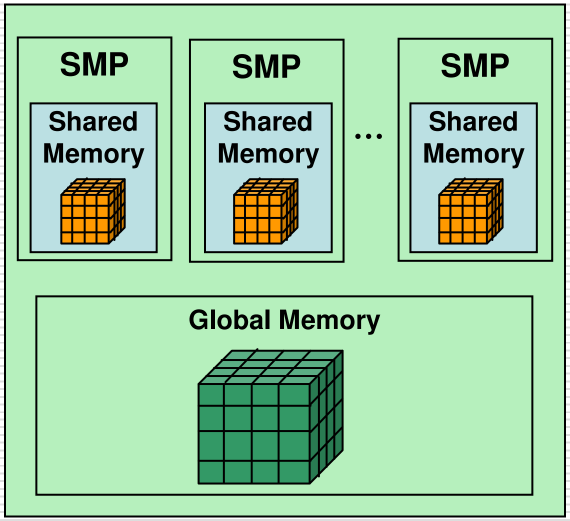


# Direct correlation on GPU

- Replaces steps of FFT, Modulation and IFFT
  - Shifting, Voxel-voxel interaction, grid summation



- Each multiprocessor accesses both grids
  - Receptor grid → Global memory
  - Ligand grid → Shared memory (duplicated)
- Multiple correlations together
  - For different energy functions



SMP

Shared Memory

SMP

Shared Memory

…

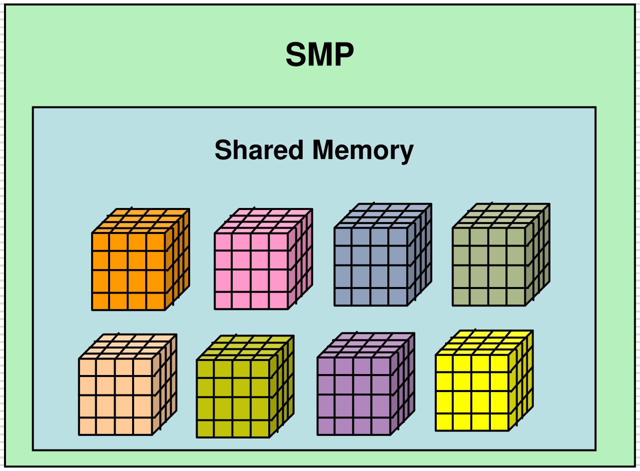
SMP

Shared Memory

Global Memory

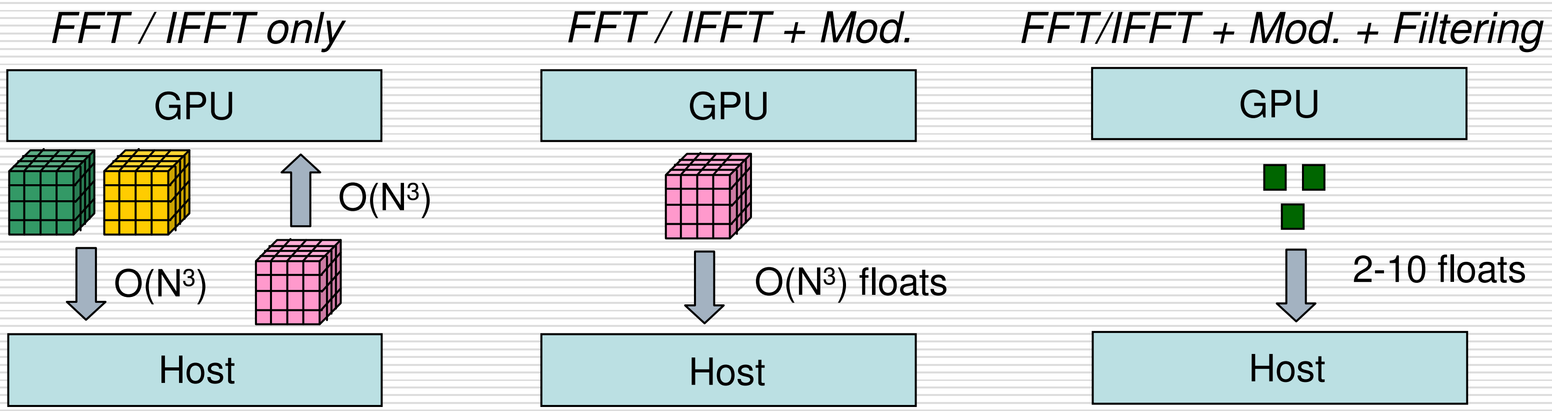
# Direct correlation on GPU

- Shared memory limits the ligand size
  - With 8 correlations - 8 cubed ligand
- For larger ligand grids
  - Store on global memory and swap pieces
  - Degrades performance
- For smaller grids - Multiple rotations
  - For 4 cubed grid - 8 rotations
  - Multiple computation per fetch
    - 2.7x improvement



# FFT Correlation on GPU

- Direct correlation is not attractive for large grids
- Multiple FFTs in serial order
  - Using NVIDIA CUFFT library
- Minimize host ↔ device data transfer
  - Perform as many steps on GPU as possible



*FFT / IFFT only*

GPU

O(N³)

O(N³)

Host

*FFT / IFFT + Mod.*

GPU

O(N³) floats

Host

*FFT/IFFT + Mod. + Filtering*

GPU

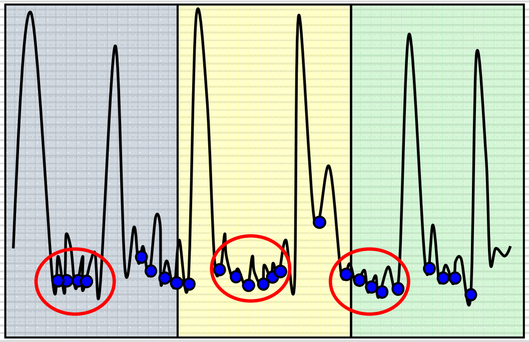
2-10 floats

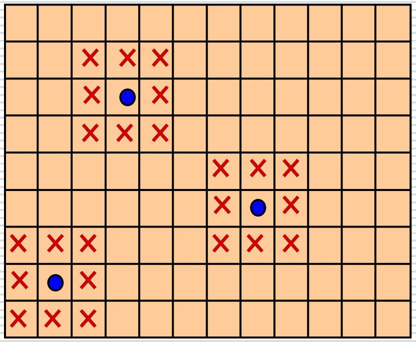
Host

# Scoring and Filtering

$$E^{total} = w1*E^{vdw} + w2*E^{elec} + w3*E^{desol}$$

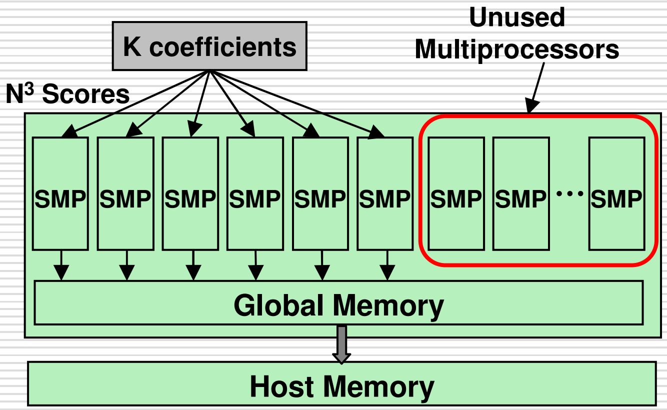
- Critical for overall performance
- Scoring
  - Multiple sets of weights
- Filtering
  - Regional Best



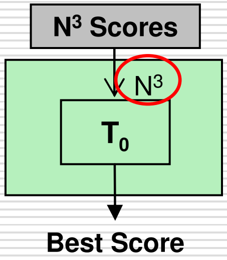


# Scoring and Filtering on GPU

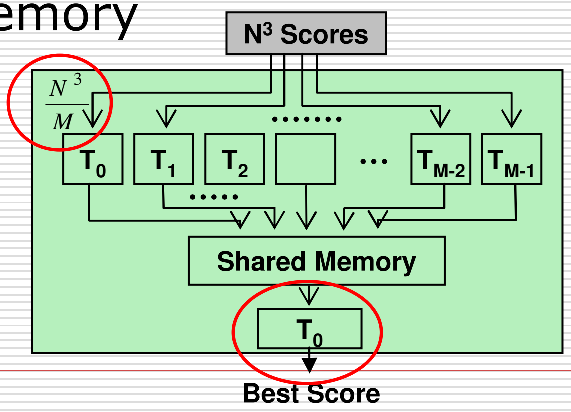
- Distribute weight-sets on multiprocessors
  - Multiprocessors underutilized



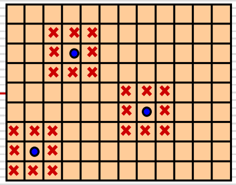
- Naïve scheme
  - Negative speedup



- Second scheme
  - Threads store scores in shared memory
  - Serialization at the end
    - Thread 0 finds best of best
    - Also performs flagging of cells



# Scoring and Filtering on GPU



- ❑ Flagging the neighboring cells
  - ❑ Serial code:
  - ❑ Does not fit in GPU shared memory

(N$^3$ entries)

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 0 | … | 1 | 0 |

- ❑ Solution 1 – Exclusion index array
  - ❑ 2-3x slowdown w.r.t. host filtering

(100 entries)

| | | | | | |
|---|---|---|---|---|---|
| 4 | 5 | 16 | 28 | …. | 45 |

- ❑ Solution 2 – Bit array on GPU global memory
  - ❑ One array for each set of weights
  - ❑ Achieves speedup over host filtering

(N$^3$ entries each)

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 0 | … | 1 | 0 |
| 0 | 0 | 0 | 1 | 1 | … | 0 | 0 |

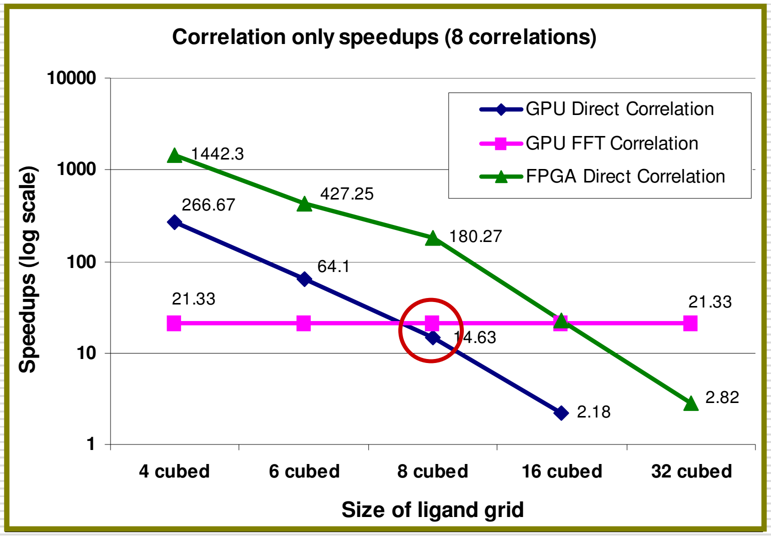
# Results

## Speedup for different phases

| | *Phase* | *CPU Time (ms)* | *GPU Time (ms)* | *Speedup* |
|---|---|---|---|---|
| Once per rotation, per energy grid | Forward FFT | 205 | 9.3 | 22 |
| | Modulation | 10 | 0.2 | 50 |
| | Inverse FFT | 205 | 11.8 | 17 |
| Once per rotation | Accumulation of desolvation terms | 240 | 4.5 | 50 |
| | Scoring and Filtering | 230 | 39.5 | 6 |
| For 22 grids | **Total runtime per rotation** | **9980** | **556** | **18** |

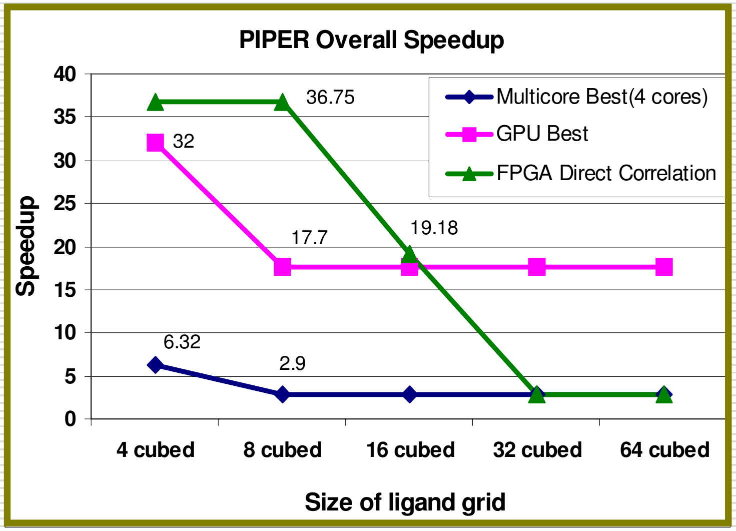
# Results

## Correlation only Speedup - FFT v/s Direct correlation



* Baseline: FFT Correlation on a single core

## Overall Speedup



* Baseline: PIPER running on single core

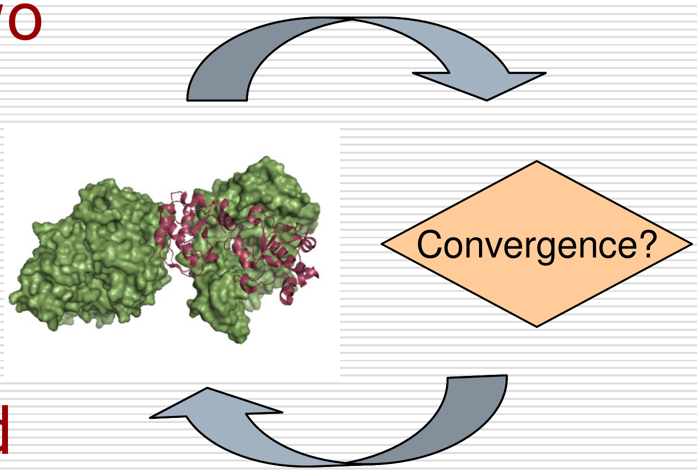
GPU: NVIDIA TESLA C1060

FPGA: Altera Stratix III

CPU: Intel Quad core Xeon @ 3GHz

# Energy Minimization

- Minimizing energy between two molecules
  - Iterative process
  - Optimization moves
- Used in Molecular Docking and Mapping Binding sites
  - To model flexible side chains



# Programs employing Energy Minimization

Docking

Mapping

CHARMM

FTMAP

EADock

CSMap

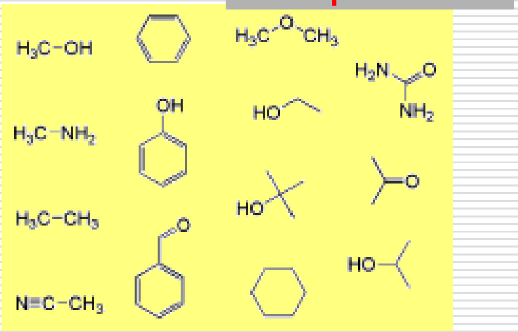
RDOCK

DARWIN

MCSS HOOK

# FTMap

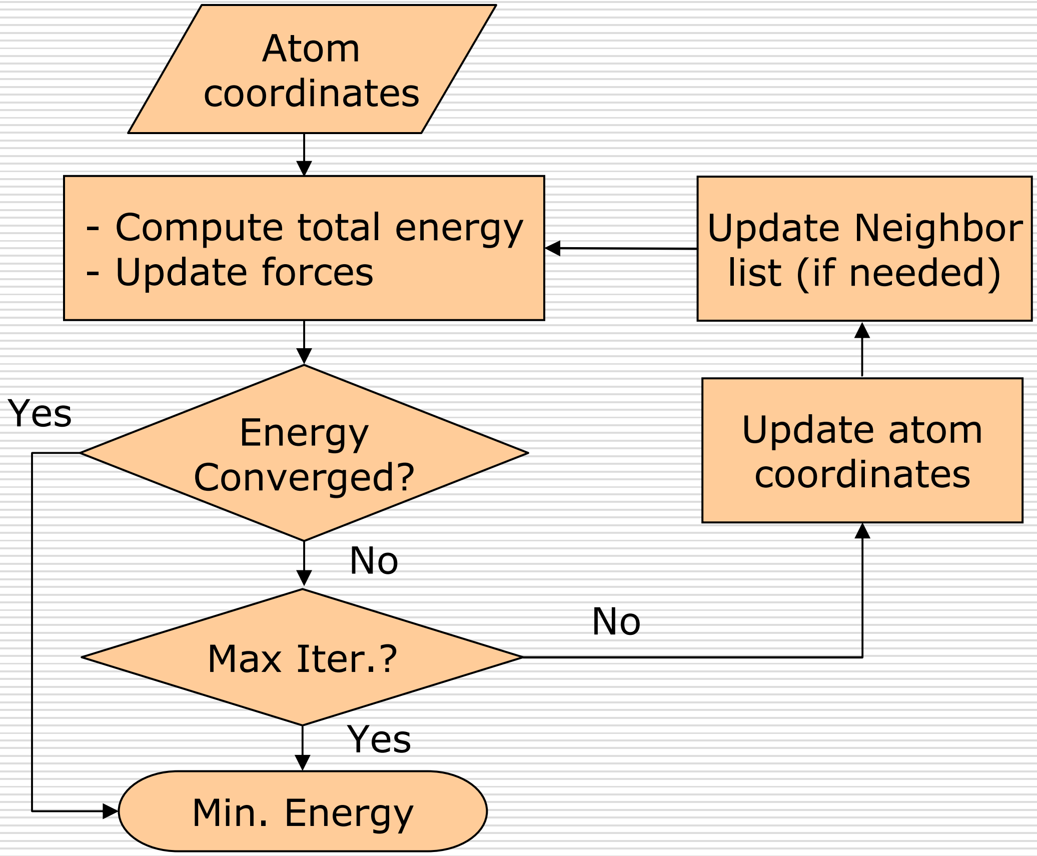
- Identification of hot-spots by docking small probes
- Rigid docking using PIPER
  - 500 rotations
  - 0.8Å grid for translation
  - 30 minutes on a single CPU



16 probes

- Minimize 2000 conformations per protein-probe complex
  - Up to 30 seconds per conformation
  - 18 hours per probe!

# Energy Minimization

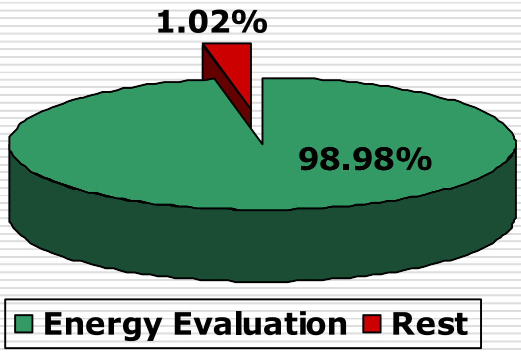


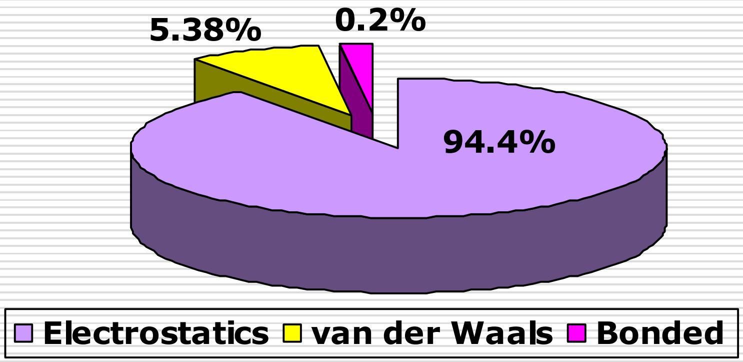
# Energy Functions

$$E^{total} = \underbrace{E^{vdw} + E^{elec}}_{\textit{non-bonded}} + \underbrace{E^{bond} + E^{angle} + E^{torsion}}_{\textit{bonded}}$$

- Looks like MD, but it’s not
  - Much smaller number of atoms
  - No motion / velocity updates
  - Similar energy terms but evaluated differently
  - Much smaller atom neighborhoods

# FTMap Profiling





FTMap Minimization Step

Energy evaluation phase

Absolute time ~ 10 ms per iteration (on single core)

$$E^{total} = \underbrace{E^{vdw} + E^{elec}}_{non\text{-}bonded} + \underbrace{E^{bond} + E^{angle} + E^{torsion}}_{bonded}$$

# FTMap Electrostatics Model

- Analytic Continuum Electrostatics (ACE)

## Atom Self Energy

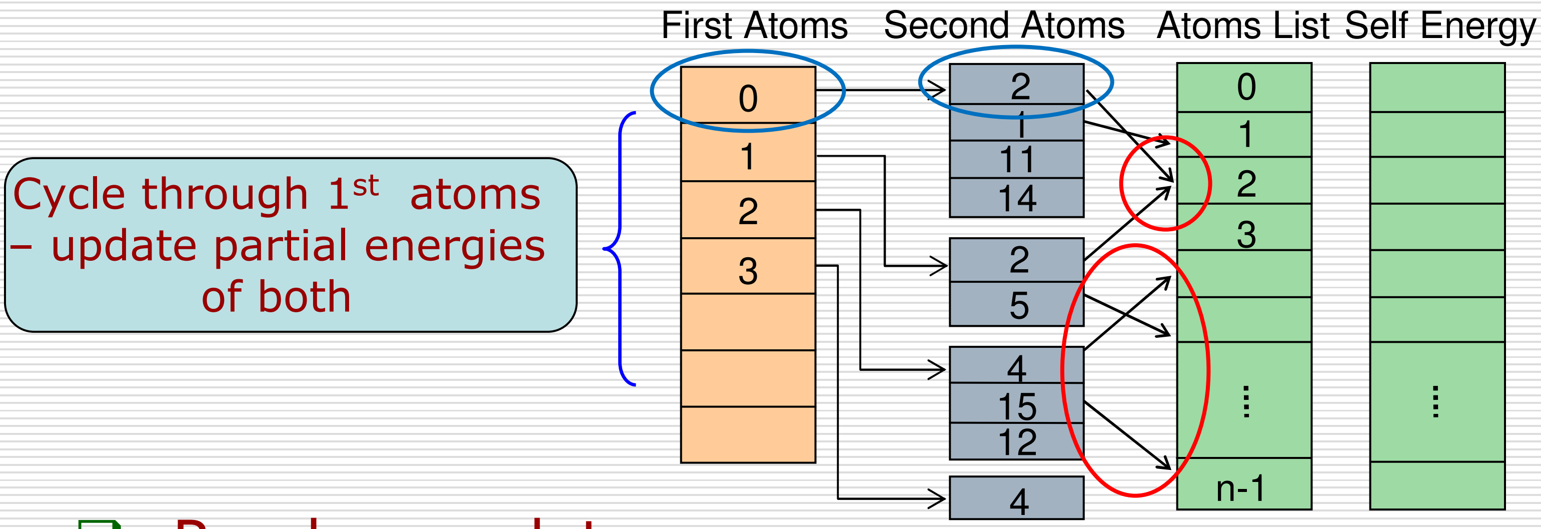
$$E_i^{self} = \frac{q_i^2}{2\varepsilon_s R_i} + \sum_{k \neq i} E_{ik}^{self} \qquad \Big| \qquad E_{ik}^{self} = \frac{\tau q_i^2}{\omega_{ik}} e^{-\left(\frac{r_{ik}^2}{\sigma_{ik}^2}\right)} + \frac{\tau q_i^2 \tilde{V}_k}{8\pi}\left(\frac{r_{ik}^3}{r_{ik}^4 + \mu_{ik}^4}\right)^4$$

## Pairwise interaction – Generalized Born eqn.

$$E_{ij}^{\text{int}} = 332 \sum_{j \neq i} \frac{q_i q_j}{r_{ij}} - 166\ \tau \sum_{j \neq i} \frac{q_i q_j}{\sqrt{r_{ij}^2 + \alpha_i \alpha_j e^{-\left(r_{ij}^2 / 4\alpha_i \alpha_j\right)}}}$$

Born Radii – depends on $E^{self}$

# Original Data Structure - Neighbor Lists



Cycle through 1st atoms – update partial energies of both

- Random updates
  - Cannot distribute the array – must stay on global memory
- Write conflicts
  - Second atom might appear in multiple neighbor lists

# Mapping to CUDA – Difficulties

- Little to no data reuse
- Small computation per iteration
- Multiple accumulations – self energy of each atom must be computed

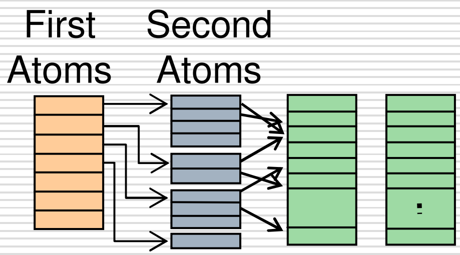
Inherent to the algorithm

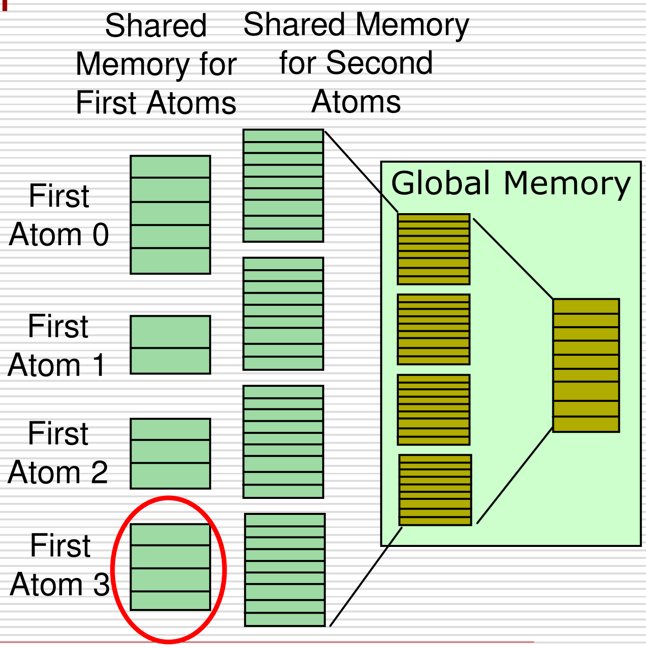
- Total runtime dominated by data transfers
- Accumulation requires serialization
- Random updates

Architecture related

# Mapping to GPU – Neighbor Lists

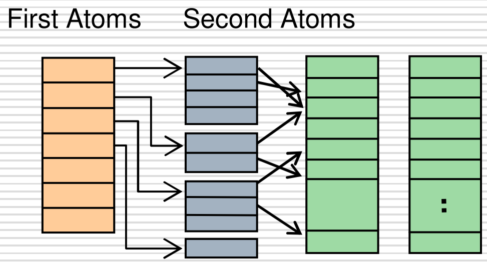
- Separate energy arrays for first and second atoms
  - Allows parallel updates by multiple threads
- Multiple copies of arrays for second atom
  - One in each thread block
  - Parallel updates – no conflicts
- First arrays reduced to single values
- Second atoms arrays merged by moving to global memory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





# Modified Data Structure - Pair List

- 2D neighbor lists → 1D pair list
  - Each pair stores energies of the two atoms involved
- Distribute pairs to multiple threads
  - More uniform work distribution
- Compute partial energies in parallel
- Perform accumulations serially



| | Atom index | | Self energy | |
|---|---|---|---|---|
| Pair id | Atom 1 | Atom 2 | Atom 1 | Atom 2 |
| 0 | 0 | 2 | | |
| 1 | 0 | 1 | | |
| 2 | 0 | 11 | | |
| 3 | 0 | 14 | | |
| 4 | 1 | 2 | | |
| 5 | 1 | 5 | | |
| 6 | 2 | 4 | | |
| 7 | 2 | 15 | | |
| 8 | 2 | 12 | | |
| 9 | 3 | 4 | | |

# Mapping Pair List – Initial Attempts

- ❑ Pairs distributed on different threads
  - ❑ Parallel evaluations, serial accumulation
- ❑ Accumulation on GPU
  - ❑ From global memory
  - ❑ Slow

| | Atom index | | Self energy | |
|---|---|---|---|---|
| Pair id | Atom 1 | Atom 2 | Atom 1 | Atom 2 |
| 0 | 0 | 2 | | |
| 1 | 0 | 1 | | |
| 2 | 0 | 11 | | |
| 3 | 0 | 14 | | |
| 4 | 1 | 2 | | |
| 5 | 1 | 5 | | |
| 6 | 2 | 4 | | |
| 7 | 2 | 15 | | |
| 8 | 2 | 12 | | |
| 9 | 3 | 4 | | |

- ❑ Accumulation on host
  - ❑ Fast, but requires energy arrays to be transferred every iteration
  - ❑ 2x-3x speedup

# Mapping Pair List – Improved Scheme

- Pair list with two changes
  - Split forward and reverse pair list
  - Static mapping of pairs onto GPU threads

# Split Pair List

- ❑ Problem due to random occurrences of second atoms
- ❑ Split into forward and reverse lists
  - ❑ Forward list: Same as before
  - ❑ Reverse list: Treat every second atom as a first atom
  - ❑ Process only first atoms of each list
  - ❑ Adds determinism → Better distribution

Forward List

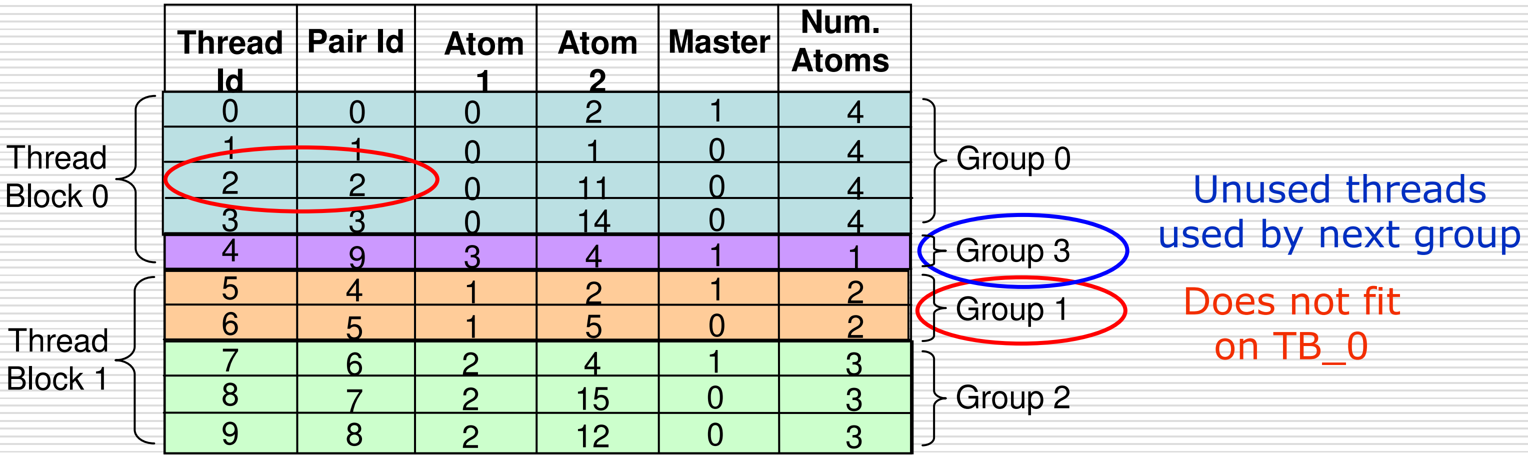
| | Atom index | | Self energy | |
|---|---|---|---|---|
| Pair id | Atom 1 | Atom 2 | Atom 1 | Atom 2 |
| 0 | 0 | 2 | | |
| 1 | 0 | 1 | | |
| 2 | 0 | 11 | | |
| 3 | 0 | 14 | | |
| 4 | 1 | 2 | | |
| 5 | 1 | 5 | | |
| 6 | 2 | 4 | | |
| 7 | 2 | 15 | | |
| 8 | 2 | 12 | | |
| 9 | 3 | 4 | | |

Reverse List

| | Atom index | | Self energy | |
|---|---|---|---|---|
| Pair id | Atom 1 | Atom 2 | Atom 1 | Atom 2 |
| 0 | 1 | 0 | | |
| 1 | 2 | 0 | | |
| 2 | 2 | 1 | | |
| 3 | 4 | 2 | | |
| 4 | 4 | 3 | | |
| 5 | 5 | 1 | | |
| 6 | 11 | 0 | | |
| 7 | 12 | 2 | | |
| 8 | 14 | 0 | | |
| 9 | 15 | 2 | | |

# Static Mapping - Assignment Table

- Pairs can be grouped by first atom
- Groups mapped to different thread blocks
  - Look for next block with enough threads



| Thread Id | Pair Id | Atom 1 | Atom 2 | Master | Num. Atoms |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 2 | 1 | 4 |
| 1 | 1 | 0 | 1 | 0 | 4 |
| 2 | 2 | 0 | 11 | 0 | 4 |
| 3 | 3 | 0 | 14 | 0 | 4 |
| 4 | 9 | 3 | 4 | 1 | 1 |
| 5 | 4 | 1 | 2 | 1 | 2 |
| 6 | 5 | 1 | 5 | 0 | 2 |
| 7 | 6 | 2 | 4 | 1 | 3 |
| 8 | 7 | 2 | 15 | 0 | 3 |
| 9 | 8 | 2 | 12 | 0 | 3 |

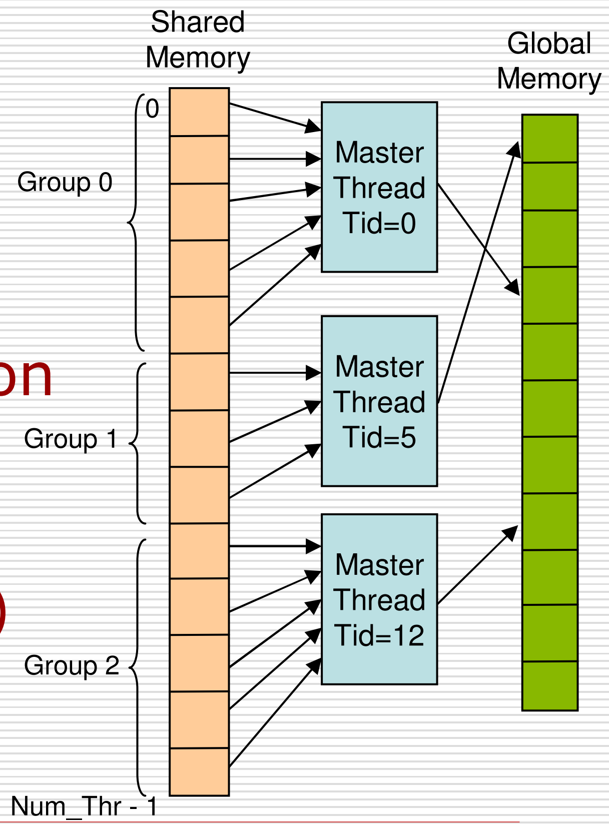
- One pair per thread (multiple if $N_{pair} > N_{threads}$)
- Reverse Assignment table for second atoms

# Computing and Accumulating Energies

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

| Thread Id | Pair Id | Atom 1 | Atom 2 | Master | Num. Atoms |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 2 | 1 | 4 |
| 1 | 1 | 0 | 1 | 0 | 4 |
| 2 | 2 | 0 | 11 | 0 | 4 |
| 3 | 3 | 0 | 14 | 0 | 4 |
| 4 | 9 | 3 | 4 | 1 | 1 |

- Master thread performs accumulation
  - 'N' locations starting from its thread id
- Multiple parallel accumulations per thread block (from shared memory)



# Results

- NVIDIA TESLA C1060
- Three GPU Kernels
  - Self energy and gradient computation
  - Pairwise interaction and gradient computation
  - Force updates

| ***Computation*** | ***Serial Time (per iteration)*** | ***GPU Time*** | ***Speedup*** |
|---|---|---|---|
| Self energies | 6.15 ms | 0.22 ms | **27.9x** |
| Pairwise | 2.75 ms | 0.23 ms | **11.9x** |
| Force updates | 0.95 ms | 0.14 ms | **6.7x** |

# Results – Overall Speedup

- 5 different protein-probe complexes
  - ~2200 atoms per complex
  - ~9800 atom-atom pairs
  - 1000 iterations per complex

| *Complex* | *Serial Time* | *GPU Time* | *Speedup* |
|---|---|---|---|
| Complex 1 | 11.9 sec | 1.098 sec | **10.8x** |
| Complex 2 | 11.87 sec | 1.078 sec | **11x** |
| Complex 3 | 11.8 sec | 1.078 sec | **10.9x** |
| Complex 4 | 10.74 sec | 0.906 sec | **11.8x** |
| Complex 5 | 11.87 sec | 1.094 sec | **10.8x** |

# Conclusion

- Docking and Mapping are computationally demanding
- GPUs provide high FP capability, but …
- … must
  - minimize host-board transfer!
  - map computations to threads efficiently!
  - perform large computations per datum transferred!

Thank You!