Accelerating Free-Energy Minimization using Graphics Processors

Bharat Sukhwani **Martin Herbordt**

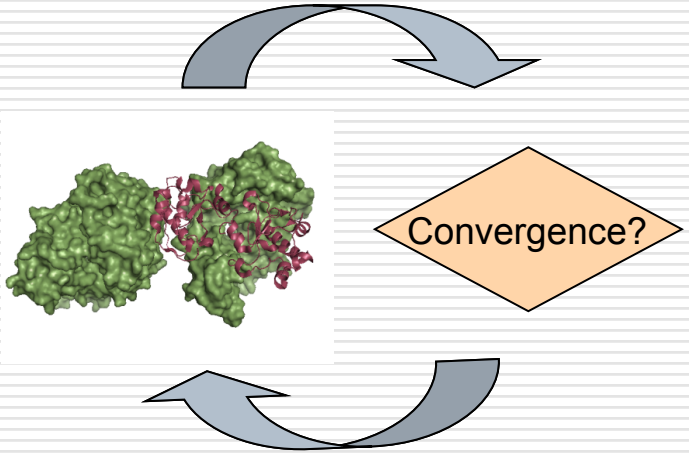
Computer Architecture and Automated Design Laboratory
Department of Electrical and Computer Engineering
Boston University
http://www.bu.edu/caadlab

* This work supported, in part, by the U.S. NIH/NCRR
+ Thanks to Sandor Vajda, Dima Kozakov and Ryan Brenke (BME at Boston University)

- Drug discovery is an expensive process
- Computational methods play an important role in faster and cost-effective drug discovery

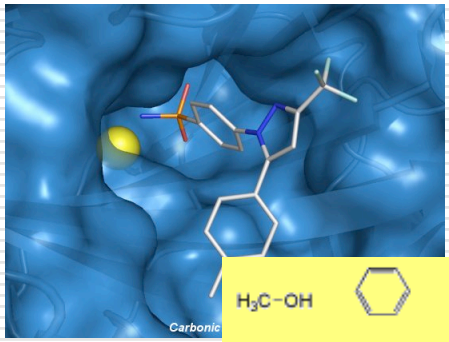
Energy Minimization

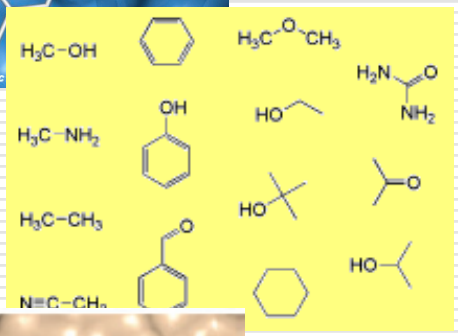
- Minimizing energy between two molecules
  - Iterative process
  - Optimization moves
- Used in Molecular Docking and Mapping
  - To model flexible side chains

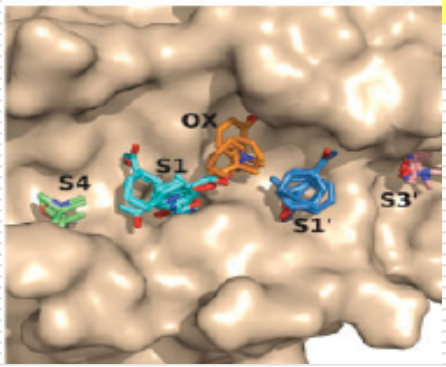


Computational Mapping

- Purpose: Identification of hot spots
  - Sites that are likely to bind inhibitors with high affinity
- Process:
  - Docking small molecule probes
  - Energy minimizing each protein-probe complex
- Rationale:
  - Hot spots are major contributors to the binding energy
  - They bind a large variety of small molecules
  - Regions binding many probes indicate hot spots

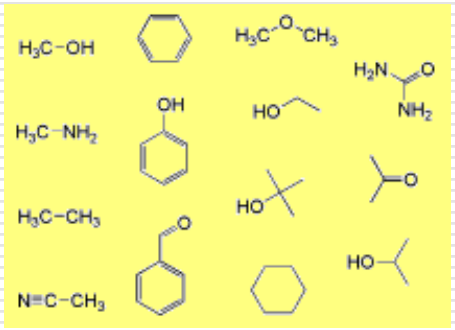




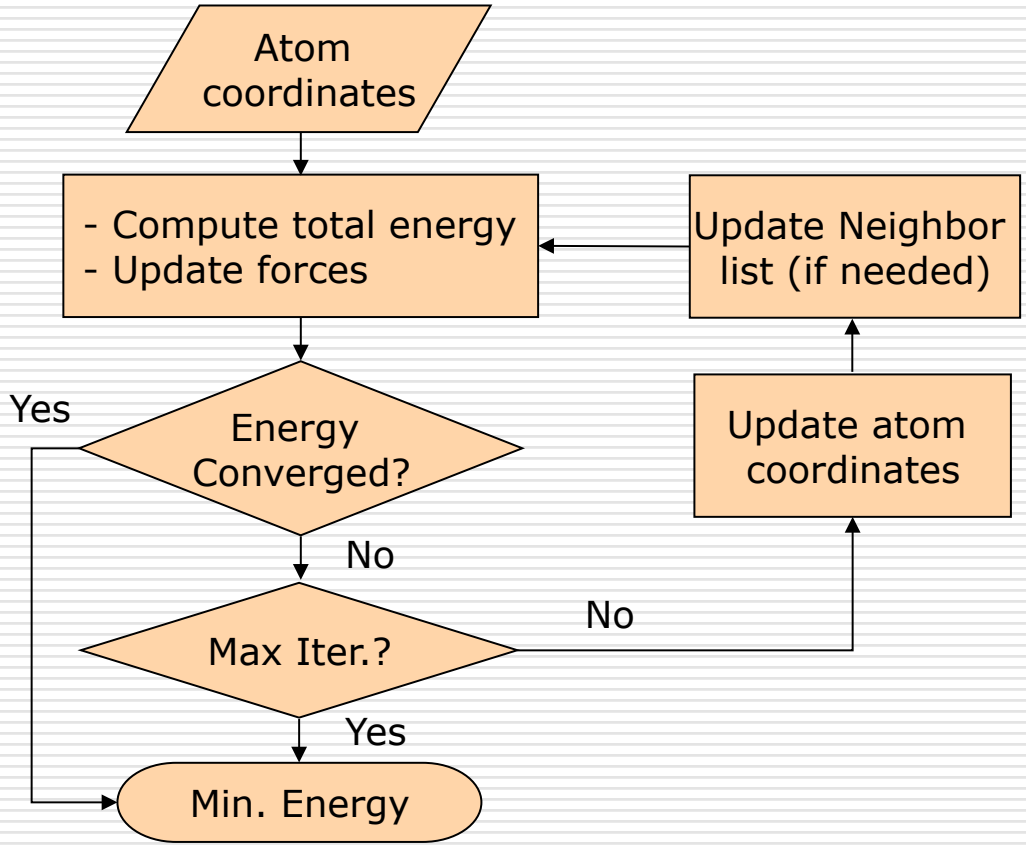


FTMap

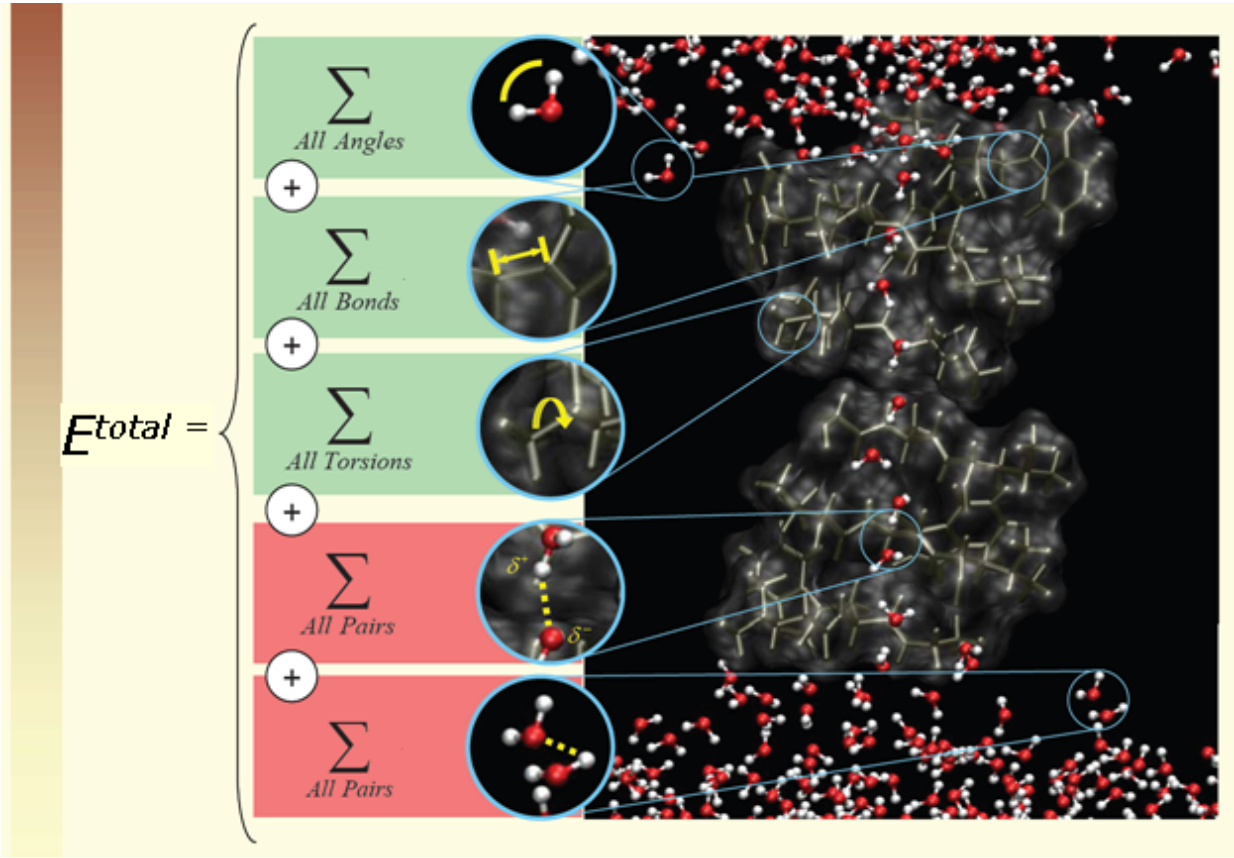
- 16 small molecule probes
- Rigid docking using PIPER
  - 500 rotations
  - 0.8Å grid for translation
  - 30 minutes on a single CPU
- Minimize 2000 conformations per protein-probe complex
  - Up to 30 seconds per conformation
  - 18 hours per probe!



Energy Minimization



Energy Functions

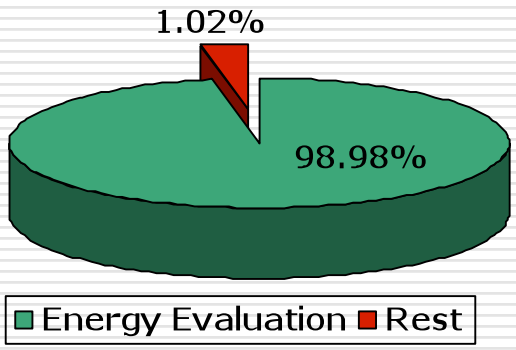


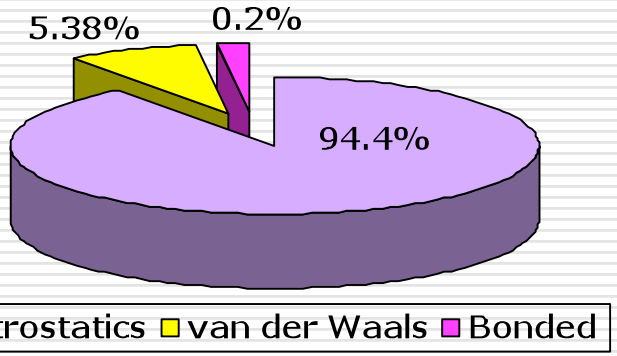
* Image courtesy of Dr. Paul Chow, University of Toronto

Looks like MD, but it’s not

- Much smaller number of atoms
  - Typically few thousand
- Move to next position
  - No motion / velocity updates
- Similar energy terms but evaluated differently
- Much smaller atom neighborhoods
  - Very small cut-off radius

FTMap Profiling





FTMap Minimization Step

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

Energy evaluation phase

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

FTMap Electrostatics Model

$$E^{total} = E^{vdw} + E^{elec} + E^{bond} + E^{angle} + E^{torsion}$$

(non-bonded: $E^{vdw} + E^{elec}$; bonded: $E^{bond} + E^{angle} + E^{torsion}$)

- Analytic Continuum Electrostatics (ACE)

Atom Self Energy

Solvation volume

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

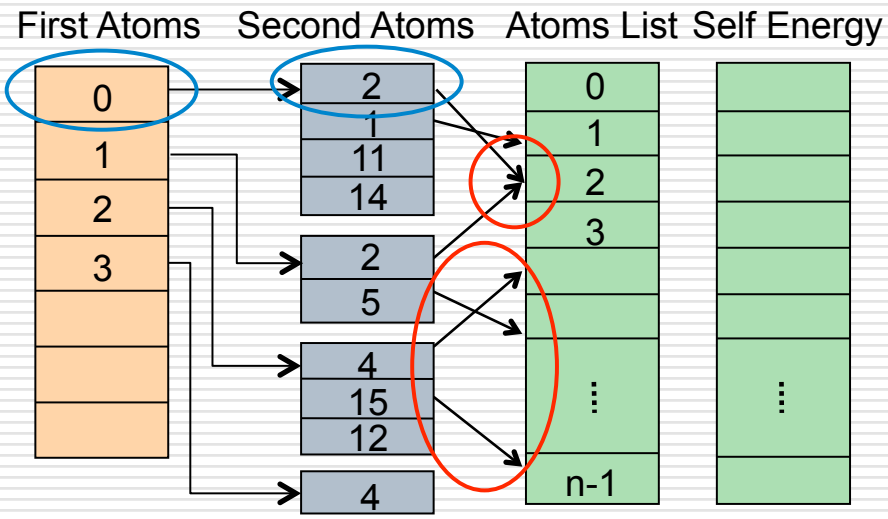
Pairwise interaction – Generalized Born eqn.

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

Born Radii – depends on E^{self}

Original Data Structure - Neighbor Lists

- List of first atoms
  - Each has a list of second atoms
- Cycles through first atoms
  - Accumulates contribution due to second atoms in neighbor list
  - Uses symmetry – computes energy of second atom too
- Random updates
  - Cannot distribute the array – must stay on global memory
- Write conflicts
  - Second atom might appear in multiple neighbor lists

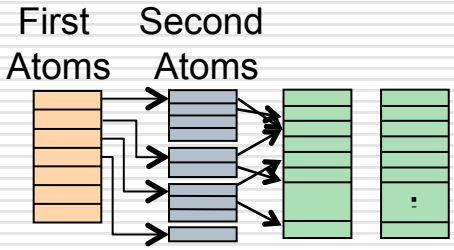


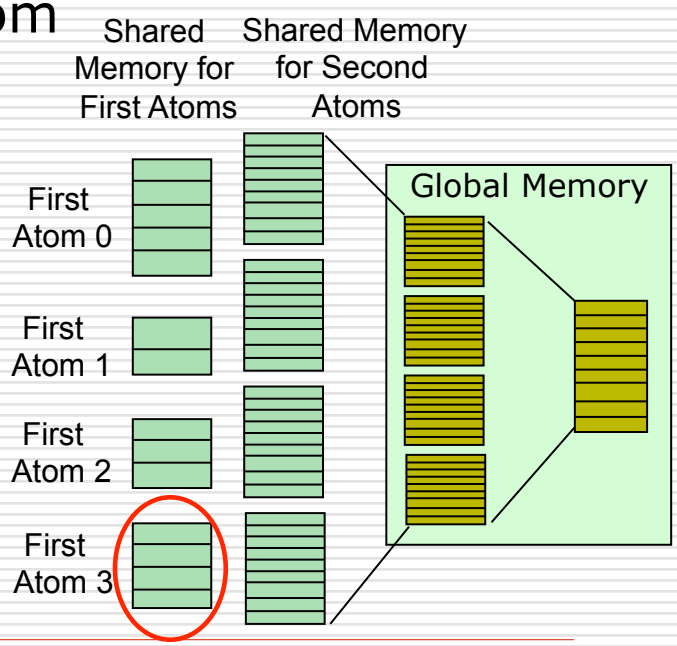
Mapping to CUDA – Difficulties

- Little to no data reuse
- Small computation per iteration
- Total runtime dominated by data transfers
- Accumulation requires serialization
- Random updates
- Multiple accumulations – self energy of each atom must be computed

Mapping to GPU – Neighbor Lists

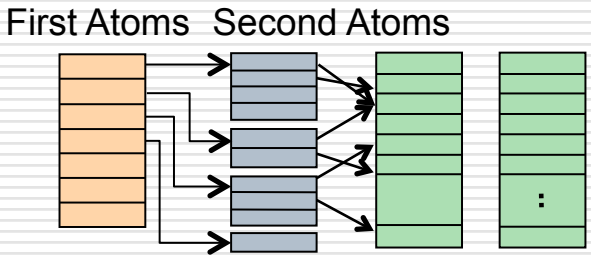
- Separate energy arrays for first and second atoms
  - Allows parallel updates by multiple threads
- Multiple copies of arrays for second atom
  - One in each thread block
  - Parallel updates – no conflicts
- First arrays reduced to single values
- Second atoms arrays merged by moving to global memory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





Modified Data Structure - Pair List

- Flatten 2D neighbor lists into 1D pair list
  - Similar to techniques used in MD codes like NAMD
  - Each pair stores energies of the two atoms involved
- Distribute pairs across 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
  - 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

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

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

Unused threads used by next group

Does not fit on TB_0

Creating Assignment Table

- Statically map pairs to threads
  - Look for next available block with enough threads
  - Loop back to use unused threads
  - Threads reused if not enough threads
- Preprocessing
  - Created only once – unless neighbor list is updated
  - Transferred to GPU only once

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	1	1	-1	-1	-1

rep. 0

rep. 1

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	8	3	4	1	1
0	4	1	2	1	2
1	5	1	5	0	2
2	6	2	4	1	2
3	7	2	15	0	2
4	-1	-1	-1	-1	-1

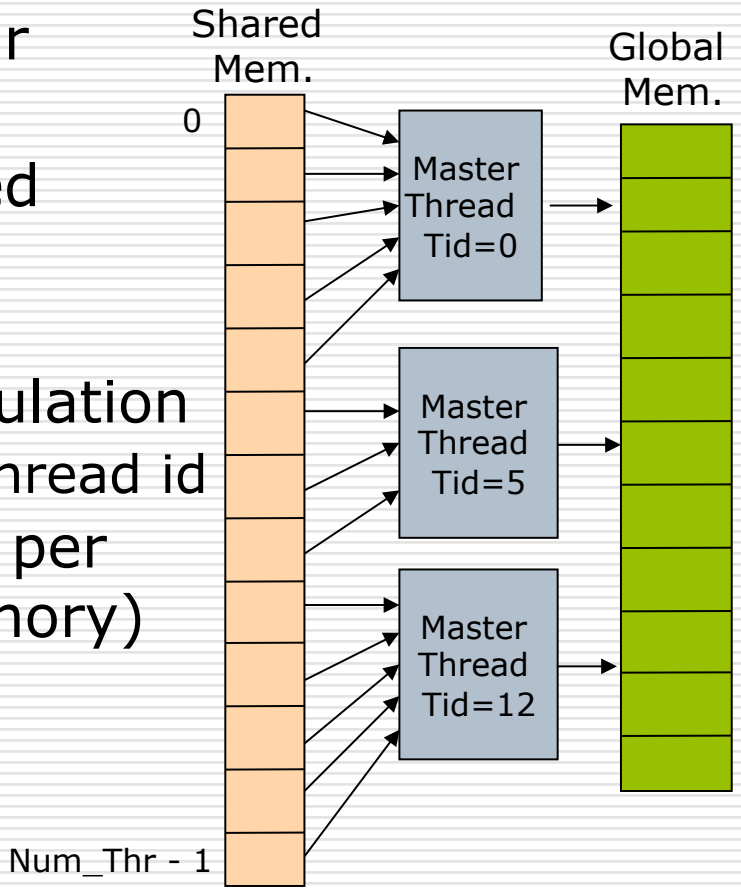
Cannot fit in first rep.

Unused during second rep.

Computing and Accumulating Energies

- Threads compute energy of pair assigned to them
- Partial energies stored 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)

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



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**

Next Steps

- van der Waals computation on GPU
- Optimization move on GPU
  - Quasi-Newtonian (L-BFGS)
  - Will avoid the need to transfer atom coordinates in every iteration

Thank You