

# CUSA and CUDE: GPU-Accelerated Methods for Estimating Solvent Accessible Surface Area and Desolvation

David Dynerman<sup>1</sup>   Erick Butzlaff<sup>2</sup>   Julie C. Mitchell<sup>1,3</sup>

<sup>1</sup>Department of Mathematics

<sup>2</sup>AMEP Program & Department of Economics

<sup>3</sup>Department of Biochemistry

University of Wisconsin

IEEE/ACM Supercomputing 2008

# Outline

- 1 Introduction
- 2 Energy Models
- 3 Technical Implementation

# The Big Picture

- Using two simple models, we derive an algorithm for the *desolvation energy* of a protein complex. We find that this algorithm is well-suited in minimization settings, with our GPU accelerated code outperforming our CPU implementation by up to *200x*.
- Goal: Understand how proteins interact with each other
- Free energy,  $\Delta G$ , is a useful measure of equilibrium
- Nature will seek out energy minimums...
- Our focus:  $\Delta G_d$ , desolvation energy

# The Big Picture

- Using two simple models, we derive an algorithm for the *desolvation energy* of a protein complex. We find that this algorithm is well-suited in minimization settings, with our GPU accelerated code outperforming our CPU implementation by up to *200x*.
- Goal: Understand how proteins interact with each other
  - Free energy,  $\Delta G$ , is a useful measure of equilibrium
  - Nature will seek out energy minimums...
  - Our focus:  $\Delta G_d$ , desolvation energy

# The Big Picture

- Using two simple models, we derive an algorithm for the *desolvation energy* of a protein complex. We find that this algorithm is well-suited in minimization settings, with our GPU accelerated code outperforming our CPU implementation by up to *200x*.
- Goal: Understand how proteins interact with each other
- Free energy,  $\Delta G$ , is a useful measure of equilibrium
  - Nature will seek out energy minimums...
  - Our focus:  $\Delta G_d$ , desolvation energy

# The Big Picture

- Using two simple models, we derive an algorithm for the *desolvation energy* of a protein complex. We find that this algorithm is well-suited in minimization settings, with our GPU accelerated code outperforming our CPU implementation by up to *200x*.
- Goal: Understand how proteins interact with each other
- Free energy,  $\Delta G$ , is a useful measure of equilibrium
- Nature will seek out energy minimums...
- Our focus:  $\Delta G_d$ , desolvation energy

# The Big Picture

- Using two simple models, we derive an algorithm for the *desolvation energy* of a protein complex. We find that this algorithm is well-suited in minimization settings, with our GPU accelerated code outperforming our CPU implementation by up to *200x*.
- Goal: Understand how proteins interact with each other
- Free energy,  $\Delta G$ , is a useful measure of equilibrium
- Nature will seek out energy minimums...
- Our focus:  $\Delta G_d$ , desolvation energy

# Our Work

- We want  $\Delta G_d$  for a protein complex
  - Intuition:  $\Delta G_d$  is the energy cost of pushing solvent molecules out of a binding location
- We want  $\Delta G_d$  *fast* so we can quickly test many configurations in search of a minimum  $\Delta G_d$
- We use a desolvation model developed by Eisenberg et. al.

# Our Work

- We want  $\Delta G_d$  for a protein complex
  - Intuition:  $\Delta G_d$  is the energy cost of pushing solvent molecules out of a binding location
- We want  $\Delta G_d$  *fast* so we can quickly test many configurations in search of a minimum  $\Delta G_d$
- We use a desolvation model developed by Eisenberg et. al.

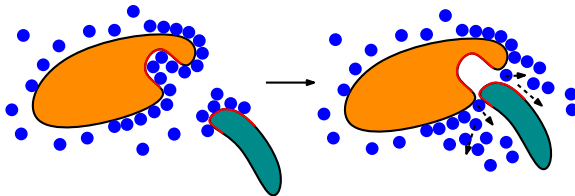
# Our Work

- We want  $\Delta G_d$  for a protein complex
  - Intuition:  $\Delta G_d$  is the energy cost of pushing solvent molecules out of a binding location
- We want  $\Delta G_d$  *fast* so we can quickly test many configurations in search of a minimum  $\Delta G_d$
- We use a desolvation model developed by Eisenberg et. al.

# Our Work

- We want  $\Delta G_d$  for a protein complex
  - Intuition:  $\Delta G_d$  is the energy cost of pushing solvent molecules out of a binding location
- We want  $\Delta G_d$  *fast* so we can quickly test many configurations in search of a minimum  $\Delta G_d$
- We use a desolvation model developed by Eisenberg et. al.

# Desolvation Energy Model

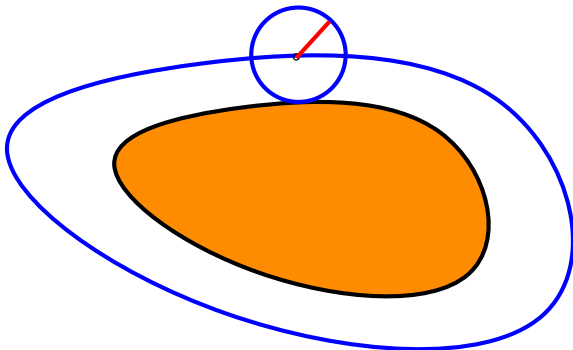


- Eisenberg: There is a linear relationship between  $\Delta G_d$  and buried molecular surface area

- $$\Delta G_d = \sum_{i \in \text{atoms}} \gamma_i \Delta A_i$$

# Solvent Accessible Surface Area

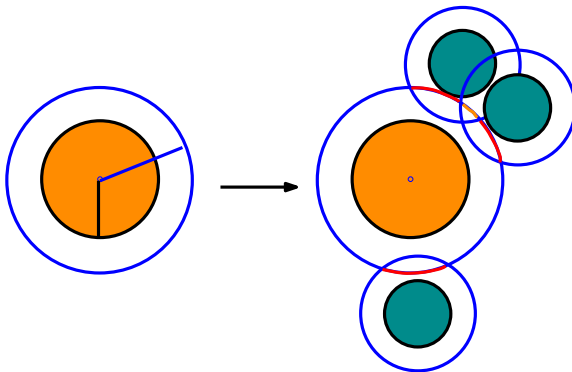
- Solvent Accessible Surface Area (SASA) (Lee-Richards, 1971) (Shrake & Rupley 1973)
- Intuition: Roll a solvent-sized sphere over the protein



# SASA Model

- SASA model due to Hasel et. al.
- Overview:
  - Start with the surface area of an atom
  - Subtract pairwise overlaps
  - Use a knowledge based factor for multiple overlaps

# SASA Model



$$A_i = S_i \prod_{j \in \text{atoms}} f_{ij}$$

$$f_{ij} = 1 - \frac{p_i b_{ij}}{S_i}, \quad S_i = 4\pi(r_i + r_s)^2$$

# CPU Implementation

- SASA Model: General  $O(n^2)$  pairwise loop
- Desolvation: Sum over all atoms
- Bottleneck is SASA
  - ...especially since SASA is a function of atomic positions

# GPU Implementation

- CUDA device kernel for SASA
  - $\Delta G_d$  sum on CPU
- Our first implementation used global memory
- Real gains: use shared memory

# SASA Kernel

- Each thread computes SASA ( $A_i$ ) for one atom

- $$A_i = S_i \prod_{j \in \text{atoms}} f_{ij}$$

$A_i \Leftarrow S_i$

$\text{my\_atom} \Leftarrow \text{d\_atoms}[i]$

**for** each atom  $j$  **do**

$$A_i = A_i \times f_{ij}(\text{my\_atom}, \text{d\_atoms}[j])$$

**end for**

# SASA Kernel

- Frequent global memory access is slow
- Cooperatively use shared memory (Schive et. al.)

$A_i \Leftarrow S_i$

my\_atom  $\Leftarrow$  d\_atoms[i]

**for** each block of protein atoms *b* **do**

    \_\_syncthreads();

    s\_atoms[ offset ] = d\_atoms[ b + offset ]

    \_\_syncthreads();

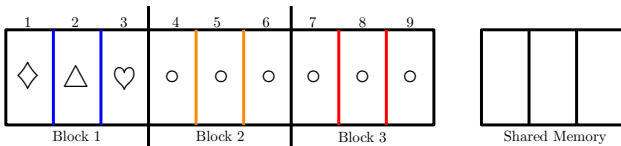
**for** each atom *j* in *b* **do**

$A_i \Leftarrow A_i \times f_{ij}(\text{my\_atom}, \text{s\_atoms}[ j \% |b| ])$

**end for**

**end for**

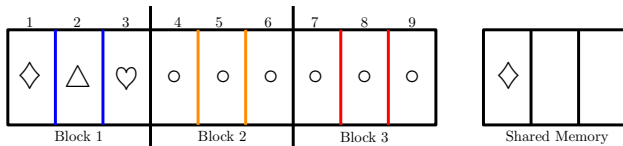
# Block-based iteration



## ● Process Block 1

- Thread 1 loads  $\diamond$  into shared memory
  - Thread 2 loads  $\triangle$  into shared memory
  - Thread 3 loads  $\heartsuit$  into shared memory
  - `__syncthreads();`
  - Now threads 1, 2, & 3 have everything required to compute pairwise interactions in block 1
- ## ● Process Block 2
- ...

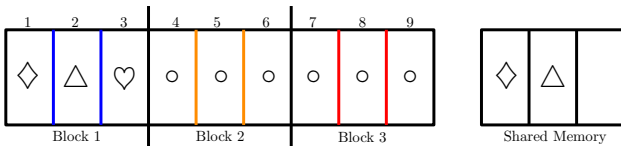
# Block-based iteration



## ● Process Block 1

- Thread 1 loads  $\diamond$  into shared memory
  - Thread 2 loads  $\triangle$  into shared memory
  - Thread 3 loads  $\heartsuit$  into shared memory
  - `__syncthreads();`
  - Now threads 1, 2, & 3 have everything required to compute pairwise interactions in block 1
- ## ● Process Block 2
- ...

# Block-based iteration



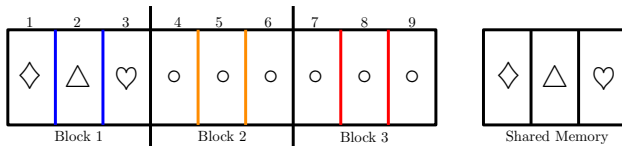
- Process Block 1

- Thread 1 loads  $\diamond$  into shared memory
- Thread 2 loads  $\triangle$  into shared memory
- Thread 3 loads  $\heartsuit$  into shared memory
- `__syncthreads();`
- Now threads 1, 2, & 3 have everything required to compute pairwise interactions in block 1

- Process Block 2

- ...

# Block-based iteration



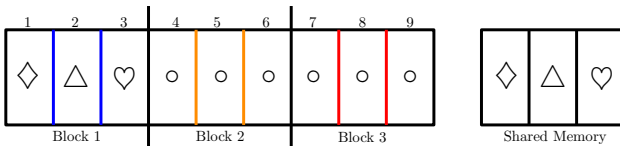
- Process Block 1

- Thread 1 loads  $\diamond$  into shared memory
- Thread 2 loads  $\triangle$  into shared memory
- Thread 3 loads  $\heartsuit$  into shared memory
- `__syncthreads();`
- Now threads 1, 2, & 3 have everything required to compute pairwise interactions in block 1

- Process Block 2

- ...

# Block-based iteration



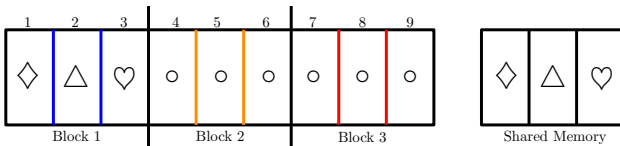
## ● Process Block 1

- Thread 1 loads  $\diamond$  into shared memory
- Thread 2 loads  $\triangle$  into shared memory
- Thread 3 loads  $\heartsuit$  into shared memory
- `__syncthreads();`
- Now threads 1, 2, & 3 have everything required to compute pairwise interactions in block 1

## ● Process Block 2

● ...

# Block-based iteration



- Process Block 1

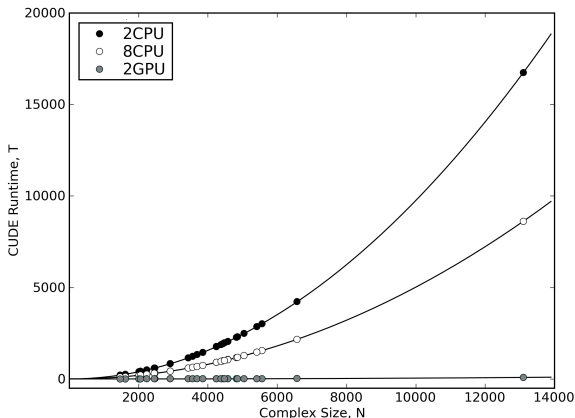
- Thread 1 loads  $\diamond$  into shared memory
- Thread 2 loads  $\triangle$  into shared memory
- Thread 3 loads  $\heartsuit$  into shared memory
- `__syncthreads();`
- Now threads 1, 2, & 3 have everything required to compute pairwise interactions in block 1

- Process Block 2

- ...

# Results

- We simulate a minimization setting by generating a large amount of protein conformations



# Future Work

- We plan to add VDW and electrostatics
  - Pairwise will be easy, others harder...
- Flexible docking using replicas

# Acknowledgements

- We gratefully acknowledge our funding sources:
  - DoE Genomics:GTL/SciDAC DE-FG02-04ER25627
  - NIH National Library of Medicine 5T15LM007359

# Bibliography

- Totrov, M. & Abagyan, R. (1997) Flexible protein-ligand docking by global energy optimization in internal coordinates. *Proteins* S1:215-220.
- Eisenberg, D. & McLachlan, A. D. (1986) Solvation energy in protein folding and binding. *Nature* 319:199-203.
- Hasel, W., Hendrickson, T. F., & Still, W. C. (1988) A rapid approximation to the solvent accessible surface areas of atoms. *Tetrahedron Computer Methodology* 1:103-116.
- NVIDIA Corporation (2007) NVIDIA CUDA Compute Unified Device Architecture Programming Guide Version 1.0.
- Schive, H.-Y., Chien, C.-H., Wong, S.-K., Tsai, Y.-C., & Chiueh, T. (2008) Graphic-card cluster for astrophysics (GraCCA) - Performance tests. *New Astronomy* 13:418-435.