Using Graphics Processors to Accelerate Protein Docking Calculations

Dave Ritchie
Orpailleur Team
INRIA Nancy – Grand Est
Protein-Protein Interactions – Why Are They Important?

- Protein-protein interactions (PPIs) define the “machinery” of life.
- Humans have about 30,000 proteins, each having about 5 PPIs.
- Understanding PPIs could lead to immense scientific advances.
- Controlling PPIs could have huge therapeutic benefits (new drug molecules).
What is Protein Docking?

- Protein docking = predicting protein interactions at the molecular level

- If proteins are rigid $\Rightarrow$ six-dimensional search space

- But proteins are flexible $\Rightarrow$ multi-dimensional space!

- Modeling protein-protein interactions accurately is difficult!
Protein Docking Using Fast Fourier Transforms

- Conventional approaches digitise proteins into 3D Cartesian grids...

![Grids and Protein Diagram]

- ...and use FFTs to calculate translational correlations:

\[ C[\Delta x, \Delta y, \Delta z] = \sum_{x,y,z} A[x, y, z] \times B[x + \Delta x, y + \Delta y, z + \Delta z] \]

- BUT have to rotate one protein and repeat, which becomes expensive!

- POLAR coordinates allow rotational nature of the problem to be exploited
Some Theory – 2D Spherical Harmonic Surfaces

- Use spherical harmonics (SHs) as orthogonal shape “building blocks”

  - Reals SHs \( y_{lm}(\theta, \phi) \), and coefficients \( a_{lm} \)

- Encode distance from origin as SH series to order \( L \):

  \[
  r(\theta, \phi) = \sum_{l=0}^{L} \sum_{m=-l}^{l} a_{lm} y_{lm}(\theta, \phi)
  \]

- Calculate coefficients by numerical integration

- ROTATIONS: \( a'_{lm} = \sum_{m'=-l}^{l} R_{mm'}^{(l)}(\alpha, \beta, \gamma) a_{lm} \)

- Good for shape-matching, not so good for docking...

Docking Needs a 3D “Spherical Polar Fourier” Representation

- Need to introduce special orthonormal Laguerre-Gaussian radial functions, $R_{nl}(r)$

- $R_{nl}(r) = N_{nl}^{(q)} e^{-\rho/2} \rho^{l/2} L^{(l+1/2)}_{n-l-1}(\rho); \quad \rho = r^2/q, \quad q = 20.$

- Surface Skin: $\sigma(r) = \begin{cases} 1; \quad r \in \text{surface skin} \\ 0; \quad \text{otherwise} \end{cases}$  
  Interior: $\tau(r) = \begin{cases} 1; \quad r \in \text{protein atom} \\ 0; \quad \text{otherwise} \end{cases}$

- Parametrise as: $\sigma(r) = \sum_{nlm} a_{nlm}^l(r) y_{lm}(\theta, \phi), \text{ etc.}$

- TRANSLATIONS: $a_{nlm}^{\sigma'} = \sum_{n'l'} T_{nl,n'l'}^{|m|}(R) a_{n'l'm}^{\sigma'}$
SPF Protein Shape-Density Reconstruction

Interior density:

$$\tau(r) = \sum_{n=1}^{N} \sum_{l=0}^{n-1} \sum_{m=-l}^{l} a_{nlm}^\sigma R_{nl}(r) y_{lm}(\theta, \phi)$$

<table>
<thead>
<tr>
<th>Image</th>
<th>Order</th>
<th>Coefficients</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Gaussians</td>
<td>-</td>
</tr>
<tr>
<td>B</td>
<td>N = 16</td>
<td>1,496</td>
</tr>
<tr>
<td>C</td>
<td>N = 25</td>
<td>5,525</td>
</tr>
<tr>
<td>D</td>
<td>N = 30</td>
<td>9,455</td>
</tr>
</tbody>
</table>

Protein Docking Using SPF Density Functions

Favourable:
\[ \int \left( \sigma_A(r_A) \tau_B(r_B) + \tau_A(r_A) \sigma_B(r_B) \right) dV \]

Unfavourable:
\[ \int \tau_A(r_A) \tau_B(r_B) dV \]

Score:
\[ S_{AB} = \int \left( \sigma_A \tau_B + \tau_A \sigma_B - Q \tau_A \tau_B \right) dV \]
Penalty Factor: \( Q = 11 \)

Orthogonality:
\[ S_{AB} = \sum_{nlm} \left( a_{nlm}^{\sigma} b_{nlm}^{\tau} + a_{nlm}^{\tau} \left( b_{nlm}^{\sigma} - Q b_{nlm}^{\tau} \right) \right) \]

Search:
6D space = 1 distance + 5 Euler rotations: \((R, \beta_A, \gamma_A, \alpha_B, \beta_B, \gamma_B)\)

Nvidia Graphics Processors

- Modern GPUs have very high compute performance
- SIMT architecture = simultaneous instructions, multiple threads

- NVIDIA GPUs:
  - Up to 4Gb memory
  - Up to 240 arithmetic “cores”
  - Up to Tera-flop performance
  - Easy API with C++ syntax
  - Grid of threads SIMT model

- BUT – for best results, need to understand the hardware...
The CUDA Device Architecture

- Typically 8–16 multi-processor blocks, each with 16 thread units

- NB. only a very small amount of fast shared memory is available
- NB. global memory is $\sim 80x$ slower than shared memory
- Strategy: aim for “high arithmetic intensity” in shared memory
CUDA Programming Example - Matrix Multiplication

• Matrix multiplication \( C = A \times B \)

• Each thread is responsible for calculating one element: \( C[i,k] \)

\[
\begin{align*}
\text{C} & = \text{A} \times \text{B} \\
\text{by} & = \text{tx} \\
\text{bx} & = \text{ty}
\end{align*}
\]

• Conventional algorithm: rows and columns
  \( C[i,k] = A[i] \times B[k] \)

• Thread-block algorithm working on tiles

• Threads co-operate by reading & sharing tiles of A & B

• Multi-processor launches multiple blocks to compute all of C

• Executing thread-blocks concurrently hides global memory latency
CUDA Programming Example – Matrix Multiplication Kernel

__global__ void matmul(int wA, int wB, float *A, float *B, float *C) {
    float Cik = 0.0; // thread-local result variable
    int bx = blockIdx.x, tx = threadIdx.x; // thread subscripts
    int by = blockIdx.y, ty = threadIdx.y; // ("this" thread is one of a 2-D grid)

    __shared__ float a_sub[16][16], b_sub[16][16]; // declare shared memory

    for (int j=0; j<wA; j+=16) { // thread-local loop over tiles of A and B
        int ij = (16*by+ty)*wA + (j+tx); // thread-local array subscripts
        int jk = (j+ty)*wB + (16*bx+tx);

        a_sub[ty][tx] = A[ij]; // copy global data to shared memory ("I/O")
        b_sub[ty][tx] = B[jk];

        __syncthreads(); // wait until all memory I/O has finished

        for (int jj=0; jj<16; jj++) {
            Cik += a_sub[ty][jj] * b_sub[jj][tx]; // multiply row*column in current tiles
        }

        __syncthreads(); // synchronise threads before starting more I/O
    }

    C[(16*by+ty)*wB + (16*bx+tx)] = Cik; // copy local result -> global memory
}
**GPU Implementation Part 1 – Rotate and Translate Protein A**

1. On CPU, calculate multiple \((\beta_A, \gamma_A)\) rotations of protein A

2. On CPU, re-index translation matrices and rotated coefficients into regular sparse arrays

3. On GPU, translate multiple protein A coefficients using tiled matrix multiplication
GPU Implementation Part 2 – Perform Multiple FFTs

- The overall aim is to calculate multiple 1D FFTs of the form:

\[
C(\alpha_B) = \sum_m e^{-im\alpha_B} \sum_{nl} A_{nlm}^{\sigma}(R, \beta_A, \gamma_A) \times B_{nlm}^{\tau}(\beta_B, \gamma_B)
\]

4. On GPU, cross-multiply transformed A with rotated B coefficients (as above)

5. On GPU, perform batch of 1D FFTs using cuFFT and save best orientations

- 3D FFTs in \((\alpha_B, \beta_B, \gamma_B)\) can be calculated in a similar way...
Results – GPU v’s CPU Docking Performance

- Key Hex functions implemented using only 5 or 6 CUDA kernels
- 1D and 3D FFTs are calculated using Nvidia’s cuFFT library
- Here, GPU = Nvidia FX-5800, CPU = Intel i7-965

- Hex 1D correlations are up to 100x faster on FX-5800 than on iCore7
- Overall, including set-up, Hex 1D FFT is about 45x faster on FX-5800 than on iCore7
Results – Multiple GPUs and CPUs

- With Multi-threading, we can use as many GPUs and CPUs as are available

- For best performance: use 2 GPUs alone, or 6 CPUs plus 2 GPUs

- With 2 GPUs, docking takes only about 15 seconds – very important for large-scale!
“Hex” and “HexServer” – Publicly Available Docking Tools

Macindoe et al. (2010), Nucleic acids Research (featured article)
Conclusions and Future Prospects

- Protein-protein docking on a GPU now takes only a few seconds:
  - This was implemented using only 5 or 6 GPU kernels
  - But a lot of low-level CPU code had to be re-written

- High-throughput multi-shape comparison is now feasible:
  - Probing PPI networks...
  - Assembling multi-component machines...
  - Electron-microscopy density fitting...
  - Full 3D small-molecule virtual screening...
  - Protein shape matching and classification...
Acknowledgments

BBSRC 1996–2000
EPSRC 2000–2006
ANR 2009–2010

Software & Papers:  http://hex.loria.fr/
HexServer:  http://hexserver.lORIA.fr/
Extra Slides
Knowledge of even only one key residue can reduce search space enormously...

This accelerates the calculation and helps to reduce false-positive predictions
5D FFT Correlations from Complex Overlap Expressions
(Ritchie, Kozakov, Vajda, (2008) Bioinformatics, 24, 1865–1873)

Complex SHs, $Y_{lm}$:  
$$ y_{lm}(\theta, \phi) = \sum_t U_{mt}^{(l)} Y_{lt}(\theta, \phi) $$

Complex coefficients:  
$$ A_{nlm} = \sum_t a_{nt} U_{lm}^{(l)} $$

Complex overlap:  
$$ S = \sum_{kjsmlnv} D_{ms}^{(j)}(0, \beta_A, \gamma_A) A_{kjs}^{*} T_{kj,nl}^{(|m|)}(R) D_{mv}^{(l)}(\alpha_B, \beta_B, \gamma_B) B_{nlv} $$

Collect coefficients:  
$$ S_{js,lv}^{(|m|)}(R) = \sum_{kn} A_{kjs}^{*} T_{kj,nl}^{(|m|)}(R) B_{nlv}, \quad k > j; n > l $$

To give:  
$$ S = \sum_{jsmlv} D_{ms}^{(j)}(0, \beta_A, \gamma_A) S_{js,lv}^{(|m|)}(R) D_{mv}^{(l)}(\alpha_B, \beta_B, \gamma_B) $$

Expand as exponentials:  
$$ D_{mv}^{(l)}(\alpha, \beta, \gamma) = \sum_{t} \Gamma_{lv}^{tm} e^{-im\alpha} e^{-it\beta} e^{-iv\gamma} $$

Hence:  
$$ S = \sum_{jsmlvrt} \Gamma_{js}^{rm} S_{js,lv}^{(|m|)}(R) \Gamma_{lv}^{tm} e^{-i(r\beta_A - s\gamma_A + m\alpha_B + t\beta_B + v\gamma_B)} $$
Translation Matrices From Fourier-Bessel Transform Theory

Using spherical Bessel transforms:

\[ \tilde{R}_{nl}(\beta) = \sqrt{\frac{2}{\pi}} \int_0^\infty R_{nl}(r) j_l(\beta r) r^2 dr; \quad R_{nl}(r) = \sqrt{\frac{2}{\pi}} \int_0^\infty \tilde{R}_{nl}(\beta) j_l(\beta r) \beta^2 d\beta \]

it can be shown that

\[ T_{n'l',nl}(R) = \sum_{k=|l-l'|}^{l+l'} A_k^{(l'l'|m)} \int_0^\infty \tilde{R}_{nl}(\beta) \tilde{R}_{n'l'}(\beta) j_k(\beta R) \beta^2 d\beta \]

where

\[ A_k^{(l'l'|m)} = (-1)^{k+l'-l} (2k + 1)[(2l + 1)(2l' + 1)]^{1/2} \begin{pmatrix} l & l' & k \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} l & l' & k \\ m & m & 0 \end{pmatrix} \]

- Can derive analytic formulae for both GTO and ETO radial functions
- Requires high precision math library (GMP)...
- Calculate once for \( R = 1, 2, 3, \ldots 50 \) Å and store on disk (\( \sim 200\text{Mb} \))
Inside Hex – High Order FFTs and GPUs

- The SPF gives an analytic way to calculate \( \text{TRANSLATIONAL} + \text{ROTATIONAL} \) correlations:

\[
S_{AB} = \sum_{jsmlvrt} \Lambda_{js}^{rm} T_{js,lv}^{(|m|)} (R) \Lambda_{lv}^{tm} e^{-i(r\beta_A-s\gamma_A+m\alpha_B+t\beta_B+v\gamma_B)}
\]

- This allows high order FFTs to be used – 1D, 3D, and 5D
- It also allows calculations to be easily ported to modern GPUs
  - Up to 240 arithmetic “cores”
  - Grid of threads SIMT model
  - Correlation speed-up \( \geq 100x \)
  - Overall speed-up = 45x
- GPU docking takes 15 seconds (475x faster than ZDOCK) – very important for large-scale!

D.W. Ritchie, et al. HealthGrid (2010), To Appear
D.W. Ritchie, V. Venkatraman (2010), In review