GPU PROGRAMMING: 2D FITTING FOR SMALL-ANGLE NEUTRON SCATTERING DATA

Helen Park
Paul Kienzle
Matt Wasbrough
DATA USED: COLLECTED BY NEUTRON SCATTERING

- Neutrons scatter off large scale structures in the material
- Reveal structure of material
- Colors refer to intensity/density of neutrons scattered
SURFACTANT DATA

- Has a hydrophobic tail, hydrophilic head
- Form variety of shapes
- Used in soap, drug delivery, even mayonnaise!
• Device spins the surfactants at different speeds
• Surfactants change shape
• How does the geometry of the material change under different frequencies?
2D FITTING

- Fitting allows a better understanding of the data
- Chose a model (such as Core-Shell-Cylinder, Lamellar, Triaxial-Ellipse) to best describe the phase of the surfactant micelles
- Manipulate variables to fit the shape and orientation of the data

- These are examples of data collected under various amounts of shear stress

0 Hz

1230 Hz

7000 Hz
SASVIEW: FITS THE DATA

- Choose model that best describes data
- Issues: Slow, crashes often
• GPU programming: faster
• CPU—found in most PCs
• GPU—previously used solely in 3D gaming
• GPU allows parallel processing, 1000s of threads, 100s more cores than CPU
• GPUs are cheap, fast, and energy efficient

<table>
<thead>
<tr>
<th>Processor</th>
<th>US$</th>
<th>Cores</th>
<th>GHz</th>
<th>GFlops</th>
<th>GFlops/W</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel i7 x6</td>
<td>1000</td>
<td>6</td>
<td>3.3</td>
<td>100</td>
<td>&lt;1</td>
</tr>
<tr>
<td>AMD Phenom II x6</td>
<td>300</td>
<td>6</td>
<td>4.0</td>
<td>100</td>
<td>&lt;1</td>
</tr>
<tr>
<td>NVidia GeForce 480</td>
<td>500</td>
<td>480</td>
<td>0.7</td>
<td>1400</td>
<td>5</td>
</tr>
<tr>
<td>ATI Radeon 5970</td>
<td>700</td>
<td>3200</td>
<td>0.7</td>
<td>4600</td>
<td>15</td>
</tr>
</tbody>
</table>
For matrix multiplication: \textbf{GPU = 150*CPU MFLOPS} (Mega floating point operations per second)

- But, need to tune algorithm and memory transfer for every sized GPU, and for each kernel and program

<table>
<thead>
<tr>
<th>Speed-ups of different projects</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>MA Hospital</td>
<td>300X</td>
</tr>
<tr>
<td>U Rochester</td>
<td>160X</td>
</tr>
<tr>
<td>U Amsterdam</td>
<td>150X</td>
</tr>
<tr>
<td>Harvard</td>
<td>130X</td>
</tr>
<tr>
<td>U Pennsylvania</td>
<td>130X</td>
</tr>
<tr>
<td>Nanyang Tech</td>
<td>130X</td>
</tr>
<tr>
<td>U Illinois</td>
<td>125X</td>
</tr>
<tr>
<td>Cambridge U</td>
<td>100X</td>
</tr>
<tr>
<td>Boise State</td>
<td>100X</td>
</tr>
<tr>
<td>Florida U</td>
<td>100X</td>
</tr>
</tbody>
</table>
The Host (CPU) passes memory buffers, kernels, and queue commands to the device, and receives the result, also in a buffer
The Context holds the GPU(s)—varying shaped & sizes
A Device (GPU) handles the computations.
Handling of memory transfer

**global → local → private**

**effect speeds**

A Device has global, local, and private memory, and many work groups that perform calculations in parallel.
OPENCL DESIGN

Central Blackboard: Global Memory

Class Blackboard: Local Memory

Notebook: Private Memory
A work group, 128x128 performs computations.

Work-groups run in parallel, do not interact.
The Kernel is the physical code, or program, that are computed by the device.
The Queue relays and manages how kernels and buffers are translated and organized.
Our Kernel: calculates scattering intensity, stored in a 2D array

- In our algorithm, work-group takes random Qx & Qy and calculates the scattering density at that point
- Performs this until every Qx & Qy complete
- Adds results and returns to CPU

Cylinder Model equations:

\[
P(q, \alpha) = \frac{\text{scale}}{V_2} \cdot f(q) + \text{bkg}
\]

\[
f(q) = 2(\rho_s - \rho) V_2 \sin[\alpha L \cos \alpha / 2] / (\alpha L \cos \alpha / 2) \frac{[q \sin \alpha]}{[q \sin \alpha]}
\]

\[
+ 2(\rho_s - \rho_{\text{model}}) V_2 \sin[q (L + 2t) \cos \alpha / 2] / q (L + 2t) \cos \alpha / 2 \frac{[q (r + t) \sin \alpha]}{[q (r + t) \sin \alpha]}
\]

\[
V_2 = \pi (R + t)^2 \cdot (L + 2t)
\]
Loop for polydispersity in CPU
- Size of polydispersity corresponds to width of bell-curve
- Allow a variety of values for a variable (like length)
- For example, high polydispersity in theta gives a larger range of angles
- Also, the more bins, the more accurate the fit

← High polydispersity, but low number of bins (5)
→ Lower polydispersity, but many bins (40)

Cost: higher bins means much slower fit, so need to balance
• In CPU, program the context, device(s), the queue to relay information, and write the buffers for variables, and return values
• Using bumps, loop (again!) over the entire program to fit different variables

```python
model = SasModel(data, CpuCylinder).
scale = 0.0104,
radius = 92.5,
length = 798.3,
sldCyl = 0.29e-6,
sldSol = 7.105e-6,
background = 5,
cyl_theta = 0,
cyl_phi = 3,
cyl_theta_pd = 22.11,
cyl_theta_pd_n = 20,
cyl_theta_pd_n sigma = 3,
radius_pd = -0.0054,
radius_pd_n = 10,
radius_pd_n sigma = 3,
length_pd = 0.453,
length_pd_n = 10,
length_pd_n sigma = 3,
cyl_phi_pd = 3,
cyl_phi_pd_n = 1,
cyl_phi_pd_n sigma = 3,

cyl_theta_range(1, 500)
cyl_theta_range(1, 400)
cyl_theta_range(-90, 100)
cyl_theta_range(0, 90)
cyl_theta_range(0, 90)
cyl_theta_range(-5, 5)
cyl_theta_range(0, 90)
cyl_theta_range(0, 100)
cyl_theta_range(0, 100)
cyl_theta_range(0, 100)
cyl_theta_range(0, 100)
```
<table>
<thead>
<tr>
<th>Model</th>
<th>Sasview</th>
<th>GPU</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cylinder</td>
<td>3977.7ms</td>
<td>202.3ms</td>
<td>19.7X</td>
</tr>
<tr>
<td>Ellipse</td>
<td>2953.2ms</td>
<td>285.5ms</td>
<td>10.3X</td>
</tr>
<tr>
<td>Core-Shell-Cylind</td>
<td>71149.9ms</td>
<td>4474.7ms</td>
<td>15.9X</td>
</tr>
<tr>
<td>Triaxial-Ellipse</td>
<td>100627ms</td>
<td>6500.2ms</td>
<td>15.5X</td>
</tr>
<tr>
<td>Lamellar</td>
<td>69.2ms</td>
<td>6.2ms</td>
<td>11.2X</td>
</tr>
</tbody>
</table>
• Day-long fit to hour-long fit
• Paul: **50X** faster—cuts out values when the polydispersity weight is low, use local memory
• If 4 GPUs: 4 times faster (200X)
• Allows increased control over simultaneous fitting, multiple-model fitting, and the angular limits of integration in 1D
• Also used models to fit various scattering data
• Here is an example of a fit for a surfactant at 0 Hz
• The left is the data, the middle is the fit, and the right is the residuals of the data
ACKNOWLEDGEMENTS

- Paul Kienzle
- Dr. Matt Wasbrough
- Aaron West
- Yusuf Ameri cainaki
<table>
<thead>
<tr>
<th>OPENCL VS CUDA</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CUDA:</strong></td>
</tr>
<tr>
<td>• Easier to understand; more tutorials online and in books</td>
</tr>
<tr>
<td>• CUDA: need to have whole toolchain available</td>
</tr>
<tr>
<td><strong>OpenCL:</strong></td>
</tr>
<tr>
<td>• Newer, so not much online; less accessible to learn</td>
</tr>
<tr>
<td>• Broader range of hardware supported</td>
</tr>
<tr>
<td>• Simply link the shared library to access</td>
</tr>
</tbody>
</table>