An Out-of-core Implementation of Block Cholesky Decomposition on A Multi-GPU System

Lin Cheng, Hyunsu Cho, Peter Yoon, Jiajia Zhao
Trinity College, Hartford, CT
Outline

• Block Cholesky decomposition
• Multi-GPU system
• Out-of-core implementation
• Inter-device communication
• Extension to disk I/O
• Performance
Block Cholesky decomposition

- Widely used matrix factorization
- Dense linear algebra routine
- Diagonally dominant, numerically stable
- Block version
  - Cache-aware block size

\[ A = G^T G \]
Procedure: Phase I

\[
\begin{array}{cccccccc}
A_{00} & A_{01} & A_{02} & \cdots & A_{0,q-1} \\
\end{array}
\]

\[ k = 0 \]
Procedure: Phase II

\[ k = 0 \]

\[
\begin{array}{cccc}
A_{00} & A_{01} & A_{02} & \cdots & A_{0,q-1} \\
\end{array}
\]

\[ k = 0 \]
Procedure: Phase III
Procedure: Repeat

\[
\begin{array}{cccc}
A_{00} & A_{01} & A_{02} & \cdots & A_{0,q-1} \\
A_{11} & A_{12} & & \cdots & A_{1,q-1} \\
A_{22} & & & \cdots & A_{2,q-1} \\
& & & \vdots & \\
& & & & A_{q-1,q-1}
\end{array}
\]
General Purpose GPU (GPGPU)

- Cost-efficient parallel platform
- Many-core approach
Multi-GPU system

- GPU devices maintain separate memory & kernel invocations
- Coordination becomes a significant issue
- Memory transfer between devices is costly
- Load-balancing is necessary to achieve high performance
Out-of-core implementation

• GPU memory as **cache** for main CPU memory

• Roughly 1/N of matrix were loaded to each device
  – Balanced load
  – Minimal communication w/ the host
  – Write back to main memory only finished parts

• Submatrix size
  – small enough to load several of them at once
  – large enough to reduce latency
Inter-device communication

• Happens whenever we transition from one phase to another

• Data transfer can be costly

• Possible solutions
  – Peer-to-peer: 2x fast
  – **Overlapping** of computation and data transfer

• Synchronization is critical.
  – CPU threads control GPU devices.
  – Between Phases II and III.
<table>
<thead>
<tr>
<th>Stream 0</th>
<th>Stream 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Do Phase II</td>
<td>Flush Phase I to Host</td>
</tr>
<tr>
<td>Communication to prepare for Phase III</td>
<td>Communication to prepare for Phase II</td>
</tr>
<tr>
<td></td>
<td>Communication to prepare for Phase II</td>
</tr>
<tr>
<td>Do Phase III</td>
<td></td>
</tr>
<tr>
<td>Do Phase I</td>
<td>Flush Phase II to Host</td>
</tr>
<tr>
<td>Communication to prepare for Phase II</td>
<td></td>
</tr>
</tbody>
</table>
Extension to disk I/O

- For larger matrices, use main memory as cache for disks
- Prefetching, delayed write: exploit locality
Performance

• CPU: dual 2.4 GHz Intel® Xeon® quad-core

• Main memory: 16GB

• GPU: four Tesla C2050 graphics cards with 3GB memory

• CUDA 4.2 Runtime

• 33x compared to PLASMA, a numerical linear algebra library for multicore CPU

• Scalable to larger systems
  – 65,000 x 65,000 matrix amounts to 32GB
Conclusion

• Our implementation is scalable to very large systems.

• We streamlined operation across three memory layers.

• We were able to apply it to image segmentation.