Multi-Platform LU-Decomposition Solution in OpenCL

Ehsan Nasiri, Rafat Rashid, Saurabh Verma
Supervisor: Vaughn Betz  Administrator: Christiana Amza  Section: 8  Project ID: 2011027

1. MOTIVATION
LU-Decomposition (LUD) is a basic algebraic operation with a wide range of applications.

Figure 1: (a) Electromagnetic solver simulation (b) Structure stress simulation (c) Spice Voltage simulation

LU-Decomposition is the most expensive part of solving a system of linear equations.

Existing challenges:
• High computing complexity: O(n³)
• Long computation/simulation times
• Existing solutions provided by Intel (MKL) and AMD (OpenCL SDK) are expensive and platform specific

2. OUR SOLUTION
Develop a high-performance code that can execute across heterogeneous devices without recompilation.

Test Hardware:
CPU: Intel Core i7-2600K @ 3.4 GHz
GPU: AMD Radeon HD 6900 Series (Cayman)
FPGA: Stratix IV GX

OpenCL
Free cross-platform parallel computing language
Recently developed by the Khronos group
• Apple, Intel, AMD, Altera, TI, etc.

3. GOALS AND OBJECTIVES
• Functional Correctness: Produce the correct L and U on each of the target platforms
• Runtime Efficiency: Beat runtime of serial C++ LUD implementation
• Portability Across Devices: Cross-platform LUD solution using OpenCL

4. SYSTEM DESIGN
4.1 BLOCKED ALGORITHM APPROACH
• Divide matrix into smaller blocks
• Dependencies: black block depends on top-most and left-most gray blocks

Why use Blocking?
• Operates on at most 3 blocks at any given time
• Correct block size takes advantage of caches, thus reducing the time required to fetch the data.

4.2 OPENCL LUD ALGORITHM DESIGN ON CPU
Our implementation on the CPU performs the computation as shown in Figure 4.b):
• Keep the CPU occupied by using parallelism within a block
• Optimal block size of 250 (with 250 concurrent threads) is used
• Utilizes CPU caches and SIMD instructions
• Dependencies between blocks impose a restriction on the level of parallelism we can achieve

4.3 OPENCL LUD ALGORITHM DESIGN ON GPU
CPU to GPU Port
Single SM + Global Memory
Aligned Coalesced Data Access
Remove Bank Conflicts
Multiple SMs
Local Memory

Two levels of parallelism exploited:
• Perform computations within a block in parallel (like it is done on the CPU)
• Compute multiple blocks in parallel (coloured blocks in Figure 8) by taking advantage of the Streaming Multiprocessors on a GPU

Figure 7: OpenCL Algorithm Design Process

5. TESTING, VERIFICATION AND RESULTS
GFLOPs = \( \frac{2}{3} \times (\text{matrix size})^3 \) / \( \text{runtime} \)

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>GFLOPs</th>
</tr>
</thead>
<tbody>
<tr>
<td>C++ Non-Blocked</td>
<td>1</td>
</tr>
<tr>
<td>C++ Blocked</td>
<td>3.02</td>
</tr>
<tr>
<td>OpenCL CPU</td>
<td>5.01</td>
</tr>
<tr>
<td>OpenCL GPU</td>
<td>7.5</td>
</tr>
</tbody>
</table>

We successfully beat the runtime of the C++ Blocked code with our OpenCL CPU and GPU implementations.

6. CONCLUDING REMARKS
The OpenCL framework makes it relatively easy to write cross-platform code. However, code that is optimized for one platform is most likely not going to perform at optimum on another platform.

7. FUTURE WORK
• Optimize algorithm for FPGA
• Utilize SIMD SSE instructions on GPU
• Extend algorithm to operate on matrices larger than 10k x 10k elements
• For a block size of m, launch m x m threads instead of the current m to exploit more parallelism

ACKNOWLEDGEMENTS
• Vaughn Betz (Supervisor)
• Christiana Amza (Administrator)
• Deshanand Singh (Altera)
• Alex Rodionov (Altera)