Designing Code Variants for Applications with Nested Parallelism on GPUs
Da Li, Michela Becchi
University of Missouri - Columbia

Abstract
The effective parallelization of applications exhibiting irregular nested parallelism is still an open problem. In particular, a naive mapping of irregular codes to the GPU hardware may lead to resource underutilization and, thereby, limited performance. In this work, we focus on two computational patterns exhibiting nested parallelism: irregular nested loops and recursive algorithms operating on tree and graph data structures. We propose different parallelization templates aimed at increasing the GPU utilization of these codes. Specifically, we investigate different mechanisms to effectively distribute irregular work to streaming multiprocessors and GPU cores. We target the Fermi and the Kepler architecture; in the latter case, we also study parallelization templates relying on dynamic parallelism and propose mechanisms to maximize the work performed.

Recursive Algorithms

- **flat parallelism**

\[
\text{flat_kernel (graph } g) \{
\text{thread-mapped-loop} (\text{node } n \in g.\text{nodes}) \{
\text{atomic}\{g.\text{descendants}[n] += g.\text{children}(c)\};
\}
\}
\]

**Example Application: Tree descendants**

\[
\text{set_low} = (i :: x[i] < TH)
\]

\[
\text{set_high} = (i :: x[i] > TH)
\]

\[
\text{thread-mapped_kernel(set_low)}
\]

\[
\text{block-mapped_kernel(set_high)}
\]

- **recursive parallelism – naïve approach**

\[
\text{naive_rec_kernel (graph } g, \text{ node } n) \{
\text{thread-mapped-loop(node } c \in g.\text{children}(n))\{
\text{if (!leaf(c)) naive_rec_kernel<1,blockSIZE>(g,c);}
\}
\}
\]

**Example Application: Hierarchical recursion**

\[
\text{hier_rec_kernel(gridSIZE,blockSIZE)(g,c);}
\]

Irregular Nested Loops

Irregular nested loop sizes lead to hardware underutilization

<table>
<thead>
<tr>
<th>X</th>
<th>7</th>
<th>326</th>
<th>9</th>
<th>21</th>
<th>46</th>
<th>234</th>
<th>22</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>iteration: 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>7</td>
<td>326</td>
<td>9</td>
<td>21</td>
<td>46</td>
<td>234</td>
<td>22</td>
<td>14</td>
</tr>
<tr>
<td>iteration: 30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>7</td>
<td>326</td>
<td>9</td>
<td>21</td>
<td>46</td>
<td>234</td>
<td>22</td>
<td>14</td>
</tr>
<tr>
<td>iteration: 50</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Experiments

- **SpMV**

- **SSSP**

- **Tree Descendants on Regular Tree**

  - GPU platform: NVIDIA K20 GPU, 13 x 192 CUDA cores, 4,800 Mbytes global memory
  - CPU platform: Intel Xeon E5620, 15MB L1 Cache
  - CUDA kernel configuration of our implementation: 192 threads per block

References


Contact us: da.li@mail.missouri.edu