Using MIC to accelerate a typical data-intensive application: the Breadth-first Search

Gao Tao, Lu Yutong and Suo Guang
Contents

1. Background
2. Optimizations
3. Experimental Results
4. Summary
1. Background

- **Data-intensive applications**
  - Draw more and more attentions
  - Graph 500 is proposed, BFS is the kernel component

- **The MIC coprocessor**
  - Be designed for highly parallel computing

- **Our research**
  - A typical data-intensive application (BFS) on MIC
The MIC architecture
2. Optimizations

- **Native optimizations**
  - Use multi-threads
  - Use 512-bits SIMD instructions

- **Offload optimizations**
  - Use MIC as a accelerator
Data Structure

Our optimizations are based on the level-synchronous BFS. The current queue holds vertexes will be explored in this level and the next queue holds vertexes should be explored in the next level. Within each level, the algorithm scans vertexes in the current queue. Neighbors, which haven’t been explored, are inserted into the next queue.

• `in_queue` : the current queue (bitmap)
• `out_queue` : the next queue (bitmap)
• `visited` : the visited bitmap
• `P` : the predecessor map
Native Optimizations

- Multi-thread optimizations – relax method
  - Two data races conditions exist
  - Traditional method uses atomic operation to make correctness
  - We relax the data races and restore the bitmap at end of a level
Native Optimizations

• SIMD optimizations
  – The relax method eliminates the atomic operations
  – SIMD instructions is used to inspect the neighbors of a vertex in parallel

```plaintext
for v0 ∈ in_queue parallel do
  for all vertexes adjacent to v0 do
    v1 ← VecLoadNeighbors(v0)
    vec_visited ← VecGather(visited, v1)
    mask ← VecTest(vec_visited, v1)
    if mask = 0 then
      continue
    VecStore(child, v1, mask)
    for i ← 0 to 16 do
      if mask[ith] = 1 ∩ P[child[i]] = ∞ then
        P[child[i]] ← v0 - n
        VecSetBitmap(out_queue, child[i])
```
Offload Optimizations

- Most vertexes are expanded in middle levels
- The scanned vertexes are much more than the expanded vertexes
1. If the vertex amount in the current queue exceeds a baseline, MIC is used to accelerate.
2. Otherwise, only CPU is used.
3. Experimental Results

- platform

<table>
<thead>
<tr>
<th></th>
<th>CPU</th>
<th>MIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Intel(R) Xeon(R) E5-2670</td>
<td>Knight Corner</td>
</tr>
<tr>
<td>clock rate</td>
<td>2.60GHz</td>
<td>1.1GHz</td>
</tr>
<tr>
<td>sockets</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>cores (per socket)</td>
<td>8</td>
<td>57</td>
</tr>
<tr>
<td>threads (per core)</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>L1 cache (per core)</td>
<td>32KB</td>
<td>32KB</td>
</tr>
<tr>
<td>L2 cache (per core)</td>
<td>256KB</td>
<td>512KB</td>
</tr>
<tr>
<td>L3 cache (per socket)</td>
<td>20MB</td>
<td>-</td>
</tr>
</tbody>
</table>
Relax optimization on CPU

- The Experimental Results on CPU

![Graph showing TEPS vs Threads for different configurations]
Relax optimization on MIC

- The Experimental Results on MIC
SIMD optimization

• Speedup of Different Process Amount

![Graph showing speedup vs scale for different process amounts]
SIMD optimization

- Speedup of One Process with Different Thread Amount
Offload optimization

• The TEPS of Offload Algorithm

![Graph showing TEPS (transactions per second) for different levels of offload with varying numbers of processes and scale values. The graph includes data points for 1 process with SCALE=24, 2 processes with SCALE=25, and 4 processes with SCALE=26, with TEPS values ranging from 0 to $15 \times 10^7$. The graph compares two categories: graph500-replicated-csc and offload.]
Offload optimization

- The Time in Every Level
Summary

• We propose the relax and SIMD optimization methods on MIC

• We propose the offload algorithm for CPU and MIC hybrid computing

• We still work on this topic and have gained some new results, which will report recently!
Thank you!