A Preliminary Study of Compiler Transformations for Graph Applications on the EMU System

Prasanth Chatarasi, and Vivek Sarkar

Habanero Extreme Scale Software Research Project
Georgia Institute of Technology, Atlanta, USA
Introduction – Graph applications

• Increasing in importance for high-performance
  — With the advent of "big data"

• Random memory access patterns
  — Inefficient utilization of memory & cache in CPU and GPU’s

• Growing interest to innovate architectures
  — To handle applications with weak-locality
EMU [Kogge et al. IA3’16]

- A highly scalable near-memory multi-processor
  - 8 nodes → 8 nodelets/node → 4 cores/nodelet → 64 threads/core
  - Cilk programming model for expressing parallelism

A Comparison b/w EMU and Xeon on a pointer-chasing benchmark
-- Hein et al. [IPDPSW'18]

http://www.emutechnology.com/products/ #lightbox/0/
1) Key features – Thread migration

- Automatic thread migrations on an access to a non-local data
  - Computation moves instead of data
- Benefits of thread migration
  - Sparse matrix vector multiply
    - Kogge et al. [IA3’17]
  - BFS algorithm
    - Belviranli et al. [HPEC’18]

\[ y = x + 1 \]
\[ z = y + 1 \]
2) Key features – Remote atomic updates

- Atomic updates that do NOT cause a thread migration
  - Sends a packet having data and operation to be performed

- Used when a thread doesn’t need a return value of atomic operation
  - Otherwise, explicit FENCE required to block the thread

![Diagram](http://www.emutechnology.com/products/#lightbox/0/)

ATOMIC_ADD
(Z, 1)

Data: 1
Operation: Add

http://www.emutechnology.com/products/#lightbox/0/

Prasanth Chatarasi et al, MCHPC 18
Challenges with the EMU system

- Overheads from thread migrations, thread creation and synchronization.

- We focus on exploring compiler transformations to reduce the overheads and improve performance
  - High-level compiler transformations
    - Node fusion and Edge flipping
  - Low-level compiler transformations
    - Use of remote atomic updates
Agenda

• Introduction
• Compiler transformations
• Evaluation
  — Conductance
  — Bellman-Ford’s algorithm for single-source shortest path
  — Triangle counting
• Conclusions and future work
1) Node fusion

1: \texttt{parallel-for(v ∈ vertices) \{} \\
2: \hspace{1em} p1[v] = ... \\
3: \} // Implicit barrier \\

4: \texttt{parallel-for(v ∈ vertices) \{} \\
5: \hspace{1em} p2[v] = f(p1[v], ...) \\
6: \} \\

\begin{itemize}
\item Repeated migrations for 
\begin{itemize}
\item Same property across parallel loops 
\item Different properties of same vertex across parallel loops
\end{itemize}
\end{itemize}

- Can be reduced with fusing parallel loops 
Can reduce thread creation and synchronization overhead
2) Edge flipping

1: for(t-loop) {
2:   parallel-for(v ∈ vertices)
3:      for(u ∈ incoming_neighbors(v))
4:         p1[v] = f(p1[u], ...);
5: }

1: for(t-loop) {
2:   parallel-for(v ∈ vertices)
3:      cont = f(p1[v], ...);
4:      for(u ∈ outgoing_neighbors(v))
5:         atomic_update(p1[u], cont);
6: }

• Back and forth migrations
  — From a vertex to each of its incoming neighbor vertices
  - Can be reduced by pushing vertex contribution to its outgoing neighbors
Agenda

• Introduction

• Compiler transformations

• **Evaluation**
  – Conductance
  – Bellman-Ford’s algorithm for single-source shortest path
  – Triangle counting

• Conclusions and future work
Experimental setup

Table 1: Specifications of a single node of the Emu system.

<table>
<thead>
<tr>
<th></th>
<th>Emu system</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microarch</td>
<td>Emu1 Chick</td>
</tr>
<tr>
<td>Clock speed</td>
<td>150 MHz</td>
</tr>
<tr>
<td>#Nodelets</td>
<td>8</td>
</tr>
<tr>
<td>#Cores/Nodelet</td>
<td>1</td>
</tr>
<tr>
<td>#Threads/Core</td>
<td>64</td>
</tr>
<tr>
<td>Memorysize/Nodelet</td>
<td>8 GB</td>
</tr>
<tr>
<td>NCDRAM speed</td>
<td>1600MHz</td>
</tr>
<tr>
<td>Compiler toolchain</td>
<td>emusim.HW.x (18.08.1)</td>
</tr>
</tbody>
</table>

- **Evaluation on a single node of the Emu system**
  - Actual hardware on FPGA

- **Two experimental variants**
  - Original version of a graph algorithm
  - Transformed version after *manually* applying compiler transformations
Graph applications

- **Graph applications**
  - Conductance
  - Bellman-Ford’s algorithm for single-source shortest path
  - Triangle counting
    - Developed using the MEATBEE framework

- **Input data sets**
  - RMAT graphs from scale of 6 to 14 as specified by Graph500
    - \#vertices = 2^{scale}
    - \#edges = 16 * \#vertices

https://github.gatech.edu/ehein6/meatbee
1) Conductance algorithm

- Computes a flow from a given partition of graph to others

```python
    def CONDUCTANCE(V, id):
        for each v ∈ V do in parallel with reduction
            if v.partition_id == id then
                ▷ Thread migration for v.partition_id value
                din+ = v.degree
        for each v ∈ V do in parallel with reduction
            if v.partition_id != id then
                dout+ = v.degree
        for each v ∈ V do in parallel with reduction
            if v.partition_id == id then
                for each nbr ∈ v.nbrs do
                    if nbr.partition_id != id then
                        dcross+ = 1
        return dcross/((din < dout)?din : dout)
```

- Repeated migrations to same nodelet for the same property from multiple parallel loops

- All the parallel loops can be fused to avoid the overheads
Results after node fusion

- Speedups of up to 2.2x (geometric mean: 1.95x)
  - Also, a geometric mean reduction of 6.06% in thread migrations
2) Bellman-Ford’s algorithm

- Compute shortest paths from a single source vertex to all the other vertices in a weighted directed graph

```python
for t ← 0 to |V| − 1 do
   for each v ∈ V do in parallel
      for each u ∈ incoming_neighbors(v) do
         temp = distance(u) + weight(u, v)
         ➤ Migration for distance(u) value
         if distance(v) > temp then
            temp_distance(v) = temp
      end
   end
endfor
```

- Edge flipping followed by remote updates can avoid back and forth migrations
Results after Edge flipping + Remote updates

- Speedups of up to 3.8x (geometric mean: 1.38x)
  - Also, a geometric mean reduction of 36.39% in thread migrations
3) Triangle counting

- Computes the number of triangles in a given undirected graph
  - Also computes the number of triangles that each node belongs to

    ```
    for each v ∈ V do in parallel
    for each u ∈ v.nbrs do
      if nbr1 > v then
        for each w ∈ v.nbrs do
          if w > u then
            if edge_exists(u, w) then
              tc_count ++; // Atomic
              tc(v) ++; // Atomic
              tc(u) ++; // Atomic
              tc(w) ++; // Atomic
    
    ▷ Above regular atomics can be replaced by the remote updates.
    
    - Regular atomic updates can be replaced with remote updates
    ```

Prasanth Chatarasi et al, MCHPC 18
Results after using Remote updates

- Speedups of up to 1.3x (geometric mean: 1.05x)
  - Also, a geometric mean reduction of 54.55% in thread migrations
Conclusions & Future work

- EMU architecture is a potential choice for graph applications
  - But, a careful attention is required to make sure that overheads don’t hurt the benefits
  - Evaluated compiler transformations for three graph applications

<table>
<thead>
<tr>
<th>Applications</th>
<th>Transformations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conductance</td>
<td>Node fusion</td>
</tr>
<tr>
<td>Bellman-Ford’s algorithm</td>
<td>Edge flipping + Remote updates</td>
</tr>
<tr>
<td>Triangle counting</td>
<td>Remote updates</td>
</tr>
</tbody>
</table>

- Future work
  - Systematically explore & evaluate more compiler transformations

Any questions?
Acknowledgements

- MCHPC’18 Program committee
- Eric Hein and Jeff Young
  - Getting setup with EMU machine and the MEATBEE framework
- CRNCH center at Georgia Tech
  - Rogues gallery

http://crnch.gatech.edu/rogues-emu