

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

Prasanth Chatarasi and Vivek Sarkar

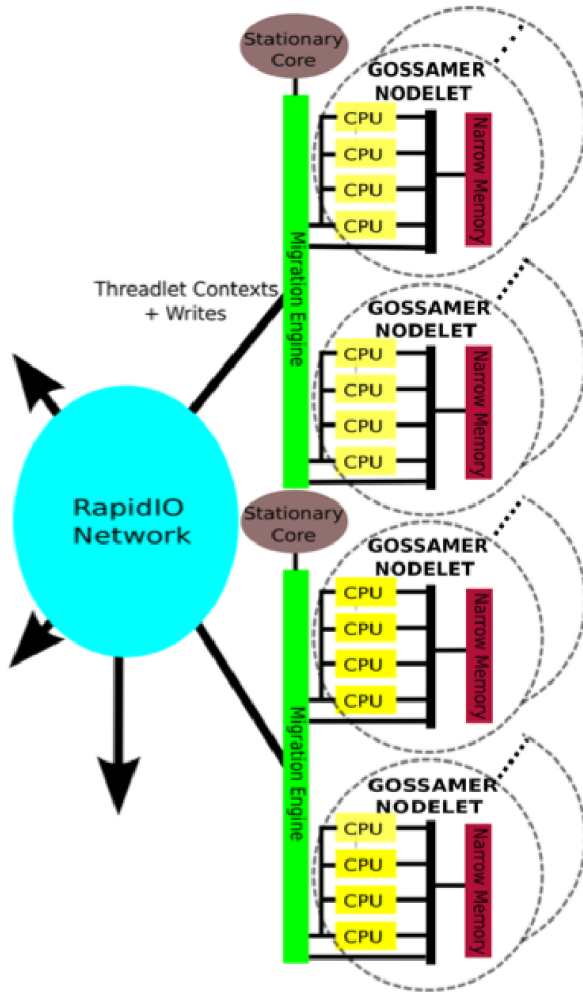
Habanero Extreme Scale Software Research Project
Georgia Institute of Technology, Atlanta, USA



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]

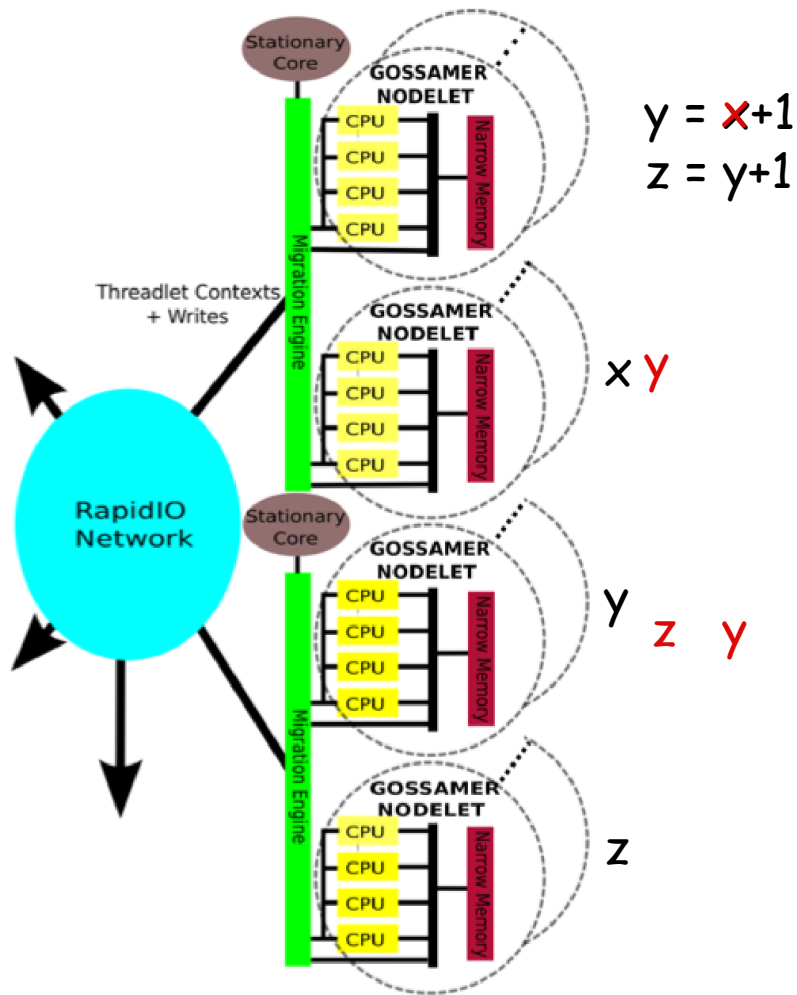


- A highly scalable near-memory multi-processor
 - 8 nodes \rightarrow 8 nodelets/node \rightarrow 4 cores/nodelet \rightarrow 64 threads/core
 - Cilk programming model for expressing parallelism

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

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

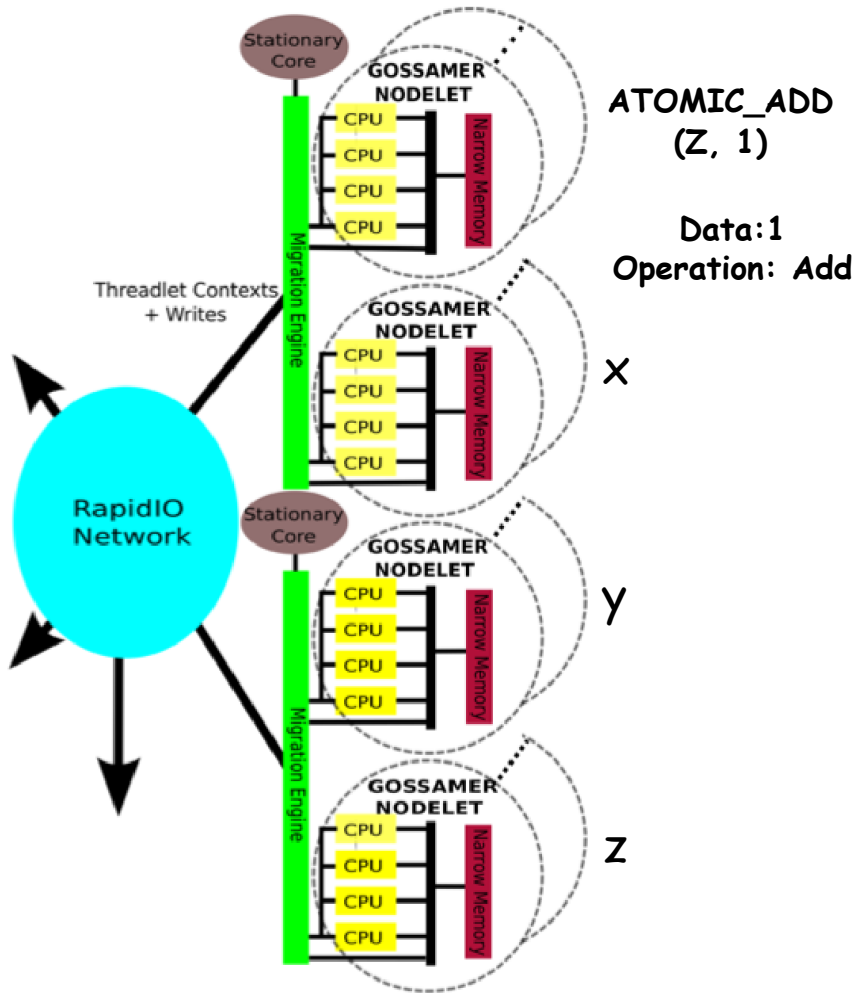
1) Key features – Thread migration



- 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]

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

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

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

Challenges with the EMU system

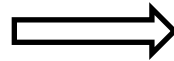
- Overheads from thread migrations, thread creation and synchronization.
- We focus on exploring compiler transformations to reduce the overheads and improve performance
 - High-level compiler transformations
 - Node fusion and Edge flipping
 - Low-level compiler transformations
 - Use of remote atomic 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: parallel-for(v ∈ vertices) {  
2:   p1[v] = ...  
3: } // Implicit barrier
```



```
4: parallel-for(v ∈ vertices) {  
5:   p2[v] = f(p1[v], ...)  
6: }
```

```
1: parallel-for(v ∈ vertices) {  
2:   p1[v] = ...  
3:   p2[v] = f(p1[v], ...)  
4: }
```

- Repeated migrations for
 - Same property across parallel loops
 - Different properties of same vertex across parallel loops
 - 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.

| | Emu system |
|--------------------|-----------------------|
| Microarch | Emul Chick |
| Clock speed | 150 MHz |
| #Nodelets | 8 |
| #Cores/Nodelet | 1 |
| #Threads/Core | 64 |
| Memorysize/Nodelet | 8 GB |
| NCDRAM speed | 1600MHz |
| Compiler toolchain | emusim.HW.x (18.08.1) |

- 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

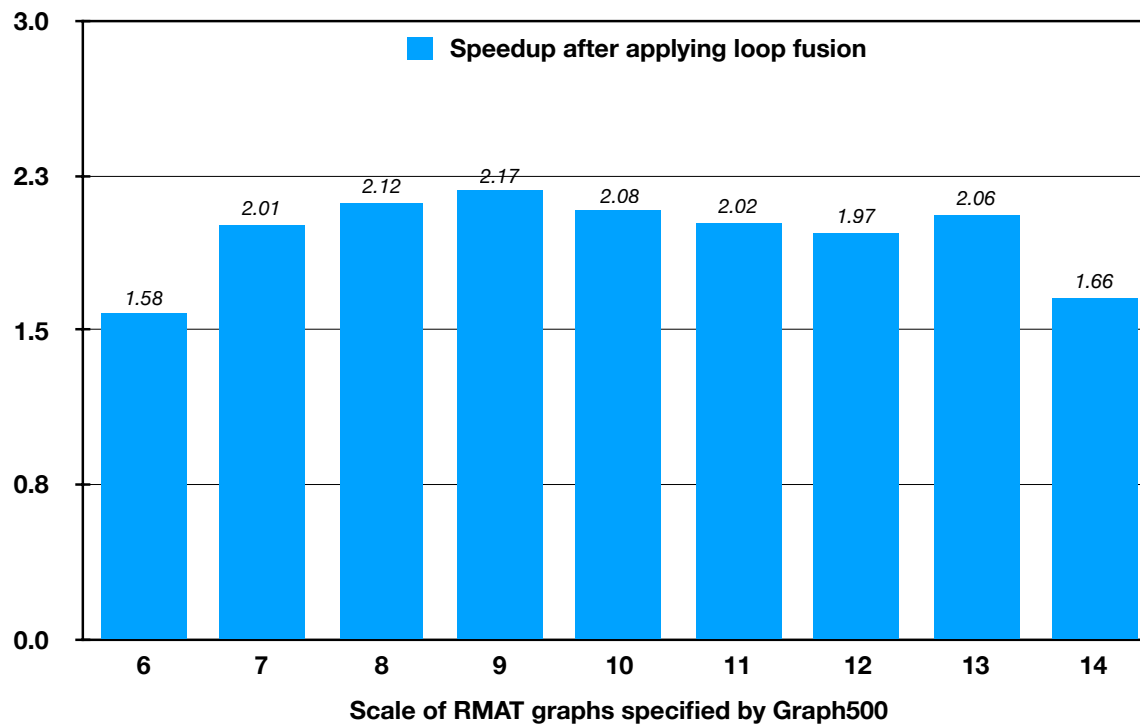
- Computes a flow from a given partition of graph to others

```
1 def CONDUCTANCE(V, id):
2   for each v ∈ V do in parallel with reduction
3     if v.partition_id == id then
4       din+ = v.degree
5     for each v ∈ V do in parallel with reduction
6       if v.partition_id != id then
7         dout+ = v.degree
8     for each v ∈ V do in parallel with reduction
9       if v.partition_id == id then
10        for each nbr ∈ v.nbrs do
11          if nbr.partition_id != id then
12            dcross+ = 1
13   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

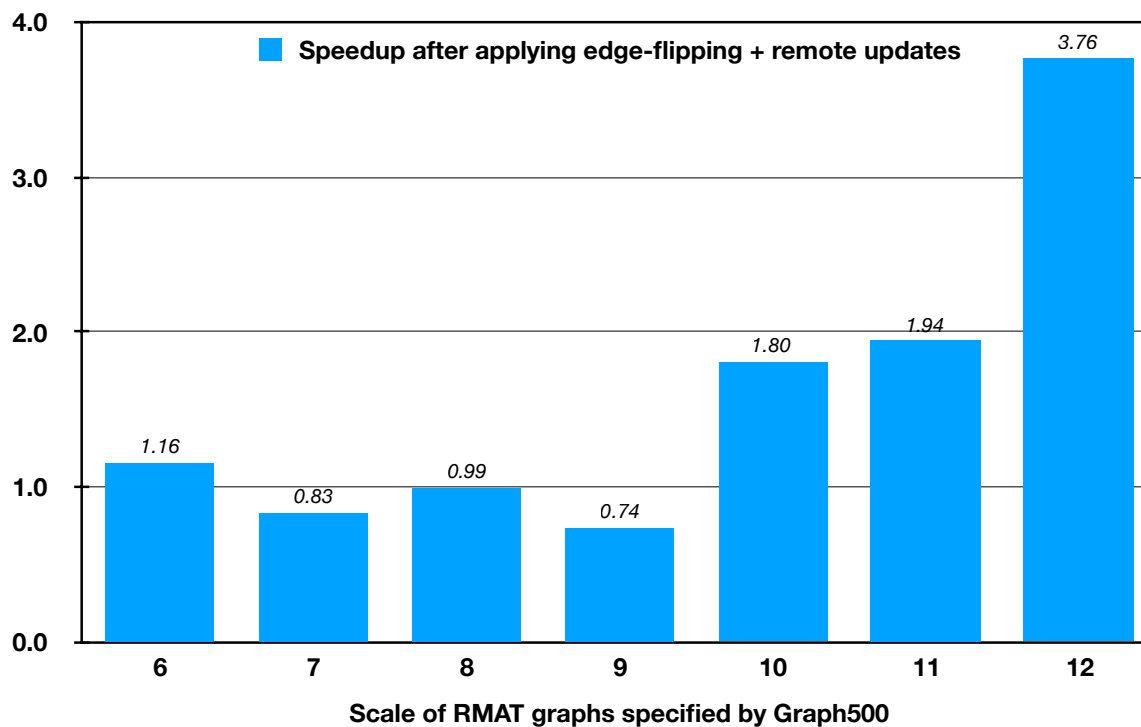
- Compute shortest paths from a single source vertex to all the other vertices in a weighted directed graph

```
5 for  $t \leftarrow 0$  to  $|V| - 1$  do
6   for each  $v \in V$  do in parallel
7     for each  $u \in \text{incoming\_neighbors}(v)$  do
8        $\text{temp} = \underline{\text{distance}(u)} + \text{weight}(u, v)$ 
9          $\triangleright$  Migration for  $\text{distance}(u)$  value
10      if  $\underline{\text{distance}(v)} > \text{temp}$  then
11         $\text{temp\_distance}(v) = \text{temp}$ 
12      end
13    endfor
```

Back and forth migration for every incoming neighbor

- 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

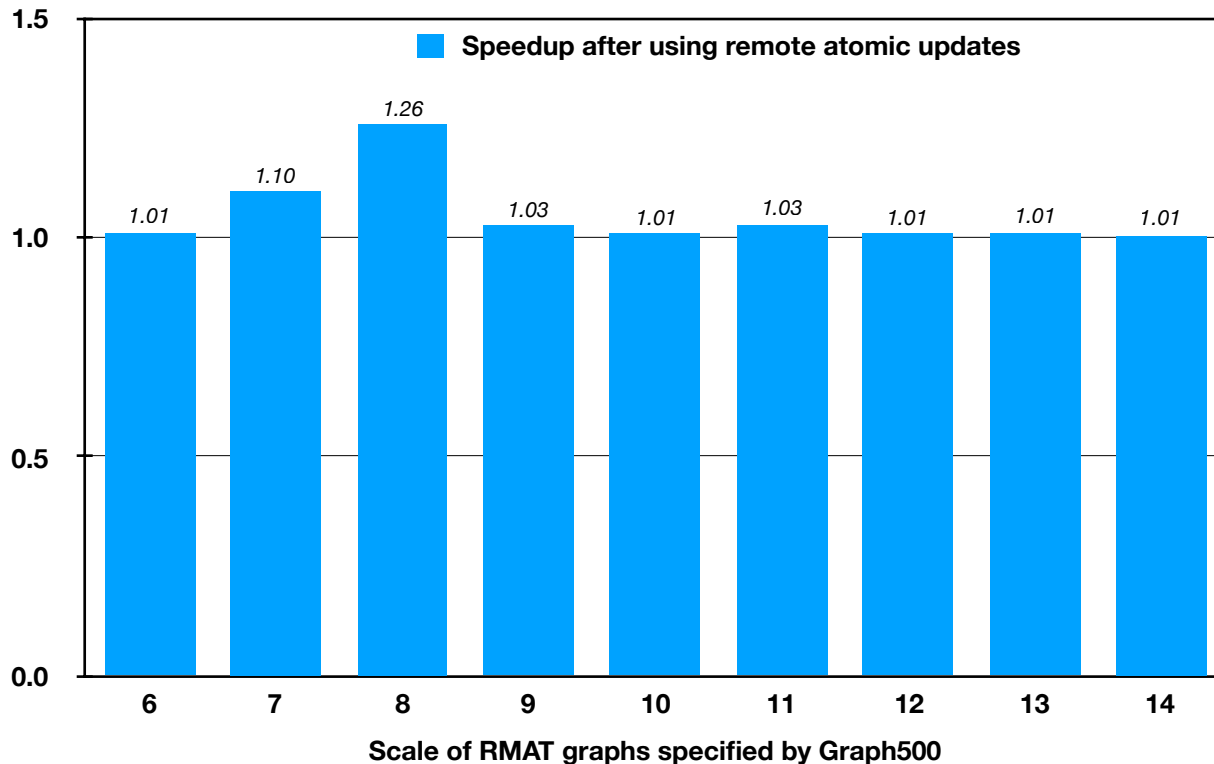
3) Triangle counting

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

```
2 for each  $v \in V$  do in parallel
3   for each  $u \in v.nbrs$  do
4     if  $nbr1 > v$  then
5       for each  $w \in v.nbrs$  do
6         if  $w > u$  then
7           if  $edge\_exists(u, w)$  then
8             tc_count ++; //Atomic
9             tc(v) ++; //Atomic
10            tc(u) ++; //Atomic
11            tc(w) ++; //Atomic
           ▷ Above regular atomics can be
             replaced by the remote updates.
```

- Regular atomic updates can be replaced with remote updates

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

| Applications | Transformations |
|--------------------------|--------------------------------|
| Conductance | Node fusion |
| Bellman-Ford's algorithm | Edge flipping + Remote updates |
| Triangle counting | Remote updates |

- 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>