Falcon: A Graph Manipulation Language for Heterogeneous Systems

Unnikrishnan C, IISc, Bangalore
Rupesh Nasre, IIT, Madras
Y N Srikant, IISc, Bangalore

April 28 , 2016

1. `Falcon` is a Domain Specific Language (DSL) for writing Graph algorithms.

1. `Falcon` is a Domain Specific Language (DSL) for writing Graph algorithms.
2. `Falcon`
   i) extends C programming language.
   ii) provides additional data types for Graph processing.
   iii) constructs for writing explicitly parallel graph algorithms.

1. `Falcon` is a Domain Specific Language (DSL) for writing Graph algorithms.
2. `Falcon`
   i) extends C programming language.
   ii) provides additional data types for Graph processing.
   iii) constructs for writing explicitly parallel graph algorithms.
3. Support for heterogeneous backends(CPU and GPU).

1. `Falcon` is a Domain Specific Language (DSL) for writing Graph algorithms.
2. `Falcon`
   i) extends C programming language.
   ii) provides additional data types for Graph processing.
   iii) constructs for writing explicitly parallel graph algorithms.
3. Support for heterogeneous backends(CPU and GPU).
4. Supports parallel execution of different algorithms on multiple devices.

1. `Falcon` is a Domain Specific Language (DSL) for writing Graph algorithms.
2. `Falcon`
   i) extends C programming language.
   ii) provides additional data types for Graph processing.
   iii) constructs for writing explicitly parallel graph algorithms.
3. Support for heterogeneous backends(CPU and GPU).
4. Supports parallel execution of different algorithms on multiple devices.
5. Supports partitioning of Graph objects and execution of a single algorithm using multiple devices. Used when graph object does not fit in a single device.
6. Supports mutation of Graph object.
7. Allows viewing Graph in different way(say collection of triangles).

Language constructs for parallelization and Synchronization in Falcon

single(t1) {stmt block1} **else** {stmt block2}	The thread that gets a lock on item t1 executes stmt block1 and other threads execute stmt block2.
single(coll) {stmt block1} **else** {stmt block2}	The thread that gets a lock on all elements in the collection executes stmt block1 and others execute stmt block2.

Table 1. single statement(Synchronization) in Falcon

Data Type	Iterator	Description
Graph	points	iterate over all points in graph
Graph	edges	iterate over all edges in graph
Graph	pptyname	iterate over all elements in new ppty.
Point	nbrs	iterate over all neighboring points
Point	outnbrs	iterate over dst point of outgoing edges (Directed Graph)
Edge	nbrs	iterate over neighbor edges
Set	item	iterate over all items in Set
Collection	item	iterate over all items in Collection

Table 2. Iterators for foreach(parallelization) statement in Falcon

parallel sections- for Multiple parallel regions on different devices.

shortest path

```
1  int <GPU> changed = 0; // Variable on GPU
2  relaxgraph(Point <GPU>p, Graph <GPU>graph) {
3      foreach (t In p.outnbrs)
4        MIN(t.dist, p.dist + graph.getWeight(p, t), changed);
5  }
6  main(int argc, char *argv[]) {
7      Graph hgraph; // graph on CPU
8      hgraph.addPointProperty(dist, int);
9      hgraph.getType() <GPU>graph; // graph on GPU
10     hgraph.read(argv[1]); // read graph on CPU
11     graph = hgraph; // copy graph to GPU
12     foreach (t In graph.points)t.dist=MAX_INT;//INFinity
13     graph.points[0].dist = 0; // source has dist 0
14     while( 1 ){
15         changed = 0;
16         foreach (t In graph.points) relaxgraph(t,graph);
17         if (changed == 0) break; //reached fix point
18     }
19     for (int i = 0; i <graph.npoints; ++i)
20           printf("i=%d dist=%d\n", i, graph.points[i].dist);
21 }
```

shortest path

```
1  int <GPU> changed = 0; // Variable on GPU
2  relaxgraph(Point <GPU>p, Graph <GPU>graph) {
3      foreach (t In p.outnbrs)
4        MIN(t.dist, p.dist + graph.getWeight(p, t), changed);
5  }
6  main(int argc, char *argv[]) {
7      Graph hgraph; // graph on CPU
8      hgraph.addPointProperty(dist, int);
9      hgraph.getType() <GPU>graph; // graph on GPU
10     hgraph.read(argv[1]); // read graph on CPU
11     graph = hgraph; // copy graph to GPU
12     foreach (t In graph.points)t.dist=MAX_INT;//INFinity
13     graph.points[0].dist = 0; // source has dist 0
14     while( 1 ){
15         changed = 0;
16         foreach (t In graph.points) relaxgraph(t,graph);
17         if (changed == 0) break; //reached fix point
18     }
19     for (int i = 0; i <graph.npoints; ++i)
20           printf("i=%d dist=%d\n", i, graph.points[i].dist);
21 }
```

shortest path

```
1  int <GPU> changed = 0; // Variable on GPU
2  relaxgraph(Point <GPU>p, Graph <GPU>graph) {
3      foreach (t In p.outnbrs)
4        MIN(t.dist, p.dist + graph.getWeight(p, t), changed);
5  }
6  main(int argc, char *argv[]) {
7      Graph hgraph; // graph on CPU
8      hgraph.addPointProperty(dist, int);
9      hgraph.getType() <GPU>graph; // graph on GPU
10     hgraph.read(argv[1]); // read graph on CPU
11     graph = hgraph; // copy graph to GPU
12     foreach (t In graph.points)t.dist=MAX_INT;//INFinity
13     graph.points[0].dist = 0; // source has dist 0
14     while( 1 ){
15         changed = 0;
16         foreach (t In graph.points) relaxgraph(t,graph);
17         if (changed == 0) break; //reached fix point
18     }
19     for (int i = 0; i <graph.npoints; ++i)
20           printf("i=%d dist=%d\n", i, graph.points[i].dist);
21 }
```

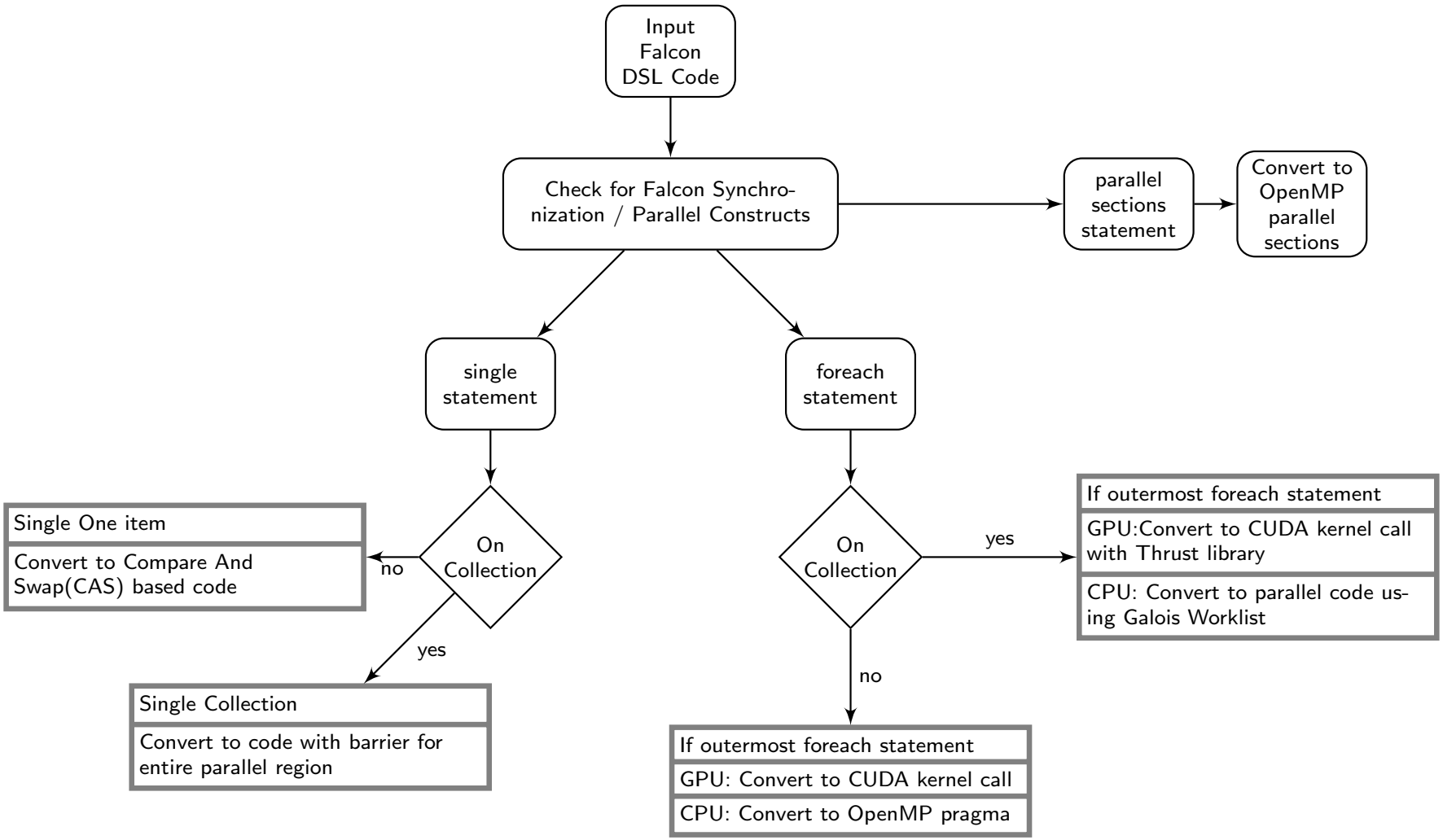
shortest path

```
1  int <GPU> changed = 0; // Variable on GPU
2  relaxgraph(Point <GPU>p, Graph <GPU>graph) {
3      foreach (t In p.outnbrs)
4        MIN(t.dist, p.dist + graph.getWeight(p, t), changed);
5  }
6  main(int argc, char *argv[]) {
7      Graph hgraph; // graph on CPU
8      hgraph.addPointProperty(dist, int);
9      hgraph.getType() <GPU>graph; // graph on GPU
10     hgraph.read(argv[1]); // read graph on CPU
11     graph = hgraph; // copy graph to GPU
12     foreach (t In graph.points)t.dist=MAX_INT;//INFinity
13     graph.points[0].dist = 0; // source has dist 0
14     while( 1 ){
15         changed = 0;
16         foreach (t In graph.points) relaxgraph(t,graph);
17         if (changed == 0) break; //reached fix point
18     }
19     for (int i = 0; i <graph.npoints; ++i)
20           printf("i=%d dist=%d\n", i, graph.points[i].dist);
21 }
```

shortest path

```
1  int <GPU> changed = 0; // Variable on GPU
2  relaxgraph(Point <GPU>p, Graph <GPU>graph) {
3      foreach (t In p.outnbrs)
4        MIN(t.dist, p.dist + graph.getWeight(p, t), changed);
5  }
6  main(int argc, char *argv[]) {
7      Graph hgraph; // graph on CPU
8      hgraph.addPointProperty(dist, int);
9      hgraph.getType() <GPU>graph; // graph on GPU
10     hgraph.read(argv[1]) // read graph on CPU
11     graph = hgraph; // copy graph to GPU
12     foreach (t In graph.points)t.dist=MAX_INT;//INFinity
13     graph.points[0].dist = 0; // source has dist 0
14     while( 1 ){
15         changed = 0;
16         foreach (t In graph.points) relaxgraph(t,graph);
17         if (changed == 0) break; //reached fix point
18     }
19     for (int i = 0; i <graph.npoints; ++i)
20           printf("i=%d dist=%d\n", i, graph.points[i].dist);
21 }
```

Falcon Compiler Code Genaration (Synchronization and parallelization constructs)



1. Using `Falcon` compiler we wrote algorithms like BFS, SSSP and Boruvka's-MST.

1. Using `Falcon` compiler we wrote algorithms like BFS, SSSP and Boruvka's-MST.
2. We wrote dynamic algorithms like Survey Propagation(SP), Delaunay Mesh refinement(DMR) and Dynamic-SSSP in `Falcon`.

1. Using `Falcon` compiler we wrote algorithms like BFS, SSSP and Boruvka's-MST.
2. We wrote dynamic algorithms like Survey Propagation(SP), Delaunay Mesh refinement(DMR) and Dynamic-SSSP in `Falcon`.
3. Performance of Falcon DSL codes were compared with
   i) **LonestarGPU**-(ISS group at the University of Texas at Austin)
   i) **Galois**- (ISS group at the University of Texas at Austin)
   ii)**Green-Marl**- DSL (PP Laboratory, Stanford University)
   iV)**Totem**(NetSysLab, University of British Columbia)

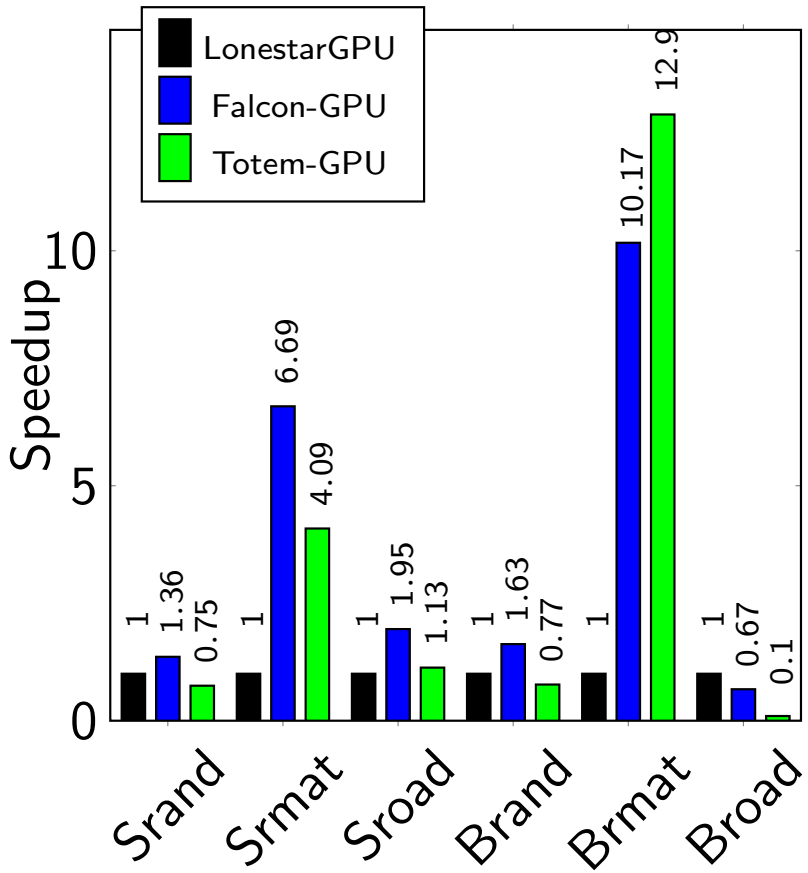
1. Using `Falcon` compiler we wrote algorithms like BFS, SSSP and Boruvka's-MST.
2. We wrote dynamic algorithms like Survey Propagation(SP), Delaunay Mesh refinement(DMR) and Dynamic-SSSP in `Falcon`.
3. Performance of Falcon DSL codes were compared with
   i) **LonestarGPU**-(ISS group at the University of Texas at Austin)
   i) **Galois**- (ISS group at the University of Texas at Austin)
   ii)**Green-Marl**- DSL (PP Laboratory, Stanford University)
   iV)**Totem**(NetSysLab, University of British Columbia)
4. **Totem**- for comparing Performance on CPU, GPU and heterogeneous execution.
5. **Galois** and **Green-Marl** for comparing Performance on CPU.
6. **LonestarGPU** for comparing Performance on GPU.

1. Using `Falcon` compiler we wrote algorithms like BFS, SSSP and Boruvka's-MST.
2. We wrote dynamic algorithms like Survey Propagation(SP), Delaunay Mesh refinement(DMR) and Dynamic-SSSP in `Falcon`.
3. Performance of Falcon DSL codes were compared with
   i) **LonestarGPU**-(ISS group at the University of Texas at Austin)
   i) **Galois**- (ISS group at the University of Texas at Austin)
   ii)**Green-Marl**- DSL (PP Laboratory, Stanford University)
   iV)**Totem**(NetSysLab, University of British Columbia)
4. **Totem**- for comparing Performance on CPU, GPU and heterogeneous execution.
5. **Galois** and **Green-Marl** for comparing Performance on CPU.
6. **LonestarGPU** for comparing Performance on GPU.
7. We were able to get performance close to and some times better than above systems. Tested on a machine with 12-core CPU and 4-GPUs.

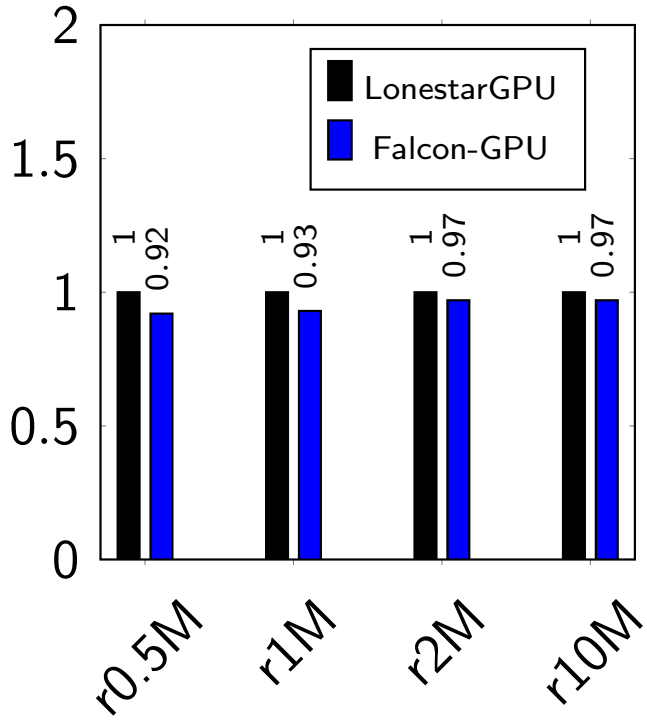
1. Using `Falcon` compiler we wrote algorithms like BFS, SSSP and Boruvka's-MST.
2. We wrote dynamic algorithms like Survey Propagation(SP), Delaunay Mesh refinement(DMR) and Dynamic-SSSP in `Falcon`.
3. Performance of Falcon DSL codes were compared with
   i) **LonestarGPU**-(ISS group at the University of Texas at Austin)
   i) **Galois**- (ISS group at the University of Texas at Austin)
   ii)**Green-Marl**- DSL (PP Laboratory, Stanford University)
   iV)**Totem**(NetSysLab, University of British Columbia)
4. **Totem**- for comparing Performance on CPU, GPU and heterogeneous execution.
5. **Galois** and **Green-Marl** for comparing Performance on CPU.
6. **LonestarGPU** for comparing Performance on GPU.
7. We were able to get performance close to and some times better than above systems. Tested on a machine with 12-core CPU and 4-GPUs.

1. Using `Falcon` compiler we wrote algorithms like BFS, SSSP and Boruvka's-MST.
2. We wrote dynamic algorithms like Survey Propagation(SP), Delaunay Mesh refinement(DMR) and Dynamic-SSSP in `Falcon`.
3. Performance of Falcon DSL codes were compared with
   i) **LonestarGPU**-(ISS group at the University of Texas at Austin)
   i) **Galois**- (ISS group at the University of Texas at Austin)
   ii)**Green-Marl**- DSL (PP Laboratory, Stanford University)
   iV)**Totem**(NetSysLab, University of British Columbia)
4. **Totem**- for comparing Performance on CPU, GPU and heterogeneous execution.
5. **Galois** and **Green-Marl** for comparing Performance on CPU.
6. **LonestarGPU** for comparing Performance on GPU.
7. We were able to get performance close to and some times better than above systems. Tested on a machine with 12-core CPU and 4-GPUs.
8. Publication- http://dl.acm.org/citation.cfm?id=2842618(ACM TACO,2015)

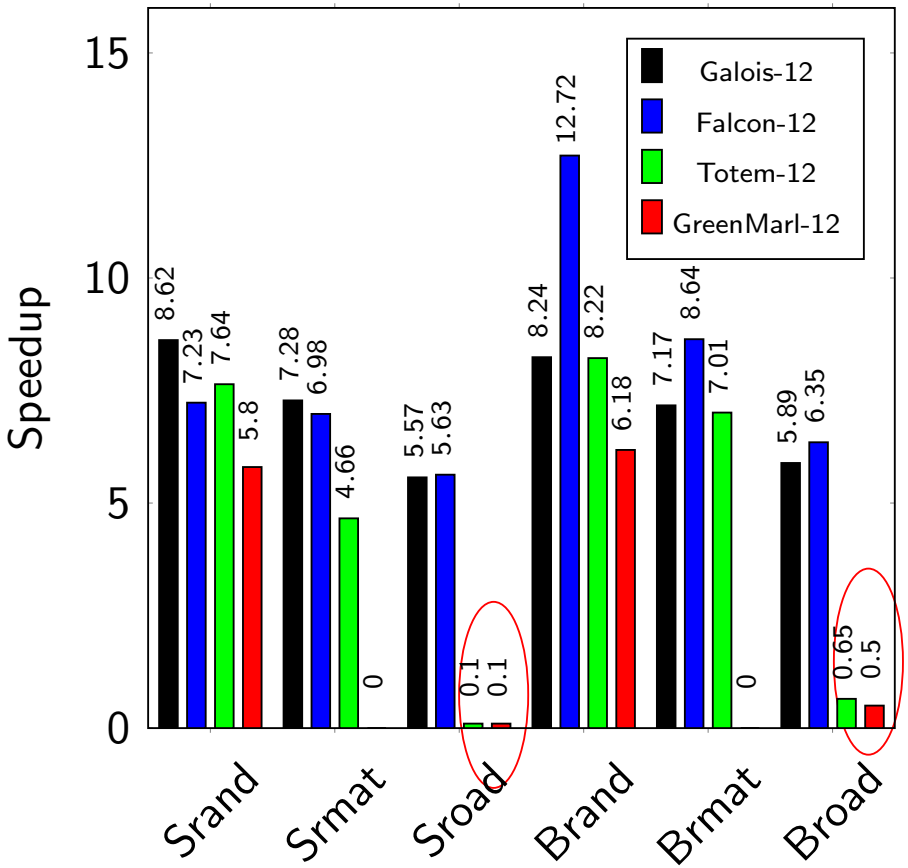
(Points,Edges) rand(32M,128M),rmat(20M,200M)
road(23M,58M)



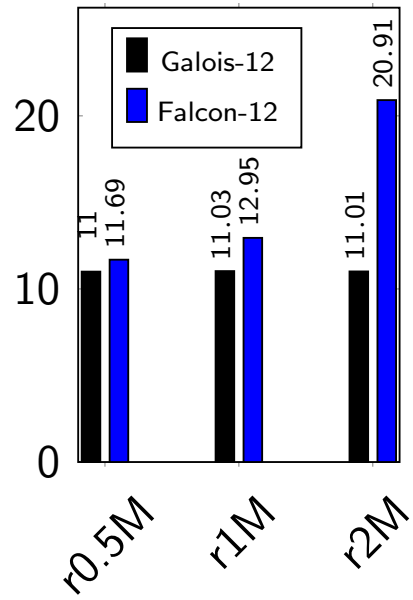
(a) Speedup of SSSP & BFS on GPU



(b) DMR speedup over LonestarGPU

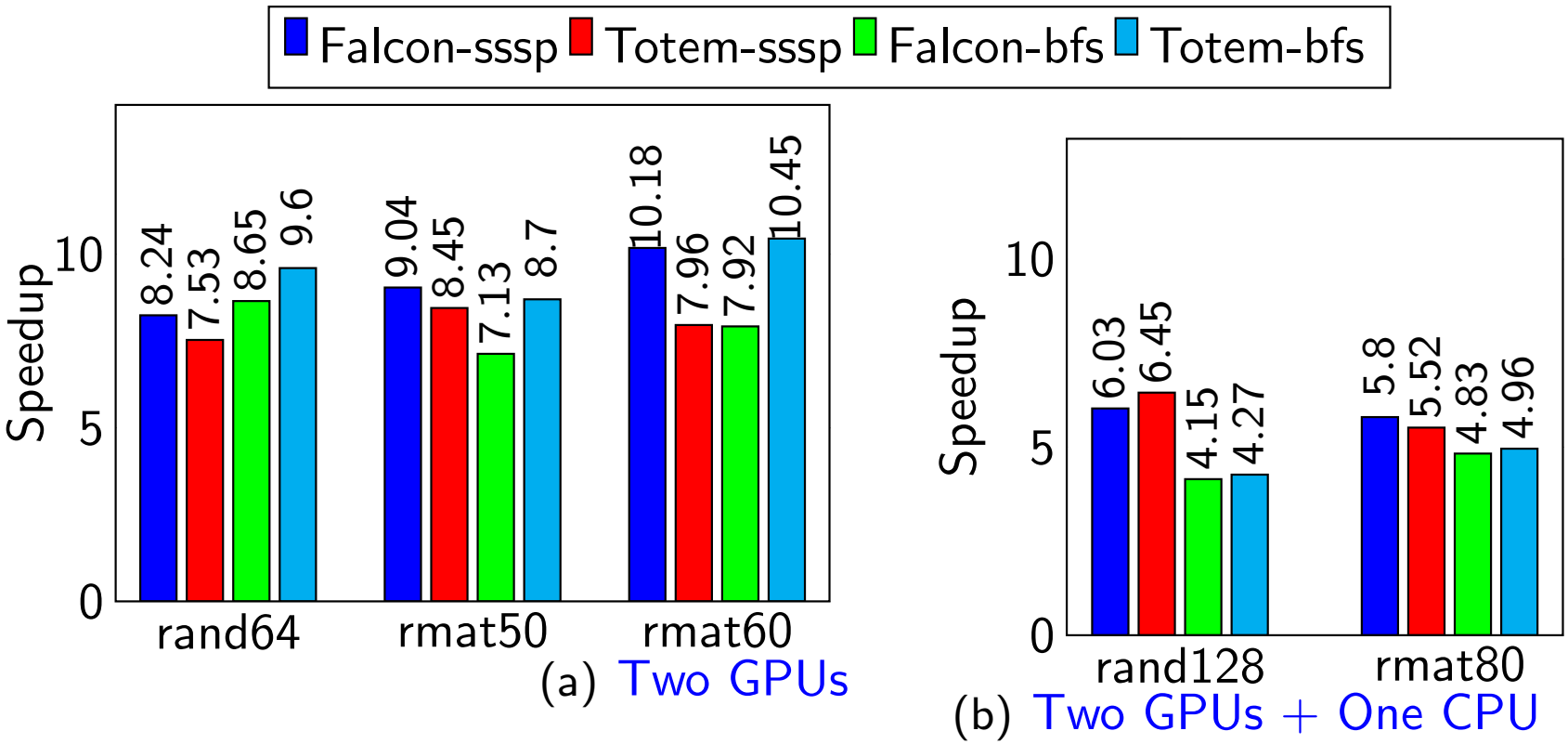


(a) SSSP & BFS speedup over Galois Single



(b) DMR speedup over Galois single

Speedup over single threaded CPU code



Heterogeneous Execution-SSSP and BFS speedup

1. We have a introduced a new DSL for Graph algorithms which targets heterogeneous architectures.

1. We have a introduced a new DSL for Graph algorithms which targets heterogeneous architectures.
2. Programmer does not have to worry on target architecture , thread & memory management.

1. We have a introduced a new DSL for Graph algorithms which targets heterogeneous architectures.
2. Programmer does not have to worry on target architecture , thread & memory management.
3. Future Works in mind
   i) to extend it for CPU clusters.
   ii) Making DSL more simple(say removing <GPU>tag).
   iii) Support for non-Nvidia GPUs by providing OpenCL backend.
   iv) adding optimizations on Compiler.

1. We have a introduced a new DSL for Graph algorithms which targets heterogeneous architectures.
2. Programmer does not have to worry on target architecture , thread & memory management.
3. Future Works in mind
   i) to extend it for CPU clusters.
   ii) Making DSL more simple(say removing <GPU>tag).
   iii) Support for non-Nvidia GPUs by providing OpenCL backend.
   iv) adding optimizations on Compiler.
4. for queries email me on unni_c@csa.iisc.ernet.in.

Questions??