Falcon: A Graph Manipulation Language for Heterogeneous Systems

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

April 28, 2016
Falcon is a Domain Specific Language (DSL) for writing graph algorithms.
Falcon is a Domain Specific Language (DSL) for writing Graph algorithms.

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

i) extends C programming language.

ii) provides additional data types for Graph processing.

iii) constructs for writing explicitly parallel graph algorithms.

Support for heterogeneous backends (CPU and GPU).

Supports parallel execution of different algorithms on multiple devices.

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.

Supports mutation of Graph object.

Allows viewing Graph in different way (say collection of triangles).
Language constructs for parallelization and Synchronization in Falcon

<table>
<thead>
<tr>
<th>single(t1)</th>
<th>stmt block1</th>
<th>else</th>
<th>stmt block2</th>
<th>The thread that gets a lock on item t1 executes stmt block1 and other threads execute stmt block2.</th>
</tr>
</thead>
<tbody>
<tr>
<td>single(coll)</td>
<td>stmt block1</td>
<td>else</td>
<td>stmt block2</td>
<td>The thread that gets a lock on all elements in the collection executes stmt block1 and others execute stmt block2.</td>
</tr>
</tbody>
</table>

Table 1. single statement (Synchronization) in Falcon

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Iterator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graph</td>
<td>points</td>
<td>iterate over all points in graph</td>
</tr>
<tr>
<td>Graph</td>
<td>edges</td>
<td>iterate over all edges in graph</td>
</tr>
<tr>
<td>Graph</td>
<td>pptynam</td>
<td>iterate over all elements in new ppty.</td>
</tr>
<tr>
<td>Point</td>
<td>nbrs</td>
<td>iterate over all neighboring points</td>
</tr>
<tr>
<td>Point</td>
<td>outnb</td>
<td>iterate over dst point of outgoing edges (Directed Graph)</td>
</tr>
<tr>
<td>Edge</td>
<td>nbrs</td>
<td>iterate over neighbor edges</td>
</tr>
<tr>
<td>Set</td>
<td>item</td>
<td>iterate over all items in Set</td>
</tr>
<tr>
<td>Collection</td>
<td>item</td>
<td>iterate over all items in Collection</td>
</tr>
</tbody>
</table>

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

parallel sections - for Multiple parallel regions on different devices.
shortest path

```c
int <GPU> changed = 0; // Variable on GPU
relaxgraph(Point <GPU>p, Graph <GPU>graph) {
    foreach (t In p.outnbrs)
        MIN(t.dist, p.dist + graph.getWeight(p, t), changed);
}
main(int argc, char *argv[]) {
    Graph hgraph; // graph on CPU
    hgraph.addPointProperty(dist, int);
    hgraph.getType() <GPU>graph; // graph on GPU
    hgraph.read(argv[1]); // read graph on CPU
    graph = hgraph; // copy graph to GPU
    foreach (t In graph.points)t.dist=MAX_INT; //INFINITY
    graph.points[0].dist = 0; // source has dist 0
    while( 1 ){
        changed = 0;
        foreach (t In graph.points) relaxgraph(t,graph);
        if (changed == 0) break; //reached fix point
    }
    for (int i = 0; i <graph.npoints; ++i)
        printf("i=%d dist=%d\n", i, graph.points[i].dist);
}  

Unnikrishnan C, Rupesh Nasre, Y N Srikant
Falcon: A Graph Manipulation Language for Heterogeneous Systems 3 / 10
shortest path

```c
int <GPU> changed = 0; // Variable on GPU
relaxgraph(Point <GPU>p, Graph <GPU>graph) {
    foreach (t In p.outnbrs)
        MIN(t.dist, p.dist + graph.getWeight(p, t), changed);
}
main(int argc, char *argv[]) {
    Graph hgraph; // graph on CPU
    hgraph.addPointProperty(dist, int);
    hgraph.getType() <GPU>graph; // graph on GPU
    hgraph.read(argv[1]); // read graph on CPU
    graph = hgraph; // copy graph to GPU
    foreach (t In graph.points) t.dist = MAX_INT; // INFINITY
    graph.points[0].dist = 0; // source has dist 0
    while(1) {
        changed = 0;
        foreach (t In graph.points) relaxgraph(t, graph);
        if (changed == 0) break; // reached fix point
    }
    for (int i = 0; i < graph.npoints; ++i)
        printf("i=%d dist=%d\n", i, graph.points[i].dist);
}
```
shortest path

int <GPU> changed = 0; // Variable on GPU
relaxgraph(Point <GPU>p, Graph <GPU> graph) {
    foreach (t In p.outnbrs)
        MIN(t.dist, p.dist + graph.getWeight(p, t), changed);
}
main(int argc, char *argv[]) {
    Graph hgraph; // graph on CPU
    hgraph.addPointProperty(dist, int);
    hgraph.getType() <GPU> graph; // graph on GPU
    hgraph.read(argv[1]); // read graph on CPU
    graph = hgraph; // copy graph to GPU
    foreach (t In graph.points)t.dist = MAX_INT; // INFINITY
    graph.points[0].dist = 0; // source has dist 0
    while( 1 ){
        changed = 0;
        foreach (t In graph.points) relaxgraph(t, graph);
        if (changed == 0) break; // reached fix point
    }
    for (int i = 0; i < graph.npoints; ++i)
        printf("i=%d dist=%d\n", i, graph.points[i].dist);
}
shortest path

```c
int <GPU> changed = 0; // Variable on GPU
relaxgraph(Point <GPU>p, Graph <GPU>graph) {
    foreach (t In p.outnbrs)
        MIN(t.dist, p.dist + graph.getWeight(p, t), changed);
}
main(int argc, char *argv[])
{
    Graph hgraph; // graph on CPU
    hgraph.addPointProperty(dist, int);
    hgraph.getType() <GPU>graph; // graph on GPU
    hgraph.read(argv[1]); // read graph on CPU
    graph = hgraph; // copy graph to GPU
    foreach (t In graph.points)t.dist=MAX_INT;//INFINITY
    graph.points[0].dist = 0; // source has dist 0
    while( 1 ){
        changed = 0;
        foreach (t In graph.points) relaxgraph(t,graph);
        if (changed == 0) break; //reached fix point
    }
    for (int i = 0; i <graph.npoints; ++i)
        printf("i=%d dist=%d\n", i, graph.points[i].dist);
}
shortest path

```c
int <GPU> changed = 0; // Variable on GPU
relaxgraph(Point <GPU>p, Graph <GPU>graph) {
    foreach (t In p.outnbrs)
        MIN(t.dist, p.dist + graph.getWeight(p, t), changed);
}
main(int argc, char *argv[]) {
    Graph hgraph; // graph on CPU
    hgraph.addPointProperty(dist, int);
    hgraph.getType() <GPU>graph; // graph on GPU
    hgraph.read(argv[1]) // read graph on CPU
    graph = hgraph; // copy graph to GPU
    foreach (t In graph.points)t.dist=MAX_INT;/INFINITY
    graph.points[0].dist = 0; // source has dist 0
    while( 1 ){
        changed = 0;
        foreach (t In graph.points) relaxgraph(t,graph);
        if (changed == 0) break; //reached fix point
    }
    for (int i = 0; i <graph.npoints; ++i)
        printf("i=%d dist=%d\n", i, graph.points[i].dist);
}
```
Falcon Compiler Code Generation (Synchronization and parallelization constructs)

- **Input**: Falcon DSL Code
- **Check for Falcon Synchronization / Parallel Constructs**
  - **Single Statement**
    - Single One item: Convert to Compare And Swap (CAS) based code
    - Single Collection: Convert to code with barrier for entire parallel region
  - **Foreach Statement**
    - **Yes**: If outermost foreach statement
      - GPU: Convert to CUDA kernel call with Thrust library
      - CPU: Convert to parallel code using Galois Worklist
    - **No**: Convert to OpenMP parallel sections
Using Falcon compiler we wrote algorithms like BFS, SSSP and Boruvka’s-MST.
Using Falcon compiler we wrote algorithms like BFS, SSSP and Boruvka’s-MST.

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

We wrote dynamic algorithms like Survey Propagation (SP), Delaunay Mesh refinement (DMR) and Dynamic-SSSP in Falcon.

Performance of Falcon DSL codes were compared with

i) LonestarGPU-(ISS group at the University of Texas at Austin)

ii) 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)

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.

Publication-

http://dl.acm.org/citation.cfm?id=2842618 (ACM TACO, 2015)
Using Falcon compiler we wrote algorithms like BFS, SSSP and Boruvka’s-MST.

We wrote dynamic algorithms like Survey Propagation (SP), Delaunay Mesh refinement (DMR) and Dynamic-SSSP in Falcon.

Performance of Falcon DSL codes were compared with
i) LonestarGPU-(ISS group at the University of Texas at Austin)
ii) 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)

Totem- for comparing Performance on CPU, GPU and heterogeneous execution.

Galois and Green-Marl for comparing Performance on CPU.

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

We wrote dynamic algorithms like Survey Propagation (SP), Delaunay Mesh refinement (DMR) and Dynamic-SSSP in Falcon.

Performance of Falcon DSL codes were compared with
i) LonestarGPU-(ISS group at the University of Texas at Austin)
ii) 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)

Totem- for comparing Performance on CPU, GPU and heterogeneous execution.

Galois and Green-Marl for comparing Performance on CPU.

LonestarGPU for comparing Performance on GPU.

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

We wrote dynamic algorithms like Survey Propagation(SP), Delaunay Mesh refinement(DMR) and Dynamic-SSSP in Falcon.

Performance of Falcon DSL codes were compared with
i) LonestarGPU- (ISS group at the University of Texas at Austin)
ii) Galois- (ISS group at the University of Texas at Austin)
iii) Green-Marl- DSL (PP Laboratory, Stanford University)
iv) Totem( NetSysLab, University of British Columbia)

Totem- for comparing Performance on CPU, GPU and heterogeneous execution.

Galois and Green-Marl for comparing Performance on CPU.
LonestarGPU for comparing Performance on GPU.

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.

(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

(a) Two GPUs

(b) Two GPUs + One CPU

Heterogeneous Execution-SSSP and BFS speedup
We have introduced a new DSL for Graph algorithms which targets heterogeneous architectures.
We have introduced a new DSL for Graph algorithms which targets heterogeneous architectures.

Programmer does not have to worry on target architecture, thread & memory management.
1. We have 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 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??