Falcon: A Graph Manipulation Language for Heterogeneous Systems

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

January 20 , 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.

After Last session of HiPEAC, you want to reach home through shortest path??!!

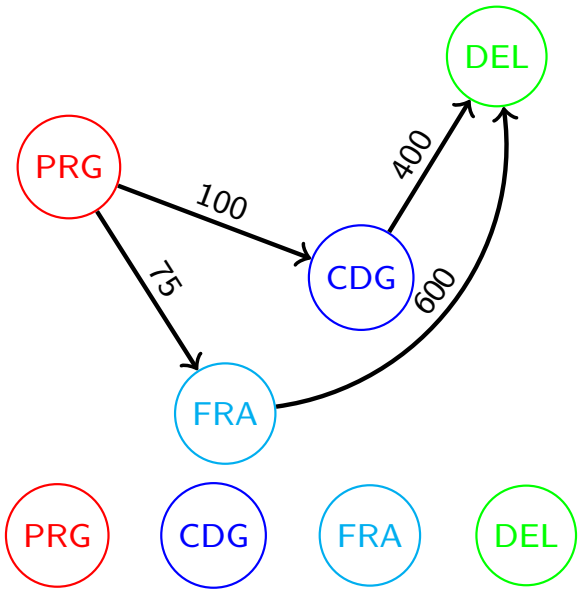
```
1  int <GPU> changed = 0; // Variable on GPU
2  relaxgraph(Point <GPU>p, Graph <GPU>graph) {
3      foreach (t In p.outnbrs)
4          MIN(t.dist, p.dist + graph.getweight(p, t), changed);
5  }
6  main(int argc, char *argv[]) {
7      Graph hgraph; // graph on CPU
8      hgraph.addPointProperty(dist, 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 }
```

A SIMPLE EXAMPLE

After Last session of HiPEAC, you want to reach home through 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 }
```

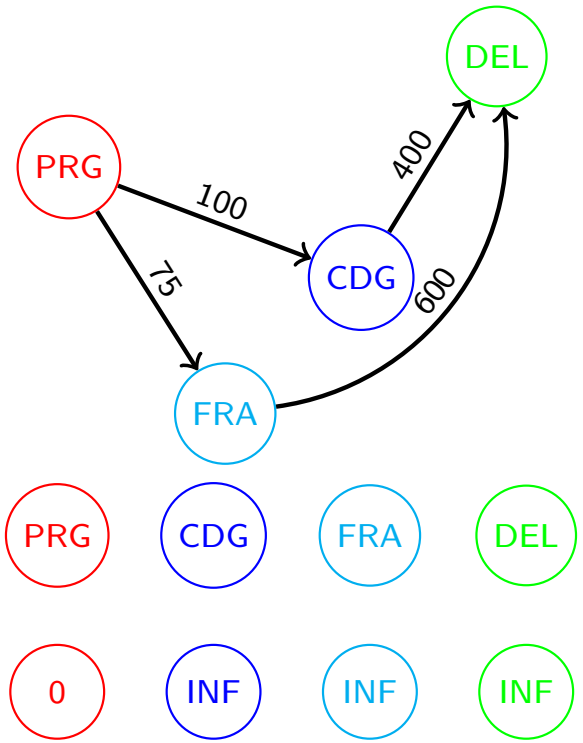
A SIMPLE EXAMPLE



After Last session of HiPEAC, you want to reach home through 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 }
```

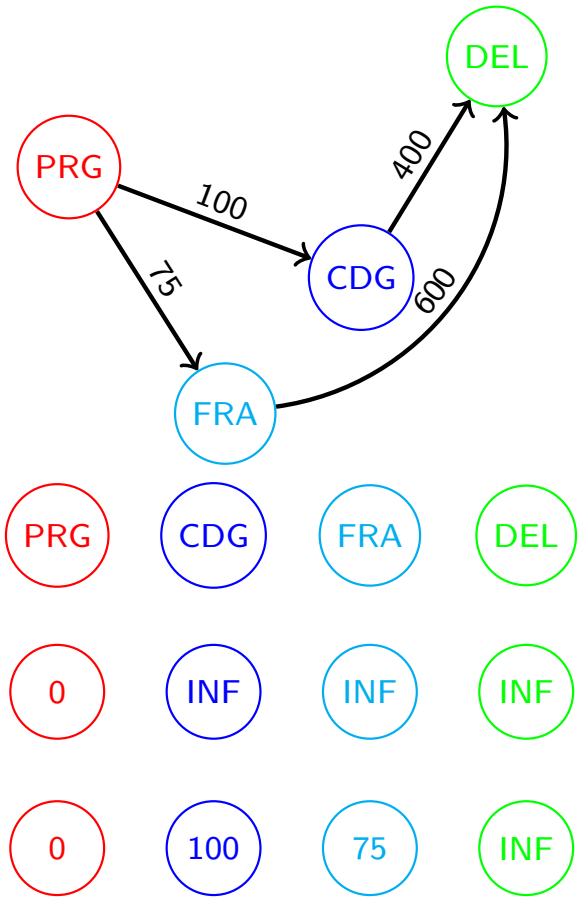
A SIMPLE EXAMPLE



After Last session of HiPEAC, you want to reach home through 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 }
```

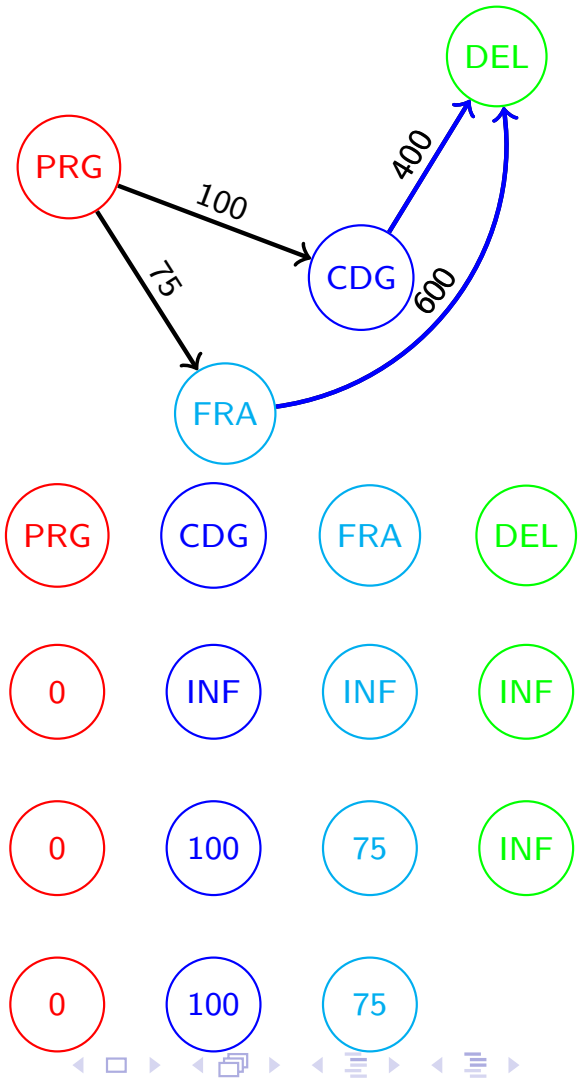
A SIMPLE EXAMPLE



After Last session of HiPEAC, you want to reach home through 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 }
```

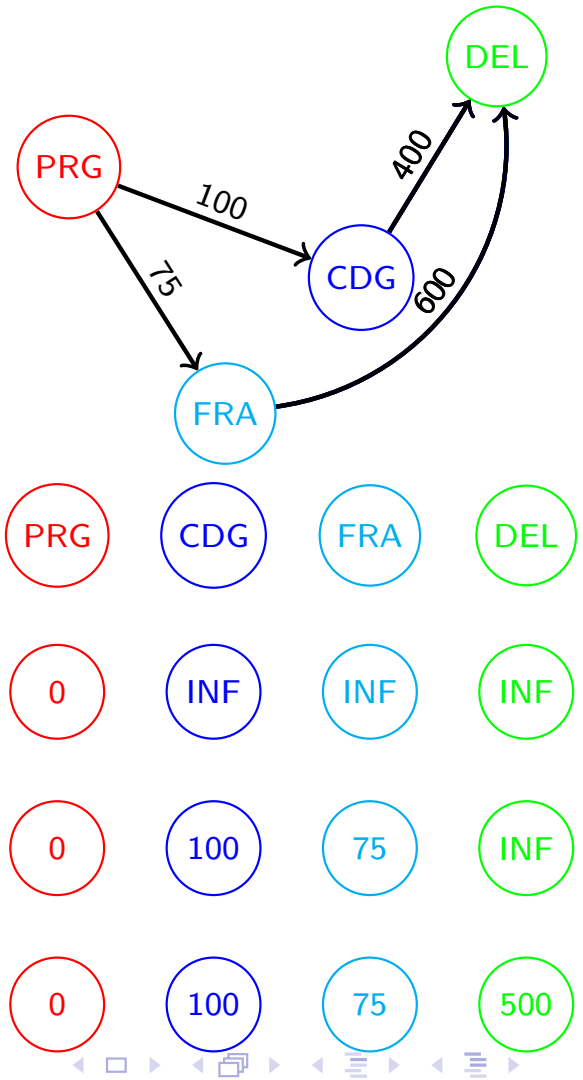
A SIMPLE EXAMPLE



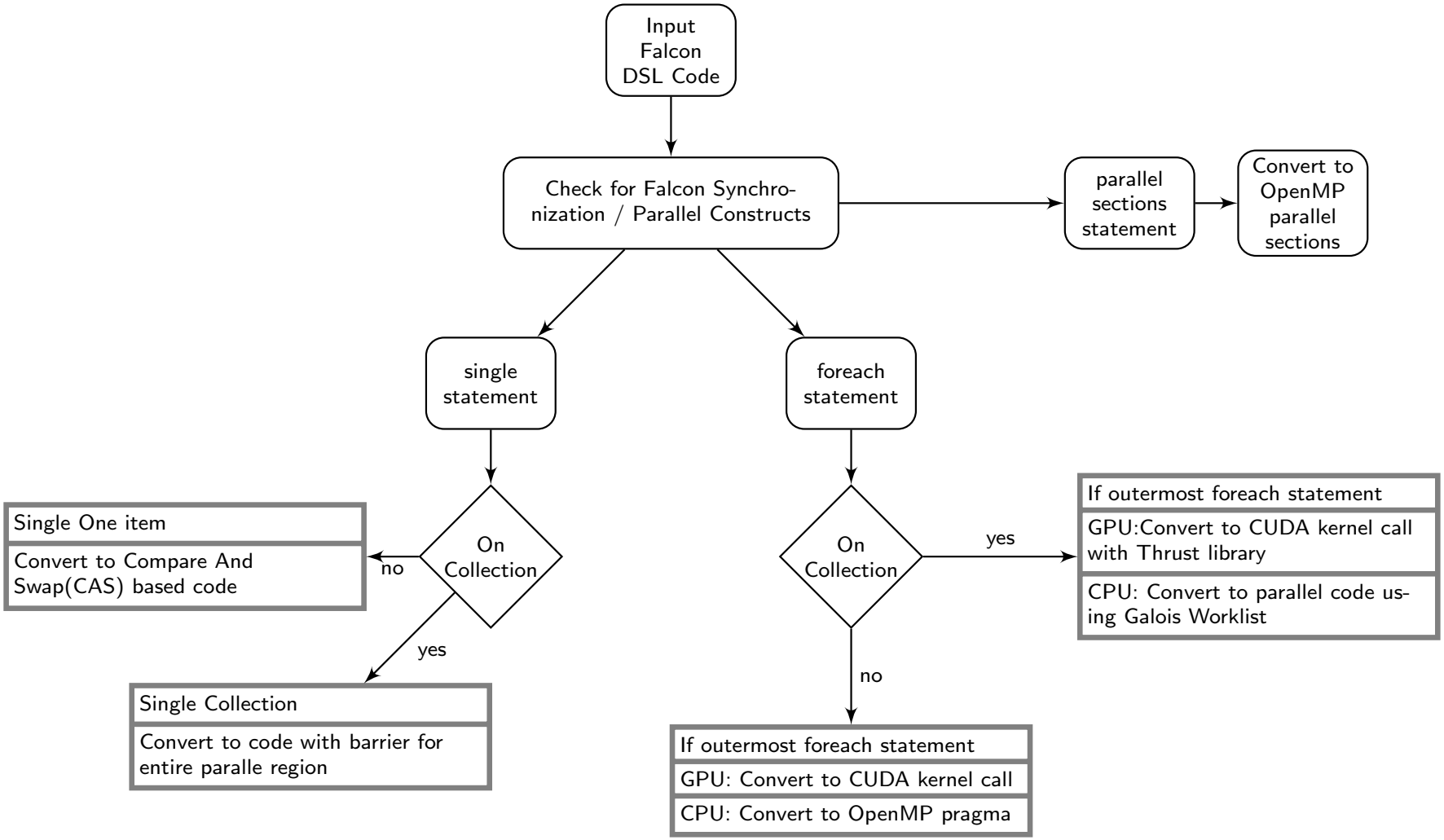
After Last session of HiPEAC, you want to reach home through shortest path??!!

```
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 }
```

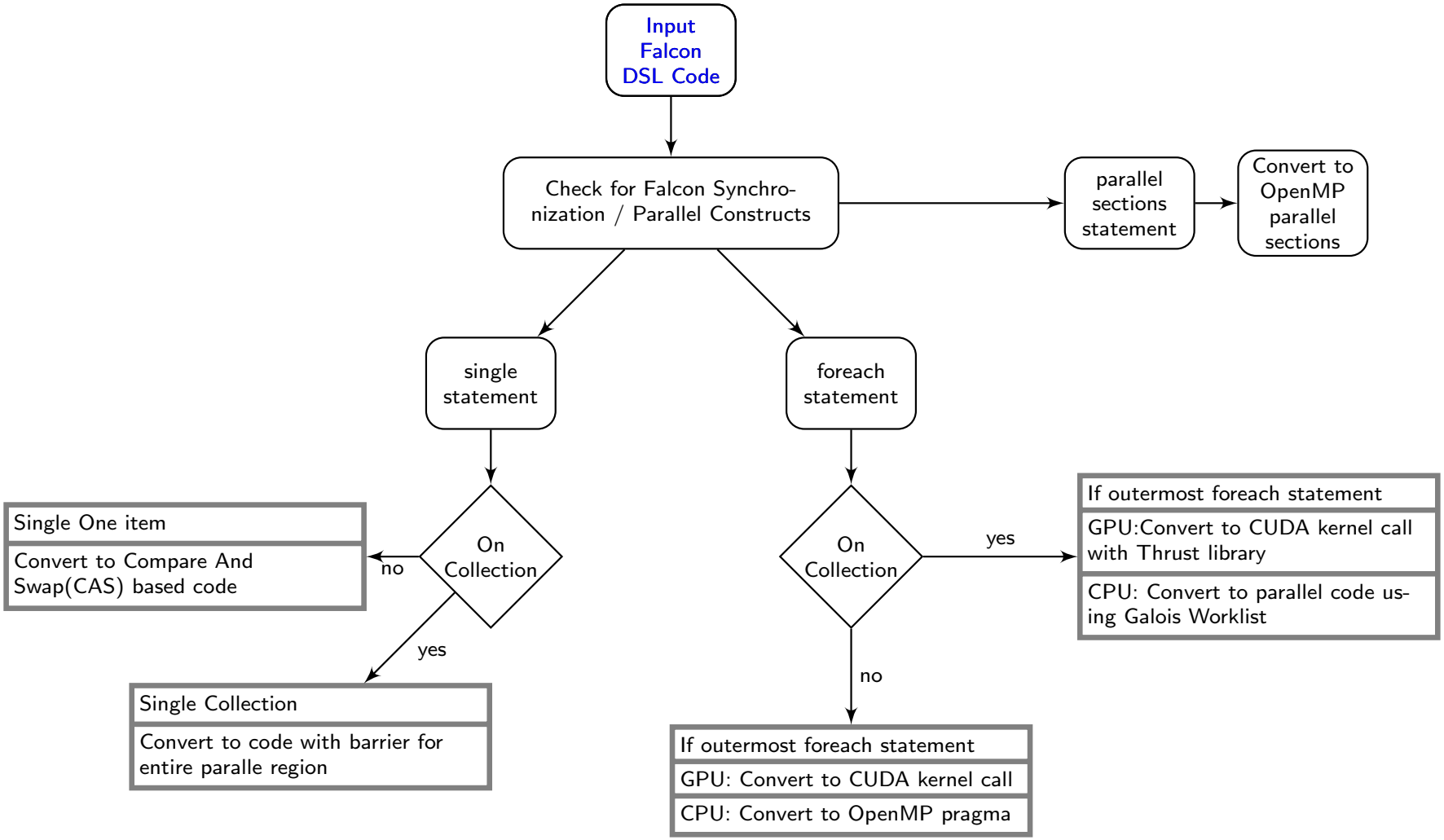
A SIMPLE EXAMPLE



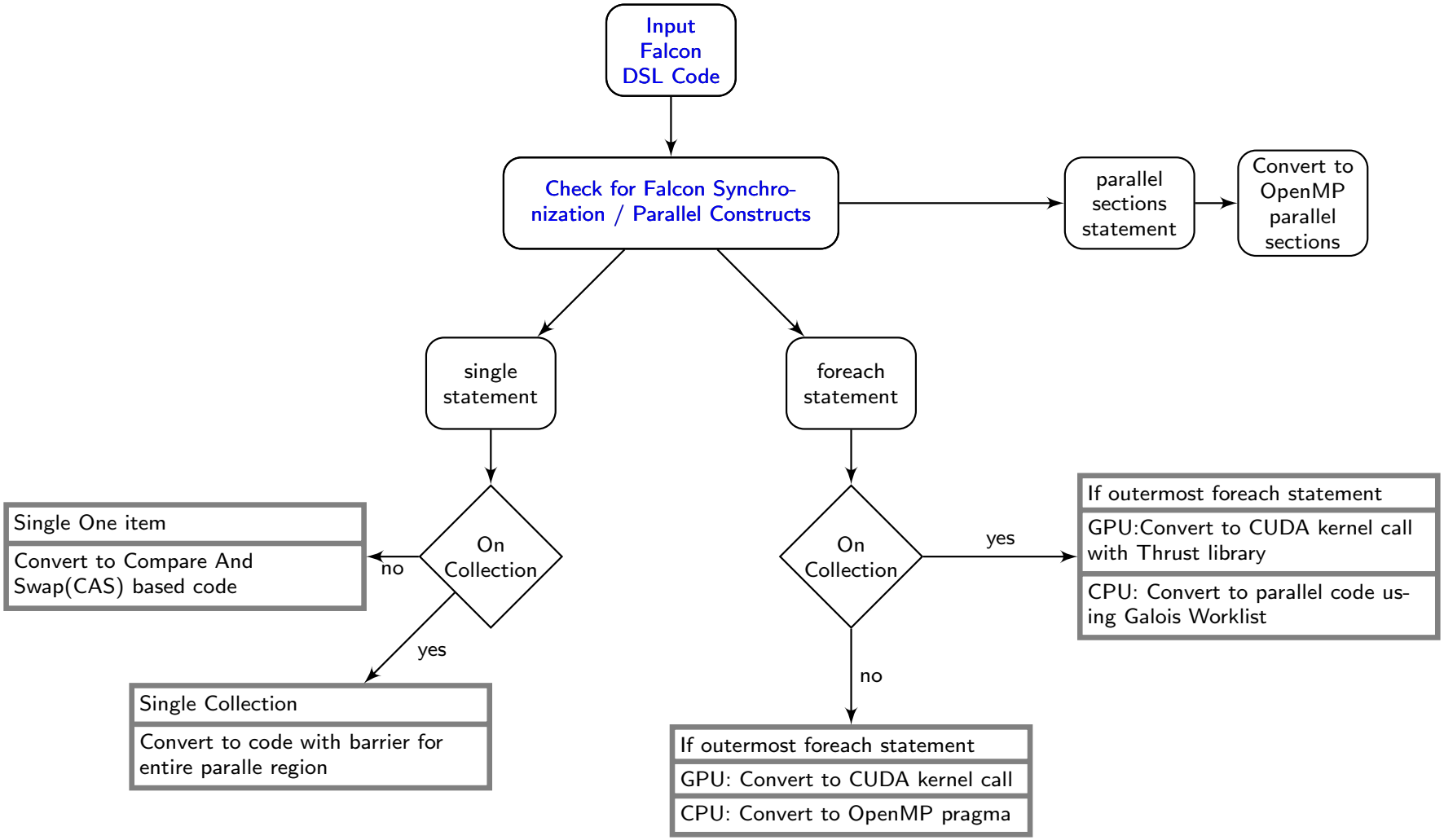
Falcon Compiler Code Genaration (Synchronization and parallelization constructs)



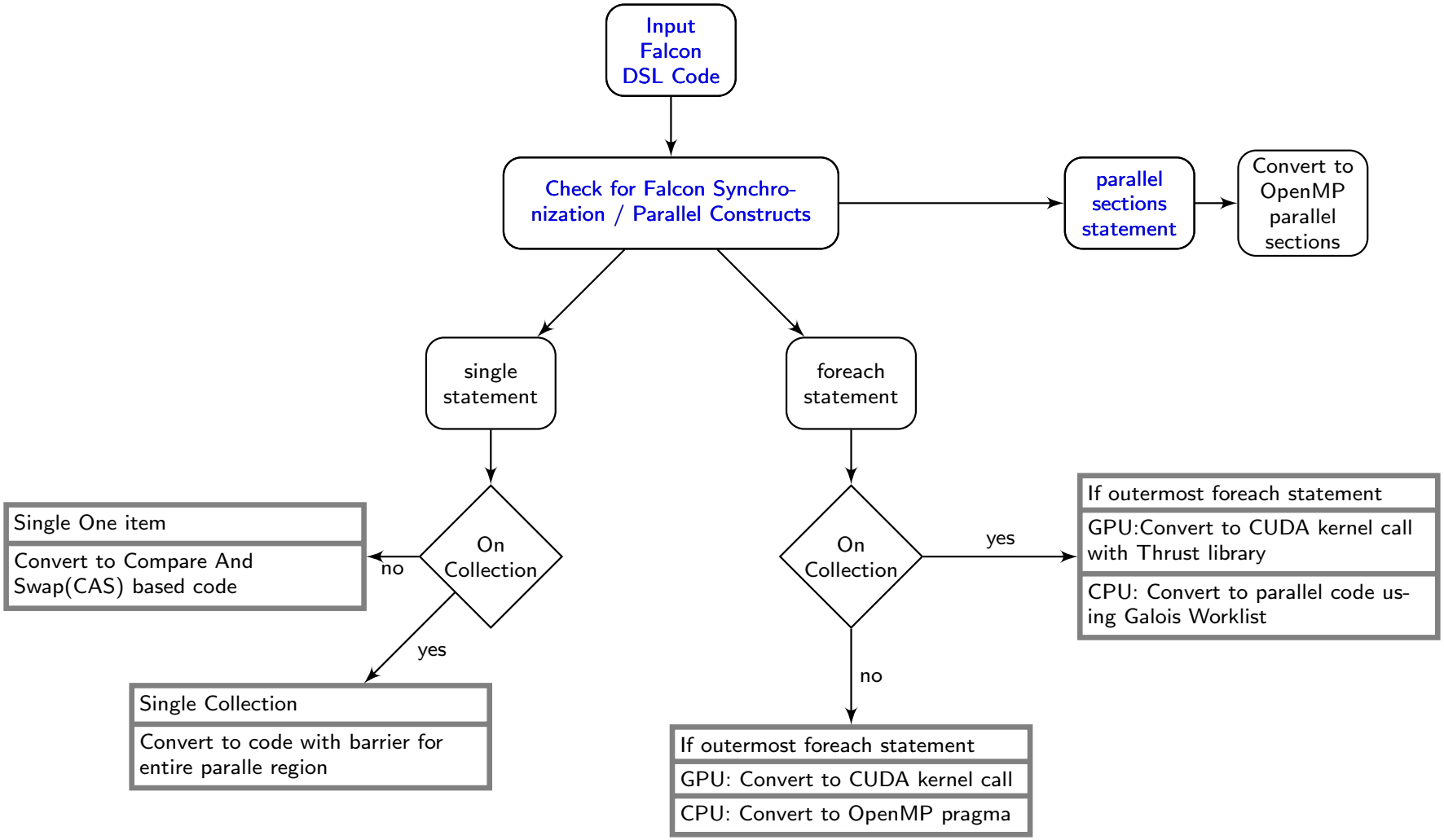
Falcon Compiler Code Genaration (Synchronization and parallelization constructs)



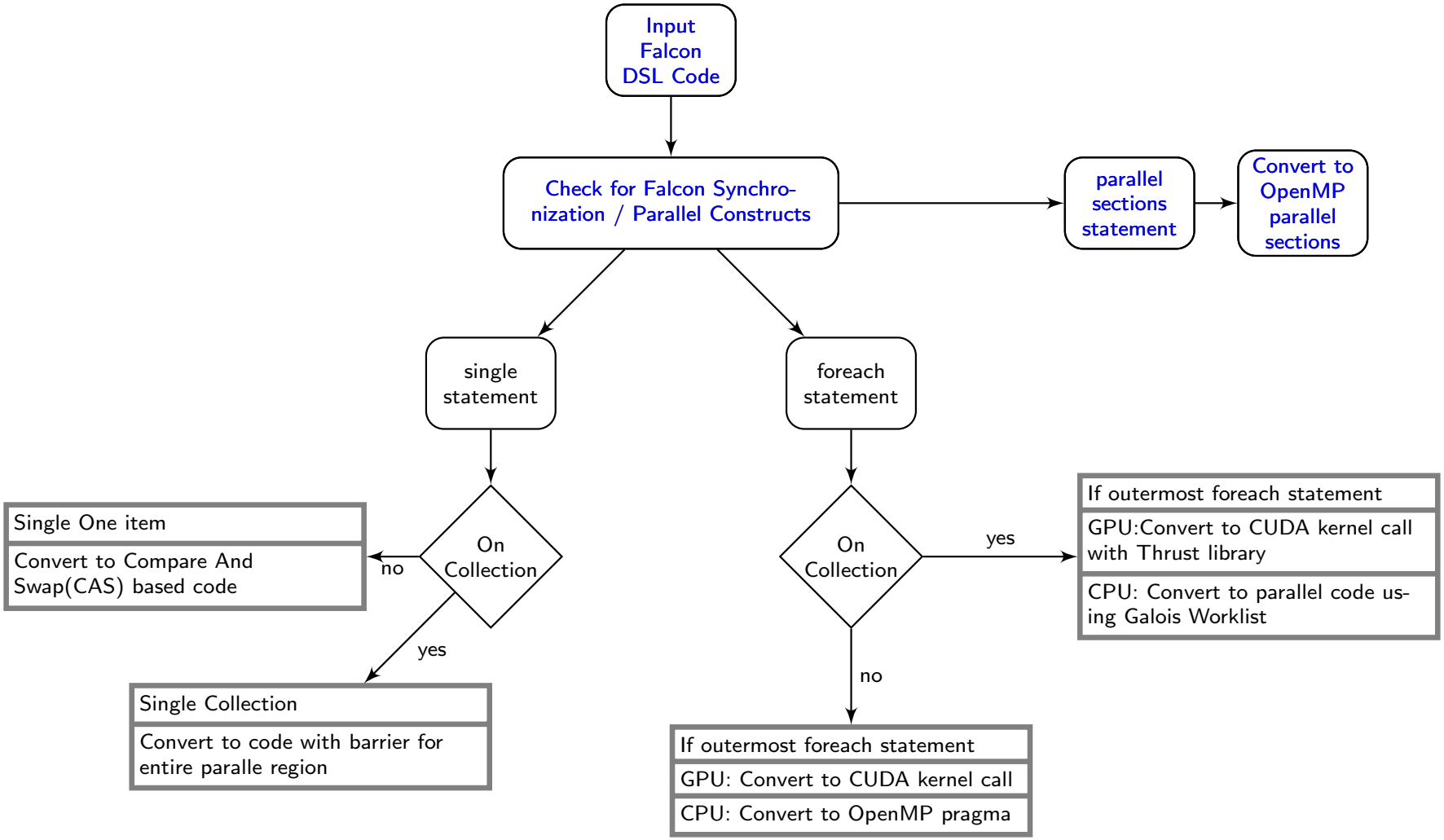
Falcon Compiler Code Genaration (Synchronization and parallelization constructs)



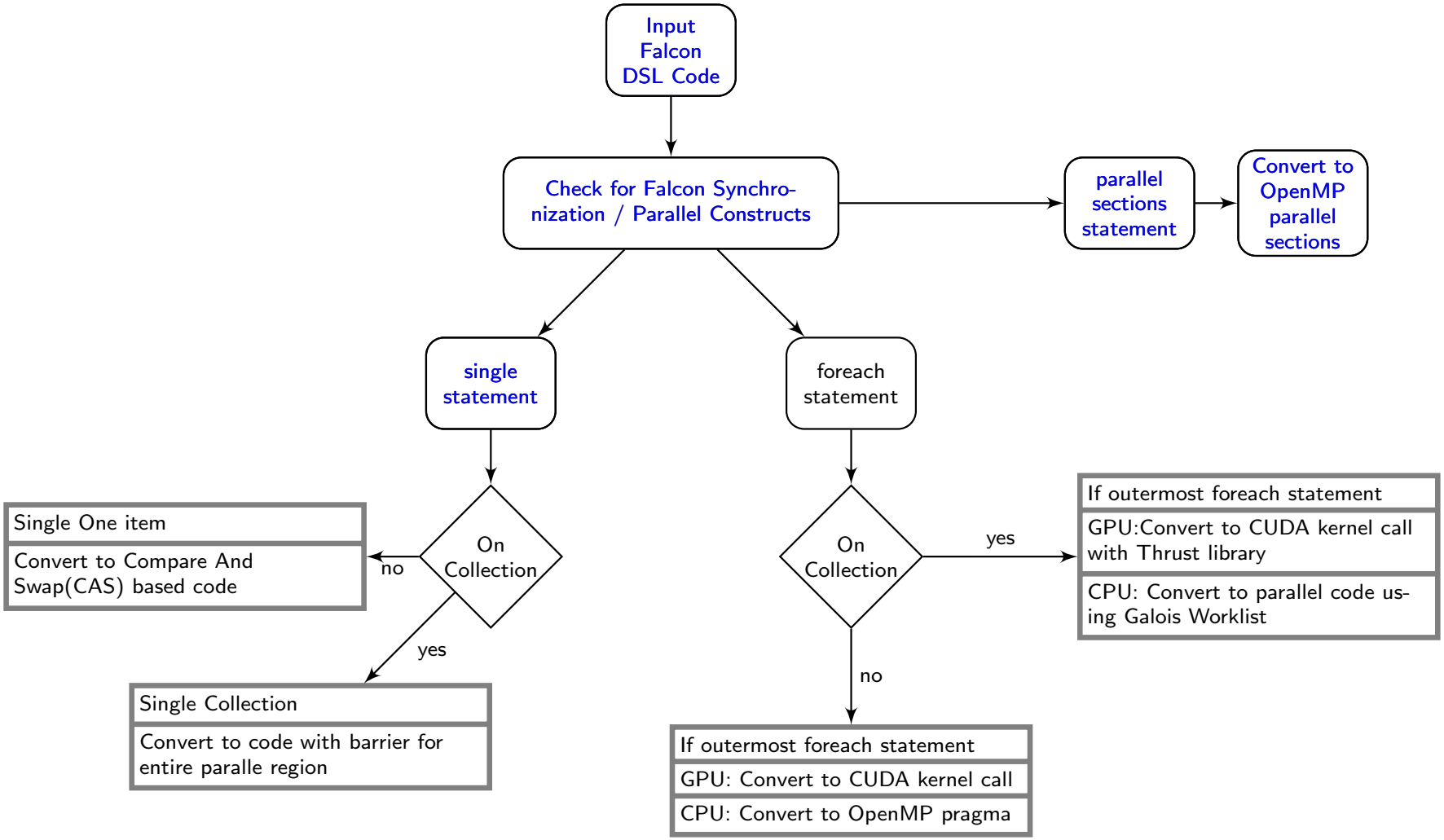
Falcon Compiler Code Genaration (Synchronization and parallelization constructs)



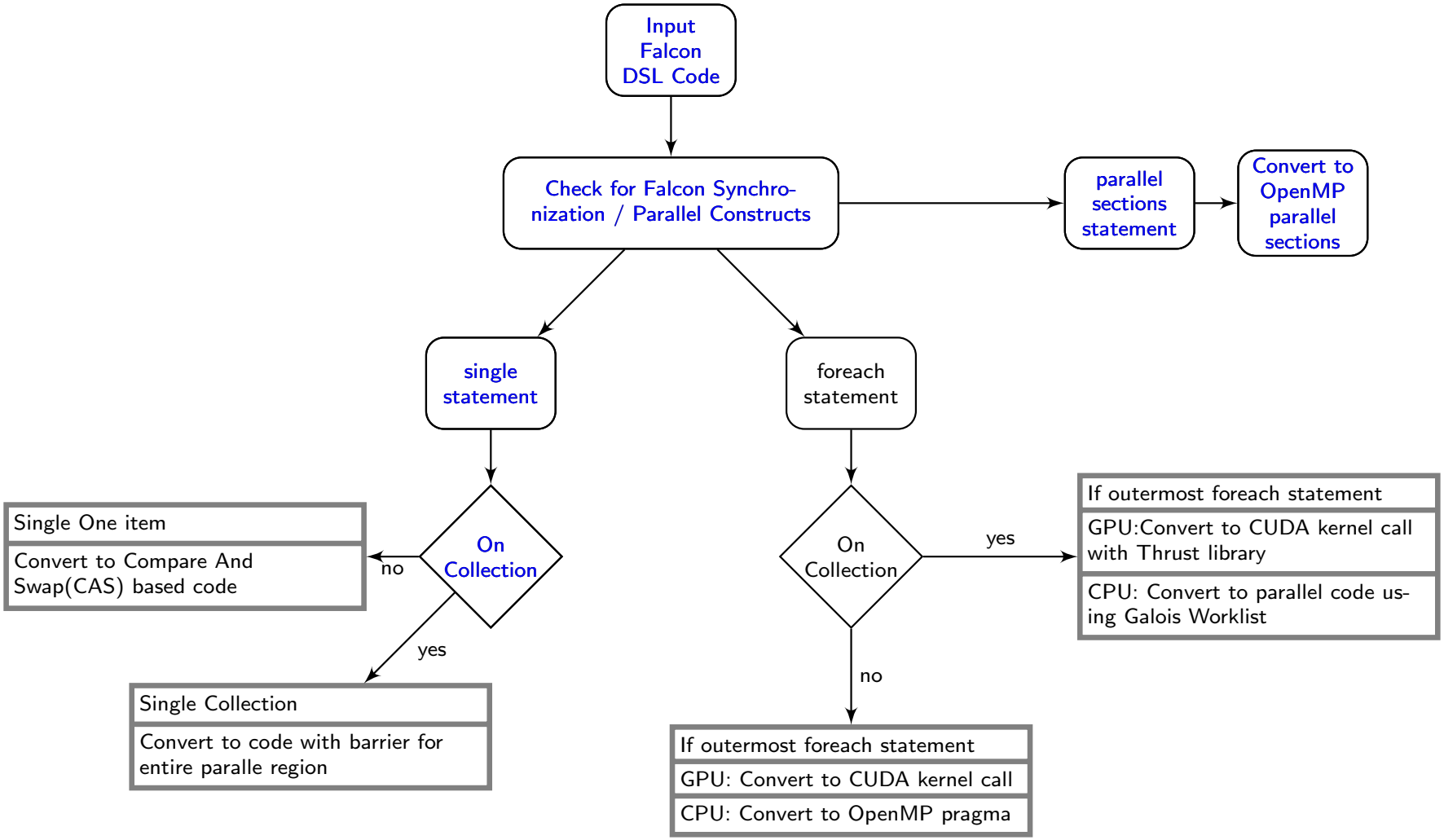
Falcon Compiler Code Genaration (Synchronization and parallelization constructs)



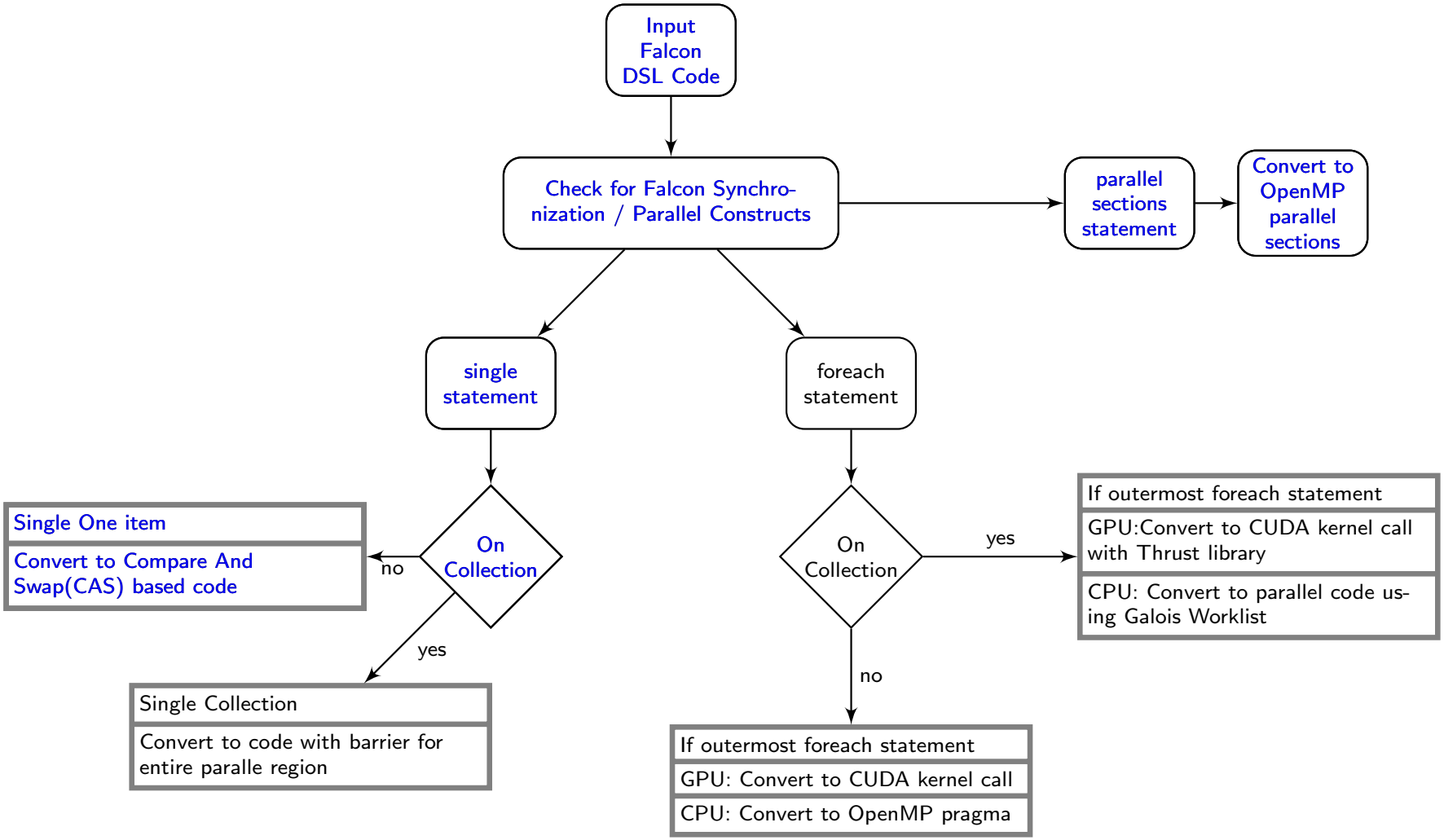
Falcon Compiler Code Genaration (Synchronization and parallelization constructs)



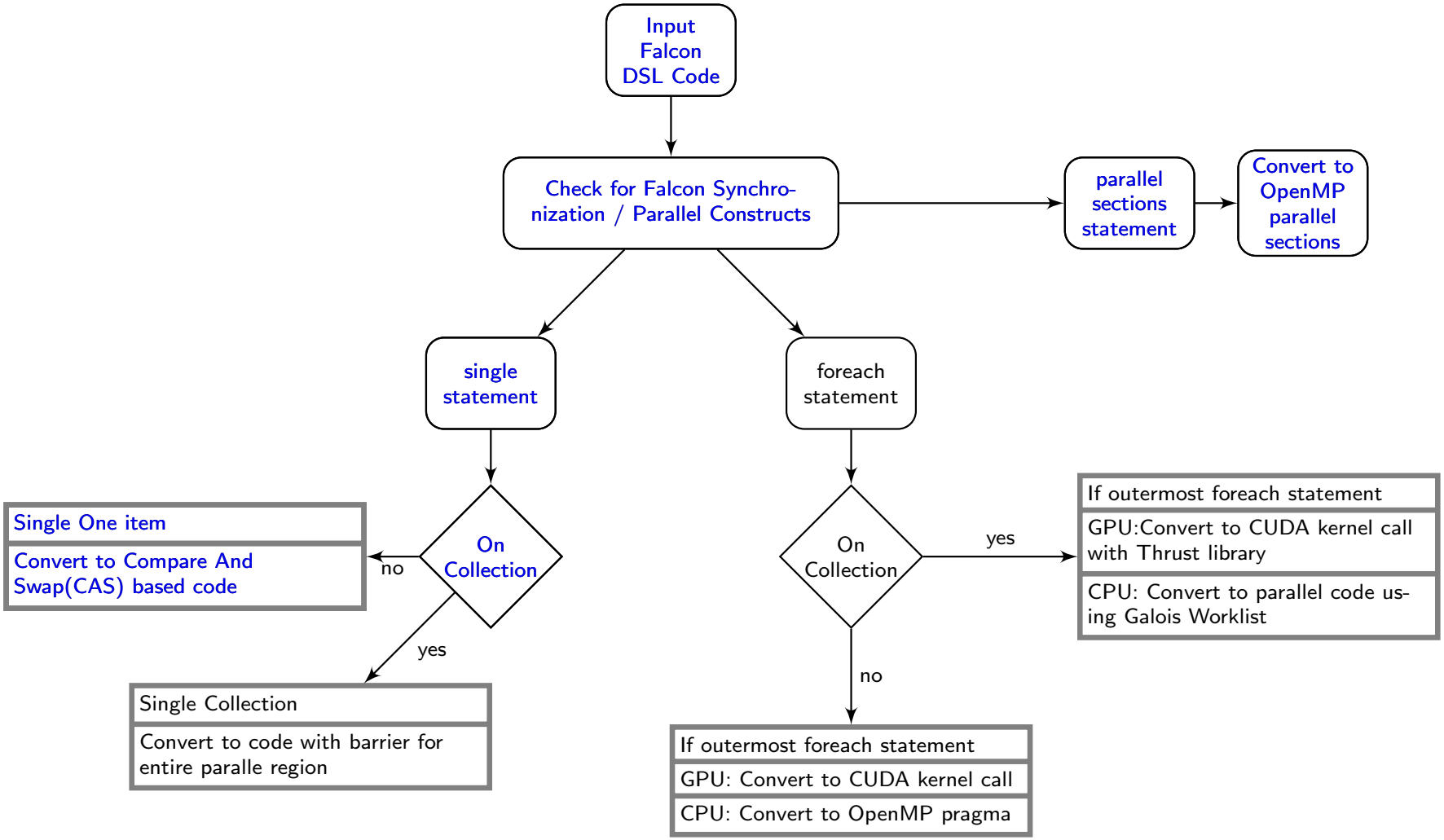
Falcon Compiler Code Genaration (Synchronization and parallelization constructs)



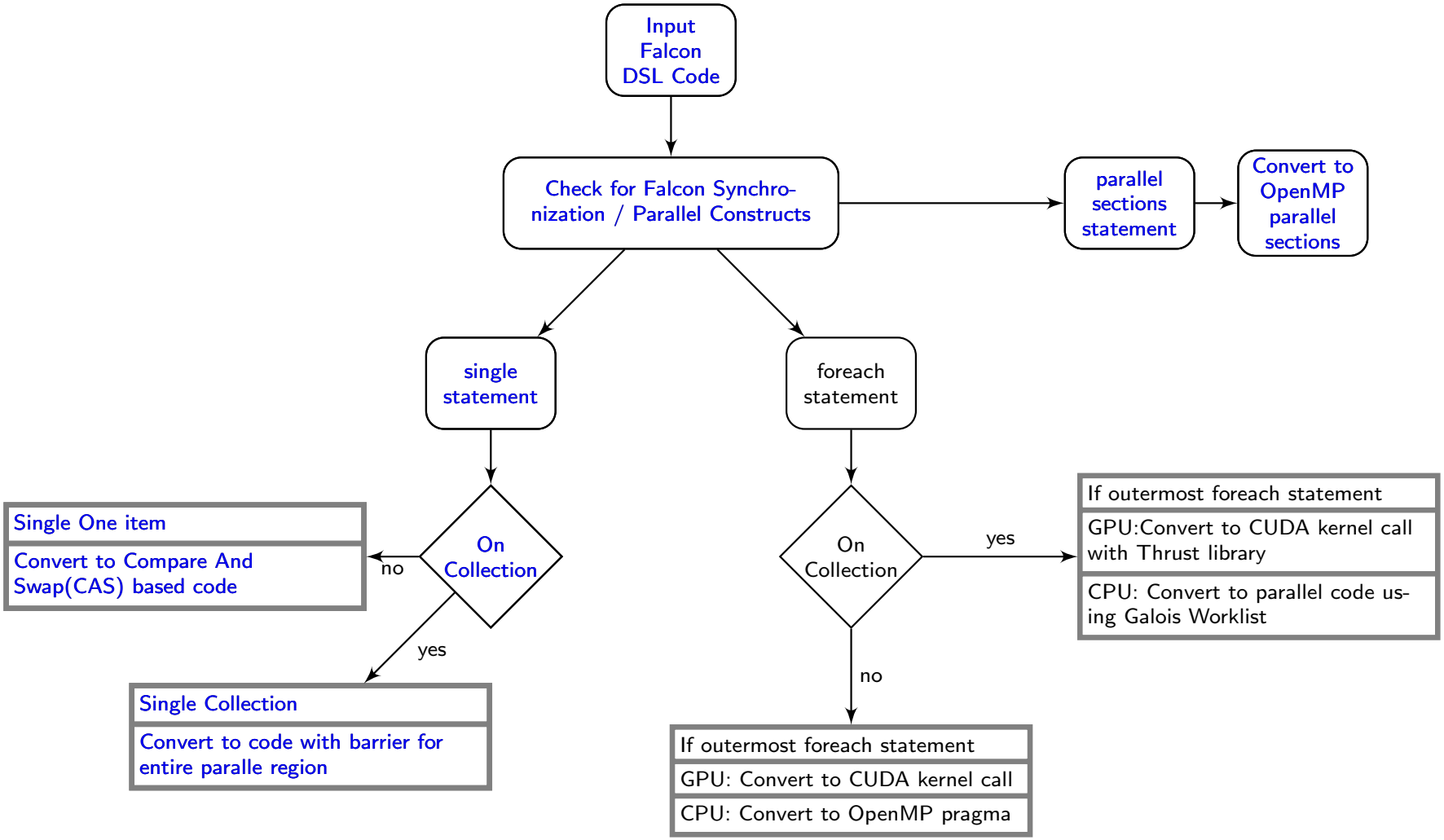
Falcon Compiler Code Genaration (Synchronization and parallelization constructs)



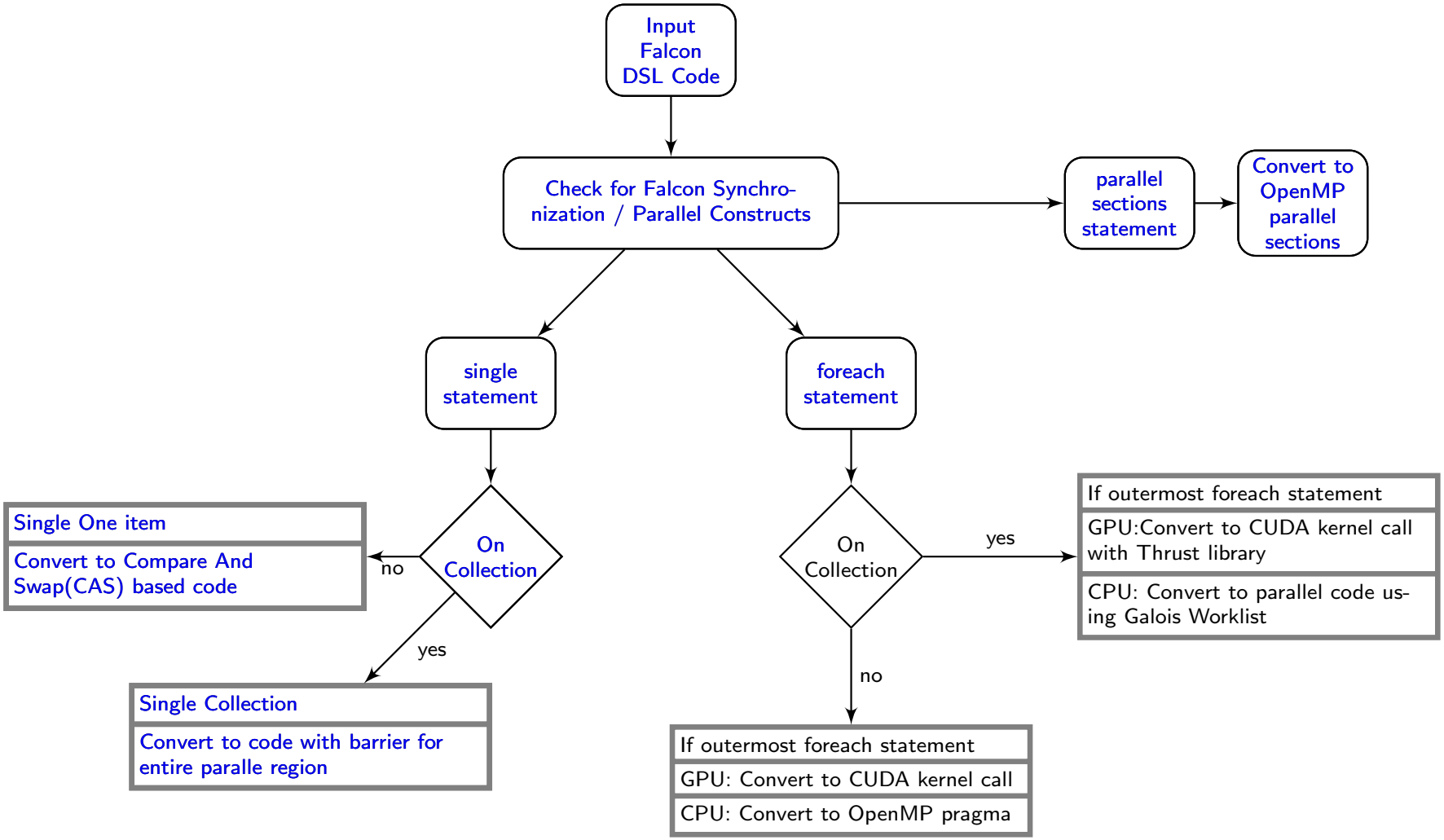
Falcon Compiler Code Genaration (Synchronization and parallelization constructs)



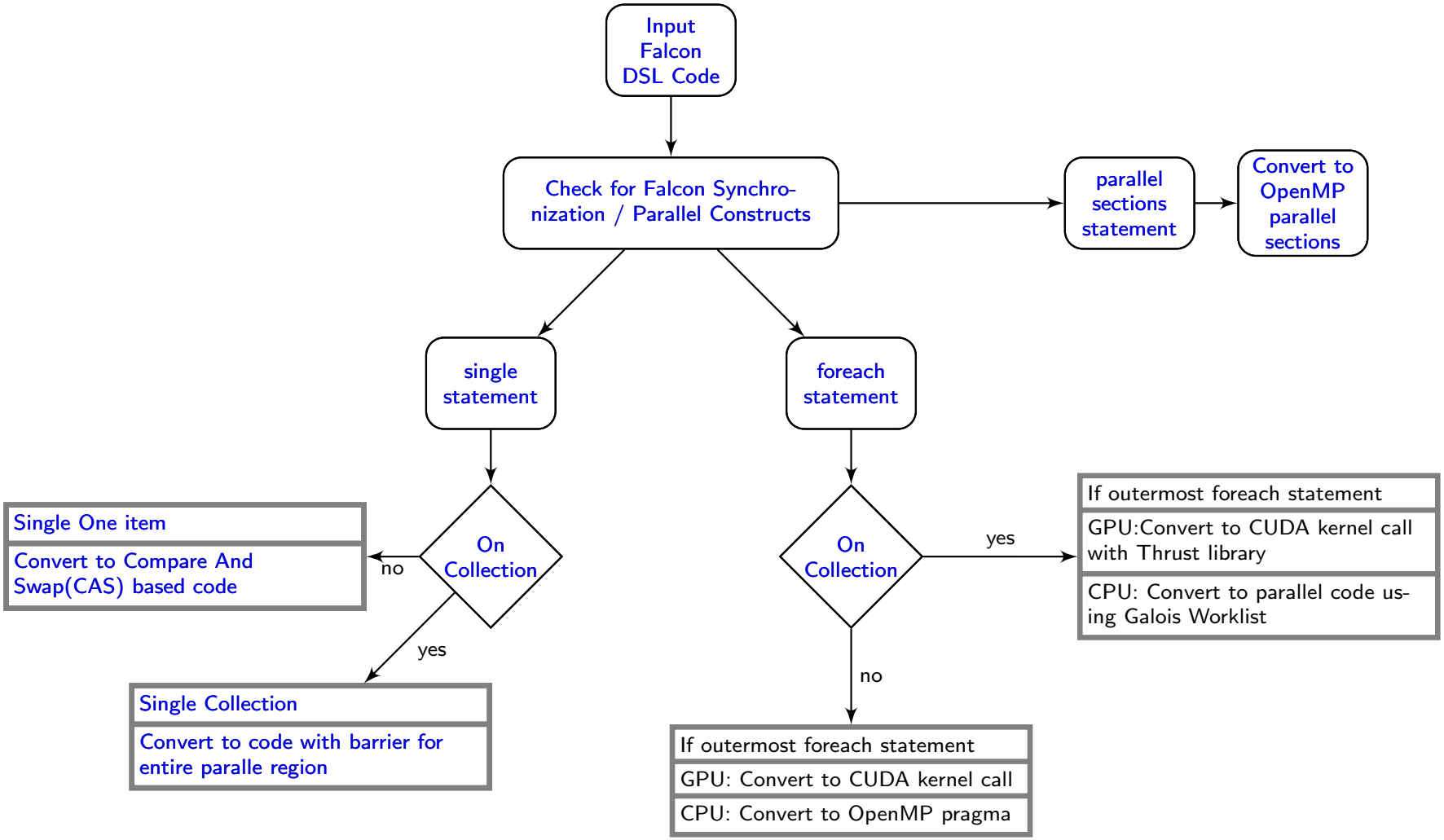
Falcon Compiler Code Genaration (Synchronization and parallelization constructs)



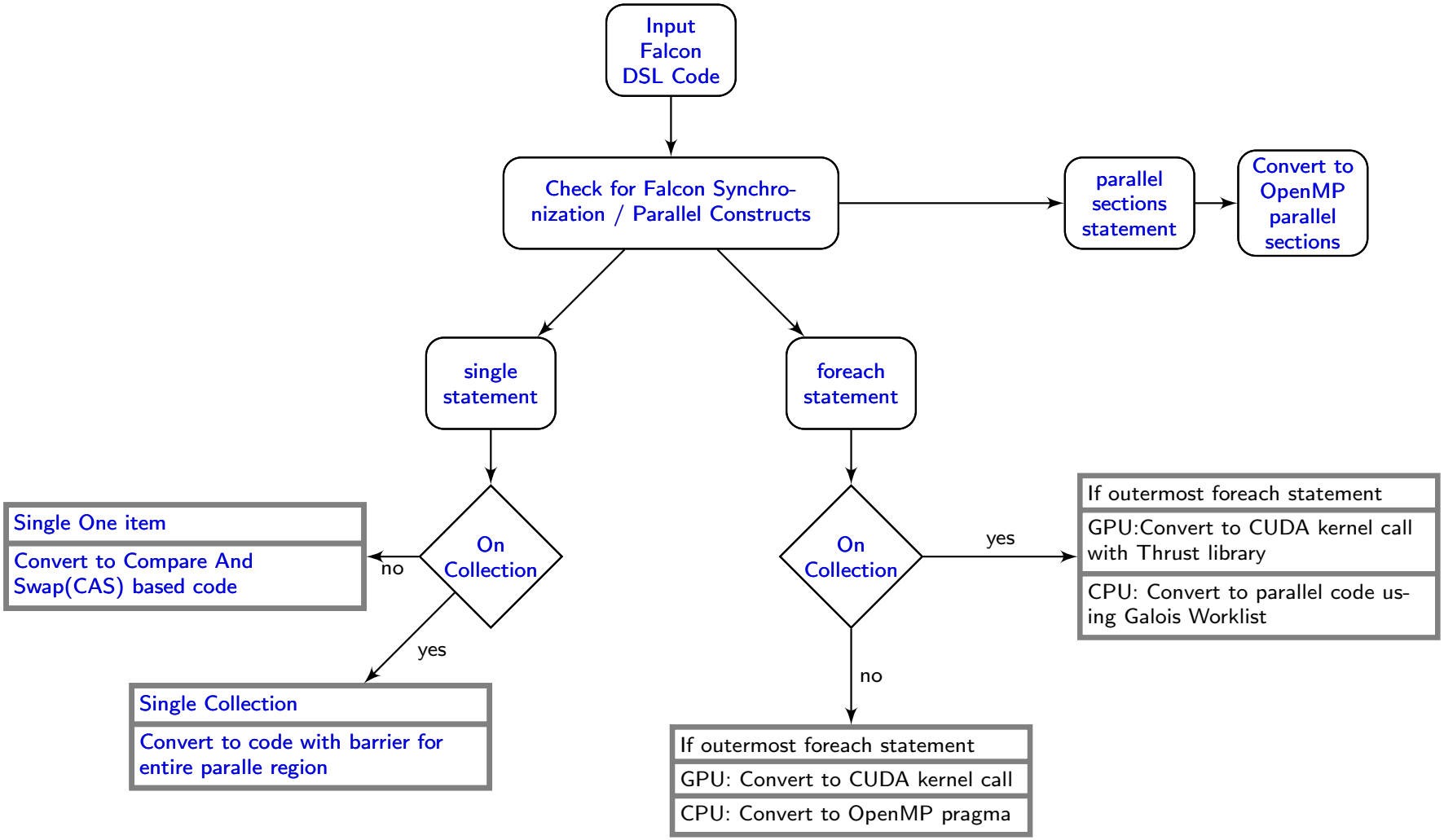
Falcon Compiler Code Genaration (Synchronization and parallelization constructs)



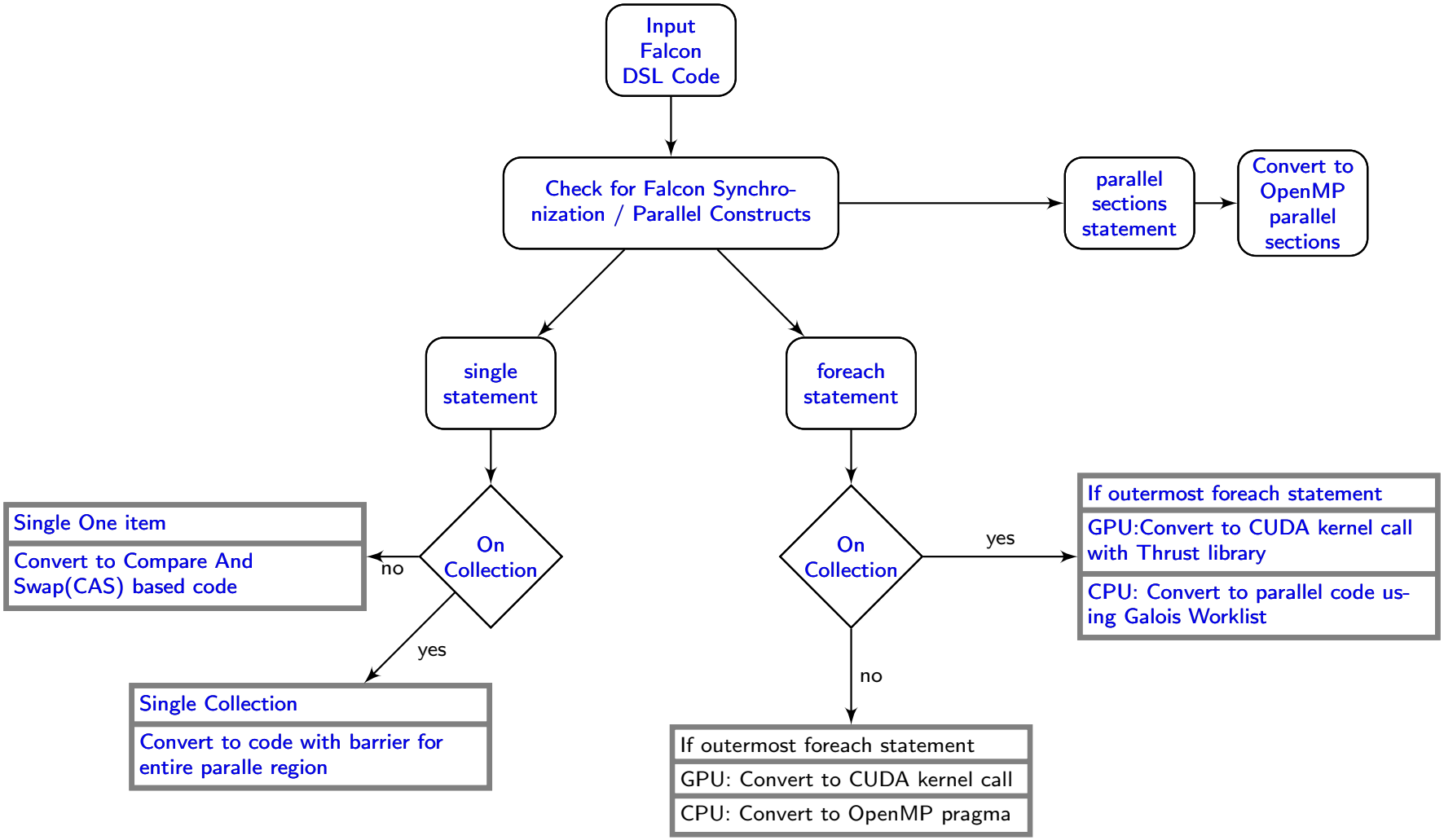
Falcon Compiler Code Genaration (Synchronization and parallelization constructs)



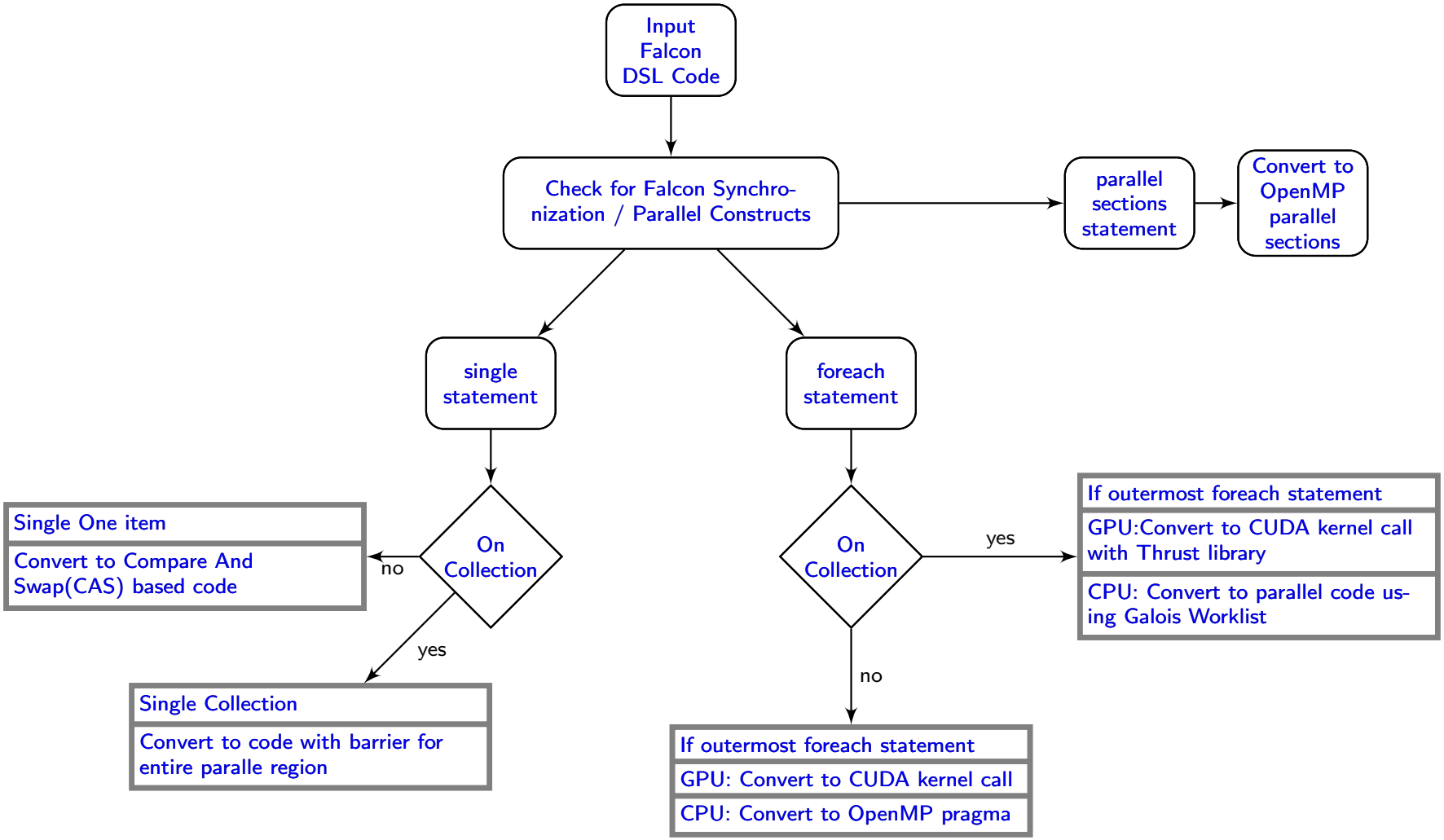
Falcon Compiler Code Genaration (Synchronization and parallelization constructs)



Falcon Compiler Code Genaration (Synchronization and parallelization constructs)



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 GPU 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 GPU 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.

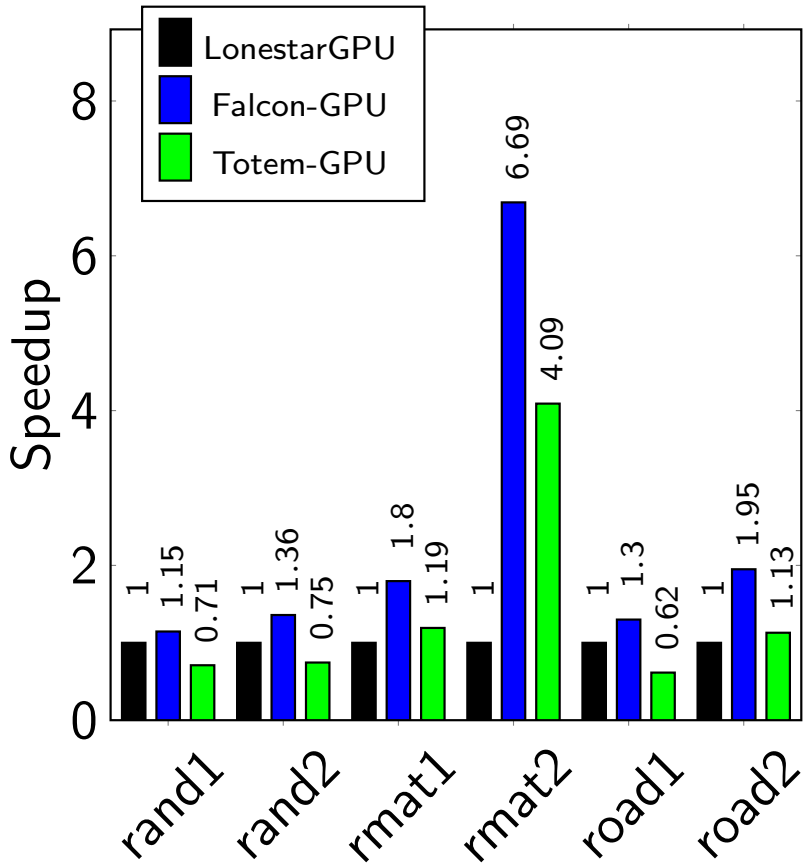
1. Using `Falcon` compiler we wrote algorithms like BFS, SSSP and Boruvka's-MST.
2. We wrote dynamic algorithms like Survey Propagation(SP), Delaunay Mesh refinement(DMR) and Dynamic-SSSP in `Falcon`.
3. Performance of GPU codes were compared with
   i) **LonestarGPU**-(ISS group at the University of 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.

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 GPU 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.

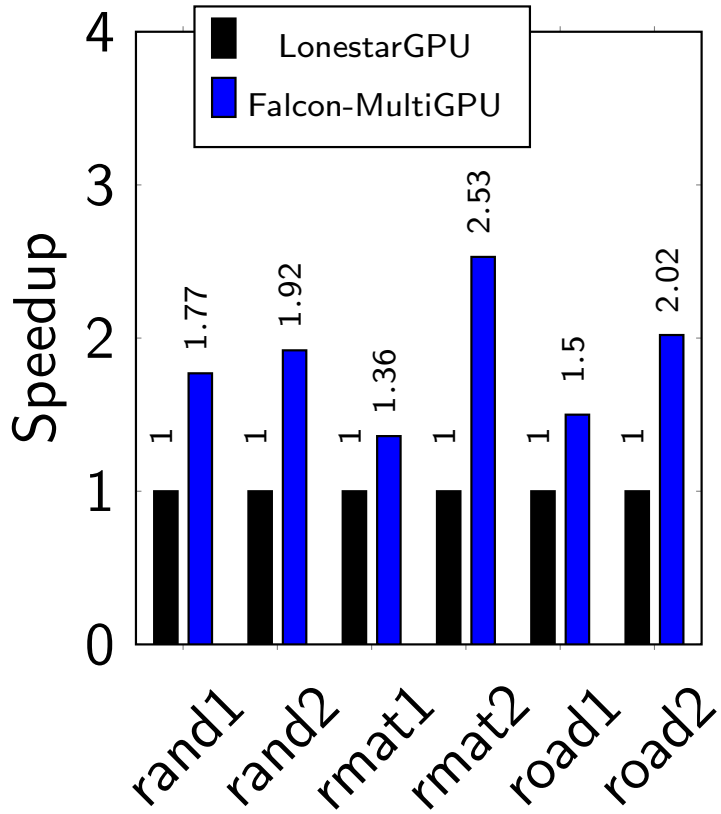
1. Using `Falcon` compiler we wrote algorithms like BFS, SSSP and Boruvka's-MST.
2. We wrote dynamic algorithms like Survey Propagation(SP), Delaunay Mesh refinement(DMR) and Dynamic-SSSP in `Falcon`.
3. Performance of GPU codes were compared with
   i) **LonestarGPU**-(ISS group at the University of 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.

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 GPU 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.
8. Tested on 12-core CPU and 4-GPU machine(1 Kepler and 3 Tesla).

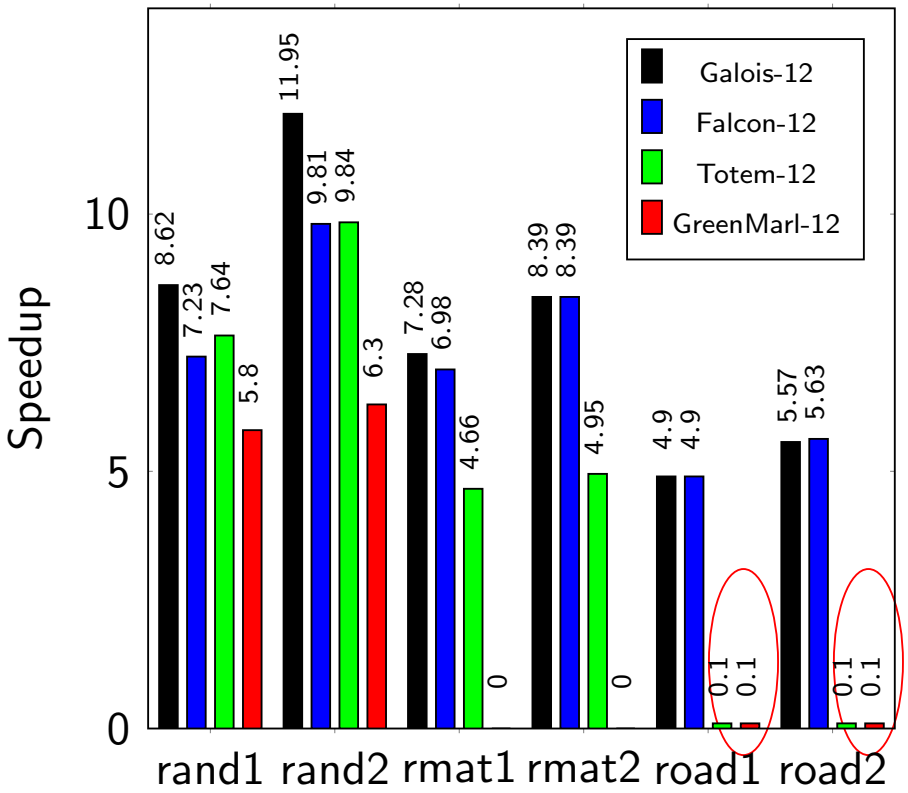
(Points,Edges) rand1(16M,64M),rand2(32M,128M)
rmat1(10M,100M),rmat2(20M,200M)
road1(14M,34M) road2(23M,58M)



Speedup of SSSP on GPU

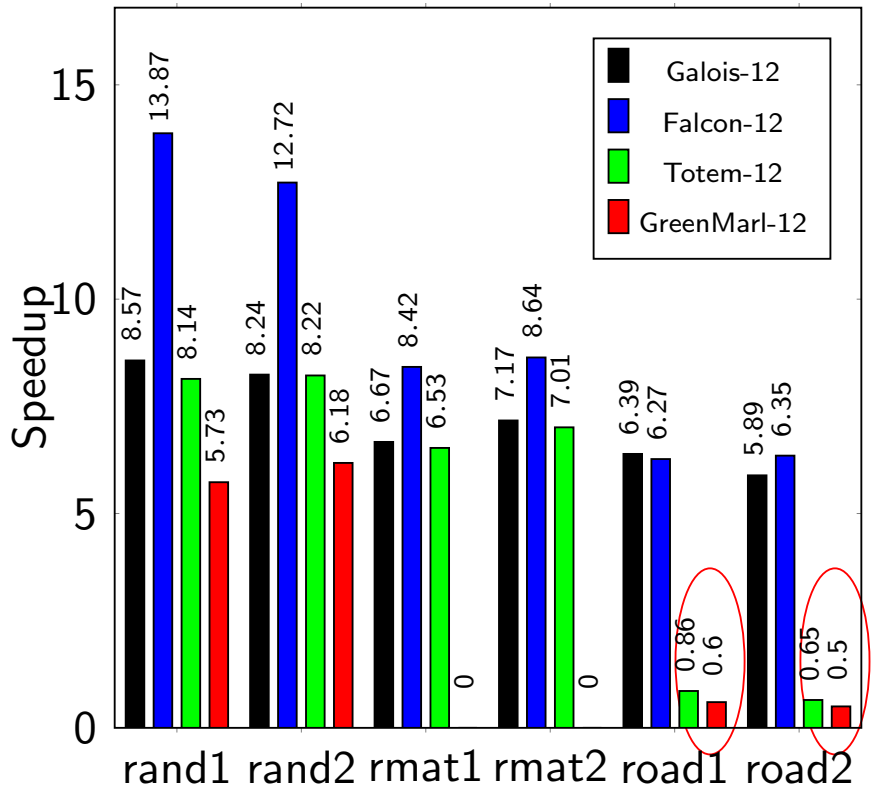


Speedup- SSSP, BFS and MST in parallel on 3 GPUs



(a)

SSSP speedup over Galois Single



(b)

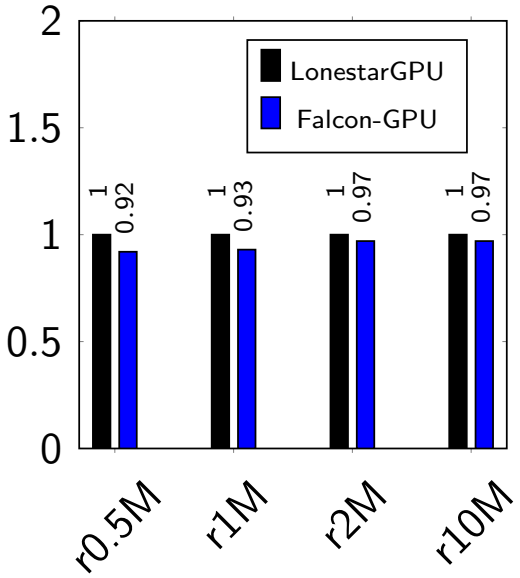
BFS speedup over Galois single

1. single on collection and barrier on GPU.

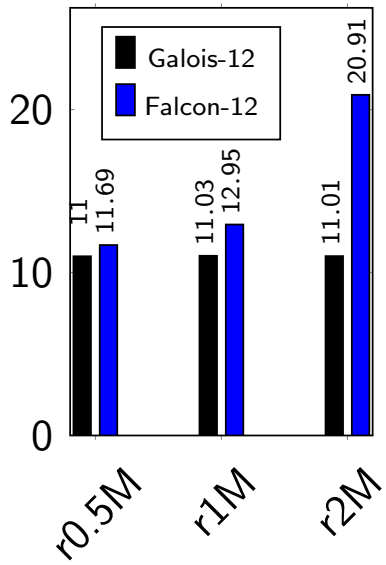
1. single on collection and barrier on GPU.
2. Shucai Xiao and Wu chun Feng. Inter-Block GPU Communication via Fast Barrier Synchronization.IPDPS 2010.

1. single on collection and barrier on GPU.
2. Shucai Xiao and Wu chun Feng. Inter-Block GPU Communication via Fast Barrier Synchronization.IPDPS 2010.
3. Importance of good runtime for Collection/Worklist.

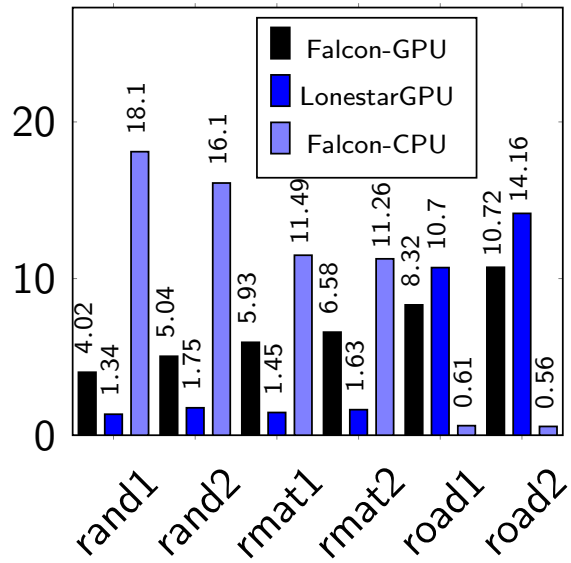
1. single on collection and barrier on GPU.
2. Shucai Xiao and Wu chun Feng. Inter-Block GPU Communication via Fast Barrier Synchronization.IPDPS 2010.
3. Importance of good runtime for Collection/Worklist.
4. Ulrich Meyer and Peter Sanders. *Delta*-Stepping: A Parallel Single Source Shortest Path Algorithm.EAS 1998.



(a) DMR speedup over LonestarGPU

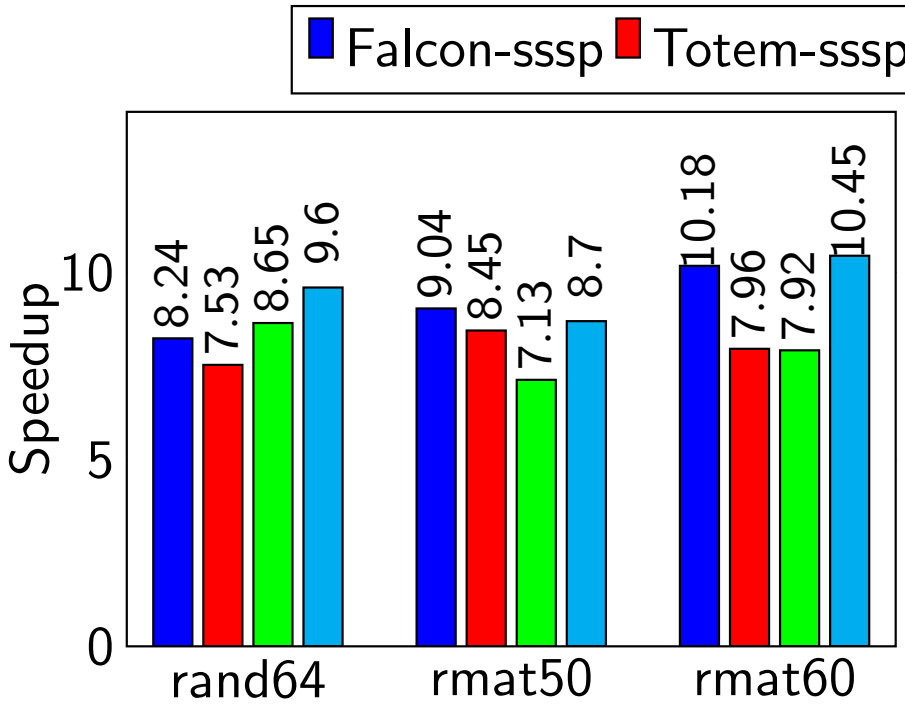


(b) DMR speedup over Galois single

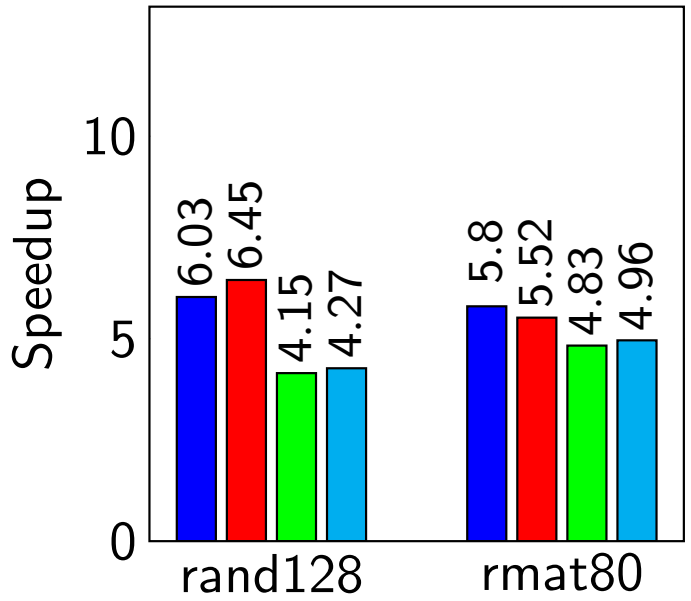


(c) Dynamic-SSSP- Self relative speedup

Morph Algorithm Results-Speedup
DMR(Delaunay Mesh Refinement) and Dynamic-SSSP



speedup on two GPUs



Speedup on two GPUs and one CPU

Heterogeneous Execution-SSSP and BFS speedup

Partitioned Execution of one Algorithm on Multiple Devices

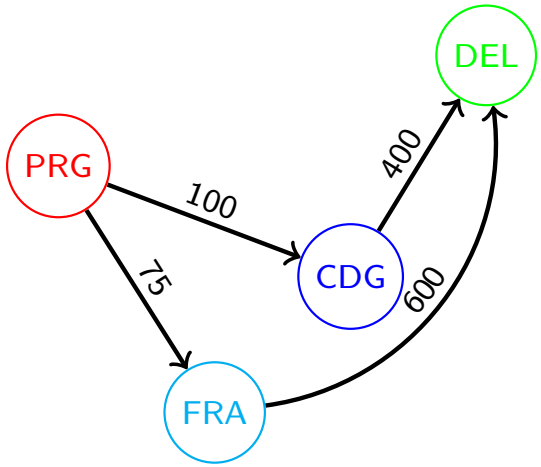
```
1  fun1(Point ori, Point incom){
2        if(orig.dist >incom.dist)
3              orig.dist=incom.dist
4  }
5  relaxgraph(Point p, HGraph hgraph){
6     foreach(t in p.outnbrs)
7        MIN(t.dist,p.dist+hgraph.getWeight(p,t), hgraph.changed[0]);
8  }
9  main(int argc, char *argv[]) {
10       HGraph hgraph;
11       hgraph.addPointProperty(dist, int);
12             hgraph.addProperty(changed, int);
13       hgraph.read(argv[1]);
14             hgraph.makePartition(1,1,ORDERED);
15             hgraph.updateFunction(fun1);
16       foreach(t In hgraph.points) t.dist=1234567890;
17       hgraph.points[0].dist=0;
18       while( 1 ){
19            hgraph.changed[0]=0;
20            foreach(t In hgraph.points)relaxgraph(t,hgraph);
21            if(hgraph.changed[0]==0)break;
22            hgraph.updatePartition();
23       }
24       for(int i = 0;i <hgraph.npoints; i++)
25          printf("%d", hgraph.points[i].dist);
26 }
```

A SIMPLE EXAMPLE

Partitioned Execution of one Algorithm on Multiple Devices

```
1  fun1(Point ori, Point incom){
2        if(orig.dist >incom.dist)
3              orig.dist=incom.dist
4  }
5  relaxgraph(Point p, HGraph hgraph){
6     foreach(t in p.outnbrs)
7        MIN(t.dist,p.dist+hgraph.getWeight(p,t), hgraph.changed[0]);
8  }
9  main(int argc, char *argv[]) {
10       HGraph hgraph;
11       hgraph.addPointProperty(dist, int);
12             hgraph.addProperty(changed, int);
13       hgraph.read(argv[1]);
14             hgraph.makePartition(1,1,ORDERED);
15             hgraph.updateFunction(fun1);
16       foreach(t In hgraph.points) t.dist=1234567890;
17       hgraph.points[0].dist=0;
18       while( 1 ){
19            hgraph.changed[0]=0;
20            foreach(t In hgraph.points)relaxgraph(t,hgraph);
21            if(hgraph.changed[0]==0)break;
22            hgraph.updatePartition();
23       }
24       for(int i = 0;i <hgraph.npoints; i++)
25          printf("%d", hgraph.points[i].dist);
26 }
```

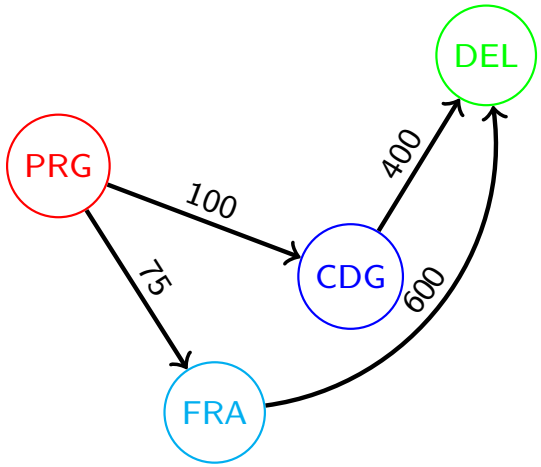
A SIMPLE EXAMPLE



Partitioned Execution of one Algorithm on Multiple Devices

```
1  fun1(Point ori, Point incom){
2        if(orig.dist >incom.dist)
3              orig.dist=incom.dist
4  }
5  relaxgraph(Point p, HGraph hgraph){
6     foreach(t in p.outnbrs)
7        MIN(t.dist,p.dist+hgraph.getWeight(p,t), hgraph.changed[0]);
8  }
9  main(int argc, char *argv[]) {
10       HGraph hgraph;
11       hgraph.addPointProperty(dist, int);
12             hgraph.addProperty(changed, int);
13       hgraph.read(argv[1]);
14             hgraph.makePartition(1,1,ORDERED);
15             hgraph.updateFunction(fun1);
16       foreach(t In hgraph.points) t.dist=1234567890;
17       hgraph.points[0].dist=0;
18       while( 1 ){
19            hgraph.changed[0]=0;
20            foreach(t In hgraph.points)relaxgraph(t,hgraph);
21            if(hgraph.changed[0]==0)break;
22            hgraph.updatePartition();
23       }
24       for(int i = 0;i <hgraph.npoints; i++)
25          printf("%d", hgraph.points[i].dist);
26 }
```

A SIMPLE EXAMPLE



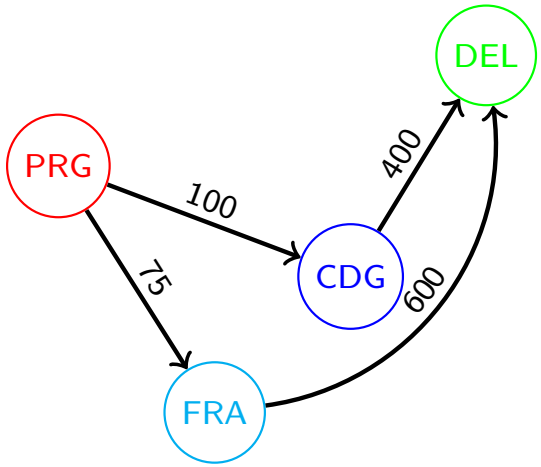
Part1(PRG, CDG)
0 , INF , INF , INF
0 , 100 , 75 , INF
0 , 100 , 75 , INF
0 , 100 , 75 , 500
0 , 100 , 75 , 500

Part2(FRA,DEL)
0 , INF, INF , INF
0 , INF , INF , INF
0 , 100 , 75 , INF
0 , 100, 75 , 675
0 , 100 , 75 , 500

Partitioned Execution of one Algorithm on Multiple Devices

```
1  fun1(Point ori, Point incom){
2        if(orig.dist >incom.dist)
3              orig.dist=incom.dist
4  }
5  relaxgraph(Point p, HGraph hgraph){
6     foreach(t in p.outnbrs)
7        MIN(t.dist,p.dist+hgraph.getWeight(p,t), hgraph.changed[0]);
8  }
9  main(int argc, char *argv[]) {
10       HGraph hgraph;
11       hgraph.addPointProperty(dist, int);
12             hgraph.addProperty(changed, int);
13       hgraph.read(argv[1]);
14             hgraph.makePartition(1,1,ORDERED);
15             hgraph.updateFunction(fun1);
16       foreach(t In hgraph.points) t.dist=1234567890;
17       hgraph.points[0].dist=0;
18       while( 1 ){
19            hgraph.changed[0]=0;
20            foreach(t In hgraph.points)relaxgraph(t,hgraph);
21            if(hgraph.changed[0]==0)break;
22            hgraph.updatePartition();
23       }
24       for(int i = 0;i <hgraph.npoints; i++)
25          printf("%d", hgraph.points[i].dist);
26 }
```

A SIMPLE EXAMPLE

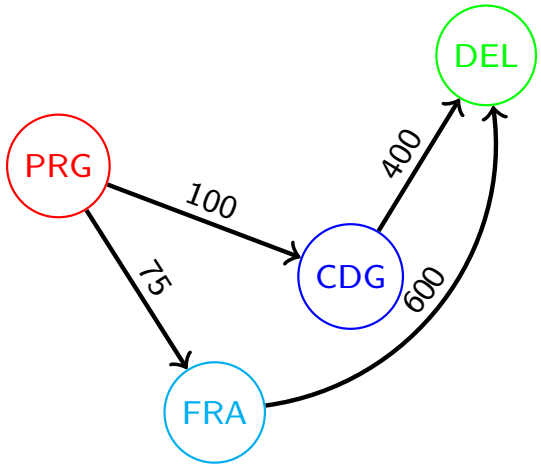


Part1(PRG, CDG)	Part2(FRA,DEL)
0 , INF , INF , INF	0 , INF, INF , INF
0 , 100 , 75 , INF	0 , INF , INF , INF
0 , 100 , 75 , INF	0 , 100 , 75 , INF
0 , 100 , 75 , 500	0 , 100, 75 , 675
0 , 100 , 75 , 500	0 , 100 , 75 , 500

Partitioned Execution of one Algorithm on Multiple Devices

```
1  fun1(Point ori, Point incom){
2        if(orig.dist >incom.dist)
3              orig.dist=incom.dist
4  }
5  relaxgraph(Point p, HGraph hgraph){
6     foreach(t in p.outnbrs)
7        MIN(t.dist,p.dist+hgraph.getWeight(p,t), hgraph.changed[0]);
8  }
9  main(int argc, char *argv[]) {
10       HGraph hgraph;
11       hgraph.addPointProperty(dist, int);
12             hgraph.addProperty(changed, int);
13       hgraph.read(argv[1]);
14             hgraph.makePartition(1,1,ORDERED);
15             hgraph.updateFunction(fun1);
16       foreach(t In hgraph.points) t.dist=1234567890;
17       hgraph.points[0].dist=0;
18       while( 1 ){
19            hgraph.changed[0]=0;
20            foreach(t In hgraph.points)relaxgraph(t,hgraph);
21            if(hgraph.changed[0]==0)break;
22            hgraph.updatePartition();
23       }
24       for(int i = 0;i <hgraph.npoints; i++)
25          printf("%d", hgraph.points[i].dist);
26 }
```

A SIMPLE EXAMPLE



Part1(PRG, CDG)	Part2(FRA,DEL)
0 , INF , INF , INF	0 , INF, INF , INF
0 , 100 , 75 , INF	0 , INF , INF , INF
0 , 100 , 75 , INF	0 , 100 , 75 , INF
0 , 100 , 75 , 500	0 , 100, 75 , 675
0 , 100 , 75 , 500	0 , 100 , 75 , 500

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.

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.
5. Some sample autogenereated code can be downloaded from my home page(http://clweb.csa.iisc.ernet.in/unni_c).

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.
5. Some sample autogenereated code can be downloaded from my home page(http://clweb.csa.iisc.ernet.in/unni_c).
6. The Falcon compiler will made available online.

Questions??

THANK YOU

DĚKUJI

```
1  relaxgraph(Point <GPU>p, Graph <GPU>graph) {
2      foreach (t In p.outnbrs)
3        MIN(t.dist, p.dist + graph.getweight(p, t), changed);
4  }
```

```
1  #define t (((struct_hgraph *)(graph.extra)))
2  __global__ void relaxgraph(GGraph graph, int x) {
       id = blockIdx.x * blockDim.x +
3                threadIdx.x + x;
4      if (id <graph.npoints){
5      int falcft0 = graph.index[id];
6      int falcft1 = graph.index[id+1]-graph.index[id];
8      for (falcft2 = 0; falcft2 <falcft1; falcft2++) {
9          int ut0 = 2 * (falcft0 + falcft2); //edge index
10         int ut1 = graph.edges[ut0].ipe; //dest point
11         int ut2 = graph.edges[ut0 + 1].ipe;
12         GMIN(&t->dist[ut1], t->dist[id] + ut2, changed);
13     }
14 }
```

Code generated for GPU

```
1  relaxgraph(Point p, Graph graph) {
2      foreach (t In p.outnbrs)
3        MIN(t.dist, p.dist + graph.getweight(p, t),
       changed);
4  }
```

```
1  #define t (((struct_hgraph *)(graph.extra)))
2  relaxgraph(int &p ,HGraph &graph) {
3      int falcft0 = graph.index[p];
4      int falcft1 = graph.index[p+1]-graph.index[p];
7      for (int falcft2 = 0; falcft2 <falcft1;
       falcft2++) {
8          int ut0 = 2 * (falcft0 + falcft2);
9          int ut1 = graph.edges[ut0].ipe;
10         int ut2 = graph.edges[ut0 + 1].ipe;
               HMIN(&t->dist[ut1],
11         t->dist[p] + ut2, ut1, changed);
12     }
14 }
```

Code generated for CPU

Code generated for relaxgraph function of SSSP example

```
1  int <GPU>changed=0,lev=0;
2  BFS(Point <GPU>p,Graph <GPU>graph,int lev) {
3      foreach( t In p.outnbrs ){
4          if(t.dist>(lev+1)) {
5          t.dist=lev+1;changed=1;
6          }
7      }
8  }
9  main(int argc, char *name[]) {
10     Graph hgraph;
11     hgraph.addPointProperty(dist,int);
12     hgraph.read(name[1]);
13     hgraph.getType() <GPU>graph;
14     graph=hgraph;
15     foreach(t In graph.points)t.dist=1234567890;
16     graph.points[0].dist=0;
17     while( 1 ){
18         changed=0;
19         foreach(t In graph.points)(t.dist==lev)
           BFS(t,graph);
20         if(changed==0)break;
21         lev++;
22     }
23     for(int i=0;i<graph.npoints;i++)
24     printf("%d\n",graph.points[i].dist);
25 }
```

Falcon code for BFS on GPU

```
1  int changed=0,lev=0;
2  BFS(Point p,Graph graph,int lev) {
3      foreach( t In p.outnbrs ){
4          if(t.dist>(lev+1)) {
5          t.dist=lev+1;changed=1;
6          }
7      }
8  }
9  main(int argc, char *name[]) {
10     Graph hgraph;
11     hgraph.addPointProperty(dist,int);
12     hgraph.read(name[1]);
13     foreach(t In hgraph.points)
       t.dist=1234567890;
14     hgraph.points[0].dist=0;
15     while( 1 ){
16         changed=0;
17         foreach(t In hgraph.points)
           (t.dist==lev)BFS(t,hgraph,lev);
18         if(changed==0)break;
19         lev++;
20     }
21     for(int i=0;i<hgraph.npoints;i++)
22     printf("%d\n",hgraph.points[i].dist);
23 }
```

Falcon code for BFS on CPU

Dynamic Graph algorithms

1. `Falcon` provides addPoint(),addEdge() function on Graph class.

Dynamic Graph algorithms

1. `Falcon` provides addPoint(),addEdge() function on Graph class.
2. `Falcon` compiler checks for such function during code generation.

Dynamic Graph algorithms

1. `Falcon` provides addPoint(),addEdge() function on Graph class.
2. `Falcon` compiler checks for such function during code generation.
3. If such a function is found in DSL, Graph is pre-allocated with more space(say 3 times).

Dynamic Graph algorithms

1. `Falcon` provides addPoint(),addEdge() function on Graph class.
2. `Falcon` compiler checks for such function during code generation.
3. If such a function is found in DSL, Graph is pre-allocated with more space(say 3 times).
4. Deletion of Points/Edges are managed by marking. No Garbage Collection.

Graph member function addProperty() function

1. graph.addProperty(struct node,triangle);

Graph member function addProperty() function

1. graph.addProperty(struct node,triangle);
2. After this you can use triangle in same way as Edge and Point in DSL code.

Graph member function addProperty() function

1. graph.addProperty(struct node,triangle);
2. After this you can use triangle in same way as Edge and Point in DSL code.
3. It provides to programmer
   i)an iterator triangle
   ii) variable ntriangle, which store number of triangles.

Graph member function addProperty() function

1. graph.addProperty(struct node,triangle);
2. After this you can use triangle in same way as Edge and Point in DSL code.
3. It provides to programmer
   i)an iterator triangle
   ii) variable ntriangle, which store number of triangles.
4. This function can be used to view Graph as Collection of triangles(DMR) or as Collection of clauses(Survey Propagation).

Code for Parallel Execution of SSSP and BFS on two GPUs in Falcon

```
1  int <GPU>changed;
2  SSSPBFS(char *name) { //begin SSSPBFS
3  Graph hgraph;//Graph object on CPU
4  hgraph.addPointProperty(dist,int);
5  hgraph.addProperty(changed,int);
6  hgraph.getType() <GPU>graph0;//Graph on GPU0
7  hgraph.getType() <GPU>graph1;//Graph on GPU1
8  hgraph.addPointProperty(dist1,int);
9  hgraph.read(name);//read Graph from file to CPU
10 graph0=hgraph;//copy entire Graph to GPU0
11 graph1=hgraph;//copy entire Graph to GPU1
12 foreach(t In graph0.points)t.dist=1234567890;
13 foreach(t In graph1.points)t.dist=1234567890;
14 graph0.points[0].dist=0;
```

```
16 parallel sections { //do in parallel
17   section {//compute BFS on GPU1
18     while(1){
19       graph1.changed[0]=0;
20       foreach(t In graph1.points)BFS(t,graph1);
21       if(graph1.changed[0]==0) break;
22     }
23   }
24   section {//compute SSSP on GPU0
25     while(1){
26       graph0.changed[0]=0;
27       foreach(t In graph0.points)SSSP(t,graph0);
28       if(graph0.changed[0]==0) break;
29     }
30   }
31 }//end SSSPBFS
32 }
```