Falcon: A Graph Manipulation Language for Heterogeneous Systems

Unnikrishnan C, Department of CSA, IISc, email: unni.c@csa.iisc.ernet.in
Rupesh Nasre, Department of CSE, IIT Madras
Y N Srikant, Department of CSA, IISc

Abstract
Graph algorithms have been shown to possess enough parallelism to keep several computing resources busy even hundreds of cores on a GPU. Unfortunately, tuning their implementations for efficient execution on a particular hardware configuration of heterogeneous systems consisting of multicore CPUs and GPUs is challenging, time consuming, and error prone. To address these issues, we propose a domain-specific language (DSL) Falcon, for implementing graph algorithms that (i) abstracts the hardware, (ii) provides constructs to write explicitly parallel programs as a higher level, and (iii) can work with general algorithms that may change the graph structure (morph algorithms). We illustrate the usage of our DSL to implement local computation algorithms (that do not change the graph structure) and morph algorithms such as Delaunay mesh refinement, surface propagation, and dynamic SSSP on GPU and multicore CPUs. Using a set of benchmark graphs, we illustrate that the generated code performs close to the state-of-the-art hand-tuned implementations.

Falcon: Introduction
1. i) extends C programming language.
   ii) provides additional data types for Graph processing.
   iii) constructs for writing explicitly parallel graph algorithms.
2. Support for heterogeneous backends (CPU and GPU).
3. Supports parallel execution of different algorithms on multiple devices.
4. 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.
5. Allows viewing Graph in different ways (say collection of triangles).

Compiler Overview

Data Types and Iterators

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Iterator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graph</td>
<td>node, iterate over all points in graph, graph</td>
<td>iterate over all edges in graph</td>
</tr>
<tr>
<td>Graph</td>
<td>edge, iterate over all edges in graph</td>
<td></td>
</tr>
<tr>
<td>Graph</td>
<td>point, iterate over all elements in new ppu, point</td>
<td>iterate over all neighboring points</td>
</tr>
<tr>
<td>Print</td>
<td>point, iterate over all neighbors of point</td>
<td></td>
</tr>
<tr>
<td>Print</td>
<td>edge, iterate over all neighbor edges</td>
<td></td>
</tr>
<tr>
<td>Set</td>
<td>item, iterate over all items in Set</td>
<td></td>
</tr>
<tr>
<td>Collection</td>
<td>item, iterate over all items in Collection</td>
<td></td>
</tr>
</tbody>
</table>

Sample Falcon DSL codes

a) Multi-GPU SSSP and BFS

```c
10. Graph hgraph;//Graph object on CPU
11. hgraph.getType()
12. hgraph.addPointProperty(dist,int);
```

b) Heterogeneous Execution (CPU and GPUs)

```c
1. int argc, char *argv[]
2. hgraph.makePartition(1,1,ORDERED);
3. hgraph.updateFunction(fun1);
4. hgraph.read(argv[1]);
```

Results

1. Using Falcon compiler we devised algorithms like BFS, SSSP and Borrucka's MST.
2. We wrote dynamic algorithms like Survey Propagation (SP), Delaunay Mesh refinement (DMR) and Dynamic SSSP in Falcon.
3. Performance of Falcon 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 (NetSysLab, University of British Columbia)
4. Testing 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).

Future works

- OpenCL extension and Support for CPU and GPU cluster.
- For queries email to unni.c@csa.iisc.ernet.in

Publications