MR-API: A Comprehensive API Framework for Heterogeneous Multi-core Systems using Map Reduce Programming Model

Prashanth Thinakaran†, Srinivas Avireddy†, Prasanna Ranganathan†, Sumalatha Ramachandran‡

†*Undergraduate Students, Department of Information Technology, Anna University, Chennai-44*

‡*Associate Professor, Department of Information Technology, Anna University, Chennai-44*

Motivation for an effective API

- Existing cloud systems map general purpose computations to CPU
- Difficulty in Automation of mapping computation to heterogeneous processing unit
- Abstraction in mapping instructions to heterogeneous unit
- Tapping GPU resources in cloud to improve the performance of general purpose applications
- Evolving effective heuristics for instruction delegation across heterogeneous cores
- Performance optimization of DAG's

MR-API Features

- Comprehensive Application Programing Interface for Heterogeneous Multi-core Multi threaded Systems
- Runtime aggregation of application specific instructions
- Enhances parallelism through processor level parallelism
- Mapping computations at runtime for heterogeneous processing units
- Provides scalability, fault tolerance and load balancing
- Performance optimization of DAG's using operation coalescing and removal of redundant temporary arrays
- Programming model of **MR-API** is library based improving the level of Abstraction
- Performance improvement by **50-90** percent in contrast to the serial CPU only architectures

Perks of MR-API

- Task based segregation Viz. Heuristics to handle flat and block data to achieve better granularity
- MR-API decreases the code complexity by hiding low level GPGPU constructs
- Application speedup increases at nearly exponential rate for N-body simulation which has $O(n^2)$ complexity

To integrate MR-API with Hadoop, the developer just needs to add two parameters to Hadoop's configuration file:

- *Djava.library.path= location of aparapi native library*
- *mapred.map.tasks=1*

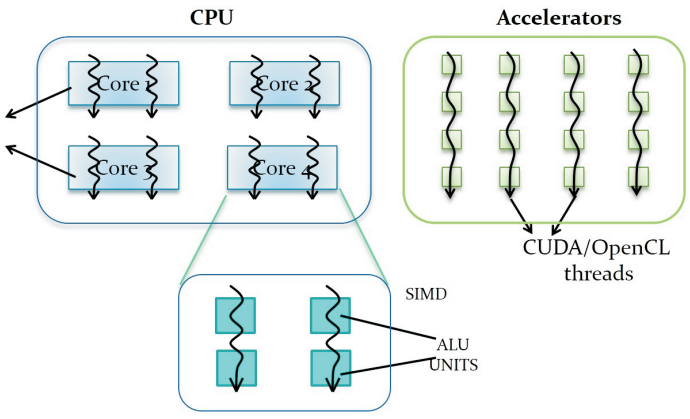

Fig 1: Typical Heterogeneous architecture comprising of CPU and GPUs

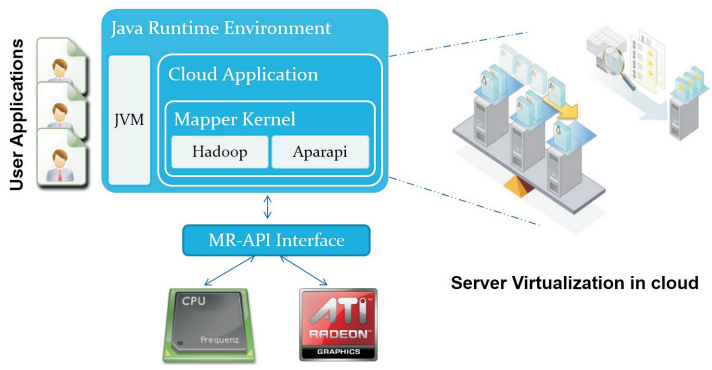

Fig 2: MR-API Software Architecture Framework

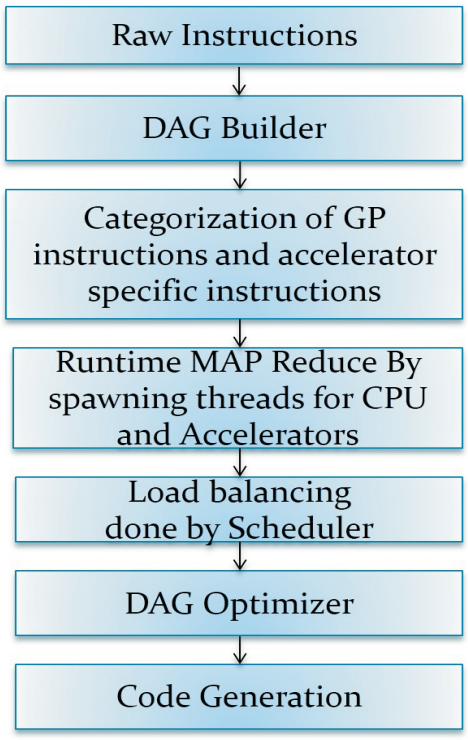

Fig 3: Compilation sequence in MR-API based heterogeneous architectures

```
public class Simple Mapper<K1 , V1 , K2 , V2>
implement s MapRunnabl e<K1 , V1 , K2 , V2> f

publ ic voi d map ( LongWr i tabl e offset,
    LongWr i tabl e size,
    Output Collector<Boolean Writable,
     LongWritable > out ,
     Reporterreporter) throws IOExceptionf
     long numInside = 0L ;
     long numOutside = 0L ;

finalPoint[]points=
    new Point[size.get()];

for(inti=0;i<size.get();i++)f
    points[i]=new Point(Math.random(),
       Math.random());
    finalfloatx=points[i].x 0.5f;
    finalfloaty=points[i].y 0.5f;
    if(x x+y y>0.25)f
       numOut side ++;
    } else {
    numInside ++;
  }
}

out.collect(new Boolean Writable(true),
    new LongWritable (numInside ));
out.collect(new Boolean Writable(false),
    new LongWritable (numOutside ));
}

public voi d run ( Recor dReader<K1 , V1> input,
    Output Collector <K2 , V2> output,
    Reporterreporter) throws IOExceptionf

    //allocatekey&valueinstances
    //thatarereusedforallentries
    K1 key = input.createKey();
    V1 value=input.createValue();

    while(input.next(key,value))f
    // map pair to output
     map((LongWritable) key,
     (LongWritable) value,
     (OutputCollector<Boolean Writable,
      LongWritable>)output,reporter);
  }
 }
}
```

Fig 4: Original MAP-REDUCE Program

```
1 public class GPUMapper extends
2 //thefirsttwotypeareinputkey/valuepair
3 //thelasttwotypeareoutputkey/valuepair
4 Mapper Kernel2Float ToBoolean<LongWritable, LongWritable,
5 Boolean Writable, LongWritable> {
6
7 @Override
8 public List<FloatTuple2> preprocess(
9    RecordReader<LongWritable, LongWritable> input,
10   Reporterreporter) throws IOException {
11
12   LongWritable key = input.createKey();
13   LongWritable value = input.createValue();
14   ArrayList<FloatTuple2> allGpuIn
15     = new ArrayList<FloatTuple2>();
16
17 while(input.next(key,value)){
18 for(inti=0;i<value.get() key.get();
19      i++)
20      allGpuIn.add(new FloatTuple2(
21      (float) Math.random(),
22      (float) Math.random()));
23   }
24   returnallGpuIn;
25 }
26
27
28 public boolean gpu(floatx,floaty){
29 returnx x+y y>0.25f;
30 }
31
32
33 @Override
34 public void postprocess(List<Boolean> gpuOut,
35   OutputCollector<Boolean Writable, LongWritable>
36   output)
37   throws IOException {
38    intnumInside = 0;
39    intnumOutside = 0;
40    for(boolean x:gpuOut)
41      if(x)
42         numInside ++;
43      else
44         numOutside ++;
45 output.collect(new Boolean Writable(true),
46      new LongWritable(numInside));
47 output.collect(new Boolean Writable(false),
48      new LongWritable(numOutside));
49   }
50 }
```

Fig 5: Modified MAP-REDUCE Program with **MR-API** Mapper Modules

Sample Results

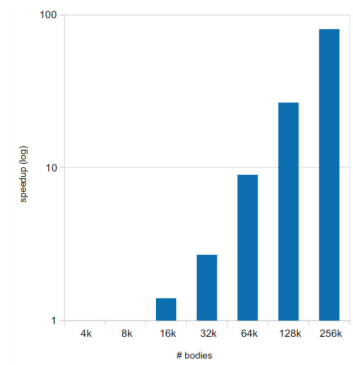

Fig 6: Speed up of Nbody simulation MR-API over CPU

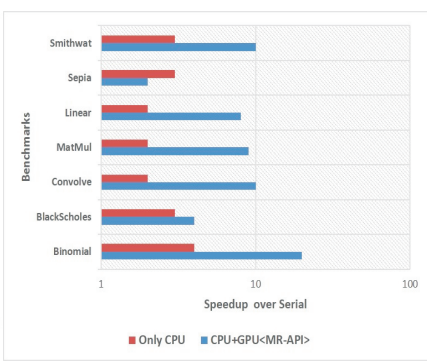

Fig 7: Speed up of other benchmarks MR-API over CPU

Future work

- Evolving MR-API framework to support multiple languages apart from Java
- Developing libraries to support varied set of accelerators including **CUDA** powered Nvdia GPUs
- Complete automation of mapping framework to support hybrid execution without programmer intervention