HeteroSpark: A Heterogeneous CPU/GPU Spark Platform for Machine Learning Algorithms

Peilong Li, Yan Luo, Yu Cao, Ning Zhang
University of Massachusetts Lowell
Background

- GPU outperforms CPU in a broad area of applications:
  - Machine learning, image processing, bioinformatics, etc.

- Pros and cons of current solutions:

<table>
<thead>
<tr>
<th>Current Solutions</th>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single GPU</td>
<td>Good data parallelism</td>
<td>Difficult to handle large scale dataset due to memory size</td>
</tr>
<tr>
<td>GPU Cluster</td>
<td>Good data parallelism</td>
<td>Complexity in data partitioning (MPI, OpenMP)</td>
</tr>
<tr>
<td>CPU Cluster</td>
<td>Scalability, programmability</td>
<td>Single node performance due to limited number of cores</td>
</tr>
</tbody>
</table>
Motivations

- **Acceleration**: Integrate GPU accelerators into current Spark platform to achieve further data parallelism and algorithm acceleration.

- **Plug-n-play**: “Plugin” style design – current Spark applications can choose to enable/disable GPU acceleration.

- **Portability**: Existing Spark code can be easily ported to the heterogeneous platform.
HeteroSpark Architecture

- Enable/disable GPU accelerators in configuration file
- Three configs:
  - Local GPU
  - Remote GPU
  - No GPU
CPU-GPU Communication

- Java Remote Method Invocation: methods of remote Java objects can be invoked from other Java virtual machines (on different hosts)
- RMI uses object serialization to marshal and unmarshal parameters
HeteroSpark Glue Logic

Call myFunc on Spark worker

- GPU Acc online & free?
  - Yes: Check myFunc on RMI registry
    - Yes: Original implementation
    - No: GPU computation
  - No: Send serialized data through RMI to remote method

- myFunc exist?
  - Yes: Send serialized data back to Spark worker and set GPU Acc free
  - No: Set GPU Acc busy

GPU computation

RMI server gets data, deserialized and send to GPU implementation
GPU Accelerator Development

- Accelerate your application:
  - Use HeteroSpark existing libraries.
  - Develop HeteroSpark accelerator libs.

Existing GPU library or source code

Java Class

File sharedLibrary = new File("FuncJNI.so");
public native int func(float[] para1, int para2 ...);
System.load(sharedLibrary.getAbsolutePath());
jdouble *para1 = (*env)->GetFloatArrayElements(env, array1, 0);
funcArray = obj.func(para1, para2 ...);
// Return data back to Java env
(*env)->ReleaseFloatArrayElements(env, array1, para1, 0);
int retArray = obj.func(para1, para2...);

• Accelerate your application:
  - Use HeteroSpark existing libraries.
  - Develop HeteroSpark accelerator libs.
## Why RMI?

<table>
<thead>
<tr>
<th>Solutions</th>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>RMI</td>
<td>Secure, faster, lightweight</td>
<td>Java-specific</td>
</tr>
<tr>
<td>COBRA</td>
<td>Language independent</td>
<td>No GC supported</td>
</tr>
<tr>
<td>SOAP</td>
<td>XML-based web service</td>
<td>Heavy and slow</td>
</tr>
<tr>
<td>Spring+JMS</td>
<td>Message queuing; simple programming</td>
<td>Spring framework learning</td>
</tr>
<tr>
<td></td>
<td>interface</td>
<td>curve, extra dependency</td>
</tr>
</tbody>
</table>

- RMI overhead is < 500 ms with 2MB of data if connected locally.
- In most cases, we will use local GPU which is connected via PICe.
Performance Evaluation

- Benchmark applications:
  - Logistic regression
  - Dataset: Criteo (click prediction task), training 11.15 GB, test 1.46 GB
  - K-Means
  - Dataset: MNIST-8M, (handwritten digits), 8.1 M data, 24.8 GB

- System setup:
  - CPU: EC2 m3.xlarge nodes
  - GPU: EC2 g2.2xlarge

![Performance Comparison Chart]

Spark 32 cores | Spark 64 cores | Spark 128 cores | HeteroSpark 8 cores/2 GPU | HeteroSpark 32 cores/8 GPU
1.001.00 | 2.092.71 | 4.606.25 | 4.333.66 | 18.5716.41
Conclusion

• **Acceleration**: HeteroSpark enhances Spark by accelerating machine learning algorithms and reducing CPU resources.

• **Plug-n-play**: zero interference with original application if choose to “mute” acceleration.

• **Portability**: non-tedious work to port existing Spark application into HeteroSpark (if using the maintained libraries).
Future Work

- **Serialization Overhead**: Utilize faster serialization technique for communication.
- **Simplified Interface**: Use Spring framework to simplify remote method innovation interface.
- **Spawning the Library**: Integrating more machine learning libraries into HeteroSpark, esp. deep learning algorithms.
Thank you!