FatMan vs. LittleBoy: Scaling up Linear Algebraic Operations in Scale-out Data Platforms

Luna Xu (Virginia Tech)
Seung-Hwan Lim (ORNL)
Ali R. Butt (Virginia Tech)
Sreenivas R. Sukumar (ORNL)
Ramakrishnan Kannan (ORNL)
HPC is used to enable scientific discovery

Scientific Simulation/Observation

Data Analysis

Scientific Discovery
Increasing role of data in HPC

Efficient big data processing → Faster HPC

- Scale of data to be analyzed is growing exponentially

- **Observation**: Data analysis in HPC is similar to data analysis in big data community

- Big data scale-out platforms can benefit data-based scientific discovery
Scale-out data processing is becoming popular
Problem: Unable to leverage accelerators
Per node performance does not scale

Sources: https://www.nextplatform.com/2015/07/13/top-500-supercomputer-list-reflects-shifting-state-of-global-hpc-trends/; Nvidia
Per node performance does not scale

ACCELERATORS/CO-PROCESSORS

Sources: https://www.nextplatform.com/2015/07/13/top-500-supercomputer-list-reflects-shifting-state-of-global-hpc-trends/; Nvidia
Our contribution: Scaling up linear algebra in scale-out data platforms

• Introduce the support for distributed dense matrix manipulations in Spark for scale-out matrix operations

• Adopt scale-up hardware acceleration for BLAS-3 operations of distributed matrices

• Design a flexible controller to decide when and whether to use hardware accelerators based on the density of matrices
Agenda

• Introduction
• Background
• Design
• Evaluation
• Conclusion
Distributed matrix support in Spark

Input file ➔ Row Matrix ➔ Indexed Row Matrix ➔ Coordinate Matrix ➔ Block Matrix

- RDD
- Sparse Matrix
Distributed matrix support in Spark

Dependency between internal components in MLlib treats all matrices as sparse matrix regardless of density
Distributed matrix support in Spark

- Input file
- Row Matrix
- Indexed Row Matrix
- Coordinate Matrix

- Ad-hoc Scala Impl

- RDD
- Sparse Matrix

- Block Matrix
Distributed matrix support in Spark

Ad-hoc implementation of sparse matrix multiplication prevents use of hardware acceleration

Ad-hoc Scala Impl
Agenda

• Introduction
• Background
• **Design**
• Evaluation
• Conclusion
Design considerations

• Enable user transparency
• Support scalable matrix multiplication
• Support dense and sparse matrices
System architecture

matrix multiplication → MLlib

Spark cluster

SELECTOR

Scala Impl

BLAS enabler

netlib-java (JNI)

Open BLAS

NV BLAS

cuBLAS

cuBLAS^xt

worker

JVM

native

CPU

GPU

GPU

GPU
Agenda

• Introduction
• Background
• Design
• Evaluation
• Conclusion
Methodology

Gramian matrix computation with GEMM

- $XX^T$
- Machine learning (PCA, SVD)
- Data analysis (all-pair similarity)
Methodology

Gramian matrix computation with GEMM

- $XX^T$

<table>
<thead>
<tr>
<th># of Rows (Cols)</th>
<th>Density</th>
<th>Raw size (GB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4873</td>
<td>1</td>
<td>0.34</td>
</tr>
<tr>
<td>14684</td>
<td>1</td>
<td>3.1</td>
</tr>
<tr>
<td>24495</td>
<td>1</td>
<td>8.4</td>
</tr>
<tr>
<td>663331</td>
<td>1</td>
<td>77</td>
</tr>
<tr>
<td>4873</td>
<td>0.05</td>
<td>0.104</td>
</tr>
<tr>
<td>14684</td>
<td>0.05</td>
<td>2.6</td>
</tr>
<tr>
<td>24495</td>
<td>0.05</td>
<td>19</td>
</tr>
<tr>
<td>663331</td>
<td>0.05</td>
<td>41</td>
</tr>
</tbody>
</table>
Methodology

• System spec:

<table>
<thead>
<tr>
<th>System</th>
<th>Rhea GPU node</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>Dual Intel Xeon E5</td>
</tr>
<tr>
<td>CPU cores</td>
<td>28 (56 HT)</td>
</tr>
<tr>
<td>Memory</td>
<td>1TB</td>
</tr>
<tr>
<td>GPU</td>
<td>Dual NVIDIA K80</td>
</tr>
<tr>
<td>GPU cores</td>
<td>4992 x 2</td>
</tr>
<tr>
<td>GPU memory</td>
<td>24 x 2 GB</td>
</tr>
<tr>
<td>CUDA</td>
<td>7.5</td>
</tr>
</tbody>
</table>

• Spark configuration:
  • Version: 1.6.1
  • 2 node cluster
  • One executor per node (56 cores, 800GB)

• BLAS configuration
  • OpenBLAS v0.2.19 (hand compile)
  • NVBLAS v7.5
Overall performance: Dense Matrix

![Graph showing speedup for different matrix sizes.](image)

- **OpenBLAS**
- **NVBLAS**

- Speedup increases with matrix size.
- Baseline speedup is indicated.
- OpenBLAS shows a 2.2x speedup compared to baseline.
- NVBLAS shows a 1.5x speedup compared to baseline.
Performance: Dense Matrix

The chart above shows the time percentage for different matrix sizes and libraries: MLlib, OpenBLAS, and NVBLAS. The y-axis represents the time percentage, while the x-axis indicates the matrix size in thousands (4873, 14684, 24495, and 66331).

- **GC**: Green bars represent the garbage collection component.
- **Shuffle**: Red bars denote the shuffle component.
- **Compute**: Yellow bars indicate the compute component.
- **Others**: Blue bars show other components.

The chart highlights that for the largest matrix size (66331), the compute component (MLlib and OpenBLAS) dominates, with values around 85.2% of the total time.
Performance: Dense Matrix

![Graph showing performance metrics for different matrix sizes and libraries. The graph compares GC, Shuffle, Compute, and Others time percentage for MLlib and OpenBLAS with matrix sizes 4873, 14684, 24495, and 66331. The highlighted bars indicate improvements with NVBLAS, showing 92.9% and 96.1% improvements respectively.](image-url)
Overall performance: Sparse Matrix

- **Speedup**
  - Matrix size:
    - 4873
    - 24495
    - 66331
    - 97708

Graph showing:
- OpenBLAS
- NVBLAS

Baseline with 10% improvement.
Overall performance: Sparse Matrix

![Graph showing speedup vs matrix size for OpenBLAS and NVBLAS. The graph indicates a 36.7% speedup at a matrix size of 97708.]
Performance: Sparse Matrix

- **GC**
- **Shuffle**
- **Compute**
- **Others**

<table>
<thead>
<tr>
<th>Matrix size</th>
<th>Time percentage (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4873 (MLlib, OpenBLAS)</td>
<td>22%</td>
</tr>
<tr>
<td>14684 (MLlib, OpenBLAS)</td>
<td>85.1%</td>
</tr>
<tr>
<td>24495 (MLlib, OpenBLAS)</td>
<td></td>
</tr>
<tr>
<td>66331 (MLlib, OpenBLAS)</td>
<td></td>
</tr>
</tbody>
</table>

VirginiaTech
Invent the Future
Conclusion

• We employ scale-up accelerations for linear algebraic operations in Spark

• Our approach decides whether to use hardware accelerations based on matrix density

• The system improves overall performance up to more than 2x compared to default Spark

• Contact: Luna Xu, xuluna@cs.vt.edu
DSSL: http://dssl.cs.vt.edu/