MadLINQ: Large-Scale Distributed Matrix Computation for the Cloud
Zhengping Qian, Xiuwei Chen, Nanxi Kang, Mingcheng Chen, Yuan Yu, Thomas Moscibroda, Zheng Zhang

Philip Leonard

October 27, 2014
Motivation

- MapReduce and DryadLINQ relational algebra operators not suitable for linear algebra computations
- Demand for efficient matrix computations;
  - Machine learning
  - Graph algorithms (graphs boil down to sparse matrices)
- Previous attempts failed to deliver;
  - ScaLAPACK [2] too low level (MPI Knowledge required)
  - HAMA built on top of MapReduce (still restrictive)
Key Components of MadLINQ

- Simple programming model for matrix computation
- New Fine Grained Pipelining (FGP) model
- Fault tolerance for FGP
- Integration with DryadLINQ [3]
Tile Algorithms

- A tile is a sub-matrix.
- Entire matrix is partitioned into a grid of tiles.
- This idea is what gives rise to parallelism in matrix computation.
- Aim is to maximise cache localisation by exploiting the structured access of matrix algorithms.
Computation Example: Cholesky Decomposition

- Takes a symmetric positive-definite matrix
- Matrix is partitioned into tiles
- On the $k$-th iteration, tile operations employed to factorise;
  - diagonally (DPOTRF)
  - $n - k$ tiles below (DTRSM)
  - trailing tiles to the right (DSYRK and DGEMM)
Cholesky Decomposition Iteration
C# constructs, allows DryadLINQ and MadLINQ integration.

Matrix data abstraction in C# encapsulates tile representation.

Programs expressed in a sequential fashion.

Linear algebra library in C#
Example Application: Collaborative Filtering

- How to predict what other movies users will like given their rating of other movies.
- $R[i,j]$ is user $j$’s rating of movie $i$.

**CF Equation**

\[(R \cdot R^T) \cdot R\]

becomes;

**CF MadLINQ Code**

```csharp
Matrix similarity = R.Multiply(R.Transpose());
Matrix scores = similarity.Multiply(R).Normalize();
```

* Matrix goes from sparse (users haven’t seen most movies) to dense (every user has predicted score for every movie)
CF: Integration with DryadLINQ

- DryadLINQ processes Netflix dataset
- This boils down to a MadLINQ Matrix
- MadLINQ does transposition, matrix multiplication and normalisation of $R$ to get scores
- DryadLINQ generates top 5 list of movies for each user.
MadLINQ Architecture

- Central Scheduler (CS)
- Schedule
- Heartbeat (sync states)
- Heartbeat (sync states)
- Machine
  - Vertex
  - Local Daemon
  - Local Store
- Schedule
- Data push
Directed Acyclic Graph (DAG)

- DAG is dynamically expanded through symbolic execution to prevent explosion ($O(n^3)$ for Cholesky Decomposition)

- $f_1$ through $f_4$ are the tile operators discussed earlier (DPOTRF, DTRSM, DSYRK and DGEMM resp.)
FGP & Fault Tolerance

- Parallelism fluctuates with matrix computations
- Pipelining exploits vertex parallelism by increasing data granularity (recursively tiling matrices)

- Failure handling: Input blocks can be reconstructed from output blocks.
- Dependencies are calculated to reduce recovery cost.
Optimisations & Configuration

From the authors experience, optimisations were made;

- Prefetching of vertex data for a close to terminating node
- Specifying order of matrix data (column or row first?)
- Auto switching between sparse (compressed) and dense matrices.

Configuration;

- Smaller tiles $\Rightarrow$ higher parallelism
- Granularity of computation is a block
- Block size determined by number of non-zero elements
Observations & Applications

- Observations;
  - Pipelining performs better on larger problems
  - Pipeline approach on average 14.4% faster than ScaLAPACK
  - ScaLAPACK failed consistently using 32 cores (with no fault tolerance)

- Real world applications;
  - MadLINQ more efficient than MapReduce
  - For Collaborative Filtering (recall $(R \cdot R^T) \cdot R$) on 20k × 500k matrix (Netflix challenge). Mahout over Hadoop took over 800 minutes, as opposed to MadLINQ 16 (albeit Mahout produces results of higher accuracy)
Conclusion & Related Work

- DAGuE, a similar use of DAG for tiled algorithms but failed to provide fault tolerance and resource dynamics

- Future research ideas:
  - Auto-Tiling of matrices for matrix algorithms
  - Dynamic Re-Tiling (dynamic changing of tile sizes for graph algorithms)
  - Sparse matrices cause load imbalance. Methods required for handling these well.

- Concludes MadLINQ fills the void that is large scale distributed matrix and graph processing, using linear algebra.
Zhengping Qian, Xiuwei Chen, Nanxi Kang, Mingcheng Chen, Yuan Yu, Thomas Moscibroda, Zheng Zhang
MadLINQ: Large-Scale Distributed Matrix Computation for the Cloud

Choi, J., Dongarra, J., Pozo, R., Walker, D.
Scalapack: A scalable linear algebra library for distributed memory concurrent computers.

Yu, Y., Isard, M., Fetterly, D., Et Al.
DryadLINQ: A system for general-purpose distributed data-parallel computing using a high-level language.
In OSDI, 2008.
Questions