Large-Scale GPU programming

Tim Kaldewey

IBM

Research Staff Member
Database Technologies
IBM Almaden Research Center
tkaldew@us.ibm.com

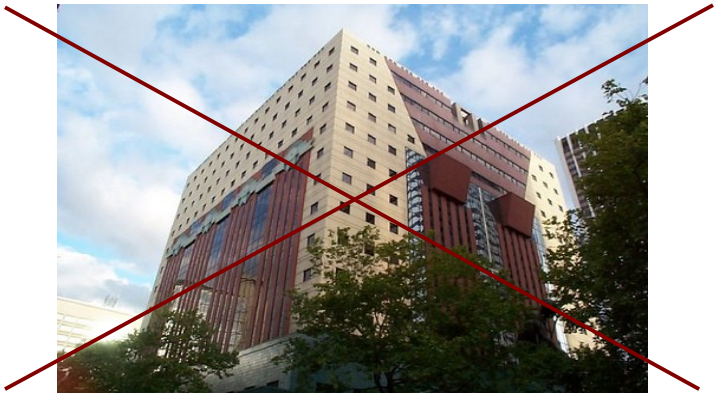
Penn Engineering

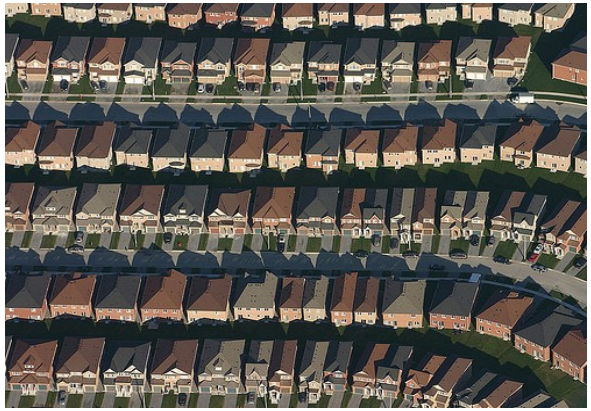
Assistant Adjunct Professor
Computer and Information Science Dept.
University of Pennsylvania
tim@kaldewey.com

Agenda

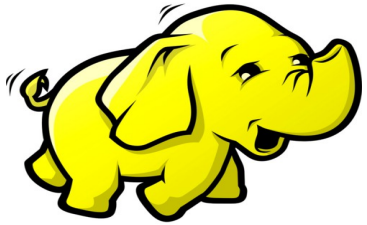
- Scale-up vs. scale-out
- GPU & scale-out (MPI)
- Introduction to MapReduce
- MapReduce & GPU
  - Building a GPU MapReduce framework (UC Davis)
  - Hadoop Pipes (Tokyo Institute of Tech.)
  - Hadoop streams (UIUC)

Scale-up vs. Scale-out





Scale-up vs. Scale-out

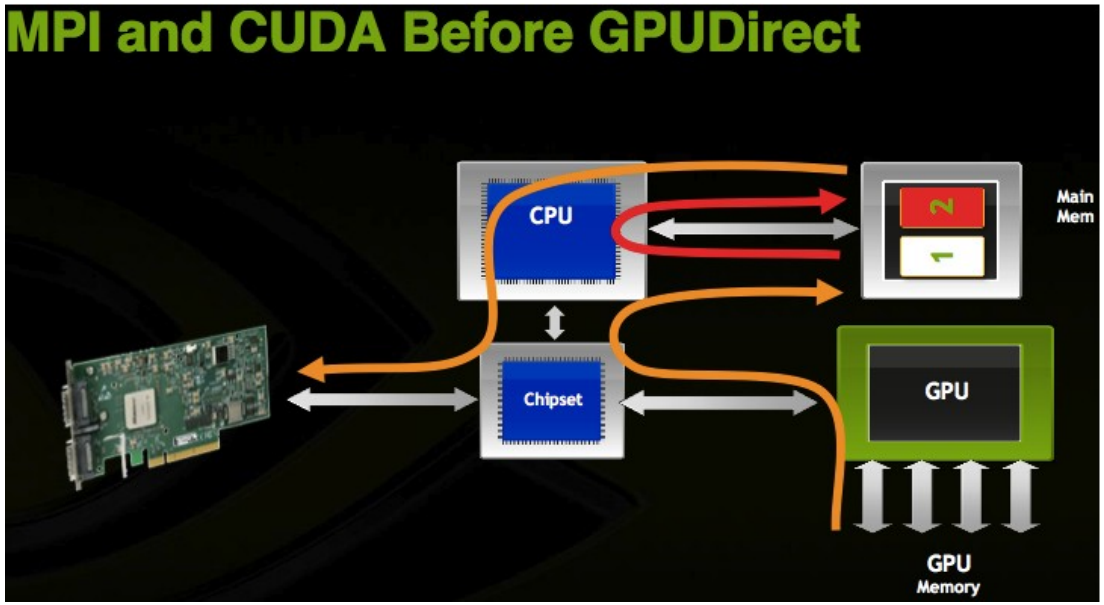


How to leverage GPUs in **scale-out** architectures

- On foot
  - Manage your own network connectivity, communication, data placement & movement, load balancing, fault tolerance, …
- MPI (Message passing interface)
  - Focus on communication (RDMA support), still have to manage data handling, load balancing, fault tolerance, ...

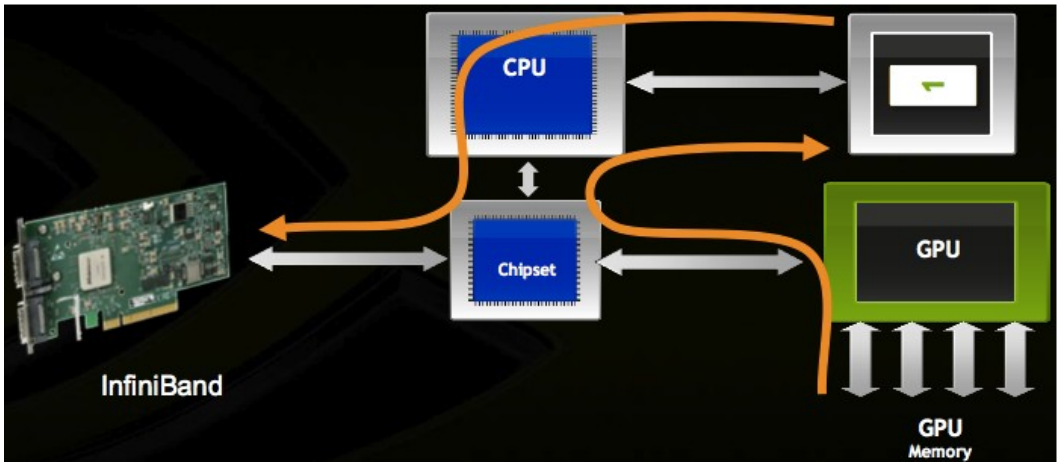
How to leverage GPUs in **scale-out** architectures

- On foot
  - Manage your own network connectivity, communication, data placement & movement, load balancing, fault tolerance, …
- MPI (Message passing interface)
  - Focus on communication (RDMA support), still have to manage data handling, load balancing, fault tolerance, ...
  - GPUDirect (nVidia & Mellanox)



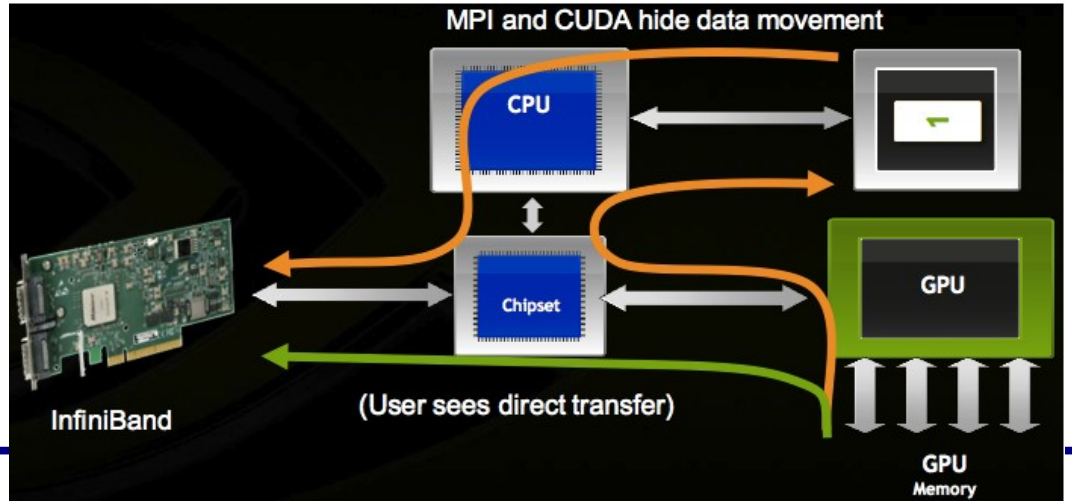
How to leverage GPUs in **scale-out** architectures

- On foot
  - Manage your own network connectivity, communication, data placement & movement, load balancing, fault tolerance, …
- MPI (Message passing interface)
  - Focus on communication (RDMA support), still have to manage data handling, load balancing, fault tolerance, ...
  - GPUDirect (nVidia & Mellanox)
    - Infiniband card and GPU share DMA/RDMA region



How to leverage GPUs in **scale-out** architectures

- On foot
  - Manage your own network connectivity, communication, data placement & movement, load balancing, fault tolerance, …
- MPI (Message passing interface)
  - Focus on communication (RDMA support), still have to manage data handling, load balancing, fault tolerance, ...
  - GPUDirect (nVidia & Mellanox)
    - Iinfiniband card and GPU share DMA/RDMA region
    - v2 uses UVA to make remote memory access transparent



Agenda

- Scale-up vs. scale-out
- GPU & scale-out (MPI)
- Introduction to MapReduce
- MapReduce & GPU
  - Building a GPU MapReduce framework (UC Davis)
  - Hadoop Pipes (Tokyo Institute of Tech.)
  - Hadoop streams (UIUC)

MapReduce

- Programming Model for Large-Volume Data Processing
- Specialized for frequent use case: aggregation queries
  - Map every input object to set of key/value pairs
  - Reduce (aggregate) all mapped values for same key into one result for that key

MapReduce

- Programming Model for Large-Volume Data Processing
- Specialized for frequent use case: aggregation queries
  - Map every input object to set of key/value pairs
  - Reduce (aggregate) all mapped values for same key into one result for that key

- Use this structure as explicit API for cluster computing
  - Submit jobs as (input data, map function, reduce function)
  - Implementation details for given architecture can be hidden
  - No need to manually implement parallelization, scheduling, fault tolerance, ...
  - Dynamic resource allocation by MR framework without code changes

Example – Word Frequency

```
map(String key, String value):
// key: document name
// value: document contents
for each word w in value:
  EmitIntermediate(w, "1");
```

```
reduce(String key, Iterator values):
// key: a word
// values: a list of counts int result = 0;
for each v in values:
  result += ParseInt(v);
  Emit(AsString(result));
```

More Examples

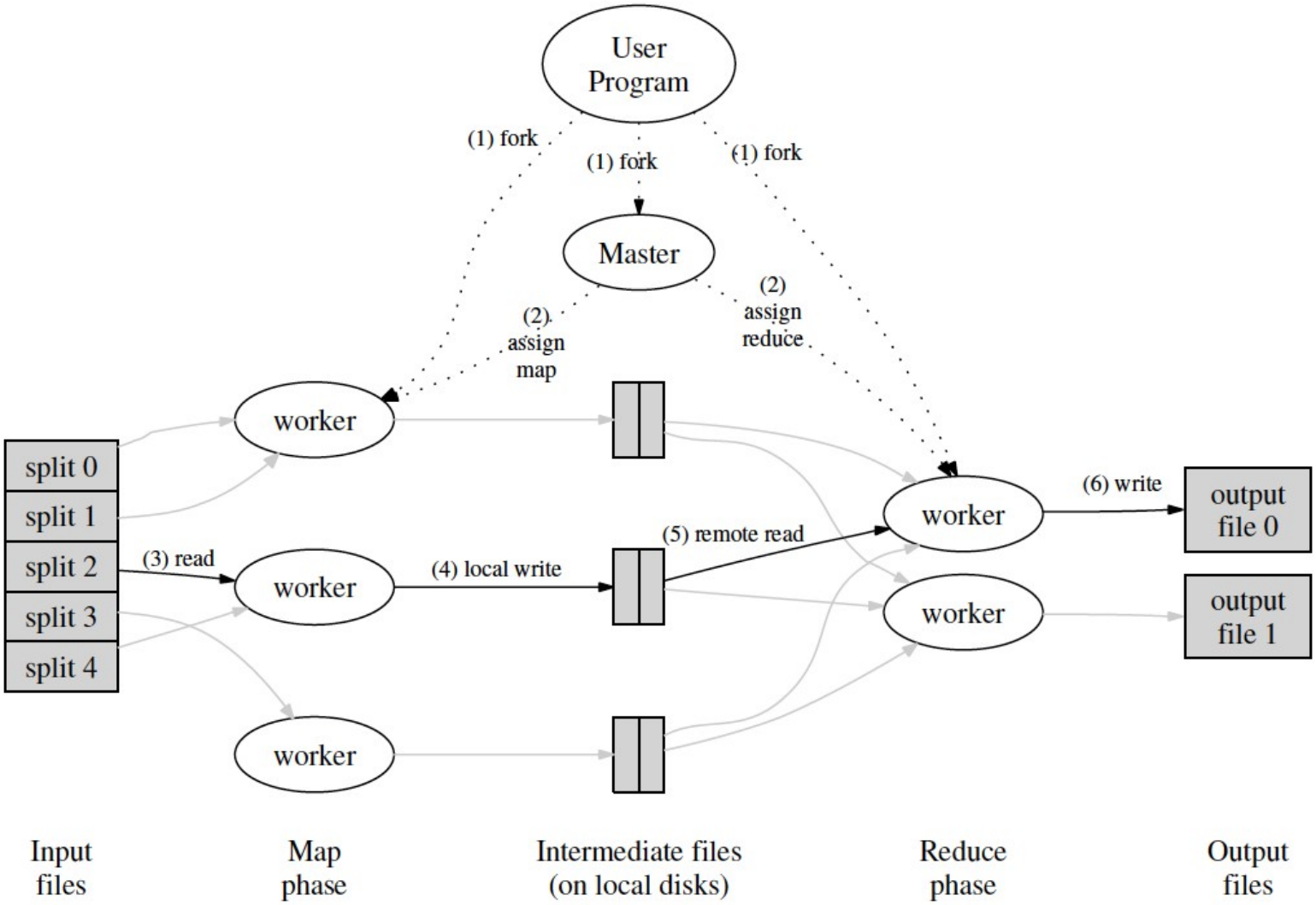
- Grep
  - Input: (many) text files
  - Map: Emit line if matches
  - Reduce: Concatenate intermediate results

- URL Access Frequency
  - Input: Web server logs
  - Map: <URL,1>
  - Reduce: Add values for URL

MapReduce Environment

- Commodity PC Machines
  - Commodity CPU, Network, Storage
  - Distributed File System (Replicated)
- Map/Reduce is framework (library)
- First described by Google
  - Dean, Ghemawat: MapReduce: Simplified Data Processing on Large Clusters, OSDI 2004.
  - Implemented in C++

Google Implementation



Fault Tolerance

- Master Data:
  - For each task
    - State: Complete / Idle / In-progress
    - Assigned worker (for non-idle tasks)
  - For each map task
    - size and location of intermediate results for each reduce job

Fault Tolerance

- Master Data:
  - For each task
    - State: Complete / Idle / In-progress
    - Assigned worker (for non-idle tasks)
  - For each map task
    - size and location of intermediate results for each reduce job

- Periodic Ping to Worker by Master
    - Reply includes status information on pending tasks
    - No reply:
      - Reschedule all its incomplete tasks
      - Also reschedule completed map tasks of that worker, results are inaccessible
      - Completed reduce tasks available in distributed file system

Fault Tolerance cont.

- Slow machines can delay whole job
  - faulty hard disks, faulty configuration
  - answer to pings, but do not complete
- Near end of job
  - process in-progress tasks redundantly
  - remaining in-progress jobs likely to be “stragglers”
  - redundant execution will overtake them
  - if inherently slow, original execution finishes first
  - typical results: 2x faster overall execution time

Fault Tolerance cont.

- Slow machines can delay whole job
  - faulty hard disks, faulty configuration
  - answer to pings, but do not complete
- Near end of job
  - process in-progress tasks redundantly
  - remaining in-progress jobs likely to be “stragglers”
  - redundant execution will overtake them
  - if inherently slow, original execution finishes first
  - typical results: 2x faster overall execution time
- What happens if the master crashes
  - Re-execute entire job
  - Unlikely as there is only 1 master per job
  - Checkpointing possible to reclaim partial results

MapReduce Scheduling

- Locality
  - Input Locality
    - Try to execute map jobs where data already is in DFS
  - Output Locality
    - User may specify reduce key-partitioning function
    - Co-locates related output in same reducers

MapReduce Scheduling

- Locality
  - Input Locality
    - Try to execute map jobs where data already is in DFS
  - Output Locality
    - User may specify reduce key-partitioning function
    - Co-locates related output in same reducers

- Combiner
  - For commutative and associative reducer functions
  - “Pre-aggregation”: Aggregate in several stages
  - Reduces network traffic
  - Combiner tasks
    - code typically same as reduce
    - output is fed to other reduce/combine functions, not output files

MapReduce Variants – Related Products

- Hadoop (Yahoo, Apache)
  - Open-source Java implementation of Map/Reduce
- Sawzall (Google)
  - Domain-specific programming language (DSL) for Map stage
- Pig Latin (Yahoo)
  - DSL for large-volume processing jobs
  - can be compiled into Map/Reduce jobs
- Hive (Facebook)
  - Data warehouse
  - SQL-like interface

- Amazon EC2 (Elastic Cloud Computing)
  - Rent machines by the hour, can be preloaded with Hadoop server images
  - Nodes equipped with 2 Tesla M2050 available for $2.60/hr ...

Agenda

- Scale-up vs. scale-out
- GPU & scale-out (MPI)
- Introduction to MapReduce
- MapReduce & GPU
  - Building a GPU MapReduce framework (UC Davis)
  - Hadoop Pipes (Tokyo Institute of Tech.)
  - Hadoop streams (UIUC)

MapReduce & GPU

- MapReduce frameworks promise to sweeten the bitter taste of distributed programming by handling system tasks:
  - communication, data placement & movement, load balancing, fault tolerance, …
- The de-facto standard

  - Implemented in Java, but
    - Can handle binaries

MapReduce & GPU

- Mars
  - frameworks for single CPU/GPU
- GPMR
  - C++ MR implementation specifically for GPU clusters
- Extending Hadoop MR to GPUs
  - Tokyo Institute of Technology
  - MITHRA

GPMR

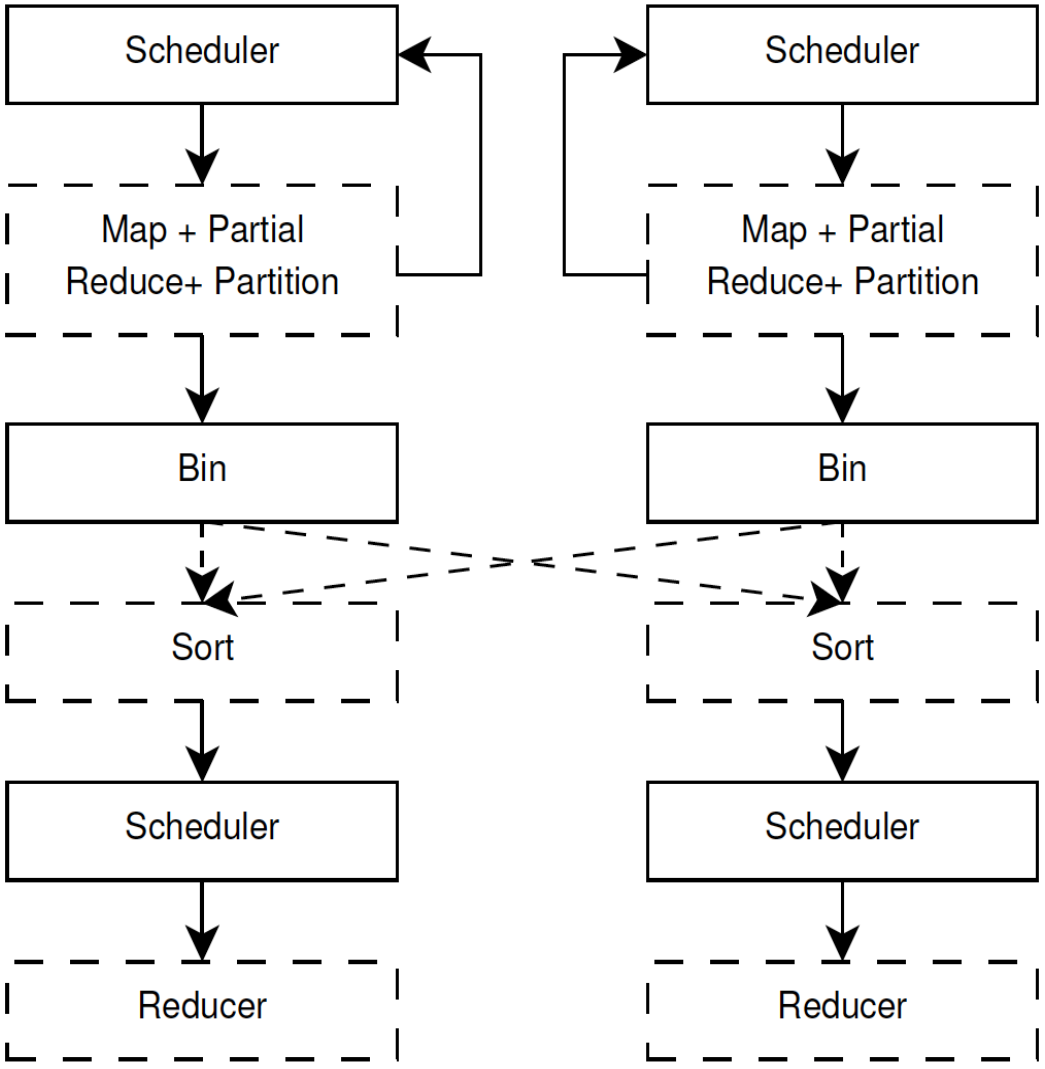
- A standalone MR library implemented in C/C++
  - Allows native use of CUDA
- Starting with a naïve implementation:
  - Copy input data to GPU
  - Map, sort, reduce → kernel calls
  - Copy data back
- What happens if the the amount of data > GPU memory?
- How to scale across nodes?

  - Partitioning is required
  - Optimizations:
    - Relax 1 thread to 1 work-item constraint
    - Use combiners for all map jobs of a node

GPMR

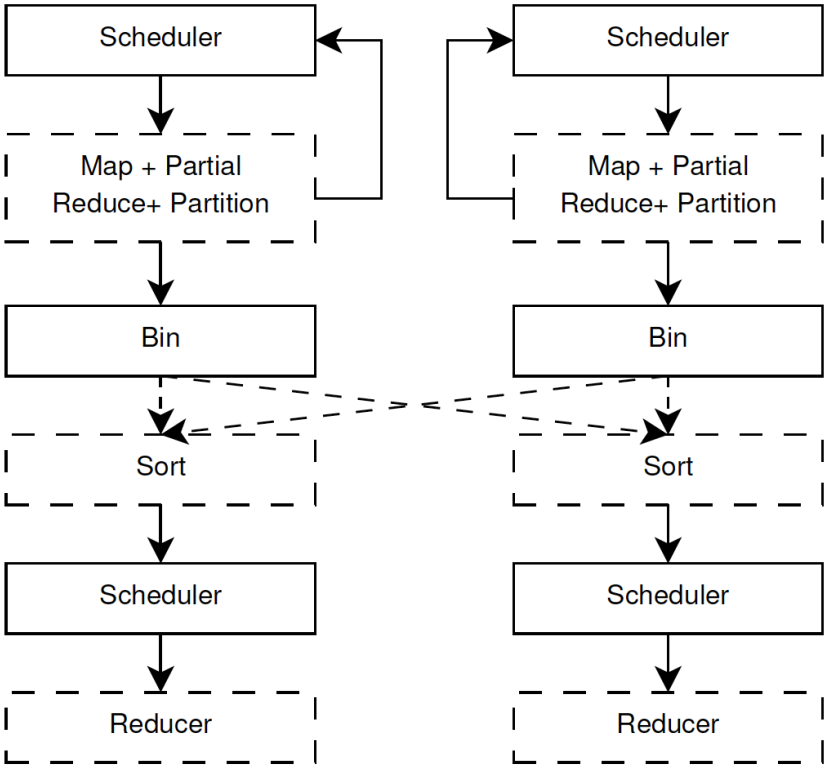
Phases

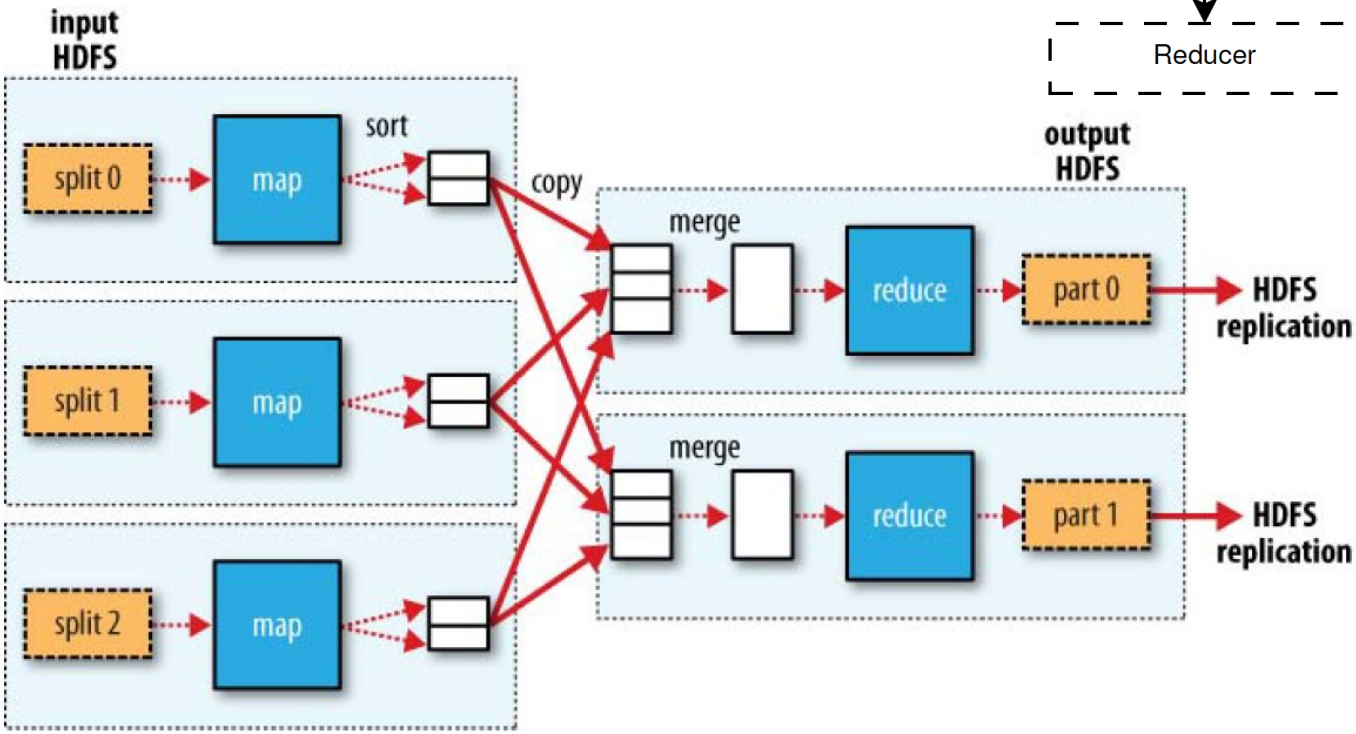
- Scheduler
  = Master
- GPU map
  = (CPU map) - sort
- GPU Partial Reduce
  = Combiner
- Partition defaults to RR
- Bin = Data tx
- Sort potentially executed on different node
- GPU Reduce = CPU reduce



MR vs. GPMR

- Confused ?
  - Many new stages
  - All stages open to the programmer ...
- Wasn't MapReduce designed to be simple ?

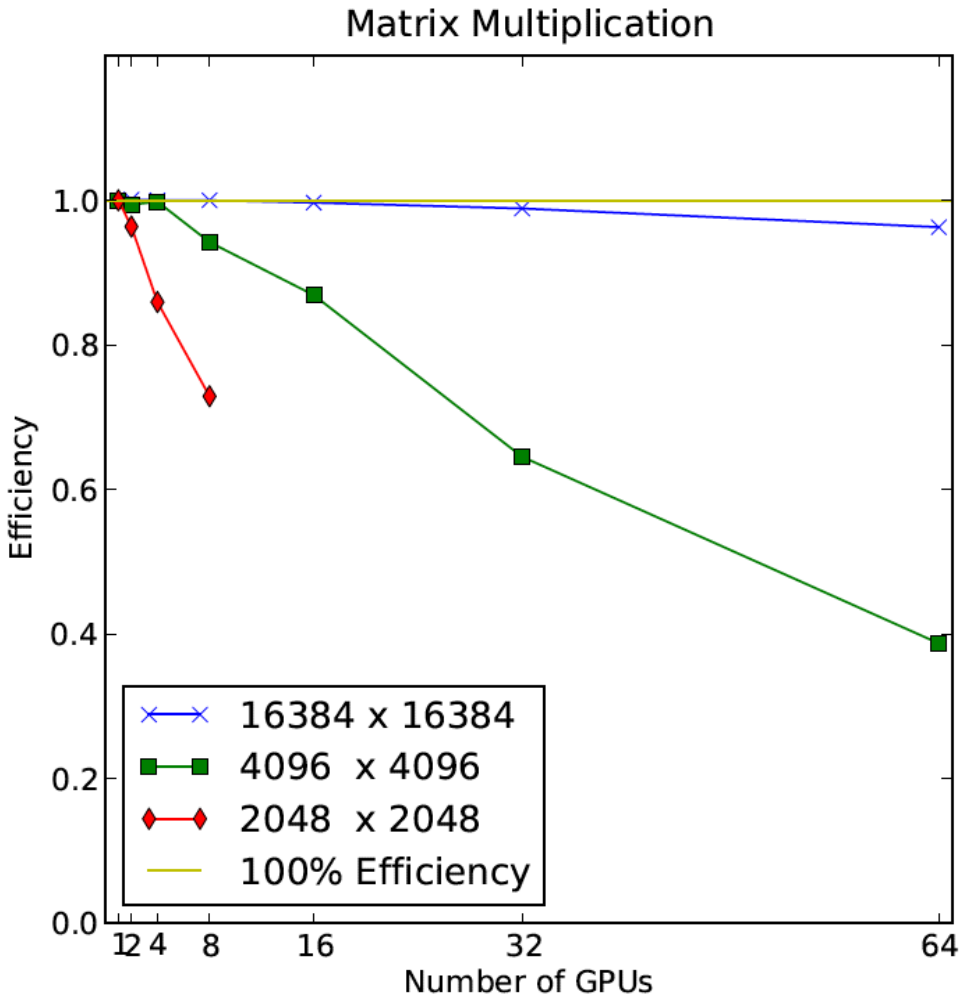




GPMR - Evaluation

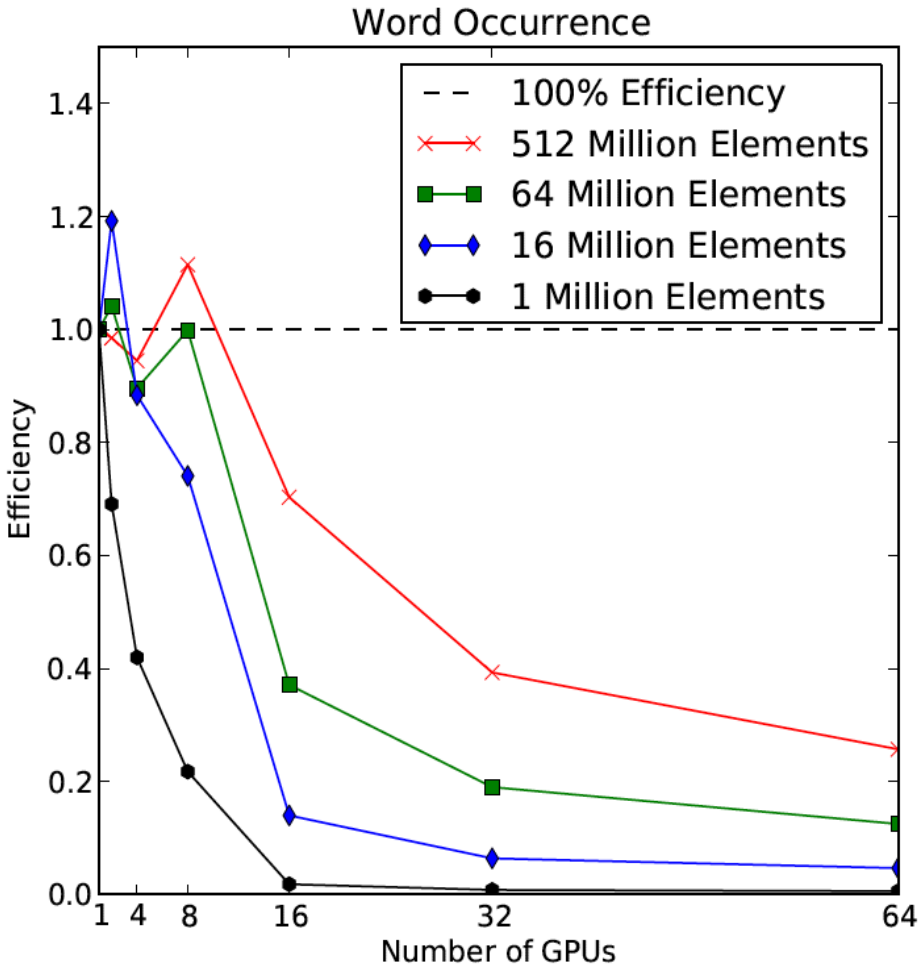
- 5 “common” benchmarks for MR clusters
  - Matrix multiply
  - Sparse integer occurrence
  - Word occurrence
  - K-Means clustering
  - Linear Regression
- Demonstrate scalability
- Comparison with existing MR libraries
  - Mars, single GPU
  - Phoenix, multi-core CPU

GPMR – Results



- C=A+B, use blocking to partition matrix
- C_block(1,1)=A_block(1,1)*B_block(1,1) + A_block(1,2)*B_block(2,1) + ...
- Each mapper processes a matrix multiplication for individual block(s)
- Reducer sums all sub-matrices.

GPMR – Results



- String processing inefficient and mismatch for GPU
  - Dictionary relatively small, 43k words
  - Use integer hashing
- Use combiners to accumulate results
  - Implemented using atomics

GPMR – Results

- Comparison with MARS running on a single GPU

	MM	KMC	WO
1-GPU Speedup	2.695	37.344	3.098
4-GPU Speedup	10.760	129.425	11.709

- Comparison with Phoenix running on 4 CPU cores

	MM	KMC	LR	SIO	WO
1-GPU	162.712	2.991	1.296	1.450	11.080
4-GPU	559.209	11.726	4.085	2.322	18.441

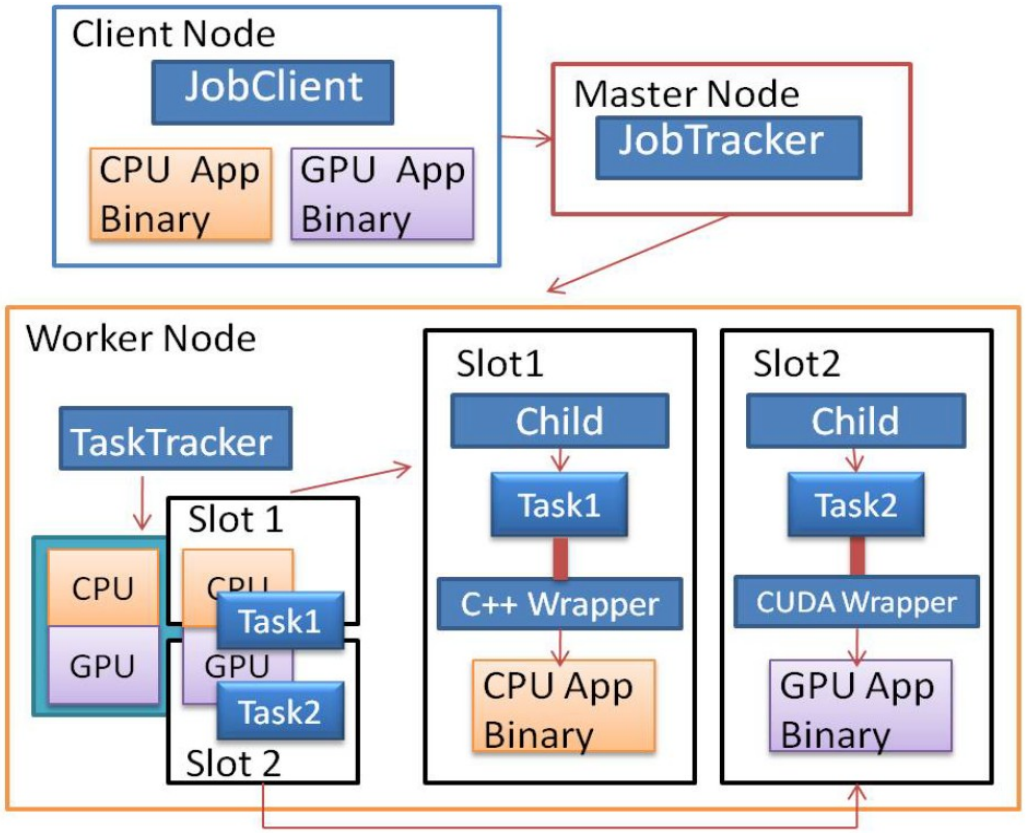
- Missing comparison with a “real” MR framework

Extending Hadoop MR

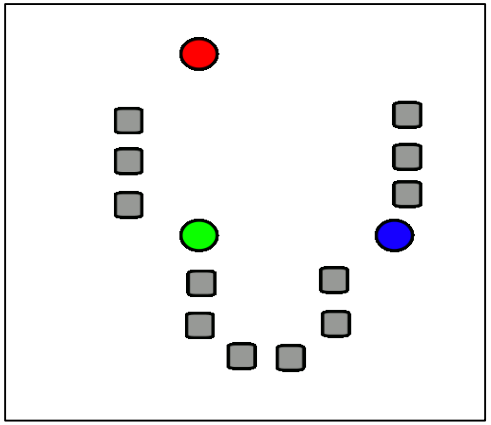
- Hadoop is implemented in Java
- Options to interface with CUDA
  - Hadoop Streaming – Unix streams with line-oriented or binary interface
    - Works for any binary, but requires input parsing
  - Hadoop Pipes – Unix sockets
    - C++ Hadoop library provides k-v abstractions
  - JNI
    - Allows to invoke external apps and libraries in C, asm, ...
    - Require platform specific libaries
    - Access to Java data structures through JNI slow

Tokyo Institure of Technology – Hadoop Pipes

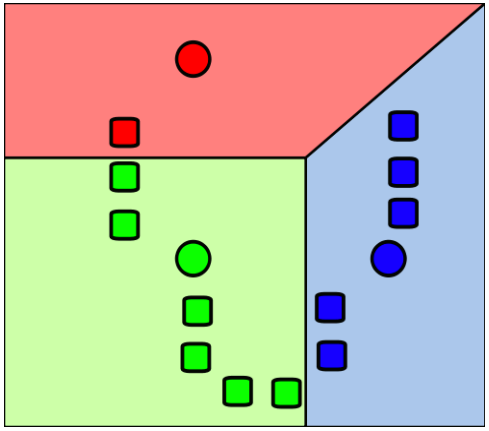
- Hybrid approach
  - CPU and GPU mappers
  - Scheduling not affected
  - Input/output formats unchanged
  - Transparent on which hardware Job runs



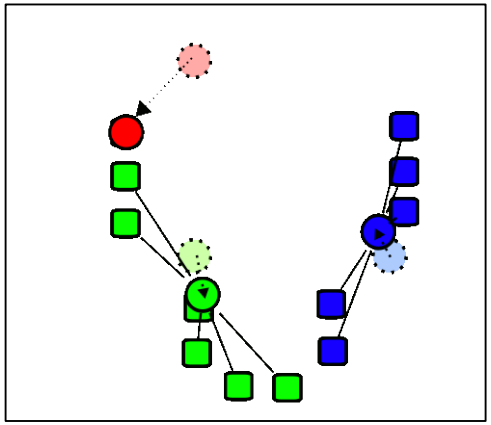
K-Means



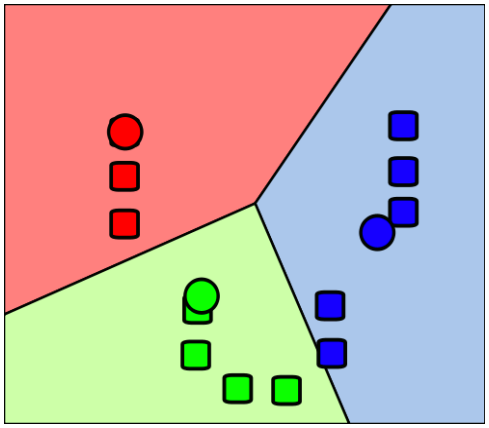
1) k initial "means" (k=3) randomly selected



2) k clusters by finding nearest mean



3) centroid of each clusters = new mean



4) Steps 2 and 3 are repeated until convergence, i.e. no more changes in assigments

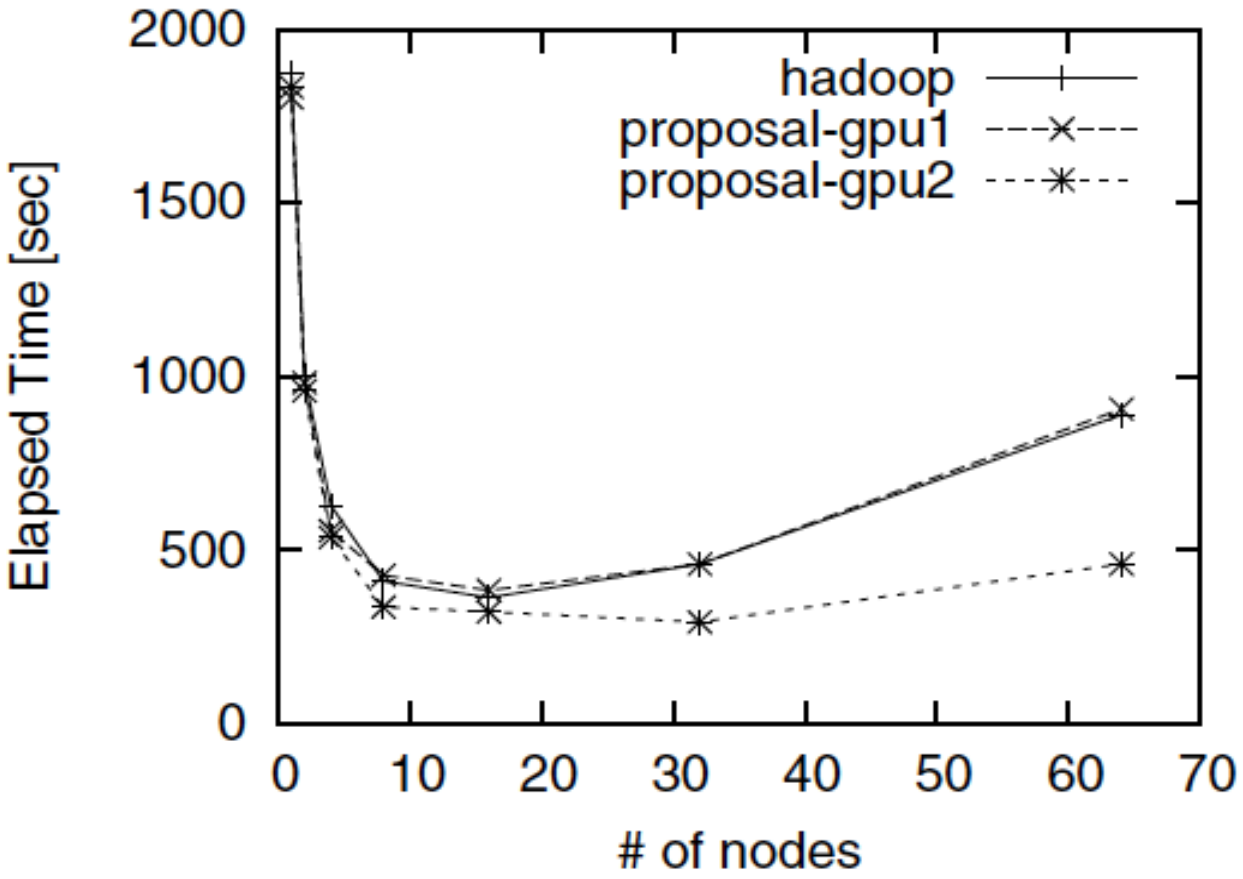
K-Means on MapReduce

- Map Phase
  - Load the cluster centers from a file
  - Iterate over each cluster center for each input key/value pair
  - Measure the distances and save the nearest center which has the lowest distance to the vector
  - Write the cluster center with its vector to the filesystem

K-Means on MapReduce

- Map Phase
  - Load the cluster centers from a file
  - Iterate over each cluster center for each input key/value pair
  - Measure the distances and save the nearest center which has the lowest distance to the vector
  - Write the cluster center with its vector to the filesystem
- Reduce Phase (we get associated vectors for each center)
  - Iterate over each value vector and calculate the average vector
  - This is the new center, save it to a file
  - Check the convergence between the cluster center that is stored in the key object and the new center
- Run this until nothing was updated anymore

Tokyo Institure of Technology – K-Means Results

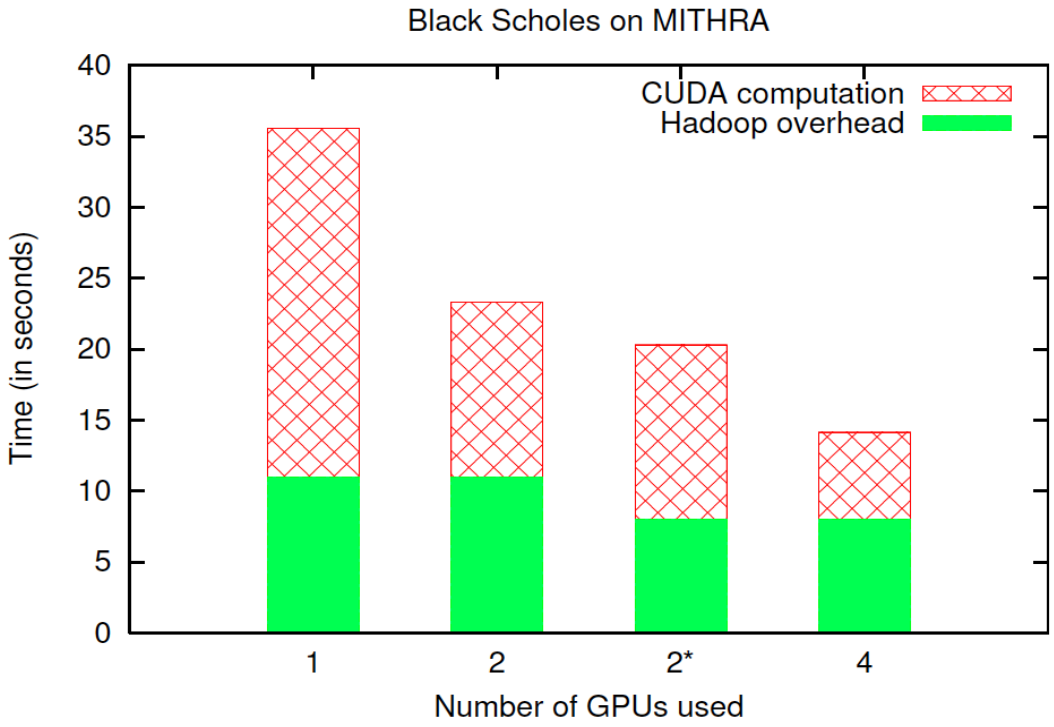


20GB input data (2-dimensional points), K=128,

each node comprises 16 CPU cores, 32GB RAM, 2 GPUs

MITHRA – Options pricing

- **M**ultiple Data **I**ndependeed **T**asks on **H**eterogenous **R**esource **A**rchitecture
- Uses CUDA SDK version of Black Scholes option pricing
- Hadoop streams to distribute and execute the CUDA binary
  - CPUs idle
- Combiners to aggregate local results



Existing Approaches - issues

- GPMR
  - Stand-alone library
  - No Fault tolerance
  - No DFS
  - No communication handling (mapper – reducer)
  - Not a complete Framework
- Tokyo Institute of Technology
  - Single application – K-means
  - Small problem set (20GB) does not scale beyond 16 nodes
  - Cost of Hadoop streams
- MITHRA
  - Single application – Options Pricing
  - Scalability evaluated on 2 nodes

References

J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. OSDI'04

J. Stuart and J. Owens. Multi-GPU MapReduce on GPU Clusters. IPDPS'11

R. Favriar, A. Verma, E.Chan, R. Campbell. MITHRA: Multiple data independent Tasks on a Heterogeneous Resource Architecture. CLUSTER'09

K. Shirahata, H. Sato, S. Matsuoka. Hybrid Map Task Scheduling for GPU-Based Heterogeneous Clusters. CloudCom'10