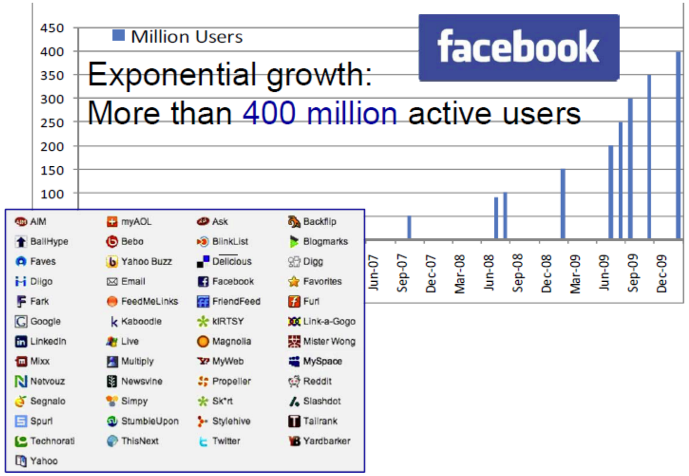
A GPU Implementation of Generalized Graph Processing Algorithm GIM-V

Koichi Shirahata[*1], Hitoshi Sato[*1,*3],
Toyotaro Suzumura[*1,*2,*3], Satoshi Matsuoka[*1,*3,*4]

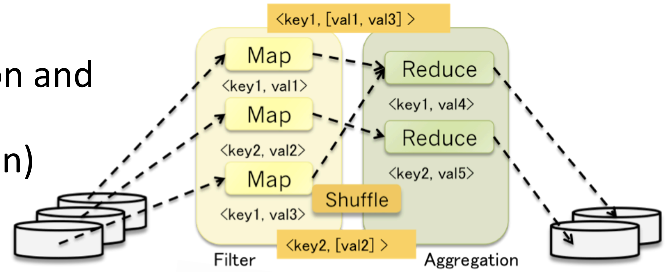
[*1] Tokyo Institute of Technology
[*2] IBM Research - Tokyo
[*3] CREST, Japan Science and Technology Agency
[*4] National Institute of Informatics

Large Scale Graph Processing with GPGPU



- Emergence of Large Scale Graph
  - Wide ranges of applications
    - Medical services, SNS, Intelligence, Biology, Smart grid, Simulation
  - The Large volume of available data, The low cost of storage

    → A need for fast processing of large scale graph

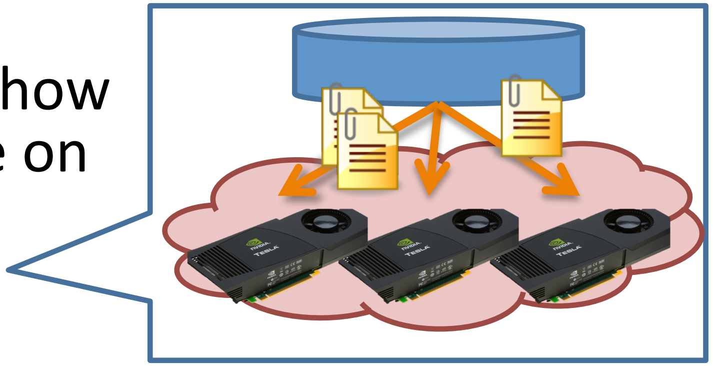


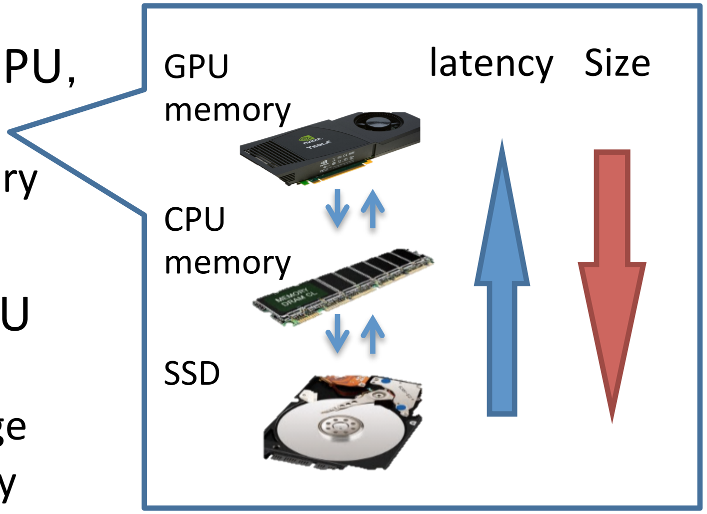
- Fast large scale graph processing methods
  - MapReduce
    - Peta-byte scale data processing by massive parallelization and automatic memory management
    - GIM-V (Generalized Iterative Matrix-Vector multiplication) model is proposed as a graph processing model by MapRedudce
  - GPGPU
    - Fast processing by many cores and memory bandwidth
    - Mars is proposed as a MapReduce system on GPU

→ Fast large MapReduce graph processing with GPGPU

Problems of Large Scale Graph Processing with GPGPU

- Applying GPU on MapReduce processing model
  - How much can MapReduce on GPU show better performance than MapReduce on CPU
- Handling on Large scale graph
  - Multi-GPU implementation
    - Delay of communication between CPU-GPU, GPU-GPU
      - Cut of communication overhead is necessary
  - Memory overflow
    - Memory on GPU is lower than that of CPU
      - ex) TSUBAME2.0 (GPU 3GB, CPU 54GB)
      - Utilization of CPU memory and local storage
      - Efficient management of memory hierarchy

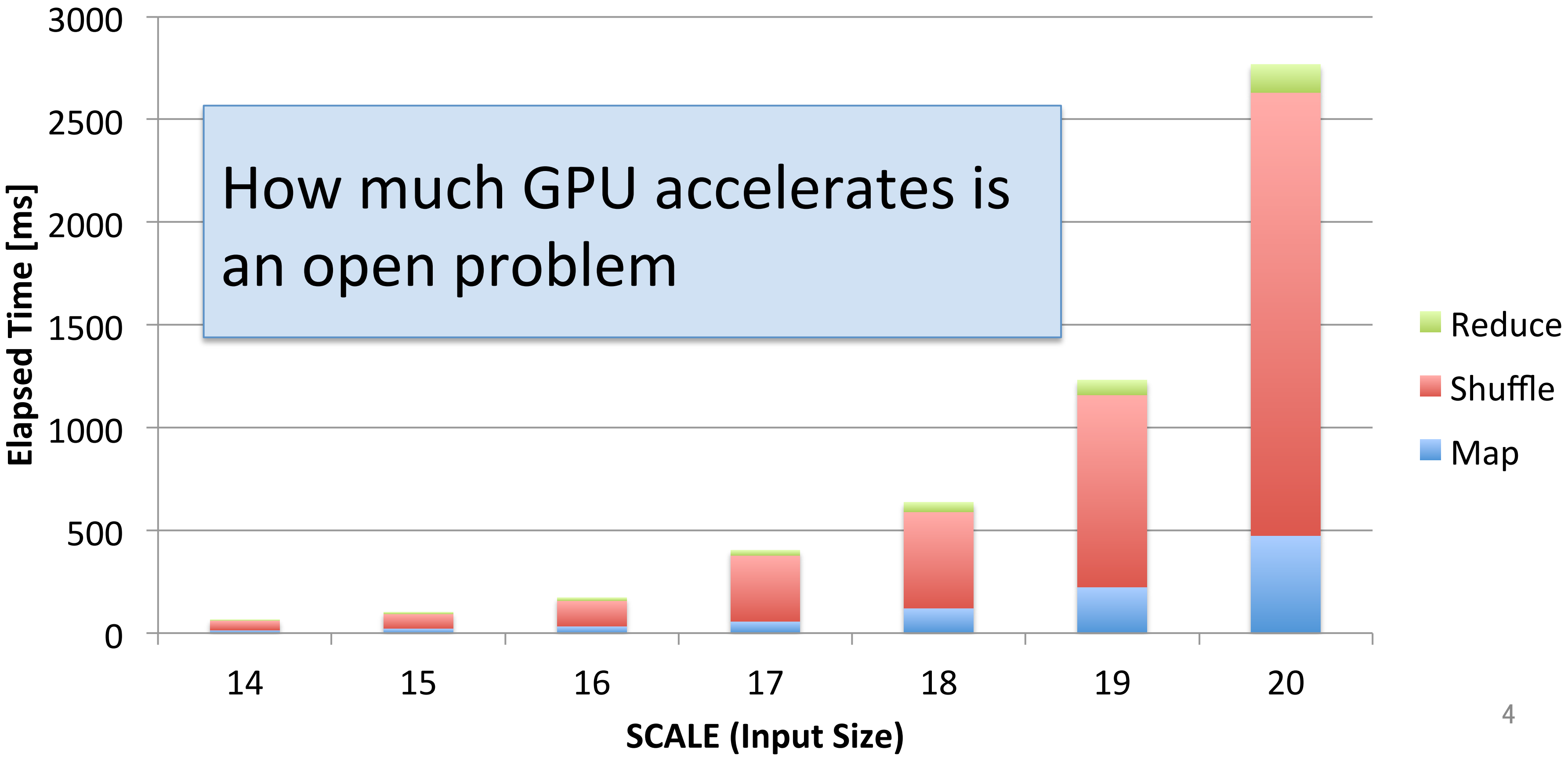




Execution time of our CPU-based Graph Processing

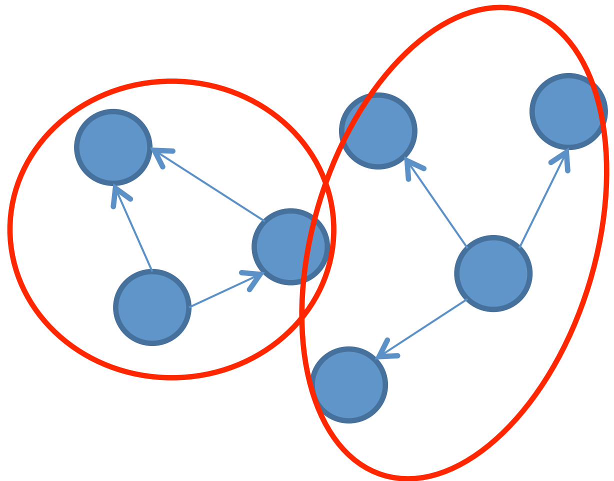
- Significant performance overheads in map and shuffle stages
  - The overheads may affect performance of graph processing with further larger size

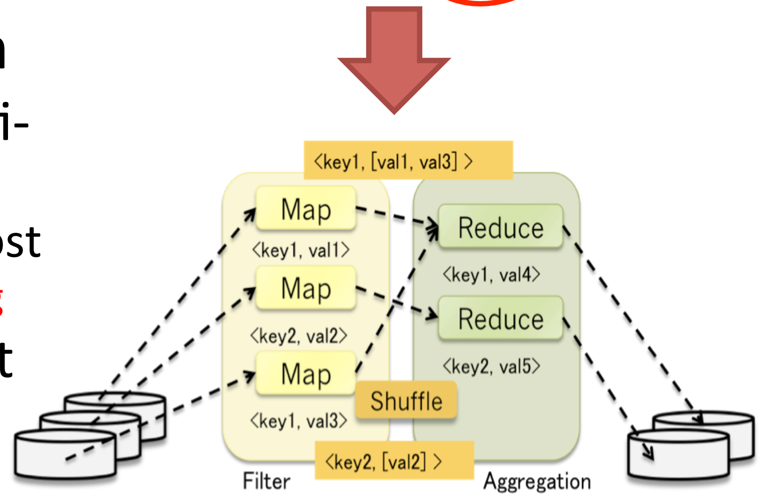
  → Could be accelerated by using GPU
- How much the applications can be accelerated using GPU is an open problem
  - Advantages: massive amount of threads, memory bandwidth
  - Uncertain factors: PCI-E overhead, performance of GPU-based algorithms



Solution: Reduction of Amount of Data Transfer Cost by Graph Partitioning

- Investigation of effectiveness of using GPU on MapReduce-based processing model
  - Comparison between existing implementation
    - Existing CPU-based implementation
    - Optimal implementation which is not based on MapReduce
- Handling extremely large scale graph
  - Add amount of memory by using multi-GPUs
    - Reduction of amount of data transfer cost
      - Reduce transfer cost by graph partitioning
  - Utilization of local storage which is not memory
    - Read data in turn from file-system and give data to GPUs
    - Scheduling for optimal data deployment





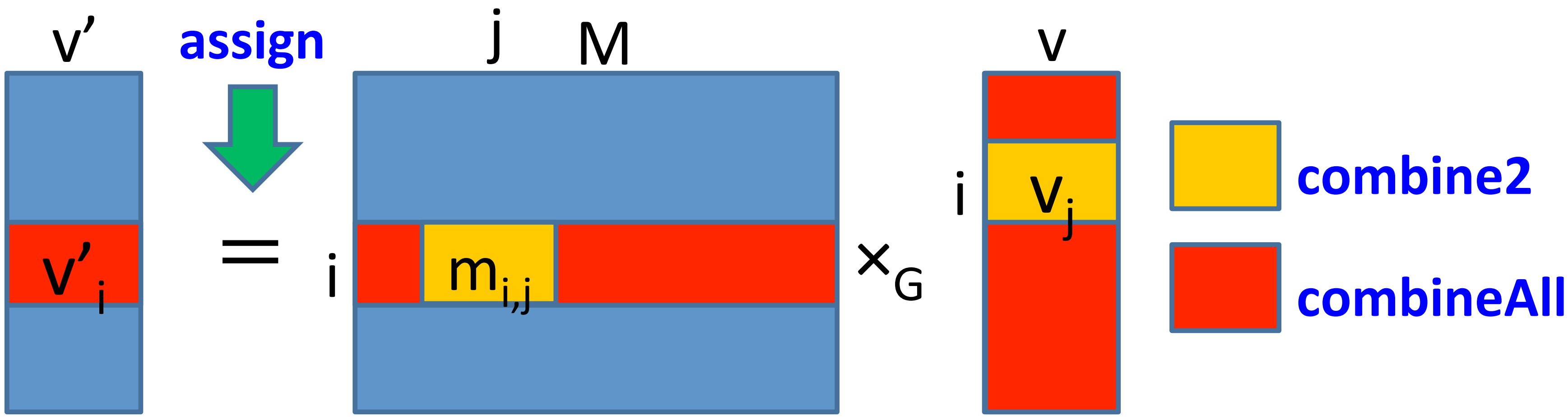
Goal and Contributions

- Goal
  - Measurement of validity of a GPU graph processing
- Conclusions
  - Acceleration using a GPU for Generalized graph processing algorithm implemented on MapReduce
    - 8.80 – 39.0x speedup compared to a Hadoop-based implementation
    - 2.72x speedup in Map stage than CPU-based implementation
    - Our GPU implementation introduces significant performance overheads in Reduce stage

Large graph processing algorithm GIM-V

- Generalized Iterative Matrix-Vector multiplication*1
  - $v' = M \times_G v$ where
    $v'_i = \text{assign}(v_j, \text{combineAll}_j(\{x_j \mid j = 1..n, x_j = \text{combine2}(m_{i,j}, v_j)\}))$ $(i = 1..n)$
  - Various graph applications can be implemented by defining above 3 functions
  - GIM-V can be implemented using 2-stage MapReduce

→ We implement GIM-V on existing GPU-based MapReduce flamework (Mars)

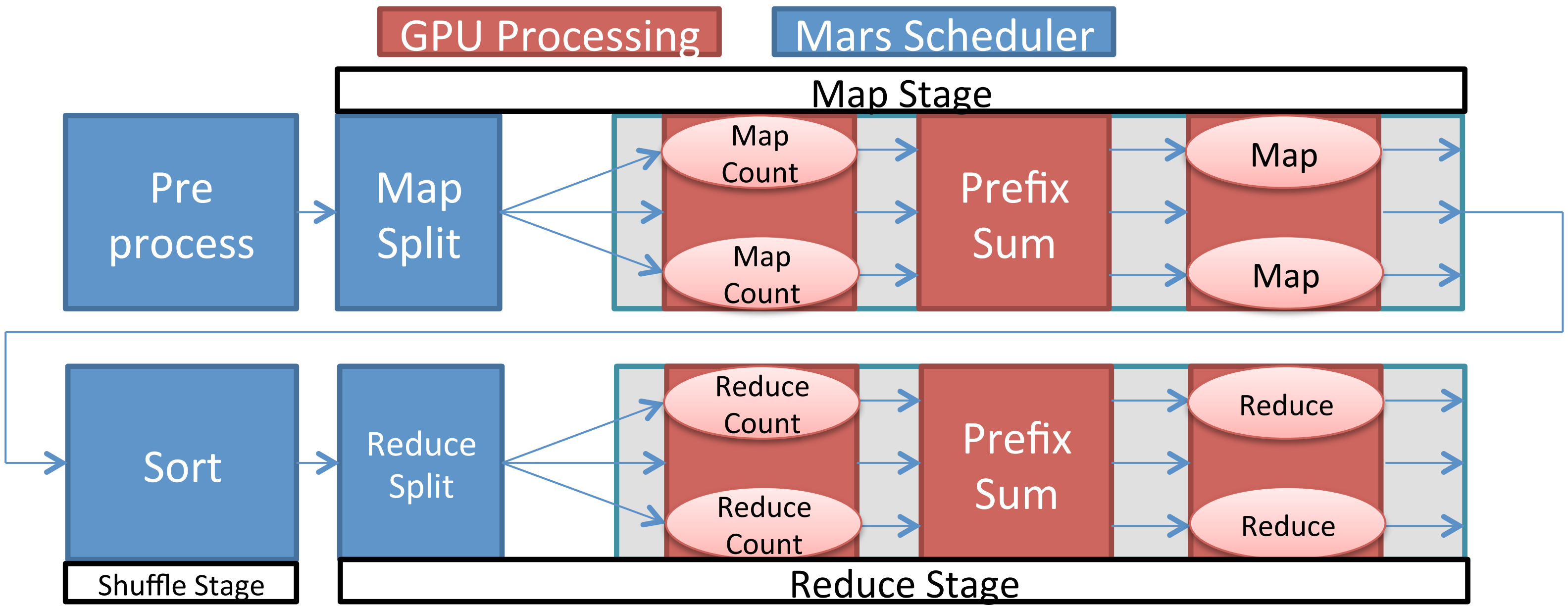


*1 : Kang, U. et al, "PEGASUS: A Peta-Scale Graph Mining System- Implementation and Observations", IEEE INTERNATIONAL CONFERENCE ON DATA MINING 2009

Structure of Mars

- Mars*1 : an existing GPU-based MapReduce framework
  - Map, Reduce functions are implemented as CUDA kernels
    - Mapper/Reducer are called in increments of a GPU thread
    - Map/Reduce Count → Prefix sum → Map/Reduce
  - Shuffle stage executes GPU-based Bitonic Sort
  - CPU-GPU communication at starting Map

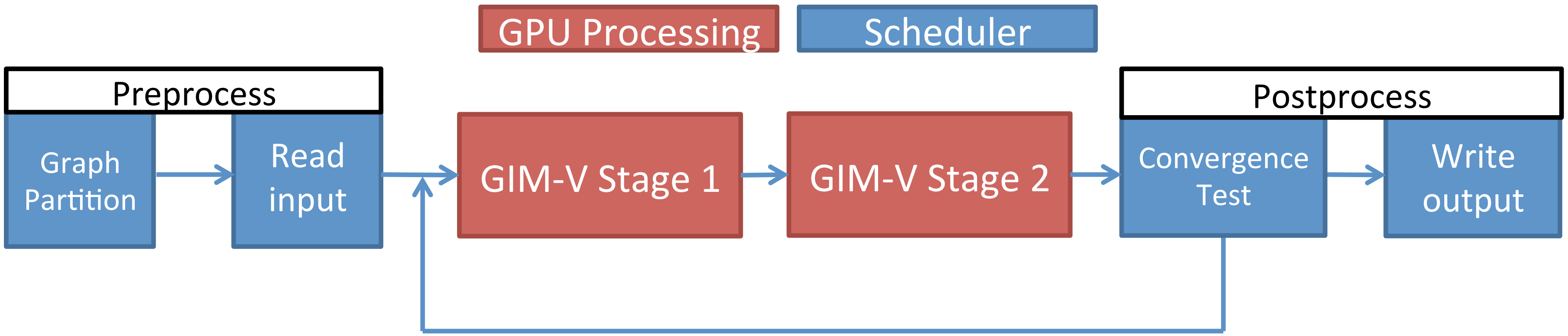
→ We extends Mars for a GPU GIM-V graph processing



*1 : Fang W. et al, "Mars: Accelerating MapReduce with Graphics Processors", Parallel and Distributed Systems, 2011

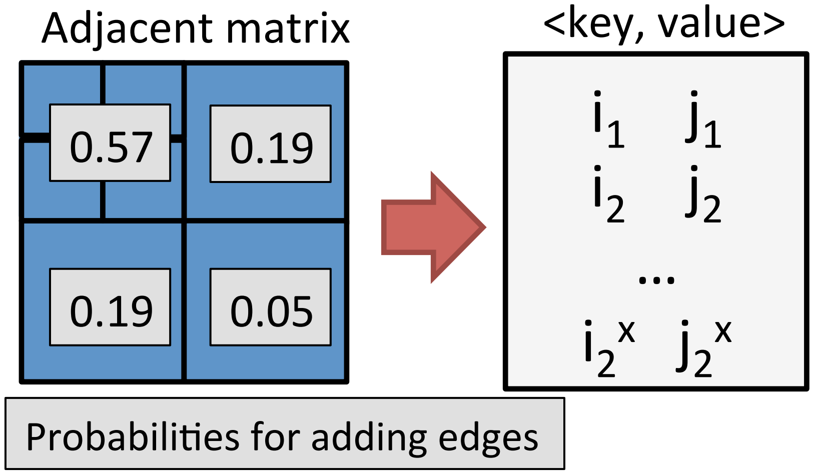
GIM-V implementation on a GPU

- GPU-based GIM-V implementation on top of Mars
  - Continuous execution feature of multi MapReduce stages
    - CPU-GPU communication at the start and the end of each iteration
    - Convergence test as a post processing



Experiments

- Questions
  - Performance of our GIM-V implementation on a GPU
- Measurement method
  - Mean time of 1 round of iterative graph processing
    - Comparison with existing CPU implementation (PEGASUS)
    - Comparison with CPU-based Mars
- Methods
  - Application
    - PageRank
      - Measures relative importance of web pages
  - Input data
    - Artificial Kronecker graph
      - Generated by generator in Graph 500
    - Parameters
      - SCALE: the base 2 logarithm of #vertices
      - #edges = #vertices × 16



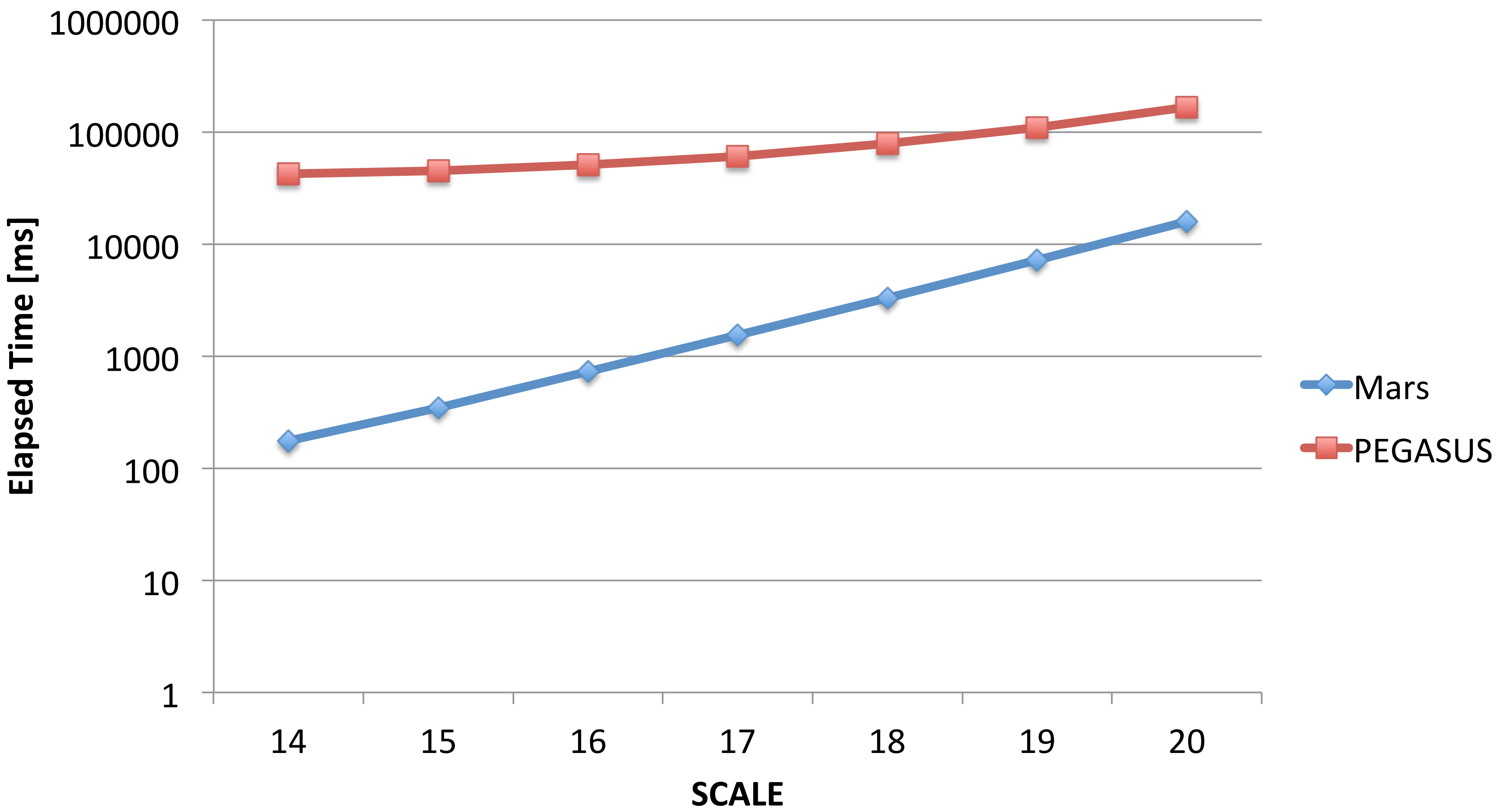
Experimental environments

- TSUBAME 2.0
  - We use 1 GPU on 1 node
    - CPU 6 cores x 2 sockets, 24 threads (HyperThread enabled)
  - GPU
    - CUDA Driver Version: 4.0
    - CUDA Runtime Version: 4.0
    - Compute Capability: 2.0
    - shared/L1 cache size: 64 KB
- Mars
  - *MarsGPU*
    - 1 GPU
    - # threads = # different keys
      - 256 threads on a thread block
  - *MarsCPU*
    - 24 threads / node
    - implemented by C instead of CUDA
    - Sort is implemented by parallel quick sort
- PEGASUS
  - Hadoop 0.21.0
  - Lustre file system as DFS

	CPU	GPU
Model	Intel® Xeon® X5670	Tesla M2050
# Physical cores	12	448
Frequency	2.93 GHz	1.15 GHz
Amount of memory	54 GB	2.7 GB (Global)
Compiler	gcc 4.3.4	nvcc 3.2

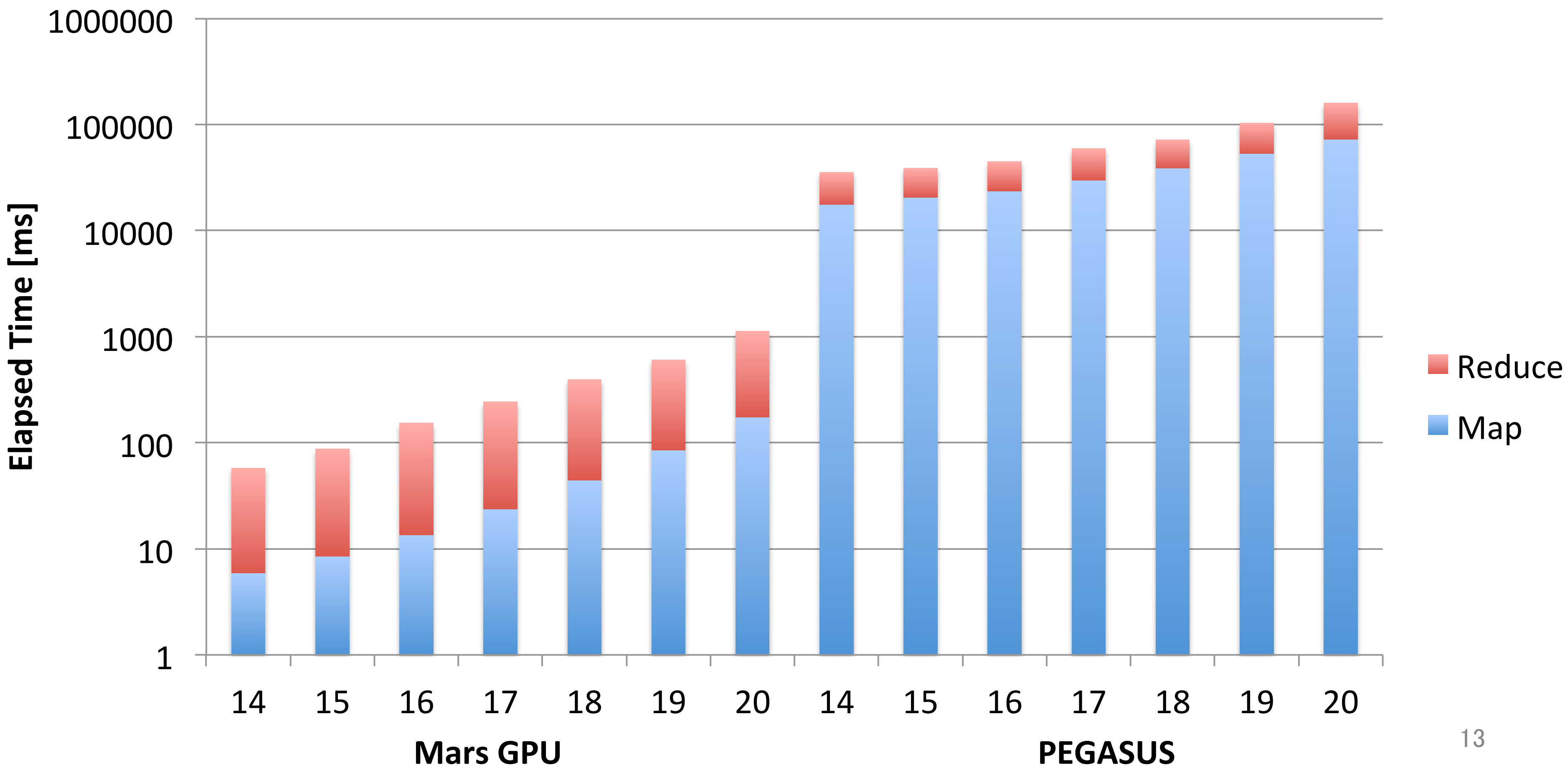
Elapsed Time: Mars vs. PEGASUS -PageRank

- Compare mean elapsed time of each iteration for Mars, PEGASUS (a Hadoop-based Graph Processing implementation)
- Mars is 8.80 – 39.0x faster than PEGASUS (8 mapper, 2 reducer / node)
  - Map and Reduce are measured from task invocation on PEGASUS
  - File I/O occurs very often during Map and Reduce executions



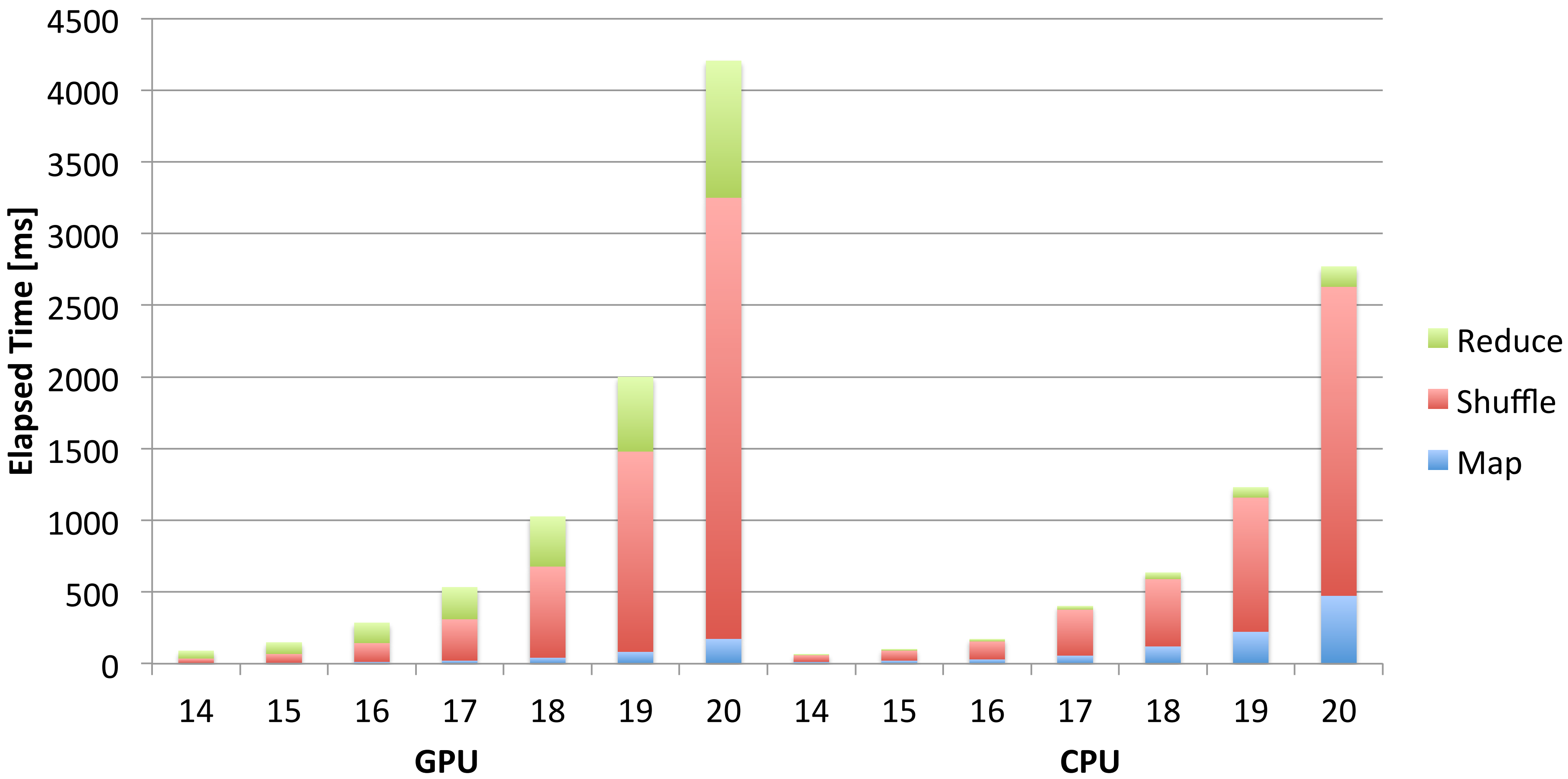
Mars vs. PEGASUS -Breakdown

- Map stage is highly accelerated by GPU
- I/O optimization between iterations
  - PEGASUS conducts read/write I/O operations in each iteration
  - Mars forwards output in Reduce to input in next Map



Elapsed Time: *MarsGPU* and *MarsCPU*

- In Map stage, *MarsGPU* is 2.72x faster than *MarsCPU*
- In Reduce stage, *MarsGPU* introduces significant overheads
  - The overhead derives from the characteristic of the graph
    - We used Kronecker graph, which has considerable locality



Related Work

- Existing large-scale graph processing systems
  - Pregel*1 : BSP-oriented implementation using Master/Worker model
    - Vertex centric model
  - Parallel BGL*2 : MPI-based C++ graph processing library
- Graph processing using GPU, MapReduce
  - Shortest path algorithms for GPU*3
    - Fast implementation of BFS, SSSP, and APSP algorithms
  - MapReduce-based shortest path problems will be released in Graph500 *4 reference implementation
- MapReduce implementations on multi GPUs, multi nodes
  - GPMR*5 : MapReduce implementation on multi GPUs
  - MapReduce-MPI*6 : MapReduce library using MPI

→ Efficient MapReduce-based graph processing using GPU

*1 : Malewicz, G. et al, "Pregel: A System for Large-Scale Graph Processing", SIGMOD 2010.
*2 : Gregor, D. et al, "The parallel BGL: A Generic Library for Distributed Graph Computations", POOSC 2005.
*3 : Harish, P. et al, "Accelerating large graph algorithms on the GPU using CUDA", HiPC 2007.
*4 : David A. Bader et al, "The Graph 500 List"
*5 : Stuart, J.A. et al, "Multi-GPU MapReduce on GPU Clusters", IPDPS 2011.
*6 : Plimpton, S.J. et al, "MapReduce in MPI for Large-scale Graph Algorithms", Parallel Computing 2011.

Conclusions

- Conclusions
  - Acceleration using a GPU for GIM-V
    - 8.80 – 39.0x speedup compared with PEGASUS
    - 2.72x speedup in Map stage than *MarsCPU*
    - *MarsGPU* introduces significant performance overheads in Reduce stage
- Future work
  - Optimization of our implementation
    - Performance improvement in Shuffle and Reduce stages
    - Multi GPU implementation
  - Data handling for out of GPU memory
    - Use local storage as well as CPU/GPU memories
    - Efficient memory hierarchy management

Reduction of I/O between iterations

- Comparison between w/ and w/o disc I/O in each iteration
- 1.6 – 9.1x faster by reducing disc I/O