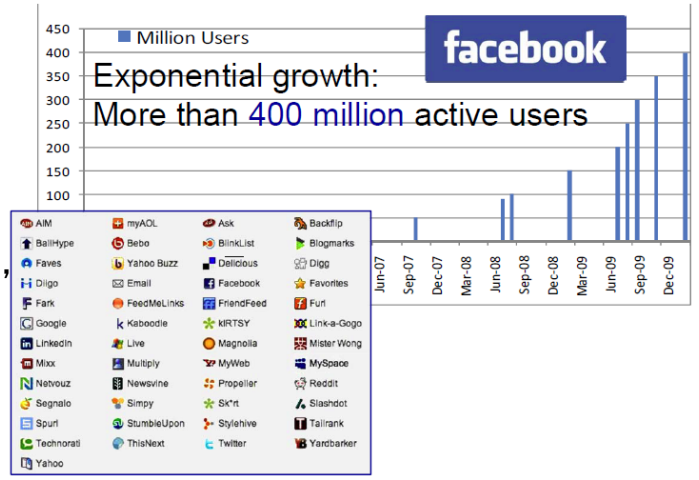
GPGPUを用いた大規模高速グラフ処理に向けて

白幡 晃一*1, 佐藤 仁*1, 鈴村 豊太郎*1,*2, 松岡聡*1, *3, *4
*1 東京工業大学
*2 IBM 東京基礎研究所
*3 国立情報学研究所
*4 科学技術振興機構

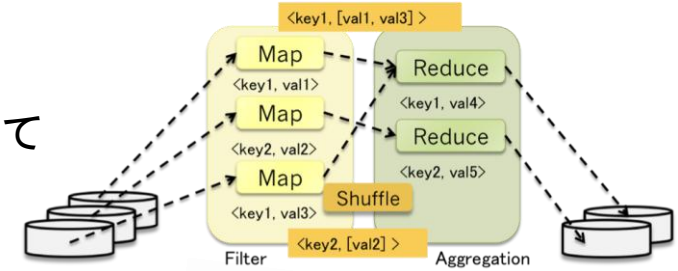
GPGPUを用いた大規模グラフ処理

- 大規模グラフの出現
  - データ量の肥大化，ストレージの省コスト化
  - 幅広い応用例
    - 医療，ソーシャルネットワーク，インテリジェンス，生物学，スマートグリッド，シミュレーション

    → 大規模グラフの高速処理が必要



- 高速大規模グラフ処理手法
  - MapReduce
    - 並列化による大規模データ処理
    - 既存の MapReduce によるグラフ処理モデルとして GIM-V モデルが提案されている
  - GPGPU
    - コア数，メモリバンド幅の活用により高速処理が可能
    - 既存のGPU 上での MapReduce 処理系として Mars が提案されている



→ GPU と MapReduce による高速大規模グラフ処理

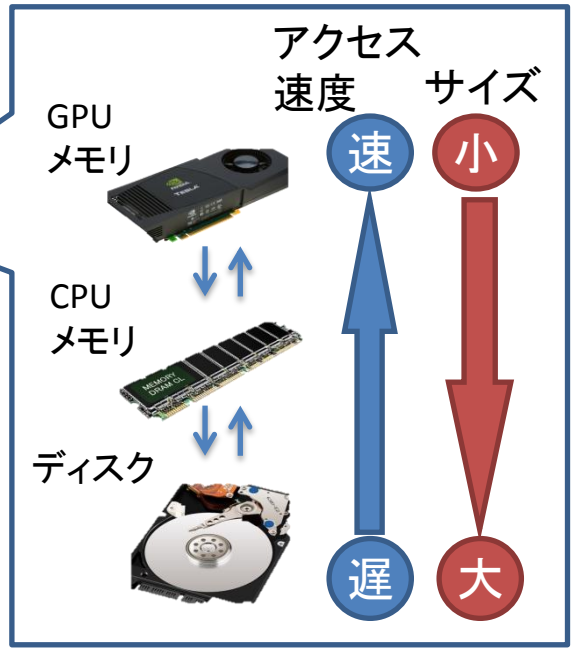
GPGPUを用いた大規模グラフ処理における問題点

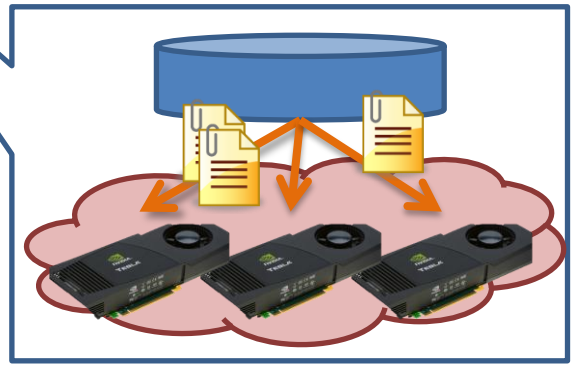
- メモリあふれへの対処
  - GPU は CPUに比べメモリ容量が少ない
  - 効率的なメモリ階層の管理が必要
- マルチ GPU 化
  - 効率的なデータの割り振り方法は明らかでない
- GIM-V モデルへの適用

本発表

  - 既存のCPU実装に対して，GPUの使用によりどの程度高速化できるか定かでない

→ GIM-V モデルを Mars 上に実装して評価





目的と成果

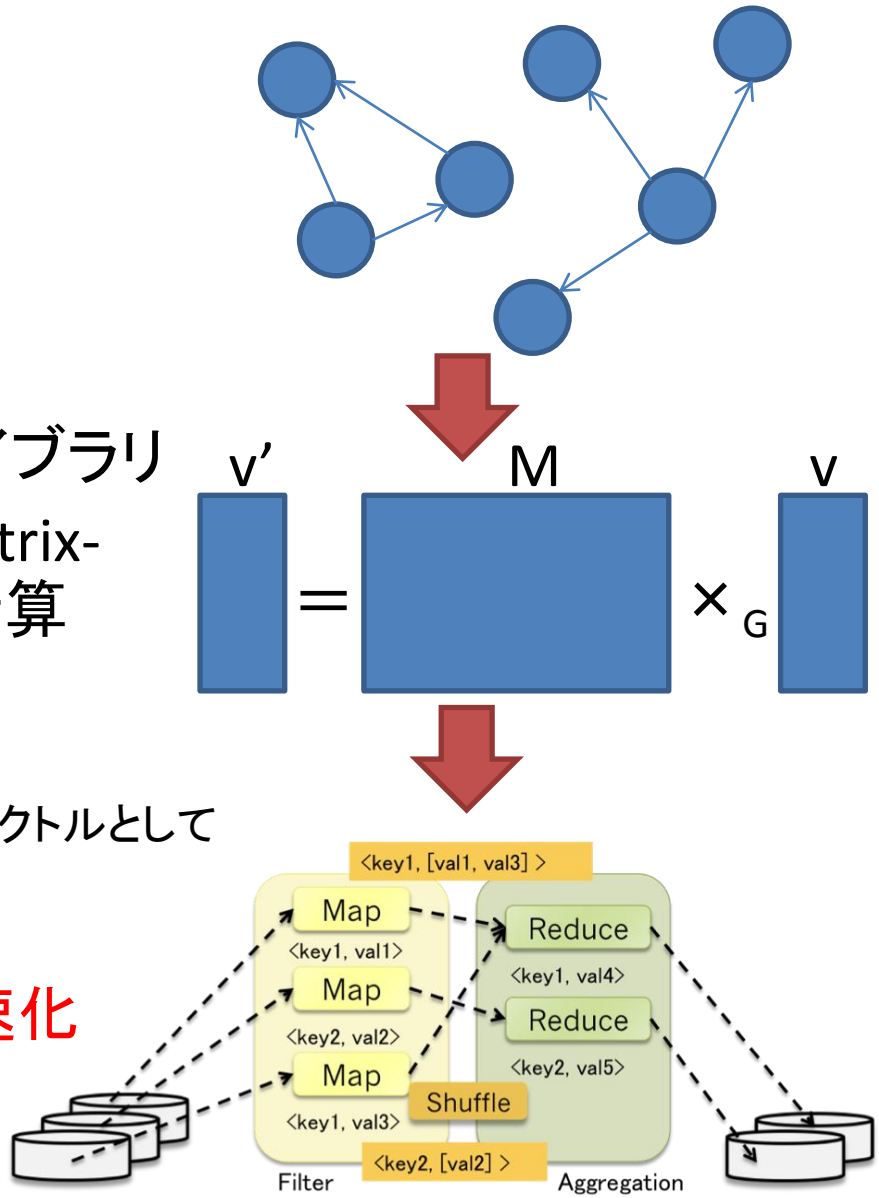
- 目的
  - GPU 上での MapReduce による高速グラフ処理
- 成果
  - GPU 上への GIM-V モデルの実装
    - Mars 上に実装
  - 評価
    - 3 種類のグラフ処理アプリケーション
      (PageRank, Random Walk with Restart, Connected Components)
    - CPUによる既存実装との比較実験

    → CPU 4コアに比べ，2.17～9.53倍の高速化

既存の大規模グラフ処理システム

- グラフ構造データ
  - 頂点集合と辺集合から成る
    - 辺: 頂点の対
- PEGASUS
  - Hadoopベースのグラフ処理ライブラリ
  - GIM-V (Generalized Iterated Matrix-Vector multiplication) による計算
    - 行列ベクトル積を一般化
    - 典型的なグラフ処理を記述可能
      - 辺集合を隣接行列，頂点集合をベクトルとして計算
    - MapReduceにより記述可能

→ GPUの使用によりどれほど高速化できるかは定かではない

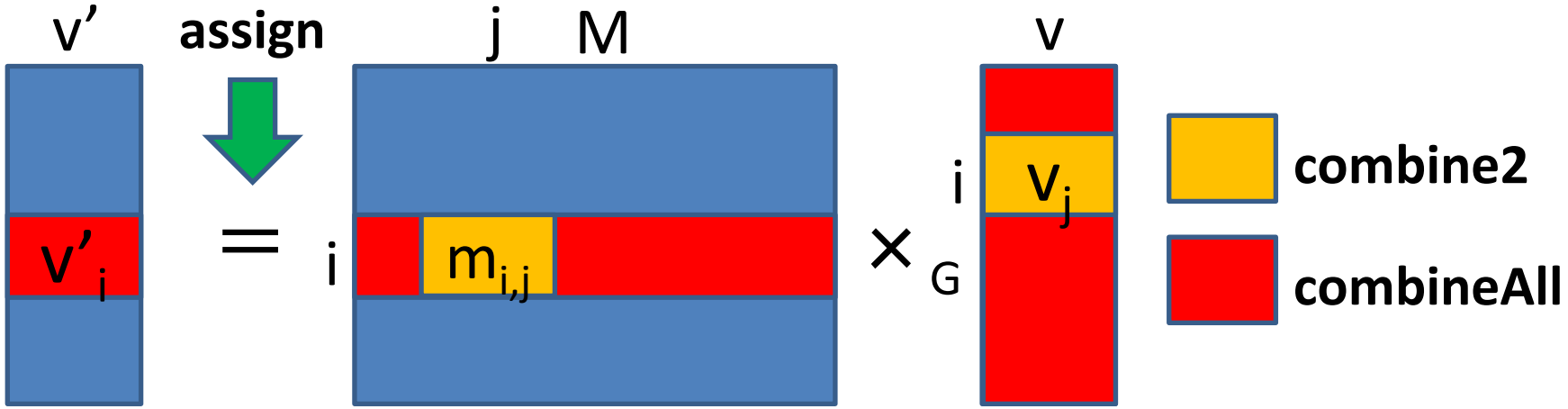


GIM-V

- Generalized Iterative Matrix-Vector multiplication*1
  - 反復行列ベクトル積の一般化
  - $v' = M \times_G v$ where

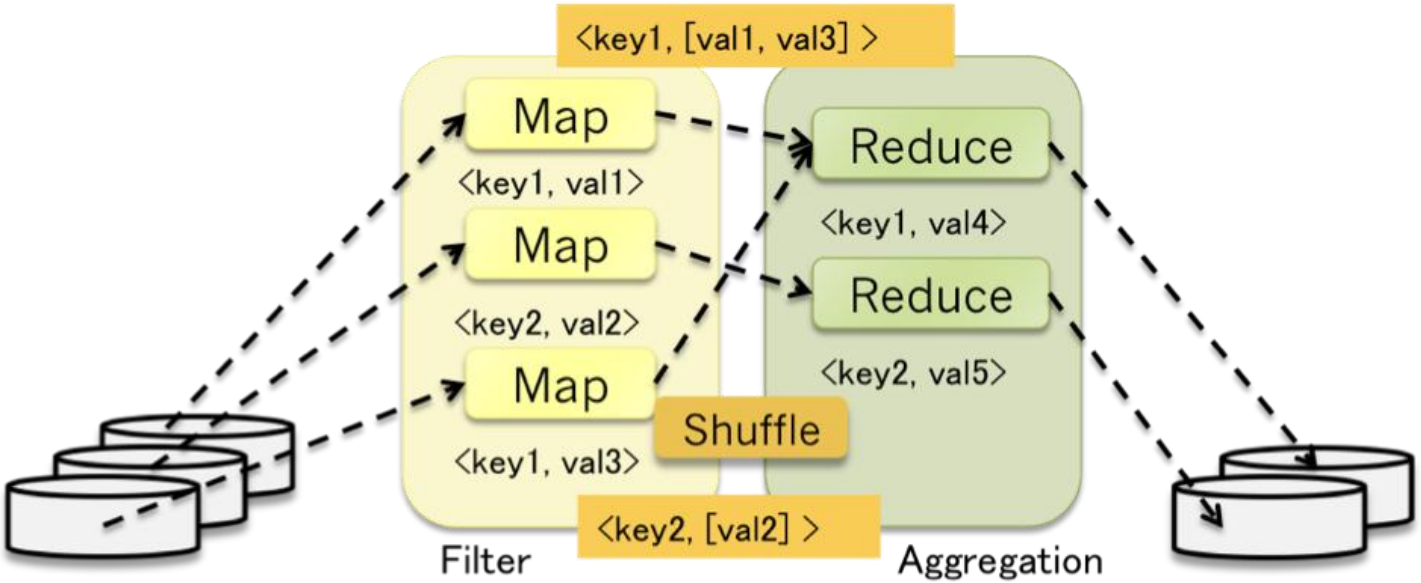
    v'_i = **assign**(v_i, **combineAll**$_i$ ({x_j | j = 1..n, x_j = **combine2**($m_{i,j}$, v_j)}))
  - 上記3つの関数を実装することで，様々なグラフ処理を記述可能
  - 演算は反復して収束するまで実行
  - GIM-V を 2つのMapReduceにより記述可能
    - PEGASUS ではHadoop上に実装

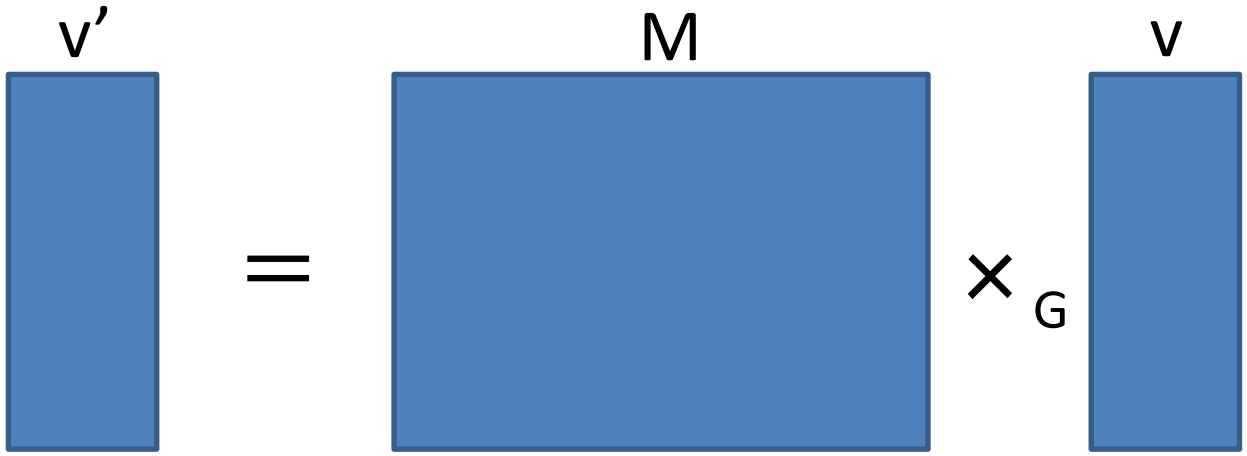
  → Mars 上に実装し評価



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

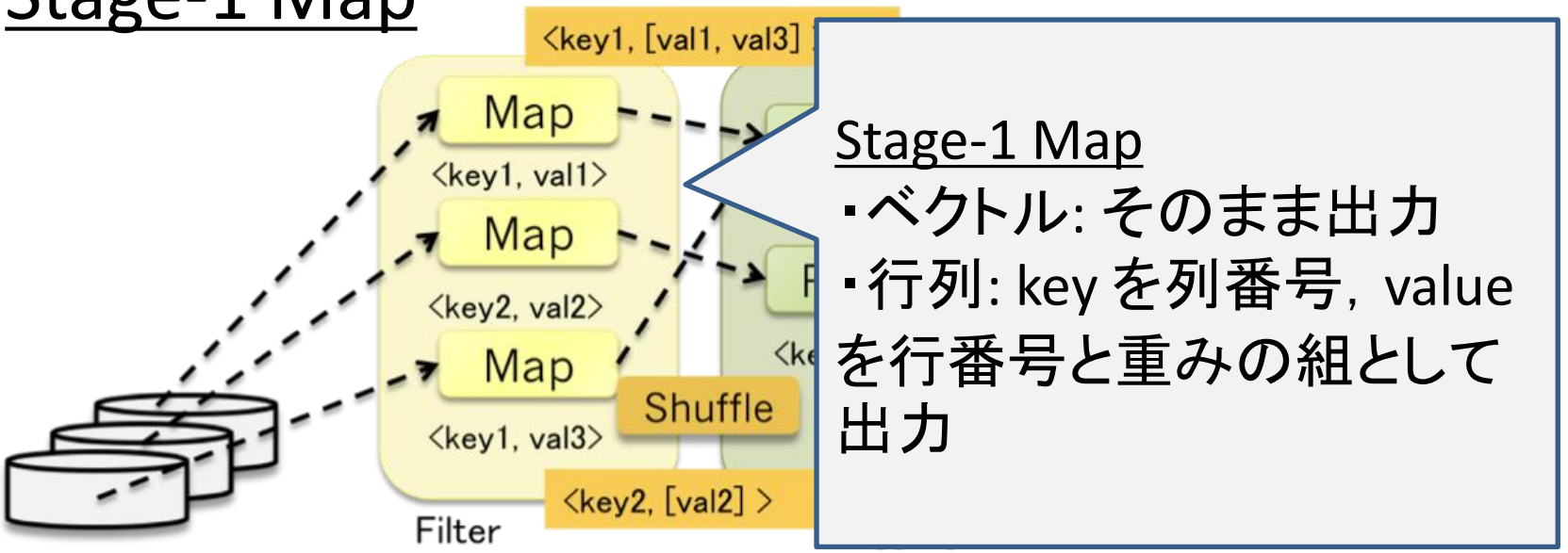
MapReduce による GIM-V 計算

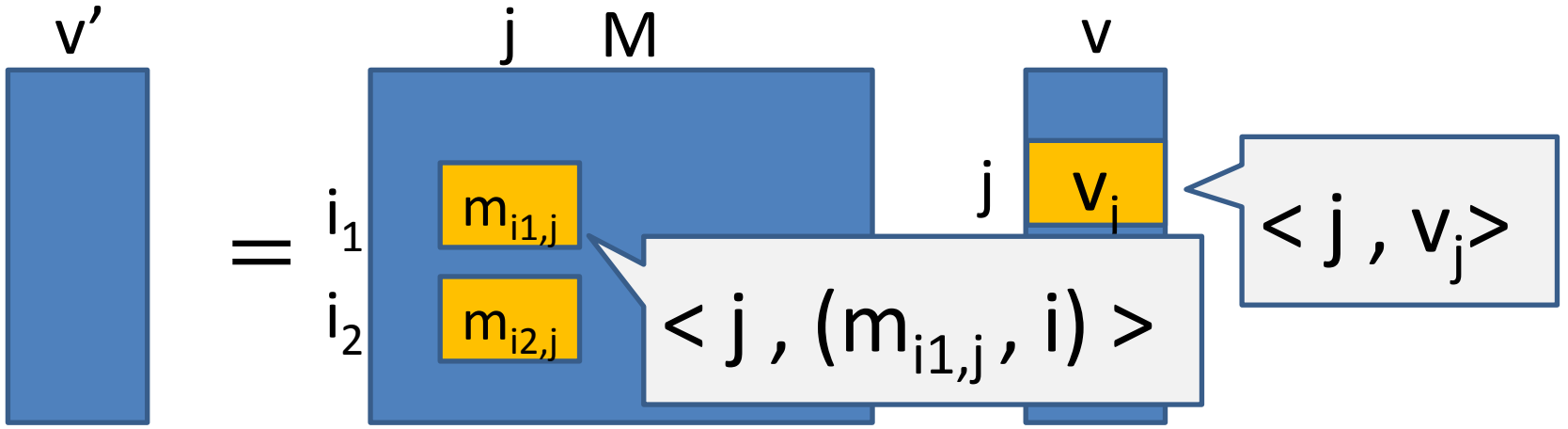




MapReduceによるGIM-V計算

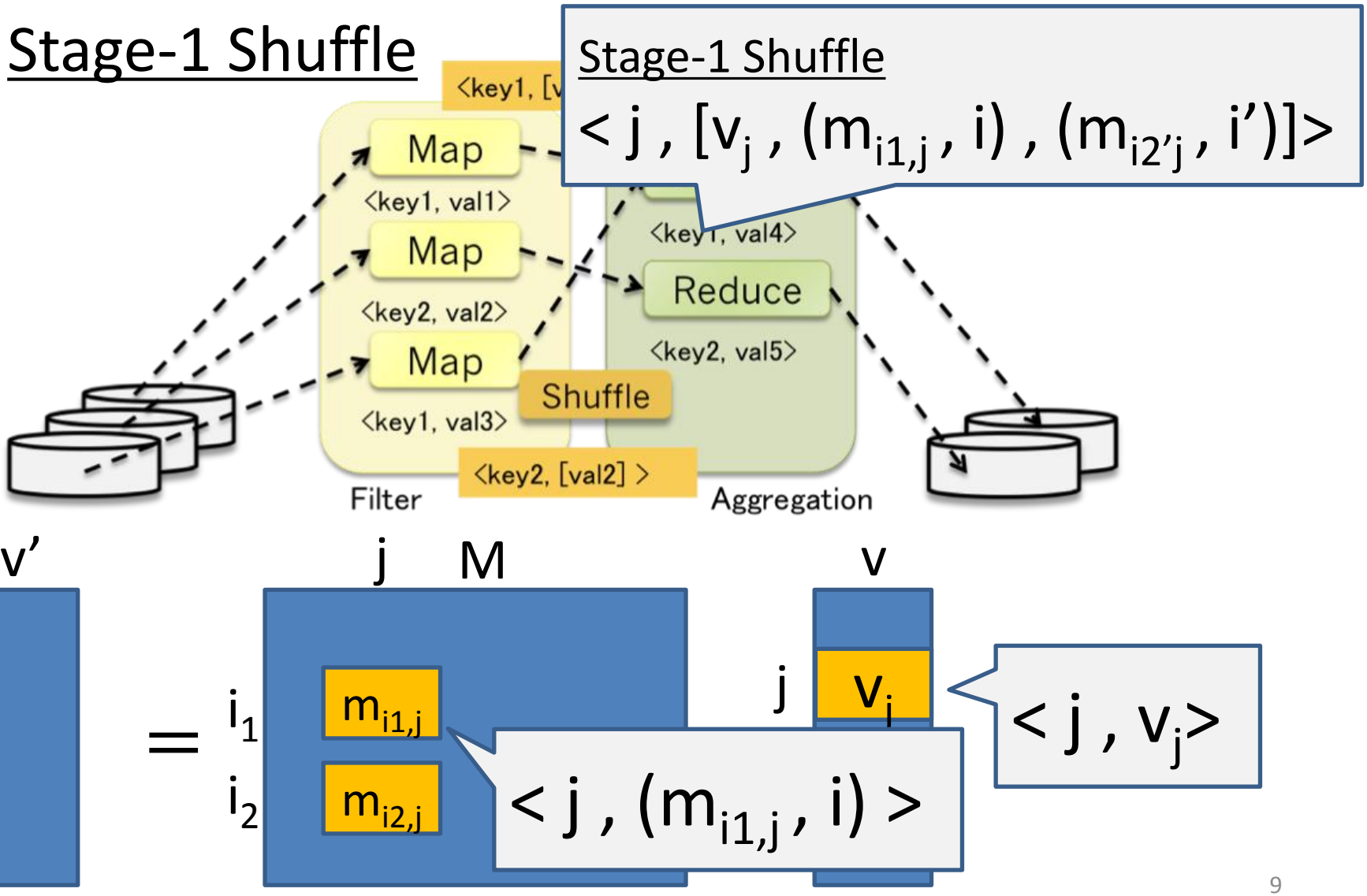
- Stage-1 Map





MapReduceによるGIM-V計算

- Stage-1 Shuffle



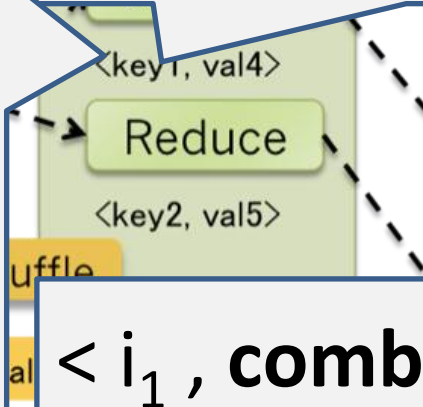
MapReduceによるGIM-V計算

- Stage-1 Reduce

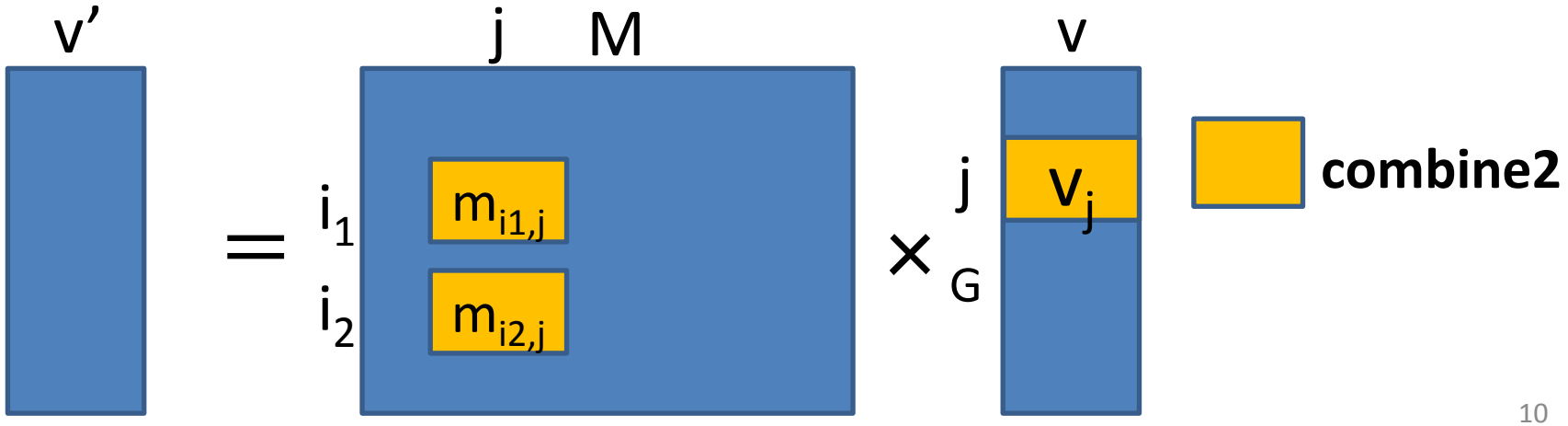
Stage-1 Shuffle

$< j , [v_j , (m_{i1,j} , i) , (m_{i2'j} , i')]>$

Stage-1 Reduce
key を行番号，value を
combine2(行列の要素の値，
ベクトルの要素の値**)**
として出力

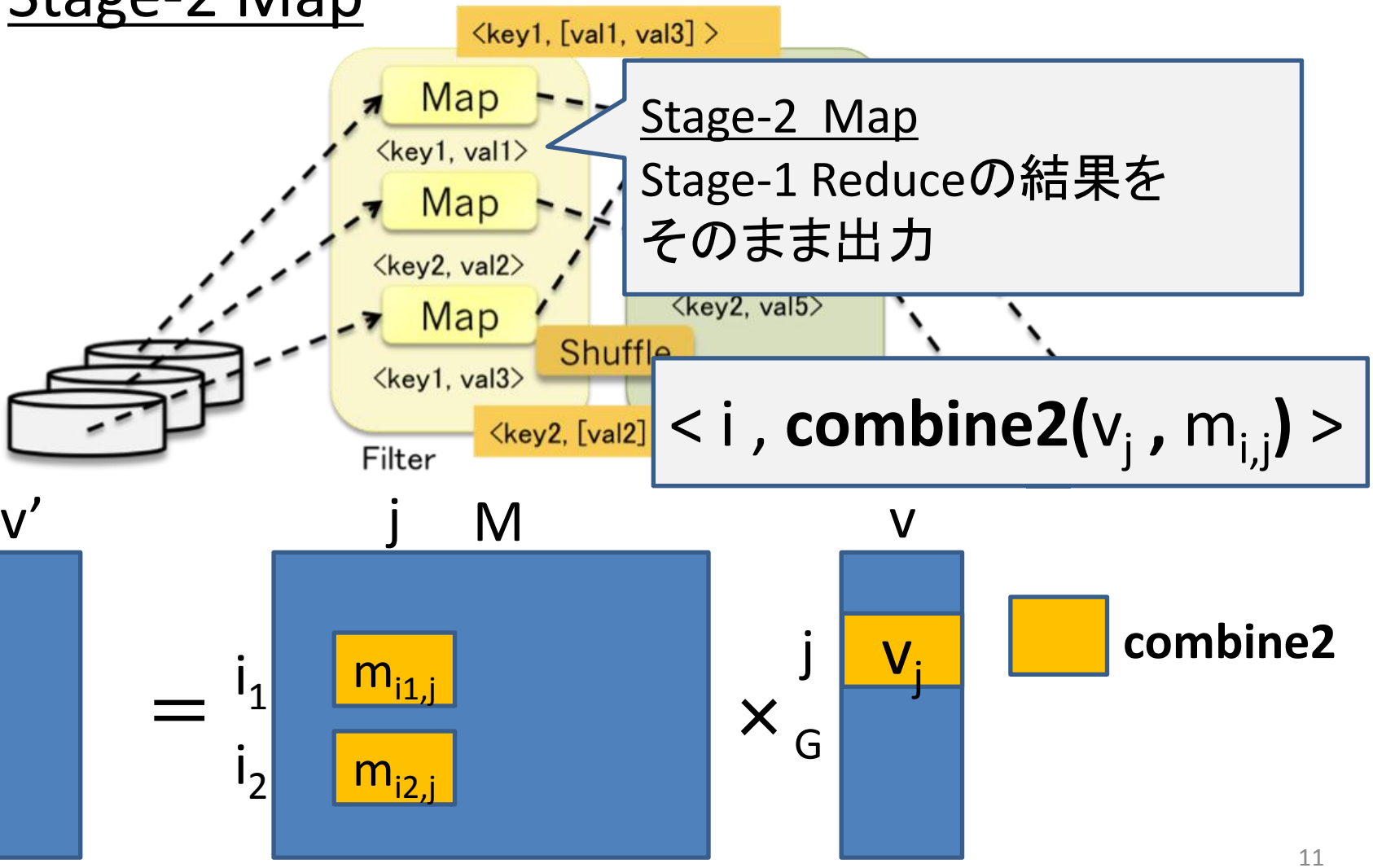


$< i_1 , \mathbf{combine2}(v_j , m_{i1,j}) >$



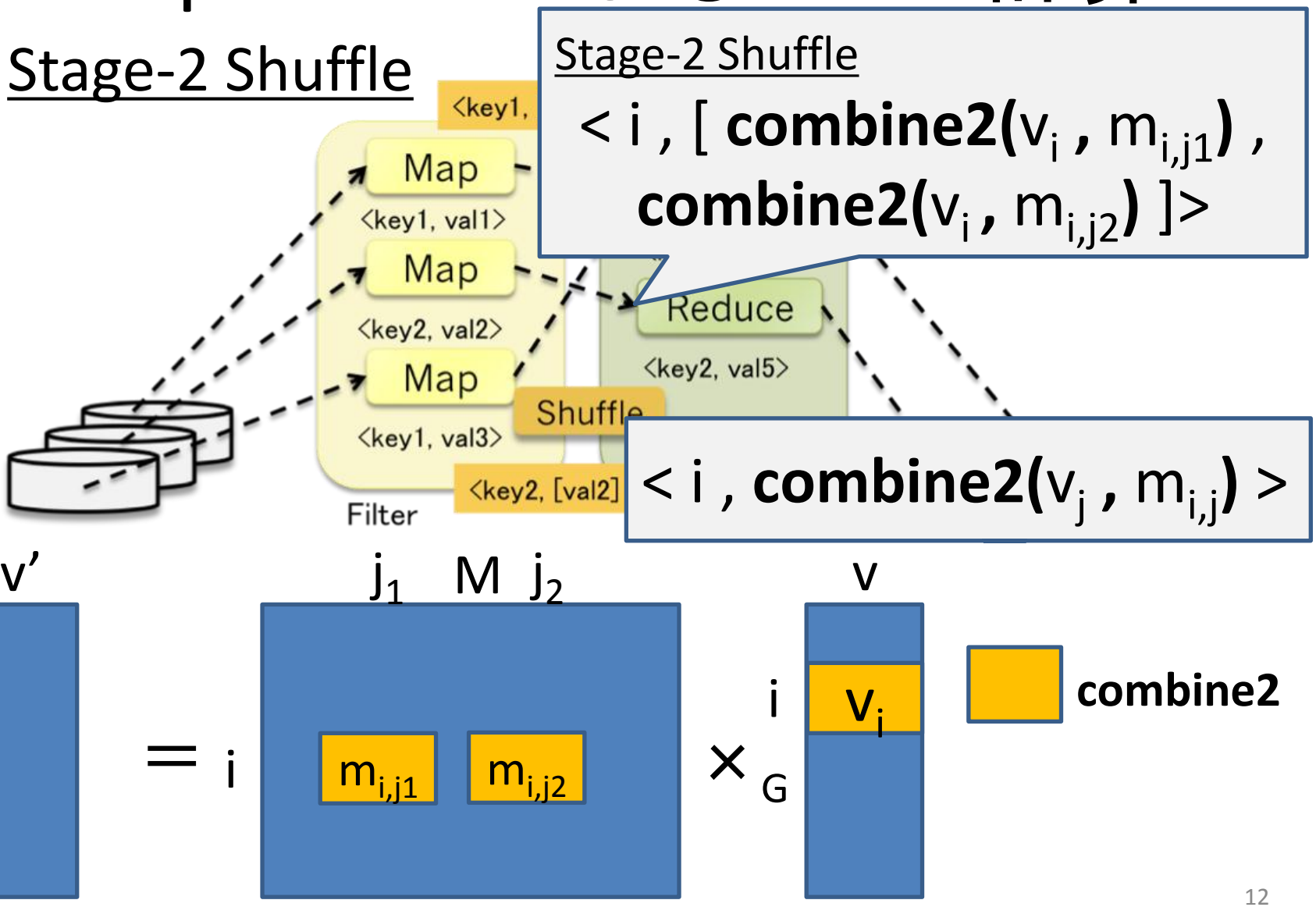
MapReduceによるGIM-V計算

- Stage-2 Map



MapReduceによるGIM-V計算

- Stage-2 Shuffle



MapReduceによるGIM-V計算

- Stage-2 Reduce

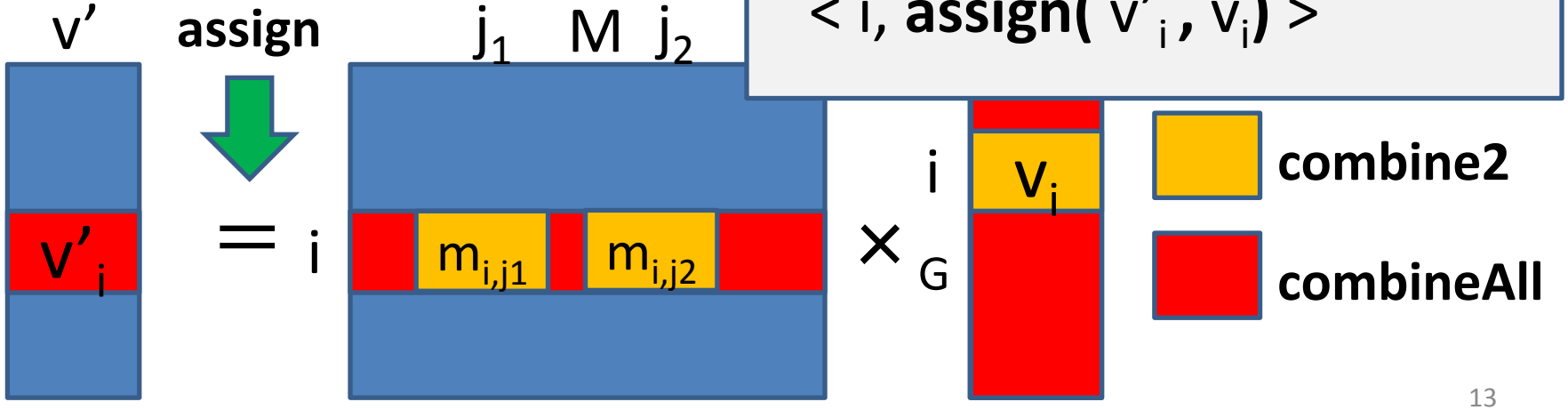
Stage-2 Shuffle

$< i , [\mathbf{combine2}(v_i , m_{i,j1}) , \mathbf{combine2}(v_i , m_{i,j2})]>$

Stage-2 Reduce

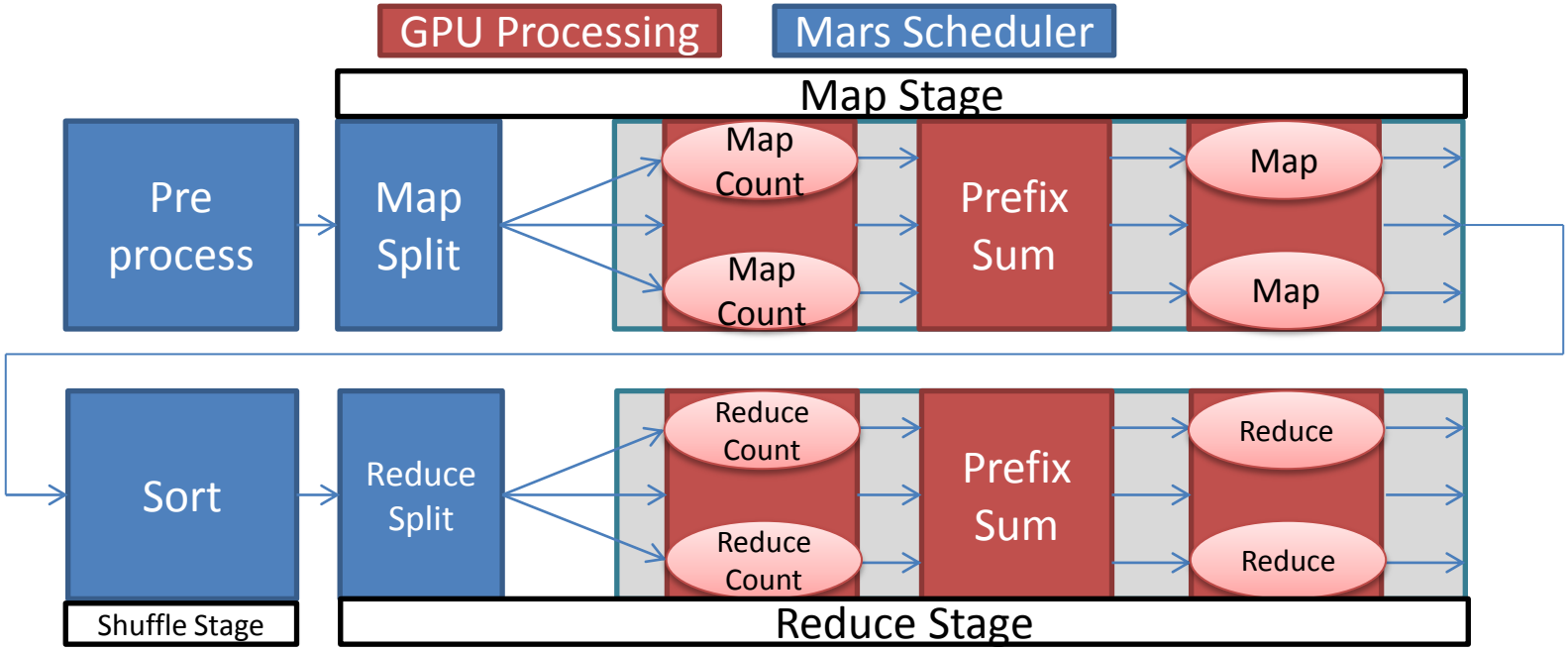
Stage-1 Reduce の**combine2**の結果に対して**combineAll**を実行し、その結果を新しいベクトルの要素として**assign**

$v'_i = \mathbf{combineAll}([\mathbf{combine2}(v_i , m_{i,j1}) , \mathbf{combine2}(v_i , m_{i,j2})]) < i, \mathbf{assign}(v'_i , v_i) >$



Mars の構造

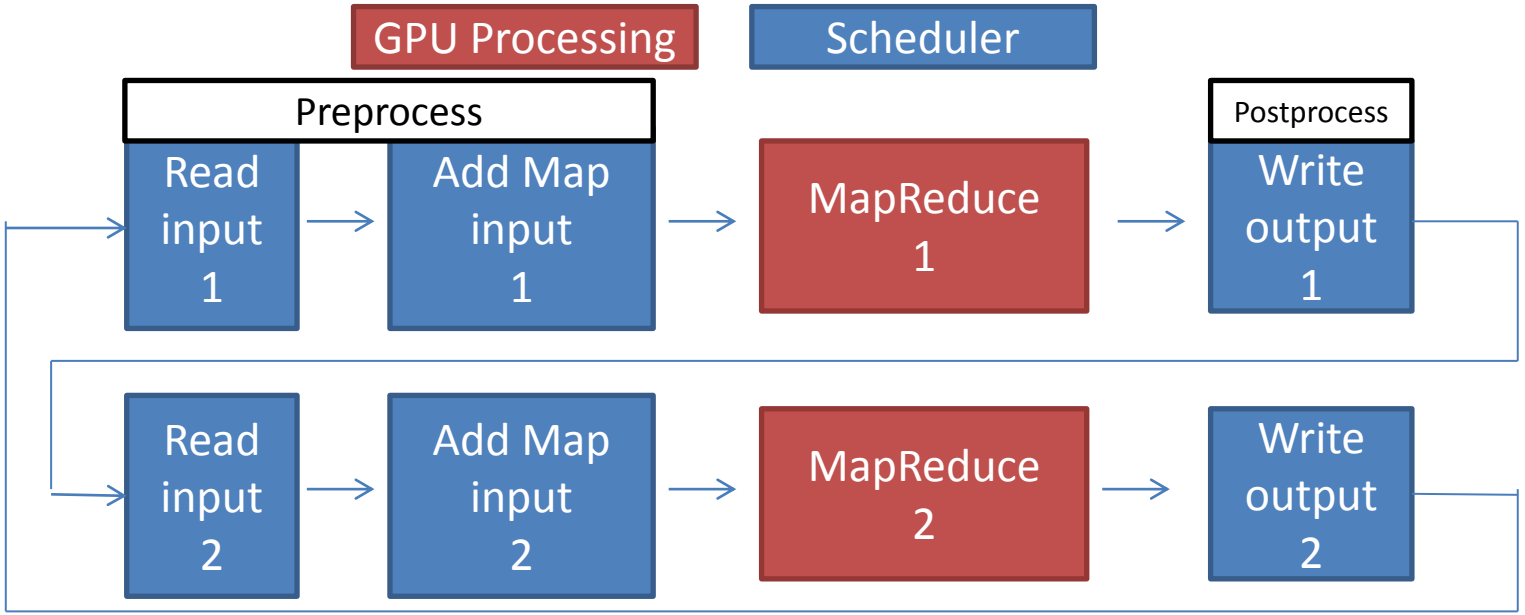
- Mars*1 : 既存のGPU用MapReduceフレームワーク
- Map, Reduce 関数をCUDAカーネルとして実装
  - GPU のスレッド単位でMapper，Reducer を呼び出し
  - データ構造: <Key size, Key offset, Value size, Value offset>
  - Map/Reduce Count → Prefix sum → Map/Reduce の順に実行
- Shuffleフェーズは GPUベースの Bitonic Sortを実行
- 各MapReduceの始めと終わりにCPU-GPU間入出力



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

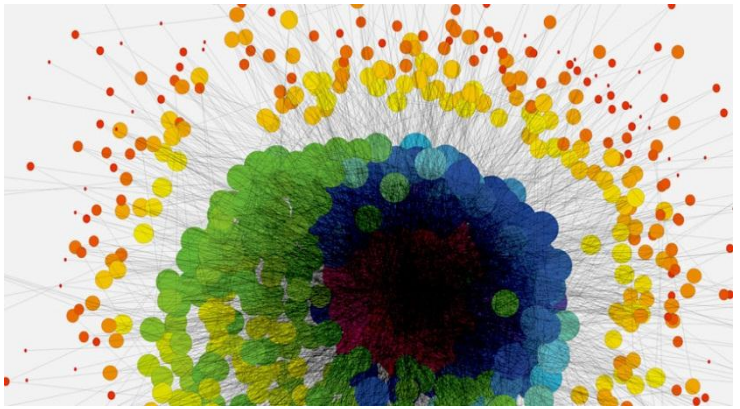
実装

- Mars 上に GIM-V を実装
  - 各MapReduce毎に入出力をファイルに読み書き
  - MapReduce処理の前処理としてMapの入力を一行ずつ(CPUメモリ内に)格納
  - GIM-V 以外の計算も必要
    - PageRank では2つ目のReduce と PostProcess で収束判定
    - RWR では前処理に2つ，収束判定に1つ MapReduce を追加 (計5つ)
    - Connected Componentsでは収束判定に1つ MapReduce を追加 (計3つ)

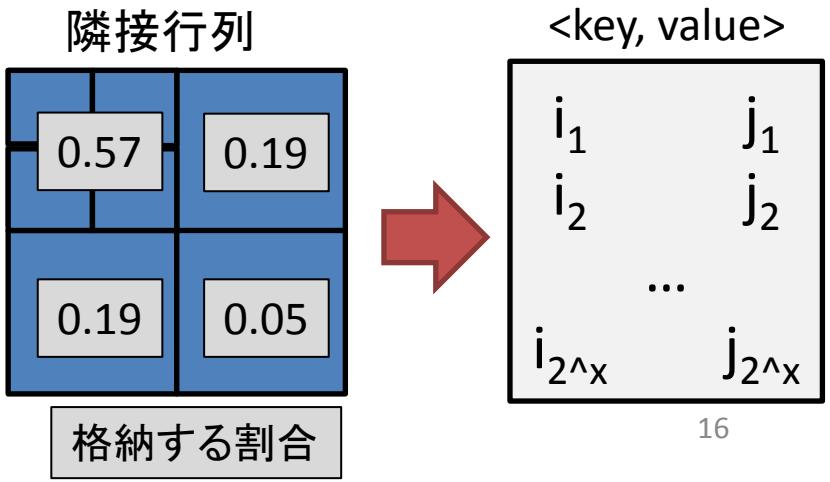


実験

- グラフ処理アプリの反復処理1ループの平均時間を測定
  - Mars と PEGASUS の比較
- ベンチマークアプリケーション
  - PageRank
    - ウェブページの重要度を決定
  - Random Walk with Restart (RWR)
    - 起点ノードと他のノードとの近接性を計算
  - Connected Components
    - グラフの連結成分を探索



- 入力データ
  - Kronecker グラフを人工的に生成し，エッジリストの形式でテキストファイルとして格納
    - Graph 500 の Generator により生成
    - スケールフリー性を持つ
  - SCALE を14～20と変化
    - SCALE: 頂点数の2の対数 (頂点数 = 2^{SCALE})
    - 辺の数 = 頂点数 × 2^4



実験環境

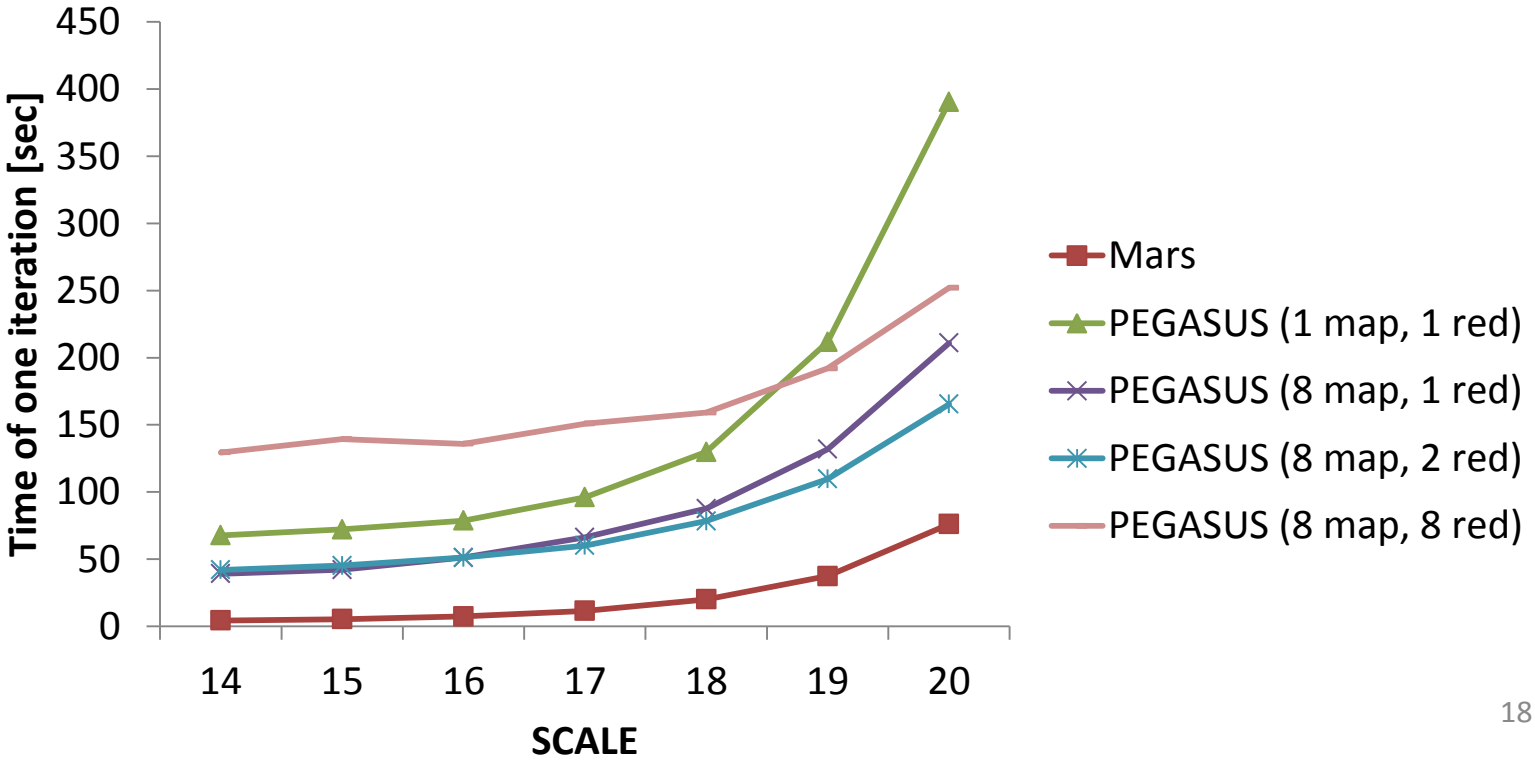
- 1ノードでの実験
- CPU 4コア 8スレッド(HyperThread オン), GPU 1台

	CPU	GPU
種類	Intel® Core™ i7	Tesla C2050
物理コア数	4	448
周波数	2.67 GHz	1.15 GHz
メモリ	12.3 GB	2.8 GB (Global)
コンパイラ	gcc 4.1.2	nvcc 3.2

- GPU
  - CUDA Driver Version: 4.0
  - CUDA Runtime Version: 3.20
  - CUDA Capability: 2.0
  - 共有/L1 キャッシュサイズ: 64KB
- PEGASUS
  - Hadoop 0.21.0, Java 1.6.0 (Oracle Java VM)
  - Mapper, Reducer の数を変化させて実験
  - JVM のヒープサイズは1GB(デフォルト)
  - GPUは使用しない

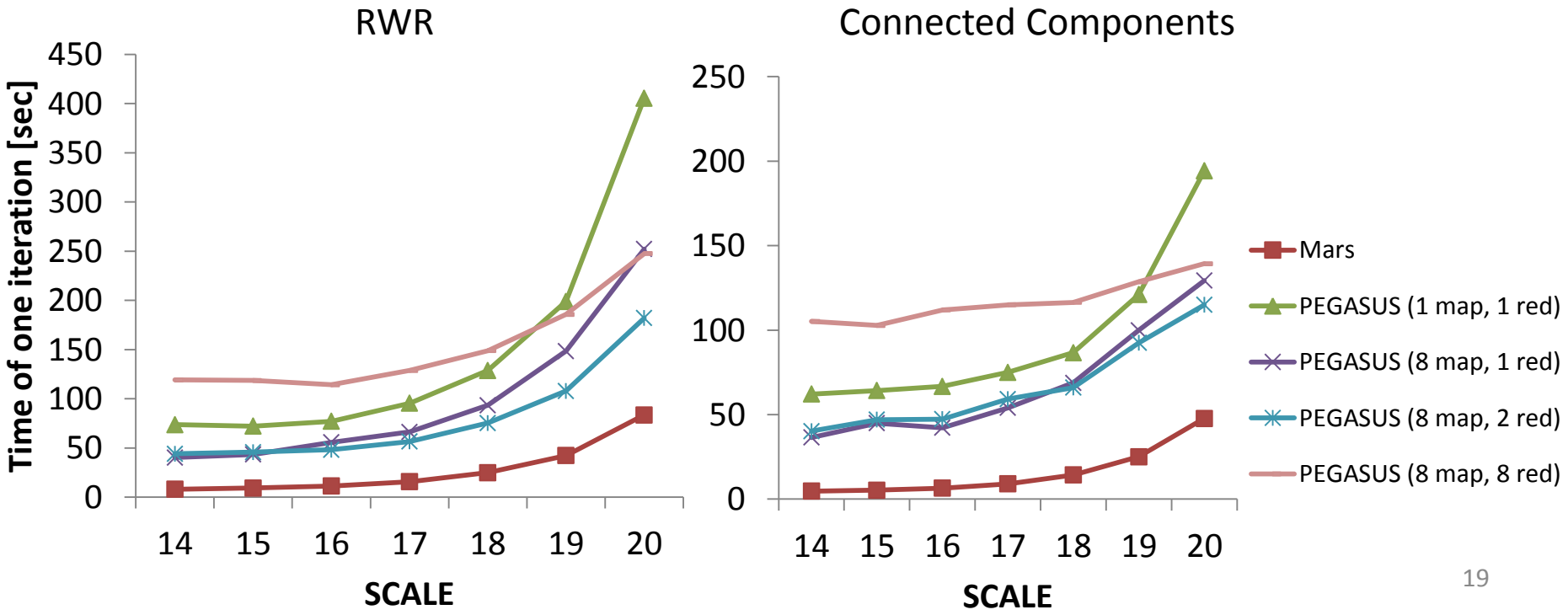
反復1回の実行時間: Mars vs. PEGASUS - PageRank

- Mars，PEGASUS に対して各反復の平均実行時間を測定
  - ファイル入出力や前処理，後処理の時間を含めて測定
- 実行結果
  - Mars では PEGASUS (8 mapper, 2 reducer)に対して2.17～9.53倍高速
  - PEGASUS ではジョブ実行中のファイル入出力によるオーバーヘッド大



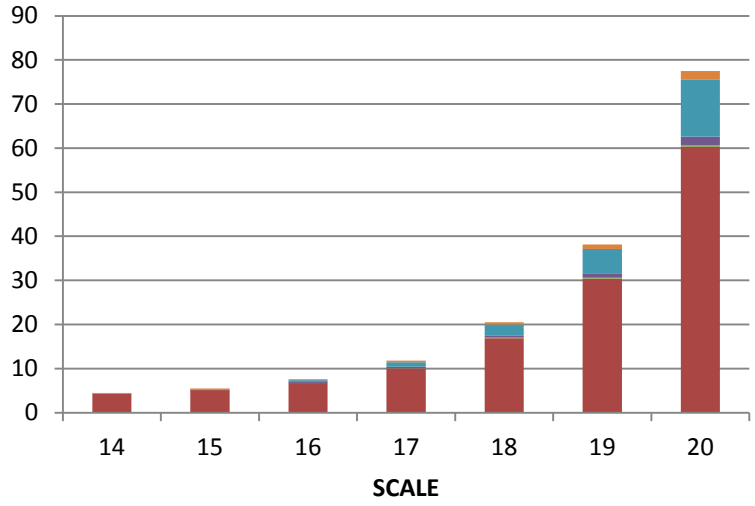
反復1回の実行時間: Mars vs. PEGASUS - RWR, Connected Components

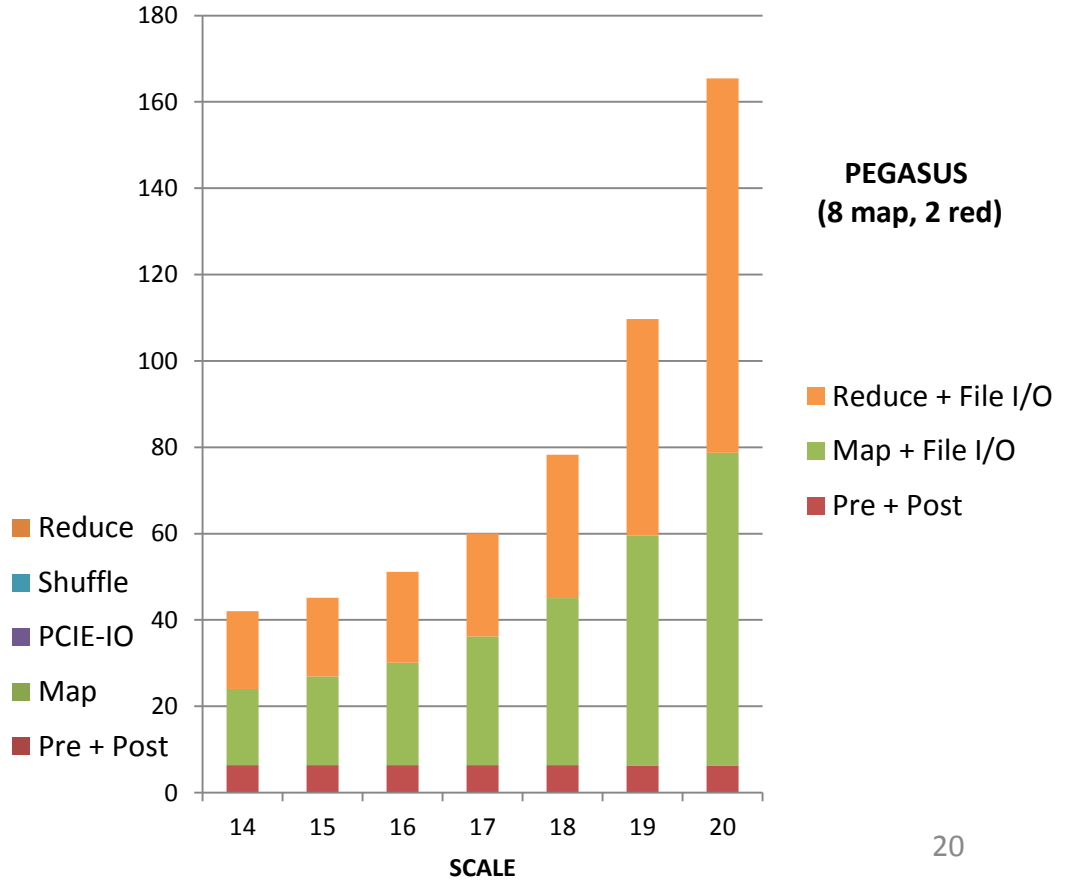
- RWR
  - 反復一回当たり5 つのMapReduce
  - 2.18 ～ 5.47倍の性能向上
- Connected Components
  - 反復一回当たり3 つのMapReduce
  - 2.41 ～ 8.46倍の性能向上
- MapReduceの数が多くなるほど性能向上率が下がる傾向



実行時間の内訳: PageRank

- Mars では，全体の実行時間に対する前処理，後処理の占める割合が大きい
  - MapReduce毎にファイル入出力が発生しているため
- PEGASUS では，Map，Reduce ともに大
  - Map, Reduce はタスクの起動から実行終了までの時間を測定
  - Map や Reduce の実行中に頻繁にファイル入出力が発生





マルチGPU化,
大規模グラフ処理に向けて

- PEGASUS では Mars に比べファイル入出力大
  - PEGASUS では Map, Reduce 毎にファイル入出力
  - Mars では 1つのMapReduce をGPUメモリ内で処理
- Mars におけるファイル入出力オーバーヘッド
  - Mars における MapReduce 間ファイル入出力を削減する必要がある
    - 実装を改善してメモリ内で処理
    - メモリ階層（GPU，CPU，SSD）の管理，マルチGPUにおけるデータの割り振り方法を模索
- MapReduce 処理によるオーバーヘッド
  - GIM-V によりグラフ処理を容易に記述できる一方，ナイーブな実装に比べ無駄が多い
    - ナイーブな実装との比較実験

関連研究

- 既存の大規模グラフ処理システム
  - Pregel*[1] : Master/Worker モデルによるC++の実装
    - Worker毎に頂点を分担して計算
  - Parallel BGL*[2] : MPIベースのC++並列グラフ処理ライブラリ
- GPU，MapReduceを用いたグラフ処理
  - GPUを用いて最短路問題を解くアルゴリズム*[3]
    - 幅優先探索，単一始点最短路問題を高速に計算
  - Graph500 *[4] のリファレンス実装にMapReduceを用いた最短路問題が公開される予定
- マルチGPU上，マルチノード上でのMapReduce実装
  - GPMR*[5] : 既存のマルチGPU上でのMapReduce実装
  - MapReduce-MPI*[6] : MPIを使用したMapReduceライブラリ

→ 今後マルチGPU化を行った上で，大規模グラフ処理における
メモリ階層管理を模索

*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.

まとめと今後の課題

- まとめ
  - GPUを使用したMapReduceによるグラフ処理
  - PEGASUSとの比較
    - PageRankアプリケーションにおいて，反復一回当たり 2.17 ～ 9.53倍の高速化
- 今後の課題
  - 性能の改善
    - ファイル入出力オーバーヘッドの削減
  - GPUメモリに乗り切らない入力データへの対応
    - グラフの構造を考慮した効率的なデータの割り振り
    - メモリ階層の管理
      - GPUメモリ，CPUメモリ，SSD へ格納するデータのスケジューリング
  - マルチGPU化
    - MPIによる複数ノード，マルチGPU処理
      - Map，Shuffle，Reduceの前後にMPIによるデータ交換