将来の大規模メニーコアプロセッサ環境に向けたビッグデータ基盤処理の性能評価

佐藤　仁

産業技術総合研究所　人工知能研究センター

1.　はじめに

近年，ビッグデータ処理が注目されており，SNS，道路の経路探索，スマートグリッド，創薬，遺伝子解析などペタ～ヨッタバイト級のデータに対する高速処理の需要が高まっている．現状のビッグデータ処理は，数台～数百台の安価なWebサーバ由来のクラスタやクラウド上のHDDにファイルとして分散し，Hadoop等のI/Oやネットワーク性能の効率の悪いプラットフォーム上でのリニアスキャンなどの処理が行われている．一方，最新のスループットプロセッサや大容量なメモリ，バーストバッファなどの不揮発性メモリを活用したI/Oアクセラレータを備え，InfinibandやOmniPathなどの広帯域で低遅延な高速ネットワークが接続されたスーパーコンピュータは，超高速な演算性能に加えて強力なI/O性能を提供できるため，ビッグデータ処理基盤として有望であると考えられ，実際，米国NERSC Cori(2016年)，東京大学Reedbushや東京工業大学TSUBAME3（2017年）をはじめ，米国SDSC Gordon(2012年)，米国LLNL Cataryst(2013年)などビッグデータ処理にも適したスーパーコンピュータが登場しはじめている．一方で，このような先進的な環境を活用したデータ処理のソフトウェア基盤は少なく，大規模実行の事例もまだまだ少ないのが現状である．本稿では，将来のメニーコアプロセッサ環境に向けたビッグデータ処理の研究開発の動向として，Graph500,メニーコアプロセッサを考慮したMapReduceや大規模分散ソートなどの具体的な事例について紹介する．

2.　Graph500

グラフは，辺と頂点で記述された連結されたオブジェクトを表現するための基本的な数学的表現である．ヘルスケア，システム生物学，ソーシャル・ネットワーク，ビジネスインテリジェンス，電力網などの様々な重要アプリケーションがグラフを用いてモデル化されている．（図 1）さらに，近年，このようなアプリケーション分野において様々なデータが大量に生成されるようになったため，大規模なグラフに対する高速処理の要求が非常に大きくなっており，高性能計算での重要なカーネルのひとつとなってきている．実際，従来のLinpackによるスーパーコンピュータの計算処理の性能を競うTop500リストに加え，近年，大規模なグラフ処理を行うことでスーパーコンピュータのビッグデータ処理能力を競うGraph500リストが登場してはじめている[1]．図 2にGraph500ベンチマークの概要を示す．現在のベンチマークではスケールフリーや直径の小ささなどの現実のネットワークをモデル化したクロネッカーグラフに対して幅優先探索(BFS)を行った際の実行時間を計測する．幅優先探索とは，ある頂点から隣接している頂点への探索を繰り返し，グラフ内で隣接した頂点を全探索するアルゴリズムである．しかし，現状では，分散メモリで構成されたスーパーコンピュータ上での実行に適した幅優先探索のアルゴリズムには多くの最適化の余地がある．そこで我々は，大規模計算環境でのビッグデータ処理のハードウェア・ソフトウェアに関連する問題点を明らかにし将来のスーパーコンピュータの設計へ活かすことを目的とし、Graph500の開発を進めている．

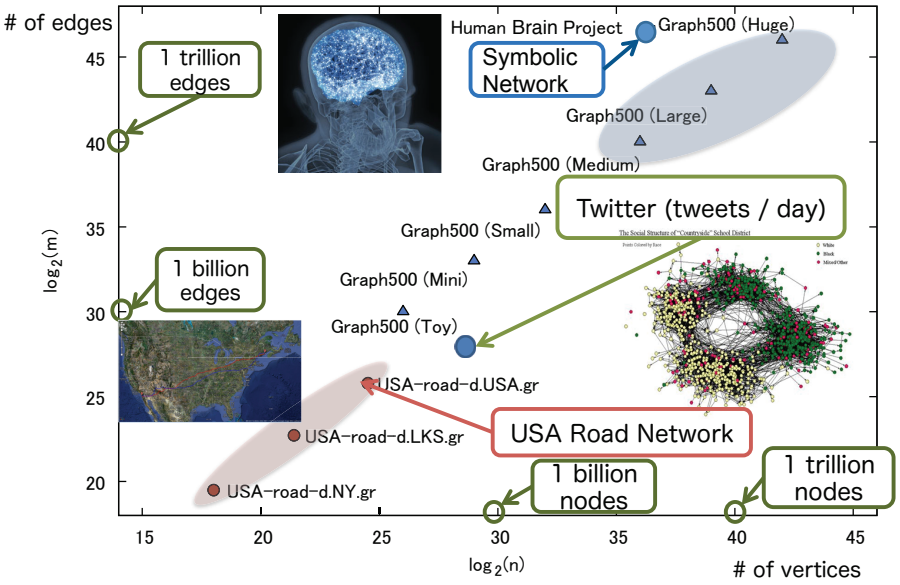

図 1 様々な大規模グラフ

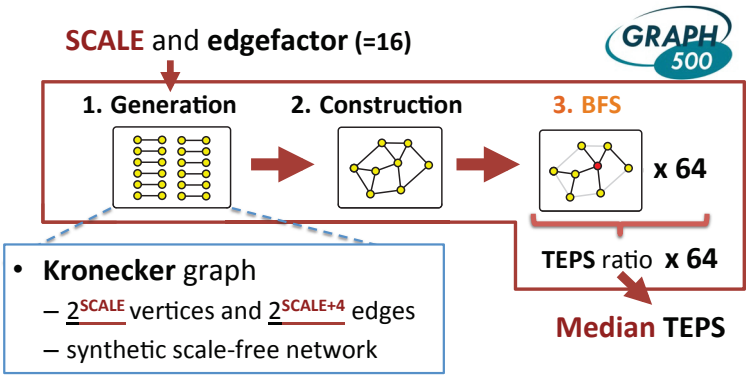

図 2 Graph500 の概要

スーパーコンピュータでは，数千～数万台の計算ノードを使った並列分散アルゴリズムが求められるが，BFS の並列分散アルゴリズムを設計するのは容易なことではない．単純なデータ構造や探索アルゴリズムでは，計算ノードの台数が数百～数千台規模に増えたときネットワーク性能に律速されて全体の性能が頭打ちになってしまう．計算ノード数が数千～数万台規模においても計算ノード数に比例した性能を得て，かつ，効率よく処理を行うためには，以下の条件を満たすアルゴリズム・データ構造が必要となる．

- 大規模に分割したときでも，メモリ使用量とアクセスコストをともに小さくできるグラフ探索向け疎行列のデータ構造
- 通信ノード数と通信データ量がともに低く抑えられる通信アルゴリズム
- グラフの分割数が増えても探索エッジ数を小さく保つ探索アルゴリズム

Graph500 の参照実装では，データ構造や通信アルゴリズムなどの問題により，数百台の計算ノード規模で性能が頭打ちになってしまう．大規模並列分散向け BFS の最新の研究として，２次元分割の Wave を使った手法があるが，この方法は，高速化に有効で他の実装でも広く用いられている Hybrid BFS の手法を使っておらず，最適な方法ではない．また，Hybrid BFS を使った分散並列アルゴリズムも提案されているが，これもデータ構造や通信アルゴリズムの問題により，千台の計算ノード程度で性能が頭打ちになっている．

今回，将来の大規模メニーコアプロセッサ環境に向けたビッグデータ基盤処理の性能評価の一環として，Oakforest-PACS で Graph500 を実行させた．今回の実行では，基本的には，GPU 版 Graph500 上野実装 v1.2 をベースに CPU モードで動作させ，-xMIC-AVX512 -gopenmp などのコンパイラオプション，numactl,taskset などのコンパイラオプション，I_MPI_XXX, KMP_XXX,OMP_XXX などの環境変数など標準的な動作方法で実行したところ，Scale32 で平均次数 16 のグラフ（2^{32} 頂点～2^{36} 辺）を，計算ノード 64 台を用いて，3.94×10^{9} TEPS 程度達成できることを確認した．技術的な内容を広範囲に検討した結果，更なる性能向上を達成するためには，やはり，ソースコードレベルでメモリ確保の際のアライメントを揃えた上で，AVX512 などの SIMD 命令を活用する最適化を導入する必要があることがわかった．今回は，工数の関係からここまでの最適化は実現せず今後の課題となったが，同様のこのような最適化手法の考え方は Intel Skylake 世代以降の標準的な CPU でも応用できそうな見込みである．

3. メニーコアプロセッサを考慮した MapReduce

大規模データ処理の分野では，MapReduce プログラミングモデル[4]が登場し，数千台規模のクラウド上のクラスタ型計算機上で局所性と耐故障性を考慮したスケーラブルなペタバイト-ヨッタバイト級のデータ並列処理を可能にしている．MapReduce では，分散した key-value のペアデータに対して局所性を考慮した統一的な操作を並列で適用するために，並列データ処理のプロセスを Map，Shuffle，Reduce の 3 つの処理に分解する．そして，Map 処理で入力データとなる key-value ペアから中間データとなる key-value ペアを生成し，Shuffle 処理で同じ key に対して value のリストを生成し，最後に Reduce 処理で中間データを Shuffle することにより得られた key と value のリストから最終出力となる key-value のペアデータを生成する．MapReduce は局所性を考慮したデータ並列処理を提供するという点でメモリ空間が階層的に分かれている GPU アクセラレータへも有効な手法であると考えられる．一方，メニーコアプロセッサを搭載した大規模なスーパーコンピュータへ MapReduce 処理を適用した事例は少ない．

そこで，我々は，現在，メニーコアプロセッサや不揮発性メモリデバイスを搭載したスーパーコンピュータ上でスケーラブルなデータ並列処理を目指したソフトウェア基盤として HAMAR(Highly Accelerated Data Parallel Processing Framework for Deep Memory Hierarchy Machines)の開発を進めている．現在の HAMAR の実装では，処理する対象のデータを動的にいくつかのチャンクに分割し，CPU と GPU 間のデータ転送と GPU アクセラレータ上での Map，Shuffle，Reduce 処理をできるだけオーバーラップさせることで，GPU アクセラレータに搭載されたメモリの容量を超えるデータセットに対しても効率のよい処理が可能になっている．また，GPU ベースの外部ソートなど Out-of-core な基本アルゴリズムの実装を多く取り入れている．

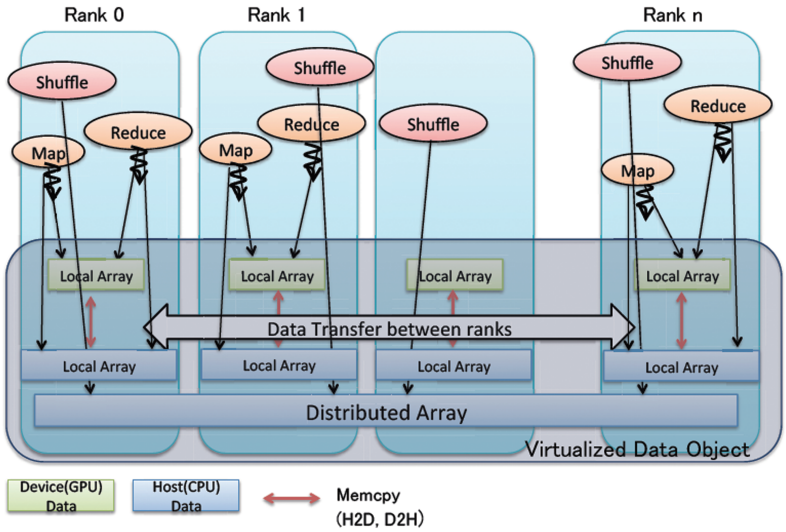

図 3 HAMAR の概要

HAMAR を用いた実装のケーススタディとして，MapReduce 型の大規模グラフ処理である GIM-V (Generalized Iterative Matrix-Vector multiplication)と呼ばれる反復行列ベクトル積の一般化されたアルゴリズム[5]を実装している．いま、i, j が{1, ..., n}の値をとるとし、$M = (m_{i,j})$ を $n \times n$ の行列，$v = (v_i)$ を大きさ n のベクトルとする．ここで，オペレータ $\times_G$ を導入することにより，GIM-V アルゴリズムを次のように定義できる．

$$v' = M \times_G v$$

where $v'_i = assign(v_i, combineAll_i(x_j \mid j = 1 \cdots n, \text{ and } x_j = combine2(m_{i,j}, v_j)))$

上式は，*combine2*, *combineAll*, *assign* という 3 つのオペレータを用いることにより表現される (図 4)．上の操作を，PageRank, Random Walk with Restart, Connected Component といったグラフアルゴリズムの実装によって定義された収束条件を満たすまで反復処理する．

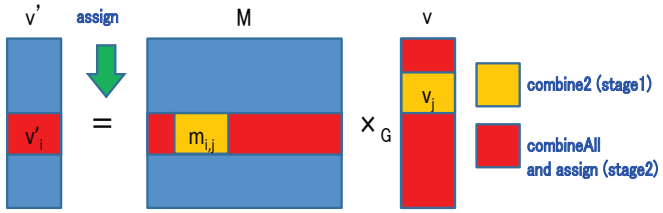

図 4 GIM-V の概要

Hamar を用いて，行列ベクトル積の一般化を MapReduce へ適用した GIM-V により PageRank アルゴリズムを実装し，TSUBAME2.5 の 1024 ノード(12288 CPU コア，3072 GPU)を用いて大規模な実証実験を行った．デバイスメモリの容量を超えるグラフデータ(171.8 億頂点、2749 億枝からなる大規模グラフ)を処理する場合に，1 ノード当たり 3GPU を使用した場合，2.81 GEdges/sec(1 秒当たりに処理した枝数，47.7GB/sec)の性能となり，CPU 上での実行に対して 2.10 倍の高速化を確認した．また，ウィークスケーリングの性能を計測した結果，1024 ノード(3072GPU)を使用した場合に，1 ノード(3GPU)を使用した場合に対して 686 倍の性能向上を

示し，良好なスケーラビリティを示すことを確認した（図 5）[6].

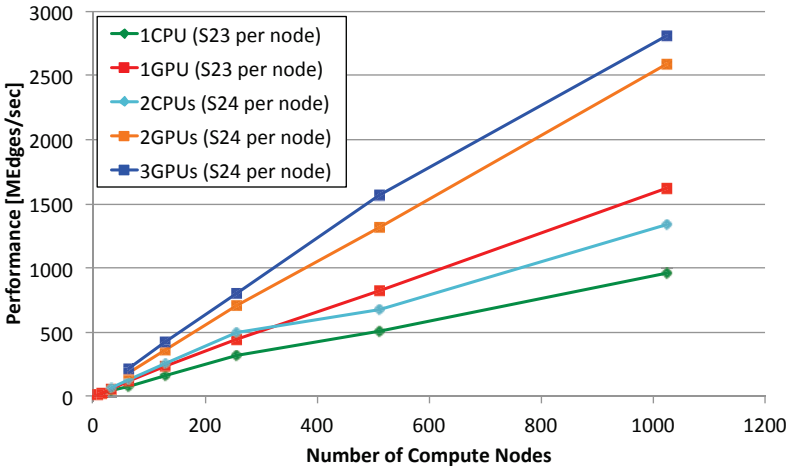

図 5 GIM-V のスケーラビリティ

4. メニーコアプロセッサを考慮した大規模分散ソート

ソートアルゴリズムは最も基本的なアルゴリズムの一つであり，ゲノム解析や多体問題など様々なアプリケーションのカーネルに使用されている．近年，このようなアプリケーションでの急激なデータの増加に伴い，膨大な量のデータをソートするために分散システム向けのソートアルゴリズムが数多く提案されてきた．その中でも，データの通信量を減らす事によって，高速にソートできるアルゴリズムとして，Splitter-based parallel sorting algorithm というアルゴリズムが広く知られている．Splitter-based parallel sorting algorithm では，スプリッターを導入することでデータの通信コストを減少させ，高速化に成功したが，相対的に計算処理のコストが高まり，全体の処理時間の大部分を占めるようになることが問題となっている．

Splitter-based parallel sorting algorithm では，以下のような手順でソートが行われる（図 6）.

1. それぞれのプロセスに配置されたデータを独立にソートする.
2. その後，データの転送時の基準となるスプリッターを選択する.
3. 選択したスプリッターに合わせてプロセス間で通信を行い，データの転送を行う.
4. 最後に，マージ操作でプロセス上のデータをソートされた状態にする.

分散システム用のマージソートや基数ソートではデータ転送フェイズが数多く行われるが，このアルゴリズムではスプリッターを元にしたデータ転送によって通信回数を減らすことで通信コストを下げている.

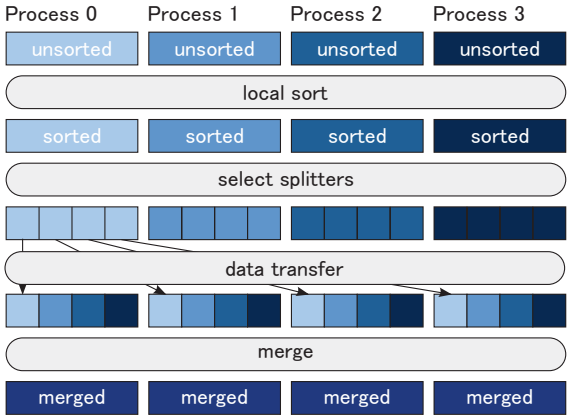

図 6 Splitter-based Parallel Sorting Algorithm

我々は，Splitter-based parallel sorting algorithm の一つである HykSort[7]を拡張し，メニーコアプロセッサの代表的な類型として GPU アクセラレータによる高速化を行った．実装では，最もコストの大きい計算処理である各プロセス上でのソート処理を GPU 上で行うように変更した．GPU メモリは DRAM と比べると非常に小さいため，GPU メモリを溢れるようなサイズのデータをソートする時には，複数個にチャンクを分けてソートを行い，最後にソートされたチャンクをマージすることで対応している．

以下は，メニーコアプロセッサ版の hyksort の性能を GPU スパコン(TSUBAME2.5)上最大 1024 ノードを使用して行ったものである．各ノード上ではソケットあたりに 1 つのプロセスが配置され，各プロセスが 2GB の 64 bit 整数のデータを所有している状況で開始する．弱スケーリングの実験では，2048 プロセス(1024 ノード，2048GPU)の時に 0.25TB/s 程のスループットが得られた．これは CPU 1 スレッドのみの実装と比べると 3.61 倍，CPU 6 スレッド並列のものと比べると 1.40 倍の性能となっている（図 7）[8]．

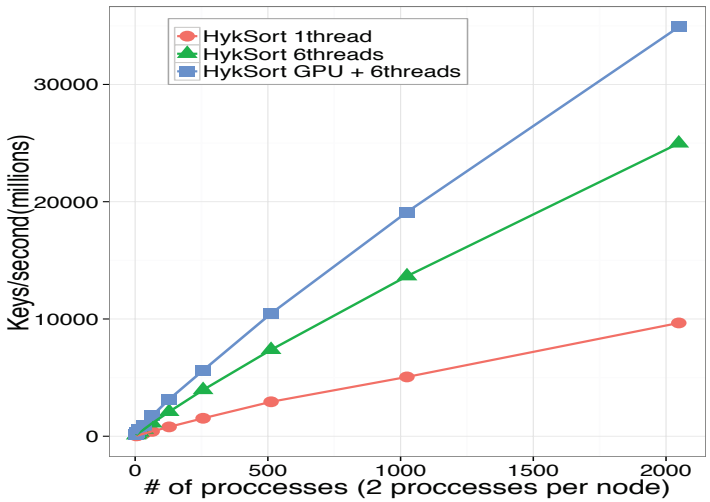

図 7 hyksort のスケーラビリティ

5. おわりに

本稿では，将来の大規模メニーコアプロセッサ環境に向けたビッグデータ処理の研究開発の動向として，Graph500，メニーコアプロセッサを考慮したMapReduce処理や大規模分散ソートなどの具体的な事例についての紹介を行った．これらの事例は2018年頃に産総研に導入されるABCI(AI Bridging Cloud Infrastructure)[9]などのエクストリームなビッグデータ処理を指向した大規模計算基盤アーキテクチャの設計などに活かされる予定である．

参考文献

[1] Graph500: http://www.graph500.org.

[2] Fabio Checconi, Fabrizio Petrini, Jeremiah Willcock, Andrew Lumsdaine, Anamitra Roy Choudhury, and Yogish Sabharwal, “Breaking the speed and scalability barriers for graph exploration on distributed-memory machines”, Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC'12), Article No.13, p13:1-13:12, 2013.

[3] Scott Beamer, Krste Asanović, and David Patterson, “Direction-optimizing breadth-first search”, Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC'12), Article No.12, p12:1-12:10, 2012.

[4] Jeffrey Dean and Sanjay Ghemawat, “MapReduce: simplified data processing on large clusters”, Communications of the ACM, Vol. 51, Issue 1, p107-113, 2008.

[5] U. Kang, Charalampos E. Tsourakakis, and Christos Faloutsos, “PEGASUS: A Peta-Scale Graph Mining System Implementation and Observations”, Proceedings of the 9th IEEE International Conference on Data Mining (ICDM '09), p229-238, 2009.

[6] Koichi Shirahata, Hitoshi Sato, Satoshi Matsuoka, “Out-of-core GPU Memory Management for MapReduce-based Large-scale Graph Processing”, 2014 IEEE International Conference on Cluster Computing (Cluster2014), p221-229, 2014.

[7] Hari Sundar, Dhairya Malhotra, and George Biros, “HykSort: a new variant of hypercube quicksort on distributed memory architectures”, Proceedings of the 27th international ACM conference on International conference on supercomputing, p293-302, 2013.

[8] Hideyuki Shamoto, Koichi Shirahata, Aleksandr Drozd, Hitoshi Sato, Satoshi Matsuoka,” GPU-Accelerated Large-Scale Distributed Sorting Coping with Device Memory Capacity”, IEEE Transactions on BigData, Vol.2, Issue 1, p57-59, 2016.

[9] 小川宏高, 松岡聡, 佐藤仁, 高野了成, 滝澤真一朗, 谷村勇輔, 三浦信一, 関口智嗣, “AI橋渡しクラウド-AI Bridging Cloud Infrastructure (ABCI)-の構想”,情報処理学会研究報告ハイパフォーマンスコンピューティング(HPC)2017-HPC-160(28),p1-7, 2017.