大規模グラフ処理ベンチマークGraph500のTSUBAME 2.0 における挑戦

鈴村 豊太郎 */** **上野 晃司** *
*東京工業大学大学院・情報理工学研究科 **IBM Research – Tokyo

Graph 500とは、スーパーコンピュータのグラフ処理性能を測定する新しいベンチマークである。
スパコンのベンチマークでは、数値計算性能を測るLinpackによるTop 500が有名だが、近年、大規模グラフ処理が、重要性を増しており、Graph500ベンチマークが広がりを見せている。Graph500のリファレンス実装は、使用されているアルゴリズムの問題により、分散メモリ環境で大規模にスケールさせることができなかった。そこで、大規模にスケール可能な2次元分割に注目した。本論文では、2次元分割をTSUBAME2.0上に実装し、1366ノードで頂点数2^36(68.7 billion)、エッジ数2^40(1.1 trillion)のグラフ(Graph500のScale 36)のBFS(幅優先探索)を10.955秒で計算した。
TEPS値は100.366 GE/sであり、2011年11月に発表されたランキングでは世界3位を獲得した。

1 はじめに

大規模グラフ処理はWebページのリンク解析、タンパク質間の相互作用解析、サイバーセキュリティ、VLSIのレイアウトや道路網、送電網の最適化など様々な応用分野あり、近年盛んに研究されている。従来、スーパーコンピュータは物理シミュレーションなどの数値計算に、主に使われてきたが、大規模グラフ処理も重要なアプリケーションとなりつつあり、そのような中、スパコンのグラフ処理性能を測る、Graph500[1]という新しいベンチマーク登場し、注目を集めている。Graph500は、スパコンの通信性能や、グラフデータを格納するメモリの大きさや、メモリへのランダムアクセス性能を測るという、データインテンシブなベンチマークであり、数値計算性能を測るTop 500ベンチマークとは計測する性能が全く異なる。本論文では、Graph500の概要、我々が提案するスケーラブルな最適化実装とTSUBAME2.0 における性能評価について述べる。

2 Graph500の概要

本章では，Graph500ベンチマークの概要と、分散BFSアルゴリズムについて述べる。

2-1 Graph500ベンチマークの概要

Graph500は、大規模なグラフに対してBFSによる探索を実行するベンチマークである。単位時間に処理できたエッジ数と、扱える最大問題サイズが評価指標となる。計算インテンシブなTop500ベンチマークと違い、Graph500ベンチマークは、データインテンシブなベンチマークである。扱える問題サイズは、グラフの頂点数=2SCALEであるようなSCALE値で表す。単位時間に処理できたエッジ数は、TEPS (Traversed Edges Per Second) 値で表す。例えば、100万TEPSとは、100万個の枝を持つ連結グラフのBFSが1秒で完了した場合の性能である。

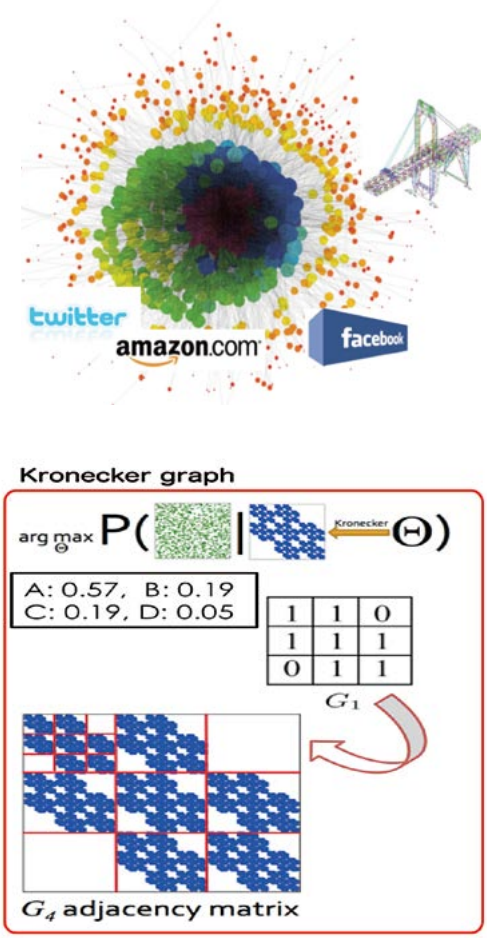

図1 クロネッカーグラフ[4]

ベンチマークを実行するプログラムは、(a)グラフデータの生成、(b)計算するのに最適なデータ構造への変換、(c)BFSによる探索、(d)計算結果の検証の4つの部分から成る。ベンチマークの実行順は次のようになっている、最初に(a)、(b)によりグラフデータを構築し、グラフから始点を64個選ぶ。次に、64個の始点それぞれに対して順番に、(c)BFSによる探索と、(d)計算結果の検証を行う。複数の始点からの探索を同時に行うことはできない。時間を計測しベンチマークとする部分は、(b)のグラフデータ構造への変換(Kernel 1) と、(c)

大規模グラフ処理ベンチマークGraph500のTSUBAME 2.0 における挑戦

のBFSによる探索(Kernel 2) のみである。(a)では、枝数が頂点数の16倍となるようなクロネッカーグラフ[4]を生成する。枝はすべて重みなし、無向辺である。ここで生成されるデータは規則性のない順番で並んだ、枝のリストである。(b)では(a)で生成された枝リストから、隣接行列のCSR (Compressed Sparse Row)や、CSC (Compressed SparseColumn)などのグラフデータ構造に変換する。(c)のBFSは、BFSで辿った頂点の軌跡であるBFS木を出力する。(d)では、このBFS木が正しいかどうかチェックする。このチェックでは、BFS木にループがないこと、枝の張っている頂点同士の深さの差が1以下であること、などの5つのルールを満たしていることをチェックする。

2-2 分散 BFS アルゴリズム

Graph500のリファレンス実装には、OpenMPで書かれた実装や、MPIで書かれた実装、Crayの共有メモリ型プログラミング環境用の実装、など複数の種類が用意されている。TSUBAME2.0で分散実行するには，MPIで書かれた実装を使用する。MPIで書かれた実装には、さらにアルゴリズムや実装方法の異なる4種類の実装が用意されている。これらの実装は、対象としているプログラミング環境や分散方法などは異なるが、全てベースとなるアルゴリズムとしてLevel-synchronized BFSを使っている。このアルゴリズムは、各レベル(深さ) について、そのレベルの頂点をすべて処理してから、次のレベルに進むというアルゴリズムである。

Algorithm I: Level-synchronized BFS

```
1   for all vertex v in parallel do
2   | pred[v]← -1;
3   pred[r] ← 0
4   Enqueue(CQ r)
5   While CQ != Empty do
6   |   NQ ← empty
7   |   for all u in NQ in parallel do
8   |   |   u ← Dequeue(CQ)
9   |   |   for each v adjacent to u in parallel do
10  |   |   |   if pred[v] = -1 then
11  |   |   |   |   pred[v] ← u;
12  |   |   |   |   Enqueue(NQ, v)
13  |   swap(CQ, NQ);
```

アルゴリズム1はLevel-Synchronized BFSの擬似コードである。まず、BFS木を格納するPREDと、頂点が訪問済みかどうかを格納するVISITEDを初期化する。PRED[v]は頂点vのBFS木における親頂点を表す、初期値-1はBFS木にまだ入っていないことを表している。VISITED[v]は頂点vが訪問済みかどうかを表す。初期値0はまだ訪問していないことを表している。次に、BFSの始点となる頂点をCQ (Current Queue) に入れ、探索を開始する。

探索においては、7~16行の1ループが1レベルに相当する。このループ中で、CQは現在のレベルで訪問する頂点、NQ (Next Queue)は次のレベルで訪問する頂点が格納されている。例えば、レベル1でCQに頂点γが入っていたとすると、11、12行目でγの隣接頂点が訪問済みかどうかチェックされ、まだ訪問していない頂点はNQに格納される。次のレベルでCQにはこれらの頂点が格納されていることになる。9行目のforと、11行目のforは並列化が可能なループである。リファレンス実装の4つのMPI実装は、基本的にはLevel-synchronized BFSを実装しているが、グラフデータの分散方法などに違いがある。4つのリファレンスMPI実装の処理の仕方の違いやTSUBAME 2.0 上における性能特性の結果は我々の先行研究[2]を参照して頂きたい。

3 2 次元分割によるスケーラブルな実装

リファレンス実装は全て1次元分割を使っているが、1次元分割はスケールさせることが難しい[2]。そこで、隣接行列を2次元に分割するアルゴリズム(2次元分割)[3]を実装したプロセッサをP = R x Cの2次元メッシュ (mesh)に配置する。このメッシュの行を「プロセッサ行」、列を「プロセッサ列」と呼ぶことにする。隣接行列を図4のようにR*C個の行とC個の列に分割し、プロセッサ(i, j)は、隣接行列のA_(i,j)^((1))~A_(i,j)^((C))のCブロックを担当する。頂点は、R x C個のブロックに分割し、プロセッサ(I, j)は、j*R+I 番目のブロックを担当する。1レベルにつき、expandとfoldの2段階の通信を行う。各プロセッサは自分の担当する頂点ブロックのCQを同じプロセッサ列の他のプロセッサに送信する。これをExpandという。Expandは1次元分割の縦分割と同じように、CQをコピーする通信であるが、隣接行列は横にもC個に分割されているので、通信は、同じプロセッサ列の他のプロセッサとだけ行う。次に、各プロセッサはCQと各プロセッサが持っている部分隣接行列から、CQの隣接頂点を探す。PREDやNQを更新するため、CQの隣接頂点を、その頂点の担当プロセッサに送信する。この通信をFoldという。PREDを更新するのに、親の頂点が必要なので、Foldでは、CQの隣接頂点と、親頂点(CQの頂点) の組みを送信することになる。Foldは1次元分割の横分割と同じように、CQの隣接頂点を担当プロセッサに送信する通信だ。しかし、2次元分割では、隣接行列の分割方法から、Foldの通信を行う必要のある相手は、同じプロセッサ行の他のプロセッサのみとなる。

2次元分割の利点は、通信で絡むプロセッサ数が少ないことである。1次元分割では、2種類の分割方法のどちらも、全対全の通信が必要だったのに対し、2次元分割の場合、Expandでは同じ列のノード(R-1)プロセッサと、foldでは同じ行のノード(C-1)プロセッサとしか通信を行わない。よって、通信するプロセッサ数を少なくすることができ、大規模に分散可能になる。

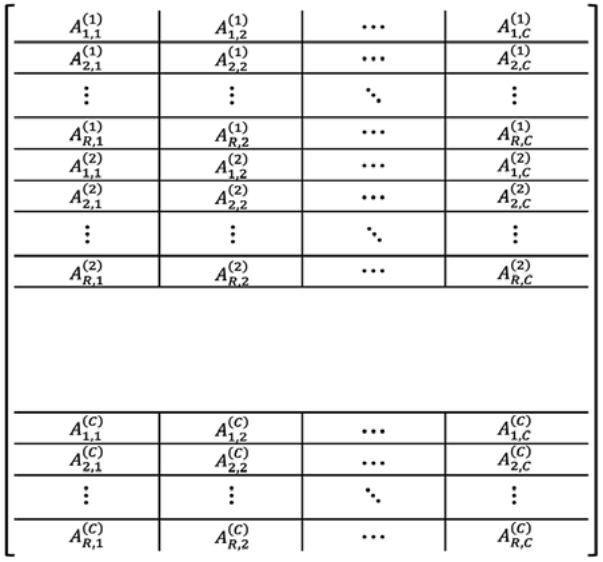

図2 隣接行列の2次元分割

4 性能評価

TSUBAME2.0上での性能評価の結果について述べる。TSUBAME2.0は、1400以上のノードがFat-Treeによるフルバイセクションのインフィニバンドネットワークで接続されている。各ノードには、Intel CPU Xeon 5670 2.93GHz（Westmere EP、6コア、256-KB L2 キャッシュ、12-MB L3）が2つ、NVIDIA M2050 GPU（Fermi）が3つ、48GBのメモリが搭載されている。通信は、各ノードはInfiniband QDRが2リンク使用可能で、合計80Gbpsの通信バンド幅を備えている。

最大1024ノードまで使用して実験した。なお、TSUBAME2.0はGPUメインのスパコンだが。GPUは使用していない、TSUBAME2.0は1ノードあたり物理コア12個だが、SMTを有効にすると仮想的に24コアになる。1ノード24コアとして、各プロセスに均等に割り振った。gcc 4.3.4（OpenMP 2.5）、MVAPICH2 1.6[4]。比較するリファレンス実装は、執筆時点で最新のversion 2.1.4である。

図3は2次元分割とリファレンス実装の比較である。横軸はノード数で縦軸はTEPS（GE/s）である。リファレンス実装のreplicated-csr、replicated-cscと最適化実装は、1ノードあたり2MPIプロセスで実行し、Simpleは1ノードあたり16MPIプロセスで実行した。図7は図6の性能をノード数で割り、1ノードあたりの性能を算出したグラフである。リファレンス実装のsimpleを参考に掲載した。リファレンス実装で、データがない部分はエラーなどで計測できなかったところである。

2次元分割の実装は、リファレンス実装のsimpleの2倍程度の速度が出ている。これは、送信処理と受信処理の並列化や、OpenMPによるプロセス内の並列化の効果によるものである。2次元分割の実装は、リファレンス実装のreplicatedと比べると、性能が低い。これはreplicatedのアルゴリズムはノード数が小さい場合には通信データ量を小さくすることができ、有利だからである。図3から分かるように、replicatedの優位性もノード数が増えるにしたがって急激に低下し、通信データ量は512ノードで2次元分割と逆転する。実際、図3からreplicated-cscはノード数128で既に性能の限界が見え始めている。また、図4は1ノードあたりのWeak Scalingによるスケーラビリティの評価だが、1024ノードまでノード数を増加させても性能が向上し、十分なスケーラビリティが得られていることがわかる。

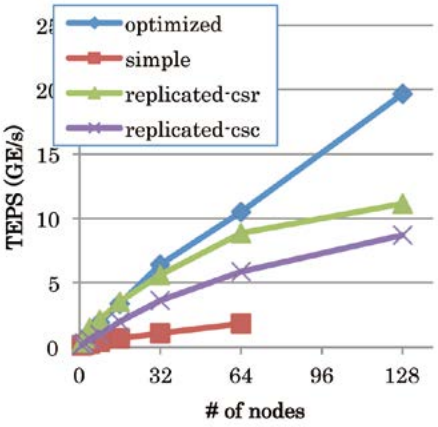

図3 2次元分割と参照実装の比較

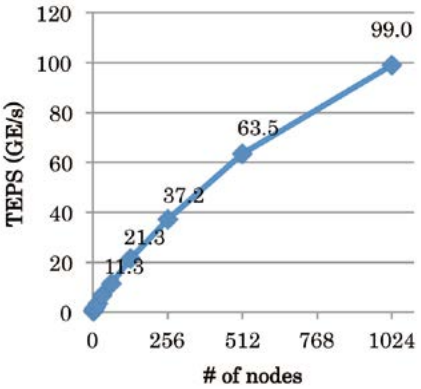

図4 1024ノードまでのスケーラビリティ

まとめと今後の展望 5

本論文では大規模分散環境でスケールさせるため2次元分割によるBFSを実装した。2011年11月のGraph500におけるスコアは、1366ノードで頂点数2^36(68.7 billion)、エッジ数2^40(1.1 trillion) のグラフ(Graph500のScale 36)のBFS(幅優先探索)を10.955 秒で計算した。TEPS値は100.366 GE/sであり、2011年11月に発表されたランキングでは世界3位を獲得した。我々は、2次元分割の他に、通信データの圧縮や頂点の並び替えなどによる最適化も行なっている。
それらの成果は、また別の機会に発表する。

謝辞

本研究の成果は、TSUBAME2.0グランドチャレンジ制度、科学技術振興機構CREST「ポストペタスケール高性能計算に資するシステムソフトウェア技術の創出」から支援を頂いた。

参考文献

[1] Graph500 : http://www.graph500.org/.
[2] Toyotaro Suzumura, Koji Ueno, Hitoshi Sato, Katsuki Fujisawa and Satoshi Matsuoka, "Performance Evaluation of Graph500 on Large-Scale Distributed Environment", IEEE IISWC 2011 (IEEE International Symposium on Workload Characterization) , 2011/11, Austin, TX, US
[3] Andy Yoo, et al, A Scalable Distributed Parallel Breadth-First Search Algorithm on BlueGene/L. SC 2005.
[4] J. Leskovec, D. Chakrabarti, J. Kleinberg, and C. Faloutsos, "Realistic, mathematically tractable graph generation and evolution, using kronecker multiplication," in Conf. on Principles and Practice of Knowledge Discovery in Databases, 2005.