pノードセンター問題の近似解法のGPGPUクラスタを用いた並列実装

2009SE072 稲本 裕貴　2009SE277 竹内 伊吹
指導教員 宮澤 元

1　はじめに

大規模グラフにおけるさまざまな配置問題を解くことは，現実の道路網における最適配置問題やセンサネットワークのスケジューリング問題など幅広い分野で役立つ．これらの配置問題の多くはNP(Non-deterministic Polynomial time) 困難であり，厳密解を求めるのにグラフが大規模になる程時間がかかる．そこで近似解を求めるヒューリスティクスが研究されてきた．例えば，古田らは，ネットワークボロノイ図を用いたpノードセンター問題の近似解法を提案している [1]．しかし，数千〜数万ノード以上の大規模グラフでは近似解法であっても十分高速であるとはいえない．

一方で，複数のPCをネットワーク接続したクラスタ環境や，GPGPU(General-Purpose computing on Graphics Processing Units) といった並列計算環境が比較的安価に利用できるようになってきている．またGPGPUを利用するためのCUDA[2] や，クラスタ環境で並列計算を行うためのライブラリであるOpenMPI[3] などのソフトウェア開発環境も整いつつある．しかし，並列計算のための問題のタスク分割や，タスクのプロセッサへの割り当ては一般には自明では無い．

本稿では，マルチコアCPUとGPGPUを備えたPC複数台をLANで接続したクラスタ (GPGPUクラスタ) を想定し，CUDAとMPIを用いてネットワークボロノイ図を用いたpノードセンター問題の近似解法を並列に実装する方法について述べる．またこの近似解法で必要となるボロノイ分割の繰り返しを効率化するためのアルゴリズムの最適化についても述べる．

2　pノードセンター問題とその近似解法

大規模グラフの配置問題のひとつにpノードセンター問題がある．pノードセンター問題は，需要点と枝から構成されるネットワーク (グラフ) 上に，需要点とその最寄りの施設との最大距離が最小となるようにp個の施設を配置する問題である．最大距離を最小にすると言う性質から現実の道路網における消防署や病院などの緊急施設の最適配置問題，センサネットワークのスケジューリング問題などに適しているとされている．

2.1　ネットワークボロノイ図を用いた近似解法

ネットワークボロノイ図はグラフ上に母点 (ジェネレータ) が与えられたとき，各頂点からの距離がグラフ上で最も近い母点によってグラフを分割するものである．分割された領域をボロノイ領域と呼ぶ．

このネットワークボロノイ図を用いてpノードセンター問題の近似解を求める近似解法の概略を示す．

1. p個のジェネレータをランダムに選ぶ．
2. グラフをジェネレータごとのボロノイ領域に分割する．
3. 各ボロノイ領域で1ノードセンターを求める．
4. 求めた各領域のセンターノードと，元のジェネレータを比較し，求めたセンターノードとジェネレータが全て等しければ，求めたp個のセンターがそのグラフのpノードセンターの近似解とする．センターノードとジェネレータが異なる時，求めたセンターノードを新しいジェネレータとして2.へ戻る．

この近似解法では最初に選ぶジェネレータの組合せによっては局所最適解で終了してしまう可能性がある．そこで初期値を変えて複数回実行するマルチスタートを用い，その中で最適なジェネレータの組合せを最適解とする．

2.2　並列解法

上記のpノードセンターの近似解を求めるアルゴリズムは以下の2つの観点から容易に並列化できる．

最短路問題　ジェネレータからネットワークボロノイ図を求める時に単一始点最短路 (SSSP:Single Source Shortest Path) 問題を複数回計算する必要があり，また1ノードセンターを求める時に全点対最短経路 (APSP:All-Pairs Shortest Path) 問題を解くためにSSSP問題を繰り返し計算する必要がある．これらの最短路問題を並列計算することができる．また，SSSP問題自体を並列に計算することもできる．

ボロノイ領域　各ジェネレータのボロノイ領域ごとに1ノードセンターを求める時，各ボロノイ領域は独立で他のボロノイ領域における計算の影響は受けないので，ボロノイ領域ごとに並列計算することができる．

文献 [4] では，文献 [1] の近似解法をこれらの視点からMPIを用いて並列実装している．

3　GPGPUクラスタにおける並列実装

GPGPUクラスタ上でネットワークボロノイ図を用いたpノードセンター問題の近似解法を実装する方法について述べる．また，ネットワークボロノイ分割を効率的に繰り返すためのアルゴリズムの最適化についても示す．

3.1　実装の概要

本実装では2.2節の最短経路問題の観点からCUDAを用いて並列化した実装 [5, 6] と，2.2節のボロノイ領域の観点からOpenMPIを用いて並列化した実装を組み合わせる．

CUDAを用いた所は，SSSP問題を解いてジェネレータからボロノイ領域に分割する箇所，SSSP問題を繰り

返し解きボロノイ領域の 1 センターを求める箇所である．MPI を用いた所は，ボロノイ領域の 1 センターを求める箇所である．
以下に実装の流れを示す．

1. 1CPU に 1MPI プロセスを割り当てる．各 MPI プロセスは rank と呼ばれる固有の番号を持っている．
2. 各 MPI プロセスで同じグラフのデータファイルからグラフを作成する．
3. 各 PC(ホストノード) の一番小さい rank から，ホストノードが持っている GPU の個数分だけ CUDA で計算するフラグをたてる．
4. GPGPU クラスタ上でネットワークボロノイ図を用いて p ノードセンター問題の近似解法を解く．
   4-1. p 個の頂点をグラフ全体から乱数で選択し，その頂点集合をジェネレータとする．
   4-2. rank0 で CUDA を使用して，ジェネレータからネットワークボロノイ図を求め，全てのプロセスに領域の情報を送る．このとき領域内の総ノード数の降順に，各領域に領域番号を付ける．
   4-3. CUDA と OpenMPI を用いて各領域のセンターを求める (図 1).

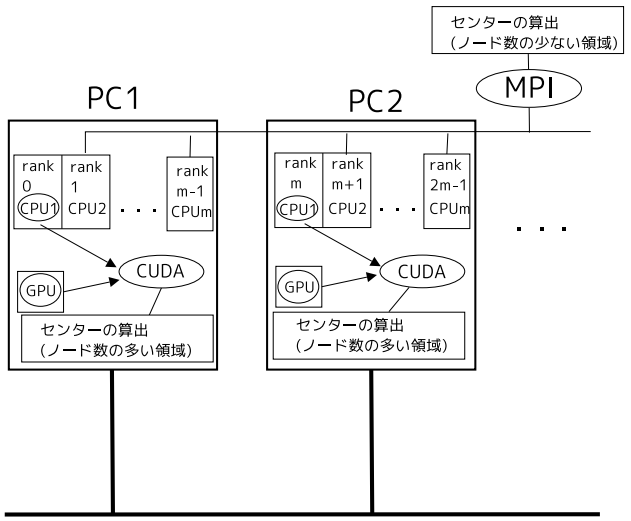

図 1 1 ノードセンターを算出する

   4-4. MPI で求めたセンターとそれに付随する情報は，GPU を使用していない全ての rank の中で rank が一番小さい MPI プロセスに集め rank0 に送る．GPU で求めたものはそれぞれ rank0 に集約する．センターとそれに付随する情報を全て rank0 に集めた後全てのプロセスに計算した全ての領域についてのセンターなどの情報を送信する．
   4-5. もし p ノードセンター問題をネットワークボロノイ図を用いて解く方法の終了条件を満たしていれば終了．満たしていなければ求めたセンターをジェネレータとして 4-1. へもどる．

3.2 計算機資源の効率的な利用

前節の 4-3. で CUDA と MPI にどのように領域を割り当てるべきかは自明ではない．1 台の PC において MPI プロセスと GPU の個数は等しくないので，同時に複数の MPI プロセスが GPU を利用することはできない．これは，GPU を利用するためには同じホストノードの CPU で CUDA カーネル関数を起動しなければならないからである．

そこで，1 センターを求める領域の CUDA と MPI への割り当ての方針として，総ノード数が多い領域に GPU を割り当て，そして総ノード数が少ない領域に MPI を割り当てて計算をさせることにした．具体的には各領域の総ノード数を調べ各領域に総ノード数の降順に領域番号をつける．そして，領域番号の小さい領域を CUDA に領域番号の大きなものに MPI を割り当てる (図 1).

実装するにあたって，CUDA と MPI の領域の割り当て方法を，静的に基準を決めて割り当てた．この割り当て基準は，領域の総ノード数が基準以上ならば CUDA に，基準未満ならば MPI に割り当てるものである．これは実験で，CUDA による 1 センターの計算は，小さすぎる領域では性能が出ず，C で計算するよりも遅くなってしまうことがわかったからである (4.3 節参照).

例として，2 台のホストノードがそれぞれ 4 個の CPU と 1 個の GPU を持つ計算環境において，割り当て基準値を 10 として，5 ノードセンター問題を解くケースを示す．グラフがボロノイ分割されており，5 つの領域 R0 から R4 に分割されている．各領域の総ノード数は R0 の 16 個が一番多く続いて R4 の 15 個，R2 の 11 個，R3 の 6 個となり一番ノード数が少ない R1 の 4 個となっている．最初のセンターを求める時，ノード数が多い領域である領域 R0，R4 を GPU に割り当て，一番ノード数が少ない領域 R1 を MPI を用いて GPU を使用していない CPU 全てを用いて解く (図 2)．そして，割り当て基準値は 10 なので GPU0 は R0 の 1 センターが計算し終えたら次に R2 を求める．MPI は次に R3 を求める．

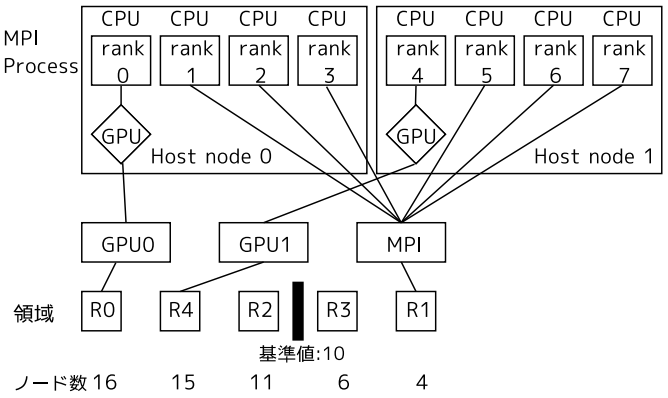

図 2 領域割り当ての例

3.3 ネットワークボロノイ分割アルゴリズムの最適化

グラフをボロノイ分割する際，設定されたジェネレータについて SSSP 問題を解くことで，全てのノードにつ

いて，どのジェネレータに一番近いかを調べることができる．これで，全てのノードはどのジェネレータの領域に属するか決定される．各領域でセンターを調べ，求めたセンターをジェネレータとして再びボロノイ分割する際，上記の分割方法を再び実行する．しかし再びボロノイ分割する際，前のボロノイ分割の時の全てのノードについて一番近いジェネレータまでの距離コストの計算結果を捨ててしまっているので，無駄である．

そこで，1 つ前のボロノイ分割の結果，すなわち全てのノードについて一番近いジェネレータまでの距離コストを差分グラフとして記録しておく．そして再びボロノイ分割する際，各領域について求めたセンターとジェネレータが違う領域だけを，差分グラフを用いてボロノイ分割するという方法をとった．

実装の概略を示す．

1. 1 回目のボロノイ分割を行う．このとき，全てのノードは一番近いジェネレータまでの距離コストを覚えておく，このグラフを差分グラフとする．
2. 各領域でセンターを求め，各領域のどれか一つでもジェネレータとセンターが異なっている場合，再びボロノイ領域にグラフを分割する．
3. ジェネレータと求めたセンターが変わっている領域について，差分グラフの領域内全てのノードの距離コストを∞にし所属領域の情報を新しく求めたセンターに属するようにする．また，新しく求めたセンターの距離コストは 0 にする．ジェネレータと求めたセンターが変わっていない領域はそのままにする．
4. ジェネレータと求めたセンターが変わっている領域のセンターから，差分グラフの距離コストの更新を更新ができなくなるまで行う．また，領域を跨いで距離コストの更新を確認する時は，距離コストの確認をされたノードからも逆に距離コストの更新できるかを確認する．
5. ジェネレータを新しく求めたセンターにする．

4 計算実験

並列計算の性能確認のためと，提案実装の評価のために計算実験を行った．

4.1 実行環境

実験は以下のような仕様の PC を用いて行った．

- OS:UbuntuServer11.10
- CPU: Intel(R) Core(TM) i7-2600 CPU @ 3.40GHz (4core 8Threads)
- GPU: NVIDIA GeForce GTX 560 . CUDA Compute Capability 2.1
- ネットワーク:1000base-T

4.3 節，4.4 節の実験では上記環境の PC を 3 台，24 スレッド，4.5 節の実験では PC1 台，1 スレッド使用した．

4.2 実験対象グラフ

今回の実験では，実用的に用いることを想定して表 1 に示す実際の道路網のデータを用いて行った．

表 1 都市の道路網

ファイル名	ノード数	総枝数
昭和区	14248	18402
阿久比町	33598	35807
瀬戸市	58168	62102

4.3 1 センターを求める予備実験

1 センターを求める時の実行時間の違いを，C のみの場合，MPI を用いる場合，CUDA のみを用いる場合の 3 パターンについて調べた．この実験で，MPI は上記実行環境の PC を 3 台，24 スレッド用いて実験を行った．また，この実験では比較のために OR_library による p メディアン問題用のグラフを使用した [7]．これは表 2 に示すノード数の少ないグラフとノード数の多い都市の道路網のグラフを比較するためである．

表 2 ノード数の少ないグラフデータ

ファイル名	ノード数	総枝数
pmed1	100	200
pmed20	400	3200

全ての実験について，グラフを 1 回実行した時間の結果を示した．

表 3 実行結果 (p=1)

	p 値	CPU (秒)	MPI (秒)	CUDA (秒)
pmed1	1	0.001	0.002	0.075
pmed20	1	0.064	0.012	0.081
昭和区	1	3979.068	760.717	202.013
阿久比町	1	43486.757	7982.393	849.209
瀬戸市	1	232734.017	20961.472	5129.429

表 3 に結果を示す．結果から，CUDA はノード数が少ないグラフの場合だと性能を発揮できないので，C や MPI を用いて解くよりも遅くなった．しかし，1 万点以上のノード数のグラフでは MPI を使って解くより CUDA を使って解く方が実行完了までの時間が短いことから，CUDA は都市の道路網のようなノード数が多いとき MPI で解く場合より高速に解くことができるということがわかった．

4.4 p ノードセンター問題の実行時間

表 1 の道路網における p ノードセンター問題を C のみの場合，MPI を用いる場合，CUDA のみを用いる場合，CUDA と MPI を組み合わせた提案実装の 4 パターンで行った．CUDA と MPI を組み合わせた実装について，総ノード数を p 値で割ったものを丸めた数値を基準値として，領域の総ノード数が基準値以上の領域を GPU に割り当て，基準値未満の領域を MPI に割り当てる (表 4)．

表 4 割り当て基準値

	ノード数	p 値	割り当て基準値
昭和区	14248	30	400
		10	1500
阿久比町	33598	30	1000
		10	3000
瀬戸市	58168	30	2000
		10	5000

この実験で，MPI は上記実行環境の PC を 3 台，24 スレッド用いて実験を行い，GPU は 1 つの PC につき 1 つ使用した．都市の道路網のグラフを用いて一回実行した時間の結果を示した．

表 5 道路網に対する p ノードセンター問題の実行結果

	p 値	CPU (秒)	MPI (秒)	CUDA (秒)	提案実装 (秒)
昭和区	30	150.875	24.240	287.632	117.272
	10	1318.547	153.111	3035.842	1138.054
阿久比町	30	661.879	90.002	766.351	593.462
	10	2342.984	276.598	1100.456	352.742
瀬戸市	30	4234.637	766.351	6106.339	2391.223
	10	39938.538	3622.691	12049.410	3171.396

表 5 に結果を示す．結果から，CUDA と MPI を組み合わせて解く場合，CUDA への割り当て基準のノード数が少ない場合，CUDA の性能が引き出せなかったため遅くなった．しかし，ノード数が 5000 ノード以上の領域については CUDA に解かせると，MPI のみや CUDA のみで解く場合よりも高速に解くことができるということがわかった．

4.5 ネットワークボロノイ分割アルゴリズム最適化の効果

ネットワークボロノイ分割を効率的に繰り返すためのアルゴリズムの最適化の効果を調べるために，C だけのプログラムで，p ノードセンター問題の計算を行い，最適化の有無で比較したものである．この実験で，PC を 1 台，1 スレッドを用いて実験を行った．都市の道路網のグラフを 1 回実行した時間の結果を示した．

表 6 道路網に対するボロノイ分割の最適化の実行結果

	p 値	最適化・有 (秒)	最適化・無 (秒)
昭和区	30	150.875	165.396
阿久比町	30	661.879	800.376
瀬戸市	30	4234.637	4244.087

表 6 に実験結果を示す．ボロノイ領域の分割の繰り返しのアルゴリズムを改良した結果，瀬戸市のグラフでは効果を発揮できていないが，おそらく差分の量が多く最適化前と同じ回数処理をしてしまい，一部増やした最適化の処理の時間で遅くなってしまったと考えられる．しかし，グラフによっては改良前より高速化することができた．また，p 値が大きいグラフの多くは最適化によって計算時間が短縮できた．これは距離コストの計算回数が最適化前に全ノードに対して行ってた距離コストの計算を，改良後は差分ノードのみに対して計算しているからだと考える．

5 まとめと今後の課題

GPGPU クラスタを想定し，ネットワークボロノイ図を用いた p ノードセンター問題の近似解法を，CUDA と MPI を用いて実装した．予備実験で CUDA はより大規模なグラフで効率的に動作することがわかったので，ネットワークボロノイ分割されたグラフの領域を，領域の総ノード数が多いものは GPU に，少ない領域は MPI に割り当てて計算した．実験の結果，CUDA に割り当てる領域のノード数が多いほど CUDA の性能を活かして高速化ができることがわかった．またボロノイ分割の繰り返しを効率化するためのアルゴリズムの最適化も行った．実験の結果，ボロノイ分割された領域の差分を用いることで一部高速化することができた．

今後の課題としては，GPU と MPI の領域割り当てを動的に行う．具体的には，1 つの MPI プロセスをサーバとして，領域の計算が終わったプロセスに，まだ計算されていない領域をオンデマンドで割り当てることができると考えている．また，実装の最適化を行い，大規模実験で有効性を確認する．

参考文献

[1] 古田壮宏, 鈴木敦夫, “ネットワークボロノイ図を利用した p ノードセンター問題 の近似解法”, アカデミア 数理情報編, 第 6 巻, 2006 年.

[2] NVIDIA Corporation, “NVIDIA_CUDA_C_Programming_Guide”, `http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html` ,2012.

[3] Indiana University, “OpenMPI:Open Source High Performance Computing”, `http://www.open-mpi.org/` ,2004.

[4] Hajime Miyazawa, Takehiro Furuta, Atsuo Suzuki, “An Approximate Parallel Solution of the Vertex p-Center Problem Using Network Voronoi Diagram”, in *Proceedings of the 2009 International Conference on Parallel and Distributed Processing Techniques and Applications(PDPTA'09)*, 2009.

[5] Pawan Harish, and P.J. Narayanan, “Accelerating large graph algorithms on the GPU using CUDA”, in *Proceedings of the 14th International Conference on High Performance Computing (HiPC'07)*,pp.197-208,2007.

[6] 奥山 倫弘, 伊野 文彦, 萩原 兼一, “CUDA による全点対最短経路問題の高速化”, 社会法人 情報処理学会 研究報告,pp.145-150 ,2008 年.

[7] J.E.Beasley, “OR-Library: distributing test problems by electronic mail”, in *Journal of the Operational Research Society*, Vol.41,pp.1069-1072,1990.