2008－MPS－71 (5)
2008／9／18

最大クリーク抽出アルゴリズムの共有メモリ型並列計算機上での並列化

若月 光夫[†,‡] 高橋 真也[†] 富田 悦次[†,‡]

† 電気通信大学 電気通信学部 情報通信工学科 〒182–8585 東京都調布市調布ケ丘 1–5–1
E-mail: ‡{wakatuki,tomita}@ice.uec.ac.jp

あらまし 無向グラフ中の最大クリークを抽出する問題は，NP 困難のクラスに属す基本的な組合せ最適化問題の一つであるが，多くの数理問題に応用できることから非常に重要である．筆者らはこれまでに，深さ優先の逐次探索に基づく，分枝限定法を用いた効率の良い最大クリーク抽出アルゴリズムを提唱してきているが，実際の問題に応用する上ではなお一層の高速化が望まれる．そこで，本稿では，この逐次探索型の最大クリーク抽出アルゴリズムを並列化し，共有メモリ型並列計算機上で実験を行い，多くの対象グラフに対して良好な実験結果を得たことを報告する．

A Parallelization of an Algorithm for Finding a Maximum Clique on Shared Memory Computers

Mitsuo WAKATSUKI[†,‡], Shinya TAKAHASHI[†] and Etsuji TOMITA[†,‡]

†Department of Information and Communication Engineering, Faculty of Electro-Communications, The University of Electro-Communications, Chofugaoka 1–5–1, Chofu, Tokyo 182-8585, Japan
E-mail: ‡{wakatuki,tomita}@ice.uec.ac.jp

Abstract. Many practical problems can be formulated as the maximum clique finding problem, and then more efficient algorithms are strongly required for this problem. We have presented efficient branch-and-bound algorithms for finding a maximum clique, where these algorithms are sequential and are based on depth first search. We parallelized and implemented our algorithm which is one of such algorithms, and evaluated this parallelized algorithm by computational experiments on shared memory computers for some random graphs and DIMACS benchmark graphs.

1 まえがき

無向グラフ中の最大クリークを抽出する問題は，NP 困難のクラスに属す基本的な組合せ最適化問題の一つであり，問題の規模に関する多項式オーダーの時間内に解を得ることは期待しがたいが，多くの数理問題に応用できることから非常に重要であり，これら実問題に適用するためにもより高速なアルゴリズムが求められている．筆者らはこれまでに，深さ優先の逐次探索に基づく，分枝限定法を用いた最大クリーク抽出アルゴリズムとして，MCQ[1] を始め，MCR[2]，MCQ*[3]，MCS[4] 等の改良アルゴリズムを提唱してきている．これらのアルゴリズムは非常に効率的に動作するが，実問題に適用する上ではなお一層の高速化が望まれる．

近年，1 つの CPU の中に複数の CPU コアを搭載したマルチコア CPU が PC 用のプロセッサとして普及しており，Dual コアや Quad コアの CPU が一般的となっている．今後も，この CPU のマルチコアの流れは続いていくと考えられ，将来的には，より多くの CPU コアを搭載した製品が現れ，安価に並列計算を行うことが可能となっていくと見られている．そこで，本稿では，上記の逐次探索型の最大クリーク抽出アルゴリズムの一つである MCQ*を基に並列化し，共有メモリ型並列計算機上で実験を行い，多くの対象グラフに対して良好な実験結果を得たことを報告する．

2 諸定義

V を節点の集合，$E \subseteq V \times V$ を枝の集合としたとき，$G = (V, E)$ で表される単純無向グラフを対象とする．節点 $v \in V$ に隣接する節点の集合を $\Gamma(v)$ で表し，その節点数を v の次数と呼ぶ．また，集合 S に含まれる要素数を $|S|$ で表す．このとき，枝密度を $|E|/\{|V|(|V|-1)/2\}$ で定義する．

節点集合 V 中の任意の 2 節点が隣接しているとき，グラフ $G=(V,E)$ は完全グラフであるという．V の部分集合 Q に対する誘導部分グラフ $G(Q)$ が完全グラフであるとき，$G(Q)$(あるいは単に Q) をクリークと呼ぶ．クリーク Q が他のクリークの真部分集合でない場合，Q を極大クリークと呼び，また最も節点数の多い極大クリークの場合，Q を最大クリークと呼ぶ．G の最大クリークサイズ (即ち，最大クリークの要素数) を $\omega(G)$，または単に ω で表す．

3 最大クリーク抽出アルゴリズム

本稿で対象とするのは，グラフのデータが入力されたとき，そのグラフの最大クリークを 1 つ見付け，出力するアルゴリズムである．ここで，本稿で提唱する並列化アルゴリズムの基本となる，最大クリーク抽出アルゴリズム MCQ*[3] について述べる．

筆者らがこれまでに開発してきた最大クリーク抽出アルゴリズム MCQ[1] およびその改良アルゴリズムは，深さ優先の逐次探索に基づいており，節点の近似彩色を分枝限定に用いることで効率的な探索を可能としたアルゴリズムである．このような深さ優先探索アルゴリズムにおいて，初期節点の並び順は，問題の解を得るまでの探索回数および実行時間に大きな影響を与えることが知られている．しかし，探索中に最大クリークの上界を得るための近似彩色手続きを行う度に，初めに整列しておいた節点の順序が徐々に変わってしまう可能性があり，MCQ や MCR[2] では適切な順序での探索ができないことがあった．MCQ*は，MCR の方式により整列された初期の節点の並びを保存しておき，これを探索の各段階で保持して，この順に従って近似彩色を行うアルゴリズムである．この方法により，MCQ や MCR に比べてより適切な探索を可能にしている．

アルゴリズム MCQ*の概略を図 1 に示す．

```
procedure MCQ*(G = (V, E))
begin
  Q := ∅;  Q_max := ∅;
  節点集合 V を MCR の前処理で整列し，
  各節点に番号 No を付与する;
  EXPAND(V, V, No);
  output Q_max { 最大クリーク }
end { of MCQ* }

procedure EXPAND(V_s, R, No)
begin
  while R ≠ ∅ do
    p := 候補節点集合 R の最後尾の要素;
    if |Q| + No[p] > |Q_max| then
      Q := Q ∪ {p};
      V_p := V_s ∩ Γ(p); { 順序は保存する }
      if V_p ≠ ∅ then
        NUMBER-SORT(V_p, newR, newNo);
        { newR, newNo の初期値は意味を持たない }
        EXPAND(V_p, newR, newNo)
      else if |Q| > |Q_max| then Q_max := Q fi
      fi
    else return
    fi
    Q := Q − {p};
    R := R − {p};
    V_s := V_s − {p} { 順序は保存する }
  od
end { of EXPAND }

procedure NUMBER-SORT(V_s, R, No) { 近似彩色 }
begin
  順序付き節点集合 V_s の順に節点を近似彩色し，
  各節点に色番号 No を付与する;
  彩色した色番号 No で節点を昇順にソートし，
  順序付き節点集合 R へ格納する
end { of NUMBER-SORT }
```

図 1: アルゴリズム MCQ*

4 アルゴリズムの並列化

並列計算機環境としては，共有メモリ型計算機 (マルチコア，マルチ CPU 等) および分散メモリ型計算機 (クラスタ，PC グリッド等) の 2 種類がある．これらの計算機上で並列処理を行うためには，並列処理用にプログラムを書き直す必要がある．例えば，共有メモリ型計算機ならばマルチスレッド化，分散メモリ型計算機ならばプロセッサ間の通信処理による並列化が必要である．本稿では，マルチコア CPU を搭載した計算機上で並列処理を行うため，プログラムのマルチスレッド化を行った．これを実現するため，共有メモリ型並列計算機用 API である OpenMP を用いて，アルゴリズム MCQ*を基に並列化した．

4.1 並列化率と高速化率の関係

計算を行うプロセッサ数を N，アルゴリズム全体における並列実行部分の割合を p とした場合，並列化に関するアムダールの法則

$$\text{速度向上比} = \frac{1}{(1-p)+p/N}$$

が成り立つことが知られている．この式より，一般に並列処理が有効となるのは，プロセッサ数 N が少ない場合，または並列実行可能な部分の割合 p が極めて 1 に近い場合に限られることがわかる．このため，複数の CPU を使用した並列処理によって十分な速度向上比を得るためには，分散処理を行うためのオーバーヘッドを最小限に抑えることが必要となる．

4.2 並列化アルゴリズム

アルゴリズム MCQ*の並列化を行うには次のような処理が必要となる．

- 探索領域の部分問題への分割
- 各プロセッサへの処理の配分
- 各プロセッサ上での探索処理の実行

この各並列処理を行うためのオーバーヘッドを最小に抑えつつ，適切に処理量を分散する方法について考える．

4.2.1 部分問題への分割・配分方法

問題の分割・処理の配分について，今回の実装では探索開始時，最初の節点を選択する部分 (手続き MCQ* 中の，EXPAND の呼び出し) で処理を分割し，空いているプロセッサに逐次部分問題を配分する，という方法を採っている．この手法の利点は，分散処理にかかるオーバーヘッドが極めて小さいことである．より複雑な問題の分割や配分方法も可能であるが，余分なオーバーヘッドがかかることにより，全体の処理速度が低下する可能性があるため，今回はこの単純な分割方式を採用している．事前に行った予備実験から，小規模の並列処理 (8CPU 程度) の場合は，この単純な分割・配分方式で大部分のグラフについては問題がないことがわかっている．

4.2.2 処理の独立性

上記の方式を採用した場合には，各プロセッサにおける探索で，スレッド間で共有して更新時に同期が必要となる変数は，探索の分枝限定に用いられる $|Q_{max}|$(現在までに見付かっている最大クリークサイズ) のみとなる．$|Q_{max}|$ は単調に増加する値であり，それほど頻繁には更新されない．また，分枝限定の条件式に用いられる変数であるため，更新時に厳密にスレッド間で同期をとる必要はない．このため各プロセッサは，互いにほぼ独立した探索処理が可能となる．よって，部分問題の配分が適切に行われているという場合に，高い並列度を得ることはそれほど難しくないと考えられる．

5 計算機実験

5.1 実験対象および環境

実験には，アルゴリズム MCQ*_bitop を使用している．これは MCQ*を基本として，グラフの隣接行列をビット列で表現し，近似彩色手続きは NUMBER-SORT に代えて PAC[5] を使用したビット演算を用いる方式に変更改良したものである．MCQ*_bitop 中の各段階での入出力関係は MCQ*と同等であるため，分枝数 (即ち，探索領域) は MCQ*と全く同一である．複数の CPU により並列処理を行う場合，Q_{max} が更新されるタイミングが逐次探索アルゴリズムと異なるため，分枝数が多少変化する．このため，1CPU 実行時との分枝数の増減の比を表すことにする．また，MCQ*_bitop は元の MCQ*に対し，メモリ使用量が 1/8 で，グラフ全般について MCQ*よりも高速に動作する，といった特徴を持つ．このアルゴリズムを共有メモリ型計算機用並列化 API である OpenMP を用いて並列化した．表 1 および表 2 に記載の，ランダムグラフおよび DIMACS ベンチマークグラフを対象として，計算機実験を行った．ここで，ランダムグラフは節点数 n，枝存在確率 p の一様分布により生成されたグラフであり，実験では各 n，p に対して 1 個を使用した．また，表 2 中の d は枝密度を表している．

使用した計算機環境は，CPU：Xeon5320 1.86GHz (Quad コア)×2 (計 8 コア)，OS：Linux，コンパイラ：gcc 4.1.1 である．

5.2 実験結果および考察

実験結果を表 1 および表 2 に示す．表中の “高速化率”，“1CPU 当りの効率” および “分枝数の増減比” の項目は全て，8CPU 並列実行時のものである．

多くのグラフに対して，CPU 数である 8 倍に近い高速化率が得られていることがわかる．一部のグラフについて高速化率が 8 を超えている場合があるが，このようなグラフにおいては，並列実行時に分枝数が減少していることから，逐次探索の場合と比較して，より早い段階で最適解が見付かることにより分枝限定が有効に働き，必要となる探索量が減少したためと考えられる．

5.2.1 問題分割方法の問題点

MANN_a45(枝密度 0.996) 等，極端に枝密度の高いグラフについて，CPU をある一定数以上に増やしても，高速化率が向上しない場合があった．これは，分割された各部分問題のサイズに極端な偏りがあるため，最も大きい部分問題の探索がボトルネックとなるためである．このようなグラフに対しては，より適切な問題の分割や配分の方法を考える必要がある．

5.2.2 節点数が大きい場合の問題点

節点数 5,000 以上のランダムグラフに対して，多少高速化率が落ちる傾向にある．これはグラフの隣接行列が CPU キャッシュ内に収まらないことから，メモリ

表 1: ランダムグラフに対する実行結果

グラフデータ				実行時間 [sec]		高速	1CPU 当り	分枝数の
データ名	n	p	ω	1CPU	8CPU	化率	の効率	増減比 [%]
200_0.9_1	200	0.9	40	150.94	20.97	7.199	0.900	+11.35
500_0.6_1	500	0.6	17	36.27	4.57	7.933	0.992	+0.16
700_0.5_1	700	0.5	14	18.94	2.41	7.846	0.981	+0.87
1000_0.5_1	1,000	0.5	15	240.71	30.23	7.963	0.995	+0.18
3000_0.3_1	3,000	0.3	11	248.47	31.21	7.961	0.995	-0.43
5000_0.2_1	5,000	0.2	9	113.38	14.75	7.687	0.961	+0.02
10000_0.1_1	10,000	0.1	7	39.60	6.75	5.872	0.734	+0.01
15000_0.1_1	15,000	0.1	8	192.99	31.18	6.190	0.774	+0.00

表 2: DIMACS ベンチマークグラフに対する実行結果

グラフデータ				実行時間 [sec]		高速	1CPU 当り	分枝数の
データ名	n	d	ω	1CPU	8CPU	化率	の効率	増減比 [%]
brock400_1	400	0.748	27	669.19	80.76	8.286	1.036	-5.21
brock400_4	400	0.749	33	237.70	20.76	11.448	1.431	-34.50
brock800_1	800	0.649	23	7,528.70	932.20	8.076	1.010	-1.89
brock800_4	800	0.650	26	3,112.38	326.59	9.530	1.191	-15.73
p_hat700-3	700	0.748	62	3,143.44	400.20	7.855	0.982	+1.70
p_hat1000-2	1,000	0.490	46	245.99	31.73	7.751	0.969	+2.99
sanr400_0.7	400	0.700	21	158.16	20.01	7.904	0.988	+0.22
MANN_a45	1,035	0.996	345	1,379.84	353.51	3.903	0.488	-0.04

へのアクセスが頻繁に発生し，バスやメモリ帯域等，共有リソースへのアクセスの競合が発生するためと考えられる．また，CPU が 4MB の 2 次キャッシュを 2CPU コアで共有している関係上，8CPU 実行時は 1CPU 時と比べて，各 CPU が実質使用できるキャッシュ容量が減るといった要因も存在する．なお，CPU のキャッシュ効率を改善して実装することにより，このようなグラフに対して実行した結果，高速化率が向上することを確認している．

今回用いたアルゴリズム MCQ*_bitop は，メモリ使用量が従来のアルゴリズム MCQ* の 1/8 であり，MCQ* を直接並列化した場合と比較して，節点数が大きいグラフに対して並列実行した場合の速度は大幅に向上している．

6 むすび

最大クリーク抽出アルゴリズムの並列化を行い，多くのグラフに対して CPU 数に近い高速化率が得られることを確認した．この並列化手法は，MCS[4] 等，MCQ 系列のアルゴリズム全般に適用可能であり，同様の傾向の実験結果が得られている．アルゴリズムの並列化による高速化は，実問題に適用する上で有用であると考えられる．

今後の課題としては，MPI を用いた分散メモリ環境での並列化による実装や，より大規模な並列環境における実問題への適用等が挙げられる．

謝辞 本研究は科学研究費補助金基盤研究 (B) および (C) の支援を受けている．

参考文献

[1] E. Tomita and T. Seki: "An efficient branch-and-bound algorithm for finding a maximum clique," DMTCS 2003, Lec. Notes in Comput. Sci. 2731, pp.278-289 (2003).

[2] E. Tomita and T. Kameda: "An efficient branch-and-bound algorithm for finding a maximum clique with computational experiments," Journal of Global Optimization, Vol.37, pp.95-111 (2007).

[3] 須谷洋一，富田悦次："最大クリーク抽出アルゴリズムの効率化と実験的評価・解析," 情報処理学会研究報告，2005-MPS-57，pp.45-48 (2005).

[4] E. Tomita: "The maximum clique problem and its applications – Invited Lecture –," IPSJ SIG Technical Report, 2007-MPS-67, pp.21-24 (2007).

[5] 卜部昌平，富田悦次，森田昭広："密なグラフに有効で単純な最大クリーク抽出アルゴリズム," 情報処理学会第 67 回全国大会，Vol.1，pp.231-232 (2005).