1A-1

階層型共有メモリプログラミング言語 MpC を用いた TSP の並列処理

小山浩生† 鈴木絢子‡ 緑川博子‡
成蹊大学大学院工学研究科† 成蹊大学理工学部‡

1. はじめに

現在，分散メモリ並列システムにおける並列プログラミングで，最も広く用いられているのは MPI であるが，最近，MPI 以外のプログラミングモデルや言語にも少し目が向けられるようになってきた．その一つが，GAS 言語とよばれるもので，UPC，CAF などがある．これらは，従来のデータをフラットに扱う純粋な共有メモリプログラミングモデルとは異なり，分散共有メモリモデル，区分化大域アドレス空間モデルに基づいている．いずれも，データのローカリティに着目し，データの分散マッピング機構を取り入れ，明示的な並列処理記述を基本としている．

筆者らは，これ以前からソフトウエア分散共有メモリ(SDSM)の研究開発[1]をもとに，ローカリティに着目したメタプロセスモデルという階層型共有メモリプログラミングモデルを提案し，これに基づくポータブルな API として MpC 言語を開発してきた．MpC コンパイラは、MpC プログラムをクラスタ向けには SDSM プログラムへ、共有メモリマシンに対しては Pthread へ変換して並列実行を行えるようにしている．すでに，MpC がコモディティクラスタや共有メモリマシンにおいて，UPC や OpenMP と比較し，性能上の優位性があることを示した [2].

今回、この MpC 言語を用いて TSP(Traveling Salesman Problem)の並列化を行い、共有並列マシン、クラスタにおいて、このような並列実行の性能と解の質について調べた．

2. メタプロセスモデルと MpC

2.1 メタプロセスモデル

メタプロセスモデルとは，従来の共有メモリモデルに，NUMA マシンなどにも対応できるような各プロセスへの共有分散データの階層構造（スコープ）を取り入れたプログラミングモデルである．メタプロセスとは，1 つの応用を協力して並列処理する複数のプロセス全体を指し，ユーザーにとっての実行単位である．

2.2 MpC

MpC 言語は、メタプロセスモデルをクラスタや共有メモリマシン上で実現するために共有データ型 shared と共有データ分散マッピング指定子を新しく導入して，C 言語を拡張したものである．新しく導入した shared 型は C の記憶クラス指定子として組み込み，共有データの分散マッピングは柔軟性の高い様々な割り付けが可能で，コンパイル時よりも実行時に処理を行うことで柔軟性を高めている．また特定のコンパイラや実装系を前提とせず，ポータビリティが高いことが特徴である．

3. TSP の並列化の方針

TSP とはセールスマンが与えられた全ての都市を 1 度だけ訪れて元の都市に戻ってくるルートのうちでコストが最小のものを求める問題である[3]．最適化問題を解くには厳密解法と近似解法があるが，厳密解法では問題サイズが大きくなると実用的な時間で解を得るのが難しくなる。そのためここでは近似解法を対象とし、構築法と改善法を組み合わせて用いる．構築法は初期ツアーを決めるもので，改善法は初期ツアーを局所探索アルゴリズムに基づいて改善していくものである．構築法には saving, Nearest Neighbor，random の 3 つの方法を，改善法には 2-Opt，3-Opt，Or-Opt の 3 つの方法を試した．いずれもいくつかの都市やルートの一部を交換した場合のコスト計算をして，コストが改善された場合にのみ最良ツアーとして採用する．逐次処理の計算フローは構築法で初期のツアーを生成し，2-Opt，Or-Opt ，3-Opt の順で各改善ができなくなるまで行う．今回の実装には、局所探索範囲（交換候補の）制限を行って高速化を図るヒューリスティックは取り入れていない．TSP の並列化の基本方針として次の２つの方法を実装した．

3.1 繰り返し構造の並列実行 (TYPE1)

改善法における交換候補生成の繰り返しを並列化し，各候補採用時のコスト評価を並列に行う．現在の最良コストと候補交換時のコストを比較し，コストが改善される場合のみロックをかけて最良コストとツアーを更新し，これ以上の改善ができなくなる（λ-optimal）までこれを繰り返す．そのためロックの回数は最良コストの更新回数とほぼ一致する．ただし，逐次処理と同じ順番でツアーの更新がされていくとは限らないため，最終のコストは逐次処理の結果と等しくなる保証はない．

3.2 複数回の試行の並列実行 (TYPE2)

近似解法では、初期ツアーを変えて複数の試行を行い，最もよい解を採用する．構築法での初期ツアー生成には，基点都市の決定に乱数を用いているため，乱数 seed を変えて複数の初期ツアーを生成し，並列に試行を行う．各

The parallel processing of TSP with hierarchical shared memory programming model MpC

†Hiroki Koyama, Department of Information Sciences, Seikei University
‡Ayako Suzuki, Hiroko Midorikawa, Department of Computer and Information Science, Seikei University

試行の終了時に現在までの最良コストと比較し、改善されればロックして最良コストのツアーを更新する．また試行回数カウンターの更新も行う．したがって，処理全体のロック回数は試行回数 x2 と等しくなる．逐次処理の実行を複数回試行することと等しく，逐次処理と同じ結果が得られる．

4. 結果

4.1 実行環境

表 2 は 3.1，3.2 の並列化手法を用い，表 1 の実行環境で試行数 16 とし，プロセス数 1，2，4 における共有メモリマシンと PC クラスタ[1]の 2 種で実行した結果である．共有メモリマシンは表 1 のマシンを 1 ノード 2CPU で、クラスタは表 1 のマシンを 1，2，4 ノード各 1CPU で実行した．TSPLIB[4]の rat575，d657，pr1002，u2152 に対し、構築法は saving，改善法は 2-Opt，Or-Opt（3Opt は除く）の近似解法を用いた時の結果である．

4.2 実行結果

得られた解の質を見ると，最適値に対する増加分は TYPE2 が 4%台で安定しているのに比べ，TYPE1 は 3～5%とばらつきが大きい．TYPE2 は逐次処理のフローを繰り返しているのに対し，TYPE1 はツアーの書き換えを並列で行っているため，逐次処理と同じ探索を行っていないのでタイミングしだいで良くも悪くもなるという結果になった．

実行時間に関しては，共有メモリマシンは Hyper threading により仮想的に 4CPU まで利用できるが、1CPU 使用時に比べ 2 CPU で約 2 割程度しか速くならず，物理 CPU 数を超えると高速化はほとんどない．図 1 に u2152 の実行時間比を示す．共有メモリマシンの実行時間比が良くないのはメモリアクセスネックが原因と思われる．クラスタの TPYE1 でみると，CPU 数を増やしても速くならない．これはロック回数が TYPE2 は 32 回に対して、TYPE1 には 1100～2600 回とロックが多用されているためだと考えられる．クラスタの TYPE2 でみると，CPU 数を 1 から 2，4 と増やすとおよそ 1.9、3.6 倍速くなる．そのため，今回の実行環境ではクラスタで TYPE2 を用いるのが一番並列化の効果が得られる．

5. 終わりに

MpC 言語を用いてクラスタ，共有メモリマシンの両方で同一 TSP プログラムの実行が出来ることを確認した．しかし並列化手法よって，メモリアクセスネック，ネットワーク通信オーバーヘッドの影響が違うことが明らかになった．今後，詳細な原因の検討を行う予定である．

参考文献

[1] 緑川，飯塚："ユーザーレベル・ソフトウエア分散共有メモリ SMS の設計と実装", 情処論文誌 HPC,Vol.42, No.SIG9(HPS 3), pp.170-190 (2001)

[2] 緑川，飯塚："メタプロセスモデルに基づくポータブルな並列プログラミングインターフェース MpC",情処論文誌：ACS, Vol.46 No.SIG4(ACS9), pp.69-85,（2005)

[3] 山本，久保：巡回セールスマン問題への招待，朝倉書店

[4] TSPLIB
http://elib.zib.de/pub/Packages/mpestdata/tsp/tsplib/tsplib.html

表 1　実行環境(PC ノードのスペック)

CPU	Intel(R)Xeon(TM) 2.80GHz
CPU 数	2CPU
OS	Linux FC5 kernel2.6.18
メモリ	1GB

表 2　実行結果(saving + 2-Opt + or-Opt　16 試行)

プロセス数		共有メモリマシン		クラスタ	
		TYPE1	TYPE2	TYPE1	TYPE2
		rat575			
1	Excess[%]	4.40	4.24	4.40	4.24
	Time[sec]	10.13	10.92	7.60	6.82
2	Excess[%]	4.40	4.24	4.60	4.24
	Time[sec]	8.07	7.20	10.85	3.58
4	Excess[%]	4.40	4.24	4.71	4.24
	Time[sec]	7.74	7.17	11.03	1.96
		d657			
1	Excess[%]	3.21	4.17	3.21	4.17
	Time[sec]	16.01	16.87	12.03	10.94
2	Excess[%]	3.17	4.17	3.40	4.17
	Time[sec]	12.76	11.54	15.56	5.65
4	Excess[%]	3.46	4.17	3.33	4.17
	Time[sec]	12.17	12.10	15.21	3.07
		pr1002			
1	Excess[%]	4.78	4.34	4.78	4.34
	Time[sec]	46.17	50.01	43.89	35.15
2	Excess[%]	4.91	4.34	5.14	4.34
	Time[sec]	39.93	35.84	45.35	17.95
4	Excess[%]	4.85	4.34	5.26	4.34
	Time[sec]	38.35	39.23	42.33	9.45

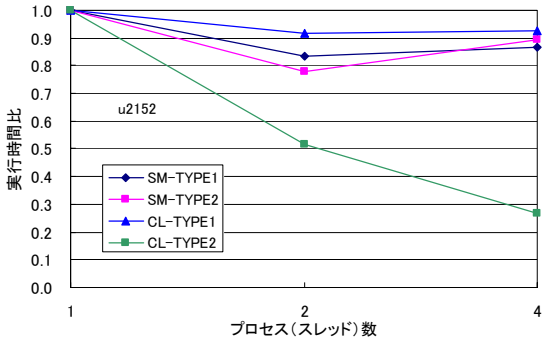

図 1　u2152 の実行時間比