3ZC-08

教育用並列プログラミング言語における CPU 制御による見かけ上の速度向上率の増加

田中　寛章‡　　水谷　泰治†

大阪工業大学　大学院情報科学研究科‡，大阪工業大学　情報科学部†

1. はじめに

近年，シングルコアプロセッサのクロック周波数の性能向上の頭打ちにより，並列化による高速化が注目されるようになった．このため，並列プログラミングの学習はより重要視されている．その学習において並列化によって大きな速度向上を体感する事は学習意欲の向上に繋がると考えられる．しかし，現状のマルチコア CPU では 2〜8 倍程の速度向上しか得られない．大きな速度向上を得るには PC の台数を増やす必要があり，環境構築や管理，運用コストが掛かる．コストを軽減するための研究には安価なコンピュータを用いた並列計算環境の構築[1]があるが，これは並列プログラミング言語に MPI を用いているため，初学者にとって難解な部分がある．そこで本研究では，著者らが提案している教育用並列プログラミング言語[2](以下 T 言語)の処理系を対象に，プログラムの実行速度を調節する事で少ない台数の PC でも見かけ上の高い速度向上率を得る手法を提案する．

2. 並列プログラムのプロセス数とコア数の関係

並列化による実行速度の向上率は，並列実行するプロセス数に比例していることが望ましいが，プロセス数を P，PC の CPU のコア数を C とすると，P>C の時は，P を増やしても速度向上しない．これは，各コアがプロセスを切り替えて実行する為である．図 1 に，4 コアのマルチコア CPU を搭載した PC が 2 台からなる，合計 8 コアを使用できる PC クラスタを用いて行列積を計算するプログラムを実行した時の実行時間とその速度向上率を示す．P>C の時は速度向上していない事が分かる．このように，多くのプロセス数で速度向上を得たい場合，その分のコアを用意する必要がある．

3. 実行速度の調整による速度向上の調整

プログラムを並列化する理由は，処理速度の向上のためであることが多い．そのため，学習において大きな速度向上を体感する事はモチベーションの向上に関わると考える．しかし，大きな速度向上を得たい場合，コア数とプロセス数の問題から，大規模な並列計算環境の構築が必要になる．これは，維持コストや構築難易度の面から用意が難しい．そこで本研究では，少ないプロセスでの実行速度を遅く調整し，多くのプロセス数で実行した時に速度が向上したように見せかける事で，少ない台数の PC でも，プロセス数を多くした時に，速度向上を得る手法を提案する．実行速度を最大にしたいプロセス数を Pmax とし，Pmax>C となる環境で，Pmax で実行した時に Pmax 倍の速度向上を得られるようにする．即ち，P で実行した時の実行速度を Pmax での実行速度の$\frac{P}{Pmax}$となるよう調節する．このような実行速度のエミュレーションを行う機能を T 言語上に実装する．

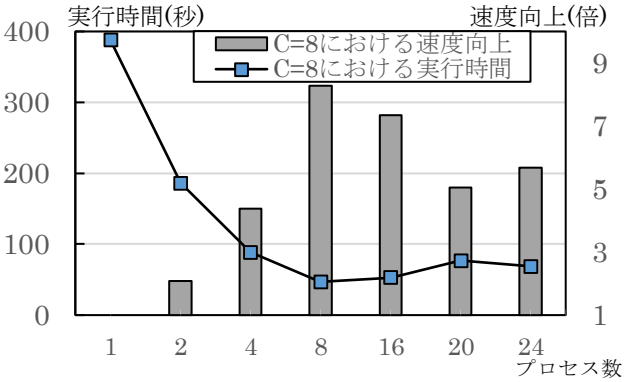

図 1 コア数 8，24 の時の実行速度と速度向上率

4. CPU 使用率の制限

並列プログラムの実行を遅らせるために，プロセスを実行する CPU の使用率を Linux のコントロールグループ(以下 cgroups)機能を用いて制限する．また，速度向上を表現するためにプロセス数毎の CPU 使用率を求める．

4.1 計算式

以下に，それぞれ P>=C の時と，P<C の時に速度が$\frac{P}{Pmax}$となるような CPU 使用率を述べる．

P>=C の時， P が変化しても実行時間は概ね一定となり，速度向上しない．この時 P が C から Pmax に増加するにしたがって速度向上し，P が Pmax の時に最大の速度向上となるようにしたい．そのためには，P での実行速度が$\frac{P}{Pmax}$となるように調整すれば良いため，CPU 使用率を$\frac{P}{Pmax}$と設定する．

P<C の時，実行に使用されるコア数は P である．この時 P を大きくすると，P=C となるまで使用す

Emulating Higher Speedup with Restricting CPU Usage on Educational Parallel Programming Language
†Hiroaki Tanaka, ‡Yasuharu Mizutani
‡Graduate School and Faculty of Information Science and Technology
†Faculty of Information Science and Tchnology, Osaka Institute of Technology

るコア数が増える．このため，CPU 使用率の制御無しでも実行速度は向上すると考えられる．実行速度を P>=C の時と合わせるために，P<C の時は P=C の時の CPU 使用率である$\frac{C}{Pmax}$とする．

以上より，CPU 使用率は以下のような式となる．

$$cpu使用率 = \frac{P}{Pmax} \quad (P \geq C) \qquad (1)$$

$$cpu使用率 = \frac{C}{Pmax} \quad (P < C) \qquad (2)$$

4.2 cgroups による CPU 使用率の制限

Linux には，cgroups と呼ばれるカーネル機能がある．cgroups を用いてプロセスの CPU 使用率を制限したい場合，再アクセス可能時間 period と，period 中の最大アクセス時間 quota を設定する．プロセスは period 時間中に quota 時間以上に CPU にアクセスしようとした場合，次の period 時間まで cgroups により CPU の使用を制限される．CPU 使用率を 1/2 とした場合の，CPU の稼働例の図 2 に示す．白色の部分は，cgroups の CPU 使用の制限により CPU が稼働していない状態を表している．

T 言語の処理系は，T 言語で書かれたプログラムを，MPI を用いたプログラムに変換する事で実装している．cgroups での制御を適応するため，変換時にプログラムの先頭部分に予め用意した cgroups を使用する記述を追加しコンパイルする．

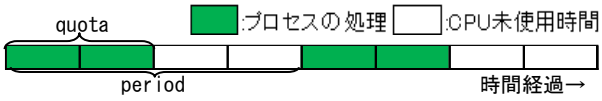

図 2 cgroups を用いた CPU 使用率の制限

4.3 1 コア当たりのプロセス数と CPU 使用率

cgroups による CPU 使用率の制御はプロセスごとに独立しているため，複数のプロセスを 1 つのコアで実行する時，period 時間内でプロセスが切り替わるため，設定した CPU 使用率よりも実質的な CPU 使用率が上がる．図 3 の上部分は，1 つ当たりのコアで 2 つのプロセスを実行する時に，(1)式を用いて CPU 使用率を 1/2 に設定しても，実質 100%の稼働となっている例である．これを考慮し，図 3 の下部分のように(1)式から 1 コア当たりの処理されるプロセス数で割る事で，CPU 使用率の向上を防ぐ．式(2)と式(3)から，CPU 使用率は P に関わらず常に$\frac{C}{Pmax}$となる．

$$CPU使用率 = \frac{P}{Pmax} / \frac{P}{C} = \frac{C}{Pmax} \quad (P \geq C) \qquad (3)$$

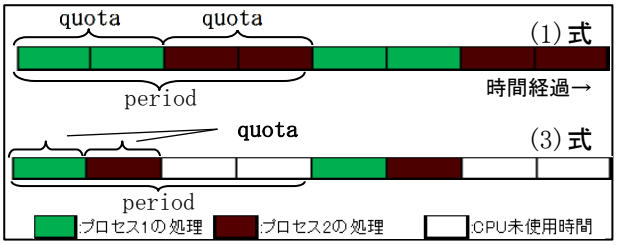

図 3 cgroups とプロセス切り替えの関係

5. 実験

4 コアのマルチコア CPU を搭載した PC からなる 1 台構成，2 台構成の PC クラスタで，提案手法を用いて Pmax=24 の時の速度向上のエミュレートを行った．行列積を行う並列プログラムを 24 プロセスまで並列実行し，提案手法を用いた場合と，6 台構成の PC クラスタを用いた実機での実行時間と速度向上率を比較したものが図 4 である．この結果により，概ね実機での実行結果に則した速度向上が得られることを確認した．

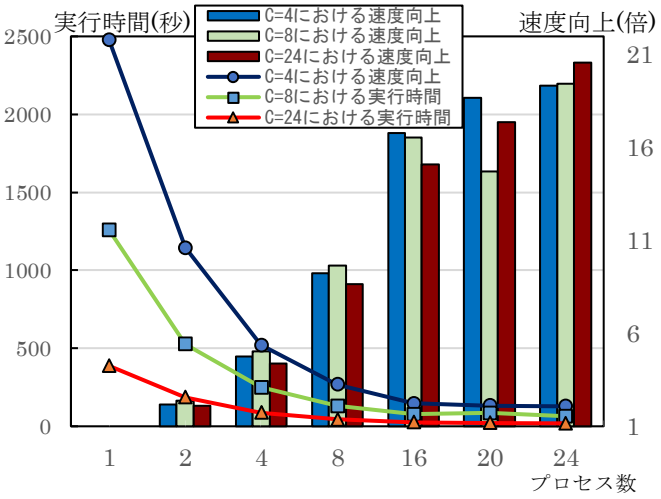

図 4 提案手法(4，8 コア)と実環境(24 コア)での実行時間と速度向上の比較

6. まとめと今後の課題

本稿では，cgroups を用いて並列プログラムの実行速度を遅らせる事で，少ない台数の PC でも大きな速度向上を体感できるような見かけ上の速度向上を体感する機能を，T 言語上に実装した．提案手法を用いて，合計 4 コア，8 コアとなる PC クラスタ環境で 24 プロセス迄の並列実行の速度向上を，24 コアの PC クラスタ環境で得られた速度向上と比較する実験を行った．結果，概ね実機実行のものと同一の速度向上が得られた．今後の課題として，プロセス間の通信に掛かる時間の制御と，通信が多発する等の効率の悪いプログラムの実行速度の再現が挙げられる．

謝辞 本研究は JSPS 科研費 JP15K21511，JP15K12008 の助成を受けたものです．

参考文献

[1] 信田圭哉，長谷川明生．"安価なコンピュータを用いた実験・教育用並列計算機環境の構築"，情報処理学会研究報告，Vol.2014-IOT-24,No.17 (2014/3)

[2] 田中寛章，藤井健太，礒淵郁也，水谷泰治．"複数のハードウェアでの共通操作に着目した教育用並列プログラミング言語の提案"．第 78 回情報処理学会全国大会，5ZC-02，(2016-03)．