5G－3

実行効率を考慮した並列プログラムの移植性確保
—— 問題の最適並列度推定 ——

須崎 有康　栗田 多喜夫　田沼 均　平野 聡　一杉 裕志

電子技術総合研究所

概要　実行計算機上で並列プログラムの最適並列度を推定する機構を備えることで、実行効率を考慮した移植性を確保する手法を提案する。最適並列度は処理するデータサイズによって変化するため、本手法では事前実行で求めたいくつかの結果から未知のデータサイズに対する最適並列度を推定する。推定手法としてはデータサイズと最適並列度の関係を離散的に推定する近接点近似と連続的に推定するスプライン補間を用いた。本手法を適用することで並列計算機上で実行効率を考慮した移植性を確保できることを行列乗算を対象とした実験で示す。

1　はじめに

逐次計算機ではアルゴリズムが仮定する実行モデルと実計算機がほぼ同一であるため、プログラムの移植性については記述言語を規定することのみで問題はなかった。しかし、並列計算機ではアルゴリズムが要求する実行モデル(プロセッサ数、ネットワークトポロジなど)と実行計算機が同一でないことがしばしばあるため、記述言語を規定することのみの移植性確保では並列計算機本来の目的である効率的実行が行なえない。

効率を考慮した並列プログラムの移植性確保のためには記述される並列アルゴリズムと並列計算機のアーキテクチャの関係を仲介する機構が必要とされる。このような機構のモデルは解析的には今だ求められていないが、我々は並列度の制御の観点から、いくつかの事前実行の結果を基に実行計算機上で最適並列度を統計的に求める手法を提案した[1]。本論文ではこの手法をプログラムの移植に用いることで実行効率を確保できることを示す。

2　並列アルゴリズムの並列度調整

多くの並列アルゴリズムは計算時間とプロセッサ数との間でトレードオフを行なう。そこでは問題のサイズに比例してプロセッサ数を要求する。例えば並列ソートである奇遇ソートでは、データサイズ n に対して $O(n)$ の計算時間を得るために $O(n)$ のプロセッサを要求する。

しかし実行並列計算機では要求されたプロセッサ数を提供できない場合がある。言語が提供する仮想プロセッサを使って記述することはできるが、効率は保証されない。奇遇ソートの例では並列アルゴリズムが最適ではない[1]ため、プロセッサが提供できなければいずれ逐次のアルゴリズム $O(n \log n)$ に追い越されてしまう。このため、並列アルゴリズムの一部を逐次処理 $O(n \log n)$ に置き換える必要がある。

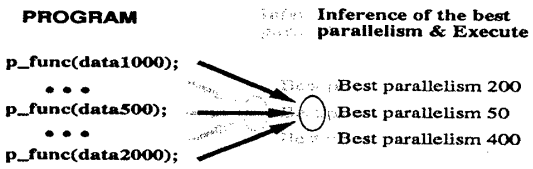

図 1: 動的に推定を行なう様子

一部の処理を逐次に置き換えた場合の最適並列度は必ずしも計算機が提供するプロセッサ台数ではない。なぜなら並列計算機では、計算と通信をオーバーラップさせることでデータ転送遅延を隠したり(レイテンシ隠蔽)、各プロセッサに割り当てられる処理を均等にする(ロードバランシング)ためには、プロセッサ台数以上の並列度で実行することが必要となるからである。

また並列部分と逐次部分のトレードオフは、データサイズ毎に考えなければならない。これはレイテンシ隠蔽やロードバランシングの効果が処理の大きさによって異なってくるためである。このため最適並列度はデータサイズごとに調節しなければならない。

3　最適並列度推定による実行効率確保

本手法は図1に示すように、ある手続きが呼び出されるときに処理するデータサイズによって最適な並列度を推定し、その並列度で実行するものである。実行計算機ごとに最適並列度を推定する基準を作成することにより、移植性を保ちながら計算機特性を考慮した効率的実行を

Portability of parallel progarm to support execution time
— Estimation of optimal parallelism —
Kuniyasu SUZAKI　Takio KURITA　Hitoshi TANUMA
Satoshi HIRANO　Yuji ICHISUGI
Electrotechnical Laboratory

[1]並列アルゴリズムが最適であるとは、その並列アルゴリズムを逐次でシミュレートした場合に計算量が逐次で最適なアルゴリズムと等しいことを言う。

行なう。

3.1 推定手法

本手法では、事前にいくつかのデータサイズに対する最適並列度を求める。この関係からプログラムの実行時に未知のデータサイズに対して、近接点近似とスプライン補間によって最適並列度を推定する。

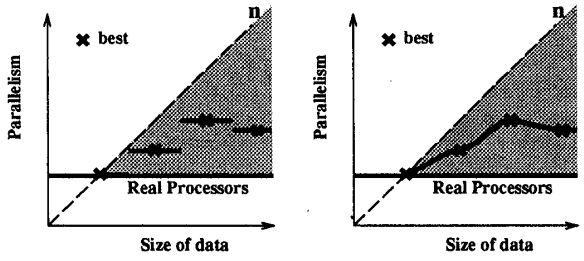

図 2: 近接点近似 (左) とスプライン補間 (右) : n を入力データサイズ、並列アルゴリズムが本来もつ並列度を $O(n)$ とした場合

近接点近似では未知のデータサイズに対して事前実行で求めたデータサイズのうち最も近いものを探し、そのとき最適並列度となった数を未知のデータサイズに対しても最適並列度であると推定する。図 2 右 では事前実行で求めたデータと最適並列度の関係を best で示し、それに基づく近接点近似の推定を太黒線で示している。

スプライン補間では事前実行で求めたデータサイズと最適並列度の関係をデータサイズに対する最適並列度の連続関数の一部としてとらえる (図 2 左)。与えられた関係 (図 2 左の best の点) から未知の部分を関数式を用いて補間する。スプライン補間を用いる理由は他の多項式補間に比べ、両端での振動が少なく、安定して滑らかに近似できるためである。

4 試作／実験

正方行列乗算を対象として、提案した手法の有効性を確かめる実験を行なった。対象計算機には、それぞれ 16 台構成の iPSC/2, Paragon を使用し、記述言語はこれらの計算機に共通して使える Dataparallel C を用いた。

図 3 に実行結果について示す。実験では正方行列のサイズを一辺が 17 から 200 までを対象とした。事前実行では、行列の一辺が 10 増えるごとに測定した。並列度は測定の都合で最大 96 までとした。図 3 より、本実験では近接点近似の方がスプライン近似より正確に推論してることがわかった。これは対象とした問題が、離散的に最適並列度を変えているためである。

それぞれの行列にかかった実行時間の総和については、iPSC/2 で最適実行した場合 (638.5sec) に対し、近接点近似が 1.005 倍 (641.9sec)、スプライン補間が 1.359 倍 (868.0sec) であり、Paragon で最適実行した場合 (89.1sec) に対し、近接点近似が 1.010 倍 (90.0sec)、スプライン補間が 1.015 倍 (90.4sec) であった。この結果より、このアルゴリズムに関しては近接点近似を用いることで効率性を確保しながら、移植性が保てることがわかった。スプライン補間が劣る原因は、連続的に推定しているために、最適並列度が離散的に変わる途中で最適でないものを推定してしまうためである。

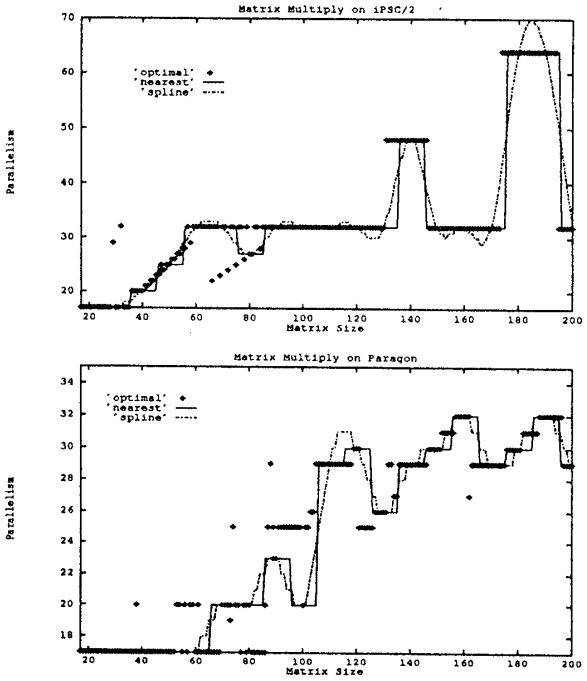

図 3: 行列乗算における最適並列度の推定

5 おわりに

実行効率を考慮した並列プログラムのポータビリティ確保のために、対象計算機上でアルゴリズムの最適並列度を推定する手法を提案した。推定手法としては近接点近似とスプライン補間を用い、行列乗算を例に簡単な実験を行なった。今後は、色々なアルゴリズムや計算機に適応してその有用性を確認していきたい。また推定方法についても、アルゴリズムと計算機間の関係をより正確に表すモデルを作っていきたい。

謝辞

本研究の一部は RWC 計画の一環として「超並列システムアーキテクチャに関する研究」で行なわれたものである。関係各位に感謝いたします。iPSC/2 を使用させて下さった東京大学の米澤明憲教授に感謝いたします。

参考文献

[1] 須崎, 栗田, 田沼, 平野, 一杉: “計算機特徴に合わせた並列アルゴリズムの最適分割法”, 第 35 回プログラミングシンポジュウム報告集, pp.63-70 (1994)