8L−3

Fortranマクロデータフロー処理におけるデータローカライゼーション

吉田 明正†, 前田 誠司†, 岡本 雅巳†, 合田 憲人†, 本多 弘樹‡, 笠原 博徳†
†早稲田大学 理工学部, ‡山梨大学 工学部

1 はじめに

マルチプロセッサシステム上におけるFortranプログラムの並列処理では、Doall、Doacross等のループ並列化(中粒度タスクの並列処理) [1] [2]が従来より広く用いられている。しかし、ループ並列化では、ループ以外の部分の並列性、例えば、サブルーチン、ループ、及び基本ブロック間の並列性を効果的に抽出することができなかった。そこで、筆者らは、マクロデータフロー処理 [3][4]と呼ぶ粗粒度レベルの並列処理手法を提案している。

本稿では、このマクロデータフロー処理手法において、マクロタスク間のデータ転送を軽減するためのデータローカライゼーション手法を提案すると共に、OSCAR上で行った性能評価についても述べる。

2 マクロデータフロー処理

本マクロデータフローコンパイレーション手法では、Fortranプログラムをマクロタスクと呼ぶ並列処理単位に分割し、擬似代入文ブロック(BPA)、繰り返しブロック(RB)、サブルーチンブロック(SB)の3種類のマクロタスクを生成する。ここで、BPAは基本的には通常の基本ブロックであるが、基本ブロックの処理時間が短い場合は、複数の基本ブロックを融合しBPAを生成する。RBは最外側ナチュラルループ [5] である。また、サブルーチンに関しては、可能な限りインライン展開を適用するが、効果的にインライン展開できない場合には、そのサブルーチンをSBとして定義する。

ただし、上述のマクロタスク定義では、十分に並列性が引き出せない場合あるいは粒度が細かすぎてスケジューリングやデータ転送オーバーヘッドが相対的に大きくなってしまう場合がありえる。そこで、本手法では、マクロタスクを再分割しマクロタスク間並列性を向上させたり、複数マクロタスクを融合することにより粒度を粗くし相対的オーバーヘッドを軽減する最適化 [4] を行う。

マクロタスク生成後、コンパイラはマクロタスク間の制御フロー、データフローを解析し、マクロフローグラフを生成する。

次に、マクロフローグラフからマクロタスク間の制御依存、データ依存を解析することにより、マクロタスク間の最大の並列性を表した最早実行可能条件 [2][3] を各マクロタスクごとに求める。この最早実行可能条件はマクロタスクグラフ(MTG)で表すことができる。

各マクロタスクは、条件分岐やマクロタスクの実行時間の変動に対処するため、実行時にダイナミックスケジューリングによってプロセッサクラスタ(PC)に割り当てられる。このダイナミックスケジューリングは、粗粒度タスクに対して適用されるため、スケジューリングオーバーヘッドは相対的に小さく抑えられる。さらに本手法では、OSコール等を用いず、コンパイラにより生成されたダイナミックスケジューリングルーチンを使用するため、オーバーヘッドをより小さく抑えることができる。

3 データローカライゼーション手法

マクロデータフロー処理では、マクロタスクは実行時にダイナミックスケジューリングによってプロセッサクラスタ(PC)あるいはプロセッサ(PE)に割り当てられる。このような方式では、マクロタスク間で共有されるデータは共有メモリに割当て、タスク間のデータ転送は全て共有メモリを介して行なうのが最も一般的な方法である。

しかし、この方法では共有メモリアクセスに伴うオーバーヘッド(タスク間データ転送オーバーヘッド)が大きくなる可能性が高い。そこで、本稿では、これらのデータ転送オーバーヘッドを軽減するためのデータローカライゼーション手法を提案する。データローカライゼーションとは、同一プロセッサに割り当てられたタスク間では、共有メモリを介さず、ローカルメモリを介してデータ転送を行なう手法である。

この手法を実現するには、データ転送量の多いマクロタスクを、同一のPC(PE)に割り当てることが必要であるが、マクロデータフロー処理では前述のようにマクロタスクのPC(PE)への割り当てが実行時に行われるため、データ転送量の多いマクロタスクが常に同一のPC(PE)に割り当てられるようにスケジューリングするのは困難である。そこで、本手法では、データ依存による結びつきの強い(転送されるべきデータ量が多い)マクロタスクをコンパイル時に融合し、実行時にこの融合マクロタスクを1つのPC(PE)に割り当てる方式をとる。

3.1 ループ分割及びマクロタスク融合手法

マクロデータフロー処理では、1つのDoallループが1台のPC(あるいはPE)だけで実行されるのをさけるため、Doallループを PC(PE)台数に等しいあるいはそれらの倍数になるように分割し、全PC(PE)で実行できるようにしている。例えば、図1(a)のデータ依存のある2つのDoallループは、図1(b)のように分割される。この分割後にデータローカライゼーションのための融合を適用しようとすると、ループ1bとループ2aの間にデータ依存が存在しているため、データ転送量(図中、データ依存を表すエッジの右横の数字は転送されるデータ数である)の多いマクロタスク(例えば、ループ1aとループ2a)を単純に融合することができない。

そこで、本手法では、データローカライゼーションのためのマクロタスク融合を適用可能とするために、以下の方法でデータの分割、すなわち、ループの分割を行なう。

1. マクロタスクグラフからデータ依存関係が存在するDoallループを抽出し、それらをグループとする。

   (図1(a)は、グループの一例である。)

2. 各グループ内で、データ依存のあるループ間のイタレーション間データ依存を解析して、各ループごとに標準イタレーションに対する比率及びずれ幅を求める。

   ここで、標準イタレーションとは、グループ内で後続データ依存ループを持たないループ(複数ある場合はその一つ)のイタレーションのことを意味し、そのループを標準ループと呼ぶ。あるループにおける比率 $rate$ とは、そのループがイタレーションごとにアクセスするデータの間隔が、標準ループがイタレーションごとにアクセスするデータ間隔の $1/rate$ 倍であることを意味する。また、あるループにおけるずれ幅 (min, max) とは、k番目の標準イタレーションが、そのループの $k*rate+min$ 番目〜$k*rate+max$ 番目のイタレーションにデータ依存していることを意味している。但し、比率及びずれ幅を求めるループと標準ループとの間に両方のループにデータ依存のあるループがある場合には、間にあるループを介した間接的なデータ依存により比率及びずれ幅を求めている。

   (図1(a)の場合、ループ2が標準ループとなるため、ループ2の比率は1、ずれ幅は(0,0)となる。次に、ループ1の場合に

A Data-Localization Scheme for Fortran Macro-Dataflow Computation

Akimasa YOSHIDA†, Seiji MAEDA†, Masami OKAMOTO†, Kento AIDA†, Hiroki HONDA‡, Hironori KASAHARA†
†Waseda University, ‡Yamanashi University

は、ループ1がイタレーションごとにアクセスするデータの間隔が、標準ループ（ループ2）がイタレーションごとにアクセスするデータ間隔と等しいので、比率は1となる。一方、ループ1のずれ幅は、標準ループ（ループ2）のk番目のイタレーションがループ1のk*1−1番目とk*1+1番目のイタレーションにデータ依存しているため、(-1,1)と求まる。）

3. グループ内の各ループのイタレーション範囲を、そのループの比率及びずれ幅を用いて標準イタレーション範囲に換算する。その後、グループ内の全てのループの標準イタレーション範囲の和集合を求め、これをグループの全標準イタレーション範囲とする。

   この全標準イタレーション範囲とは、グループ内の全ループで使われるデータの範囲を、標準ループのイタレーションの範囲で表したものである。

   （図1(a)の場合、ループ1のイタレーション範囲（I=1〜300）は、標準イタレーション範囲に換算すると2〜299となる。一方、ループ2の方は2〜299となるので、全標準イタレーション範囲は2〜299となる。）

4. 全標準イタレーション範囲を複数個（現在はクラスタ台数個）に均等に分割した範囲と、各ループの比率及びずれ幅に基づいて、ループのイタレーション分割範囲を決定し、ループを分割する。

   （例えば、図1(a)の場合、全標準イタレーション範囲（2〜299）をかりに3PC用に均等分割すると、分割後の範囲は、2〜101,102〜201,202〜299となる。また、ループ1は、比率が1、ずれ幅が(-1,1)であることから、例えば、標準イタレーション範囲102〜201に対応するループ1の範囲は、103〜200となる。）

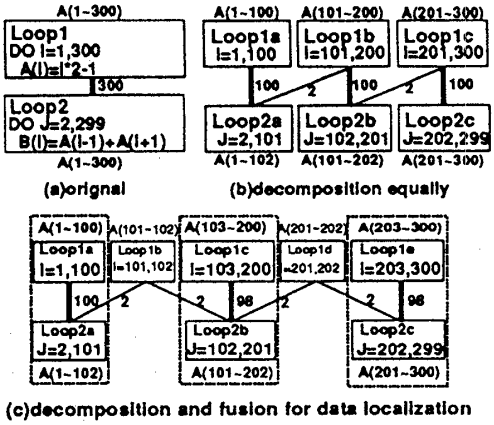

図1: ループ分割及びマクロタスク融合の例

上記で述べたループ分割を行うと、図1(a)の分割後のマクロタスクグラフは、図1(c)のようになる。また、分割後の各ループのデータの出力範囲および入力範囲は、図中のループ（実線の長方形）の上または下に書いてある範囲となる。この図1(c)において、分割後のループ2aとループ2bの両方が参照しているループ1のイタレーションは、ループ1bのような独立した小ループに分割されているため、ループ1cとループ2aの間にデータ依存が存在しない。したがって、同じデータ領域のループ（図中、点線で囲んだ部分）は、融合することができる。

3.2 データローカライゼーションの実行方式

本手法では、3.1で述べた方法で生成された融合マクロタスクに、データローカライゼーションを適用する。このとき、すべての配列変数をローカライズすると、融合マクロタスクの処理時間が延びてしまうことがあるので、本手法では、融合マクロタスク内のマクロタスク間でフロー依存および入力依存のある配列変数だけにローカライズを適用する。ローカライズの適用されない配列変数は、共有メモリのデータを直接アクセスすることになる。

ローカライズを行う配列変数の場合、実行中はローカルメモリのデータにアクセスするため、その配列変数の参照までに、参照する範囲のデータが、ローカルメモリにおかれていなければならない。このため、本方式では、融合マクロタスク内で、参照前に定義されない範囲のデータは、その融合マクロタスク実行前に共有メモリからローカルメモリに転送している。一方、ローカルメモリ上に定義した配列変数は、その融合マクロタスク実行後にローカルメモリから共有メモリに転送するようにしている。

このような方法でローカライゼーションを実現すると、融合されたマクロタスク内での共有メモリアクセスが減るため、データ転送オーバーヘッドを軽減することができる。

4 OSCARのアーキテクチャ

OSCAR [6]は、ローカルメモリと分散共有メモリを持つ32ビットRISCプロセッサ16台を、集中型の共有メモリに3本のバスで接続した共有メモリ型マルチプロセッサシステムである。OSCARは、PEを共有メモリに平等結合したアーキテクチャとなっているが、複数のプロセッサエレメントをグループ化することにより、3クラスタまでのマルチプロセッサクラスタシステムを効率よくシミュレートできる。

5 OSCAR上での性能評価

ここでは、連立1次方程式間接解法のCG法プログラムを用いる。このプログラムで実行時間のほとんどが費やされる収束計算ループをOSCAR上の1PEを用いてシーケンシャル実行すると処理時間は19.26[ms]である。一方、3PEを用いて、マクロデータフロー処理により実行すると、マクロタスク間の並列性が少ないため、処理時間は16.95[ms]となり、あまり短縮されない。そこで、本稿で提案するループ分割（データ分割）を適用するとマクロタスク間の粗粒度並列性が引き出されるため処理時間は7.30[ms]、さらに、マクロタスク融合を適用するとダイナミックスケジューリングのオーバーヘッドの軽減により6.92[ms]に短縮される。次に、融合マクロタスク内でデータローカライゼーションを適用すると共有メモリアクセスに伴うデータ転送オーバーヘッドが軽減されるため処理時間は6.13[ms]となり、シーケンシャル処理と比べて3.14倍の速度向上が得られる。このように3PEでの実行により3倍以上の速度向上が得られているのは、単一PEでの処理においてはマクロタスク間のデータ授受に集中共有メモリを使用しているが、ローカライゼーション後は一部のデータ授受がローカルメモリ上で行なわれているためである。

6 むすび

本稿では、データローカライゼーションのためのループ（データ）分割手法と融合手法、及び、融合したマクロタスク内でのデータローカライゼーション手法について述べた。今後の課題としては、多次元配列変数を使ったプログラムでのデータ分割手法の開発、及び、スタティックスケジューリングによるマクロデータフロー処理でのより広範囲なデータローカライゼーションの実現などがあげられる。

参考文献

[1] D.A.Padua, M.J.Wolfe: "Advanced Compiler Optimizations for Super Computers", C.ACM, 29, 12, pp.1184-1201 (Dec.1986).

[2] 笠原:"並列処理技術", コロナ社(1991-06).

[3] 本多, 岩田, 笠原: "Fortranプログラム粗粒度タスク間の並列性検出手法", 信学論(D-I), J73-D-I, 12, pp951-960 (1990-12).

[4] 笠原, 合田, 吉田, 岡本, 本多: "Fortranマクロデータフロー処理のマクロタスク生成手法", 信学論(D-I), J75-D-I, 8, pp.511-525 (1992-08).

[5] A.V.Aho,R.Sethi and J.D.Ullman: "Compilers(Principles, Techniques, and Tools)", Addison Wesley (1988).

[6] 笠原, 成田, 橋本: "OSCARのアーキテクチャ", 信学論(D), J71-D, 8 (1988-08).