数値演算ループの多次元展開技法

4T－5

今野 明† 古関 聰† 小松 秀昭‡ 深澤 良彰†

†早稲田大学理工学部 ‡日本IBM(株) 東京基礎研究所

1 はじめに

プログラムのループ部分において、各繰返しを展開して条件ジャンプ命令を取り除く手法がある。この最適化を行うと、ループ制御に相当する条件判定とジャンプの命令が各繰り返しごとに省略されるだけでなく、ジャンプによる命令パイプライン分断が抑制される。特に、superscalar,VLIW等の命令レベル並列計算機での実行を考えた場合、繰返し間の制御依存が解消され、命令の繰返しを越えた移動が可能になり、並列度が大幅に高まる可能性がある。

並列化の対象になるループが自由交換可能(fully permutable)な多重ループをなしている場合、基本的にはどのループを選んで展開してもよい。しかし、得られる並列化の効果は、対象となるループにかかわる繰返し間の依存関係および記憶データの再利用の関係により異なる。展開するループによっては、配列の参照が共通化でき、1回の展開当たりのメモリ参照命令の個数を減らして、命令実行ユニットが飽和するまでにより多くの繰返しを展開できるものもある。そこで今回の発表では、多重ループに存在する依存関係および記憶データの再利用の関係から、最適と思われる各ループの展開回数を見積る手法について述べる。

以下の議論では、対象とする多重ループは自由交換可能であることを仮定する。

2 多次元展開の性質

従来の研究では、最も内側のループに対して展開したり、software pipelining変換をしていた。今回の研究の目標は、多重ループの各ループを同時に展開するように拡張することである。

多次元展開の例を図1に示す。(a)のループに対し、iについて2回、jについて3回の繰り返しからなるブロックを構成し、ブロック内の全ての繰返しを展開し、1個の繰り返しにまとめたものが(b)である。このような変換を行うには、途中でiのブロック内ループとjのブロック制御ループを交換していることから、ループ本体bdy(i,j)にはiのループとjのループの交換を禁止するような繰返し間の依存関係があってはならないことがわかる。

```
for i:=imin to imax do
  for j:=jmin to jmax do
    bdy(i,j);
```

(a) 元の2重ループ

```
for i:=imin to imax step 2 do
  for j:=jmin to jmax step 3 do begin
    bdy(i,j);   bdy(i,j+1);   bdy(i,j+2);
    bdy(i+1,j); bdy(i+1,j+1); bdy(i+1,j+2);
  end;
```

(b) ブロッキングし展開した2重ループ

図1 多次元展開の例

3 メモリ参照の共通化

ループ展開時に適用可能な最適化技法の一つに、メモリ参照の共通化がある。これは、各繰り返しにおいて配列参照等によりメモリの同一の番地を参照することがあらかじめわかっている場合、展開後のループ本体ではその番地の参照を1回に抑え、残りはレジスタへの参照に置き換えることである。この利点は2つある。一つは、メモリ参照命令の発行回数を減らし、余った命令実行能力を他の命令に割り当てることができ、その分並列度が向上することである。もう一つは、キャッシュミスの発生回数を高々1回に抑えられることである。このような性質は各ループの展開による効果が加算されるので、多重ループの最も内側のループだけでなく、外側のループも含む複数のループを同時に展開した方が有利になる。

4 多次元展開による実行性能の見積り

一般には各ループの展開回数が多い程全体の並列度が向上するが、その効果は展開後のループ本体の命令数が多くなりすぎて計算機の命令並列実行能力を超えるころから低下し、更にレジスタの個数および命令キャッシュの容量の制約があるので、むやみに大きく取ることはできない。したがって、適当な大きさで打ち切ることが重要である。

1サイクル当たりの命令実行能力は有限のため、ループ本体の命令間依存関係により決まる最小の実行サイクル数（依存サイクル数）Cの間に実行可能な命令の個数に対して、展開ループ本体の命令総数が上回るようになると、命令数の飽和が起きて、実際の展開ループ本体の実行に要するサイクル数が延長する。

そのために、コンパイル時に得られる多重ループに関するパラメタから、展開後の実行性能つまり1サイク

A Method of Unrolling Loop Nests for Scientific Programmes
Akira KONNO†, Akira KOSEKI†, Hideaki KOMATSU‡, Yoshiaki FUKAZAWA†
† School of Science and Engineering, Waseda University
‡ IBM Japan, Ltd. Tokyo Research Laboratory

ル当たりの元のループの繰返し実行回数、および、展開ループ本体の命令数が飽和するかどうかを、ループの変換に先だって見積る必要がある。

展開の対象にする最内側の n 重のループに、外側から順に $1,2,3,\ldots$ と番号を付ける。各ループの展開回数 k_i（展開しない時は 1 とする）を決めるにあたって、以下のパラメタを考慮する。

$m(>0)$	1 サイクルに実行できる命令の個数
$n_0(>0)$	展開前のループ本体の命令の個数
$c_0(\geq c_i)$	展開前の依存サイクル数
$c_i(\geq 0)$	i 番目を 1 回展開して増えるサイクル数
n_i	i 方向に 1 回展開した時に減る命令数

m だけが計算機による定数で、他はすべてプログラムの性質による。

このとき、以下の値を見積る。

$C \equiv c_0 + \sum_i c_i(k_i - 1)$　展開後の依存サイクル数

$N \equiv n_0 - \sum_i n_i \frac{k_i - 1}{k_i}$
展開後の繰返し 1 回当たりの命令数

$S \equiv N \prod_i k_i - m/C$　飽和条件 ($S > 0$ のとき飽和)

$P_{NS} \equiv \frac{\prod_i k_i}{C}$　飽和しないときの性能

$P_S \equiv m/N$　飽和したときの性能

変数の再利用がないとき、つまり $n_i = 0$ のときは、性能 P は、命令実行能力が飽和したときに最大になる。また、飽和したときの性能は $P = m/n_0$ で一定になる。したがって、$S = 0$ を満たす点は全て最適解になる。しかし、ループ展開に伴うメモリ参照の共通化に伴って繰り返し 1 個当たりの命令数が若干減るという効果がある。このため、性能は、飽和した場合でも k_i の値の増加にともない緩慢に増大し、$\forall i k_i \to \infty$ の極限で

$$\frac{m}{n_0 - \sum_i n_i}$$

に近付く。もちろん、これはループ本体の命令数が無限大の極限において成り立つので、実際には、これより若干低い性能を与える $\{k_i\}$ の値で妥協しなければならない。しかし、もし、そのときの性能が極限値と大差なければ、十分「最適」であるといえる。

一般に、展開回数 k_i の値が増加したとき、性能が増加する割合は、飽和している場合は飽和していない場合にくらべて低いので、飽和領域の境界付近を選ぶのが賢明である。その場合でも、性能最大の点と命令数最小の点は一般には一致しない。したがって、展開回数はこれを求めるアルゴリズムに依存する。

5 展開回数の決定アルゴリズム

今回の発表では、展開回数の組み $\{k_i\}$ を決定する一手法を紹介する。k_i の値は、1 以上の整数に限られるから、まず $\forall i k_i = 1$ から出発し、i の値を適当に選んで k_i の値を 1 つずつ増していき、飽和領域に入ったら終了する。このとき、性能がより大きくなる方向を選んで k_i の値を増やせばよい。このアルゴリズムを以下に示す。

1. すべての i について $k_i = 1$ に初期化する
2. このときの $\{k_i\}$ について S の値を計算し、$S \geq 0$ ならば終了する
3. すべての i について $P_i \equiv P(k_1,\ldots,k_i+1,\ldots,k_n)$ を計算し、P_i の最大値を与える i を選び、k_i の値を 1 増す
4. 2. に戻る

6 評価

今回の発表では、Livermore Fortran Kernel の No. 4,8,18,21,23 のプログラムを例にとり、VLIW 型並列計算機を対象に、最適展開回数を見積り、その時得られる性能つまり 1 サイクル当たりの繰返し実行回数を調べた。表 1 に、見積った展開数およびその時に実際に得られた実行性能を示す。ここで、A は最内ループのみを展開する手法によるもの、B は本手法によるものを表す。また、評価の条件は、1 サイクルあたり浮動小数点演算および浮動小数点 load/store を最大 4 個、その他の命令は無制限実行可能な VLIW 型計算機で、浮動小数点演算および浮動小数点 load/store は 2 サイクル、その他は 1 サイクルで実行が完了するものとした。

表 1 各 kernel に対する展開回数および実行性能

No.	A 展開数	A 性能	B 展開数	B 性能
4	(1,5)	1.000	(3,2)	1.333
8	(1,1)	0.074	(1,1)	0.074
18	(1,2)	0.096	(1,2)	0.096
21	(1,1,6)	0.774	(3,2,2)	1.091
23	(1,1)	0.083	(5,5)	0.213

この表から、各 kernel とも 1 次元展開したときの実行性能よりも多次元展開したときの実行性能の方が若干高くなっていることがわかる。これは、多次元展開を行うと、1 次元展開の場合に比べて、記憶再利用の効果がより強く現れるためである。特に、kernel No.23 においては、多次元展開すると実行性能が格段に向上する。このループには、各ループに繰返し間の依存関係があるため、1 方向のみのループの展開では取り出せる並列度に限度がある。2 方向を同時に展開した場合は、展開ループ本体の増加が 2 次元的になるのに対し、依存サイクル数の伸びは 1 次元的にしかならないので、展開ループ本体の命令数が飽和するまで限りなく並列度を高められる。

7 まとめ

今回の発表では、多重ループに対する展開の手法ならびにその性質、特に共通化による命令数削減の効果と実行ユニット飽和の関係を説明し、各ループの展開回数を決定するアルゴリズムを提案した。この手法により、2 重以上の自由交換可能多重ループから細粒度並列計算機の実行能力を最大限に引き出すループ変化が常に可能であることがわかった。今後は、ループ本体に関する繰り返し間の依存および再利用の関係を調べ、このアルゴリズムの適用に必要なパラメタをコンパイラにより自動的に求める手法の開発を行う予定である。