計算機アーキテクチャ　79－8
（１９８９．１１．２１）

並列計算機ADENAに対する最適化コンパイルの一手法

若谷彰良，材木幸治，岡本理
松下電器産業(株)半導体研究センター

並列計算機ADENAの言語/ADETRANに対する最適コンパイル手法、特に、doループ内の演算の短縮及び並列計算機特有のオーバーヘッドであるデータ転送の効率化について述べる。前者については、doループ内には同じ配列の異なったインデックスの演算が多いと考え、一度ロードしたデータはレジスタに残しておき、ループをアンロールし、かつ、レジスタ使用もサイクリックにかえる方法を提案する。後者については、まず、転送の型の分類を行ない、演算とデータ転送がオーバーラップする場合が効率良いことを示し、その型への変形方法を述べる。最後に、例を用いて、本手法の評価を行なった。

A compiler optimization for the ADENA system

Akiyoshi Wakatani , Kouji Zaiki , Tadashi Okamoto
Matsushita Electric Industrial Co.,Ltd. Semiconductor Research Center
3-25 , Yakumonakamachi , Moriguchi 570 , Japan

This paper discusses the optimization method for a parallel computer ADENA , especially 1) the code optimization within the do-loop and 2) the reduction of the explicit data transfer time which is an overhead of parallel computations. For the first problem, a new method for reducing memory access frequency is proposed. This is realized by taking advantages of common characteristics in numerical programs: to remain loaded data in register until the following step, to un-roll do-loop to some extent using cyclically shifted register allocation. For the second problem, after classifying data-transfer types, one type with best efficiency is selected and proposed, where the data-transfer operations are embedded in processor operations. Some methods to change other types to this most efficient type are also described. Finally, these methods are evaluated by the sample programs.

(0) はじめに

ADENAは、数値シミュレーションを主目的とした並列計算機で、京都大学との共同で開発した。(1),(2)
ADENAにおけるプログラミングは、FORTRANに並列化用構文を付加した言語ADETRANを用いて行なわれる。
今回の報告では、ADETRAN及びADENAの概要を述べたあと、ADETRANのコンパイルにおける演算部及び転送部に関する手法を述べ、doループ内の差分式計算に対する効率化について検討する。

(1) ADETRAN及びADENAの概要 (3),(4),(5)

1) ADENA
ADENAは、1つのホスト計算機と256個のPE(Processing Element)とADENA network からなる。PEは、EPU(Element Processing Unit)、TCU(Transfer Control Unit)、LDM(Local Data Memory)、LIM(Local Instruction Memory)からなる。

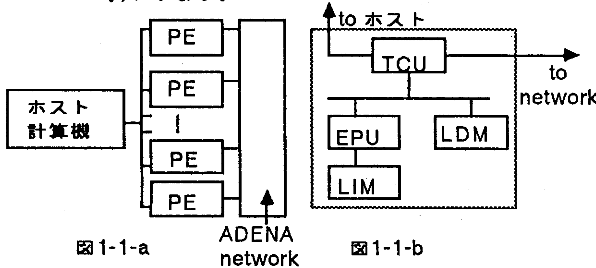


図1-1-a　図1-1-b

ADENA networkは、ADENAを特徴づけるもので、3次元シミュレーションに適応したnetworkである。構成としては、転送を制御するSRC(SendReceive Controller)とバッファとしての役目をするBMU(Buffer Memory Unit)からなる。
接続関係は、下に示す様になる。

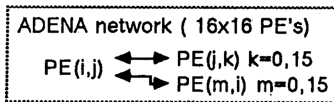


図1-2

これを実現するために、BMUを4096個立方体状に並べ、SRCを256個正方形状に2セット並べる(実際は、BMUは16ユニットで1チップ、SRCは4ユニットで1チップなので、それぞれ256個、128個用いる)。

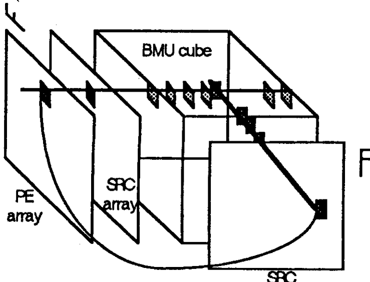


図1-3

2) ADETRAN
ADETRANは、ADENAに対するプログラミングを行なうための高級言語である。基本的な文法はFORTRANに従っている。特徴としては、
* インデックスの揃っている配列の、インデックスを方向付配列と呼び、そのインデックスを、'/'で囲む。
* 並列実行をpdo文で表す。
* 異なった方向付配列のデータ交換(転送)を、pass文で表す。

が挙げられる。
まず、次のFORTRANプログラムを考える。

```
do 1 k=1,N
do 1 j=1,N
do 1 i=1,N
  u(i,j,k)=p*(v(i+1,j,k)-v(i-1,j,k))
1 continue
do 2 k=1,N
do 2 j=1,N
do 2 i=1,N
  w(i,j,k)=q*(u(i,j+1,k)-u(i,j-1,k))
2 continue
```

図1-4

以後、配列の第1インデックスをx方向、第2をy方向、第3をz方向とする。このプログラムにおいて、ラベル1のループは、y方向とz方向のインデックスが同じであるので、yz平面をPEの平面と一致させるとx方向の計算が各PEで並列に行なえる。
これを、ADETRANプログラムで記述すると、

```
pdo j , k = 1,N
do 1 i=1,N
  u(i,/j,k/)=p*(v(i+1,/j,k/)-v(i-1,/j,k/))
1 continue
pend
```

図1-5

となる。'/'で囲まれているインデックスは、論理的なPE番号を表わす。
同様にラベル2のループを、ADETRANプログラムで記述すると、

```
pdo k,i=1,N
do 2 j=1,N
  w(/i,j,/k)=q*(u(/i,j+1,/k)-u(/i,j-1,/k))
2 continue
pend
```

図1-6

となる。
ここで、ラベル1のループで確定したu(i,/j,k/)をラベル2のループのu(i/,j,/k)で用いるのだが、この2つのプログラムを連続して書いた場合、データの存在する場所が一致しない。つまり、ラベル1のループのu(i,/j,k/)はPE(j,k)の配列データだが、ラベル2のループのu(i/,j,/k)はPE(k,i)の配列データである。よって、これらのPE間でデータ転送が必要となる。これを表すのが、pass文である。
以上をまとめると、図1-4のFORTRANプログラムは、

```
pdo j , k = 1,N
do 1 i=1,N
  u(i,/j,k/)=p*(v(i+1,/j,k/)-v(i-1,/j,k/))
1 continue
pend
pass i,j,k=1,N
 u(i/,j,/k)=u(i,/j,k/)
pend
pdo k,i=1,N
do 2 j=1,N
  w(i/,j,/k)=q*(u(i/,j+1,/k)-u(i/,j-1,/k))
2 continue
pend
```

図1-7

となる。

(2) 演算部最適化(6) (cyclic register allocation)

FORTRANにおいて、演算の密度の高い部分はdoループの中である。ADENAにおいてもdoループを並列処理して高速化を図っている。したがって、ループ内の計算時間を縮小することは効果的である。

また、偏微分方程式の 解法を初めとする数値シミュレーションにおいては、doループ内の演算は、ループインデックスを用いた差分式になっていることが多い。これに対する最適化を検討する。

```
DO 1 i = 1,21
u(i,/j,k/)=v(i-1,/j,k/)+v(i,/j,k/)+v(i+1,/j,k/)
1 CONTINUE
```

図2-1

この例を疑似コードにコンパイルすると、

```
_loop_entry i=1,21,1
ENTRY:
_ld #v(i-1),fr0
_ld #v(i),fr1          ロード命令
_ld #v(i+1),fr2
_add fr0,fr1,fr3       浮動小数演算命令
_add fr3,fr2,fr4
_st fr4,#u(i)          ストア命令
_loop_return ENTRY
```

fr: レジスタ
_XX: 疑似コードを表わす。

図2-2

となる。このように、ループ内ではロード命令が3つ、ストア命令が1つ、浮動小数演算命令が2つあり、演算効率が悪い。つまり、2つの浮動小数演算のために合計6命令必要となっている。

そこで、図2-1の代入の式に着目すると、ループインデックスの差分式になっており、同じデータを複数回ロードしている。つまり、例えば、i=2の時、v(1, /j,k/)とv(2, /j,k/)とv(3, /j,k/)をロードし、i=3の時、v(2, /j,k/)とv(3, /j,k/)とv(4, /j,k/)をロードし、i=4の時、v(3, /j,k/)とv(4, /j,k/)とv(5, /j,k/)をロードするので、v(3, /j,k/)は3回ロードする。よって、ロードしたデータをレジスタに残しておき、loop return前に、レジスタ間ムーブすることにより無駄なロードはなくなる。

```
_ld #v(0),fr0
_ld #v(1),fr1
_loop_entry i=1,21,1
ENTRY:
_ld #v(i+1),fr2        ロード命令
_add fr0,fr1,fr3       浮動小数演算命令
_add fr3,fr2,fr4
_st fr4,#u(i)          ストア命令
_mv fr1,fr0
_mv fr2,fr1
_loop_return ENTRY
```

図2-3

しかし、ロード命令は減ることになるが、ムーブ命令が入ることにより大幅な演算時間削減にはならない。

そこで、レジスタ間ムーブをなくすために、コード自体を3倍に展開(アンローリング)し、使用レジスタの番号をサイクリックに移動させる。

```
_ld #v(0),fr0
_ld #v(1),fr1
_loop_entry i=1,21,3
ENTRY:
_ld #v(i+1),fr2        ロード命令
_add fr0,fr1,fr3       浮動小数演算命令    fr3=fr0+fr1
_add fr3,fr2,fr4                           fr4=fr3+fr2
_st fr4,#u(i)          ストア命令
_ld #v(i+2),fr0        ロード命令
_add fr1,fr2,fr3       浮動小数演算命令    fr3=fr1+fr2
_add fr3,fr0,fr4                           fr4=fr3+fr0
_st fr4,#u(i+1)        ストア命令
_ld #v(i+3),fr1        ロード命令
_add fr2,fr0,fr3       浮動小数演算命令    fr3=fr2+fr0
_add fr3,fr1,fr4                           fr4=fr3+fr1
_st fr4,#u(i+2)        ストア命令
_loop_return ENTRY
```

図2-4

このように、ロード命令が3つ、ストア命令が3つ、浮動小数演算命令が6つとなり、演算効率が高くなる。

	最適化前	最適化後
総コード数	6	12
浮動演算数	2	6
FLOPS*	1	1.5
FLOPS**	0.2(=2/10)	0.3(=6/21)

図2-5

* 最適化前を1とする
** ADENAの場合の比較

この手法を定式化する。

(1) doインデックスを固定したループ内で使用される配列インデックスの範囲(i-min,i+max)を求める。
(2) i-minからi+max-1までの配列要素をloop_entryの前で、ロードする。レジスタの割り当ては一定規則で順に割り当てる。
(3) loop_bodyの先頭で、i+maxの配列要素をロードする。
(4) 計算式に合わせてコードを生成する。但し、ストア命令は最小インデックスのみ生成する。
(5) レジスタ割り当てをサイクリックに変え、(min+max+1)回のアンローリングを行なう。
(6) loop_returnを作成し、その後にループ内のストア命令で最小インデックス以外のコードを書く。

この方法により、数値演算などのdoループ内に配列演算を多く含む場合には、効率よいコードを生成できる。

(3) 転送部最適化

ADENAにおいて、並列実行に伴うPE間のデータ転送は、pass文によって陽に記述されている。よって、コンパイラによる転送動作の認識は不要である。しかし、PE間のデータ転送は、逐次実行時には存在しないのでオーバーヘッドとなり、できるだけ減らすことが必要である。

ADENAでは、転送専用のLSI(TCU)をもっているので演算しながら転送をおこなうDMA動作ができる。特に、EPUがメモリへのライトと同時にTCUへシグナルを送り、TCUはそのシグナルを受けたとき、1レコードだけ転送を実行するモード(FIFOemulationモード)を備えている。

よって、ソースコードよりデータ転送の型を検出し、できるだけ有利な転送型へ変形することが必要となる。

1) 転送型

pass文とそれに先行するpdo文のパラメータによって転送型が決まる。

past型　・・pdo内で変更されない配列の転送
future型・・pdo内で変更される配列の転送
now型　・・pdo内で変更されるが、pdo range とpass rangeが同じで、if文の外

a) past型
転送とpdo内の演算を同時に行なえる。

```
pdo j, k = 1, N
 do 1 i = 1, N
  u(i,j,k) = v(i,j,k)
1 continue
pend
pass i, j, k = 1, N
 w(i,j,k) = x(i,j,k)
pend
```

図3-1-a

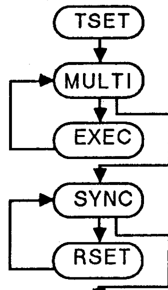


図3-1-b

b) future型
pdo内の演算が終了した後に、転送を開始する。

```
pdo j, k = 1, N
 do 1 i = 1, N
  u(i,j,k) = v(i,j,k)
1 continue
pend
pass i, j, k = L, M
 w(i,j,k) = u(i,j,k)
pend
```

図3-2-a

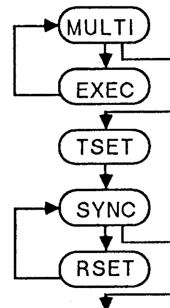


図3-2-b

c) now型
pdo内のストア動作が終了した後に、1レコード毎転送を開始する。

```
pdo j, k = 1, N
 do 1 i = 1, N
  u(i,j,k) = v(i,j,k)
1 continue
pend
pass i, j, k = 1, N
 w(i,j,k) = u(i,j,k)
pend
```

図3-3-a

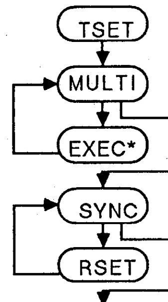


図3-3-b

TSET : TCUに対する命令をセットする
MULTI : EPUの多重使用処理を行なう
EXEC : pdo内の演算命令列
SYNC : 転送終了の検出及び同期
RSET : 再転送命令のセット

EXEC*: ストア命令をシグナル付ストア命令にかえる

past型転送は、バスに空きがあればいつでも転送を行なう。now型転送は、EPUからのシグナルを受け、かつバスに空き時間があれば転送を行なう。future型転送は、EPUの処理が終わった時点で初めて転送を開始する。

よって、以上より明らかなように、

past > now > future

の順に高速転送が期待できる。そこで、future型転送をnow型転送に、now型転送をpast型転送に変形することは有効な最適化手法となる。

now型転送は、EPUのDOループごとに1回の転送を行ない、past型転送はDOループにかかわらず転送を行なう。しかし、各ループ内にnow型転送を行なうのに十分なバスの空き時間があれば、past型転送を行なうのにも十分であり、なければ、いずれの転送もうまくはできない。

したがって、now型転送をpast型転送に変形することは、システムの高速化の点からは、あまり重要ではない。よって以後、future型転送をnow型転送に変形することを考える。

2) 転送型変形

future型転送とnow型転送は、いずれも先行するpdo文中で変更された配列を転送するものである。future型転送をnow型転送に変形するためには、

* if文やgoto文の影響をなくして、規則的にストアをする
* pdoとpassのrangeを合わせる

ことが必要である。よってこれらに対する方法を述べる。

a) デフォルト代入の追加

if文中の代入は条件判定により、配列への代入(ストア)が行なわれたり、行なわれなかったりする。しかし、now型の転送には規則的なストアが必要である。そこで、一時変数を用意し、do_bodyの先頭で代入される配列データをロードしておき、do_bodyの最後でその一時変数をストアする。その他のdo_bodyの演算は、その一時変数に対して行なう。

例外として、ブロックif文の場合は、else文を追加する。

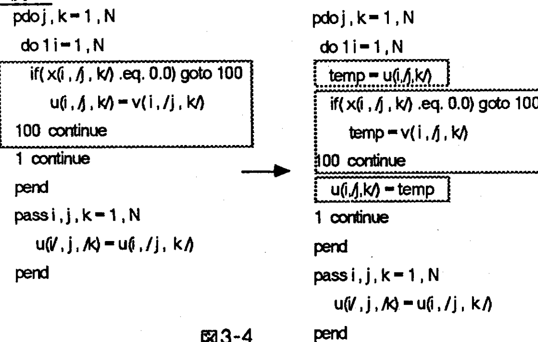


図3-4

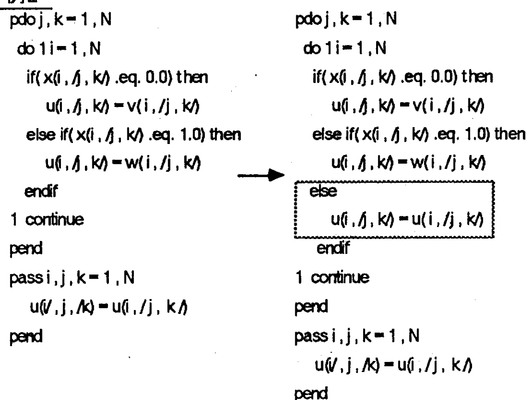


図3-5

b) rangeのマッチング

pdo rangeとpass rangeが異なる場合、EPUの出せるシグナルの数と転送すべきデータの数があわなくなり、now型転送ができない。

そこで、pass rangeにあわせてpdo/doループの分割/拡張を行なう。

まず、pass rangeが狭い場合は、pdo/doループをpass rangeと一致するpdo/doループとそうでないものに分割し、前者のストア命令のみをシグナル付にかえる。次に、pass rangeが広い場合は、pdo/doループを広い分だけ拡張し、不足分の転送すべき配列のロード/シグナル付ストアを書き込む。

これらの操作により、任意のrangeを持つfuture型転送が、now型転送に変形できる。しかし、複雑に分割/拡張手法を用いるので、実行時のオーバーヘッドが増えることになり、有効性の少ない場合も考えられる。ここでは、まず比較的簡単な例を示す。

例3は、代入されるdoループの最内側のrangeがpass rangeよりも大きい場合である。この時、はみ出したpdo内の代入をpdoの前で行なっておく。

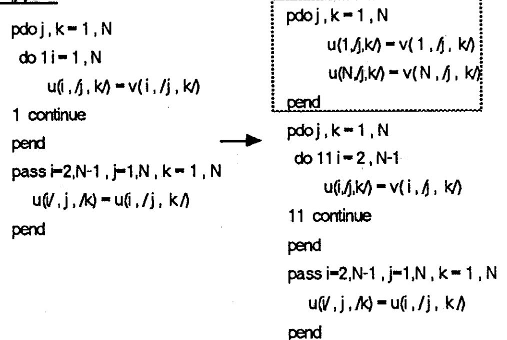


図3-6

例4は、例3の反対に代入されるdoループの最内側のrangeがpass rangeよりも小さい場合で、たらない代入を追加する。

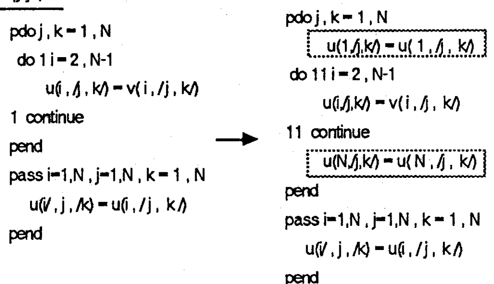


図3-7

次に、より複雑な最内側以外のrangeも異なる場合についてだが、下の例文のrangeの関係を図で示す。

```
pdo j,k=10,100
 do 1 i=10,100
   u(i,/j,k/)=f(v(i,/j,k/))
1 continue
pend
pass i,j,k=1,30
  u(i/,j,/k)=u(i,/j,k/)
pend
```

図3-7

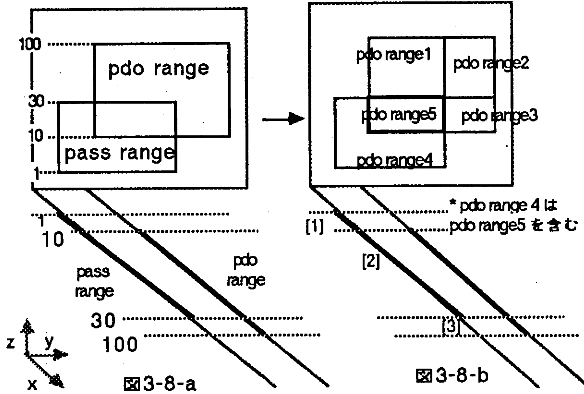


図3-8-a　図3-8-b

図中の矩形はpdo rangeとそれに対応するpass rangeを表わし、斜線はdo loopのrangeとそれに対応するpass rangeを表わす。図3-8-aのように、pdo rangeとpass rangeの一部が重なり、future型転送となる。
この矩形部を図3-8-bのように5つのrangeに分け、斜線部を3つに分ける。これに次の操作を行なう。

1. pdo range 5に対するマスクデータを作成する
2. pdo range 1～3に対するコードを作る(シグナル無)
3. pdo range 4に対するコードを作り、[1]に対しては、自分自身への代入のコード(シグナル有)を作り、[2]に対しては、マスクデータが真の場合は演算代入(シグナル有)を、そうでない場合は、自分自身への代入のコード(シグナル有)を作り、[3]に対しては、マスクデータが真の場合は、マスクデータが真の場合は演算代入(シグナル無)を、そうでない場合は、自分自身への代入のコード(シグナル無)を作る。

これに従い、前記の例を変形すると、

```
pdo j,k=10,100
  mask(1,/j,k/)=.true.
pend
pdo j=30,100,k=10,100
 do 1 i=10,100
  u(i,/j,k/)=f(v(i,/j,k/))
1 continue
pend
pdo j=10,30,k=30,100
 do 2 i=10,100
  u(i,/j,k/)=f(v(i,/j,k/))
2 continue
pend
pdo j=1,30,k=1,30
 do 3 i=1,10
   u(i,/j,k/)=u(i,/j,k/) (有)
3 continue
 do 4 i=10,30
   if(mask(1,/j,k/)) then
     u(i,/j,k/)=f(v(i,/j,k/)) (有)
   else
     u(i,/j,k/)=u(i,/j,k/) (有)
   endif
4 continue
 do 5 i=30,100
   if(mask(1,/j,k/)) then
     u(i,/j,k/)=f(v(i,/j,k/)) (無)
   else
     u(i,/j,k/)=u(i,/j,k/) (無)
   endif
5 continue
pend
pass i,j,k=1,30
  u(i/,j,/k)=u(i,/j,k/)
pend
```

* 有・・シグナル有
無・・シグナル無

図3-9

となる。図3-9の斜線で囲った部分は、続くpass文と同時に行なえ、転送はnow型となる。
この例のように、元のプログラム(図3-7)よりもプログラムは長くなり、オーバーヘッドも生じている(maskへの代入、maskによるif文、不要な代入文の増加)。しかし、通常のプログラム(pdo内でデータが確定し、そのデータをpass文で転送する)で考えると、pass rangeとpdorangeの差は、±1～3の場合が多い。
よって、この方法が有効な場合は多い。

(4) 差分式計算

偏微分方程式の反復解法等でよくでてくるものとして、doループを用いた差分式計算がある。ここでは、例を示して評価を行なう。

$$f_{i,j,k} = -V_{i,j-1,k} + (2-r)*V_{i,j,k} - V_{i,j+1,k} - V_{i,j,k-1} + (2-r)*V_{i,j,k} - V_{i,j,k+1}$$

$(i,j,k=2\sim N-1)$

図4-1

これを、FORTRANプログラムで記述すると、

```
do 1 k=2,N-1
do 1 j=2,N-1
do 2 i=2,N-1
  f(i,j,k)=-v(i,j+1,k)-v(i,j-1,k)
           -v(i,j,k+1)-v(i,j,k-1)+2*(2-r)*v(i,j,k)
2 continue
1 continue
```

図4-2

となる。一方、ADETRANプログラムで記述すると、

```
pdo k,i=2,N-1
 do 2 j=2,N-1
  f(i/,j, k)=-v(i/,j+1,/k)-v(i/,j-1,/k)+2*(2-r)*v(i/,j,/k)
2 continue
pend
pass i,j,k=1,N
 f(/i,j/,k)=f(i/,j,/k)
 v(/i,j/,k)=v(i/,j,/k)
pend
pdo i,j=2,N-1
 do 3 k=2,N-1
  f(/i,j/, k)=f(/i,j/,k)-v(/i,j/,k+1)-v(/i,j/,k-1)
3 continue
pend
```

図4-3

となる。
このプログラムのpdo rangeをやや一般化したものに対し、前述の手法を適用し、比較する。

1) 1個のPEで全計算をする場合
(当然、データ転送は無い)
2) 256個のPEで計算する場合
(future型転送で、ループ内最適化無)
3) 256個のPEで計算する場合
(future型転送で、ループ内最適化有)
4) 256個のPEで計算する場合
(now型転送で、ループ内最適化無)
5) 256個のPEで計算する場合
(now型転送で、ループ内最適化有)

上記パターンについて、配列サイズと最適化の効果及びpdoとpassのrangeの違いによる並列化及び最適化の効果を検討する。

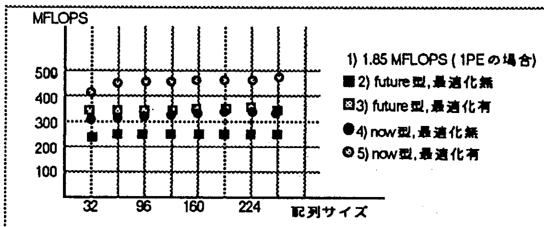
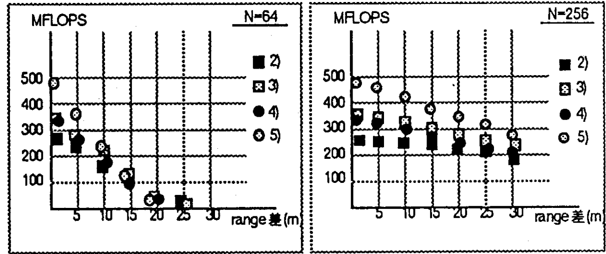


図4-4



pdo rangeとpass rangeの差と本方式の最適化の効果

図4-5-a　　図4-5-b

図4-4は、配列サイズNとFLOPSの関係を示し、図4-5は、

```
pdo k,i=m,N-m+1
 do 2 j=m,N-m+1
  f(i/,j, k)=-v(i/,j+1,/k)-v(i/,j-1,/k)+2*(2-r)*v(i/,j,/k)
2 continue
pend
pass i,j,k=1,N
 f(/i,j/,k)=f(i/,j,/k)
 v(/i,j/,k)=v(i/,j,/k)
pend
```

図4-6

のように、pdoとpassのrangeがmだけ異なっている時のFLOPSを示す。
図4-4からわかるように、ループ内最適化の効果は、配列サイズ(ベクトル計算のベクトルにあたる)が比較的小さい時でもすでに30%の効率上昇を得ており、サイズを大きくするにつれてその効果は大きくなっている。また、図4-5より、mが5〜20位になった時、future型からnow型への変形が効果を失っていることがわかる。しかし、通常のmは、1〜3位であり、本手法が効果を持つと考える。

(5) おわりに

以上述べてきたように、doループ最適化手法は、既存のベクトル演算方式とは異なり、ループサイズが比較的小さい時でも効果的であり、約30%の実行速度の向上が見られた。この方法は、ADENA以外の計算機でも有効である。
また、future型転送をnow型転送に変形することは、pdo rangeとpass rangeの差が小さい時(ループ長の10%以下)には極めて効果的であるが、差が大きくなる(ループ長の20%以上)と、最適化、さらには並列化の効果も下がってくる。しかし、通常の計算ではその差は小さく、また、その差の大きいと判断できる場合でも、now型変形制御文の挿入等を行なってコンパイラを制御することで、本手法も効果的になると思われる。

謝辞

本研究にあたり御指導御鞭たつをいただきました
京都大学野木助教授、半導体研究センター超LSI
デバイス研究所間野所長、廉田主任技師に感謝致
します。

参考文献

(1) H.Kadota and T.Nogi , " VLSI Computer with Data TransferNetwork : ADENA " , Proc. of 1989 International Conf.on Parallel Processing , pp. 319〜322 , 1989
(2) T.Nogi , " Parallel Computaion " , Studies in Mathematics and Its Applications , No.18 , North-Holland , Amsterdam pp. 279〜318 , 1986
(3) K.Kaneko et al. , " 64bit RISC Microprocessor for the Parallel Computer System " , 1989 International Solid-State Circuits Conf. Dig. Technical Papers , pp.78〜79 , 1989
(4) K.Nakakura , " A Versatile Data-Transfer Control Unit for a Parallel Processor System " , 1989 Symp. VLSI Circuits Dig. Technical Papers , 1989
(5) 谷川、金子" 並列計算機ADENA " , CPSY88-11,1989
(6) 若谷、野木、"並列計算機用言語ADETRAN処理系の開発"、情報処理学会第36回全国大会, p.153, 1988