# 並列性向上を意識した大域最適化の方法と実装

5G－4

遠藤浩太郎、竹内陽一郎、境隆二

（株）東芝　情報・通信システム技術研究所

## 1　はじめに

スーパスカラ、ＶＬＩＷなどの並列アーキテクチャをもつ計算機ではコンパイラが命令を並列にスケジュールして性能を引き出している。一方、並列スケジュールに先立って、コンパイラはループ不変式の削除、強度軽減、共通部分式の削除などの最適化（スカラ最適化）も行っている。

スカラ最適化は計算機アーキテクチャに依存しない技術であるが、従来のように逐次実行することを前提としている場合には、並列アーキテクチャに適さないコードを生成することがある。こういった場合、並列スケジュールによって並列度をあげることが困難となり、従って最適化アルゴリズムにおいて並列アーキテクチャを前提としたスカラ最適化が必要となる。

## 2　並列化の障害とスカラ最適化での対策

並列化の障害のひとつはレジスタの依存関係である。レジスタの依存関係はプログラムの意味から発生する本質的な前後関係なので、これを取り除くためにはスカラ最適化での式の変形が必要である。例えば（図２）では演算の順序を変えて式の高さを最小とすることにより並列度が上がっている。また（図４）では演算を畳み込むことによって並列度が上がっている。

もうひとつの並列化の障害はメモリアクセスの依存関係である。レジスタ間接アドレッシングによるメモリアクセスは他のメモリアクセスとの独立性が明らかでないために並列化の妨げとなる。メモリアクセスの独立性を保証するためにはアドレスの解析が必要であるが、アドレスを畳み込まれた式とするスカラ最適化でその解析が可能となる（図５）。

```
do
 {
  a[i][0]= j × a[i][1];
  i = i + 1 ;
}
while (i<100) ;
```

```
DO
  t ← [a + 8 × i + 4]
  [a + 8 × i] ← j × t
  i ← i + 1
WHILE (i<100)
```

（図１）並列性のあるループ

```
DO
  t ← [a + 8 × i + 4]
  [a + 8 × i] ← j × t
  i ← i + 1
  t ← [a + 8 × i + 4]
  [a + 8 × i] ← j × t
  i ← i + 1
  t ← [a + 8 × i + 4]
  [a + 8 × i] ← j × t
  i ← i + 1
  t ← [a + 8 × i + 4]
  [a + 8 × i] ← j × t
  i ← i + 1
WHILE (i<100)
```

（図３）ループ内の計算を展開

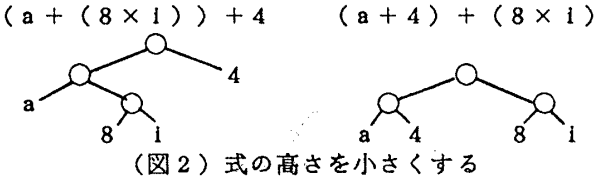
```
[a + 8 × i] ← j × t 1
i ← i + 1
t 2 ← [a + 8 × i + 4]
```

```
[a + 8 × i] ← j × t 1
t 2 ← [a + 8 × i + 1 2]
i ← i + 1
```

（図４）演算の畳み込み

```
t 2 ← [a + 8 × i + 1 2]
[a + 8 × i] ← j × t 1
i ← i + 1
```

（図５）ストアとロードは独立




（図２）式の高さを小さくする

---

Global optimization strategy to improve the parallel scheduling effect
Kotaro Endo, Yoichiro Takeuchi, Ryuji Sakai
TOSHIBA Corporation

スカラ最適化で演算を畳み込んで式としてとらえるという方法が、レジスタの依存関係を解消する効果をもち、また式の変形によって並列度を向上させることができ、さらにメモリアクセスの独立性の判定にも有用であるということができる。

3　具体例　－　ループアンロール

スカラ最適化によって並列度が顕著に向上する例としてループアンロールをあげる。仕組みはループ内の計算を展開するだけの単純なものであるが、スカラ最適化の機能により並列度が向上する。

例として（図１）のループを考える。展開した直後の（図３）ではｉのインクリメントが壁となってこのままでは展開前と並列性は変わらない。つぎの（図６）では演算の畳込みとレジスタリネーミング、そしてアドレスの解析が行われ並列度が向上した。最後に強度軽減が行われた（図７）。ここで並列性を得るためそれぞれのアドレス式ごとに強度軽減を行っている。

参考として当社ＶＬ２０００シリーズ（４並列ＶＬＩＷマシン）での並列スケジュールをした結果を（図８）に示す。

```
ＤＯ
  ｔ１←［ａ＋８×ｉ＋４］
  ｔ２←［ａ＋８×ｉ＋１２］
  ｔ３←［ａ＋８×ｉ＋２０］
  ｔ４←［ａ＋８×ｉ＋２８］
  ［ａ＋８×ｉ］←ｊ×ｔ１
  ［ａ＋８×ｉ＋８］←ｊ×ｔ２
  ［ａ＋８×ｉ＋１６］←ｊ×ｔ３
  ［ａ＋８×ｉ＋３２］←ｊ×ｔ４
  ｔ←ｔ４
  ｉ←ｉ＋４
ＷＨＩＬＥ（ｉ＜１００）．
```

（図６）ループアンロール

```
  ｒ１←ａ＋（８×ｉ）
  ｒ２←（ａ＋４）＋（８×ｉ）
  ｒ３←（ａ＋８）＋（８×ｉ）
  ｒ４←（ａ＋１２）＋（８×ｉ）
  ｒ５←（ａ＋１６）＋（８×ｉ）
  ｒ６←（ａ＋２０）＋（８×ｉ）
  ｒ７←（ａ＋２４）＋（８×ｉ）
  ｒ８←（ａ＋２８）＋（８×ｉ）
ＤＯ
  ｔ１←［ｒ２］
  ｔ２←［ｒ４］
  ｔ３←［ｒ６］
  ｔ←［ｒ８］
  ［ｒ１］←ｊ×ｔ１
  ［ｒ３］←ｊ×ｔ２
  ［ｒ５］←ｊ×ｔ３
  ［ｒ７］←ｊ×ｔ４
  ｉ←ｉ＋４
  ｒ１←ｒ１＋３２
  ｒ２←ｒ２＋３２
  ｒ３←ｒ３＋３２
  ｒ４←ｒ４＋３２
  ｒ５←ｒ５＋３２
  ｒ６←ｒ６＋３２
  ｒ７←ｒ７＋３２
  ｒ８←ｒ８＋３２
ＷＨＩＬＥ（ｉ＜１００）
```

（図７）並列性を意識した強度軽減

4　今後の課題

スカラ最適化による並列度の向上には物理レジスタの使用量が増えるという問題がある。例えば（図２）では少なくとも１個の物理レジスタを余分に必要としている。限界を越えたレジスタの使用はスピルの発生を引き起こし、性能を劣化させる。またこの問題はそのほかの大域最適化によっても引き起こされうるので、レジスタ使用量を増加させるそれらの要因を総合して調整する方法の確立がこれからの課題である。

参考文献

(1) Chow, F. A Portable machine independent global optimizer Ph.D. dissertation, Stanford University, 1984.

(2) 竹内陽一郎，境隆二　微視的並列度向上のための中間コード最適化戦略　情報処理学会第４３回全国大会，１９９１

```
add(r3,#28,r8)   sll(r0,#3,r1)    add(r3,#12,r7)
add(r3,#4,r4)    add(r3,#16,r14)  add(r3,#24,r6)   add(r3,#20,r13)
movrh(r2,f0)     add(r13,r1,r5)   add(r3,#8,r13)   add(r4,r1,r4)
add(r6,r1,r12)   add(r0,#9,r6)    add(r8,r1,r9)    add(r0,#4,r8)
add(r14,r1,r14)  add(r7,r1,r13)   add(r13,r1,r15)  add(r1,r3,r7)
LOOP:
fld([r4],f1)     tstgt(r6,#100,@0)
fld([r13],f2)    add(r4,#32,r4)   add(r6,#4,r6)    fmul(f0,f1,f1)
fld([r5],f2)     mov(r8,r0)       fst(f1,[r7])     fmul(f0,f2,f1)
fld([r9],f2)     add(r7,#32,r7)   fst(f1,[r15])    fmul(f0,f2,f1)
add(r9,#32,r9)   add(r1,#32,r1)   fst(f1,[r14])    fmul(f0,f2,f1)
add(r12,#32,r12) add(r13,#32,r13) fst(f1,[r12])    [0f]branch(#LOOP)
add(r15,#32,r15) add(r5,#32,r5)   add(r14,#32,r14) add(r8,#4,r8)
```

（図８）