「PC アーキテクチャ向け命令スケジューリングにおけるデータスタックの仮想多重使用とその効果」

長倉 裕叔（静岡大学大学院　情報学研究科　情報学専攻）
梅谷 征雄（静岡大学　情報学部　情報科学科）

1　はじめに

PC の代表的なプロセッサとして Pentium がある．Pentium アーキテクチャでは FPU（Floating Point Unit）データスタック [1] の存在が命令スケジューリングのネックとなっており，それは式の長い演算やループアンロールコードでは顕著である．ロード・ストア命令の Out of Order 実行が有効であることは知られているが，実数演算に関してはデータスタックによる拘束がスケジューリングの障害となる．そこで，我々はデータスタックを仮想多重使用することで，上記の障害を克服する事を思いついた．そして，仮想多重使用の機能を Pentium のスタック入れ替え命令を使って実現した．またスタック入れ替え命令の自動挿入アルゴリズムを考案し，アセンブラのコンバータとして実装した．これを遺伝的命令スケジューラ [2, 3] に組み込み，実験をおこなった．遺伝的命令スケジューラは実行時間の計測値に準拠した最適化を行なうため，それぞれのマシンに最適な命令列を出力するスケジューラである．LINPACK コード・LFK[4] を対象に評価し，その有効性を確認した．

2　データスタックの仮想多重使用

Pentium は FPU データレジスタをスタックのように使用している．そのため，浮動小数点演算において，そのスタックのトップのデータとの依存が強くなる．すなわち，浮動小数点演算命令の順序を入れ替えることができず，スケジューリングの効果が期待できない．

そこで，我々は SDN（Stack Data Number) というパラメータを各命令に設定する事にした．SDN の値は，スタックのデータの有効範囲に番号付けをしたものである．そして，スタックによる依存を削除する代わりに，この SDN を使って依存を生成することとした．

図 1 は評価の対象とした LINPACK のループを 2 回展開した UNROLL コードの一部である．命令間の依存グラフは図 2 の (a) のようになる．UNROLL によって計算が並列となっているが，スタックによる依存のために浮動小数点演算命令が入れ替えられない．これでは，UNROLL することによって得られる効果は非常に小さいと予想される．命令 2 と命令 7 の fld はロードであり，スタックにデータをプッシュする命令である．また，命令 6 と命令 11 の fstp はストア後にトップのデータをポップする命令である．この命令列では，命令 2・3・5・6，命令 7・8・10・11 のデータ有効範囲がある．この場合，命令 2・3・5・6 の SDN 値を 1，同様に命令 7・8・10・11 の SDN を 2 であると設定する．そして，SDN が等しい命令間に対して，依存を生成する．つまり，命令間の依存グラフは図 2 の (b) のようになる．

しかし，依存グラフ (b) を用いてスケジューリングを行なうと，FPU データスタックがプログラムで意図しているように使用されていないコードを生成してしまうことがある．そこで，スタック入れ替え命令 fxch 命令 [5] を挿入し，次の命令が必要としているデータを正しい位置に移動させることにした．この命令はスタックの内容を入れ替える命令である．「fxch ST(i)」ならば，ST(0) レジスタの内容と ST(i) レジスタの内容を入れ換えることになる．ST(0) レジスタとはスタックのトップのレジスタである．また，この fxch 命令はほとんどコストゼロの命令であり，余分なサイクルを消費しない [6]．よって，この命令を挿入しても，実行時間には影響しないと予想される．例えば，図 3 のように，fxch 命令を挿入する．挿入前は命令 7 でロードしたデータを命令 6 でストアしてしまうという動作になってしまっている．しかしながら，fxch 命令の挿入によって，スタックデータが入れ替わり，命令 5 の結果を命令 6 がストアするという正しい動作にすることができる．

3　仮想多重使用の実現方式

3.1　SDN の割り付け方法

ここでは，SDN の割り付け方法について述べる．浮動小数点命令は SDN が 1 以上，それ以外の命令は SDN が 0 と設定する．SDN による依存生成は SDN が 1 以上の命令に対して行なう．SDN を割り付けるために，作業用スタックと変数 i を用意する．i の初期値は 0 とする．そして，以下の手順で SDN を設定していく．

```
double da,dx[ ],dy[ ];                  1   @122:
int i,m,n;                              2           fld     qword ptr [ebp+12]
                                        3           fmul    qword ptr [ecx]
for (i = m; i ¡ n; i = i + 2)           4           add     edx,2
{dy[i] = dy[i] + da*dx[i];              5           fadd    qword ptr [eax]
dy[i+1] = dy[i+1] + da*dx[i+1];         6           fstp    qword ptr [eax]
}                                       7           fld     qword ptr [ebp+12]
                                        8           fmul    qword ptr [ecx+8]
                                        9           add     ecx,16
                                       10           fadd    qword ptr [eax+8]
                                       11           fstp    qword ptr [eax+8]
                                       12           add     eax,16
                                       13           cmp     esi,edx
                                       14           jg      short @122
```

図 1: ソースの一部とそのアセンブリ命令列

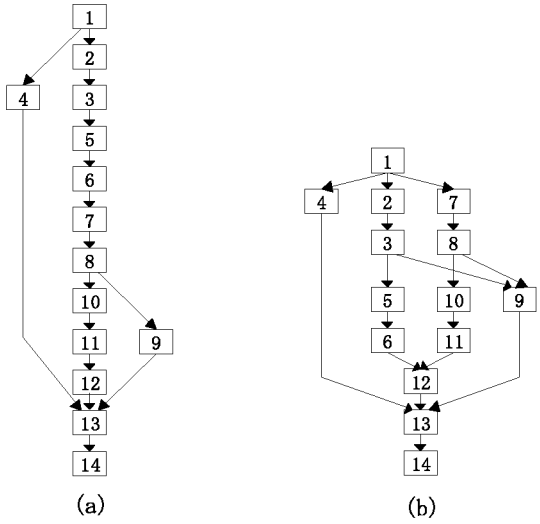

図 2: 図 1 のアセンブリ命令列の依存グラフ

手順１： 命令が浮動小数点命令でなければ，SDN を 0 とする．

手順２： 浮動小数点命令のロード命令であれば，i の値を 1 増やす．そして，その i の値を作業用スタックにプッシュし，SDN を i とする．

手順３： 浮動小数点命令であり，ロードまたはポップ命令でない命令の場合，作業用スタックのトップにある値を SDN とする．

手順４： 浮動小数点命令のポップ命令であれば，作業用スタックのトップにある値を SDN とする．その後，作業用スタックをポップする．

手順５： 対象とする命令列が終わってなければ，手順 1 へ戻り，次の命令の SDN を設定する．

3.2 SDN に基づく依存グラフの生成とスケジューリング

SDN をデータリソースと見なして，通常の方法で依存グラフを作成する．依存グラフの制約内で遺伝的命令スケジューリングを行なう．

5	fadd	qword ptr [eax]	5	fadd	qword ptr [eax]
7	fld	qword ptr [ebp+12]	7	fld	qword ptr [ebp+12]
6	fstp	qword ptr [eax]		fxch	st(1)
			6	fstp	qword ptr [eax]
		挿入前			挿入後

図 3: fxch 挿入例

3.3 fxch の自動挿入法

スケジューリング後は，作業用スタックを用意し，以下の手順で fxch 命令を挿入する．

手順１： 命令が浮動小数点命令のロード命令であれば，作業用スタックにその SDN をプッシュする．

手順２： 命令が浮動小数点命令であり，さらにロード命令でなければ，その命令の SDN と作業用スタックのトップの SDN の値を比較する．それらの値が等しければ，fxch 命令を挿入する必要はない．しかし，異なる場合は，作業用スタックのトップにその SDN の値がくるように fxch 命令を挿入し，その後，作業用スタックの値を交換する．

手順３： FPU データスタックをポップする命令である場合は，作業用スタックをポップする．

手順４： 対象とする命令列が終わってなければ，次の命令へ行き，手順１へ戻る．

4 有効性評価

ここでは遺伝的命令スケジューラに仮想多重使用とスタック入れ替え命令の自動挿入を組みこみ，LINPACK と LFK ベンチマークを利用して評価を行なう．LFK は FORTRAN で書かれた 24 個のカーネルループプログラムから構成されている．今回の実験では，FORTRAN から C に書き直したものを使用した．今回の実験で使用したマシンは GATEWAY Performance800（メモリ 192MB）であり，PentiumⅢ800MHz を搭載している．OS は Window98SE である．コンパイラは Borland C++ Builder4 の最適化オプションを使用した．また，実行時間の評価には Borland C++ Builder4 にある clock 関数を使用した．clock 関数は 2 つのイベント間の時間を ms 単位で計測できる．

4.1 評価結果

LINPACK の評価結果を表 1 に，LFK のうち効果の大きかったカーネルループの効果を表 2 に示す．それぞれの表において，
BCC:Borland C++ Builder4 コンパイラによる実行時間
GA:遺伝的命令スケジューラによる実行時間
GA+fxch:遺伝的命令スケジューラに仮想多重使用を追加した場合の実行時間
改良度 GA(BCC):100-GA/BCC*100
改良度 GA+fxch(BCC):100-(GA+fxch)/BCC*100
改良度 GA+fxch(GA):100-(GA+fxch)/GA*100
となっている．

表 1 において，NOROLL は C 言語レベルでループ展開をしていないものであり，UNROLL はソースレベルでループ展開したものである．UNROLL の数値はそれぞれ何回のループ展開をしたかを示している．

4.2 考察

LINPACK では，BCC と GA では UNROLL の効果がほとんど出ていないが，fxch 命令を入れることにより UNROLL に大きな効果が得られた．これは演算命令とロード・ストア命令が最適にスケジュールされた結果と思われる [7].

表 1: LINPACK 評価結果

LINPACK	BCC (ms)	GA (ms)	GA+fxch (ms)	改良度 GA(BCC) (%)	改良度 GA+fxch(BCC) (%)	改良度 GA+fxch(GA) (%)
NOROLL	1032	1029	1029	0.3	0.3	0.0
UNROLL2	1013	1013	825	0.0	18.6	18.6
UNROLL3	1047	1047	801	0.0	23.5	23.5
UNROLL4	1030	1026	810	0.4	21.4	21.1

表 2: LFK 評価結果

LFK No.	BCC (ms)	GA (ms)	GA+fxch (ms)	改良度 GA(BCC) (%)	改良度 GA+fxch(BCC) (%)	改良度 GA+fxch(GA) (%)
7	959	949	844	1.0	12.0	11.1
8	194	189	182	2.6	6.2	3.7
9	1554	1533	1395	1.4	10.2	9.0
16	344	344	309	0.0	10.2	10.2

LFK では LINPACK の UNROLL ほどではないが，４個のカーネルで顕著な効果が得られた．これは，fxch 命令の挿入により演算命令が良くスケジュールされた結果である．ちなみに，効果は LINPACK の UNROLL3 で最も大きかった．その時に挿入された fxch 命令の数は４個であった．

5 結言

スケジューラに仮想多重使用とスタック入れ替え命令の自動挿入アルゴリズムを組みこむことで，データスタックによる命令間依存を回避し，多くの浮動小数点計算を含む Pentium アセンブリ命令列を効果的にスケジューリングできた．

今後の課題として，次の点が挙げられる．

- より大規模な問題を対象とした評価．
- リストスケジュール等の命令スケジューラへのスタック仮想多重使用の適用．

参考文献

[1] Intel Corporation:“ インテル・アーキテクチャ・ソフトウェア・ディベロッパーズ・マニュアル　上巻　基本アーキテクチャ ”, 資料番号 243190J, (1999)

[2] 伊東勇, 梅谷征雄: “ 遺伝的アルゴリズムを用いる命令スケジューリング方式とその効果 ”, 情報処理学会論文誌, Vol. 41, No. 4, pp. 1073–1085 (2000).

[3] 長倉裕叔　梅谷征雄 :“ PC アーキテクチャにおける遺伝的命令スケジューリングの適用実験 ”, 情報処理学会研究報告 Vol. 2000, No. 57, pp. 31–36(2000)

[4] McMahon, F.H.: “ The livermore fortran kernels: A computer test of numerical performance range ”, Technical report, Lawrence Livermore National Laboratory (Dec. 1986).

[5] Intel Corporation:“ インテル・アーキテクチャ・ソフトウェア・ディベロッパーズ・マニュアル　中巻　命令セット・リファレンス ”, 資料番号 243191J, (1999)

[6] Intel Corporation:“ インテルアーキテクチャ最適化リファレンスマニュアル ”, 資料番号 730795J-001,(1999)

[7] Manoj Franklin and Gurindar S.Sohi:“ A Hardware Mechanism for Dynamic Reordering of Memory References ”, IEEE TRANSACTIONS ON COMPUTERS, Vol. 45, No. 5, pp. 552–571 (May. 1996)