ＢＬアドレシングを用いた数値計算の高速化

1X-6

市川　周一[1]、佐藤　三久[1]、後藤　英一[2,3,1]

1.新技術開発事業団、 2.東京大学理学部、 3.理化学研究所

はじめに

筆者らは、アドレス計算に簡単な範囲検査機構と分岐制御を付加することによって、LISPやFORTRANなど広い範囲の応用に対して有効な機能を提供できることを示し、それをＢＬアドレシングと名付けた[1,2]。さらに、ＢＬアドレシング・モードを実装する計算機FLATS2を設計し、コンパイラを作成して、各種の応用に対する評価をすすめている。

本稿では、典型的な数値演算応用にＢＬアドレシングを適用し、高速化する方法を示す。さらに、いくつかのベンチマーク・プログラムをシミュレータ上で評価した結果を報告する。

ＢＬアドレシング

数値計算プログラムでは、数値的処理が実行時間の大半を占めるため、演算器に連続してデータを供給することが重要である。数値処理はループ構造の中で行なわれる比率が高く、しかもループ内の演算は均一で単純である場合が多い。このような場合、ループ内の数値演算時間に比してループ構造を回るオーバーヘッドが大きくなり、ループ内での演算密度（演算数と命令数の比）が下がりやすい。

ＢＬアドレシングはループ制御を演算自体とオーバーラップして実行する。従ってループのオーバーヘッドは実効上0になり、演算密度を高く保つことができる。以下に、ＢＬアドレシングの概要を示す。

ＢＬアドレシングでは、アドレス計算を行なう際に必ずアクセス範囲の検査を行なう。具体的には、アクセスするアドレスがベース・アドレス（Ｂ）とリミット・アドレス（Ｌ）の間に含まれているか否かを検査する。

アドレスがＢＬの範囲内にあればアクセスは許可され、オペランドがメモリからフェッチされる。このとき同時に、命令内で指定した番地へジャンプする。アドレスが範囲外であった場合、次の番地へ進む（またはトラップされる）。

アドレシング・モードに副作用がある場合（自動インクリメント／自動デクリメント等）、副作用の書き込みも同時に行なわれる。

こうしてフェッチされたオペランドは、命令の指示にしたがって処理され、結果が格納される。

結局、ＢＬアドレシングを用いた命令では、以下の５つの作業を１命令内で（１命令サイクル内に）並行して行なうことになる：

(1) オペランドのアドレス計算、
(2) アドレスの範囲検査と分岐、
(3) アドレシングの副作用の書き込み、
(4) オペランドのフェッチ、
(5) 演算の実行。

数値演算においては、(1)はオペランドのアクセスに使用され、(2)はループの脱出判定に、(3)はループ変数の変更に、(4)と(5)はループ本体の実行に使用される。

ＢＬアドレシングを実装するには、２組のアドレス比較器と、若干の分岐制御論理が必要である。しかし、それ以外のハードウェアについては、もともと備わっているものを流用できる。

例えば、ＢＬアドレシングでは全ての命令に分岐フィールドがあるが、通常の条件分岐／無条件分岐命令を実行するためのハードウェアを共用することができるので、わずかな制御論理を付加するだけでＢＬアドレシングを実装することができる。

また、範囲検査を行なうためには、アドレスを比較する時間が必要になるが、その時間はメモリ・アクセスのサイクル・タイムにオーバーラップさせることができるので、範囲検査によって遅延時間が不要に増加することはない。

数値演算への適用

数値演算では、データの配列に連続的にアクセスして、一定の操作を行なう場合が多い。ここでは例として、配列の総和を計算するプログラムを、ＢＬアドレシングでどのように実現するか示す。

例：　配列の総和

```
for i := 0 to IMAX do
    S := S + array[i];
od;
```

このプログラムは、ＢＬアドレシング・モードで、以下の様な１命令ループに最適化される：

```
loop:  add.j  BL:(B)++4, S, S, loop
```

ただし、あらかじめＢレジスタにベース・アドレス（array[0]のアドレス）、Ｌレジスタにリミット・アドレス（array[IMAX]）を用意しておく必要がある。

add.jは「ジャンプ付き加算命令」を表わし、メモリ・アクセスが成功すれば「loop」に分岐する。この場合、loopは自分自身のアドレスなので、自己命令ループとなる。

The optimization of numerical applications with BL-addressing

Shuichi Ichikawa[1], Mitsuhisa Sato[1], and Eiichi Goto[2,3,1]

1. Research Development Corporation of Japan,
2. University of Tokyo, Faculty of Science,
3. The Institute of Physical and Chemical Research

「BL:」は、BとLレジスタでアドレス範囲検査を行なうことを示す。この検査で失敗するのは配列外へアクセスした場合（配列へのアクセスが終了した場合）であるから、ループを脱出して次の命令へ進むことになる。つまり、この範囲検査で、ループの脱出判定を行なうことになる。

「(B)」は実効アドレスがBレジスタの値であることを示し、「++4」はアドレス生成後にBレジスタの値を4増やすことを示す。これは、いわゆる副作用付きアドレシング・モードであるが、ここではループ変数の更新を兼ねている。「S」はレジスタの名前で、メモリ上のオペランドとSレジスタの値を足して、Sレジスタに格納することを示す。

この例では、ハードウェアの削減のため、インデックス型のアドレシングをポインタ型アドレシングに変形して用いているが、インデックスを副作用で修飾するアドレシング・モードを実装すれば、このような変形は不要になる。

この例でもわかるように、BLアドレシング法を用いると、ループの境界判定や変数更新が演算自体にオーバーラップして実行されるため、事実上ループのオーバーヘッドが0になる。そのため、ループ内では命令あたりの演算数が1となり、演算器にオペランドを連続して投入することができる。数値計算では演算器を遊ばせないことが重要であるため、このような工夫が性能向上に大きく寄与する。

ベンチマーク・プログラムでの評価

このようなBLアドレシングの利点を実証するため、筆者らはBLアドレシングを命令セットに含む計算機FLATS2を設計し[1]、評価を進めている。

以下では、FLATS2の命令セットを用いて、主要な数値演算用ベンチマーク・プログラムを評価した結果を示す。評価にあたっては、GCC（GNU C コンパイラ）をFLATS2用に移植したものを用いた。ベンチマーク・プログラム（C言語版）は、netlib[3]から入手した。

まず、配列演算の多い2つのベンチマーク・プログラムについて、その最内周ループがどの程度BLアドレシングを用いて最適化できるのか示す（表1、表2）。表からわかるように、最内周での演算密度は1に近い。最内周ループは繰り返し（多くの回数）実行されるので、ここでの演算密度が演算性能の上限を決める。FLATS2では、BLアドレシングによって、演算密度が極めて高くなっている。

表1： LINPAC最下位ルーチン

関数名	意味	最内周命令数	演算／命令
dasum	Σ | x |	2	1
daxpy	y := y + ax	2	1
dcopy	y := x	1	−
ddot	Σ(xy)	1	2
drot	2次元の回転	6	1
dscal	x := ax	1	1
dswap	x と y の交換	3	−
idamax	絶対値最大の添字	4	1／4

表2： リバモア・ループ

番号	意味	最内周命令数	演算／命令
3	内積	1	2
4	線形方程式	4	1／2
5	三角行列変形	4	1／2
6	線形リカレンス	3	2／3
11	積分	2	1／2
12	差分	2	1／2
21	行列積	2	1

次に、ベンチマークでの性能（予測）を示す。68020は実測、FLATS2はシミュレータでの評価、SPARCとR2000は公称値である。

性能比較

Architecture CPU + FPU	clock MHz	公称 MIPS	Dhrystone KD/s	Whetstone MW/s	Linpac MFLOPS
68020+68881	20	3	4.3	1.0	0.12
FLATS2(1CPU)	3.7	3	5.4	2.6	1.5
SPARC+WEITEK	16	10	19.	3.9	1.1
R2000+R2010	16	12	25.	9.0	1.8

BLアドレシングを用いたFLATS2では、MIPS値に対するMFLOPS値の比がたいへん大きくなっている（演算密度が高い）。そのため、MIPS値が低いにも関わらず、Linpacの性能が高くなる。

FLATS2のクロックは低いが、これは実装密度が低い（MSI＋多層基板）ためであって、BLアドレシングのせいではない。他の3例と同じ構成（CPU＋FPU）で、BLアドレシングを持つ計算機は設計できる。

RISC等のアーキテクチャでは、BLアドレシングで実現しているような『命令内の並列性』を充分に活用していないのではないだろうか。クロックを上げてパイプラインに連続的に命令を供給することは重要ではあるが、パイプライン的オーバーラップでは、ループのオーバーヘッドは0にはならない。並列に実行できる仕事は完全にオーバーラップさせて実行し、見かけの実行時間を0にすることが必要である。

BLアドレシングは、このような並列性を極めて自然な形で利用していると考えられる。

参考文献

[1] 市川・加藤・後藤：循環パイプライン計算機 FLATS2、情処研報、88-ARC-72-1、1988.

[2] M.Sato, S.Ichikawa, and E.Goto: "RUN-TIME CHECKING IN LISP BY INTEGRATING MEMORY ADDRESSING AND RANGE CHECKING", Proc. 16th Int'l Symp. on Comp. Arch., ACM, pp.290-297, 1989.

[3] J.Dongarra and E.Grosse: "Distribution of mathematical software via electronic mail", CACM, vol.30, no.5, pp.403-407, 1987.