4P-1　高並列計算機における遠隔データの先行フェッチのためのループ変換手法

戸田賢二　西田健次　内堀義信　島田俊夫

(電子技術総合研究所)

1、はじめに

並列処理計算機においては演算装置の要素数が増加するに従い、通信網での遅延が増加し遠隔データのアクセスに時間がかかるようになる。従って、高並列処理のためにはこの遅延に対し耐性のあるアーキテクチャが要求される。

データ駆動アーキテクチャは、命令レベルの並列性を取り出すことによって、通信網を含んだ循環パイプラインを実現しこの遅延をキャンセルできると考えられた。しかし、2オペランド命令の実行に2個のデータ入力を必要とする実現においては、データ待ち合わせ部に最初のデータが到着した際、命令実行部には実行すべき命令が送られずパイプラインブレイクを生じてしまう(ここでは、データ待ち合わせ部と命令実行部のパイプライン処理速度を同一と見なしているが、この仮定は、同じ実現技術を用い、両処理部の高い利用率を確保しようとする場合、一般的なものであろう。)。この問題の解決には、演算レジスタや実行順序制御の導入等の方法によってデータ転送数を減少させる工夫が必要である[1]。

一方、逐次処理型の演算装置を用いた並列処理計算機においては、個々の演算装置内では連続する命令列による縦型パイプライン処理が行なわれている。従ってマルチコンテキスト用の演算レジスタセットがサポートされていたとしても、コンテキスト切り替えの際演算パイプラインに空きが生じてしまうので、できるだけコンテキスト切り替えを行なわない様にする必要がある。すなわち、プロセス間の同期のため待が生じた場合は止むを得ないが、すでに値が入っている遠隔変数を読む場合はできるだけ待が生じないようにしてコンテキスト切り替えの頻度や演算装置のアイドル時間を減少させる必要がある。先行フェッチを可能にするアーキテクチャについては、データ読みだし要求をその番地の内容が確定するまでその番地で待たせる機能を持ったメモリを導入する方法が提案されている[2]。筆者らは、レジスタ間演算を基本とするＲＩＳＣアーキテクチャを適用するため、演算レジスタに対し遠隔データのフェッチを行える機構を示した[3]。本稿では、先行フェッチのために必要なハードウェア機能を示した後、プログラムのループ構造部分において、遠隔変数に対するデータフェッチ要求をその使用に先だって先行発行するための手法について述べる。

2、先行フェッチのためのハードウェアサポート

ここでは先行フェッチのために必要なハードウェアの機能を述べる。全体のシステムは複数の基本演算装置(ＰＥ)からなり、個々のＰＥは、演算部とそれが直接アクセスできる局所記憶を持っている。演算部は、他のＰＥの局所記憶(ＰＥと独立した共有記憶を仮定してもよい)に遠隔アクセスすることによって変数を共有することができる。個々の記憶要素にはデータの存在を示すフラグがあり、データの存在していない記憶要素に対する参照は、そこにデータが書き込まれるまで待たされる。演算部は、遠隔地のデータを局所記憶にフェッチする要求を出すと同時にその局所記憶要素の存在フラグを落とす。演算部がその局所記憶要素を参照した時、それが(遠隔地からのデータ転送によって)読みだし可能になっていれば演算を継続し、もしまだデータが存在していなければ、他のコンテキストの実行に切り替えるか、アイドルしてデータ到着を待つ。(実行の順序等によって)遠隔地のデータが読みだし前に必ず存在していることが保証されていれば、記憶要素でデータ参照要求をそこにデータが書き込まれるまで待たせる機構が不要になり、データ存在のフラグがあればよいことになる。

3、ループにおける先行フェッチ手法

遠隔データのフェッチ命令とその使用命令との間にそれと独立な命令を挿入する必要がある。原理的には、変数のデータ依存解析を行い複数の遠隔データのフェッチに対して全体として最大数の命令が挿入できるようにプログラムを変換すればよい。しかし、この時同時に次のことを考慮しなければならない。

○フェッチの先行によるコンテキスト間の並列性の減少。
○フェッチを一時に多量に発生することによる、通信網での衝突増加。
○特定遠隔記憶要素にフェッチが集中した際の応答の遅れ。
○多量の先行フェッチによる局所記憶資源の浪費。

良い変換を行なうためには、プログラムの性質とマシンの仕様を勘案しなければならない。最適解を求めることはそれほど重要でなく、ある程度有効な解を簡単な操作で得ることが重要である。従って、ここでは、科学技術計算で中心的な制御構造であるループについて簡単で有効な書換え規則を与える。以下では、簡単化のため配列要素は遠隔データであり、それ以外は局所記憶上のデータとする。

(1)半世代先行フェッチ

現在の世代で配列要素を用いた処理が終わり次第、次の世代で用いる配列要素の先行フェッチを行なう。最初の世代のフェッチはループの前処理として、ループの実行に先だって行ない、最後の世代の実行は、次の世代のフェッチが必要ないため、ループの後処理として、ループ実行の後行なう(図1)。フェッチは、現在の世代の残りの処理とループ制御処理の分だけ先行する。先行フェッチのための局所記憶領域は1世代分である。

配列の添え字を求めるのに計算が必要な場合、それはフェッチの前に処理されなければならない。従って、そ

A Loop Optimizing Technique
for the Remote-Data Prefetch
Kenji TODA, Kenji NISHIDA, Yoshinobu Uchibori
and Toshio SHIMADA
Electrotechnical Laboratory

の処理が大きくて配列要素に対する処理が小さい場合は、この変換によるフェッチ先行の程度は少なくなる。

（２）複数世代先行フェッチ

ｎ世代を融合して１つの世代として扱う手法で、図２に示すように個々の世代のフェッチとその演算部分をｎ個分互い違いに組み合わせたものである。ループの前処理と後処理は何れも、ｎ個分となる。フェッチはｎ－１世代分先行し、そのための局所記憶領域はｎ世代分である。この変換は、ｎ世代分のフェッチとその演算部分をそれぞれ一括して行なう方法と比較して、フェッチ要求発生の平均化、フェッチの先行度の均一化の利点がある。

現在の世代を実行しないと次の世代の配列の添え字が決まらない場合は、この変換は適用できない。

上に述べた（１）または（２）の変換を行なうに当たって、次の規則を適用し、配列要素に対するアクセス数の減少とループ間の依存関係の緩和を行なう。

○世代内での同一配列要素に対する複数参照や複数代入を局所記憶の利用により単一化する。
○現在の世代で値の定まった配列要素を先の世代で参照する場合、その値を局所記憶に保持することにより、その配列要素の参照を取り除く。この結果、その配列要素に対する代入も不要になればそれも取り除く。最終的な値が必要な場合は、ループの後処理の部分でその代入を行なう。

```
                              /*半世代先行フェッチ*/
for(i=0; i<n; i++){           0世代のフェッチ;
    ループ本体;          ⇨    for(i=0; i<n-1; i++){
}                                 i世代の演算;
                                  i+1世代のフェッチ;
                              }
                              n-1世代の演算;
```

図１．半世代先行フェッチの構造

```
/*２世代先行フェッチの例（ｎ：偶数）*/

0世代のフェッチ;          /*局所記憶領域Ａ*/
1世代のフェッチ;          /*局所記憶領域Ｂ*/
for(i=0; i<n-2; i+=2){
    i世代の演算;                /*局所記憶領域Ａ*/
    i+2世代のフェッチ; /*局所記憶領域Ａ*/
    i+1世代の演算;              /*局所記憶領域Ｂ*/
    i+3世代のフェッチ; /*局所記憶領域Ｂ*/
}
n-2世代の演算;                  /*局所記憶領域Ａ*/
n-1世代の演算;                  /*局所記憶領域Ｂ*/
```

図２．複数世代先行フェッチの構造（２世代の例）

４、先行フェッチと条件処理

ループ内に配列要素のアクセスに関する条件処理がある場合、先行フェッチの処理に注意が必要である。すなわち、使用しない配列要素に対して先行フェッチを行なった場合、その応答が帰ってきたのを確認した後でないと現在のプロセスを終了できないというものである。これは、データ駆動計算機における残留パケットの問題と同じであって、遠隔フェッチに対応する局所記憶要素を必ず参照するコードを生成することによって解決する。ある条件の成立する確率が低いことが分かれば、その条件が成立した時必要とされる配列については、先行フェッチを行なわない方が効率的である。一方、ループからのブレイクの様に通常はループが実行される場合、ブレイクした時点で発行されている先行フェッチについては、ブレイク後その対応する局所記憶要素を漏れなく参照するコードを生成しなければならない。

５、おわりに

本手法は、与えられたループの構造をそのまま利用するため、手軽な反面、適用できるループの構造に対する制限、その変換コードの効率等において改善の余地がある。しかしながら、リバモアループ（ローレンスリバモア研究所が科学技術計算の分野で代表的であるとして選んだ２４個のループ構造）に適用したところ、大部分のループに対して複数世代先行フェッチが可能であった。本手法によって、一般的なループ構造において十分な先行フェッチが可能であると思われる。ここでは、プログラムのソースレベルの書換えによって先行フェッチを実現しているが、変換がかなり機械的であるため、コンパイラに組み込んでマシンの仕様に適した先行フェッチを自動的に行なう様にすることが可能であろう。

筆者らが先に示したアーキテクチャ［３］によると、先行フェッチのための局所記憶として演算レジスタを用いることができ、レジスタへのロード命令が遠隔記憶要素に対するフェッチ要求を発行し、通常のレジスタを参照する命令において、そのデータ存在がチェックされる。従って、専用のフェッチ命令によるオーバヘッドが避けられる上プロセス切り替えの機会が減少するので、レジスタ間演算中心のＲＩＳＣ流の動作が効率よく行えることが期待できる。

謝辞

本研究の機会を与えて下さった、棟上昭男情報アーキテクチャ部長、有益なご討論を頂いた計算機方式研究室の同僚諸氏に感謝致します。

参考文献

［１］Sakai,S. et al., "An Architecture of Data-flow Single Chip Processor", pp.46-53, Proc. of ISCA, June 1989.

［２］Buehrer,R. and Ekanadham,K., "Incorporationg Data Flow Idears into von Neumann Processors for Parallel Execution",IEEE Trans. on Computers, Vol. C-36, No.12, December 1987.

［３］戸田　内堀　島田、「マクロデータフローアーキテクチャの検討」、並列処理シンポジウム、１９８９年２月.