

CODA におけるスキップ機構の導入による

7 P-6

共有変数アクセスの効率化*

西田健次[†] 戸田賢二, 内堀義信, 島田俊夫電子技術総合研究所[‡]

1 はじめに

CODA[2]は、並列処理環境においても処理の持つ局所性を活用し、演算器の利用効率を高く保つことを目標としている。CODA プロセッサ[3]は、RISCアーキテクチャにバケット通信機能を融合したもので、レジスタ上で変数の同期をとることが可能であり、命令挿入機構を用いることによりバケットからのデータを直接レジスタに取り込むことができる。プロセッサ間の通信はバケットを通じて行なわれるが、これにはプロセスの生成・発火、共有変数を介したプロセス間通信が含まれる。

共有変数に対するアクセスはバケットを使用するのを基本とするが、自PEのメモリに割り付けられた共有データに対してバケットを使用することは、本来不要なバケットを発生することによるネットワークの負荷の増大、および、メモリアクセスの局所性を生かしていないという点で望ましいものではない。そこで、バケット送出命令にスキップ機能を組み込むことにより、実行命令数を増加させずに、遠隔メモリへのアクセスと自メモリへのアクセスでの処理の切替えを行なう方法を提案する。

2 CODA における共有データアクセス

CODAの実行方式[1]では、プロセス起動時の引数の同期はレジスタ上で可能となっている。一方、プロセス間で共有される変数に対するアクセスは書き込み/読み出しの順序を完全に規定することはできないので、メモリ上で同期をとらなくてはならない。そこで、共有変数からの読み出しを行なう際に、共有変数の値が未定義であれば、読み出しリクエストを共有変数のメモセル上で待たせておくリクエストフック機構[2]を用いる。リクエストフック機構は、メモリ読み出し命令とフラグ判定・メモリ書き込みを同時に行なう複合命令の2命令を不可分に実行することで実現

される[1]。共有変数の読み出し操作(リクエストフックリード)では、1命令目でメモリを読み出し、2命令目でメモリのフラグ判定を行ない、データが未定義ならば読み出そうとしたプロセスのIDをメモリに書き込む。共有変数の書き込み操作(リクエストフックライト)では、1命令目でメモリを読み出し、2命令目でフラグ判定を行ない、データが未定義の場合にデータを書き込む(データが既に定義されている場合はエラー)。

実際のプログラム上では共有変数に対するアクセスは、バケット送出命令を実行するのみで、リクエストフック操作はバケットからの命令挿入で行なう。バケット送出には2命令を必要とし、また、リクエストフック操作も2命令を必要とする。しかし、共有変数が自メモリに割り付けられた場合、バケット送出を行わずに直接リクエストフック操作を行なうことが可能である。これによりバケット数の削減によってネットワーク負荷を軽減し、また、バケット送出・受信のためのオーバーヘッドを削除することにより、システム全体の処理効率を改善することができる。

共有変数の割り付けは動的に行なわれるため、どちらの命令コードを実行するかをコンパイル時に決定することはできない。CODAでは、メモリアドレスの最上位ビットによって遠隔メモリアクセスと自メモリアクセスの区別を行なうため、条件分岐命令を用いて命令コードを切替えることも可能であるが、(バケット数の削減する効果はあるものの)余分な命令を実行することになり処理効率の改善を妨げることになる。そこで、バケット送出命令にスキップ機能を組み込むことにより実行命令数を増加させずに遠隔アクセスと自メモリへのアクセスでの処理を切替える。

3 遅延スキップ機構の実装

バケット送出操作は、バケットヘッダのバケットポートへの書き込みと、バケットデータのポートへの書き込みの2命令によって実行される。バケットデータの書き込みと同時にバケット送出信号を発生することによりネットワークにバケットが送出される。

スキップ機能はバケットヘッダの書き込み命令(バ

*Delayed Skip Mechanism for Optimization of Shared Variable Access in CODA

[†]Kenji NISHIDA, Kenji TODA, Toshio SHIMADA, Yoshinobu UCHIBORI

[‡]Electrotechnical Laboratory

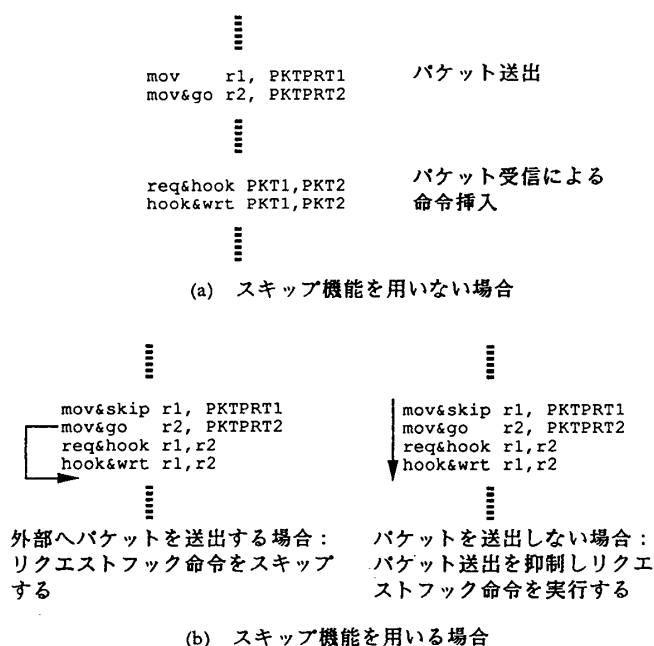


図 1: スキップ機構を利用したコーディングの例

ケット送出操作の 1 命令目) に組み込む。スキップ機能を組み込んだ場合は図 1(b) の様にコーディングする。パケットヘッダのデスティネーションアドレスが遠隔メモリである場合、メモリアドレスの最上位ビットが 1 となっている。この場合、次のパケットデータの書き込み命令を実行した後に続く 2 命令 (リクエストフック操作) をスキップする。パケットヘッダのデスティネーションアドレスが自メモリを示す場合 (最上位ビットは 0)、パケット送出信号を無効化し、スキップは行なわない。これによりパケットの送出を抑止すると同時に自メモリに対するリクエストフック操作を行なう。

CODA では、1 サイクルの命令先行フェッチを行っているため、スキップ判定から 1 命令の遅延の後に実際にスキップが行なわれる。これは、次々命令のアドレス計算を行なう際にスキップすべき命令数を足し込むことで実現されるが、2 命令のスキップを実現するためには、メモリアドレスの最上位 1 ビットを次々命令計算用の ALU の下位 2 ビット目に加えるのみでよい。次々命令アドレス計算用の ALU は分岐命令のために必須のものであるが、分岐命令以外では使用されていないので、スキップ機構を実装することにより次々命令アドレスの計算に余分なハードウェアを付加する必要はない。

4 まとめ: 遅延スキップ機構の得失

パイプラインに空きを作らないためには、スキップ判定の後 1 命令の遅延を必要とする。CODA での共有変数へのアクセスは常に 2 命令のパケット送出操作と

して実行するため、一命令目にスキップ機構を組み込むことにより、スキップによるオーバーヘッドは発生しない。

遅延スキップ機能を組み込んだ場合でも、遠隔メモリに対するアクセスを行なう場合にはパケット送出命令を実行するのみであるから、実行命令数は増加しない。また、自メモリに対するアクセスを行なう際には、パケット送出命令をスキップ判定のために 2 命令実行した後にリクエストフック操作を行なうため、2 命令増加しているように見える。しかしスキップ機能を用いずにパケット送出を行なった場合、最終的にはパケット到着によってリクエストフック操作が命令挿入されることを考えると、実行命令数には差がないと考えられる。

共有変数のアクセス (リクエストフック操作) にスキップ機能を組み込むことの利点は、(1) 自メモリに割り付けられた共有変数に対するアクセスを高速化できる、(2) 不要なパケット発生を抑制することによりネットワーク負荷を軽減できる、(3) 実行命令数を増加させない等があげられる。一方、スキップ機能によってパケット生成を抑止することによる欠点は、自メモリアクセスと遠隔メモリアクセスの両方の命令コードを記述しておかなくてはならないため静的なプログラムサイズが増加することがあげられる。

以上の考察により、パケット送出操作にスキップ機能を組み込むことは不要なパケットの送出を抑止することによりネットワークでトラフィックの増加による処理効率低下を防ぐと共に、自 PE のメモリに対するアクセスを高速化することが可能になり、全体の処理効率を向上することができると考えられる。

謝辞

本研究を遂行するにあたり御指導、御討論いただいた棟上情報アーキテクチャ部長ならびに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] 戸田賢二, 西田健次, 内堀義信, 島田俊夫. マクロデータフロー計算機 CODA - アーキテクチャ -. 並列処理シンポジウム JSPP'90 論文集, pp. 185-192, May 1990.
- [2] 戸田賢二, 内堀義信, 島田俊夫. 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 JSPP'89 論文集, pp. 79-84, February 1989.
- [3] 西田健次, 戸田賢二, 内堀義信, 島田俊夫. マクロデータフロー計算機 CODA - ハードウェア設計 -. 並列処理シンポジウム JSPP'90 論文集, pp. 193-200, May 1990.