計算機アーキテクチャ 113−21
（1995. 8. 24）

ロード先行実行機構によるデータプリフェッチ

中済 光昭, 堀口 進†, 岡本 秀輔, 曽和 将容

電気通信大学 大学院 情報システム学研究科
†北陸先端科学技術大学院大学 情報科学研究科

分散共有メモリを用いた並列計算機は, 既存のプログラムを大きく変更することなく, 並列化のメリットを得られることから, 注目されている. しかし, 並列計算機システムの大規模化と, 要素プロセッサの高速化に伴い, メモリアクセス遅延が性能に大きく影響するようになってきた.

これを解決する方法としてデータのプリフェッチを行なう命令を前もって発行する方法が提案されており, アクセス遅延の隠蔽に有効である. しかしプリフェッチ命令の実行に伴うオーバーヘッドが無視できない場合がある.

この問題を解決する1手法として 筆者らは並列計算機上の各プロセッサにロード命令を先取り実行するユニットを設け, 自動的にプリフェッチを行なう機構を提案したが, 当機構が最大の効果を発揮するためには, 分岐における当機構の動作が重要になる. 本稿では, 分岐命令フェッチ時の当機構の動作について検討するとともに高位論理合成システム PARTHNON 上に本方式を構築し分岐時の動作を検証する..

Data prefetch scheme by Hardware unit executed LOAD pre-execution

Mitsuaki NAKASUMI,Susumu HORIGUCHI†,Shusuke OKAMOTO,Masahiro SOWA

Graduate School of Information Systems
University of Electro-Communications
1-5-1 Chofugaoka,Chofu,Tokyo,182,JAPAN
E-mail:{nakasumi,okam,sowa}@sowa.is.uec.ac.jp

†Graduate school of Information Science
Japan Advanced Institute of Science and Technology
15 Asahidai,Tatsunokuchi,Ishikawa, 923-13,JAPAN
† E-mail:hori@jaist.ac.jp

Distributed shared memory systems are remarkable, Because these Parallel systems accept exist programs with minimum modify. But Distributed shared memory systems have a disadvantage as data latency brought by both the size of these systems and the speed gap between memory and processor.

To hide reference latency on a Distributed shared memory system, Software Prefetcing has been widely used. but this method requires overhead to issue prefetch instructions. We have already proposed a new Hardware unit based on data prefetching scheme for these systems. The basic idea is to place the Hardware unit which is able to execute load instructions before executed by processor. In this paper, We argue that how the unit executes prefetching when it meets conditioned Branch. because it's important to make full use of its ability. and We make a simulation of the Hardware unit on a CAD called PARTHENON.

1 はじめに

分散共有メモリによる並列計算機システムは,現存するプログラムを大きく変更することなく並列処理の効果を得られることから注目されている. しかし並列計算機システムの大規模化はリモートデータアクセスにおいてデータ転送時間を大きくし, データ待ちにより性能が大きく低下するという問題がある. システムの性能を最大限に発揮するにはメモリアクセスレイテンシの隠蔽が必須である. この問題に関して, プログラムによるプリフェッチ方式が提案されている [1]. この方法では,prefetch 命令自体のオーバーヘッド[1]により性能が落ちる事がある. 他方, ロード命令のプリロードがある [2]. ロード命令とロードしたデータを使用する命令の距離を離したコードを生成し, ノンブロックロード命令[2]を用い, ロードに時間がかかる場合, 並行して他の命令を実行することでメモリアクセスオーバーヘッドを隠蔽する方法である. この方法は効果的であるが, レジスタ割り当てやロード命令の配置の良否が問題となる.

以上の問題を解決するために, 筆者らはPUが実行する前にロード命令を先取り実行するハードウエアによってプリフェッチを行なう方式を提案した [5] が, 分岐命令を当機構がどう扱うかについて検討されていなかった. 特に条件分岐時プリフェッチをいかに行うかはプリフェッチの効果に大きく影響する. 本稿では, 分岐命令フェッチ時の動作について検討するとともに高位論理合成システム PARTHNON 上に本方式を構築し分岐時の動作を検証する.

2 前提

以下に当機構の前提を示す.

- 処理する命令:ロード, 無条件/条件分岐を処理する. 算術演算命令は, 処理せず PU に渡す. 詳細は表 1 の通りである.

- アドレッシング:アドレス指定は相対アドレス方式に依っており, データ参照においてはインデックスレジスタ修飾, 分岐命令においては後述のフェッチカウンタに命令語のオペランドを加えた値を指定アドレスとする.

- レジスタ:インデックスアドレスを格納するためにインデックスレジスタを持つ. インデックスレジスタは 2 つあり通常時と条件分岐命令フェッチ後にそれぞれ使われる.

3 本方式の概要

図 1に本方式の構成を示す. UP は要素プロセッサである.UP はネットワークを介して多くの UP と接続されている. PU は UP の命令処理部であり, IM は命令メモリ, DM はデータメモリである. また, IB は命令を前もってコピーしておくための命令バッファ, DB はデータをコピーしておくためのデータバッファである. PFU はメモリ内の命令とデータをそれぞれのバッファにコピーするプリフェッチユニットである. RMM はデータがリモート UP (自分以外の UP) にある時にデータを転送するためのユニットである. 図 2は PFU の詳細と分岐時の動作を示す図である. PFU はフェッチカウンタ (FC), 条件分岐フェッチカウンタ (CFC), バッファ(B) からなっている. FC は, IM から命令をフェッチする番地を表すカウンタであり, CFC,B は条件分岐の時使われる.

PU は, 基本的に IB から命令を, DB からデータを取り出し実行する. PFU は PU とは別にロード命令を前もって実行し, プログラムの要求するデータがリモート UP にあるときには, PFU は RMM を通してそのデータを要求し, 到着したデータを DB に入れる. 基本動作は以下のようである.

1) プログラム実行開始時,FC に命令開始アドレスをセットする.

2) PFU は FC が表す番地から IM の内容を IB に書き込み, FC の内容 (命令開始アド

[1] prefetch 命令実行およびこの命令のためのアドレス計算

[2] ロードの完了を待たず後続命令の実行が行なえるロード命令

レス) をPUに送る. 同時にIBからロード命令をPFUの作業域にコピーする.

3) PFUはコピーしたロード命令がローカルメモリに関するものならDMからデータをフェッチしてDBに格納し, リモートメモリに関するものならRMMにデータ送出要求を出してリモートのDMから返って来たデータをDBに格納する.

4) PUはIBから命令をフェッチしプログラムの実行を開始する (PFU,PUは並行に実行).

5) RMMは,PFUから要求が来るとその要求に対するデータをリモートUPに要求する. データが送られてきたらそれをDBに格納する. 以上の手順により,PUがリモートデータを必要とするときには, そのデータのほとんどがDBに格納されるようになるので, データアクセスの遅延がさけられる.

分岐がある場合を図2を用いて説明する. 図のαは, 通常時の動作を表し,βは, 条件分岐時の動作を表す. 動作の詳細は以下の通りである.

1) プログラム実行開始時,FCに命令開始アドレスをセットする.

2) PFUはFCが表す番地からIMの内容をIBに書き込む. 同時にIBを検索し命令が分岐命令ならば, PFUはそれをコピーする.

3) PFUはコピーした分岐命令の分岐先アドレスをもとめる.

4) 無条件分岐の場合,PFUはFCを分岐先アドレスに書き換え, 基本動作にもどる. 条件分岐の場合,PFUは分岐先アドレスをCFCに書き込み,FCとCFCのアドレスに基づき分岐, 無分岐先の命令フェッチを行なう.

5) PFUはPUからの命令フェッチの際出力されるアドレスによって分岐先が確定した時CFCの値を削除する. 以上により分岐する場合, 分岐しない場合の両方に存在するロード命令の先行実行を行なうことが可能となる.

次に割り込みについて検討する. 割り込みは, 分岐時期が不定である分岐として捉えられる. そこで分岐命令の扱いの中で割り込みを検討することにする. 以下では割り込みに対する当機構の対応について述べる. 本機構がサポートする割り込みとして以下のものがある.

- 内部割り込み
  - 命令メモリ保護違反:命令フェッチ時の不当なアドレスへのアクセス
  - TRAP命令:トラップ命令実行
  - データアドレスフォルト:リモートメモリへのアクセス
  - データメモリ保護違反:データアクセスの際の不当な領域へのアクセス
- 外部割り込み
  - リセット:リセットが押された時
  - I/Oリクエスト:I/O装置からのメッセージ受信時

上記以外の割り込みは,PUによって処理される.

割り込み処理は以下の方式で実現する. 当機構は図3のような状態遷移に基づいて動作している. 図においてStageは, 独立して動作するユニットであり命令フェッチ (if) と命令実行 (exec) が存在する. 各ステージはStateを持っており, クロックごとにStateを遷移させ処理を行なう. ここではState f1は通常時命令フェッチであり分岐命令を読み込むまではこのStateのみが起動される. State f2は分岐命令フェッチ後, 追加される動作である. State ext1は通常時命令実行であり分岐命令を読み込むまではこのStateのみが起動される. State ext2は分岐命令フェッチ後, 追加される動作である.

- イベント検知:内部割り込みのイベントについては, 各イベントごとにそれが発生するStageにおいて発生の有無を確認する. 割り込みイベントごとに発生するStageが決まっているので当該ステージでイベント発生の有無を調べる.

- 優先順位判定:プリフェッチ機構内の各ステージではイベント状態を所定の記憶域 (以下 ISS) に書き込むようになっており, あるステージで発生した内部割り込みは, その上流ステージで優先順位の高い割り込みが発生していないかを調べ, もし書き込まれていなければ当該割り込みイベントを ISS に書き込む. そしてその命令を無効化して状態更新を起こさせないようにする.

- 割り込み処理:毎クロックごとに, ステージの実行を終了した命令の ISS を調べ, その命令が内部割り込みイベントを起こしているか, あるいは外部割り込みイベントがペンディングしていれば, 次の割り込み処理を行なう.

  - フェッチ中の命令を無効化しフェッチを中断する.
  - フェッチカウンタと条件分岐フェッチカウンタをメモリ上の割り込み退避域に退避する.
  - 割り込みの緊急度が低ければ, 現在実行中のロードが完了し,PU に当機構に残る全ての命令が送られるまで, 当機構の処理を続け, 緊急度が高ければ, 割り込み退避域にデータバッファの内容を退避し割り込み処理に移る.
  - 割り込み許可を禁止にする.
  - フェッチカウンタに割り込み処理ルーチンのアドレスを設定して, 割り込み処理のための命令フェッチを始める

- 割り込みからの復帰:リターン命令によって, メモリ上の割り込みアドレス退避域のアドレスをフェッチカウンタに設定し, 割り込み禁止を許可に戻す.

4 簡単な実行例

当機構は,NTT で開発された論理合成システム PARTHENON 上のハードウエア記述言語 SFL を用いて記述している. 現在のところ, 通常動作部分の設計を終え,SFL 記述上での動作検証を行なっている.

以下に示すプログラムを実行させ, 分岐命令フェッチ時の動作を検証する. アドレス fd,ff にはアドレスが格納されており, インデックスとして用いられる.

```
LDXI 0xfd    /* x <- op2 */
LDAX         /* a <- (x) */
CLC          /* c <- 0 */
LDXI 0xfe    /* x <- op2 */
ADCX         /* a <- a + (x) + c */
LDXI 0xff    /* x <- op2 */
STAX         /* (x) <- a */
SEC          /* c <- 1 */
BC   0x0b    /* if (c) pc <- op2 */
LDXI 0xfd    /* x <- op2 */
LDAX         /* a <- (x) */
```

図 4:サンプルプログラム

これを PARTHENON のシミュレータ seconds を用いてシュミレーションを行なった結果が以下である. 以下では clk はクロック,fc はフェッチカウンタ, cfc は条件分岐フェッチカウンタ,op1 は通常時当機構が処理する命令コード, op2 は条件分岐フェッチ後, 処理される命令コードである.

```
clk  fc cfc op1 op2
0    00  00 uu  uu
1    00  00 uu  uu
2    00  00 uu  uu
3    01  00 83  uu
4    02  00 83  uu
5    03  00 01  uu
6    03  00 01  uu
  (略)
20   0d  0b 8d  uu
21   0e  0b 01  uu
22   0e  0c 01  8d
23   0e  0c 01  8d
24   0f  0c 08  8d
25   0f  0d 08  0b
```

図 5:実行例

当機構に条件分岐命令に対する考慮がされていない場合は, 条件決定までプリフェッチをブロックすることが必要になるが, これではプリフェッチの効果を得られない. 図5によれば, 条件分岐フェッチ後2アドレスに対してフェッチが行なわれ, 条件決定を待たずに必要なデータ要求を行なえることがわかる.

5 まとめ

プロセッサがロード命令を実行する以前にそのロード命令を実行する機構を用い, ローカルメモリにないデータを前もってリモートメモリに要求しそのデータをプロセッサに供給する方式における分岐命令の扱いについて検討し, 論理合成システム PARTHNON 上で当機構を検証した.

本稿では, 概念の提示に重点をおいたが, 今後詳細なアーキテクチャについて検討し, 詳細な性能評価を行なう予定である.

参考文献

[1] T.C.Mewry,M.S.Lam "Design and Evaluation of a Compiler Algorithm for Prefetching", Proceeding of 5th ASPLOS(1991)

[2] W.Y.Chen,S.A.Mahlke,P.P.Chang,W.W.Hue "Data access microarchitectures for superscalar processors with compilar-assisted data prefetching ", Proceeding of Microcomputing 24(1991)

[3] Daniel Lenoski,James Laudon,Truman Joe,
David Nakahira,Luis Stevens,Anoop Gupta ,
John Hennessy, "The DASH Prototype:Logic Overhead and Performance", IEEE Trans. on Parallel and distributed systems,vol.4,No.1,pp.41-61(1993)

[4] 中濟, 堀口, "マルチプロセッサシステムにおけるロード/ストア方式", 電気関係学会北陸支部連合大会 E-15,pp266(1993)

[5] 中濟, "大規模分散共有メモリシステムにおけるデータプリフェッチに関する研究", 北陸先端科学技術大学院大学修士論文(1994)

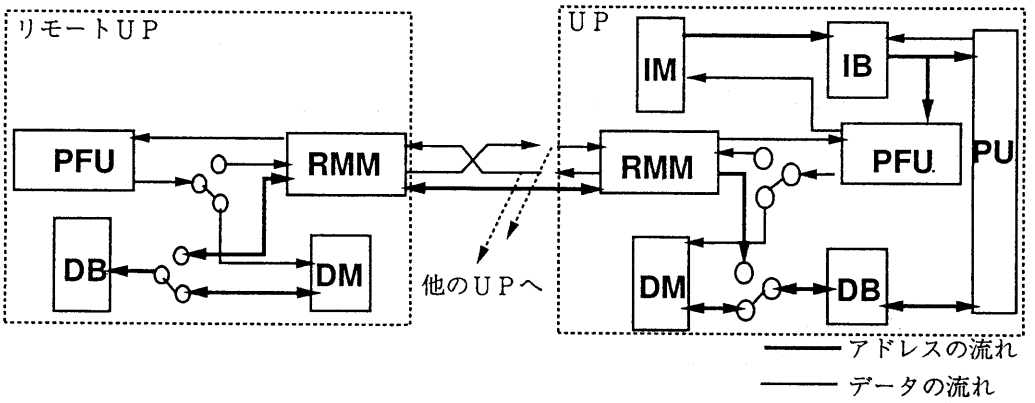

図 1: プリフェッチユニット

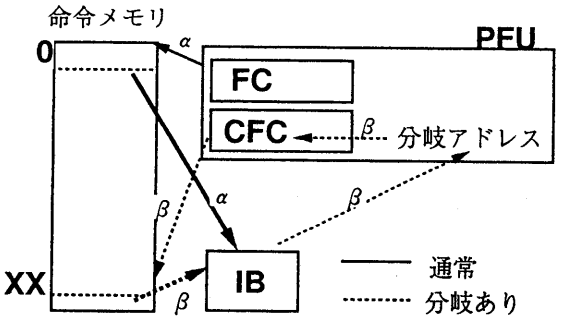

図 2: 分岐時の動作

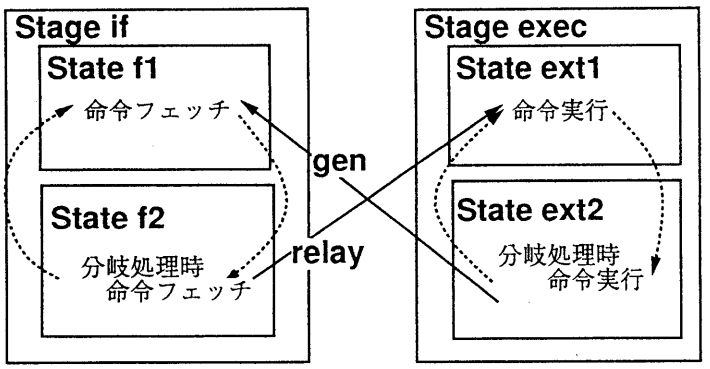

図 3: 状態遷移図

No.	命令	ニーモニック	長	形式	命令コード	機能
1	Increment X	INX	1	RR	00000001	X=X+1,PC=PC+1
2	Load A	LDAX	1	RX	00001000	A=(X),PC=PC+1
3	Load Immediate A	LDAI #op2	2	RI	00010000	A=op2,PC=PC+2
4	Load Immediate X	LDXI #op2	2	RI	00010001	X=op2,PC=PC+2
5	Load X	LDXM #op2	2	XM	00011000	X=(op2),PC=PC+2
6	Branch on Carry	BC #op2	2	BR	00011100	if C==1 then PC=op2 elsePC=PC+2
7	Branch on Zero	BZ #op2	2	BR	00011101	if Z==1 then PC=op2 elsePC=PC+2
8	Branch	B #op2	2	BR	00011110	PC=op2

表 1: 当機構が処理する命令