計算機アーキテクチャ 125－17
（1997. 8. 21）

UPCHMSにおけるパイプライン処理の適用

牧 晋広† 石田 朗‡ 岡本 秀輔† 曽和 将容†
電気信大学† 信州大学‡

処理能力が大幅に低減するキャッシュミスを避けるため，キャッシュメモリとは異なる新しい階層メモリシステムUPCHMSを提案している．このUPCHMSはメインメモリ同様に線形的なアドレスを持つ高速なメモリHMに，プログラムによりデータを1ワード単位で転送することで，HMをキャッシュメモリよりも効率的に利用することが可能になる．またこのことによりキャッシュメモリよりも高速にデータを供給することが可能になることが期待される．
今までの報告における評価ではパイプライン処理を考慮していなかった．そこで本稿ではパイプライン処理の導入について述べる．

Applying pipelining to UPCHMS

Nobuhiro Maki† Akira Ishida‡ Shusuke Okamoto† Masahiro Sowa†
University of Electro-communication† Sinshu University‡

To reduce the cache miss penalty, which becomes heavy bottleneck for processor, we have proposed new hierarchical memory system, we call UPCHMS. UPCHMS enables memory to supply the data to processor faster than cache memory system does. The reason why UPCHMS can do this shows follow. In UPCHMS, *data transfer program* controls the data on the HM, which corresoponds to cache memory and has no main memory address but has its own linear address. It is possible for program-control to use the data on HM more efficiently.
In this paper, we describe the UPCHMS with pipelineing.

1 はじめに

近年プロセッサの処理能力が飛躍的に向上し，それに伴い高速大容量なメモリが必要不可欠になっている．これを実現する手段として現在は，キャッシュメモリを用いる．キャッシュメモリは，頻繁に利用するメモリ領域をキャッシングし，そのメモリ領域に関するアクセスを高速化することが可能となる．しかし，キャッシュメモリはキャッシュミスが発生する．キャッシュミスが発生すると，プロセッサの処理能力は大幅に低下する [1]

このキャッシュメモリの発生を削減する目的でキャッシュプリフェッチが行われている．キャッシュプリフェッチ方式は，ハードウェアプリフェッチ方式とソフトウェアプリフェッチ方式に大別される [11, 12, 3, 5, 4, 6]．ハードウェアプリフェッチ方式は，キャッシュミス時に必要とするブロックだけを転送するのではなく，その転送されるブロックに隣接したブロックをも転送する方式である．この方式は，コードを何ら変更することなく実装することができるので，実装コストが低いという特徴を持つ．しかし必要とされるデータに隣接したデータブロックを同時に転送するため，命令参照のような逐次的なデータ参照では有効であるが，ランダムなデータ参照が多いアプリケーションプログラムでは，逆に不必要なデータをキャッシュメモリに転送することがあり，本来発生しないキャッシュミスが発生し実行性能を下げる可能性がある [4, 11]．

ソフトウェアプリフェッチ方式は，キャッシュメモリをソフトウェアにより制御する方式である．この方式は，キャッシュミスが発生する可能性の高いロード命令が実行される前に，プリフェッチ命令によりキャッシュメモリに先行してデータを転送する方式である．この方式では，実行前にプログラムのデータ参照動作を解析することで，キャッシュミスが起る可能性があるロード命令を検出し，そのロード命令が実行される前にプリフェッチができているようにプリフェッチ命令をプログラムに挿入する．この方式では，ハードウェアプリフェッチとは異なり，将来必要とされるブロックをプリフェッチできるので，キャッシュミスをハードウェアプリフェッチ以上に削減できる可能性がある．しかし，ブロックを単位として転送を行うので，プロセッサにより使用されないデータも転送されることがあり，ハードウェアプリフェッチほどではないがキャッシュメモリを無駄に使用することがある．また，キャッシュ飽和時はプリフェッチされるブロックによりキャッシュメモリ上の必要なデータを追い出すこともある [6]．

ハードウェアプリフェッチやソフトウェアプリフェッチ方式を含めた従来のキャッシュメモリシステムでこ

のキャッシュミスが発生する原因の1つは，転送するデータをキャッシュメモリのどこに置くかをLRU法などの固定的なアルゴリズムにより行うことにあると考え，これを解決するために著者らは新しいキャッシュメモリシステム「明示化キャッシュ」を提案している [2].

この明示化キャッシュ方式では，ブロックを転送の単位として，ソフトウェアによりキャッシュメモリにデータを先行して転送することはソフトウェアプリフェッチと同様であるが，プリフェッチ命令による転送位置 (アドレス) を明示的に指定できることが大きく異なる．これはキャッシュメモリの内容がプログラマに見えることを意味するので，将来利用することがわかっているキャッシュメモリ上のデータの置換を極力避けることが可能である．しかしこの方法でもデータ転送がブロックを単位として行われるので，無駄なデータを転送することがあり，きめ細かなキャッシュメモリ上のデータの制御ができず，キャッシュメモリの使用効率が悪くなる場合がある．

著者らはキャッシュという概念をなくした新しい階層メモリシステム『ユーザプログラム制御階層メモリシステム：UPCHMS』を提案している．UPCHMSはキャッシュメモリに相当するメモリに主メモリ同様の線形なアドレスを与え，そのメモリにデータをプログラムにより1ワード単位で転送する方式であり，プログラマにキャッシュレベルメモリの内容をみせるようにした階層メモリシステムである [7]．このUPCHMSには2つの特徴がある．

- プログラムにより1ワード単位でデータ転送をする
- データを実行プログラムと並列に転送する

である．これらの特徴から以下のような効果が期待される．

- プログラムにより1ワード単位でデータを転送するため必要なデータだけを転送可能，キャッシュレベルメモリ上のデータで将来必要とされるデータをできる限り維持するように制御ができる，これらよりキャッシュレベルメモリを有効に利用することができる．
- データ転送を並列に行うためデータ転送時間を隠蔽することができる．
- プログラムにより並列にデータ転送をすることができるため実行処理で必要なデータを必要とするときまでに用意することができる．

過去の報告では，UPCHMSの評価は各ユニットの命令の実行にパイプライン処理を考慮しない前提で行っていた．しかし現状の計算機システムは，命令の実行にパイプラインを行うことは一般的になっている．そこで現状の計算機に合わせるため，UPCHMSにパイプライン処理の導入する．本稿では，UPCHMSの構成動作，パイプライン処理の構成および動作を示す．

2 UPCHMS

2.1 構成

図1は，UPCHMSのブロック図である．PUは，一般的な演算処理を行うプロセッサユニット，IMはプログラムメモリ，DMはデータメモリである．DMは，超高速メモリVHM(レジスタに相当)，高速メモリHM(キャッシュメモリに相当)，主メモリMMの3階層により構成される．HMは，MM同様独立した線形アドレスを持つ．

HU，MUは階層メモリ間のデータ転送処理を専門に行うプロセッサユニットで，HUはVHMとHM間のデータ転送，MUはHMとMM間のデータ転送を行う．HU，MUは，それぞれのIMに格納された専用のメモリ操作プログラムを実行する．MUのみデータ一時保存用レジスタmrを持つ．tcは，トークンカウンタとよばれるカウンタで，ユニット間でトークンを送受することにより命令実行の同期を取るためのものである [8]．トークンとは1ビットの信号である．cfq，CF lineは条件分岐のための機構である．CF lineは，PUで決められた分岐の是非に関する情報をHU，MUに伝える通信線，cfqはその情報を格納するキューである．分岐情報は2ビット（分岐情報の有無と分岐の是非を表す）からなる．

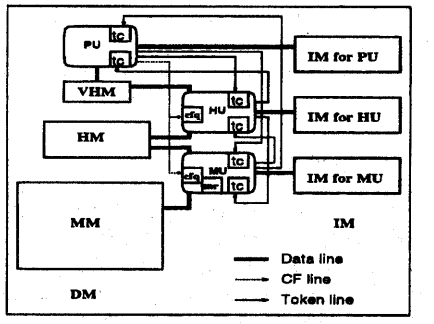

図 1: UPCHMSのブロック図

2.2 プログラムの動作

図2はMMのm0番地からm9番地に，それぞれデータa(1)からa(10)が格納されているとき，$\sum_{i=2}^{10} a(i)+a(i-1)$ の計算を行うプログラムである．ここでIPU，IHU，IMUはそれぞれPU，HU，MU用の命令流であり，IPUは主に計算処理，IHUはVHM-HM間のデータ転送処理，IMUはHM-MM間のデータ転送処理を行う．各命令間に付けられたアークは実行の先行関係を表し，例えば，命令M1とH1の関係では，M1命令終了後にH1命令が実行可能となることを意味している．ここで命令M1の

ldh m0,h0 は，MM の m0 番地のデータを HM の h0 番地に転送するロード命令，命令 H1 の ldv h0,v4 は HM の h0 番地のデータを VHM の v4 番地に転送する命令，命令 P8 の add v2,v4,v2 は VHM の v2 番地と v4 番地の内容を加算してその結果を v2 番地に格納する命令である．

図 2で，M1，H1，P8命令のプログラム動作を説明する．この動作では，MM の m0 番地のデータ a(1) は M1 命令によって HM の h0 番地に転送され，転送されたデータは，H1 命令によって VHM の v4 番地に転送され P8 命令により処理される．P7 命令以前は M1，H1 命令で転送されるデータを利用しないので，IPU の P1 から P7 と IHU，IMU の M1，H1 命令とは並列に実行される．

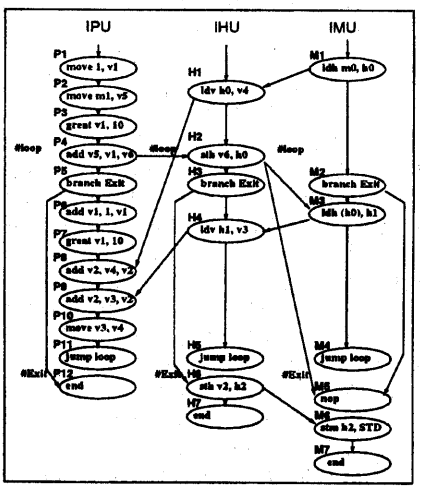

図 2: UPCHMS のプログラム

2.3 同期

命令流間の同期を示す．図 2の M3 と H4 の同期を例として述べる．今 M3 命令の実行が修了したとすると，M3 を実行した MU から HU に対してトークンが送られる．トークンが HU に到着すると HU のトークンカウンタ tc がインクリメントされる．トークンの送信は，図 1 の MU から HU ののトークン通信線（Token line）を通して行われ，その通信線に接続された tc がインクリメントされる．トークンを受け取った HU は tc をディクリメントした後 H4 命令を実行する．HU が H4 命令を実行しようとしたとき，もしトークンが未到着（すなわち，トークンカウンタが 0）ならばトークンの到着まで H4 の実行を延期する．tc の値は送信と命令実行によりインクリメントとディクリメントを繰り返すので，通常は小さな値である．tc による同期はカウンタのインクリメントとディクリメントで行われるので高速である [8].

2.4 分岐

UPCHMS には無条件分岐と条件分岐がある．無条件分岐は，各ユニットがそれぞれその命令を持ち，ユニット毎でその命令が実行された時点で分岐が行われる．図 2では，P11,H5,M4 が無条件分岐命令である．そのためユニット毎で分岐が実行される時間帯が異なる．

条件分岐は，分岐の是非を決定する分岐先決定命令とその決定に従い実際に分岐する分岐命令に別けられる．図 2では条件分岐は次のように行われる．各命令流が分岐命令 (IPU では P5, IHU では H3, IMU では M2) に達したとき，分岐情報がすでに到着していればそれにしたがって分岐命令を実行する．もし，分岐情報が未到着ならば到着するまで待つ．分岐情報は PU の分岐先決定命令によって作られ CF line を通して HU，MU の cfq に送られる (IPU の P7). 分岐情報がそれぞれの分岐命令の実行に先だって送られていれば，分岐によるオーバヘッドは最小になる．このとき分岐は分岐命令に達した命令流から行われるので，命令流ごとに独立して分岐する．図 2 のプログラムでは，分岐情報を決定するための演算 P6 の実行後次の P7 命令で分岐情報を IHU，IMU に放送する．

2.5 ユニット毎のパイプライン構成

UPCHMS のユニットは，ユニットによりアクセスするメモリが異なるので，ステージ構成はユニット毎で異なる．次にユニット毎のステージ構成について示す．ただし今回は，HM を 2 ポートとし，MM は 1 ポートとした場合の構成を示している．VHM ホワーディング (バイパス) 処理を考慮していない．

- PU

PU は次の 4 ステージの命令パイプラインである．
1. IF：命令フェッチ
2. ID：命令デコード，tc チェック，分岐チェック，およびデータフェッチ (ステージ後半)
3. EX：実行，トークン送出
4. WB：VHM 書き込み (ステージ前半)

tc チェックは，HU と同期する命令をデコードする場合トークンカウンタの値をチェックすることを意味し，tc の値が 0 の場合 1 以上になるまでストールする．トークン送出は，HU に同期させる命令が実行されるとトークンを HU に送ることを意味する．

ID ステージで，トークン待ちを含めたパイプラインハザードの是非を判定し，必要があればストールする．また，データフェッチはステージ後半で行い，WB では VHM 書き込みをステージ前半で行う．そのため同一クロックにおける ID ステージと WB

ステージの依存関係の無いVHM上のデータアクセスはストールしない．依存関係がある隣接命令間ではIDステージで1ステージ分ストールする．VHMのアクセスをステージ前半後半と分けられる理由は，VHMのアクセスはステージを実行する時間(クロックサイクル)に比べて短いためである．

- HU

HUは次のような5ステージの命令パイプラインである．このHUではトークンの送出ステージが命令の種類送出先(PU、MU)により変化する．
1. IF ：命令フェッチ
2. ID ：命令デコード，tcチェック，分岐チェック，データフェッチ(ステージ後半)，およびトークン送出(MU)
3. AD：加算，トークン送出(ST:PU)
4. HM ：HMアクセス，トークン送出(LD:PU)
5. WB ：VHM書き込み(ステージ前半)

HMは，HMへのアクセスであり，このステージのみHMにアクセスする．ストア命令ではWBステージで何も行わない．ADはインデックス計算を行うためのステージである．IDステージのトークン送出(MU)は，MUへのトークン送出はこのステージで行われることを意味する．ADステージの(ST:PU)は，ストア命令でPUと同期をとる必要がある場合トークンをPUに送ることを意味する．HMステージの(LD:PU)は，上と同様にロード命令でPUと同期をとる必要がある場合トークンをPUに送ることを意味する．

IDステージで，トークンによる同期待ちを含めたパイプラインハザードの是非を判定し，必要があればストールする．それ以外のステージではストールは起こらない．

- MU

MUは命令の種類によりステージ数が5，9と異なる．これは，ロード，ストア命令を2命令に分割するためである．これによりデータ転送は，MMとMU内部レジスタ間通信とMU内部レジスタとHM間通信の2行程により行われる．2命令に分割した理由は1命令におけるHMへのアクセス回数を減らすためである．

・構成1
1. IF ：命令フェッチ
2. ID：命令デコード，tcチェック，分岐チェック，mrデータフェッチ(ステージ後半)，およびトークン送出
3. AD：加算
4. HM ：HMアクセス
5. WB ：VHM書き込み(ステージ前半)

・構成2
1. IF ：命令フェッチ
2. ID：命令デコード，tcチェック，分岐チェック，mrデータフェッチ(ステージ後半)，およびトークン送出
3. AD：加算
4. MM1：MMアクセス
5. MM2：MMアクセス
6. MM3：MMアクセス
7. MM4：MMアクセス
8. MM5：MMアクセス
9. WB ：VHM書き込み(ステージ前半)

MM 1～5は，MMアクセスを意味し，5ステージ使ってMMをアクセスする．

ID, MM 1ステージで，パイプラインハザードの是非を判定し，必要があればストールする．それ以外のステージではストールは起こらない．IDステージのチェック項目は，他ユニットのそれと同じである．MM 1ステージでは，MMのアクセスが自命令だけであるかをチェックする．もし他命令がMMをアクセスしている場合は，そのアクセスが終るまでストールする．

2.6 ユニットの連係動作

上記で示したようにユニットによりトークン送出ステージが異なる．特にHUでは，命令の種類，トークン送出先(PU，MU)により送出ステージが異なる．これは，PU, HU, MUの各ユニットで効率的にデータを受渡しを可能にするためである．このことを示すため図3を用いる．図は，MM上のデータをHM経由でVHMに転送し，PUで演算を行った後，ふたたびMMに書き戻す過程を示している．左から順にIPU，IHU，IMUを示している．縦方向の四角列は1命令を，1つの四角は命令のそれぞれのステージを表している．また命令の先頭に，その命令の動作内容を示している．実線矢印はトークンの流れを，破線矢印はデータの流れを示す．命令M 1のldmrはMMからMU内部レジスタ(mr)にデータを転送する，M 2のldhはmrからHMにデータを転送する，H 1のldvはHMからVHMにデータを転送する，H 2のsthはVHMからHMにデータを転送する，P 1はVHMに転送されたデータを演算する，ことを意味している．

図を見るとMUのM 1命令でMMからmrにデータを転送し，M 2命令で(M 1命令でmrに転送されるデータを必要とするため) 6クロックストールした後mrからHMにデータを転送する．このときM 2のIDステージでトークンが送られる(実線矢印1)．M 1命令はトークン待ちで7クロックストールした後，M 2命令からトークン(矢印1)を受とり処理を再開する．ここでM 2命令のIDステージでトークンが送られているので，M 2によりHMに転送されたデータは，次のクロックですぐに実行を再開したH 1命令のHMステージでフェッチされる(破線矢印1')．同様にH 1とP 1の関係では，H 1のHMステージでトークンが送られP 1のIDステージでトークンを受けるので，データはH 1のWB前半でVHMに転送され同クロックのP 1のIDステージでフェッチされる(破線矢印2')．これらのように

データの受渡しが最適になるようにトークン送受信が行われる．

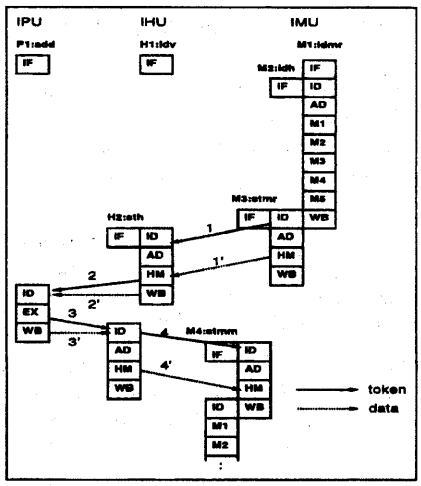

図 3: UPCHMS の各ユニットの命令処理過程

また，ユニット間のデータのやりとりを最適化するため，ユニット毎でトークンの送出時期を変化させているが，HU ではさらに命令とトークン送出先によりトークンの送出時期を変えている．命令により送出ステージを変えるのは，命令によりデータのやりとり時期が異なる HU にどののユニットに対しても最適な時期にデータをやりとりできるようににするためである．しかし命令だけでトークン送出時期を決定すると，トークンの送出の追い越しが起こる．つまり前の命令のトークン送出がその後の命令のトークン送出時期よりも遅くれ，トークンの送出時期が逆転する．これは例えば連続してた異なる命令 (ldv,sth など) が同じユニットにトークンを送る場合に発生する．この原因としては ldv は PU を対象に，sth は MU を対象に最適化しているため，対象とするユニットとは異なるユニットにトークンを起こる時矛盾が生じる．そこでこのことを避けるため，命令だけでなくどのユニットにトークンを送るかによってもトークン送出ステージを変化させる．このことを図 4に示す．図では IPU，IHU の実行過程を示している．矢印はトークンのやりとりを示している．いま sth のトークン送出は ID ステージ，ldv のトークン送出は HM ステージであるとすると H 2 のトークン送出 (破線) が H 1 よりも早くなってしまいトークン送出順序が逆転する．これを避けるため，sth のトークン送出は MU に送る場合は ID で，PU に送る場合は AD ステージで行う．これによりトークンの追い越しは避けられる．ただしこの場合トークンの送出が同一クロックで複数発生する可能性があるのでこれを考慮した設計にする必要がある．

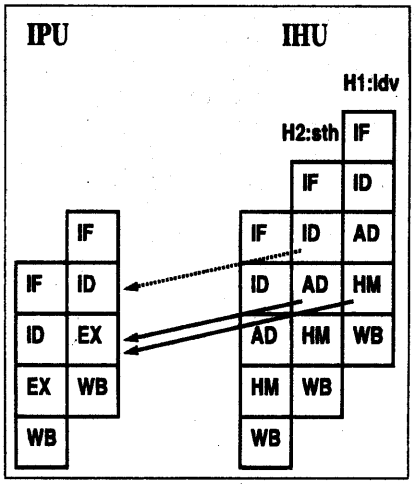

図 4: トークンの追い越し

2.7 ユニット毎の分岐処理

各ユニットの無分岐処理は，デコード時に行う．そのため分岐ハザードは 1 クロックストールですむ．(現時点ではディレイドブランチは考慮していない) 図 5に PU の無条件分岐の実行過程を示す．P 1命令の無条件分岐が ID ステージで実行されるため，P 2命令の命令は P 1のデコード時に IF をした命令をフラッシュし分岐先の命令 add を実行する．

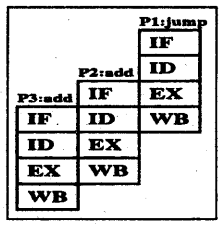

図 5: UPCHMS の無条件分岐の命令処理過程

条件分岐は分岐先決定命令により先行して分岐の是非を決定できるので，分岐命令を実行時は，そのID ステージで分岐を開始することが可能である．このことを図 6を用いて示す．図は分岐が成功する場合の IPU，IHU，IMU の条件分岐実行過程を示している．P 1命令は分岐先決定命令を，P 2，H 1,M 1命令は分岐命令を示している．図では P 1命令の EX ステージで分岐の是非が決定され，各ユニットに分岐フラグが送出される．PU では P 2の分岐命令の ID ステージで分岐が行われ，次の命令はフェッ

チした内容をフラッシュし分岐先の命令をフェッチしなおし処理を続ける．HUではIPUの分岐先決定命令の実行よりもHUの分岐命令が先に実行されるため，分岐フラグが届くまでストールする．MUではIPUの分岐先命令の実行よりもM 1の分岐命令が遅く実行されるため，実行された時には分岐フラグが到着していてストールすること無く分岐命令が実行される．また分岐命令の次の命令は分岐先が変更されるのでフェッチした内容をフラッシュし新しい分岐先の命令をフェッチする．

このように条件分岐を分岐先決定命令と分岐命令の2つに分けることで3つのユニットが並列に動作するUPCHMSの分岐による遅延を減らすことができる．

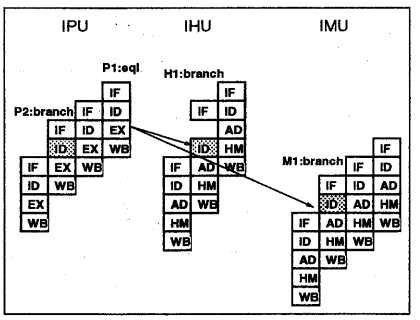

図 6: UPCHMSの条件分岐の命令処理過程

3 おわりに

本稿では，UPCHMSにパイプライン処理を適用することに関する諸事について述べた．UPCHMSでは各ユニットが最適な位置で同期をとるためデータの受渡しは無駄な待ちをすること無く効率良く行うことが可能である．これらの処理によりユニット間のデータのやりとりに必要とする時間を削減でき，効率の良いデータ転送ができることが期待される．

現在UPCHMSの性能を評価するため，シミュレータを製作している．この結果については後日論文として報告する予定である．

参考文献

[1] J. L. Hennessy, and D. A. Patterson, "Computer Architecutre:A Quantitiative Aproach.", Morgan Kaufmann, 1990

[2] 佐藤正樹, 有田隆也, 曽和将容, "並列処理によるキャッシュ操作の明示化," 並列シンポジウムJSPP,Vol.1,pp.1-7,1990

[3] D. Callahan , K.Kennedy , and A. Porterfield, "An Architecture for Sortware-Controlled Data Prefetching.", Proc.the 4 th International Conference on Architectural Suport for Programming Langages and Operationg Systems, pp.40–52, 1991

[4] T. Mowry, and A. Gupta, "Design and evalution of a compiler algorithm for prefetching.", Proc. of the 5th Intl.Conf. on Architectural Support for Programming Languages., pp.62–73, 1992

[5] T. Chen, and J. Bear, "A Performance Study of Software and Hardware Data Prefetching Schemes", Computer Architecture News Spring, vol.22, no.2, pp.223–232, 1994

[6] W. Y. Chen, R. A. Bringmann, and S. A. Mahlke, J. E. Sicolo, "An Effecient Architecture for loop Based Data Preloading.", Proc. of ACM MICRO-25, pp.92–101, 1992

[7] 牧 晋広, 岡本 秀輔, 曽和 将容, "ユーザプログラム制御階層メモリシステム", 情処学論, vol.37, no. 10, pp.1512–1526, 1996

[8] 高木浩光, 河村忠明, 有田隆也, 曽和将容, "問題の持つ先行関係だけを保証する高速な静的実行順序制御機構構", 並列情報処理シンポジウムJSPP, vol.1, pp.57–64, 1990

[9] A. R. Lebeck, and D. A. Wood, "Cache Profiling and the SPEC Benchmarks:A CASE Study.", COMPUTER , pp.15–26,num.10,vol.27,October,1994

[10] A. S. Tanenbaum, "Minix Operating Systems:Design and Implementation.", Prentice Hall, pp.255–261, 1987

[11] N. Drach, "Hardware Implementation Issues of Data Prefetching",International Conference on SuperComputing, pp.245–253, 1995

[12] F. Dahlgren, and M. Dubois, and P. Strenstrom, "Fixed and Adaptive Sequential Prefetching in Shared Memory Multiprocessors", International Conference on Parallel Processing, vol.I, pp.56–63, 1993