ハイパフォーマンス
コンピューティング 68－2
（1997. 10. 17）

UPCHMSのパイプライン処理適用による評価

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

著者らは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，プログラムにより1ワード単位で階層メモリ間を配置および転送制御するキャッシュメモリを持たない階層メモリシステムを提案している．このメモリシステムではタグメモリをもたないキャッシュメモリ相当のメモリ(すなわち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1ワード単位でデータ転送を行う．この操作によりHMをキャッシュメモリよりも更に効率的に利用することが可能になる．これによりキャッシュメモリよりも高速にデータをプロセッサに供給することが可能になることが期待される．
本稿では，命令をパイプライン実行する本提案システムの評価を示す．

Evaluations of Pipelined User Program Controlled Hierarchical Memory System

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 processors, we have proposed new hierarchical memory system, we call this UPCHMS. UPCHMS enables memory latency to be reduced more than cache memory system. 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 tag memory 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 principle and hardware organization of pipelined UPCHMS. And then, we evaluate the pipelined UPCHMS. It is reported that the pipelined UPCHMS can execute benchmark program 2 times faster than a computer with conventional cache.

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きている．しかし，プロセッサデータ消費サイクルに対しメモリのデータ供給サイクルが遅いため，これを回避する手段として現在では，キャッシュメモリを用いる．キャッシュメモリは，プロセッサにより頻繁に利用されるメインメモリ上のデータのサブセットをその中に保持することで，その領域に関するプロセッサからのアクセスを高速化することが可能となる．しかし，プロセッサがキャッシュ上にない部分をアクセスしようとするとき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].

現在，このキャッシュメモリの発生を削減する手法の1つにキャッシュプリフェッチがある．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 [5, 6, 7, 9, 10]．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 [5, 9].

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可能性がある [7].

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 [2, 3]. UPCHMSは，タグメモリを持たないキャッシュ相当のメモリ(HM) にプログラム制御により1ワード単位でデータ転送を行うメモリシステムである．このUPCHMSでは，キャッシュの特徴であるプログラマからその中身が見えないキャッシュメモリではなく，MM同様に何処にどのデータがあるかが自明である (タグメモリを持たない)HMを用いることで，プログラマの意志が反映されたより効率的なデータ配置可能なメモリシステムが実現できる．

過去の報告では，UPCHMSの評価でプロセッサの命令の実行がパイプライン処理をしていないために，命令のパイプライン処理が一般的である既存のシステムと異なる評価になっていた．そこで本稿では，現行の計算機システムの構成に近づける目的で命令をパイプライン実行するUPCHMSを提案する．まずは，パイプライン化したUPCHMSの構成，動作，HMへのデータ配置手法の1例，UPCHMSの各ユニットのパイプラインステージ構成を示し，その後に既存のキャッシュメモリを用いた計算機と比較評価を行う．

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，キャッシュメモリとは異なりタグメモリを持たない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間のデータ転送および単純なアドレス計算を行う．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．

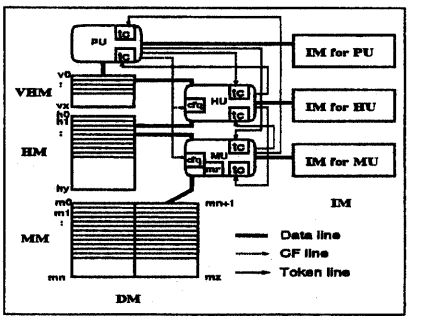

図 1: UPCHMSのブロック図

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，$\sum_{i=1}^{9} a(i)$ 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間のデータ転送と簡単なアドレス計算処理を行う．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M5とH2の関係では，M5命令終了後にH2命令が実行可能となることを意味している．ここで命令M2のldmra mr1,m0(mr7)はMMのm0+mr7番地にあるデータをMUの内部レジスタmr1番地に転送するロード命令，M5のldh h1,mr1はMUの内部レジスタmr1のデータをHMのh1番地に転送するロード命令，命令H1のldv v2,h1はHMのh1番地のデータをVHMのv2番地に転送する命令，命令P6のadd v3,v2,v3はVHMのv2番地とv3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，M2，M5，H2，P6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M2命令により内部レジスタmr1に格納され，mr1に転送されたデータはM5命令によりHMのh1番地に転送される．転送されたデータはH2命令によってVHMのv2番地に転送されP6命令によ

り処理される．P5 命令以前は M5，H2 命令で転送されるデータを利用しないので，IPU の P1 から P5 と IHU の H1,H2，IMU の M1 から M4 命令とは並列に実行される．

このプログラムでは，プログラム動作が単純で再利用性がないためタグメモリを持たない HM をプログラムコントロールする効果を明確には示せないが，HM の特定の番地にデータを転送できるため，HM 上で将来的に利用されないもしくは空いている領域に MM のデータを優先的に転送する，また将来的に利用しないデータを優先的に MM に書き戻す等の処理が可能になる．

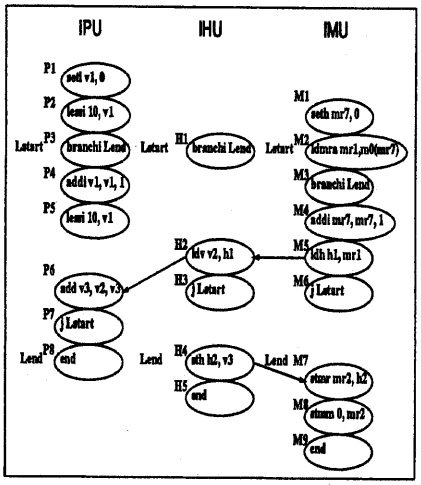

図 2: UPCHMS のプログラム

2.3 アプリケーションプログラムが扱うデータの HM への配置手法例

UPCHMS のプログラムでは，いかに効率良く HM 上のどの位置にどのデータを配置するかがデータ供給処理能力を左右する重要な鍵となる．ここでは，ループ処理でデータの再利用性があるプログラム (LFK の kernel1) の HM へデータを配置する手法の一例を示す．この LFK, kernel1 のソースプログラムを下に示す．

このプログラムでは，データ列 ZX(k), Y(k) を逐次的に参照する処理を Loop 回繰り返す．この場合 HM 上で再利用性を高めるように ZX(k), Y(k) を配置するには，次のように ZX(k), Y(k) を配置する．ただしここでは，ZX(k) と Y(k) のデータの個数の和 (2n+1) が HM の容量の 1 倍以上で 2 倍を超えていないものと仮定する．この場合 UPCHMS のプログラムでは最内ループを 3 パートに分ける．

1) 初期パート，2) 上書きパート，3) 再利用パートである．

この 3 つのパートの処理の流れを図 3を用いて説明する．

1 Initial Area のように (ユニット間のデータ通信領域や X(k) の書き込み領域 othr area を除いた領域)HM の領域 ZX(k) と Y(k) を HM に配置する

2 上の 1 の処理だけでは全てのデータが配置できないとき，Overwrite Area のように HM に配置しきれない ZX(k) と Y(k) をそれぞれの領域の先頭部分を残すように上書きする

3 L ループ (外ループ) に移るとき 2 で上書きせずに残した部分を再利用する (再利用パート)

4 Overwrite Area に再利用できない部分を再度上書き転送する

5 2,3,4 のステップを繰り返し行う．

このようにプログラムの持つ再利用性を積極的に引き出すことで HM 上のデータを有効的な利用が可能になる．

```
-----------------------------------------
DO 1 L = 1,Loop
DO 1 k = 1,n
1 X(k)= Q + Y(k)*(R*ZX(k+10) + T*ZX(k+11))
-----------------------------------------
LFK kernel1
```

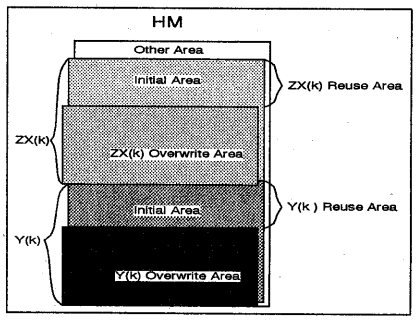

図 3: HM 上のデータの配置図

3 UPCHMS のパイプライン化

UPCHMS の各ユニットの命令をパイプライン実行させる．UPCHMS では，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．以下にユニット毎のステージ構成を示す．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．またVHM間のホワーディング(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と同期する命令のIDステージで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に同期され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上の重複データアクセスではストールしない．隣接命令間で依存関係がある場合は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)に比べて十分に短いためである．

- 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はマルチサイクル命令で，ステージ数が5，9の2種類がある．これは，多くのステージ数を必要とするMMにアクセスする命令(ロード・ストア)とそれ以外の命令のオーバラップを助長させるためである．またデータ転送命令は，その命令を2命令に分け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．これによりMMから内部レジスタへのロード中にHMから内部レジスタへのストア処理命令がストールすることなく処理が可能になる．

・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1
5. MM2：MMアクセス2
6. MM3：MMアクセス3
7. MM4：MMアクセス4
8. MM5：MMアクセス5
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．

4 性能評価

パイプライン化UPCHMSの性能評価を行う．評価では，性能向上の基準としてキャッシュメモリを持つDLX(C-computer)と比較を行う．

4.1 評価モデル

評価に用いるプロセッサモデルを以下に示す．

- U-computer：パイプライン化したUPCHMSを持つ計算機システムを指す.
- C-computer：キャッシュ付きDLXを指す[1].

今回の実装では，U-computer，C-computerの両者はバイパス処理、ディレイドブランチ等の機能を持っていない．次にU-computer, C-computerのメモリモデルを示す．各階層のメモリの容量(size)とメモリアクセス時間(clock)を表1のようにする．メモリ容量，メモリアクセス時間の単位はそれぞれワード(wd)，ナノ秒(ns)である．メモリはVHM，HM，MMとも1ワード32bitとする．メモリサイズは比較的小規模に設定した．またC-computerでは，ブロック転送を行うことを考慮して，キャッシュミス時の1ワードあたりのデータ転送速度はUPCHMSの1ワード転送の速度の2倍とし，それにより得られるキャッシュミスペナルティは10クロック(5clock/word*4word*0.5)である．

C-computerでは，HMと同容量のキャッシュメモリを持ち，LRU置換，write through，フルアソシエィティブキャッシュ，1ブロック4ワードとしている．

命令メモリは理想的なものとし，HMのアクセス時間で1命令をフェッチできるものとする．プログラムは，実行開始前にすべてのものがMMに格納されているとし，HMもそれらプログラムに静的に割り当てられるとした．

次に評価用ベンチマークプログラムを示す．ベンチマークプログラムとしてLivermoreloop Fortran Kernel:LFKを用いた．

表1: U-computer，C-computerのメモリの容量および各階層のアクセス時間

		VHM	HM	MM
U-computer	size (wd)	16	128	65535
C-computer	acs (clock)	0.25	1	5

4.2 評価結果

4.2.1 実行時間の比較

図4は，U-computerとC-computerの実行結果を示している．棒グラフは，ストール時間(Stall time)とそれ以外の処理時間(Calc time)の積み重ねで表されている．時間の単位はクロックである．

全体では，すべての評価プログラムでU-computerがC-computerの0.47から0.77倍の時間で実行している．Calc timeはU-computer，C-computerでほぼ同値であるのに対し，UPCHMSのStall timeはCMSに比べ減少している．このことより，UPCHMSの実行時間の短縮は命令ストール時間を短縮することにより得られると考えられる．

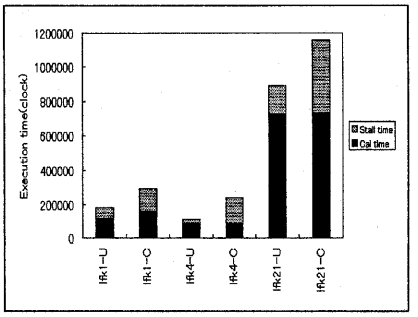

図4: U-computerとC-computerの実行時間とストール時間

4.2.2 HMの有効利用度

次にU-computerとC-computerがHMをどの程度有効に利用しているかを調査する．このHMの有効利用を調べるためHMの利用率を定義する．HMの利用率はMUによりMMからHMに転送されるデータ数とHUがデータをHMからVHMに転送する回数の比を利用率と定義して用いる．この利用率が大きいことほど，HMを有効に利用していることを意味する．

表3はU-computerとC-computerの利用率を比較したものである．Arcはアーキテクチャで，U-computerおよびC-computerを示す．Ref.はHUがHM上のデータをVHMに転送する回数もしくはプロセッサがキャッシュメモリ上のデータを参照する回数を，Trans.はMMからHMもしくはキャッシュメモリに転送するデータ数を示す．また，Util.はHMのデータの利用率で，Ref.とTrans.の比(Ref./Trans.)で示される．

表3より全てのベンチマークプログラムでU-computerがC-computerよりも利用率が高いことがわかる．これは，UPCHMSの方がより有効にHM上のデータを利用していることを表している．これはプログラムによりHM上のデータをうまく制御できたためと考えられる．

lfk4ではC-computerではキャッシュミスが頻繁に発生する．そのため利用率が非常に低い．しかしU-computerでは1ワード単位でHMを細かく管理するためより良い利用率を示している．

lfk21ではU-computerとC-computerのどちらも利用率1を超えている．U-computerではもともと局所性があり利用率が高くなる傾向にあるプログラムでもC-computerを上回る利用率を示している．

表 2: 評価プログラムの名前，内容, 特徴

Program	Description	Characteristics
lfk1	ICCG excerpt	データ読み参照に再利用あり, 逐次にデータを参照, データの解析可
lfk4	banded linear equations	データの読み参照に再利用あり, キャッシュミス頻発 , データ解析可
lfk21	matrix*matrix	データ読み参照に再利用あり, 離散的なデータ参照, データ解析可

また，図 4と表 4.2.2で利用率とスピードアップの関係を考えて見ると，U-computer と C-computer の利用率の差が大きいもの程スピードアップしていることがわかる．

以上より U-computer では HM を有効に利用することで C-computer を超える実行性能を示すと考えられる．

表 3: U-computer と C-computer のデータの利用率

Prog	Arc	Ref.	Trans.	Util.
lfk1	U-computer	20201	16201	1.25
	C-computer	20201	28580	0.71
lfk4	U-compter	21110	10707	1.97
	C-computer	21110	48928	0.43
lfk21	U-computer	129600	65600	1.95
	C-computer	129600	88980	1.46

5 さいごに

プログラムによりタグメモリを持たない高速なメモリ:HM を 1 ワード単位で制御するユーザプログラム制御階層メモリシステム:UPCHMS の原理, 構成, 動作, パイプラインステージ構成を示した後, パイプライン化 UPCHMS の評価について述べた. 評価結果では，UPCHMS を持つ計算機システム (U-computer) がキャッシュメモリを持つ計算機 (C-computer) に対して最大 0.47 倍の実行時間で実行することを示した. また，HM 上のデータがどの程度利用されているかを調査する評価では，UPCHMS が 4 倍以上有効に利用していることが確認された.

本稿では，コンテキストスイッチングを考慮していないので，これらの実現方法は今後の課題である.

また UPCHMS では階層メモリ間のデータ管理がプログラマの責任になるので，プログラマの負担が大きくなる. そのため，自動的にメモリ操作プログラムを出力するコンパイラの開発も今後の課題である.

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] 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.1512–1516, 1996

[4] 牧晋広, 石田朗, 岡本秀輔, 曽和将容,"UPCHMS におけるパイプライン処理の適用", 情処研報, vol.97, no.76, ARC-125,pp97–102, 1997

[5] 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

[6] 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

[7] 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

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

[10] 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