4K－4

参照属性に基づくハードウエアプリフェッチ方式

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†

1 はじめに

共有メモリ型並列コンピュータにおいて, 共有メモリアクセスにおける遅延は, 性能低下の主要因である. 遅延を隠蔽し性能を向上するため, プログラム中に明示的に prefetch 命令を挿入し演算とデータアクセスをオーバーラップして遅延を隠蔽しようとする方法などが提案されている. この方法ではプリフェッチ命令に関連する計算のオーバーヘッドが問題となっている. 筆者らは並列コンピュータ上の各プロセッサが要求する以前にデータ参照を処理するユニット (以下 PFU) によるプリフェッチを提案したが [1], 本稿では, プログラム内に示されたデータ参照属性情報を PFU に送りプリフェッチを行なう方法を提案する.

2 データの参照属性

データの参照属性は, データの使われ方とデータ間の参照関係からなる. データの使われ方は, リードのみ/更新/同時書き込みに分けられる. これは, 変数ごとに定義されデータの複製を許すかどうかとデータ整合性プロトコルを切替えるための情報として使われる.

リードのみは, データの複製を必要に応じて作るモードであり複製データの整合性について考慮しなくてよいため, プリフェッチを容易に導入できる. 更新は, データの複製を必要に応じて作るモードかつ処理の直列性により, 複製データの整合性を保たなければならないためいつプリフェッチを行なうかの制御が必要である. 同時書き込みは, 更新可能だがロジック上同時に更新されないことがわかっているモードである. プロトコル上では整合性の緩いチェックを行なうがリードのみと同様に扱える.

データ間の参照関係は,1) 式/レコード (構造体)/配列における位置の局所性と 2) データの再利用性として時間軸に対する局所的参照関係と考えられ, 局所的に塊としてアクセスされる可能性の高いデータ群として示される.

1) は, 数値計算では配列への規則的参照が得られることが多く, 静的にアクセスパターンを作ることができることが多い. しかし文字列処理などではメモリ参照の規則性が少なく, アクセスパターンを静的に求めることは難しい. 例えばリスト構造によって作られるデータベースのテーブルを参照する場合, キー部のデータアドレスと検索内容を示すデータアドレスには参照の規則性を見出すことは困難である. これに関して, プログラムに示される情報からプリフェッチ戦略を作ることを考える. レコードは,C 言語の構造体で示される. 構造体を構成する各変数は, 構造体が互いに関連の高い変数を塊として管理するという性質上参照局所性が高い場合が多い. データベースの検索ではキー部が確定した時, キーに関連するデータ部をプリフェッチすれば参照される可能性が高い.

配列の参照について以下の２つのパターンを考える.

stride access: ループ内で配列を一定のストライドで読み出す.
random access: ループ内で配列を可変のストライドで読み出す.

stride access はプリフェッチ戦略が単純であり, ストライドと初期アドレス, 終了アドレスが設定されればプリフェッチが可能である. random access では, ストライドの性質によりプリフェッチの有効性が変化する. ストライドが計算式によって与えられる場合, 参照のタイミングの制御が可能であればプリフェッチは可能であるが, 検索などでユーザーがキーを入れるまでアクセス先が不明の場合キー確定までプリフェッチは困難である.

2) について,PFU はバッファを持ちこれにプリフェッチしたデータを一時保管するが, データを更新しストアする場合, 再利用の有無によりコピーを置かずに, もとのメモリ位置に書き込みコピーを消去する, またはコピーを保持し再利用するという選択ができれば, 性能を最適にできる.

A Hardware Prefetch Method by attributes of Data references
Mitsuaki NAKASUMI,Susumu HORIGUCHI†,Shusuke OKAMOTO and Masahiro SOWA
University of Electro-Communications,1-5-1 Chofugaoka Chofu-shi Tokyo 182,JAPAN
Japan Advanced Institute of Science and Technology†,15 Asahidai Tatsunokuchi-cho Nomigun Ishikawa 923-12,JAPAN

3 プログラミング言語の拡張

以上の参照属性をPFU命令として表しプリフェッチを行なう. プリフェッチはCPUのロード/ストア命令をCPUより前に実行することである. PFU命令はプログラム文中に挿入されCPU命令と変数定義以外のPFU命令の並び順は, CPUとPFUの同期制御を表す.PFU命令はコメント形式になっており, PFUをサポートするコンパイラのみが解釈する.PFU命令の定義を図1に示す.

PFU命令定義 (変数定義)

```
/*@PF-BEGIN(PARALLELWRITE)*/-> 同時書き込み可
/*@PF-END(PARALLELWRITE)*/  -> 同時書き込み可
/*@PF-BEGIN(READONLY)*/     -> リードのみ
/*@PF-END(READONLY)*/       -> リードのみ
/*@PF-BEGIN(UPDATE)*/       -> 更新
/*@PF-END(UPDATE)*/         -> 更新
```

PFU命令定義 (プリフェッチ制御)

```
/*@PF-LOCK(PFU)*/-> プリフェッチ不可
/*@PF-REL(PFU)*/  -> プリフェッチ可
```

PFU命令定義 (再利用制御)

```
/*@PF-LOCK(VAR{変数リスト})*/-> 再利用可
/*@PF-REL(VAR{変数リスト})*/-> 再利用不可
```

PFU命令定義 (位置の局所性によるプリフェッチ制御)

```
/*@PF-BULK(LOOP,BEGIN{式},STRIDE{式},END{式},DATA{変数リスト}) */
```

図1:PFU命令

PFU命令の変数定義は,'@PF-BEGIN()' と'@PF-END()' に囲まれた変数定義部の変数についてプリフェッチがいつでも可能か, ある処理実行後にプリフェッチ可能になるかをPFUに知らせる. プリフェッチ制御では'@PF-LOCK(PFU)' と'@PF-REL(PFU)' に囲まれたCPU命令がプリフェッチ不可であることを示す. 再利用制御では,'@PF-LOCK(VAR)' でPFUのバッファにあるデータの再利用を許し, '@PF-REL(VAR)' で禁止する. 位置の局所性によるプリフェッチ制御は, 以下のような命令を書き, プリフェッチを行う.

```
/*@PF-BULK(LOOP,BEGIN{i=0},STRIDE{1},END{i<98},DATA{a[99],b[99]})*/
 for(i = 0; i < 99; i++)sum += a[i]*b[i];
                    :
```

図2:プリフェッチ制御 (LOOP)

図2では初期値と終値, ストライドが確定しているので属性がリードのみなら任意の時点でプリフェッチが可能である. 構造体や式では,PFU命令の変数定義により, プリフェッチ戦略を作成する. 以上のプログラムは, 単一の実行形式ファイルに変換される. 実行形式ファイルには,PFU命令とCPU命令が混在する. 命令はPFUを通りCPUに供給されるが, その時PFU命令は取り除かれ,CPUへはCPU命令のみが送られる.

4 まとめ

プログラムの性質に合わせたデータプリフェッチを行うPFU命令を定義した. 今後はハードウエア自動合成システム パルテノン [2] でのPFUの記述を行い,PFUの評価を進める. また多岐にわたるアプリケーションの記述を通じてPFU命令の精錬を行う予定である.

参考文献

[1] 中済, 堀口, 岡本, 曽和," ロード先行実行機構によるデータプリフェッチ", 情報処理学会研究報告 95-ARC-113,pp.161-167,1995

[2]NTTデータ通信 (株),"PARTHENON USER'S MANUAL",NTTデータ通信 (株),1990

[3]A.S.Tanenbaum, "DISTRIBUTED OPERATIONG SYSTEMS",Prentice Hall,pp.345-353,1995