並列アーキテクチャ

小西将人

平成１０年5月8日

1　マルチプロセッサ概論

1.1　マルチプロセッサの分類

1.1.1　プロセッサアーキテクチャとしての分類

1. SPP
   (a) スーパスカラ方式
   (b) VLIW 方式
   (c) スーパパイプライン方式
   (d) マルチスレッド方式
2. VPP
   パイプライン方式のベクトルプロセッサ

1.1.2　メモリアーキテクチャとしての分類

1. 共有か非共有か

   共有メモリ方式　１つの物理アドレス空間をシステム全体で共有する

   非共有メモリ方式 個々のプロセッサでアドレス空間が閉じている

2. 集中管理か分散管理か

1.1.3　ネットワークアーキテクチャによる分類

1.2　プログラム分割とプロセッサ割り付け

1.2.1　粒度問題

分割と割り付けにおいて処理量と通信量のバランスを採る問題。

分割数が多いと（細粒度）並列度増大→処理時間減少、通信時間増大。

分割数が少ないと（粗粒度）通信時間減少、処理時間増大。

1.2.2　プログラム分割モデル

プログラムは命令系列とデータに別れているため次のような分割モデルがある。

メモリ共有方式 命令系列のみを分割する

共有メモリが集中/分散共有方式ならば、メモリアクセスにＯＳの介在がないので高速→細粒度向け。

※ 共有メモリが分散非共有方式で実現されている場合もあるが、普通は上の方式をさす。

メッセージ交換方式 命令系列、データともに分割。

通常、物理メモリが分散非共有方式であるもので用いる。通信にＯＳが介在し遅い→租粒度向け。

2　メモリ共有型マルチプロセッサ

メモリ共有型マルチプロセッサは、メモリが多数のプロセッサで共有されているものを指す。

プロセッサに内蔵されたキャッシュを生かして、データ参照の局所性を生かすこと、無駄な通信を極力削減することが要求される。

2.1　共有メモリの構成方式

- 集中共有メモリ
- 集中共有メモリ、ローカルメモリ付き
- 集中共有メモリ、キャッシュ付き
- 分散共有メモリ
- 分散共有メモリ、キャッシュ付き

共有メモリを次のように分類することもある。

ＵＭＡモデル 共有メモリが全プロセッサから等距離にあり、アクセス時間が同一。

ＮＵＭＡモデル 共有メモリが全プロセッサから等距離にない。
書き込み置いてきぼり制御が導入されることがある。
あるプロセッサで遠隔メモリへの書き込みがなされるとする。そのときストアバッファへ書き出し、時間をかけてメモリのに書き込む。プロセッサはメモリの変更を待たずに次の命令にすすむ。

2.2 メモリコンシステンシモデル

メモリがいつ正しい状態を示すか、どの時点で正しい状態にする必要があるのか、がメモリコンシステンシ問題である。

- ノンコンシステンシ
- 逐次コンシステンシ
  (1) 各プロセッサの命令はならべられた命令順に実行する。
  (2) プロセッサは先行命令でのメモリの状態変化が完了しない間に後続の命令の発行はできない。

  そのためすべての書込み操作でオーバーヘッドが生じる。

書き込み完了待ちをまいかいする必要のない場合がある。臨界領域へのアクセスではその前後を同期命令 (Acquire と Release) で挟むことにより達成できる。
これを達成するモデルに次のようなものがある。

ウィークコンシステンシ 同期命令は、その前の先行メモリ操作完了後に発行。

同期命令後の、メモリ操作は同期命令完了後に発行。

リリースコンシステンシ ウィークモデルの制約を更に弱めたもの。臨界領域前後のメモリ操作が臨界領域に無関係の場合に適用できる。

Acquire は先行するメモリ操作の完了を待つ必要がない。また Release の完了を待たずに後続メモリ操作を発行できる。

3 キャッシュコヒーレンス

共有メモリと各プロセッサのキャッシュの内容をどう一致させるか、がキャッシュコヒーレンス問題である。

3.1 キャッシュコヒーレンスとメモリコンシステンシの関係

あるプロセッサでキャッシュへの書き込みがあったとき、他のプロセッサのキャッシュに存在する同一ブロックをどの時点で変更しなければならないか、はメモリコンシステンシの方式で決まる。

ノンコンシステンシの場合 極端な場合変更しなくてよい。

逐次コンシステンシの場合 他のプロセッサのキャッシュの変更が完了してから次の命令に移る。

ウィークコンシステンシの場合 同期命令の発行前にキャッシュの変更を完了する。

リリースコンシステンシの場合 Release 同期命令発行前にキャッシュの変更を確認する。

キャッシュコヒーレンスの方式には、次のものがある。

1. ハードウェアによるもの
   (a) スヌープキャッシュ方式
   (b) ディレクトリ方式
2. ソフトウェアによるもの

3.2 スヌープキャッシュ法

3.2.1 単純な構成

プロセッサごとにスヌープコントローラーがあり、書き込みアドレスを監視している。
あるプロセッサ Ｐ１ で書き込みがあると、

ストアスルーの場合 メモリにも書き込み、他のプロセッサの同一ブロックを無効化、もしくは更新する。

ストアインの場合 他のプロセスの同一ブロックを無効化、もしくは更新する。

メモリに最新データがないので、Ｐ１にのみ存在するブロックへアクセスするときはＰ１のキャッシュのデータをキャッシュ間で転送する。このときメモリに書き戻すかしないか２つの方式がある。

どの方式においてもトラフィックが大きく高速な処理が困難である。

3.2.2 各ブロックにタグを持たせる方式

書き込みが多数回なされるブロックにはＤｉｒｔｙタグを付け、システム内に１つしか存在しないようにする。

ストアインの場合には４つのプロトコルがある。

バークレイプロトコル 無効化、無書き戻し

イリノイプロトコル 無効化、書き戻し

ドラゴンプロトコル 更新、無書き戻し

ファイアフライプロトコル 更新、書き戻し

スヌープ方式ではプロセッサ台数が多くなると、バスが飽和し性能低下が著しいという欠点がある。

3.3 ディレクトリ方式

メモリ内の各ブロックに、そのブロックがどのプロセッサにキャッシュされているかを示すディレクトリがある。ウィーク/リリースコンシステンシが使われる。

つぎのような方式が提案されている。プロセッサ台数を N とする。

- フルマップ方式 :メモリの各ブロックごとに N ビットのディレクトリ。
- リミテッド方式 :各ディレクトリに log N ビットのフィールドが複数存在しプロセッサへのポインタとなる。
- LimitLESS 方式 :リミテッド方式の変更を加えたもの。ポインタプールを用意しディレクトリに格納しきれないポインタを格納する。
- チェイン方式 :メモリのディレクトリに log N ビットのフィールドが 1 つ。キャッシュ内のディレクトリにポインタで連結される。

フルマップ方式はディレクトリ容量が大きいため小規模システムに使われる。

他の方式は大規模システムに使用され拡張性も確保できる。

ポインタ数、チェイン数は無効化/更新プロトコル、ブロックサイズに依存する。

更新方式ではブロックを共有するキャッシュが増大していくので無効化方式よりもポインタ数、チェイン数ガ増える。

またブロックサイズを大きくするとディレクトリ容量が減らせるが、偽共有問題が生ずる。

上に述べた方式のほかに、疑似フルマップ方式があり、ＪＵＭＰ－１で使われている。

3.4 ソフトウェアによる方式

定型的な処理でコンパイル段階でプロセッサ間のデータの参照関係がわかるばあいはソフトウェアの方式が有利になる場合がある。

ソフトウェアによる方式には次のことが必要となる。

- ストアスルー方式
  ストアイン方式ではハードウェアが複雑になる。
- 同期命令
- キャッシュ内容の選択的無効化

ソフトウェアによる方式をならべると次のようになる。

- キャッシュブロック無効化命令の装備
- マーキング法
  各ブロックにソフトウェアで操作できるタグをつける。共有しないブロック、読み出し専用ブロックはタグを 1 としてキャッシュ利用

を可とする。共有する可能性のあるのもはタグを 0 としてキャッシュ利用を不可とする。

3.5 キャッシュを利用した通信量の削減

データの属性ごとにプロトコルを切り替えることで、通信量を減らすことができる。

たとえば、データがあるプロセッサでいったん書き込まれると長時間そのプロセッサのみで処理されるようなデータは、無効化プロトコルがよい。

逆に、処理するプロセッサが頻繁に入れ替わる場合は更新が有利になる。

これらのデータタイプの検出がコンパイラにできればよいが、できなければユーザが指定しなければならない。

3.6 キャッシュオンメモリ方式

この方式では、メモリは存在せずプログラムやデータはシステム内のどこかのキャッシュに存在する。またキャッシュディレクトリは階層構造となっている。データ参照の局所性が有効に生かすことができる。

参考文献

[1] 富田 眞治 並列コンピュータ工学 昭晃堂 1996

[2] 富田 眞治 コンピュータアーキテクチャ1 丸善 1994