参考資料：並列処理概論

四回生 杉野秀和

平成１０年5月6日(水)

1 はじめに

現代の汎用大型計算機には数十MIPS (Million Instructions Per Second)、数百MFLOPS (Mega FLOating point Par Second,Mega は10^6) の性能をもつものもある。しかし大規模な計算を処理するとなると、これらの計算機でも多くの時間がかかってしまう。並列処理はこの時間を短縮する重要な手段である。まずここでは、並列処理についての勉強を進めるにあたり最低限知っておくべき用語や概念について説明をする。

2 並行と並列

2.1 並行と並列の違い

並行とは論理的には同時に実行できることで、並列とは物理的にも同時に実行できることを指す。計算機上での処理における並行と並列は次のように区別される。

並行処理 :1つの処理装置が複数のプロセス間で切換え使用される。
並列処理 :1つのプロセスを複数の処理に分割して、複数の処理装置上で同時に実行する。
並行ではプロセス間で情報のやりとりをするが、並列では各処理は独立である。

2.2 並列の種類

実世界での場合を例にとってみよう。受付の窓口に多数の人が並ぶような場合を考えると、次のような分類ができる。

1. 空間並列
   空間上に並べられた複数の窓口を指示者が同時に仕事をさせる。横に並ぶ人達は互いに独立である。
2. 時間並列
   いくつかの窓口を順にまわることで手続きをすますやりかた。

また、このような分類のしかたも考えられる。

1. データ並列
   同じ業務を取り扱う窓口が複数あること。
2. コントロール並列
   通常では窓口全体に1人指示者がいるが、各窓口に指示者がいて、自分で仕事をする。

人をプロセスやデータ、窓口を処理装置、業務を処理方法、指示者を制御装置に置き換えればコンピュータでの話になる。

3 粒度による分類

粒度とは、並列処理する単位の大きさを指す。粒度で分類すると主に次のようになる。

- 細粒度
- 中粒度
- 粗粒度

細粒度とは演算または命令を単位とする方法で、演算装置を複数持つ計算機に適している。中粒度とはループなどの文を単位とする方法で、マルチプロセッサ型の計算機に適している。粗粒度とはサブルーチンを単位とする方法で、これもマルチプロセッサ型の計算機に適している。また、この分類では述べられないものもある。粒度を細くすると、並列度 (並列実行時の最長命令系列の長さ $*$ 並列度 $=$ 普通に実行した時の全命令系列の長さ) は大きくなるが、通信時間 (レイテンシ) が増大してしまう。この通信時間が処理にとってのオーバヘッドととなる (オーバヘッドとは本来 OS が使用するシステム資源の時間的、容量的割合またはそれらの絶対量をさすものだが、この場合、真の実行以外の無駄な処理のことをいう。)。逆に粒度が粗くなると、並列度が上がらなくなってしまう。従って、粒度をどのように設定すると最適になるか、というのが一つの課題である。

4 駆動方式

並列コンピュータは機械命令の起動形態で分類すると、制御駆動方式とデータ駆動方式の 2 つになる。ここではこの 2 方式について説明し、今日普及している方式をさらに分類する。

4.1 制御駆動方式

この方式は、プログラムカウンタ (以下 PC) に指示された機械命令が集中制御の下で逐次的に実行される、いわゆるノイマン型計算機である。並列実行可能な命令があっても PC に指示されないと実行できない。

4.2 データ駆動方式

この方式では、PC が存在せず、各機械命令はオペランドデータがそろえば全て同時に実行できる。プログラムはデータの依存関係を示すデータフローグラフで表現され、これに従って機械命令が実行され、原理的には、プログラム内の 全ての並列性を引きだせる。この方式には静的方式と動的方式の 2 つがあり、動的方式では同一プログラムを複数地点で同時に呼んだり、再帰的に呼んだりできる。しかし、データ駆動方式にも問題点がいくつかある。

- 演算レベルが四則演算レベルなので、並列の粒度が細くなり、通信のオーバヘッドが大きい。
- 通常のメモリの逐次アクセスの性質が利用しにくく、必要なメモリバンド幅 (データ転送の際の毎秒あたりのバイト転送量) が大きくなってしまう。
- 動的方式では呼び出されている多数のプログラムを識別する必要があるのだが、識別機構が複雑であり、処理の中でボトルネックとなる。(そこで時間がかかり、処理の流れが詰まってしまう。)

4.3 並列コンピュータの種類と実際の諸問題

主なものとして以下の 3 つが挙げられる。

- SIMD(Single Instruction Multiple Data) 方式
- パイプライン方式
- MIMD(Multiple Instruction Multiple Data) 方式

4.3.1 SIMD 方式

一つの制御装置のもとで複数の演算装置を動作させるもので、空間並列とデータ並列の両方の特徴をもつ。分岐などのない、定型的な処理に対して高速実行が可能である。

4.3.2 パイプライン方式

ジョブ内の命令をいくつかの基本処理に分解し、それらを対応するユニットで順次行い、ジョブ全体の実行時間を短縮させる。時間並列の特徴をもつ。高速性、連続性、単純性、汎用性、単一性を満たすため科学技術計算で普及しているスーパーコンピュータの分野において今のところ主流となっている。

- パイプラインにおける問題点
  実際にはパイプラインの実行を乱すものがあり、パイプラインのステージに待ちが生じてしまう。ここでは、その要因の主なものを挙げる。

1. データ依存
   フロー依存 :先の命令の結果を後の命令が使用する場合
   出力依存 :先の命令と後の命令が同じ場所に書きこみをする場合
   逆依存 :先の命令が読んだ後に後の命令が書きこみをする場合
   (出力依存、逆依存はレジスタリネーミングなどで回避できるが、フロー依存はどうしようもない。)
2. 制御依存
   並列化したい文の中にIF 文があったり、実行中に割込みがある場合
3. ハードウェア資源の競合
   資源獲得の失敗や、マシンサイクルの長い命令が存在する場合

4.3.3 MIMD 方式

MIMD 方式にはマルチプロセッサ、マルチコンピュータの2つがあり、どちらもコントロール並列である。前者は複数のマイクロプロセッサを結合して動作させるもので、後者は多数のコンピュータがネットワークで接続されているシステムのことであり、ネットワーク上で負荷の高いコンピュータがあれば、それのいくつかのジョブを他の送って処理させることで単位時間あたりの処理量 (スループット) を向上させるもの。実際のところは無駄が多く、台数を増やしても効果が出ないことがある。

- マルチプロセッサの大分類
  マルチプロセッサのメモリ構成は、共有メモリ型とメッセージパシング型の2つに大きく分類できる。

  - 共有メモリ型と同期の手法
    メモリが多数のプロセッサで物理的に共有されているもの。計算結果が間違うのを防ぐためデータ通信ではプロセス間で同期をとる必要がある。一般に同期は相互排他 (共有資源にアクセスできるのは1つだけ) と条件同期 (データの書きこみを確認してから受信側プロセスが読み込みをする動作) の2種類がある。最近ではバリア同期 (プロセスがバリアという同期ポイントに達したら協調動作中の全てのプロセスがバリアに到達するまで待つ) という手法がよく用いられる。
  - メッセージパシング型
    各プロセッサが共有メモリを介さず直接相互結合網で結ばれたもの。メモリは各プロセッサごとに閉じており、他のプロセッサのメモリを直接には参照できない。

各プロセッサは1対1でメッセージ通信を行う。通信の方法として、送信者、受信者の両方が準備できるまで待つ同期型(ブロッキング通信)と、送信受信側が共にメッセージをバッファに書きこむ非同期型(ノンブロッキング通信)がある。

このようなバッファをおいてプロセスを待たせずにすます方法をバッファリングとよび、通常は複数のバッファをおいて切換えをする(ダブルバッファリング)。ただしコストとの関係上、バッファの数には制限があり、どれだけのバッファで受信、送信側を待たせずにできるかという問題がある(有限バッファ問題)。

- MIMD方式の性能向上率

  プロセッサの個数を増やすことによって性能がどのくらい向上するかを考える。プログラムの中で並列化できるものの割合をαとし、並列化した部分がK台で実行されるならば、並列化して実行した時間は逐次実行した場合の時間に比べて、$\frac{K}{\alpha+(1-\alpha)*K}$倍速くなる。これをアムダール($E.Amdahl$)の法則という。この式で表現されるように、並列化できない部分が少しでもある場合に台数がある程度の数を越えると、効果つまりスピードアップ率が上がらなくなる(飽和 :$satulation$)。また全体としてのスケーラビリティにも影響を及ぼす。

5　プロセス間でおきる諸問題

これはマルチプロセッサに限らず、マルチプログラミング環境ならどこでも起こりうる問題である。

プロセッサに割りあてられたプロセスはデータ、プログラム、時にはプロセスそのもの(これらをまとめて資源とする)を要求する。要求した資源が得られないとプロセスはその資源が解放されるまで待つことになるのだが、場合によっては2つのプロセスが相手の欲しい資源を持ちあって動作できなくなる、という状態になる。この状態をデッドロックとよぶ。

デッドロックから回復する方法として以下の2つがある。

1. デッドロック状態にあるプロセスの1つを消滅させる。

2. デッドロック状態にあるプロセスの1つを元の状態に戻し、(後退復帰)その後に再びそのプロセスが資源を要求する。

どのプロセスを消滅または後退復帰させるかはシステムが決定するのだが、場合によっては常に1つのプロセスが選択されることがある。このような選択によってプロセスが永久に仕事ができなくなることを飢餓(starvation)とよぶ。飢餓がおこらないように回復させる手段として、中断と再開の機構がある。多くのシステムではチェックポイントを作り、そこに達したらその時点での状態を退避させ、後で再開するという方法をとっている。しかしチェックポイントの位置によっては中断と再開の繰り返しがおこったりもする。(ライブロック)

6　終わりに

並列処理の分野には上で述べた以外にも様々な方式や問題点がある。また応用の面でも様々な研究が進められている。(各方面の細かい説明は、後の人にお任せします。)

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)

[3] 情報科学辞典, 岩波書店

[4] 笠原 博徳, 並列処理技術, コロナ社 (1991)

[5] 村岡 洋一, 並列処理, 昭晃堂 (1986)

[6] 萩原、津田、大久保, 現代オペレーティイングシステムの基礎, オーム社 (1988)

[7] 白鳥、滝沢, 分散処理, 丸善 (1996)