並列計算機KUMP/Dのシステムプログラムの概要

2F－5

坂本和昭　富安洋史　雨宮真人
九州大学大学院 システム情報科学研究科

1　はじめに

並列計算機では、同期や通信メッセージなどに伴うレイテンシによって、CPUの稼働率が低くなるという問題がある。この問題に対して、レイテンシが生じる間に実行可能な細粒度の別プロセスへ切り替えてCPUの稼働率を上げる手法[1][2]がある。筆者らは現在、細粒度並列処理を行なう分散共有メモリ型の並列計算機KUMP/D (Kyushu University Multimedia Processor on Datarol-II)[3]を設計試作中である。図1にKUMP/Dの構成を示す。各PE(Processor Element)は、2次元トーラス網のネットワークで結合されており、市販のCPUとメモリマップドI/Oによってアクセスされる付加回路FMP(Fine-grain Message Processor)から構成されている。

効率の良い細粒度並列処理を実現するにはさまざまな問題がある。プロセスが細粒度になることによって増えるコンテクストスイッチや、メモリの獲得解放、スレッド間の同期、通信メッセージのハンドリング等のオーバーヘッドが特に問題である。FMPは、これらのオーバーヘッドを軽減するための機能を持っており、システムプログラムは、このFMPを用いて効率の良い細粒度並列処理のための環境を提供する。

本稿では、KUMP/Dにおけるシステムプログラムで特にコンテクストスイッチとメモリ獲得解放について述べる。

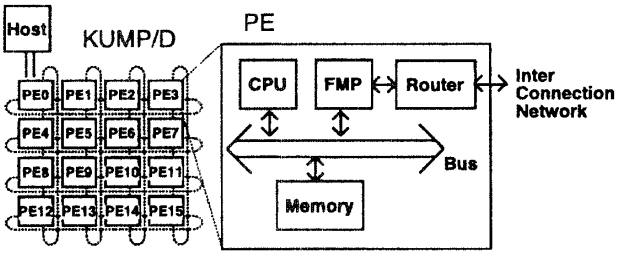

図1: KUMP/Dの構成

2　細粒度メッセージ駆動方式

KUMP/Dは、細粒度メッセージ駆動方式FMD(Fine-grain Message Driven Mechanism)に基づいた処理を行なう。

FMDは、メッセージパッシングによってプログラムを実行する方式である。関数はスレッドの集合によって構成され、スレッド間のメッセージによって実行が制御される。ここでいうスレッドとは、途中で中断されることなく逐次に実行できる命令列であり、コンテクストスイッチの単位である。関数起動によって生成されたインスタンスは、フレームと呼ばれる記憶領域を持つ。

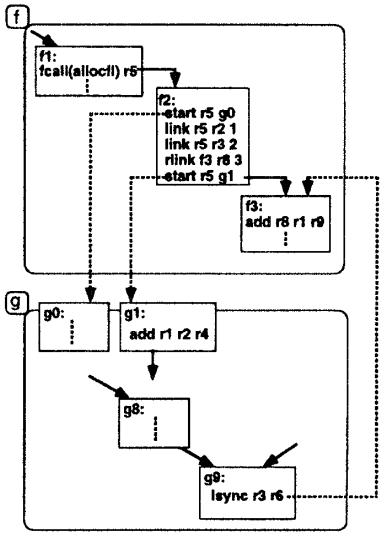

図2: FMDにおける関数適用

図2にFMDにおける関数適用を示す。図2中のアークはメッセージを表す。

まず、関数適用を行なうインスタンス(caller、ここではf)は、fcall命令によりcallee(ここではg)で使用するフレームを獲得する。次にcallerは、calleeに対してstart命令によりスレッド起動メッセージを送り、初期化スレッドg0を起動する。次にcallerは、リモートメモリ書き込みにより引数および関数の返り値の戻り先をcalleeに送った後、start命令によりスレッドg1を起動する。ここで、PE間ネットワークにおいて、同一の宛先に対してはメッセージの順序性が保証されていることを前提とする。

スレッドの同期もメッセージにより実現される。複数の入力を持つスレッドは、全てのメッセージが揃った時点で起動される。スレッド間のメッセージの依存関係に従ってスレッドが順次活性化され、プログラムが実行される。

3　システムプログラム

3節では、スレッド間の同期や、コンテクストスイッチ、メモリの獲得解放に関するFMPの支援機構を用いたシステムプログラムについて述べる。

通信メッセージのハンドリングはFMPで処理されるため、ここでは説明を省略する。

3.1　スレッド間の同期とスレッドキュー

スレッド間の同期は、同期カウンタを用いて実現する。各スレッドに入力されるメッセージの数

System Program for Parallel Processor KUMP/D
Kazuaki Sakamoto, Hirosi Tomiyasu,
Makoto Amamiya
Graduate School of Information Science and Electrical Engineering, Kyushu University

を同期カウンタに保持し、メッセージの到着時にFMPが同期カウンタをチェックする。同期が取れて実行可能となったスレッドは、スレッドキューと呼ぶ実行待ち行列に入れられる。

スレッドキューは、メインメモリ上にあり、リングバッファとして管理する。スレッドキューには、スレッドのエントリポインタとそのスレッドが属するインスタンスで使用されているフレームポインタが入っており、スレッドを実行する際にはスレッドキューからエントリポインタとフレームポインタが取り出される。

FMPで同期カウンタの管理を行なうため、CPUはその間にプログラムを実行でき、スレッド間の同期に要するオーバーヘッドを削減できる。

3.2 コンテクストスイッチ

コンテクストスイッチは、各スレッドの実行が終了する度に発生する。次に実行可能なスレッドへ切り替えるには、各スレッドの末尾にスレッド切り替えルーチンを挿入して実現する。以下にスレッド切り替えルーチンを示す。

1. スレッドキューが空かどうか?(HEADとTAILを比較する。)
2. 空でなければ3へ。空ならば6へ。
3. 新しいスレッドのエントリポインタとフレームポインタをセット。
4. スレッドキューのHEADを更新してFMP内のレジスタに反映する。
5. 新しいスレッドへジャンプ。
6. FMP内のレジスタを読み込み、スレッドキューのTAILを更新して1へ。

このスレッド切り替えルーチンはスレッドキューが空でなければ、10 clock程度になるので、効率の良いコンテクストスイッチが期待できる。

3.3 メモリの獲得解放

KUMP/Dでは、数種類(試作機では4種類)の固定長のフレームと任意の大きさのフレームを使用する。固定長のフレームを使用する事によって、メモリ管理を簡単にする。任意の大きさのフレームは、実行するプログラムにおいて、固定長のフレームで対応できない場合に使用される。

固定長のフレームを管理するためにメインメモリ上に、空きフレームのポインタを格納するスタックをサイズ毎に用意する。フレーム管理の高速化のため、各フレームスタックに対してそれぞれ高速なバッファをFMP内のメモリに持っており、フレームの獲得解放は通常、このバッファに対しての読み書きで行なわれる。フレームの獲得解放ルーチンはバッファや、スタックが空(あるいは一杯)になってから起動する。図3にメインメモリ上のスタックとFMP内のバッファの関係を表す。

FMPがフレームポインタのバッファを持つ事によって、他PEからのフレーム獲得要求が来た場合

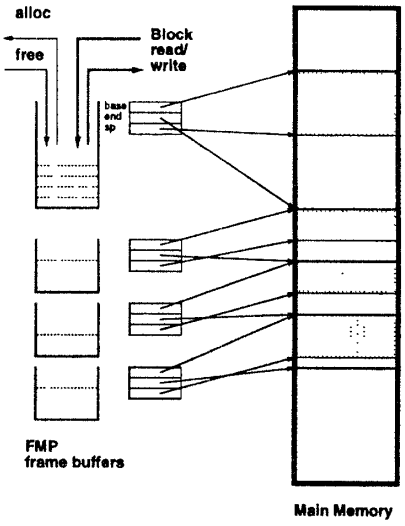

図3: フレーム管理

に、FMPのバッファからフレームポインタを獲得する事ができるため、CPUがプログラムの実行を中断する必要がなくなる。任意の大きさのフレームの獲得要求が来た場合は、メモリ獲得ルーチンを起動する必要があるが、プログラムで固定長のフレームを使用する割合が高ければ、頻繁にメモリ獲得ルーチンが起動される事はない。

このように、FMPの機能を用いる事によってシステムプログラムでは、不必要にメモリの獲得解放ルーチンを起動しなくて済むため、メモリの獲得解放によるオーバーヘッドを削減できる。

4 おわりに

KUMP/Dのシステムプログラムの概要について述べた。FMPの機能を活用したシステムプログラムによって、スレッド間の同期や、コンテクストスイッチ、メモリの獲得解放等のオーバーヘッドを抑え、効率の良い細粒度並列処理が行なわれる。

今後の課題として試作機では、16、32、64、128変数の固定長のフレームを使用するが、プログラムの実行に最適なフレームの大きさを検討する必要がある。

参考文献

[1] R.S.Nikhil, G.M.Papadopoulos and Arvind:*T:A Multithread Massively Parallel Architecture,*Proc. 19th ISCA*,pp.156-167(1992).

[2] Kawano,T.,Kusakabe,S.,Taniguchi,R. and Amamiya,M.:Fine-grain Multi-thread Processor Architecture for Massively Parallel Processing,*Proc. HPCA '95*,pp.308-317(1995).

[3] Tomiyasu,H.,Kawano,T.,Taniguchi,R. and Amamiya,M.:KUMP/D:the Kyushu University Multi-medea Processor, *Proc. Computer Architectures for Machine Perception '95*,pp.367-374(1995).