パーティショニングアルゴリズムを用いた時分処理の実装方法

3M－8

須崎 有康　田沼 均　平野 聡　一杉 裕志　Chris Connelly　塚本 享治

電子技術総合研究所
suzaki@etl.go.jp, http://www.etl.go.jp/Organization/Bunsan/Fluid/TSS.html

1　はじめに

筆者らは文献[1]においてパーティショニングアルゴリズムを使った時分割処理方式を提案した。時分割処理方式の性能はシミュレータにより、投入された全タスクの処理時間、タスクの応答性、プロセッサ利用率などで従来のパーティショニングアルゴリズムより効率が良いことが示された。本論文では実際の計算機に実装するに当たり、その方法と実装に必要となる要素技術について述べ、筆者らが所有するAP1000+に適した設計方針を明らかにする。

2　パーティショニングアルゴリズムを使った時分割処理

パーティショニングアルゴリズムは空間分割を行ないマルチタスクを実現する。筆者らはこのアルゴリズムを時分割処理に拡張する方法を提案した[1]。本方式では対象とする並列計算機と同じ構造の仮想並列計算機(スライス)を提供する。各スライスへのタスクの割り当てはパーティショニングアルゴリズムで行なう。既存のスライスへ割り当てられなくなったら新しいスライスを作成し、そのスライスにタスクを割り当てる。スライスはラウンドロビンで一定時間実計算機に割り当てられ、スライスに割り当てられたタスクの処理を進める。また本方式では、あるスライスのタスクが占有している領域が別のスライスで未使用の場合、空き領域のあるスライスにそのタスクを割り当て、プロセッサの利用率を上げることができる。時分割処理の動作を図1左に示す。複数のスライスに跨るタスクをマルチプルタスク、一つのスライスにのみ存在するタスクをシングルタスクと呼ぶ。

3　実装方法

2章で説明した時分割方式を実装するための方法を図1右に示す。ここでは各PEがローカルタスクリストを持つ。各タスクリストはスライス毎にリスト要素を取り、リストの長さはどのPEでも同一となる。図1右例ではslice1がリストの一番上の要素、slice2が二番目の要素、slice3が三番目の要素となる。

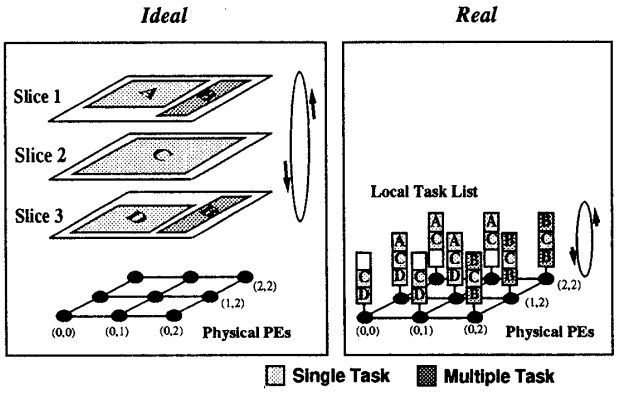

図1: TSSのモデルとその実装方法

投入されるタスクはスライスを想定したパーティショニングアルゴリズムで配置PEを決定し、それぞれのPEのリストにタスクを登録する。この際リストの末尾に登録されるわけではない。スライスの順番に対応するリスト要素の位置に登録される。また、スライスに対応してリスト要素を確保するため、実行すべきタスクがないリスト要素も存在する。例えば図1右のPE(0,0),(0,1)では、slice1でタスクが割り当てられていないため、一番目の要素に実行すべきタスクがない。逆にマルチプルタスクは複数個のリスト要素に登録される。図1右のPE(0,2),(1,2),(2,2)では、slice1,slice3に跨るマルチプルタスクが存在するため、一番目の要素と三番目の要素に同一のタスクBが割り当てられる。

各リスト要素はタイムクォンタが経過する度にラウンドロビンで巡回する。その際タスクコンテキストスイッチは個々のPE上で行なう。各コンテキストスイッチは並列プログラムを効率的に実行するため、同期して行なわれる。

4　要素技術

3章で述べた実装方法を実現するに当たって、要素技術に幾つかの選択肢がある。ここではAP1000+に対応して各要素技術について述べる。

The implementation method for time sharing systems that use a partitioning algorithm
Kuniyasu SUZAKI　Hitoshi TANUMA　Satoshi HIRANO
Yuuji ICHISUGI　Chris Connelly　Michiharu TUKAMOTO
Electrotechnical Laboratory

4.1 タスク切替え

並列計算機上でのタスク切替えでは、逐次計算機で行なわれているCPUのコンテキストスイッチの他にネットワーク上に存在するメッセージのコンテキストスイッチが問題となる。特にレイテンシを隠蔽するため同時期に多段のネットワークスイッチを経由するワームホールルーティングでは、コンテキストスイッチ時に複数のネットワークスイッチ上に一つのメッセージが分割して存在するため、この問題が複雑化する。この解決方法としては、ネットワーク上のメッセージをCPUのコンテキストスイッチと同期して退避／復帰する方法、メッセージにタスク識別子を付けて複数タスクのメッセージの混在を許す方法がある。前者の方法はCM5,RWC-1に採用されており、CPUのコンテキストスイッチと同期して起こるネットワークコンテキストスイッチの機構をハードウェアとして備えている。コンテキストスイッチ時にはスイッチ上のバッファにメッセージを退避する。後者の方法はAP1000+上のChorus[2]で採用されており、メッセージの送信先のPEでコンテキストスイッチが起こっていてもタスクを識別し、正しく送信できる。

AP1000+の場合、ネットワークコンテキストスイッチのハードウェアを有してない。しかし、既にChorusで実現されているような送信先タスク識別子付きメッセージで時分割処理は実現可能である。その代わり一回のメッセージ通信にオーバーヘッドがかかる。

4.2 コンテキストスイッチの同期

筆者らが提案した時分割処理方式では、全プロセッサが一斉に同期してタスク切替えを行なう。ネットワーク上のメッセージについては、4.1章で述べたようにコンテキストスイッチを省略することができるが、CPUコンテキストスイッチには一斉同期のメカニズムが必要である。K2プロジェクトのオペレーティングシステムChagoriでも時分割処理の実現には一斉同期がハードウェアとして必要であることが強調されている[3]。

AP1000+ではCPUからのブロードキャスト用ネットワークのB-netが装備されており、これを利用すればコンテキストスイッチの一斉同期が可能である。

4.3 閉パーティショニング

パーティショニングアルゴリズムの多くは、対象計算機に閉パーティショニングを要求する。閉パーティショニングとは割り当てた個々のタスクのメッセージが他のタスクが利用するネットワーク上を通らずに送ることができる性質である。この性質により、個々のタスクはお互いに干渉することなく処理を進めることができる。

AP1000+ではメッセージの宛先範囲チェックを行なうハードウェアを有しており、閉パーティショニングを保証できる。しかし、時分割処理の実現でネットワークコンテキストスイッチを行なわない方法を採用すれば、コンテキストスイッチ直後はコンテキストスイッチ以前のメッセージがネットワーク上に存在し、現在実行しているタスクが干渉されることが予想される。このため閉パーティショニングを完全に保証できない。しかし、論文[4]によるとIntel Paragonを対象とした結果のみだが、タスク同士のメッセージによる干渉は小さく、閉パーティショニングを気にする必要はないと述べられている。筆者らの時分割処理では、メッセージの干渉がコンテキストスイッチ直後に限定されているため小さいく、影響は少ないと予想する。

4.4 メモリマッピングとプロテクション

各タスクが使用するメモリに対するマッピングとプロテクションは逐次計算機の時分割処理同様に重要な課題である。特に現在の並列計算機ではリモートメモリアクセスも必須の機構となっている。これに対処して、時分割処理によって休止した相手タスクのリモートメモリに正しく書き込めるマッピング機構も必要である。

AP1000+では、リモートメモリアクセスに対するマッピング機構をネットワークハードウェアとして有しており、これを利用することで休止したタスクに対するマッピングとプロテクションを実現できる。

4.5 その他

時分割処理化では共有資源の扱いに注意しなければならない。特にAP1000+では同期用ネットワークS-netがあり、これに対処しなければならない。

5 まとめ

本論文では筆者らが提案した時分割処理方式を実際の計算機に実装する方法とそれに必要な要素技術について述べた。特に筆者ら所有するAP1000+を対象として実現可能性を検討した。ここでは機能面について調査したが効率面ではまだ不確定な要素があり、今後は実際の使用に耐えうる効率が出せるかを検討する必要ある。

謝辞 本研究の一部はRWC計画の一環として「超並列システムアーキテクチャに関する研究」で行なわれたものである。関係各位に感謝する。

参考文献

[1] K. Suzaki, H. Tanuma, S. Hirano, and Y. Ichisugi. A time sharing system scheme that uses a partitioning algorithm for mesh-connected parallel computers. *7th IEEE Symposium on Parallel and Distributed Processing*, 1995.

[2] N. Imamura, N. Fujisaki, H. Ishihara, and M. Ikesaka. Implementation and Performance of the Chorus Micro-Kernel IPC on the AP1000+. *The Fourth Parallel Computing Workshop(PCW'95)*, 1995.

[3] P. Steiner. Extending multiprogramming to a DMPP. *Future Generation Computer Systems*, (8), 1992.

[4] W. Liu, V. Lo, K. Windish, and B Nitzberg. Non-contiguous Processor Allocation Algorithms for Distributed Memory Multicomputers. *Supercomputing*, 1994.