マクロデータフロー処理のためのジョブスケジューリング手法

2H－5

合田 憲人　笠原 博徳　成田 誠之助
早稲田大学理工学部

1　はじめに

マルチプロセッサシステム上で各々が複数PEを用いて実行される並列プログラム(ジョブ)をマルチジョブ実行する場合、単一ジョブ内タスクの効率的な並列処理と複数ジョブ間での効率的な並列実行の両方が要求される。しかし、従来から行なわれている複数ジョブ内のタスクをFCFS(First Come First Served)、RR(Round Robin)ベースでスケジューリングする手法[1]では、単一ジョブ内タスクの並列性抽出が十分に行なわれていないだけでなく、OSによるタスクスケジューリングのオーバーヘッドが大きいため、単一ジョブ内タスクの効率的な並列処理が行なわれていない。

本稿では、ジョブ内部でマクロデータフロー処理[2, 3]が行なわれるジョブ間のマルチジョブ実行手法について述べる。本手法では、各ジョブ内部でマクロタスクが生成され、それらのマクロタスクがマクロデータフロー処理により効率的に並列処理される。また、複数ジョブ間のスケジューリングも、コンパイル時の解析により得られる各ジョブを実行するために必要なPE数をもとにして効率良く行なわれる。

2　マクロデータフロー処理手法

本節では、Fortranプログラムのマクロデータフロー処理手法について概説する。

マクロデータフロー処理では、はじめにFortranプログラムをマクロタスクと呼ぶ粗い粒度をもつタスクに分割する。マクロタスクは、基本ブロック(BB)あるいは複数の基本ブロックからなるブロック(BPA)、繰り返しブロック(RB)、サブルーチンブロック(SB)から構成される。プログラム分割終了後、マクロタスク間の制御フロー、データ依存を解析することにより、これらの情報を表現するマクロフローグラフ(MFG)を生成する。次にマクロタスク間の制御依存、データ依存関係を解析することにより、各マクロタスクの並列性を抽出した最早実行開始条件[2]を解析し、これをグラフ表現したマクロタスクグラフ(MTG)を生成する。

各マクロタスクは、MTGの情報をもとにして、条件分岐やマクロタスク処理時間の変動に対処するために、Dynamic-CP法により実行時にプロセッサに割り当てられる[3]。一般的なOSあるいはライブラリコールによるダイナミックスケジューリングは実行時のオーバーヘッドが大きいが、本手法では、コンパイラが各Fortranソースプログラム毎に自動生成する専用のスケジューリングコードによって、マクロタスクのダイナミックスケジューリングが行なわれるため、オーバーヘッドを小さく抑えることができる。

3　ジョブスケジューリング手法

本節では、各々がマクロデータフロー処理されるジョブをマルチジョブ環境下で実行するためのジョブスケジューリング手法について述べる。

3.1　ジョブの実行PE数

マクロデータフロー処理では、ジョブを実行するために必要なPE数$ExecPE$を以下の式によりコンパイル時に算出するものとする。

$$ExecPE = \left\lceil \frac{SeqTime}{CPlength} \right\rceil$$

ここで$SeqTime$は、プログラムのシーケンシャル実行時間であり、MFG上のコントロールフローをプログラムの出口から入口へ走査し、コントロールフローパス上の各マクロタスクの処理時間の和として算出する。$CPlength$は、MTGのクリティカルパスである。ただし、MFG、MTGは条件分岐を含んでいるため、$SeqTime$、$CPlength$の算出に際し、条件分岐を含むマクロタスクについては、分岐確率(推定不可能な場合は確率的に平等とする)によって重み付けをした値を用いる。

3.2　Fit Processors First Scheduling

ここでは、マクロデータフロー処理されるジョブのジョブスケジューリング手法であるFPFS(Fit Processors First Scheduling)について述べる。

マクロデータフロー処理では、コンパイル時に各ジョブの実行に必要なPE数が決定できるため、各ジョブに

A Job Scheduling Scheme for Macro-dataflow Computation
Kento AIDA, Hironori KASAHARA, Seinosuke NARITA
WASEDA University

```
int idlePE;                    /* number of idle PE */
struct job_type {
   int execute_pe;             /* number of PE job needs */
   int cost;                   /* execution time of job (if predictable) */
struct job_type *target_job;   /* job to schedule */
struct queue_type {
   struct job_type *job;       /* job */
} queue[QUEUEMAX];             /* job queue */

Fit_Pe_First_Scheduling()
{
   int target;

   SORT_QUEUE(queue);
   target = 1;
   target_job = queue[target].job;
   while((target_job != NULL)&&(idlePE != 0)) {
        if(target_job.execute_pe <= idlePE) {
             SCHEDULE(target_job);
             target++;
             target_job = queue[target].job; }
        else {
             target++;
             target_job = queue[target].job; } }
}

SORT_QUEUE(queue)
{
   sort jobs in job queue;
}

SCHEDULING(target_job)
{
   schedule target_job;
}
```

図 1: Fit Processors First Scheduling

対して必要な PE 数分のプロセッサを割り当てる必要がある。FPFS は、キュー内のジョブをキューの先頭からサーチし、ジョブの実行に必要な PE 数がシステムのアイドル PE 数より少ないジョブ (=スケジューリング可能なジョブ) に対してプロセッサを割り当てる。また、キュー内のジョブを実行に必要な PE 数等でソートすることにより、プロセッサ利用率を向上させることができる。FPFS のアルゴリズムを図 1に示す。

4 性能評価

本節では、3節で述べたジョブスケジューリング手法の性能をシミュレーションにより評価する。

対象とするシステムは、32PE から構成される UMA 型マルチプロセッサシステムとする。ここで、全ジョブは Non-preemptive に実行され、スケジューリングオーバーヘッドは、ジョブの処理時間に比べて十分小さいと仮定して考慮しないものとする。また、実行するジョブの条件として、ジョブを実行するために必要な PE 数を 1〜32PE の一様乱数、ジョブの実行時間を平均値を 10 単位時間とする指数分布乱数により決定する。100 ジョブから構成されるジョブセットを用意し、ジョブセットを実行した場合の平均応答時間およびプロセッサ利用率を測定する。

ジョブスケジューリング手法として、FPFS、FPFS においてキュー内のジョブを実行に必要な PE 数もとにソートした FPMPFS(Fit Processors Most Processors First Scheduling)、FCFS を用いた場合の全ジョブ平均応答時間とプロセッサ利用率 (100 種類のジョブセットに対するシミュレーション結果の平均値) を図 2に示す。図 2では、左縦軸に平均応答時間、右縦軸にプロセッサ利用率をとっている。また、横軸の WR(Workload Ratio)

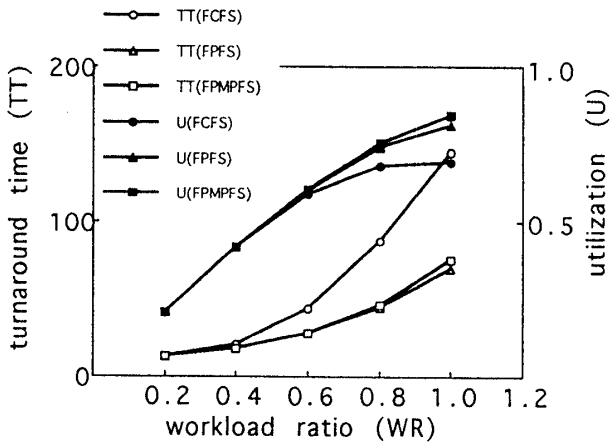

図 2: 平均応答時間・利用率

は、対システム負荷比であり、以下の式より算出される。

$$WR = \frac{\lambda(JobCost \cdot JobPE)}{SysPE}$$

ここで、$JobCost$、$JobPE$はそれぞれジョブセット中の全ジョブの実行コスト、実行に必要な PE 数の平均値である。$SysPE$は対象とするシステムの PE 数 (=32) である。λは単位時間内のジョブの平均到着数で、ジョブはλを平均値としたポアソン分布で到着するものとする。λの値は、各ジョブセット毎に WRが一定となるように決定される。

図 2より、FPFS は、応答時間およびプロセッサ利用率の面で、WRが高い場合に特に FCFS より優れていることがわかる。また FPMPFS は、FPFS に比べてプロセッサ利用率の面で優れているが、応答時間の面では劣ることが確認された。

5 むすび

本稿では、ジョブ内部のマクロタスクがマクロデータフロー処理されるジョブのマルチジョブ実行手法について述べた。ジョブスケジューリング手法についてシミュレーションによる評価を行なった結果、FPFS が FCFS より優れていることが確認された。今後、マクロデータフロー処理と FPFS の統合的な評価を行なっていく予定である。

参考文献

[1] S.T.Leutenegger, M.K.Vernon: The Performance of Multiprogrammed Multiprocessor Scheduling Policies, Proc. 1990 ACM SIGMETRICS Conf. on Measurement and Modeling of Computer Systems, pp226-236, May 1990

[2] 本多, 岩田, 笠原：Fortran プログラム粗粒度タスク間の並列性検出手法, 信学論 D-I, Vol.J73-D-I, No.12, pp951-960, (1990-12)

[3] 本多, 合田, 岡本, 笠原: Fortran プログラム粗粒度タスクの OSCAR における並列実行方式, 信学論 D-I, Vol.J75-D-I No.8, pp526-535,(1992-08)