システムソフトウェアと
オペレーティング・システム 73－13
（1996．8．27）

マルチプロセッサシステム上での並列ジョブのスケジューリング手法の評価

合田 憲人　笠原 博徳　成田 誠之助

早稲田大学理工学部

E-mail:aida@oscar.elec.waseda.ac.jp

マルチプロセッサシステム上で各ジョブにジョブが要求する数のプロセッサを割り当てるジョブスケジューリング手法では，従来より，到着順でジョブにプロセッサグループを割り当てる手法が提案されているが，これらの手法では，プロセッサフラグメンテーションが大きいという問題点がある．本稿では，プロセッサフラグメンテーションを小さく抑え，プロセッサ利用率を向上させる手法である Fit Processors First Served（FPFS）および Fit Processors Most Processors First Served（FPMPFS）のマルチプロセッサシステム NEC Cenju-3 上での性能評価ついて述べる．これらの性能評価結果より，FPFS および FPMPFS が従来手法よりも，実システム上でのプロセッサ利用率を 9%～19%向上させる等，有効性，実用性の高いことが確認された．

Evaluation of a Scheduling Scheme of Parallel Jobs on a Multiprocessor System

Kento AIDA　Hironori KASAHARA　Seinosuke NARITA

WASEDA University

E-mail:aida@oscar.elec.waseda.ac.jp

Job scheduling schemes that dispatch required numbers of processors to jobs in First Come First Served manners on a multiprocessor system have been proposed. However, these schemes have a drawback such that processor fragmentation is large. This paper presents performance evaluation of Fit Processors First Served (FPFS) and Fit Processors Most Processors First Served (FPMPFS), which can reduce processor fragmentation and improve processor utilization, on a multiprocessor system NEC Cenju-3. These evaluation results show that FPFS and FPMPFS improve processor utilization by 9% - 19%, and are more effective and practicable than conventional schemes.

1 はじめに

マルチプロセッサシステム上での単一ジョブの並列処理方式として，SPMD 方式等を初めとしたループ並列処理 [1] やマルチグレイン並列処理手法 [2] のように，ジョブを実行するプロセッサ（PE）数をプログラミング時またはコンパイル時に決定し，データローカリティを有効利用しようとする方式がよく用いられる．このようなジョブがマルチプロセッサシステムに動的に到着する場合，従来より OS ジョブスケジューラが各ジョブが要求する数の PE を割り当てる方式がとられていた．

このような OS によるジョブスケジューリング手法としては，従来より，First Fit，Best Fit，非連続割り当て等の方式でメッシュ結合網等で接続された PE を各ジョブに割り当てる手法 [3, 4, 5, 6, 7] が提案されているほか，多段結合網等で接続された任意の位置にある PE を割り当てる手法 [8] が実用化されている．しかしこれらの手法では，ジョブをプロセッサグループに割り当てる順序がジョブの到着順（FCFS）で行われており，プロセッサフラグメンテーションが大きいためにプロセッサ利用率が下がるという問題があった [5, 9]．

著者らは，このような問題を改善するために，スケジューリング時にジョブキューを検索してその時点でのアイドルプロセッサ数に適合するジョブをプロセッサグループに割り当てることにより，プロセッサフラグメンテーションを小さく抑え，プロセッサ利用率を向上させる手法である Fit Processors First Served（FPFS）および Fit Processors Most Processors First Served（FPMPFS）を提案してきた [10]．本稿では，FPFS および FPMPFS のマルチプロセッサシステム NEC Cenju-3 上での性能評価について述べる．

以後，2節では，本稿が対象とするジョブスケジューリングのモデルについて述べ，3節では，FPFS および FPMPFS について述べる．4節では，FPFS，FPMPFS のマルチプロセッサシステム NEC Cenju-3 上での性能評価について述べる．

2 ジョブスケジューリングモデル

本節では，本稿が対象とするジョブスケジューリングのモデルについて述べる．

2.1 マルチプロセッサシステム

本稿では，対象とするマルチプロセッサシステムの PE は，クロスバーネットワークまたは多段結合網などにより平等に接続されているものとする．またこのようなマルチプロセッサシステムでは，ジョブを実行する PE の物理的な配置がジョブの実行時間に与える影響が小さいため，ジョブに対して任意の位置にある PE を割り当てられるものとする．

2.2 ジョブスケジューラ

本稿が対象とするジョブスケジューラモデルでは，図1に示すジョブスケジューラが，動的にジョブキューに到着するジョブに対して，ユーザまたはコンパイラにより決定された数の PE を割り当てる．

ジョブスケジューラは，PE 上で実行されているジョブやアイドルプロセッサ数等のマルチプロセッサシステム上の PE の状況を管理しており，ジョブキュー内のジョブに対して，ジョブが要求する数（要求プロセッサ数）のアイドルプロセッサを割り当てる．ここで，2.1節で述べたように，ジョブスケジューラは，ジョブに対して，PE をマルチプロセッサシステム上の物理的な配置に関係なく割り当てることができるものとする．

3 ジョブスケジューリング手法

本節では，Fit Processors First Served（FPFS）および，Fit Processors Most Processors First Served（FPMPFS）について述べる．

従来より用いられている First Come First Served（FCFS）では，ジョブスケジューラは常にジョブキューの先頭のジョブに対してアイドルプロセッサを割り当てるため，ジョブキュー先頭のジョブの要求プロセッサ数に比べてアイドルプロセッサ数が少ない場合，ジョブの割り当てが行なわれず，プロセッサ利用率が下がる原因となっていた．これに対し，FPFS，FPMPFS では，ジョブスケジューラが，以下に示す方法を用いてジョブキューを検索することにより，その時点でのアイドルプロセッサ数に適合するジョブをプロセッサグループに割り当てるため，プロセッサ利用率の向上を図ることができる．

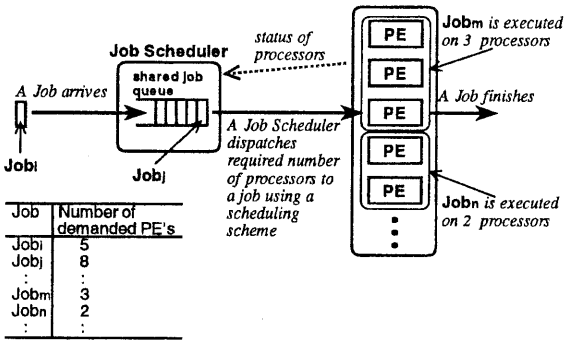

図 1: ジョブスケジューラモデル

3.1 FPFS

FPFS では，ジョブスケジューラがマルチプロセッサシステム内のアイドルプロセッサ数をもとにしてジョブキューの先頭のジョブから順にジョブの検索を行い，要求プロセッサ数がアイドルプロセッサ数よりも少ないジョブにプロセッサグループを割り当てる．

FPFS におけるジョブへのプロセッサグループ割り当て手順を以下に示す．ここで，ジョブキュー内のジョブを Job_i で表し，ジョブはジョブキューの先頭から，$Job_1, Job_2, \cdots, Job_n$ の順で並べられているものとする．また，Job_i の要求プロセッサ数を $DemandedPE_i$，マルチプロセッサシステム上のアイドルプロセッサ数を $IdlePE$ とする．

1. $i = 1$.
2. $DemandedPE_i$ と $IdlePE$ を比較する．
   (a) $DemandedPE_i > IdlePE$ の場合，3へ．
   (b) $DemandedPE_i \leq IdlePE$ の場合，Job_i に $DemandedPE_i$ 数の PE から構成されるプロセッサグループを割り当てる．$IdlePE = IdlePE - DemandedPE_i$．3へ．
3. $i = i + 1$. $i \leq n$ かつ $IdlePE > 0$ ならば，2へ．
4. 1へ．

また上記の手順のほかに，PE 上で実行されているジョブが終了した場合に，以下の処理を行う．

1. 実行が終了したジョブに割り当てられていた PE の解放
2. $IdlePE = IdlePE$ + 実行が終了したジョブに割り当てられていた PE 数

FPFS におけるスケジューリングコストは，ジョブキュー内のジョブ数を n とすると，$O(n)$ である．またこのスケジューリングの実行時間は，実行されるジョブの実行時間に比べると十分に小さいことが実測により確認されている．

3.2 FPMPFS

本節では，プロセッサ利用率をさらに向上させるための FPMPFS について述べる．FPMPFS では，はじめにジョブキュー内のジョブをジョブの要求プロセッサ数の非増大順でソーティングし，FPFS と同じ方式でジョブへのプロセッサグループ割り当てを行う．本稿で対象とするジョブスケジューリング問題をジョブキュー内のジョブをアイドルプロセッサに配置する Bin-packing 問題として考えると，Bin-packing 問題では，パッキングする対象物をそのサイズの非増大順でソーティングすることにより，パッキングの効率が向上するため [11]，FPMPFS においても同様の効果が期待できる．

3.3 Wait Limit の設定

FPFS および FPMPFS では，ジョブキューの検索によって，要求プロセッサ数がアイドルプロセッサ数よりも少ないジョブに，優先的にプロセッサグループが割り当てられるため，スタベーションが発生する可能性がある．このようなスタベーションの発生を防ぐため，各ジョブにプロセッサグループ割り当て待ち期限（$WaitLimit$）を設定する．

本 $WaitLimit$ を用いた FPFS および FPMPFS

では，ジョブスケジューラは，スケジューリング時にジョブキュー内での待ち時間が $WaitLimit$ 以上のジョブ（$Job_{overWaitLimit}$）が存在した場合，ジョブキュー内の $Job_{overWaitLimit}$ 以降のジョブへのプロセッサグループ割り当てを行わない．$WaitLimit$ を用いた場合の FPFS および FPMPFS における具体的なジョブへのプロセッサグループ割り当て手順は，3.1で示した手順のうち，手順2aを以下のように変更したものである．

(a) $DemandedPE_i > IdlePE$ の場合，
  i. Job_i のジョブキュー内での待ち時間が $WaitLimit$ 以上である場合，4へ．
  ii. Job_i のジョブキュー内での待ち時間が $WaitLimit$ 未満である場合，3へ．

また FPMPFS におけるジョブのソーティングでも，ジョブキュー中に $Job_{overWaitLimit}$ が存在する場合，ジョブキュー中の $Job_{overWaitLimit}$ より前の位置に新たなジョブの登録を行なわない．

4 Cenju-3 上での性能評価

本節では，FPFS，FPMPFS のマルチプロセッサシステム NEC Cenju-3 上での性能評価について述べる．

4.1 Cenju-3 のアーキテクチャ

Cenju-3 は，最大256台の PE および1台のホストコンピュータを多段結合ネットワークにより接続した構成である．各 PE は，マイクロプロセッサ VR4400 および最大 64MByte のローカルメモリから構成され，またネットワークの通信性能は，40MByte/s である [12]．今回用いたシステムは，8PE から構成される Cenju-3 Model 8S で，単一 PE のピーク性能は 33.3MFLOPS，単一 PE 上のメモリ容量は 32MByte である．またホストコンピュータとして，NEC EWS4800/330EX （CPU:75MHz）が接続されている．

4.2 ジョブスケジューラ

4.2.1 Cenju-3 上でのジョブ管理システム

Cenju-3 上のジョブスケジューリング機構は，ホストコンピュータ上で動作するジョブ管理システムおよびプロセス管理システムから構成されている．ジョブ管理システムは，Cenju-3 上の PE の利用状況をもとに，ジョブキュー内のジョブに対して，FCFS でジョブが要求する台数の PE を割り当て，プロセス管理システムに通知する．プロセス管理システムは，Cenju-3 上の PE や PE 上で実行中のジョブの状態を管理し，ジョブ管理システムに対してこれらの情報を通知するとともに，ジョブ管理システムから通知されたジョブを実際に Cenju-3 上で実行させる [8]．

4.2.2 ジョブスケジューラの実装

FCFS，FPFS，FPMPFS の性能評価を行うために，Cenju-3 上のジョブ管理システムを改良し，ジョブスケジューラを開発した．本ジョブスケジューラに実装した FCFS，FPFS，FPMPFS のスケジューリングルーチンの基本構成を図 2に示す．図 2 中，FCFS_Scheduler，FPFS_Scheduler は，それぞれ，FCFS，FPFS におけるジョブへのプロセッサグループ割り当てルーチンを示し，ジョブスケジューラは本ルーチンを繰り返し実行する．また FPMPFS では，ジョブのソーティングをジョブのジョブキューへの登録時に行い，ジョブへのプロセッサグループ割り当ては，FPFS_Scheduler を用いる．FPMPFS におけるジョブのジョブキューへの登録を行うルーチンを FPMPFS_job_spooler に示す．また Processor_relinquishment は，実行中のジョブが終了した場合に実行されるルーチンで，実行が終了したジョブに割り当てられていたプロセッサグループを解放する．

4.3 性能評価結果

本性能評価では，有限要素法および境界要素法を用いた電磁界解析プログラム [13]，3 次元マルチグリッド法プログラム [14]，ガウスザイデル法によるスパース行列解法プログラムから 500 個のジョブを構成し，一様乱数により決定した順序で実行した．これら 500 個のジョブ中，各ジョブの要求プロセッサ数の分布は一様であり，また，プログラムのロード等，実行に必要な処理も含むジョブの実行時間の平均値は，32[sec.] である．各ジョブの到着時刻はポアソン分布乱数によって決定し，ジョブの到着間隔は (1) 式によって表す．

$$load = \frac{\lambda \cdot p}{m \cdot \mu} \qquad (1)$$

```
queue[QUEUEMAX];   /* job queue */
queue[i].job;      /* a job at i th position in a job queue */
queue[i].job.penum;  /* number of demanded PE's by queue[i].job */
queue[i].job.wl;   /* WaitLimit of queue[i].job */
QueueLength;       /* number of jobs in a job queue */
IdlePeNumber;      /* number of idle processors */
ArrivalJob;        /* arrival job */
ArrivalJob.penum;  /* number of demanded PE's by ArrivalJob */
FinishJob;         /* job which finished its execution */
FinishJob.penum;   /* number of demanded PE's by FinishJob */

FCFS_Scheduler()
{
 while (){
  if(queue[1].job.penum <= IdlePeNumber){
   Dispatch_Processors_to_queue[1].job;
   IdlePeNumber - = queue[1].job.penum;
  }
 }
}

FPFS_Scheduler()
{
 while (){
  i = 1;
  while(i <= QueueLength && IdlePeNumber > 0){
   if(queue[i].job.penum <= IdlePeNumber){
    Dispatch_Processors_to_queue[i].job;
    IdlePeNumber - = queue[i].job.penum;
    i++;
   } else {
    if(queue[i].job.wl <= waiting_time_of_queue[i].job )
     break;
    else
     i++;
   }
  }
 }
}

FPMPFS_job_spooler()
{
 i = QueueLength;
 while(i > 0){
  if(queue[i].job.wl <= waiting_time_of_queue[i].job )
   break;
  if(queue[i].job.penum < Arrivaljob.penum)
   i - -;
  else
   break;
 }
 insert_Arrivaljob_into_(i+1 )th_position_in_a_job_queue;
}

Processor_relinquishment()
{
 Release_processors_dispatched_to_Finishjob;
 IdlePeNumber += FinishJob.penum;
}
```

図 2: Cenju-3 上でのジョブスケジューリングルーチン基本構成

ここで，λはジョブの到着率，$1/\mu$はジョブの平均実行時間，pはジョブの平均要求プロセッサ数，mはマルチプロセッサシステムの PE 数とする．また，FPFS および FPMPFS では，$WaitLimit = 600$[sec.] とした．

図 3，図 4に本ジョブを Cenju-3 上で実行した場合のプロセッサ利用率と平均応答時間をそれぞれ示す．また，本性能評価における FCFS では，Cenju-3 上で標準的に提供されているものと，これに対してスケジューリングオーバーヘッド軽減などを行って新たに実装したものを用いる．図中では，前者を FCFS(native)，後者を FCFS(improved) と表す．

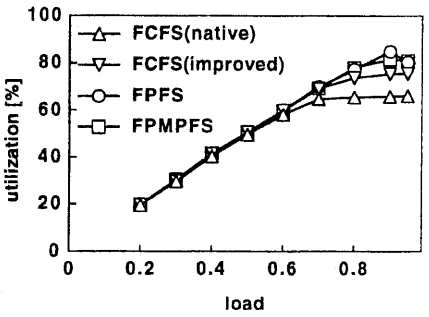

図 3: Cenju-3 上でのプロセッサ利用率

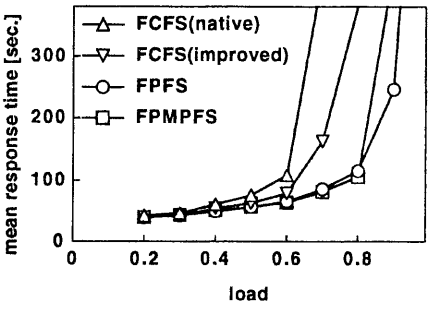

図 4: Cenju-3 上での平均応答時間

図 3，図 4より，FPFS および FPMPFS が，FCFS に比べてプロセッサ利用率を向上させるとともに，平均応答時間を低く維持できていることがわかる．FPFS におけるプロセッサ利用率は，FCFS(improved) に比べて最大で 9[%]，また FCFS(native) に比べて最大で 19[%] 向上している．また FCFS(native)，FCFS(improved) における平均応答時間は，FCFS では高負荷時にプロセッサを有効利用できないためにオーバーロードとなり，それぞれ $load = 0.7$ 時，$load = 0.8$ 時に 350[sec.] 以上に急騰しているが，FPFS，FPMPFS における平均応答時間は，80[sec.] から 115[sec.] に抑えられている．

$load = 0.95$ 時に FPFS，FPMPFS のプロセッサ利用率が 81[%] に低下するのは，$WaitLimit$ の設定により，プロセッサグループ割り当て時のジョブキューの検索範囲が制限されたためである．また $load = 0.9$ 時に，FPMPFS のプロセッサ利用率が FPFS より

も3[%]低下するのは，FPMPFSでは，*WaitLimit*によってジョブキューの検索範囲が制限された回数がFPFSに比べて約12倍と多かったためである．

FCFS(improved)では，図2に示すように，アイドルプロセッサ数の管理をジョブスケジューラのローカル変数を用いて行っており，ジョブへのプロセッサグループ割り当て時および解放時に，直ちにアイドルプロセッサ数が更新される．これに対してFCFS(native)では，プロセス管理システムに対して定期的にアイドルプロセッサ数を問合わせる方式をとっているため，アイドルプロセッサ数更新のためのオーバーヘッドが大きく，FCFS(native)は，FCFS(improved)よりも性能が低下している．

5 むすび

本稿では，動的に到着する並列ジョブをジョブが要求する数のPEから成るプロセッサグループに割り当てるジョブスケジューリング手法であるFPFSおよびFPMPFSのマルチプロセッサシステムNEC Cenju-3上での性能評価について述べた．本稿で示した性能評価結果より，

1. FPFS，FPMPFSが，従来手法であるFCFSに比べて，実システム上でのプロセッサ利用率を9%～19%向上させる等，プロセッサ利用率を向上させるとともに，FCFSでは平均応答時間が急騰してしまう高負荷時でも，平均応答時間を低く抑えることができ，有効性，実用性が高い．
2. FPFSは，FPMPFSよりもスケジューリングのための処理が少ないにもかかわらずFPMPFSと同様の性能を示すことから，本稿で取り上げたジョブスケジューリング手法中，実運用にはFPFSが適している．

ことが確認された．

FPFSおよびFPMPFSにおける*WaitLimit*の決定方法については，プロセッサ利用率とジョブの応答時間のトレードオフを考慮する必要があるため，今後最適な*WaitLimit*を決定する手法の開発が必要である．さらに，今後の課題として，大規模システム上でのFPFS，FPMPFSの性能評価があげられる．

謝辞

Cenju-3の使用環境を提供していただいている
NEC C&C研究所に感謝いたします．

参考文献

[1] High Performance Fortran Forum: High Performance Fortran Language Specification Version 1.0 (1993).

[2] Kasahara, H., Honda, H., Aida, K., Okamoto, M. and Narita, S.: OSCAR Fortran Compiler, *Proc. Workshop on Compilation of Languages for Parallel Computers*, pp. 30–37 (1991).

[3] Li, K. and Cheng, K.: A Two-Dimensional Buddy System for Dynamic Resource Allocation in a Partitionable Mesh Connected System, *J. Parallel and Distributed Computing*, Vol. 12, pp. 79–83 (1991).

[4] Chuang, P. and Tzeng, N.: An Efficient Submesh Allocation Strategy for Mesh Computer Systems, *Proc. International Conference on Distributed Computing Systems*, pp. 256–263 (1991).

[5] Zhu, Y.: Efficient Processor Allocation Strategies for Mesh-Connected Parallel Computers, *J. Parallel and Distributed Computing*, Vol. 16, pp. 328–337 (1992).

[6] Liu, W., Lo, V., Windisch, K. and Nitzberg, B.: Non-contiguous Processor Allocation Algorithms for Distributed Memory Multicomputers, *Proc. Supercomputing '94*, pp. 227–236 (1994).

[7] Feitelson, D. G. and Rudolph, L.: Prallel Job Scheduling: Issues and Approaches, *Job Scheduling Strategies for Parallel Processing*, Springer-Verlag, pp. 1–18 (1995).

[8] 日本電気株式会社: NEC並列コンピュータCenju-3利用者の手引 (1994).

[9] Krueger, P., Lai, T. and Dixit-Radiya, V. A.: Job Scheduling Is More Important than Processor Allocation for Hypercube Computers, *IEEE Trans. Parallel and Distributed Systems*, Vol. 5, No. 5, pp. 488–497 (1994).

[10] 合田, 岡本, 笠原, 成田: 階層並列実行ジョブ間スケジューリング手法, 情処研報, Vol. ARC-111-1, pp. 1–8 (1995).

[11] Coffman, E. G., Garey, M. R. and Johnson, D. S.: Approximation Algorithms for Bin-paking – An Updated Survey, *Algorithm Design for Computer System Design*, Springer-Verlag, pp. 49–106 (1984).

[12] 丸山, 加納, 広瀬, 中田, 村松, 浅野, 稲村: 並列コンピュータCenju-3のアーキテクチャとその評価, 信学論, Vol. J78-D-I, No. 2, pp. 59–67 (1995).

[13] 坂本, 前川, 若尾, 小貫, 笠原: 有限要素法と境界要素法を利用した電磁界解析の並列処理, 第52回情報処理学会全国大会講演論文集, pp. 6–135 – 6–136 (1996).

[14] Bailey, D., Barszcz, E., Barton, J., Browning, D., Carter, R., Dagum, L., Fatoohi, R., Fineberg, S., Frederickson, P., Lasinski, T., Schreiber, R., Simon, H., Venkatakrishnan, V. and Weeratunga, S.: THE NAS PARALLEL BENCHMARKS, Technical Report BNR-94-007, NASA Ames Researce Center (1994).