計算機アーキテクチャ 139－21
（2000．8．4）

マルチプロセッサ・システムに於けるスケジューリング支援ハードウェアのシミュレーション評価

佐々木 敬泰　西村 直己　弘中 哲夫　吉田 典可

広島市立大学大学院 情報科学研究科

並列処理環境の性能を引き出す上で問題となる様々なレイテンシを隠蔽し，かつ移植性のあるプログラムを記述することは重要である．この実現方法の1つとして細粒度並列処理がある．しかしながら，細粒度並列処理を従来のOSを用いて実行するには問題がある．何故なら，細粒度になるに従いコンテキスト・スイッチやスケジューリングの回数が増えるため，それらに起因するオーバヘッドが増加し，大幅な性能低下を招く危険性があるためである．そこで，本稿ではスケジューリング支援ハードウェア(Scheduling Support Hardware;SSH)を用いたマルチプロセッサ・アーキテクチャを提案する．これは，OSの機能の一部である，スレッドのスケジューリング，CPU資源の割り当て/解放の機能をハードウェアで支援することで，細粒度な並列性を有効利用し，かつ高速なコンテキスト・スイッチやスケジューリングの実現を目指すものである．また，本稿では，Verilog-HDLにてSSHを用いたマルチプロセッサを設計し，シミュレーションにより性能評価を行っている．

Scheduling Support Hardware for Multiprocessor System and its Evaluations

Takahiro Sasaki　Naoki Nishimura　Tetsuo Hironaka　Noriyoshi Yoshida

Hiroshima City University, Graduate School of Information Sciences

This paper proposes a methodology in order to exploit fine grain parallelism effectively by introducing the scheduling support hardware(SSH). It is important to make a program that exploits the full hardware performance, and also has portability among different multiprocessor architectures on the same time. Although in order to hide various types of latency, most tuned programs, that exploit high performance of the multiprocessor architecture, depend on its hardware architecture. Fine grain parallelism, in which a program is decomposed into many fine grain threads for parallel execution, is one solution for this problem. Fine grain parallelism achieves high performance, but it has a problem. As the grain becomes finer, scheduling and context switching are needed more frequently, which may affect the performance. This paper proposes the multiprocessor systems with scheduling support hardware to reduce above scheduling overhead. This paper describes its details and performance evaluation by simulation with Verilog-HDL.

1　はじめに

細粒度並列処理は並列処理環境をより有効に利用でき，また，メモリ・アクセス・レイテンシやI/Oレイテンシ，ネットワーク遅延等の差を吸収できる可能性を持っている．更に，粗粒度並列処理よりも多くの並列性を利用できるため，大規模並列計算機の有効利用も可能となる．細粒度並列処理を利用することで，より効率的な並列処理を実現することが可能であるが，従来のマルチプロセッサ環境，およびOSを用いて細粒度並列処理を行った場合，スケジューリング・オーバヘッドが増加し，逆に性能が劣化する危険性がある．これは，スレッド数が増加することにより，1) スケジューリングすべきスレッド数が増加し，その結果スケジューリング時間が増加する，2) スレッド1つ当りの処理時間が短くなるため，スケジューリング時間の割合が相対的に大きくなり，スケジューリング時間の増加が性能に与える影響が大きくなるためである．

本研究では，この問題を低減するため，従来の共有メモリ型マルチプロセッサ構成にスケジューリング専用のハードウェアを導入することで効率的な細粒度並列処理の実現を目指す．具体的には，一般的な対称型マルチプロセッサ(Symmetric Multipro-

cessor;SMP) に，スケジューリング支援ハードウェア (Scheduling Support Hardware;SSH)[1, 2, 3, 4] を導入し，CPU の動作と並行してスレッドのスケジューリングを行うことで，スケジューリング時間の隠蔽を行う．更に，CPU にプリフェッチ・ユニットを付加することで，スケジューリングした結果を予め取得することが可能となる．これにより，従来の SMP 型マルチプロセッサに大きな変更を加えることなく，細粒度並列処理を利用できる環境を目指す．

本稿は，次のような各節からなっている．まず，次節で先行研究の概括として代表的なスケジューラについて述べる．第3節では，本稿で提案している SSH の原理，およびその詳細について述べ，第4節では試作設計した SSH を用いたマルチプロセッサ環境について簡単にふれる．そして，第5節では設計に用いた Verilog-HDL のコードを用いた評価を行い，最後に，第6節で結論と今後の展望を述べる．

2 先行研究

マルチプロセッサ用のタスク・スケジューリング方式に関する研究は広く行われているが，その多くは OS やマルチスレッド・ライブラリ等，ソフトウェアでタスクのスケジューリングを行うものである．また，タスク・スケジューリング，あるいは OS 全体のハードウェア化による高速化の研究も行われているが，その多くはユニ・プロセッサを対象としている．以下に，スレッドを用いた並列処理に関する代表的な先行研究についてまとめる．

2.1 STRON のハードウェア・スケジューラ

仲野らによる STRON[5, 6] は，リアルタイム OS である μITRON の一部をハードウェア化することで高速化を目指したものである．STRON では，OS の支援を行うハードウェアを導入することで高速化を行っている．しかしながら，導入しているハードウェアはハードウェアで実現したサブルーチンであり，CPU と協調処理を行うわけではない．一方，本稿で提案している手法は，CPU で実行されるソフトウェア部分と専用ハードウェアで実行される処理が並列に動作する点が異なる．これにより，ハードウェアによる高速化だけでなく，処理時間の隠蔽も可能にしている．

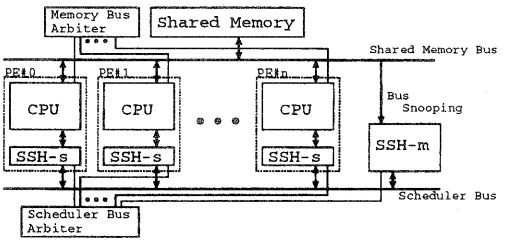


図 1: SSH を用いたマルチプロセッサ環境

2.2 飯田らのスレッド制御ハードウェア

飯田らのスレッド制御ハードウェア [7] も，STRON と同様，ハードウェアを用いたスケジューラである．

しかし，飯田らの研究では，スケジューラの実装に FPGA を使用することを想定している．これは，スケジューリング・ポリシを FPGA のリコンフィグレーションにより実現しているためである．一方，本稿で提案している手法では，予め FIFO やラウンド・ロビン等のスケジューリング・ポリシを実装しているため，ハードウェアの実現方法は FPGA に限らず，ASIC やフルカスタム LSI でも実現可能である．また，飯田らの研究では，実行時のスケジューリング・ポリシの変更を許可していない．更に，全スレッドが同一のスケジューリング・ポリシに従ってスケジュールされる．つまり，飯田らの研究は用途を絞ることにより高速化を目指している．これに対し，本研究ではスレッドのみでなく，プロセスを含めたスケジューリングを可能対象としており，UNIX 等の汎用的な OS を用いたマルチプロセッサ環境の高速化を目指している．

3 スケジューリング支援ハードウェア (SSH)

3.1 SSH を用いたマルチプロセッサ環境

図 1に SSH を用いたマルチプロセッサ環境を示す．提案するマルチプロセッサ環境は，複数の PE(Processing Element)，SSH-m(SSH-master)，共有メモリ (Shared Memory)，メモリ・バス調停器 (Memory Bus Arbiter)，スケジューラ・バス調停器 (Scheduler Bus Arbiter) から成る．メモリ・アーキテクチャとしては，集中あるいは分散共有メモリを想定しており，同一のアドレス空間を共有しているものとする．これは，スレッドを容易に任意の PE に割り当て可能にするためである．

PEは，以下の2つのユニットで構成される．すなわち，プログラムの実行を行うCPUと，SSHの一部であるSSH-s(SSH-slave)である．SSH-sはCPUとSSH-mのインタフェースを提供するもので，CPUの動作と並行して次に実行されるスレッドの取得，新規スレッドのキューイング等を行う．CPUとSSH-sはオンチップでの実装を想定している．

スレッドのスケジューリングは，主にSSH-mで行う．SSH-mでは，SSH-sを介して送信された新規スレッドを管理し，スケジューリング・ポリシに従ってスケジュールする．また，SSH-sから要求があった場合，最も優先度の高いスレッドの管理情報を当該SSH-sに返す．SSH-sとSSH-mは専用のスケジューラ・バスを介して通信する．

3.2 SSHの詳細

SSHの構成要素には，スケジューリング等の処理を行うSSH-mと，CPUとSSH-mとのインタフェースを提供するSSH-sがある．

3.2.1 SSH-m

図2にSSH-mのブロック図を示す．SSH-mはメモリ・バスを監視するBus Snooping Unitとスレッドのスケジューリングを行うHardware Scheduler，および実行可能キュー等を保持するGlobal Queueから成る．

3.2.2 SSH-s

図3にSSH-sのブロック図を示す．SSH-sではCPUとSSH-mのインタフェースを行う．SSH-sは，CPUとのインタフェースを行うCPU-SSH Interface Unit，スケジューラ・バスを介してSSH-mと通信を行うSSH-SSH Interface Unit，およびGlobal Queueから先行してロードしたスレッドやGlobal Queueに登録すべきスレッドをバッファリングするためのRegister Unitから成る．

3.3 ユーザ・プログラムとのインタフェース

SSHは，ユーザ・プログラムから一般的なスレッド・ライブラリの一つであるPthread[8]のAPIを通して制御する．そのため，Pthreadに準拠して記述されたプログラムは，ほぼ変更なしに提案アーキテクチャ上で実行できるという特徴を有している．SSH制御用ライブラリは，C言語，およびMIPS R3000のアセンブラ言語を用いて記述している．

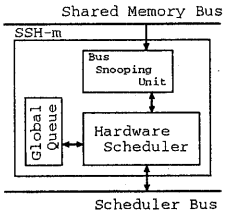


図2: SSH-mのブロック図

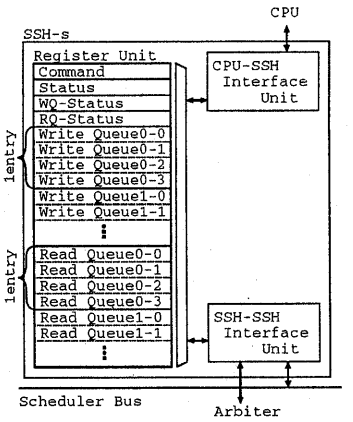


図3: SSH-sのブロック図

4 SSHを用いたマルチプロセッサ環境の設計

本研究では，図1に示したマルチプロセッサ環境のプロトタイプ設計を行った．CPU，および周辺回路を含め，マルチプロセッサ環境の設計はすべてVerilog-HDLを用いて行った．CPU部分には，MIPS R3000のサブ命令セットを持つRISCプロセッサを用いた．除外した命令は，乗算／除算器および一部のコプロセッサ命令である．

また，現在SSH-mで行うデータ・バスのスヌーピングは実装していない．スケジューリングの手法は，ソフトウェアのアルゴリズムをそのままハードウェア上に実装した．

5 性能評価

本節では，SSHの有効性を示すため，設計に用いたVerilog-HDLのビヘイビア記述を用いて，シミュレーションによる性能評価を行う．本研究では，CPU，SSHを含めたマルチプロセッサ環境の設計，評価，および検証をすべてVerilog-HDLを用いて行っている．そのため，Verilog-HDLによるシミュ

表 1: シミュレーション評価環境

WS	Sun Ultra-60 (2CPU)
CPU	UltraSPARC-II 360MHz
Memory	1GB
Verilog-HDL Simulator	Cadence Verilog-XL 2.7

表 2: 使用したプログラム開発ツール

用途	ツール名	開発元
C Compiler	LCC Ver 4.1	AT&T
Assembler	GNU assembler Ver 2.9.1	GNU
Linker	GNU ld Ver 2.9.1	GNU

レーション評価結果は，実機を用いた場合の動作結果と完全に等しくなる．

評価は，一般的な並列処理プログラムをモデル化したものを取り上げ，SSH を用いて実行した場合と，ソフトウェアのみを用いて実行した場合との比較を行い，SSH の有効性を示す．以降，本稿では，SSH を用いた方式を「SSH」，従来通りソフトウェアのみで実装した場合を「Software」で表す．

5.1 評価環境

評価は Verilog-HDL シミュレータを用いて行う．評価環境を表 1に示す．評価は RTL レベルの記述を用いて行う．また，評価プログラムは C 言語を用いて作成する．本研究の特徴の 1 つとして，一般的なスレッド・ライブラリの 1 つである Pthread の API に準拠しているということがある．そのため，本研究で使用した評価用プログラムは，Pthread を実装している環境であればコンパイル，および実行することが可能である．

また，設計したプロセッサは，MIPS R3000 のサブ命令セットを実装している．そこで，C 言語を用いて作成した評価用プログラムをクロス開発環境を用いて，Ultra-60 上でコンパイルし，シミュレーションに利用する．表 2に使用したクロス開発環境を示す．

5.2 対象とするマルチ・プロセッサ環境

マルチプロセッサ環境は，図 1のものを対象とする．PE は 1 ～ 16 台までの任意の台数に設定できる．Memory Bus は全 PE から等距離で，アクセス時間は均一であり，またバスの優先順位は全 PE とも等しい．Scheduler Bus も Memory Bus と同様にラウンド・ロビンにより優先順位が決定される

```
main(){
   初期化処理 [Init]
   スレッド Task() 生成 [Create × Thr_num]
   スレッドの結合 [Join × Thr_num]
   終了処理 [Term]
}

void Task(){
   並列処理 [Thr_load / Thr_num]
}
```

図 4: 並列処理モデル

表 3: 評価パラメータ

パラメータ	意味
PE	利用できる PE 数
Thr_num	生成するスレッド数
$Init$	初期化処理に要するサイクル数
$Create$	スレッド 1 つ生成するのに要するサイクル数
$Join$	スレッド 1 つと結合するのに要するサイクル数
$Term$	終了処理に要するサイクル数
Thr_load	Task() を逐次実行した場合に要するサイクル数 *1

*1 スレッド 1 つあたりのサイクル数は $\frac{Thr\_load}{Thr\_num}$ となる

が，SSH-mからの要求には最も高い順位の優先度が割り当てられている．命令メモリは，各 PE が専用のメモリを持っている．これは，全 PE がヒット率 100% の命令キャッシュを持っているのと等価である．メモリ・バスは 32 ビットのアドレス／データ共有型，命令バスは 32 ビットのアドレス／データ分離型である．

5.3 評価内容

SSH の性能を評価するために，図 4の様な並列処理モデル，および表 3のパラメータを用いてシミュレーション評価を行う．図 4中の鉤括弧内は，処理に要するサイクル数を示す．プログラムは「初期化処理」を行った後，スレッド Task() を Thr_num 個生成する．Pthread には，スレッドを複数個同時に生成する方法はないため，ループ文を用いて逐次的にスレッドの生成を行う．親スレッドは，生成した順番に子スレッドを回収していき，全スレッドを回収した後，「終了処理」を行ってプログラムを完了する．本研究では並列処理部分のみの高速化に関する研究であるため，逐次処理部分，すなわち，$Init$, $Term$ は 0 サイクルとする．処理に要するサイクル数の内，$Create$ と $Join$ は SSH の実装方式，およ

表 4: *Create*, *Join* のオーバヘッド

パラメータ	サイクル数	
	SSH	ソフトウェア
Create	690	724
Join	841	1,180

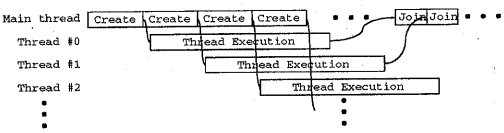


図 5: タスク・グラフ

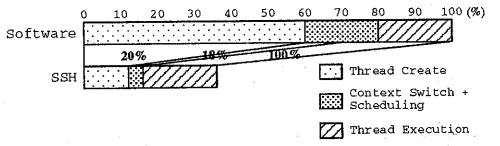


図 6: 各処理にかかる総サイクル数の割合

びライブラリに依存する．本研究で実装したライブラリでは，SSH, Software はそれぞれ表4に示す値となった．図5に評価プログラムのタスク・グラフを示す．

このモデルを用いて，PE 数，およびスレッド数に対する実行時間を計測する．なお，本評価では処理の実行時間を示す指標としてクロック・サイクル数を用いる．

5.4 評価結果 1:SSH 導入による効果の確認

SSH による処理時間の短縮率を見るために，各処理に要するサイクル数を比較する．このサイクル数とは，各処理に全 PE で費やされるサイクル数の総和である．図6に，$Thr\_num = 16$, $Thr\_load = 30000$, $PE = 8$ の場合における，Software の処理全体を 100% としたときの各処理にかかる総サイクル数の割合を示す．同図より SSH を用いた場合，Software と比較して，スレッドの生成のサイクル数が 20% に，コンテキスト・スイッチのサイクル数が 18% に減少しているのがわかる．

次に SSH，および Software がどの程度のスレッド粒度に適しているかを調べるために，スレッド当りのサイクル数に対する性能評価を行う．図7に $PE = 8$, $Thr\_load = 40$ に固定した上で，スレッド当りのサイクル数を変化させた場合の逐次実行に

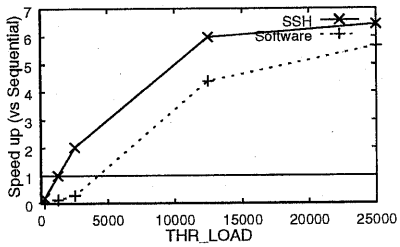
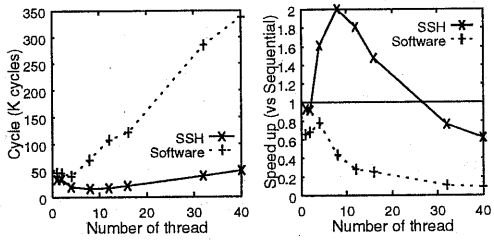


図 7: スレッド当りのサイクル数に対する性能の変化



(A) サイクル数　　(B) 逐次処理に対する性能比

図 8: 一般的な並列処理モデル（スレッド数に対するサイクル数）

対する性能比を示す．図7より，SSH ではスレッド当りのサイクル数が 1,250 サイクル以上で並列化の効果が得られるのに対し，Software では 5,000 サイクル以上なければ並列化の効果が得られないことが確認できる．このことから，SSH は Software と比較して，より細粒度の並列性を利用できることがわかる．

5.5 評価結果 2: スレッド数に対する処理時間の変化

次に $PE = 8$ に，$Thr\_load = 30,000$ に固定した上で，スレッド数 Thr_num に対するサイクル数の変化を評価する．この時のスレッド当りのサイクル数は，$30000/Thr\_num$ となる．図8に評価結果を示す．同図 (A) は処理に要したサイクル数を，(B) は逐次実行した場合に対する性能比を示す．また，同図 (A) は横軸がスレッド数，縦軸がサイクル数を，(B) は横軸がスレッド数，縦軸が逐次実行の場合に対する性能比を表す．同図 (A), (B) より，SSH, Software とも Thr_num が増加する，すなわち，より細かくプログラムを分割するに従い，処理に要するサイクル数が短くなっているが，ある程度以上になると逆にサイクル数が長くなっている．これは一般に，並列処理で粒度を細かくし過ぎた場合に起こる現象で，並列化に伴うオーバヘッドが，並列化による性能向上よりも大きくなるこ

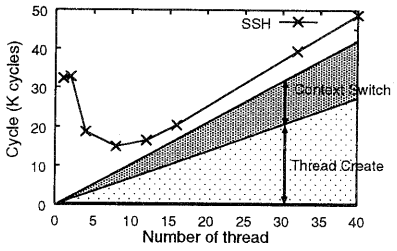


図 9: 一般的な並列処理モデル（スレッド数に対するサイクル数：SSH のみ）

とに起因する．ここでSSH に注目してみる．図9にSSH の場合のスレッド数に対するサイクル数の変化を示す．同図よりスレッド数8の時に最も良い性能を得ており，それ以上スレッドが増加すると，逆に処理時間が増加していることがわかる．このグラフにスレッドの生成にかかる時間の合計（図中の“Thread Create”）を重ねてみると，性能低下の原因としてスレッド生成のオーバヘッドの影響が大きいことがわかる．粗粒度並列処理では，$Create \ll (Thr\_load/Thr\_num)$ となるため，スレッド生成のオーバヘッドは大きな問題にならないが，細粒度並列処理ではスレッドの実行時間とスレッドの生成時間が同程度になるため，スレッド生成のオーバヘッドの増加は大幅な性能低下につながる危険性がある．

ここで，スレッドの生成時間に加え，コンテキスト・スイッチに必要な時間を図9に重ねてみる（図中の“Context Switch”）．このコンテキスト・スイッチの時間は，CPU リソースの退避，復帰の時間であり，スケジューリングの時間は含んでいない．図9より，スレッド数40 の場合の総処理時間の内，実に87% がスレッド生成，およびコンテキスト・スイッチに費やされていることがわかる．これより，細粒度並列処理においては，スレッド生成の時間，およびコンテキスト・スイッチの時間を小さくすることが重要であることがわかる．

以上より，より高速な細粒度並列処理を実現するには，スレッド生成の時間，およびコンテキスト・スイッチの時間を0サイクルに近づける必要がある．

6 結論と今後の展望

本稿では，従来のSMP 型マルチプロセッサにスケジューリング支援ハードウェア (SSH) を導入することで，細粒度並列処理を有効に利用可能なシステムの構築を行った．従来のソフトウェアのみで実現したスレッド・ライブラリでは，スレッド数，プロセッサ数の増加にともない，性能が著しく劣化するという問題があった．そのため，細粒度並列処理への適用が困難であった．本稿で提案しているSSH を導入することで，スレッドのスケジューリングを並列に，かつ高速に行うことができ，細粒度並列処理においても高速な処理を実現できる．第5節で行った評価によると，PE 数が8，スレッド数が40 という条件において，ソフトウェアのみで実装した場合ではスレッド当りのサイクル数が5,000 サイクル以上なければ並列化の効果が得られないのに対し，SSH では1,250 サイクル以上あれば並列化の効果が得られることがわかった．

今後の展望としては，次のことが考えられる．すなわち，(1) スレッド生成時間の更なる短縮，(2) コンテキスト・スイッチのハードウェア支援，(3) より大規模なマルチプロセッサ環境での評価である．

参考文献

[1] Sasaki, T., Hironaka, T., Nishimura, N. and Fujino, S.: Microprocessor LSI with Scheduling Support Hardware for Operating System on Multiprocessor System, *Thr 6th Asia Pacific Conference on cHip Design Languages (APCHDL'99)*, pp. 67–72 (1999).

[2] 佐々木敬泰，弘中哲夫：スケジューリング支援ハードウェアを混載したマルチプロセッサ・システム，平成11 年度電気・情報関連学会中国支部連合大会講演論文集, pp. 276–277 (1999).

[3] 西村直己，佐々木敬泰，木山真人，弘中哲夫，藤野清次：マルチプロセッサ・システムに於けるスケジューリング支援ハードウェア, 電子情報通信学会技術報告書, Vol. CPSY99–57, pp. 79–86 (1999).

[4] 西村直己, 佐々木敬泰, 木山真人, 弘中哲夫, 藤野清次：スケジューリング支援ハードウェアとその評価環境の構築，第3 回システムLSI 琵琶湖ワークショップ，pp. 247–250 (1999).

[5] 仲野巧, 小松平良樹, 塩見彰睦, 今井正治：リアルタイムOS チップの性能評価と分散OS への拡張，電子情報通信学会技術報告書, Vol. CPSY–51, pp. 23–30 (1998).

[6] Nakano, T., Komatsudaira, Y., Shiomi, A. and Imai, M.: Performance Evaluation of STRON: A Hardware Implementation of a Real-Time OS, *IEICE Transactions*, Vol. E82-A, No. 11, pp. 2375–2382 (1999).

[7] 飯田全広，久我守弘，末吉敏則：マルチスレッド制御ライブラリのハードウェア化によるオンチップ・マルチプロセッサの構成，並列処理シンポジウム（JSPP'97 IPSJ Symposium Series), pp. 337–344 (1997).

[8] Lewis, B. and Berg, D.: マルチスレッドプログラミング入門，アスキー出版局 (1996).