SMP型計算機を活用する軽量プロセス・ライブラリ

小熊 寿[†] 海江田 章裕[†] 森本 浩通[†,☆]
田村 友彦[†,☆☆] 鈴木 貢[†] 中山 泰一[†]

近年，SMP型計算機がより身近なものとなった．このようなアーキテクチャを活用する並列処理機構として，軽量プロセス（スレッド）が注目されている．筆者らは現在，SMP型計算機を有効に活用し，特定のOSに依存しないスレッド・ライブラリの設計・実現を行っている．本ライブラリでは，スレッドを並列に動作させるために，複数のUNIXプロセス（仮想プロセッサ）を生成する．実行されるスレッドのコンテクストは，すべての仮想プロセッサから見える巨大な共有メモリ領域に保存する．巨大な共有メモリ領域の確保には，メモリマップト・ファイルを用いる方法を提案する．実験により，移植性の高さとCPU台数に見合った性能向上が確認された．

Design and Implementation of a Light-weight Process Library on SMP Computers

Hisashi Oguma,[†] Akihiro Kaieda,[†] Hiroyuki Morimoto,[†,☆]
Tomohiko Tamura,[†,☆☆] Mitsugu Suzuki[†]
and Yasuichi Nakayama[†]

Recently, SMP computers have been commonly used. In this paper, we present a design of the light-weight process (thread) library for SMP computers. For this library, we require portability, low-cost thread generation and switching, and efficient parallel execution of threads. We assume that the operating system can executes UNIX processes in parallel on multiple processors. Our thread library creates UNIX processes as virtual processors, which execute user-level threads. Our library also creates a memory-mapped file as the large shared memory space, in which thread contexts are saved. We have implemented a new thread library on SMP computers. Experimental results confirm that our thread library satisfies the above-mentioned requirements.

1. はじめに

並列計算機は近年，パーソナル・コンピュータ（PC）でも実現されるようになり☆☆☆，より身近なものとなってきた．これによって，SMP型計算機（symmetric multiprocessor）と呼ばれる対称型の密結合型並列計算機に対応したUNIX（たとえばFreeBSDやLinux）が提供されるようになった．

SMP型計算機は，すべてのプロセッサがシステムのすべての資源（メモリやI/Oなど）を平等に扱えるため，非対称型で問題となるI/Oや割込みハンドリングといったOS機能のボトルネックを起こしにくい．SMP対応型のUNIXでは，そのようなハードウェアの特性を活かして，OS機能の分散化を実現している．また，UNIXプロセスの各プロセッサへの割当てを工夫することにより，UNIXプロセスレベルでの並列処理を実現し，システム全体としての性能を高めている．

このようなSMPに対応したUNIX上で，効率的に並列処理をサポートする機構を実現することが急務とされる．とくに，UNIXプロセスより細かい実行単位（軽量プロセスまたはスレッドと呼ばれる）による処理方法が必要とされている．

スレッドには，カーネルに手を加えることを必要とするもの（たとえばScheduler Activations[3]）と，ユーザ・レベルのみで実現するもの（たとえばPTL[1]）と

† 電気通信大学電気通信学部情報工学科
Department of Computer Science, The University of Electro-Communications
☆ 現在，ソニー株式会社パーソナルAVカンパニー
Presently with Sony Corporation
☆☆ 現在，株式会社富士通ターミナルシステムズ第二開発統括部
Presently with Fujitsu Terminal Systems Limited

☆☆☆ Intel社が公開した仕様（MultiProcessor Specification Version 1.4 July 1995）による．

がある．カーネル・レベルでスレッドを実現すると，計算機アーキテクチャに適合したシステムの構築が可能であるが，移植性を損なうという欠点がある．

それに対しユーザ・レベルのみで実現するもの，すなわちスレッド・ライブラリは，アーキテクチャやOSに依らずに利用できるという利点がある．

また，ユーザがスレッドを取り扱ううえで，ユーザとスレッド・ライブラリとの間にインタフェースが必要になる．OS 間で移植性を持つインタフェースにPOSIX という規格が存在する．この規格の中では，スレッドのインタフェースについても定められている．この POSIX で定められたユーザ・インタフェースは一般に Pthread と呼ばれている[6]．

スレッド・ライブラリの構成方式に関しては，これまで様々な研究がなされている[1),5),9),12)]．しかし移植性に優れていて，かつ，複数のスレッドを複数のプロセッサに割当て可能なライブラリはこれまで存在しない．たとえば，PTL[1)]はインターバル割込みによるマルチスレッド・システムを実現しているが，SMP 型計算機上で並列には動作しない．また，LinuxThreads[5)]のように SMP を想定したライブラリでも，Linux でしか動かないために移植性に優れているとはいえない．

本来，SMP 型計算機のような環境上では，マルチ・スレッドによる並列実行処理が可能となり，CPU 台数に見合った時間短縮が可能となるはずである．

本論文では，SMP 型計算機を活用するために，並列に動作し，移植性にも優れたスレッド・ライブラリの設計・実現について述べる．ライブラリは，UNIX が提供するシステム機能のみを使用している．

スレッドを並列に動作させるために，複数の UNIX プロセス（仮想プロセッサ）を生成する．実行されるスレッド・コンテクストは，すべての仮想プロセッサから見える領域に保存する．スレッド・コンテクストには，レジスタ情報やスレッドが個別に保持するスタック領域も含まれる．数多くのスレッド・コンテクストを保存するためには巨大な共有メモリ領域が必要となる．巨大な共有メモリ領域の確保には，メモリマップト・ファイルを用いる方法を採用した．

本論文では性能評価実験として，従来のスレッド・ライブラリとの処理時間比較を行った．CPU 台数などの実験環境を変化させ実験を行った結果，本論文で提案した方式は，CPU 台数の増加に見合った性能向上を得られた．

以下，2 章ではこれまでに提案されてきたスレッド・ライブラリについて述べる．3 章では本論文で提案するライブラリの設計・実現に関する問題と解決法について述べる．4 章で実現したライブラリの評価・結果について述べる．

2. 従来のスレッド・ライブラリ

多田ら[9),10)]のライブラリは，移植性に優れ，実行の一時中断と再開の制御が可能である．内部で行われているコンテクスト切替えの方法は，後続の研究にも適用されており，この分野では草分け的なライブラリである．

安倍ら[1),2)]による PTL は，BSD UNIX 上で移植性を考慮して実現されたライブラリであり，POSIX に準拠したインタフェースとインターバル割込み機能を持つ．また，動的なスタック領域拡張機能も備えている．

Provenzano[12)]による MIT pthreads は，システム構成が PTL に似通ったものとなっている．スレッドごとのスタック領域の確保および切替えに多田らの方式を採用することで，移植性が高まっている．

Leroy[5)]による LinuxThreads は，SMP 型計算機を想定してるため，スレッドの並列実行が可能となっているライブラリである．しかし，対象 OS として Linux のみを想定しているために移植性がない．

各スレッド・ライブラリを比較すると，**表 1** のようになる．単一プロセッサを想定したスレッド・システムは，CPU 台数に見合った性能向上が得られず，SMP 型計算機を活用することはできない．その理由は，以下のとおりである．

単一プロセッサを想定した従来のスレッド・システムは**図 1** のようになっている[4),13)]．単一の UNIX プロセスが，スレッド実行のための仮想プロセッサとして使われる．

従来のスレッド・ライブラリを SMP 型計算機上で動かすと，**図 2** のようになる．たとえ，カーネルが CPU 台数分だけの UNIX プロセスに対して CPU を割り当てることで UNIX プロセスの並列実行が可能であるとはいっても，スレッド・システムを実行する仮想プロセッサは 1 つしか存在しない．したがって，スレッドが並列に実行されることはない．

表 1 各スレッド・ライブラリの比較
Table 1 Comparison of the thread libraries.

	移植性	POSIX	SMP 活用
多田ら	○	—	×
PTL	△†	○	×
MIT pthreads	○	○	×
Linux Threads	×	○	○
本論文	○	○	○

† PTL は，SVR4 系 UNIX では動作しない．

このように単一プロセッサに対するマルチスレッド・システムにより，プログラミングのしやすさと，UNIXプロセスと比較して生成・切替え速度の向上が図れたが，それ以上のことは期待できない．このままのシステムをSMP型計算機上で動かしても，CPU台数に見合った性能向上は見込めない．

本論文では，今までのスレッド・ライブラリの利点を活かしつつ，SMP型計算機を活用するための方法を提案し，これらの条件を満たしたスレッド・ライブラリを設計する．

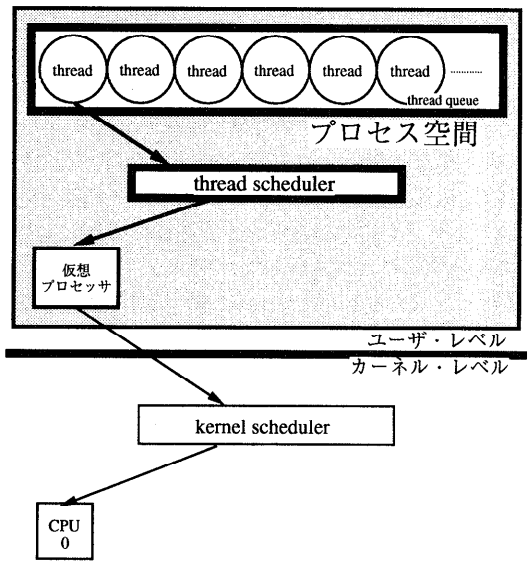


図1 従来のシステム構成
Fig. 1 Previous thread library model.

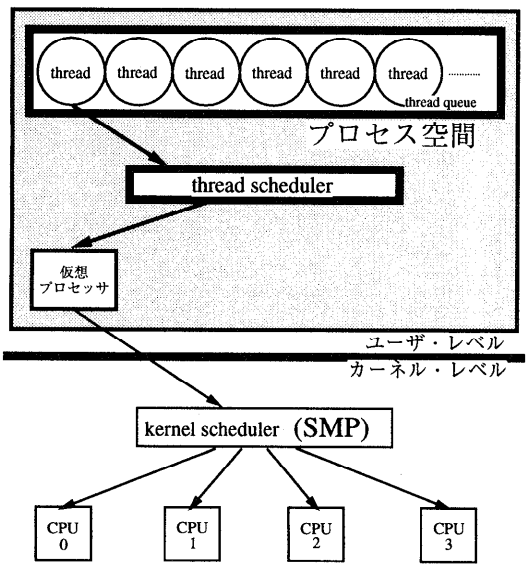


図2 SMPで動かした場合
Fig. 2 Previous thread library model on SMP machine.

3. 設計方針

SMP型計算機上でのスレッド・ライブラリの設計について，概要と問題点，そして問題点の解決方法について述べる．

3.1 システムの概要

本論文では，**図3**に示す構成のシステムを提案する．

スレッドを並列に動作させるために，複数のUNIXプロセス（仮想プロセッサ）を生成する．実行されるスレッドのコンテクストは，すべての仮想プロセッサから見える領域に保存する．すなわち，すべての仮想プロセッサはスレッドのコンテクストを共有しているので，任意のスレッドを実行できる．本論文で提案するシステムがSMP型計算機上で実現されれば，各UNIXプロセスが同時に複数のプロセッサに割り当てられ，プロセスが並列に動く．これによってスレッドが並列に実行される．また，すべての仮想プロセッサから見える共有メモリ領域は，スレッド数が動的に変化しても領域の拡張を必要としないだけの大きさを，初期化時にあらかじめ確保しておく．

3.2 実行機構に関する要求

上記のシステムを実現する際には，

- 移植性
- インターバル割込みによるコンテクスト切替え
- 標準的なユーザ・インタフェース

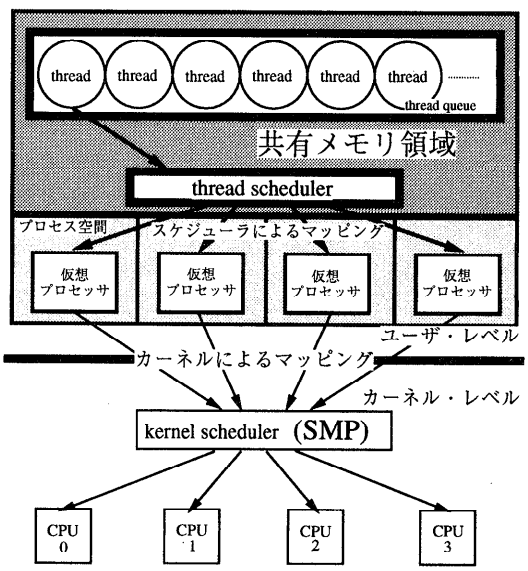
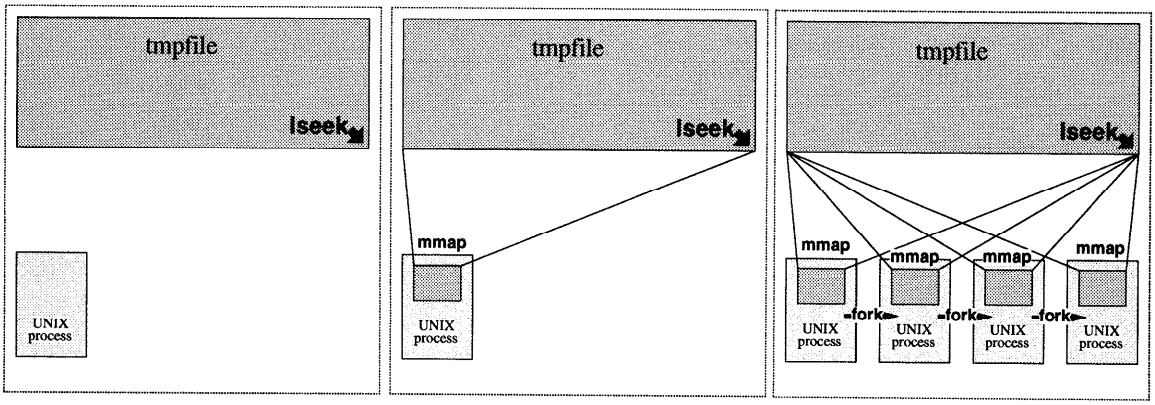


図3 本研究のシステム構成
Fig. 3 Our thread library model.



開いたファイルに対して，lseek でファイル内のポインタを進めて，仮想的な巨大ファイルをつくり出す．

mmap でメモリ空間に巨大なファイルをマッピングする．

新たに生成した仮想プロセッサからも，メモリマップト・ファイルは共有できる．

図4 巨大な共有メモリ領域の生成方法
Fig. 4 Huge shared memory space.

- 巨大な共有メモリ領域
- 仮想プロセッサ間での相互排除

が要求される[7]．

以下，本論文では，移植性と巨大な共有メモリ領域を中心に，その実現法について述べる．

3.3 移 植 性

ライブラリは，特定のアーキテクチャや OS に依存しないことが望ましい．そのために本論文で設計したライブラリは，すべて C 言語で記述し，かつ，UNIX が標準的に提供する機能のみを用いて実現した．コンテクスト切替えは，多田ら[9),10)]と同様に，setjmp 関数と longjmp 関数を用いて行うこととした．

本論文のライブラリは，UNIX プロセスの並列実行ができる OS であれば動作する．また，次節で述べるようなメモリマップト・ファイルの機能を必要とする．

実現したライブラリで OS に依存せざるをえない部分は，C 言語のコードにして 2，3 行しかない．また，新しいアーキテクチャや OS に対しても，容易に移植が可能と考えられる．

3.4 巨大な共有メモリ領域の実現法

各仮想プロセッサがスレッドを任意に実行するために，本論文のライブラリでは図 3 で提案したように，巨大な共有メモリ領域（たとえば 1 G バイト程度の共有メモリ領域）を用意し，すべてのスレッド・コンテクストを共有メモリ領域内に保存する．スレッド・コンテクストには，レジスタ情報やスレッドが個別に保持するスタック領域も含まれる．数多くスレッド・コンテクストを保存するためには巨大な共有メモリ領域が必要である．巨大な共有メモリ領域を実現するために本論文では，メモリマップト・ファイルを採用する．

一般に，共有メモリ領域を生成する方法として，shmget，shmat システム・コールを用いる方法は容易に思いつく．しかしこの方法では，本論文が対象とするシステムに必要不可欠な「巨大な共有メモリ領域」が生成できない．

メモリマップト・ファイルは，巨大な領域を実現するためには適した方法といえるが，ファイルの生成コストが大きくなることが心配される．しかし，本論文で提案する以下の手順で共有メモリを生成することで，生成に要するコストを小さくできる（図 4）．これにより，移植性が高くかつ高速に，巨大な共有メモリ領域が生成できる．

- open システム・コールで一時的なファイルを開く．
- lseek システム・コールで，仮想的に巨大なファイルにする（最後の 1 ブロックだけは write システム・コールを用いる）．
- mmap システム・コールでファイルと仮想メモリ空間の間でマッピングを行う．

この方法は，仮想的に巨大なファイルを作っただけでファイルの実体は小さいものとなる．本論文のライブラリでは，初期化時に論理空間の上限値まで，共有メモリ領域を生成している．そのために，共有メモリ領域を動的に拡張する必要がない．本論文で提案する手順を踏むことにより，shmget システム・コールを用いた共有メモリ領域と比べて，その生成コストに差異がないと予想される．これについては，次章の実験で検証する．

4. 評 価

本論文のライブラリは，現在，表 2 に記された OS

表2 動作確認したOS
Table 2 List of operating systems on which our system has been checked to work as designed.

OS名	タイプ	特徴
SunOS 4.1.4	BSD UNIX	単一プロセッサのみ対応
SunOS 5.3	SVR4 UNIX	SMPに対応
FreeBSD 2.2	BSD UNIX	単一プロセッサのみ対応
FreeBSD 3.0	BSD UNIX	SMPに対応
Linux 2.0	SVR4 UNIX	SMPに対応

表3 スレッド生成などの時間（マイクロ秒）
Table 3 Execution time of basic thread operations (μsec).

	shmget	mmap
初期化	1435	3984
スレッド生成	192	240
スレッド切替	188	164

上での動作を確認している．ライブラリはすべてC言語で記述し，かつ，UNIXが標準的に提供する機能のみを用いているため，それぞれの環境においてコード記述の変更を2，3行程度しか必要としない．

4.1 共有メモリ領域の生成法とコスト比較

まず，システム内部の共有メモリ領域を，メモリマップト・ファイルで実現したときと，一般的な方法として知られている **shmget** システム・コールで実現したときとの2つの場合で，性能の違いを検討する．

実験機は2台のPentium Pro（150 MHz）CPUと，64Mバイトのメモリを持つSMP型計算機を用い，OSとしてSMPに対応しているFreeBSD 3.0を用いた．

表3 は，システムの初期化，スレッド生成，スケジューリングにかかる時間をシステム内部の共有メモリ領域の構成方法ごとに分けた表である．

きわだった違いはシステムの初期化にかかる時間である．**shmget** では約1.4ミリ秒，メモリマップト・ファイルでは約4.0ミリ秒となる．しかし，初期化は1度しか行われないため，大規模なアプリケーション・プログラムであれば実行時間に影響しない．その他の項目については，それほど目立った違いが見られない．

次に，性能評価プログラムとして，正方行列の2乗計算を実行させた（**図5**）．横軸は行列の要素数，縦軸はプログラムの総実行時間とする．

shmget システム・コールを利用するとき，共有メモリ領域には大きさに制限がある（実験環境では，3Mバイトまでしかとれない）．そのためこの実験では，256次正方行列（要素数65536）までしか計測ができない．256次までで双方を比べてみると，目立った違いは見られずにほぼ同性能であるといってよい．メモリマップト・ファイルで共有メモリを生成する方法は，ギガバイト規模の大きい領域をとることができ，優れているといえる．

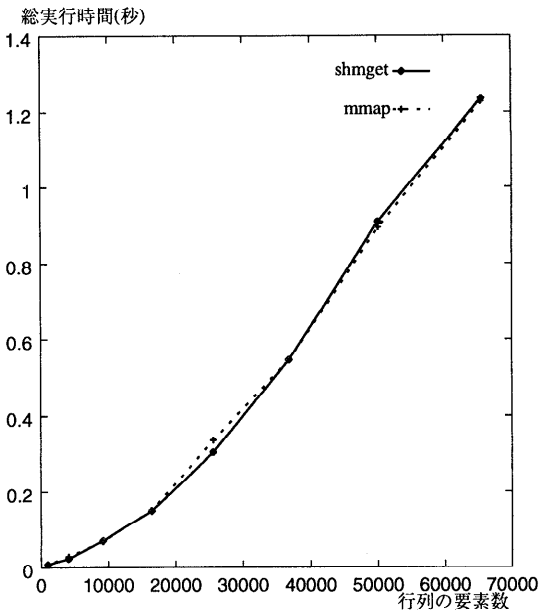


図5 shmgetとmmapの総実行時間比較
Fig. 5 Comparison of turnaround time between shmget and mmap.

この結果より本論文では，メモリマップト・ファイルで巨大な共有メモリ領域を採用することにした．

4.2 PTLとの実行時間比較

本論文のライブラリの性能を評価するうえで，PTLを比較対象とした．

実験環境は前節と同じ2台のPentium Proを持つSMP型計算機を用い，OSをFreeBSD 3.0とした．性能評価プログラムとして，正方行列の2乗計算を実行させた．

図6 は，性能評価プログラムが生成するスレッド数を4に固定し，問題の大きさを変化させてPTLとの総実行時間の違いを見たものである．

この結果から，本論文で提案した方式はCPU台数に見合った性能が得られることが確認できた．PTLは単一プロセッサでもCPU2台のSMP型計算機でも性能に違いが生じない．これに対し，本システムを単一プロセッサ上で実験を行ったときには，PTLと同程度の性能が得られ，CPU2台のSMP型計算機を使って実行させたときには，スレッドの並列実行による性能向上が現れる．このことより本システムは，SMP型計算機の特徴を活かしたライブラリといえる．

図7 は，行列を512次正方行列（要素数は262144）に固定し，スレッド生成数による総実行時間の違いを

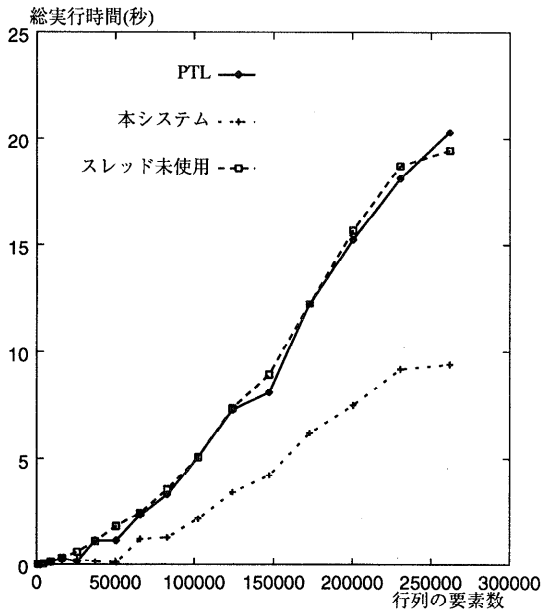


図6 本システムと PTL との総実行時間比較（スレッド数4）
Fig. 6 Comparison of turnaround time between PTL and our system (4 threads).

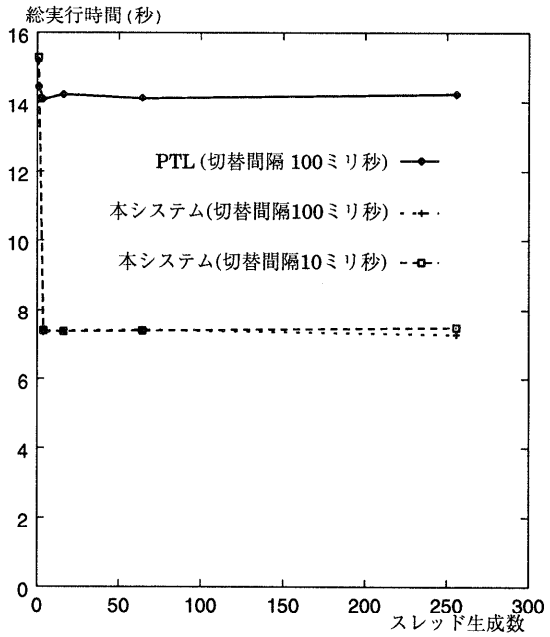


図7 本システムと PTL との総時間比較（要素数一定）
Fig. 7 Comparison of turnaround time between PTL and our system (512×512 matrix).

見たものである．

スレッド総数が多いと，コンテクスト切替えにともなって，ページフォルトによるスワップ・アウトがより多く発生することが心配される．しかし，本システム，PTL ともにスレッド数が増加し，スレッドの粒度が細かくなっても性能が悪化しなかった．また，スレッドの切替え間隔を変化させて実験を行ってみた．スレッド数が256の場合，スケジューラは，切替え間隔100ミリ秒で120回，切替え間隔10ミリ秒で448回起動した．スレッド・システムをユーザ・レベルで実現しているため，スレッド生成と切替えの際に生じるオーバヘッドが小さいことが示された．

続いて，同期変数（Mutex）のロック，アンロックについて実行時間を測定した．複数スレッドが頻繁に同期をとるプログラムのために，スレッド・ライブラリでは，Mutex の制御が高速で実行されることが望ましい．

表4 に示すとおり，少ない処理時間で Mutex の制御ができていることが確認された．

本論文のライブラリでは，仮想プロセッサ間の排他制御にシステム・コールによるセマフォを利用している[7),11)]．PTL との処理時間の差は，システム・コールにかかる時間が加わったことによるものと考えられる．

4.3 台数増による性能向上と同期機構の性能

本節では，実験機を8台の CPU（60 MHz）とメモリを 512M バイト持つ SPARC Server 1000 とし，OS を SMP に対応した SunOS 5.3 として，仮想プロセッサ数を変化させた実験を行った．

表4 Mutex 変数制御にかかる時間（マイクロ秒）
Table 4 Execution time of Mutex control (μsec).

	mutex_lock	mutex_unlock
PTL	36	40
本システム	58	58

図8 は，スレッド数を16に固定し，前節までと同様に正方行列の2乗のプログラムについて，仮想プロセッサ数1～8台で，総実行時間の違いを見たものである．なお，仮想プロセッサが1つのときは，図2のモデルと同じ状態であり，スレッドは並列に動作していない．

仮想プロセッサ数を CPU 台数に近付けるに従って，図8に示すように実行時間が短縮されている．256次元正方行列の計算を仮想プロセッサ8台で解いた場合，スレッド未使用と比べて，約6.7倍の性能向上が得られた．仮想プロセッサ数を CPU 台数に近付けることにより，スレッド実行の並列度が高くなっていることが明らかになった．

次に，スレッドごとで同期操作をともなう問題を，本論文のライブラリで解かせた．アプリケーション内の共有変数に対し，排他的なアクセスによって同期が多く起こるプログラムとして，巡回セールスマン問題

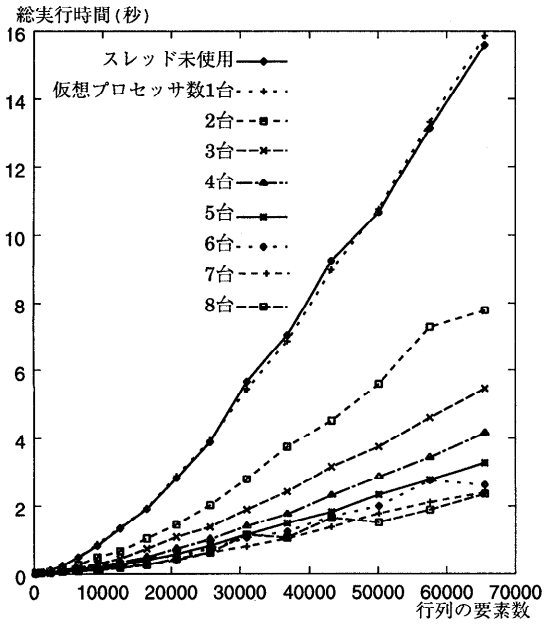


図8 仮想プロセッサ数ごとの総実行時間の違い
Fig. 8 Turnaround time for the number of processors (matrix multiplications).

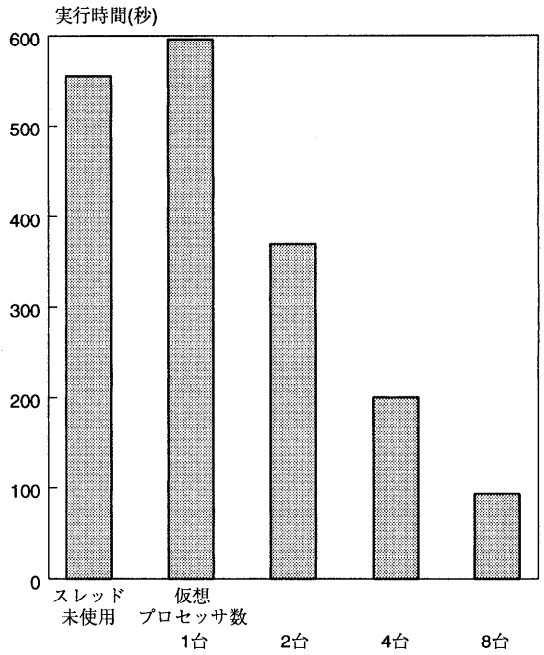


図9 仮想プロセッサ数ごとの総実行時間の違い
(巡回セールスマン問題)
Fig. 9 Turnaround time for the number of processors (Traveling Salesperson Problem).

を用いた．図9では，都市数12，スレッドを121個生成し，仮想プロセッサ数を1，2，4，8として実験を行った結果である．

図9でも仮想プロセッサ数をCPU台数に近付けるほど，実行時間が短縮された．仮想プロセッサ数8台のとき，スレッド未使用と比べて，約6.8倍の性能向上が得られた．

本論文のライブラリでは，仮想プロセッサ間での同期にシステム・コールを用いている．仮想プロセッサ数が増えると，仮想プロセッサ間の同期のオーバヘッドも増加するが，スレッドの並列処理の効果の方がはるかに大きく，CPU台数に見合った性能向上を示せた例といえる．

5. おわりに

本論文ではSMP型計算機を活用するために，スレッドを並列に動作させることが可能で，移植性にも優れたスレッド・ライブラリの設計・実現を行った．SMP型計算機に対応したUNIXでは，UNIXプロセスの並列実行が可能となっている．そこで，本論文では，並列動作できるUNIXプロセスを利用して，スレッドの並列実行を試みた．並列に動作させるために本論文では，UNIXプロセスを複数生成し，それぞれのプロセスでスレッドを処理させるシステム構成を提案した．本論文では，メモリマップト・ファイルを用いて巨大な共有メモリ領域を実現し，そこにスレッドのコンテクストを格納した．これにより，移植性に優れ，スレッドの並列実行が可能なライブラリが実現できた．性能比較実験により，本論文で提案したスレッド・ライブラリは，CPU台数に見合った性能向上が確認できた．

現在，筆者らはこのライブラリを利用して，様々なアプリケーションの移植を行い，スレッド・ライブラリに要求される機能について検討を行っている[8]．今後はこれらのアプリケーションが要求する機能を追加していく予定である．

謝辞 PTLのソースコードを提供していただいた，大阪市立大学学術情報総合センター安倍広多氏に感謝いたします．

本研究を進めるにあたり，貴重なご助言をいただいた電気通信大学情報工学科野下浩平教授，渡邊坦教授，柳井啓司助手に感謝いたします．また，実験環境の構築にご協力くださった中山研究室の各位に感謝します．なお，本研究は一部，文部省科学研究費補助金奨励研究（A）課題番号08780253による．

参考文献

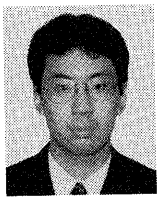
1) 安倍広多，松浦敏雄，谷口健一：BSD UNIX上で

の移植性に優れた軽量プロセス機構の実現，情報処理学会論文誌，Vol.36, No.2, pp.296–303 (1995).
2) 安倍広多：PTL – Portable Thread Library (current 970709), http://www.media.osaka-cu.ac.jp/~k-abe/PTL/.
3) Anderson, T.E., Bershad, B.N., Lazowska, E.D. and Levy, H.M.: Scheduler Activations: Effective Kernel Support for the User-Level Management of Parallelism, *ACM Trans. Computer Systems*, Vol.10, No.1, pp.53–79 (1992).
4) 福田 晃：並列オペレーティング・システム，情報処理，Vol.34, No.9, pp.1139 1149 (1993).
5) Leroy, X.: Linuxthreads – POSIX 1003.1c kernel threads for Linux, http://pauillac.inria.fr/~xleroy/linuxthreads.
6) Nichols, B., Buttlar, D. and Farrell, J.P.: *Pthreads Programming*, O'Reilly & Associates (1996).
7) 小熊 寿，海江田章裕，森本浩通，鈴木 貢，中山泰一：SMP 型計算機を活用する軽量プロセス・ライブラリ，情報処理学会プログラミング研究会報告，97-PRO-16-3 (1997).
8) 鈴鹿倫之，鈴木 貢，中山泰一：MPEG 再生系のマルチ・スレッドによる高速化，電子情報通信学会総合大会講演論文集，D-3-4 (1998).
9) 多田好克，寺田 実：移植性・拡張性に優れたCのコルーチンライブラリーの実現法，電子情報通信学会論文誌，Vol.J73-D-1, No.12, pp.961–970 (1990).
10) 多田好克：機種に依存しない利用者 threads ライブラリ，情報処理学会プログラミング—言語・基礎・実践—研究会報告，92-PRG-8-22 (1992).
11) 田村友彦，森本浩通，海江田章裕，小熊 寿，鈴木 貢，中山泰一：SMP 型計算機を活用する軽量プロセス・ライブラリ—スレッド間同期機構の実現と評価，第 56 回情報処理学会全国大会論文集，2D-05 (1998).
12) Provenzano, C.: Pthreads version 1_70, http://www.mit.edu:8001/people/proven/pthreads.html.
13) Vahalia, U.: *UNIX Internals – The New Frontiers*, Prentice Hall (1996).

(平成 9 年 10 月 31 日受付)
(平成 10 年 7 月 3 日採録)

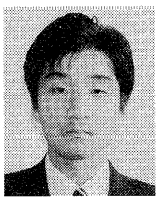
小熊 寿（学生会員）

1973 生．1997 年電気通信大学情報工学科卒業．現在，同大学院電気通信学研究科情報工学専攻博士前期課程在学中．並列・分散処理，ネットワーク，モバイル・コンピューティングに関する研究に興味を持つ．日本ソフトウェア科学会会員．



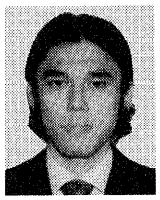
海江田章裕（学生会員）

1974 生．1998 年電気通信大学情報工学科卒業．現在，同大学院電気通信学研究科情報工学専攻博士前期課程在学中．並列プログラミング，システム・ソフトウェアに興味を持つ．



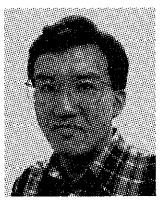
森本 浩通（正会員）

1973 年生．1998 年電気通信大学情報工学科卒業．1998 年 4 月より，ソニー（株）パーソナル AV カンパニー勤務．DVCR のシステム開発に従事．



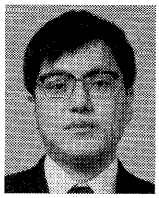
田村 友彦（正会員）

1975 年生．1996 年群馬工業高等専門学校電子情報工学科卒業．1998 年電気通信大学情報工学科卒業．1998 年 4 月より，(株) 富士通ターミナルシステムズ第二開発統括部勤務．IC カード関連ソフトウェアの開発に従事．



鈴木 貢（正会員）

電気通信大学情報工学科助手．記憶管理アルゴリズム，並列アルゴリズム，プログラミング言語処理系等に興味を持つ．ACM，電子情報通信学会，日本ソフトウェア科学会各会員．



中山　泰一（正会員）

1965年生．1988年東京大学工学部計数工学科卒業．1993年同大学院工学系研究科情報工学専攻博士課程修了．工学博士．1993年4月より，電気通信大学情報工学科助手．現在，同学科講師．オペレーティング・システム，並列・分散処理，ゲーム・プログラミングに関する研究に従事．日本ソフトウェア科学会，電子情報通信学会，IEEE，CSA，ICCA等各会員．