プログラミング 16-3
(1997. 11. 21)

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, 鈴木 貢, 中山 泰一
電気通信大学 情報工学科
oguma-h@igo.cs.uec.ac.jp

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。本研究では、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・ライブラリを設計・実現する。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、全てのUNIXプロセスから見える領域に保存する。すなわち、プロセス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。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。

A Study of Light-weight Process Library on SMP Computers

Hisashi Oguma, Akihiro Kaieda, Hiroyuki Morimoto,
Mitsugu Suzuki and Yasuichi Nakayama
Department of Computer Science,
The University of Electro-Communications
Chofu, Tokyo 182, Japan

Recently, SMP computers have been popular. In this paper, we present a design of the light-weight process (thread) library for SMP computers. For this library, we require portability, shorter thread generation and switching time, 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. Partial memory space is shared among all the virtual processors. 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[1]、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や割り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り当てを工夫す

[1] Intel社が公開した仕様 (MultiProcessor Specification Version 1.4 July 1995) による

ることにより、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(たとえばスケジューラ・アクティベーション[3])と、ユーザ・レベルのみで実現するもの(たとえばPTL[1])とがある。カーネル・レベルで実現すると、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。また、SolarisでもSMP型計算機上で動くマルチ・スレッド環境を提供しているが、これもカーネル・レベルで実現されているスレッド・ライブラリであるため、移植性に乏しい。

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によらずに利用できるという利点がある。

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, 2, 5, 8, 9, 10]。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り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]はインターバル割り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言えない。

本来、SMP型計算機のような環境上では、マルチ・スレッド環境で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になる。本研究では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・ライブラリを設計・実現する。ライブラリは、UNIXが提供するシステム機能のみを用い、ユーザ・レベルで設計・実現を行なうために、カーネルには手を加えない。これによりライブラリは、これまでの安定したカーネルの機能をそのままにして実現が可能となり、アプリケーション・ライブラリもそのまま利用することができる。

本研究では性能評価実験として、これまでに発表されたライブラリと処理時間の比較を行なった。その結果、本研究で提案した方式は、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章では本研究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

多田らのライブラリ[8, 9]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。内部で行なわれているコンテクスト切替の方法は、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。

安倍らによるPTL[1, 2]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に準拠したインターフェイスとインターバル割り込み機能を持つ。また、動的なスタック領域拡張機能も備えている。

Provenzanoらによる MIT pthread[10]は、PTLと同様の機能をもつ。スタック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によるLinuxThreads[5]は、SMP型計算機を想定しているがLinuxのみを想定しているために移植性がない。

以上のことを表1にまとめる。

単一プロセッサに対応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, 6]。単一のUNIXプロセスを作りだし、これにはスケジューリングによってカーネルからプロセッサが割り当てられる(これを仮想プロセッサと呼ぶ)。1つのプロセスには複数のスレッドがあり、各スレッドはこのプロセス内で並行に実行される。ある時間で見るとプロセスは1つだけしかスレッド

表 1: 各スレッド·ライブラリの比較

	移植性	POSIX	SMP 対応
多田ら	○	—	△
PTL	△	○	△
MIT pthread	○	○	△
Linux Threads	×	○	○
本研究	○	○	○

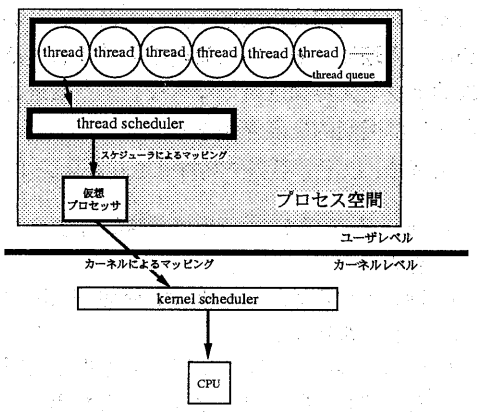

図 1: 従来のシステム構成

を処理できないため、各スレッドが並列には動作しない。

このように単一プロセッサに対するマルチスレッド·システムによって、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で SMP 型計算機上で動かしても、CPU 台数分の性能向上は見込めない。本研究では、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。

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

本研究では、図 2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、全ての UNIX プロセスから見える領域に保存する。すなわち、プロセスはスレッドのコンテクスト共有しているので、任意のスレッドを実行できる。本研究のように SMP 型計算機上であれば、各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。

3.2 実行機構に関する要求と問題点

上記のシステムを実現する際には、以下のことが要求される。

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。移植性が高いことは、ライブラリとして普及する上において重要である。

インターバル割り込みによるコンテクスト切替え 各スレッドを並列に実行させるために、スレッドに割り込みをかける機構が必要となる。スレッドを切替えるためには、実行中のコンテクストの保存と復元が可能でなければならない。また、切替のタイミングを定期的に行なうために、カーネルからシステムにシグナルを発行しなければならない。

標準的なユーザ·インターフェイス ユーザに対するインターフェイスは、標準的である方が良い。移植性、割り込み機構とこのインターフェイスに関する要求は、スレッド·ライブラリとして必要不可欠なものである。

さらに SMP 型計算機でスレッドを並列に動作させるためには、以下のものも必要となる。

全てのプロセスから見える共有メモリ領域 各プロセスが全てのスレッドを任意に実行するためには、スレッドのコンテクストが全てのプロセスから見えなければならない。従って全ての

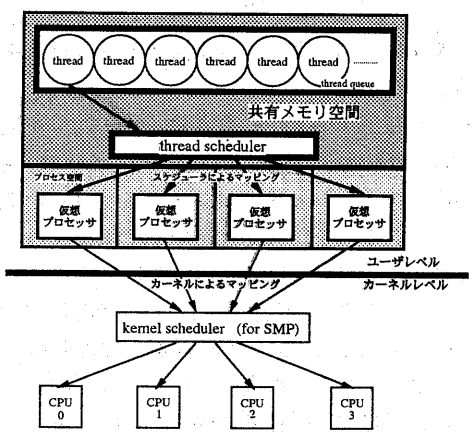

図 2: 本研究のシステム構成

UNIX プロセスが共有するメモリ領域を作成し、そこにスレッドのコンテクストを格納する。

プロセス間の相互排除　上で述べたように、スレッドのコンテクストは全てのプロセスから触ることができる。ここで、1 つのスレッド·コンテクストを複数のプロセスが操作しないようにするために、プロセス間での相互排除機能が必要となる。

3.3 各要求の実現法

前章で述べた問題点に対して、本研究では以下のようにして解決する。

移植性　移植性を保つためには、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する。コンテクスト切替えは、多田ら [8, 9] と同様に、`setjmp` 関数と `longjmp` 関数を用いて行なう。

インターバル割り込み　インターバル割り込みは、PTL[1, 2] と同様に `setitimer` システム·コールを用いる。タイマ割り込みのハンドラにおいて、コンテクスト切替を行なう。

インターフェイス　インターフェイスは [1, 2, 5, 10] と同様に、POSIX の規格をインターフェイスとして適用する。この規格で定められたスレッドは Pthread[7] と呼ばれている。

共有メモリ領域　全てのプロセスから見える共有メモリ領域を作成して、そのなかにスレッドの制御に関する情報を集める。共有メモリ領域を実現するために本研究では、2 つの方法を試みる。

1 つは `shmget`、`shmat` システム·コールを用いる方法であり、もう 1 つは、`mmap` システム·コールを用いる方法である。

前者は、メモリ空間に複数のプロセスから見える空間を動的に生成する方法である。この方法は、実際にメモリ内に領域を生成するため比較的アクセス時間は気にならないが、領域の大きさには機能的に限界がある。

後者は、以下の手順で共有メモリを生成する。

- `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
- `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なう

この方法は、仮想的に巨大なファイルを作っただけで実体は小さくできる。ある程度の大きさの共有メモリ領域であれば、`shmget`、`shmat` システム·コールを用いた時と性能に差異が生まれない。

相互排除　複数の UNIX プロセス間で、UNIX で標準的に提供されているセマフォを利用し、相互排除をして競合を避ける。

4 評価

4.1 実験環境

本研究のライブラリは、現在、表 2に記された OS 上での動作を確認している。

実験機は 2 台の Pentium(150MHz)CPU と、64Mbyte のメモリを持つ SMP 型計算機を用い、OS として FreeBSD 3.0 を用いる。

性能評価実験プログラムには、正方行列の乗算を用いる。行列の乗算は各行列要素が do-all ループで求まるため、並列化に適している。

表 2: 動作確認した OS

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 5.4	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

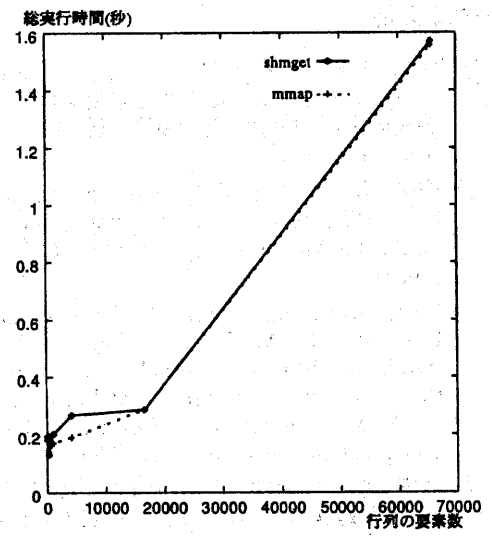

図 3: shmget と mmap の総実行時間比較

4.2 実験結果

まず、共有メモリ領域の実現方法として、shmget システム·コールと mmap システム·コールの場合を比較して、性能の違いを検討する。FreeBSD 3.0 に対し、同じプログラムを実行させた (図 3)。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·コールを利用する時、共有メモリ領域には大きさに制限があるために、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。mmap システム·コールを利用した方が、大きい領域を取ることができ、優れている。

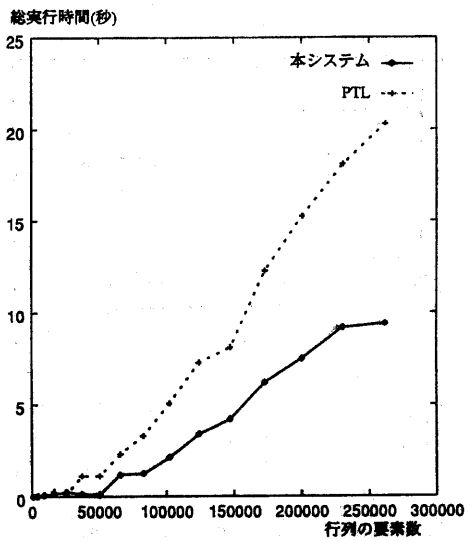

図 4: 本システムと PTL との総実行時間比較 (スレッド数 4)

以下の実験では mmap システム·コールを採用したシステムを用いる。

図 4は、アプリケーションが生成するスレッド数を 4 に固定し、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、本研究で採用した方式は 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なった場合には、PTL と同程度の性能が得られた。また、CPU2 台の SMP 型計算機を使って実行させた時には、スレッドの並列実行による性能向上が現れる。上記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 5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見たものである。

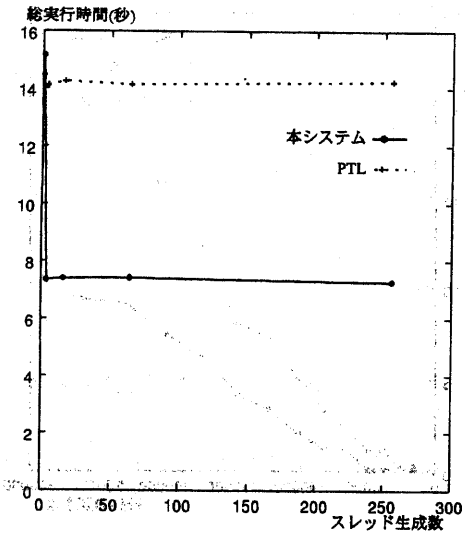

図 5: 本システムと PTL との総時間比較 (要素数一定)

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。これにより、スレッド生成と切替の際に生じるオーバー・ヘッドが小さいことが示された。

5 おわりに

本研究では 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・実現を行なった。並列に動作させるために本研究では、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。性能比較実験により、本研究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ない、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なっている。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 大阪市立大学学術情報総合センターの安倍広多先生には PTL のソースコードを提供して頂き、また励ましのお言葉を頂きました。深く感謝いたします。

また、実験環境の構築に協力くださった電気通信大学 情報工学科 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 PTL-current-970709, http://www.media.osaka-cu.ac.jp/~k-abe/PTL/ (1997).

[3] T. E. Anderson, B. N. Bershad, E. D. Lazowska, H. M. Levy: Scheduler Activations: Effective Kernel Support for the User-Level Management of Parallelism, ACM Transaction on 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] X. Leroy: Linuxthreads - POSIX 1003. 1c kernel threads for Linux, http://pauillac.inria.fr/~xleroy/linuxthreads (1996).

[6] B.Lewis,D.J.Berg 共著, 岩本 信一 訳: マルチスレッドプログラミング入門, アスキー出版局 (1996).

[7] B.Nichols,D.Buttlar,J.P.Farrell: Pthreads Programming,O'Reilly&Associates,Inc (1996).

[8] 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).

[9] 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).

[10] C.Provenzano: Pthreads version 1_70, http://www.mit.edu:8001/people/proven/pthread.html (1996).