Vol.2010-ARC-189 No.2
Vol.2010-OS-114 No.2
2010/4/21

複数カーネル実行機構を利用したアプリケーション実行環境の設計と実装

下沢 拓[†1] 石川 裕[†1,†2]

本稿では，筆者らが提案した複数カーネル実行機構 SHIMOS を用いたアプリケーションの実行環境を提案する．提案する実行環境は，I/O 処理を行うカーネルと計算のみに機能を限定した二つのカーネルを同時に動作させることで，アプリケーションの動作に与える OS の影響を低減することを目的とする．本稿では，この設計と実装の現状を示し，マイクロベンチマーク及び OpenMP を用いたベンチマークによって，計算に機能を限定したカーネルの性能とその結果のばらつきについて報告を行う．

Design and Implementation of An Application Execution Environment by Using A Multiple Kernel Execution Mechanism

Taku Shimosawa[†1] and Yutaka Ishikawa[†1,†2]

We propose a new application execution environment using the SHIMOS mechanism, a multiple kernel execution mechanism. In this environment, two different kernels, one that processes I/O and another that concentrates on calculating, run on a machine. We expect that this environment mitigate the operating system effects on the applications. We show the design and implementation of the proposed environment, and evaluate the performance and variability of the calculating kernel by several microbenchmarks and an application benchmark using OpenMP.

†1 東京大学大学院情報理工学系研究科
Graduate School of Information Science and Technology, The University of Tokyo
†2 東京大学情報基盤センター
Information Technology Center, The University of Tokyo

1. はじめに

現在，コモディティハードウェアにおいても，CPU のマルチコア化が進み，さらに CPU とメモリや I/O への距離が均一ではない NUMA と呼ばれる環境，特に ccNUMA 環境が一般的になりつつある[1),2)]．さらにスーパーコンピューターや計算機クラスタなど，高性能計算の分野においてもこのようなコモディティハードウェアを採用したものが増えている[3),4)]．また，ソフトウェアの面からも，I/O 処理を行うノード，計算を行うノードそれぞれにおいて，専用のカーネル[5),6)] から Linux をはじめとする汎用オペレーティングシステムを採用することも多くなっている[7),8)]．

このような計算を主な目的とする環境において，汎用的なオペレーティングシステムを採用する際の課題としては，ハードウェア割り込みやデーモンの動作による OS ノイズ[9)]，前述した NUMA 環境に起因するメモリの割り当てやスケジューリング[10)] などが挙げられる．これらは，性能の不確定的な変動を引き起こし，複数のノードを用いる並列計算においては全体的な性能低下を引き起こすことになる．他方，専用カーネルを用いる手法においては，汎用 OS に比べて機能が制限されていることが多く，たとえばスクリプト言語などの利用が困難であることも多い．また，保守の面においても，専用カーネルをハードウェアの更改にあわせていくコストは大きい．

我々は，クラスタ環境のノードにおいて，両方の短所を補うことを目標として，デバイス操作などの I/O 処理を行う OS と計算を行うカーネルを分離して実行する環境を提案する．これは，I/O などを受け持つカーネルとして汎用 OS を動作させ，計算カーネルとして他の CPU 上で，OS なしに直接実行，あるいは，汎用 OS をベースに最小限に機能を落としたものを用いて計算を行うものである．これにより，専用カーネルの機能の少なさを補い，汎用 OS の OS ノイズ等の削減を行うことができる．

この実行環境の実現には，複数のカーネルを実行する際に，それ自体によるオーバーヘッドにより計算性能が低下することは避けなくてはならない．このため，仮想マシンの手法を用いることはできない．そこで，我々が提案した複数カーネル実行機構 SHIMOS[11),12)] を用いる．複数カーネル実行機構 SHIMOS は，カーネルに対して変更を加えることで，仮想マシンと異なりオーバーヘッドなく複数のカーネルを実行させることができるものである．論文[11)] において，x86 アーキテクチャを対象とした実装で，仮想マシンモニタの一つである Xen よりも 134%の高速化を実現し，通常の Linux とほぼ変わらない性能を実現したことを示した．また，論文[12)] で，カーネル間通信機構を用いたデバイスの共有を実現し，仮

Vol.2010-ARC-189 No.2
Vol.2010-OS-114 No.2
2010/4/21

想デバイスを用いたベンチマークで，ネイティブと同じか場合により性能が改善したことを示した．

本稿では，複数カーネルを用いた実行環境の設計を述べ，x86_64 アーキテクチャの SHIMOS Linux によって動作する計算用カーネル libsms 及び計算用 Linux の実装を示す．また，マイクロベンチマークとアプリケーションでの実行性能を計測することで評価を行う．このうち，NAS Parallel Benchmark を用いたベンチマークにおいては，標準偏差が約 1/3 となり，性能のばらつきが減少していることを示す．

本稿の構成は，以下の通りである．次節において，提案する実行環境の設計を述べる．第 3 節において，x86_64 アーキテクチャの Linux 上における実装を示す．第 4 節では，OS のジッタや OpenMP の性能を計測するマイクロベンチマークとアプリケーションによるベンチマークを行い，提案する実行環境についての評価を示す．第 5 節で関連研究を示し，第 6 節で結論を述べる．

2. 設　　計

本節では，複数カーネルを実行することによるアプリケーション実行環境の設計について述べる．提案する手法においては，SHIMOS 機構を用いて，I/O を主に処理するカーネルと，計算を行うカーネルと，異なる役割をするカーネルを複数同時に動かすことにより，各カーネルの短所を補うことを目標とする．

以下においては，SHIMOS 機構の概要，全体構成，OS なしでの実行を実現するカーネル *libsms* の構成，アプリケーションの実行について述べる．

2.1 SHIMOS 機構

SHIMOS 機構は，筆者らが提案した複数カーネル実行機構である．これは，カーネルに変更を加えることで，カーネル自身が利用するマシン上の資源を制限することで，資源分割を実現し，複数のカーネルの実行を可能とするものである．CPU も含めた資源分割を行うので，CPU コアの共有は行われないため，複数カーネルを実行するのに，仮想マシン等による方法と異なり，オーバーヘッドがかからない．これらの資源制限は，特殊な共有メモリ領域に保存される各カーネルの資源割り当て表もしくは起動時パラメータによって指定される．

また，他方，デバイスも同様にカーネル間で分割され専有されるが，共有メモリを利用したカーネル間通信機構 (IKC) の仕組みがあり，これを用いることで，カーネル間の通信，あるいは，別カーネルに専有されているデバイスの共有利用も行うことができる．

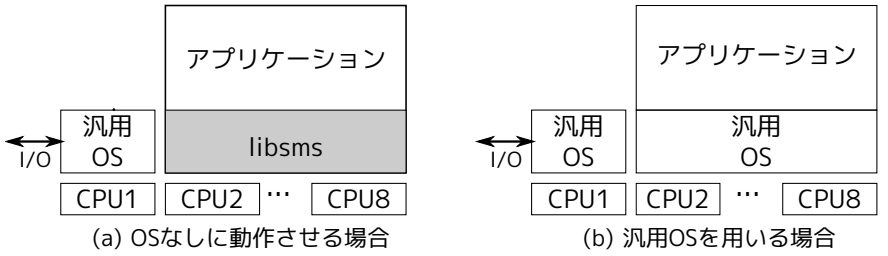

図 1　カーネルの構成

2.2 カーネルの構成と役割

提案手法で実行するカーネルには二種類があり，その構成を**図 1** に示す．一つは I/O 処理等の機能をもつ汎用 OS であり，他方は，計算アプリケーションを動作させるためのカーネルである．後者のカーネルとしては，OS なしに実行するためのプロセス実行ライブラリ libsms あるいは，汎用 OS のカーネルを新たに起動し用いることができる．

I/O 処理を最初の OS に集中させることにより，計算カーネルとして，実行ライブラリを用いる場合でも，汎用 OS でも，不必要なデバイス等を取り扱うコードを省くことができ，それに付随する割り込みやカーネルスレッド，デーモンなどのユーザープロセスを省略することができる．

2.3 libsms の構成

libsms は，アプリケーションを OS なしで起動させるためのライブラリである．アプリケーションのコンパイル時にリンクし，後述する Kloader によってメモリ上にロードされ，割り当てられた CPU 上で実行される．

libsms の行う処理は，次の二つに分かれている．CPU の初期化・終了処理，メモリの確保等の特権命令を必要とする部分と，libc やスレッドライブラリといったユーザーレベルでのライブラリである．特権命令を必要とする部分以外では，CPU はユーザーモードで動作し，実際のアプリケーションは，そのためユーザーモードで動作する．libsms は，単純化とスケジューリング等の影響を排除するために，1 つの CPU コアにつき 1 つのスレッドのみ実行する．そのため，Kloader によるロード時に，スレッド数は明示的に指定され，その数で固定される．`fork` などのプロセス・スレッドの動的な生成は行われない．

また，libsms には，CPU とメモリ以外の資源に対する処理は含まれない．このため，I/O 処理を必要とする場合は，カーネル間通信インタフェースを libsms が媒介することで，実

Vol.2010-ARC-189 No.2
Vol.2010-OS-114 No.2
2010/4/21

現する．

2.4 アプリケーションの実行

アプリケーションの実行には，SHIMOS 機構におけるカーネル実行の仕組みである Kloader と呼ばれるカーネルモジュールを用いる．このカーネルモジュールは，SHIMOS 機構によって実行されているカーネルの一つで動作させるもので，メモリ上に指定されたカーネルを配置し，割り当てられた CPU コアを起動し，そのカーネルコードを実行させることにより，新たなカーネルの実行を開始させるものである．

本提案手法においては，計算カーネルとして libsms を利用した場合は，リンクしたアプリケーションの ELF 形式のバイナリを，汎用 OS を利用した場合は，その OS のバイナリを指定し，Kloader によって起動する．この際，割り当てられた CPU やメモリ等の情報も，カーネルに対するパラメータとして与える．

3. 計算カーネルの実装

本節では，前節で述べた設計に基づき，x86_64 アーキテクチャ上での実装を述べる．以下では，計算カーネル libsms の実装，計算カーネルとしての Linux の構成について，それぞれ示す．

3.1 libsms の実装

libsms の処理について以下では，CPU，メモリ，デバイスについての処理，ユーザーライブラリの処理について，また，現状の実装について，それぞれ述べる．

3.1.1 CPU

2.3 節で述べたように，CPU の初期化と終了処理を行う．具体的には，リアルモードから 64bit モードへの移行，例外ハンドラの設定，次節で述べる仮想メモリマップの作成を行う．また，マルチスレッドアプリケーションを実行する場合には，メインスレッドとなる CPU 以外では，スレッドの実行待ち状態に移行する．メインスレッドであるかどうかは，Kloader によるパラメータにより指定される．それぞれの CPU は，OpenMP の並列化開始処理や pthread のスレッド作成関数によって，スレッドの実行を開始する．なお，スレッドの実行中にプリエンプションは行われない．

3.1.2 メ モ リ

Kloader により与えられるパラメータに基づいて，共有領域およびそのカーネルの専有領域部分をマップする．各専有領域はその CPU から最も近いメモリの中から割り当てる．マルチスレッドアプリケーションを実行する場合においては，同一のメモリ空間で動作させる

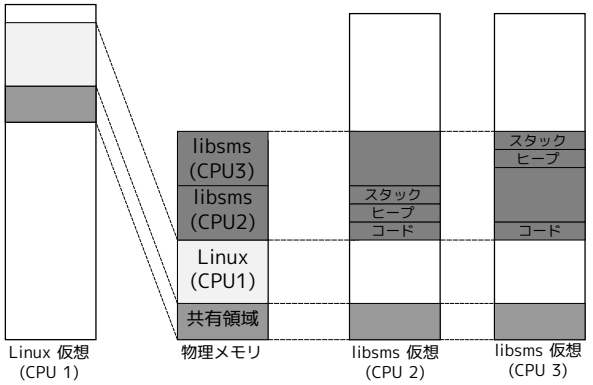

図 2 libsms 上での仮想メモリマップ

カーネルに対する専有領域すべてをマップする．例として，CPU2 と CPU3 の二つの CPU を用いてマルチスレッドアプリケーションを動作させる場合のメモリマップを**図 2** に示す．CPU2 と CPU3 に割り当てられたメモリの両方を CPU2 はマップしており，同一のものを CPU3 も使用することで，この二つの CPU コアで同一メモリ空間でのスレッドの動作を可能にする．この時，コード部分は共通のものを用い，スタックはそれぞれの専有領域を用いる．共有領域部分は，カーネル間通信用の領域として用いられるほか，アーキテクチャの制約上ブート時に必要になる領域として用いられる．

仮想メモリは，libsms のコードおよび，text 領域や data 領域等のアプリケーションのイメージであらかじめ指定される領域では，物理メモリアドレスと同一のものを用いる．これは，起動時に物理アドレスと同一の仮想アドレスマップを必要とするアーキテクチャ上の制約があるためで，初期化処理を単純化する点からもこれを採用している．これ以外の領域については，デマンドページングにより，最初にページを要求した CPU のメモリノードからページが確保される．

3.1.3 デ バ イ ス

libsms では，open や read 等 POSIX システムコールの一部が提供され，これらを介したファイル操作はカーネル間通信機構によって，I/O を行う Linux へ転送される．ただし，非同期 I/O については，提供されない．

3.1.4 ユーザーライブラリ

libc の主要な関数，fortran の実行ライブラリの一部の実装が提供される．コンパイラは，

Vol.2010-ARC-189 No.2
Vol.2010-OS-114 No.2
2010/4/21

GNU gcc を用いるものと仮定し，glibc および fortran ライブラリ関数を置き換える形で提供される．

また，スレッド関連の機能として，POSIX pthread ライブラリ[13] の一部および OpenMP ランタイム[14] の一部が実装されている．ただし，前述の CPU であげた制限により，あらかじめ割り当てられた CPU コア数を超える数のスレッドを同時に実行することはできない．

3.1.5 現状の実装

現状の実装においての制限事項を掲げる．仮想メモリと物理メモリの対応は，起動時に静的に決定され，デマンドページングについては未実装である．また，デバイスについては，コンソールデバイスのみサポートし，そのほかのデバイスや一般のファイルの取り扱いについては未実装である．

3.2 計算用 Linux の構成

I/O 部分を最初に動いている Linux に委任することを前提とし，計算用 Linux からは各種デバイスドライバを削除する．また，ディスクキャッシュによる影響を排除し，ディスクデバイスを必要としないように，動作させるアプリケーションは initrd と呼ばれる RAM ディスク上に置くこととする．Kloader からカーネルイメージと initrd はメモリ上に配置され，起動時にコマンドラインパラメータとして，その位置等が指定される．RAM ディスク上以外のファイル等の I/O 処理が必要な場合は，仮想デバイスドライバを用いて，カーネル間通信機構によって，I/O を担当する Linux が処理を代行する．

なお，計算用 Linux については，SHIMOS 機構の一部が x86_64 アーキテクチャへの移植が完了していないため，I/O を行う OS と同時実行する部分は，未実装となっている．

4. 評　　価

本節では，前節で述べたうち，実装の完了している libsms についての評価を示す．用いたマシン環境は，**表 1** に示すもので，クアッドコアの CPU を 4 ソケット持つマシンである．また，ccNUMA 構成となっており，各ソケットは，2GB のローカルメモリをもっている．ソフトウェアについては，用いたコンパイラは，GNU gcc 4.3.3 であり，FORTRAN としてはこれに付属する GNU fortran を用いた．また，比較対象としての OpenMP ランタイムとしては，同じくこれに付属する GNU libgomp(以下，GOMP と表記) を用いた．

以下では，まず，マイクロベンチマークとして，割り込み等による OS ノイズ，単純なメモリアクセスプログラムによる性能測定，また EPCC OpenMP Microbenchmarks[15] を利用した OpenMP の同期性能の測定を行う．次に，アプリケーションベンチマークとして，

表 1　評価環境

モデル	APPRO Opteron 4-way Server
CPU	AMD Opteron 8378 (Quad-core, 2.4GHz) x 4
Cache per Socket	L1: 64KB(I)+64KB(D) x 4, L2: 512KB x 4, L3: 6MB
メモリ	DDR2 800MHz FB-DIMM 1024 MB x 2 x 4
OS, Userland	Linux 2.6.31 + Ubuntu 9.10
NIC	Intel 82546GB (e1000, PCI Express) x 2

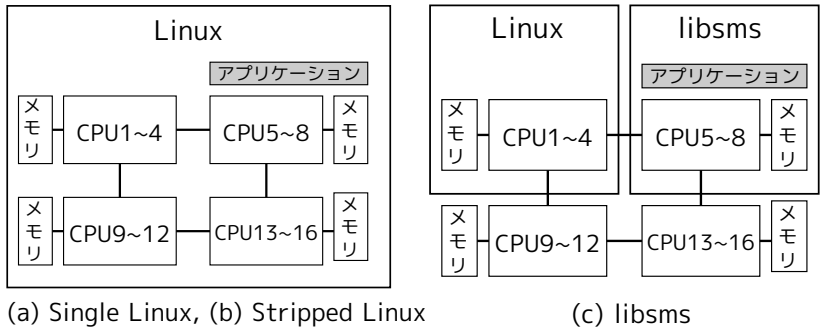

図 3　実験の設定と CPU の割り当て

NAS Parallel Benchmarks[16] を動かした場合の性能について示す．

以下では，複数カーネルを実行せず単一の Linux を使用した場合 (以下，Single Linux と表記)，3.2 節で述べたように Linux の機能を最低限にした場合 (以下，Stripped Linux と表記)，提案手法である複数カーネル実行機構を用いて libsms で実行した場合 (以下，libsms と表記) のパターンについて述べる (**図 3**)．なお，特に指定のない場合，公平を期するために Linux を用いた場合でも，libsms のうち libc や OpenMP ランタイムなどのユーザーライブラリは，同等の実装を用いることとした．そのため，時間測定にはすべてタイムスタンプカウンタを用いている．また，Linux 上でアプリケーションが使用する CPU コアおよびメモリノードの指定には，`numactl` コマンドを用いた．

4.1 マイクロベンチマーク

最初に，割り込みなどによる OS のノイズの状態について評価を行う．評価方法は，ベンチマークプログラムでタイムスタンプカウンタを繰り返し 2^{28} 回読み，その差分を記録した．その差分が他と大きく異なるとき，OS や他スレッドの実行がその間で入ったことがわかる．

Vol.2010-ARC-189 No.2
Vol.2010-OS-114 No.2
2010/4/21

表 2 TSC 間隔の実験結果

間隔 (クロック)	Single Linux	Stripped Linux	libsms
32 - 63	268,431,092	268,433,709	268,435,456
64 - 127	1	1	0
128 - 255	0	0	0
256 - 511	0	0	0
512 - 1023	2,181	872	0
1024 - 2047	2,126	817	0
2048 - 4095	51	55	0
4096 - 8191	5	2	0

*全計測回数は，268,435,456 (2^{28})

```
unsigned long a[32M], b[32M];
int i,j;

for(i = 0; i < 32M; i++){
    a[i] = a[i] + b[i];
}
```

図 4 メモリアクセスベンチマークのコア部分

なお，このベンチマークに所要する時間はおよそ 8.7 秒であった．その結果を**表 2** に示す．Single Linux と比較して Stripped Linux では，割り込みが少なくなっており，libsms では，OS が存在しないので割り込みが一度も起こっていないことが分かる．

次に，256MB のメモリ 2 箇所へのアクセスを 40 回シーケンシャルに行うプログラム (そのコア部分を**図 4** に示す) を走らせ，その中央 10 回の実行時間の平均の測定を繰り返し行った．その結果を**表 3** に示す．Stripped Linux では Single Linux と比較して，1.7%性能が向上している．libsms でも同様の結果となることが期待されたが，Single Linux よりも若干悪い結果となっており，Stripped Linux と比較して 2.1%性能が低下している．ただし，標準偏差については，若干改善されている．なお，libsms のユーザーライブラリを利用したことにより，glibc を使用した場合と比較して，同じ Linux でも 2.0%性能が向上している．

また，パフォーマンスカウンタを用い，キャッシュおよび TLB のミス数について計測を行った．測定には，Linux においては，カーネルのパフォーマンスカウンタサポートを用いた．その結果を**表 4** に示す．全体的な実行性能としては，libsms が早くなっているが，これは Linux カーネル内におけるパフォーマンスカウンタの処理に時間がかかったためと考えられる．この結果においては，特に libsms が Linux の二つと比べて L2 Cache Miss が高い．このために，図 3 の場合で Linux よりも低い性能となったものと考えられる．

表 3 メモリアクセスベンチマークの結果

環境	平均 (s)	標準偏差 (s)
Single Linux	0.231949	1.76×10^{-4}
Stripped Linux	0.228019	1.50×10^{-4}
libsms	0.232909	4.61×10^{-5}
Stripped Linux(glibc)	0.247296	1.29×10^{-4}

*時間は，1 回のシーケンシャルアクセスあたりの elapsed time

表 4 パフォーマンスカウンタを用いた結果

環境	平均 (s)	標準偏差 (s)	L2 Cache Miss	TLB Miss
Single Linux	0.251183	2.13×10^{-4}	1.656M	130.9K
Stripped Linux	0.246606	1.81×10^{-4}	1.658M	131.0K
libsms	0.237697	8.67×10^{-5}	1.900M	130.8K

*時間は，1 回のシーケンシャルアクセスあたりの elapsed time
L2 Cache Miss，TLB Miss は，ユーザー空間内の miss のみ測定．

4.2 OpenMP マイクロベンチマーク

OpenMP のランタイムについての性能評価を示す．用いたベンチマークは，EPCC OpenMP Microbenchmarks のうちの `syncbench` で，おもに同期プリミティブの性能を計測するベンチマークである．4 スレッド (1 つのソケットを使用) で動作させた場合の結果を，**図 5** に示す．REFERENCE は，計測するプリミティブの間に挿入されたループの実行時間，それ以外はそれぞれ OpenMP のディレクティブに対応するものである．なお，libsms 上に未実装の部分があるため，一部のベンチマークは行われていない．Stripped Linux と libsms を比較すると，FOR と BARRIER を除き libsms のほうがよい結果を得られている．FOR については実際に並行処理を行わないため，実質的に開始と終了におけるバリアの性能を左右していると思われる．

また，ベンチマークを 100 回繰り返した際の BARRIER ベンチマークの分布について**図 6** に示す．結果値のばらつきについては，libsms が最も少ないことが見え，Stripped Linux と Single Linux はほぼ変わらない．

4.3 アプリケーションベンチマーク

最後に，OpenMP 版の NAS Parallel Benchmark を動作させた場合の評価について述べる．

利用したベンチマークは，NAS Parallel Benchmark 3.3 の OpenMP 版であり，そのうち

Vol.2010-ARC-189 No.2
Vol.2010-OS-114 No.2
2010/4/21

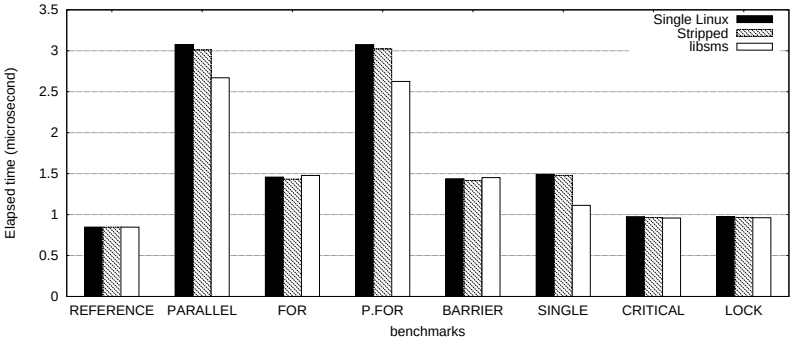

図 5 OpenMP マイクロベンチマーク

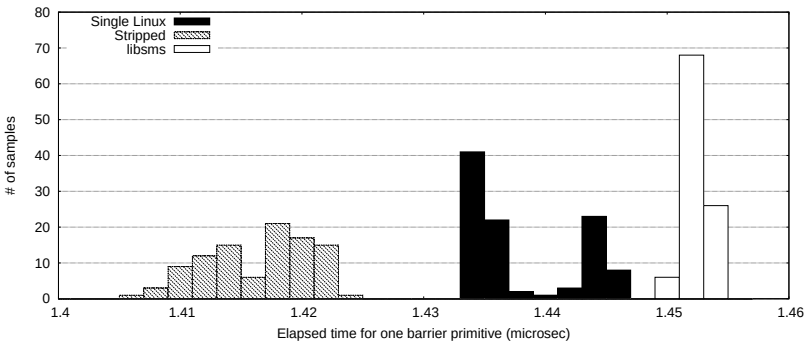

図 6 BARRIER 時の頻度分布

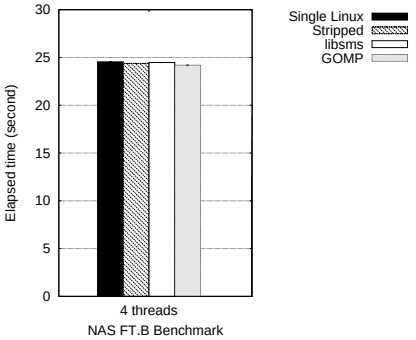

図 7 FT ベンチマークの結果 (平均)

表 5 FT ベンチマークの結果 (標準偏差)

環境	標準偏差 (s)
Single Linux	1.769×10^{-2}
Stripped Linux	1.598×10^{-2}
libsms	4.827×10^{-3}
GOMP	5.609×10^{-2}

FT ベンチマークの結果を示す．使用した問題サイズ (class) は B である．これを 4 コア (1 ソケット) でそれぞれの環境で実行した場合の経過時間を**図 7** に示す．またその標準偏差を**表 5** に示す．また，比較のために，Stripped Linux 上で GNU OpenMP(GOMP) を利用した場合をともに示す．

どの場合も概ね近い結果であるが，libsms は Stripped Linux と比べて 0.4%ほど悪い結果となっている．また，GOMP を利用した場合では，Stripped Linux よりも 0.75%良い結果である．ただし，結果のばらつきの点で標準偏差を見た場合，GOMP を使った場合，もっともばらつきが大きく，libsms が最も小さい値となっている．

4.4 考　　察

OS のない libsms でのアプリケーションの実行に関しては，4.1 節で述べたように，理由を明らかにすることはできなかったが，何らかの理由でメモリアクセスが遅くなっていることが要因で，OpenMP の一部のベンチマークおよびアプリケーションベンチマークでの実行結果が，Stripped Linux より悪いものとなってしまったと考えられる．

他方，Linux では，割り込みによるノイズによって，OpenMP マイクロベンチマークやアプリケーションベンチマークでの結果のばらつきに関しては，libsms より若干悪い結果になったと考えられる．同様に，GOMP を利用した場合にも結果のばらつきは libsms と比較して大きい結果となった．このように，libsms で目標としたノイズのない環境において，結果のばらつきが減少することが確認できたと考えられる．

Vol.2010-ARC-189 No.2
Vol.2010-OS-114 No.2
2010/4/21

5. 関連研究

OS ノイズの影響の測定については，例えば，Beckman らは，論文[17] において Selfish と呼ばれるツールを用いスーパーコンピュータ等の環境で測定した．また，Shmueli ら[18] は IBM BlueGene/L 上で動作する専用の計算カーネルを Linux で置き換えた場合の影響と対策について述べている．ただし，ここでは，TLB ミスの数がパフォーマンス低下の大きな要因とし，本稿で対象とする環境とは状況が異なる．

スーパーコンピュータにおいては，計算ノードと I/O ノードを分離し，それぞれで異なる OS を動作させるものがある．例えば，Cray XT5[4] や IBM BlueGene/P[19] においては，計算ノードで Linux を用いるか専用のカーネル CNK を用いるかの違いはあるが，計算ノードからのファイル等の I/O 処理はインターコネクトを用いて，I/O ノードで処理される．これらは，ハードウェアがそのような構成で設計されたものを対象としたものであり，また，ノード内で異なるカーネルにより，コアごとに役割を分担するものではない．

Barrelfish[20] は，カーネル内において，コア間の通信をキャッシュの一貫性を利用した共有メモリによる方式から，明示的なメッセージパッシング方式にすることで，スケーラビリティを確保しようとするものである．しかし，ドライバやライブラリについては，カーネル構造が異なることから，汎用 OS のものを用いることができず，移植をする必要がある．

6. おわりに

本稿では，コモディティCPU を用いたクラスタノード上で計算を行う場合に，汎用 OS を用いた場合に発生する問題点を解決することを目的としたアプリケーション実行環境を示した．そして，その性能についての評価を行い，割り込み等のノイズに関して OS なしの環境では期待通りの結果となってることを示した．また，OpenMP を用いたマイクロベンチマークの結果から，libsms 上での OpenMP ランタイム性能が，FOR と BARRIER を除き Linux よりも向上していることを示した．最後に，NAS Parallel Benchmark のうち FT Benchmark を用いたベンチマークでは，結果のばらつきが Linux を用いた場合よりも改善していることを示した．

今後の課題としては，未実装部分の実装と評価，libsms と Linux を比較した場合に起こる性能の低下に対する実装の改善，I/O 処理を含めた実装と評価，さらに，複数のノード間の通信を伴うような場合の評価等を行う必要があると考える．

謝辞 本研究の一部は，科学技術振興機構 戦略的創造研究推進事業 (CREST) (領域名：実用化を目指した組込みシステム用ディペンダブル・オペレーティングシステム) 技術課題：「高信頼組込みシングルシステムイメージ OS」による．

参考文献

1) Intel: First the Tick, Now the Tock: Intel(R) Microarchitecture (Nehalem), `http://www.intel.com/technology/architecture-silicon/next-gen/319724.pdf`.
2) Advanced Micro Devices: AMD Opteron Processor Product Data Sheet, `http://www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/23932.pdf`.
3) Nakashima, H.: T2K Open Supercomputer: Inter-university and Inter-disciplinary Collaboration on the New Generation Supercomputer, *ICKS '08: Proceedings of the International Conference on Informatics Education and Research for Knowledge-Circulating Society (icks 2008)*, Washington, DC, USA, IEEE Computer Society, pp.137–142 (2008).
4) Cray Inc.: Cray XT5 Brochure, `http://www.cray.com/Assets/PDF/products/xt/CrayXT5Brochure.pdf`.
5) Kelly, S.M. and Brightwell, R.: Software Architecture of the Light Weight Kernel, Catamount, *Proceedings of the 2005 Cray User Group Annual Technical Conference*.
6) Moreira, J., Brutman, M., Casta nos, J., Engelsiepen, T., Giampapa, M., Gooding, T., Haskin, R., Inglett, T., Lieber, D., McCarthy, P., Mundy, M., Parker, J. and Wallenfelt, B.: Designing a highly-scalable operating system: the Blue Gene/L story, *SC '06: Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, New York, NY, USA, ACM, p.118 (2006).
7) Wallance, D.: Compute Node Linux: Overview, Progress to Date & Roadmap, *Proceedings of the 2007 Cray User Group Annual Technical Conference*.
8) Brightwell, R., MacCabe, A.B. and Riesen, R.: On the Appropriateness of Commodity Operating Systems for Large-Scale, Balanced Computing Systems, *IPDPS '03: Proceedings of the 17th International Symposium on Parallel and Distributed Processing*, Washington, DC, USA, IEEE Computer Society, p.68.1 (2003).
9) Petrini, F., Kerbyson, D.J. and Pakin, S.: The Case of the Missing Supercomputer Performance: Achieving Optimal Performance on the 8,192 Processors of ASCI Q, *SC '03: Proceedings of the 2003 ACM/IEEE conference on Supercomputing*, Washington, DC, USA, IEEE Computer Society, p.55 (2003).
10) Li, T., Baumberger, D., Koufaty, D. A. and Hahn, S.: Efficient operating system scheduling for performance-asymmetric multi-core architectures, *SC '07: Proceedings of the 2007 ACM/IEEE conference on Supercomputing*, New York, NY, USA, ACM, pp. 1–11 (2007).
11) Shimosawa, T., Matsuba, H. and Ishikawa, Y.: Logical Partitioning without Archi-

Vol.2010-ARC-189 No.2
Vol.2010-OS-114 No.2
2010/4/21

tectural Supports, *IEEE International Computer Software and Applications Conference*, pp.355–364 (2008).

12) Shimosawa, T. and Ishikawa, Y.: Inter-kernel Communication between Multiple Kernels on Multicore Machines, *IPSJ Transactions on Advanced Computing Systems (ACS)*, Vol.2, No.4, pp.261–279 (2009).
13) IEEE: *IEEE Std. 1003.1c-1995 thread extensions* (1995).
14) OpenMP Architecture Review Board: OpenMP Specifications, `http://openmp.org/wp/openmp-specifications/`.
15) Reid, F. J.L. and Bull, J.M.: OpenMP Microbenchmarks Version 2.0, `http://www.hpcx.ac.uk/research/hpc/technical_reports/HPCxTR0411.pdf`.
16) Bailey, D. H., Barszcz, E., Barton, J. T., Browning, D. S., Carter, R. L., Dagum, L., Fatoohi, R. A., Frederickson, P. O., Lasinski, T. A., Schreiber, R. S., Simon, H.D., Venkatakrishnan, V. and Weeratunga, S.K.: The NAS parallel benchmarks—summary and preliminary results, *Supercomputing '91: Proceedings of the 1991 ACM/IEEE conference on Supercomputing*, New York, NY, USA, ACM, pp. 158–165 (1991).
17) Beckman, P., Iskra, K., Yoshii, K., Coghlan, S. and Nataraj, A.: Benchmarking the effects of operating system interference on extreme-scale parallel machines, *Cluster Computing*, Vol.11, No.1, pp.3–16 (2008).
18) Shmueli, E., Almasi, G., Brunheroto, J., Castanos, J., Dozsa, G., Kumar, S. and Lieber, D.: Evaluating the effect of replacing CNK with linux on the compute-nodes of blue gene/l, *ICS '08: Proceedings of the 22nd annual international conference on Supercomputing*, New York, NY, USA, ACM, pp.165–174 (2008).
19) Sosa, C. and Knudson, B.: *IBM System Blue Gene Solution: Blue Gene/P Application Development* (2009). ISBN 0738433330.
20) Baumann, A., Barham, P., Dagand, P.-E., Harris, T., Isaacs, R., Peter, S., Roscoe, T., Schüpbach, A. and Singhania, A.: The multikernel: a new OS architecture for scalable multicore systems, *SOSP '09: Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, New York, NY, USA, ACM, pp.29–44 (2009).