2009－OS－110 (13)
2009／1／29

マイクロカーネル方式による Cell/B.E. 向け OS 構成法の提案と MINIX 3 による実現

野尻祐亮† 並木美太郎†

Cell/B.E. は，PPE, SPE の各プロセッサコアからなる，ヘテロジニアス構成のマルチコアプロセッサである．従来，PPE では OS を動作させシステム全体の制御を行い，SPE ではアプリケーションによって大規模演算を行うのが一般的であった．しかし，OS サーバやデバイスドライバは，多数ある SPE の演算能力を生かすことができなかった．そこで，SPE をアプリケーションにだけでなく，OS 機能の実行に利用し，プロセッサコアを有効活用したい．また，SPE でも PPE と同様のプロセスモデルを実現し，プログラミングを容易としたい．本研究では，MINIX のマイクロカーネル方式を利用することで，OS 機能を各プロセッサコアに割当て，並列実行するモデルが成り立つと考え，Cell/B.E. に対応した MINIX の開発を行う．MINIX 3 に，ヘテロジニアスマルチコアに対応したプロセス管理機構を実装する．また，プロセスを実行するプロセッサコアの種類が異なっても，透過的にプロセス間通信を行える通信機構を実装する．評価の結果，SPE におけるプロセス間通信の性能は問題ないことが分かった．

Proposal of Microkernel-Based OS Structure for Cell/B.E. and its Implementation using MINIX 3

Yusuke NOJIRI † and Mitaro NAMIKI†

Cell/B.E. is heterogeneous multicore processor which includes processor cores of PPE and SPE. Cell/B.E. programmers commonly use PPE for OS execution and control of the whole system, and SPE for large-scale calculation. However, OS servers and device drivers could not be executed on SPEs. SPE should be utilized not only for applications but for OS functions. Programming on Cell/B.E. is not easy because the process model on SPE different from PPE. It is possible to assign OS functions to each processor cores and execute them parallel by applying MINIX's microkernel. In this study, authors develop MINIX 3 for Cell/B.E. Authors will implement process management for heterogeneous multicore processors on MINIX. Authors will also implement interprocess communication that makes transparent communication with different processor cores possible. As a result of evaluation, it turned out that performance of interprocess communication with SPE has no problem.

1. はじめに

本研究では，Cell/B.E. (Cell Broadband Engine) プロセッサを対象に，ヘテロジニアスマルチコアプロセッサに適した OS, **PS3 MINIX** の設計，実現を行っている．本研究の前回の報告[1] では，ヘテロジニアスマルチコアプロセッサにおける問題点を指摘し，それを解決するために，MINIX 3 をベースとしたヘテロジニアスマルチコア対応 OS を実現すると述べた．その段階としてまず，MINIX 3 を Cell/B.E. に移植して，単一のプロセッサコアで動作させた．そして，その OS に実装すべき，マルチコアの管理方式の検討を行った．今回の報告では，さらに詳細なマルチコア管理方式の設計と，その実装について述べる．そして，その設計を評価するため，性能測定を行う．

† 東京農工大学
Tokyo University of Agriculture and Technology

2. 研究概要

近年のプロセッサ開発の潮流はマルチコアへと向かっているが，その流れのなかでも特にヘテロジニアス（非対称型）マルチコアという形態が注目されている．ヘテロジニアスマルチコアは，汎用的なプロセッサコアを複数搭載する（ホモジニアスマルチコア）よりも，目的の用途に特化したプロセッサコアを含めることで，高いパフォーマンスを実現しつつ製造コストや消費電力を下げ，高い効率を狙うという設計思想である．本研

究で対象としている Cell/B.E. は，システム全体の制御用の PPE (PowerPC Processor Element) と，ベクトル演算用の SPE (Synergistic Processor Element) の，2 種類のプロセッサコアを搭載している．

しかし，ヘテロジニアスマルチコアでは，ソフトウェアの設計・開発が複雑になるという問題点がある．プロセッサコアごとに異なる命令セットにしたがってプログラミングすることや，それぞれのプロセッサコアの特性を引き出すプログラミングをすることが容易でない．Cell/B.E. では命令セットの違いのほか，メモリへのアクセス手段も異なる．SPE は，ローカルで小さく高速なメモリ（ローカルストレージ）を持つ代わりに，メインメモリへのアクセスは DMA による処理を明示的に記述する必要がある．また，システムソフトウェアによるヘテロジニアスマルチコアのサポート状況について見ると，Cell/B.E. 機においてもっぱら使われている Linux では，カーネルはヘテロジニアスなプロセッサコアのタスク管理をせず，管理がアプリケーションに委ねられるため，アプリケーションプログラマにとって負担である．そのほか，システムの制御，プロセス管理，リソース管理，デバイスドライバなどの OS の処理は，特に工夫しない限り 1 個のプロセッサコア（あるいは 1 種類のプロセッサコア）でしか実行されない．しかし，それらの処理は各プロセッサコアへ分散させることが望ましい．プロセッサの開発は，今後ますますプロセッサコア数を増やす方向に向かっていっており，アプリケーションソフトウェアだけでなく，システムソフトウェアを含むプログラミングシステム全体の機能を各プロセッサコアへ分散させることが重要であると考える．

このような問題点を解決するため，Cell/B.E. において，すべてのプロセッサコアを統合的に管理する新たな OS を実現する．その OS において，単なる並列化とは異なる新規な点として，次の点を目標とする．

- **OS の機能の一部を，PPE だけでなく SPE などの別のプロセッサコアでも実行できるようにする．**
  特定の機能については，SPE など別のプロセッサコアで実行した方が高速であるかもしれない．たとえ別のプロセッサコアにおいて高速にならなくても，遊休しているプロセッサコアの活用により，全体としての性能向上につながる．
- **上位層で実行するプログラムのために，統一したプロセスモデルを提供する．**
  デバイスドライバやアプリケーションを，異なる命令セットを持つ別のプロセッサコアに移植することが容易になる．ヘテロジニアスマルチコアプロセッサを利用するにあたっての敷居を下げることができる．

並列化すべき OS の機能としては，例えばデバイスドライバが挙げられる．画面表示のためのフレームバッファドライバ，あるいはストレージやネットワーク入出力におけるデータ暗号化，圧縮のフィルタドライバは処理量が大きく，並列化することが望ましい．MINIX においては，デバイスドライバや OS サーバなどはすべてプロセスとして存在している．SPE においても PPE にできるだけ近いプロセスモデルを実現し，SPE でそれらのプロセスを動作させることで，全体として性能の向上を図る．

本研究で開発する OS は，マイクロカーネル構造を持つ MINIX[2] をベースとする．マイクロカーネルでは，OS の機能が小さなモジュールとして独立しており，OS の機能を各プロセッサコアに割り当てて，並列に実行させるモデルが成立すると考えた．それにより，OS の機能を別のプロセッサコアでも実行できるようにするという目標が達成できる．

OS の実現の段階としては，(1) まず MINIX 3 を Cell/B.E. に移植し，(2) 次にその MINIX をマルチコアに対応させるという風に分けられる．前回の報告[1] は主に (1) の移植の部分について述べた．本稿ではそれを踏まえて，以降，PS3 MINIX で実現するプロセスモデル，プロセッサコア管理方式の設計，その実装について述べていく．

3. OS アーキテクチャとプロセスモデル

本研究の方針は，SPE 上でも，できるだけ従来の MINIX プロセスモデルと同じモデルのプロセスを再現し，各プロセスをハードウェア的に並列実行することで，OS 機能の並列化を達成することである．ここではまず，OS 機能を並列化させる，PS3 MINIX の OS アーキテクチャを示す．そして，それを実現するためのプロセスモデルと，プロセス間連携の方式について述べていく．

3.1 OS アーキテクチャ

マルチコアで OS 処理を行う OS アーキテクチャを**図 1** に示す，主に二つの処理モデルを想定している．

一つは，OS 機能を各プロセッサコアで分担するモデルである．例えばデバイスドライバを SPE プロセスとして作成し実行する．OS 機能を担う一部のプロセスを SPE 用に作成すれば，プロセッサコア同士の連携は，MINIX のプロセス管理にのっとって行われるので，それで目標である OS 機能の各プロセッサコ

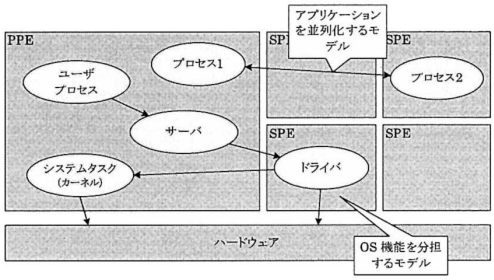

図 1　PS3 MINIX の OS アーキテクチャ

アへの分散が達成できる．システム全体としての性能の向上を狙うには，例えばフレームバッファドライバのような，演算処理量の多いプロセスを SPE に移すことが効果的である．

もう一つは，アプリケーションをマルチプロセスで作成し，並列実行するモデルである．このモデルを実現するためには，特にマイクロカーネルを適用する必要性はない．しかし，各プロセッサコアで同様のプロセスモデルが実現できていて，SPE でも PPE と同様にプログラミングできるという，PS3 MINIX の長所は依然としてある．

3.2　プロセスモデル

SPE においても，従来の MINIX で提供していたプロセスモデルとできるだけ同じモデルを提供できるようにする．MINIX プロセスの概略は次の通りである．

メモリ空間　各プロセスにおいて，ローカルメモリとして仮想アドレス空間が利用できる．利用できるローカルメモリサイズは静的に（ビルド時に）定まる．物理アドレス空間へのアクセス，リモート（別のプロセス，別のプロセッサコア，I/O）な仮想アドレス空間へは，カーネルコールを直接ないし間接的に利用してアクセスする．

プロセス間のメッセージ転送　`send`, `receive`, `sendrec`, `notify` の各メッセージ転送コールを行うことで，メッセージ転送を実行できる．

プロセス間のメモリ転送　カーネルコールの呼出しにより実行する．

システムコール・カーネルコール　POSIX システムコール，カーネルコールがすべて利用できる．

ライブラリ　C 標準ライブラリ，システムコールライブラリなどを PPE, SPE 各アーキテクチャ用に提供する．各アーキテクチャ用のものを，静的リンクによってあらかじめ実行ファイルに取り込んでおく．

ハードウェア入出力　カーネルが代行するか，メモリマップ I/O により直接行う．

できるだけ SPE でも同じプロセスモデルを実現するが，制約を設けざるを得ない部分も存在する．SPE プロセスにおける制約を次に示す．

- ローカルメモリ以外のメモリやハードウェアに直接アクセスできない．
- ローカルメモリは 256KB しか利用できない．
- 同時に存在できる SPE プロセス数は SPE の個数以下．
- テキスト書換えのような不正なメモリアクセスを検出，防止できない．

また，PPE と SPE でシステムコールの互換性を保つため，仕様を若干変えなければならないシステムコールもある．詳しくは 4.4 節で述べる．ここで挙げた点以外は，SPE プロセスでも PPE プロセスと同様のことを行うことができる．

3.3　プロセス同士の連携

これまで述べたモデル上で，プロセス同士の連携がどのように実現されるのかを示す．プロセス同士の連携は，システムコールなどのプロセス間関数呼出しで行う．MINIX におけるプロセス間関数呼出しには次のようなものがある．

- **POSIX システムコール**
  `fork`, `open`, `times` など，UNIX で共通して使われているシステムコールである．プロセス管理サーバ，ファイルシステムサーバなどが処理する．
- **MINIX カーネルコール**
  サーバやドライバが呼び出し，システムタスクが処理する．プロセスの生成・破棄，入出力ポートの操作 (x86)，外部セグメントへのメモリ転送，物理アドレスの取り扱い，システム情報の取得など，カーネルモードでしか行うことのできない処理を担う．`sys_*` の関数名を持つ．

その他，例えばドライバとサーバの間の通信など，当該プロセス同士の取り決めにより，独自の仕様のメッセージを交換することも多い．本節では，これらのプロセス間の連携をシステムコールとしてまとめて考えて，その実現方法を示す．

システムコール処理の流れを**図 2** として示す．サーバプロセス☆は，メッセージ転送コール `receive` でメッセージの到着を常に待つようにしておく．クライアントプロセスは，システムコールを呼び出すと，クライアント側においてまずシステムコールライブラ

☆ ここで用いている「サーバプロセス」という語は，プロセス管理サーバ，ファイルシステムサーバにとどまらず，広義の「機能を提供する側のプロセス」を指す．

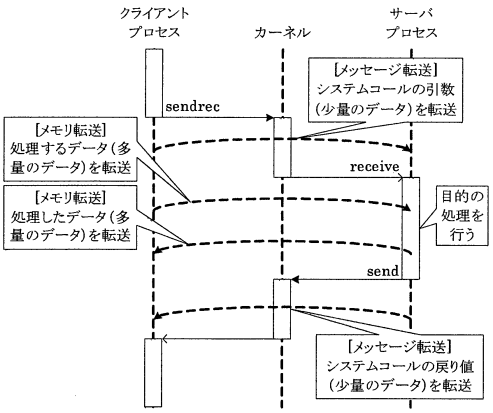

図 2 システムコール処理の流れ

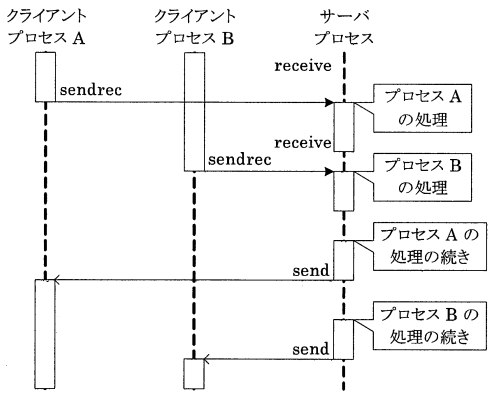

図 3 サーバの非同期処理の流れ

リ内でシステムコール番号や引数がメッセージ化され，sendrec によってカーネルへと制御が渡される．sendrec はメッセージの送信と受信を続けて行うメッセージ転送コールである．カーネルでメッセージ転送が行われ，サーバプロセスへと制御が移る．少量のデータはメッセージに乗せて渡すことができるが，多量のデータは，メッセージに乗せられないので，サーバプロセスにおいてカーネルのメモリ転送コールを呼び出すことで，データを交換する．典型的なケースでは，データへのポインタだけメッセージで渡し，データ本体はメモリ転送コールで転送する．処理が終了したら，メッセージ転送コール send で，処理結果をメッセージに乗せてクライアントプロセスへ送り返す．クライアントプロセスのシステムコールライブラリは，そのメッセージを読み取り，システムコールの戻り値などとして解釈する．

ここで述べたシステムコールの流れはネストすることもある．例えば，ファイル入出力を行うシステムコールは，クライアントプロセス→ファイルシステムサーバ→デバイスドライバ→カーネルのように呼び出される．

サーバプロセスは，特に工夫しない限り，システムコールの処理を直列に行う．しかし，**図 3** に示すように，複数の処理内容を同時に保持して管理し，非同期で処理を進めることで，擬似的に並列にシステムコールを処理できる．プロセス管理サーバ，ファイルシステムサーバは非同期で処理する設計となっている．

4. 設　　計

PS3 MINIX では複数の異なる種類のプロセッサコアを管理する構造が必要である．また，SPE においても PPE と同様のプロセスモデルを実現するため，カーネルにおいて SPE プロセスとのメッセージ交換やメモリ転送をサポートする必要がある．本章では，それらの設計について述べていく．

4.1 プロセッサコアの管理方式

プロセッサコアの管理方式について述べる．プロセッサ管理を行う仕組みはカーネル内に設け，メインコア上でのみ実行する．カーネル内に設けるのは，プロセッサコア管理が，スケジューリング，メモリ配置，データ転送，割込みなどカーネルの機能と密接に関わるためである．カーネルではプロセッサコアは，メインのプロセッサコア，サブのプロセッサコアに関わらず，一様のインタフェースを持つオブジェクトとして扱う．

全体図を**図 4** に示す．カーネル内のスケジューラ，割込み管理，メモリ転送などの各機能は，プロセッサオブジェクトのインタフェースやデータを利用して，プロセッサコアに対して操作を実行する．プロセッサオブジェクトはそれぞれランキューを持つ．ランキューはスケジューラによって，そのプロセッサコアにおいてどのプロセスを実行するのかを決定するのに利用される．

システムイメージに含まれる初期プロセスと，exec システムコールで実行するプロセスでは，実行ファイルのロード方法が異なる．初期プロセスの場合，すでに実行ファイルはメインメモリに配置されているので，プロセッサの種類によっては，別のメモリ空間へ転送し，再配置する処理を行う必要がある．exec システムコールで実行するプロセスの場合，実行ファイルの読込みを行うのはファイルシステムサーバ，配置するのはカーネルである．イメージの配置は，カーネルが

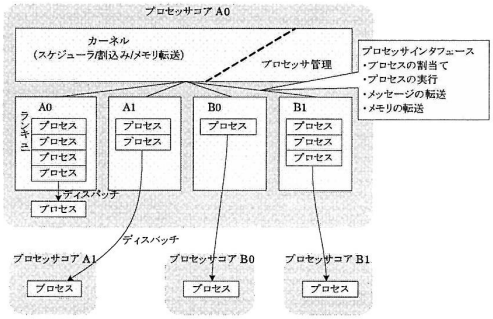

図 4　カーネルによるプロセッサコア管理

プロセッサインタフェースを通じて行うようにする．

SPE においてはプロセススイッチを行わず，一つのプロセスで一つのプロセッサコアを専有させる．それは，SPE におけるプロセススイッチのコストが高く，プロセススイッチを実装してもいい性能が得られないと予想しているためである．しかし，何らかの工夫により，プロセススイッチのコストを抑え，SPE の個数以上の SPE プロセスを動作させられるようになれば，利便性が非常に高まる．1 個の SPE 上で複数のプロセスを動作させることは今後の課題である．

ここでは，プロセッサコアの管理とメモリの管理は分離せず，プロセッサコア単位でメモリの管理を行うこととする．しかし，本章で提案する管理方式を Cell/B.E. 以外のアーキテクチャにも適用するならば，メモリの管理方針はプロセッサコアの管理から分離する設計とすることを考えるべきである．

4.2　データ構造とインタフェース

プロセッサコアを管理するためにカーネルが扱うデータ構造を**図 5** に図示する．主要なデータ型は `cpu_group_t` と `cpu_t` の二つで，それぞれプロセッサコアの各種類，各プロセッサコアに対応する．

4.2.1　cpu_group_t 型

PPE/SPE などのプロセッサコアの種類ごとに，`cpu_group_t` 型のデータを設ける．その種類におけるすべてのプロセッサコアを束ねる役割を持つ．プロセッサコア間でプロセスのマイグレーションを行う際には，このデータに含まれるプロセッサコア同士でマイグレーションを行う．それぞれ，`cpu_group_operations_t` 型で定義するインタフェースを持つ．

4.2.2　cpu_t 型

前節で述べたプロセッサオブジェクトに相当し，1 つのプログラムカウンタを持つ各プロセッサコアを表す．スケジューリングがプロセッサコアごとに行えるようにするため，それぞれ独立したランキューを持つ．`cpu_operations_t` 型で定義するインタフェースを持つ．メッセージ交換を異なる種類のコア間で行えるようにするため，メッセージ交換操作のインタフェースを `get_message` や `put_message` として統一している．

4.3　メッセージ交換とメモリ転送

SPE プロセスがどのようにメッセージ交換を行うのかについて述べる．SPE のメッセージ転送も，PPE プロセスと同様，PPE 上のカーネルを経由する形式とする．メッセージの送信元・受信先が PPE-PPE, PPE-SPE, SPE-SPE のどの場合も，いったんカーネルによる処理を経る．カーネルにおける PPE-SPE 間のデータの転送方法として，PPE から，メモリマップされたローカルストレージを読書きする方法を用いる．ローカルストレージ上に設けた，メッセージ交換のための通信領域を利用して，カーネルはメッセージや戻り値などをやりとりする．

プロセス間のメモリ転送も，PPE のカーネル上で行う．メモリ転送を行うカーネルコールの引数には，転送元と転送先それぞれのプロセス番号と仮想アドレスが含まれる．プロセス番号から，そのプロセスが存在するプロセッサコアが特定できるので，カーネルは転送元と転送先のプロセッサコアの組合わせに応じて適切な方法で転送を行う．カーネル内において SPE のローカルストレージを読書きするには，メッセージ転送と同様，メモリマップ I/O の方法を用いる．

4.4　SPE における制約への対処

仮想アドレスを物理アドレスに変換するカーネルコール `sys_umap` がある．しかし，SPE のローカルストレージアドレスをメインメモリの物理アドレスに変換することは，プロセッサの仕組みからしてできない．そこで，そのカーネルコールの代わりに新たなカーネルコール `sys_getbuf`, `sys_putbuf` を設ける．そのカーネルコールは，カーネル内のバッファを貸与し，呼出し側にそのバッファの物理アドレスを返す．カーネルは貸与時または貸与終了時にそのバッファとローカルストレージの内容を同期させる．`sys_umap` を容易に置き換えられるよう，`sys_getbuf` のインタフェースは `sys_umap` に似せてある．

SPE においては，他のプロセスのメモリや，メモリマップ I/O などの，ローカルメモリ以外のメモリに直接アクセスすることができず，カーネルにアクセスを依頼するか，DMA 転送などの方法を用いる必要がある．SPE プロセスがカーネルへアクセスを依頼せずに済むよう，SPE が DMA 転送で自力でアクセスするためのライブラリを整備する．これにより，SPE

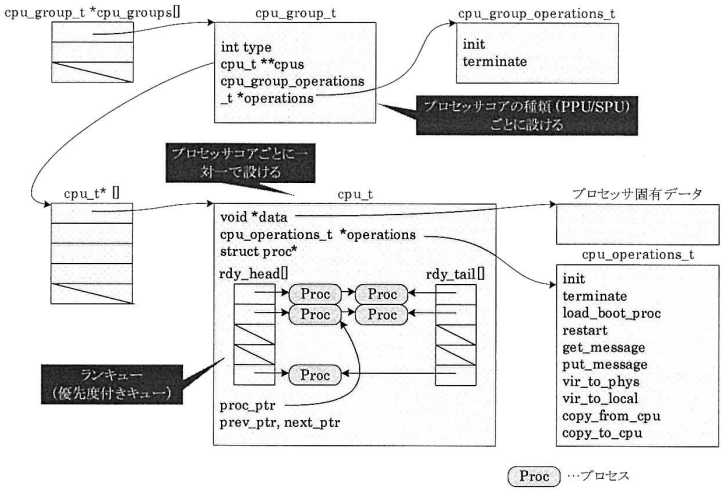

図 5 プロセッサコア管理のためのデータ構造

で動作させるプログラムを SPE に特化した書き方とすることができるようにもする.

5. 実　　装

5.1 開発・実行環境

開発は, Intel x86 機上でクロスコンパイルの形で行う. コンパイルには, Cell SDK 2.1 に含まれる GNU Toolchain を用いる. 実行は, Cell/B.E. 搭載の家庭用ゲーム機 PLAYSTATION 3 で行う. クロスコンパイルで得られた MINIX 実行ファイルを, ブートローダ kboot でブートする.

5.2 実装状況

実装したものを次に挙げる.

- カーネル
  - プロセッサコアの初期化, プロセスディスパッチ
  - マルチコアに対応したプロセススケジューリング
  - 異種プロセッサコア間のメッセージ転送
  - 異種プロセッサコア間のメモリ転送
- SPE プロセス用
  - 実行起動ルーチン (crtso)
  - PPE へのメッセージ転送

実装の結果, SPE 上のプロセスから `printf` などのライブラリ関数や, `open`, `write` など, 多くの POSIX システムコール, MINIX カーネルコールが利用できるようになった. これにより, PPE 用のプログラムとほぼ同様に SPE 用のプログラムを記述することができる.

シグナル, プロセス生成・破棄, SPE におけるメモリ転送ライブラリなどについては実装中である.

6. 評価と考察

6.1 システムコールオーバヘッドの測定

ここでは, 本稿で提案したプロセッサコア間メッセージ交換機構によるシステムコールの, オーバヘッドがどの程度であるのかを測る. 図 2 で示したシステムコールの流れを想定し, 呼出し側における `sendrec` メッセージ転送コールのターンアラウンドタイムを計る.

呼び出される側においては, メッセージ交換以外の処理を一切行わない. 呼び出す側において, 呼び出す直前にカウンタ☆の値を保存しておき, `sendrec` を実行したあと, 制御が再び戻った直後に再度カウンタの値を読み出し, 差を取る. 10 回計測し, 平均を取る. これによって得られる時間を, システムコールにかかるオーバヘッドとする.

呼び出す側, 呼び出される側における, PPE プロセス, SPE プロセスの 4 通りすべての組合わせについて, 計測を行う. ただし, PPE プロセス同士の組合わせは, 同じ PPE 上でのプロセスの組合わせとし, SPE プロセス同士の組合わせは, 異なる SPE 上でのプロセスの組合わせとする.

☆ カウンタとして PPE においてはタイムベースレジスタ, SPE においてはデクリメンタレジスタを用いる. 以下, 同様である.

表 1　システムコールにかかったオーバヘッド

呼び出す側→呼び出される側	μsec
PPE → PPE	49.74
PPE → SPE	64.37
SPE → PPE	43.62
SPE → SPE	44.54

計測した結果を**表 1** に示す．PPE 側の実行時間としては，PPE → PPE の 49.74 μsec に対し，プロセッサコアをまたぐ PPE → SPE は約 29% 悪化している．SPE 側の実行時間としては，SPE → PPE は，PPE → PPE に対して約 12% 短く，SPE → SPE は約 10% 短くなっている．

6.2 メモリ転送性能の測定

システムコールなどにより他のプロセスに処理を依頼する際には，処理するデータを送ったり，処理されたデータを受け取ったりするために，そのプロセスとのメモリ転送を行う必要もある場合が多い．ここでは，本稿で提案したメモリ転送機構の性能を測定する．

メモリ転送にかかる時間として，カーネルコール `sys_vircopy` の実行にかかる時間を計測する．時間の計測方法は 6.1 節の測定と同様である．15 回計測し，中央値を取る．カーネルコールを呼び出す側が，転送元プロセスと転送先プロセスの 2 通り，転送元・転送先が PPE プロセス，SPE プロセスの 4 通り，総合して全部で 8 通りの場合について計る．転送サイズは 4B, 128B, 4KB, 128KB の 4 通りとする．

計測した結果を**表 2** に示す．転送サイズが 4B の列を見ると，PPE からの呼出しよりも，SPE の呼出しのほうがかかる時間が 5〜6μsec 短くなっている．PPE からよりも SPE からのほうがかかる時間が短い傾向は，前節の評価と同様の傾向である．転送相手が SPE であっても，特に長い時間がかかるなどの傾向は見て取れず，PPE でのメモリ転送が全体的に見てやや長時間かかる傾向がある．SPE から PPE へのメモリ転送 (PPE READ from SPE, SPE WRITE to PPE) は，特に長時間かかる傾向がある．

6.3 考　　察

システムコールのオーバヘッドの測定結果を見て分かることは，SPE からのシステムコールにかかる時間は，PPE 上のカーネルを経由するにもかかわらず，PPE でのシステムコールの時間に比べて長くなく，短い場合も多いということである．むしろ，PPE でのシステムコールにかかる実行時間の長さが目立つ．これは，PPE 上でメッセージ転送コールがなされて，カーネルから制御が戻る際の，プロセススイッチによるオーバヘッドがあるためと考えられる．SPE 上でメッセージ転送コールがなされ，カーネルから制御が戻る場合は，カーネルは SPE の動作を再開させるだけで済むので，PPE でのプロセススイッチに比べ処理量が少ない．

PPE → SPE の場合に限っては，PPE → PPE の場合に比べ明らかに悪化して，それぞれの転送コールにおいて約 27%〜約 47%実行時間が延びている．しかし，従来 PPE で行っていた処理を SPE に分担させるので，この程度の悪化分の元を取ることは容易であると考えている．

次に，メモリ転送性能の測定についてであるが，SPE のローカルストレージへのデータ転送に特に時間がかかっている様子はなく，単にメモリコピーの実装が時間の差として表れているようである．PPE でのメモリ転送がやや長時間かかったのは，PPE キャッシュの書出しがメモリコピーの実装に含まれているからである．また，PPE でのメモリコピーの実装はあまり最適化していないが，SPE でのメモリコピーの実装は十分に最適化していた．特に長時間かかった PPE READ from SPE について，PPE キャッシュの書出し処理を除いて再度，実行時間を計測したところ，128KB の転送サイズにおいて，1985.29μsec が 1799.75μsec へと改善した．このことからも，メモリコピーの実装が時間の差として表れていることが分かる．

従来の MINIX を並列化し，各プロセッサコアに処理を割り当てて性能向上を狙うという観点からは，有望な結果が得られた．

7. 関 連 研 究

本章では，すでに行われている，ヘテロジニアスマルチコアプロセッサ向けのシステムソフトウェアの研究についてまとめる．

- **Cell broadband engine, SPE assisted user space device driver**[3)]
  この発表では，Linux の　カーネルモードでの処理，特にデバイスドライバの処理を　SPE に分担させる仕組みを提案，実装している．カーネルモードの処理を SPE に分担させるということで，課題であった，システムソフトウェアの機能の並列化を目指している．この研究では，処理をいったんユーザモードに引き出し，ユーザモードのデーモンを通して SPE と通信を行っている．この方式では，ソフトウェアの構成が複雑になってしまったり，オーバヘッドの増大を招いてしまう．一方，本稿で提案している方式では，OS が直接 SPE を扱うようにすることにより，設計を

表 2　メモリ転送にかかった時間

呼出し側/転送方向/転送相手	4B	128B	4KB	128KB
PPE READ from PPE	49.20	50.80	104.00	1846.78
PPE WRITE to PPE	49.10	50.78	104.01	1842.99
PPE READ from SPE	49.27	51.13	109.21	1985.29
PPE WRITE to SPE	49.55	50.36	95.40	1551.37
SPE READ from PPE	43.27	44.00	89.64	1563.33
SPE WRITE to PPE	42.79	44.52	102.76	1979.71
SPE READ from SPE	42.52	44.04	95.04	1719.11
SPE WRITE from SPE	42.19	44.04	95.26	1735.39

単位: μsec

単純にでき，モード遷移などのオーバヘッドを最低限にすることができる．

- **CellVM: A Homogeneous Virtual Machine Runtime System for a Heterogeneous Single-Chip Multiprocessor**[4]
  この研究では，Java のスレッドを一対一で SPE に割当て，SPE 上の仮想マシンでスレッドを実行するモデルを提案している．仮想マシンのレイヤを挟むことで，PPE, SPE の違いを意識させない実行環境を実現したり，SPE においてキャッシュを実現したりすることができているシステムソフトウェアを工夫し，プログラミングしやすい実行環境を提供している．本研究では，ネイティブで実行するプログラムに対して，プログラミングしやすい実行環境を提供することを目的としている．
- **タスクの動作特性に適合可能なヘテロジニアスマルチコア CPU 向け OS の開発**[5]
  この研究では，本研究と同様，ヘテロジニアスマルチコアに向けた OS の機能として，Cell/B.E. の SPE 管理機構を提案している．ただしこの研究とは，PPE 上の プロセスを SPE 上のプロセスを統合的に考えるかどうかにおいて，根本的に異なる．この研究は Linux をベースにしていて，OS の機能を分担するという考え方ではないが，一方，本研究はマイクロカーネルのヘテロジニアスマルチコアへの適用可能性について考えるものである．

8. ま と め

本稿では，ヘテロジニアスマルチコアプロセッサ Cell/B.E. において，デバイスドライバなどの OS の処理を異種のプロセッサコアでも実行できるようにし，プロセッサコアを有効活用するための仕組みとして，マイクロカーネルを適用することを考え，PS3 MINIX の開発を進めてきた．ヘテロジニアスマルチコアのプログラミングを容易とするため，SPE でも PPE と極力，同様となるようなプロセスモデルを定義した．そして，種類が異なる複数のプロセッサコアを管理する機構の設計と実装について述べた．評価の結果，プロセス間通信の性能は問題ないと分かり，本稿で提案した設計の妥当性を確認した．

今後の課題としては，さらに SPE 用のライブラリの充実を図り，SPE で実行できる処理の種類を増やし，目標である OS 処理の分散を達成できるようにすることである．特に SPE でデバイスドライバを動作させるためには，メモリマップ I/O が扱えるようにすべきである．そして，実践的なプログラムを SPE 上で動作させ，性能の検証を行い，本研究で提案しているマイクロカーネル方式の利用の有効性について詳細に評価したい．プロセッサの動作をモニタするためのインタフェースを整備することも検討したい．

参 考 文 献

1) 野尻祐亮, 並木美太郎: MINIX 3 の Cell B.E. への移植と SPE の管理方式の検討, 情報処理学会研究報告 2008-OS-108, Vol. 2008, No. 35, pp. 83–90 (2008).
2) Tanenbaum, A.S. and Woodhull, A.S.: *Operating Systems Design and Implementation*, Prentice Hall, 3rd edition (2005).
3) 町田浩之: Cell broadband engine, SPE assisted user space device driver, http://tree.celinuxforum.org/CelfPubWiki/JapanTechnicalJamboree13?action=AttachFile&do=get&target=20070222-Cell-Cloop-e.pdf (2007).
4) Noll, A., Gal, A. and Franz, M.: CellVM: A Homogeneous Virtual Machine Runtime System for a Heterogeneous Single-Chip Multiprocessor, *Workshop on Cell Systems and Applications* (2008).
5) 小林良岳, 東賢一郎, 前川守: タスクの動作特性に適合可能なヘテロジニアスマルチコア CPU 向け OS の開発, 情報処理学会研究報告 2007-OS-106, Vol. 2007, No. 83, pp. 55–62 (2007).