2007－OS－106 (8)
2007／8／3

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

小林 良岳　東 賢一朗　前川 守
電気通信大学 大学院情報システム学研究科

概要

ヘテロジニアスマルチコアCPUは，パフォーマンス面だけでなく，その運用コストや性能面に関しても注目され，多様なアプリケーションへの応用が期待されている．例えば，Cell Broadband Engineでは，OSなどの実装に用いる汎用的なコアと，演算に最適化されているコアの2種類のコアが搭載されている．この種のCPUを，多種多様なアプリケーションに適用しようとした時，それぞれのアプリケーションに対して最適な環境を提供するためには，プログラミング面での困難さという問題もあるが，個々のコアを適切に管理し制御しなければならないという問題もある．そこで本稿では，ヘテロジニアスマルチコアCPU向けのOSに必要な機能として，SPEタスク自身がSPEの制御を行うことが可能，SPEの制御方針をカーネルから分離してプログラマが自由に作成可能，SPEを独立に管理し個々のSPEがそれぞれ独自の環境を持つことが可能という3つの特徴を持つ環境の提案し．プロトタイプの実装についても述べる．

An Operationg System for Heterigeneous Multicore CPU to Provide Individual Environment for Each Task

Yoshitake KOBAYASHI　Kenichiro HIGASHI　Mamoru MAEKAWA
Graduate School of Information Systems, The University of Electro-Communications

Abstract

Heterogeneous multicore CPU is one of the solutions of performance and cost problem. For example, Cell Broadband Engine consists of two types of cores on a die. Former one is PowerPC Processor Element which is used for general-purpose such as operating system. Latter one is Synergistic Processor Element which is used for computation. However, this kind of CPU is difficult for programmer not only to programming but also to manage each core. In this paper, we propose a computing environment for heterogeneous multicore processor to manege and control cores as individual. This environment has three features: first, it supports individual core management, second, it has low dependency between different type of cores, third, it is easy to change management policy. Finally, we evaluate the overhead to realize this environment.

1 はじめに

既存のCPU開発では，クロック周波数の向上やCPUの1サイクルで実行できる命令数を示すIPC(Instructions Per Cycle)などをどのようにして向上するかという点に着目していた．特に，CPUアーキテクチャの設計にあたっては，高性能，低消費電力，低価格という用件を満たす必要がある．しかし，プロセッサ設計，プロセッサと主記憶との連携，並列化技法，アプリケーションソフトウェアへの適応などの分野において，これらの用件は相反するものである．

例えば，マルチメディアアプリケーションでは，高速処理が求められるだけでなく，画像処理に特化した複雑な処理を行うための特殊な機能が必要

となる場合がある．よって，CPU に対しては，これらの機能を実現するための機能を追加する必要があり，結果として CPU には多くのトランジスタが必要となる．また，それだけのトランジスタを実装するためには，大きなダイが必要となり構造も複雑となってしまう．さらに，これらの CPU を高速に動作させるためには，それ相応の電力と冷却システムが必要となる．

そこで，プロセッサ開発の問題や，多様なアプリケーションに対応するために，複数の異なるアーキテクチャのコアを 1 つの CPU パッケージにまとめたヘテロジニアスマルチコアが登場した．例えば，SONY，東芝，IBM が共同開発した Cell Broadband Engine Architecture (CBEA) がある [1],[2]．CBEA では，2 つの異なるコアを搭載しており，一つは制御に関する処理を行い，もう一つは演算に特化した処理を行う．このような構成をとることにより，個々のコアでは，シンプルな構造をとることが可能となり，また複数の異なる特性を持ったコアを有効に活用することで，多くのアプリケーションの要求に応えられるようになった．

しかし，既存のオペレーティングシステム (OS) では，個々のコアに対して一つの管理ポリシを適用することしかできず，個々のコアに対して特化した環境を提供することが困難である．また，OS には，個々のコアの上で動作しているアプリケーションに対して，CPU が本来持っているカスタマイズ可能な機能をコントロールする機能がないという問題もある．もし，OS が個々のコアを管理し，同時に動作しているそれぞれのアプリケーションに対して個別の実行環境を提供可能となれば，個々のアプリケーションに最適な実行環境を CPU が提供できる．

そこで，本稿ではヘテロジニアスマルチコア CPU 上で，個々のコアを管理可能とする環境の提案を行う．この環境は，OS のカーネルレベルおよびユーザレベルの双方で実現され，次の 3 つの特徴を持つ．まず 1 つ目の特徴としては，個々のコアを，それぞれ独自したものとして管理し，それぞれのコアが独自の管理ポリシーを持つことが可能であるという点である．2 つ目の特徴としては，異なるタイプのコア間のプログラミングに置ける依存性を最小限にするということである．既存のプログラミングモデルでコアの管理を実現しようとした場合，片方のプログラムの変更がもう一方のコア上で動作するプログラムに対しても影響をあたえることがあるが，これを最小にすることによって，プログラムへの変更が容易になる．3 つ目の特徴

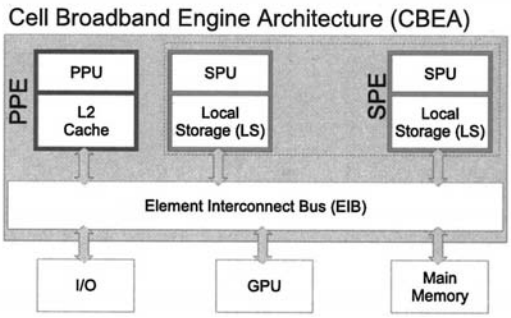

図 1: CBEA の構成

としては，これらのコアを管理するための管理ポリシーを，プログラムの他の部分から分離しておくことによる変更が容易さである．API を統一して管理ポリシーを分離しておくより，管理ポリシーのみの変更を動的に行うことが可能となる．

本稿の構成は以下の通りである．ます，第 2 章では本稿での実装に用いた CBEA と，提案するシステムの構成について述べる．第 3 章では実装について述べる．第 4 章では実装したシステムについて，実装の妥当性に関して評価するとともに，提案システムがどのようなアプリケーションに対して適用可能かどうかを考察する．第 5 章では関連研究について触れ，第 6 章でまとめる．

2 システムの概要

2.1 Cell Broadband Engine Architecture

本章では，提案システムの構成について述べる．まず，システムの実装に用いた Cell Broadband Engine Architecture について延べ，次にシステムの設計について述べる．

図 1 に CBEA の構成を示す．CBEA は，2 つの異なるアーキテクチャのコアを持ち，その組み合わせによりさまざまなアプリケーションに対応可能となるように設計されている．また，コアの数などの構成を変更することによって，必要なパフォーマンスを提供したりコストを抑えることが可能となるように設計されている，

CBEA には，PowerPC Processor Element(PPE) というコアと，Synergistic Processor Element (SPE) と呼ばれるコアの 2 種類のコアが搭載されている．PPE は，OS などが実装されたり，複雑な処理を行うアプリケーションが動作することが想定されている汎用的なコアである．これに対して，SPE は高速な演算が必要なアプリケーションに対して利

用されるとが想定されており，そのための命令が用意されている．これらの SPE を PPE 側で適切に管理することで，さまざまな種類のアプリケーションが要求するパフォーマンスに対して，柔軟に対応することが可能となる．

CBEA では，複数のコアを用いてパフォーマンスを得ることができるという特徴がある反面，異なるアーキテクチャのコアを用いてプログラミングを行わなければならないため，プログラミングの複雑性が指摘されている．この問題を解決するためのプログラミングモデルも提案されている [3].

しかし，これらのプログラミングモデルを用いた場合でも，現状の CBEA では，PPE で動作する OS が，すべての SPE を同じ制御方針で管理しているという問題がある．そのため，既存の OS では，各コアで動作する性質の異なるアプリケーションごとに適したコアごとの制御を設定することができないという問題がある．また，SPE 上で動作するアプリケーションソフトウェアが OS と直接連絡を取り合って SPE 自身の設定を設定したり変更したりする枠組みがないため，各コア上で動作しているアプリケーションソフトウェア自身が主体となってコアの制御を行い，アプリケーションソフトウェア自身の特徴を考慮したコア管理が行うこともできない．

ヘテロジニアスマルチコア CPU の特性上，対応する機器によっては，プロセッサ内にあるコアの構成を柔軟に変更する可能性もある．しかし，既存のオペレーティングシステムの枠組みだけでは，OS が動作する汎用的なコアにプロセッサ内にあるすべてのコアが依存するために，コアの構成が変更になった場合は，アプリケーションソフトウェアも新しい OS や制御するコアに対応しなければならない．よって，OS がヘテロジニアスマルチコア CPU を生かす設計になっていなければ，ヘテロジニアスマルチコア CPU が本来持っている，柔軟性や応用性などの特性を最大限に発揮することができないという問題がある．

そこで，本稿では，CBEA を対象として，次に示す特徴を持つ環境を構築することにより，これらの問題に対処する．

1. 個々の SPE を独立して管理する機構
2. SPE 上で動作するプログラム自身が主体となって SPE の動作環境を決定可能とする機構
3. SPE の管理方針をカーネルと分離することで OS に依存せずに管理方針を切り換えることが可能な機構

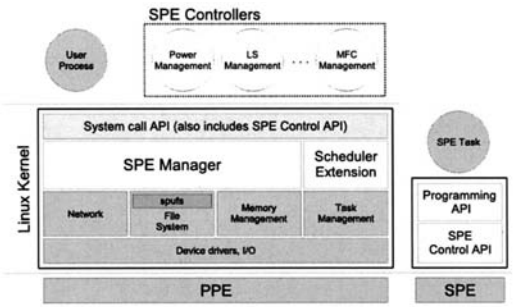

図 2: System Overview

2.2 システムの設計

CBEA 向け Linux[4][5] と libspe[6] によるプログラミングでは，一つのキューによって SPE 上で動く SPE スレッドを管理している．また，PPE 上で動作するプログラムと SPE スレッドの連携によって一つのプログラムが動作するような構成になっている．そのため，複数のプログラムがある特定の SPE を占有したり，特定の SPE 通しの組み合わせで実行されるようなプログラミング環境を実現することができないという問題がある．

CBEA ではサポートされていないものの，各コアの消費電力を，個々のコアのパフォーマンスを細かく変化させることによって最適化する場合，コアの動作速度をなるべく一定に保つ方が，消費電力を抑えることができる．このような理由から，ヘテロジニアスマルチコア CPU では，特定のコアや特定のコアグループの動作をサポートすることが望ましいと考えられる．

各コアごとに独自の環境を提供し，特定のコア同士の組み合わせを可能とするためには，OS はコアごとに管理をする必要がある．また，特定のコアで動作するタスクが，そのタスク自身の実行環境を制御できるようにする必要もある．我々の提案する環境では，個々の SPE に着目した管理を行うことを可能とすることを目標とし，図 2 に示すようなシステムの設計を行った．具体的には，次に示す機能を実現した．

- SPE 制御機構群 (SPE Controller)
- SPE 管理機構 (SPE Manager)
- SPE 制御 API(SPE Control API)

これらの環境は，現在，CBEA に対応した Linux を対象に実装している．SPE 管理機構は Linux カーネルへの拡張として実装し，SPE 制御機構群は PPE 上のユーザレベルで動作するプログラムとして実装した．このような実装を行った理由としては，以

下のものがある．

- SPE を制御するためのレジスタは，OS などの特権ソフトウェアによってのみ設定したり読み込んだりすることが可能である
- 特権ソフトウェアには SPE タスクから SPE 制御機構群への制御依頼情報の受け渡しを行う機能が必要となる

SPE 制御機構群では，個々の SPE に対しての制御ポリシーのみを提供する．実際にどのような制御を行うかについては，SPE 上で動作するプログラムの実行単位である SPE タスクが決定し，その依頼に基づいて SPE 制御機構群がポリシーに基づいた処理を行う．SPE 制御機構群は，図 2 において PPE 上で動作するユーザレベルプロセスとして実装されることが前提となっているが，このような設計とした理由は以下のものがある．

- SPE 制御方針を OS から分離するため
- プログラマが容易に SPE 制御方針の変更を行えるようにするため

このような構成にすることにより，新しい要求が生じても，SPE 制御機構群のプロセスを止め，新しいプロセスを実行することによって対応することが可能となる．また，カーネル側の変更が必要となる可能性が少なくなるだけでなく，プログラマは SPE 制御方針を容易に作成しテストすることが可能となる．

SPE の制御を SPE 自身が行うためには，どのような制御を行うかを特権ソフトウェアに対して伝える必要がある．我々の実装した環境では，制御依頼のための情報は，Mailbox を通して伝えられる．CBEA においてメッセージを伝える機構としては，他にも Memory Flow Controller(MFC) の DMA 転送を使う方法やアシストコールを使う方法もあるが，制御依頼情報の長さを考えた結果，今回の実装では Mailbox を用いることとした．

2.3 SPE タスクと SPE 制御機構群間の通信

SPE タスクが SPE の状態や SPE タスク自身を制御するために，SPE タスクは図 3 に示すような流れで制御依頼メッセージを送る．これらは，次の 3 つのステップに分類できる．1) SPE タスクはコントロールメッセージを Mailbox へ書き込む，2) SPE 管理機構はメッセージを読み取って対応する SPE 制御機構群に渡す，3) SPE 制御機構群が受け取る．制御依頼メッセージには，SPE タスクが SPE をどのように制御したいかという情報が含まれている．

また，個々の SPE が独立した環境を提供できるようにするために，SPE 管理機構では図 4 に示すようなタスクの待ち行列を用いて SPE タスクを管理する．これにより，ある一連の SPE タスクが特定の SPE を用いて実行を行うことが可能となる．

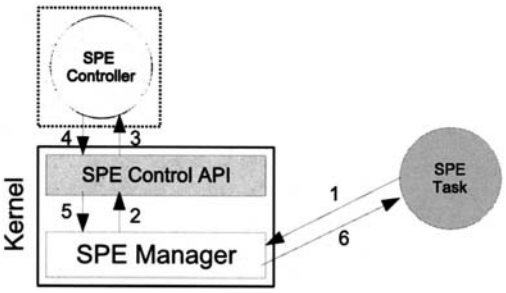

図 3: SPE タスクと SPE 制御機構群間の呼び出し

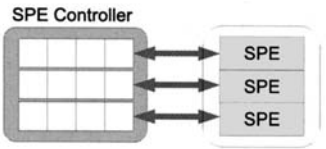

図 4: SPE 管理機構による SPE 割り当て

3 実装

本章では，SPE 制御機構郡および SPE 管理機構などの実装の詳細について述べる．

3.1 SPE 制御 API

表 1 に SPE タスクで利用可能な SPE 制御 API の一覧を示す．これらは，SPE 特有の機能を制御するためのものである．SPE 制御 API は電力制御，MFC 制御，SPE 実行状態制御，バージョン管理などに分類することができる．

プログラマはこれらの API を用いて，SPE 自身や SPE タスクの実行状態をコントロールすることが可能となる．例えば，分散環境において異なるバージョンの CBEA が動作していて，LS のサイズに違いがある場合がある．この時，バージョン番号を SPE タスク自身が取得し，LS のサイズを `get_ls_limit()` を用いて適切に取得し調整することにより，後方互換性を保ったり，LS のサイズを有効に活用することが可能となる．

3.2 制御依頼メッセージ

制御依頼メッセージは，SPE 上で実行されている SPE タスクや SPE 自身を制御するために用いら

表 1: SPE タスクが SPE 制御に用いる API

分類	API 名	行数
電力制御	get_pm_CbeaState()	35
	get_thermal_info()	41
	set_thermal_interrupt()	40
MFC 制御	get_mfc_state()	28
	set_mfc_cntl()	17
SPE 状態制御	get_spe_PrivCntl()	26
	set_spe_SingleStep()	16
	set_spe_RunDefault()	17
	get_spe_RunCntl()	26
	set_spe_RunCntl()	18
バージョン管理	get_spe_version()	27
	get_spe_rivision()	26
LS 制御	get_ls_limit()	32
	set_ls_limit()	26
SPE 制御機構群の管理	spe_default_spe_control()	15
	set_spe()	185

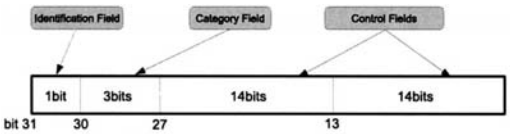

図 5: SPE 管理用のメッセージ形式

れる．SPE タスクが SPE 制御機構群にメッセージを送る際には，Mailbox が用いられる．Mailbox は 32 ビット幅で，表 5 に示すような 4 つのフィールドに分割されている．カテゴリフィールドは，表 1 に示した各分類を示すために用いられる．また，パラメータフィールドは，それぞれの API で必要となるパラメータが設定される．

SPE タスクでは，まず Mailbox に書き込むためのメッセージを作成する．このメッセージ作成のための API として `make_scm()` が用意されており，これを用いる．次に，SPE タスクは Mailbox にメッセージを書き込むが，これには `Write Channel` 命令 [7] か，C/C++拡張 [8] で実装されている `spu_write_out_mbox()` 関数を用いている．

3.3 SPE 管理機構

SPE 管理機構の主な機能は，特権レジスタの設定，SPE タスクと SPE 制御機構群間のメッセージの仲介にある．メッセージの受け渡しは，現在の実装では，スケジューリングのタイミングに合わせて行われている．

SPE 管理機構は，個々の SPE を図 4 に示したように管理するために，初期化の際に SPE の個数だけ要素がある配列 `spe_array` を作成する．`spe_array` における各要素は，`spu` 構造体へのポインタを含み，`spu` 構造体に含まれる `ctx` フィールドが個々の SPE タスクのコンテキストとなっている．`spu` 構造体と `ctx` 構造体は CBEA Linux のものを拡張して実装されている．

表 2: 優先度による応答時間の違い

nice 値	応答時間 (ms)	比
0	1224.49	1
-20	1163.41	0.95

3.4 SPE 制御機構群

SPE 制御機構群は，CBEA Linux においてユーザレベルで動作するデーモンプロセスとして実装されている．アルゴリズムとしては，カーネル内部にも組み込むことが可能であるが，プログラマの実装の容易さと機能変更時の容易さという 2 つの理由により，ユーザレベルで実装している．

ただし，SPE 制御機構群は SPE タスクからの要求に対して，迅速な応答が求められるため，nice 値の調整により他のプロセスより高い優先度で実行されている．実際に，通常の優先度で実行したときと，高い優先で実行したときの応答時間を測定した結果，表 2 のように，およそ 5%程度の性能差があることが確認されている．

3.5 SPE 制御機構群の実装例

プログラマは，デフォルトで提供される SPE 制御機構群の他に，独自の SPE 制御機構群を作成することが可能である．図 6 に SPE 制御機構群の簡単な実装例を示す．各 SPE 制御機構群は，以下に示す 2 つの関数を含まなければならない．

- `sys_set_default_spe()` : SPE と SPE 制御機構群の関連付け
- `sys_set_spe()` : SPE タスクからの制御依頼メッセージの受け取り

図 6 に示した以外の部分は，プログラマによって自由に決定することが可能である．SPE 制御機構群では SPE タスクからのメッセージを受け取った後，そのメッセージの内容を解析し，それに応じた動作を行う．これらの動作には，SPE の電力管理や LS の管理だけでなく，自由な作業を行うような実装をしてもよい．

3.6 SPE 管理機構との連携

本説では，SPE 管理機構と SPE 制御機構群の連携方法について述べる．SPE 制御機構群は，ユーザレベルでデーモンプロセスとして実行される．実

```
main()
{
  int data, SPE_number;
  /* との関連付けSPE */
  sys_set_default_spe(SPE_number);
  for(;;){
    sys_set_spe(&data, SPE_number);
    /* データの解釈      */
    /* 処理の実行        */
    ...
  }
}
```

図 6: SPE 制御機構群の簡単な実装例

```
select_spe_control(data, SPE_number)
{
  switch(SPE_number){
  case SPE0:
    flag_for_mailbox_SPE0 = TRUE;
    wake_up_SPE_manager(SPE_number);
    break;
  case SPE1:
    ...
  }
}
```

図 7: SPE 制御機構群へのメッセージ受け渡し

行開始時に SPE 制御機構群は，個々のプロセス ID と管理する SPE 番号を SPE 管理機構に伝える．これにより，SPE 管理機構は SPE 制御機構群がどの SPE を制御しているかが特定でき，SPE からのメッセージを渡すことが可能となる．

その後，SPE 制御機構群は，SPE タスクからの制御依頼メッセージを受け取るまで実行を停止する．SPE タスクでは，SPE 管理機構を通して制御依頼メッセージの受け渡しを行う．そのため，SPE 管理機構は SPE 制御機構群と同様に SPE タスクからの制御依頼メッセージを待つ．

SPE 管理機構が SPE タスクからのメッセージを受け取った後，SPE 管理機構は SPE タスクが動作している SPE に関連付けられた SPE 制御機構群に対してメッセージを送信し，実行可能に遷移させる（図 7）．これによって，SPE タスクから SPE 制御機構群への制御依頼が可能となる．

4 評価および考察

本章では，実装したシステムに関する評価と考察について述べる．なお，全ての実験は，CBEA 3.2GHz，メモリ 256MB を搭載した SONY PLAYSTATION 3 上に，Fedora Core 5 をインストールして行った．

表 3: メッセージ送信にかかる時間の比較

	時間 (ns)	比
MFC	100.25	1
Mailbox	18.80	0.19

4.1 メッセージ送信の評価

まず，SPE タスクから SPE 制御機構群へのメッセージを送る際にかかるオーバヘッドを測定した．測定項目としては，次の 2 つがある．

1. SPE タスクと SPE 管理機構間のオーバヘッド
2. SPE 管理機構と SPE 制御機構群の間のオーバヘッド

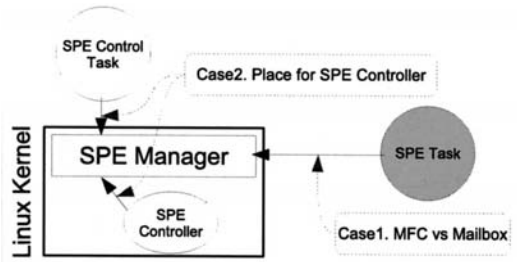

図 8: 評価項目

SPE タスクと SPE 管理機構間のメッセージ送信のオーバヘッドの測定では，メッセージ送信に Mailbox を使う場合と，`spu_mfcdma32()` を用いた MFC 転送の比較を行った（図 8 の Case 1）．我々の実装では，実際には Mailbox を用いているが，MFC を用いても同様の機能が実現可能なため，どちらを用いることが処理速度の面で適切であるかという観点から評価を行った．

また，SPE 管理機構と SPE 制御機構群間のメッセージ受け渡しの比較では，SPE 制御機構群をユーザレベルで実装した場合と，カーネルレベルで実装した場合の両方を比較した (図 8 の Case 2)．応答性能のことを考えた場合，SPE 制御機構群はカーネル内部に実装した方が性能面で有利であるものの，それがどの程度なのかを明確にすることで，今後の実装の方針について検討する．

まず，表 3 に，図 8 の Case 1 について実験を行った結果を示す．実験では，4 バイトの整数を MFC と Mailbox それぞれを用いて送信した．表 3 の結果から，MFC では 5 倍程度の処理時間がかかっていることがわかる．これは，MFC がローカルストレージからメインメモリへのデータの転送を必要

表 4: SPE 制御機構群の実装方法による処理時間の違い

	処理時間 (μs)	比
Kernel	1086.45	1
User Level	1163.41	1.07

とするのに対して，Mailbox ではそのような転送が必要ないためである．しかし，Mailbox では送ることができるメッセージ長に制限があるため，長いメッセージの送受信には，オーバヘッドがかかるものの MFC を用いてメッセージ送信することが必要である．

次に，表 4 に SPE 制御機構群をカーネル内部に実装した場合と，ユーザレベルに実装した場合の処理時間を計測した結果を示す．表 4 の結果から，カーネル内部に SPE 制御機構群を実装した方が，7%程度良い結果が出ていることがわかる．しかし，プログラマに SPE 制御機構群を自由に作成可能とするには，実装の容易さとテスト段階など動作が不安定なものでもシステム全体への影響を与えないようにする必要がある．そのような点を考慮すると，ユーザレベルでの実装でも運用上問題ない程度のパフォーマンスは得られるのではないかと考えられる．しかし，応答性能の安定性などを求められるアプリケーションに対しては，より確実に処理が可能な方式を選択する必要がある．

4.2 本システムの適用範囲に関する考察

提案システムでは，SPE の制御レジスタレベルで各コアを制御することが可能である．そのため，ひとつのプロセッサだけで，各コアごとに特徴の異なる SPE タスクを同時実行することができる．この適応範囲の例として，CBEA 間での互換性挙げることができる．CBEA は今後，シュリンク版 CBEA の開発だけでなく，AV 機器向けや携帯機器向けなど特定の分野に特化した SPE に改良し，コンシューママーケットに進出する予定もある [9]．また．現在 SPE に搭載しているローカルストレージの記憶容量は 256KB だけであるので，今後記憶容量の増加も予想できる．そのため，将来的には多様なバージョンの CBEA や SPE が市場に存在することになる．

この様な状況になった場合，提案システムが提供する SPE のバージョン情報取得を行う制御 API や，ローカルストレージ使用範囲を設定する制御 API を利用して各コアごとに設定を行い，過去のバージョンを対象にした CBEA と次世代 CBEA との間で容易に SPE のプログラムを移植することが可能である．さらに，SPE 制御 API を利用すれば，コアごとに旧世代の SPE 環境と次世代 SPE との環境を提供できるため，OS や CPU アーキテクチャに修正を加えることなく，提案システムだけで後方互換をサポートすることもできる．

例えば，SPE のバージョン管理とローカルストレージの利用制限変更とを用いて，CBEA の後方互換性を保つ場合が考えられる．具体的には，128KB までローカルストレージにアクセスする CBEA に対する処理内容を，256KB までアクセスできる初期バージョンの CBEA 上で実行する場合を想定してみる．この時，バージョン管理とローカルストレージの制御依頼メッセージを利用して，処理を実行する前に動作する SPE の実行環境を設定するために，SPE のバージョン情報を取得するように SPE 制御機構群に依頼する．次に，得たバージョン情報に基づき，どのくらいローカルストレージを利用するかの設定を行い実行する．このような実装方法によって，後方互換性を保つことが可能となる．

5 関連研究

本稿では，CBEA 用 Linux 上で実装を行った．CBEA 用 Linux では，libspe や Spufs[10] といった SPE の制御のための API を提供している [11]．これらは，PPE から SPE を管理するインターフェイスである．現在，libspe では SPE の制御は可能であるが，特定の SPE に限定した制御は困難となっている．また，Spufs では SPE で実行されているスレッドの制御や，LS，Mailbox の取得などが可能であるが，SPE の特権レジスタなどへのアクセスは限定的なものである．さらに，現在提案されているプログラミングモデルでは，PPE が主体となり SPE を制御することが前提としてあり，SPE 自身が SPE の制御を行うことが困難である．この種のプログラミングモデルは，PPE プロセスと SPE スレッドを利用した並列処理に適しているが，SPE スレッドのみでプログラミングを行うことは困難である．我々の提案した SPE 管理機構では，SPE 自身が SPE の制御を行うことに焦点を当てており，また特定の SPE に限定した資源割当てを行うことに焦点を当てている．

現在の実装は，Linux を拡張するという方法で行ったが，SPE 管理機構や SPE 制御機構群の実装方法は，他の OS に適用することは難しくない．例えば，Exokernel[12][13] のような構成の OS には，

本稿で示した手法の適用性は高い．SPE 管理機構のような低レベルで行う動作だけをカーネル内部にまとめ，それ以外の機構はユーザレベルに用意された SPE 制御機構群ライブラリの組み合わせによって実現するという構成をとることができる．

6 まとめ

本稿では，ヘテロジニアスマルチコア CPU 向けの OS 機能として，SPE 管理機構，SPE 制御機構群，制御 API からなる環境の提案および実装を示した．この環境は，1) SPE タスク自身が SPE の制御を行うことが可能である，2) SPE の制御方針をカーネルから分離してプログラマが自由に作成可能である，3) SPE を独立に管理し個々の SPE がそれぞれ独自の環境を持つことが可能である，という 3 つの特徴を持つ．

これにより，個々の SPE をコンフィギュレーションに基づいてまとめて管理したり，特定の番号の SPE についてタスクを割り当てたりすることが可能となる．また，SPE タスクの独立性を高くしていることから，今後は，SPE タスクを要求される資源の設定に基づいて実行可能な分散環境の構築を行うことを考えている．

参考文献

[1] D. Pham, S. Asano, M. Bolliger, MN Day, HP Hofstee, C. Johns, J. Kahle, A. Kameyama, J. Keaty, Y. Masubuchi, et al. The Design and Implementation of a First-Generation CELL Processor. In *Proceedings of the IEEE International Solid-State Circuits Symposium*, pp. 184–185, San Francisco, 2 2005.

[2] B. Flachs, S. Asano, SH Dhong, P. Hofstee, G. Gervais, R. Kim, T. Le, P. Liu, J. Leenstra, J. Liberty, et al. A Streaming Processing Unit for a CELL Processor. In *Solid-State Circuits Conference, 2005. Digest of Technical Papers. ISSCC. 2005 IEEE International*, pp. 134–135, San Francisco, 2 2005.

[3] D.A. Bader, V. Agarwal, and K. Madduri. On the design and analysis of irregular algorithms on the cell processor: A case study on list ranking. In *Proceedings of the 21th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, March 2007.

[4] IBM. *IBM Full-System Simulator User's Guide Modeling Systems based on the Cell Broadband Engine Processor Version 0.02*, 2006.

[5] Barcelona Supercomputer Center. Linux on Cell BE-based Systems. http://www.bsc.es/projects/deepcomputing/linuxoncell/.

[6] IBM, SCEI, and Toshiba. *SPE Runtime Managemnet Library Version1.1*, Feb 2006.

[7] IBM, SCEI, and Toshiba. *Synergistic Processor Unit Instruction Set Architecture Version 1.1*, Aug 2006. http://cell.scei.co.jp.

[8] IBM, SCEI, and Toshiba. *SPU C/C++ Language Extensions Version 2.1*, Oct 2005. http://cell.scei.co.jp.

[9] 林宏雄, 斎藤光男, 増渕美生. Cell Broadband Engine の設計思想. 東芝レビュー, June 2006.

[10] Arnd Bergmann. Spufs: The Cell Synergistic Processing Unit as a virtual file system. Technical report, The IBM Linux Technology Center, 2005.

[11] IBM, SCEI, and Toshiba. *Cell Broadband Engine Linux Reference Implementation Application Binary Interface Specification Version 1.1*, Nov 2006.

[12] Dawson R. Engler, M. Frans Kaashoek, and James O'Toole Jr. Exokernel: An operating system architecture for application-level resource management. In *Proceedings of the fifteenth ACM symposium on Operating systems principles*, pp. 251–266, 1995.

[13] M. Frans Kaashoek, Dawson R. Engler, Gregory R. Ganger, Hector M. Briceno, Russel Hunt, David Mazières, Thomas Pinckney, Robert Grimm, John Janotti, and Kenneth Mackenzie. Application performance and flexibility on exokernel systems. In *Proceedings of the 16th Symposium on Operating Systems Principles*, pp. 52–65, 1997.