Vol.2011-OS-118 No.6
2011/7/27

# マルチコア・メニーコア混在型並列計算機向け OS の構想

佐藤 未来子[†1]　辻田　祐一[†2]
堀　敦史[†3]　並木　美太郎[†1]

ハイパフォーマンスコンピューティング（HPC）の分野では，ペタフロップス級のスーパーコンピュータ開発が現実のものとなり，エクサフロップス，ゼッタフロップスという単位のシステム開発が次の技術課題となっている．本研究では，エクサコンピュータの実現に向けて，今後 HPC の主流となるメニーコアアーキテクチャを有効活用するための基盤ソフトウェアの研究開発を行っている．従来のマルチコアアーキテクチャを汎用コアとして活用しつつ，演算性能や並列性を強化しているメニーコアアーキテクチャを混在させたシステムアーキテクチャを想定し，アプリケーションプログラムからはメニーコアとマルチコアの特性を活かした単一システムとして見せられるメニーコア OS カーネルの実現を目指す．本論文では，メニーコア OS カーネルの検討課題を述べ，プロセス管理，メモリ管理，I/O 管理についての基本設計を述べる．

## Overview of an Operating System for a parallel computer with both Many-core and Multi-core architecture

Mikiko Sato,[†1] Yuichi Tsujita,[†2] Atsushi Hori[†3]
and Mitaro Namiki [†1]

In the field of high performance computing (HPC), the super computer development of the PetaFLOPS class becomes the one of the reality. The development of the ExaFLOPS system is the following technological opportunity. In this study, the infrastructure software to use Many-core architecture that will become the main current of HPC in the future effectively is researched and developed aiming at the achievement of ExaFLOPS system. We are assuming the system architecture to use past Multi-core architecture together with Many-core Architecture. We aim at OS Architecture that can use Many-core and Multi-core from the application program as a single system. In this paper, we show the overview of the OS and discuss on problems of the OS for Many-core architecture system. Moreover we describe the management of processes, threads, memory and also inter-OS communication handling.

## 1. はじめに

プロセスの微細化による単位面積あたりのコア数増加を背景に，スーパーコンピュータは着実に性能向上を実現している．TOP500[1] に掲載される上位システムの性能はペタフロップスを実現しており，トレンドグラフでは 2020 年目前にエクサフロップス達成の可能性が示されている．このエクサフロップスを達成するための技術としてメニーコアプロセッサが注目されている．Intel 社は，Intel Architecture の汎用コアを搭載したメニーコアプロセッサを商用化することを発表している[2]．また，NVIDIA 社も従来の GPGPU アーキテクチャに汎用コアを統合し，Streaming Multiprocessor(SM) 数を増大することによりエクサフロップスの達成を目指している[3]．

本研究では，エクサコンピュータの実現に向けて，今後ハイパフォーマンスコンピューティング（HPC）の分野で主流となるメニーコアアーキテクチャを備えるシステムを対象として，プログラム実行基盤ソフトウェアの研究開発を行っている．従来のマルチコアアーキテクチャを汎用コアとして活用しつつ，演算性能や並列性を強化しているメニーコアアーキテクチャを混在させた「メニーコア混在型並列計算機アーキテクチャ」において，アプリケーションプログラムからは，メニーコアとマルチコアの特性を活かした単一システムとして見せるための OS を実現する．本システム実現のために，マルチコア上の汎用 OS と，メニーコアを管理・制御するためのメニーコア軽量 OS（以下 軽量 OS）とを連携させる機構を設け，汎用 OS と軽量 OS とを適材適所で利用し，汎用 OS による利便性と軽量 OS による演算性能を両立させる．

本論文では，汎用 OS の利便性を維持しつつメニーコア軽量 OS を併用することでシステム全体の性能向上を追求する OS の構想に関して説明し，既存の OS の問題点，本 OS の検討課題，そして，OS の基本設計を述べる．

†1 東京農工大学
Tokyo University of Agriculture and Technology
†2 近畿大学
Kinki University
†3 理化学研究所計算科学研究機構
RIKEN Advanced Institure for Computational Science

## 2. メニーコアプロセッサを搭載するシステムにおける課題

従来のメニーコアプロセッサを搭載するシステムでは，GPGPU に代表されるように，汎用 OS を搭載するホスト計算機が主体となり，並列演算に特化したメニーコア用のプログラムを起動し，実行終了を待ち，メモリに書きこまれた計算結果をホスト計算機で収集するといったスタイルをとる．本方式により，比較的単純で並列度の高いプログラムをメニーコアで並列実行させて高い演算性能を得ているが，ファイルシステムやネットワークへのアクセスといった，汎用 OS で提供されているアプリケーション・プログラミング・インタフェース（API）をメニーコア上のプログラムから直接活用することはできない．例えば，メニーコアにおける計算結果やエラー発生時の状況をディスク装置へ出力したり，フィルタリング処理を施したデータをネットワーク I/F へ出力するなどの手続きは，結果を一度メモリを介してホスト計算機へ転送した上で，別途ホスト側で結果を処理する手続きを実行する必要がある．

本研究では，今後メニーコアプロセッサをさまざまな分野のアプリケーションプログラムから有効活用できるようにすることが重要であると考える．メニーコアプロセッサ上で実行するプログラムを，HPC 系の並列演算に特化したプログラムだけに限定せず，マルチコアや分散システムなどの汎用 OS 上で実行していた並列プログラムもメニーコア上で実行できるようにするためには，プログラム実行基盤が重要課題となる．

## 3. 対象とするアプリケーション

本研究では，以下に示すサーバ系アプリケーションプログラムを扱えるようにすることを目標としている．

**（1）ハイパフォーマンスコンピューティング（HPC）系アプリケーション**

ベクトル処理に向いた並列計算系のアプリケーションプログラムである．

**（2）並列処理と I/O 処理が必要なアプリケーション**

マルチメディア処理系アプリケーションなど，繰り返し行う計算処理とファイルアクセスなどの I/O が織り交ぜて全体の処理性能の高速化が求められるようなプログラムである．

**（3）I/O バウンドなアプリケーション**

Web 系アプリケーションのように，クライアントからの要求を受けてはサーバ側での処理を実行するタイプのプログラムである．

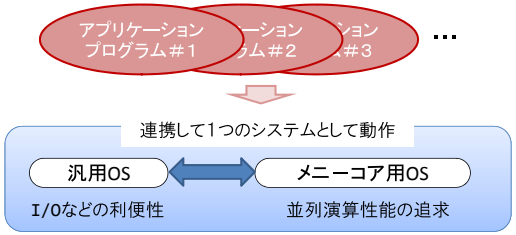


**図 1** OS 連携の概念図
Fig. 1 Cooperation of Operating System

## 4. 開 発 目 標

本研究では，マルチコア上で動作する汎用計算機と，メニーコアを駆使する専用計算機とを単一システムとして協調動作させることにより，メニーコアプロセッサを HPC 分野で有効活用することを目指す．その概念を**図 1** に示す．すなわち，3 章で示したアプリケーションプログラムに対して，I/O バウンドな処理や，複雑な制御系のコードをマルチコア上の汎用計算機を使い，並列性能を追求するベクトル計算機向けの処理をメニーコア上の専用計算機を使うといった，アプリケーションプログラムの特性によってシステム側で適材適所にメニーコアとマルチコアとを使い分ける実行基盤を目標とする．

本章では，本研究で指向するシステムアーキテクチャを示し，OS の開発目標を述べる．

### 4.1 システムアーキテクチャ

**図 2** に，本研究で想定するマルチコア・メニーコア混在型並列計算機アーキテクチャを示す．ハードウェアとして，マルチコアプロセッサとメニーコアプロセッサを混在させたハイブリッド型の計算機とし，これを一つのノードとする．各ノードには，SSD 等の高速ストレージデバイスや Infiniband 等の高速インタフェースを備え，ノード間を高速ネットワークで接続してクラスタ化する．

本研究で対象とするメニーコアプロセッサは，従来の汎用コアの命令セットをサポートしたホモジニアスなメニーコアプロセッサである．従来のマルチコアプロセッサに比べて，コア数は多いがコアの単体性能は低く，コアごとに備える内蔵メモリやキャッシュメモリの容量も少量であるが，一方で，マルチスレッディングによるスレッド並列性やベクトル演算機能を強化したプロセッサを対象としている．

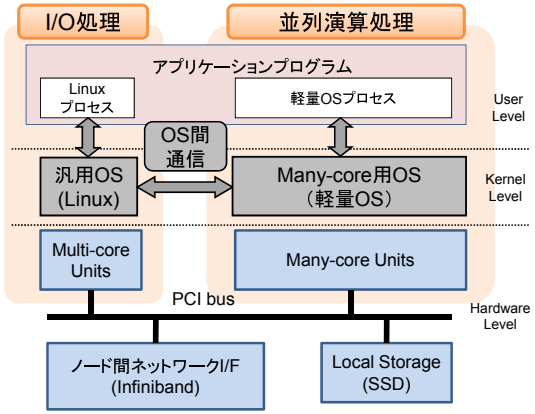


**図 2** マルチコア・メニーコア混在型並列計算機アーキテクチャの概念図
Fig. 2 The system architecture for Multi-core and Many-core

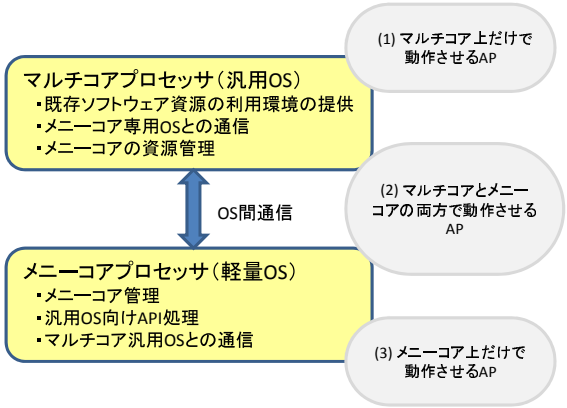


**図 3** OS カーネルの役割分担とプログラム実行モデル
Fig. 3 The roles of operating system on Multi-core and Many-core

### 4.2 OS の開発目標

4.1 節に示したシステムアーキテクチャにおいて，アプリケーションプログラムからはマルチコアとメニーコアの特性を活かせる単一のシステムとして見えるプログラム実行基盤のための OS カーネルの実現を目指す．マルチコア上では Linux 等の汎用 OS を稼働させ，メニーコア上では並列演算処理に特化した軽量な OS（以下，軽量 OS）を稼働させ，これらを連携させことでノード全体を単一システムにみせる．単一システム上で実行するアプリケーションプログラムが，汎用 OS の利便性と軽量 OS の演算性能を適材適所で活用できるようすることを目標とする．

## 5. 設　　計

本章では，マルチコア・メニーコア混在型並列計算機における OS カーネルの設計方針を述べ，本システムにおけるプロセスモデル，アドレス空間管理，汎用 OS・軽量 OS を連携させるための機能について述べる．

### 5.1 設 計 方 針

本研究では，汎用 OS と軽量 OS のハイブリッドな OS 構成とし，軽量 OS ではメニーコアを活用した並列プログラムの高効率実行に専念し，入出力デバイスの管理を含めて本システム全体の管理を汎用 OS が担う．例えば，軽量 OS 側のプログラムから入出力装置に対する処理が要求された場合に，これを汎用 OS が代行するなど，汎用 OS と軽量 OS とが連携して機能を分担しながらアプリケーションプログラムを実行する．以下，各 OS カーネルの役割分担について述べる（**図 3** 参照）．

**(a) 汎用 OS**

汎用 OS では，本システムアーキテクチャ全体を管理しながら，既存のソフトウェア資源の利用環境も提供する．すなわち，従来の汎用 OS と同様に，マルチコア上で実行するプロセスの管理，主記憶の管理を担当する．また，ファイルシステムや，ディスク／ネットワークなどの入出力デバイスの管理を従来と同様に汎用 OS が担当する．
なお，ハイブリッド OS 構成のための機能として，メニーコア上の軽量 OS 用のプログラムを管理・制御する機能や，軽量 OS と連携するための OS 間通信機能を提供する．

**(b) 軽量 OS**

軽量 OS は，メニーコア上の演算コアを制御し，並列プログラムを効率よく実行する役割を担う．なお，前述の汎用 OS で述べたように，本システムアーキテクチャ全体を汎用 OS が管理する方針であるため，軽量 OS は汎用 OS から指示されるコア資源やメモリ資源に基づいて並列プログラムを実行する．また，軽量 OS で実行中のプログラムにおいて，汎用 OS の API を呼び出すことにより汎用 OS が管理する入出力デバイスを利用できるよう，軽量 OS と汎用 OS とが連携するための OS 間通信機能を提供する．

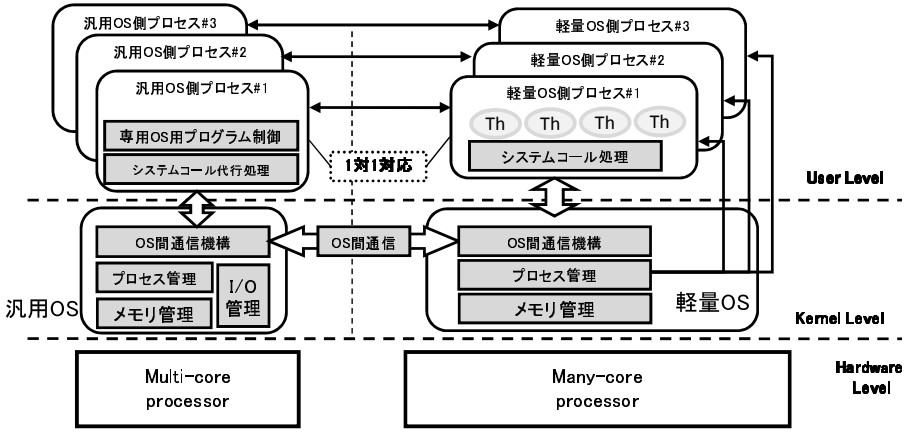


**図 4** 汎用 OS 側プロセスと軽量 OS 側プロセスの対応関係
Fig. 4 Correspondence relation of multi-core process and many-core process

以上のように各 OS が適材適所で機能分担することにより，3 章で述べた三つのアプリケーションプログラムのいずれにも対応可能なシステムとする方針である．

### 5.2 プロセス管理

汎用 OS 側プロセスと軽量 OS 側プロセスの対応関係を**図 4** に示す．本研究のプログラム実行基盤では，マルチコアプロセッサを割り当てる汎用 OS 側のプロセスと，メニーコアプロセッサの複数のコアを割り当てる軽量 OS 側のプロセスを 1 対 1 で対応させて 1 つの実行単位とする．すなわち，本研究のプロセスモデルは，汎用 OS 側で実行するプロセスの手続きの一部をメニーコアプロセッサ上の別プロセスで実行する，というモデルである．

汎用 OS 側のプロセスには，マルチコアプロセッサ上のコア，メモリ資源，I/O 資源を割り当てる．軽量 OS 側のプロセスには，メニーコアプロセッサ上のコアとメモリ資源を割り当てる．軽量 OS 側で I/O 資源を使いたい場合には，5.4 節で示す OS 連携機構を介して，対応する汎用 OS 側プロセスへ割り当てた I/O 資源を活用する．

軽量 OS 側のプロセスでは，軽量な実行実態としてスレッドを生成して，プロセスに割り当てたメニーコアの資源を活用しながら並列実行することで演算性能を追求する．軽量なスレッド管理に関しては，既存研究としてマルチスレッドプログラムの高効率実行を目的としたマルチスレッドライブラリ MULiTh(Userlevel Thread Library for Multithreaded

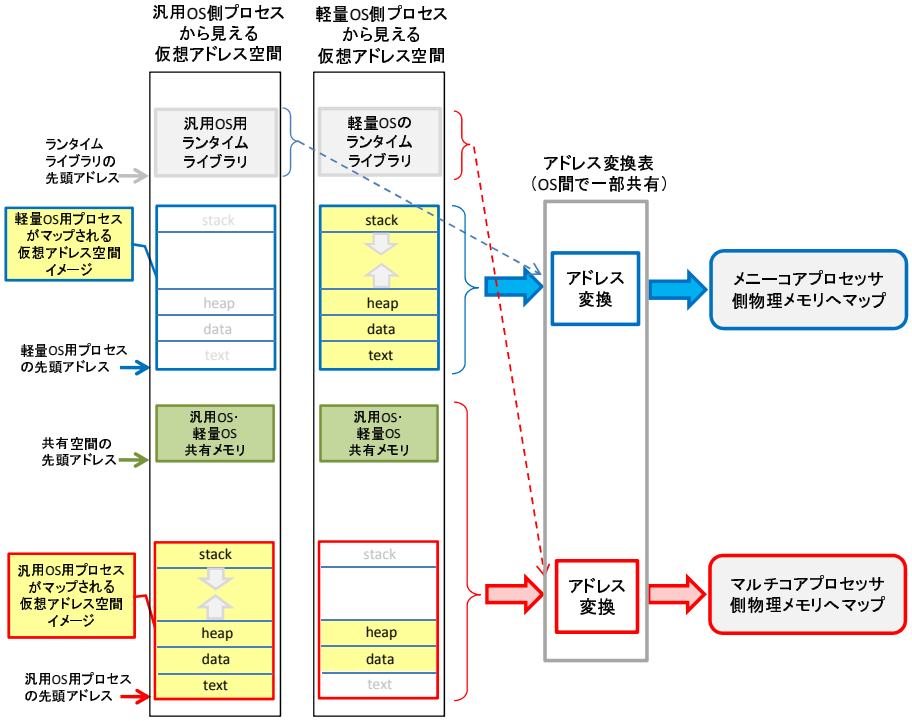


**図 5** 仮想アドレス空間
Fig. 5 Virtual address space

architecture)[4] を提案している．本研究では，MULiTh と同様な様々な軽量 OS をメニーコア上で稼働させて，アプリケーションプログラムの特性に応じて汎用 OS 側プロセスからそれらを活用することで，メニーコアによる性能追求が可能であると考える．ただし，軽量 OS には，5.3 節にて述べるアドレス空間の管理機構と，5.4 節で述べる汎用 OS との連携機構を備えることが必要である．

### 5.3 メモリ管理

5.2 節に示したプロセスモデル実現のために，**図 5** に示す仮想アドレス空間を提案する．基本的に汎用 OS はマルチコアプロセッサ側のメモリ資源を管理し，軽量 OS ではメニーコアプロセッサ側のメモリ資源を管理する．各 OS では，プロセスへ割り当てた仮想アドレ

ス・物理アドレスの対応関係を，従来のページ管理方式と同様にアドレス変換表を用いてページ単位で管理する．

本メモリ管理においては，次に示す各領域の先頭アドレス情報と，仮想アドレスから物理アドレスへ変換するための「アドレス変換表」の一部を，汎用 OS と軽量 OS とで共有し，共有領域の管理をしやすくする．

- 汎用 OS 用プロセスの仮想アドレス空間
- 軽量 OS 用プロセスの仮想アドレス空間
- 汎用 OS 用プロセスと軽量 OS 用プロセス間で共有する空間
- 各 OS で利用するランタイムライブラリ

汎用 OS と軽量 OS との間では，OS 間通信処理などのシステムで使用することを目的とした領域と，汎用 OS 側のヒープ領域とデータ領域を共有するという設計であり，汎用 OS がマルチコアプロセッサ側のメモリ資源を割り当てる．例えば，軽量 OS 側プロセスが汎用 OS 側の仮想アドレス空間の一部にアクセスできるようになることで，次のようなアプリケーションプログラムの処理を，専用のデータ共有領域を別途確保することなく実行できるようになる．

- 汎用 OS のファイルシステムで管理している画像を，軽量 OS 側プロセスから読み出し，並列画像変換を実現する．
- 汎用 OS のネットワーク I/O から受信したデータを，軽量 OS 側プロセスから随時読み出し，データ変換処理を並列実行する．

軽量 OS 用プロセスにおいて，共有領域にアクセスした場合の軽量 OS と汎用 OS の処理の流れの概略を示す．

(1) 共有する領域への最初のアクセスで軽量 OS がページフォルトを検出する．
(2) ページフォルトを起こしたアドレスを確認し，共有領域であることを確認する．
(3) 汎用 OS からアドレス変換情報をもらう．もうすでに共有していれば，つぎの処理へ進む．
(4) アドレス変換情報から，所望のページの汎用 OS が管理するマルチコアプロセッサ側の物理アドレスを得る．
(5) 共有領域の所望のデータへアクセスする．

なお，メモリ管理の実装上の課題としては，PCI バス経由で各 OS が管理するメモリを共有をする場合のアクセス性能を考慮する必要がある．実装方法に関しては，いくつかの方式を実装して，比較提案する方針である．

### 5.4 異種 OS 間の連携機構

汎用 OS と軽量 OS とが連携して機能を分担するための異種 OS 間の連携機構（以下，OS 間連携機構）に関しては，ホモジニアスマルチコア CPU 上での試作提案を行っている[5]．汎用 OS の Linux と，並列実行に特化した独自開発の OS「Future」とを連携させ，並列プログラムの API を汎用 OS が代行する機構を試作し，マルチコア CPU での実装方法について提示している．マルチコアでの試作ではあるが，OS 間連携機構で提供する機能や，OS 間通信インタフェースの仕様については，本研究のメニーコア混在型計算機において流用可能である．本節では，マルチコア CPU で実現した OS 間連携機構の概要を示す．

#### 5.4.1 既存研究における OS 間連携機構の概要

既存研究では，マルチコア CPU 上の一つのコアで Linux を稼働させておき，残りの複数コアを OS Future が管理し，Future 用プログラムを実行するというシステムアーキテクチャをとる．その際に，次に示す OS 間連携機構を用いて OS 同士が連携して単一システムとして動作する．

**Linux からの Future の起動**

Linux から Future のカーネルをロードし，ブートする．ロード及びブートは Linux のユーザープロセスからデバイスドライバを経由して行う．

**Future 用プログラムのロード及び実行**

Linux のファイルシステム上で管理している Future 用プログラムを Linux からロードし，Future 上で実行させる．また，実行時に Future プロセスで使用する CPU コア数やプログラムへ渡すパラメータを引数として指定する．

**Future 用プログラムにおけるシステムコールの代行処理**

Future 用プログラムの実行途中で発行されるシステムコールを Linux に代行させることによって，Linux 側で管理している I/O などの資源を利用可能にする．

上記の各種依頼は，OS 間通信という機構の「メッセージ」という単位でやり取りする．既存研究においては，OS 間通信用の専用メモリ領域にメッセージの内容を格納したうえで，CPU 間割り込みによって各 OS に対してメッセージを通知している．

実装に関して，Linux ではデバイスドライバにより OS 間通信を実現している．表 1 に Linux 側のデバイスドライバのインターフェースの一覧を示す．また，Future 側は表 2 に示

**表 1** Linux 側のデバイスドライバのインターフェース
Table 1 The interface of Linux side device driver

| ハンドラ名 | 機能 |
|---|---|
| read | Future からメッセージを受信（メッセージが来るまでスリープ） |
| write | Future へメッセージを送信 |
| mmap | Future 用領域を Linux プロセスのアドレス空間にマップ |
| ioctl(BOOT_FUTURE) | CPU#1 のリセットベクタを指定し，CPU#1 を起動させる |
| ioctl(CREATE_PROC) | Future のプロセス情報の生成（pid やロード先アドレスなど） |
| ioctl(RUN_PROC) | Future プロセスの実行依頼を送信する |

**表 2** Future 側の OS 間通信用関数
Table 2 The interface of Future side device driver

| ハンドラ名 | 機能 |
|---|---|
| l2f_read | Linux からメッセージを受信 |
| f2l_write | Linux へメッセージを送信 |

す OS 間通信用関数を実装して，割り込みハンドラやカーネル内部の処理で利用している．

#### 5.4.2 メニーコア混在型計算機における OS 間連携機構

本研究のメニーコア混在型システムにおける OS 間連携機構の場合，既存研究における Linux が本研究の汎用 OS に相当し，OS Future が本研究の軽量 OS に相当する．これらの OS 間で交わされる OS 間通信インタフェースの仕様，メッセージのデータ構造，メッセージ送受信のプロトコルなどについては，本研究にて流用可能である．すなわち，本研究の OS 間連携機構では次の機能を提供する．

- 汎用 OS から軽量 OS の起動
- 軽量 OS 用プログラムのロードおよび実行
- 軽量 OS 用プログラムにおけるシステムコールの代行処理

ただし，既存研究では単一のマルチコア CPU および物理メモリ資源を複数の OS で分割・共有するシステム構成での試作である．一方，本研究では PCI バスを介したマルチコアプロセッサ・メニーコアプロセッサ間での実装となり，物理メモリ資源もお互いに備えているモデルである．OS 間連携機構における OS 間通信用の専用メモリ領域で送受信するデータや，軽量 OS 用プログラムおよびそのプログラムで使う汎用 OS と軽量 OS の共有データなどを，お互いの物理メモリ間でどのような実装で共有させるかが重要な課題となる．

## 6. おわりに

本論文では，汎用 OS の利便性を維持しつつメニーコア軽量 OS を併用することでシステム全体の性能向上を追求する OS の構想に関して述べた．従来のマルチコアアーキテクチャを汎用コアとして活用しつつ，演算性能や並列性を強化しているメニーコアアーキテクチャを混在させた「メニーコア混在型並列計算機アーキテクチャ」を提案し，アプリケーションプログラムからは，メニーコアとマルチコアの特性を活かした単一システムとして見せるための OS の設計について述べた．

今後は，プロセスの管理・制御方式，マルチコア・メニーコア間でアドレス空間共有する技術，マルチコア・メニーコア間における OS の連携方式をさらに検討し，実装・評価していく．

**謝辞** 本研究を進めるにあたり，東京大学情報基盤センター長の石川 裕教授から有益なご意見をいただいた．なお，本研究は，科学技術振興機構「JST」の戦略的創造研究推進事業「CREST」における研究領域「ポストペタスケール高性能計算に資するシステムソフトウェア技術の創出」によるものである．

## 参 考 文 献

1) Supercomputer Sites, available from ⟨http://www.top500.org/⟩ (accessed 2011-06-20).
2) Kirk B. Skaugen: HPC Technology – Scale-Up & Scale-Out，ISC'10 Keynote (online)， available from ⟨http://lecture2go.uni-hamburg.de/konferenzen/-/k/10940⟩, (2010.05).
3) Dally, B.: GPU Computing To Exascale and Beyond (online), available from ⟨http://www.nvidia.com/content/PDF/sc_2010/theater/Dally_SC10.pdf⟩ (accessed 2011-06-20).
4) 佐藤未来子，磯部 泰徳，十山 圭介，野尻 徹，入江 直彦，内山 邦男，並木 美太郎：汎用ホモジニアスコアプロセッサにおける OS とスレッドライブラリ，情報処理学会第 20 回コンピュータシステムシンポジウム，ポスターセッション，入手先⟨http://www.ipsj.or.jp/sig/os/index.php?ComSys2008%2Fposter⟩（参照 2008-11-12）.
5) 磯部 泰徳，佐藤 未来子，並木 美太郎：ホモジニアスマルチコア CPU における異種 OS 間の連携機構の試作, 情報処理学会「システムソフトウェアとオペレーティング・システム」第 111 回研究報告, Vol.2009-OS-111, No.17, pp.1–8 (2009.04).