Vol.2012-HPC-134 No.7
2012/6/1

マルチコア・メニーコア混在型計算機における演算コア側資源管理の代行方式

深沢 豪[1,a)]　佐藤 未来子[1,5]　長嶺 精彦[1]　坂本 龍一[1]　吉永 一美[2,5]　辻田 祐一[2,5]　堀 敦史[3,5]
石川裕[4,3]　並木美太郎[1,5]

概要：本研究では，マルチコア・メニーコア混在型計算機において，マルチコアプロセッサ上の管理 OS が，メニーコアプロセッサ上の演算 OS の機能を代行することで，演算 OS を軽量化して並列スレッドプログラムの実行制御に専念させるシステムソフトウェアを提案している．これまでに提案した I/O 管理の代行方式に加え，演算 OS におけるプロセス管理やメモリ管理の機能を管理 OS が代行することにより，演算 OS をより軽量化して並列制御に専念させることを目指す．Intel Xeon X5690 プロセッサを用いて，本論文で提案したハイブリッド OS を試作し，演算プロセスの生成時間を評価した結果，110μsec 程度でプロセス生成が可能であることを確認した．

キーワード：オペレーティングシステム，メニーコアプロセッサ，メモリ管理，プロセス管理

Remote Resource Management on a Hybrid Operating System for a Multi/Many-core Parallel Computer

Go Fukazawa[1,a)]　Mikiko Sato[1,5]　Kiyohiko Nagamine[1]　Ryuichi Sakamoto[1]
Kazumi Yoshinaga[2,5]　Yuichi Tsujita[2,5]　Atsushi Hori[3,5]　Yutaka Ishikawa[4,3]　Mitaro Namiki[1,5]

Abstract: This paper describes the design of an operating system to manage the hybrid computer system architecture with multi-core and many-core processors for Exa-scale computing. In this study, a host operating system on a multi-core processor performs some functions of a lightweight operating system (computing-OS) on a many-core processor. In particular, to ensure that computing-OS execution does not disturb the application program executed on the many-core processor, the functions such as process management, memory management, and I/O management are delegated to the host OS. To demonstrate this design, we made an prototype system equipped with a multi-core processor and a many-core processor using an Intel Xeon X5690 processors. The Linux and original computing-OS are loaded on to each processor and the overhead for creating the process for computing-OS from Linux process is at least 110 μsec.

Keywords: Operating System, Many-core processor, Memory Management, Process Management

[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 AICS
[4] 東京大学
University of Tokyo
[5] 独立行政法人科学技術振興機構 CREST
JST CREST
[a)] fzawa@namikilab.tuat.ac.jp

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の性能は向上をつづけている．TOP500[1] に掲載されているトレンドグラフによれば，2018 年にはエクサフロップスを達成すると予想されている．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 社が開発

© 2012 Information Processing Society of Japan

をすすめている Many Integrated Core(MIC) アーキテクチャのメニーコアプロセッサ（以下，MIC プロセッサと呼ぶ）[4] は，世界で広く普及している Intel 社の x86 アーキテクチャコアを多数集積したホモジニアス構成のプロセッサである．従来の x86 プログラミング環境で構築したシステムソフトウェアやアプリケーションプログラムを移行しやすい一方，従来の x86 のマルチコアプロセッサに比べてコア単体の性能や，コアあたりのキャッシュメモリ容量を抑えた設計とすることでコアの集積数を上げている．このような MIC プロセッサのシステムを従来の汎用 OS で管理する場合，OS 自身の実行性能も下がることが予想され，プロセス管理，メモリ管理，I/O 管理などの OS 実行による擾乱が，アプリケーションプログラムの実行性能にも影響することが懸念される．

本研究では，MIC プロセッサを備えたシステムへの適用を想定して，メニーコアプロセッサの並列演算性能を生かすためのハイブリッド構成の OS 設計を提案している [5]．本研究で想定するマルチコア・メニーコア混在型計算機では，数値演算処理が中心となる並列プログラムを高速に実行することを目的としている．そのために，システムの資源全体を管理するための OS（以下，管理 OS）としてマルチコアプロセッサ上に汎用 OS を搭載し，メニーコアプロセッサ上には演算に特化した専用 OS（以下，演算 OS）を搭載する．そして，マルチコアプロセッサ上の管理 OS が，メニーコアプロセッサ上の演算 OS の資源管理機能の一部を代行し，演算 OS ではメニーコアプロセッサ上で軽量な実行実体であるスレッドを効率よく管理・制御することで並列演算性能の向上を目指している．また，メニーコアプロセッサで実行するアプリケーションプログラムに対して，演算 OS が POSIX 標準 API[6] のサブセットをサポートすることで，プログラミングの利便性も追究している．

これまで，演算 OS の処理軽減のために I/O 管理に関する機能を管理 OS へ代行させる機構について提案した．本論文では，I/O 管理に加えて，演算 OS におけるプロセス管理やメモリ管理の一部機能を管理 OS が代行する方式を提案する．プロセス管理，メモリ管理の代行方式を Intel Xeon のデュアルコアプロセッサシステムで試作し，メニーコアプロセッサの資源の代行管理の実現可能性を確認した．

本論文では，2 章においてシステム構成について述べ，3 章では本研究で提案するハイブリッド OS の設計について述べる．4 章では，本ハイブリッド OS を実現するための OS の構成について述べ，5 章では試作システムでの基本性能の評価にいて述べる．そして 6 章でまとめを行う．

2. システム概要

本章では，本研究で想定するマルチコア・メニーコア混在型計算機のハードウェアとシステムソフトウェアの概要を述べる．

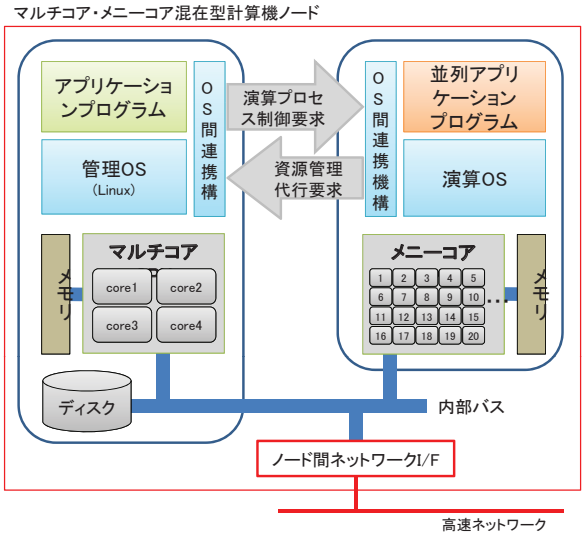

図 1 本研究で対象とするマルチコア・メニーコア混在型計算機
Fig. 1 Hybrid computer system architecture with multi-core and many-core processors.

2.1 ハードウェアの概要

本研究で対象とするマルチコア・メニーコア混在型計算機を**図 1** に示す．本研究では，Intel 社の MIC プロセッサなどのメニーコアプロセッサと，既存のマルチコアプロセッサとを備え，広帯域低遅延の内部バスを介してプロセッサ間通信とメモリ共有が可能なハイブリッド型の計算機を対象としている．このハイブリッド型計算機を一つのノードとし，ノード間を InfiniBand 等の高速ネットワークで接続して大規模な並列計算機クラスタを構成する．

マルチコアプロセッサとメニーコアプロセッサはそれぞれの物理メモリを備え，お互いの物理メモリをアドレス空間へマッピングするために必要となるアドレス空間管理のための例外やページ管理機構などを備えている．また，プロセッサ間割り込みを備えており，共有メモリを用いたマルチコアとメニーコア間の高速通信が行える．

2.2 システムソフトウェアの概要

本研究では，図 1 に示すようにメニーコア上 OS とマルチコア上の OS とが連携しながら，メニーコア向け並列プログラムを実行する．マルチコアプロセッサには，Linux 等の汎用 OS を管理 OS として搭載し，管理 OS がマルチコア・メニーコア混在型計算機のノードの資源を管理する．演算 OS は計算機資源の管理など負荷の重い OS の処理を管理 OS へ委託する．本構成により，演算 OS を軽量化し，メニーコアプロセッサ上で並列プログラムの実行に専念できるようにする．また，マルチコアとメニーコアの OS 間で連携するための機構として，両 OS に「OS 間連携機構」を設ける．本 OS 間連携機構を用いて，管理 OS は演算 OS 向けのプログラムの実行を制御し，演算 OS はメニーコアプロセッサの資源管理を管理 OS へ委託する．

© 2012 Information Processing Society of Japan

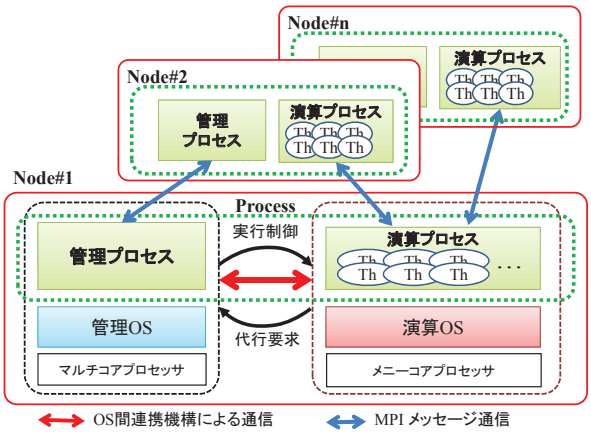

図 2 ハイブリッド OS におけるプロセスモデル
Fig. 2 Process model on the hybrid operating system.

3. ハイブリッド OS の基本設計

本章では，メニーコアプロセッサの並列演算性能を生かすためのハイブリッド OS の基本設計を述べる．

3.1 プロセスモデルの概要

本研究のプロセスモデルの概念図を**図 2** に示す．各ノード上で，管理 OS 上のプロセス（以下，管理プロセス）と，演算 OS 上のプロセス (以下，演算プロセス) が一つの実行単位として「Process」を形成する．管理プロセスには，マルチコアプロセッサ上のコア資源とメモリ資源，I/O 資源を割り当て，演算プロセスには，メニーコアプロセッサ上の複数のコア資源とメモリ資源を割り当て，演算プロセスの I/O アクセス要求は，管理 OS が代行処理する．Process 内では，管理プロセスが演算プロセスの生成，一時停止，再開，停止などを制御し，演算プロセス内では軽量なスレッド（図 2 における “Th”）を多数並列実行させることで，メニーコアプロセッサの並列演算性能を追求する．

なお，本プロセスモデルでは，各ノードの Process 間で従来の MPI 通信を用いて同期をとることを想定している．特に，各演算プロセスのスレッド間においては，高速ネットワークを用いた MPI 通信方式を別途検討している [7]．

3.2 管理 OS による演算プロセスの資源管理

本節では，管理 OS による，演算プロセスのメニープロセッサ上のコア資源，物理メモリ資源，I/O アクセス要求の管理方式について述べる．

3.2.1 コア資源管理

管理 OS は，演算 OS の代わりにメニーコアプロセッサ上の CPU コアを管理する．具体的には，メニーコアプロセッサ上のどの CPU コアをどの演算プロセスへ割り当てるかを管理し，演算プロセスが使用中の CPU コアと空き CPU コアに関するコア数と物理コア番号の情報を管理する．

管理 OS が割り当てた複数の CPU コア上では 3.3 節で述べるスレッドを実行するが，管理 OS が割り当てる CPU コアの組み合わせでも，演算プロセスの処理性能に差異が発生すると予想している．本研究では，まずメニーコアプロセッサの CPU コア管理方式を提案し試作するが，様々な割り当てパターンを想定した評価に基づくコアの割り当てアルゴリズムについては今後検討する方針としている．

3.2.2 メモリ管理

管理 OS は，演算 OS の代わりにメニーコアプロセッサの物理メモリや，演算プロセスの仮想アドレス空間を管理する．代行する機能は次のとおりである．

- メニーコアプロセッサに備える物理メモリを管理し，演算 OS や演算プロセスへ割り当てる．
- 割り当てた物理メモリのアドレスを仮想アドレスへ変換するためのページテーブルを準備する．
- 演算 OS や演算プロセスの実行ファイルを割り当てた物理メモリへロードする．これと同時にスタックやヒープ領域の物理メモリをあらかじめ割り当てる．

本研究で提案するハイブリッド OS では，メニーコアプロセッサ向けの実行ファイルを管理 OS のファイルシステムで管理する．したがって，実行ファイルをファイルシステムから読み出し，解析してページテーブルを準備するまでの作業を，メニーコアプロセッサよりも CPU 性能の高いマルチコアプロセッサ側で行う方が実行性能の面で得策だと考えた．

しかし本設計では，演算 OS でページフォルトを検出した場合に，ページフォルトの処理を管理 OS へ委託することになる．従来のデマンドページング方式で必要に応じてページを割り当てる場合には，ページフォルトの度に演算 OS から管理 OS への通知が発生し，並列演算性能が低下してしまうことが懸念される．そこで本研究では，従来のアクセラレータの利用時の様に，演算プロセス生成時にあらかじめ実行ファイルをロードし，スタックやヒープ領域を十分に割り当てることで，できる限りページフォルト例外を発生させずにプログラムを実行するという方針にした．

3.2.3 I/O 管理

本ハイブリッド OS では，I/O デバイスの管理をすべて管理 OS が担う．演算プロセスから発行される I/O アクセス要求については，文献 [5] で提案した I/O 代行方式により，管理 OS がすべて代行処理する．管理 OS は演算プロセスから，I/O アクセスに必要となるファイルやデバイスの情報や，データやサイズの情報を受け取り，演算 OS の代わりに I/O 処理を行う．代行処理による I/O 性能を向上させるためには，演算プロセスから管理 OS 側への通信ボトルネックを緩和するための工夫を要すると考えている．現状では管理 OS 側の代行処理の並列処理方式 [8] を別途提案しているが，I/O 要求をまとめて管理 OS へ渡すなど演算 OS 側での工夫も今後検討する方針としている．

© 2012 Information Processing Society of Japan

3.3 演算 OS におけるスレッド管理

演算 OS は管理 OS から割り当てられた資源で一つの演算プロセスを実行する．演算 OS の主な役割は以下のとおりである．

- 管理 OS が演算プロセスへ割り当てた物理 CPU コアを管理する．
- 管理する CPU コアを仮想化してスレッドを管理する．
- 演算プロセスの仮想アドレス空間は管理しない．ただしページフォルト例外を検出して管理 OS へ通知する役割を担う．
- 実行中のスレッドからの I/O アクセス要求を管理 OS へ委託する．

本研究の演算 OS は，割り当てられた演算プロセスの資源に対してプロセス切り替えを行わない方針とし，さらに，演算 OS で管理するスレッドも，割り当てた CPU コアをできる限り横取りせずに処理終了まで実行させる．この設計方針により，プロセスやスレッドの切り替え処理オーバヘッドを必要最小限に抑え，CPU コア上のキャッシュメモリや TLB（Translation Look-aside Buffer）のヒットを促すことで性能の向上を狙う．

3.4 OS 間連携機構

本研究では，ハイブリッド OS が連携しながら Process を実行するための通信機構のことを「OS 間連携機構」と呼んでいる．本ハイブリッド OS では，**表 1**, **表 2**, **表 3** に示す通知項目を OS 間で通知しながら，管理 OS は演算 OS 向けのプログラムの実行を制御し，演算 OS はメニーコアプロセッサの資源管理を管理 OS へ委託する．以下，OS 間連携機構の通知項目について説明する．

(1) 演算プロセス管理関連の通知項目

管理 OS は演算プロセスを管理・制御するために，**演算プロセスの生成**，**一時停止**，**再開**，**停止**などのプロセス制御項目を演算 OS へ通知する．これらの通知を受け取った演算 OS が，演算プロセスを生成して起動し，実行状態を管理する．これらの通知は，処理結果を依頼元へ返答するため，双方向の同期通信としている．また，演算プロセスがエラーなどで実行を中止する場合，管理 OS でも演算プロセスへ割り当てた資源を解放する必要があるため，演算 OS は**演算プロセスの状態通知**を単方向通信で管理 OS へ通知する．

(2) メモリ管理関連の通知項目

演算 OS がページフォルトに伴うページ管理が必要な場合や，ヒープ領域やスタック領域を動的に確保したい場合には，管理 OS へメモリ管理を委託する．**ページフォルトの発生**は，ページフォルト例外が発生した状態で演算 OS が管理 OS へ通知するので，双方向の同期通信としている．また，**ページ割り当ての要求**は，演算プロセス実行中の動的メモリ確保用のヒープメモリが不足する場合に備え，演算 OS があらかじめ管理 OS へ要求することを想定し，双方向の非同期通信としている．

表 1 演算プロセス管理関連の通知項目
Table 1 LW-process management notification items.

項目	通知元	通知先	通信タイプ
演算プロセスの生成	管理 OS	演算 OS	双方向同期通信
演算プロセスの一時停止	管理 OS	演算 OS	双方向同期通信
演算プロセスの再開実行	管理 OS	演算 OS	双方向同期通信
演算プロセスの停止	管理 OS	演算 OS	双方向同期通信
演算プロセスの状態通知	演算 OS	管理 OS	単方向通信

表 2 メモリ管理関連の通知項目
Table 2 Memory management notification items.

項目	通知元	通知先	通信タイプ
ページフォルトの発生	演算 OS	管理 OS	双方向同期通信
ページ割り当ての要求	演算 OS	管理 OS	双方向非同期通信

表 3 I/O アクセス関連の通知項目
Table 3 I/O access notification items.

項目	通知元	通知先	通信タイプ
デバイスの open	演算 OS	管理 OS	双方向非同期通信
デバイスの close	演算 OS	管理 OS	双方向非同期通信
デバイスからの read	演算 OS	管理 OS	双方向非同期通信
デバイスへの write	演算 OS	管理 OS	双方向非同期通信

(3)I/O アクセス関連の通知項目

演算プロセスが，ハードディスクやネットワークデバイスへアクセスするときに，演算 OS から管理 OS へ**デバイスの open, close, read, write** といった項目を通知する．これらは，POSIX のインタフェースをそのまま通知することを想定しており，アクセスするデータサイズにより I/O 処理に要する時間が変動する可能性があるため，すべて双方向の非同期通信としている．

4. ハイブリッド OS の構成

管理 OS と演算 OS を実現するためのソフトウェアの構成を**図 3** に示す．また，本ハイブリッド OS で管理するアドレス空間の概念図を**図 4** に示す．二つの OS 間では，通信データの受け渡しに共有メモリを使い，通信開始のトリガーにプロセッサ間割り込み (Inter-processor interrupt, 以下 IPI) を用いて，メモリコピーのない OS 間連携を行う．

4.1 管理 OS の構成

管理 OS は，ユーザレベルの「OS 間連携ライブラリ」と，カーネルレベルの「OS 間連携カーネルモジュール」で構成する．OS 間連携ライブラリが主に管理 OS の資源管理処理を行うことにより，資源管理に伴うカーネルレベル遷移のオーバヘッドを極力削減する方針である．

4.1.1 OS 間連携ライブラリ

OS 間連携ライブラリには次の三つの機能を備える．

© 2012 Information Processing Society of Japan

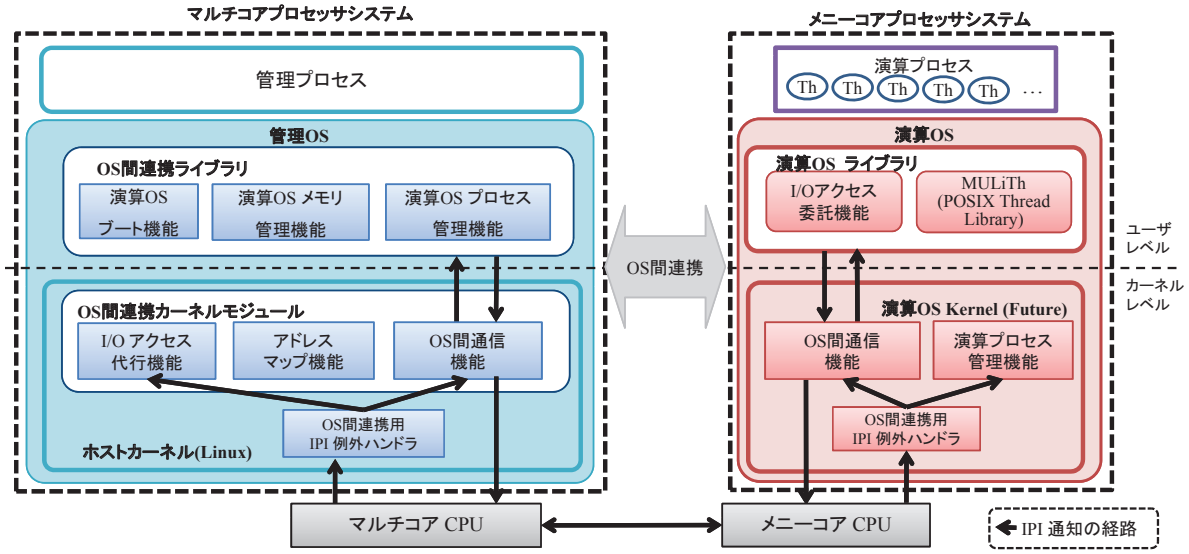

図 3　ハイブリッド計算機システムにおけるソフトウェア構成
Fig. 3　Software structure on multi-core processor and many-core processor.

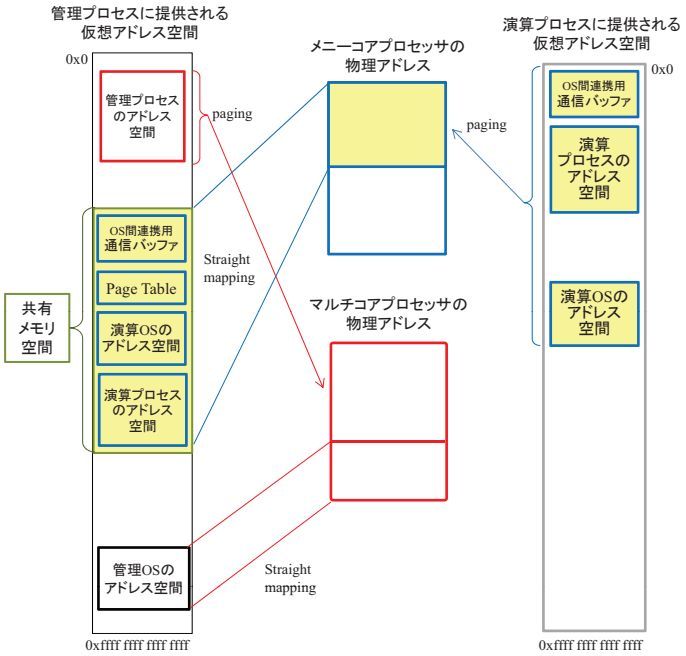

図 4　仮想アドレス空間の概念
Fig. 4　Virtual memory mapping process address spaces to physical memories.

- **演算 OS ブート機能**
  演算 OS のための物理メモリを割り当てて演算 OS をロードし，IPI を発行することでメニーコアプロセッサ上で演算 OS を起動する．
- **演算 OS メモリ管理機能**
  演算 OS のメモリ管理機能を代行する機能であり，ライブラリ内の他の機能ブロックから利用される．次の三つの役割がある．

(a) 物理メモリの割り当て機能
メニーコアプロセッサの物理メモリをブロック（現在は 4MB）に分けて管理し，演算 OS のブートや演算プロセス生成の際に必要なサイズを割り当てる．

(b) 実行ファイルのロード機能
機能 (a) を用いて確保した物理メモリに対して，演算 OS や演算プロセスの実行ファイルを解析してロードする．これと同時に，スタックやヒープ領域のための物理メモリも確保する．

(c) ページテーブル管理機能
ページテーブル用の物理メモリを確保して，機能 (b) で解析した演算プロセスの仮想アドレスと物理アドレスの変換情報を書き入れる．演算 OS からページフォルトが通知された場合に，機能 (a) を用いて物理メモリを確保してページテーブルを更新する役割も担う．

- **演算 OS プロセス管理代行機能**
  4.4 節で示す演算プロセス制御用 API を介して指示される演算プロセスの生成，一時停止，再開，終了などの要求を処理する．また，メニーコアプロセッサの物理コアを管理し，演算プロセス生成時に指定されたコア数を割り当てる．演算 OS メモリ管理機能を呼び出し，演算プロセス実行のためのメモリを準備する．生成した演算プロセスに関して，次に示すコンテキスト情報を管理する．
  – 演算プロセスが使う物理コア情報
  – 演算プロセスへ割り当てた物理メモリ情報
  – 演算プロセス用ページテーブルの先頭アドレス
  – OS 間連携用通信バッファの先頭アドレス
  演算プロセスが終了した場合に，これらのコンテキスト情報を参照しながら資源を解放する．

© 2012 Information Processing Society of Japan

4.1.2 OS 間連携用カーネルモジュール

OS 間連携用カーネルモジュールには次の三つの機能を備える．

- I/O アクセス代行機能
  ディスクやネットワークなど管理 OS 側で管理するデバイスに対して，演算 OS から委託される I/O アクセス要求を代行する機能である．
- アドレス空間マッピング機能
  OS 間連携ライブラリが割り当てたメニーコアプロセッサの物理アドレスを，管理プロセスの仮想アドレス空間へマッピングする機能である．本機能により，OS 間連携ライブラリが管理する演算プロセスのアドレス空間，ページテーブル，OS 間連携機構の通信バッファなどを，OS 間連携ライブラリが直接読み書きできるようにしている．
- OS 間通信機能
  OS 間連携機構に用いる IPI を OS 間連携ライブラリが送受信するための機能である．本機能ではデバイスファイルを介する ioctl システムコールを用いて，ユーザレベルから IPI を送信できるようにしている．IPI の送信先に関しては OS 間連携ライブラリが管理している物理コアの情報で決める．また，演算 OS からの IPI に対しては，OS 間連携機構用 IPI 例外ハンドラを介して受信し，本機能が OS 間連携ライブラリへアップコールする．

4.2 演算 OS の構成

演算 OS は，スレッド管理や POSIX API に対応するためのユーザレベルの「演算 OS ライブラリ」と，管理 OS と連携するための「演算 OS カーネル」で構成する．演算 OS ライブラリが，可能な限りカーネルレベルへ遷移せずに，ユーザレベルでスレッドの並列実行を管理することで並列演算性能を追求する方針である．

4.2.1 演算 OS ライブラリ

演算 OS ライブラリは，従来の汎用 OS 上で開発したプログラムのコードをできる限り改変せずに実行できるようにするために，POSIX の API をサポートするライブラリである．現状では，次の二つの機能を備える．

- POSIX Thread 機能
  演算プロセスで生成するスレッドを管理するための機能である．ユーザレベルに備えるスレッド管理用のライブラリ “MULiTh (User-level Thread Library for Multithreaded architecture)[9]” で，演算プロセスに割り当てられている複数の CPU コア資源をスレッドへ割り当て，スレッドの並列実行を制御する．
- I/O アクセス委託機能
  演算プロセス内のスレッドから呼び出された I/O アクセス要求を，次節の演算 OS カーネルの OS 間通信機能と連携して管理 OS へ通知する機能である．

4.2.2 演算 OS カーネル

演算 OS カーネルには次の二つの機能を備える．

- OS 間通信機能
  演算 OS が管理 OS と連携するために用いる IPI を，演算 OS ライブラリが送受信するための機能である．例えば，演算 OS ライブラリが I/O アクセス要求を管理 OS へ委託する場合，OS 間連携機構の通知内容を共有メモリへ格納し，本機能を用いて IPI を送信することで管理 OS へ委託内容を通知する．また， 管理 OS からの IPI に対しては，OS 間連携機構用 IPI 例外ハンドラを介して受信し，本機能が演算 OS ライブラリへアップコールする．
- 演算プロセス管理機能
  管理 OS から演算プロセスの生成，一時停止，再開，終了などの要求を OS 間連携機構で受信し，プロセスコンテキストを管理，制御する機能である．次の四つの役割がある．

  (i) 演算プロセスのコンテキスト管理機能
  管理 OS から OS 間連携機構で受信した演算プロセスのコンテキストを管理する．コンテキストの内容は，次のとおりである．
  – 演算プロセスが使うコアの個数やコア ID などの物理コア情報
  – OS 間連携用通信バッファの先頭アドレス
  – 演算プロセスのページテーブルの先頭アドレス
  – 演算プロセスのエントリポイントのアドレス

  (ii) ページテーブル設定機能
  4.1.1 節で述べた管理 OS のページテーブル管理機能が準備した演算プロセスのページテーブルを，CPU コアのページディレクトリベースレジスタへ設定する．

  (iii) 演算プロセス実行機能
  演算プロセスの実行開始の準備ができ次第，実行を開始することを管理 OS へ通知し，演算プロセスのエントリポイントのアドレスからプログラムをユーザモードで実行する．

  (vi) 演算プロセス制御機能
  管理 OS から演算プロセスの実行を制御するための一時停止，再開，終了などの要求を受信した場合，MULiTh へ実行状態の変更をアップコールで指示する． 終了の場合には，すべてのスレッドを終了させた後，上記 (i) の演算プロセスのコンテキストを廃棄する．

この他，演算 OS カーネルは，演算プロセスの実行時エラー（例えばアドレスエラー例外など）に伴う例外ハンドリングも行う．例外発生時には，プロセスの実行を終了するが，ページフォルトに関しては管理 OS へ通知してメモリ管理を委託する．

© 2012 Information Processing Society of Japan

表 4　演算プロセス制御用 API
Table 4　The computing-process management API.

API 関数名	仕様	引数項目
lwp_create	軽量プロセスを LWOS 上に生成し実行する．演算プロセス ID を返す.	実行ファイルパス, 利用するコア数, スタックサイズ, ヒープサイズ
lwp_suspend	軽量プロセスを一時停止する.	演算プロセス ID
lwp_resume	軽量プロセスの動作を再開する.	演算プロセス ID
lwp_stop	軽量プロセスの動作を終了させる	演算プロセス ID
lwp_wait	軽量プロセスの終了通知を待つ	演算プロセス ID

4.3　OS 間連携機構の通信方式

本研究では，管理 OS と演算 OS の間での通信データの受け渡しに共有メモリを使い，OS 間連携の開始を IPI で通知する．OS 間連携では，管理 OS と演算 OS のどちらからもユーザレベルでアクセス可能な共有メモリ上にある「OS 間連携用通信バッファ」を介して，データコピーを排除した通信を行う．この方式により OS 間通信オーバヘッドをできる限り抑えている．この OS 間連携用通信バッファ上には，双方向にデータキューを設け，送信と受信を独立したパスで相互に通信できるようにしている．

4.4　演算プロセス制御インタフェースの仕様

管理 OS が提供する演算プロセスの制御用 API を**表 4** に示し，それらを用いて管理プロセスから演算プロセスを生成・制御する疑似コードを**図 5** に示す．管理プロセスから演算プロセスを作成する場合，(a) メニーコアプロセッサ用の実行ファイルを格納してある管理 OS 上のファイルパス情報，(b) 演算プロセスへ割り当てる CPU コアの数，(c) スタック領域のメモリサイズ，そして，(d) ヒープ領域のメモリサイズなどを指定して，「`lwp_create`」を実行する (図中の (1))．この関数を管理 OS の OS 間連携ライブラリで処理し，演算プロセスのための資源を確保し，演算 OS へプロセス生成を通知する．実行途中で「`lwp_suspend`」により一時停止したり，「`lwp_resume`」で再開することができる．演算プロセスの終了を待つ場合は「`lwp_wait`」を使う（図中の (2)）．

5.　演算プロセス管理方式の評価

5.1　試作システム

Intel Xeon X5690 (6 コア, 3.47GHz) プロセッサを 2 個搭載した計算機をマルチコア・メニーコア混在型計算機に見立てて評価環境を構築した．一方のプロセッサをマルチコアプロセッサとして Linux を搭載し，管理 OS の機能を実装した．もう一方のプロセッサをメニーコアプロセッサとして，独自に開発中の演算 OS カーネル “Future” を実装した．Linux のプロセスから `lwp_create` を実行し，Future 上の演算プロセスを生成して実行するまでの実行基盤を試作した．

```
void main() {
    lwpid = lwp_create(              //--- (1)
          < executable file path>,  // (a)
          <number of cores>,        // (b)
          [<stack size>],           // (c)
          [<heap size>] );          // (d)
    lwp_suspend( lwpid );  //suspend process
    lwp_resume(lwpid );    //resume process
    lwp_wait( lwpid );               //--- (2)
}
```

図 5　演算プロセスを制御する疑似コード
Fig. 5　The pseudo code which has controlled computing-process.

5.2　評価と考察

まず，管理プロセスから演算プロセスを生成するまでの時間を評価した．`lwp_create` により，即終了する演算プロセスを管理プロセスが生成するプログラムの実行時間と，参考値として管理 OS と見立てた Linux 上で子プロセス生成（fork-wait）のオーバヘッドを計測した．結果を**表 5** に示す．演算 OS においてプロセスコンテキストを生成して，管理 OS が演算 OS からの返答を受信するまでのラウンドトリップ時間は約 110μsec であり，Linux における子プロセス生成の実行時間は 75μsec であった．`lwp_create` では，実行ファイルの読み込みと解析，メモリへのロード，演算プロセスのページテーブルの作成，OS 間連携機構による演算 OS へのプロセス生成通知などがある．これらの処理時間の内訳を解析したところ，実行ファイルの読み込みと解析，メモリへのロードに全体の 85%を要しており，これがデマンドページング方式の Linux のプロセス生成処理時間との差として表れていると考える．

次に，1 次キャッシュメモリサイズよりも大きい 32KB のローカルデータを宣言して，キャッシュミスとタイマー割り込みを誘発する簡易プログラムで実行時間を比較した．演算プロセスにはあらかじめ十分なローカルメモリを割り当てた．その結果，表 5 に示すとおり提案方式で約 0.5%の優位性を確認した．前述の評価では Linux に比べてプロセス生成時のオーバヘッドが増加したが，演算 OS ではプロセス実行中に OS の資源管理による擾乱が起きず，プログラムの実行に専念できたことが性能向上につながったと考える．本研究では，MULiTh によるユーザレベルでの軽量なスレッド制御でメニーコアの並列演算性能を生かすことが本来の目標であり，OS の擾乱を受けにくいスレッド実

© 2012 Information Processing Society of Japan

表 5 演算プロセス管理方式の基礎評価
Table 5 The evaluation of computing-process management.

マイクロベンチマーク	演算 OS	Linux
演算プロセス生成	110μsec	75μsec
(実行ファイルサイズ)	(0.9KB)	(1.1KB)
32KB データ演算処理	32.41sec	32.46sec
(実行ファイルサイズ)	(1.1KB)	(1.2KB)

行基盤の実現は，並列演算性能の向上に有効に機能すると考える．また，コア単体性能を抑えた MIC プロセッサ上ではメモリ管理などの重たい処理を可能な限り排除する本ハイブリッド OS の構成は，性能向上につながると考える．

6. 関連研究

Bproc[10] や Mosix[11] はホモジニアス Linux クラスタを対象にして，マスターノードでプロセス管理を集約させてシングルシステムイメージを実現している．本研究では，CPU アーキテクチャの異なるプロセッサを対象に，OS 機能を管理 OS へ委託することで演算 OS の負荷を軽減している．BlueGene/L[12] や Shimizu 等 [13] は，計算専用ノードとこの上で実行するプログラムの管理や I/O を代行する管理ノードを備えたクラスタシステムを提案している．計算専用ノードには軽量の OS を搭載し，管理ノードと OS 間で協調する点は同じであるが，本研究では，内部バスを介してプロセッサ間通信とメモリ共有が可能な点を生かし，メモリ管理を管理 OS 側へ委託した点が異なる．

7. おわりに

本論文では，マルチコア・メニーコア混在型計算機において，マルチコアプロセッサ上の管理 OS が，メニーコアプロセッサ上の演算 OS の機能を代行することで演算 OS の軽量化を図り，メニーコアプロセッサは並列プログラムの実行にできる限り専念させるためのシステムソフトウェアアーキテクチャを提案した．これまでに提案してきた I/O 管理に加えて，演算 OS におけるプロセス管理やメモリ管理の一部機能を管理 OS が代行する方式について述べた．Intel Xeon X5690 プロセッサを用いて，本論文で提案したハイブリッド OS を試作し，演算プロセスの生成時間を評価し，管理 OS が演算 OS からの返答を受信するまでのラウンドトリップ時間が 110μsec 程度で実現可能であることを示した．

今後は，メニーコアプロセッサの CPU コアの様々な割り当てパターンを想定した評価に基づくコアの割り当てアルゴリズムや演算 OS ライブラリにおけるスレッド管理などを検討し，メニーコアプロセッサの並列性能を向上させていく．また，演算 OS ライブラリの機能を拡張し，様々なベンチマークプログラムにより本ハイブリッド OS の評価を続けていく．

謝辞 本研究は，科学技術振興機構（JST）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」研究課題「メニーコア混在型並列計算機用基盤ソフトウェア」によるものである．

参考文献

[1] Top500: Supercomputer Sites (online), Available via “http://www.top500.org/” (accessed Mar.30.2012).
[2] Intel: Many Integrated Core (MIC) Architecture - Advanced (online), Available via “http://www.intel.com/content/www/us/en/architecture-and-technology/many-integrated-core/intel-many-integrated-core-architecture.html” (accessed Mar.30.2012).
[3] Dally, B.: GPU Computing To Exascale and Beyond (online), Available via “http://www.nvidia.com/content/PDF/sc_2010/theater/Dally_SC10.pdf” (accessed Mar.30.2012).
[4] Skaugen, K.: Petascale to Exascale - Extending Intel’s HPC Commitment (online), Available via “http://download.intel.com/pressroom/archive/reference/ISC_2010_Skaugen_keynote.pdf” (accessed Mar.30.2012).
[5] 長嶺精彦, 坂本龍一, 佐藤未来子, 下沢拓, 吉永一美, 辻田祐一, 堀敦史, 石川裕, 並木美太郎: メニーコア混在型並列計算機におけるスレッド管理方式, 情報処理学会「ハイパフォーマンスコンピューティング」第 130 回研究報告, Vol. 2011-HPC-132, No. 30, pp. 1–8 (2011.11.21).
[6] the Open Group: POSIX Certified by IEEE and The Open Group (online), Available via ”http://www.opengroup.org/certification/idx/posix.html” (accessed April.30.2012).
[7] Yoshinaga, K., Mugurama, H., Tsujita, Y., Hori, A., Namiki, M., Sato, M., Shimosawa, T., Sawada, T. and Ishikawa, Y.: Realizing Scalable MPI Communication on a Heterogeneous Platform with Multi-Core and Many-Core CPUs, *Proceedings of the Work in Progress Session, PDP2012* (2012).
[8] 深沢 豪，長嶺精彦，佐藤未来子，並木美太郎: マルチコア・メニーコア混在型計算機における資源管理コア向け OS 間連携機構の試作, 情報処理学会 第 74 回全国大会講演論文集, 5J-1 (2012.03.08).
[9] 笹田 耕一, 佐藤 未来子, 河原 章二, 加藤 義人, 大和 仁典, 中條 拓伯, 並木 美太郎: マルチスレッドアーキテクチャにおけるスレッドライブラリの実装と評価, 情報処理学会論文誌, Vol. 44, No. SIG11(ACS3), pp. 215–225 (2003).
[10] Hendriks, E.: Bproc: the beowulf distributed process space, In *Proceedings of the 16th international conference on Supercomputing*, ICS ’02, pp. 129–136 (2002).
[11] Barak, A.: Scalable cluster computing with mosix for linux, In *In Proceedings of Linux Expo’ 99*, pp. 95–100 (1999).
[12] Moreira, J., Brutman, M., Castaños, J., Engelsiepen, T., Giampapa, M., Gooding, T. *et al.*: Designing a highly-scalable operating system: the Blue Gene/L story, In *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, SC ’06 (2006).
[13] Shimizu, M., Yonezawa, A.: Remote process execution and remote file i/o for heterogeneous processors in cluster systems, In *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, CCGRID ’10, pp. 145–154, (2010).

© 2012 Information Processing Society of Japan