Xeon Phi搭載システムの稼働率向上のための マルチタスクオフロードスケジューラ

宮本孝道† 石坂一久† 細見岳生†

高い演算処理性能を有する Xeon Phi コプロセッサが付加された高性能サーバーが登場している．本稿では，オフロードモデルで記述されたタスクをこのような高性能サーバーで複数同時動作させる場合に，サーバーの稼働率を上げてシステム性能を向上させるマルチタスクオフロードスケジューラを提案する．提案するスケジューラは，Xeon Phi の負荷が高い場合に，通常 Xeon Phi で動作するオフロード部分をホストプロセッサでの実行に切り替える機能を有する．評価した結果，Xeon Phi の負荷が高いときに，ホストプロセッサの稼働率が低くなりシステム性能が低下する問題を解消してシステム性能を向上させることができることを確認した．

Offload Scheduler for high System Utilization on Xeon Phi Server

Takamichi Miyamoto,† Kazuhisa Ishizaka† and Takeo Hosomi†

High-performance servers which consist of Xeon processor and Xeon Phi coprocessor appear, and it is important to achieve high utilization of both Xeon and Xeon Phi for high-performance. A popular usage for Xeon Phi is offload model which executes a part of a workload on Xeon Phi. When Xeon Phi server is shared by multiple offload workloads whose Xeon Phi part is dominant, low Xeon utilization problem is occurred. In this paper, we propose an offload scheduler which switches the Xeon Phi execution to the Xeon execution with considering Xeon Phi load for achieving high Xeon utilization. We describe an effectiveness of the proposed method with evaluations.

1. はじめに

プログラムの並列性を利用することで汎用のプロセッサに対して高い演算処理性能を発揮できるメニコアアクセラレータの活用が高性能を必要とする分野で有望視されている．このようなメニコアアクセラレータとしては，GPU を汎用計算に利用する GPGPU（General-Purpose Computing on Graphics Processing Unit）[1),2)] が代表的ではあるが，汎用のプロセッサコアを持つ Intel Xeon Phi コプロセッサ（以下，Xeon Phi）[2),3)] が登場し，様々な分野への展開が期待されている．

従来，メニコアアクセラレータの活用は主に単一タスクで占有利用した場合の高速化が議論されてきた[4),5)]．複数のタスクからメニコアアクセラレータを共有して活用することは，GPGPU のマルチタスク実行機能が不十分であったこともあり，用途が限定的[6)]で十分な議論がなされていなかった．一方，Xeon Phi は，x86 命令をサポートする汎用プロセッサコアを有しマルチタスクをサポートする Linux OS が動作する[3)] ことから，複数タスクでの共有利用が容易となっている．そこで，本論文では Xeon Phi がアクセラレータとして付加された高性能サーバーにおける複数タスク実行時のシステム性能について議論する．

Xeon Phi が付加されたサーバーを活用するためには，プログラムにおける高並列化可能な部分をホストプロセッサから Xeon Phi に切り出して実行するオフロードモデルが一般的である．プログラム中の Xeon Phi に切り出して実行する部分はオフロード部と呼ばれる．オフロードモデルはプログラムの高並列化することができない部分を単一スレッド性能が高いホストプロセッサで実行し，高並列化可能なオフロード部を Xeon Phi で実行するため，ホストプロセッサと Xeon Phi の双方の特徴を活かした効率の良い実行ができる．

マルチタスクをサポートする Xeon Phi では，ホストプロセッサ上の複数のタスクが同時にオフロードを行うことができる．ただし，複数のタスクが無造作にオフロードを行うため，Xeon Phi 上でリソース競合が起こり性能が低下する．特に Xeon Phi のハード

† NEC グリーンプラットフォーム研究所
NEC Green Platform Research Laboratories

ウェアスレッド数を超えるスレッドが起動される過負荷状態では著しい性能劣化が起きることが知られている[7]．このため，オフロードモデルの複数タスクから利用される Xeon Phi のプロセッサリソースを管理する必要がある．

複数タスク実行時のリソース管理方法として，ホストプロセッサと Xeon Phi のプロセッサリソースを静的に空間分割して各タスクに割り当てる方法が考えられる．この方法ではタスクの実行開始から終了までの間，タスクは割り当てられたホストプロセッサと Xeon Phi のプロセッサリソースを占有する．しかし，両プロセッサを占有しているにも関わらず，オフロードモデルで記述されたタスクは同時に一方しか利用できず，もう一方が利用されない時間が発生してしまう．そのため，ホストプロセッサと Xeon Phi のプロセッサ稼働率が低くなってしまう．

Xeon Phi のプロセッサ稼働率が低い問題を解決するためにタスク単位より詳細なオフロード単位で動的にプロセッサリソースを管理する手法が提案されている[7),8)]．これらの手法は，タスクへの Xeon Phi のプロセッサリソース割当をオフロード単位で行い，タスクによる Xeon Phi の占有利用期間をオフロード実行区間に限定する．これにより，複数のタスクからのオフロード実行を効率よくスケジュールして Xeon Phi のプロセッサ稼働率を向上させている．

しかし，オフロードモデルで記述された単一のタスクの Xeon Phi の実行割合は 40%から 91%と多岐に渡ることが知られている[7]．そのため，タスク中の Xeon Phi の実行割合が大きいタスクが多い場合には，Xeon Phi のプロセッサ稼働率は高いがホストプロセッサのプロセッサ稼働率が低い問題が発生する．このため，サーバーのシステム性能を十分に活用できないという問題がある．

一方，x86 ベースのコアから構成される Xeon Phi はオフロード部を標準の C/C++で記述することができるため，オフロード部をホストプロセッサでも実行することができるという特徴を持つ．そこで本論文ではこの特徴を利用し，Xeon Phi の負荷が高い場合はオフロード部を Xeon で実行する様にスケジュールすることでホストプロセッサも効率的に利用する手法を提案する．

本論文の貢献は以下の 2 点である．

- 通常 Xeon Phi で実行されるタスク中のオフロード部分をホストプロセッサでの実行に切り替えるオフロードスケジューラを提案する．
- 処理をモデル化したワークロードで評価を実施したところ，従来 Xeon Phi の負荷が高くホストプロセッサが活用されていないケースにおいて，提案するオフロードスケジューラの導入によりシステム性能が最大 56%向上することを確認した．

以下，第 2 章で本論文が取り扱う問題について述べる．第 3 章で提案するオフロードスケジューラについて述べる．第 4 章で提案する手法を我々が想定するワークロードを用いて評価した結果を示す．第 5 章で関連研究について述べる．第 6 章で本論文のまとめについて述べる．

2. 複数タスクによる Xeon Phi サーバ共有時の課題

本章では，従来研究で十分な議論がなされてこなかった Xeon Phi の付加されたサーバを複数のタスクで共有する場合の課題について詳細に述べる．

2.1 Intel Xeon Phi コプロセッサ[3)]

まず Intel Xeon Phi のアーキテクチャ，ソフトウェア環境について述べる．Xeon Phi 5110P は，x86 プロセッサコアを 60 個持つ．各プロセッサコアは，4 つのハードウェアスレッドをサポートし，512bit のベクトル演算器を持ち，1.053GHz のインオーダー実行で動作する．また，各コアは 512KB の L2 キャッシュを持つ．ピーク単精度演算性能は 2TFLOPS を超える．ホストプロセッサである Xeon プロセッサと比較すると，シングルスレッド性能は低いが，ソケット性能は高いという特徴を持つ．

Xeon Phi のソフトウェア環境として，Manycore Platform Software Stack (MPSS[9)]) が Intel によって提供されている．Xeon Phi 上では Linux OS が動作しているため，複数タスクの並列動作が容易となっている．また，MPSS 環境ではオフロード実行用の管理デーモンが Xeon Phi の特定の 1 コアを占有して動作する．そのため，オフロードで利用できる Xeon Phi のプロセッサコア数は 59 個となっている．

2.2 オフロード実行モデル

図 1 に本論文が対象とするオフロード実行モデルのプログラムコードと実行の様子を示す．本例では，プログラムはホストプロセッサ実行，Xeon Phi へのオフロード実行を逐次的に実行する（同期オフロード）．プログラム中のオフロードされるコードブロック（オフロード部）は，`#pragma offload target(mic)` という指示文によって指定される．オフロード部の内部は，例えば OpenMP によって並列化されることで Xeon Phi 上で並列実行される．単一タスクのホストプロセッサ実行部分とオフロード部の実行割合はアプ

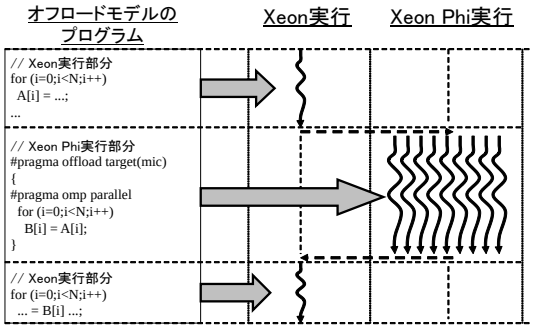

図 1 オフロードモデルによる Xeon と Xeon Phi の利用
Fig. 1 Xeon and Xeon Phi usage by offload model

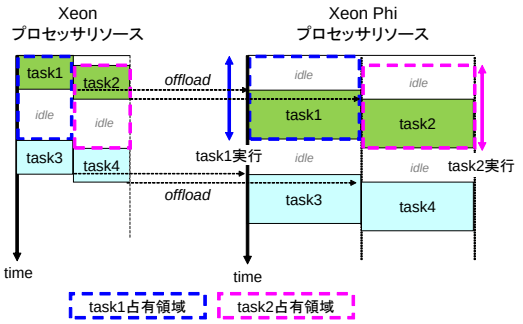

図 2 オフロードタスクに対してタスク単位で静的にプロセッサリソースを分割する手法におけるプロセッサ稼働率が低い課題
Fig. 2 The low processor utilization problem of Xeon and Xeon Phi by static resource partitioning per task for offload task

リケーションによって異なり，タスクのオフロード部の実行割合は 40%から 91%と多岐に渡ることが知られている[7]．

2.3 タスク単位での静的プロセッサ分割

複数のタスクがホストプロセッサ（以下，Xeon）と Xeon Phi を共有利用する場合，例えば 2 個のタスクに対して Xeon の 8 個のプロセッサコアを 4 個と 4 個，Xeon Phi の 59 個のプロセッサコアを 30 個と 29 個に分割し，一方のタスクが Xeon のプロセッサ 4 個と Xeon Phi のプロセッサ 30 個を利用し，もう一方のタスクが Xeon のプロセッサ 4 個と Xeon Phi のプロセッサ 29 個を利用するように管理する手法が考えられる．この手法は，タスク数に合わせて静的に Xeon と Xeon Phi のプロセッサリソースを分割し，各タスクにそれぞれのプロセッサリソースを割り当てることで，各タスクに各プロセッサリソースの一部を占有利用させる．

図 2 に，Xeon と Xeon Phi のプロセッサリソースに静的に分割する手法を適用した場合の例を示す．この手法はタスク全体の実行時間単位で分割された Xeon および Xeon Phi の両プロセッサリソースを占有利用し（図中の点線枠），次のタスクは占有利用が終わるまで開始することができない．そのため，タスクの開始から終了までの間でみると，Xeon あるいは Xeon Phi のプロセッサリソースのどちらか片方が常に利用されない状態となり，プロセッサ稼働率が低くシステム性能を十分に引き出せない問題が存在する．

ここで，単一タスクにおいてホストプロセッサ実行と Xeon Phi へのオフロード実行を並列に動作させることでタスク実行中のプロセッサリソースが使われない時間を軽減する方法が考えられる（非同期オフロード）．しかし，この手法は両実行の間にデータ依存関係がある場合には適用ができない．また，両実行の実行時間に大きな差がある場合もやはり片方のプロセッサリソースが十分に活用されないという問題が残る．

2.4 オフロードスケジューリング

タスク単位で静的にリソース管理を行った場合のプロセッサ稼働率が低い問題に対して，オフロード単位で動的にリソース管理をする手法が提案されている[7),8)]．本論文では，このような手法をオフロードスケジューリングと呼ぶ．

これら手法は Xeon Phi のリソースを利用する最小単位であるオフロード単位でタスクに Xeon Phi のリソースを割り当てる．あるタスクにおける Xeon Phi のリソース未使用期間を他のタスクからのオフロード実行に割り当てることで Xeon Phi のプロセッサ稼働率を向上させる．また，オフロードスケジューリングでは，Xeon Phi が過負荷とならないように，Xeon Phi の負荷状態を考慮して，オフロードを直ちに実行するか，Xeon Phi の負荷が下がるまで，すなわち他のタスクのオフロードが終了するまで待つかを制御する．

オフロードスケジューリングを行った場合の例を図 3 に示す．本例では 4 つのタスクが繰り返し実行されている．オフロード単位でリソース管理が行われるため，例えば Xeon 上でのタスク 1 の終了直後に，タスク 3 を実行することができる．これにより，タスク 1 のオフロード終了直後に，タスク 3 がオフロードを行うことができ，Xeon および Xeon Phi 両方のプロセッサ稼働率を向上させることができている．

2.5 共有時の Xeon プロセッサ稼働率の課題

図 3 の例は，各タスクの Xeon と Xeon Phi の実行時間が等しい場合の例であるため，両プロセッサの稼働率が十分に高い．しかし，Xeon Phi 実行割合が大きいタスクが複数ある場合では，Xeon に十分な負荷がかからずに Xeon のプロセッサ稼働率が低い問題が残る．図 4 に Xeon Phi 実行割合が大きいタスクに対

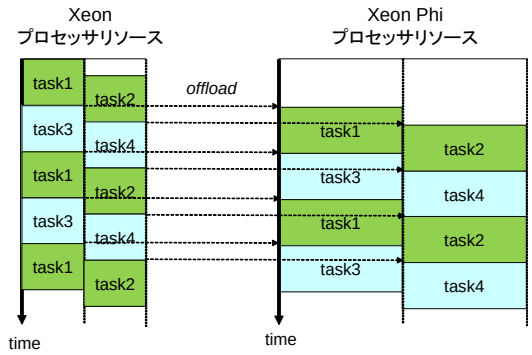

図 3 オフロードスケジューリングの効果
Fig. 3 The effectiveness of managing Xeon Phi resource per offload

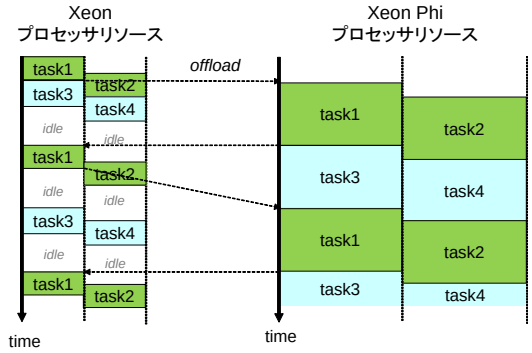

図 4 オフロード単位での Xeon Phi リソース管理における Xeon のプロセッサ稼働率が低い課題
Fig. 4 The low processor utilization problem of Xeon by dynamically managin Xeon Phi resource per offload

して前記手法を適用した場合の動作を示す．Xeon Phi を見ると，オフロード単位のリソース割当で，異なるタスクのオフロード部分を切り替えながら実行し，プロセッサ稼働率は高くなっている．しかし，Xeon Phi を利用中以外のタスクはオフロード実行待ち状態となるため，Xeon には空いているリソースを埋めるだけの十分な処理が存在せず，プロセッサ稼働率が低い問題が残ってしまう．

ここで，一般に広く利用されている 2CPU 構成のサーバーに 1 台の Xeon Phi を搭載した場合で両プロセッサの性能を比較する．Xeon Phi に非常に適したアプリケーションでは Xeon Phi は Xeon の約 10 倍に達するが，それ以外のアプリケーションでは 1.3〜3.5 倍という結果が報告されている[10]．後者の場合では，Xeon はサーバーのシステム性能の 22〜43%を占めており，Xeon Phi に加えて，Xeon を効率的に利用することが重要となる．このため，Xeon の稼働率が低いという問題のある従来のオフロードスケジューリングは十分ではない．

3. 提案手法：ホストプロセッサ実行を考慮したオフロードスケジューラ

本論文では，Xeon Phi 共有時の Xeon のプロセッサ稼働率が低い問題を解決するため，オフロード部を Xeon でも実行するように制御するオフロードスケジューラを提案する．x86 ベースのプロセッサコアで構成される Xeon Phi の特徴の 1 つは，Xeon とプログラムのソースコードを共有できることである．このため，オフロードモデルを用いたプログラムは，オフロード部に対して Xeon Phi 用と Xeon 用のオブジェクトコードを生成し，Xeon Phi を持たないサーバーで実行する場合に，オフロード部は Xeon で実行されるという特徴を持つ．提案するオフロードスケジューラはこの特徴を利用して，Xeon Phi を搭載したサーバーにおいても Xeon Phi の負荷状況に応じてオフロード部を Xeon で実行するように制御することで Xeon プロセッサの稼働率を向上させる．

このようなオフロード部のホストプロセッサ実行を考慮したスケジューリングは，

(1) オフロードスケジューラからの指示に応じてオフロード部の実行プロセッサを切り替える機能を有するタスク，
(2) 各タスクのオフロード部の実行プロセッサを決定するオフロードスケジューラ，

から構成される．

なお，本論文ではオフロードスケジューラと連携しないタスクによる負荷は無視できると想定する．プライベートな計算機センターなどよく管理された環境では無理の無い想定である．

3.1 オフロードスケジューラと連携動作を行うためのタスク実装

オフロード部を含むタスクは各オフロード部前後に Application Programming Interface（API）を呼び出すことによってオフロードスケジューラへ各オフロード区間を通知する．本タスクはオフロードスケジューラから指示される実行プロセッサ情報に応じて各オフロード部を Xeon または Xeon Phi で実行することができる．また，本タスクはオフロードスケジューラからの指示が到達するまではオフロード部の実行を停止する機能を有する．

図 5 にオフロードスケジューラからの指示に応じてオフロード部の実行プロセッサを切り替える機能を有するタスクのプログラム例を示す．タスクはオ

```
1  sched_acquire(taskid, phithreads,
2                xeonthreads, &phiflag);
3
4  // offload part
5  #pragma offload target(mic) if(phiflag)
6    {
7  #pragma omp parallel
8        for (i=0;i<N;i++)
9            B[i] = A[i];
10   }
11
12 sched_release(taskid);
```

図 5 オフロードスケジューラと連携動作を行うタスクのプログラムコード
Fig. 5 The program code example cooperating with offload scheduler

フロード部で実行するプロセッサをオフロードスケジューラに問い合わせるために，オフロード部の直前に `sched_acquire` API を呼び出す（行 1-2）．次に，タスクはオフロードスケジューラからの指示に従ってオフロード部の Xeon Phi 実行（オフロード実行コード）と Xeon 実行を切り替える（行 5-10）．実行を切り替える機能は `pragma` 文中の `if(phiflag)` によって実現される（行 5）．この `pragma` 文中の条件式が `TRUE` の時にオフロード部は Xeon Phi で実行され，条件式が `FALSE` の時にオフロード部は Xeon で実行される．最後に，タスクはオフロード部で利用していたプロセッサリソースの解放指示をオフロードスケジューラに通知するために，オフロード部の直後に `sched_release` API を呼び出す (行 12)．

オフロードスケジューラと連携するための 2 種類の API（`sched_acquire`，`sched_release`）について詳細に述べる．

まず，オフロード部の直前にプロセッサリソース確保要求を行う `sched_acquire` API のフローチャートを図 6 に示す．`sched_acquire` API は以下に示す 6 つのステップから成る．

A1 オフロードスケジューラに対して，問い合わせ種別（リソース確保要求），タスク番号，オフロード部の Xeon Phi 実行の最大スレッド数情報，オフロード部の Xeon 実行の最大スレッド数情報を通知し，ステップ A2 へ．

A2 オフロードスケジューラからの指示を待つ．実行プロセッサ情報，スレッド数情報，スレッドアフィニティ情報を含む指示を受け取ったらステップ A3 へ．オフロードスケジューラからの指示がこなければオフロードスケジューラからの指示を待つためにステップ A2 へ．

A3 実行プロセッサ情報が Xeon Phi であればステップ A4 へ，実行プロセッサ情報が Xeon であればステップ A5 へ分岐する．

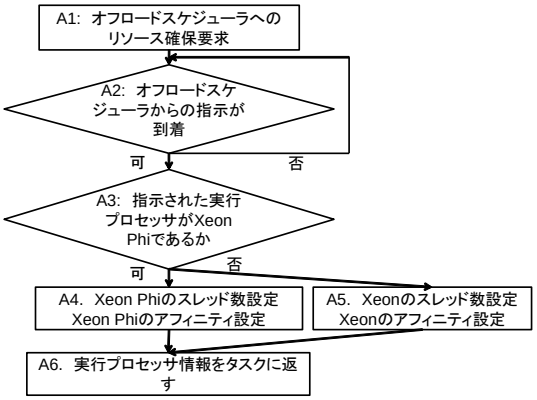

図 6 `sched_acquire` API のフローチャート
Fig. 6 The flowchart of `sched_acquire` API

A4 Xeon Phi のスレッド数の設定と，Xeon Phi のスレッドアフィニティを設定し，ステップ A6 へ．

A5 Xeon のスレッド数の設定と，Xeon のスレッドアフィニティを設定し，ステップ A6 へ．

A6 実行プロセッサ情報をタスクに返す．呼び出し元のタスクの実行に戻る．

次に，オフロード部の直後にプロセッサリソース解放指示を行う `sched_release` API について述べる．`sched_release` API は以下に示す 1 つのステップから成る．

R1 オフロードスケジューラに対して，問い合わせ種別（リソース解放指示），タスク番号を通知する．

本タスクはこれら 2 種類の API をオフロード部前後で呼び出すことによって，オフロードスケジューラに必要な情報を与えることができる．

3.2 オフロードスケジューラ

提案するオフロードスケジューラは，各タスクからのオフロード部実行に対するリソース確保要求に対して Xeon 実行か Xeon Phi 実行かを指示する．この時，それぞれのプロセッサリソースの利用方法も決定して指示を行う．また，各タスクからのリソース解放指示に対してオフロード部実行の停止したタスクが存在する場合には実行停止したタスクに解放されたリソースを割り当てることで実行停止したオフロード部の実行を再開させる．

図 7 に，タスクからのリソース確保要求を受けた時の動作を示す．リソース確保要求を受けた時のオフロードスケジューラの動作は以下に示す 5 つのステップから成る．

S1 Xeon Phi のリソースの空き状況とタスクからの

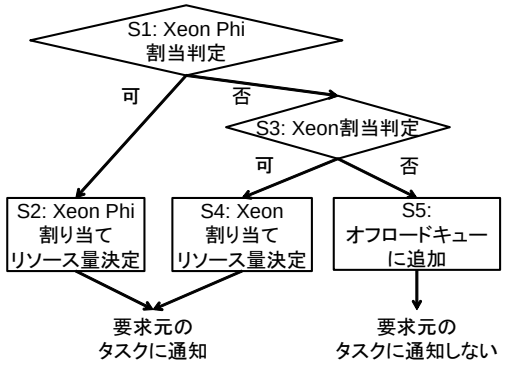

図 7 タスクからのリソース確保要求に対するスケジューラの動作
Fig. 7 The flowchart of offload scheduler corresponding to resource requirement by tasks

要求スレッド数からオフロード部を Xeon Phi に即時割り当てるかを判断する．Xeon Phi に割り当てると決定した場合はステップ S2 へ，割り当てないと決定した場合はステップ S3 へ．

S2 Xeon Phi 割り当てるハードウェアスレッドを決定してタスクに通知する．この時，通知を受けたタスクはオフロード部の実行を行う．

S3 Xeon に割り当てるか Xeon Phi が空くのを待つかを Xeon Phi の負荷情報を元にしたしきい値から決定する．Xeon に割り当てると決定した場合はステップ S4 へ，Xeon Phi が空くのを待つと決定した場合はステップ S5 へ．

S4 Xeon に割り当てるハードウェアスレッドを決定してタスクに通知する．この時，通知を受けたタスクはオフロード部の実行を行う．

S5 スケジューラの管理するオフロードキューにタスク情報を投入する．この時，リソース確保要求元のタスクへの通知を行わないため，リソース確保要求元のタスクはオフロードスケジューラからの通知待ち状態となる．

一方，図 8 に，タスクからのリソース解放指示を受けた時の動作を示す．リソース解放指示を受けた時のオフロードスケジューラの動作は以下に示す 2 つのステップから成る．

S6 オフロードキューにタスクが存在するかを確認する．タスクが存在したらステップ S7 へ．タスクが存在しなければ何もしない．

S7 オフロードキューからタスクを取得する．取得したタスクに対してリソース確保要求時のオフロードスケジューラの処理へ移行する（前記ステップ S1 へ）．これにより，前記ステップ S5 でオフロードキューに投入されて応答待ち状態になっていたタスクの一つが再実行されることになる．

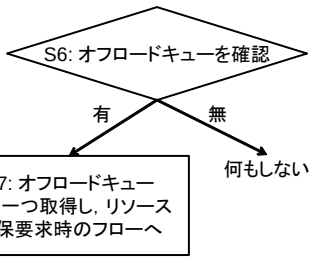

図 8 タスクからのリソース解放指示に対するスケジューラの動作
Fig. 8 The flowchart of offload scheduler corresponding to resource release by tasks

これらの動作のうちステップ S3 とステップ S4 が提案するオフロードスケジューラに特徴的な動作であり，オフロード部を Xeon で実行するための動作である．この動作において，スケジューラは Xeon Phi に空きリソースが無い場合でも，ただちに Xeon に割り当てるのではなく Xeon Phi の空きリソースができるまで待つことを可能としている．これは性能が高い Xeon Phi を優先して利用するため，Xeon Phi に空きが発生した場合に，直ちに次のオフロードを実行するためである．

3.3 オフロードスケジューラの実装

本節では，本論文の評価に利用したオフロードスケジューラの具体的な実装について述べる．

Xeon Phi への割り当て判断 (S1) 今回の実装では，タスクが API で要求したスレッド数に関わらず，Xeon Phi 上にしきい値以上の空きハードウェアスレッドがある場合は，Xeon Phi で実行すると決定する．また，割り当てるハードウェアスレッド数は，要求スレッド数か空きハードウェアスレッド数の最小値である．これも Xeon Phi を優先して利用するためであるが，要求スレッド数が，Xeon Phi の利用可能な総ハードウェアスレッド (236) に対して，十分に少ない（〜32 程度）場合を想定しており，要求スレッド数より少ないスレッドが割り当てられることによって，実行時間が著しく増大することはない．

Xeon への割り当て判断 (S3) 今回の実装では，オフロードキューに溜められた Xeon Phi への割り当て待ちのタスク数がしきい値以上の場合は Xeon へ割り当てる．またオフロードキュー内でのタスクの並び替えは行わない．本論文の評価ではしきい値として 1 を用いた．

3.4 オフロードスケジューラによる効果

図 9 に，第 2 章で示した Xeon のプロセッサ稼働率の低い課題が提案手法によって解決される様子を示

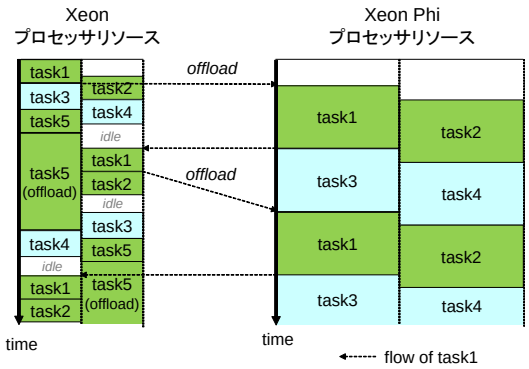

図 9 提案手法による Xeon のプロセッサ稼働率向上
Fig. 9 The effectiveness by switching Xeon Phi execution to Xeon execution for Xeon utilization

表 1 システム環境
Table 1 System environment

	プロセッサ	コア数	動作周波数
システム A	E5-2620	6	2.00GHz
	Xeon Phi 7120P	61	1.238GHz
システム B	E5-2670	8	2.60GHz
	Xeon Phi 5110P	60	1.053GHz

表 2 評価ワークロード
Table 2 Evaluated workloads

	前処理	本処理
ワークロード 1	sGESV （データサイズ可変）	sGESV （4096x4096）
ワークロード 2	sGEMM （データサイズ可変）	sGEMM （4096x4096）

す．本実行例では，タスク 3 とタスク 4 のオフロード時に，オフロードスケジューラは，Xeon Phi を割り当てているが，過負荷を避けるためオフロードキューに投入して，実行を待たせている．一方，タスク 5 のオフロード時には，既に 2 つの待ちオフロードがあるため，Xeon を割り当てている．これにより，Xeon のプロセッサ稼働率の低下を防ぐことができている．

4. 性能評価

本章では，評価環境，提案するオフロードスケジューラの評価結果と考察について述べる．

提案手法は，Xeon のプロセッサ稼働率を向上させることでサーバーのシステム性能を向上させる手法であるから，(1) タスク中の Xeon Phi の実行割合が多い場合，(2)Xeon Phi と Xeon の性能差が少ない場合により効果がある．そこで，評価ワークロードの Xeon Phi の実行時間比率を変えた場合について性能評価を行う．また，ハードウェア性能が異なる複数のシステムと，Xeon Phi への適性の異なる複数のワークロードを用いた評価を行う．

4.1 評価環境

本節では，評価環境について述べる．

Xeon Phi 搭載サーバーとして Xeon 性能と Xeon Phi 性能の比率が異なる場合の振舞いを評価するために 2 種類のシステム環境を用意した．1 つは Xeon E5-2620 が 1 ソケットと Xeon Phi 7120P が 1 カードのシステム（システム A）であり，もう 1 つは Xeon E5-2670 が 1 ソケットと Xeon Phi 5110P が 1 カードのシステム（システム B）である．**表 1** に各システムにおける各プロセッサのコア数と動作周波数をまとめる．

評価に用いたワークロードは Xeon で実行される前処理とオフロード可能な本処理で構成され，前処理の実行後に本処理の実行を行う処理を 300 回繰り返すものを用いた．前処理と本処理は Intel Math Kernel Library(MKL)[11] から行列積を行う sGEMM と連立 1 次方程式の解を求める sGESV を用いて実現した．前処理と本処理を sGESV で実現したもの（ワークロード 1）と sGESMM で実現したもの（ワークロード 2）の 2 種類を用意した．本処理のデータサイズは 4096x4096 固定とし，前処理のデータサイズは単一タスク実行時の前処理の実行割合を変えるために複数のパターンを用意した．Xeon で実行される前処理の実行スレッド数は 1 スレッドとし，本処理の実行スレッド数は Xeon Phi 実行時に最大 32 スレッド，Xeon 実行時に 1 スレッドとした．各ワークロード中の本処理には第 3 章で説明したオフロードスケジューラと連携動作を行うためのタスク実装を施している．**表 2** に評価ワークロードをまとめる．

各ワークロードに対して複数タスク実行した場合に同時に複数タスクが実行されている状態のスループット性能を従来手法と提案手法のそれぞれで計測した．従来方式はオフロードごとに Xeon Phi の実行開始タイミングとスレッドアフィニティの制御を行った方式[8] である．提案手法はオフロードごとに Xeon 実行と Xeon Phi 実行を Xeon Phi 用のオフロードキューに溜められたタスク数に応じて切り替えを行う手法である．ただし，本評価では Xeon Phi 用のオフロードキューにタスクを溜めずに Xeon Phi のリソースに空きが無い場合には即時 Xeon 実行を指示する．

従来手法 各タスクはオフロード単位で Xeon Phi のプロセッサリソースの重なりが無いように制御を行う．各オフロードの実行スレッドは近傍のハードウェアスレッドに集中するようにアフィニティ

表 3　各システム上でのワークロードごとの Xeon に対する Xeon Phi の素性能
Table 3　Xeon Phi performance against Xeon using sGESV or sGEMM on Xeon E5-2620 and Xeon Phi 7120P system or on Xeon E5-2670 and Xeon Phi 5110P system

	sGESV	sGEMM
システム A $\left(\frac{\text{Xeon Phi 7120P}}{\text{Xeon E5-2620}}\right)$	2.01	8.56
システム B $\left(\frac{\text{Xeon Phi 5110P}}{\text{Xeon E5-2670}}\right)$	1.14	4.93

設定を行う．

提案手法　各タスクのオフロード部に対して Xeon 実行か Xeon Phi 実行を制御する．Xeon Phi の全てのプロセッサコアが利用中の場合にオフロード部を Xeon 実行に切り替える．Xeon Phi 実行の場合はプロセッサリソースの重なりが無いように制御を行い，各オフロードの実行スレッドは近傍のハードウェアスレッドに集中するようにアフィニティ設定を行う．

4.2　予 備 実 験

各システム上における sGESV と sGEMM の複数実行時の Xeon のスループット性能と Xeon Phi のスループット性能のそれぞれを予備実験によって求める．予備実験用ワークロードは第 4.1 節で述べたワークロードの前処理を行わず，本処理を Xeon で実行するものと Xeon Phi で実行するものをそれぞれ用いた．Xeon 性能（Xeon 素性能）は 1 スレッド実行のワークロードを 1 ソケットあたりのハードウェアスレッド数だけ複数実行した場合のスループット性能値を計測した．Xeon Phi 性能（Xeon Phi 素性能）は 32 スレッド実行のワークロードを 8 個同時に実行した場合のスループット性能値を計測した．

表 3 に本予備実験から得られた Xeon 素性能に対する Xeon Phi 素性能の比を示す．システム A では Xeon E5-2620 に対する Xeon Phi 7120P のスループット性能は sGESV では 2.01，sGEMM では 8.56 となった．システム B では Xeon E5-2670 に対する Xeon Phi 5110P のスループット性能は sGESV では 1.14，sGEMM では 4.93 となった．

Xeon と Xeon Phi の両プロセッサが十分に利用された場合，理論的なシステムの合計性能（以下，理想システム性能）は，これらの素性能の和になると考えられる．そのため，システム A の理想システム性能は sGESV で 3.01，sGEMM で 9.56，システム B の理想システム性能は sGESV で 2.14，sGEMM で 5.93 となる．

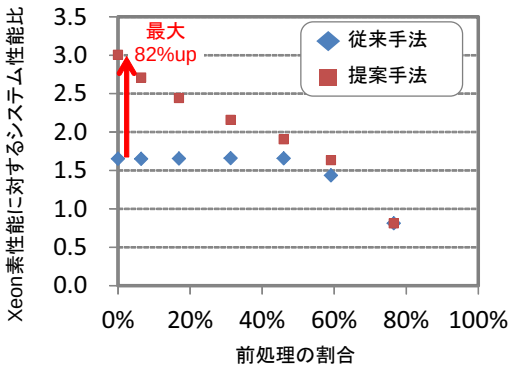

図 10　システム A でのワークロード 1(sGESV) の Xeon 素性能に対するシステム合計性能　提案手法は，前処理の割合が少ない場合に Xeon の空きプロセッサを効果的に利用することで，従来手法に対してシステムの合計性能を向上させている
Fig. 10　**Total system performance of Workload1 on SystemA**　The proposed scheme imporves total system performance when ratio of initialization phase is small.

本予備実験からワークロードごとに Xeon Phi で実行することによる効果が異なることと，Xeon と Xeon Phi の構成ごとに効果が異なるという 2 点がわかる．

4.3　16 タスク同時実行時の評価結果

本節では，Xeon と Xeon Phi に対して十分に負荷をかけるために必要となる 16 タスクを同時に実行した場合の従来手法と提案手法の評価結果について述べる．

図 10 にシステム A 上で前処理の実行割合を変えたワークロード 1 を 16 個同時実行した場合の従来手法と提案手法の評価結果を示す．図 10 は横軸にワークロードに占める前処理の実行割合を示し，縦軸に評価システムにおける Xeon 素性能を 1 とした時のシステム全体で得られたスループット性能比を示す．本評価では，前処理の実行割合が 0%，6%，17%，31%，46%，59%，77%となるポイントを評価した．

本評価結果から，前処理の実行割合が 40%より小さい場合，オフロード部分を Xeon で実行する提案手法は従来手法に対して 30%から 82%の性能向上が得られた．一方で，前処理の割合が 40%以上の場合には，元々Xeon での実行割合が大きいため，従来手法でも Xeon のプロセッサ稼働率が十分に高い．このため提案手法との差は大きくない結果が得られた．本結果から，提案手法はタスク中の Xeon Phi の実行割合が多い場合により効果があることが確かめられた．

4.4　システムとワークロードの組合せごとの評価結果

従来手法に対する提案手法の性能向上が最大となる

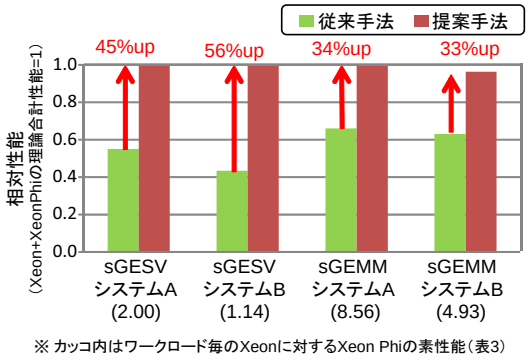

図 11 **ワークロード毎の相対性能** Xeon に対する Xeon Phi の素性能が低いワークロードで提案手法の効果が大きい．また全てのワークロードで提案手法によりシステムの理論合計性能の約 95%以上が得られている．

Fig. 11 **Relative Performance for the workloads** The proposed scheme shows better improvement for sGESV workload, and it shows at least around 95% of theoretical total system performance for all workloads.

前処理の実行割合が 0%となる場合において従来手法と提案手法のピーク性能を評価した．

図 11 にシステムとワークロードの組合せごとに予備実験から得られた Xeon と Xeon Phi のスループット性能を合計した理想システム性能に対する従来手法と提案手法のスループット性能比を示す．図 11 は横軸にシステムとワークロードの組合せを示し，縦軸に理想システム性能に対する従来手法と提案手法のスループット性能比を示す．

本結果から，提案手法は異なるシステムと異なるワークロードにおいても従来手法に対して 33%から 56%の性能向上が得られた．また，提案手法は理想システム性能に対して 96%から 100%のスループット性能が得られた．ワークロードとシステムの組合せによる違いから，Xeon の素性能と Xeon Phi の素性能に大きな差が見られない場合には提案手法の効果が大きい結果が得られ，Xeon の素性能と Xeon Phi の素性能に大きな差が見られる場合には提案手法の効果が比較的小さいことがわかった．本結果から，提案手法は Xeon Phi と Xeon の性能差が少ない場合により効果があることが確かめられた．

5. 関 連 研 究

マルチタスクにおける Xeon Phi のプロセッサ稼働率を向上させる研究が挙げられる[7),8)]．これら手法はオフロードモデルのタスクが複数実行された場合の Xeon Phi 単体のプロセッサ稼働率を向上させる手法である．また，Xeon Phi を複数のタスクで共用利用する場合に，実行スレッド数過多による顕著な性能低下を抑制することの重要性とアフィニティ設定の重要性を合わせて示している．ただし，タスクのホストプロセッサ実行部とオフロード実行部の実行割合ごとのホストプロセッサと Xeon Phi の双方のプロセッサ稼働率を述べてはいない．

マルチタスクで GPU を共有利用する研究が挙げられる[6),12)~14)]．Peters らは，Tesla 世代 GPU を対象として，複数の関数を含む単一のカーネルを GPU で動かし続けて，データだけを非同期に送ることでマルチタスクを実現する手法を提案している[6)]．これは NVIDIA GPU の Tesla 世代を対象としたものであるが，Kepler への世代交代によって，32 タスクまでのマルチタスクは可能となった．Sun らは，シングルプログラムで異なるデータサイズの処理を GPU でシェアするために，異なるデータに対する複数の同一カーネルをマージして GPU にオフロードすることで GPU の演算効率を向上する手法を提案している[12)]．Li らは，Fermi 世代の GPU を対象として，複数のタスクのカーネル部分の I/O，演算，データ転送情報を用いて GPU で並列実行するタスクのサブセットをグルーピングすることで GPU の演算効率を向上する手法を提案している[13)]．Ino らは，GPU にオフロードするタスクを分割することで GPU の時間方向の演算器リソースを充填させる手法を提案している[14)]．これら研究は，GPGPU でマルチタスクを実現することを目的とするため，我々の提案するようなオフロードを空間的にスケジューリングする研究では無い．

また，マルチタスク環境におけるプロセッサリソース管理の研究[15),16)] が挙げられる．Adriaens らは，アプリケーションのプロファイル情報を元にして，GPU の SM を静的に分割し，分割された SM にカーネルを割り当てることでリソース稼働率を上げる手法を提案している[16)]．Mok らは，リアルタイムシステムに対して 2 種類のリソース分割モデルを行い，タスクグループレベルでパーティションへのスケジューリングを行う手法を提案している[15)]．これらは静的に分割したリソースにタスクを割り当てるため，我々の動的な分割と割り当てを行う提案手法とは異なる．

6. ま と め

本論文では，Xeon と Xeon Phi で構成されたサーバーが複数のタスクによって共有利用される場合にタスク中のオフロード部分を Xeon 実行に切り替えるオフロードスケジューラを提案した．提案するオフロードスケジューラは Xeon と Xeon Phi の利用状況

と Xeon Phi のオフロードキューに溜められたタスク数に応じて Xeon 実行への切り替えを制御する．オフロードスケジューラを Xeon と Xeon Phi の組合せが異なる 2 種類のシステムと Xeon と Xeon Phi のスループット性能の比が異なる 2 つのワークロードで評価を行い，Xeon Phi のリソース管理のみを行う手法に対してシステム性能を向上することを確認した．また，提案手法はタスク中の Xeon Phi の実行割合が多い場合，および，Xeon と Xeon Phi の素性能に大きな差が見られないシステムとワークロードの場合に特に効果的であることが確認できた．

参 考 文 献

1) NVIDIA Corp. Kepler gk110 white paper. http://www.nvidia.com/content/PDF/kepler/NVIDIA-Kepler-GK110- Architecture-Whitepaper.pdf.
2) A. Heinecke, M. Klemm, and H. Bungartz. From gpgpu to many-core: Nvidia fermi and intel many integrated core architecture. *Computing in Science Engineering*, 14(2):78 –83, march-april 2012.
3) Intel. The intel®xeon phi™product family. http://www.intel.com/ content/ dam/ www/ public /us/ en/ documents/ product- briefs/ high- performance- xeon- phi- coprocessor-brief.pdf.
4) Gang Chen, Guobo Li, Songwen Pei, and Baifeng Wu. High performance computing via a gpu. In *Information Science and Engineering (ICISE), 2009 1st International Conference on*, pages 238 –241, dec. 2009.
5) Mian Lu, Jiuxin Zhao, Qiong Luo, Bingqiang Wang, Shaohua Fu, and Zhe Lin. Gsnp: A dna single-nucleotide polymorphism detection system with gpu acceleration. In *Parallel Processing (ICPP), 2011 International Conference on*, pages 592 –601, sept. 2011.
6) H. Peters, M. Koper, and N. Luttenberger. Efficiently using a cuda-enabled gpu as shared resource. In *Computer and Information Technology (CIT), 2010 IEEE 10th International Conference on*, pages 1122–1127, 2010.
7) Srihari Cadambi, Giuseppe Coviello, Cheng-Hong Li, Rajat Phull, Kunal Rao, Murugan Sankaradass, and Srimat Chakradhar. Cosmic: middleware for high performance and reliable multiprocessing on xeon phi coprocessors. In *Proceedings of the 22nd international symposium on High-performance parallel and distributed computing*, HPDC '13, pages 215–226, New York, NY, USA, 2013. ACM.
8) Takamichi Miyamoto, Kazuhisa Ishizaka, and Takeo Hosomi. A dynamic offload scheduler for spatial multitasking on intel xeon phi coprocessor. *The 18th Workshop on Synthesis And System Integration of Mixed Information technologies*, pages 261–266, Nov 2013.
9) Intel. Offload compiler runtime for the intel®xeon phi™coprocessor. http ://download -software.intel.com/ sites/ default/ files/ article/ 366893/ offload- compiler- runtime-for-the-intel-xeon-phi-coprocessor- 130315.pdf.
10) Intel. The intel®xeon phi™product family performance. http:// www. intel. com/ content/ dam/ www/ public/us/en/ documents/ performance -briefs/xeon-phi-product-family- performance -brief.pdf.
11) Intel®math kernel library. http://software.intel. com/ en-us/ sites/ default/ files/ Intel_ Math_ Kernel_ Library_ v12_ PB-1.pdf.
12) Siqi Sun, Zhuo Zhang, Liang Wang, Wenfeng Shen, Weimin Xu, and Yanheng Zheng. A study of the single-program multiple-task model on gpu computing. In *Automatic Control and Artificial Intelligence (ACAI 2012), International Conference on*, pages 296–300, 2012.
13) Teng Li, V.K. Narayana, E. El-Araby, and T.El-Ghazawi. Gpu resource sharing and virtualization on high performance computing systems. In *Parallel Processing (ICPP), 2011 International Conference on*, pages 733–742, 2011.
14) F. Ino, A. Ogita, K. Oita, and K. Hagihara. Cooperative multitasking for gpu-accelerated grid systems. In *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*, pages 774–779, 2010.
15) A.K. Mok, Xiang Feng, and Deji Chen. Resource partition for real-time systems. In *Real-Time Technology and Applications Symposium, 2001. Proceedings. Seventh IEEE*, pages 75–84, 2001.
16) J.T. Adriaens, K. Compton, Nam Sung Kim, and M.J. Schulte. The case for gpgpu spatial multitasking. In *High Performance Computer Architecture (HPCA), 2012 IEEE 18th International Symposium on*, pages 1 –12, feb. 2012.