OpenMPDによるマルチコア向け省電力機構の開発

木村英明[†1] 李珍泌[†1]
坂西隆之[†2] 福田喬士[†2]

近年，クラスタシステムに代表される並列分散メモリ環境が急速に普及している．しかしながら，分散メモリ環境では一般的にプログラミングが難しい．さらに，このような環境における消費電力の増加も問題となってきている．本プロジェクトではこれらの問題に対し並列言語による支援を行う．我々が開発している OpenMPD と省電力化技術を拡張・協調することで，様々な並列化に対応し，クラスタシステムの省電力化を実現する．実証プラットフォームとして低消費電力プロセッサ Atom を搭載したクラスタシステムを構築し，評価を行った．その結果，OpenMPD と省電力化技術の協調により，低電力システムのさらなる電力削減を実現できることが分かった．

Task Parallelization and Power-aware Extension on OpenMPD for Multi-core Clusters

Hideaki Kimura,[†1] Jinpil Lee,[†1] Takayuki Banzai[†2]
and Takashi Fukuda [†2]

Recently, many distributed shared memory (DSM) system has been developed such as PC clusters. However, programming cost on PC cluster system is more expensive than serial programming or shared memory programming. On the other hand, an increasing power consumption of PC cluster has been becoming a big issue. In this project, we design power-aware parallel language system by extended OpenMPD to solve these problems. Extended OpenMPD supports task-parallel programming model, and reduces the power of the PC cluster by combining with the power state control. We have designed and developed low power PC cluster system using Atom processor for the evaluation. As a result, we achieved energy-efficient computing on a low power system by cooperation of the OpenMPD and power-aware technology.

1. はじめに

近年，複数の計算ノードをネットワークを介して接続したクラスタシステムが広く普及している．しかしながら，クラスタシステムに代表される分散メモリ環境は，各ノードが利用可能なメモリ領域の制限によりプログラミングが難しい．マルチコアプロセッサ環境，すなわち共有メモリ環境におけるプログラミング手法として pthread や OpenMP を用いる方法があるが，これらは比較的容易に利用することができる．たとえば，OpenMP では逐次プログラム中に指示文と呼ばれる行を数行追加するだけで並列化を実現することができる．一方，クラスタシステムなど分散メモリ環境に対するプログラミングは一般的に難しく，アプリケーション開発者に対して大きな負担を強いることになる．分散メモリ環境を有効に活用するためにはスケーラビリティの高いプラットフォーム構築のみならず，扱いやすい分散メモリ環境向け並列言語による支援[1)2)3)4)] が必要不可欠である．

我々は，これまでに分散メモリ環境向け並列プログラミング言語 OpenMPD を設計・開発してきた[5)6)]．OpenMPD は逐次プログラムに対し単純な指示文を追加することで分散メモリ環境における並列化を実現する．そのため，逐次プログラムからの並列化も容易であり，アプリケーション開発者に対する負担も小さい．通常，分散メモリ環境では MPI（Message Passing Interface）を利用した並列化が一般的であるが，MPI ではデータの転送や同期処理を明示的に記述しなければならず，アプリケーション開発者に対する負担が大きい．

OpenMPD は大規模並列プログラムを容易に記述することを念頭に開発を進めている．そのため，大規

†1 筑波大学大学院 システム情報工学研究科
Graduate School of Systems and Information Engineering, University of Tsukuba
†2 筑波大学 第三学群 情報学類
College of Information Science, Third Cluster of Colleges, University of Tsukuba

模並列プログラムで頻繁に利用されるデータ並列の支援を念頭に開発を進めてきた．しかしながら，近年ではデータ並列のみならずタスク並列による並列化も重要になりつつある．そこで本プロジェクトでは，OpenMPD においてタスク並列拡張を実現することを目的とする．

一方，“グリーン IT” という言葉に象徴されるように計算機システムの消費電力が重要になりつつある．これは，クラスタシステムにおいても例外ではない．近年，クラスタシステムの各ノードを構成するコンピュータの消費電力が増加したことによって，クラスタシステム全体の消費電力が増加している．これにより，電力コストの増加，高価な冷却装置必要性，信頼性の低下といった様々な問題が発生している．クラスタシステムの消費電力を減少させる手法としては，

- 組み込み機器向けプロセッサなどを利用することによる，システムの “低電力化”[7)8)9)]
- DVFS（Dynamic Voltage and Frequency Scaling）機構などシステムで利用可能な技術を利用し，必要に応じて高性能モード・省電力モードを切り替える “省電力化”[10)11)12)]

が考えられる．我々は，これまでにプロセッサの省電力機構である DVFS 機構を利用することで並列システムの省電力化を実現してきた[13)14)]．クラスタシステム，電力測定環境 PowerWatch[15)] を構築し，実測による評価を行ってきた．本プロジェクトでは，クラスタシステムの消費電力問題を解決するためにノードの電源状態制御を利用した省電力機構を構築する．DVFS 機構ではプロセッサの省電力化は実現できるものの，メモリなどその他コンポーネントの電力削減は実現できない．本プロジェクトでは，OpenMPD による並列化支援とノードの電源状態制御機構を組み合わせることによりクラスタシステムの省電力化を実現する．評価のためのプラットフォームとして組み込みやモバイルインターネットデバイスを対象とした Intel Atom Processor を搭載した低電力クラスタシステムを構築し，実機による性能評価を行う．すなわち低電力な環境でさらなる省電力化を目指す．

本プロジェクトの目的を以下にまとめる．

- 分散メモリ環境向け並列言語 OpenMPD のタスク並列拡張を実現する．
- 低電力マルチコアプロセッサプラットフォームを設計・構築し，省電力化技術について実際に電力測定を行うことで評価する．
- 分散メモリ環境向け並列言語 OpenMPD と省電力化技術が協調することにより，省電力を実現するディスパッチャを構築する．異なる技術を組み合わせることにより，“OpenMPD によるマルチコア向け省電力機構” を実現する．

本稿の構成は以下のようになっている．第 2 章では，OpenMPD とそのタスク並列拡張について述べる．第 3 章では，省電力タスクスケジューリング方式，特に OpenMPD との連携による省電力化方式について述べる．第 4 章では評価について述べる．第 5 章では関連研究について述べ，最後にまとめと今後の課題を述べる．

2. OpenMPD とタスク並列拡張

2.1 分散メモリ環境向けデータ並列言語 OpenMPD

現在，分散メモリ環境における並列化手法として MPI が広く知られている．これは，ノード間でデータの送受信を明示的に記述することによって高い性能を達成するが，その反面プログラマに対する負担が大きい．また，逐次プログラムからの並列化も難しい．MPI では “どのノードがどのようなデータを保持するか” を意識したプログラミングが必要となり，逐次プログラムから多くの箇所を変更する必要がある．

そこで，我々は OpenMPD を提案している．これは，以下のような特徴をもつ．

- 分散メモリ環境を対象とする．
- 共有メモリ環境向け並列言語 OpenMP に似た，指示文による並列化を実現する．
- MPI で記述したコードと同等に，高い性能を実現可能である．

OpenMPD は C 言語の指示文による拡張として設計されている．アプリケーション開発者は，逐次プログラムにいくつかの指示文を挿入することで容易に分散メモリ環境における並列化を実現できる．

OpenMPD の記述例を**図 1** に示す．これは，逐次プログラムに対して並列化のための指示文**#pragma ompd for** を追加したものである．このように非常に単純な指示文を一行追加するだけで分散メモリ環境における並列化を実現できる．分散メモリ環境において頻繁に利用される MPI では，どのノードにどのようなデータが保持されているかを意識したプログラミングが必要となり，通信を明示的に記述する必要がある．例えば Laplace プログラムでは，各ノードが処理するデータ領域の両端を別のノードに送受信するためのコードを明示的に記述しなければならない．どのノードが，どのような処理を実行し，どのようなデータを保持しているかを意識したプログラミングが必要となり，アプリケーション開発者に対して大きな負担を与える．一方，OpenMPD を用いた並列化ではデータの送受信を明示的に記述する必要はない．逐次コードに対して指示文を挿入するのみで並列化を実現でき，プログラマに対する負担は小さい．

図 2 に OpenMPD 処理系の概要を示す．OpenMPD 処理系は C 言語の拡張として設計されている．処理系は Omni OpenMP Compiler[16)] をベースに実装されている．OpenMPD のソースコードは処理系のフロン

```
int array[YMAX][XMAX];
#pragma ompd distvar(var=array; dim=2)
main()
{
   int i, j, res;
   res = 0;
#pragma ompd for affinity(array) reduction
     (+:res)
   for (i=0; i<YMAX; i++) {
      for (j=0; j<XMAX; j++) {
         array[i][j] = func(i, j);
         res += array[i][j];
      }
   }
}
```

図 1 OpenMPD の記述例

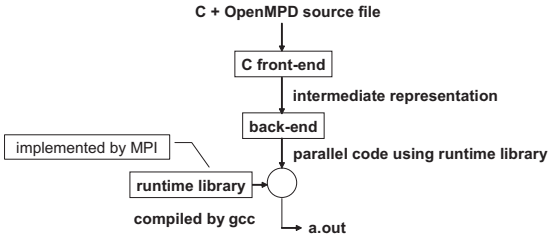

図 2 OpenMPD 処理系の概要

トエンドによって中間言語に変換される．OpenMPD の並列化は中間言語を用いて行われる．処理系のバックエンドが逐次コードと OpenMPD 指示文から並列コードを生成する．並列コードは中間言語から C 言語のソースコードに変換され，ランタイムライブラリにリンクされる．その結果，並列プログラムを生成する．並列コードは OpenMPD のランタイムライブラリを用いてノード間通信による並列処理を行う．現在の実装では通信に MPI を用いるため，ランライムライブラリとして MPI の API 関数を用いる．ユーザは性能最適化のために OpenMPD のコード中で MPI 関数を利用することが可能であり，高い性能を達成することが可能である．

我々は，科学技術計算で頻繁に利用されるデータ並列化を念頭に OpenMPD の設計・開発を行ってきた．科学技術計算では，非常に膨大なデータに対して同一の処理を繰り返し実行することが非常に多い．データ並列化によって，数多くの科学技術計算の並列化を支援してきた．

しかしながら，データ並列化のみによる支援では OpenMPD の利用可能範囲が限定される．すなわち，一部の大規模並列プログラムのみが対象となる．データ並列性を十分に利用できないプログラムに対しては，大量の利用可能ノードが存在する場合であっても高い性能を達成することはできない．より汎用的な並列化モデルを提供するためには，処理されるデータ間の並列性のみならず処理そのものに着目しなければならない．すなわち，処理そのものの並列性を記述するための “タスク並列拡張” が必要となる．本プロジェクトでは，既存の OpenMPD に対して新たな指示文を追加することにより，OpenMPD タスク並列拡張を実現する．

2.2 タスク並列拡張のための言語拡張

OpenMPD では，指示文を定義することによって様々な言語拡張を行うことができる．本プロジェクトでは，タスク並列拡張を実現するために指示文 **task_parallel** と指示文 **task** を導入する．これらの指示文にはタスク間の依存関係に関する情報が含まれる．以降，各指示文の仕様について述べる．

- **task_parallel(master**=*node*)

task_parallel はタスクスケジューリングを行う区間を明示するための指示文である．task_parallel の内部に存在する互いに独立なタスクは並列分散実行される．また，これは並列実行可能なタスクが存在する場合のみ行われ，指示文 task の dep 節によって明示された依存関係にしたがって処理を行う．**master** 節によって指定されたノードが逐次処理部を実行する．

- **task(name**=*task_name*; **dep**=*dependency*; **sync**=*synchronization*)

task はタスクスケジューリングの対象となるタスクを指定するための指示文である．各タスクは **name** 節によって名前をつけることが可能である．**dep** 節はタスク間の依存関係を表す．ある task に dep 節が指定された場合，その task は dep 節で指定されたタスクの完了後にスケジューリングされる．データの整合性を保障するための同期機構として **sync** 節を提供する．

2.3 タスク並列拡張の記述例

図 3 に OpenMPD タスク並列化の記述例を，**図 4** に 2 ノードでプログラムを実行した際の処理の流れを示す．

ここでは，各関数の直前に並列化のための指示文を挿入し，4 つのタスクに分割している．ここで，func_A と func_B の戻り値は func_C で必要となる．すなわち，タスク A とタスク B，タスク C 間には依存関係が存在しており，依存関係を明示するために dep 節が必要となる．さらに，sync 節によってデータの整合性を保障する．

3. 省電力タスクディスパッチャ

3.1 クラスタシステムにおける省電力化技術

DVFS 機構を効果的に制御することにより，省電力化を実現することができる．DVFS 機構とはプロセッサの動作周波数・動作電圧を動的に制御するための機構であり，負荷に応じてこれを制御する EIST（Enhanced Intel Speedstep Technology）[17] や Cool'n'Quit[18]，PowerNow![19] といった様々な制御技術も利用可能になってきた．プロセッサのアクティブ消費電力は動作電

```
#pragma ompd task_parallel (master=0)
{
#pragma ompd task(name=A)
   x = func_A();
#pragma ompd task(name=B)
   y = func_B();
#pragma ompd task(name=C; dep=A,B; sync=(A:x),(B:y))
   func_C(x, y);
#pragma ompd task(name=D)
   func_D();
}
```

図 3 OpenMPD のタスク並列拡張

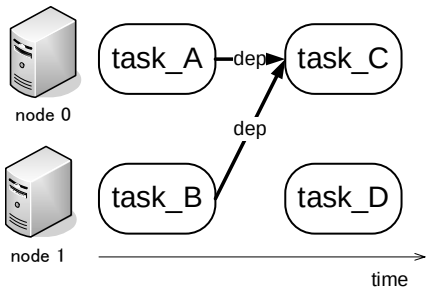

図 4 タスク並列に関するスケジューリング

圧の 2 乗と動作周波数に比例する．このため，DVFS 機構を適切に制御することによってプロセッサのアクティブ電力を削減し，省電力化を実現できる．DVFS 機構は単一ノードに対する制御としても有効活用可能であるが，クラスタシステムにおいても効果的に利用することが可能である．一般的に並列システムにおいて通信時は計算時より負荷が小さく低い動作周波数・動作電圧を選択した場合であっても性能低下は少ない．そこで通信時に低い周波数を選択するよう制御することによってわずかな性能低下で大幅な電力削減を実現することが可能となる[15)20)]．また，並列プログラムは常に負荷が均衡するとは限らない．このような場合においても DVFS 機構を効果的に制御することにより同期などの待ち合わせのための時間（slack time）を削減し，並列システム全体の性能を低下させることなく消費エネルギーを削減することが可能である[10)21)]．

しかしながら，DVFS 機構の制御ではプロセッサのアクティブ消費電力を削減するのみであり，システムバスやメモリ，HDD などといったその他構成要素についての電力削減を実現することができない．また，近年ではプロセスルールの微細化によってリーク電流をはじめとするプロセッサのスタティック電力も無視できなくなってきた．これらは，DVFS 機構を適切に制御しても削減することはできない．

一方，本プロジェクトで対象とするクラスタシステムでは，各ノードは独立に動作可能であり，必要に応じて計算ノード数を制御することが可能である．負荷の小さい時や並列度の低い場合などは一部の計算ノードで処理を行い，余剰となる遊休ノードは電源を遮断することにより省電力化を実現することができると考えられる．ノードの電源状態制御を行うことで，プロセッサのスタティック電力，システムバスやメモリ，HDD といったプロセッサ以外の構成要素が消費する電力を削減することが可能である．

ノードの電源状態制御により，様々な電源状態に遷移させることが可能である．各ノードをどのような状態に遷移させるかについては復帰時間とのトレードオフを考慮しなければならない．**表 1** に，頻繁に利用される ACPI ステートとその電源状態を示す．

表 1 ACPI ステータスと状態

ACPI status	status
S0	稼動状態
S3	データをメモリに退避．他デバイスは OFF
S4	データを HDD に退避．デバイスを OFF
S5	ソフトウェアによる電源 OFF

ACPI S3 ステートでは、データをメモリ内に保存しプロセッサのみを休止状態とする．これは，高速な復帰が可能であるがメモリには通電し続けるため電力削減幅が小さくなる．ACPI S4 ステートではすべてのデータを退避し電源するため，消費電力を大幅に削減することが可能である．しかしながら，復帰に要する時間が長くなる．ACPI S5 ステートでは一般的な電源 OFF と同等の処理を行う..

本プロジェクトでは OpenMPD と連携するため，プログラム開始時に必要な計算ノード数を指定する．したがって，負荷状態に対する追従性・応答性を考慮する必要は無く，復帰に要する時間は重要ではない．また評価に利用する環境においては，カーネルイメージの軽量化によって ACPI 各ステートから復帰に要する時間に大きな差がないことを確認している．本プロジェクトではより積極的な電力制御を実現するために，ACPI S5 ステートと Wake on LAN 機能による遠隔電源投入を組み合わせる．

3.2 OpenMPD との連携による省電力化

現在，マルチコア環境が急速に普及しており，これらを効果的に活用するためのスケジューラが重要になってきた．しかしながら，これまでに提案されているスケジューリング規則の多くは性能のみに着目しており，電力を考慮したものは少ない．また，クラスタシステムのような分散メモリ環境も増加しており，マルチコア・分散メモリ環境における適切なスケジューラ・ディスパッチャに対する期待が高まってきている．

本プロジェクトでは，OpenMPD と連携し省電力を実現するためのディスパッチャの構築を目指す．OpenMPD では，プログラム実行時に前もって使用するノード数を与える．ここで並列度の低いプログラムを実行するような場合には，クラスタシステムの全ノード数

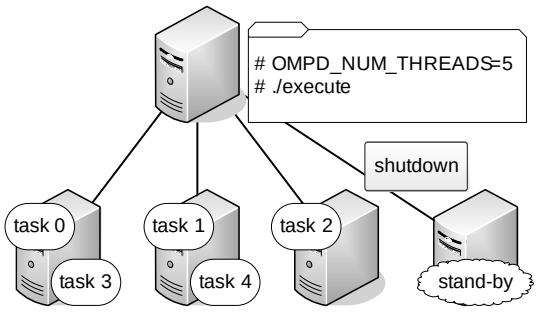

図 5 OpenMPD との連携による省電力化

と比べて少ないノード数のみで十分である可能性が高い．確保されなかったノードは遊休状態となりプログラムを実行することはない．しかしながら，遊休状態にあってもシステムの電源は投入された状態にあり，電力を消費する．

本プロジェクトではOpenMPDをタスクディスパッチャとして利用し，ノード電源状態制御を組み合わせることによって必要ノード数の確保と不必要ノードの電源遮断によるクラスタシステムの省電力化を同時に実現する．これにより，高性能・省電力を同時に実現するシステムを実現する．

3.3 CPU コア割り当て方式による省電力化

OpenMPD との連携により省電力化を実現するためのシステムの概要を**図 5** に示す．

ユーザは並列プログラム実行前に使用するプロセッサコア数を宣言する．ここでマルチコア並列クラスタ環境を想定すると，様々なタスクの割り当て方法が考えられる．プロセッサを確保する方法として以下の 2 種類について考える．

- ノード間並列化を重視し，すべてのノードにタスクを平均的に割り当てる
- ノード内並列化を重視し，一部のノードにタスクを集中的に割り当てる

ノード間並列化を重視する事で，各タスクはキャッシュやメモリバス，メモリなどを占有することができる．したがって，ノード内並列化を重視した場合と比較して高い性能を達成できる可能性が高い．一方，多ノードの電源状態制御を行うためには一部のノードにタスクを集中させるべきである．本プロジェクトでは，より積極的な電力削減を達成するためにノード内並列化を重視する方式を採用する．

4. 評　　価

4.1 低電力プラットフォーム：Atom Cluster

評価用プラットフォームとして，Intel Atom Processor[22] を搭載した 8 ノードクラスタ（計算ノード），および管理ノードからなるシステムを構築した．クラスタシステムを構成する計算ノードの構成を**表 2**，管理ノードの構成を**表 3** に示す．管理ノードは，HDD と追加の NIC を搭載した点を除いてはクラスタノードと同等の構成とした．管理ノードにのみ HDD を搭載し，計算ノードは pxeboot によって起動する．

表 2 計算ノード

CPU	Intel Atom 330 1.6GHz, L2: 512kB *2
Mother Board	Intel D945GCLF2
Memory	DDR2 SDRAM 2GB
Power Supply	Enhance ENP-2730H SFX
number of nodes	8

表 3 管理ノード

CPU	Intel Atom 330 1.6GHz, L2: 512kB *2
Mother Board	Intel D945GCLF2
Memory	DDR2 SDRAM 2GB
HDD	HDP725050GLA360
Power Supply	ENP-2730H SFX

Intel Atom Processor は 2008 年 3 月に Intel 社より発表されたモバイルインターネットやシンプルで低コストなシステムを対象とした低消費電力プロセッサである．2008 年 9 月よりデュアルコア版 Atom Processor も登場し，利用の範囲はますます広がっている．Atom Processor は高級小型端末や組み込み機器向けの Z シリーズ（開発コード：Silverthorne）と低価格ネットトップ向けの N シリーズ（開発コード：Diamondville）の 2 種類に分類することができる．本プロジェクトでは，安価で比較的入手が容易な低価格ネットトップ向けデュアルコア版 Atom Processor を利用し，1) 安価に，2) 低消費電力なシステムを，3) 高密度実装した．

また，今回使用した Intel Atom 330 では EIST をサポートしておらず，動作周波数・電圧を動的に制御することはできない．ノードの電源状態制御によってのみ省電力化を実現することが可能である．

クラスタシステム製作の様子を**図 6**，**図 7**，完成したクラスタシステムの概観を**図 8** に示す．

計算ノードでは，Linux kernel 2.6.27.10 を用いた．今回使用したマザーボード Intel D945GCLF2 では，NIC を正常に認識させるために Linux kernel 2.6.24 以降を使用する必要がある．またコンパイラとして icc-11.0.074，ifort-11.0.074 を用いた．これは，gcc，gfortran が Atom Processor に正式対応していないためである．

電力測定のために，我々が開発した PowerWatch[15] を用いた．これは，電流センサ，接続 BOX，A/D 変換ボード，管理用 PC などからなる．ホール素子を用いた非接触型の電流センサを用いる．これにより，測

図 6 Atom Cluster 作成の様子（1）

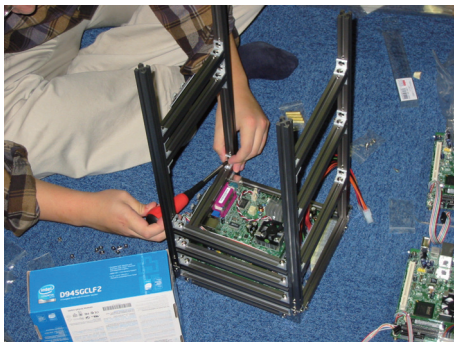

図 7 Atom Cluster 作成の様子（2）

図 8 Atom Cluster の概観

定対象を非破壊で電力測定できる．今回使用した非接触型電流センサは，電流値をアナログ電圧データとして出力する．複数のデータを接続 BOX によってまとめ，A/D 変換ボードによりデジタルデータに変換し，PC において解析を行う．

4.2 単一ノードにおける性能評価

Atom Processor は HT（Hyper Threading Technology）機構を持っており，デュアルコア版 Atom Processor では OS 上からは 4 コアが存在するように見える．予備評価として HT を ON にした場合と OFF にした場合の性能を比較する．予備評価のためのベンチマークとして，高性能計算分野で頻繁に利用される NPB[23]（NASA Parallel Benchmarks）からカーネルベンチマーク 5 種を用いる．

表 4 に各ベンチマークを実行した際の性能（相対値，Atom 330 w/o HT を 1 とする）を示す．OpenMP 版の NPB を使用し，CLASS は B，最適化オプションとして-O3 -QxSSE2 を与えた．

並列性が非常に高くメモリアクセスも少ない EP（乗算合同法による乱数生成）では HT の効果が非常に大きいことが分かる．一方で，まばらなメモリアクセスが発生する IS（整数ソート）や FT（FFT による微分方程式解法）では HT を利用することで性能が低下している．これは，HT を使用することにより演算器やキャッシュの競合が発生するためである．同様にまばらなメモリアクセスが発生する MG（マルチグリッド法を用いたポアソン方程式解法）では一度に大きなメモリアクセスが発生するため HT による性能低下は見られない．一般的なデスクトップ向けプロセッサである Intel Core2Duo E8400（3GHz，6MB L2 cache）を搭載したシステムと性能比較では，Atom 330 の CG（共役勾配法を用いた固有値解法）実行時の性能が非常に低い事が確認できる．これは，Atom Processor ではメモリアクセスがシングルチャネルとなっている事が理由として挙げられる．CG では，非常に細かいメモリアクセスが定常的に行われる．ここでメモリバンド幅がボトルネックとなり，プロセッサ性能を十分に発揮できていない．CG 以外のベンチマークでは，2.8 倍から 5.8 倍の性能となっている．これらの差は in-order 実行などプロセッサアーキテクチャの違いや動作周波数に起因すると考えられる．

表 4 NPB 実行時の相対性能，
（Atom 330 w/o HT を 1 とする）

	Atom 330 (HT)	Core2Duo E8400
EP	1.85	2.81
MG	1.21	2.80
CG	1.47	12.2
IS	0.94	4.10
FT	0.64	5.78

Atom Processor の TDP（Thermal Design Power）は 8W，Core2Duo Processor は 65W である．以上の結果より，Atom Processor の単体性能は低いものの，電力効率は非常に高いと考えられる．

本プロジェクトでは，OpenMPD によるタスク並列化を対象としている．対象となるアプリケーションの多くはデータを共有せず，依存関係はあるものの各タスクは独立に実行可能である．予備評価結果より，このようなアプリケーションに対しては HT の効果が高いことが分かる．よって，HT を ON として以降の評価を行う．

```
if (ompd_task_sched(0))
   x = func_A();

if (ompd_task_sched(1))
   y = func_B();

if (ompd_task_sched(2)) {
   // タスクA と B の終了を待つ
   ompd_sync_barrier(0);
   omsc_sync_barrier(1);

   // 変数x と y の同期を行う
   // ソースは指定されたタスクを実行するノード
   omsc_sync_var(&x, OMPD_TYPE_INT, 0);
   omsc_sync_var(&y, OMPD_TYPE_INT, 1);

   // タスクを実行
   func_C(x, y);
}

if (ompd_task_sched(3))
   func_D();

//task_parallel の終了
omsc_sync_barrier_all();
```

図 9 OpenMPD 処理系によって処理された並列コード

4.3 OpenMPD タスク並列拡張によるコード変換

分散メモリ環境向け並列プログラム OpenMPD によるタスク並列拡張を実現した．指示文 task_parallel と指示文 task を実装し，変換された並列コードが利用するランタイムライブラリ関数を追加する．図 3 のコードを OpenMPD の処理系によって変換した結果を**図 9** に示す．

各タスクは指示文 task によって固有の名前が付けられる．処理系はタスクの名前からタスク番号を生成し，スケジュールのために利用する．あるノードでタスクを実行するかどうかはランタイムライブラリ関数 omsc_task_sched() によって決定される．omsc_task_sched() は引数のタスク番号とノード番号に関する内部データからスケジューリングを行う．指示文 task_parallel の内部に存在するタスクはスケジューリングによってクラスタシステムの各ノードに分散され，並列に処理される．現在の実装ではラウンドロビン方式の静的なスケジューリングが行われる．その結果，指定した全ノードに対して均等なタスク分散が行われる．OpenMPD のタスク並列機構はノード内の並列化を考慮し，マルチスレッドでのタスクスケジューリングをサポートしている．

task に dep 節が指定された場合，実行タイミングの同期（バリア同期）を行う．依存関係のあるタスクを先に実行することを保障して，正しい計算結果が得られるようにスケジューリングを行う．バリア同期にはランタイムライブラリ関数 ompd_sync_barrier() を用いる．ompd_sync_barrier() は引数で指定されたタスクが終了されるまでプログラムの実行を停止する．対象となるタスクが他ノードで実行されている場合は，ノード間のメッセージ交換によるバリア同期が行われる．対象となるタスクが同一ノードで実行されている場合は，スレッド間のバリア同期を行いタスク実行順序の正しさを保証する．

変数の同期を行い，タスクが正しい値にアクセスできるよう指定するために sync 節が提供されている．変数と関連付けられたタスクが同期のソースとなる．処理系はタスクの名前を番号に変換してランタイムライブラリ ompd_sync_var() の引数として利用する．対象のタスクが他のノードで実行された場合は，ノード間通信によってデータをソースから受信する．タスクが実行中である可能性があるため，通信の前にノード間のバリア同期が行われる．同一ノードの他スレッドで実行されている場合は，スレッド間のバリア同期が行われ，関数が終了する．sync が指定されている場合はバリア同期が自動的に挿入されるため，dep による依存関係の記述は必要ではない．しかし，コードの意味を明確にするため，両方の記述を行うことが推奨される．dep と sync がともに記述されている場合，処理系ではバリア同期を一回のみ行うよう最適化する．

全てのタスクが実行され task_parallel の内部の処理が全て終了すると，全てのノードやスレッドでバリア同期を行う．ランタイムライブラリ関数 ompd_sync_barrier_all() が実行されてタスク並列の処理が終了する．

図 3 には記述されていないが，task_parallel において変数の同期を記述することも可能である．この場合は，先頭部において task の sync 節と同様の処理が行われる．

4.4 OpenMPD タスク並列拡張時の電力性能評価

タスクを 16 個生成し，タスクを実行するノード数，ノード内スレッド数を変化させたときの実行時間を**表 5** に，消費エネルギーを**表 6** に示す．生成したタスクはいずれも同程度の時間で終了するものとし，依存関係はないものとした．

8 ノードを使用し各ノードで 2 スレッドを生成した際に実行時間が最短となっている．HT を使用したことにより OS 上からは論理的には 4 コア存在するように見えるが，演算器の競合により，依存性のないタスクであっても論理 CPU コア数分の性能向上は見られないことが予備評価の結果から分かっている．本評価結果においても，4 ノード 4 スレッド実行時に 8 ノード 2 スレッド実行時と比較して若干ながら性能が低下することを確認した．今回の評価においてオーバヘッドの大部分はハードウェアに起因するものであり，OpenMPD そのもののオーバヘッドは小さい．また，使用ノード数とノード内スレッド数の積に反比例して実行時間は長くなる．これは，システム全体で同時に実行可能なスレッド数が制限された場合には，当該ス

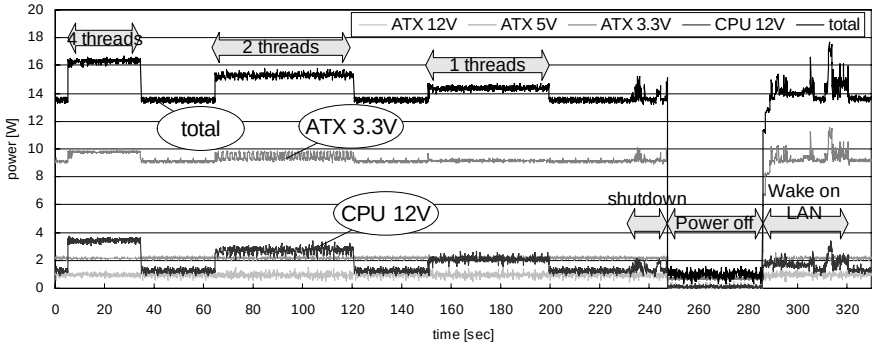

図 10 並列プログラム実行時の消費電力遷移

表 5 16 タスク実行時の実行時間 [sec]

	1 thread	2 threads	4 threads
1 node	391	224	132
2 nodes	195	112	62.0
4 nodes	97.8	56.0	29.6
8 nodes	48.9	27.5	*

表 6 16 タスク実行時の消費エネルギー [Wsec]

	1 thread	2 threads	4 threads
1 node	8169	4894	3021
2 nodes	6696	4060	2377
4 nodes	5988	3643	2049
8 nodes	5623	3373	*

レッドがタスクを完了した後に新たなタスクを逐次実行するためである．

一方消費エネルギーについて評価を行うと，4 ノード 4 コア時が最適であることが分かる．各ノードにおいて ACPI S5 ステートを積極的に利用し，OpenMPD（タスクディスパッチャ）においてノード内並列化を重視することにより消費エネルギーを最小化した．本プロジェクトで開発した “OpenMPD によるマルチコア向け省電力機構” では確保するプロセッサコア数を指定するとノード内並列化を重視した並列化を実現する．分散メモリ環境向け並列言語 OpenMPD とシステムの省電力化機構を適切に組み合わせることにより，効果的な電力削減，高性能・省電力を両立するシステムを実現した．

図 10 に OpenMPD によりタスク並列実行した際の特定ノードの消費電力遷移を示す．図 10 より，実行するスレッド数に応じて CPU 消費電力が変化していることが確認できる．構成要素別に見ると，CPU で消費する電力と比較して ATX 3.3V 系統の消費電力が非常に大きいことが分かる．これはネットトップ向け Atom Processor において頻繁に利用されている 945GC+ICH7 チップセットによるものであると考えられる．評価システムにおいてはチップセットにのみ冷却用ファンが搭載されている．

プロセッサが遊休状態であったとしても 14W 程度の電力を消費するが，ACPI S5 ステートを併用することにより 1W 程度まで電力を削減することができる．これより，低電力プラットフォームにおいても ACPI S5 ステートの積極的な利用によりシステム全体の消費エネルギーを大幅に削減できることが分かる．

5. 関 連 研 究

分散メモリ環境における並列言語による支援としては Unified Parallel C（UPC）[24)25)]，Co-Array Fortran（CAF）[26)27)28)]，High Performance Fortran（HPF）[29)] などが挙げられる．いずれも，C 言語や Fortran 言語の拡張により分散メモリ環境でのデータ並列化を言語によって支援している．これらは，高性能計算分野で頻繁に利用されるデータ並列化を念頭に開発が進められてきた．UPC ではタスク並列化の導入が検討されているものの製品に実装はされていない．また，CAF や HPF ではタスク並列化は実現されていない．共有メモリ環境では OpenMP を用いた並列化が一般的である．OpenMP ver3.0[30)] よりタスク並列化の概念が導入されている．本プロジェクトでは，分散メモリ環境において同等の概念を実現している．OpenMP の仕様においてはタスクの依存関係を明示することはできないため我々は独自の拡張を加えている．

一方，近年ではシステムの省電力化，グリーン IT の実現へ向けた数多くの提案がなされている．クラスタシステムの省電力化を実現する方法としては，DVFS 機構の利用やスタンバイ状態を含めたノード状態制御が挙げられる．高性能計算分野では，プログラムの特性に着目し省電力化を実現する技術が数多く提案されている．Ge ら[31)] や Min ら[32)] は並列計算の通信に着目し，電力削減を実現している．ノード間通信時やメモリアクセス時は低い動作周波数・動作電圧を選択す

ることで省電力化を実現できる．Rajamani ら[33] や Rusu ら[34]，今田ら[35] は，スタンバイ状態を含めたノード状態制御により省電力化を実現する手法を提案している．これらは Web サーバクラスタシステムを対象としており，リクエストレートが低くなると予測された際に余剰となるノードをスタンバイ状態に遷移させる．本プロジェクトで開発したシステムは，プログラムを実行するユーザが必要とするノード数を指定する点が異なる．また，これらの研究は対象アプリケーションを分散型 Web サーバに限定しておりアクセスレートの変化によるシステム負荷の変動を利用した最適化が行われている．本プロジェクトでは，負荷の変動ではなく対象となるアプリケーションの並列度に着目した省電力化となっている．また，分散メモリ環境向け並列言語との協調により省電力化を実現している点も特徴である．

6. おわりに

本プロジェクトでは，以下について実現・評価を行った．

- 分散メモリ環境向け並列言語 OpenMPD のタスク並列拡張を実現した．
- 低電力マルチコアプロセッサ環境を構築し，省電力化技術の評価を行った．
- 分散メモリ環境向け並列言語 OpenMPD と省電力化技術の組み合わせにより “OpenMPD によるマルチコア向け省電力機構” を実現した．

分散メモリ環境向け並列言語 OpenMPD のタスク並列拡張により，既存の分散メモリ環境において困難であったタスク並列による並列化を支援した．これは，データ並列を重視した既存の OpenMPD を拡張することで実現している．現在，タスク並列化の重要性は広く認知されるようになってきており，本プロジェクトにおいて開発した OpenMPD はその先駆けである．今後は，タスクの動的スケジューリングや指定したタスクを任意の CPU で実行するといった機能拡張を検討する．

低電力マルチコアプロセッサである Atom Processor を搭載したシステムを構築し，省電力化技術を組み合わせることによって，さらなる省電力化を実現した．Atom Processor は非常に高い電力性能を達成できることを確認した．

さらに，OpenMPD と省電力化技術を協調させることにより省電力化機構を構築した．プログラムの並列度が低いなど，クラスタシステム全ノードを必要としない場合，OpenMPD がタスク割り当てを行うとともに電源状態制御を行う．電力測定の結果より，低電力クラスタシステムにおいて ACPI S5 ステートと Wake on LAN の組み合わせによる省電力化の可能性を示した．今後は，ノードの休止・復帰に要するペナルティを考慮した省電力化機構の開発や，OpenMPD 言語処理系とのさらなる協調により動作周波数制御を含めた電力削減手法について検討する．

謝辞 本プロジェクトの遂行にあたり，指導教員である佐藤三久教授には多くのご助言，ご指摘等をいただきました．また，システム開発型研究プロジェクト中間報告会にて様々なご意見をいただきました皆様，HPCS 研究室の皆様に御礼申し上げます．本研究の一部は，魅力ある大学院教育イニシアティブ “実践 IT 力を備えた高度情報学人材育成プログラム” による．

参考文献

1) Jarek Nieplocha, Bruce Palmer, Vinod Tipparaju, Manojkumar Krishnan, Harold Trease, and Edo Apra. Advances, Applications and Performance of the Global Arrays Shared Memory Programming toolkit. *International Journal of High Performance Computing Applications*, Vol.20, No.2, pp. 203–231, 2006.
2) C.Bell, W.Chen, D.Bonachea, and K.Yelick. Evaluating Support for Global Adress Space Languages on the Cray X1. In *ICS 2004*, 2004.
3) W.Chen, D.Bonachea, C.Iancu, and K.Yelick. Automatic Nonblocking Communication for Partitioned Address Space Programs. In *ICS 2007*, 2007.
4) J. Nieplocha, RJ Harrison, and RJ Littlefield. Global Arrays: A portable shared memoy model for distributed memory computers. In *Supercomputing'94*, 1994.
5) Jinpil Lee, Mitsuhisa Sato, and Taisuke Boku. Design and Implementation of OpenMPD. *International Workshop on OpenMP 2007*, 2007.
6) Jinpil Lee, Mitsuhisa Sato, and Taisuke Boku. OpenMPD: a Directive-based Data Parallel Language Extension for Distributed Memory Systems. In *P2S2*, 2009.
7) N.R. Adiga et al. An Overview of the BlueGene/L Supercomputer. *SC2002*, p.60, 2002.
8) Michael S. Warren, Eric H. Weigle, and Wu-Chun Feng. High-Density Computing: A 240-Processor Beowulf in One Cubic Meter. *SC2002*, p.61, 2002.
9) Hiroshi Nakashima, Hiroshi Nakamura, Mitsuhisa Sato, Taisuke Boku, Satoshi Matsuoka, Satoshi Matsuoka, Daisuke Takahashi, and Yoshihiko Hotta. MegaProto: 1 TFlops/10kW Rack Is Feasible Even with Only Commodity Technology. *SC05*, p.28, 2005.
10) Guangyu Chen, Konrad Malkowski, Mahmut T. Kandemir, and Padma Raghavan. Reducing Power with Performance Constraints for Parallel Sparse Applications. In *HPPAC in*

IPDPS, 2005.

11) Chung-Hsing Hsu and Ulrich Kremer. The Design, Implementation, and Evaluation of a Compiler Algorithm for CPU Energy Reduction. In *PLDI*, pp. 38–48, 2003.
12) Hiroshi Sasaki, Yoshimichi Ikeda, Masaaki Kondo, and Hiroshi Nakamura. An intra-task dvfs technique based on statistical analysis of hardware events. In *Computing Frontiers 2007*, 2007.
13) 木村英明, 佐藤三久, 堀田義彦, 朴泰祐, 高橋大介. DVS による負荷不均衡のある並列プログラムの電力量削減手法. 情報処理学会論文誌 Vol.47 No.SIG12 （ACS15）, pp. 285–294, 2006.
14) 木村英明, 佐藤三久, 堀田義彦, 今田貴之. 影響の少ないインスツルメント手法と電力最適化のためのプログラム領域分割. 情報処理学会論文誌 Vol.48 No.SIG13 （ACS19）, pp. 247–259, 2007.
15) Y. Hotta, M. Sato, H. Kimura, S. Matsuoka, T.Boku, and D.Takahashi. Profile-based Optimization of Power-Performance by using Dynamic Voltage Scaling on a PC cluster. In *HPPAC in IPDPS*, 2006.
16) Mitsuhisa Sato, Sigehisa Sato, Kazuhiro Kusano, and Yoshio Tanaka. Design of OpenMP Compiler for an SMP Cluster. In *EWOMP*, 1999.
17) *Enhanced Intel Speedstep Technology*. http://download.intel.com/design/network/papers/30117401.pdf.
18) *Cool 'n' Quiet*. http://www.amd.com/us-en/assets/content_type/DownloadableAssets/Cool_N_Quiet_Installation_Guide3.pdf.
19) *Power Now!* http://www.amd.com/us-en/assets/content_type/DownloadableAssets/Power_Now2.pdf.
20) Rong Ge, Xizhou Feng, and Kirk W. Cameron. Performance-constrained Distributed DVS Scheduling for Scientific Applications on Power-aware Clusters. In *SC05*, 2005.
21) Hideaki Kimura, Mitsuhisa Sato, Yoshihiko Hotta, Taisuke Boku, and Daisuke Takahashi. Empirical Study on Reducing Energy of Parallel Programs using Slack Reclamation by DVFS in a Power-scalable High Performance Cluster. In *CLUSTER 2006*, 2006.
22) *Intel Atom Processor Z5xx Series*. http://download.intel.com/design/processor/datashts/319535.pdf.
23) D. Bailey et. al. THE NAS PARALLEL BENCHMARKS. *RNR Technical Report*, 1994.
24) CristianCoarfa et. al. An Evaluation of Global Address Space Languages: Co-Array Fortran and Unified Parallel C. In *PPoPP 2005*, 2005.
25) *Unified Parallel C*. http://upc.gwu.edu.
26) Yuri Dotsenko, Cristian Coarfa, and John Mellor-Crummery. A multi-platform co-array fortran compiler. In *PACT 2004*, 2004.
27) Cristian Coarfa, Yuri Dotsenko, Jason Lee Eckhardt, and John Mellor-Crummery. Co-array Fortran Performance and Potential: An NPB Experimental Study. In *LCPC 2003*, 2003.
28) *Co-Array Fortran*. http://www.co-array.org.
29) *High Performance Fortran*. http://hpff.rice.edu/.
30) *OpenMP 3.0*. http://openmp.org.
31) Rong Ge, Xizhou Feng, and Kirk W. Cameron. Performance-constrained Distributed DVS Scheduling for Scientific Applications on Power-aware CLusters. In *SC05*, 2005.
32) MinYeol Lim, VincentW. Freeh, and DavidK. Lowenthal. Adaptive, Transparent Frequency, and Voltage Scaling of Communication Phases in MPI Programs. In *SC06*, 2006.
33) Rajamani K. and Lefurgy C. On Evaluating Request-Distribution Schemes for Saving Energy in Server Clusters. In *ISPASS 2003*, 2003.
34) Cosmin Rusu, Alexandre Ferreira, Claudio Scordino, and Aaron Watson. Energy-Efficient Real-Time Heterogeneous Server Clusters. In *RTAS 2006*, 2006.
35) Takayuki Imada, Mitsuhisa Sato, Yoshihiko Hotta, and Hideaki Kimura. Power Management of Distributed Web Servers by Controlling Server Power State and Traffic Prediction for QoS. In *HPPAC 2008*, 2008.