Vol.2011-ARC-197 No.7
Vol.2011-HPC-132 No.7
2011/11/28

ハードウェア同期機構を用いた省電力 MPI の実装と評価

堀　敦史[†1]　亀山豊久[†1]　並木美太郎[†2]
辻田祐一[†3]　石川裕[†1,†4]

コンピュータの省電力化は、現在のそして将来のスーパーコンピュータを実現するための壁として知られると同時に注目を集めている。これまで並列アプリケーションでは、ユーザレベル通信が高性能な通信を実現する手法として広く使われている。そこでは受信メッセージをポーリングで待つために、電力が無駄に消費されてしまう。本論文では、良く知られた 2 フェーズの待ちの技法を、メッセージの待ちに適用する方式を提案する。この提案手法は、1) スピンループとブロッキングシステムコールの組み合わせ、2) スピンループと x86 の monitor/mwait 同期命令を用いた組み合わせ、の 2 種類で実装された。この提案手法は、これまでに提案されている CPU の DVFS を用いた省電力化の手法と異なり、アプリケーションの性能を損なわずに省電力化が可能になる。評価結果から、NAS 並列ベンチマークの FT では計算ノード全体の消費電力を 3% 削減することができた。

Low Energy Consumption MPI Using Hardware Synchronization

Atsushi Hori,[†1] Toyohisa Kameyama,[†1]
Mitarou Namiki,[†2] Yuichi Tsujita[†3]
and Yutaka Ishikawa[†1,†4]

The power wall of current and future supercomputer is gathering attentions. The technique of user-level communication to achieve high communication performance is widely used by parallel applications, and processes are spinning-wait for the incoming messages. This spinning loop in the absence of incoming messages is simply wasting energy and thus increase the power consumption of a parallel computer. In this paper, it is proposed to decrease the power used for the waiting loop by applying the well-known two-phase synchronization technique. This technique is implemented in two ways; 1) combination of spin-loop and blocking system call, and 2) combination of spin-loop and using the x86 monitor/mwait synchronization instructions which put computational core into a low-power mode. Unlike the techniques using the DVFS function of CPU, our proposed technique does not sacrifice application performance but can save energy. Evaluations show that more than 3% total system power can be saved with the FT application of NAS parallel benchmarks.

1. はじめに

10 PFLOPS の性能を誇る「京」コンピュータの消費電力は 20 MWatt 以下と公表されている[14]。今後 10 年以内のエクサ級のコンピュータが登場すると予測されており、「京」の 100 倍の性能を「京」と同じ消費電力で実現する必要があるとされている[2]。同時に、スーパーコンピュータのランニングコストも大きな問題となっており、たった数パーセントの消費電力の違いでも無視できない金額となる。このように、スーパーコンピュータの消費電力は現在および将来に渡り重要な関心事となっている。

HPC 分野におけるプロセス間通信では MPI が広く使われている。その内部で使われている低レベル通信デバイスの多くは、ユーザレベル通信[13] で実現されており、割込を用いず、OS カーネル機能をバイパスすることで低遅延、高バンド幅の通信を可能としている。しかしながら OS カーネル機能をバイパスした代償として、メッセージの待つためにポーリングする必要が生じている。

ポーリング中のプロセッサは電力を積極的に消費する。もし、あるプロセスが受信メッセージ待ち以外に有効な処理が存在しない場合には、ポーリングで消費される電力は無駄に費やされることになる。逆に、ユーザレベル通信ではなく、割込を用いてシステムコールでブロッキングするようにした場合には、消費電力が低減される可能性がある一方で、通信性能の低下が懸念される。

HPC においては、アプリケーションの性能を維持したまま、消費電力を低減する方式が望まれる。本稿では、このポーリングで消費される電力の無駄を減らすために、MPI における新しいメッセージ待ちの方式を提案し、アプリケーションの性能低下を抑え、消費電力を低減することが可能であることを示す。

†1 理研 計算科学研究機構 / RIKEN AICS
†2 農工大 / Tokyo University of Agriculture and Technology
†3 近畿大学 / Kinki University
†4 東京大学 / The Univercity of Tokyo

Vol.2011-ARC-197 No.7
Vol.2011-HPC-132 No.7
2011/11/28

2. 関連研究

Choi らは CPU の電圧と動作周波数を変化させる DVFS (Dynamic Voltage and Frequency Scaling) 機能を使って、プログラムが頻繁にメモリアクセスする局面で省電力モードに移行させる方式を提案し、CPU の消費電力を最大 70% 低減させることが可能であると報告している[1)]。しかしながら、最大 20% の処理性能ペナルティも同時に発生している。

Huang と Feng は CPU のパフォーマンスカウンタを用いて CPU の状態を検知し、その結果から DVFS 制御をおこなうするデーモンプロセスを用いたシステムを構築し、NPB[7)] ベンチマークプログラムを使って評価している[3)]。その結果、例えば NPB の FT では、10% 以上の性能ロスが生じながらも 30% 以上の CPU 消費電力の削減に成功している。

三輪らは、CPU のパフォーマンスカウンタを用いずに、メモリの消費電力から CPU の状態を推定し、その結果に基づいて DVFS 制御をおこなう方式を提案し[16)]、OpenMP で並列化された NPB の LU で 3% の性能ロスに対し 10% の消費電力削減に性能している。

Zamani らは Myrinet や QsNet を用いた HPC クラスタ上で、MPI の peer-to-peer 通信および collective 通信における DVFS と性能の関係のプロファイルを予め作っておき、それを実行時に適用する方式を提案している[15)]。

Vishnu らは ARMCI[8)] における片方向通信を DVFS を用いて省電力化する方式を提案している[12)]。この論文では、片方向通信での消費電力と性能の評価に留まっている。

以上のように、これまでに提案されてきた消費電力の削減手法は DVFS を用いている。この方式では、CPU の性能を低下させてもアプリケーションの性能に大きな影響を与えないコード部分や状況をいかに見つけるかが焦点となっている。また、DVFS を用いて省電力化した場合に性能低下が伴うとの報告が大半である。

本稿で提案する方式は、DVFS 制御を用いないで消費電力の削減を試みるという点、もうひとつは、ブロッキングシステムコールを用いてもユーザレベル通信よりも少しの性能低下で消費電力の削減の可能性を示す、という 2 点でこれまでの研究に対しユニークと考える。

3. 省電力な MPI メッセージ待ちの実装

3.1 2 フェーズの待ち

Intel X86 の拡張命令である mwait 命令は、直前の monitor 命令で指定されたアドレスのメモリの内容が書き変わるまでコアを省電力モードにして待つ[4)]。この命令を備える CPU 上で動く Linux では、カーネルがアイドル時にコアが省電力になるように monitor 命令と mwait 命令が用いられている。この結果、ブロッキングシステムコールによりコアがアイドル状態になれば、そのコアは省電力モードになる。

ブロッキングシステムコールを用いて MPI などのメッセージを受信すると、消費電力の低減が期待できるが、通信性能が低下してしまう。一方、ブロッキングシステムコールを用いないユーザレベル通信[13)] では、高い通信性能が実現できるが消費電力が高くなってしまう。これと似たような問題として、古くにプロセスあるいはスレッド間の同期制御がある[6),9)]。この方法では、最初はポーリングでしばらく待ち、その後にブロッキングするシステムコールで待つ、という 2 フェーズで待つ方式が提案されている。この方式が提案された背景にはポーリングでは、他のプロセスに CPU 資源を明け渡すことという目的があった。しかし、現在の HPC においては細かい粒度でのマルチタスクは不要であるが、同じ方式が、消費電力を減らすという目的に応用することが可能である。

この 2 フェーズの待ちを用いて MPI 通信の受信待ちを実現する方法を考える。MPI ではノード内の通信とノード間の通信で異なる通信方式（MPI のデバイス）が実装されていることが多いが、以下本稿では、ノード内での実現方式にのみ焦点をあてる。

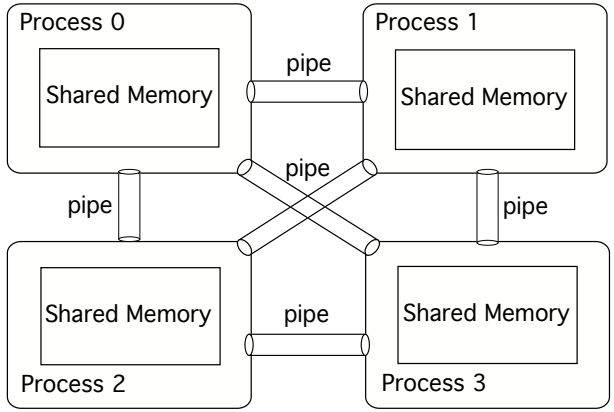

図 1 ブロッキング方式

ノード内通信における現状の多くの実装では、通信し合う 2 つのプロセスで共有されるメモリ領域を用いて実現されている。相手にメッセージを送る時、メッセージの内容を共有メモリ領域内のバッファに書込む。受信側はメッセージが書込まれた事を示すフラグをポー

Vol.2011-ARC-197 No.7
Vol.2011-HPC-132 No.7
2011/11/28

リングでチェックすることで、受信メッセージの待ちが実現されている。複数のコアからなる最近の CPU では、受信すべき相手のプロセスは複数あるため、それぞれの送信元に対し、このフラグをチェックする。

この通信をブロッキング方式とするためには、各プロセスの間を pipe で結んでおき、送信メッセージを書込んだ時点で、1 バイト pipe に書込む。受信側では複数の相手からの受信を待つ pipe に対し `select()` システムコールによりブロッキングして待つ。この方式は、データの送受信はユーザレベルの機構を用いているが、受信メッセージを待つという同期の部分にのみブロッキングするシステムコールを用いていることになる（図 1)。この方式を以下本稿では「ブロッキング方式」と呼ぶこととする。

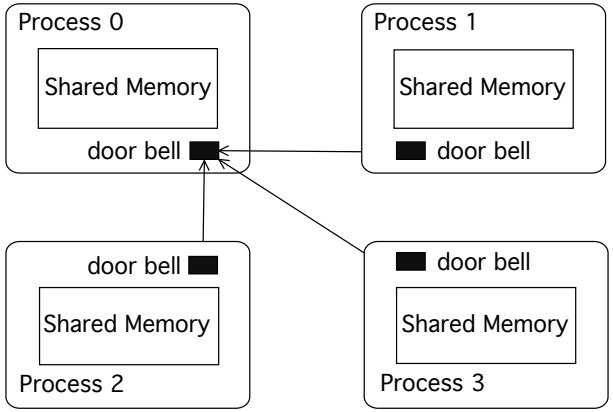

図 2 MINT 方式

一方、ユーザレベル通信においてシステムコールでなく直接 monitor/mwait 命令を用いれば、システムコールのオーバヘッドがなく高性能な通信を維持したまま、コアを省電力化することができる。しかしながら、先に説明した共有メモリ上のバッファにあるメッセージ受信のフラグに対し、直接 monitor/mwait 命令対で待つことはできない。Monitor/mwait の命令対で待つことができるアドレスはひとつに限られている。多くの場合、ノード内には 3 つ以上のプロセスが存在し、自分以外の複数のプロセスからメッセージが送られてくる可能性があり、それら全てのバッファに対しフラグをチェックしなければならないからである。この問題を回避するために、各プロセスは共有メモリ領域内にそれぞれひとつの “door bell” 変数を持ち、これに対し monitor/mwait 命令対で待つ方式を考える。プロセスがメッセージを送信したとすると、そのプロセスはそのメッセージの受信プロセスに対応する door bell 変数の内容を変更するようにする。この変更により受信プロセスは mwait 命令から抜けて、メッセージの受信を知ることができる（図 2)。これは先に説明したブロッキング方式における pipe を door bell 変数を置き換えたものと考えることができる。

文献[17] では、monitor/mwait 命令による待ちは決して速いものではない。したがい、ここでも上記ブロッキングの場合と同様に、最初はポーリングで、その後 monitor/mwait 命令対で待つ、という 2 フェーズの待ち方式が適切と考えられる。この方式を本稿では “MINT (Monitor INstruction Technoque)” と呼ぶことにする。

Monitor/mwait 命令を持つアーキテクチャ上での Linux においては、上記ブロッキング方式も MINT 方式も最終的には monitor/mwait 命令での待ちにより、コアが省電力モードになる。しかしながら、ブロッキング方式では、monitor/mwait 命令に到達するまでにかなりのオーバヘッドがある。もし受信待ちが発生してから monitor/mwait 命令に到達するまでの時間内にメッセージが到着することが頻繁にあれば、その時間はまるで無駄になってしまう。その結果、オーバヘッドで遅くなった分、かえって消費電力が増えてしまう可能性も考えられる。一方、door bell 変数に対し直接 monitor/mwait 命令を発行する方式ではこのようなオーバヘッドが生じない。

3.2 MPI におけるメッセージ受信待ち

図 3 は MPICH 実装における progress ルーチンの典型的な実装例を疑似コードで示したものである。MPICH において送信メッセージキューと受信メッセージキューがあり、ポーリング関数の中で送信メッセージキューにあるメッセージの送信と、ネットワークからの受信をそれぞれ試みる。

```
MPID_progress_function( ... ) {
    if( send_queue ) send_message( send_queue );
    try_recv_message( recv_queue );
}
```

図 3 MPI における progress ルーチンの疑似コード

ここで、単純に受信関数 `try_recv_message()` 関数内の受信メッセージ待ちを第 3.1 章で示した 2 フェーズの待ちに置き換えることは著しい性能低下を引き起こす可能性がある。なぜなら、送信キューにある送信メッセージの送出が、メッセージ受信の待ちにより遅延さ

Vol.2011-ARC-197 No.7
Vol.2011-HPC-132 No.7
2011/11/28

れてしまう可能性があるからである。したがい、図 3 のコードは以下の図 4 のようになるべきである。

```
MPID_progress_function( ... ) {
    if( send_queue ) {
        send_message( send_queue );
        try_recv_message( recv_queue );
    } else {
        lowp_recv_message( recv_queue );
    }
}
```

図 4 省電力 MPI における progress ルーチンの疑似コード

ここで、受信関数 `lowp_recv_message()` 関数は第 3.1 章で示した 2 フェーズの待ちにより、省電力なメッセージ受信関数である。これはつまり、送信すべきメッセージが送信キューに存在しない時にのみ、省電力な方式で受信メッセージを待つ、という方式である。

4. 評価実験

Intel Xeon (X5560、2.8 GHz、6 cores) を 2 ソケット持つサーバ（計 12 物理演算コア）上で、本稿で提案する省電力 MPI の有効性を検証するための評価実験をおこなった。なお、x86 の monitor 命令、および mwait 命令は特権命令なので、Kernel Mode Linux[11] を用いて計測した。SCore 7[10] パッケージ内の MPICH に対し、第 3 章に示した MPI を省電力化するための改良を施した。評価に用いたアプリケーションプログラムは NPB[7] の中から、CG, FT, LU, IS, MG の 5 つを用いた。これらのプログラムの実行において、問題の大きさは Class-C を、プロセス数は 8 とした。EP は MPI 通信がほとんどないこと、その他は実行に必要なプロセス数が先の 5 つと異なるため、本稿の評価から除外した。

図 5 に本実験における電力の計測の結線を示す。計測に用いた積算電力計は株式会社アイエスエー社製の DN-4104A である[5]。本製品には Ethernet があり、`rsh` コマンドにより遠隔から積算電力（KWh）の値を読み出すことが可能である。しかしながら積算電力の更新間隔は 1 分であるため、サンプリング誤差を考慮して、同じ NPB プログラムを 30 分以上繰り返し実行し、その時間区間で消費された電力量を計測した。

実験結果

図 6 は 2 フェーズにおけるフェーズが移行するしきい値（ポーリングループの回数、横

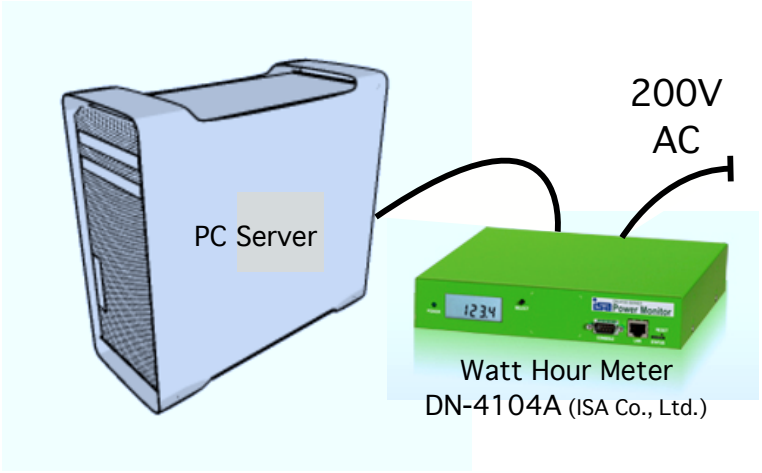

図 5 電力の計測方法

軸、図中 ”Threshold” と表記）を変化させた時の、性能（NPB における “Mop/s total” の値）と消費電力量の値を、省電力の改造を施さない通常の MPI を用いたときの実行を 1 としたときの割合で示したものである。図 6 中、ブロッキングシステムコール（実際には Linux の `select()` システムコール）と、第 3.1 章で説明した MINT を用いた場合の両方のデータが示されている。

ブロッキングシステムコールの場合、しきい値が小さいとアプリケーションのペナルティが大きく、かつ消費電力も増大していることが分かる。しきい値が小さい場合は短い時間の待ちにおいてもブロッキングシステムコールが呼ばれ、そのオーバヘッドが性能および消費電力に悪影響を与えたものと考えられる。消費電力の増大に関しては、オーバヘッドによる実行時間が長くなる事で消費電力量が増えたものと考えられる。一方、MINT 方式では、しきい値の影響は、ブロッキングの場合に比べ非常に小さい。

図 7 は、図 6 におけるしきい値の値が 10^6 の場合における、消費電力量の比を X 軸、速度の比を Y 軸にとり、NPB プログラムのそれぞれの値をプロットしたものである。このグラフにおいて、左側が通常の MPI 実行より消費電力が少ないことを意味し、上側が速度が速いことを意味する。

この図から、ブロッキング方式でも MINT 方式でもしきい値の値が適切であれば、性能

Vol.2011-ARC-197 No.7
Vol.2011-HPC-132 No.7
2011/11/28

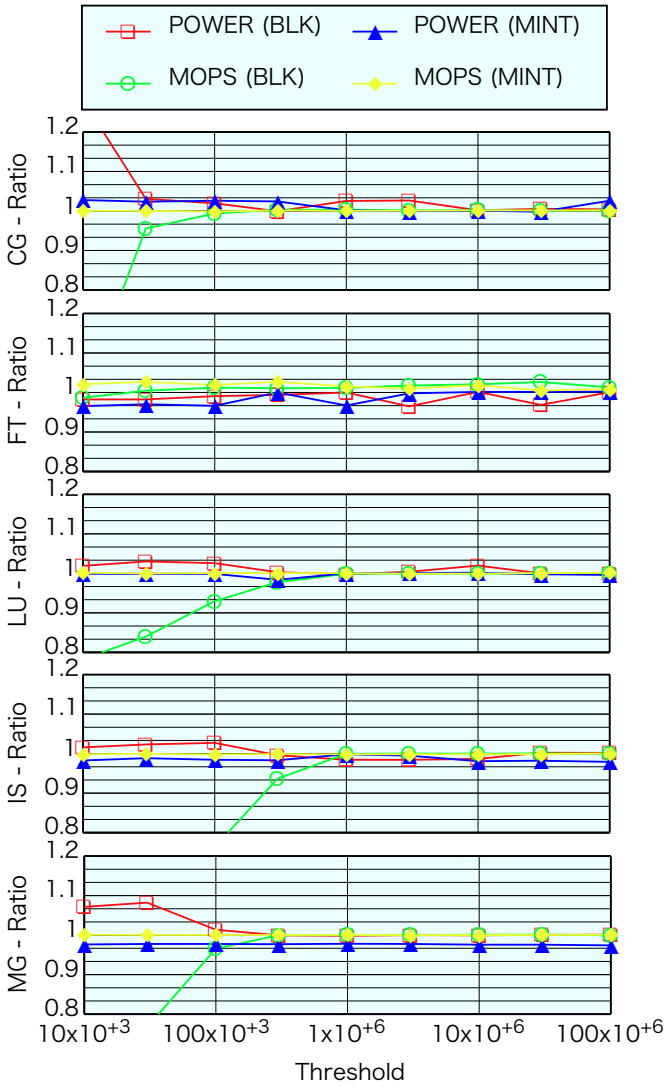

図 6 しきい値を変化させた時の電力と速度の変化

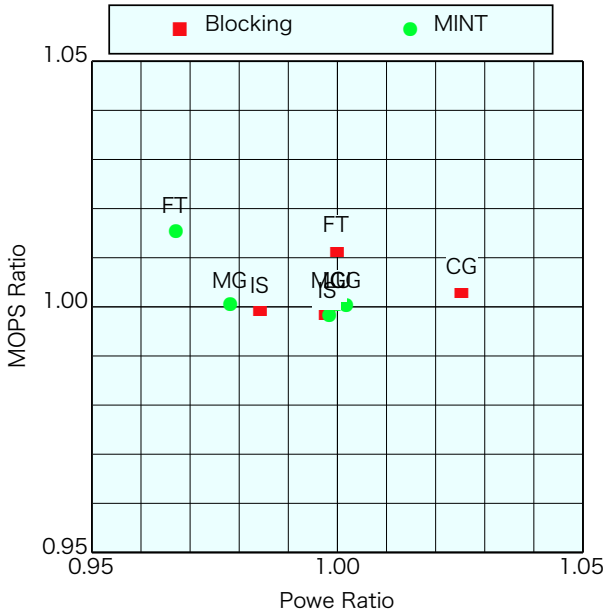

図 7 しきい値が 10^6 の時の電力と速度の比

低下がほとんど生じていないことが分かる。ブロッキング方式では IS が 2% 弱、MINT 方式では FT が 3% 強、MG で 2% 強、消費電力量が低くなっているのが分かる。残念ながら、ブロッキング方式の CG では 2%強程電力が増えてしまっている。

5. 考　　察

評価実験の結果から、MINT 方式の場合では、2 フェーズの待ちにおけるしきい値の値の影響は少ないが、ブロッキング方式においては、ポーリングループの回数が小さい場合に、性能が著しく低下することが確認された。ポーリングループの回数が大きくなるほど、省電力モードへの移行の機会が少なくなるため、消費電力の低減効果が減る。このためブロッキング方式の消費電力効果が MINT より少なくなったものと思われる。しかしながら、ブロッキング方式でそれなりに省電力の効果が見られた IS が、MINT 方式で同様の結果が見られなかったことの理由については現時点で不明である。

Vol.2011-ARC-197 No.7
Vol.2011-HPC-132 No.7
2011/11/28

論文[12] では、ARMCI 片方向通信[8] において DVFS を用いて省電力化を実現している。この論文では、ポーリングベースの実装と、ブロッキングシステムコールの実装がそれぞれ比較されている。ブロッキングシステムコールの場合では、ネットワークからの割込待ちの時に DVFS を下げ、割込があった時点で DVFS を上げることで、性能の低下を不正で省電力を実現しようとしている。この考え方は、基本的に本稿での提案と同じであるが、本稿では DVFS の代わりに monitor/mwait 命令を用いている点が大きく異なる。

論文[12] では ARMCI (Infiniband)、論文[15] では MPI (Myrinet と QsNet)、論文[3] では MPI (GbE) 、論文[16] では OpenMP を用いてそれぞれ評価がされている。論文[16] 以外では、全てノード間通信ネットワークを用いての評価であり、ノード内通信に比べ通信遅延が大きい。このため、省電力モードの時間が大きく取れる可能性が高い。MPI におけるノード内通信という低遅延での省電力化の評価は本稿が初めてと考える。論文[16] ではメモリアクセスが性能上のボトルネックとなっている局面で DVFS により省電力化を狙ったものであり、本稿のように通信の待ちの局面を対象とはしていない。

おわりに

本稿では、MPI におけるメッセージ受信待ちにおいて、ブロッキングシステムコールを用いた方法と monitor/mwait 拡張命令を直接用いた方法により、消費電力を減らす手法を提案し、計算ノード内の共有メモリを用いいた通信において、NPB を用いて評価した。計算ノード内通信という低遅延環境にも関わらず、特にブロッキングシステムコールにおいて、2 フェーズの技法を用いることで、NPB の IS における性能低下がほとんどなく、2% 弱の消費電力を減らすことに成功した。Monitor/mwait 命令を直接用いた MINT 方式では、2 フェーズのしきい値の影響が少なく、かつ FT で 3% 強、MG で 2% 強の省電力を達成することができた。

評価結果では、特にブロッキング方式において、アプリケーションの性能低下を伴わずに低消費電力化が実現できる可能性を示すことができたことの意義は大きいと考える。これまで提案されてきた DVFS を用いた方式よりもより容易に実装可能であるからである。

本稿での提案手法のより高精度な評価、および通信パターンと 2 フェーズにおけるしきい値の関係、Infiniband を用いた場合の実装方法とその評価、などについては今後の研究課題としたい。

謝辞 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) Choi, K., Soma, R. and Pedram, M.: Fine-grained dynamic voltage and frequency scaling for precise energy and performance tradeoff based on the ratio of off-chip access to on-chip computation times., *IEEE Trans. on CAD of Integrated Circuits and Systems*, pp.18–28 (2005).
2) Dongarra, J., Choudhary, A., Kale, S. et al.: The International Exascale Software Project Roadmap, White paper, Argonne National Laboratory (2010).
3) Huang, S. and Feng, W.: Energy-Efficient Cluster Computing via Accurate Workload Characterization, *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, CCGRID '09, Washington, DC, USA, IEEE Computer Society, pp.68–75 (2009).
4) Intel Corporation: *Intel 64 and IA-32 Architectures Software Developer's Manual* (2011).
5) ISA Co., Ltd.: DN-4104A. http://www.isa-j.co.jp/product/RS-485/it-sekisan/products/dn-4104a.html.
6) Lim, B.-H. and Agarwal, A.: Waiting algorithms for synchronization in large-scale multiprocessors, *ACM Trans. Comput. Syst.*, Vol.11, pp.253–294 (1993).
7) NASA: NAS Parallel BenchmarkS.
8) Nieplocha, J., Tipparaju, V., Krishnan, M. and Panda, D.K.: High Performance Remote Memory Access Communication: The Armci Approach, *Int. J. High Perform. Comput. Appl.*, Vol.20, No.2, pp.233–253 (2006).
9) Ousterhout, J.: Scheduling Techniques for Concurrent Systems, *3rd International COnference on Distributed Computing Systems*, IEEE, pp.22–30 (1982).
10) PC Cluster Consortium: SCore. http://www.pccluster.org/.
11) Toshiyuki Maeda: Kernel Mode Linux : Execute user processes in kernel mode. http://web.yl.is.s.u-tokyo.ac.jp/tosh/kml/.
12) Vishnu, A., Song, S., Marquez, A., Barker, K., Kerbyson, D., Cameron, K. and Balaji, P.: Designing Energy Efficient Communication Runtime Systems for Data Centric Programming Models, *Proceedings of the 2010 IEEE/ACM Int'l Conference on Green Computing and Communications & Int'l Conference on Cyber, Physical and Social Computing*, GREENCOM-CPSCOM '10, Washington, DC, USA, IEEE Computer Society, pp.229–236 (2010).
13) von Eicken, T., Basu, A., Buch, V. and Vogels, W.: U-Net: a user-level network interface for parallel and distributed computing (includes URL), *SIGOPS Oper. Syst. Rev.*, Vol.29, pp.40–53 (1995).
14) Yokokawa, M., Shoji, F., Uno, A., Kurokawa, M. and Watanabe, T.: The K com-

Vol.2011-ARC-197 No.7
Vol.2011-HPC-132 No.7
2011/11/28

puter: Japanese next-generation supercomputer development project, *Proceedings of the 17th IEEE/ACM international symposium on Low-power electronics and design*, ISLPED '11, Piscataway, NJ, USA, IEEE Press, pp.371–372 (2011).

15) Zamani, R., Afsahi, A., Qian, Y. and Hamacher, C.: A feasibility analysis of power-awareness and energy minimization in modern interconnects for high-performance computing, *Proceedings of the 2007 IEEE International Conference on Cluster Computing*, CLUSTER '07, Washington, DC, USA, IEEE Computer Society, pp. 118–128 (2007).

16) 三輪真弘, 中島耕太, 平井聡, 成瀬彰：メモリ消費電力に基づく CPU 周波数の動的制御, 研究報告「ハイパフォーマンスコンピューティング（HPC）」No.2011-HPC-130, 情報処理学会 (2011).

17) 堀敦史, 山本啓二, 大野善之, 今田俊寛, 亀山豊久, 石川裕：ハードウェア同期機構を用いた超軽量スレッドライブラリ, 研究報告「ハイパフォーマンスコンピューティング（HPC）」No.2011-HPC-130, 情報処理学会 (2011).