社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2006－HPC－107 (46)
2006／8／2

ヘテロなOSの計算資源を活用するグリッドRPCの設計

上村佳史[†] 中島佳宏[†,††] 佐藤三久[†]

大規模グリッド環境では従来のクラスタに加え，研究室やオフィスなどで用いられているPCを利用することが考えられる．このようなグリッド環境では各々のPCにはWindowsやLinuxといった様々なOSが用いられていると想定される．そのためヘテロなOS環境に対応した仕組みがGrid RPCに必要となる．本論文では，ヘテロなOS環境に対応するGrid RPCシステムとして，LinuxからWindowsを計算資源として利用するためのシステムを提案する．WindowsをGrid RPCのワーカに利用し，再コンパイルなしにLinuxプログラムを実行する方法として，Linuxプログラムを Windows上でそのまま実行可能とするためのLinuxプログラム実行環境BEEの設計・開発を行った．BEEを用いた評価実験によりシステムコールの実行性能はWindowsのネイティブプログラムと同程度の性能が得られた．また，OmniRPCを用いた実アプリケーションをLinuxとWindowsの混在環境での実行を行い，WindowsとLinuxのヘテロなOS環境でWindowsをGrid RPCのワーカとして利用できることを示した．

Design of Grid RPC system utilizing computing resources of heterogeneous OS

YOSHIFUMI UEMURA,[†] YOSHIHIRO NAKAJIMA[†] and MITSUHISA SATO [†]

To build a large scale grid environment, PCs in laboratory and offices are expected to be used as computational resources in addition to dedicated conventional clusters. PCs are running under various kinds of OS including Windows and Linux. To make use of such PCs as workers of Grid RPC system, we propose a mechanism of Grid RPC system that utilizes Windows PCs as computing resources from Linux PC. We have designed and implemented a system called BEE to allows a workers program exported from Linux to run under Windows, and it is integrated into OmniRPC system. BEE enables Linux binary program to run under Windows without recompilation. We have evaluated its performance of system calls and realistic applications in heterogeneous OS environment that contain Windows and Linux. The result shows that system calls on BEE can be executed at almost the same performance with a Windows native binary, and the Grid RPC system successfully utilizes PC of Windows OS as computing resources in heterogeneous OS environment.

1. はじめに

近年，ネットワーク技術の進歩により広域に分散した計算機資源を統合し，共有することで大規模計算などを行うグリッドコンピューティングに関する技術が注目されている．このグリッドコンピューティング環境上に分散した計算資源を利用するための並列プログラミングモデルの一つとしてGrid RPCが提案されている．Grid RPCでは，広域に分散された計算機資源を遠隔手続き呼び出し(Remote Procedure Call)の形で利用することが出来る．

多くの計算機資源を利用することが必要となる大規模グリッド環境を考えた場合，従来のPCクラスタに加えて，オフィスや研究室などで利用されているPCを利用することが考えられる．しかし，このようなPCにはWindowsやLinuxなどの様々なOSが用いられていると想定される．このようなヘテロなOS環境に対応し，オフィスなどで用いられているPCを計算機資源に利用するための方法がGrid RPCに必要となる．本稿では，このようなヘテロなOS環境に対応したGrid RPCシステムとして，WindowsとLinuxからなるヘテロなOS環境においてLinuxからWindowsをGrid RPCのワーカに利用するためのシステムを提案する．

LinuxからWindowsをGrid RPCのワーカに利用する方法として，次のような方法が考えられる．仮想マシンとLinuxプログラムをWindowsプログラムへの再コンパイルである．まず，VMware[8]やQEMU[2]などの仮想マシンを用いる方法の場合，この仮想マシン上にLinux環境を構築することができる．これにより仮想マシン上のLinuxをGrid RPCのワーカとして利用できる．

次に，Linuxプログラムを Windowsプログラムへ再

† 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
†† 日本学術振興会特別研究員
Research Fellow of the Japan Society for the Promotion of Science

コンパイルする方法である．これはCygwin[3]環境を用いることで簡単に実現できる．Cygwinではプログラムとソースコードでの互換性があるため，Cygwin上でLinuxプログラムを再コンパイルすることにより，Windowsで利用できるプログラムを作成することができる．

我々は，WindowsとLinuxのヘテロなOS環境においてLinuxからWindowsを利用する方法として，Linuxで作成されたプログラムを，そのままWindows上で実行するシステムとしてLinuxプログラム実行環境BEEを設計した．

本稿の構成は以下のようになっている．まず，2章で提案するLinuxプログラムをWindows上で実行する方式について述べる．3章では提案システムであるLinuxプログラム実行環境BEEの設計と実装について述べる．4章ではGrid RPCとして用いたOmniRPCの概要と，BEEを用いた場合のOmniRPCのRPCシステムについて述べる．5章では性能評価について述べる．最後にまとめと今後の課題について述べる．

2. Linuxプログラムの Windows上での実行方式

ここでは，提案するシステムであるLinuxプログラムをWindows上でそのまま実行するシステムの特徴を，仮想マシンや再コンパイルする方法と比較を通して述べる．

仮想マシンを用いることで，通常のLinuxと同じように扱うことができる．これによりGrid RPCシステムにWindowsに対応する仕組みを加えることなくWindowsを計算機資源として利用することが可能となる．しかし，Windows上に仮想マシンを導入し，さらにLinuxのOSを導入しなければならないという問題がある．さらに，仮想マシンは専用のメモリやディスク領域を多く使用する．そのためホストOSが仮想マシンに占有されてしまう問題がある．また，WindowsをGrid RPCのワーカとして利用するのにLinux環境全てが必要なわけではない．

Cygwinを利用してLinuxプログラムをWindowsプログラムに再コンパイルする方法の場合，ソースコードでの互換性があるため，再コンパイルすることでWindowsプログラムとすることができる．しかし，この場合，Linuxに加えてWindows環境を用意し，Windowsプログラムを用意する必要がある．また，もとのGrid RPCにWindowsに対応した仕組みを加えなければならない問題がある．

提案するLinuxプログラムをそのままWindows上で実行するシステムの場合，Windowsに仮想マシンほどの環境やLinuxOS環境は必要とせずにすむ．そのため，仮想マシンで使用されるメモリやディスクといった専用の領域が必要ではなくなる．また，Linuxプログラムを Windows上で実行できるため，Windows用のプログラムを必要としなくなる．

3. Linuxプログラム実行環境BEEの設計と実装

ヘテロなOS環境において，LinuxからWindowsをGrid RPCのワーカとなる計算機資源として利用するために，Linuxプログラム実行環境としてBEEを開発した．ここでは，BEEの設計と実装について述べる．

Linuxは様々なアーキテクチャに対応しているが，BEEではLinuxプログラムをそのままWindows上で実行することを目的として実装しているため，IA-32アーキテクチャ上で動作するLinuxのプログラムを対象とする．

BEEのシステムは，2つの部分から構成されている．1つはプログラムローダである．LinuxではELFなどの多種のプログラムのフォーマットが用いられているが，WindowsではPEフォーマットなどがプログラムのフォーマットとなっている．そのため，Linuxプログラムのフォーマットは Windowsでは対応していないため実行することができない．そこで，Linuxプログラムのフォーマットを解析，実行するためのプログラムローダの実装をしている．

もう1つはシステムコールである．WindowsとLinuxで同じIA-32アーキテクチャを用いた場合，プログラム自体はそのまま実行することは可能である．しかし，システムコールの実装はOS毎に異なっている．そのため，同じアーキテクチャであってもWindows上でLinuxプログラムをそのまま実行することが出来ない．そこで，Linuxのシステムコールを実行するための機能を実装している．

次に，プログラムローダとシステムコールのそれぞれの実装について述べる．

3.1 プログラムローダ

Linuxのプログラムを実行するには，そのフォーマットに対応したプログラムローダを実装することで実行可能となる．Linuxでは，プログラムのフォーマットは多くの種類が存在している．BEEでは一般的に普及しているフォーマットであるELF[1]を対象としてプログラムローダの実装を行った．また，ELFには動的リンクと静的リンクがある．BEEではプログラム単体で実行することが可能となる静的リンクした実行プログラムのみを対象とした．これは，動的リンクの場合，共有ライブラリをWindowsが持たなければならなくなるためと，共有ライブラリのバージョンが異なっていた場合に実行することができなくなってしまうという問題が起きるためである．プログラムローダはLinuxプログラムのフォーマットの解析を行い，ローダプログラム上にプログラムを展開し実行する．このときの仮想メモリの状態を**図1**に示す．

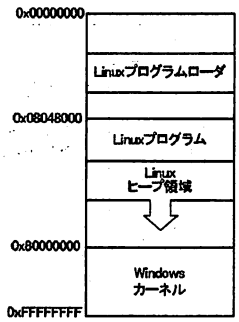

図 1 プログラムローダによる Linux プログラムの展開

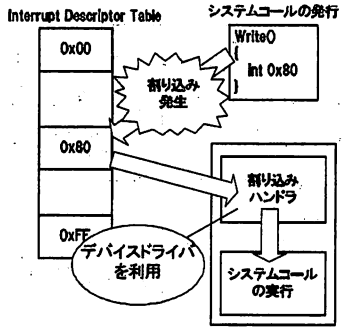

図 2 並列固有値計算プログラムによる性能評価

3.2 システムコール

入出力などの処理はシステムコールを発行することで実行できる．このシステムコールはOSにより異なる実装がなされている．そのためLinuxプログラムをWindows上で実行する場合，システムコールの実装が異なるため実行することができない．Windowsでのシステムコールは，NT/2000の場合は *int 2E*，XPなどの場合では *sysenter* 命令を用いて，ソフトウェア割り込みを発生させることで実行している．Linuxのシステムコールは *int 0x80* により，ソフトウェア割り込みを発生させ実行している．このようにWindowsでは *int 0x80* に対応する割り込み処理は無いためである．

WindowsではLinuxのシステムコールに用いられている *int 0x80* に対応する割り込み処理は特に指定されていない．そこで，BEEでは *int 0x80* に対応するソフトウェア割り込み処理を，新たに追加することでシステムコールの実行を可能にしている．新たにソフトウェア割り込みの処理を追加する方法としてBEEではデバイスドライバを利用している．

割り込み処理を追加には，割り込みディスクリプタテーブル(Interrupt Descriptor Table)[5]を操作する必要があり，割り込みディスクリプタテーブルに指定する割り込み処理はカーネル上に用意しておかなければならない．通常のユーザプログラムではユーザ空間上にプログラムが展開されてしまうため，これを割り込み処理として割り込みディスクリプタテーブルに指定することができない．しかし，デバイスドライバを用いた場合は可能となる．デバイスドライバはOSの起動時にOSに組み込まれカーネル上に展開される．そのためデバイスドライバを利用して，カーネル上に新たな割り込みの処理を追加することができる．これにより，Linuxのシステムコールに対応した処理などの任意のソフトウェア割り込み処理などを実装することができる．デバイスドライバを利用して割り込みディスクリプタテーブルに割り込み処理を追加したときの実行の流れを**図2**に示す．

システムコールのソフトウェア割り込み処理はデバイスドライバを利用することで可能となるが，実際のシステムコールの処理を実装する必要がある．Linuxではシステムコールは多くの種類が存在する．しかし，WindowsをGrid RPCのワーカとして利用するのに，このような数多くのシステムコールを実装する必要はない．そこで，BEEではワーカとして利用するのに必要最低限のシステムコールのみを実装する．現在実装しているシステムコールはwrite/readなどの基本的なファイルI/Oと，マスタとのデータのやり取りに必要なネットワークI/Oのみを実装している．

4. OmniRPCの概要とBEEを用いたOmniRPCのRPCシステム

Linuxプログラム実行環境BEEで用いるGrid RPCシステムとして，OmniRPC[7][6]を使用した．OmniRPCは，ローカルなクラスタ環境から広域ネットワークで構成されたグリッド環境までのシームレスな並列プログラミングを可能にするマスタ/ワーカ型のGrid RPCシステムである．

OmniRPCのシステムを**図3**に示す．OmniRPCのLinuxでの実装では，マスタからの遠隔手続き呼び出しにより，リモート実行プログラムを実行する．このとき，直接ワーカのリモート実行プログラムを実行はせずに，sshなどにより認証を行った後にAgentをリモートノード上に起動する．この後，マスタからのRPCコールはAgentを経由し，リモート実行プログラムの実行はAgentが行う．

BEEを用いた場合のWindowsでのOmniRPCシステムを**図4**に示す．Windowsはワーカとしての利用をするため，マスタとして利用するのはLinuxのみである．OmniRPCはAgentをsshなどによる認証を行った後に起動するが，Windowsの場合，このようなsshによる認証などは通常はできない．そのためAgentは常に起動した状態で動作するように変更を加える必要がある．また，リモート実行プログラムはLinuxプログラムであるので，AgentはLinuxプログラム実行環境であるBEEを経由してプログラムを実行するように変更する．後はLinuxでのOmniRPCと同じように

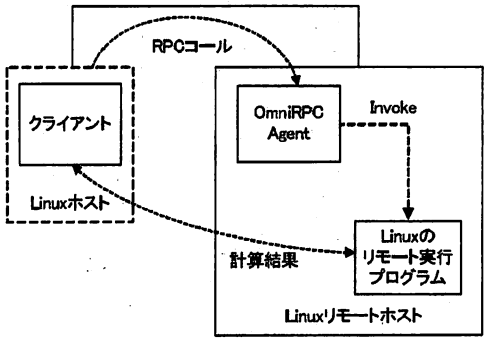

図 3 OmniRPC の RPC システム

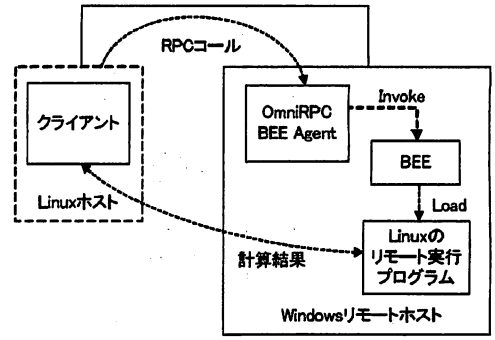

図 4 BEE を用いた OmniRPC の RPC システム

BEE により実行された Linux プログラムがマスタである Linux ホストと通信を行う.

また, OmniRPC で想定しているグリッド環境では, インターネット上で複数のクラスタ環境が相互接続されている環境である. そのような環境では, それぞれのクラスタ環境内では通常ファイルの共有がなされている. しかし, 大規模グリッド環境を想定した遊休 PC を利用する場合では, ノードは単一の PC として存在するためファイルの共有をすることができない. そこで, 単一の PC を簡便に利用できるようにするために, ワーカが必要なリモート実行プログラムを所持していない場合には, マスタが自動的に必要とするリモート実行プログラムを配布する機能を実装した. これにより, Windows や Linux においてリモート実行プログラムなどをあらかじめ取得しておくといった必要がなくなった.

5. 性能評価

提案システムである BEE との性能比較には, VMware と Cygwin を用いて行った. 性能評価には, それぞれの環境におけるシステムコールの実行性能と通信バンド幅の性能, 実アプリケーションによる性能として OmniRPC を用いた並列固有値計算プログラムにより評価する. また, ヘテロな OS 環境で実行として Windows と Linux の混在環境で動作確認を行った.

5.1 実験環境

実験は本システムを用いる環境である, マスタに Linux を使用しワーカに Windows を使用する環境で行った. **表 1** にそれぞれのマシン環境と VMware 環境, 使用したノード数を示す. VMware と Cygwin 環境は Windows と同じマシン環境上に構築した.

5.2 システムコールの性能評価

システムコールの性能評価は, 基本的なファイルへの Write/Read システムコールにより行った. 測定の方法は, 1 万回のシステムコールを発行し, バッファサイズを変化していったときの実行時間を比較することにより評価をした. また, 性能比較の基準として Windows のネイティブプログラムの測定も行っている.

表 1 測定環境

Master	Xeon 2.4GHz, メモリ 1GByte, Gigabit Ethernet Linux kernel 2.6.9, gcc 3.4.5, 1 ノード
Worker (Windows)	Xeon 3.2GHz, メモリ 2GByte, Gigabit Ethernet WindowsServer2003 Enterprise Edition SP1 4 ノード
Worker (Linux)	Xeon 2.4GHz, メモリ 1GByte, Gigabit Ethernet Linux kernel 2.6.9, 4 ノード
VMware	仮想メモリ 1GByte, ネットワーク接続 ブリッジ Linux kernel 2.6.11, gcc 4.0.0

ファイルへの write システムコールを行ったときの測定結果を**図 5**, read システムコールを行ったときの測定結果を**図 6** に示す. 図の縦軸が, 1 万回のシステムコールを発行するのに要した時間であり, 10 回測定した平均値である. 横軸がそのときのバッファサイズである.

BEE は Windows のネイティブプログラムや Cygwin と同程度の性能が得られているが, 1 回のシステムコールの発行に平均で, Windows のネイティブプログラムと 0.6μsec, Cygwin と 0.5μsec ほど遅くなっている. これは, BEE の実装では Linux プログラムによるシステムコール発行時と Windows API を呼び出す時の 2 度の割り込みの発生してしまうため, 通常のプログラムよりも割り込みが多く発生することによるオーバヘッドのためであると考えられる. また, VMware では Write, Read ともにバッファサイズが 10KByte あたりから実行性能が大きく低下している.

5.3 通信バンド幅の性能評価

通信バンド幅の性能測定は Linux とそれぞれの環境で行った. **図 7** に結果を示す. 図の横軸がメッセージサイズである. このメッセーサイズを用いて, 各ノード間でメッセージの ping-pong を行い, この送受信に要した時間から転送バンド幅を算出した.

通信バンド幅においても BEE は Windows のネイティブプログラムと同程度の 93.9MByte/sec の性能が

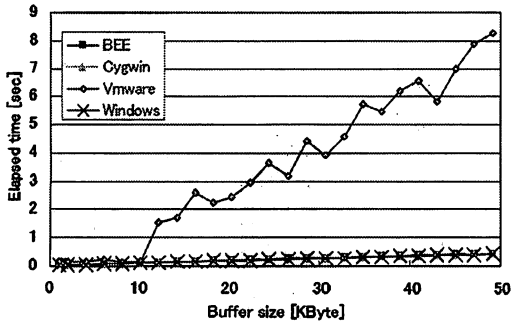

図 5　ファイルへの write システムコールの実行時間

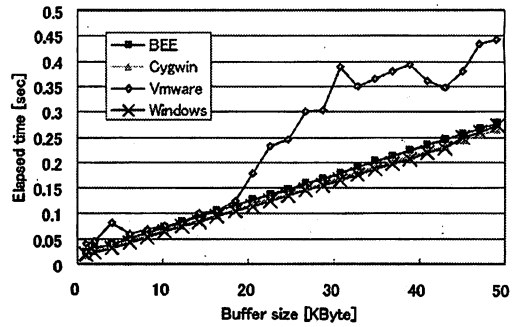

図 6　ファイルへの read システムコールの実行時間

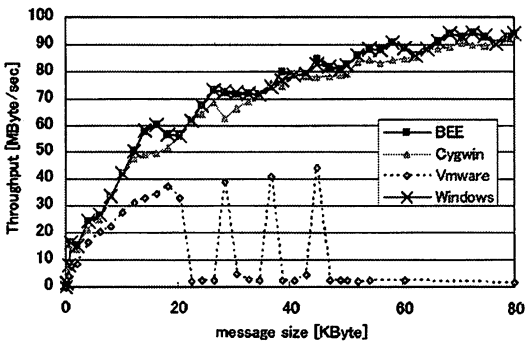

図 7　環境によるスループットの比較

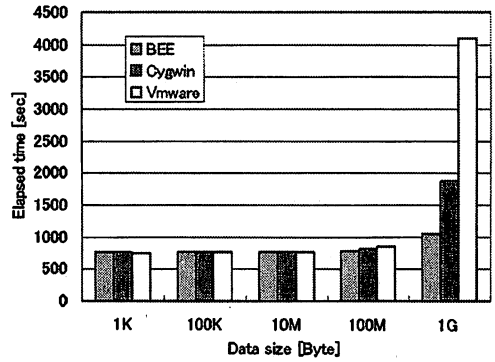

図 8　OmniRPC を用いたデータサイズによる実行速度の比較

得られた．しかし，Cygwin では 93.6MByte/sec と多少性能が低下している．これは Cygwin ではメッセージ転送の際にメモリコピーが発生しているためである．また，VMware ではバッファサイズが 20KByte あたりから急激な性能低下が発生している．

5.4 OmniRPC での実行性能

予備評価として，Grid RPC を用いて 1 ジョブに 30 秒を必要とし，これが 100 ジョブある場合の性能を示す．Grid RPC システムとして OmniRPC を用い，それぞれの環境で Windows を 4 ノード用いた．**図 8** に結果を示す．横軸が 1 ジョブを実行するときにワーカに転送するデータサイズである．

この結果より，データサイズが 100MByte までは実行性能にほとんど差は見られない．データサイズが 1GByte になるとデータ転送に時間がかかってしまい，BEE や Cygwin では実際に使用されたのは 3 ノード，VMware ではネットワーク速度などの性能が悪いため 2 ノードであった．また，逐次での実行性能に比べ，4 ノードで 4 倍の実行性能が出ている．

5.5 並列固有値計算プログラムを用いた性能評価

実アプリケーションとして OmniRPC を用いた並列固有値計算プログラム[9] による性能評価を行った．このプログラムは，大規模な一般固有値問題のための解法であり，複素空間上の円周上の点に対応する方程式を並列に解くことにより，その円周内にある固有値を効率的に求めるものである．**図 9** に，BEE，Cygwin，VMware のそれぞれの環境で，ワーカを 2 ノード，4 ノードとしたときの実行結果を示す．図の縦軸は 5 回実行したときの実行時間の平均値である．

この結果より，BEE 環境が Cygwin 環境よりも若干遅くなっていることがわかる．このような結果となったのは，まず OmniRPC の通信は 1KByte 単位で送受信を行っている．そのため，Cygwin と比較した場合に通信性能で，BEE の割り込みによるオーバヘッドが現れるためと考えられる．また，並列固有値計算プログラムでは通信に加えて mmap システムコールが多くの回数実行されている．しかし，現在の BEE での mmap システムコールの実装は，あまり効率的な方法とはなっていない．そのため，mmap システムコール

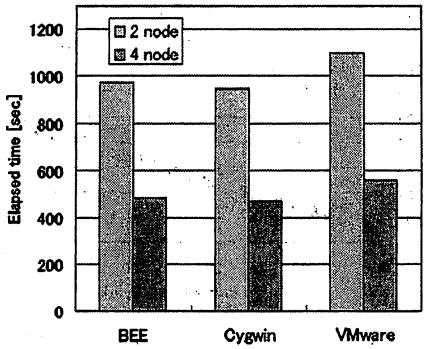

図 9 並列固有値計算による性能評価

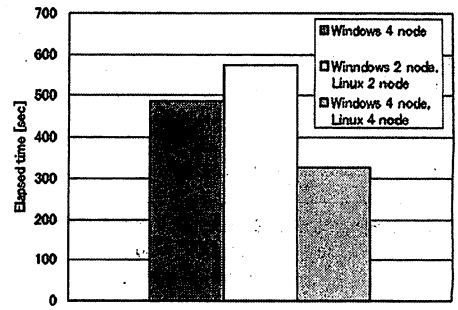

図 10 並列固有値計算の Windows と Linux の混在環境での実行

が多くの回数実行されるようなプログラムでは，かなりのオーバヘッドとなってしまうと考えられる．

5.6 Windows と Linux の混在環境での実行

並列固有値計算プログラムを用いて，Windows と Linux の混在環境として，Windows と Linux を 2 ノードずつと 4 ノードずつの場合で測定を行った．**図 10** に結果を示す．ここで Windows2 ノードと Linux2 ノードの実行時間が Windows4 ノードよりも遅くなっているのは，Linux マシンの性能が Windows マシンの性能と異なるためである．この結果より，Windows と Linux が混在した場合でも利用できることが確認できた．

6. まとめと今後の課題

本稿ではヘテロな OS を計算機資源とする Grid RPC システムとして，Linux と Windows のヘテロな OS 環境における Linux からの Windows のワーカ利用の方法として，Windows 上で Linux プログラムを実行するための実行環境 BEE を開発・評価を行った．

BEE ではプログラムローダと，システムコールのための割り込み処理を Windows に追加することで，Windows 上で Linux プログラムを実行させることを可能にした．BEE により，仮想化環境や Windows 用のプログラムを必要とせずに Grid RPC のワーカとして利用できるようになった．

性能評価の結果，システムコールの実行性能は Windows ネイティブプログラムと同程度の性能が得られた．また，Linux と Windows の混在環境においても実行を確認し，ヘテロな OS 環境で Windows を Grid RPC のワーカとして利用することが可能となった．

今後の課題として，Linux kernel のバージョンが 2.6 になってから，スレッド関連の実装[4] が変化している．また，新しいスレッドモデルの実装は glibc などのバージョンなどとも関連している．そのためプログラムの実行方法も変化しているため，現在の BEE では実行することが不可能である．このような問題があるため，動作できるように BEE を変更していくことを考えている．また，Linux と Windows 間でのプロセスマイグレーションの実装を検討している．

謝辞 本研究の一部は，文部科学省科学研究費補助金 基盤研究 A 課題番号 172002，特別研究員奨励費 課題番号 17・7324，および，日仏共同研究プログラム (SAKURA) による．

参 考 文 献

1) *TIS Committee: Tool Interface Standard(TIS) Executable and Linking Format(ELF) Specification version 1.2* (1995).
2) Bellard, F.: QEMU, a Fast and Portable Dynamic Translator, *Proceedings of USENIX 2005 Annual Technical Conference*, pp. 41–46 (2005).
3) Cygwin: *http://sourceware.org/cygwin/*.
4) Drepper, U. and Molnar, I.: The Native POSIX Thread Library for Linux, *http://people.redhat.com/drepper/nptl-design.pdf*.
5) Intel(R): *IA-32 IntelR Architecture Software Developer's Manuals* (2004).
6) Nakajima, Y., Sato, M., Boku, T., Takahashi, D. and Gotoh, H.: Performance Evaluation of OmniRPC in a Grid Environment., *SAINT Workshops*, pp. 658–665 (2004).
7) OmniRPC: *http://www.omni.hpcc.jp/OmniRPC/*.
8) VMware: *http://www.vmware.com/*.
9) 櫻井鉄也, 多田野寛人, 早川賢太郎, 佐藤三久, 高橋大介, 長嶋雲兵, 稲富雄一, 渡邊寿雄梅田宏明: 大規模固有値問題の master-worker 型並列解法, 情報処理学会 ACS 論文誌, Vol. 46, No. 10, pp. 1–8 (2005).