情報処理学会研究報告
IPSJ SIG Technical Report

Vol.2009-HPC-121 No.12
2009/8/4

シームレスなMPI環境を実現するMPI-Adapterの設計と性能評価

住元 真司[†1] 中島 耕太[†1] 成瀬 彰[†1]
久門 耕一[†1] 安井 隆[†2] 鴨志田 良和[†3]
松葉 浩也[†3] 堀 敦史[†3] 石川 裕[†3]

本論文では、クラスタシステム上でシームレスなMPI実行環境を実現するFortranとC言語対応版のMPI-Adapterの設計と性能評価について述べる。Fortran用のMPIライブラリは、FortranからC言語へのABI(Application Binary Interface)変換ライブラリとして実現される場合が多い。この場合、Fortran用のMPIライブラリからC言語用のMPIライブラリを直接呼び出すため、MPI-Adapterを適用した場合にC言語版のMPI-Adapterが呼び出されABI不整合となる。この問題を回避するために、実行時に呼び出し先を変更するライブラリを開発した。性能評価の結果、ABI変換機能が動作すること、その性能劣化は最小限に抑えられることがわかった。

A Design and Evaluation of MPI-Adapter for Seamless MPI Computing Environment

Shinji Sumimoto,[†1] Kohta Nakashima,[†1]
Akira Naruse,[†1] Kouichi Kumon,[†1] Takashi Yasui,[†2]
Yoshikazu Kamoshida,[†3] Hiroya Matsuba,[†3]
Atsushi Hori[†3] and Yutaka Ishikawa [†3]

This paper presents a design and evaluation of MPI-Adapter, which supports both of Fortran and C languages for seamless MPI computing environment on cluster systems. Many MPI libraries for Fortran language are developed as ABI translation library from Fortran to C language. In such a case, a Fortran MPI library calls functions on MPI library for C language directly, therefore, in MPI-Adapter environment, a Fortran MPI library calls MPI-Adapter library for C language instead of calling MPI library. This causes ABI mismatch and MPI-Adapter does not work properly. To solve the problem, We have developed a library which changes function call address on runtime. Our evaluation results show that MPI ABI translation is worked properly, and its overhead is very small.

1. はじめに

我々は、クラスタシステム上でシームレスなMPI通信ライブラリ実行環境を実現するMPI-Adapter[1)]を開発している。本論文におけるシームレスなMPI通信ライブラリ実行環境とは、MPI通信ライブラリの実装に依存せず、OpenMPIやMPICH2、そしてベンダ固有のMPIなどの様々な実装上で一つの実行バイナリが実行できる環境を意味する。文献[1)]で報告した通り、MPI通信ライブラリはAPI (Application Program Interface)を規格しているが、ABI (Application Binary Interface)を規格化していない。このため、MPI通信ライブラリ規格で規定されているデータ型やオブジェクトの実装方法は異なる。例えば、OpenMPIのMPI_Comm型はアドレス型で表現され、64bitアドレスのプロセッサ上では64bitの長さを持ち、MPICHのMPI_Comm型は整数型で32bitの長さを持つ。このため、OpenMPI環境でコンパイルされた実行バイナリは、OSおよびアーキテクチャが同一であってもMPICH2などOpenMPI環境以外のライブラリ環境では実行できない。文献[1)]では、シームレスなMPI通信ライブラリ実行環境の実現可能性を確認するために、C言語に限定したMPI-Adapterを試作評価し、その有用性とオーバヘッドが小さいことを示した。

本論文では、FortranとC言語に対応したMPI-Adapterの設計と性能評価について述べる。Fortran用のMPI処理系の実装はFortran用MPI ABIをC言語用MPI ABIに変換するABIトランスレータとして実装されているものが多い。これは、Fortran用のMPIライブラリからC言語用のMPIライブラリを直接呼び出す構成であることを意味している。したがって、Fortran用のMPI-Adapterを適用した場合、Fortran ABIを変換した後に再度C言語用のMPI-Adapterが呼び出されることになり、ABIに不整合が生じる。この問題を回避するために、Fortran ABIを変換するライブラリのC言語用の関数呼び出し先を実行時に変更するライブラリ（実行時関数スワップライブラリ）を開発した。このライブラ

†1 富士通研究所
FUJITSU LABORATORIES
†2 日立製作所
HITACHI
†3 東京大学
The University of Tokyo

リは、ダイナミックリンクライブラリ (DLL) の外部関数呼び出しの仕組みを利用して実現しているため、DLL を再コンパイルすることなく、実行時に特定の関数群の呼び出し先をプログラムで変更できる。

実行時関数スワップライブラリを実装して Fortran と C に対応した MPI-Adapter を実装、評価した。評価の結果、ABI 変換の機能が問題なく動作し、MPI-Adapter の性能差のオーバヘッドは小さいことがわかった。

本論文では、まず 第 2 章で目指す MPI-Adapter の実現方針と課題について述べる。第 3 章で、課題を解決するための MPI-Adapter の設計について述べる。第 4 章で MPI-Adapter の実装、第 5 章で評価、第 6 章で関連研究について述べる。

2. Fortran 対応 MPI-Adapter の実現方針と課題

MPI-Adapter の実現はシームレスな MPI の実行環境を実現するために行っており、その実現方針は次の 3 つである。

(1) 既存 MPI 実装との動作互換性を最大限に確保する (動作互換性)。
(2) 設定と利用の簡単さを実現する (利用簡易性)。
(3) シンプルな作りでより多くの MPI 処理系に適用可能とする (適用性, 実現容易性)。

以上の方針に基づき、MPI-Adapter は、バイナリ互換性を確保するため、ABI を変換する DLL として実現されており、低オーバヘッドで ABI 変換を実現している。

本章では、Fortran 対応の MPI-Adapter の実現課題を述べる。まず、既存の Fortran 対応の MPI 処理系のアーキテクチャについてまとめた後、MPI-Adapter を実現するための課題について述べる。

2.1 Fortran 対応の MPI 処理系のアーキテクチャ

図 1 に典型的な Fortran 対応の MPI 処理系として Open MPI のアーキテクチャを示す。図 1 において、C 用の MPI ライブラリ関数 (*libmpi.so*) が実際の MPI 関数処理と通信処理を行う。また、Fortran 用の MPI ライブラリ (*libmpi_f77.so*) は、ABI を変換し C 用の MPI ライブラリを直接呼び出す (F2C 変換)。この場合、図 1 の右上の C で書かれたプログラムは直接 C 用の MPI ライブラリが使われることになるが、左上の Fortran で書かれたプログラムは、一度 Fortran 用の MPI ライブラリで ABI を変換された後、C 用の MPI ライブラリで実際の MPI 処理がされることになる。

この F2C 変換のアーキテクチャとすることにより、C 用のライブラリに処理を集中し、実装を簡略化すると共に、複数の言語が混在された場合にも通信リソースの一貫性を確保する構成となっている。また、Fortran だけでなく、C++用の MPI ライブラリも同様のアーキテクチャとなっている。

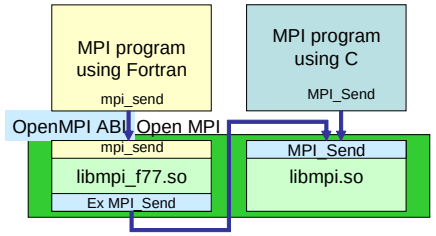

図 1 典型的な Fortran 対応の MPI 処理系の構成

F2C 変換により実現している MPI 処理系は、Open MPI, MPICH2, HP MPI, そして MPICH2 から派生した Intel MPI, Microsoft MPI がある。

2.2 Fortran 版の MPI-Adapter の課題

本節では、前節で述べたアーキテクチャを持つ MPI 処理系に MPI-Adapter を適用することを考える。図 2 に、Open MPI のバイナリを MPICH2(MPICH) の実行環境で動作させる MPI-Adapter の構成を示す。図 1 の構成に Fortran 用と C 用の MPI-Adapter が挿入される構成になっている。

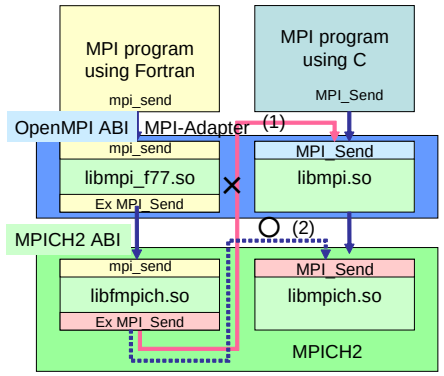

図 2 Fortran 対応の MPI-Adapter の構成と課題

図 2 における、MPI-Adapter のライブラリの動作を考えると、C で書かれた MPI プログラムの場合は、C 用 MPI-Adapter ライブラリから、実際の MPICH2 ライブラリを呼び出す動作になり問題はない。しかし、Fortran で書かれた MPI プログラムの場合は Fortran 用 MPI-Adapter ライブラリ (*libmpi_f77.so*) から Fortran 用 MPICH2 ライブラリ (*libfmpich.so*) が呼ばれるまでは問題ないが、Fortran 用 MPICH2 ライブラリが C 用の MPI ライブラリを呼び出した時点で、本来呼ばれるべき MPICH2 用のライブラリ (*libmpich.so*) は呼び出されず (図 2, (2))、ライブラリの依存関係が先の C 用 MPI-Adapter ライブラリ (*libmpi.so*) 上の関数が呼び出されることになる (図 2, (1))。この場合、ABI が異なる関数を呼び出すため、引数の型、引数の意味する値が異なると正常に動作しない。

このように、Fortran 用の MPI-Adapter を実現する場合、この ABI の不一致を回避する必要がある。

3. Fortran 対応 MPI-Adapter の設計

本章では、Fortran 対応の MPI-Adapter の設計について述べる、最初に既存の仕組みによる解決手法の問題点を明らかにし、設計の方向性を決める。そして、これを解決する実行時関数スワップ方式による Fortran 対応 MPI-Adapter の設計について述べる。

3.1 既存の仕組みによる解決手法

ここでは既存の仕組みによる解決手法について述べる。実行バイナリを変更しないで既存の仕組みにより解決する手法として次の 3 つが考えられる。

(1) 複数言語の混在を許可しない
(2) 実行環境の Fortran ライブラリを再構築
(3) 実行環境の Fortran と C の ABI 変換を MPI-Adapter で実現

それぞれの手法について得失を議論する。

3.1.1 複数言語の混在を許可しない (混在禁止)

この手法は、複数言語の混在を許可しないことにより、課題を回避する手法である (図 3)。具体的には、Fortran の場合には空の C 用ライブラリを準備して DLL の依存関係を解決させ、実行環境の C 用の MPI ライブラリを呼び出させる手法である。実行環境用の MPI ライブラリの名前を変更してもよい。それぞれの言語用に準備して、LD_LIBRARY_PATH により実行時に切り替えることにより実現できる。

この手法の得失は次のようになる。

Pros: 特別な工夫をすることなく、簡単に実現が可能

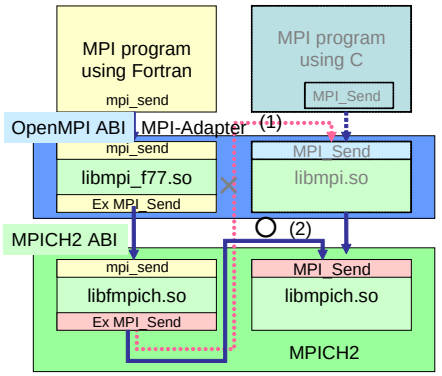

図 3 複数言語の混在を許可しない

Cons: 複数言語混在時には対応不能で動作互換性が確保できない。実行バイナリ毎に C 用か Fortran 用かを判断して環境切り替えが必要で利用が複雑になる。

3.1.2 実行環境の Fortran ライブラリを再構築 (実行環境再構築)

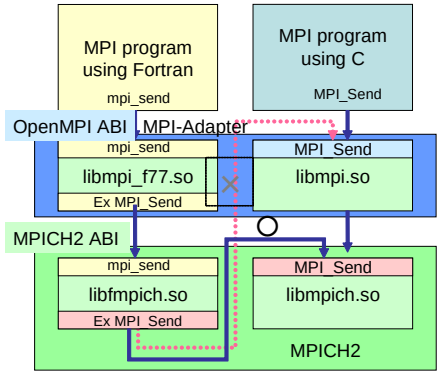

図 4 実行環境の Fortran ライブラリを再構築

この手法は、実行環境の MPI ライブラリを再構築して、Fortran 用のライブラリを変更し、C 用のライブラリの呼び出しを固定化する手法である (図 4)。固定化の手法としては、次の 2 つがある。

(1) Fortran 用ライブラリの関数の名前を変更、C 用のライブラリには、変更した名前を追加する。(ex. weak シンボルで定義)

(2) MPI-Adapter の仕組みを使って、関数呼び出しを固定する。

この手法の得失は次のようになる。

Pros: 全体の構成がシンプルになり、確実に目的の関数を呼び出せる。

Cons: 実行環境を再コンパイルする必要があるため、ソースとコンパイル環境が必要。ソースが手に入らないベンダ製 MPI には対応不能で適用 MPI 処理系に制限がある。

3.1.3 実行環境の Fortran と C の ABI 変換を MPI-Adapter で実現 (F2C 実現)

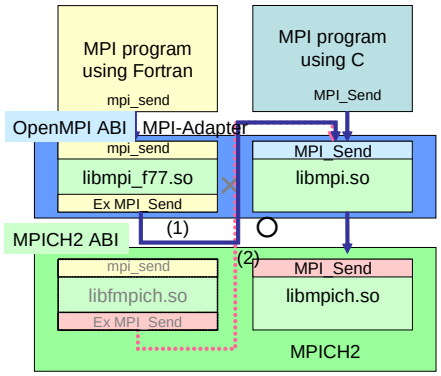

図 5 実行環境の Fortran と C の ABI 変換を MPI-Adapter で実現

この手法は、Fortran 用の MPI-Adapter で F2C 変換を行うことにより、実行環境の Fortran 用の MPI ライブラリの呼び出しを不要にするものである (図 5)。

この手法の得失は次のようになる。

Pros: 全体の構成がシンプルになり、確実に目的の関数を呼び出せる。

Cons: MPI 処理系毎に F2C の仕様をエミュレートする必要があり、実装が複雑になる。自動化は難しい。処理系により、数値→ポインタ→数値変換になりオーバヘッドが増え、性能上のインパクトがある。

この手法の発展形として、直接実行バイナリの Fortran ABI を実行時の C 用の MPI ライブラリに直接変換する手法も考えられるが、自動化は難しく、処理系の組み合わせ毎に解析する必要があり、実現容易性に問題がある。

3.1.4 Fortran 対応 MPI-Adapter 実現の方針

第 2 章で述べた Fortran 対応 MPI-Adapter 実現の方針に照らし合わせ、これまで述べた既存の方式を評価する。

表 1 既存方式の比較

	動作互換性	利用簡易性	適用性	実現容易性
混在禁止	×	×	○	○
実行環境再構築	○	○	△	△
F2C 実現	○	○	○	×

表 1 に既存の 3 つの手法の比較結果を示す。表 1 において実行環境再構築の適用性が△であるのは、ソースがあれば実現可能だが、ソースがないと実現できないので、△としている。表 1 の比較結果から、すべての条件を満たす既存の方式はないことがわかる。

ここで、ABI の不一致の問題の本質を考えてみると、実行環境の Fortran 用 MPI ライブラリの呼び出し先を実行環境の C 用 MPI ライブラリに固定できないのが問題の本質である。この課題を実行環境再構築手法では、再コンパイルで解決しようとしていたが、本来、再コンパイルすることなく実行環境の Fortran 用 MPI ライブラリの呼び出し先を実行環境の C 用 MPI ライブラリに固定できる手法を実現すべきである。

3.2 実行時関数スワップ方式の提案

前節で議論した ABI の不一致の問題を解決するために、実行時関数スワップ方式を提案する。実行時関数スワップ方式は DLL 自体が実行時に呼び出し先のアドレスを解決する仕組みを利用して、前節の実行環境再構築方式と同様の動作を、実行環境を再構築することなく実現する方式である。本節では、Linux における、実行時リンクの仕組みを述べた後、実行時関数スワップの動作について述べる。

3.2.1 Linux 上の実行時リンクの仕組み

Linux の実行環境 (Ex. CentOS, Fedora Core) においては、実行バイナリは ELF フォーマットを用いているものが多い。この実行フォーマット上で、動的リンクの仕組みは、Procedure Linkage Table(PLT) と Global Offset Table(GOT) の 2 つのテーブルにより実現されている。図 6 に Linux における ELF 実行バイナリの構成の抜粋と実行時におけるプロセス空間のマップイメージを示す。左上が ELF バイナリから、PLT、実行テキスト、GOT のセクションを抜き出した図である。DLL の ELF バイナリにおいては、プロセス実行時のマップアドレスからの相対アドレスでの動作を想定したプログラムとなっている。各セク

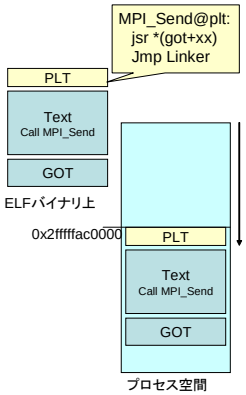

図 6 Linux における実行バイナリと実行時リンクの仕組み

ションについて述べる。

Procedure Linkage Table(PLT): 実行時リンクの基本コードが埋め込まれている。GOT に格納されたアドレスにより、飛び先が変更され動作が変わる。

Text: 実行テキストが格納されている。外部の関数を呼び出す場合は、PLT にある関数エントリを呼び出す。

Global Offset Table (GOT): 各 PLT 毎に割り当てられたテーブルである。初期化時は PLT 内の jsr 命令の次のアドレスで初期化されている。

図 6 でプログラム実行時において、テキスト上の *call MPI_Send* が最初に呼ばれたとき、PLT 内の *MPI_Send@plt* が実行される。*jsr *(got+xx)*[⋆1] の実行時、GOT の初期値は *jsr *(got+xx)* の次のアドレスで初期化されており、動的リンカが呼び出され関数アドレスを解決する。その関数アドレスが新たに GOT に格納され、次回からは格納されたアドレスに実行が移動する。

この過程において、関数の実行先を変更するには関数名に応じ GOT に格納されたアドレスを変更すればよいことが分かる。

3.2.2 実行時関数スワップ方式の動作

実行時関数スワップ方式は、前節の PLT と GOT の仕組みを利用して、実行環境再構築方式と同様の動作を、実行環境を再構築することなく実現する。

図 7 に実行時関数スワップ方式適用時の動作を示す。図 7 において、MPI-Adapter に対応した *libmpi_f77.so* により、Open MPI の Fortran ABI が MPICH2 の ABI に変換され、*libfmpich.so* が呼び出される。実行時関数スワップ方式では、*libfmpich.so* の GOT を初期化時に *libmpich.so* 内の関数に変更することにより、*libfmpich.so* 内から直接 *libmpich.so* 内の関数を呼び出す (図 7 の 1) 2))。

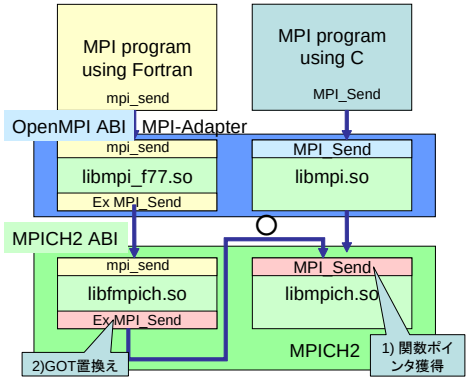

図 7 実行時関数スワップ方式の動作

3.3 その他の設計

その他の設計については、C 用の MPI-Adapter[1] と同じである。

各 MPI 型実装の違い： Fortran の MPI 型はすべて 32 ビット数値型の実装であるため、数値間の変換だけで実装可能である。

各 MPI 型の予約語の抽出と変換： *mpif.h* から半自動で抽出可能にし、変換関数の自動生成が可能な作りとする。

4. Fortran 対応の MPI-Adapter の実装

本章では、Fortran 対応の MPI-Adapter の実装について述べる。Fortran 対応の MPI-Adapter は、C 用の MPI-Adapter[1] と同じく Linux 上で実現されているが、実行時関数スワップを実現するライブラリの実現と、32 ビット数値型の型変換に限定されるため、型変換のコードが簡略化されている。本章では特に実行時関数スワップの実装について述べる。

⋆1 実際のアセンブラコードから簡略化している

4.1 実行時関数スワップの実装

実行時関数スワップの実装は、第 3.2 節に述べたように、DLL 上の GOT の格納アドレスを置き換えることにより実装している。その手順と実装を次に示す。

(1) 実行環境の Fortran 用 MPI ライブラリ (図 7 の *libfmpich.so*) の ELF ファイルから、置き換え対象の各関数の GOT の DLL 内の相対アドレスの獲得
- 対象ライブラリの ELF ファイルから、対象のセクションを読み出して生成する。
- ELF 参照用のライブラリとして *libelf* ライブラリを使用している。

(2) 実行環境の Fortran 用 MPI ライブラリのプロセス空間上の仮想アドレスの獲得
- 対象ライブラリのファイル名を元に */proc/PID/maps* ファイルから対象ライブラリの先頭マップアドレスを獲得している。

(3) 実行環境の C 用 MPI ライブラリ (図 7 の *libmpich.so*) での置き換える関数の呼び出しアドレスの獲得
- C 用の MPI-Adapter[1] と同様に、*dlopen()*, *dlsym()* を用いて実装している。

(4) GOT の DLL 内の相対アドレス、先頭マップアドレスから、*libfmpich.so* がマップされた GOT のアドレスを計算し、(3) で取得した実行環境の C 用 MPI ライブラリの関数呼び出しアドレスを実行プロセス空間内の *libfmpich.so* の GOT に格納する。

4.2 その他の実装

C 用の MPI-Adapter[1] に比べて次の点を改良している。

全体の作りの変更： 複数の種類の MPI 処理系に対応するために、スケルトン用コードと型値格納用のファイルに分類し、MPI ヘッダから半自動で MPI のプリセットの値を抽出可能にしている。

変換関数の効率化： 関数の呼び出し段数を減らし高速化を実現している。

5. MPI-Adapter の評価

本章では、Fortran 対応の MPI-Adapter の評価を行う。MPI-Adapter の評価環境として、SCore[2] 搭載のクラスタで評価する。表 2 に評価環境を示す。ネットワークは Gigabit Ethernet と共有メモリであり、低レベル通信層は PMX/Etherhxb と PMX/Shmem である。MPI は MPICH2/SCore を用いている。評価では OpenMPI バイナリを MPI-Adapter を用いて MPICH2/SCore で実行した結果と MPICH2/SCore の Native 実行結果を比較する。

本環境において、MPI-Adapter のオーバヘッドと NAS 並列ベンチマークにより、MPI-Adapter による性能差の程度を調べる。

表 2 評価環境

Computation Node	DUAL Xeon 3.8GHz Primargy RX200S2
Ethernet (1Gbps)	Intel E1000 NIC Netgear 48port GigE Switch
OS	CentOS 5.1 x86_64 (2.6.18-8.el5 kernel) SCore7.0

5.1 MPI-Adapter の呼び出しオーバヘッド

MPI-Adapter のオーバヘッドを調べるため、MPI の pinpong プログラム (C 版、Fortran 版) を作成し、4 バイトの pingpong のラウンドトリップ時間を MPICH2/SCore と Open MPI のバイナリを MPI-Adapter を用いて測定した。用いたデバイスは共有メモリ上の PMX/Shmem デバイスである。

表 3 MPI ラウンドトリップ時間 (共有メモリデバイス)

	Fortran	C	Overhead (per MPI call)
OpenMPI+MPI-Adapter	3.154 μs	3.065 μs	0.082(0.022) μs
MPICH2/SCore	3.103 μs	3.055 μs	0.048(0.012) μs
Overhead (per MPI call)	0.051(0.013) μs	0.010(0.0025) μs	0.034(0.0085) μs

MPI-Adapter の結果は MPICH2/SCore の処理を含む

表 3 に測定結果を示す。表 3 より、F2C 方式で実現された Fortran 対応の MPI ライブラリのオーバヘッドは、C 用の MPI ライブラリに比べて MPI 関数あたり 0.012 μs であった（MPICH2/SCore の場合）。これに MPI-Adapter+MPICH2/SCore のオーバヘッドが MPI 関数あたり 0.022 μs であるので、MPI-Adapter 自体のオーバヘッドは 0.010 μs となる。これは機構上、MPI-Adapter のオーバヘッドは F2C のオーバヘッドと同等レベルであると言える。

また、共有メモリ上での通信時間に対してのオーバヘッドは 3.154 μs に対して 0.051 μs であるので、1.6%に相当する。F2C のオーバヘッドより小さく無視できる程度である。

5.2 NAS 並列ベンチマークによる性能差

アプリケーション性能における MPI-Adapter の性能差を調べるために、NAS 並列ベン

チマーク BT, CG, FT, LU, MG, SP の (Class A,B,C) を用いて測定を行った。利用に用いた Fortran コンパイラは gfortran である。

表 4 に 16 ノード (ノードあたり 1 プロセス) の測定結果を元にした MPI-Adapter のオーバヘッドの算出結果を示す。

表 4 NAS 並列ベンチマークの実行オーバヘッド

(16node)	BT	CG	FT	LU	MG	SP
Class-A	-0.1%	3.4%	0.2%	0.4%	-3.7%	0.0%
Class-B	-2.2%	3.7%	2.9%	-0.5%	-0.8%	1.6%
Class-C	0.3%	1.2%	0.4%	0.1%	-5.0%	0.6%

マイナスの場合は MPI-Adapter が性能が高いことを意味する

表 4 より、性能劣化はアプリケーション全般的に 3.7%以下となった。いくつかの結果で MPI-Adapter の結果の方が性能がよい結果となった。元々、MPI-Adapter 自体のオーバヘッドは、第 5.1 節での評価より、無視可能な程度小さく抑えられているので、直接的なものである可能性は低く、タイミングの問題などの間接的なものであると考えられる。

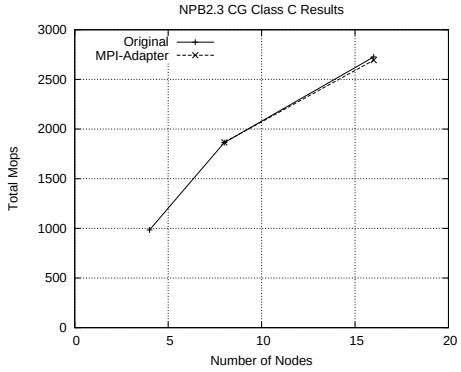

図 8 CG Class B の性能比較

このうち、性能差の出た CG Class B, FT Class B, MG Class C について、それぞれ、図 8,9,10 に性能のグラフを示す。各アプリケーションが良好な性能向上を示していることがわかる。

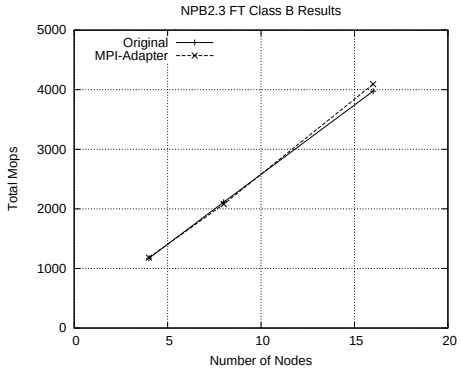

図 9 FT Class B の性能比較

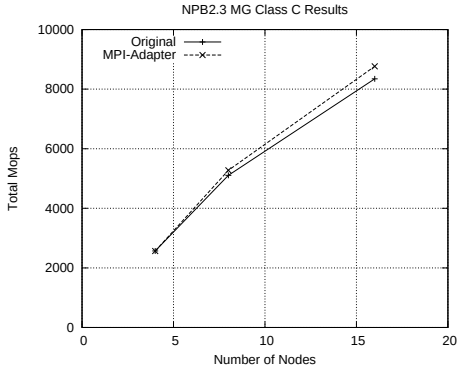

図 10 MG Class C の性能比較

測定した結果から、MPI-Adapter による直接的な性能劣化は特に気にするものでないことが分かる。間接的な性能劣化については調査中である。

5.3 実行時関数スワップ方式について

本論文で提案した実行時関するスワップ方式は、今回は MPI-Adapter の実現のために開発したが、本来、より汎用的、かつ応用できる方式であると考えている。

例えば、

- 関数をフックして異なる関数を置き換える時

- 統計用の関数を挿入する場合
- デバッグ用など、

他にも多くの適用が可能である。特に、プログラムの初期化時だけでなく、プログラム次第で実行時に動的に状況に応じて切り替えられるのが特徴である。

6. 関連研究

シームレスな MPI 実行環境に、関連する研究として、MPI フォーラムにおける ABI の統一化と MorphMPI がある。

- MPI フォーラムにおける ABI の統一化：MPI Forum では、次期の MPI の規格である MPI 3.0 の規格検討を進めており、その中で、Application Binary Interface Working Group[3] において、ABI の統一化の検討が進められている。
- MorphMPI[4]、共通の mpi.h(mpif.h) を実現し、かつ、実行ランタイム機能を含めて実現している。実際のつくりは、単純に mpi.h(mpif.h) で MPI 関連のシンボルを置き換え、変換ライブラリで MPI 関数を呼び出すことにより、シンボルの衝突を避けている。

以上の、既存研究と異なり、MPI-Adapter のアプローチは個々の MPI プログラムの MPI ランタイム実装を変更すること無く、バイナリ実行の可搬性を実現している点が異なる。

実行時関数スワップ方式については、DLL に関連したソフトウェアと研究が関連研究となる。

- glibc の ld[5]: プログラムの実行時にプログラムアドレスが未解決の関数を指定されたディレクトリパスから探し出して、プログラム空間にロードする。先行するライブラリが優先される。
- 仮想マシン (Xen[6], Wine[7] など): プログラムの性質上、DLL を制御して実行される関数を変更することで機能を実現している。

実行時関数スワップ方式は、MPI のシームレス実行のために特定の実行関数を置き換えている。

7. まとめ

本論文では、クラスタシステム上でシームレスな MPI 実行環境を実現する Fortran と C 言語対応版の MPI-Adapter の設計と性能評価について述べた。C 言語用と比較して Fortran 用の MPI ライブラリは、Fortran から C 言語への変換ライブラリとして実現される場合が多い。この場合、Fortran 用の MPI ライブラリから C 言語用の MPI ライブラリを直接呼び出すため、MPI-Adapter を適用した場合に C 言語版の MPI-Adapter が呼び出され ABI 不整合となる。この問題を回避するために、実行時に呼び出し先を変更する実行時関数スワップライブラリを開発した。性能評価の結果、ABI 変換機能が動作すること、その性能劣化は最小限に抑えられることがわかった。

今後の予定として、未実装部分の実装の完成度を高める、MPICH2 から OpenMPI 対応、他 MPI の対応を行う予定である。開発されたプログラムは SCore[2] の distribution に含まれる予定である。

謝辞

本研究は、文部科学省「e サイエンス実現のためのシステム統合・連携ソフトウェアの研究開発」からの支援を受けている。

参考文献

1) 住元真司, 中島耕太, 成瀬彰, 久門耕一, 安井隆, 鴨志田良和, 松葉浩也, 堀敦史, 石川裕. 並列プログラムの実行可搬性を実現する MPI 通信ライブラリの設計. 情報処理学会研究報告 09-HPC-119 (HOKKE'09). 情報処理学会, Feb 2009.
2) SCore Cluster System Software: http://www.pccluster.org/.
3) Application Binary Interface Working Group: https://svn.mpi-forum.org/trac/mpi-forum-web/wiki/AbiWikiPage.
4) MorphMPI: http://morphmpi.sourceforge.net/.
5) Glibc: http://www.gnu.org/software/libc/.
6) Xen: http://www.xen.org/.
7) Wine: http://www.winehq.org/.