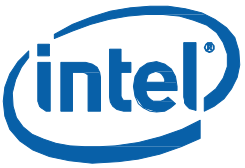

インテル® MPI ライブラリー for Linux*

ユーザーズガイド (5.1 Update 3)

目次

著作権と商標について

本資料は、明示されているか否かにかかわらず、また禁反言によるとよらずにかかわらず、いかなる知的財産権のライセンスも許諾するものではありません。

インテルは、明示されているか否かにかかわらず、いかなる保証もいたしません。ここにいう保証には、商品適格性、特定目的への適合性、知的財産権の非侵害性への保証、およびインテル製品の性能、取引、使用から生じるいかなる保証を含みますが、これらに限定されるものではありません。

本資料には、開発の設計段階にある製品についての情報が含まれています。この情報は予告なく変更されることがあります。最新の予測、スケジュール、仕様、ロードマップについては、インテルの担当者までお問い合わせください。

本資料で説明されている製品およびサービスには、不具合が含まれている可能性があり、公表されている仕様とは異なる動作をする場合があります。

MPEG-1、MPEG-2、MPEG-4、H.261、H.263、H.264、MP3、DV、VC-1、MJPEG、AC3、AAC、G.711、G.722、G.722.1、G.722.2、AMRWB、Extended AMRWB (AMRWB+)、G.167、G.168、G.169、G.723.1、G.726、G.728、G.729、G.729.1、GSM AMR、GSM FR は、ISO、IEC、ITU、ETSI、3GPP およびその他の機関によって制定されている国際規格です。これらの規格の実装、または規格が有効になっているプラットフォームの利用には、Intel Corporation を含む、さまざまな機関からのライセンスが必要になる場合があります。

性能に関するテストに使用されるソフトウェアとワークロードは、性能がインテル® マイクロプロセッサー用に最適化されていることがあります。SYSmark* や MobileMark* などの性能テストは、特定のコンピューター・システム、コンポーネント、ソフトウェア、操作、機能に基づいて行ったものです。結果はこれらの要因によって異なります。製品の購入を検討される場合は、他の製品と組み合わせた場合の本製品の性能など、ほかの情報や性能テストも参考にして、パフォーマンスを総合的に評価することをお勧めします。

Intel、インテル、Intel ロゴ、Intel Xeon Phi、Xeon は、アメリカ合衆国および / またはその他の国における Intel Corporation の商標です。

* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

Microsoft、Windows、Windows ロゴは、米国 Microsoft Corporation の、米国およびその他の国における登録商標または商標です。

Java は、Oracle Corporation 及びその子会社、関連会社の米国及びその他の国における登録商標です。

Bluetooth は商標であり、インテルは権利者から許諾を得て使用しています。

インテルは、Palm, Inc. の許諾を得て Palm OS ready マークを使用しています。

OpenCL および OpenCL ロゴは、Apple Inc. の商標であり、Khronos の使用許諾を受けて使用しています。

最適化に関する注意事項

インテル® コンパイラーでは、インテル® マイクロプロセッサーに限定されない最適化に関して、他社製マイクロプロセッサー用に同等の最適化を行えないことがあります。これには、インテル® ストリーミング SIMD 拡張命令 2、インテル® ストリーミング SIMD 拡張命令 3、インテル® ストリーミング SIMD 拡張命令 3 補足命令などの最適化が該当します。インテルは、他社製マイクロプロセッサーに関して、いかなる最適化の利用、機能、または効果も保証いたしません。本製品のマイクロプロセッサー依存の最適化は、インテル® マイクロプロセッサーでの使用を前提としています。インテル® マイクロアーキテクチャーに限定されない最適化のなかにも、インテル® マイクロプロセッサー用のものがあります。この注意事項で言及した命令セットの詳細については、該当する製品のユーザー・リファレンス・ガイドを参照してください。

注意事項の改訂 #20110804

1. はじめに

インテル® MPI ライブラリー for Linux* ユーザーズガイドは、一般的なシナリオでインテル® MPI ライブラリーを使用する方法を説明しています。MPI アプリケーションのコンパイル、リンク、実行、およびデバッグに関する情報とともに、クラスター環境への統合に関する情報を提供します。

このユーザーズガイドでは、以下の情報を提供します。

ドキュメント構成

セクション	説明
第 1 章 – はじめに	このドキュメントについて
第 2 章 - 利用モデル	インテル® MPI ライブラリーを導入するための利用モデル
第 3 章 - インストールとライセンス	インストールの手順を説明し、ライセンスに関する情報を提示
第 4 章 - コンパイルとリンク	MPI アプリケーションのコンパイルとリンクの方法を説明
第 5 章 - アプリケーションの実行	アプリケーションを実行する手順を説明
第 6 章 - デバッグとテスト	デバッガーからアプリケーションを実行する方法を説明
第 7 章 - プロセス管理	プロセス管理とパスワードなしの ssh 接続の設定に関連する情報
第 8 章 - `mpitune` ユーティリティーによるチューニング	インテル® MPI ライブラリー向けの最適な設定を見つけるため `mpitune` ユーティリティーを使用する方法を説明
第 9 章 - サポートされるジョブ・スケジューラー	ジョブ・スケジューラーとの統合方法を説明
第 10 章 - 一般的なクラスターに関する考察	MPI を利用する上で考慮すべき一般的なクラスターに関する情報を説明
第 11 章 – トラブルシューティング	一般的なトラブルシューティングの手順と例
第 12 章 - インテル® MIC アーキテクチャーでインテル® MPI ライブラリーを使用する	インテル® MIC アーキテクチャーとインテル® MPI ライブラリーを使用する際のいくつかの特殊な考慮事項について説明

1.1. インテル® MPI ライブラリーの紹介

インテル® MPI ライブラリーは、メッセージ・パッシング・インターフェイス 3.0 (MPI-3.0) 仕様を実装する、マルチファブリックをサポートするメッセージ・パッシング・ライブラリーです。インテル® プラットフォームに標準ライブラリーを提供します。

- 企業、事業部、部門および作業グループにおけるハイパフォーマンス・コンピューティング向けに業界で最高の性能を提供します。インテル® MPI ライブラリーは、インテル® アーキテクチャー・ベースのクラスター上でアプリケーションのパフォーマンスを向上することに注目します。
- 開発者の要求に応じて、MPI-3.0 に機能を導入できるようにします。

この製品には、次のコンポーネントが含まれています。

- インテル® MPI ライブラリー・ランタイム環境 (RTO) には、スケーラブルなプロセス管理システム (Hydra)、多目的デーモン (MPD)、サポート・ユーティリティーなどプログラムの実行に必要なツール、共有 (.so) ライブラリー、ドキュメントなどが含まれています。
- インテル® MPI ライブラリー開発キット (SDK) には、すべてのランタイム環境コンポーネントに加え、`mpiicc` などのコンパイラー・コマンド、インクルード・ファイルとモジュール、スタティック (.a) ライブラリー、デバッグ・ライブラリー、およびテストコードなどが含まれます。

1.2. この資料の対象

このユーザーズガイドには、経験豊富な開発者がインテル® MPI ライブラリーを導入するのを支援する、使い方の手順と例題による主要な機能の説明が含まれています。完全な説明は、『インテル® MPI ライブラリー for Linux* リファレンス・マニュアル』をご覧ください。

1.3. このドキュメントの表記について

このドキュメントでは、以下のフォント規則を使用しています。

表 1.3-1 このドキュメントの表記

`This type style`	コマンド、引数、オプション、ファイル名を示します。
`THIS_TYPE_STYLE`	環境変数を示します。
<この書式>	実際の値を挿入します。
`[ 項目 ]`	オプションの項目です。
`{ 項目 | 項目 }`	縦線で区切られた選択可能な項目です。
(SDK のみ)	ソフトウェア開発キット (SDK) 利用者向け

1.4. 関連情報

インテル® MPI ライブラリーに関する詳しい情報については、次のリソースをご覧ください。

- 『インテル® MPI ライブラリー・リリースノート』には、要件、技術サポート、および既知の問題に関する最新情報が含まれます。
- 『インテル® MPI ライブラリー・リファレンス・マニュアル』には、製品の機能、コマンド、オプションおよび環境変数に関する詳しい情報が含まれます。
- インテル® MPI ライブラリー for Linux* ナレッジベース (英語) では、トラブルシューティングのヒントや秘訣、互換性ノート、既知の問題、技術ノートが提供されます。

次のリソースもご覧ください。

インテル® MPI ライブラリー製品ウェブサイト

インテル® MPI ライブラリー製品サポート

インテル® クラスターツール製品ウェブサイト

インテル® ソフトウェア開発製品ウェブサイト

2. 使用モデル

インテル® MPI ライブラリーは、次の手順で導入します。

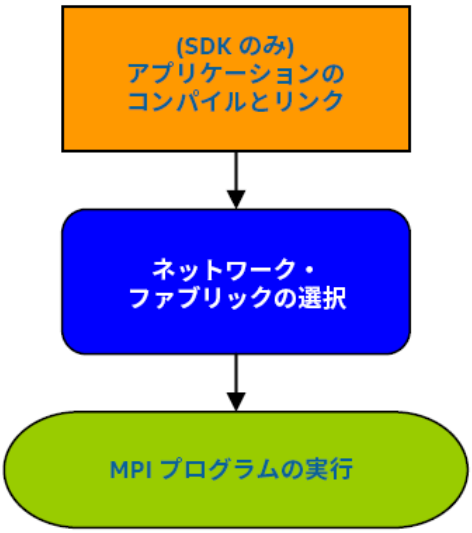

図 1: インテル® MPI ライブラリーを導入するための利用モデル

3. インストールとライセンス

このセクションでは、インテル® MPI ライブラリー・ランタイム環境 (RTO) とインテル® MPI ライブラリー・ソフトウェア開発キット (SDK)のインストール手順とライセンスに関する情報を提供します。

3.1. インテル® MPI ライブラリーのインストール

最新のインテル® MPI ライブラリー for Linuxs* をインストールする際に、以前のバージョンがインストールされている場合それをアンインストールする必要はありません。

次のコマンドで、`l_mpi[-rt]_p_<version>.<package_num>.tar.gz` パッケージを展開します。

```
tar –xvzf l_mpi[-rt]_p_<version>.<package_num>.tar.gz
```

このコマンドは、`l_mpi[-rt]_p_<version>.<package_num>` というサブディレクトリーを作成します。

インストールを開始するには、`install.sh` を実行します。インテル® MPI ライブラリーのデフォルトのインストール先は以下です。

```
/opt/intel/compilers_and_libraries_2016.<update>.<package#>/linux/mpi
```

2 つの異なるインストール方法があります。

- PRM ベースのインストール - インストールには root のパスワードが必要です。この製品は、自動共有ファイルシステムまたはクラスターの各ノードにインストールできます。
- 非 PRM インストール - このインストール方法では、root 権限は必要ありません。すべてのスクリプト、ライブラリー、およびファイルは目的のディレクトリー (通常は、ユーザーの `$HOME`) にインストールされます。

スクリプト、`include` ファイル、そしてライブラリーは、対象のアーキテクチャーごとのディレクトリーに格納されます。デフォルトでは、`<installdir>/<arch>` ディレクトリー下にライブラリーと必要なスクリプトがインストールされます。例えば、インテル® 64 アーキテクチャー向けの `<arch>` は、`bin64` です。

注意

インテル® 64 アーキテクチャー上ですべての機能を使用するには、`<installdir>/bin64` を環境変数に設定します。提供されるスクリプトを使用して、環境変数を簡単に設定することができます。対象の開発環境向けに適切な値を設定するには、`source <installdir>/bin[64]/mpivars.[c]sh` を実行します。

インストールに関する完全な説明は、『インテル® MPI ライブラリー for Linux* インストール・ガイド』をご覧ください。ドキュメントには、サイレントモードでの製品のインストールと、役立つインストーラーのオプションが紹介されています。

3.2. インテル® MPI ライブラリー・ランタイム環境 (RTO) とインテル® MPI ライブラリー・ソフトウェア開発キット (SDK) のライセンス

インテル® MPI ライブラリーには、2 つのライセンスがあります。

- インテル® MPI ライブラリー・ランタイム環境 (RTO) ライセンス。このライセンスは、ランタイム・プロセス管理、ユーティリティーのサポート、共有ライブラリー (.so) およびドキュメントを含む MPI プログラムの実行に必要なツールに適用されます。ライセンスには、インテル® MPI ライブラリー・ベースのアプリケーションを実行するために必要なすべてが含まれており、無料で永続的に利用できます。
- インテル® MPI ライブラリー・ソフトウェア開発キット (SDK) ライセンス。このライセンスには、コンパイルツールとともにランタイム環境のすべてのコンポーネントが含まれます: コンパイラー・コマンド (`mpiicc`、`mpicc` など)、ファイルとモジュール、スタティック・ライブラリー (.a)、デバッグ・ライブラリー、トレース・ライブラリー、およびテスト用ソース。このライセンスは有料で、ソフトウェア利用許諾契約 (EULA) に記載されているようにいくつかの種類があります。

ライセンスの詳細は、EULA またはインテル® MPI ライブラリー製品ウェブサイトをご覧ください。

4. コンパイルとリンク

このセクションでは、インテル® MPI ライブラリーを使用したアプリケーションのコンパイルとリンクの手順を説明し、異なるデバッガーとコンパイラーのサポートに関する詳細を説明します。次の用語がこのセクションで使用されています。

- MPI プログラムのコンパイル
- デバッグ情報の追加
- その他のコンパイラー・サポート

4.1. MPI プログラムのコンパイル

ここでは、インテル® MPI ライブラリー開発キットのみを使用して MPI プログラムのコンパイルとリンクを行う際に必要な手順を説明します。インテル® MPI ライブラリーを使用して MPI プログラムをコンパイルおよびリンクするには、次の手順に従ってください。

1. 適切な環境設定を行うには、適切なスクリプトを `source mpivars.[c]sh` します (以下の例では Bash* シェルを使用)。
   `$ .<installdir>/bin64/mpivars.sh`
2. 適切な `mpixxx` コンパイラー・コマンドを使用して MPI プログラムをコンパイルします。例えば、C 言語で記述されたプログラムをコンパイルするには、次のコマンドを使用します。
   `$ mpiicc <installdir>/test/test.c -o testc`

カレント・ディレクトリーに実行形式ファイル `testc` が生成されます。アプリケーションを起動する方法の詳しい説明は、このドキュメントの「アプリケーションの実行」をご覧ください。

注意

デフォルトでは、実行形式ファイル `testc` は、最適化されたマルチスレッド版のインテル® MPI ライブラリーにリンクされます。ほかの設定を使用するには、「インテル® MPI ライブラリーの構成」をご覧ください。

その他のコンパイラーでも、通常のコンパイラー・コマンドに `mpi` のプリフィクスの付いた同等のスクリプトが提供されます。サポートされるコンパイラーの詳細については、『インテル® MPI ライブラリー for Linux* リファレンス・マニュアル』をご覧ください。

4.2. デバッグ情報の追加

アプリケーションをデバッグする場合、`-g` オプションを追加します。これにより、バイナリーにデバッグ情報が追加されます。アプリケーションのデバッグには、任意のデバッガーを利用できます。

```
$ mpiicc –g test.c -o testc -g
```

4.3. その他のコンパイラー・サポート

インテル® MPI ライブラリーは、異なるオペレーティング・システム (異なる `glibc*` のバージョン) と異なるコンパイラーをサポートするためバインディング・ライブラリーを提供します。これらのライブラリーには、C++、F77、および F90 インターフェイスが用意されています。

GNU* とインテル® コンパイラーのバインディング・ライブラリー:

- `libmpicxx.{a|so}` – g++ >= 3.4 向け
- `libmpifort.{a|so}` – GNU* とインテル® コンパイル向けの g77/gfortran インターフェイス

`mpicc`、`mpicxx`、`mpifc`、`mpif77`、`mpif90`、`mpigcc`、`mpigxx`、`mpiicc`、`mpiicpcor`、および `mpiifort` コンパイラー・コマンドを使用する場合、アプリケーションは適切な GNU* およびインテル® コンパイラーのバインディング・ライブラリーにリンクされます。

サードパーティーのコンパイラー向けには、インテル® MPI ライブラリー for Linux* に特定のコンパイラーのサポートを追加するバインディング・キットが用意されています。

注意

インテル® MPI ライブラリーは、インテル® コンパイラーだけではく GNU* コンパイラーをサポートしています。このコマンドに関しては、『インテル® MPI ライブラリー・リリースノート』をご覧ください。

さらに、インテル® MPI ライブラリーは、PGI* C、PGI* Fortran、および Absoft* Fortran77 コンパイラー向けのソースキットをバンドルしています。使用にあたっては、以下の注意が必要です。

- PGI* コンパイラーでコンパイルするソースでは、`long double` の実体を転送してはいけません。
- Absoft* ベースのビルドでは、`-g77` と `-B108` コンパイラー・オプションを指定する必要があります。
- 適切なコンパイラーをインストールして選択してください。
- それぞれのコンパイラーのランタイムが、各ノードに正しくインストールされていることを確認する必要があります。

PGI* C++、PGI* Fortran 95、および Absoft* Fortran 95 および GNU* Fortran 95 バージョン 4.0 以降とのバインドをサポートする場合、インテル® MPI ライブラリーのバインディング・ライブラリーを別途ビルドする必要があります。

バインディング・キットでは、必要なすべてのソースファイル、便利なスクリプト、および手順書が提供されます。

バインディング・キットと詳しい説明は、インストール先の `binding` ディレクトリーに格納されています。特定のツールに対するバインディングをサポートするには、インテル® プレミアサポートサイトから、インテル® MPI ライブラリー for Linux* 製品への要望をお送りください。

5. アプリケーションの実行

アプリケーションのコンパイルとリンクが完了したら、インテル® MPI アプリケーションを実行する準備ができました。ここでは、アプリケーションを実行する手順を説明します。

5.1. インテル® MPI プログラムの実行

MPI プログラムを実行する最も簡単な方法は、mpirun コマンドを使用することです。

```
$ mpirun -n <プロセス数> ./myprog
```

このコマンドは、デフォルトで Hydra プロセス管理を使用するため mpiexec.hydra コマンドを起動します。mpirun のコマンドラインで mpiexec.hydra へのオプションを指定します。

注意

mpirun と mpiexec.hydraare コマンドは互換性があります。

次の理由から、MPI プログラムを実行する際には mpirun コマンドの使用を推奨します。

1. mpirun コマンドで mpiexec.hydra のすべてのオプションを指定できます。
2. PBS Pro* や LSF* などのスケジューラーを使用して割り当てられたセッション内から MPI ジョブが送出された場合、mpirun コマンドはそれを検出します。そのため、MPI プログラムがバッチ・スケジューラーやジョブ管理下で実行されている場合、mpirun コマンドを使用することが推奨されます。

MPI プロセス数を設定するには、-n オプションを使用します。-n オプションが指定されていない場合、プロセス管理はジョブ・スケジューラーのホストリストを参照するか、マシン上のコア数をプロセス数として使用します。

デフォルトでは、ノード間の通信に ssh プロトコルが使用されます。rsh を使用する場合、-r オプションを指定します。

```
$ mpirun -r rsh -n <プロセス数> ./myprog
```

正しく実行するため、すべてのノードでパスワードなしの ssh 接続を設定します。ファブリック制御に関する詳しい情報は、『インテル® MPI ライブラリー for Linux* リファレンス・マニュアル』をご覧ください。

インテル® MPI ライブラリーを使用したアプリケーションが正しく実行できたら、そのアプリケーションは再リンクすることなくノード間で異なるファブリックを使用するほかのクラスターへ移行できます。問題が生じた場合、「デバッグとテスト」で解決策を探してください。

5.2. インテル® MPI ライブラリーの設定

インテル® MPI ライブラリーを設定するには、mpivars.c[c]sh スクリプトに適切な引数を指定して source します。次に例を示します。

```
$ .<installdir>/bin64/mpivars.sh release
```

このスクリプトの引数として以下が指定できます。デフォルトで、マルチスレッド版の最適化されたインテル® MPI ライブラリーが使用されます。

引数	定義
release	シングルスレッド版の最適化されたインテル® MPI ライブラリーを使用するにはこの引数を指定します。
debug	シングルスレッド版のデバッグ向けインテル® MPI ライブラリーを使用するにはこの引数を指定します。
release_mt	マルチスレッド版の最適化されたインテル® MPI ライブラリーを使用するにはこの引数を指定します。
debug_mt	マルチスレッド版のデバッグ向けインテル® MPI ライブラリーを使用するにはこの引数を指定します。

注意

インテル® MPI ライブラリーを異なる設定で使用する場合、アプリケーションを起動する前に対象の引数を指定して `mpivars.[c]sh` スクリプトを実行します。アプリケーションを再コンパイルする必要はありません。

5.3. マルチスレッド・アプリケーション

OpenMP* アプリケーションを実行し、ドメイン内でスレッドを固定するには、`KMP_AFFINITY` 環境変数が、対応する OpenMP* の機能を使用するように設定されている必要があります。

アプリケーションを実行する:

```
$ mpirun –genv OMP_NUM_THREADS 4 -n <プロセス数> ./myprog
```

OpenMP* との相互利用に関する詳しい情報は、『インテル® MPI ライブラリー for Linux* リファレンス・マニュアル』をご覧ください。

5.4. ファブリックの選択

インテル® MPI ライブラリーは、デフォルトで `I_MPI_FABRICS_LIST` に指定されるファブリックのリストをベースにネットワーク・ファブリックを選択します。特定のファブリックの組み合わせを選択するには、`-genv` オプションを使用して `I_MPI_FABRICS` 環境変数に割り当てます。また、`set` コマンドを使用して値を割り当てることもできます。

指定されたファブリックが利用できない場合、インテル® MPI ライブラリーは、`I_MPI_FABRICS_LIST` に指定されるリストから次に利用可能なファブリックを選択します。

このフォールバックの振る舞いは、`I_MPI_FALLBACK` 変数を使用して無効にできます。

```
$ export I_MPI_FALLBACK=0
```

デフォルトでは、フォールバックは有効です。`I_MPI_FABRICS` が設定されると、フォールバックは無効になります。

5.4.1. TCP ソケット接続

クラスターで利用可能なイーサーネット接続を使用して TCP ソケットを介して MPI プログラムを実行するには、次のコマンドを使用します。プログラムは、ノード内で共有メモリーを使用しません。

```
$ mpirun -genv I_MPI_FABRICS tcp -n <プロセス数> ./myprog
```

5.4.2. 共有メモリー

共有メモリーファブリック (shm) のみを介して MPI プログラムを実行するには、次のコマンドを使用します。

```
$ mpirun -genv I_MPI_FABRICS shm -n <プロセス数> ./myprog
```

5.4.3. 共有メモリーと DAPL* 接続

ノード内の通信に共有メモリーを使用し、ノード間の通信に Direct Access Programming Library* (DAPL*) を使用するには、次のコマンドを使用します。

```
$ mpirun -genv I_MPI_FABRICS shm:dapl -n <プロセス数> ./myprog
```

DAPL User Datagrams* (DAPL UD*) プロトコルによる接続を有効にするには、I_MPI_DAPL_UD 環境変数を設定します。

```
$ export I_MPI_DAPL_UD=enable
```

```
$ mpirun -genv I_MPI_FABRICS shm:dapl -n <プロセス数> ./myprog
```

これは、ファブリック・オプションが指定されていない場合のデフォルトです。

5.4.4. 共有メモリーと TMI*

ノード内の通信に共有メモリーを使用し、ノード間の通信に Tag Matching Interface* (TMI*) を使用するには、次のコマンドを入力します (ldd コマンドの検索パスに libtml.so ライブラリーがあることを確認してください)。

```
$ mpirun -genv I_MPI_FABRICS shm:tmi -n <プロセス数> ./myprog
```

これは、インテル® Omni Scale ファブリック (インテル® True Scale) や Myricom* MX インターフェイスを使用する際に推奨される方法です。

5.4.5. 共有メモリーと OFA*

ノード内通信に共有メモリーを使用し、ノード間通信に OpenFabrics* Enterprise Distribution (OFED*) verbs を使用するには、次のコマンドを使用します。

```
$ mpirun -genv I_MPI_FABRICS shm:ofa -n <プロセス数> ./myprog
```

これは、OpenFabrics* Enterprise Distribution (OFED) ソフトウェア・スタックを使用する際に推奨される方法です。

5.4.6. 共有メモリーと OFI*

ノード内通信に共有メモリーを使用し、ノード間通信に OpenFabrics Interface* (OFI*) を使用するには次のコマンドを使用します。

```
$ mpirun -genv I_MPI_FABRICS shm:ofi -n <プロセス数> ./myprog
```

これは、OpenFabrics Interface* (OFI*) ソフトウェア・スタックを使用する際に推奨される方法です。

5.4.7. マルチレール機能を使用

クラスターシステムが複数の通信カードや複数ポートのカードを装備している場合、次のコマンドを使用して通信帯域を向上させることができます。

```
$ export I_MPI_FABRICS=shm:ofa
```

```
$ export I_MPI_OFA_NUM_ADAPTERS=<num>
```

<num> には通信アダプターの数を指定します (デフォルトは 1)。

通信カードが複数のポートをサポートする場合、次の環境変数を使用してポート数を指定できます。

```
$ export I_MPI_OFA_NUM_PORTS=<num>
```

ファブリック制御に関する詳しい情報は、『インテル® MPI ライブラリー for Linux* リファレンス・マニュアル』をご覧ください。

インテル® MPI ライブラリーを使用して任意のファブリックでアプリケーションが正しく実行できたら、そのアプリケーションは再リンクすることなくノード間で異なるファブリックを使用するほかのクラスターへ移行できます。問題が生じた場合、「デバッグとテスト」で解決策を探してください。

また、PBS Pro* や LSF* などのリソース管理を利用する場合、mpirun を使用することを推奨します。

例えば、PBS 環境下でアプリケーションを実行するには、次の手順に従ってください。

1. 必要なノード数とインテル® MPI ライブラリーの環境を指定する、PBS 起動スクリプトを作成します。例えば、次の内容で pbs_run.sh ファイルを作成します。

   ```
   #PBS -l nodes=2:ppn=1
   #PBS -l walltime=1:30:00
   #PBS -q workq
   #PBS –V
   # Set Intel MPI environment
   mpi_dir=<installdir>/<arch>/bin
   cd $PBS_O_WORKDIR
   source $mpi_dir/mpivars.sh
   # Launch application
   mpirun -n <プロセス数> ./myprog
   ```

2. PBS の qsub コマンドを使用してジョブを送信します。

   ```
   $ qsub pbs_run.sh
   ```

ジョブ・スケジューラーから mpirun を起動する場合、ノード数を指定する必要はありません。インテル® MPI ライブラリーは、Hydra プロセス管理を介して自動的に利用可能なノード数を検出します。

5.4.8. I_MPI_FABRICS

このトピックは、『インテル® MPI ライブラリー for Linux* リファレンス・マニュアル』からの抜粋であり、I_MPI_FABRICS 環境変数について詳しく説明します。

通信に使用する特定のファブリックを選択します。

構文

```
I_MPI_FABRICS=<fabric>|<intra-node fabric>:<inter-nodes fabric>
```

ここで

- <fabric> := {shm, dapl, tcp, tmi, ofa, ofi}
- <intra-node fabric> := {shm, dapl, tcp, tmi, ofa, ofi}
- <inter-nodes fabric> := {dapl, tcp, tmi, ofa, ofi}

引数

引数	定義
<*fabric*>	ネットワーク・ファブリックを定義
shm	共有メモリー
dapl	InfiniBand*、iWarp*、Dolphin* や XPMEM* (DAPL* を介して) などの DAPL ケーブル・ネットワーク・ファブリック
tcp	イーサーネットや InfiniBand* (IPoIB* を介して) のような TCP/IP ケーブル・ネットワーク・ファブリック
tmi	Tag Matching Interface TMI を介したタグ一致機能を備えたネットワーク・ファブリック (インテル® True Scale ファブリックや Myrinet*)
ofa	Open Fabrics Alliance* (OFA*) によって提供される InfiniBand* (through OpenFabrics* Enterprise Distribution (OFED*) verbs) などのファブリック
ofi	インテル® True Scale Fabric と TCP (OFI* API を介した) を含む OFI (OpenFabrics Interfaces*) ケーブル・ネットワーク・ファブリック

例えば、winOFED* InfiniBand* デバイスを選択するには、次のコマンドを使用します。

```
>mpirun -n <プロセス数>   \ -env I_MPI_FABRICS shm:dapl <実行形式>
```

これらのデバイスは、<*provider*> が指定されていない場合、/etc/dat.conf ファイルの最初の DAPL* プロバイダーが使用されます。shm ファブリックは、インテル製マイクロプロセッサーおよび互換マイクロプロセッサーで利用可能ですが、インテル製マイクロプロセッサーにおいてより多くの最適化が行われる場合があります。

注意

選択されているファブリックが利用可能であることを確認してください。例えば、すべてのプロセスが共有メモリーを介してのみ通信可能な場合、/dev/shm デバイスを使用します。すべてのプロセスが単一の DAPL プロバイダーを介してのみ通信可能な場合、dapl を使用します。

---

6. デバッグとテスト

ここでは、異なるデバッグツールによる MPI アプリケーションのデバッグ方法について説明します。

6.1. GDB: GNU* プロジェクト・デバッガー

インテル® MPI ライブラリーから GDB デバッガーを起動するには、次のコマンドを使用します。

```
$ mpirun -gdb -n 4 ./testc
```

シングルプロセスのアプリケーションをデバッグするように DGB デバッガーを利用できます。並列プログラムのデバッグに関する詳細は、http://www.gnu.org/software/gdb/ (英語) にある GDB のドキュメントをご覧ください。

実行中のジョブをアタッチするには次のコマンドを使用します。

```
$ mpirun –n 4 -gdba <pid>
```

<pid> には、実行中の MPI ランクのプロセス ID を指定します。

6.2. TotalView* デバッガー

インテル® MPI ライブラリーは、Rogue Wave* Software, Inc の TotalView* デバッガーをサポートします。MPI プログラムをデバッグするには、mpirun の引数に -tv オプションを追加します。

```
$ mpirun -tv -n 4 ./testc
```

注意

ssh で通信を行う場合、TVDSVRLAUNCHCMD 環境変数を ssh に設定する必要があります。

ジョブを停止状態で開始するかどうか確認する TotalView からのポップアップ・ウィンドウが表示されます。停止状態で開始すると、TotalView のウィンドウが表示され、ソースウィンドウにアセンブリー・コードが表示されます。main 関数のソースを表示するには、スタックウィンドウ (左上) の main( ) をクリックします。TotalView は、プログラム (すべてのプロセス) が MPI_INIT( ) 呼び出しで停止したことを通知します。上記の手順で TotalView によるデバッグを行った場合、インテル® MPI ライブラリーのジョブを再起動する前に TotalView を終了する必要があります。

TotalView でデバッグするためセッションを再起動するには、次のコマンドを使用します。

```
$ totalview python -a ‘which mpirun‘ -tvsu <mpirun_args> <prog> <prog_args>
```

TotalView 8.1.0 以降を使用する場合、間接起動の機能を利用できます。

1. TotalView を次のように開始します。
   ```
   $ totalview <prog> -a <prog_args>
   ```
2. **[Process/Startup Parameters]** コマンドを選択します。
3. 結果ダイアログ・ボックスの **[Parallel]** タブを選択し、並列システムとして **[MPICH2]** を選択します。
4. **[Tasks]** フィールドで使用するタスク数を設定します。
5. **[Additional Starter Arguments]** フィールドに、mpirun に渡すその他の引数を入力します。

TotalView に実行中のジョブをアタッチする場合、ジョブを開始する際に mpirun コマンドに -tvsu オプションを加える必要があります。このオプションを使用すると、MPI_Init( ) 内部にバリアが追加され、スタートアップのパフォーマンスに影響する可能性があります。MPI_Init( ) からすべてのタスクがリターンすると、その後このオプションによるパフォーマンスの低下はありません。

6.3. DDT* デバッガー

DDT* デバッガーを使用して MPI アプリケーションをデバッグすることができます。インテルは、このデバッガーに対するサポートを提供していません。開発者は、Allinea* 社にコンタクトする必要があります。DDT* のユーザーガイドは、http://www.allinea.com/products/ddt-support/ (英語) から入手できます。

実行前に TotalView* の変数を設定し、-tv オプションを指定して DDT* を実行できます。

```
$ export TOTALVIEW=DDT_INSTALLATION_PATH/bin/ddt-debugger-mps
```

```
$ mpirun -np 4 -tv ./your_app
```

DDT デバッガーで問題が生じた場合、DDT のドキュメントを参照してください。

6.4. ログの取得

時には、アプリケーションのデバッグよりも、ログを利用することが有効なことがあります。実行中のアプリケーションのログを取得するには、いくつかの方法があります。

6.4.1. デバッグ情報の取得

環境変数 I_MPI_DEBUG は、実行中の MPI アプリケーションの情報を取得する非常に便利な機能を提供します。この変数には、0 (デフォルト) から 1000 の値を設定できます。値が大きくなるほど詳しいデバッグ情報を取得できます。

```
$ mpirun -genv I_MPI_DEBUG 5 -n 8 ./my_application
```

注意

I_MPI_DEBUG に大きな値を設定すると、多くの情報を取得できますが、アプリケーションの実行パフォーマンスを大幅に低下させます。始めに一般的なエラーを検出するには、I_MPI_DEBUG=5 で開始することを推奨します。I_MPI_DEBUG に関する詳しい情報は、『インテル® MPI ライブラリー for Linux* リファレンス・マニュアル』をご覧ください。

6.4.2. アプリケーションのトレース

-t または -trace オプションを使用してインテル® Trace Collector ライブラリーとのリンクを行います。これは、mpiicc やほかのコンパイラー・スクリプトに -profile=vt オプションを指定するのと同じ効果があります。

```
$ mpiicc -trace test.c -o testc
```

このオプションを使用するには、次の操作を行う必要があります。

- インテル® Trace Analyzer & Collector を最初にインストールします。このツールは、インテル® Parallel Studio XE Cluster Edition スイートの一部としてのみ提供されます。
- VT_ROOT 環境変数に、インテル® Trace Collector のインストール先のパスを含めます。ほかのプロファイル・ライブラリーを指定するには、I_MPI_TRACE_PROFILE 環境変数に <プロファイル名> を設定します。例えば、I_MPI_TRACE_PROFILE に vtfs を設定すると、フェイルセーフ版のインテル® Trace Collector とリンクを行います。

6.4.3. 正当性のチェック

-check_mpi オプションを使用してインテル® Trace Collector の正当性チェック・ライブラリーとのリンクを行います。これは、mpiicc やほかのコンパイラー・スクリプトに -profile=vtmc オプションを指定するのと同じ効果があります。

```
$ mpiicc -profile=vtmc test.c -o testc
```

もしくは

```
$ mpiicc -check_mpi test.c -o testc
```

このオプションを使用するには、次の操作を行う必要があります。

- インテル® Trace Analyzer & Collector を最初にインストールします。このツールは、インテル® Parallel Studio XE Cluster Edition スイートの一部としてのみ提供されます。
- VT_ROOT 環境変数に、インテル® Trace Collector のインストール先のパスを含めます。ほかのプロファイル・ライブラリーを指定するには、I_MPI_CHECK_PROFILE 環境変数に <プロファイル名> を設定します。

インテル® Trace Analyzer & Collector の詳しい情報は、製品に含まれるドキュメントをご覧ください。

6.4.4. 統計収集

アプリケーションが使用する MPI 関数の統計情報を収集するには、I_MPI_STATS 環境変数に 1 から 10 の値を設定します。この環境変数は、収集する統計情報の量を制御し、ログをファイルに出力します。デフォルトでは、統計情報は収取されません。

統計収集に関する詳しい情報は、『インテル® MPI ライブラリー for Linux* リファレンス・マニュアル』をご覧ください。

6.5. インストールのテスト

インテル® MPI ライブラリーがインストールされ正しく機能していることを確認するには、テストプログラムのコンパイルと実行に加え、次の手順で一般的なテストを行います。

インストールをテストする (クラスターの各ノードで):

1. PATH に <installdir>/<arch>/bin が設定されていることを確認します。
   ```
   $ ssh <nodename>which mpirun
   ```
   各ノードでパスが正しいことを確認してください。

   (SDK のみ) インテル® Parallel Studio XE Composer Edition パッケージを使用している場合、PATH と LD_LIBRARY_PATH 環境変数に適切なディレクトリーが設定されていることを確認してください。
   ```
   $ mpirun -n <# of processes> env | grep PATH
   ```
   各ノードでパスが正しいことを確認してください。正しくない場合、compilervars.sh [csh] スクリプトを呼び出します。例えば、インテル® Parallel Studio XE 2015 Composer Edition 向けには次の source コマンドを使用します。
   ```
   $ ./opt/intel/composer_xe_2015/bin/compilervars.sh intel64
   ```
2. 状況によりますが、LD_LIBRARY_PATH に <installdir>/<arch>/lib を含める必要がある場合があります。LD_LIBRARY_PATH の設定を確認するには次のコマンドを使用します。
   ```
   $ mpirun -n <プロセス数> env | grep LD_LIBRARY_PATH
   ```

6.5.1. テストプログラムのコンパイルと実行

テストプログラムをコンパイルするには、次の手順に従ってください。

1. **(SDK のみ)** 製品に含まれるテストプログラムのいずれかをコンパイルします。
   ```
   $ cd /opt/intel/samples_2016/en/mpi
   $ mpiicc -o myprog test.c
   ```
2. InfiniBand*、Myrinet* またはその他の RDMA-ケーブル・ネットワーク・ハードウェアとソフトウェアを使用している場合、それぞれのネットワークのテスト機能を使用してすべてが機能していることを確認してください。

3. クラスターのすべての利用可能な構成でテストプログラムを実行します。

- TCP/IP ネットワーク・ファブリックをテストするには、次のコマンドを使用します。

  ```
  $ mpirun -n 2 -genv I_MPI_DEBUG 2 -genv I_MPI_FABRICS tcp ./myprog
  ```

  各ランクからの出力される情報だけでなく、TCP/IP ネットワーク・ファブリックを使用することを示す出力をデバッグします。

- DAPL ネットワーク・ファブリックをテストするには、次のコマンドを使用します。

  ```
  $ mpirun -n 2 -genv I_MPI_DEBUG 2 -genv I_MPI_FABRICS shm:dapl ./myprog
  ```

  各ランクからの出力される情報だけでなく、共有メモリーと DAPL 対応ネットワーク・ファブリックが使用されていることを表示する出力をデバッグします。

- 次のコマンドでほかのファブリックをテストします。

  ```
  $ mpirun -n 2 -genv I_MPI_DEBUG 2 -genv I_MPI_FABRICS <fabric> ./myprog
  ```

  <fabric> は、サポートされているファブリックを指定します。詳細は、ファブリックの選択を参照してください。

mpirun コマンドを使用すると、ランクごとに 1 行のメッセージと使用しているファブリックのデバッグ情報が表示されるはずです。ファブリックは、I_MPI_FABRICS の設定と一致している必要があります。

インテル® MPI ライブラリー開発者キットの /opt/intel/samples_2016/en/mpi ディレクトリーには、test.c に加えほかのテストプログラムが含まれます。

7. プロセス管理

ここでは、インテル® MPI ライブラリーに含まれるプロセス管理について説明します。

7.1. プロセス管理の選択

`mpirun` スクリプトは、`I_MPI_PROCESS_MANAGER` 環境変数に指定されたプロセス管理を使用します。デフォルトでは、`mpirun` は Hydra プロセス管理を使用します。`I_MPI_PROCESS_MANAGER` に `hydra` を設定することで明示的に Hydra を選択し、`mpd` にすることで明示的に MPD を選択します。また、プロセス管理は直接 `mpiexecfile` を呼び出すことで選択できます: `mpiexec.hydra` (Hydra) または `mpiexec` (MPD)。

7.2. スケーラブルなプロセス管理システム (Hydra)

Hydra は、シンプルでスケーラブルなプロセス管理です。Hydra は、既存のリソース管理がプロセスを実行する場所を決め、各ホストのプロキシーを使用してターゲット間に分散するのをチェックします。プロキシーは、プロセスの起動、クリーンアップ、I/O 転送、信号転送、およびほかのタスクで使用されます。

`mpiexec.hydra` を使用して Hydra を起動できます。スケーラブル・プロセス管理システム (Hydra) コマンドの詳細は、『インテル® MPI ライブラリー・リファレンス・マニュアル』をご覧ください。

注意

マルチプロセス・デーモン (MPD) は、インテル® MPI ライブラリー 5.0 リリースでは使用されなくなりました。並列ジョブを開始するには、スケーラブル・プロセス管理システム (Hydra) を使用します。

7.3. 多目的デーモン (MPD*)

MPD は、MultiPurpose Daemon の略です。これは、すべてのノード上で並列ジョブを開始するために必要なインテル® MPI ライブラリーのプロセス管理システムです。MPD は、システムとハードウェアに関する情報だけでなく、必要な情報をやり取りするため相互に通信します。例えば、MPD リングは、MPD プロセス管理下で正しいピニングを行うため必要となります。

注意

マルチプロセス・デーモン (MPD) は、インテル® MPI ライブラリー 5.0 リリースでは使用されなくなりました。並列ジョブを起動する代わりに、スケーラブルなプロセス管理システム (Hydra) を使用します。

7.4. MPI プロセスの配置を制御する

`mpirun` コマンドは、プロセスのランクがクラスターのノードにどのように割り当てられるかを制御します。デフォルトでは、`mpirun` コマンドはグループ・ラウンドロビン割り当てにより、ノードのすべてのプロセッサー・ランクに連続する MPI プロセスを投入します。この配置アルゴリズムは、対称型マルチプロセッサー (SMP) ノードを持つクラスターでは、アプリケーションにとって最良の選択ではないかもしれません。

各ノードに割り当てられる隣接するランクのペアのジオメトリーは、<ランク数> = 4 および <ノード数> = 2 と仮定します (例えば、2 ウェイ SMP ノードの場合)。クラスターノードを参照するには、次のコマンドを入力します。

```
cat ~/mpd.hosts
```

結果は以下のようになります。

```
clusternode1
clusternode2
```

アプリケーションの 4 つのプロセスを 2 ウェイ SMP クラスターに均等に配置するには、次のコマンドを入力します。

```
mpirun –perhost 2 –n 4 ./myprog.exe
```

myprog.exe の実行結果は、次のように出力されます。

```
Hello world: rank 0 of 4 running on clusternode1
Hello world: rank 1 of 4 running on clusternode1
Hello world: rank 2 of 4 running on clusternode2
Hello world: rank 3 of 4 running on clusternode2
```

別の方法として、引数を使用して各ホスト上で実行されるプロセス数を明示的に設定することができます。一般的な例として、アプリケーション内でマスター・ワーカー・モデルを用いる場合があります。例えば、次のコマンドは、clusternode1 と clusternode2 に 4 つのプロセスを均等に分散します。

```
mpirun –n 2 –host clusternode1 ./myprog.exe :-n 2 –host clusternode2
```

```
./myprog.exe
```

関連情報

ローカルのオプションに関する詳細は、『インテル® MPI ライブラリー・リファレンス・マニュアル』をご覧ください。

MPI プロセスの配置の制御に関する詳しい情報は、オンラインの「インテル® MPI ライブラリーのプロセス配置制御」(英語)もご覧ください。

8. mpitune ユーティリティーによるチューニング

この章では、インテル® MPI ライブラリー向けの最適な設定を見つけるため mpitune ユーティリティーを使用する方法を説明します。

- クラスター固有のチューニング
- アプリケーション固有のチューニング
- 時間制限の設定
- ファブリック・リストの設定
- プロセス数の範囲を設定
- ホストを使用する制限を設定
- 最後に保存したセッションから mpitune をリストア
- 手動でアプリケーションをチューニング

8.1. クラスター固有のチューニング

インテル® MPI ライブラリーには 100 個以上のパラメーターがあります。デフォルトは、一般的な利用向けの設定が行われ、ほとんどのクラスターとアプリケーションで優れたパフォーマンスを提供します。しかし、より高いパフォーマンスを求める場合、mpitune ユーティリティーを利用できます。このユーティリティーは、ベンチマーク・プログラムとしてインテル® MPI Benchmarks (IMB) を使用し、異なるパラメーターでテストを数回実行してクラスターシステムに最適な設定を探します。mpitune ユーティリティーは次のコマンドで起動します。

```
$ mpitune
```

その後、チューニングされた設定を有効にするには、アプリケーション起動時に -tune オプションを指定します。

```
$ mpirun -tune -perhost 8 –n 64 ./your_app
```

最良の結果を得るには、チューニングされたパラメーターのデフォルトの格納先である <installdir>/<arch>/etc への書き込み権限を持つアカウントで mpitune を実行します。このディレクトリーへの書き込み権限が無い場合、カレント・ディレクトリーに新しい設定ファイルが保存されます。

デフォルトで、mpitune はベンチマークとしてインテル® MPI Benchmarks (IMB) を使用します。また、次のようなコマンドでベンチマークを置き換えることができます。

```
$ mpitune -test \”your_benchmark –param1 –param2\”
```

新しい設定を適用するには、クラスター固有のチューニングをご覧ください。

インテル® MPI Benchmarks の実行可能ファイルは、デフォルトで非インテル互換プロセッサーよりもインテル® マイクロプロセッサーに最適化されています。そのため、インテル® マイクロプロセッサーと非インテル互換プロセッサーでは、チューニングの設定が異なることがあります。

8.2. アプリケーション固有のチューニング

アプリケーション固有の最適化設定を見つけるには、mpitune ユーティリティーを使用します。

```
$ mpitune --application \”your_app\” –-output-file yourapp.conf
```

“your_app” には、実際に起動するアプリケーションを指定します。次に例を示します。

```
“mpitune –-application \"./my_test\" –-output-file $PWD/my_test.conf
```

チューニングされた設定は yourapp.conf ファイルに保存されます。それを適用するには、mpirun を次のように起動します。

```
$ mpirun -tune $PWD/yourapp.conf -perhost 8 –n 64 ./your_app
```

“your_app” には、実行可能ファイルだけでなく異なるプロセスで開始するスクリプトを指定することもできます。

注意

スクリプトでは、I_MPI_* 環境変数を変更してはいけません。

8.3. 時間制限の設定

チューニングの過程には時間がかかります。各クラスターとアプリケーションの設定にはさまざまな要因があるため、チューニングの時間は予測できないことがあります。

チューニングの時間を制限するには、-time-limit オプションを使用します。例えば、チューニングを 8 時間 (480 分) に制限するには、次のコマンドを実行します。

```
$ mpitune --time-limit 480
```

時間は分単位で指定します。

8.4. ファブリック・リストの設定

テストするファブリックを定義するには、-fabrics-list オプションを使用します。

```
$ mpitune --fabrics-list shm,ofa,dapl
```

利用可能なファブリックには以下があります: shm:dapl、shm:tcp、shm、dapl、shm:ofa、shm:tmi、ofa、tmi、tcp。

8.5. プロセス数の範囲を設定

1 つのノード上で実行するプロセスを制限するには、perhost-range min:max オプションを使用します。例えば、次のコマンドは各ノード上の MPI ランク数を 4 から 8 に設定します。

```
$ mpitune --perhost-range 4:8
```

8.6. ホストを使用する制限を設定

チューニングを実行するノードを制限するには、host-range min:max オプションを使用します。例えば、次のコマンドは 8 から 16 ノードでのみ実行するように制限します。

```
$ mpitune --host-range 8:16
```

8.7. 最後に保存したセッションから mpitune をリストア

mpitune を実行中に予測しないイベントが発生することがあります。このケースでは、mpituner_session_<session-id>.mts ファイルに保存された途中結果を使用します。最後に保存したセッションから mpitune を開始します。

```
$ mpitune --session-file ./mpituner_session_<session-id>.mts
```

<session-id> には、チューニングを開始するタイムスタンプを指定します。

8.8. 手動でアプリケーションをチューニング

一連の I_MPI_ADJUST_* 環境変数を使用して、インテル® MPI ライブラリーの集合操作を手動でチューニングすることができます。メッセージサイズの範囲を設定し、異なるアルゴリズムを選択するとアプリケーションのパフォーマンスを向上することができます。I_MPI_ADJUST に関する詳しい情報は、『インテル® MPI ライブラリー for Linux* リファレンス・マニュアル』をご覧ください。

9. サポートされるジョブ・スケジューラー

インテル® MPI ライブラリーは、HPC 市場で一般的に利用されているジョブ・スケジューラーの大部分をサポートしています。Linuxs* では、次のジョブ・スケジューラーがサポートされます。

- Altair* PBS Pro*
- Torque*
- OpenPBS*
- IBM* Platform LSF*
- Parallelnavi* NQS*
- SLURM*
- Univa* Grid Engine*

Linux* では、このサポートは `mpirun` ラッパースクリプトで実装されています。`mpirun` は、特定の環境変数をチェックして起動されたジョブ・スケジューラーを確認した後、アプリケーションを起動する適切な方法を選択します。

9.1. Altair* PBS Pro*、TORQUE*、および OpenPBS*

これらのいずれかのジョブ・スケジューラーを使用し、`$PBS_ENVIRONMENT` に `PBS_BATCH` もしくは `PBS_INTERACTIVE` が設定されている場合、`mpirun` は `mpirun` マシンファイルとして `$PBS_NODEFILE` を使用します。この場合、`-machinefile` オプションを直接指定する必要はありません。

バッチジョブ・スクリプトの例:

```
#PBS -l nodes=4:ppn=4
#PBS -q queue_name
cd $PBS_O_WORKDIR
mpirun -n 16 ./myprog.exe
```

9.2. IBM* Platform LSF*

IBM* Platform LSF* ジョブ・スケジューラーを使用し、`$LSB_MCPU_HOSTS` が設定されていると、並列ジョブ向けのホストリストを取得するため解析が行われます。`$LSB_MCPU_HOSTS` は、メインプロセス名を格納していません。そのため、ローカルホスト名がホストリスクのトップに追加されます。このホストリストに基づいて、`mpirun` 向けのマシンファイルが、一意の名前で生成されます。`/tmp/lsf_${username}.$$` ジョブが終了すると、マシンファイルは削除されます。

例:

```
$ bsub -n 16 mpirun -n 16 ./myprog.exe
```

9.3. Parallelnavi* NQS*

Parallelnavi NQS* ジョブ・スケジューラーを使用し、`$ENVIRONMENT`、`$QSUB_REQID`、`$QSUB_NODEINF` オプションが設定されると、`$QSUB_NODEINF` ファイルが `mpirun` のマシンファイルとして使用されます。また、起動時にプロセス管理によるリモートシェルとして `-r plesh` が使用されます。

9.4. SLURM*

$SLURM_JOBID が設定されると、mpurun 向けのマシンファイルを生成するため $SLURM_TASKS_PER_NODE と $SLURM_NODELIST 環境変数が使用されます。マシンファイル名は次のようになります。

/tmp/slurm_${username}.$$ ジョブが終了すると、マシンファイルは削除されます。

例:

```
$ srun -N2 --nodelist=host1,host2 –A
$ mpirun –n 2 myprog.exe
```

9.5. Univa* Grid Engine*

Univa* Grid Engine* ジョブ・スケジューラーを使用し、$PE_HOSTFILE が設定されると、2 つのファイルが生成されます: /tmp/sge_hostfile_${username}_$$ と /tmp/sge_machifile_${username}_$$。後者は、mpirun 向けのマシンファイルとして使用されます。ジョブが終了すると、これらのファイルは削除されます。

9.6. SIGINT と SIGTERM シグナルをインターセプト

ジョブに割り当てられたリソースの制限を超えた場合、ほとんどのジョブ・スケジューラーは、すべてのプロセスにシグナルを送信してジョブを強制終了します。

例えば、Torque* はジョブに SIGTERM を 3 回送信し、ジョブがまだアクティブであれば強制終了させるため SIGKILL を送信します。

Univa Grid Engine* では、ジョブを強制終了するデフォルトシグナルは SIGKILL です。インテル® MPI ライブラリーは、mpirun が引き起こすジョブ全体を強制終了するシグナルを処理またはキャッチできません。次のキュー設定により、強制終了シグナルを変更することができます。

- 利用可能なキューを参照するには、次のコマンドを使用します。
  ```
  $ qconf –sql
  ```
- キュー設定を変更するには、次のコマンドを使用します。
  ```
  $ qconf -mq <queue_name>
  ```
- 「terminate_method」を検索し、値を SIGTERM に変更します。
- キュー設定を保存します。

10. 一般的なクラスターに関する考察

この章では、MPI を利用する上で考慮すべき一般的なクラスターに関す情報を説明します。

- 使用するノードの定義
- パスワードなしの SSH 接続
- ヘテロジニアス・システムとジョブ

10.1. 使用するノードの定義

インテル® MPI ライブラリーは、デフォルトで `mpd.hosts` ファイルを検索します。このファイルには、アプリケーションが利用できるクラスター上のすべてのノードが含まれている必要があります。`mpd.hosts` ファイルの形式は、ノード名のリストを 1 行に 1 つ定義します。空白行と # に続く行は無視されます。

`-f` オプションを使用して、このファイルへのパスを指定できます。

ジョブ・スケジューラーの下で実行している場合、ホストはスケジューラーで決定されるため `-f` オプションは必要ありません。

10.2. パスワードなしの SSH 接続

プロセスがリモートで開始される場合、プロセスを起動するためデフォルトで `ssh` が使用されます。パスワードなしの SSH 機能を有効にしないと、ジョブを起動する際にパスワードが要求されます。インテル® MPI ライブラリーのインストール・パッケージで提供されるスクリプトは、SSH キーを自動生成して配布します。スクリクトの名称は `sshconnectivity.exp` で、`tar` 形式のファイルを展開した後にメインフォルダー下に配置されます。システム上でスクリプトが動作しない場合、次の手順でキーを生成および配布することができます。

1. パブリックキーの生成
   ```
   local> ssh-keygen -t dsa -f .ssh/id_dsa
   ```
   パスワードが共有されたら、空白のまま Enter キーを押します。

   `.ssh` ディレクトリーに `id_dsa` と `id_dsa.pub` という 2 つのファイルが作成されます。後者はパブリックキーです。
2. リモートノードへパブリックキーを配布します

   `.ssh` ディレクトリー移動します。リモートマシンへパブリックキーをコピーします

   ```
   local> cd .ssh
   local> scp id_dsa.pub user@remote:~/.ssh/id_dsa.pub
   ```
   リモートマシンにログインして、リモートマシンの .ssh ディレクトリーに移動します。

   ```
   local> ssh user@remote remote> cd .ssh
   ```
   クライアントのパブリックキーをリモートサーバー上のパブリックキーに追加します。

   ```
   remote> cat id_dsa.pub >> authorized_keys
   remote> chmod 640 authorized_keys
   remote> rm id_dsa.pub
   remote> exit
   ```

次回からリモートサーバーにログインする際、パスワードは必要ありません。

注意

ssh の設定はクライアントのディストリビューションに依存します。

10.3.ヘテロジニアス・システムとジョブ

すべてのクラスターは、ホモジニアス (同種) ではありません。すべてのジョブも、ホモジニアス (同種) ではありません。インテル® MPI ライブラリーは、複数のコマンドと引数を 1 つのコマンドで実行することができます。これには 2 つの方法があります。

最も簡単な方法は、設定ファイルを作成して `-configfile` オプションを使用することです。設定ファイルは、`mpirun` への引数を行ごとに 1 つのグループとして記述します。

```
-n 1 –host node1 ./io <io_args>
-n 4 –host node2 ./compute <compute_args_1>
-n 4 –host node3 ./compute <compute_args_2>
```

また、オプションのセットをコマンド行でグループごとに ':' で区切って渡すことができます。

```
mpirun –n 1 –host node1 ./io <io_args> :-n 4 –host node2 ./compute
<compute_args_1> :-n 4 –host node3 ./compute <compute_args_2>
```

プロセスが起動されると、作業ディレクトリーは、ジョブが起動されたマシンの作業ディレクトリーに設定されます。変更するには、`-wdir <パス>` オプションを使用します。

1 つのプロセスグループのみに環境変数を適用するには、`–env <変数> <値>` オプションを使用します。すべてのプロセスグループに環境変数を適用するには、`–genv` を使用します。デフォルトでは、すべての環境変数が起動時に実行環境から継承されます。

11. トラブルシューティング

このセクションでは、次のようなトラブルシューティングに関する情報を提供します。

- インテル® MPI ライブラリーの一般的なトラブルシューティングの手順
- 障害が発生した際の典型的な MPI の失敗に関する出力メッセージと対処法
- 潜在的な原因と解決策に関する提言

11.1.一般的なトラブルシューティングの手順

インテル® MPI ライブラリーの使用中にエラーや障害が発生した際の、一般的なトラブルシューティングの手順を以下に示します。

1. システム要件のセクションとインテル® MPI ライブラリーのリリースにある既知の問題をチェックします。
2. ホストが利用できるかチェックします。`mpirun` を使用して、ホスト・プラットフォーム上で簡単な非 MPI アプリケーション

   (例えば、`hostname` ユーティリティーなど) を実行します。

   例
   ```
   $ mpirun -ppn 1 -n 2 -hosts node01,node02 hostname
   node01
   node02
   ```

   これは、環境の問題 (MPI のリモート・アクセス・ツールが適切に設定されていないなど) や接続の問題 (ホストに到達できないなど) を明確にするのに役立ちます。
3. 環境変数 `I_MPI_DEBUG=6` および (または) `I_MPI_HYDRA_DEBUG=on` に設定し、デバッグ情報を有効にして MPI アプリケーションを実行します。より詳しいデバッグ情報を取得するには、設定する整数値の増やしてください。これは、問題のあるコンポーネントを特定するのに役立ちます。
4. 製品ページからインテル® MPI ライブラリーの最新バージョンをダウンロードしてインストールし、問題が解決しないか確認してください。
5. 問題が解決しない場合、インテル® プレミアサポート (英語) から問題を報告してください。

11.2.MPI の失敗の例

このセクションでは、エラーの説明、エラーメッセージ、および関連する推奨事項などを含む典型的な MPI のエラーの例を紹介します。ここでは、次のような MPI の障害について説明します。

- 通信の問題
- 環境の問題
- その他の問題

11.2.1. 通信の問題

インテル® MPI ライブラリーとの通信の問題は、通常強制終了 (`SIGTERM`、`SIGKILL` やその他のシグナル) によって引き起こされます。このような強制終了は、ホストのリブート、予期しないシグナルの受信、アウトオブメモリー (OOM) マネージャー・エラーなどに起因する可能性があります。

このような障害に対処するには、MPI プロセスの強制終了の原因を探ります (システムのログファイルを確認するなど)。

例 1

症状 / エラーメッセージ

```
[50:node02] unexpected disconnect completion event from [41:node01]
```

および / または:

```
===================================================================================
= BAD TERMINATION OF ONE OF YOUR APPLICATION PROCESSES
= PID 20066 RUNNING AT node01
= EXIT CODE:15
= CLEANING UP REMAINING PROCESSES
= YOU CAN IGNORE THE BELOW CLEANUP MESSAGES
===================================================================================
```

テーブル中にレポートされる実際のノードと MPI プロセスは、初期化の障害を反映しない可能性があります。

原因

node01 上の MPI プロセスの 1 つがシグナル (SIGTERM や SIGKILL など) で強制終了されました。MPI アプリケーションは dapl ファブリックを介して実行されました。

解決方法

MPI プロセスが終了した原因を特定してください。この強制終了は、ホストのリブート、予期しないシグナルの受信、アウトオブメモリー (OOM) マネージャー・エラーなどに起因する可能性があります。システムのログファイルを確認します。

例 2

症状 / エラーメッセージ

```
rank = 26, revents = 25, state = 8
Assertion failed in file ../../src/mpid/ch3/channels/nemesis/netmod/tcp/socksm.c
at line 2969:(it_plfd->revents & POLLERR) == 0
internal ABORT - process 25
Fatal error in PMPI_Alltoall:A process has failed, error stack:
PMPI_Alltoall(1062).......:MPI_Alltoall(sbuf=0x9dd7d0, scount=64, MPI_BYTE,
rbuf=0x9dc7b0, rcount=64, MPI_BYTE, comm=0x84000000) failed
MPIR_Alltoall_impl(860)...:
MPIR_Alltoall(819)........:
MPIR_Alltoall_intra(360)..:
dequeue_and_set_error(917):Communication error with rank 2rank = 45, revents = 25,
state = 8
Assertion failed in file ../../src/mpid/ch3/channels/nemesis/netmod/tcp/socksm.c
at line 2969:(it_plfd->revents & POLLERR) == 0
internal ABORT - process 84
...
Fatal error in PMPI_Alltoall:A process has failed, error stack:
PMPI_Alltoall(1062).......:MPI_Alltoall(sbuf=MPI_IN_PLACE, scount=-1,
MPI_DATATYPE_NULL, rbuf=0x2ba2922b4010, rcount=8192, MPI_INT, MPI_COMM_WORLD)
failed MPIR_Alltoall_impl(860)...:
MPIR_Alltoall(819)........:
MPIR_Alltoall_intra(265)..:
MPIC_Sendrecv_replace(658):
dequeue_and_set_error(917):Communication error with rank 84
...
```

および / または:

```
===================================================================================
= BAD TERMINATION OF ONE OF YOUR APPLICATION PROCESSES
= PID 21686 RUNNING AT node01
= EXIT CODE:15
= CLEANING UP REMAINING PROCESSES
= YOU CAN IGNORE THE BELOW CLEANUP MESSAGES
===================================================================================
```

テーブル中にレポートされる実際のノードと MPI プロセスは、初期化の障害を反映しない可能性があります。

原因

MPI プロセスの 1 つがシグナル (SIGTERM や SIGKILL など) で強制終了されました。MPI アプリケーションは tcp ファブリックを介して実行されました。この場合、MPI アプリケーションがハングアップする可能性があります。

解決方法

MPI プロセスが終了した原因を特定してください。この強制終了は、ホストのリブート、予期しないシグナルの受信、out-of-memory (OOM) マネージャー・エラーなどに起因する可能性があります。システムのログファイルを確認します。

例 3

症状 / エラーメッセージ

```
[mpiexec@node00] control_cb (../../pm/pmiserv/pmiserv_cb.c:773): connection to
proxy 1 at host node01 failed
[mpiexec@node00] HYDT_dmxu_poll_wait_for_event
(../../tools/demux/demux_poll.c:76): callback returned error status
[mpiexec@node00] HYD_pmci_wait_for_completion
(../../pm/pmiserv/pmiserv_pmci.c:501): error waiting for event
[mpiexec@node00] main (../../ui/mpich/mpiexec.c:1063): process manager error
waiting for completion
```

原因

node01 上のリモート pmi_proxy プロセスが、SIGKILL (9) シグナルによって強制終了されます。

解決方法

pmi_proxy プロセスが終了した原因を特定してください。この強制終了は、ホストのリブート、予期しないシグナルの受信、out-of-memory (OOM) マネージャー・エラーなどに起因する可能性があります。システムのログファイルを確認します。

例 4

症状 / エラーメッセージ

```
Failed to connect to host node01 port 22:No route to host
```

原因

MPI 計算ノードのうちの 1 つ (node01) が、ネットワーク上に見つかりませんでした。この場合、MPI アプリケーションがハングアップする可能性があります。

解決方法

ノードのネットワーク・インターフェイスをチェックし、ホストがアクセスできることを確認してください。

例 5

症状 / エラーメッセージ

```
Failed to connect to host node01 port 22:Connection refused
```

原因

MPI リモートノードのアクセス方式は SSH です。SSH サービスが `node01` 上で起動されていません。

解決方法

すべてのノードの SSH サービスの状態をチェックしてください。

11.2.2. 環境の問題

環境のエラーは、必須のシステムサービスが実行されていなかったり、共有リソースが利用できないなどの問題によって発生する可能性があります。

環境のエラーが発生した場合、環境をチェックします。例えば、重要なサービスの現在の状態を確認します。

例 1

症状 / エラーメッセージ

```
librdmacm:Warning: couldn't read ABI version.
librdmacm:Warning: assuming:4
librdmacm:Fatal: unable to get RDMA device list
```

または:

```
CMA: unable to get RDMA device list
librdmacm: couldn't read ABI version.
librdmacm: assuming:4
```

原因

OFED* スタックがロードされていません。MPI アプリケーションは `dapl` ファブリックを介して実行されました。この場合、MPI アプリケーションがハングアップする可能性があります。

解決方法

OFED* スタックの使用法の詳細は、OFED* のドキュメントを参照してください。

例 2

症状 / エラーメッセージ

```
[0] MPI startup():Multi-threaded optimized library
[1] DAPL startup(): trying to open DAPL provider from I_MPI_DAPL_PROVIDER: ofa-
v2-mlx4_0-1
[0] DAPL startup(): trying to open DAPL provider from I_MPI_DAPL_PROVIDER: ofa-
v2-mlx4_0-1
[1] MPI startup():DAPL provider ofa-v2-mlx4_0-1
[1] MPI startup(): dapl data transfer mode
[0] MPI startup():DAPL provider ofa-v2-mlx4_0-1
[0] MPI startup(): dapl data transfer mode
```

この場合、MPI アプリケーションがハングアップする可能性があります。

原因

サブネット管理 (`opensmd*`) サービスが起動されていません。MPI アプリケーションは `dapl` ファブリックを介して実行されました。`I_MPI_DEBUG=2` に設定すると、次のような出力が得られます。

解決方法

サービスの状態をチェックします。opensmd* 使用法の詳細は、OFED* のドキュメントを参照してください。

例 3

症状 / エラーメッセージ

```
node01-mic0:MCM:2b66:e56a0b40:2379 us(2379 us): scif_connect() to port 68,
failed with error Connection refused
node01-mic0:MCM:2b66:e56a0b40:2494 us(115 us): open_hca:SCIF init ERR for
mlx4_0
Assertion failed in file
../../src/mpid/ch3/channels/nemesis/netmod/dapl/dapls_module_init.c
at line 761:0
internal ABORT - process 0
```

原因

mpxyd デーモン (CCL-proxy) が起動されていません。MPI アプリケーションは dapl ファブリックを介して実行されました。この場合、MPI アプリケーションがハングアップする可能性があります。

解決方法

サービスの状態をチェックします。mpxyd の使用法の詳細は、DAPL* のドキュメントを参照してください。

例 4

症状 / エラーメッセージ

```
node01-mic0:SCM:2b94:14227b40:201 us(201 us): open_hca: ibv_get_device_list()
failed
node01-mic0:SCM:2b94:14227b40:222 us(222 us): open_hca: ibv_get_device_list()
failed
node01-mic0:CMA:2b94:14227b40:570 us(570 us): open_hca: getaddr_netdev ERROR:No
such device.Is ib0 configured?
...
Fatal error in MPI_Init:Other MPI error, error stack:
MPIR_Init_thread(784).................:
MPID_Init(1326).......................: channel initialization failed
MPIDI_CH3_Init(141)...................:
dapl_rc_setup_all_connections_20(1386): generic failure with errno = 872609295
getConnInfoKVS(849)...................:PMI_KVS_Get failed
```

原因

ofed-mic サービスが起動されていません。MPI アプリケーションは dapl ファブリックを介して実行されました。この場合、MPI アプリケーションがハングアップする可能性があります。

解決方法

サービスの状態をチェックします。ofed-mic の使用法の詳細は、インテル® MPSS のドキュメントを参照してください。

例 5

症状 / エラーメッセージ

```
pmi_proxy: line 0: exec: pmi_proxy: not found
```

原因

インテル® MPI ライブラリーのランタイム・スクリプトが利用できません。共有ストレージにアクセスできない可能性があります。この場合、MPI アプリケーションがハングアップする可能性があります。

解決方法

共有パスが、すべてのノード上で利用可能であるか確認してください。

例 6

症状 / エラーメッセージ

```
[0] DAPL startup:RLIMIT_MEMLOCK too small
[0] MPI startup(): dapl fabric is not available and fallback fabric is not enabled
```

または:

```
node01:SCM:1c66:3f226b40: 6815816 us:DAPL ERR reg_mr Cannot allocate memory
```

原因

誤ったシステムの制限: 最大ロックメモリーが小さすぎます。MPI アプリケーションは `dapl` ファブリックを介して実行されました。

解決方法

システムの制限と必要であれば更新を確認します。次のコマンドで正しいシステムの制限を確認します。

```
$ ulimit –a
core file size (blocks, -c) 0
data seg size (kbytes, -d) unlimited
scheduling priority (-e) 0
file size (blocks, -f) unlimited
pending signals (-i) 256273
max locked memory (kbytes, -l) unlimited
max memory size (kbytes, -m) unlimited
open files (-n) 1024
pipe size (512 bytes, -p) 8
POSIX message queues (bytes, -q) 819200
real-time priority (-r) 0
stack size (kbytes, -s) unlimited
cpu time (seconds, -t) unlimited
max user processes (-u) 1024
virtual memory (kbytes, -v) unlimited
file locks (-x) unlimited
```

例 7

症状 / エラーメッセージ

```
Are you sure you want to continue connecting (yes/no)?The authenticity of host
'node01 (<node01_ip_address>)' can't be established.
```

このメッセージは、手動で中断されるまで繰り返し表示されます。

原因

MPI のリモートノードのアクセス方式は SSH です。SSH が正しく設定されていません: 標準入力 (`stdin`) に予期しないメッセーがあります。

解決方法

問題があるノードへの SSH 接続を確認してください。

例 8

症状 / エラーメッセージ

```
Password:
```

原因

MPI リモートノードのアクセス方式は SSH です。SSH がパスワードなしではありません。この場合、MPI アプリケーションがハングアップする可能性があります。

解決方法

SSH の設定を確認します: パブリックキーによるパスワードなし認証が有効で設定されていること。

11.2.3. その他の問題

例 1

症状 / エラーメッセージ

```
cannot execute binary file
```

原因

実行形式のバイナリーのフォーマットもしくはアーキテクチャーが不正です。

このエラーは、x86_64 アーキテクチャー向けにビルドした実行形式バイナリーを k1om ノード (インテル® Xeon Phi™ コプロセッサー) で実行した場合に発生します。この場合、MPI アプリケーションがハングアップする可能性があります。

解決方法

バイナリーファイルの形式とコマンドラインのオプションを確認します。

例 2

症状 / エラーメッセージ

```
node01.9234ipath_userinit: assign_context command failed:Invalid argument
node01.9234Driver initialization failure on /dev/ipath (err=23)
```

原因

インテル® True Scale IBA のリソース枯渇。MPI アプリケーションは tmi ファブリックを介して実行されました。

一部のインテル® True Scale ファブリック・ハードウェアでは、PSM* がノードの CPU オーバーサブスクリプションをサポートしていません。ノードで実行できる最大プロセス数は、インテル® True Scale ファブリック・ハードウェアと CPU コアの数に依存します。

解決方法

ノード当たりの MPI プロセス数を制限します。

12. インテル® MIC アーキテクチャーでインテル® MPI ライブラリーを使用する

インテル® MPI ライブラリーをインテル® メニー・インテグレーテッド・コア (インテル® MIC) アーキテクチャー・カードと組み合わせて使用するのは、ほかのノードを使用するのと似ていますが、いくつか特殊な考慮事項があります。ここでは、この特殊な考慮事項について説明します。

12.1. ライブラリー

インテル® MIC アーキテクチャーは、ホストと異なるバイナリーとライブラリー使用し、それらはカード上に存在する必要があります。必要なファイルをカードにコピーするには、次のコマンドを使用します。

```
(host)$ scp <installdir>/mic/bin/* host0-mic0:/bin/
(host)$ scp <installdir>/mic/lib/* host0-mic0:/lib64/
```

この例では、カードのホスト名を `host0-mic0` と仮定します。アプリケーションが必要とするライブラリーも、同様の方法でコピーすることができます。

12.2. 複数のカード

1 つのジョブで複数のカードを使用するため、インテル® メニーコア・プラットフォーム・ソフトウェア・スタック (インテル® MPSS) は、ピアツーピアをサポートするように設定し (詳細は、インテル® MPSS のドキュメントをご覧ください)、ホストは IP フォワード機能を有効にする必要があります。

```
(host)$ sudo sysctl -w net.ipv4.ip_forward=1
```

各ホスト/カードはほかのすべてのホスト/カードに `ping` できなければならず、起動ホストは一般的なクラスターと同じようにすべてのターゲットに接続できる必要があります。

12.3. インテル® Xeon Phi™ コプロセッサーでインテル® MPI ライブラリーを使用する

インテル® MIC アーキテクチャー向けのインテル® MPI ライブラリーは、インテル® Xeon Phi™ コプロセッサー (コード名: Knights Corner) のみをサポートします。

12.3.1. インテル® MPI アプリケーションのビルド

ホストノードとインテル® Xeon Phi™ コプロセッサー向けの MPI アプリケーションをビルドするには、次の手順に従います。

1. コンパイラーとインテル® MPI ライブラリー向けの環境設定を行います。
   ```
   $ .<install-dir>/compilers_and_libraries/linux/bin/compilervars.sh intel64
   $ .<install-dir>/compilers_and_libraries/linux/mpi/intel64/bin/mpivars.sh
   ```
2. インテル® Xeon Phi™ コプロセッサー向けにネイティブ・アプリケーションをビルドします。
   ```
   $ mpiicc -mmic myprog.c -o myprog.mic
   ```
3. インテル® 64 アーキテクチャー向けにアプリケーションをビルドします。
   ```
   $ mpiicc myprog.c -o myprog
   ```

12.3.2. インテル® MPI アプリケーションの実行

ホストノードとインテル® Xeon Phi™ コプロセッサー向けの MPI アプリケーションを実行するには、次の手順に従います。

1. ホストとインテル® Xeon Phi™ コプロセッサー間で NFS が適切に設定されていることを確認します。インテル® Xeon Phi™ コプロセッサー上での NFS の設定に関する情報は、インテル® Xeon Phi™ コプロセッサーの開発者コミュニティー (http://www.isus.jp/article/idz/mic-developer/) を参照ください。
2. インテル® Xeon Phi™ コプロセッサー上で実行する場合、`mic` 拡張子を追加するため `I_MPI_MIC_POSTFIX` 環境変数を使用します。
   ```
   $ export I_MPI_MIC_POSTFIX=.mic
   ```
3. `~/mpi.hosts` ファイルが、インテル® Xeon® プロセッサー (ホスト) とインテル® Xeon Phi™ コプロセッサーのマシン名を含んでいることを確認します。次に例を示します。
   ```
   $ cat ~/mpi.hosts
   clusternode1
   clusternode1-mic0
   ```
4. ホストから実行形式ファイルを起動します。
   ```
   $ export I_MPI_MIC=on
   $ mpirun -n 4 -hostfile ~/mpi.hosts ./myprog
   ```

注意

`-configfile` と `-machinefile` オプションを使用することもできます。

アプリケーションをインテル® Xeon Phi™ コプロセッサー上でのみ実行する場合、次の手順に従い `mpi.hosts` がインテル® Xeon Phi™ コプロセッサーの名前のみを含んでいることを確認します。

関連情報

ローカルのオプションに関する詳細は、『インテル® MPI ライブラリー for Linux* リファレンス・マニュアル』をご覧ください。

インテル® Xeon Phi™ コプロセッサー上のインテル® MPI ライブラリーに関する詳細は、「インテル® Xeon Phi™ コプロセッサー上で MPI プログラムを実行する方法」(英語) をご覧ください。