インテル® MPI ライブラリー入門ガイド

(参考訳)

著作権と商標について

本資料に掲載されている情報は、インテル製品の概要説明を目的としたものです。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を許諾するためのものではありません。製品に付属の売買契約書『Intel's Terms and Conditions of Sales』に規定されている場合を除き、インテルはいかなる責を負うものではなく、またインテル製品の販売や使用に関する明示または黙示の保証 (特定目的への適合性、商品性に関する保証、第三者の特許権、著作権、その他、知的所有権を侵害していないことへの保証を含む) に関しても一切責任を負わないものとします。インテル製品は、医療、救命、延命措置、重要な制御または安全システム、核施設などの目的に使用することを前提としたものではありません。

本資料およびこれに記載されているソフトウェアはライセンス契約に基づいて提供されるものであり、その使用および複製はライセンス契約で定められた条件下でのみ許可されます。本資料で提供される情報は、情報供与のみを目的としたものであり、予告なく変更されることがあります。また、本資料で提供される情報は、インテルによる確約と解釈されるべきものではありません。インテルは本資料の内容およびこれに関連して提供されるソフトウェアにエラー、誤り、不正確な点が含まれていたとしても一切責任を負わないものとします。

機能や命令の中に「予約済み」または「未定義」と記されているものがありますが、その機能が存在しない状態や何らかの特性を設計の前提にしてはなりません。これらの項目は、インテルが将来のために予約しているものです。インテルが将来これらの項目を定義したことにより、衝突が生じたり互換性が失われたりしても、インテルは一切責任を負わないものとします。

本資料で説明されているソフトウェアには、不具合が含まれている可能性があり、公開されている仕様とは異なる動作をする場合があります。現在までに判明している不具合の情報については、インテルのサポートサイトをご覧ください。

Intel、インテル、Intel ロゴは、アメリカ合衆国およびその他の国における Intel Corporation またはその子会社の商標または登録商標です。

* その他の社名、製品名などは、一般に各社の商標または登録商標です。

MPI の法務情報

インテル MPI ライブラリーは、アルゴンヌ国立研究所* (ANL) によって開発された MPI の MPICH2* 実装の一部を基にしています。

また、オハイオ州立大学 NBCL (Network-Based Computing Laboratory) で開発された MVAPICH2* による InfiniBand* アーキテクチャー RDMA ドライバーの一部も基にしています。

目次

概要

インテル® MPI ライブラリーは、マルチファブリック対応のメッセージ・パッシング・ライブラリーで、MPI (Message Passing Interface)、v2 (MPI-2) 仕様を実装します。再リンクせずに、相互接続ファブリックを切り替えることができます。

この『入門ガイド』では、インテル MPI ライブラリーを使用して、簡単な MPI プログラムのコンパイルと実行について説明します。また、基本的な使用例やトラブルシューティングのヒントも提供します。

このインテル MPI ライブラリーのリリースでは、次の主な機能がサポートされています。

- MPI-1 仕様と MPI-2 仕様の準拠
- 下記のファブリックの切り替え、マルチファブリック・サポート:
  - TCP (Ethernet*、ソケット、Gigabit Ethernet)
  - 共有メモリー
  - Hybrid TCP と共有メモリー、SMP ノードのクラスターにおける使用
  - InfiniBand、Myrinet*、その他の RDMA 対応のファブリック方式のネットワーク (DAPL* プロバイダー経由)
  - Hybrid TCP、共有メモリー、InfiniBand、Myrinet、およびその他の RDMA 対応のファブリック方式のネットワーク (DAPL プロバイダー経由)、SMP ノードのクラスターにおける使用
- インテル® C++ コンパイラー Linux* 版 7.1 以上、インテル® Fortran コンパイラー Linux 版 7.1 以上、GNU コンパイラーを使用する IA-32 アーキテクチャー・クラスターと Itanium® アーキテクチャー・クラスターのサポート
- インテル C++ コンパイラー Linux 版 8.1 以上、インテル Fortran コンパイラー Linux 版 8.1 以上、GNU コンパイラーを使用するインテル® エクステンデッド・メモリー 64 テクノロジー (インテル® EM64T) のサポート
- C、C++、Fortran-77 および Fortran-90 言語のバインディング
- 動的リンクまたは静的リンク
- 同種のプロセッサー・アーキテクチャーとオペレーティング環境のみによるクラスター

MPI-2 仕様は、MPI-1 のフルサポートおよび次の新しい機能を提供します。

- 一方向通信 (RDMA 読み取りと書き込み)
- 拡張された集合演算
- 強化され、標準化された I/O 機能
- MPD デーモン (Multi-Purpose Daemons) と `mpiexec` コマンドによる標準化されたジョブ・スタートアップ

MPI-2 実装の制限についての情報は、製品のリリースノート (英語) の「Known Limitations」を参照してください。

インテル® MPI ライブラリーの使用

はじめに

インテル® MPI ライブラリーを使用する前に、ライブラリー、スクリプト、ユーティリティー・アプリケーションがインストールされていることを確認してください。インストールについての説明は、製品のリリースノート (英語) を参照してください。

使用モデル

インテル MPI ライブラリーを使用するステップを次に示します。

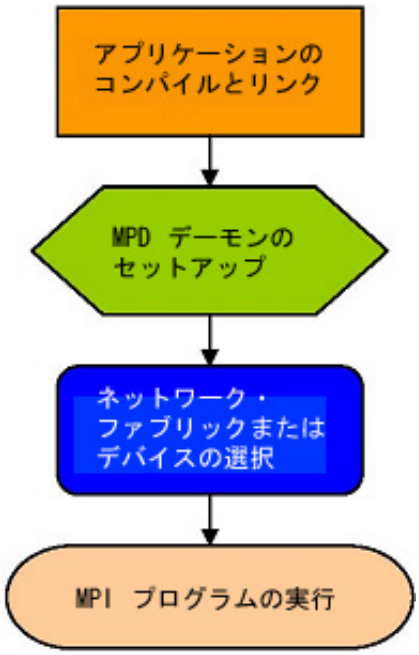

図 1: インテル MPI ライブラリーで作業するための使用モデルを表したフローチャート

1. コンパイルとリンク

MPI プログラムをコンパイルしてインテル MPI ライブラリーとリンクするには:

1. 使用するコンパイラーと関連するソフトウェアが PATH にあることを確認してください。
2. インテル® コンパイラーを使用する場合は、コンパイラー・ライブラリーのディレクトリーが LD_LIBRARY_PATH 環境変数に含まれていることを確認してください。例えば、インテル® C++ コンパイラー 8.1 と インテル® Fortran コンパイラー 8.1 の場合、次のセットアップ・スクリプトを実行します。

   `/opt/intel_cc_80/bin/iccvars.[c]sh`、および

   `/opt/intel_fc_80/bin/ifortvars.[c]sh`

3. MPI プログラムを適切な `mpi` コマンドでコンパイルします。例えば、GNU C コンパイラーを使用して C コードをコンパイルするには、次のように `mpicc` コマンドを使用します。

   ```
   $ mpicc <installdir>/test/test.c
   ```

   サポートされるすべてのコンパイラー用に、標準的なコンパイラー・コマンドに接頭辞 `mpi` が付いた対応するコマンドが用意されています。例えば、インテル Fortran コンパイラー 8.1 以上 (`ifort`) 用のインテル MPI ライブラリー・コマンドは `mpiifort` です。

2. MPD デーモンのセットアップ

インテル MPI ライブラリーでは、MPD (Multi-Purpose Daemon) ジョブ・スタートアップ・メカニズムが使用されます。`mpicc` (または関連する) コマンドを使用してコンパイルされたプログラムを実行するには、MPD デーモンをセットアップします。

システム・アドミニストレーターにシステム上のすべてのユーザーにより使用される MPD デーモンを一度に起動させるのではなく、常に、専用の MPD デーモンのセットを起動し、管理します。このセットアップにより、システムのセキュリティが強化され、実行環境の制御を柔軟に行うことができます。

MPD デーモンをセットアップするには:

1. `cshrc` ファイルまたは `.bashrc` ファイルなどで、適切な値とディレクトリーを使用して環境変数を設定します。

   `PATH` 変数に、`<installdir>/bin` ディレクトリー (インテル® EM64T 64 ビット・モードの場合は `<installdir>/bin64`) が含まれていることを確認してください。この変数を設定するには、インテル MPI ライブラリーに含まれている `mpivars.[c]sh` スクリプトを使用します。

   `PATH` 変数に、Python* 2.2 以上のディレクトリーが含まれていることも確認してください。

   インテル® コンパイラーを使用する場合は、`LD_LIBRARY_PATH` 変数にコンパイラー・ライブラリーのディレクトリーが含まれていることを確認してください。この変数は、コンパイラーに含まれている `*vars.[c]sh` スクリプトを使用して設定します。

   アプリケーションが使用するその他の環境変数も設定します。

2. `$HOME/.mpd.conf` ファイルを作成します。MPD パスワードを設定するには、このファイルに次の行を入力します。

   ```
   secretword=<mpd secret word>
   ```

   Linux ログイン・パスワードは使用しないでください。任意の `<mpd secret word>` 文字列だけが、様々なクラスターユーザーによる MPD デーモンへのアクセスを制御します。

3. `chmod` コマンドを使用して `$HOME/.mpd.conf` ファイルを保護し、他のユーザーが読み取りおよび書き込み権限を持たないようにします。

   ```
   $ chmod 600 $HOME/.mpd.conf
   ```

4. PATH 設定と `mpd.conf` コンテンツをクラスターのすべてのノード上の rsh で参照できることを確認します。例えば、クラスターの各 `<node>` で次のコマンドを使用します。

   ```
   $ rsh <node> env
   ```

   ```
   $ rsh <node> cat $HOME/.mpd.conf
   ```

1 つのノードだけではなく、各ノードが他のノードに接続できることを確認します。

クラスターで `rsh` の代わりに ssh が使用される場合は、次の「メモ」を参照してください。

5. 1 行に 1 つのホスト名が記されたクラスターノードのリスト、`mpd.hosts` テキストファイルを作成します。
6. `mpdallexit` コマンドを使用して MPD デーモンをシャットダウンします。

   ```
   $ mpdallexit
   ```

7. `mpdboot` コマンドを使用して MPD デーモンを起動します。

   `$ mpdboot -n <#nodes> $PWD/mpd.hosts` ファイルがある場合は、このファイルがデフォルトとして使用されます。ホストファイルがない場合は、`mpdboot` コマンドによりローカルマシンの 1 つの MPD デーモンが起動されます。

8. `mpdtrace` コマンドを使用して MPD デーモンのステータスを特定します。

   `$ mpdtrace` 出力は、現在 MPD デーモンを実行しているノードのリストです。このリストは、`mpd.hosts` ファイルの内容と一致します。

メモ

クラスターで `rsh` の代わりに `ssh` が使用される場合は、パスワードを入力しなくても各ノードが他のノードに接続できることを確認してください。`ssh` セットアップについての詳細は、システムのマニュアルを参照してください。

`rsh` の代わりに `ssh` がクラスターで使用される場合は、`-r ssh` オプションを `mpdboot` 起動文字列に追加します。

3. ネットワーク・ファブリックまたはデバイスの選択

インテル MPI ライブラリーでは、MPI プロセス間の異なる通信チャネルをサポートするために、複数の動的に選択可能な MPI デバイスが提供されています。

MPI デバイスを選択するには、I_MPI_DEVICE 環境変数を次のいずれかの値に設定します。

I_MPI_DEVICE 値	サポートされるファブリック
sock	TCP/Ethernet/ソケット
shm	共有メモリーのみ (ソケットなし)
ssm	TCP + 共有メモリー (Ethernet を介して接続される SMP クラスター用)
rdma[:<provider>]	InfiniBand、Myrinet、その他 (指定の DAPL プロバイダー経由)
rdssm[:<provider>]	TCP + 共有メモリー + DAPL (RDMA 対応のファブリックを介して接続される SMP クラスター用)

選択されたファブリックが利用可能であることを確認します。例えば、shm デバイスは、すべてのプロセスが共有メモリーを介して互いに通信できる時のみに使用します。同様に、rdma デバイスは、すべてのプロセスが 1 つの DAPL プロバイダーを介して互いに通信できる時のみに使用します。

4. MPI プログラムの実行

インテル MPI ライブラリーがリンクされたプログラムを起動するには、mpiexec コマンドを使用:

```
$ mpiexec -n <# of processes> ./myprog
```

-n オプションで、プロセス数を設定します。このオプションは、mpiexec コマンドで唯一、必須のオプションです。

デフォルトのファブリックではなくネットワークのファブリックを使用する場合は、-genv オプションを使用して I_MPI_DEVICE 変数に割り当てられる値を指定します。

例えば、rdssm デバイスを使用して 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。

```
$ mpiexec -genv I_MPI_DEVICE rdssm -n <# of processes> ./a.out
```

rdma デバイスの場合、次のコマンドを使用します。

```
$ mpiexec -genv I_MPI_DEVICE rdma -n <# of processes> ./a.out
```

任意のサポートデバイスを選択できます。詳細は、「ネットワーク・ファブリックまたはデバイスの選択」を参照してください。

インテル MPI ライブラリーを使用してアプリケーションが実行できれば、他の MPI ライブラリーでも実行できます。これで、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してアプリケーションをあるクラスターから別のクラスターへ移動させることができます。問題が発生した場合は、「トラブルシューティング」 を参照してください。

トラブルシューティング

次のセクションでは、インテル® MPI ライブラリーのインストール、セットアップ、アプリケーションの実行に関わる問題のトラブルシューティングについて説明しています。

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、一般的なテストを行い、テストプログラムをコンパイルして実行します。

インストール・テストを行うには:

1. 次のコマンドを使用して Python v2.2 以上が PATH に含まれていることを確認します。

   ```
   $ rsh <nodename> python -V
   ```

   または

   ```
   $ mpiexec -n <# of processes> python -V
   ```

   エラーメッセージが返されたり、返された値が 2.2 よりも低い場合は Python v2.2 以上をインストールし、インストールした Python が PATH に含まれていることを確認します。

2. python-xml* または libxml2-python* などの Python XML モジュールがインストールされていることを確認します。

   ```
   $ rpm -qa | grep python-xml
   ```

   ```
   $ rpm -qa | grep libxml2-python
   ```

   出力に "python-xml" または "libxml2-python" という名前とバージョン番号が含まれていない場合は、必要なモジュールをインストールします。

3. expat* または pyxml* などの XML パーサーがインストールされていることを確認します。

   ```
   $ rpm -qa | grep expat
   ```

   ```
   $ rpm -qa | grep pyxml
   ```

   出力に "expat" または "pyxml" という名前とバージョン番号が含まれていない場合は、必要なモジュールをインストールします。

4. `<installdir>/bin` (インテル® EM64T 64 ビット・モードの場合は `<installdir>/bin64`) が PATH に含まれていることを確認します。

   ```
   $ rsh <nodename> which mpiexec
   ```

   テストする各ノードの適切なパスが表示されます。

5. インテル® コンパイラーを使用する場合は、適切なディレクトリーが PATH 環境変数と LD_LIBRARY_PATH 環境変数に含まれていることを確認します。`$ mpiexec -n <# of processes> env | grep PATH` テストする各ノードのパス変数に対する適切なディレクトリーが表示されます。表示されない場合は、適切な `*vars.[c]sh` スクリプトを呼び出します。 例えば、インテル® C++ コンパイラー 8.1 の場合、次のソースコマンドを使用します。

   ```
   $ . /opt/intel_cc_80/bin/iccvars.sh
   ```

6. ある特定の状況では、<installdir>/lib ディレクトリー (インテル® EM64T 64 ビット・モードの場合は <installdir>/lib64) を LD_LIBRARY_PATH に含める必要があります。次のコマンドを使用して、LD_LIBRARY_PATH 設定を確認します。

```
$ mpiexec -n <# of processes> env | grep PATH
```

MPD セットアップのトラブルシューティング

このセクションでは、MPD デーモンがセットアップされ、実行されていることが前提となっています。診断を始める前に、次のコマンドを使用して、MPD デーモンがすべてのノードで実行されていることを確認してください。

```
mpdtrace
```

実行されているすべての MPD デーモンがリストされるか、またはエラーが表示されます。mpdtrace の出力に従って次のように問題を解決します。

ケース 1: 実行されているすべての MPD デーモンがリストされていない

mpdtrace の出力リストに必要なノードが含まれていない場合は、次の操作を行います。

1. 次のコマンドを使用して MPD デーモンを再起動します。
   a. 実行されているすべての MPD デーモンを終了します。
      ```
      $ mpdallexit
      ```
   b. 各ノードで、すべてのデーモンが終了されていることを確認します。
      ```
      $ rsh <nodename> ps -ael | grep python
      $ rsh <nodename> kill -9 <remaining python processes>
      ```
   c. MPD デーモンを再起動します。適切な構成オプションとホストファイルが使用されていることを確認します。
      ```
      $ mpdboot [<options>]
      ```
   d. すべての MPD デーモンが実行されていることを確認します。
      ```
      $ mpdtrace
      ```
2. mpdtrace コマンドによる出力結果で、実行されていない MPD デーモンがある場合は、次の操作を行います。
   a. の説明にあるように、MPD デーモンを終了し、デバッグオプションと詳細オプションを mpdboot コマンドに追加して再起動します。
      ```
      $ mpdboot -d -v [<options>]
      ```
      ステップ a の出力にある rsh コマンドに注意してください。
      例:
      ```
      cmd=:rsh  <nodename> -n '<installdir>/bin/mpd ¥
       -d -h <nodename> -p <port-number> < /dev/null':
      ```
   b. rsh コマンドからの出力行をコピーして貼り付け、stdin リダイレクトを含めます。

例:

```
$ rsh  <nodename> -n '<installdir>/bin/mpd ¥
 -d -h <nodename> -p <port-number> < /dev/null'
```

c. 編集された rsh コマンドを実行します。結果出力を参考に、問題を診断し、修正します。例えば、最も一般的な問題には次のようなものがあります。

- rsh コマンドで <nodename> にアクセスできない。
- その他の rsh コマンドの失敗 (システム・セットアップの問題など)。
- <installdir>/bin/mpd コマンドが見つからない、または実行できない。
- mpd.conf ファイルが見つからない、または読み取れない (アクセスエラー)。

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して実行するには、次の操作を行います。

1. 製品リリースに含まれているテストプログラムを次のようにコンパイルします。

```
$ cd <installdir>/test
$ mpicc test.c
```

2. InfiniBand、Myrinet、またはその他の RDMA 対応のネットワーク・ハードウェアとソフトウェアを使用している場合は、すべてが適切に動作していることを確認します。
3. クラスター上のすべての利用できるデバイスでテストプログラムを実行します。

a. 次のコマンドで sock デバイスをテストします。

```
$ mpiexec -n 2 -env I_MPI_DEBUG 2 -env I_MPI_DEVICE sock ¥ ./a.out
```

各ランクに 1 行の出力と、sock デバイスが使用されていることを示すデバック出力が表示されます。

b. 次のコマンドを使用して ssm デバイスをテストします。

```
$ mpiexec -n 2 -env I_MPI_DEBUG 2 -env I_MPI_DEVICE ssm ./a.out
```

各ランクに 1 行の出力と、ssm デバイスが使用されていることを示すデバック出力が表示されます。

c. 次のコマンドを使用して、その他のファブリック・デバイスをテストします。

```
$ mpiexec -n 2 -env I_MPI_DEBUG 2 -env I_MPI_DEVICE <device>
¥ ./a.out
```

<device> は shm、rdma、または rdssm のいずれかです。使用される各 mpiexec コマンドで、各ランクに 1 行の出力と、どのデバイスが使用されているかを示すデバック出力が表示されます。デバイスが、I_MPI_DEVICE 設定と一致していることを確認してください。

メモ

インテル MPI ライブラリー開発キットの <installdir>/test ディレクトリーには、test.c のほか、テストに使用できるプログラムが含まれています。