チュートリアル:
インテル® MPI ライブラリー向け MPI Tuner

バージョン 5.1 Update 3 (Linux*)

著作権と商標について

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

概要

インテル® MPI ライブラリー向けの MPI Tuner を使用して、ランタイム・ライブラリーの最適化構成ファイルを自動的に取得する方法を説明します。また、このチュートリアルを通して基本的なトラブルシューティングのヒントを得ることができます。

チュートリアルの概要	このチュートリアルでは、開発者のクラスターとアプリケーション向けにインテル® MPI ライブラリーのパフォーマンスを最適化するいくつかの方法を紹介します。以下を含みます。 • クラスターのチューニング時間を最小化する • クラスターのチューニング中、デフォルトのパラメーター・グリッドの欠損を含める • アプリケーションのチューニング中の最適な設定の構成 • MPI tuner を利用する際によくある問題の解決方法
所要時間	15-20 分
目的	このチュートリアルでは、以下のトピックについて説明します。 • クラスターやアプリケーションの構成に関連するインテル® MPI ライブラリーの最適な設定を見つけるため、MPI Tuner を使用します。 • MPI Tuner を利用する際によくある問題の解決方法を示します。
その他の情報	インテル® MPI ライブラリー向けの MPI Tuner 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。 製品 Web サイト インテル® MPI ライブラリーのサポート インテル® クラスターツール製品 インテル® ソフトウェア開発製品

必要条件

インテル® MPI ライブラリー向けの MPI Tuner を使用する前に、ライブラリー、スクリプトおよびユーティリティーがインストールされていることを確認してください。インストール方法の詳細は、『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。

ナビゲーションのクイックスタート

MPI Tuner を使用するには:

1. `mpitune` ユーティリティーを使用して、最適化された設定ファイルを作成します。
2. `mpirun` コマンドの通常の実行に `-tune` オプションを追加して設定ファイルを使用します。

注:

MPI Tuner を使用する前に、実行するタスクをチェックできます。実際に実行する前に、`mpitune` の適用範囲を確認するには、`--scheduler-only` (`-so`) オプションを使用します。

```
$ mpitune ...-so
```

MPI Tuner を起動する

MPI Tuner を使用するには次のコマンドを起動します。

```
$ mpitune
```

注:

このコマンドは、インテル® Xeon Phi™ コプロセッサー上ではネイティブ起動できませんが、ホストから起動することでインテル® MPI ライブラリーでサポートされるプラットフォームをチューニングできます。

MPI Tuner コマンド

MPI Tuner ユーティリティーは、次の 4 つのモードで動作します。

- クラスター固有。インテル® MPI ライブラリーの最適な設定を見つけるため、インテル® MPI Benchmarks やユーザーから提供されるベンチマーク・プログラムを使用して、クラスター環境を評価します。このモードがデフォルトで使用されます。
- アプリケーション固有。特定のアプリケーション向けにインテル® MPI ライブラリーの最適な設定を見つけるため、MPI アプリケーションの性能を評価します。
- 高速アプリケーション固有。アプリケーション固有モードと異なり、このモードは実際のアプリケーションを起動する代わりに、インテル® MPI ライブラリーの統計ベースのマイクロカーネルのテストを行います。このアプローチはそれほど入念ではありませんが、ホストやコミュニケーターごとのプロセス数やメッセージサイズを含むランクの配置など、適切ではない設定でパッケージを使用しなくてはいけない、非典型的なパターンのアプリケーションを高速化できます。チューニングの範囲は、`KN_RADIX` サブセットと `I_MPI_ADJUST_REDUCE_SEGMENT` を除く、`I_MPI_ADJUST` ファミリーの環境変数に関連する通常の集団操作に関係します。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。
- トポロジーを考慮したランク配置の最適化。最適なランク配置を見つけるため、MPI アプリケーションのランク間のデータ転送と、クラスター特性を評価します。このアプローチは、ステンシル、ランクのサブセットの集合操作、隣接操作など、通信パターンがローカルグループであるアプリケーションに適しています。

クラスター固有のチューニング

クラスター固有モードで MPI Tuner を使用するには次の操作を行います。

1. 次のコマンドを使用して、デフォルトの `<installdir>/<arch>/etc` ディレクトリーにチューニングされた設定ファイルを作成します。

   ```
   $ mpitune -hf <hostfile>
   ```

   もしくは、`-odr` オプションを使用して、チューニングされた設定ファイルを作成するディレクトリーを指定します。

   ```
   $ mpitune -hf <hostfile> -odr <path_to_result_directory>
   ```

2. ディレクトリーを指定して、あるいはデフォルト・ディレクトリーから引数ファイルを読み込む場合は引数を指定せず、`-tune` オプションを使用します。次に例を示します。

   ```
   $ mpirun -tune -ppn 8 -n 128 ./my_app
   ```

   ```
   $ mpirun -tune <path_to_result_directory> -ppn 8 -n 128 ./my_app
   ```

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モードで MPI tuner を使用するには次の操作を行います。

1. 提供される環境とコマンド設定で指定するワークロードをチューニングするには、`--application` (`-a`) オプションを使用します。MPI Tuner は、新しい最適な設定を `myprog.conf` ファイルに生成します。

   ```
   $ mpitune --application \"mpirun -n 32 ./myprog\" –of ./myprog.conf
   ```

2. アプリケーションの実行時に生成された最適な設定を読み込むには、`–tune` オプションを使用します。

   ```
   $ mpirun -tune ./myprog.conf -n 32 ./myprog
   ```

タスク 1: クラスター固有モードでのチューニング時間を最小化する

クラスターのチューニング時間を減らすには、クラスターで最も頻繁かつ広範囲に使用されている MPI ワークロードについて考慮する必要があります。それらが以下に関してどのように実行されているかメモしておきます。

- 使用されるホスト数の範囲
- ホストごとのランク数
- ファブリックの使い方 (`I_MPI_FABRICS`)
- 典型的なメッセージサイズ
- 最も一般的な MPI 関数

ホストの範囲

例えば、大部分のワークロードがクラスター 4 から 16 の間のホスト数を使用する場合、-hr <n:m> オプションを使用してそれらの上限と下限を設定します。

```
$ mpitune ...-hr 4:16
```

mpitune ユーティリティーは、4 から 16 の間の 2 の累乗となるすべてのホスト範囲を構築します。ここでは、4 つのホスト、8 つのホスト、そして 16 のホスト向けの設定を生成します。

ホストごとのランク数

ホストごとのランク数を設定するには、-pr <n:m> オプションを使用します。

```
$ mpitune ...-pr 1:16
```

同様に、mpitune ユーティリティーは、1 から 16 の間の 2 の累乗となるランク範囲を構築します。例えば、ランク番号 1、2、4、8、16 のすべてのケース向けにチューニングされた設定を作成します。

```
$ mpitune ...-pr 24:24
```

上限と下限が同じである場合、mpitune ユーティリティーは、ppn=24 のみをチューニングします。

ファブリックの使い方 (I_MPI_FABRICS)

チューニングで使用するファブリックを指定するには、-fl オプションを使用します。

```
$ mpitune ...-fl shm:dapl,dapl,shm:ofa,ofa
```

mpitune ユーティリティーは、列挙されたファブリックのみを使用します。

メッセージサイズ

チューニングするメッセージの対象サイズ範囲を指定するには、-mr オプションを使用します。

```
$ mpitune ...-mr 16:2097152
```

この場合、mpitune ユーティリティーは、指定された 16 から 2097152 バイト間の 2 の累乗のメッセージサイズの MPI 操作をチューニングします。

最も一般的な MPI 関数

MPI 関数の利用とパフォーマンスに関する統計情報がある場合、要求に応じてチューニング範囲を調整することができます。チューニングを開始する前に、『インテル® MPI ライブラリー for Linux* ユーザーズガイド』の MPI 関数の各種チューニングオプションについてご覧ください。

最も広く使用される MPI ルーチンから始めて、より複雑な関数へと作業を進めることを推奨します。例えば、集合操作のチューニングの前に p2p のチューニングを行います。

最初に p2p に深く関連するオプションを指定します。

1. option_set 変数下で、最も一般的な MPI 関数を集めます。

```
$ export option_set=I_MPI_RDMA_TRANSLATION_CACHE\
,I_MPI_DAPL_RNDV_BUFFER_ALIGNMENT\
,I_MPI_SHM_FBOX_SIZE\
,I_MPI_SHM_CELL_SIZE\
```

```
    ,I_MPI_SSHM_BUFFER_SIZE\
    ,I_MPI_EAGER_THRESHOLD\
    ,I_MPI_DAPL_BUFFER_SIZE\
    ,I_MPI_INTRANODE_EAGER_THRESHOLD\
    ,I_MPI_DAPL_DIRECT_COPY_THRESHOLD
```

1. option_set でチューニング・セッションを実行します。ここでは、上記の環境変数に指定された値を基に、最適なインテル® MPI ライブラリー・クラスターの一連の設定を作成します。

   ```
   $ mpitune ...-os $option_set
   ```

2. 次の集合操作をチューニングします。

   ```
   $ mpitune ...--collective-only
   ```

完了したら、2 つの設定ファイルを 1 つにマージし、実行オプションに -tune または -config を指定してそのファイルを使用します。

注:

さらにチューニング時間を短縮するため、必要に応じて改善率を指定するか、許容可能なパフォーマンスを示すオプションを除外することができます。

タスク 2: クラスターのチューニング中、デフォルトのパラメーター・グリッドの欠損を含める

mpitune ユーティリティーは、2 の累乗であるほとんどの変数の値を列挙してチューニングするだけです。アプリケーションが非定型レイアウトやデータサイズを使用することが事前に判明している場合、カスタマイズしたセットで mpitune を実行しデフォルトを上書きすることができます。

<installdir>/<arch>/etc ディレクトリーに書き込み権限があることを確認してください。

mpitune ユーティリティーは、自己構成のため <installdir>/<arch>/etc にある *.xml ファイルをクラスター固有モード向けに何がどのように実装されているかを示す 2 つのファイル options.xml と Benchmarks/imb.xml があります。

例えば、I_MPI_EAGER_THRESHOLD 環境変数のチューニングをカスタマイズしたい場合、変更方法については次のハイライトされたテキストを参照してください。

```
options.xml:
...
    <option name="I_MPI_EAGER_THRESHOLD" type="global" group="collective"
weight="1.0">
    <actions>
        <step order="1" storage="first">
            <additive>
                <env name="I_MPI_FALLBACK_DEVICE" type="global"
value="disable" />
            </additive>
            <range name="range_vars">int_range(8192:524288:*:2)</range>
<!-- explicit range from 8k to 512k with power of 2 -->
```

```
            <format>@range_vars()</format>
            <result format="[msg_size]" limit="1" separator="" />
        </step>
    </actions>
    <requirements>
        <param name="hosts" value="2:2" /> <!--use 2 hosts -->
        <param name="perhost" value="1:1" /> <!-- with 1 process on host
-->
        <param name="processes" value="2:2" /> <!-- and 2 processes total
-->
        <param name="devices" value="shm:dapl,shm:tmi" /> <!-- for
shm:dapl and shm tmi fabrics (I_MPI_FABRICS) -->

    </requirements>
    <result <!-- internal format description -->

        format="#first#"
        quotes="no"
        quotesInline="no"
    />
</option>
...

Benchmarks/imb.xml:
    <test title="IMB Sendrecv" weight="1.0">
        <description>Sendrecv test from IMB benchmark for OUTPUT
mode</description>
        <executable>"IMB-MPI1" -npmin %proc% -iter 5 -msglen
@msglen_file() Sendrecv</executable>
        <function
title="msglen_file">range_file(768:1536:+:256;"value[endl]")</function>
<!-- msg len file of IMB with range: 768, 1024, 1280 and 1536 bytes -->

        <launch_line>%mpiexec% %globals% %locals% %executable%</launch_line>
        <requirements> <!-- values for requirements section are
calculated as intersection with the same block from options.xml file.
Results are in the mpitune schedule -->

            <param name="hosts" value="1:-1" />
            <param name="perhost" value="1:-1" />
            <param name="processes" value="2:-1" />
            <param name="devices"
value="rdssm,rdma,shm,ssm,sock,shm:dapl,shm:tcp,dapl,tcp,shm,shm:ofa,shm:
tmi,ofa,tmi" />
        </requirements>
        <options_filter filter="exclusive"> <!-- this section enumerates
options to tune by this benchmark-->
```

```
            <option type="global" name="I_MPI_EAGER_THRESHOLD" />
            <option type="global" name="I_MPI_INTRANODE_EAGER_THRESHOLD"
/>
        </options_filter>
        <result <!-- format to parse benchmark output -->

            source="brtime"
            paramGroup="4"
            paramTitle="t[usec]"
            paramTarget="min"
            paramLeftMarginGroup="2"
            paramRightMarginGroup="3"
            paramChooseMode="heaviest"
            paramDiffDelta="0.001"
            msgGroup="0"
            msgTitle="Bytes"
            iterationCompare="min"
            startline=".*(\#bytes\s+\#repetitions).*"
            dataline="\s+(\d+)\s+(\d+)\s+([\d\.]+)\s+([\d\.]+)\s+([\d\.]+)"
            solidatalines="1"
        />
    </test>
...
```

注:

オプションのチューニングでカスタム範囲を定義する場合、次のパラメーターを考慮する必要があります。

`test->result->source`

このパラメーターは、使用するベンチマークの設定ファイルで指定します。例えば、`thtime` 値を設定すると、明示的に定義された範囲が使用されます。ただし、範囲のすべての中間値が自動的に計算される `brtime` パラメーターを使用する場合、上限と下限を指定する必要があります。

タスク 3: アプリケーション固有のチューニング

クラスター固有のチューニングが完了すると、アプリケーション向けにインテル® MPI ライブラリーを最適化することに集中できます。前述のクラスター固有のチューニング手法に、次のような変更を加えてアプリケーション固有のチューニングに適用できます。

1. hosts やホストごとのプロセス数などのアプリケーション固有の設定は、`mpitune` ではなく `mpirun` のコマンドラインで指定されます。
2. 前述のマイクロベンチマークの代わりにアプリケーションを使用します。
3. `imb.xml` の代わりに、`app.xml` 設定ファイルを適用します。

高速アプリケーション固有のチューニング

高速アプリケーション固有モードで MPI Tuner を使用するには次の操作を行います。

1. このモードに切り替えるには、`--fast` (`-f`) オプションを使用し、提供される環境とコマンドライン設定、もしくは以前収集された統計を渡す `–stats` (`-s`) オプション向けに指定されワークロードをチューニングする `--application` (`-a`) オプションを使用します。MPI Tuner は、新しい最適な設定を `myprog.conf` ファイルに生成します。

   ```
   $ mpitune –-fast --application \"mpirun -n 32 ./myprog\" -o
   ./myprog.conf
   ```

   もしくは

   ```
   $ mpitune –f -s ./stats.txt -o ./myprog.conf
   ```

注:

このモードで利用可能なそのほかのオプションを見るには、`--fast` (`-f`) と `–help` (`-h`) オプションを同時に使用します。

---

2. アプリケーションの実行時に生成された最適な設定を読み込むには、`–tune` オプションを使用します。

   ```
   $ mpirun -tune ./myprog.conf -n 32 ./myprog
   ```

トポロジーを考慮したランク配置最適化コマンド

ランク配置を最適化するため MPI Tuner を使用には、システム上で MPI ランクを分散する順番にホストを列挙したファイルが必要です。ホストごとに 1 つ以上のプロセスを配置する場合、複製が必要になります。これは、MPI プロセス管理、クラスター・ジョブ・スケジューラー、またはリソース管理の特定の引数に依存します。例えば、4 つのホストにそれぞれ 2 つのプロセスを配置するには、通常次のようになります。

```
$ cat hostfile.in
$ host1
$ host1
$ host2
$ host2
$ host3
$ host3
$ host4
$ host4
```

1. `--rank-placement` (`-rp`)、`--hostfile-in` (`-hi`) および `--config-out` オプションを使用します。MPI Tuner は、新しい最適な設定を `myprog.conf` ファイルに生成します。

   ```
   $ mpitune --rank-placement --application \"mpirun -n 32 ./myprog\"
   --hostfile-in hostfile.in --config-out ./myprog.conf
   ```

2. アプリケーションの実行時に生成された最適な設定を読み込むには、`-tune` オプションを使用します。

```
$ mpirun -tune ./myprog.conf -n 32 ./myprog
```

もしくは

1. `--rank-placement` (`-rp`)、`--hostfile-in` (`-hi`) および `--hostfile-out` (`-ho`) オプションを使用します。MPI Tuner は、最適化されたホストリストを `hostfile.out` ファイルに生成します。

```
$ mpitune --rank-placement --application \"mpirun -n 32 ./myprog\"
--hostfile-out ./hostfile.out
```

2. アプリケーションの実行時に生成された最適な設定を読み込むには、`-machine` オプションを使用します。

```
$ mpirun -machinefile ./myprog.ho -n 32 ./myprog
```

注:

このモードで利用可能なその他のオプションを見るには、`--rank-placement` (`-rp`) と `--help` (`-h`) オプションを同時に使用します。

また、この機能は Hydra プロセス管理の `-use-topology-app` オプションを実行時に指定して利用できます。これにより、起動時間が大幅に増えますが、起動の瞬間のクラスター状態 (ヘルス、リソース競合など) が考慮されるため効果的です。このコマンドに関しては、『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。

トラブルシューティング

ここでは、MPI Tuner を実行する際に見られる一般的な問題の解決方法について説明します。

問題	原因と解決法
`mpitune` のスケジューラーが空である場合	1. `mpitune` の引数が相互に矛盾しないことを確認してください。例えば、`--options-set` と `--options-exclude` がオーバーラップしないなど。 2. ファブリックやデバイスのパスがチェックされない場合、同じ設定で MPI テスト・アプリケーションを実行してみてください。問題は、誤った hostfile や不適切なクラスター設定による可能性があります。
`mpitune` の実行時間が非常に長い場合	1. `–so` オプションを使用して、実際に実行する前にスケジュールのみの確認を行ってください。 タスク 1 とタスク 2 で説明した方法と (もしくは) それらを組み合わせて不要なジョブをスキップします。