インテル® Xeon Phi™ プロセッサー向けのプロセスとスレッド・アフィニティー

この記事は、インテル® デベロッパー・ゾーンに公開されている「Process and Thread Affinity for Intel® Xeon Phi™ Processors」の日本語参考訳です。

インテル® MPI ライブラリーと OpenMP* ランタイム・ライブラリーは、プロセス/スレッドとハードウェア・リソース間のアフィニティーを作成できます。アフィニティーは、MPI プロセス/OpenMP* スレッドが別のハードウェア・リソースへ移動し、プログラムの実行速度に大きく影響するのを防ぎます。

ハードウェア・スレッド

インテル® Xeon Phi™ プロセッサー (開発コード名 Knights Landing) は、コアあたり最大 4 つのハードウェア・スレッド・コンテキストをサポートします。図 1 に示すように、2 つのコアが 1 つの L2 キャッシュを共有し、1 つのタイルを構成します。

図 1: インテル® Xeon Phi™ プロセッサー x200 製品ファミリーのタイルは、2 つのコア、4 つのベクトル・プロセシング・ユニット (VPU)、1MB の共有キャッシュ、キャッシュ・ホーム・エージェントで構成される。

追加のハードウェア・スレッドはレイテンシーを隠蔽するのに役立ちます。1 つのハードウェア・スレッドがストールしても、別のハードウェア・スレッドでコアのスケジューリングを行うことができます。コアまたはタイルあたりの最適なハードウェア・スレッド数は、アプリケーションに依存します。アプリケーションによっては、ハードウェア・タイルあたり 1 スレッドでも利点が得られることがあります。この記事のすべての例では、インテル® Xeon Phi™ プロセッサーに 34 のタイルがあります。

OpenMP* スレッド・アフィニティー

OpenMP* は、ハードウェア・リソースの割り当てとハードウェア・リソースへのスレッドのピニングを切り離します。

インテル® コンパイラー 13.0 以降は、OpenMP* 4.0 のアフィニティー設定とインテルの OpenMP* ランタイム拡張をサポートします。次の設定は、ハードウェア・リソースを割り当て、OpenMP* スレッドをハードウェア・リソースにピニングします。

	OpenMP* 4.0 アフィニティー	インテルの OpenMP* ランタイム拡張
ハードウェア・スレッドの割り当て	OMP_PLACES	KMP_PLACE_THREADS
OpenMP* スレッドをハードウェア・スレッドにピニング	OMP_PROC_BIND	KMP_AFFINITY

インテルの OpenMP* ランタイム拡張のスレッド・アフィニティー

KMP_PLACE_THREADS は、ハードウェア・リソースの割り当てを制御します。OpenMP* アプリケーションには、コア数とコアあたりのスレッド数を割り当てることができます。ここで、C はコア、T はスレッドを表します。例えば、68c,4t は 68 コアでコアあたり 4 スレッドを、34c,2t は 34 コアでコアあたり 2 スレッドを指定します。

KMP_AFFINITY は、OpenMP* スレッドとリソースのバインド方法を制御します。一般的なタイプは、compact、scatter、balanced です。粒度は、core または thread に設定できます。図 2、図 3、図 4 にそれぞれのアフィニティー・タイプを示します。

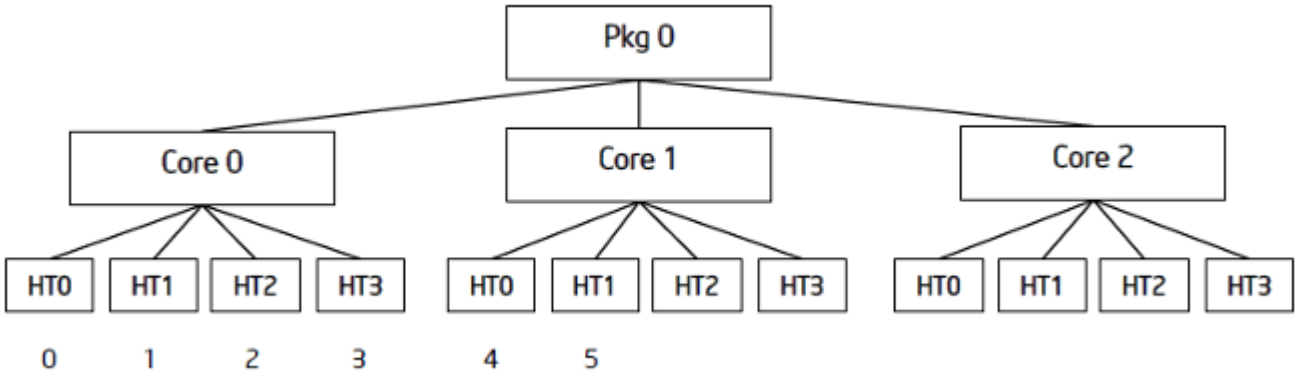

図 2: KMP_AFFINITY=compact

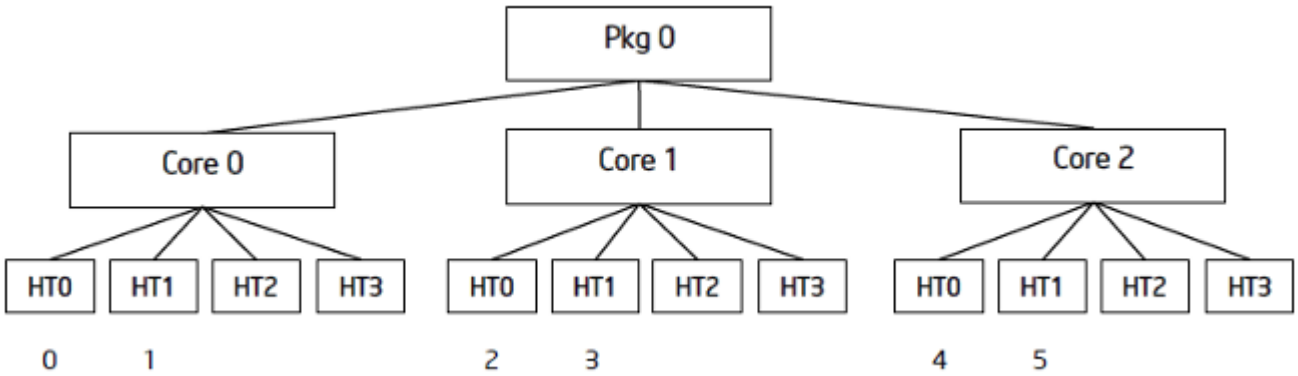

図 3: KMP_AFFINITY=balanced

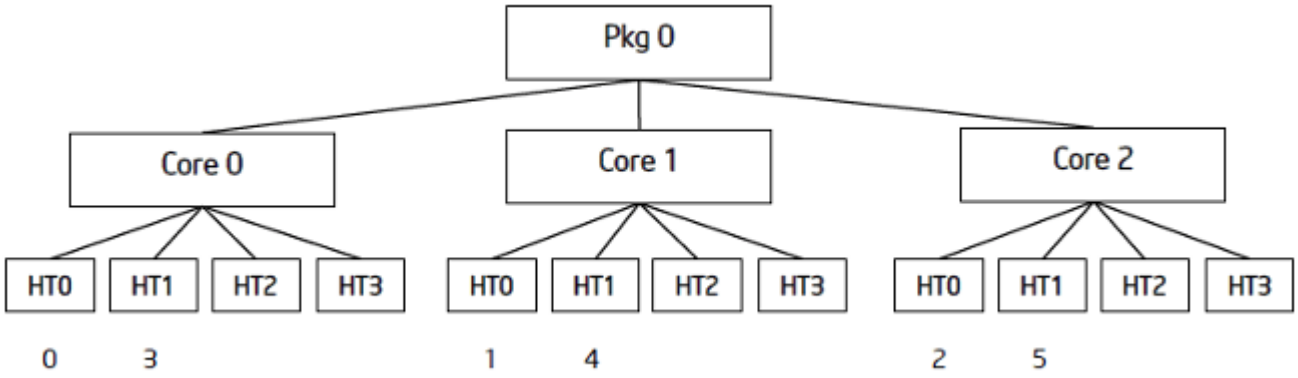

図 4: KMP_AFFINITY=scatter

KMP_PLACE_THREADS と KMP_AFFINITY に関する詳細は、『インテル® コンパイラー・デベロッパー・ガイドおよびリファレンス』の「スレッド・アフィニティー・インターフェイス」を参照してください。

次の例は、インテル® Xeon Phi™ プロセッサー x200 製品ファミリーベースの Linux* システム上で、インテルの OpenMP* ランタイム拡張を使用して、OpenMP* スレッドを特定のタイルまたはコアあたりのスレッド数にピニングします。

タイルあたり 1 スレッド
KMP_AFFINITY=proclist=[0,2,4,6,8,10,12,14,16,18,20,22,24,26,28,30,32,34,36,38,40,42,44,46,48,50,52,54,56,58,60,62,64,66],explicit

コアあたり 1 スレッド
KMP_PLACE_THREADS=1T KMP_AFFINITY=compact

コアあたり 2 スレッド
KMP_PLACE_THREADS=2T KMP_AFFINITY=compact

コアあたり 3 スレッド
KMP_PLACE_THREADS=3T KMP_AFFINITY=compact

コアあたり 4 スレッド
KMP_PLACE_THREADS=4T KMP_AFFINITY=compact

ヒント: KMP_AFFINITY VERBOSE 修飾子を使用して、スレッドと OS プロセッサーのマップと OS プロセッサーと物理コアのマップを確認できます。

同様の設定は、コアをアンダーサブスクライブする (利用可能なすべてのコアを使用しない) 場合にも利用できます。

タイルあたり 1 スレッド
KMP_AFFINITY=proclist=[0,2,4,6,8,10,12,14,16,18,20,22,24,26,28,30,32,34,36,38,40,42,44,46,48,50,52,54,56,58,60,62,64,66],explicit OMP_NUM_THREADS=4

コアあたり 1 スレッド
KMP_PLACE_THREADS=1T KMP_AFFINITY=compact OMP_NUM_THREADS=8

コアあたり 2 スレッド
KMP_PLACE_THREADS=2T KMP_AFFINITY=compact OMP_NUM_THREADS=16

コアあたり 3 スレッド
KMP_PLACE_THREADS=3T KMP_AFFINITY=compact OMP_NUM_THREADS=24

コアあたり 4 スレッド
KMP_PLACE_THREADS=4T KMP_AFFINITY=compact OMP_NUM_THREADS=32

OpenMP* 4.0 のスレッド・アフィニティー

OpenMP* 4.0 では、OMP_PLACES 環境変数および OMP_PROC_BIND 環境変数によって制御されるアフィニティー設定が追加されました、OMP_PLACES は、ハードウェア・リソースを指定します。設定可能な値は cores または threads で、プレースのリストを表す抽象名またはプレースの明示的なリスト (非一般的) を指定できます。OMP_PROC_BIND は、OpenMP* スレッドとリソースのバインド方法を制御します。
OMP_PROC_BIND の一般的な値は close と spread です。

次の例は、OpenMP* によりスレッド化されたアプリケーションを、OpenMP* 4.0 のアフィニティーを使用してコアあたり 1 ～ 4 ハードウェア・スレッドで実行します。

タイルあたり 1 スレッド
OMP_PROC_BIND=spread OMP_PLACES=threads OMP_NUM_THREADS=34

コアあたり 1 スレッド
OMP_PROC_BIND=spread OMP_PLACES=threads OMP_NUM_THREADS=68

コアあたり 2 スレッド
OMP_PROC_BIND=spread OMP_PLACES=threads OMP_NUM_THREADS=136

コアあたり 3 スレッド
OMP_PROC_BIND=spread OMP_PLACES=threads OMP_NUM_THREADS=204

コアあたり 4 スレッド
OMP_PROC_BIND=close OMP_PLACES=threads

次の例では、OpenMP* 4.0 のアフィニティーを使用して、コアをアンダーサブスクライブします。

タイルあたり 1 スレッド
OMP_PROC_BIND=spread OMP_PLACES=”threads(32)” OMP_NUM_THREADS=4

コアあたり 1 スレッド
OMP_PROC_BIND=spread OMP_PLACES=”threads(32)” OMP_NUM_THREADS=8

コアあたり 2 スレッド
OMP_PROC_BIND=close OMP_PLACES=”cores(8)” OMP_NUM_THREADS=16

コアあたり 3 スレッド
OMP_PROC_BIND=close OMP_PLACES=”cores(8)” OMP_NUM_THREADS=24

コアあたり 4 スレッド
OMP_PROC_BIND=close OMP_PLACES=threads OMP_NUM_THREADS=32

OpenMP* 4.0 のアフィニティー設定を使用する入れ子のスレッド・アフィニティー

アプリケーションで複数のレベルの OpenMP* スレッドを使用する場合、OMP_PLACES と OMP_NUM_THREADS に追加の値を指定します。次の例は、コアあたり 1 ハードウェア・スレッドで入れ子のスレッドを実行します。2 つ以上のハードウェア・スレッドを使用する場合は、使用するハードウェア・スレッド数に応じて、OMP_NUM_THREADS の 2 つ目の値を 4、6、または 8 にします。

```
OMP_NESTED=1
OMP_MAX_ACTIVE_LEVELS=2
KMP_HOT_TEAMS=1
KMP_HOT_TEAMS_MAX_LEVEL=2
OMP_NUM_THREADS=34,2
OMP_PROC_BIND=spread,spread
OMP_PLACES=cores
```

インテル® MPI ライブラリーのアフィニティー

インテル® MPI ライブラリーのアフィニティーは、I_MPI_PIN_PROCESSOR_LIST 環境変数によって制御されます。設定可能な値は、論理プロセッサーの明示的なリストまたはキーワードによって定義されたプロセッサーのセットです。一般的なキーワードは、all、allcores、grain、shift です。

- all は、ハードウェア・スレッドを含むすべての論理プロセッサーを指定します。
- allcores は、物理コアを指定します。
- grain は、ピニングの粒度を指定します。
- shift は、ラウンドロビン方式のスケジュールの粒度を GRAIN 単位で指定します。

次の例は、タイルあたり 1 ランク、コアあたり 1、2、4 ランクで MPI 実行ファイルを実行します。

タイルあたり 1 ランク
```
mpirun -perhost 34 -env I_MPI_PIN_PROCESSOR_LIST all:shift=cache2
```

コアあたり 1 ランク
```
mpirun -perhost 68 -env I_MPI_PIN_PROCESSOR_LIST allcores
```

コアあたり 2 ランク
```
mpirun -perhost 136 -env I_MPI_PIN_PROCESSOR_LIST all:grain=2,shift=2
```

コアあたり 4 ランク
```
mpirun -perhost 272 -env I_MPI_PIN_PROCESSOR_LIST all
```

ヒント:

- I_MPI_DEBUG を 4 以上に設定すると、ランクと OS プロセッサーのマップを確認できます。
- インテル® MPI ライブラリーの cpuinfo ユーティリティーを利用して、OS プロセッサーと物理キャッシュのマップを確認できます。

インテル® MPI ライブラリーと OpenMP* の相互運用性

インテル® MPI ライブラリーと OpenMP* のアフィニティー設定を組み合わせてハイブリッド実行することができます。次の例では、OpenMP* ランタイム拡張または OpenMP* 4.0 のアフィニティーを使用して、すべてのコアを使って、コアあたり 1 ～ 4 スレッドで実行します。

インテル® Xeon Phi™ プロセッサー上で、インテルの OpenMP* ランタイム拡張を使用して、インテル® MPI ライブラリー/OpenMP* のアフィニティーを指定する例:

コアあたり 1 スレッド
mpirun -env KMP_PLACE_THREADS 1T -env KMP_AFFINITY compact

コアあたり 2 スレッド
mpirun -env KMP_PLACE_THREADS 2T -env KMP_AFFINITY compact

コアあたり 3 スレッド
mpirun -env KMP_PLACE_THREADS 3T -env KMP_AFFINITY compact

コアあたり 4 スレッド
mpirun -env KMP_PLACE_THREADS 4T -env KMP_AFFINITY compact

インテル® Xeon Phi™ プロセッサー上で、OpenMP* 4.0 のアフィニティーを使用して、インテル® MPI ライブラリー/OpenMP* のアフィニティーを指定する例:

タイルあたり 1 スレッド
mpirun -env OMP_PROC_BIND spread -env OMP_PLACES threads -env OMP_NUM_THREADS 8

コアあたり 1 スレッド
mpirun -env OMP_PROC_BIND spread -env OMP_PLACES threads -env OMP_NUM_THREADS 17

コアあたり 2 スレッド
mpirun -env OMP_PROC_BIND spread -env OMP_PLACES threads -env OMP_NUM_THREADS 34

コアあたり 4 スレッド
mpirun -env OMP_PROC_BIND close -env OMP_PLACES threads

インテル® MPI ライブラリーは、MPI ランクと OpenMP* スレッドを起動する実行ファイル向けに I_MPI_PIN_DOMAIN 環境変数も提供します。この環境変数は、論理プロセッサーのオーバーラップしないサブセットの数を定義し、1 つの MPI ランクを各ドメインにバインドします。明示的なドメインのバインドは、コアをアンダーサブスクライブする場合に役立ちます。次の例は、インテル® MPI ライブラリーと OpenMP* のハイブリッド実行ファイルをインテル® Xeon Phi™ プロセッサーの 68 コア未満で実行します。

コアあたり 1 スレッド、2 つのクワドラント
mpirun -perhost 2 -env I_MPI_PIN_DOMAIN 68 -env KMP_PLACE_THREADS 1T -env KMP_AFFINITY compact

12 ランク、タイルあたり 1 ランク、コアあたり 2 スレッド
mpirun -perhost 12 -env I_MPI_PIN_DOMAIN 8 -env KMP_PLACE_THREADS 2T -env KMP_AFFINITY compact

ヒント: I_MPI_PIN_DOMAIN を設定すると、I_MPI_PIN_PROCESSOR_LIST は無視されます。

今後の予定

インテル® Xeon Phi™ プロセッサー x200 製品ファミリーの NUMA ドメインでプロセスとスレッドの配置を容易にするため、2016 年および 2017 年にインテル® MPI ライブラリーとインテルの OpenMP* ランタイム拡張を拡張する予定です。

まとめ

インテル® MPI ライブラリーと OpenMP* ランタイム拡張は、MPI ランクと OpenMP* スレッドを特定のプロセッサーにバインドするメカニズムを提供します。この記事では、例を用いて異なるコア数とハードウェア・スレッド数で、インテル® Xeon Phi™ プロセッサー (開発コード名 Knights Landing) 上で実行する方法を示しました。ここで紹介した例に従って、アプリケーションに最適なコアあたりのハードウェア・スレッド数と、MPI ランクと OpenMP* スレッドの組み合わせを見つけることができるでしょう。

関連情報

『インテル® Fortran コンパイラー・デベロッパー・ガイドおよびリファレンス』の「スレッド・アフィニティー・インターフェイス」 https://software.intel.com/en-us/intel-fortran-compiler-17.0-user-and-reference-guide (英語)
『インテル® C++ コンパイラー・デベロッパー・ガイドおよびリファレンス』の「スレッド・アフィニティー・インターフェイス」 https://software.intel.com/en-us/intel-cplusplus-compiler-17.0-user-and-reference-guide (英語)
OpenMP* 4.0 仕様 http://www.openmp.org/specifications/ (英語)
『インテル® MPI ライブラリー for Linux* デベロッパー・リファレンス』の「プロセスピニング」 https://software.intel.com/en-us/mpi-developer-reference-linux (英語)
『インテル® MPI ライブラリー for Linux* デベロッパー・リファレンス』の「OpenMP* API との相互運用性」 https://software.intel.com/en-us/mpi-developer-reference-linux (英語)
「OpenMP* で入れ子の並列処理を使用する」 https://software.intel.com/en-us/videos/using-nested-parallelism-in-openmp (英語)
「MPI/OpenMP* のハイブリッド・プログラミングを始めよう」 https://www.isus.jp/products/mpi/hybrid-mpiopenmp-development/

コンパイラーの最適化に関する詳細は、最適化に関する注意事項を参照してください。