インテル® MPI ライブラリー: インテル® Xeon® プロセッサー、インテル® Xeon Phi™ コプロセッサー、インテル® Xeon Phi™ プロセッサー間の互換性

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「Intel® MPI Library: Compatibility among Intel® Xeon® Processors, Intel® Xeon Phi™ Coprocessors and Intel® Xeon Phi™ Processors」の日本語参考訳です。

---

1. はじめに

メッセージ・パッシング・インターフェイス (MPI) 並列プログラミングが分散メモリー型システムの並列コンピューティングの業界標準になるにつれて、インテル製品をサポートする上でインテル® MPI ライブラリーが果たす役割が大きくなっています。特定のインテル製品を利用する場合、ユーザーは使い方の違いを知っておく必要があります。

この記事の前半では、3 つの異なるマイクロアーキテクチャー (インテル® Xeon® プロセッサー、インテル® Xeon Phi™ コプロセッサー、インテル® Xeon Phi™ プロセッサー) でインテル® MPI ライブラリーを使用する際の類似点と相違点を簡単に説明します。

また記事の後半では、インテル® Xeon Phi™ プロセッサーでインテル® MPI ライブラリーを使用する上で役立つベスト・プラクティスを紹介します。

2. インテル® MPI ライブラリーを使用する際の類似点と相違点

現在のインテル® MPI ライブラリーは、MPI 3.0 仕様を実装しており、それぞれのインテル® プラットフォーム向けに最適化されています。ユーザーは、次の点を考慮して、ハードウェアに関係なくライブラリーを使用することができます。

コンパイラー・オプション

インテル® MPI ライブラリーを使用するには、適切な環境を設定し、mpiicc (C の場合)、mpiicpc (C++ の場合)、または mpiifort (Fortran の場合) でコンパイルする必要があります。

```
$ source <compilerinstalldir>/bin/compilervars.sh intel64
$ source <installdir>/bin64/mpivars.sh
$ mpiicc application.c
```

インテル® ハードウェアの優れたパフォーマンスを引き出すには、ハードウェア固有のオプションを指定してアプリケーションをコンパイルします。インテル® Xeon® プロセッサーでは、-xHost オプションを指定すると、コンパイルを行うホスト・プロセッサーで利用可能な最上位の命令セット向けのコードを生成できます。

```
$ mpiicc -xHost application.c
```

インテル® C/C++ コンパイラーは、インテル® Xeon Phi™ コプロセッサー上では起動できません。そのため、インテル® Xeon® プロセッサー上でインテル® Xeon Phi™ コプロセッサー向けにクロスコンパイルする場合、-mmic オプションを指定してアプリケーションをビルドします。

```
$ mpiicc -mmic application.c
```

また、インテル® Xeon® プロセッサー上でインテル® Xeon Phi™ プロセッサー向けにコンパイルする場合、-xMIC-AVX512 オプションを指定してアプリケーションをビルドします。

```
$ mpiicc -xMIC-AVX512 application.c
```

インテル® Xeon Phi™ プロセッサー上でインテル® Xeon Phi™ プロセッサー向けにコンパイルする場合も、-xMIC-AVX512 または -xHost オプションを指定します。-xHost オプションを指定すると、コンパイルを行うホスト・プロセッサーで利用可能な最上位の命令セット向けのコードが生成されます。そのため、-xHost オプションを使用してインテル® Xeon Phi™ プロセッサー上で生成したバイナリーは、インテル® Xeon® プロセッサー上では実行できません。

バイナリー互換

-xCOMMON-AVX512 オプションを指定してビルドした MPI プログラムは、インテル® アドバンスト・ベクトル・エクステンション 512 (インテル® AVX-512) 命令に対応したインテル® Xeon® プロセッサーおよびインテル® Xeon Phi™ プロセッサーで実行できます。インテル® Xeon® プロセッサーまたはインテル® Xeon Phi™ プロセッサー向けにビルドした MPI プログラムは、インテル® Xeon Phi™ コプロセッサーでは実行できません。逆も同様です。

プログラミング・モデル

インテル® Xeon® プロセッサー、インテル® Xeon Phi™ コプロセッサー、インテル® Xeon Phi™ プロセッサーは、MPI プログラムをネイティブに実行します。つまり、特定のプラットフォーム向けに MPI プログラムをビルドすると、生成された実行ファイルをそのプラットフォーム上で直接実行できます。

インテル® Xeon Phi™ コプロセッサーは、インテル® Xeon® プロセッサー・ホストに接続されるため、次のオプションがあります。

- オフロードモデル: MPI ランクはインテル® Xeon® プロセッサー・ホストで起動され、インテル® Xeon Phi™ コプロセッサーにワークをオフロードします。オフロードモデルは、オフロード・ディレクティブ、インテル® マス・カーネル・ライブラリー (インテル® MKL) の暗黙的オフロード、および明示的オフロードを使用して実装できます。
- シンメトリック・モデル: MPI ランクは、インテル® Xeon® プロセッサー・ホストとインテル® Xeon Phi™ コプロセッサーの両方に存在します。このモデルでは、インテル® Xeon® プロセッサーとインテル® Xeon Phi™ コプロセッサーはどちらもヘテロジニアス計算ノードです。

インテル® Xeon Phi™ コプロセッサー上で直接実行する場合、ホストからまたは直接コプロセッサーから実行ファイルを起動できます。ホストから起動する場合、I_MPI_MIC 環境変数を設定する必要があります。

```
$ export I_MPI_MIC=on
```

この環境変数設定は、インテル® Xeon Phi™ コプロセッサー向けに最適化された設定を選択するため、インテル® Xeon Phi™ コプロセッサー上でネイティブ起動する場合にも有効です。

スレッド化モデル

3 つのアーキテクチャーは、インテル® MKL、インテル® スレッディング・ビルディング・ブロック (インテル® TBB)、インテル® Cilk™ Plus、OpenMP*、Pthreads をサポートします。

インテル® Xeon® プロセッサー・ベースのシステムでは、ハイパースレッディング・テクノロジーが有効な場合、各物理コアは 2 つの論理コアと見なされます。一方、インテル® Xeon Phi™ コプロセッサーおよびインテル® Xeon Phi™ プロセッサーでは、利用可能なハードウェア・スレッド数は利用可能なコア数の 4 倍になります。

次の表は、相違点を要約したものです。

	インテル® Xeon® プロセッサー	**インテル® Xeon Phi™ コプロセッサー**	**インテル® Xeon Phi™ プロセッサー**
コンパイラー・オプション	-xHost	-mmic	-xHost または -xMIC-AVX512
インテル® Xeon® プロセッサーとの互換性		×	○
プログラミング・モデル	ネイティブ	ネイティブ、 オフロード、 シンメトリック	ネイティブ
スレッド化モデル	インテル® MKL、 インテル® TBB、 インテル® Cilk™ Plus、 OpenMP*、Pthreads	インテル® MKL、 インテル® TBB、 インテル® Cilk™ Plus、 OpenMP*、Pthreads	インテル® MKL、 インテル® TBB、 インテル® Cilk™ Plus、 OpenMP*、Pthreads
最大スレッド数	コア数の 2 倍	コア数の 4 倍	コア数の 4 倍
命令セット	インテル® ストリーミング SIMD 拡張命令 (インテル® SSE)、 MME、 インテル® AVX、 インテル® AVX2	インテル® イニシャル・メニー・コア命令 (インテル® IMCI)	インテル® SSE、 MME、 インテル® AVX、 インテル® AVX2、 インテル® AVX-512
メモリーの種類	DDR4	GDDR5	DDR4 および MCDRAM

3. インテル® Xeon Phi™ プロセッサーでの操作

インテル® Xeon Phi™ プロセッサーは、高帯域幅メモリー、コアをクラスター化する機能 (「クラスターモード」と呼ばれます)、新しい命令セット・アーキテクチャー (ISA) の命令をサポートしている点で、ほかの 2 つのアーキテクチャーと異なります。クラスターモードを活用するには、スレッド・アフィニティーとアプリケーションに適した変更方法を理解することが重要です。このセクションでは、インテル® Xeon Phi™ プロセッサー上で高帯域幅メモリーおよびアフィニティーを扱う際のベスト・プラクティスを紹介します。

3.1 インテル® Xeon Phi™ プロセッサーの高帯域幅メモリーの使用

高帯域幅メモリーの割り当て方法は、memkind ライブラリーを明示的に呼び出す、または AutoHBW を有効にするの 2 つです。インテル® MPI ライブラリーは、高帯域幅メモリーの割り当てに memkind ライブラリーを使用しません。memkind ライブラリーを使用する場合、MPI アプリケーション自身で高帯域幅メモリーを割り当てます。

あるいは、AutoHBW を使用することで、ソースコードを変更/再コンパイルすることなく、インテル® Xeon Phi™ プロセッサー・ベースのシステム上で高帯域幅メモリーを自動的に割り当てることができます。**「Jemalloc および Memkind と AutoBW ライブラリーの使用」** (英語) では、MPI プログラムで AutoHBW を使用するスクリプトの作成方法を説明しています。

次の例では、MCDRAM を搭載した 72 コアのインテル® Xeon Phi™ プロセッサーで "numactl -H" コマンドを実行すると、このマシンの 2 つの NUMA ノード (ノード 0 に 288 (72 x 4) の OS プロセッサー (CPU) と 96GB のメモリー、ノード 1 に 8GB の MCDRAM メモリー) が表示されています。AutoHBW を使用して、コードを変更する前に、MPI プログラムで MCDRAM の割り当てにより恩恵が受けられるかどうか確認することができます。

```
$ numactl -H
available: 2 nodes (0-1)
node 0 cpus: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52
53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79
80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104
105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124
125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144
145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164
165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184
185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204
205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224
225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244
245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264
265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284
285 286 287
node 0 size: 98200 MB
node 0 free: 92912 MB
node 1 cpus:
node 1 size: 8192 MB
node 1 free: 7901 MB
node distances:
node   0   1
  0:  10  31
  1:  31  10
```

3.2 インテル® Xeon Phi™ プロセッサーにおける OpenMP* アフィニティー

インテルの アフィニティー拡張または新しい OpenMP* 4.0 のスレッド・アフィニティー環境変数を使用して、インテル® Xeon Phi™ プロセッサー上でアフィニティーを設定することができます。

KMP アフィニティー拡張では、KMP_PLACE_THREADS 環境変数でスレッドの配置を制御し、使用するコア数とコアあたりのスレッド数を定義します。KMP_AFFINITY 環境変数には、スレッド・アフィニティーのタイプ (compact、scatter など) を指定します。インテル® Xeon Phi™ コプロセッサーと同様に、デフォルトは scatter です。

インテル® Xeon Phi™ プロセッサー・ベースのシステムでは、各物理コアに 4 つのハードウェア・スレッドがあるため、Linux* は各物理コアに 4 つの論理コアを割り当てます。インテル® Xeon Phi™ プロセッサーは、各物理コアの最初の論理コアの番号が 0 から n-1 (n は物理コアの数)、2 つ目の論理コアの番号が n から 2n-1、3 つ目の論理コアの番号が 2n から 3n-1、4 つ目の論理コアの番号が 3n から 4n-1 のように論理コアの番号が割り当てられることを想定していますが、インテル® Xeon Phi™ プロセッサーで動作するすべてのバージョンの Linux* には、ランダムな方法で OS プロセッサーを割り当てる不具合があります。そのため、特定の番号付けを仮定した方法は失敗します。この問題は、OpenMP* の明示的なアフィニティー制御を行う場合、特に重要です。

OS プロセッサーと物理コアのマップを特定するには、/proc/cpuinfo ファイルを確認します。

```
$ cat /proc/cpuinfo
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 87
model name      : 06/57
stepping        : 0
microcode       : 0xffff002d
cpu MHz         : 1200.000
cache size      : 1024 KB
physical id     : 0
siblings        : 288
core id         : 0
cpu cores       : 72
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 13
wp              : yes
<以下略>
```

ファイルの各項目にはプロセッサーの番号と対応するコアの番号が含まれています。"processor" は論理コアの番号を示し、"core id" は物理コアの番号を示します。このファイルを利用して、すべての論理プロセッサーとマップしているコアを取得する方法を次に示します。

```
$ grep &apos;processor\|core id&apos; /proc/cpuinfo
```

出力から、すべてのコアとマップしているプロセッサーを簡単に特定できます。例えば、物理コア 0 には 4 つの論理プロセッサー 0、109、169、229 があります。

core: 0 procs: 0 109 169 229
core: 1 procs: 50 110 170 230
core: 2 procs: 51 111 171 231
core: 3 procs: 52 112 172 232
core: 4 procs: 53 113 173 233
core: 5 procs: 54 114 174 234
core: 6 procs: 55 115 175 235
core: 7 procs: 56 116 176 236
core: 8 procs: 57 117 177 237
core: 9 procs: 58 118 178 238

……

また、KMP_AFFINITY 環境変数に verbose を設定することで、スレッドと OS プロセッサーのマップを確認することができます。

コアあたり 1 スレッド: 72 コアのインテル® Xeon Phi™ プロセッサーにおいて次のコマンドは、72 コアすべて、最初の 36 コア、および最初の 18 コアを使用してそれぞれコアあたり 1 スレッドで、OpenMP* プログラム "a.out" を実行します。

```
$ KMP_PLACE_THREADS=72C,1T KMP_AFFINITY=compact,verbose ./a.out
$ KMP_PLACE_THREADS=36C,1T KMP_AFFINITY=compact,verbose ./a.out
$ KMP_PLACE_THREADS=18C,1T KMP_AFFINITY=compact,verbose ./a.out
```

コアあたり 2 スレッド: 次のコマンドは、72 コアすべて、最初の 36 コア、および最初の 18 コアを使用してそれぞれコアあたり 2 スレッドで、OpenMP* プログラム "a.out" を実行します。

```
$ KMP_PLACE_THREADS=72C,2T KMP_AFFINITY=compact,verbose ./a.out
$ KMP_PLACE_THREADS=36C,2T KMP_AFFINITY=compact,verbose ./a.out
$ KMP_PLACE_THREADS=18C,2T KMP_AFFINITY=compact,verbose ./a.out
```

コアあたり 4 スレッド: 次のコマンドは、72 コアすべて、最初の 36 コア、および最初の 18 コアを使用してそれぞれコアあたり 4 スレッドで、OpenMP* プログラム "a.out" を実行します。

```
$ KMP_PLACE_THREADS=72C,4T KMP_AFFINITY=compact,verbose ./a.out
$ KMP_PLACE_THREADS=36C,4T KMP_AFFINITY=compact,verbose ./a.out
$ KMP_PLACE_THREADS=18C,4T KMP_AFFINITY=compact,verbose ./a.out
```

インテル® Xeon Phi™ プロセッサーのコア数を *n* とします。次の表は、KMP_PLACE_THREADS および KMP_AFFINITY 環境変数を使用して、すべてのコア、n/2 コア、n/4 コアで、コアあたり 1 スレッド、2 スレッド、および 4 スレッドを生成する方法を要約したものです。

	n コアすべてを使用	*n/2* コアのみ使用	*n/4* コアのみ使用
コアあたり 1 スレッド	`KMP_PLACE_THREADS="n"C,1T KMP_AFFINITY=compact`	`KMP_PLACE_THREADS="n/2"C,1T KMP_AFFINITY=compact`	`KMP_PLACE_THREADS="n/4"C,1T KMP_AFFINITY=compact`
コアあたり 2 スレッド	`KMP_PLACE_THREADS="n"C,2T KMP_AFFINITY=compact`	`KMP_PLACE_THREADS="n/2"C,2T KMP_AFFINITY=compact`	`KMP_PLACE_THREADS="n/4"C,2T KMP_AFFINITY=compact`
コアあたり 4 スレッド	`KMP_PLACE_THREADS="n"C,4T KMP_AFFINITY=compact`	`KMP_PLACE_THREADS="n/2"C,4T KMP_AFFINITY=compact`	`KMP_PLACE_THREADS="n/4"C,3T KMP_AFFINITY=compact`

アフィニティーを設定する 2 つ目の方法は、OpenMP* 4.0 で定義されている OMP_PLACES 環境変数を使用することです。この環境変数を使用して、論理スレッドをプロセッサー・スレッド、コアまたはソケットにピニングできます。OMP_NUM_THREADS 環境変数と組み合わせると、スレッドを生成して、生成したスレッドを OS プロセッサーにピニングできます。

タイルあたり 1 スレッド: 72 コアのインテル® Xeon Phi™ プロセッサーにおいて次のコマンドは、タイルあたり 1 スレッドで OpenMP* プログラム "a.out" を実行します。

使用するコア数を指定する代わりに、OpenMP* スレッドに割り当てるプロセッサー・スレッド数を指定します。インテル® Xeon Phi™ プロセッサーは、それぞれ 2 つのコアを含むタイルで構成されます (各コアには 4 つのスレッドがあります)。そのため、36 の OpenMP* スレッドを 288 のプロセッサー・スレッド 、18 の OpenMP* スレッドを 144 のプロセッサー・スレッド、9 つの OpenMP* スレッドを 72 のプロセッサー・スレッドに割り当てて、それぞれタイルあたり 1 OpenMP* スレッドでプログラム "a.out" を実行するコマンドは次のようになります。

```
$ OMP_PLACES="threads(288)" OMP_NUM_THREADS=36 ./a.out
$ OMP_PLACES="threads(144)" OMP_NUM_THREADS=18 ./a.out
$ OMP_PLACES="threads(72)" OMP_NUM_THREADS=9 ./a.out
```

コアあたり 1 スレッド: 次のコマンドは、288 の OS プロセッサー (72 コア)、144 の OS プロセッサー (36 コア)、72 の OS プロセッサー (18 コア) を使用して、それぞれコアあたり 1 スレッドで OpenMP* プログラム "a.out" を実行します。

```
$ OMP_PLACES="threads(288)" OMP_NUM_THREADS=72 ./a.out
$ OMP_PLACES="threads(144)" OMP_NUM_THREADS=36 ./a.out
$ OMP_PLACES="threads(72)" OMP_NUM_THREADS=18 ./a.out
```

コアあたり 2 スレッド: 次のコマンドは、288 の OS プロセッサー (72 コア)、144 の OS プロセッサー (36 コア)、72 の OS プロセッサー (18 コア) を使用して、それぞれコアあたり 2 スレッドで OpenMP* プログラム "a.out" を実行します。

```
$ OMP_PLACES="threads(288)" OMP_NUM_THREADS=144 ./a.out
$ OMP_PLACES="threads(144)" OMP_NUM_THREADS=72 ./a.out
$ OMP_PLACES="threads(72)" OMP_NUM_THREADS=36 ./a.out
```

コアあたり 4 スレッド: 次のコマンドは、288 の OS プロセッサー (72 コア)、144 の OS プロセッサー (36 コア)、72 の OS プロセッサー (18 コア) を使用して、それぞれコアあたり 4 スレッドで OpenMP* プログラム "a.out" を実行します。

```
$ OMP_PLACES="threads(288)" OMP_NUM_THREADS=288 ./a.out
$ OMP_PLACES="threads(144)" OMP_NUM_THREADS=144 ./a.out
$ OMP_PLACES="threads(72)" OMP_NUM_THREADS=72 ./a.out
```

OMP_PROC_BIND 環境変数を使用して、スレッドをプロセッサー間で移動できるかどうか、およびスレッド・アフィニティー・ポリシーを指定することもできます。この環境変数を TRUE に設定すると、スレッドはコア間で移動されません。デフォルトでは TRUE に設定されています。スレッド・アフィニティー・ポリシーは、パラメーター MASTER (マスタースレッドと同じパーティション)、CLOSE (連続するパーティション)、SPREAD (パーティション間で分割) で指定します。

```
$ OMP_PROC_BIND=close OMP_NUM_THREADS=288 ./a.out
$ OMP_PROC_BIND=close OMP_NUM_THREADS=144 ./a.out
$ OMP_PROC_BIND=close OMP_NUM_THREADS=72 ./a.out
```

インテル® Xeon Phi™ プロセッサーのコア数を *n* とします。次の表は、OMP_PLACES および OMP_NUM_THREADS 環境変数を使用して、すべてのコア、n/2 コア、n/4 コアで、コアあたり 1 スレッド、コアあたり 2 スレッド、コアあたり 4 スレッドを作成する方法を要約したものです。

	n コアすべてを使用	*n/2* コアのみ使用	*n/4* コアのみ使用
タイルあたり 1 スレッド	`OMP_PLACES="threads(n)" OMP_NUM_THREADS=n/8`	`OMP_PLACES="threads(n/2)" OMP_NUM_THREADS=n/16`	`OMP_PLACES="threads(n/4)" OMP_NUM_THREADS=n/32`
コアあたり 1 スレッド	`OMP_PLACES="threads(n)" OMP_NUM_THREADS=n/4`	`OMP_PLACES="threads(n/2)" OMP_NUM_THREADS=n/8`	`OMP_PLACES="threads(n/4)" OMP_NUM_THREADS=n/16`
コアあたり 2 スレッド	`OMP_PLACES="threads(n)" OMP_NUM_THREADS=n/2`	`OMP_PLACES="threads(n/2)" OMP_NUM_THREADS=n/4`	`OMP_PLACES="threads(n/4)" OMP_NUM_THREADS=n/8`
コアあたり 4 スレッド	`OMP_PLACES="threads(n)" OMP_NUM_THREADS=n`	`OMP_PLACES="threads(n/2)" OMP_NUM_THREADS=n/2`	`OMP_PLACES="threads(n/4)" OMP_NUM_THREADS=n/4`

3.3 インテル® Xeon Phi™ プロセッサーにおけるインテル® MPI ライブラリーのアフィニティー

サンプルプログラム <installdir>/test/test.c をコンパイルして実行します。

```
$ mpiicc -xMIC-AVX512 test.c
$ mpirun -n 4 -env I_MPI_DEBUG 4 ./a.out
```

MPI ランクと CPU (OS プロセッサー) のマップを表示するには、I_MPI_DEBUG 環境変数を 4 以上に設定します。72 コアのインテル® Xeon Phi™ プロセッサー・ベースのマシンでは、各コアに 4 つのハードウェア・スレッドがあるため、CPU の総数は 72 x 4 = 288 です。上記のプログラムは 4 つの MPI ランクを起動するため、各ランクは 288 / 4 = 72 CPU にピニングされます。MPI ランクのマップを次に示します。

```
[0] MPI startup(): Multi-threaded optimized library
[0] MPI startup(): Rank    Pid      Node name   Pin cpu
[0] MPI startup(): 0       244756   knl4
{0,50,51,52,53,54,55,56,57,58,59,60,61,62,63,64,65,66,109,110,111,112,113,114,115
,116,117,118,119,120,121,122,123,124,125,126,169,170,171,172,173,174,175,176,177
,178,179,180,181,182,183,184,185,186,229,230,231,232,233,234,235,236,237,238,239
,240,241,242,243,244,245,246}
[0] MPI startup(): 1       244757   knl4
{67,68,69,70,71,72,73,74,75,76,77,78,79,80,81,82,83,84,127,128,129,130,131,132,13
3
,134,135,136,137,138,139,140,141,142,143,144,187,188,189,190,191,192,193,194,195
,196,197,198,199,200,201,202,203,204,247,248,249,250,251,252,253,254,255,256,257
,258,259,260,261,262,263,264}
[0] MPI startup(): 2       244758   knl4
{85,86,87,88,89,90,91,92,93,94,95,96,97,98,99,100,101,102,145,146,147,148,149,150
,151,152,153,154,155,156,157,158,159,160,161,162,205,206,207,208,209,210,211,212
,213,214,215,216,217,218,219,220,221,222,265,266,267,268,269,270,271,272,273,274
,275,276,277,278,279,280,281,282}
[0] MPI startup(): 3       244759   knl4
{1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30
,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,103,104,105,106,107,108
,163,164,165,166,167,168,223,224,225,226,227,228,283,284,285,286,287}
```

タイルあたり 1 MPI ランク: 72 コアのインテル® Xeon Phi™ プロセッサー・ベースのシステムで 36 の MPI ランクを起動した場合、各 MPI ランクは 8 つ (72 x 4 / 36) の OS プロセッサーにマップされます。次の出力は、最初のランクが 8 つの OS プロセッサー (2 コア) {0,50,109,110,169,170,229,230}、2 つ目のランクが次の 8 つの OS プロセッサー {51,52,111,112,171,172,231,232}、... 以下同様 ... にマップされることを示しています。各 MPI ランクはタイルの 2 つのコア間で移動できます。

```
$ mpirun -n 36 -env I_MPI_DEBUG 4 ./a.out
[0] MPI startup(): Rank    Pid      Node name   Pin cpu
[0] MPI startup(): 0       245851   knl4        {0,50,109,110,169,170,229,230}
[0] MPI startup(): 1       245852   knl4        {51,52,111,112,171,172,231,232}
[0] MPI startup(): 2       245853   knl4        {53,54,113,114,173,174,233,234}
[0] MPI startup(): 3       245854   knl4        {55,56,115,116,175,176,235,236}
[0] MPI startup(): 4       245855   knl4        {57,58,117,118,177,178,237,238}
[0] MPI startup(): 5       245856   knl4        {59,60,119,120,179,180,239,240}
[0] MPI startup(): 6       245857   knl4        {61,62,121,122,181,182,241,242}
[0] MPI startup(): 7       245858   knl4        {63,64,123,124,183,184,243,244}
<以下略>
```

コアあたり 1 ランク (タイルあたり 2 ランク): 72 コアの同じソケットで 72 の MPI ランクを起動した場合、各 MPI ランクは 4 つ (72 x 4 / 72) の OS プロセッサーにマップされます。次の出力は、最初のランクが 4 つの OS プロセッサー {0,109,169,229}、2 つ目のランクが次の 4 つの OS プロセッサー {50,110,170,230}、... 以下同様 ... にマップされることを示しています (以下同様)。

```
$ mpirun -n 72 -env I_MPI_DEBUG 4 ./a.out
[0] MPI startup(): Rank    Pid      Node name   Pin cpu
[0] MPI startup(): 0       245953   knl4        {0,109,169,229}
[0] MPI startup(): 1       245954   knl4        {50,110,170,230}
[0] MPI startup(): 2       245955   knl4        {51,111,171,231}
[0] MPI startup(): 3       245956   knl4        {52,112,172,232}
[0] MPI startup(): 4       245957   knl4        {53,113,173,233}
[0] MPI startup(): 5       245958   knl4        {54,114,174,234}
[0] MPI startup(): 6       245959   knl4        {55,115,175,235}
[0] MPI startup(): 7       245960   knl4        {56,116,176,236}
<以下略>
```

コアあたり 2 ランク (タイルあたり 4 ランク): 144 の MPI ランクを起動した場合、各ランクは 2 つ (72 x 4 / 144) の OS プロセッサーにマップされます。

```
$ mpirun -n 144 -env I_MPI_DEBUG 4 ./a.out
[0] MPI startup(): Rank    Pid      Node name   Pin cpu
[0] MPI startup(): 0       247281   knl4        {0,109}
[0] MPI startup(): 1       247282   knl4        {169,229}
[0] MPI startup(): 2       247283   knl4        {50,110}
[0] MPI startup(): 3       247284   knl4        {170,230}
[0] MPI startup(): 4       247285   knl4        {51,111}
[0] MPI startup(): 5       247286   knl4        {171,231}
[0] MPI startup(): 6       247287   knl4        {52,112}
[0] MPI startup(): 7       247288   knl4        {172,232}
<以下略>
```

コアあたり 4 ランク (タイルあたり 8 ランク): 288 の MPI ランクを起動した場合、各ランクは 1 つ (72 x 4 /288) の OS プロセッサーにマップされます。

```
$ mpirun -n 288 -env I_MPI_DEBUG 4 ./a.out
[0] MPI startup(): Rank    Pid      Node name   Pin cpu
[0] MPI startup(): 0       247918   knl4        0
[0] MPI startup(): 1       247919   knl4        109
[0] MPI startup(): 2       247920   knl4        169
[0] MPI startup(): 3       247921   knl4        229
[0] MPI startup(): 4       247922   knl4        50
[0] MPI startup(): 5       247923   knl4        110
[0] MPI startup(): 6       247924   knl4        170
[0] MPI startup(): 7       247925   knl4        230
<以下略>
```

次の表は、流動性を要約したものです。インテル® Xeon Phi™ プロセッサー・ベースのシステムのコア数を *k* とします。各ランクは OS プロセッサー間を移動できます。

	流動性
タイルあたり 1 ランク	`n = k / 2`
コアあたり 1 ランク	`n = k`
コアあたり 2 ランク	`n = k * 2`
コアあたり 4 ランク	`n = k * 4`

各 MPI ランクを特定の OS プロセッサーにバインドするには、I_MPI_PIN_PROCESSOR_LIST 環境変数を使用します。I_MPI_PIN_PROCESSOR_LIST 環境変数は、ランク数がコア数未満の場合に設定すると良いでしょう。

ピニングを制御する方法は 2 つあり、I_MPI_PIN_DOMAIN または I_MPI_PIN_PROCESSOR_LIST を使用します。

- I_MPI_PIN_DOMAIN 環境変数が定義されている場合、I_MPI_PIN_PROCESSOR_LIST 環境変数は無視されます。
- I_MPI_PIN_DOMAIN 環境変数が定義されていない場合、MPI ランクは I_MPI_PIN_PROCESSOR_LIST 環境変数に従ってピニングされます。

インテル® MPI ライブラリーは、プロセスピニングを制御する I_MPI_PIN_DOMAIN 環境変数を設定して、ノード上の論理プロセッサーのオーバーラップしないサブセット (ドメイン) の数を定義します。各 MPI ランクはドメインにピニングされます。各 MPI ランクは対応するドメイン内で実行する子スレッドを作成できます。

プロセスピニング環境変数の値を指定しない場合、デフォルトは I_MPI_PIN_DOMAIN=auto:compact です。auto は、#cpu / #rank (#cpu は論理プロセッサー数、#rank は MPI ランク数) のドメインサイズを指定します。compact は、ドメインメンバーが共通リソース (コア、キャッシュ、ソケットなど) にできるだけ近い位置に配置されるようにします。

タイルあたり 1 ランクで各ランクを OS プロセッサーにピニング: I_MPI_PIN_PROCESSOR_LIST 環境変数を all:map=scatter に設定します。

```
$ mpirun -n 36 -env I_MPI_PIN_PROCESSOR_LIST all:map=scatter -env \
I_MPI_DEBUG 4 ./a.out
[0] MPI startup(): Rank    Pid      Node name   Pin cpu
[0] MPI startup(): 0       42345    knl4        0
[0] MPI startup(): 1       42346    knl4        51
[0] MPI startup(): 2       42347    knl4        53
[0] MPI startup(): 3       42348    knl4        55
<以下略>
```

コアあたり 1 ランク (タイルあたり 2 ランク) で各ランクを OS プロセッサーにピニング:
I_MPI_PIN_PROCESSOR_LIST を all:shift=4 に設定します。

```
$ mpirun –n 72 -env I_MPI_PIN_PROCESSOR_LIST all:shift=4 -env I_MPI_DEBUG
4 ./a.out
[0] MPI startup(): Rank    Pid      Node name   Pin cpu
[0] MPI startup(): 0       42622    knl4        0
[0] MPI startup(): 1       42623    knl4        50
[0] MPI startup(): 2       42624    knl4        51
[0] MPI startup(): 3       42625    knl4        52
<以下略>
```

コアあたり 2 ランク (タイルあたり 4 ランク) で各ランクを OS プロセッサーにピニング:
I_MPI_PIN_PROCESSOR_LIST を all:grain=2,shift=2 に設定します。

```
$ mpirun –n 144 -env I_MPI_PIN_PROCESSOR_LIST all:grain=2,shift=2 -env
I_MPI_DEBUG 4 \ ./a.out
[0] MPI startup(): Rank    Pid      Node name   Pin cpu
[0] MPI startup(): 0       43082    knl4        0
[0] MPI startup(): 1       43083    knl4        109
[0] MPI startup(): 2       43084    knl4        50
[0] MPI startup(): 3       43085    knl4        110
<以下略>
```

コアあたり 4 ランク (タイルあたり 8 ランク) で各ランクを OS プロセッサーにピニング:
I_MPI_PIN_PROCESSOR_LIST を all:map=bunch に設定します。

```
$ mpirun –n 288 -env I_MPI_PIN_PROCESSOR_LIST all:map=bunch -env I_MPI_DEBUG 4 \
./a.out
[0] MPI startup(): Rank    Pid      Node name   Pin cpu
[0] MPI startup(): 0       276413   knl4        0
[0] MPI startup(): 1       276414   knl4        109
[0] MPI startup(): 2       276415   knl4        169
[0] MPI startup(): 3       276416   knl4        229
[0] MPI startup(): 4       276417   knl4        50
[0] MPI startup(): 5       276418   knl4        110
[0] MPI startup(): 6       276419   knl4        170
[0] MPI startup(): 7       276420   knl4        230
<以下略>
```

次の表は、インテル® Xeon Phi™ プロセッサーで MPI ランクを起動するパラメーターを要約したものです。このシステムのコア数を k とします。各コアを OS プロセッサーにピニングします。

	OS プロセッサーにピニング
タイルあたり 1 ランク	`n = k / 2` `I_MPI_PIN_PROCESSOR_LIST all:map=scatter`
コアあたり 1 ランク	`n = k` `I_MPI_PIN_PROCESSOR_LIST all:shift=4`
コアあたり 2 ランク	`n = k * 2` `I_MPI_PIN_PROCESSOR_LIST all:grain=2,shift=2`
コアあたり 4 ランク	`n = k * 4` `I_MPI_PIN_PROCESSOR_LIST all:map=bunch`

3.4 インテル® Xeon Phi™ プロセッサーにおけるハイブリッド MPI + OpenMP* アフィニティー

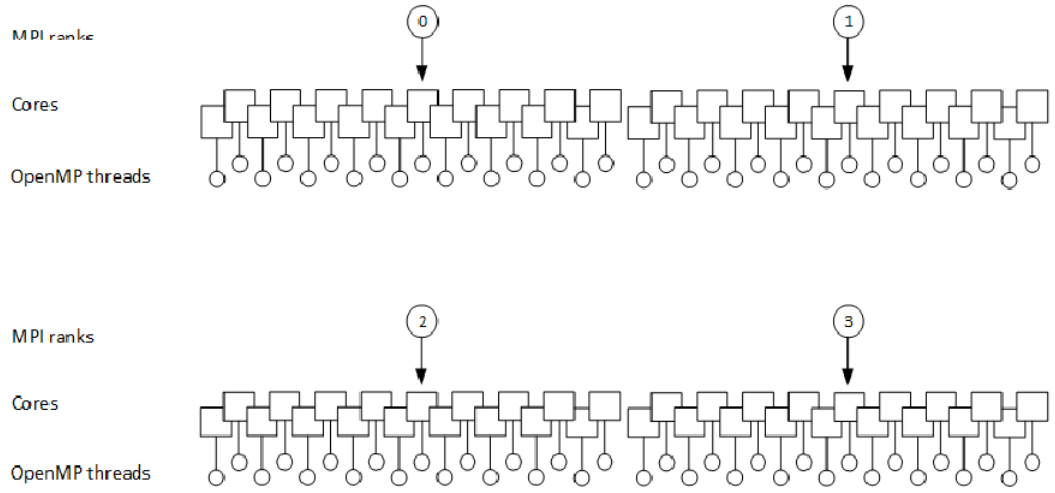
上記のアフィニティー制御の手法を組み合わせて、インテル® Xeon Phi™ プロセッサー上でハイブリッド MPI + OpenMP* アフィニティーを利用できます。スレッドと OS プロセッサーのマップを確認するには、KMP_AFFINITY 環境変数を verbose に設定します。

次の例では、72 コアのインテル® Xeon Phi™ プロセッサーで 4 つの MPI ランクを起動します。各ランクには OpenMP* スレッドのチームがあり、各スレッドを OS プロセッサーにピニングします。

この方法は 2 つあり、KMP_AFFINITY 環境変数を使用するか、OpenMP* 4.0 の OMP_PLACES 環境変数を使用して行います。

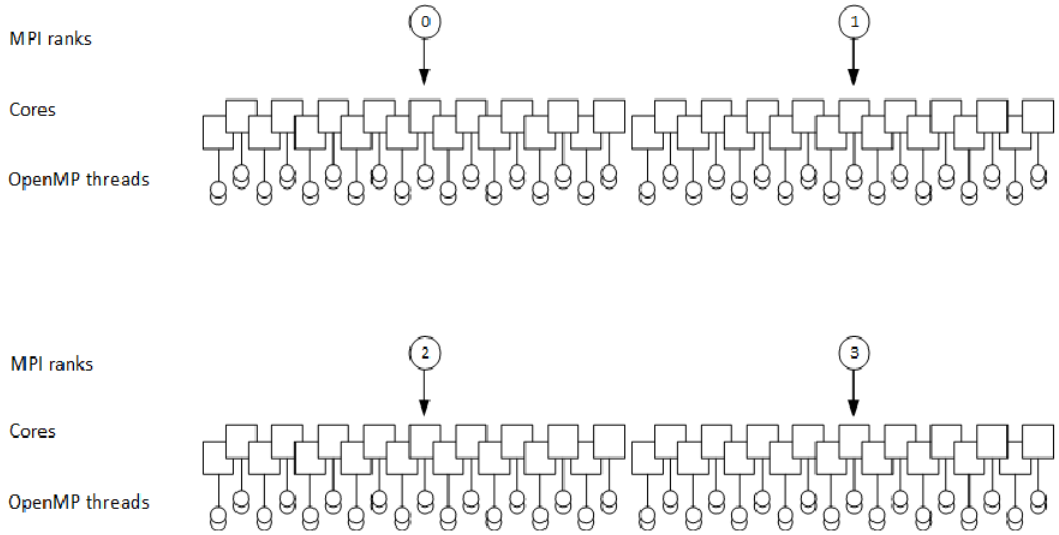
コアあたり 1 スレッド: 72 コア、4 ランク、各ランクを 72 / 4 = 18 コアにマップ。環境変数 KMP_PLACE_THREADS=18C,1T は、18 のコアを使用し、コアあたり 1 スレッドであることを示します。

```
$ mpirun -n 4 -env KMP_PLACE_THREADS=18C,1T ./a.out
```



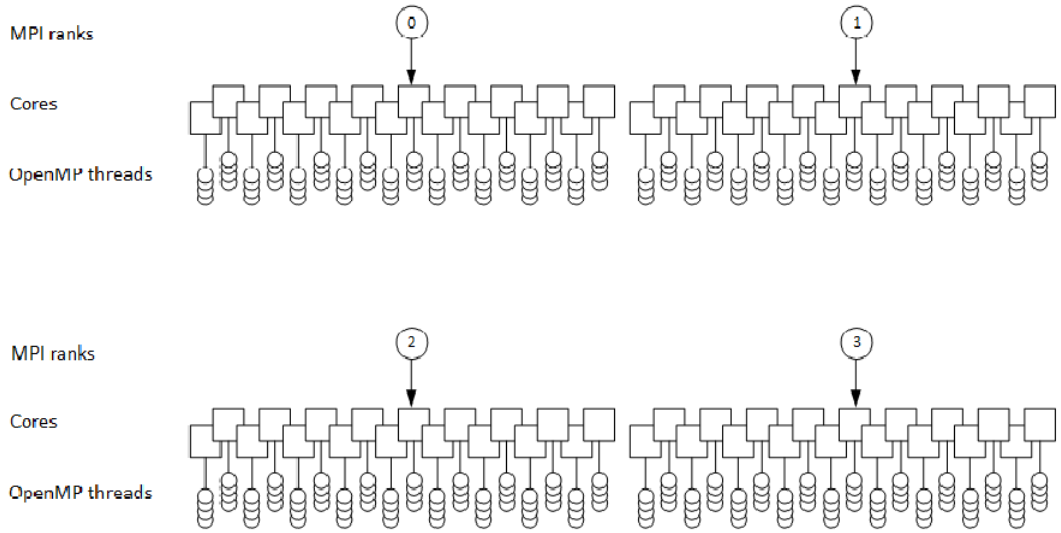
コアあたり 2 スレッド

```
$ mpirun -n 4 -env KMP_PLACE_THREADS=18C,2T ./a.out
```



コアあたり 4 スレッド

```
$ mpirun -n 4 -env KMP_PLACE_THREADS=18C,4T ./a.out
```



インテル® Xeon Phi™ プロセッサーのコア数を *n* とします。次の表は、各ランクが n/4 スレッド (コアあたり 1 スレッド)、n/2 スレッド (コアあたり 2 スレッド)、n スレッド (コアあたり 4 スレッド) の場合に 4 つの MPI ランクを起動する方法を要約したものです。

	4 つの MPI ランクを起動
コアあたり 1 スレッド	KMP_PLACE_THREADS="n/4"C,1T
コアあたり 2 スレッド	KMP_PLACE_THREADS="n/4"C,2T
コアあたり 4 スレッド	KMP_PLACE_THREADS="n/4"C,4T

OpenMP* 4.0 の OMP_PLACES 環境変数を使用することもできます。

タイルあたり 1 スレッド: 72 コア、4 ランク、各ランクを 72 / 4 = 18 コアにマップ。環境変数 OMP_NUM_THREADS は各ランクに 9 つのスレッドがあることを示し、環境変数 OMP_PLACES threads(72) はこれらのスレッドが対応する 72 の OS プロセッサー (18 x 4) に配置されることを示します。

```
$ mpirun -n 4 -env OMP_PLACES threads(72) -env OMP_NUM_THREADS 9 ./a.out
```

コアあたり 1 スレッド: 72 コア、4 ランク、各ランクを 72 / 4 = 18 コアにマップ。環境変数 OMP_NUM_THREADS は各ランクに 18 のスレッドがあることを示し、環境変数 OMP_PLACES threads(72) はこれらのスレッドが 72 の OS プロセッサー (ハードウェア・スレッド) に配置されることを示します。

```
$ mpirun -n 4 -env OMP_PLACES threads(72) -env OMP_NUM_THREADS 18 ./a.out
```

コアあたり 2 スレッド

```
$ mpirun -n 4 -env OMP_PLACES threads(72) -env OMP_NUM_THREADS 36 ./a.out
```

コアあたり 4 スレッド

```
$ mpirun -n 4 -env OMP_PLACES threads(72) -env OMP_NUM_THREADS 72 ./a.out
```

インテル® Xeon Phi™ プロセッサーのコア数を *n* とします。次の表は、各ランクが n/4 スレッド (コアあたり 1 スレッド)、n/2 スレッド (コアあたり 2 スレッド)、n スレッド (コアあたり 4 スレッド) の場合に 4 つの MPI ランクを起動する方法を要約したものです。

	4 つの MPI ランクを起動
コアあたり 1 スレッド	`OMP_PLACES="threads(n)"` `OMP_NUM_THREADS=n/4`
コアあたり 2 スレッド	`OMP_PLACES="threads(n)"` `OMP_NUM_THREADS=n/2`
コアあたり 4 スレッド	`OMP_PLACES="threads(n)"` `OMP_NUM_THREADS=n`

I_MPI_PIN_DOMAIN 環境変数を使用して、各 MPI ランクがサブセットにピニングされる論理プロセッサー数を定義することもできます。次の例は、72 コアのシステムの 2 つのクワドラントで実行している 2 つの MPI ランクを示しています。各 MPI ランクは 18 の OpenMP* スレッドのチーム (コアあたり 1 スレッド) を作成します。

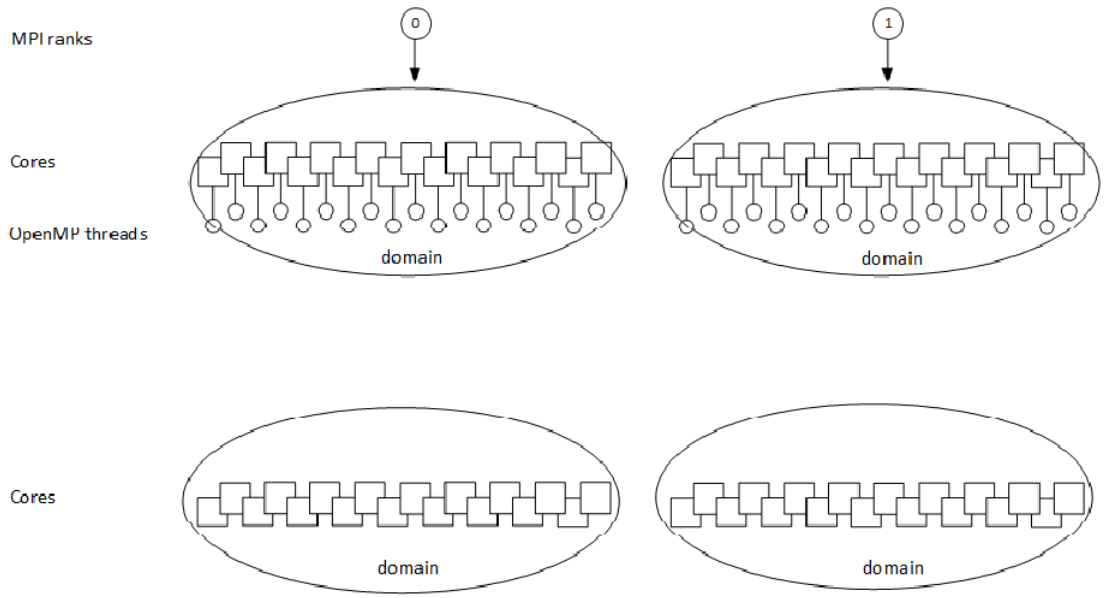
コアあたり 1 スレッド、2 つのクワドラント: 各ドメインは 72 の OS プロセッサーまたは 72 / 4 = 18 コアで構成されます。

```
$ mpirun -n 2 -env I_MPI_PIN_DOMAIN 72 -env KMP_PLACE_THREADS 18C,1T ./a.out
```

または

```
$ mpirun -n 2 -env I_MPI_PIN_DOMAIN 72 -env OMP_PLACES "threads(72)" \ -env
OMP_NUM_THREADS 18 ./a.out
```

インテル® Xeon Phi™ プロセッサー・ベースのシステムのコア数を *k* とします。4 つの MPI ランクで開始すると、MPI ランクおよび OpenMP* スレッドは次のように分散されます。



4. まとめ

この記事の前半では、インテル® Xeon® プロセッサー、インテル® Xeon Phi™ コプロセッサー、インテル® Xeon Phi™ プロセッサーでインテル® MPI ライブラリーを使用する場合の類似点と相違点を説明しました。この記事の後半では、インテル® Xeon Phi™ プロセッサーでの操作 (高帯域幅メモリーの使用、OpenMP* スレッド・アフィニティーの設定、MPI プロセス・アフィニティーの設定、ハイブリッド MPI + OpenMP* アフィニティーの設定) を行う際に役立つベスト・プラクティスを紹介しました。

5. 参考文献

- インテル® MPI ライブラリー – ドキュメント
- OpenMP* 仕様 (英語)
- OpenMP* スレッド・アフィニティーの制御
- Jemalloc および Memkind と AutoBW ライブラリーの使用 (英語)