インテル® Xeon Phi™ プロセッサーでの MPI-3 共有メモリーの使用法

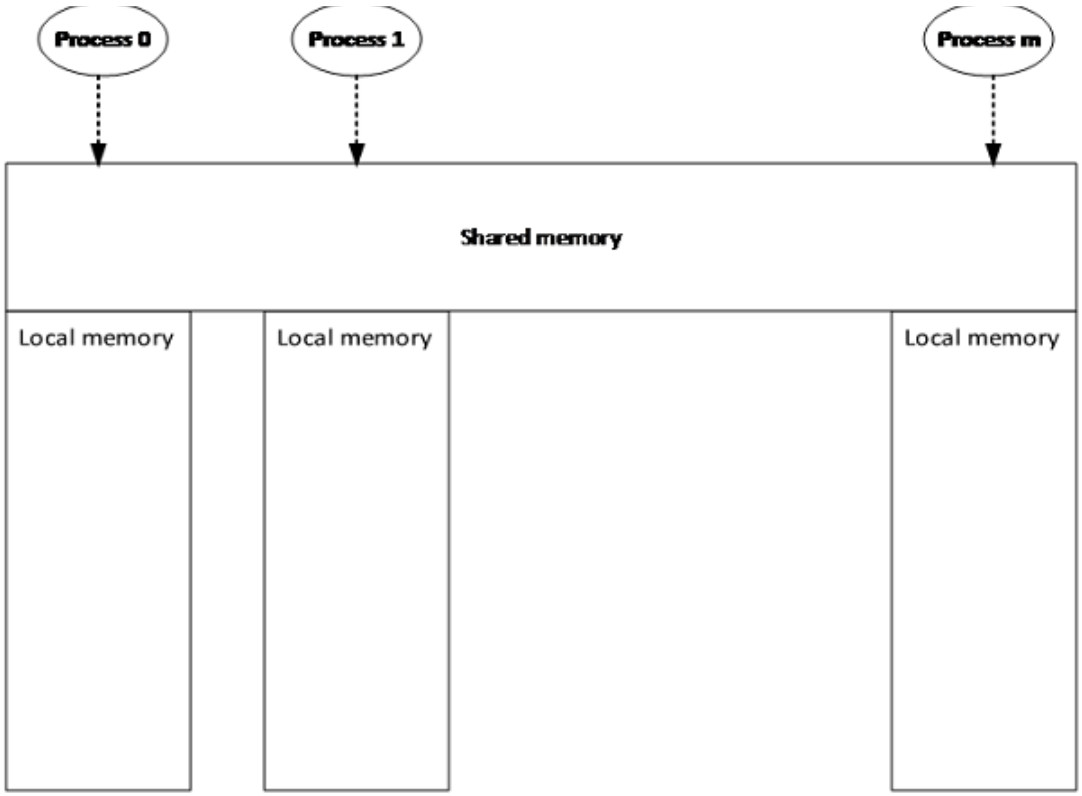
この記事は、インテル® デベロッパー・ゾーンに公開されている「How to use the MPI-3 Shared Memory in Intel® Xeon Phi™ Processors」の日本語参考訳です。

この記事では、MPI-3 共有メモリー機能、対応する API、およびインテル® Xeon Phi™ プロセッサーでの MPI-3 共有メモリーの使用法を示すサンプルプログラムを紹介します。

MPI-3 共有メモリーについて

MPI-3 共有メモリーは、メッセージ・パッシング・インターフェイス (MPI) 標準 3.0 で追加された機能で、インテル® MPI ライブラリー 5.0.2 以降に実装されています。MPI-3 共有メモリーは、複数の MPI プロセスが計算ノードで共有メモリーを割り当ておよび利用できるようにします。複数の MPI プロセスが大きなローカルデータをやり取りするアプリケーションでは、この機能によりメモリー・フットプリントを軽減し、パフォーマンスを大幅に向上することができます。

MPI 標準では、各 MPI プロセスは個別のアドレス空間を持っていますが、MPI-3 共有メモリーを利用することで、各 MPI プロセスは個々のメモリーをほかのプロセスが利用できるようにします。次の図は、共有メモリーの概念を図解したものです。各 MPI プロセスは、個別のローカルメモリーを割り当てそして維持し、その一部を共有メモリー領域にしてすべてのプロセスがアクセスできるようになります。共有メモリー機能を利用することで、ユーザーはプロセス間のデータのやり取りを軽減できます。



デフォルトでは、MPI プロセスによって作成されるメモリーはプライベートです。MPI-3 共有メモリーは、メモリーのみを共有し、ほかのすべてのリソースはプライベートにする必要があるケースに最適です。共有メモリーを使用する場合、各プロセスが共有メモリー領域にアクセスできるため、プロセスの同期に注意する必要があります。

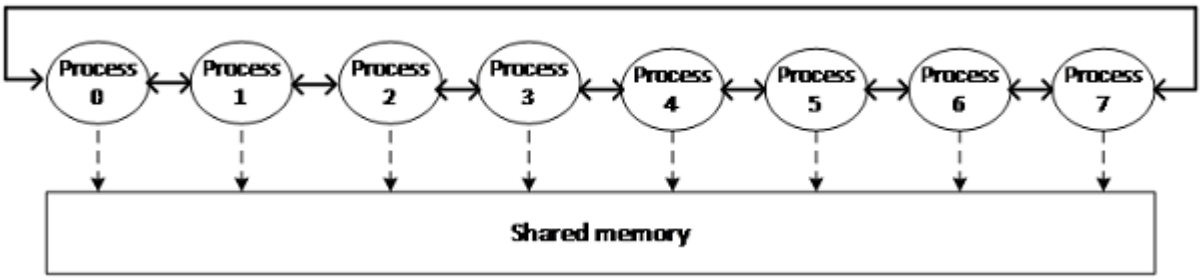
サンプルコード

このセクションでは、サンプルコードを用いて MPI-3 共有メモリーの使用法を示します。

ノードでは合計 8 つの MPI プロセスが生成されます。各プロセスは 3200 万要素の long 型の配列を維持します。配列の各要素 $_j$ に対して、プロセスはこの要素の値を、その現在の値と 2 つの直近のプロセスの対応する配列の要素 $_j$ の値に基づいて更新します。この処理は、配列全体に対して行われます。以下の疑似コードは、プログラムを 8 つの MPI プロセスで 64 回実行します。

次の処理を 64 回繰り返す:
for each MPI process *n* from *0* to *7*:
  for each element *j* in the array *A[k]*:
    $A_n[j] \leftarrow 0.5*A_n[j] + 0.25*A_{previous}[j] + 0.25*A_{next}[j]$

ここで、A_n はプロセス *n* の long 型の配列で、$A_n[j]$ はプロセス *n* の配列の要素 *j* の値です。このプログラムでは、各プロセスは直近の 2 つのプロセスの配列のみを必要としますが (例えば、プロセス 0 はプロセス 1 と 7 のデータが必要、プロセス 1 はプロセス 0 と 2 のデータが必要、…)、各プロセスのローカルメモリーが共有されているため、すべてのプロセスがすべての配列にアクセスします。



サンプルプログラムでは、MPI プログラミングに使用される基本 API に加えて、次の MPI-3 共有メモリー API を使用します。

- `MPI_Comm_split_type`: すべてのプロセスが共通プロパティーを共有する新しいコミュニケーターを作成します。サンプルプログラムでは、`MPI_COMM_WORLD` などの親コミュニケーターから共有メモリーを作成するため、`MPI_COMM_TYPE_SHARED` を引数として渡して、コミュニケーターを共有メモリー・コミュニケーター *shmcomm* に分解します。
- `MPI_Win_allocate_shared`: 共有メモリー・コミュニケーターのすべてのプロセスがアクセス可能な共有メモリーを作成します。各プロセスは、異なるサイズのローカルメモリーを割り当てることができ、それぞれのローカルメモリーをほかのすべてのプロセスと共有します。デフォルトでは、共有メモリー全体が連続して割り当てられます。共有メモリーが連続している必要がない場合は、ヒント "`alloc_shared_noncontig`" を渡します。そうすることで、ハードウェア・アーキテクチャーによっては、パフォーマンスの向上がもたらされます。
- `MPI_Win_free`: メモリーを解放します。
- `MPI_Win_shared_query`: MPI プロセスの共有メモリーのアドレスを照会します。

- `MPI_Win_lock_all` および `MPI_Win_unlock_all`: ウィンドウのすべてのプロセスに対するアクセスエポックを開始します。共有エポックのみ必要です。呼び出しプロセスは、すべてのプロセスの共有メモリーにアクセスできます。
- `MPI_Win_sync`: ローカルメモリーから共有メモリーへのコピーの完了を確認します。
- `MPI_Barrier`: すべてのプロセスがバリアに到達するまでノード上の呼び出し元プロセスをブロックします。バリア同期 API は、すべてのプロセスに対して有効です。

インテル® Xeon Phi™ プロセッサー向けの基本パフォーマンス・チューニング

インテル® Xeon Phi™ プロセッサー 7250 (1.40GHz、68 コア) 搭載の Red Hat* Enterprise Linux* 7.2、インテル® Xeon Phi™ Processor Software 1.5.1、インテル® Parallel Studio 2017 Update 2 がインストールされたシステムで、サンプルプログラムをコンパイルして実行します。デフォルトでは、インテル® コンパイラーはコードのベクトル化を試み、各 MPI プロセスはシングルスレッドで実行します。後で使用するため、ループレベルで OpenMP* プラグマが追加されています。最初に、次のコマンドを実行して、コードをコンパイルしてバイナリー mpishared.out を生成します。

```
$ mpiicc mpishared.c –qopenmp-simd -o mpishared.out
$ mpirun -n 8 ./mpishared.out
Elapsed time in msec: 5699 (after 64 iterations)
```

次に、スレッド並列処理を利用してコアごとに 4 スレッドを実行し、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) 命令を利用するため `-xMIC-AVX512` を指定して再コンパイルします。プロセスが利用できるスレッド数を 4 にするため、環境変数 `OMP_NUM_THREADS` を設定して実行します。

```
$ mpiicc mpishared.c -qopenmp -xMIC-AVX512 -o mpishared.out
$ export OMP_NUM_THREADS=4
$ mpirun -n 8 ./mpishared.out
Elapsed time in msec: 4535 (after 64 iterations)
```

このシステムの MCDRAM は、インテル® Xeon Phi™ プロセッサーが 2 つの NUMA ノードとして認識されるフラットモードに設定されています。ノード 0 にすべての CPU とオンプラットフォーム・メモリーの DDR4 があり、ノード 1 にはオンパケット・メモリーの MCDRAM があります。

```
$ 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
node 0 size: 98200 MB
node 0 free: 92775 MB
node 1 cpus:
node 1 size: 16384 MB
node 1 free: 15925 MB
node distances:
node   0   1
  0:  10  31
  1:  31  10
```

MCDRAM (ノード 1) にメモリーを割り当てるには、次のように numactl –m 1 コマンドを使用します。

```
$ numactl -m 1 mpirun -n 8 ./mpishared.out
Elapsed time in msec: 3070 (after 64 iterations)
```

この単純な最適化手法により、パフォーマンス速度が大幅に向上します。

まとめ

この記事では、MPI-3 共有メモリー機能、そして MPI-3 共有メモリー API を使用するサンプルコードを紹介しました。疑似コードを用いて、共有メモリー API の説明とともにプログラムの動作を説明しました。そして、プログラムをインテル® Xeon Phi™ プロセッサー上で実行して、単純な手法によりさらに最適化しました。

参考文献

1. MPI Forum, MPI 3.0 (英語)
2. Message Passing Interface Forum, MPI: A Message-Passing Interface Standard Version 3.0 (英語)
3. The MIT Press, Using Advanced MPI (英語)
4. James Reinders, Jim Jeffers, Publisher: Morgan Kaufmann, Chapter 16 - MPI-3 Shared Memory Programming Introduction, High Performance Parallelism Pearls Volume Two (英語)

付録

サンプル MPI プログラムのコードはこちらからダウンロードできます。

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。