# Parallel STL 入門ガイド

Alexey M. (Intel)、Andrey F. (Intel)

この記事は、2018 年 6 月 12 日時点の、インテル® デベロッパー・ゾーンに公開されている「Getting Started with Parallel STL」の日本語訳です。

Parallel STL は、一般に C++17 と呼ばれる C++ 標準の次期バージョンの Working Draft N4659 で指定されている、実行ポリシーをサポートする C++ 標準ライブラリー・アルゴリズムの実装です。この実装は、ISO C++ Working Group Paper P0076R3 で指定されている、順序関係が存在しない実行ポリシーもサポートしています。

Parallel STL は、インテル® プロセッサー向けアルゴリズムの並列実行とベクトル実行の両方を効率的にサポートします。シーケンシャル実行は、C++ 標準ライブラリーの利用可能な実装に依存します。

Parallel STL は、インテル® Parallel Studio XE およびインテル® System Studio の一部として利用できます。

## 準備

Parallel STL を使用するには、次のソフトウェアをインストールする必要があります。

- 次の機能をサポートする C++ コンパイラー:
  C++11
  OpenMP* 4.0 SIMD 構造
- 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) 2018

Parallel STL アルゴリズムのパフォーマンスを引き出すには、最新バージョンのインテル® C++ コンパイラーを推奨します。

コマンドラインで Parallel STL を使用するアプリケーションをビルドするには、コンパイルとリンクの環境変数を設定する必要があります。これらの環境変数を設定するには、`compilervars.{sh|csh|bat}` のようなスイートレベルの環境スクリプトを呼び出すか、`<install_dir>/{linux|mac|windows}/pstl/bin` で `pstlvars.{sh|csh|bat}` を実行して Parallel STL の環境変数を設定します。

*`<install_dir>`* はインストール・ディレクトリーです。デフォルトでは次の場所にあります。

**Linux* および macOS*:**

- スーパーユーザーの場合: `/opt/intel/compilers_and_libraries_<version>`
- 通常のユーザーの場合: `$HOME/intel/compilers_and_libraries_<version>`

**Windows*:**

- `<Program Files>¥IntelSWTools¥compilers_and_libraries_<version>`

# Parallel STL の使用

次の手順に従って、アプリケーションに Parallel STL を追加します。

1. コンパイラーのインクルード・パスに `<install_dir>/pstl/include` フォルダーを追加します。スクリプトを使用する場合は `pstlvars` スクリプトを呼び出します。
2. コードに `#include "pstl/execution"` を追加します。次に、使用するアルゴリズムに応じて、次の行のサブセットを追加します。
   - `#include "pstl/algorithm"`
   - `#include "pstl/numeric"`
   - `#include "pstl/memory"`
3. アルゴリズムと実行ポリシーを使用するとき、C++17 標準ライブラリーのベンダー実装がない場合は `std::execution` 名前空間、その他の場合は `pstl::execution` 名前空間を指定します。この後の「サンプル」セクションを参照してください。
4. 実装されているアルゴリズムでは、呼び出しの最初の引数として値の 1 つ (`seq`、`unseq`、`par` または `par_unseq`) をアルゴリズムに渡して、実行ポリシーを指定します。ポリシーには次の意味があります。

| 実行ポリシー | 意味 |
|---|---|
| `seq` | シーケンシャル実行。 |
| `unseq` | SIMD を使用します。提供されているすべての関数が SIMD セーフである必要があります。 |
| `par` | マルチスレッドを使用します。 |
| `par_unseq` | `unseq` と `par` の効果を組み合わせます。 |

5. コードを C++11 (またはそれ以降) としてコンパイルし、ベクトル化のコンパイラー・オプションを使用します。
   - インテル® C++ コンパイラーの場合:
     Linux* および macOS*: `-qopenmp-simd` または `–qopenmp`
     Windows*: `/Qopenmp-simd` または `/Qopenmp`
   - ほかのコンパイラーの場合は、OpenMP* 4.0 SIMD 構造を有効にするオプションを使用します。

優れたパフォーマンスを得るには、ターゲット・プラットフォームを指定します。インテル® C++ コンパイラーの場合、適切なオプションは次のとおりです。

- Linux* および macOS*: `-xHOST`、`-xSSE4.1`、`-xCORE-AVX2`、`-xMIC-AVX512`
- Windows*: `/QxHOST`、`/QxSSE4.1`、`/QxCORE-AVX2`、`/QxMIC-AVX512`

異なるコンパイラーを使用している場合は、そのコンパイラーのドキュメントを参照してください。

6. 並列処理を行うにはインテル® TBB ダイナミック・ライブラリーとリンクします。インテル® C++ コンパイラーの場合、次のオプションを使用します。
   - Linux* および macOS*: `-tbb`
   - Windows*: `/Qtbb` (オプション、`#pragma comment(lib, <libname>)` で処理)

## バージョンマクロ

バージョンに関するマクロを次に示します。これらのマクロは再定義すべきではありません。

`PSTL_VERSION`

現在の Parallel STL のバージョン。値は、*xyy* 形式の 10 進数です。ここで、*x* はメジャーバージョン番号、*yy* はマイナーバージョン番号です。

`PSTL_VERSION_MAJOR`

PSTL_VERSION/100; つまり、メジャーバージョン番号。

`PSTL_VERSION_MINOR`

PSTL_VERSION - PSTL_VERSION_MAJOR * 100; つまり、マイナーバージョン番号。

## マクロ

`PSTL_USE_PARALLEL_POLICIES`

このマクロは、並列ポリシーの使用を制御します。

0 に設定すると、par および par_unseq ポリシーが無効になり、これらのポリシーを使用するとコンパイルエラーになります。unseq ポリシーのみを使用するベクトル化されたコードで、インテル® TBB ランタイム・ライブラリーへの依存を回避するために推奨します。

マクロが定義されていない場合 (デフォルト)、またはマクロが非ゼロ値に評価された場合、すべての実行ポリシーが有効になります。

PSTL_USE_NONTEMPORAL_STORES

このマクロは、unseq ポリシーのアルゴリズム std::copy、std::copy_n、std::fill、std::fill_n、std::generate、std::generate_n で #pragma vector nontemporal を使用できるようにします。プラグマの詳細は、**『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』** (https://software.intel.com/en-us/node/524559) (英語) を参照してください。

マクロが非ゼロ値に評価された場合、#pragma vector nontemporal を使用できるようにします。

マクロが定義されていない場合 (デフォルト)、または 0 に設定された場合、マクロは何も行いません。

# サンプル

### サンプル 1

このサンプルはベクトル化された copy を呼び出します。

```
#include "pstl/execution"
#include "pstl/algorithm"
void foo(float* a, float* b, int n) {
    std::copy(pstl::execution::unseq, a, a+n, b);
}
```

### サンプル 2

このサンプルは fill_n の並列化バージョンを呼び出します。

```
#include <vector>
#include "pstl/execution"
#include "pstl/algorithm"


int main()
{
    std::vector<int> data(10000000);
    std::fill_n(pstl::execution::par_unseq, data.begin(), data.size(), -1);
// ベクトルを -1 でフィル
    return 0;
}
```

# 実装されているアルゴリズム

Parallel STL は、次の表にリストされているアルゴリズムでのみ、前述のすべての実行ポリシーをサポートします。その他の C++ 標準ライブラリー・アルゴリズムにポリシー引数を追加すると、シーケンシャル実行になります。

| アルゴリズム | cppreference.com のアルゴリズム・ページ (英語) |
|---|---|
| adjacent_find | http://en.cppreference.com/w/cpp/algorithm/adjacent_find |
| all_of | http://en.cppreference.com/w/cpp/algorithm/all_any_none_of |
| any_of | http://en.cppreference.com/w/cpp/algorithm/all_any_none_of |
| copy | http://en.cppreference.com/w/cpp/algorithm/copy |
| copy_if | http://en.cppreference.com/w/cpp/algorithm/copy |
| copy_n | http://en.cppreference.com/w/cpp/algorithm/copy_n |
| count | http://en.cppreference.com/w/cpp/algorithm/count |
| count_if | http://en.cppreference.com/w/cpp/algorithm/count |
| destroy | http://en.cppreference.com/w/cpp/memory/destroy |
| destroy_n | http://en.cppreference.com/w/cpp/memory/destroy_n |
| equal | http://en.cppreference.com/w/cpp/algorithm/equal |
| exclusive_scan | http://en.cppreference.com/w/cpp/algorithm/exclusive_scan |
| fill | http://en.cppreference.com/w/cpp/algorithm/fill |
| fill_n | http://en.cppreference.com/w/cpp/algorithm/fill_n |
| find | http://en.cppreference.com/w/cpp/algorithm/find |
| find_end | http://en.cppreference.com/w/cpp/algorithm/find_end |
| find_first_of | http://en.cppreference.com/w/cpp/algorithm/find_first_of |
| find_if | http://en.cppreference.com/w/cpp/algorithm/find |
| find_if_not | http://en.cppreference.com/w/cpp/algorithm/find |
| for_each | http://en.cppreference.com/w/cpp/algorithm/for_each |
| for_each_n | http://en.cppreference.com/w/cpp/algorithm/for_each_n |
| generate | http://en.cppreference.com/w/cpp/algorithm/generate |
| generate_n | http://en.cppreference.com/w/cpp/algorithm/generate_n |
| inclusive_scan | http://en.cppreference.com/w/cpp/algorithm/inclusive_scan |

| | |
|---|---|
| is_heap | http://en.cppreference.com/w/cpp/algorithm/is_heap |
| is_heap_until | http://en.cppreference.com/w/cpp/algorithm/is_heap_until |
| is_partitioned | http://en.cppreference.com/w/cpp/algorithm/is_partitioned |
| is_sorted | http://en.cppreference.com/w/cpp/algorithm/is_sorted |
| is_sorted_until | http://en.cppreference.com/w/cpp/algorithm/is_sorted_until |
| lexicographical_compare | http://en.cppreference.com/w/cpp/algorithm/lexicographical_compare |
| max_element | http://en.cppreference.com/w/cpp/algorithm/max_element |
| merge | http://en.cppreference.com/w/cpp/algorithm/merge |
| min_element | http://en.cppreference.com/w/cpp/algorithm/min_element |
| minmax_element | http://en.cppreference.com/w/cpp/algorithm/minmax_element |
| mismatch | http://en.cppreference.com/w/cpp/algorithm/mismatch |
| move | http://en.cppreference.com/w/cpp/algorithm/move |
| none_of | http://en.cppreference.com/w/cpp/algorithm/all_any_none_of |
| partial_sort | http://en.cppreference.com/w/cpp/algorithm/partial_sort |
| partition_copy | http://en.cppreference.com/w/cpp/algorithm/partition_copy |
| reduce | http://en.cppreference.com/w/cpp/algorithm/reduce |
| remove_copy | http://en.cppreference.com/w/cpp/algorithm/remove_copy |
| remove_copy_if | http://en.cppreference.com/w/cpp/algorithm/remove_copy |
| replace | http://en.cppreference.com/w/cpp/algorithm/replace |
| replace_copy | http://en.cppreference.com/w/cpp/algorithm/replace_copy |
| replace_copy_if | http://en.cppreference.com/w/cpp/algorithm/replace_copy |
| replace_if | http://en.cppreference.com/w/cpp/algorithm/replace |
| search | http://en.cppreference.com/w/cpp/algorithm/search |
| search_n | http://en.cppreference.com/w/cpp/algorithm/search_n |
| sort | http://en.cppreference.com/w/cpp/algorithm/sort |
| stable_sort | http://en.cppreference.com/w/cpp/algorithm/stable_sort |
| swap_ranges | http://en.cppreference.com/w/cpp/algorithm/swap_ranges |
| transform | http://en.cppreference.com/w/cpp/algorithm/transform |

| | |
|---|---|
| `transform_exclusive_scan` | http://en.cppreference.com/w/cpp/algorithm/transform_exclusive_scan |
| `transform_inclusive_scan` | http://en.cppreference.com/w/cpp/algorithm/transform_inclusive_scan |
| `transform_reduce` | http://en.cppreference.com/w/cpp/algorithm/transform_reduce |
| `uninitialized_copy` | http://en.cppreference.com/w/cpp/memory/uninitialized_copy |
| `uninitialized_copy_n` | http://en.cppreference.com/w/cpp/memory/uninitialized_copy_n |
| `uninitialized_default_construct` | http://en.cppreference.com/w/cpp/memory/uninitialized_default_construct |
| `uninitialized_default_construct_n` | http://en.cppreference.com/w/cpp/memory/uninitialized_default_construct_n |
| `uninitialized_fill` | http://en.cppreference.com/w/cpp/memory/uninitialized_fill |
| `uninitialized_fill_n` | http://en.cppreference.com/w/cpp/memory/uninitialized_fill_n |
| `uninitialized_move` | http://en.cppreference.com/w/cpp/memory/uninitialized_move |
| `uninitialized_move_n` | http://en.cppreference.com/w/cpp/memory/uninitialized_move_n |
| `uninitialized_value_construct` | http://en.cppreference.com/w/cpp/memory/uninitialized_value_construct |
| `uninitialized_value_construct_n` | http://en.cppreference.com/w/cpp/memory/uninitialized_value_construct_n |
| `unique_copy` | http://en.cppreference.com/w/cpp/algorithm/unique_copy |

# 既知の制限

ランダム・アクセス・イテレーターが提供される場合、並列実行とベクトル実行は前述のアルゴリズムのサブセットでのみサポートされ、残りの実行はシリアルのままです。

# 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 |



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。