OpenMP* 検証サンプル
omp simd 初級編

このセッションは、「インテル® コンパイラーを使用した OpenMP* 5.0 による並列プログラミング・セミナー【パート 2】」で公開されているオンラインセミナーの補足セッションです

内容

- ベクトル化の確認
- OpenMP* に関連するコンパイラー・オプション
  - omp simd とコンパイルオプション
- コンパイラーのプラグマとディレクティブ
  - 古いプラグマとの関係 (#pragma simd)
  - #pragma omp simd とベクトル化レポート
- 例題: omp simd ベクトル化における問題と解決

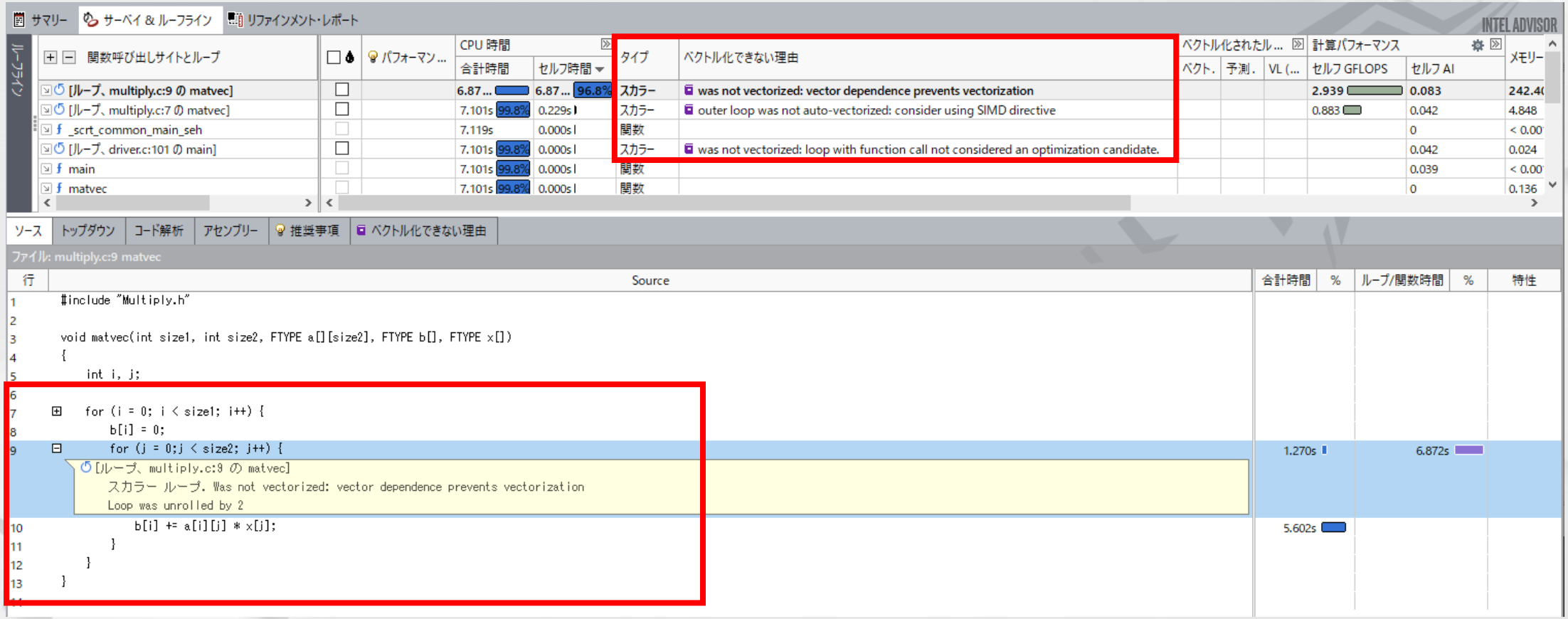
ここで使用する環境

- Windows* 10 Pro 2004
- インテル® Core™ i7-9700K プロセッサー (8 コア、8 スレッド)、メモリー 16GB
- インテル® C++ コンパイラー 19.1 Update 2 (VS2017 ライブラリー)
- インテル® VTune™ プロファイラー 2020 Update 2 (日本語パッケージ)
- インテル® Advisor 2020 Update 2 (日本語パッケージ)

ベクトル化の確認

- インテル® Advisor によるベクトル化の確認
- 最適化レポートによるベクトル化の確認
- インテル® VTune™ プロファイラーによるベクトル化の確認

インテル® Advisor によるベクトル化の確認



- インテル® Advisor の [サーベイ解析] により、ループの [タイプ] と [ベクトル化できない理由] カラムを確認
- [ソース] タブでソースコードの正確な位置を確認

最適化レポートによるベクトル化の確認

- /Qopt-report[:n] /Qopt-report-phase:vec (Windows*)
- -qopt-report[=n] –qopt-report-phase=vec (Linux*)

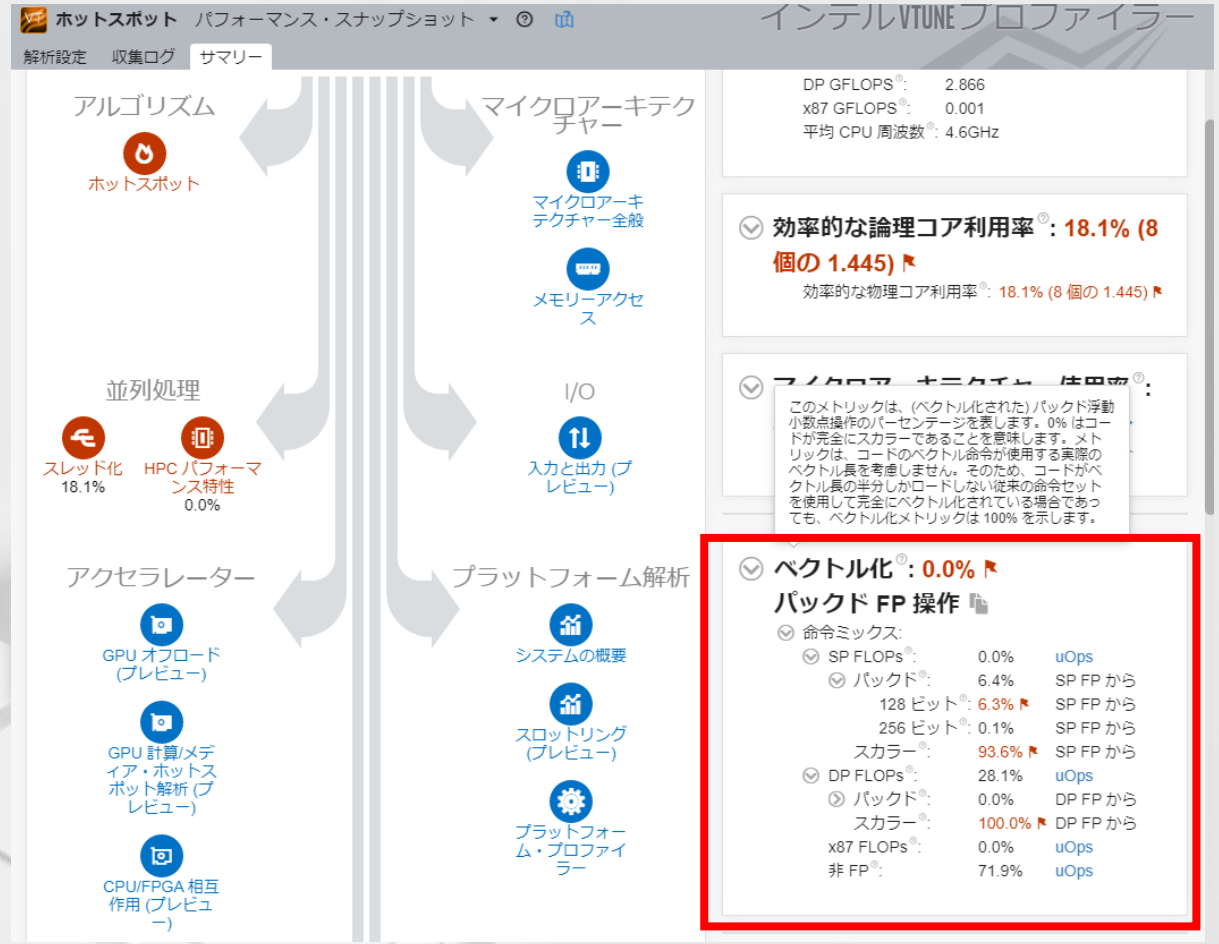
vec: ベクトルの最適化:

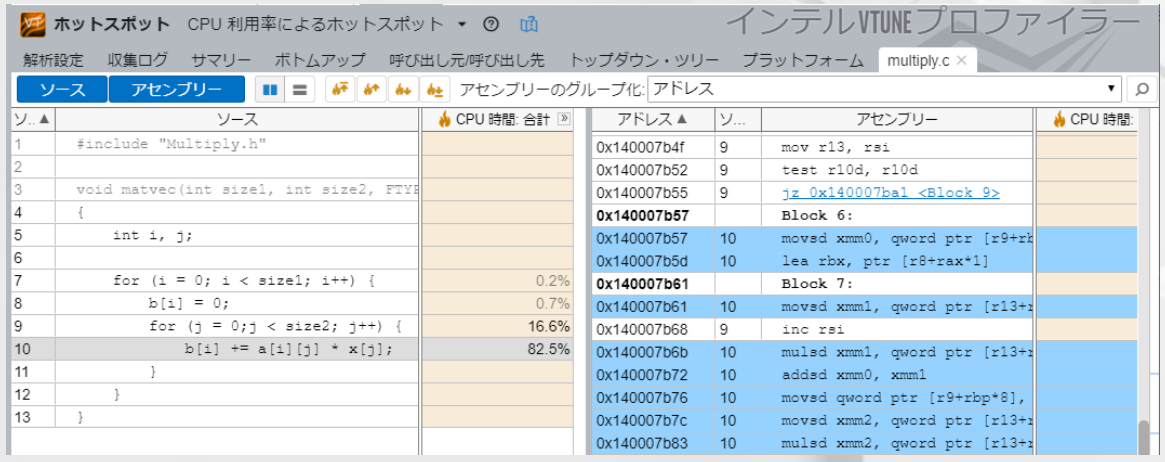
- レベル 1: ベクトル化されたループのレポート
- レベル 2: レベル 1 + ベクトル化されなかったループとその理由のレポート
- レベル 3: レベル 2 + ループのベクトル化のサマリー
- レベル 4: レベル 3 + ループがベクトル化された (されなかった) 理由の詳細なレポート
- レベル 5: レベル 4 + ベクトルを妨げている変数/メモリーの依存関係に関する情報

「ソースファイル名.optrpt」
でテキスト形式のレポート
が生成されます

```
ループの開始 C:¥matmul¥multiply.c(7,2)
  リマーク #15541: 外部 ループ は自動ベクトル化されませんでした: SIMD ディレクティブの使用を検討してください。

  ループの開始 C:¥matmul¥multiply.c(9,3)
    リマーク #15344: ループ はベクトル化されませんでした: ベクトル依存関係がベクトル化を妨げています。
    最初の依存関係を以下に示します。詳細については、レベル 5 のレポートを使用してください。
    リマーク #15346: ベクトル依存関係: FLOW の依存関係が b[i] (10:4) と b[i] (10:4) の間に仮定されました。
  ループの終了

  ループの開始 C:¥matmul¥multiply.c(9,3)
  <剰余>
  ループの終了
ループの終了
```

インテル® VTune™ プロファイラーによるベクトル化の確認





- パフォーマンス・スナップショットのベクトル化のサマリーでベクトル化の比率を確認
- ソースコード・ビューでアセンブリー・ソースを表示して命令レベルで確認

OpenMP* に関連するコンパイラー・オプション

オプションの関係: /Qopenmp と /Qopenmp-simd (1)

Linux* 環境では、/Q に代わって –q を指定します

```
float add_fl(float *a, float *b, float *c, float sum, int *p)
{
    #pragma omp parallel for simd reduction(+:sum)

    for (int i=0; i<*p; i++){
        a[i] = b[i] * c[i];
        sum = sum + a[i];
    }
    return sum;
}
```

注: omp simd はオプションがなくても
デフォルトで有効になります

- /Qopenmp と /Qopenmp-simd は異なる意味を持ちます
  - /Qopenmp はスレッド化有効にします (simd ベクトル化は有効です)
  - /Qopenmp-simd は simd ベクトル化を有効にします
  - /Qopenmp-simd- (Linux* では -qno-openmp-simd) は simd ベクトル化を無効にします
  - /Qopenmp /Qopenmp-simd- (プラグマは無視されます)
  - /Qopenmp- はスレッド化を無効にします (simd ベクトル化は有効)

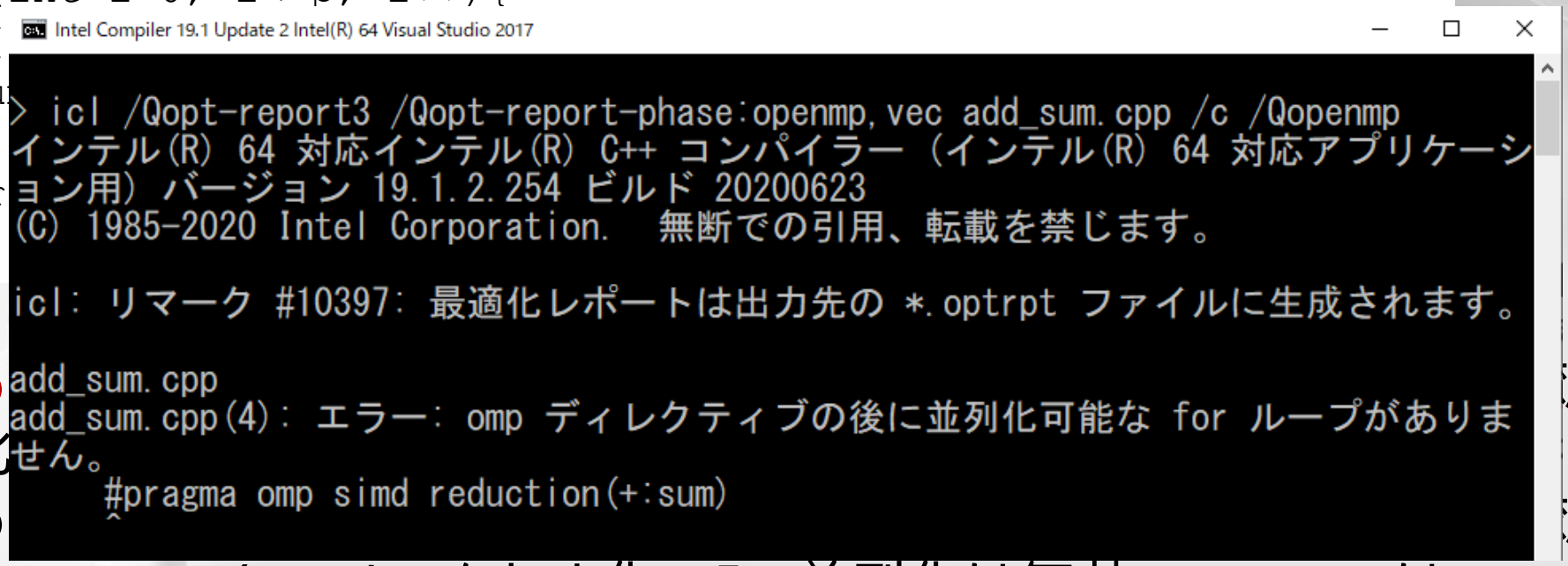
オプションの関係: /Qopenmp と /Qopenmp-simd (2)

```
float add_fl(float *a, float *b, float *c, float sum, int *p)
{
   #pragma omp parallel for reduction(+:sum)
   #pragma omp simd reduction(+:sum)
   for (int i=0; i<*p; i++){
      a[i] = b[i] * c[i];
      sum = sum + a[i];
   }
   return sum;
}
```

- /Qopenmp /Qopenmp-simd- (並列化のみ、simd ベクトル化は無効。自動ベクトル化でベクトル化される可能性があります)
- /Qopenmp- /Qopenmp-simd (simd ベクトル化のみ、並列化は無効)
- /Qopenmp- (simd ベクトル化のみ、並列化は無効。Linux* では -qno-openmp)

オプションの関係: /Qopenmp と /Qopenmp-simd (2)

```
float add_fl(float *a, float *b, float *c, float sum, int *p)
{
    #pragma omp parallel for reduction(+:sum)
    #pragma omp simd reduction(+:sum)
    for (int i=0; i<*p; i++){
        a[
        su
    }
    retur
}
```



- /Qo 効。自動ベクトル化
- /Qo 効)
- /Qopenmp- (simd ベクトル化のみ、並列化は無効。Linux* では -qno-openmp)
- /Qopenmp のみ …

複合構造とコンパイラー・レポート

```
#pragma omp parallel for simd reduction(+:sum)
```

/Qopenmp で生成されるレポート

/Qopenmp-simd 、または OpenMP* オプションなしで生成されるレポート

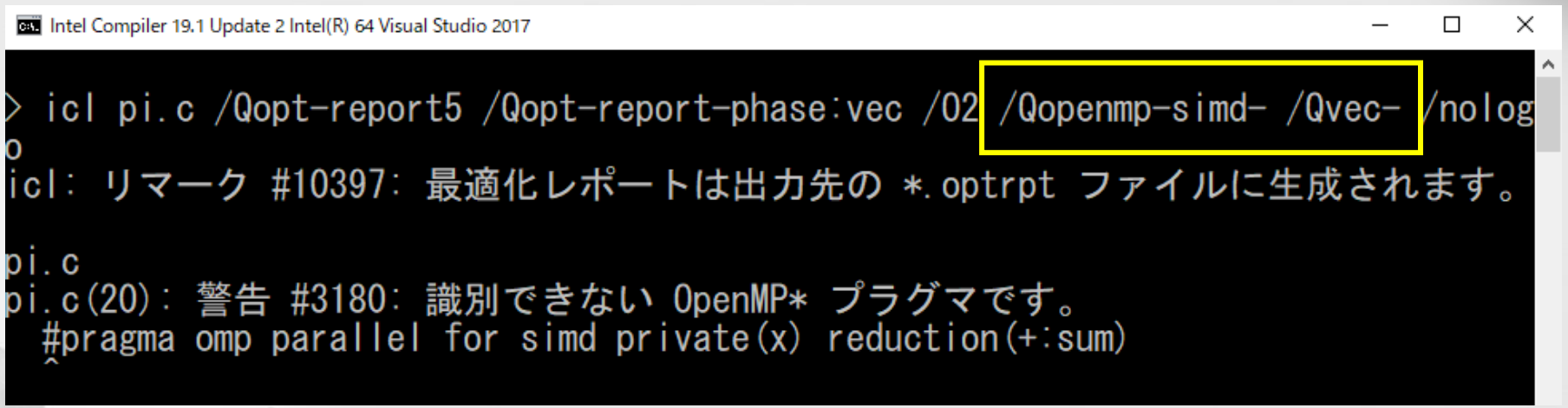
OpenMP* による並列化とベクトル化は個別にコントロールできます

注: omp simd はオプションがなくてもデフォルトで有効になります

```
ループの開始 C:¥add_sum.cpp(3,4)
<ベクトル化のピールループ>
ループの終了

ループの開始 C:¥add_sum.cpp(3,4)
   リマーク #15301: SIMD ループ がベクトル化されました。
   リマーク #15442: すべてのループは剰余ループとして実行されます。
   リマーク #15448: マスクなしアライン・ユニット・ストライド・ロード: 2
   リマーク #15449: マスクなしアライン・ユニット・ストライド・ストア: 1
   リマーク #15450: マスクなし非アライン・ユニット・ストライド・ロード: 1
   リマーク #15475: --- ベクトルのコストサマリー開始 ---
   リマーク #15476: スカラーのコスト: 12
   リマーク #15477: ベクトルのコスト: 3.250
   リマーク #15478: スピードアップの期待値: 3.400
   リマーク #15488: --- ベクトルのコストサマリー終了 ---
ループの終了

ループの開始 C:¥add_sum.cpp(3,4)
<代替アライメントでベクトル化されたループ>
ループの終了

ループの開始 C:¥add_sum.cpp(3,4)
<ベクトル化の剰余ループ>
ループの終了
```

自動ベクトル化も無効にする場合

```
#pragma omp parallel for simd private(x) reduction(+:sum)
      for (i=0; i<num_steps; i++) {
        x = (i + .5)*step;
        sum += 4.0/(1.+ x*x);
      }
```



/Qvec- (Windows*)
-no-vec (Linux*)
オプションを使用

ループの開始 C:¥pi.c(31,2)
　リマーク #15540: ループ はベクトル化されませんでした: 自動ベクトル化が /Qvec- オプションにより無効にされています。
ループの終了

コンパイラーのプラグマとディレクティブ

コンパイラー・プラグマ (例えば、#pragma vector)

```
4 void matvec(int size1, int size2, FTYPE a[][size2], FTYPE b[], FTYPE x[]) {
5       int i, j;
6 #pragma vector always または #pragma ivdep など
7       for (i = 0; i < size1; i++) {
8               b[i] = 0;
9               for (j = 0;j < size2; j++) {
10                      b[i] += a[i][j] * x[j];
11              }
12      }
13 }
```

ループの開始 C:¥multiply.c(7,2)
リマーク #15541: 外部 ループ は自動ベクトル化されませんでした: SIMD ディレクティブの使用を検討してください。

```
6       #pragma vector always
7               for (i = 0; i < size1; i++) {
//
//ループの開始 C:¥multiply.c(7,2)
//   リマーク #15541: 外部 ループ は自動ベクトル化されませんでした: SIMD ディレクティブの使用を検討してください。
```

#pragma simd と #pragma omp simd

```
float add_fl(float *a, int n){
float sum = 0.0f;
float *p = a;
int step = 4;

#pragma simd
for (int i=0; i<n; i++){
      sum += *p;
      p += step;
   }
   return sum;
}
```

- インテル® C++ コンパイラーでは、インテル® Cilk™ Plus の拡張機能の一部として、#pragma simd による明示的なベクトル化がサポートされていました
- V19 以降ではインテル® Cilk™ Plus が非推奨の機能となり #pragma simd を含むソースをコンパイルするとコンパイル時に警告が出力されます

```
add_sum2.cpp
add_sum2.cpp(7): 警告 #3948: simd pragma は古いオプションで、将来のリリースではサポートされなくなります。詳細および推奨される代替オプションについては、リリースノートを参照してください。
  #pragma simd
           ^
```

#pragma omp simd とベクトル化レポート: 例 1

/Qopt-report3 /Qopt-report-phase:vec,openmp でレポートを生成

```
float add_fl(float *a, int n){
float sum = 0.0f;
float *p = a;
int step = 4;

#pragma omp simd
for (int i=0; i<n; i++){
      sum += *p;
      p += step;
   }
   return sum;
}
```

```
ループの開始 C:¥add_sum2.cpp(9,4)
  リマーク #15335: ループ はベクトル化されませんでした: ベクトル化は可能ですが非効率です。オーバーライドするには vector always ディレクティブまたは /Qvec-threshold0 を使用してください。
  リマーク #15452: マスクなしストライドロード: 1
  リマーク #15475: --- ベクトルのコストサマリー開始 ---
  リマーク #15476: スカラーのコスト: 4
  リマーク #15477: ベクトルのコスト: 9.250
  リマーク #15478: スピードアップの期待値: 0.430
  リマーク #15488: --- ベクトルのコストサマリー終了 ---
ループの終了
```

```
ループの開始 C:¥add_sum2.cpp(9,4)
  リマーク #15301: SIMD ループ がベクトル化されました。
  リマーク #15452: マスクなしストライドロード: 1
  リマーク #15475: --- ベクトルのコストサマリー開始 ---
  リマーク #15476: スカラーのコスト: 4
  リマーク #15477: ベクトルのコスト: 9.250
  リマーク #15478: スピードアップの期待値: 0.430
  リマーク #15488: --- ベクトルのコストサマリー終了 ---
ループの終了
```

#pragma omp simd とベクトル化レポート: 例 2

```
float add_fl(float *a, float *b, float *c,
float sum, int *p)
{
    #pragma omp simd reduction(+:sum)
    for (int i=0; i<*p; i++){
        a[i] = b[i] * c[i];
        sum = sum + a[i];
    }
    return sum;
}
```

omp simd なし、または /Qopenmp-simd- でコンパイル

```
ループの開始 C:¥vector¥add_sum.cpp(4,4)
   リマーク #15523: ループ はベクトル化されませんでした: ループ制御変数 i は見つかりましたが、ループを実行する前にループの反復数を計算できません。
ループの終了
```

デフォルトまたは /Qopenmp-simd でコンパイル

```
ループの開始 C:¥add_sum.cpp(4,4)
<ベクトル化のピールループ>
ループの終了

ループの開始 C:¥add_sum.cpp(4,4)
   リマーク #15301: SIMD ループ がベクトル化されました。
   リマーク #15442: すべてのループは剰余ループとして実行されます。
   リマーク #15448: マスクなしアライン・ユニット・ストライド・ロード: 2
   リマーク #15449: マスクなしアライン・ユニット・ストライド・ストア: 1
   リマーク #15450: マスクなし非アライン・ユニット・ストライド・ロード: 1
   リマーク #15475: --- ベクトルのコストサマリー開始 ---
   リマーク #15476: スカラーのコスト: 13
   リマーク #15477: ベクトルのコスト: 3.250
   リマーク #15478: スピードアップの期待値: 3.670
   リマーク #15488: --- ベクトルのコストサマリー終了 ---
ループの終了

ループの開始 C:¥add_sum.cpp(4,4)
<代替アライメントでベクトル化されたループ>
ループの終了

ループの開始 C:¥add_sum.cpp(4,4)
<ベクトル化の剰余ループ>
ループの終了
```

#pragma omp simd とベクトル化レポート: 例 3

```
float add_fl(float *a, float *b, float *c,
float sum, int *p)
{
    int max = *p;
    for (int i=0; i<max; i++){
        a[i] = b[i] * c[i];
        sum = sum + a[i];
    }
    return sum;
}
```

ベクトル化を妨げる要因をソースコードの修正で対応し、自動ベクトル化が適用されるケースと omp simd による強制では生成されるコードが異なる場合があります

omp simd による
ベクトル化

```
最適化レポート開始: add_fl(float *, float *, float *, float, int *)

   レポート: ベクトルの最適化 [vec]


ループの開始 C:¥add_sum.cpp(4,4)
<ベクトル化のピールループ>
ループの終了

ループの開始 C:¥add_sum.cpp(4,4)
  リマーク #15301: SIMD ループ がベクトル化されました。
  リマーク #15442: すべてのループは剰余ループとして実行されます。
  リマーク #15448: マスクなしアライン・ユニット・ストライド・ロード: 2
  リマーク #15449: マスクなしアライン・ユニット・ストライド・ストア: 1
  リマーク #15450: マスクなし非アライン・ユニット・ストライド・ロード: 1
  リマーク #15475: --- ベクトルのコストサマリー開始 ---
  リマーク #15476: スカラーのコスト: 13
  リマーク #15477: ベクトルのコスト: 3.250
  リマーク #15478: スピードアップの期待値: 3.670
  リマーク #15488: --- ベクトルのコストサマリー終了 ---
ループの終了

ループの開始 C:¥add_sum.cpp(4,4)
<代替アライメントでベクトル化されたループ>
ループの終了

ループの開始 C:¥add_sum.cpp(4,4)
<ベクトル化の剰余ループ>
ループの終了
=================================================================================
```

```
ループの開始 C:¥add_sum.cpp(5,4)
<マルチバージョン v2, 分散チャンク1>
  リマーク #15304: ループ はベクトル化されませんでした: マルチバージョンのベクトル化できないループ・インスタンスです。
ループの終了

ループの開始 C:¥add_sum.cpp(5,4)
<剰余, マルチバージョン v2, 分散チャンク1>
ループの終了

ループの開始 C:¥add_sum.cpp(5,4)
<ベクトル化のピールループ, マルチバージョン v2, 分散チャンク2>
ループの終了

ループの開始 C:¥add_sum.cpp(5,4)
<マルチバージョン v2, 分散チャンク2>
  リマーク #15301: 部分ループ がベクトル化されました。
  リマーク #15442: すべてのループは剰余ループとして実行されます。
  リマーク #15448: マスクなしアライン・ユニット・ストライド・ロード: 1
  リマーク #15475: --- ベクトルのコストサマリー開始 ---
  リマーク #15476: スカラーのコスト: 5
  リマーク #15477: ベクトルのコスト: 1.250
  リマーク #15478: スピードアップの期待値: 3.480
  リマーク #15488: --- ベクトルのコストサマリー終了 ---
ループの終了

ループの開始 C:¥add_sum.cpp(5,4)
<ベクトル化の剰余ループ, マルチバージョン v2, 分散チャンク2>
ループの終了
=================================================================================
```

例題: omp simd ベクトル化における問題と解決

サンプルコードの構造

driver.c

```
#define REPEATNTIMES 1000000
…
for (k = 0;k < REPEATNTIMES;k++) {
      matvec(size1,size2,a,b,x);
      x[0] = x[0] + 0.000001;
}
```

multiply.c

```
void matvec(int size1, int size2, FTYPE a[][size2], FTYPE b[], FTYPE x[])
{
    int i, j;

    for (i = 0; i < size1; i++) {
         b[i] = 0;
         for (j = 0;j < size2; j++) {
              b[i] += a[i][j] * x[j];
         }
    }
}
```

最適化前のパフォーマンスとレポート

```
3  void matvec(int size1, int size2, FTYPE a[][size2], FTYPE b[], FTYPE x[])
4  {
5      int i, j;
6
7      for (i = 0; i < size1; i++) {
8          b[i] = 0;
9           for (j = 0;j < size2; j++) {
10              b[i] += a[i][j] * x[j];
11          }
12      }
13 }
```

> icl driver.c multiply.c /O2 –o default

```
Intel Compiler 19.1 Update 2 Intel(R) 64 Visual Studio 2017

> default

ROW:101 COL: 101
Execution time is 7.037 seconds
GigaFlops = 2.899247
Sum of result = 195853.999899
```

```
最適化レポート開始: matvec(int, int, double (*)[*], double *, double *)

   レポート: ベクトルの最適化 [vec]

ループの開始 C:¥matmul¥multiply.c(7,2)
  リマーク #15541: 外部 ループ は自動ベクトル化されませんでした: SIMD ディレクティブの使用を検討してください。

  ループの開始 C:¥multiply.c(9,3)
    リマーク #15344: ループ はベクトル化されませんでした: ベクトル依存関係がベクトル化を妨げています。最初の依存関係を以下に示します。詳細については、レベル 5 のレポートを使用してください。
    リマーク #15346: ベクトル依存関係: FLOW の依存関係が b[i] (17:4) と b[i] (17:4) の間に仮定されました。
  ループの終了

  ループの開始 C:¥multiply.c(9,3)
  <剰余>
  ループの終了
ループの終了
================================================================================
```

omp simd または IPO 最適化

```
void matvec(int size1, int size2, FTYPE a[][size2], FTYPE b[], FTYPE x[]) {
      int i, j;
#pragma omp simd
      for (i = 0; i < size1; i++) {
            b[i] = 0;
            for (j = 0;j < size2; j++) {
                  b[i] += a[i][j] * x[j];
            }
      }
}
```

icl driver.c multiply.c

icl driver.c multiply.c /Qipo /Qopenmp-simd-

なぜ omp simd のパフォーマンスは ipo より低いのか

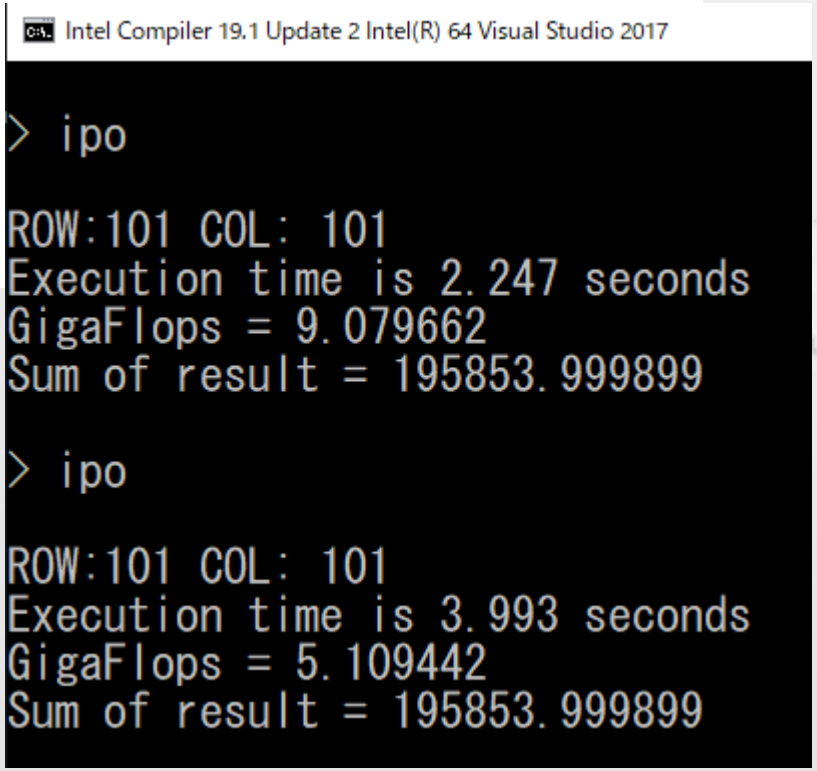
omp simd と IPO 最適化

```
void matvec(int size1, int size2, FTYPE a[][size2], FTYPE b[], FTYPE x[]) {
     int i, j;
#pragma omp simd
     for (i = 0; i < size1; i++) {
          b[i] = 0;
          for (j = 0;j < size2; j++) {
               b[i] += a[i][j] * x[j];
          }
     }
}
```

icl driver.c multiply.c /Qipo /Qopenmp-simd-

icl driver.c multiply.c /Qipo /Qopenmp-simd

omp simd がパフォーマンス低下を招いている



ここまでのまとめ

omp simd なしの
/Qipo オプション

omp simd ありの /Qipo
と /Qopenmp-simd-
オプション

>

omp simd ありで /Qipo
オプションなし

omp simd ありで /Qipo
オプションあり

コンパイラーの最適化レポートや、インテル® Advisor で確認したところ、最もパフォーマンスが高いコードは matvec 関数がインライン展開されていることが判明

パフォーマンスが中間のコードは matvec 関数がインライン展開されていないことが判明

- 関数内の omp simd プラグマやディレクティブは、コンパイラーの最適化よりも優先されます
- 関数内に omp simd が記述されていると、その関数はインライン展開から除外されます

ここまでのまとめ

omp simd なしの
/Qipo オプション

omp simd ありの /
と /Qopenmp-simd
オプション

```
#define REPEATNTIMES 1000000
…
for (k = 0;k < REPEATNTIMES;k++) {
      matvec(size1,size2,a,b,x);
      x[0] = x[0] + 0.000001;
}
```

omp simd ありで /Qipo
オプションなし

d ありで /Qipo
ンあり

コンパイラーの最適化レポートや、インテル® Advisor で確認したところ、最もパフォーマンスが高いコードは matvec 関数がインライン展開されていることが判明

パフォーマンスが中間のコードは matvec 関数がインライン展開されていないことが判明

- 関数内の omp simd プラグマやディレクティブは、コンパイラーの最適化よりも優先されます
- 関数内に omp simd が記述されていると、その関数はインライン展開から除外されます

インライン展開を強制

```
#define REPEATNTIMES 1000000
…
for (k = 0;k < REPEATNTIMES;k++) {
#pragma forceinline
      matvec(size1,size2,a,b,x);
      x[0] = x[0] + 0.000001;
}
```

コンパイラー・オプション:
/O2
/Qopt-report5
/Qopt-report-phase:vec,loop,ipo
/Qopt-report-annotate-position:both

```
void matvec(int size1, int size2, FTYPE a[][size2], FTYPE b[], FTYPE x[]) {
     int i, j;
#pragma omp simd safelen(4) aligned(a, b, x:16)
     for (i = 0; i < size1; i++) {
          b[i] = 0;
          for (j = 0;j < size2; j++) {
               b[i] += a[i][j] * x[j];
          }
     }
}
```

Intel Compiler 19.1 Update 2 Intel(R) 64 Visual Studio 2017

```
> simd2in

ROW:101 COL: 101
Execution time is 3.988 seconds
GigaFlops = 5.115848
Sum of result = 195853.999899
```

最適化レポートは?

```
最適化レポート開始: matvec(int, int, double (*)[*], double *, double *)
   レポート: プロシージャー間の最適化 [ipo]
インライン展開レポート: (matvec(int, int, double (*)[*], double *, double *)) [9/1=900.0%] C:¥multiply_simd2.c(4,1)
   レポート: ループの入れ子、ベクトルの最適化 [loop, vec]
ループの開始 C:¥multiply_simd2.c(11,2)
  リマーク #15388: ベクトル化のサポート: 参照 b[i] にアラインされたアクセスが含まれています。 [ C:¥multiply_simd2.c(12,3) ]
  リマーク #15388: ベクトル化のサポート: 参照 b[i] にアラインされたアクセスが含まれています。 [ C:¥multiply_simd2.c(15,4) ]
  リマーク #15388: ベクトル化のサポート: 参照 b[i] にアラインされたアクセスが含まれています。 [ C:¥multiply_simd2.c(15,4) ]
  リマーク #15328: ベクトル化のサポート: 非ユニットストライド ロードがエミュレートされました (変数 <a[i][j]>、ストライドは コンパイラーに不明)  [ C:¥multiply_simd2.c(15,4) ]
  リマーク #15305: ベクトル化のサポート: ベクトル長 2
  リマーク #15309: ベクトル化のサポート: 正規化されたベクトル化のオーバーヘッド 0.286
  リマーク #15301: SIMD ループ がベクトル化されました。
  リマーク #15448: マスクなしアライン・ユニット・ストライド・ロード: 1
  リマーク #15449: マスクなしアライン・ユニット・ストライド・ストア: 2
  リマーク #15452: マスクなしストライドロード: 1
  リマーク #15475: --- ベクトルのコストサマリー開始 ---
  リマーク #15476: スカラーのコスト: 14
  リマーク #15477: ベクトルのコスト: 7.000
  リマーク #15478: スピードアップの期待値: 1.990
  リマーク #15488: --- ベクトルのコストサマリー終了 ---

  ループの開始 C:¥multiply_simd2.c(14,3)
    リマーク #15548: ループと外部ループがベクトル化されました。
  ループの終了
ループの終了

ループの開始 C:¥multiply_simd2.c(11,2)
<ベクトル化の剰余ループ>

  ループの開始 C:¥multiply_simd2.c(14,3)
    リマーク #25460: ループの最適化はレポートされませんでした。
  ループの終了
ループの終了
================================================================================
```

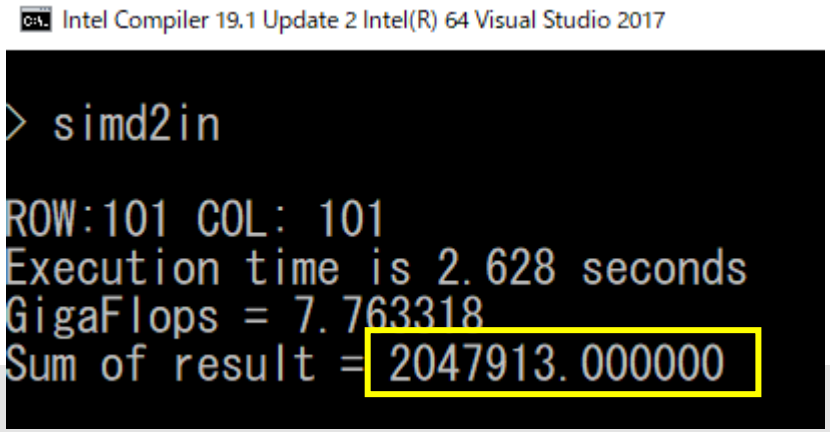
#pragma forceinline はコンパイラーに対する指示ですが、omp simd が優先されるため、インライン展開されていません

omp simd プラグマの位置を変えてみる

```
#define REPEATNTIMES 1000000
…
for (k = 0;k < REPEATNTIMES;k++) {
#pragma forceinline
      matvec(size1,size2,a,b,x);
      x[0] = x[0] + 0.000001;
}
```

コンパイラー・オプション:
/O2
/Qopt-report5
/Qopt-report-phase:vec,loop,ipo
/Qopt-report-annotate-position:both

```
void matvec(int size1, int size2, FTYPE a[][size2], FTYPE b[], FTYPE x[]) {
     int i, j;

for (i = 0; i < size1; i++) {
          b[i] = 0;
#pragma omp simd safelen(4) aligned(a, b, x:16)
          for (j = 0;j < size2; j++) {
               b[i] += a[i][j] * x[j];
          }
     }
}
```



最適化レポートを確認

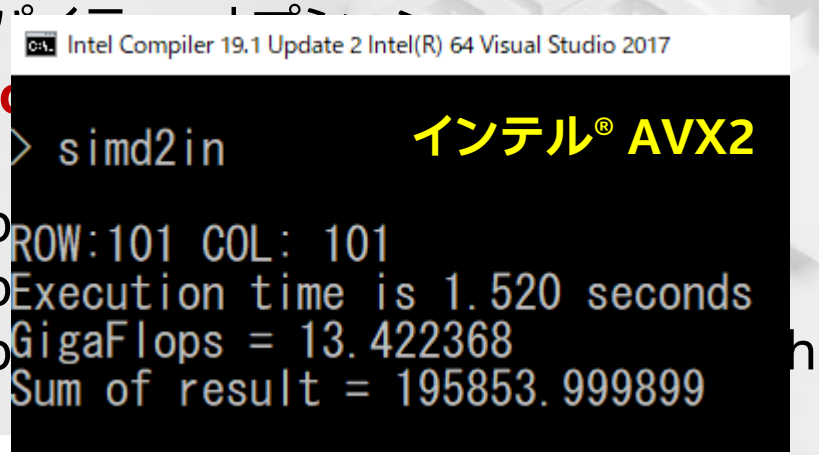
```
ループの開始 C:¥multiply_simd2.c(11,2)
  リマーク #15542: ループ はベクトル化されませんでした: 内部ループがすでにベクトル化されています。

  ループの開始 C:¥multiply_simd2.c(14,3)
    リマーク #15389: ベクトル化のサポート: 参照 a[i][j] にアラインされていないアクセスが含まれています。
[ C:¥multiply_simd2.c(15,4) ]
    リマーク #15388: ベクトル化のサポート: 参照 x[j] にアラインされたアクセスが含まれています。
[ C:¥matmul¥multiply_simd2.c(15,4) ]
    リマーク #15381: ベクトル化のサポート: ループ本体内でアラインされていないアクセスが使用されました。
    リマーク #15305: ベクトル化のサポート: ベクトル長 4
    リマーク #15399: ベクトル化のサポート: アンロールファクターが 2 に設定されます。
    リマーク #15309: ベクトル化のサポート: 正規化されたベクトル化のオーバーヘッド 0.184
    リマーク #15301: SIMD ループ がベクトル化されました。
    リマーク #15448: マスクなしアライン・ユニット・ストライド・ロード: 1
    リマーク #15450: マスクなし非アライン・ユニット・ストライド・ロード: 1
    リマーク #15475: --- ベクトルのコストサマリー開始 ---
    リマーク #15476: スカラーのコスト: 10
    リマーク #15477: ベクトルのコスト: 4.750
    リマーク #15478: スピードアップの期待値: 2.060
    リマーク #15488: --- ベクトルのコストサマリー終了 ---
    リマーク #25456: ループで置換された配列参照スカラーの数: 1
  ループの終了
```

```
リマーク #15475: --- ベクトルのコストサマリー開始 ---
リマーク #15476: スカラーのコスト: 14
リマーク #15477: ベクトルのコスト: 7.000
リマーク #15478: スピードアップの期待値: 1.990
リマーク #15488: --- ベクトルのコストサマリー終了 ---
```

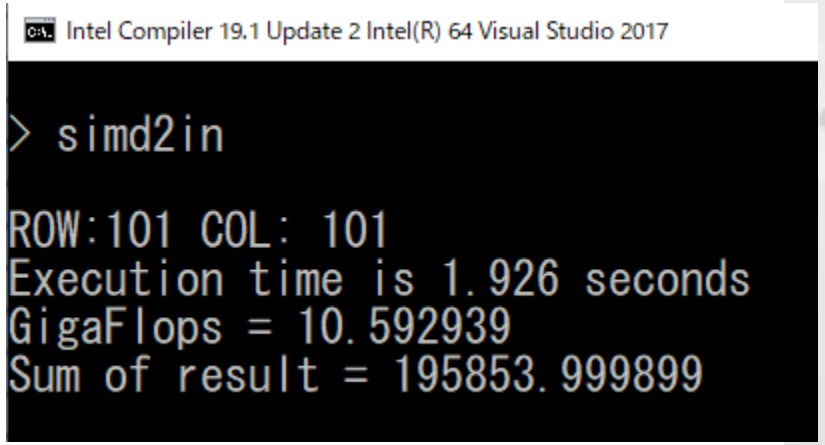
omp simd プラグマの位置を変えてみる + ipo

```
#define REPEATNTIMES 1000000
…
for (k = 0;k < REPEATNTIMES;k++) {
#pragma forceinline
      matvec(size1,size2,a,b,x);
      x[0] = x[0] + 0.000001;
}
```

コンパイラー・オプション
/Qipo
/O2
/Qop
/Qop
/Qop

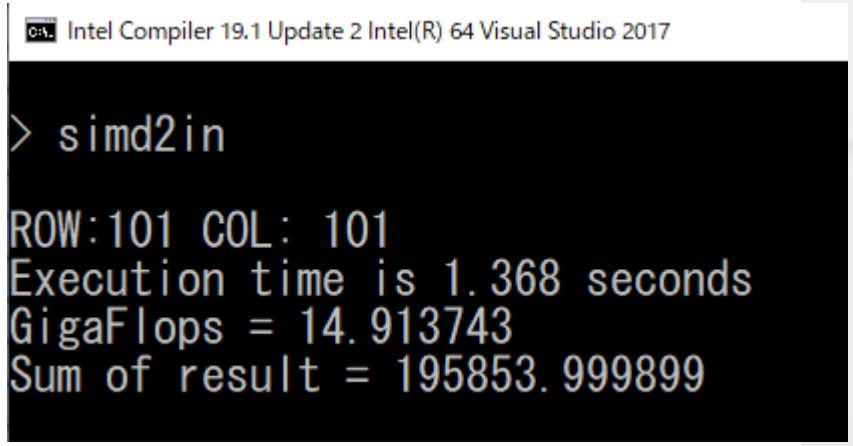



```
void matvec(int size1, int size2, FTYPE a[][size2], FTYPE b[], FTYPE x[]) {
     int i, j;

for (i = 0; i < size1; i++) {
          b[i] = 0;
#pragma omp simd safelen(4) aligned(a, b, x:16)
          for (j = 0;j < size2; j++) {
               b[i] += a[i][j] * x[j];
          }
     }
}
```



さらに…

```
void matvec(int size1, int size2, FTYPE a[][size2], FTYPE b[], FTYPE x[]) {
     int i, j;

for (i = 0; i < size1; i++) {
          b[i] = 0;
#pragma omp simd safelen(8) aligned(a, b, x:16)
          for (j = 0;j < size2; j++) {
               b[i] += a[i][j] * x[j];
          }
     }
}
```



■ safelen を大きくすると、コンパイラーはベクトル化に加えてアンロールを積極的に導入する

リマーク #15305: ベクトル化のサポート: ベクトル長 4
リマーク #15309: ベクトル化のサポート: 正規化されたベクトル化のオーバーヘッド 2.500

リマーク #15305: ベクトル化のサポート: ベクトル長 8
リマーク #15399: ベクトル化のサポート: アンロールファクターが 2 に設定されます。
リマーク #15309: ベクトル化のサポート: 正規化されたベクトル化のオーバーヘッド 1.029

まとめ

- omp simd による明示的なベクトル化は有効であるが、
  - プラグマ/ディレクティブを挿入する位置は重要
  - コンパイラーに詳細情報を伝える各種の指示節も重要
  - コンパイラー・オプションとコンパイラーの最適化機能との相互関係も重要
- コンパイラーや各種ツールを活用して適切で効果的な位置にプラグマ/ディレクティブを挿入してください
- 導入後の評価も忘れずに

iSUS

ソフトウェア・セミナー