インテルコンパイラー概要

エクセルソフト株式会社

インテル® Parallel Studio XE
包括的なソフトウェア開発ツールスイート

Composer Edition

ビルド
コンパイラーとライブラリー

インテル® C/C++、Fortran コンパイラー

インテル® MKL[1]

インテル® DAAL[2]

インテル® TBB[3]
C++ スレッド・ライブラリー

インテル® IPP[4]
画像、信号、データ処理

インテル® Distribution for Python*
ハイパフォーマンスな Python*

Professional Edition

解析
解析ツール

インテル® VTune™ Amplifier
パフォーマンス・プロファイラー

インテル® Inspector
メモリー/スレッドのデバッガー

インテル® Advisor
ベクトル化の最適化、スレッドのプロトタイプ生成、フローグラフ解析

Cluster Edition

スケール
クラスターツール

インテル® MPI ライブラリー
メッセージ・パッシング・インターフェイス・ライブラリー

インテル® Trace Analyzer & Collector
MPI チューニングと解析

インテル® Cluster Checker
クラスター診断エキスパート・システム

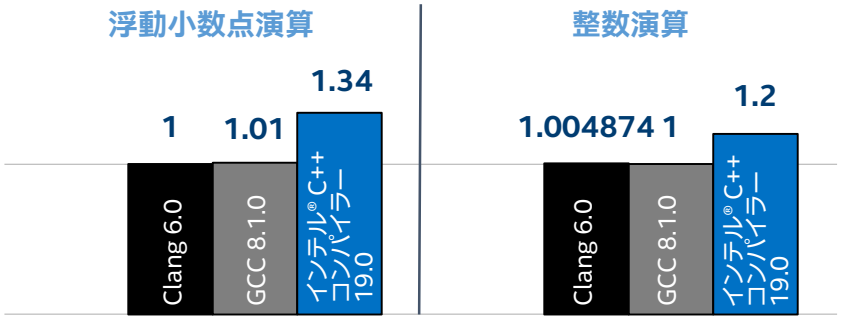
オペレーティング・システム: Windows*、Linux*、macOS*[5]

インテル® アーキテクチャー・ベースのプラットフォーム

[1] インテル® マス・カーネル・ライブラリー
[2] インテル® データ・アナリティクス・アクセラレーション・ライブラリー
[3] インテル® スレッディング・ビルディング・ブロック
[4] インテル® インテグレーテッド・パフォーマンス・プリミティブ
[5] インテル® Parallel Studio XE Composer Edition でのみ利用可能

インテル® C++/Fortran コンパイラーによる優れたアプリケーション・パフォーマンス — Linux* (数値が大きいほど高性能)

インテル® C++ コンパイラーによる優れた C++ アプリケーション・パフォーマンス — Linux*

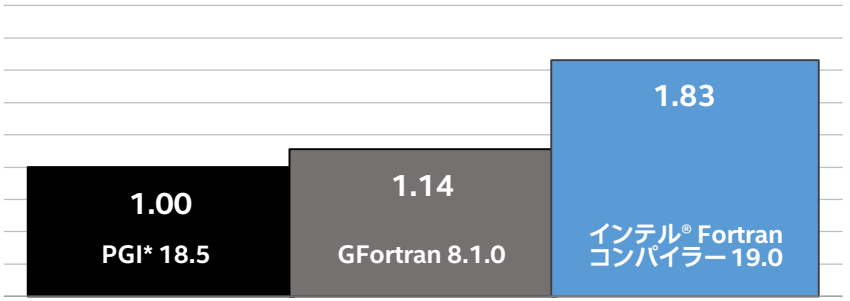


SPEC* CPU2017 の幾何平均の推定値
浮動小数点レートベースの C/C++ ベンチマーク

SPECint*_rate_base2017 の推定値

相対 (相乗平均) パフォーマンス SPEC* ベンチマーク (数値が大きいほど高性能)

インテル® Fortran コンパイラーによる優れた Fortran アプリケーション・パフォーマンス — Linux*



相対 (相乗平均) パフォーマンス Polyhedron* ベンチマーク (数値が大きいほど高性能)

パフォーマンス結果は 2018 年 8 月 26 日時点のテスト結果に基づいたものであり、公開されている利用可能なすべてのセキュリティー・アップデートが適用されていない可能性があります。詳細については、構成の開示を参照してください。絶対的なセキュリティーを提供できる製品はありません。性能に関するテストに使用されるソフトウェアとワークロードは、性能がインテル® マイクロプロセッサー用に最適化されていることがあります。SYSmark* や MobileMark* などの性能テストは、特定のコンピューター・システム、コンポーネント、ソフトウェア、操作、機能に基づいて行ったものです。結果はこれらの要因によって異なります。製品の購入を検討される場合は、他の製品と組み合わせた場合の本製品の性能など、ほかの情報や性能テストも参考にして、パフォーマンスを総合的に評価することをお勧めします。詳細については、パフォーマンス・ベンチマーク・テストの開示 (英語) を参照してください。

2018 年 8 月 26 日時点のインテルによるテスト。システム構成: ハードウェア: インテル® Xeon® Platinum 8180 プロセッサー @ 2.50GHz、384GB RAM、ハイパースレッディング有効。ソフトウェア: インテル® コンパイラー 19.0、GCC 8.1.0、PGI* 18.5、Clang/LLVM 6.0。Linux* OS: Red Hat* Enterprise Linux* Server 7.4 (Maipo)、3.10.0-693.el7.x86_64。SPEC* ベンチマーク (www.spec.org (英語))。SPECint* ベンチマーク測定時の CXX テストには SmartHeap* 10 を使用。SPECint*_rate_base_2017 コンパイラー・オプション: C++ テストには SmartHeap* 10 を使用。インテル® C/C++ コンパイラー 19.0: -xCORE-AVX512 -ipo -O3 -no-prec-div -qopt-mem-layout-trans=3。GCC 8.1.0: -march=znver1 -mfpmath=sse -Ofast -funroll-loops -flto。Clang 6.0: -march=core-avx2 -mfpmath=sse -Ofast -funroll-loops –flto。SPECfp*_rate_base_2017 コンパイラー・オプション: インテル® C/C++ コンパイラー 19.0: -xCORE-AVX512 -ipo -O3 -no-prec-div -qopt-prefetch -ffinite-math-only -qopt-mem-layout-trans=3。GCC 8.1.0: -march=skylake-avx512 -mfpmath=sse -Ofast -fno-associative-math -funroll-loops -flto。Clang 6.0: -march=znver1 -mfpmath=sse -Ofast -funroll-loops -flto。SPECint*_speed_base_2017 コンパイラー・オプション: C++ テストには SmartHeap* 10 を使用。インテル® C/C++ コンパイラー 19.0: -xCORE-AVX512 -ipo -O3 -no-prec-div -qopt-mem-layout-trans=3 -qopenmp。GCC 8.1.0: -march=znver1 -mfpmath=sse -Ofast -funroll-loops -flto -fopenmp。Clang 6.0: -march=core-avx2 -mfpmath=sse -Ofast -funroll-loops -flto -fopenmp=libomp。SPECfp*_speed_base_2017 コンパイラー・オプション: インテル® C/C++ コンパイラー 19.0: -xCORE-AVX512 -ipo -O3 -no-prec-div -qopt-prefetch -ffinite-math-only -qopenmp。GCC 8.1.0: -march=skylake-avx512 -mfpmath=sse -Ofast -fno-associative-math -funroll-loops -flto -fopenmp。Clang 6.0: -march=skylake-avx512 -mfpmath=sse -Ofast -funroll-loops -flto -fopenmp=libomp。

2018 年 8 月 26 日時点のインテルによるテスト。システム構成: ハードウェア: インテル® Core™ i7-8700K プロセッサー @ 3.70GHz、64GB RAM、ハイパースレッディング有効。ソフトウェア: インテル® Fortran コンパイラー 19.0、PGI* Fortran 18.5、GFortran 8.1.0。Linux* OS: Red Hat* Enterprise Linux* Server 7.4 (Maipo)、3.10.0-693.el7.x86_64。Polyhedron* Fortran ベンチマーク (www.fortran.uk (英語))。Linux* コンパイラー・オプション: GFortran: -Ofast -mfpmath=sse -flto -march=haswell -funroll-loops -ftree-parallelize-loops=6。インテル® Fortran コンパイラー: -fast -parallel -xCORE-AVX2 -nostandard-realloc-lhs。PGI* Fortran: -fast -Mipa=fast,inline -Msmartalloc -Mfprelaxed -Mstack_arrays -Mconcur=bind -tp haswell。

インテル® コンパイラーの最適化オプション

段階的なパフォーマンス・チューニング
1つの変更のみを行い、変更毎に性能と動作を確認し、評価する

1. 一般的な最適化
2. 特定のプロセッサー向け最適化
3. 自動並列化 (マルチスレッド化)
4. IPO (プロシージャー間の最適化)
   および PGO (プロファイルに基づく最適化)

一般的な最適化

デフォルトの /O2 (-O2) で動作と性能を確認する

- /O1 ではベクトル化が行われない
- ループが多用されるアプリケーションでは、/O3 (-O3) を試す

/O1, /O2, /O3 (-O1, -O2, -O3) の
いずれかを基準として、他のオプションを付け加えていく

コンパイルまたは実行に失敗する場合には、/Od (-O0) で
コンパイラーの最適化の影響を外し、問題を切り分ける

一般的な最適化オプション

※いずれか1つのみ有効

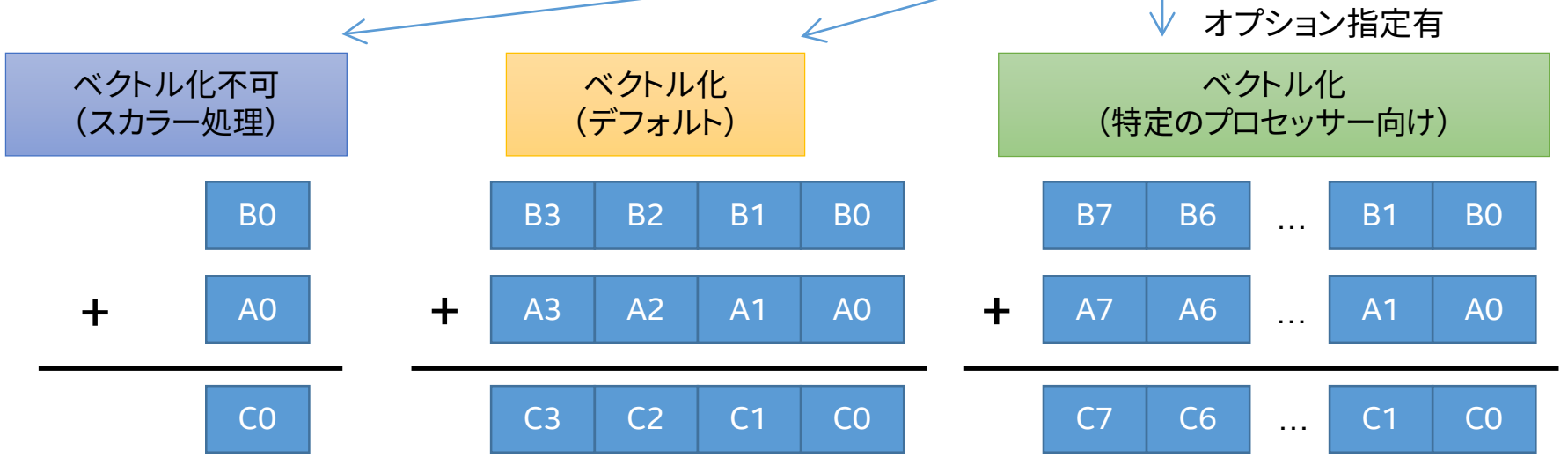
Windows*	Linux*	
/Od	-O0	最適化を行いません。 アプリケーション開発の初期段階およびデバッグ時に指定します。
/O1	-O1	コードサイズを増やさずに実行速度を向上させます。 大きなコードサイズに起因するメモリーページングが問題となる 大規模なサーバー/データベース・アプリケーションで有効です。
/O2	**-O2**	実行速度を最大化します（デフォルト設定）。 ベクトル化を含む、多くの最適化を有効にします。 多くの場合 /O1 (-O1) よりも高速なコードを生成します。
/O3	-O3	/O2 (-O2) に加え、より積極的なループとメモリーアクセスの最適化を有効にします（スカラー置換、ループアンロール、キャッシュの効率的な利用のためのループ・ブロッキング、データ・プリフェッチなど）。 これらの積極的な最適化の結果、アプリケーションの種類により /O2 (-O2) と比べて遅くなることもあります。

ベクトル化

ループ処理について SIMD 操作を
用いるよう解釈し、実行効率を高める

ソースコード（C/C++）

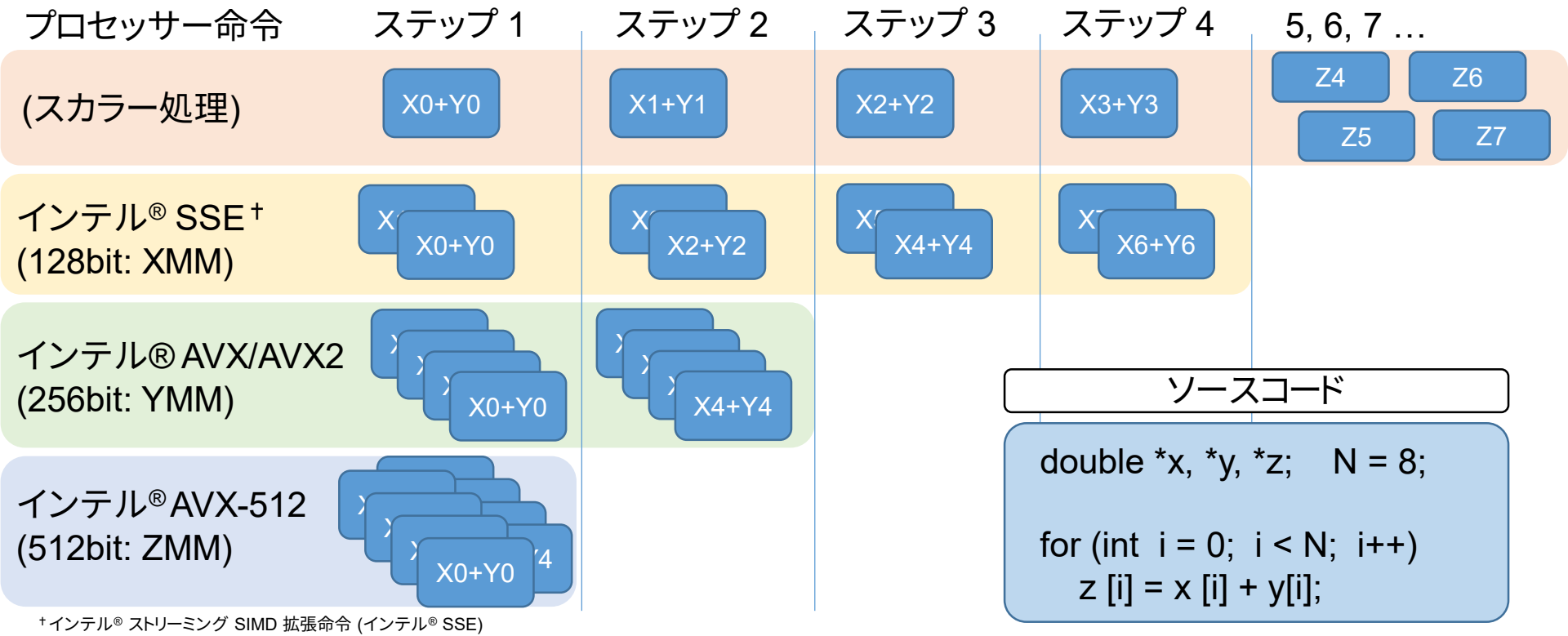
```
for ( i=0; i<N; i++ ) {
    C[i] = A[i] + B[i];  }
```



SIMD : Single Instruction Multiple Data

SIMD 拡張命令セットの進歩

1回の命令実行で、より多数の計算結果を得る



†インテル® ストリーミング SIMD 拡張命令 (インテル® SSE)

特定のプロセッサー向け最適化

/QxHOST (-xhost) を追加して性能と動作を確認する

- /O2 (-O2) または /O3 (-O3) は
  デフォルトの命令セットにより自動ベクトル化を行う
- /Qvec- (-no-vec) により自動ベクトル化を無効にできる

アプリケーションを実行するシステムのプロセッサーを確認する

- コンパイルしたプログラムを別のシステムへ移動させて
  実行させる場合には、適当な /Qx (-x) オプションを設定する
- 実行するシステムが一様でない場合には、
  代わりに適当な /Qax (-ax) オプションを指定する

特定のプロセッサー向け最適化オプション

/Qx<target>、 -x<target>

※命令セットはいずれか1つのみ有効

<target> で指定した命令セットをサポートするインテル® プロセッサー向けの
専用コードを生成します。実行可能ファイルは、非インテル製プロセッサーや、
下位の命令セットのみをサポートするインテル製プロセッサー上では動作しません。
<target> に指定可能な命令セットは次のとおりです（上位から下位）。

CORE-AVX512, MIC-AVX512, COMMON-AVX512, CORE-AVX2, CORE-AVX-I, AVX, SSE4.2, ATOM_SSE4.2, SSE4.1, ATOM_SSSE3, SSSE3, SSE3, SSE2

注: **MIC-AVX512** (インテル® Xeon Phi™ プロセッサー x200 製品ファミリー向けの命令セット) は **CORE-AVX512** のサブセットではありません。
COMMON-AVX512 は両方の共通のサブセットです。

特定のプロセッサー向け最適化オプション

/QxHOST （ -xhost ）

- コンパイルを行うホストシステム上でサポートされる
  最上位の命令セットを利用するコードを生成

/arch:<target> （ -m<target> ）

- <target> で指定した命令セットをサポートするインテル® プロセッサーと
  非インテル製の互換プロセッサー向けに専用コードを生成
- target: **SSE2**, SSE3, SSSE3, SSE4.1, SSE4.2, AVX
  および CORE-AVX2 (Linux では -march=core-avx2)

適したオプションの確認方法：型番で検索し、“命令セット拡張” を確認
https://ark.intel.com/ja 例：Core i7-6770、Xeon E5-2620 v4

特定のプロセッサー向け最適化オプション

/Qax<target>、 -ax<target>

※命令セットはカンマ区切りで複数指定可

<target> で指定した命令セットをサポートするインテル® プロセッサー向けの
専用コードとデフォルトコードを生成します。

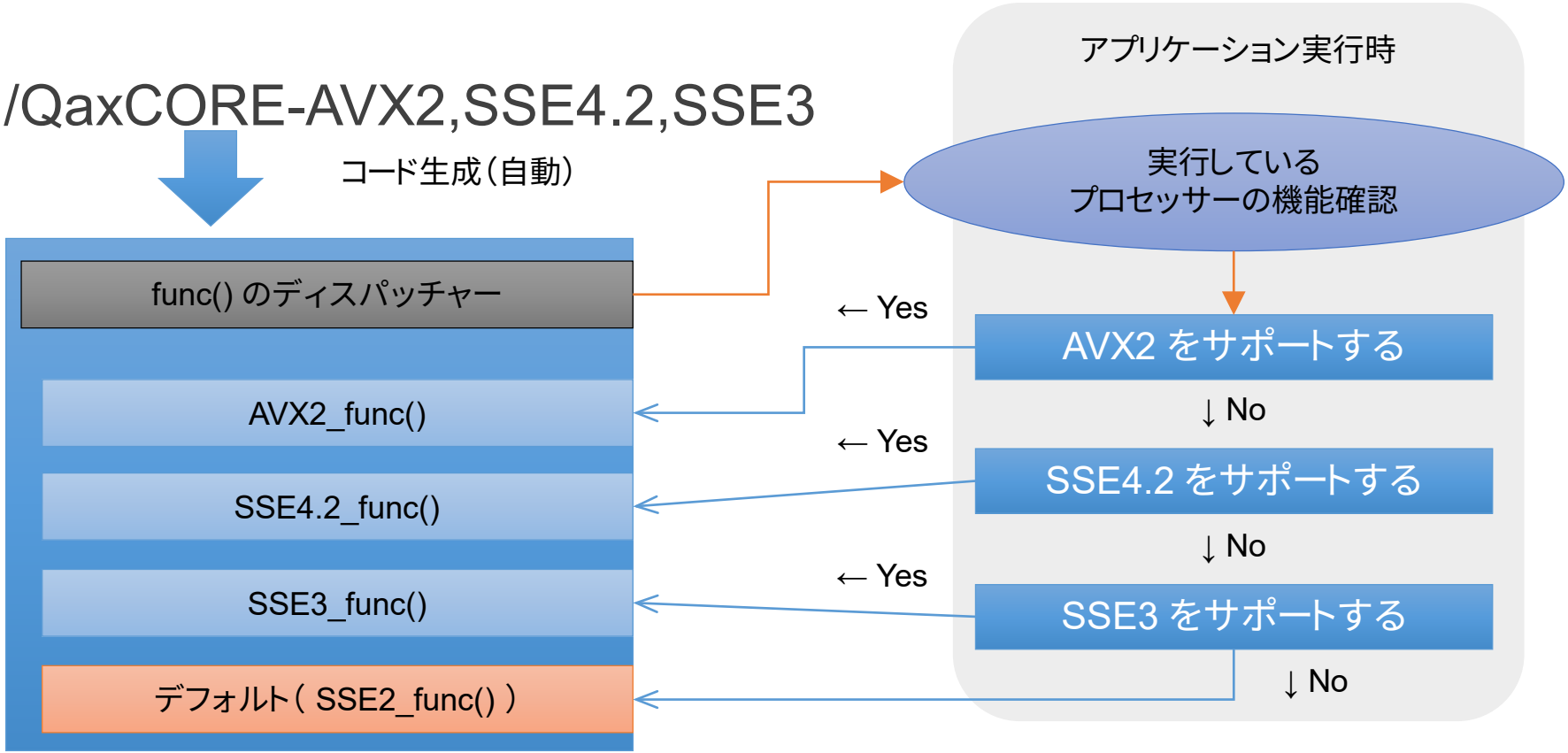
アプリケーションは実行時にインテル製のプロセッサーで実行されているか自動検出し、最適なコードパスを選択します。インテル製のプロセッサーが検出されなかった場合、デフォルトのコードパスが選択されます。
<target> に指定可能な命令セットは次のとおりです（上位から下位）。

CORE-AVX512, MIC-AVX512, COMMON-AVX512, CORE-AVX2,
CORE-AVX-I, AVX, SSE4.2, SSE4.1, SSSE3, SSE3, SSE2

/Qax, -ax オプション適用時の動作

/QaxCORE-AVX2,SSE4.2,SSE3

コード生成（自動）



自動並列化（マルチスレッド化）

/Qparallel (-parallel) を追加して性能と動作を確認する

- 並列スレッド数の指定
  - /Qpar-num-threads:<n> (-par-num-threads=<n>)
  - 環境変数 ループ・スケジュールの変更OMP_NUM_THREADS

  - /Qpar-schedule-static-steal (-par-schedule-static-steal)
  - /Qpar-schedule-auto (-par-schedule-auto)

システムの CPU 数、コア数とスレッド数を確認する

- 性能向上の理論的な最大値は（物理）コア数

自動並列化の基本動作

ループ処理を対象とし、
ループインデックス
(0, 1, 2, ... , N) を各スレッドへ
分配できるよう変換する

ソースコード（C/C++）

```
for ( i = 0 ; i < N ; i++ ) {
   // 何らかの処理
}
```

変換

```
for ( i = begin , i < end , ++ ) {
  // 何らかの処理
}
```

スケジュール例 (4 スレッドで均等に分配、N=128)

タスク1		タスク2		タスク3		タスク4	
begin	end	begin	end	begin	end	begin	end
0	32	32	64	64	92	92	128
0 から 31 まで		32 から 63 まで		64 から 91 まで		92 から N まで	

IPO および PGO

/Qipo（ -ipo ）オプションを追加して性能と動作を確認する

- ソースコードが複数ファイルで構成されており、
  すべてのソースコードへアクセス可能な場合に有効
- ファイルをまたぐインライン展開などにより
  コンパイラーの最適化が改善されることがある

プロファイルに基づく最適化（PGO）により、これまで適用した
コンパイラーによる最適化が改善されるかどうか確認する

- /Qprof-gen と /Qprof-use（ -prof-gen と -prof-use ）

プロシージャー間の最適化（IPO）とは

ソースコード全体を解析し、特定の最適化が有効かどうか検証する

有効な最適化

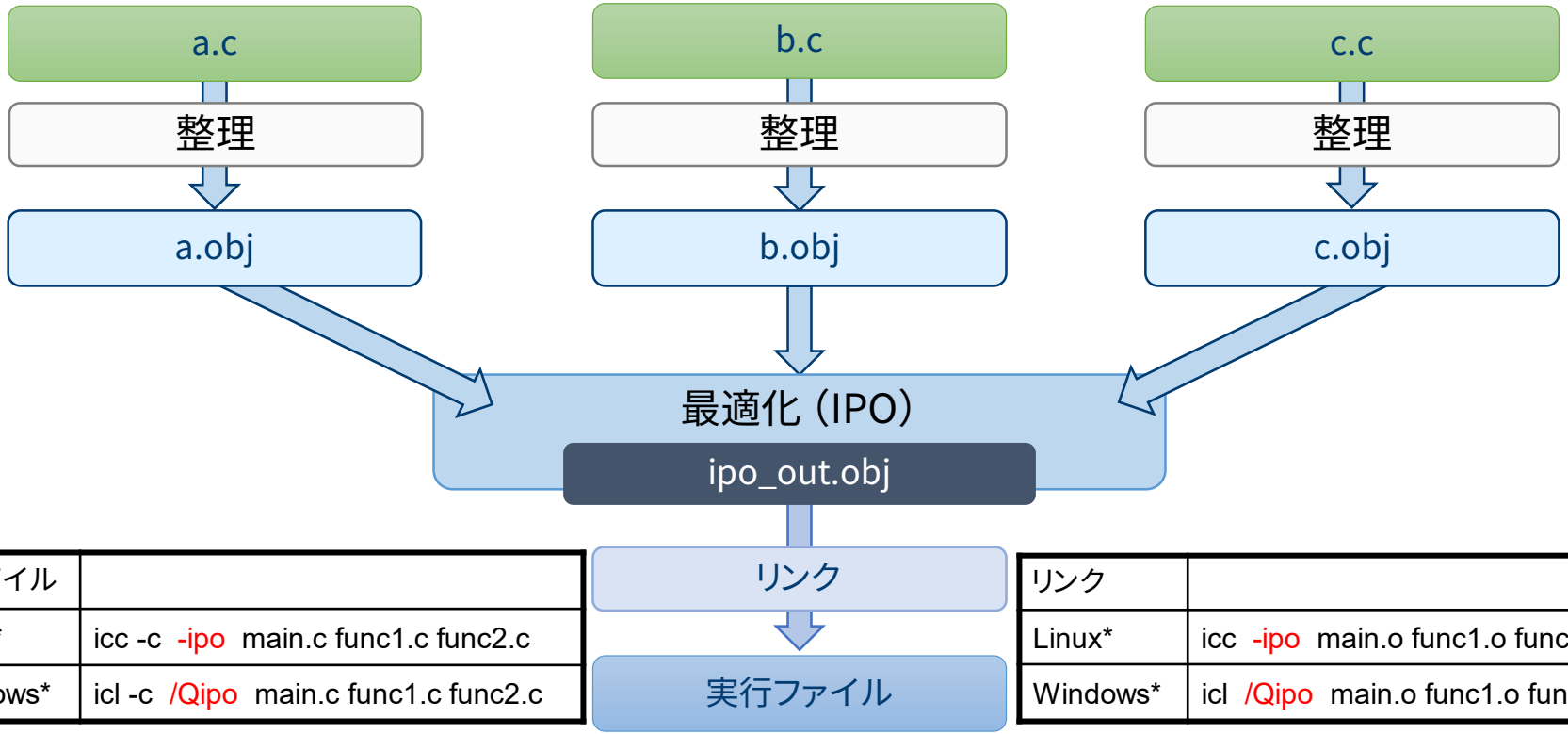
- 関数/サブルーチンのインライン展開
- プロシージャー間での定数伝播
- ポインターやアドレスの解析
- 不要なコード（関数定義や呼び出し、変数）の削除

他のコンパイラーでの類似機能（非互換）：
GCC, Clang/LLVM : リンク時の最適化（LTO） -flto
Microsoft* Visual C++* : プログラム全体の最適化（WPO） /GL, /LTCG

プロシージャー間の最適化（IPO）オプション

Windows*	Linux*	
/Qip	-ip	現在のソースファイルを対象にしたインライン展開を含む、単一ファイル内でのプロシージャ感の最適化を行います。
/Qipo[n]	-ipo[n]	複数のソースファイルにまたがるインライン展開やその他のプロシージャー間の最適化を許可します。オプションの引数 n は、リンク時コンパイルの最大数（オブジェクト・ファイル数）を制御します。デフォルトは n=0 です（コンパイラーが判断します）。 **注**：状況によりコンパイル時間とコードサイズが大幅に増えることがあります。
xilink xilib	xild xiar	/Qipo (-ipo) オプションを指定した場合に使用される、リンカーおよびアーカイバのラッパーコマンドです。

複数ファイルをまたぐ処理（2つのステップ）



コンパイル	
Linux*	icc -c -ipo main.c func1.c func2.c
Windows*	icl -c /Qipo main.c func1.c func2.c

リンク	
Linux*	icc -ipo main.o func1.o func2.o
Windows*	icl /Qipo main.o func1.o func2.o

C のインライン展開例

ファイルをまたいだ関数呼び出しがあると有効
→ 関数の内部処理を呼び出し先に展開

ファイル構成：
square_charge.h：関数宣言
square_charge.c：main 関数

twod_int.c
trap_int.c
func.c　：以下関数の定義

```
// twod_int() の一部, ny = 10000
for ( j=1; j<ny; j++ ) {
    y = y0 + j*g;
    sumy = sumy + trap_int(y,x0,xn,nx,xp,yp);
}
```

```
// trap_int() の一部, nx = 10000
for ( i=1; i<nx; i++ ) {
    x = x0 + i*h;
    sumx = sumx + func(x,y,xp,yp);
}
```

```
float func(float x, float y, float xp, float yp) {
    float denom;

    denom = (x-xp)*(x-xp) + (y-yp)*(y-yp);
    denom = 1.0f/sqrtf(denom);

    return denom;    }
```

プロファイルに基づく最適化（PGO）とは

アプリケーション実行時の情報をフィードバックして最適化を検証する
- 命令キャッシング、ページング、分岐予測を支援

有効な最適化
- 基本ブロック、関数の並び替え
- 適切なレジスター割り当て
- Switch 文の最適化
- 関数のインライン展開における効率性判断
- 自動ベクトル化、自動並列化における効率性判断

プロファイルに基づく最適化（PGO）オプション

Windows*	Linux*	
/Qprof-gen[:kywd]	-prof-gen[=kywd]	プロファイル情報を生成するためにインストルメントを行います。kywd= threadsafe はスレッド化されたアプリケーションのプロファイル生成を可能とします。kywd= srcpos と globdata は、関数やデータの並び替えに役立つ追加情報を収集します。
/Qprof-use	-prof-use	最適化に収集されたプロファイル情報を使用します。
/Qprof-dir <dir>	-prof-dir <dir>	プロファイル結果の出力ファイル（*.dyn および *dpi）を格納するディレクトリーを指定します。 代わりに環境変数 PROF_DIR により指定することもできます。

PGO：適用のための 3 つのフェーズ

フェーズ1：インストルメント済みの実行ファイルを生成

Linux* : icc -o prog-inst.exe -prof-gen ... prog.c

Windows* : icl /Fe: prog-inst.exe /Qprof-gen ... prog.c

フェーズ2：典型的なデータセットを用いてプログラムを実行

プロファイル情報が *.dyn ファイルに格納される

条件によりコードパスが異なる場合は複数回実施

コンパイル前に *.dyn ファイルがマージされた pgopti.dpi が生成される

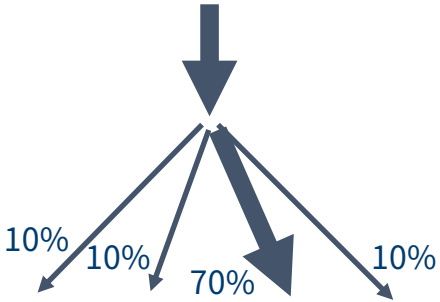
フェーズ3：フィードバック・コンパイル

Linux* : icc -o prog-release.exe -prof-use ... prog.c

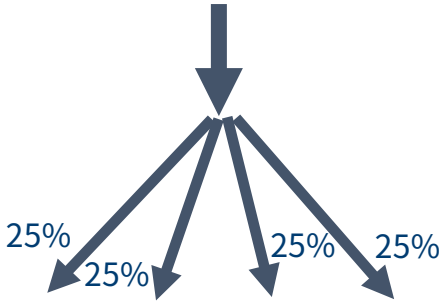
Windows* : icl /Fe: prog-release.exe /Qprof-use ... prog.c

PGO が有効となるプログラム

一貫性のあるホットパス
多数のループや if 文または switch 文
入れ子の if 文または switch文



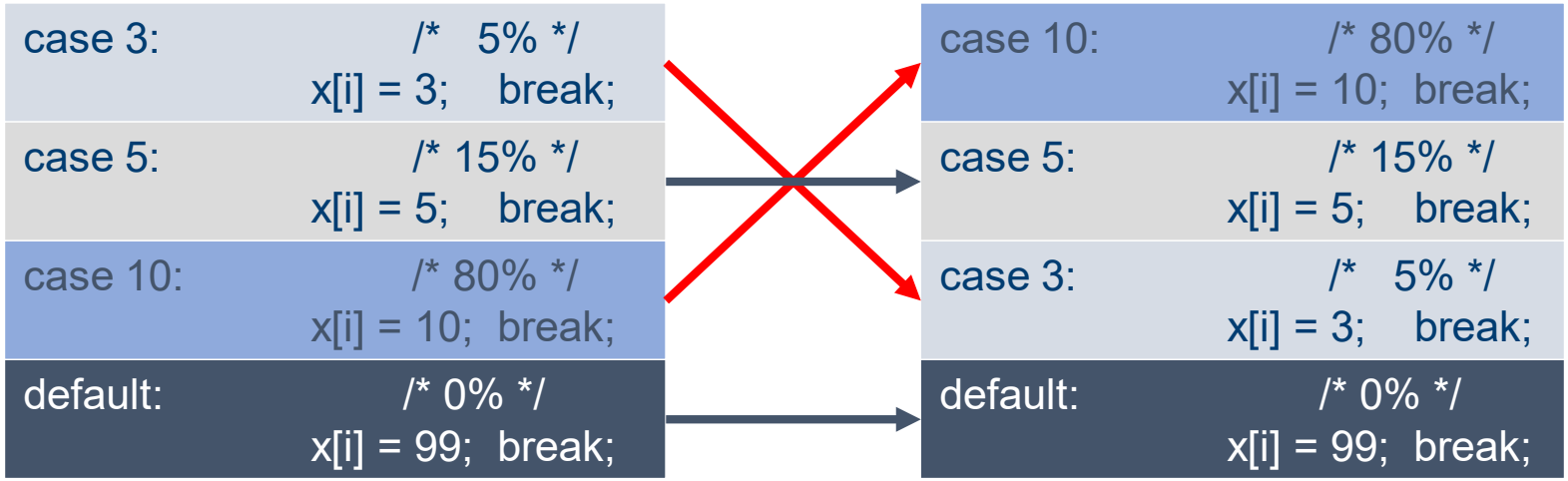
多大な利点



多少の利点

PGO : Switch 文の最適化

```
for ( int i = 0;  i <  NUM_BLOCKS;  i++ ) {
      switch ( check(i) ) {
```




```
    }
}
```

PGO による2度目のコンパイルでは各 case の頻度を
考慮してコードを生成し、プロセッサーによる分岐予測を助ける

主なオプション

プログラム全体に対して実行速度の最大化を試みる

/fast (Windows*), -fast (Liunx*)

- 以下の各オプションを指定することと同じです

OS	オプションの組み合わせ
macOS*	-O3 -ipo -no-prec-div -mdynamic-no-pic -fp-model fast=2 -xHost
Linux*	-O3 -ipo -no-prec-div **-static** -fp-model fast=2 -xHost
Windows*	/O3 /Qipo /Qprec-div- /fp:fast=2 /QxHost

/Qprec-div- , -no-prec-div : 割り算(除法)について精度が低いが高速な方法を用いることを許可します

Linux での -static について: 最近の Linux ディストリビューションでは、GCC や glibc の
静的リンク版ライブラリーを別途インストールする必要があります
対応が難しい場合は代わりに -static を除いたオプションの組み合わせを指定します

最適化オプションのまとめ

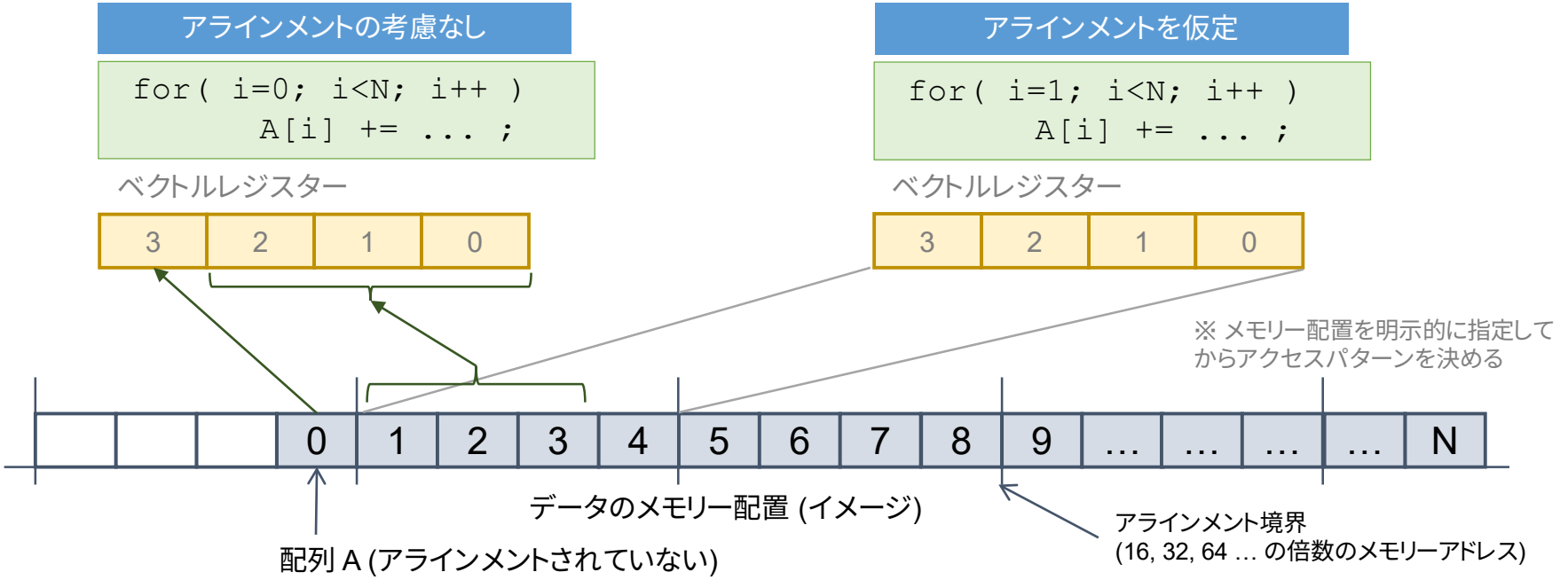
	Windows*	Linux* / macOS*
一般的な最適化	/O3	-O3
プロセッサー固有の最適化	/QxHOST	-xhost
自動並列化	/Qparallel	-parallel
プロシージャー間の最適化	/Qipo	-ipo
プロファイルに基づく最適化	→ スライド 24 参照	
浮動小数点数値演算の最適化制御	/fp:fast=2	-fp-model fast=2

※ コンパイルエラーを除いてコードの変更を伴わない、実行時間の最小化を意図した選択

アライメントについて

データ配置とアクセスパターン

アラインメントはベクトル演算の高速化に重要



ベクトル化されたループの内訳

ベクトル化されたボディー

- ループの大部分

最高速！

オプションのピール (剥離) 領域

- ループ中のアライメントされない参照に利用スカラーまたは低速なベクトル命令を使用

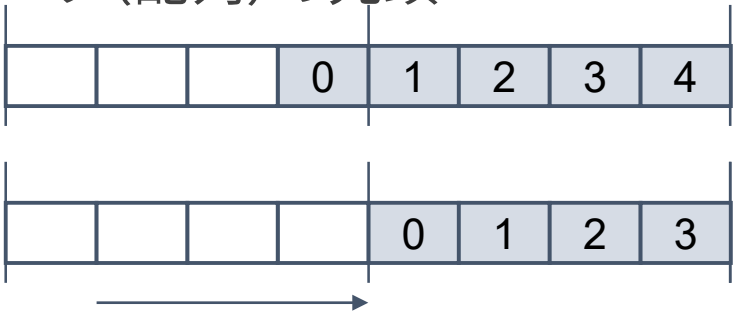
リマインダー (余剰) 領域

- ループ反復数がベクトル長で割り切れない場合に利用スカラーまたは低速なベクトル命令を使用

それほど高速ではない

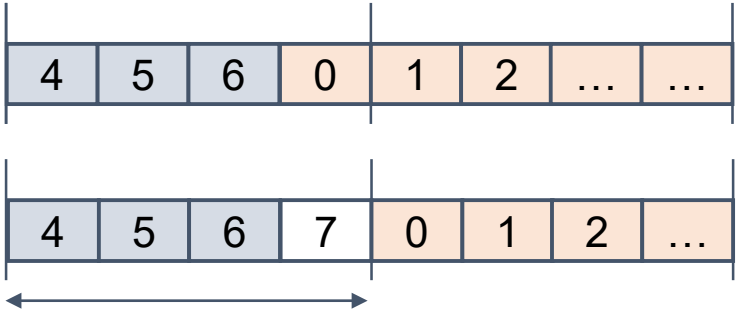
明示的なメモリー・アラインメント

データ（配列）の先頭



アラインメントを指定する

- データの中間点
  （2次元以上の配列や、構造体の配列）



1 次元目の要素数、または構造体の大きさが
指定の倍数になるように空の要素を入れる

早見表	アラインメント境界 (バイト数)		
大きさ (バイト数)	16 (インテル® SSE)	32 (インテル® AVX)	64 (Xeon Phi / AVX-512)
int / float (32 bit 要素)	4	8	16
double (64 bit 要素)	2	4	8

メモリーアクセスのアライメントを指定する

アライメントの最適化は、SIMD ハードウェアでは重要です。

OpenMP* 4.0 では、aligned(n) 節を使用してアライメント情報をコンパイラーに知らせます。

コンパイラーが最新のインテル® アーキテクチャー・ベースのシステムで最適な SIMD コードを生成するには、8 バイト、16 バイト、32 バイト、64 バイト・アライメントを指定します。

Aligned 節を使ったアラインメントの指定

配列 *x* とポインター *y* はそれぞれ、32 バイト・アライメントとしてマークされています。

メモリー参照オフセット *n1* は、**mod 8 = 0** でアサートされています。

すべてのベクトルメモリー参照が 32 バイトでアライメントされていることをコンパイラーに知らせます。

```
void arrayref(float *restrict x, float *y, int n, int n1) {
        _assume(n1%8=0);
        #pragma omp simd aligned(y:32)
        for (int k=0; k<n; k++) {
                x[k] = x[k] + y[k] + y[k+n1] + y[k-n1];
        }
}
```

データのアラインメントを取る方法

C/C++

変数への指定

- __attribute__((aligned(n)))
  (変数定義の前または最後に追加)

メモリー確保関数

- Void* __mm_malloc (site_t size, size_t align)
- Void __mm_free (void *p)

Fortran

```
    double precision, allocatable  :: a(:,:), b(:,:), c(:,:)
!DIR$ attributes align:64 :: a, b, c
```

**aligned 節は対象のデータがアラインメントされていることを
コンパイラーに知らせますが、対象のデータをアラインメントする
機能を持ちません。**

OpenMP* SIMD を使ったプログラムのコンパイル

コンパイルオプション：-qopenmp-simd (Windows: /Qopenmp-simd)

```
$ icc|icpc -qopenmp-simd sample.c
```

```
$ ifort -qopenmp-simd sample.f90
```

インテル® C/C++ および Fortran コンパイラー19.0 では、デフォルトで /Qopenmp-simd (-qopenmp-simd) オプションが有効になります

- 無効にするには、-qopenmp-simd- を指定します