インテル® コンパイラーの概要

インテル コーポレーション 開発製品部門
シニア･テクニカル･コンサルティング･エンジニア
Igor Vorobtsov

内容

- インテル® コンパイラーを使用する理由
- 高レベルの最適化
- 最適化レポート
- プロシージャー間の最適化 (IPO)
- 自動ベクトル化
- プロファイルに基づく最適化 (PGO)
- 自動並列化

© 2018 Intel Corporation. 無断での引用、転載を禁じます。Intel、インテル、Intel ロゴは、アメリカ合衆国および / またはその他の国における Intel Corporation の商標です。
* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。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。

インテル® コンパイラーを使用する理由

ソフトウェア開発者にとって重要なポイント		課題	インテル® コンパイラーの利点
	パフォーマンス – 高速なアプリケーションを開発する必要がある	最新のハードウェア・イノベーションを利用しなければならない	最新の x86 互換プロセッサーと命令セットを最大限に利用できる
	生産性 – 生産性を向上でき、使いやすくなければならない	主要な言語とプログラミング・モデルがサポートされていなければならない	最新の Fortran、C/C++、OpenMP* 標準規格をサポートし、主要なコンパイラーおよび IDE と互換性がある
	スケーラビリティー – アプリケーションをローカルで開発・デバッグし、グローバルに配置する必要がある	急速に増え続けるコア数とベクトル幅に合わせてコードを保守しなければならない	新しい世代のプロセッサーでもコードを変更することなくスケーラブルなパフォーマンスを実現できる

© 2018 Intel Corporation. 無断での引用、転載を禁じます。Intel、インテル、Intel ロゴは、アメリカ合衆国および / またはその他の国における Intel Corporation の商標です。
* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。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。

一般的なコンパイラーの機能

多くの標準規格をサポート

- C11 と C++14 の完全サポート、C++ 17 の部分サポート
  - -std オプション (-std=c++14 など) を使用して制御
- Fortran 2008 の完全サポート、Fortran 2018 の部分サポート

インテル® C/C++ コンパイラーとソース/バイナリー互換

- Linux*: gcc
- Windows*: Visual C++* 2013 以上

ベクトル化ですべての命令セットをサポート

- インテル® ストリーミング SIMD 拡張命令 (インテル® SSE)
- インテル® アドバンスト・ベクトル・エクステンション (インテル® AVX)
- 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)

OpenMP* 4.5 の完全サポート

最適化された数学ライブラリー

- libimf (スカラー) および libsvml (ベクトル): GNU* libm よりも高速
- ドライバーは libm の前に libimf を自動的にリンク
- math.h を mathimf.h に置換

多くの高度な最適化

- 詳細で構造化された最適化レポート

© 2018 Intel Corporation. 無断での引用、転載を禁じます。Intel、インテル、Intel ロゴは、アメリカ合衆国および / またはその他の国における Intel Corporation の商標です。
* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。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。

高レベルの最適化

icc -O による基本的な最適化

-O0 最適化なし、デバッグ用に -g を設定

-O1 スカラーの最適化
　コードサイズが増える最適化を除く

-O2 icc/icpc の**デフォルト** (-g を除く)
　自動ベクトル化、一部のループ変換 (アンロール、ループ交換など) を含む
　ソースファイル内のインライン展開
　このオプションで開始 (-O0 でデバッグした後)

-O3 より積極的なループの最適化
　キャッシュ・ブロッキング、ループ融合、プリフェッチなどを含む
　浮動小数点演算を多用するループや大きなデータセットを処理するループを含むアプリケーションに最適

© 2018 Intel Corporation. 無断での引用、転載を禁じます。Intel、インテル、Intel ロゴは、アメリカ合衆国および / またはその他の国における Intel Corporation の商標です。
* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。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。

コンパイラーのレポート – 最適化レポート

最適化レポートの詳細レベルの指定

- `-qopt-report[=n]`
- 引数なしで指定した場合、詳細レベル 2 の最適化レポートが stdout に出力される

最適化レポートの出力の指定

- `-qopt-report=<filename>`
- オプションを指定しない場合、<filename>.optrpt ファイルが生成される

特定のフェーズのみの最適化レポート

- `-qopt-report-phase[=list]`
  指定できるフェーズ:
  - all – すべてのフェーズの最適化レポート (デフォルト)
  - loop – 入れ子のループおよびメモリーの最適化
  - vec – 自動ベクトル化および明示的なベクトル・プログラミング
  - par – 自動並列化
  - openmp – OpenMP* を使用したスレッド化
  - ipo – プロシージャー間の最適化 (インライン展開を含む)
  - pgo – プロファイルに基づく最適化
  - cg – コード生成

© 2018 Intel Corporation. 無断での引用、転載を禁じます。Intel、インテル、Intel ロゴは、アメリカ合衆国および / またはその他の国における Intel Corporation の商標です。
* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。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。

プロシージャー間の最適化 (IPO)

マルチパスの最適化

`icc -ipo`

関数/ソースファイル境界を解析および最適化

- 関数のインライン展開、定数の伝播、依存性解析、データ/コードレイアウトなど

2 ステップのプロセス

- コンパイルステップ – オブジェクトは中間表現を含む
- リンクステップ – すべてのオブジェクトをコンパイルして最適化
- シームレス: リンカーは -ipo およびコンパイラー・オプションを指定してビルドされたオブジェクトを自動的に検出
- 場合によっては、ビルド時間とバイナリーサイズが増えることがある
- `-ipo=n` を指定してビルドを並列化できる
- ホットなモジュールのみビルド (プログラム全体をビルドする必要はない)

多くの小さな関数を含むアプリケーションで特に効果的

`-qopt-report-phase=ipo` を指定してインライン展開された関数のレポートを取得

© 2018 Intel Corporation. 無断での引用、転載を禁じます。Intel、インテル、Intel ロゴは、アメリカ合衆国および / またはその他の国における Intel Corporation の商標です。
* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。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。

プロシージャー間の最適化 (IPO)

ファイル境界の最適化を拡張

`-ip`	1 つのソースファイルのモジュール間のみ
`-ipo`	複数のファイル/アプリケーション全体のモジュール

IPO なし

コンパイルおよび最適化 → file1.c

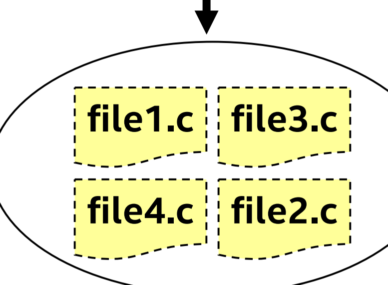
コンパイルおよび最適化 → file2.c

コンパイルおよび最適化 → file3.c

コンパイルおよび最適化 → file4.c

IPO あり

コンパイルおよび最適化



© 2018 Intel Corporation. 無断での引用、転載を禁じます。Intel、インテル、Intel ロゴは、アメリカ合衆国および / またはその他の国における Intel Corporation の商標です。
* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。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。

数学ライブラリー

icc はインテルの最適化された数学ライブラリーを利用

- libimf (スカラー) および libsvml (スカラー/ベクトル)
- GNU* libm よりも高速
- ドライバーは libm の前に libimf を自動的にリンク
- 追加の関数 (math.h を mathimf.h に置換)

libm を明示的にリンクしない!

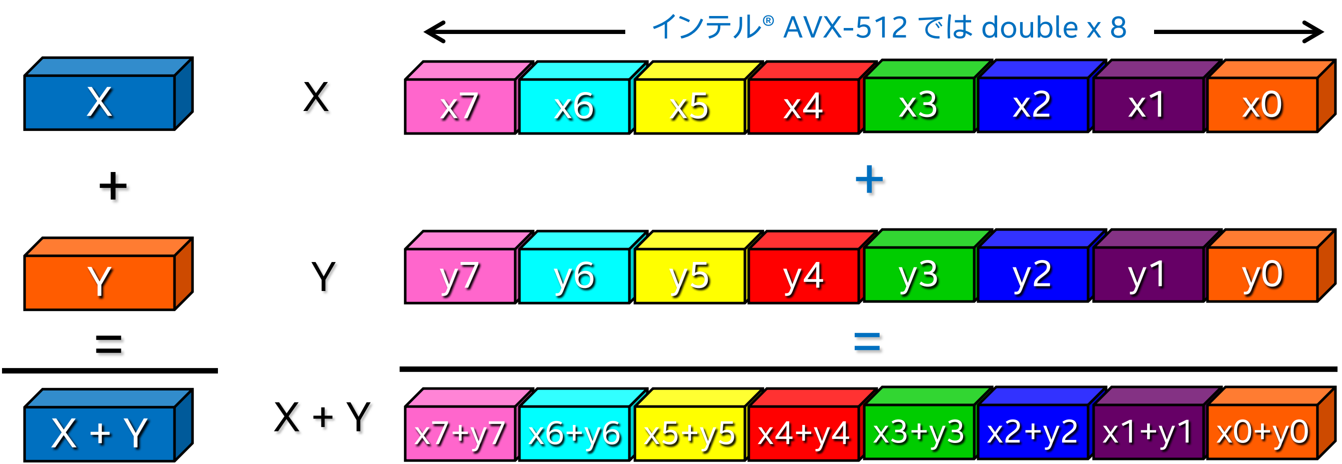
- リンクすると遅い libm の関数が使用される
- インテルのドライバーは libm を明示的にリンクしない
- gcc は -lm を使用するため古い makefile によく含まれている

© 2018 Intel Corporation. 無断での引用、転載を禁じます。Intel、インテル、Intel ロゴは、アメリカ合衆国および / またはその他の国における Intel Corporation の商標です。
* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。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。

SIMD: (Single Instruction Multiple Data)

```
for (i=0; i<n; i++) z[i] = x[i] + y[i];
```

- スカラーモード
  - 1 つの命令で 1 つの結果を生成
  - 例: vaddss (vaddsd)
- ベクトル (SIMD) モード
  - 1 つの命令で複数の結果を生成可能
  - 例: vaddps (vaddpd)



© 2018 Intel Corporation. 無断での引用、転載を禁じます。Intel、インテル、Intel ロゴは、アメリカ合衆国および / またはその他の国における Intel Corporation の商標です。
* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。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。

ベクトル化

ベクトル化は上記のパックド SIMD 命令を使用したループの生成

さまざまな選択肢:

- 自動ベクトル化: コンパイラーがすべて処理、-O2 以上を指定
  - コンパイラーが正当性を保証する必要がある
- アシスト付きベクトル化: ユーザーがプラグマの使用、言語構造やソースコードの変更により追加のヒントを提供する
- 明示的なベクトル化: ユーザーが OpenMP* プラグマを使用してベクトル化を指示する
  - ユーザーが正当性を保証する必要がある
  - コンパイラーが自動ベクトル化できない場合やループのベクトル化を制御する場合に使用

© 2018 Intel Corporation. 無断での引用、転載を禁じます。Intel、インテル、Intel ロゴは、アメリカ合衆国および / またはその他の国における Intel Corporation の商標です。
* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。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。

サポートされるプロセッサー固有のコンパイラー・オプション

インテル® プロセッサーのみ	他社製プロセッサーを含む (-m は gcc でも指定可能)
-xsse2	-msse2 (デフォルト)
-xsse3	-msse3
-xssse3	-mssse3
-xsse4.1	-msse4.1
-xsse4.2	-msse4.2
-xavx	-mavx
-xcore-avx2	
-xcore-avx512	
-xHost	-xHost (-march=native)
インテルの CPUID をチェック	CPUID をチェックしない
サポートしていないプロセッサーで実行した場合はメッセージを表示	サポートしていないプロセッサーで実行した場合は不正な命令エラーになる

© 2018 Intel Corporation. 無断での引用、転載を禁じます。Intel、インテル、Intel ロゴは、アメリカ合衆国および / またはその他の国における Intel Corporation の商標です。
* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。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。

依存性 – ベクトル化を妨げる要因

ベクトル化は SIMD レーンが別の (前の) レーンの結果に依存しない場合にのみ安全

```
for (int i = 0; i < N; i++)
   a[i + 2] = a[i] + c;
```

この例では、i=2,a[4] の結果は i=0,a[2] の結果に依存している
これは「ループ伝播」、「ベクトル」または「フロー」の依存性
a[4] と a[2] の両方が同じ SIMD 命令で使用された場合、
a[4] を計算するときに a[2] の結果がまだ利用できないため、
a[2] のオリジナルの値が使用され、a[4] の結果は不正な値になる

コンパイラーはこのループを自動ベクトル化しないため、手動でベクトル化する必要がある

© 2018 Intel Corporation. 無断での引用、転載を禁じます。Intel、インテル、Intel ロゴは、アメリカ合衆国および / またはその他の国における Intel Corporation の商標です。
* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。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。

コンパイラーは保守的であること

すべての潜在的な依存性が実際に影響するとは限らない

```
void scale(int *a, int *b)
{
    for (int i = 0; i < 10000; i++) b[i] = a[i] + 4;
}
```

- コンパイラーはポインター a と b が「エイリアス」されると仮定する
- データ依存性にはループのマルチバージョンが役立つ
- コンパイラーがベクトル化できるように支援する
  - `-fargument-noalias` を指定してコンパイルする
  - restrict キーワードを使用する: `void scale(int *a, int *restrict b)`
  - for ループの前に `#pragma ivdep` を挿入

© 2018 Intel Corporation. 無断での引用、転載を禁じます。Intel、インテル、Intel ロゴは、アメリカ合衆国および / またはその他の国における Intel Corporation の商標です。
* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。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。

ベクトル化可能なコードを記述するためのガイドライン

- **単純な for ループを使用する** (トリップカウントがループの入り口で判明するようにする)
- **前のループ反復に依存しないようにする**
- **分かりやすいコードを記述する** (次の表記はできるだけ避ける)
  - 多くの関数呼び出し (インライン展開された/単純な数学関数を除く)
  - マスク付きの代入として処理できない分岐
- **ポインターの代わりに配列を使用する**
  - ヒントがないと、コンパイラーはポインターを含むコードを安全にベクトル化できるかどうか判断できない
  - カウンターをインクリメントして配列アドレスに使用する代わりに、ループ・インデックスを配列インデックスで直接使用する
- **効率的なメモリーアクセスを使用する**
  - 内部ループとユニットストライドを使用する (連続メモリーアクセス)
  - 間接アドレス指定は最小限に抑える (b[i] = a[index[i]]; など)
  - できるだけ一貫した方法でデータをアライメントする (インテル® AVX-512 では 64 バイト境界)

© 2018 Intel Corporation. 無断での引用、転載を禁じます。Intel、インテル、Intel ロゴは、アメリカ合衆国および / またはその他の国における Intel Corporation の商標です。
* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。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。

ループが自動ベクトル化されない理由

- ループに入る前にトリップカウント (反復回数) が判明していない
- 前の反復に (明示的またはポインター・エイリアシングによる潜在的な) 依存性がある
  - ループ内部の関数呼び出し
- コンパイラーがパフォーマンス・ゲインがないと判断した
  - 非線形または非連続のメモリーアクセス
  - トリップカウント (反復回数) が SIMD レーンの数と比較して少ない
- 制御フローが複雑
  - 大量の switch 文、goto 文、例外処理、入れ子の if
- データ型が複雑 (ユーザー定義の演算子を含む)
- 入れ子のループの外側のループ

© 2018 Intel Corporation. 無断での引用、転載を禁じます。Intel、インテル、Intel ロゴは、アメリカ合衆国および / またはその他の国における Intel Corporation の商標です。
* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。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。

最適化レポート – 例

$ icc -c -xcommon-avx512 **-qopt-report=3** -qopt-report-phase=loop,vec foo.c

コンパイラーが実行または試行した最適化を要約した foo.optrpt を作成
詳細レベルは 0 (レポートなし) から 5 (最高)
-qopt-report-phase=loop,vec はベクトル化とループの最適化レポートのみ作成
レポートの内容:

ループの開始: foo.c(4,3)
マルチバージョン v1
  リマーク #25228: データの依存関係のループをマルチバージョンにしました。
  リマーク #15300: ループがベクトル化されました。
  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 #15451: マスクなし非アライン・ユニット・ストライド・ストア: 1
  .... (ループのコストサマリー) ....
ループの終了

ループの開始: foo.c(4,3)
<マルチバージョン v2>
  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

```
#include <math.h>
void foo (float * theta, float * sth)  {
  int i;
  for (i = 0; i < 512; i++)
       sth[i] = sin(theta[i]+3.1415927);
}
```

© 2018 Intel Corporation. 無断での引用、転載を禁じます。Intel、インテル、Intel ロゴは、アメリカ合衆国および / またはその他の国における Intel Corporation の商標です。
* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。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。

intel Software

最適化レポート – 例

```
$ icc -c -xcommon-avx512  -qopt-report=4 -qopt-report-phase=loop,vec -qopt-report-file=stderr -fargument-noalias foo.c
…
ループの開始: foo.c(4,3)
…
  リマーク #15417: ベクトル化のサポート: 浮動小数点数をアップコンバートします (単精度から倍精度 1)。[ foo.c(5,17) ]
  リマーク #15418: ベクトル化のサポート: 浮動小数点数をダウンコンバートします (倍精度から単精度 1)。[ foo.c(5,8) ]
  リマーク #15300: ループがベクトル化されました。
  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 #15451: マスクなし非アライン・ユニット・ストライド・ストア: 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: 111
  リマーク #15477: ベクトルのコスト: 10.310
  リマーク #15478: スピードアップの期待値: 10.740
  リマーク #15482: ベクトル化された算術ライブラリーの呼び出し: 1
  リマーク #15487: 型変換: 2
  リマーク #15488: --- ベクトルのコストサマリー終了 ---
  リマーク #25015: ループの最大トリップカウントの予測=32
ループの終了
```

foo.optrpt の代わりに stderr に出力

```
#include <math.h>
void foo (float * theta, float * sth)  {
  int i;
  for (i = 0; i < 512; i++)
      sth[i] = sin(theta[i]+3.1415927);
}
```

© 2018 Intel Corporation. 無断での引用、転載を禁じます。Intel、インテル、Intel ロゴは、アメリカ合衆国および / またはその他の国における Intel Corporation の商標です。
* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。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。

最適化レポート – 例

```
$ icc -S -xcommon-avx512 -qopt-report=4 -qopt-report-phase=loop,vec -qopt-report-file=stderr -fargument-noalias foo.c

ループの開始 foo2.c(4,3)
...
リマーク #15305: ベクトル化のサポート: ベクトル長 32
リマーク #15300: ループがベクトル化されました。
  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 #15451: マスクなし非アライン・ユニット・ストライド・ストア: 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: 109
  リマーク #15477: ベクトルのコスト: 5.250
  リマーク #15478: スピードアップの期待値: 20.700
  リマーク #15482: ベクトル化された算術ライブラリーの呼び出し: 1
  リマーク #15488: --- ベクトルのコストサマリー終了 ---
  リマーク #25015: ループの最大トリップカウントの予測=32
ループの終了

$ grep sin foo.s
        call      __svml_sinf16_b3
        call      __svml_sinf16_b3
```

```
#include <math.h>
void foo (float * theta, float * sth)  {
  int i;
  for (i = 0; i < 512; i++)
     sth[i] = sinf(theta[i]+3.1415927f);
}
```

© 2018 Intel Corporation. 無断での引用、転載を禁じます。Intel、インテル、Intel ロゴは、アメリカ合衆国および / またはその他の国における Intel Corporation の商標です。
* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。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。

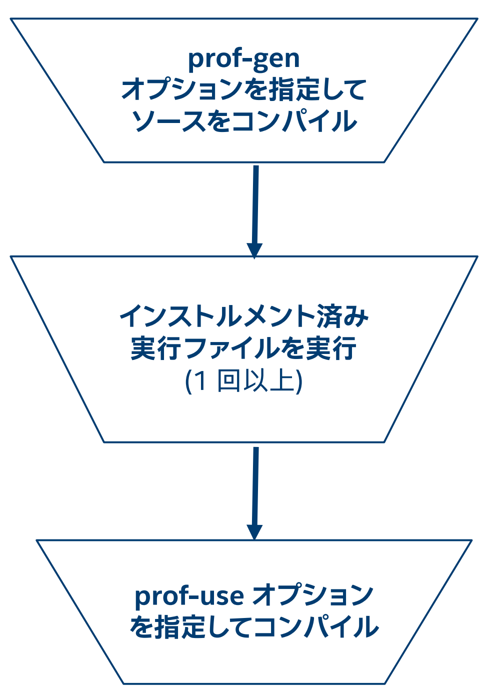
プロファイルに基づく最適化 (PGO)

次のような問題をスタティックに解析して最適化の可能性を探る

- x > y になる頻度は?
- カウントのサイズは?
- どのコードがどの程度の頻度で実行されるか?

```
if (x > y)
      do_this();
 else
      do that();
```

```
for(i=0; i<count; ++i)
do_work();
```



実行時間フィードバックを使用して (最終的な) 最適化をガイド
PGO による拡張:

- より正確な分岐予測
- 基本ブロックの移動による命令キャッシュ効率の向上
- 効率良い関数のインライン展開 (IPO で役立つ)
- 関数の順序の最適化
- switch 文の最適化
- 効率良いベクトル化

© 2018 Intel Corporation. 無断での引用、転載を禁じます。Intel、インテル、Intel ロゴは、アメリカ合衆国および / またはその他の国における Intel Corporation の商標です。
* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。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。

PGO の使用: 3 ステップのプロセス

ステップ 1

インストルメンテーションを追加してコンパイル/リンク
`icc -prof-gen prog.c -o prog`

→ インストルメント済み実行ファイル:
`prog`

ステップ 2

インストルメント済みプログラムを実行
`./prog` (特定のデータセット)

→ 動的プロファイル:
`12345678.dyn`

ステップ 3

フィードバックを使用してコンパイル/リンク
`icc -prof-use prog.c -o prog`

→ マージ後の .dyn ファイル:
`pgopti.dpi`

→ 最適化された実行ファイル:
`prog`

© 2018 Intel Corporation. 無断での引用、転載を禁じます。Intel、インテル、Intel ロゴは、アメリカ合衆国および / またはその他の国における Intel Corporation の商標です。
* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。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。

自動並列化

OpenMP* ランタイムベース

次のオプションを指定すると、コンパイラーはループを等価なマルチスレッド･コードに自動的に変換する

```
-parallel
```

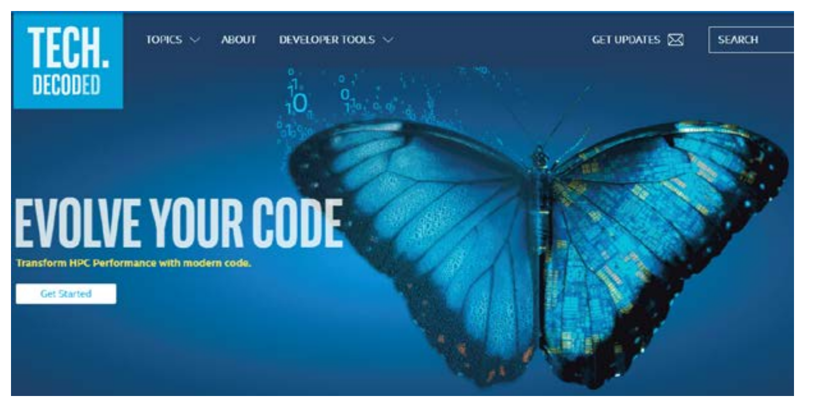
自動パラレライザーは並列で安全に実行できる単純な構造のループを検出して、それらのループのマルチスレッド･コードを自動的に生成する

次のオプションを指定すると、自動パラレライザーのレポートにコンパイラーが並列化したプログラムのセクションに関する情報が含まれる

```
-qopt-report-phase=par
```

© 2018 Intel Corporation. 無断での引用、転載を禁じます。Intel、インテル、Intel ロゴは、アメリカ合衆国および / またはその他の国における Intel Corporation の商標です。
* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。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。

Tech.Decoded の情報を確認



開発者向けの情報を提供

最新のハードウェアを最大限に活用し、競争力を高め、市場投入期間を短縮できるように作成された継続的に増えているナレッジ・ライブラリーにアクセス可能。

全体的な状況を把握できるビデオ—テクノロジーの将来を予測する人々による、さまざまなトピックに関する主な概念を紹介。

要点の深い掘り下げ—アプリケーションとソリューションのパフォーマンスの最適化に役立つ手法、演習、ツールを紹介するオンデマンドのウェビナー。

概念のコード化に役立つクイックヒント—特定の開発ツールを使用した特定のプログラミング・タスクの方法を説明する短いビデオや記事。

https://techdecoded.intel.io (英語)

DISRUPT THE FLOW

© 2018 Intel Corporation. 無断での引用、転載を禁じます。Intel、インテル、Intel ロゴは、アメリカ合衆国および / またはその他の国における Intel Corporation の商標です。
* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。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。

法務上の注意書きと最適化に関する注意事項

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、www.intel.com/benchmarks (英語) を参照してください。

本資料に掲載されている情報は、現状のまま提供されます。本資料は、明示されているか否かにかかわらず、また禁反言によるとよらずにかかわらず、いかなる知的財産権のライセンスも許諾するものではありません。製品に付属の売買契約書『Intel's Terms and Conditions of Sale』に規定されている場合を除き、インテルはいかなる責任を負うものではなく、またインテル製品の販売や使用に関する明示または黙示の保証 (特定目的への適合性、商品性に関する保証、第三者の特許権、著作権、その他、知的財産権の侵害への保証を含む) をするものではありません。

© 2018 Intel Corporation. 無断での引用、転載を禁じます。Intel、インテル、Intel ロゴは、アメリカ合衆国および / またはその他の国における Intel Corporation の商標です。

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

TECH.
DECODED

intel
Software