インテル® コンパイラー・オプション: インテル® Xeon® プロセッサー

（インテル® 64 を含む）

ソフトウェア＆ソリューションズ統括部

ソフトウェア製品部

Rev 12/13/2006

コースの目的

最新のインテル® アーキテクチャーに対応したソフトウェアの最適化を行う

主要なコンパイラーの最適化オプションを使用する

コースの内容

概要

オプションを使用した最適化

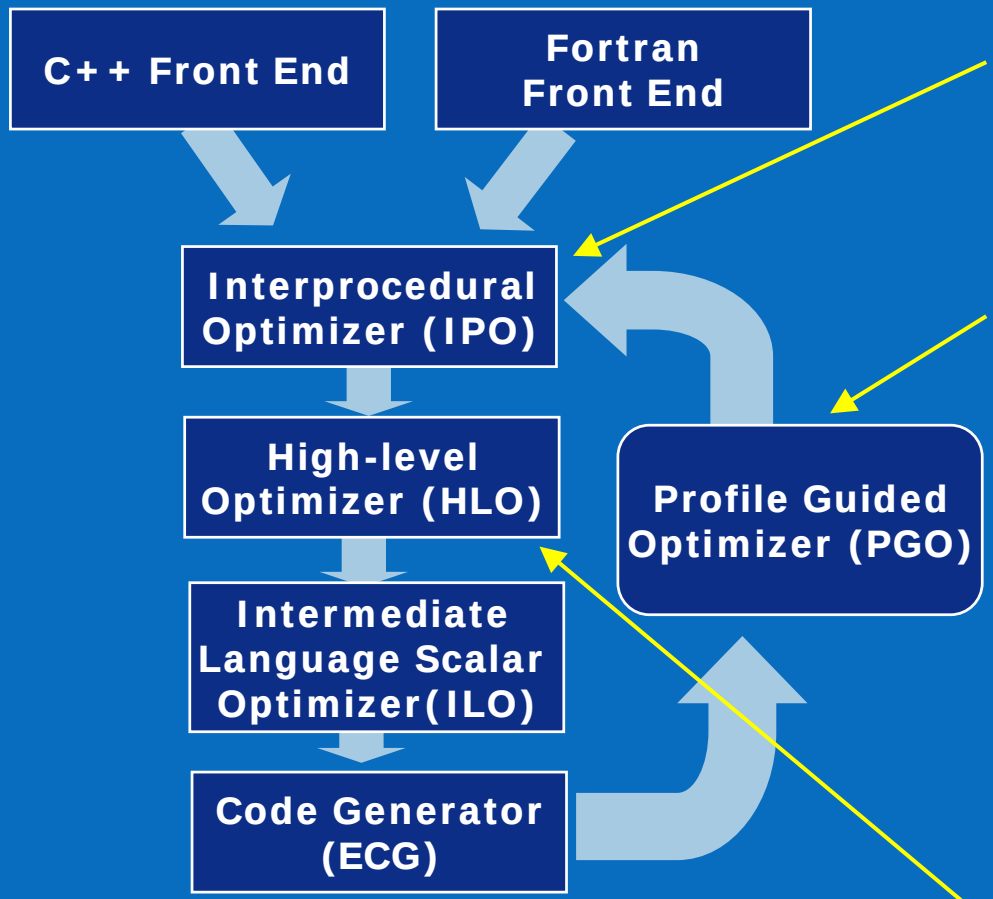
- 基本的な最適化
- 高レベルの最適化
- SIMD – SSE、SSE2、SSE3 サポート
- プロセッサー固有の最適化
- 高度な最適化
- マルチパスの最適化

その他のオプション

数値演算ライブラリー

演習

概要:コンパイラーの最適化と構造



複数ファイル間のインライン、
関数間の定数伝播、
コード置き換え、データ置き換え

スペキュレーション、プレディケーション、
関数の分割、ループ最適化、
ソフトウェア・パイプライン、
基本ブロックの順番、インライン、
レジスター・アロケーション

ループアンロールとロードの最適化、
データプリフェッチ、
レジスター・ブロッキング、
ループ変換、ベクトル演算など
メモリー階層の最適化

/Qopt-report /Qopt-report-phase[Option]
で各フェーズのレポートを表示可能

このセッションで使用する例題と最適化の進め方について

POVRAY*
レイトレーシングによる
3次元レンダリングのための
フリーソフトウェア
http://www.povray.org
で最新版を入手可能

POVRAY をインテル® コンパイラーの代表的な幾つかのコンパイル・オプションを使用しコンパイルし、描画時間を計測

演習 1:
POV-Ray*

初期コンパイル

コマンド・ウィンドウからインテル® C++ コンパイラーを使用して POV-Ray をコンパイルする

受講者用ワークブック「インテル® コンパイラー・スイッチ: インテル® EM64T 対応システム (Windows*) 」を参照。

```
> e:
> cd ¥ClassFiles¥Compiler¥Class3_Ray¥source¥RayTrace2
> lab1.bat
```

もしくは

```
> nmake /f raytrace2.mak clean
> nmake /f raytrace2.mak CPP="icl"  もしくは CPP="cl"
> raytrace2 320 240
```

コースの内容

概要

オプションを使用した最適化

- 基本的な最適化
- 高レベルの最適化
- SIMD – SSE、SSE2、SSE3 サポート
- プロセッサー固有の最適化
- 高度な最適化
- マルチパスの最適化

その他のオプション

数値演算ライブラリー

演習

基本的な最適化オプション

Linux*	Windows*	
-O0	/Od	最適化を無効にする
-g	/Zi	シンボルを作成する
-O1	/O1	コードサイズを増やさずにコードの実行速度を最適化する（ライブラリー関数のインライン化を 無効にする）
-O2	/O2	コードの実行速度を最適化する
-O3	/O3	高レベルな最適化を行う

基本的な最適化オプション
Windows 環境

/Ob[0-2]	インラインの制御
/Og	グローバル最適化
/Oi[-]	組み込み関数のインライン化
/Op[-]	浮動小数点演算の精度向上
/Os	コードサイズを増加しないスピードの最適化
/Ot	スピードの最適化
/Oa[-]	aliasの問題が無いと仮定する
/Ow[-]	関数内には Alias の問題が無いが、関数間ではあると仮定する
/Oy	EBP レジスターを汎用レジスターとして使用する

オプションを使用した最適化

高レベルの最適化 (/O3, -O3)

概要:

/O2 (/Ob2gyti) に加え以下の最適化

- ループレベルの最適化
  （ループのアンロール、キャッシュ・ブロックなどを含む）
- より積極的な依存性解析
- スカラー置換

使用方法:

(Linux*) -O3

(Windows*) /O3

ループは特定の基準を満たしている必要がある

演習 2: POV-Ray* 高レベルの最適化(HLO)

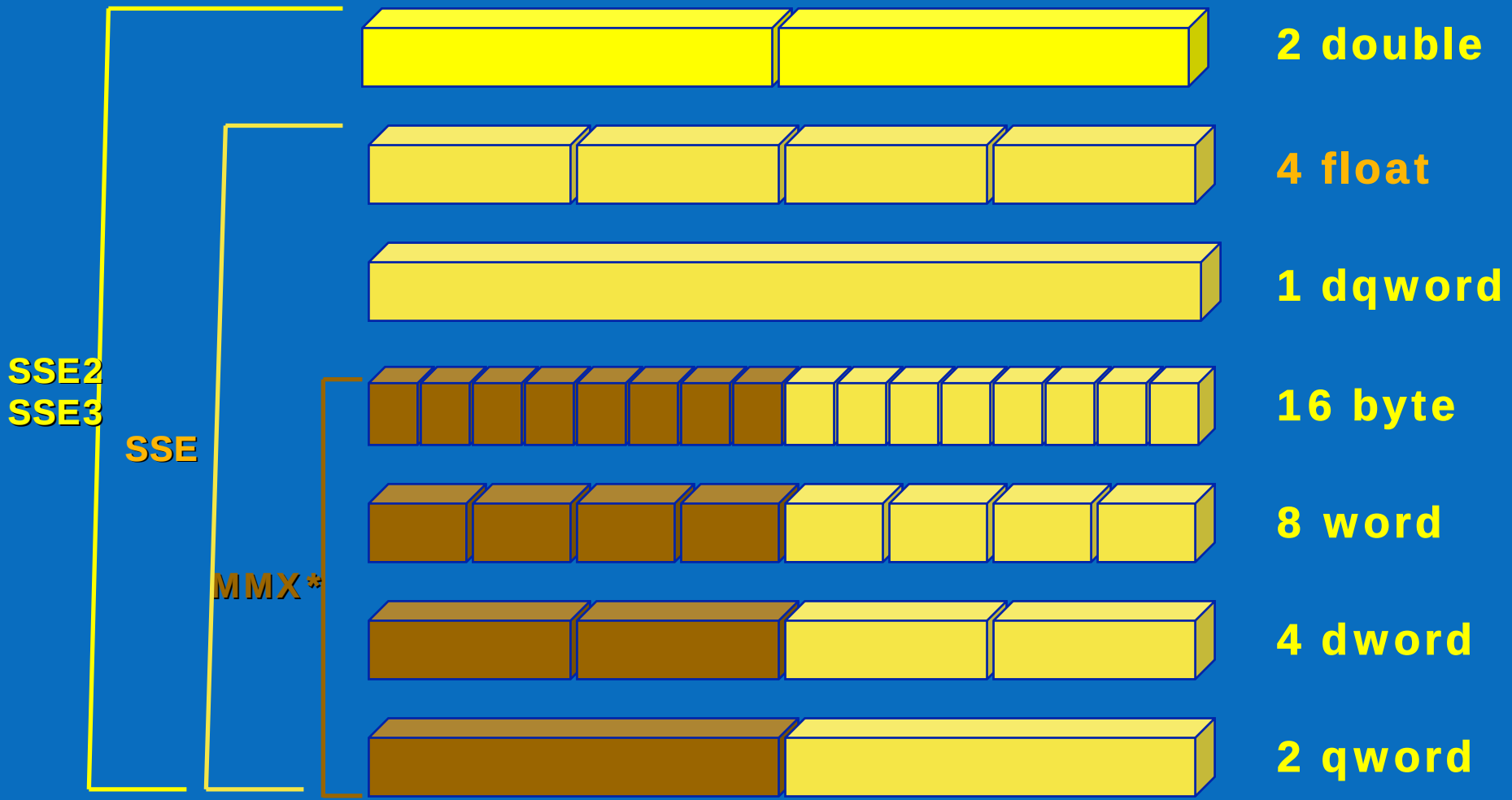
コマンド・ウィンドウからインテル® C++ コンパイラーで高レベルの最適化オプションを使用して POV-Ray をコンパイルする

受講者用ワークブック「インテル® コンパイラー・スイッチ： インテル® EM64T 対応システム (Windows*) 」を参照。

```
> cd
¥ClassFiles¥Compiler¥Class3_Ray¥source¥RayTrace2
> lab2.bat
  もしくは
> nmake /f raytrace2.mak clean
> nmake /f raytrace2.mak CF="/O3"
> raytrace2 320 240
```

オプションを使用した最適化

SIMD – SSE、SSE2、SSE3 サポート



* MMX® は x87 浮動小数点レジスタを使用する。SSE、SSE2、および SSE3 は新しい SSE レジスタを使用する

インテル® ソフトウェア開発製品

オプションを使用した最適化

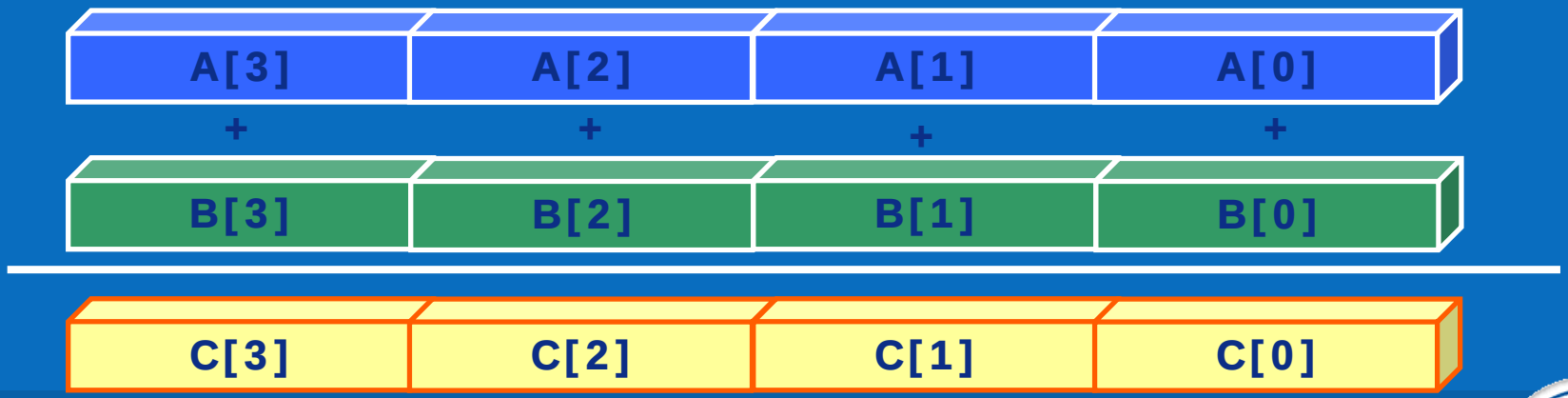
SSE レジスターの使用方法

/QxP (-xP) オプションは、ループのベクトル化、浮動小数点操作、およびスカラー操作で使用される

例:

```
for (i = 0;i <= MAX;i++)
    c[i] = a[i]+b[i];
```

使用方法: **/QxP (-xP)**



インテル® ソフトウェア開発製品

オプションを使用した最適化
プロセッサー固有の最適化

- **SSE 対応 /QxK (-xK)**
  - インテル® Pentium® III プロセッサー
- **SSE2 対応 /QxW (-xW)、/QxN (-xN)**
  - インテル® Pentium® 4 プロセッサーおよびインテル® Xeon® プロセッサー
- **SSE2 対応 /QxB (-xB)**
  - インテル® Pentium® M プロセッサー
- **SSE3 対応 /QxP (-xP)**
  - インテル® EM64T対応のインテル® Pentium® 4 プロセッサー
  - インテル® Xeon® プロセッサー
  - Pentium® D プロセッサー、Pentium® Extreme Edition プロセッサー
  - インテル® Core™ Duo プロセッサー
- **SSE3拡張 対応 /QxT (-xT)**
  - インテル® Xeon® プロセッサー 5100番以降
  - インテル® Core™ 2 Duo プロセッサー
  - インテル® Core™ 2 Extreme プロセッサー

オプションを使用した最適化

プロセッサー固有の最適化

IA32 および インテル® 64 プロセッサーで有効なプロセッサー固有オプション

説明	使用方法	Linux*	Windows*	Mac OS*
MMX®、SSE、SSE2 を含むインテル® Pentium® 4 プロセッサーおよび互換性のあるプロセッサー用の命令を生成して、最適化を行う	W	-xW	/QxW	なし
インテル® エクステンデッド・メモリー 64 テクノロジー対応のインテル® プロセッサーおよびデュアルコア・プロセッサー用の命令を生成して、最適化を行う。これらのプロセッサは、MMX、SSE、SSE2、SSE3 をサポートする	P	-xP, -axP	/QxP /QaxP	/QxP /QaxP （デフォルト）
インテル® Core™ アーキテクチャー対応のインテル® プロセッサー用の命令を生成して、最適化を行う。これらのプロセッサは、MMX、SSE、SSE2、SSE3、SSE3拡張 をサポートする	T	-xT, -axT	/QxT /QaxT	/QxT /QaxT

オプションを使用した最適化

プロセッサー固有の最適化

IA32 でのみ有効で、インテル® 64 プロセッサーでサポートされないオプション

説明	使用方法	Linux*	Windows*	Mac OS*
MMX®、SSE、SSE2 を含むインテル® Pentium® 4 プロセッサーおよび互換性のあるプロセッサ用の命令を生成して、最適化を行う	N	-xN, -axN	/QxN	なし
MMX、SSE を含むインテル® Pentium® III プロセッサー用の命令を生成して、最適化を行う	K	-xK, -axK	/QxK /QaxK	なし
MMX、SSE、SSE2 を含むインテル® Pentium® M プロセッサー用の命令を生成して、最適化を行う	B	-xB, -axB	/QxB /QaxB	なし

オプションを使用した最適化

自動プロセッサーディスパッチ – ax[?]

単一の実行ファイル

- ターゲット・プロセッサー用に最適化され、すべてのインテル® プロセッサー上で実行できる汎用コードを生成する

各ターゲット・プロセッサーに対する動作

- プロセッサー固有の命令を使用する
- ベクトル化を行う

オーバーヘッドが少ない

- コードサイズは多少増える

オプションを使用した最適化
手動プロセッサーディスパッチ

```
// 宣言
__declspec(cpu_dispatch(generic, pentium_4_sse3, ...))
void array_sum(int num) {
   /* ここには、コンパイラーによってコードが追加されるので、コードを追加しない */
}

// すべての定義
      __declspec(cpu_specific(generic))
      void array_sum(int num) {
         /* スカラ x87 浮動小数点用のコードをここに追加する */
      }

      __declspec(cpu_specific(pentium_4_sse3))
      void array_sum(int num) {
         /* SIMD-FP 用のコードと組み込み関数をここに追加するか、
            またはベクトル・クラス・ライブラリーを追加する */
      }...
```

最適化された複数の関数を含む 1 つのバイナリー

コースの内容

概要

オプションを使用した最適化

- プロセッサー固有の最適化
- 高度な最適化
  - 高度なイディオム認識 – ベクトル化
  - OpenMP*
  - 自動並列処理
- マルチパスの最適化

その他のオプション

数値演算ライブラリー

演習

高度な最適化

高度な語法認識 – ベクトル化

インテル® プロセッサーは、さまざまな語法の SIMD 命令を提供する

- 型変換および飽和を伴う型変換
- 飽和算術演算
- クリッピング
- AVG、ABS 演算

/Qx[W/N/B/P/T] オプションを使用すると、インテル® コンパイラーは、ベクトル化中にこれらの用法を検出する

- パターン・マッチングによる検出はしない

高度な最適化

高度な語法認識 – 飽和とクリッピング命令

```
unsigned char a[256], b[256];
..
for (i = 0; i < 256; i++) {  int x = (a[i] < 200) ? a[i]+55 : 255;    if (x > b[i]) b[i] = x;}

..B1.9:                                                    # Preds ..B1.7 ..B1.9
        movslq    %eax, %rdx                               #4.39
        movdqa    (%rdx, %rdi), %xmm1                      #4.39
        paddusb   %xmm0, %xmm1                             #4.46
        pmaxub    (%rdx,%rsi), %xmm1                       #4.46
        movdqa    %xmm1, (%rdx,%rsi)                       #5.52
        addl      $16, %eax                                #4.1
        cmpl      %r10d, %eax                              #4.1
        jb        ..B1.9                                   #Prob 99% #4.1
        jmp       ..B1.14                                  #Prob 100%          #4.1
```

インテル® コンパイラー が自動処理

高度な最適化

高度な語法認識 – 飽和とクリッピング命令

```
unsigned char a[256], b[256];
..
for (i = 0; i < 256; i++) {  int x = (a[i] < 200) ? a[i]+55 : 255;   if (x > b[i]) b[i] = x;}
```

```
$B1$9:                                                  ; Preds $B1$7 $B1$9
        movsxd    rax, r9d                              ;4.39
        movdqa    xmm1, XMMWORD PTR [rax+rcx]           ;4.39
        paddusb   xmm1, xmm0                            ;4.46
        pmaxub    xmm1, XMMWORD PTR [rax+rdx]           ;4.46
        movdqa    XMMWORD PTR [rax+rdx], xmm1           ;5.52
        add       r9d, 16                               ;4.1
        cmp       r9d, r8d                              ;4.1
        jb        $B1$9                                 ; Prob 99% ;4.1
        jmp       $B1$14                                ; Prob 100% ;4.1
```

インテル® コンパイラーが自動処理

高度な最適化
OpenMP*

宣言子を使用して簡単に行えるマルチスレッド化

インテル® ソフトウェア開発ツールでは OpenMP をサポート

- インテル® Fortran および C++ コンパイラーによる OpenMP* 2.0（V8.1）、OpenMP* 2.5+（V9.0以降）のサポート
- インテル® VTune™ パフォーマンス・アナライザー
- インテル® スレッド・チェッカー/プロファイラー

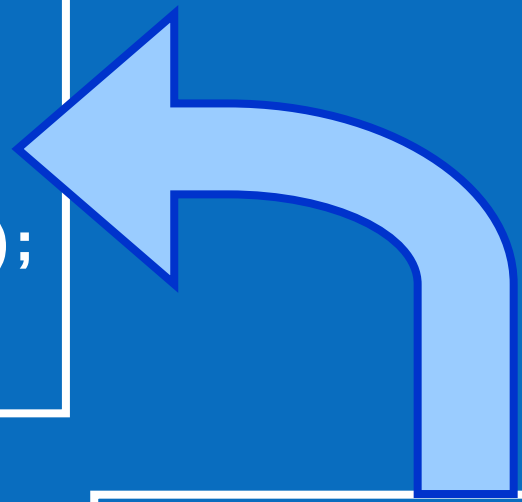
OpenMP* スイッチ：

Linux*	Windows*
-openmp	/Qopenmp
-openmp_report[n]	/Qopenmp_report[n]

ループは特定の基準を満たしている必要がある

宣言子の例

```
foo(float *a, float *b, float *c)
{
  int i;
#pragma parallel for
  for (i=0; i<N; i++) {
     *c++ = (*a++)*bar(b++);
  };
}
```



並列化コードを生成するために簡単な宣言子を使用

OpenMP* 宣言子

```
void foo(int n)
{ int a[1000], b[1000], c[1000], x[1000], i;
    /* 並列化領域 */
#pragma omp parallel for private(i), firstprivate(n) //ループのワークシェアを支持
        for (i = 0; i< n; i++) {
          x[i] = bar(a[i], b[i], c[i], n);
        }
}
```

```
icl /Qopenmp -c foo.c                    { -openmp  Linux}
foo.c
foo.c(4) : (col. 1) remark: OpenMP DEFINED LOOP WAS PARALLELIZED.
```

高度な最適化
自動並列化

自動並列化では、手動で OpenMP* 宣言子を
挿入することなく、ループを自動でスレッド化する

Linux*	Windows*
-parallel	/Qparallel
-par_report[n]	/Qpar_report[n]

OpenMP* 宣言子を使用することを推奨:
- コンパイラーは、並列化できる候補を識別できるが、規模の大きなアプリケーションでは解析が困難

自動並列化の例

```
float a[N], b[N], c[N];
int i;
for (i=0; i<N; i++)
    c[i] = a[i]*b[i];
```

```
icl  /Qparallel foo.c                { -parallel  Linux}
....
foo.c
foo.c(4) : (col. 4) remark: LOOP WAS AUTO-PARALLELIZED.
...
./foo.exe – 実行コードはプロセッサーの数を認識しスレッド数を決定
...
-Qpar_report[n] – 自動並列化に関するメッセージを表示
```

演習 3:
POV-Ray* ベクトル化

コマンド・ウィンドウからインテル® C++ コンパイラーでベクトル化の最適化オプションを使用して POV-Ray をコンパイルする

受講者用ワークブック「インテル® コンパイラー・スイッチ: インテル® EM64T 対応システム (Windows*) 」を参照。

```
> cd
¥ClassFiles¥Compiler¥Class3_Ray¥source¥RayTrace2
> lab3.bat
  もしくは
> nmake /f raytrace2.mak clean
> nmake /f raytrace2.mak CF="/QxP"
> raytrace2 320 240
```

コースの内容

概要

オプションを使用した最適化

- 高度な最適化
- マルチパスの最適化
  - プロシージャー間の最適化（IPO）
  - プロファイルに基づく最適化（PGO）

その他のオプション

数値演算ライブラリー

演習

マルチパスの最適化

プロシージャー間の最適化（IPO）

ip: 単一ファイルのコンパイルで
プロシージャー間の最適化を
有効にする

Linux*	Windows*
-ip	/Qip
-ipo	/Qipo

ipo: 複数ファイルにわたる
プロシージャー間の最適化を有効にする

他のオプションと併用することで最適化を支援する

マルチパスの最適化

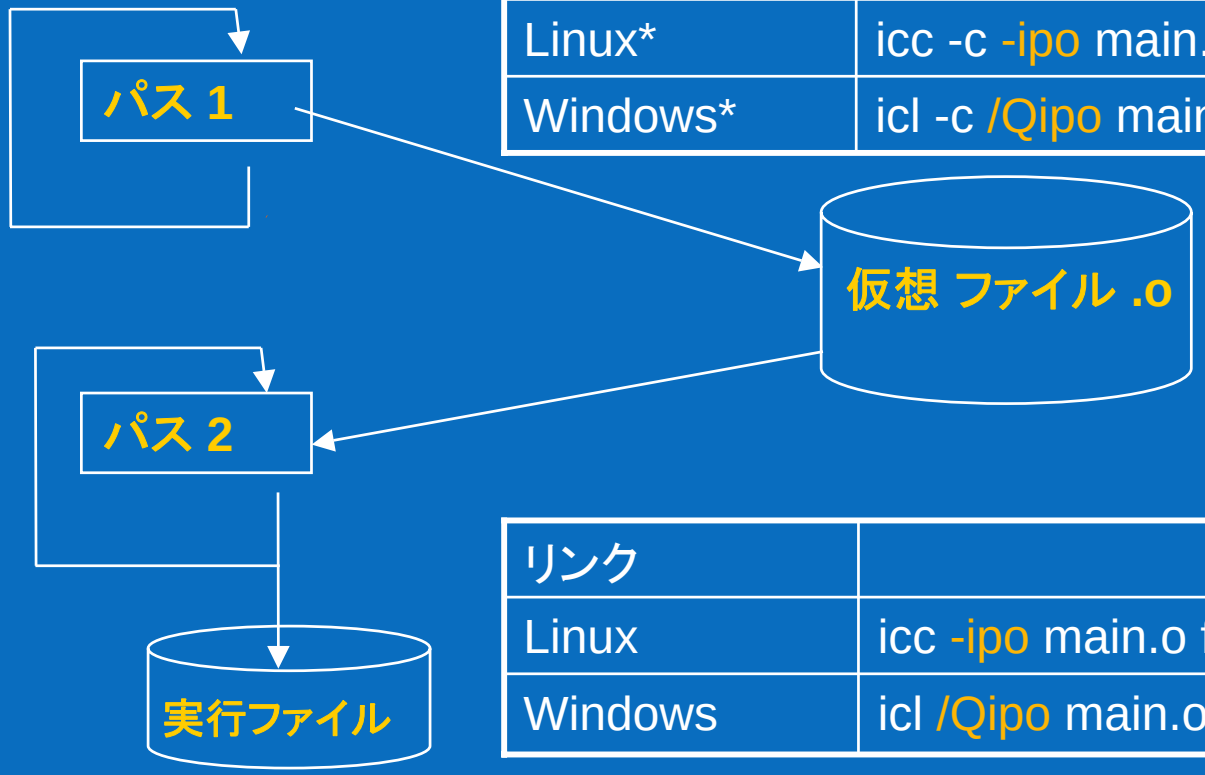
プロシージャー間の最適化（IPO）

インライン展開以外に多くの利点を提供

- 部分的なインライン展開
- プロシージャー間での定数伝播
- レジスターでの引数の受け渡し
- ループ不変コードの移動
- 不要コードの排除
- ベクトル化やメモリーの明確化を支援

マルチパスの最適化 - IPO

2 つのステップによる処理



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

演習 4: POV-Ray* プロシージャー間の最適化（IPO）

コマンド・ウィンドウからインテル® C++ コンパイラーのプロシージャー間の最適化オプションを使用して POV-Ray をコンパイルする

受講者用ワークブック「インテル® コンパイラー・スイッチ： インテル® EM64T 対応システム (Windows*) 」を参照。

```
> cd
¥ClassFiles¥Compiler¥Class3_Ray¥source¥RayTrace2
> lab4.bat
  もしくは
> nmake /f raytrace2.mak clean
> nmake /f raytrace2.mak CF="/Qipo" LF="/Qipo"
> raytrace2 320 240
```

コースの内容

概要

オプションを使用した最適化

- 基本またはプロセッサー固有の最適化
- 高度な最適化
- マルチパスの最適化
  - プロシージャー間の最適化（IPO）
  - プロファイルに基づく最適化（PGO）

その他のオプション

数値演算ライブラリー

演習

マルチパスの最適化

プロファイルに基づく最適化（PGO）

実行時のフィードバックを使用して、その他に利用できるコンパイラーの最適化を検証

命令キャッシュ、ページング、分岐予測を支援

有効な最適化：

- 基本ブロックの並び替え
- 適切なレジスターの割り当て
- 関数のインライン化における的確な判断
- 関数の並び替え
- スイッチ文の最適化
- ベクトル化における的確な判断

マルチパスの最適化

PGO: 3 つのステップによる処理

ステップ 1

インストルメント済みのコンパイル
(Linux*) icc -prof_gen prog.c
(Windows*) icl -Qprof_gen prog.c

→ インストルメント済みの実行ファイル

ステップ 2

インストルメント済みの実行
通常のデータセット上でプログラムを実行

→ DYN ファイルには動的情報が含まれる: .dyn

ステップ 3

フィードバック・コンパイル
(Linux) icc -prof_use prog.c
(Windows) icl -Qprof_use prog.c

→ マージ済み DYN サマリファイル: .dpi
情報を含めない場合は、古い dyn ファイルを削除

マルチパスの最適化

PGOが有効となるプログラム

多数の関数、呼び出し、または分岐を含むプログラム

- 例: データベース、意思決定支援システム、MCAD
- 演算を幅広く多用するプログラム
- ループの反復回数を知ることで、より積極的な最適化を行える場合

考慮事項:

- ビルドの異なるパラダイム: 3 つのステップ
- コードが安定した開発の最終段階で、スケジュールを行う
- 表現可能なデータセットを使用

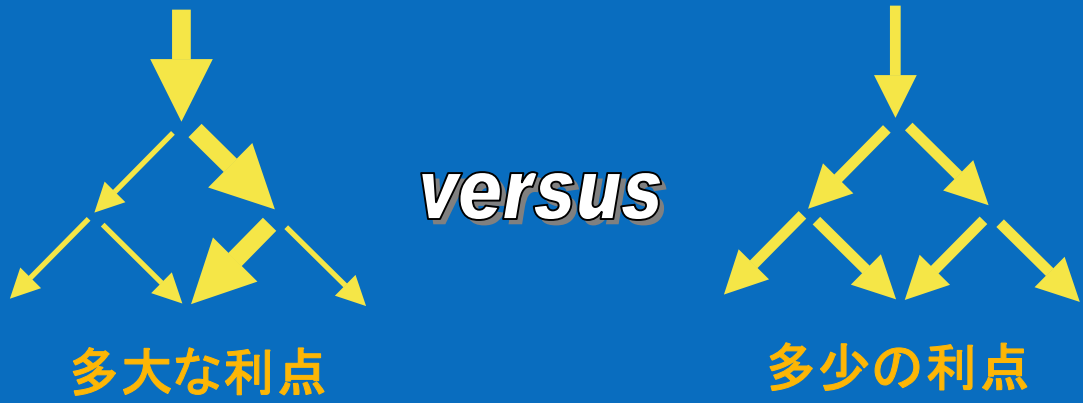
マルチパスの最適化

PGO: 利点を得られるプログラム

一貫性のあるホットパス

多数の if 文または switch文

ネストされた if 文または switch文



マルチパスの最適化

PGO: 利点を得られるプログラム

予測不可能な間接分岐

- 条件付き分岐との比較
- 一般的に switch 文で発生
- 直接分岐より大きい相対レイテンシ

インテル® コンパイラーの処理

- 条件分岐を使用すると思われるケースを最適化

マルチパスの最適化

PGO の例

```
for (i=0; i < NUM_BLOCKS; i++) {
        switch (check3(i))   {
            case 3:                 /* 25% */
                x[i] = 3;  break;
            case 10:                /* 75% */
                x[i] = 10;  break;
            default:                /* 0% */
                x[i] = 99;  break;  }
}
```

PGO は、一般的な case へのジャンプを排除できる

マルチパスの最適化

PGO の例

PGO 前

```
cmpl    $3, %edx
jne     ..B1.6    //jumps 75% of time
movl    $3, -4(%esp,%ebx,4)
jmp     ..B1.9  //goto loop termination

..B1.6:
cmpl    $10, %edx
jne     ..B1.8
movl    $10, -4(%esp,%ebx,4)
jmp     ..B1.9  //goto loop termination

..B1.8:
movl    $99, -4(%esp,%ebx,4)

..B1.9:  //loop termination
cmpl    $100, %ebx
jl      ..B1.4
```

PGO 後

```
cmpl    $3, %edx
je      ..B1.10  //jumps 25% of time
cmpl    $10, %edx
jne     ..B1.9
movl    $10, -4(%esp,%ebx,4)
jmp     ..B1.7  //goto loop termination

..B1.10:
movl    $3, -4(%esp,%ebx,4)
jmp     ..B1.7  //goto loop termination

..B1.9:
movl    $99, -4(%esp,%ebx,4)
jmp     ..B1.7  //goto loop termination

$B1$7: //loop termination
cmpl    $100, %ebx
jl      ..B1.4
```

マルチパスの最適化

ソフトウェア・ベースの推論による事前実行（SSP）

ヘルパースレッドを使用して、必要となるデータをキャッシュに事前読み込みを行うことで、メモリー・アクセスの遅延を隠匿する

IA-32 Windows* と Linux* コンパイラーでサポートされ、HTテクノロジーやマルチコア・プロセッサーが必要

SSP を使用した最適化のステップ:

1. /Qprof_gen オプションを指定しインストルメントを行うバイナリーを作成
2. バイナリーを実行し動的なプロファイル情報を作成
3. /Qprof_use と /Qprof_gen_sampling を使用し、プロセッサーのモニタリング・ユニットから情報を収集するバイナリーを作成
4. profrun –dcache *.exe コマンドを実行し、キャッシュのプロファイル情報を作成
5. /Qprof_use と /Qssp オプションを指定し、SSP最適化が有効なバイナリーを作成する

演習 5: プロファイルに基づく最適化（PGO）
POV-Ray* プロファイルを行うバイナリーを作成

コマンド・ウィンドウからインテル® C++ コンパイラーのプロファイルに基づく最適化オプションを使用して POV-Ray をコンパイルする

受講者用ワークブック「インテル® コンパイラー・スイッチ: インテル® EM64T 対応システム (Windows*) 」を参照。

```
> cd ¥ClassFiles¥Compiler¥Class3_Ray¥source¥RayTrace2
> lab5gen.bat
  もしくは
> nmake /f raytrace2.mak clean
> nmake /f raytrace2.mak CF="/Qprof_gen /Qprof_dir="
E:¥ClassFiles¥Compiler¥Class3_Ray¥source¥raytrace2¥dyn""
> raytrace2 320 240
```

演習 5: プロファイルに基づく最適化（PGO）
POV-Ray* プロファイルを参照して最適化

コマンド・ウィンドウからインテル® C++ コンパイラーのプロファイルに基づく最適化オプションを使用して POV-Ray をコンパイルする

受講者用ワークブック「インテル® コンパイラー・スイッチ: インテル® EM64T 対応システム (Windows*) 」を参照。

```
> cd E:¥ClassFiles¥Compiler¥Class3_Ray¥source¥RayTrace2
> lab5use.bat
  もしくは
> nmake /f raytrace2.mak clean
> nmake /f raytrace2.mak CF="/Qprof_use /Qprof_dir="
E:¥ClassFiles¥Compiler¥Class3_Ray¥source¥raytrace2¥dyn""
> raytrace2 320 240
```

コースの内容

概要

オプションを使用した最適化

- 基本またはプロセッサー固有の最適化
- 高度な最適化
- マルチパスの最適化
  - プロシージャ間の最適化（IPO）
  - プロファイルに基づく最適化（PGO）

その他のオプション

数値演算ライブラリー

演習

その他のオプション

オプション (Linux*)	オプション (Windows*)	説明
-ssp	/Qssp	ソフトウェア・ベースの推論による事前実行を有効にする
	/Qvcn	Microsoft* Visual* C のと互換性を指示する n=(vc6, vc7, vc7.1,vc8)
-gcc-version=n		コンパイラーの動作を gcc と互換にする。 n=320, gcc3.2 互換 n=330, gcc3.3 互換 n=340, gcc3.4 互換 n=400, gcc4.0 互換

その他のオプション

浮動小数点の精度

オプション (Linux*)	オプション (Windows*)	説明
-mp1	/Qprec	ANSI に近い精度で、ANSI より速度が向上する
-prec_div	/Qprec_div	除算を逆数の乗算に変換しない
-pc*n*	/Qpc*n*	浮動小数点の仮数部の精度を制御可能にする n={32,64,80}
-fp_port	/Qfp_port	代入時とキャスト時に浮動小数点の結果を丸める (実行速度は多少遅くなる)
-rcd	/Qrcd	浮動小数点から整数への変換で切り捨てを行う コードを削除する
-fp:model	/Qfp:model	浮動小数点演算モデルを指定する。(except, double, extended, fast, precise, source, strict)

その他のオプション

その他のオプション

『コンパイラー・ユーザーズ・ガイド』と
『リファレンス・ガイド』を参照

(Linux*)　　icc –help, man icc

(Windows*)　　icl /help

www.intel.co.jp/jp/developer/software/products/

その他のオプション

Fortran オプション

ローカル変数		
(Linux*) (Windows*)	-autoscalar /Qautoscalar	スカラー変数のみをスタックに割り当てる（デフォルト）
(Linux) (Windows)	-auto /Qauto	すべての変数をランタイム・スタックに割り当てる（OpenMP* のデフォルト）
(Linux) (Windows)	-save /Qsave	すべての変数を静的メモリーに割り当てる
(Linux) (Windows)	-zero /Qzero	ゼロに初期化する（静的変数の場合）

Fortran95 を使用しない場合	
-FI	ソースファイルが固定形式であることを指定する
-w95	Fortran95 の警告メッセージを抑止する

コースの内容

概要

オプションを使用した最適化

- 基本またはプロセッサー固有の最適化
- 高度な最適化
- マルチパスの最適化
  - プロシージャー間の最適化（IPO）
  - プロファイルに基づく最適化（PGO）

その他のオプション

数値演算ライブラリー

演習

数値演算ライブラリ

(Linux*)　　インテル® libimf

(Windows*)　　インテル® LIBM

SVML (Short Vector Mathematical Library)

- 数値演算関数を含むループのベクトル化で使用

必要なときに自動的に使用される

- LD_LIBRARY_PATH　環境変数

共通の数値演算関数

- sin/cos/tan/exp/sqrt/log ...

それぞれの IA プロセッサー用のプロセッサー・ディスパッチ

プログラミングのヒント

インテル® コンパイラーで生成したオブジェクトと他のコンパイラーで作成したオブジェクトの混在

```
OPTION DOTNAME
ASSUME  CS:FLAT,DS:FLAT,SS:FLAT
INCLUDELIB <libmmt>
INCLUDELIB <LIBCMT>
INCLUDELIB <libirc>
INCLUDELIB <OLDNAMES>
INCLUDELIB <bufferoverflowu>
```

コンパイラーはインテル・ライブラリーのリンクパスをオブジェクト中に挿入

このオブジェクトをインテルのライブラリー・パスが設定されていない他社コンパイラーでリンクするとエラーが発生。*/ ZI* オプションでリンク・パスの生成を抑制。

Linux* 上のライブラリー

-i_dynamic	共有ライブラリーへのリンク（デフォルト）
-static	スタティック・ライブラリーへのリンク
-shared	共有オブジェクトの作成
-Vaxlib	移植ライブラリーへのリンク (Fortran)

演習 6:
POV-Ray*

まとめ

コマンド・ウィンドウからインテル® C++ コンパイラーの複数の最適化オプションを使用して POV-Ray をコンパイルする

受講者用ワークブック「インテル® コンパイラー・スイッチ: インテル® 64 対応システム (Windows*) 」を参照。

```
> cd ¥ClassFiles¥Compiler¥Class3_Ray¥source¥RayTrace2
> lab6.bat
  もしくは
> nmake /f raytrace2.mak clean
> nmake /f raytrace2.mak CF="/O3 /QxP /Qipo /Qprof_use
/Qprof_dir=
"C:¥ClassFiles¥Compiler¥Class3_Ray¥source¥raytrace2¥dyn""
LF="/Qipo"
> raytrace2 320 240
```

まとめ

インテル® コンパイラーの主要な最適化オプション

- 基本的なオプション
- ベクトル化および高レベルの最適化
- プロファイルに基づく最適化（PGO）
- プロシージャー間の最適化（IPO）

インテル® コンパイラーがインテル® アーキテクチャーの利点を活用する方法

インテル® コンパイラーを使用して最適化された POV-Ray*

リファレンス

トレーニングおよびサポートの詳細情報

Web ベースおよびクラスルーム・トレーニング

www.intel.com/software/college/ (英語)

ホワイトペーパーおよびテクニカル・ノート

www.intel.com/ids/ (英語)

www.intel.co.jp/jp/developer/software/products/

製品サポート情報

www.intel.com/software/products/support/ (英語)

補足資料

オプションを使用した最適化

プロセッサー・チューニング

特定のプロセッサーの命令レイテンシーとキャッシュ・ラインのサイズに対して最適になるように命令をスケジュールする

	Linux*	Window*
インテル® Pentium® 4 プロセッサーおよびそれ以降	-tpp7（デフォルト）	/G7（デフォルト）
デュアルコア インテル® Itanium® 2 プロセッサー 9000番台およびそれ以降	-mtune=itanium2-p9000	/G2-p9000

オプションを使用した最適化

プロセッサー・チューニング /G7, -tpp7 の例

インテル® コンパイラーは、さまざまな最適化を使用できる

- 定数との乗算を加算に変更
- 定数の値だけ左シフトを加算に変更
- 符号で拡張した値にシフトを使用しない
- シフトを積極的に減らす

コンパイラーはこれらの違いを自動的に処理する

オプションを使用した最適化

命令スケジューリングの例

```
for (int i=0;i<length;i++) {p[i] = q[i] * 32; }
```

インテル® 64 対応のインテル® Xeon® プロセッサーまたは
インテル® Pentium® 4 プロセッサー

```
..B1.4:         # -tpp7
    movl (%rsi,rcx,4), %r9d
    shll   $5, % r9d
    movl %r9d, (%rdi,%rcx,4)
```

オプションを使用した最適化

命令スケジューリングの例

```
for (int i=0;i<length;i++) {p[i] = q[i] * 32; }
```

インテル® 64 対応のインテル® Xeon® プロセッサーまたは
インテル® Pentium® 4 プロセッサ

```
..B1.4:        # -G7
    mov  r11d, DWORD PTR [rdx+r10*4]
    shl    r11d, 5
    mov  DWORD PTR [rcx+r10*4], r11d
```

本資料に掲載されている情報は、インテル製品の概要説明を目的としたものです。製品に付属の売買契約書『Intel's Terms and conditions of Sales』に規定されている場合を除き、インテルはいかなる責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、知的所有権を侵害していないことへの保証を含む) に関しても一切責任を負わないものとします。

インテル製品は、予告なく仕様が変更されることがあります。

* その他の社名、製品名などは、一般に各社の商標または登録商標です。