intel Software

インテル® Parallel Studio XE における浮動小数点結果の再現性

プレゼンター: Martyn Corden シニア・ソフトウェア・エンジニア
コントリビューター: Barbara Perz & Xiaoping Duan テクニカル・コンサルティング・エンジニア
インテル コーポレーション

内容

再現性について

インテル® Parallel Studio XE に含まれる各ツールにおける再現性

- インテル® コンパイラー
  - 同じ実行ファイル
  - 異なる実行ファイル
- インテル® パフォーマンス・ライブラリー
  - インテル® マス・カーネル・ライブラリー (インテル® MKL)
  - 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)
- インテル® MPI ライブラリー

まとめ

再現性について

浮動小数点演算の有限精度は、浮動小数点演算結果に**固有の不確実性**をもたらす

- 結果はこの不確実性の範囲内で変わる可能性がある
  - 異なる最適化レベル、異なるプロセッサーなど
  - 通常、これは問題にならない

次のような状況ではより高い再現性が求められる

- 品質保証、変更制御、機能安全、法的責任
  - 例: 金融サービス、自動運転、人工知能 (AI)、天気予報、衝突シミュレーション
- 高い再現性はパフォーマンスを低下させる

「再現可能」は必ずしも精度が高いわけではない

差異の原因

最適化が主な原因

- プロセッサー依存の可能性がある (異なる命令セット、SIMD 幅など)
- 並列実行は最適化の 1 つ
- 大規模なアプリケーションを最適化しないことはめったにない

精度の違い

- 近似 (例: 数学関数、高速除算と平方根、非正規化数をゼロにフラッシュ、など)
- FMA (Fused Multiply Add) 命令 (乗算と加算を個別に行うよりも高い精度が得られる)

操作順序の変更

- 特に並列アプリケーションにとって最大の原因
- 丸め誤差の累積方法が異なる
- 異なる結果は必ずしも精度が低いわけではない
  - ユーザーは最適化されていない結果を "正しい" と見なすことが多い

操作順序の変更

例

- (x[i] + y) + z → x[i] + (y + z);
- a*b + a*c → a*(b+c)
- これらの変換は数学的には同じである
  - しかし有限精度演算では異なる
- これらの例はシーケンシャル・アプリケーションと並列アプリケーションに同様の影響を与える
- コンパイルされるコードでは、コンパイラー・オプションでこれらを抑止できる

OpenMP* SIMD プラグマを含む、シングルスレッド・アプリケーションの浮動小数点結果の再現性に対するコンパイラーによる最適化の影響については、https://www.isus.jp/products/c-compilers/consistency-of-floating-point-results/ を参照

操作順序の変更 - リダクション

例えば、合計の場合 – 積、最大、最小なども使用可能

- 並列実装は部分的な結果を生成し、それらを結合する
  - 例: SIMD レーン、OpenMP* スレッド、または MPI ランクごとに 1 つの結果
- 結果はシーケンシャル評価とは異なる可能性がある
- しかし優れたパフォーマンスを提供する

```
float Sum(const float A[], int n)
{
    float sum=0;
    for (int i=0; i<n; i++)
        sum = sum + A[i];
    return sum;
}
```

```
float Sum( const float A[], int n, int nt ) {
  float sum=0.;         // 合計
  float part_sum[nt]; // スレッドごとに 1 つの部分和
  for (int it=0; it<nt; it++) part_sum[it]=0.;

  for (int it=0; it<nt; it++)  {
      for (int j=it*n/nt; j<(it+1)*n/nt; j++)
          part_sum[it] += A[j];
  }                          // 部分和をインクリメント
  for (int it=0; it<nt; it++) sum += part_sum[it];
                             // 部分和を結合
return sum;  }
```

並列リダクションを再現可能にする方法 (1)

ドメイン分割 (部分和の構成) は変更しない

- SIMD: ベクトル長は変更しない。データ・アライメントに依存しないようにする (-qno-opt-dynamic-align)
- OpenMP*: スレッド数は変更しない
- インテル® TBB: parallel_deterministic_reduce() を使用する
  - タスクへの分割は固定、タスクのスレッドへの割り当ては変更可
- インテル® MPI ライブラリー: ランク数は変更しない

並列リダクションを再現可能にする方法 (2)

部分和の結合順序は変更しない

- SIMD: 自動
- OpenMP* スレッド: 標準では任意の順序
  - スレッド数が少ない場合のデフォルトは先着順
  - 再現性には KMP_DETERMINISTIC_REDUCTION=yes と静的スケジューリングを使用
  - スレッド数が多い場合のデフォルト
- インテル® TBB: parallel_deterministic_reduce() を使用
  - 部分和は事前定義した順序で結合される
- インテル® MPI ライブラリー: トポロジーを意識しないリダクション・アルゴリズムを選択する (後述)

インテル® コンパイラーにおける再現性

同じ実行ファイル、異なる実行ファイル

同じ実行ファイル: 実行ごとの変動 (シングルスレッド)

同じプロセッサー上で同じ入力を使用して同じ実行ファイルを再実行した場合、同じ結果になるのか?

- **同じ結果にはならない!!** (「予想される FP の不確実性の範囲内では一貫した結果になる」)

外部環境の変更により、データ・アライメントは実行ごとに変わる可能性がある

- ベクトル化されたカーネルで実行されるループの反復が変わる可能性がある
- ピールループとリマインダー・ループの反復にメインカーネルとは異なる最適化が適用される可能性がある
- リダクション・ループでは、部分和が変わり、丸めと最終結果が異なる

アライメントへの依存を回避するため、-qno-opt-dynamic-align オプションを指定してコンパイル

- -fp-model precise や consistent よりもパフォーマンスへの影響が少ない

または、データをアライメントする
例: _mm_malloc(size,64) (C/C++)、-align array64byte (Fortran) など

実行ごとの変動 (Linux* の例)

```
        gettimeofday(&start, NULL);
        na = (int)start.tv_sec%16;    //  a のサイズは時間によって異なる
        a = (float *) malloc(4*na);    //  a の可変サイズにより b のアライメントが変わる
        b = (float *) malloc(4*nb)
         …
//    ***********  ベクトル化されたリダクション  ***********
        for (int i=0; i<nb; i++) sum = sum + b[i];    // 加算の順序はアライメントに依存する

> icc  -xavx  -o run_to_run_icc  run_to_run.cpp;  ./run_to_run.sh   // 1 秒に 1 回実行
time in secs %16 =  8   start address = 0x609950   alignment = 16   sum = -8.819996
…
time in secs %16 = 12  start address = 0x609960   alignment = 32   sum = -8.82
…
time in secs %16 =  0   start address = 0x609930   alignment = 48   sum = -8.819996
…
time in secs %16 =  4   start address = 0x609940   alignment = 0    sum = -8.82
```

実際の例では、日付、時間、ユーザー名、実行ディレクトリー名を保持するため malloc で文字列が割り当てられていた (必ずしもユーザーコードに含まれているとは限らない)

同じ実行ファイル: 異なるプロセッサー

主な違いは数学ライブラリーのランタイム・ディスパッチに起因する

- 異なるプロセッサーでは関数実装が異なる
  - 例: インテル® AVX2 以降では FMA を利用可能
  - 精度が高く (そして差異が大きく)、パフォーマンスが優れている
- 一貫した数学関数の結果を得るには -fimf-arch-consistency=true を使用する
  - -qno-opt-dynamic-align と一緒に使用する
  - FMA などの新しい機能を使用できないためパフォーマンスに影響する

インテル® コンパイラー 18.0 以降では再現性が向上

- 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) (-xavx) 以降をターゲットとする場合、libsvml でランタイム・ディスパッチが不要
  - -fimf-arch-consistency=true の代わりに -fimf-use-svml を使用
  - パフォーマンスが向上する可能性がある

同じ実行ファイル: 異なるドメイン分割

異なるスレッド数/プロセス数

- ループ・パラメーター (反復回数など) は変わる可能性がある
  - データ・アライメントの変更
  - ベクトルカーネルで計算されるものや、ピールループ/リマインダー・ループの変更

優れた一貫性とパフォーマンスを実現するには:

- OpenMP* や MPI の標準のリダクションを使用しない
- -qopt-no-dynamic-align を使用してアライメントへの依存を抑止する
- -fimf-use-svml を使用してリマインダーとカーネルの間の数学関数の一貫性を保持する
- リマインダー・ループとベクトルカーネルは独立して最適化されているため、-fp-model precise や -fp-model consistent が必要になることがある

異なる実行ファイル

同じプロセッサー上で異なる最適化レベルの間で再現性を実現するには:

-fp-model precise -no-fma (/fp:precise /Qfma-)

異なる最適化レベルと異なるプロセッサー・タイプの間で浮動小数点演算結果の再現性を実現するには:

-fp-model consistent (/fp:consistent)

- -fp-model precise -fimf-arch-consistency=true -no-fma と同等

インテル® コンパイラー 18 または 19 では、-fimf-use-svml (/Qimf-use-svml) を追加してみる

- 再現性を維持しながらベクトル数学関数を再度有効にする
- パフォーマンスへの影響を軽減する可能性がある
- 結果はわずかに精度が低くなる可能性がある

異なるオペレーティング・システムでの再現性

Windows* と Linux* の間で再現性に違いはあるのか?

- 現在検証またはサポートされていない
  - インテルの数学ランタイム・ライブラリーでは意図的な違いはない
  - ベクトル化の既知の違いにより数学ライブラリー呼び出しが異なる
    - libm と SVML
  - 18.0 以降では、/Qimf-use-svml (-fimf-use-svml) を使用して回避できる

- 最良のオプション:
  /fp:consistent /Qimf-use-svml と -fp-model consistent -fimf-use-svml
  - Windows* から Linux*への移行に役立つ
  - 開発中 …

異なるインテル® コンパイラー・バージョン間での再現性は?

-fp-model precise (/fp:precise) を指定しても再現性が得られるとは限らない

- コンパイラーにより生成されたコードの結果は変更すべきではない
- 数学関数の結果は変更可
  - 通常、パフォーマンスではなく、精度を向上するために行われる
  - 期待される精度が保持される
    - libm (libimf) の場合 0.6ulp
    - libsvml (ベクトル化されたループのデフォルト) の場合 4ulp 未満、高精度の場合 1ulp 未満

ulp = 最後の桁の単位、1 ビット

- -fp-model consistent (/fp:consistent) を試してみる
  - FMA などの新しい命令を除外する
  - または、-fimf-precision=high (/Qimf-precision:high) で差異を最小化する
  - 別の回避方法: 両方のコンパイラーで最新のランタイム・ライブラリーを使用する

数学関数の最終的な標準に準拠することで一貫性は向上するが、パフォーマンスが低下する可能性がある

インテル® パフォーマンス・ライブラリーにおける再現性

インテル® MKL とインテル® TBB

インテル® MKL

線形代数、FFT、スパースソルバー、統計など

- 高度に最適化およびベクトル化されている
- OpenMP* またはインテル® TBB を使用して内部でスレッド化されている
- デフォルトでは、同じプロセッサー上で繰り返し実行した場合、同一の結果にならない可能性がある
  - 操作順序の変更が原因
- 異なるプロセッサー上で繰り返し実行した場合、追加の差異が発生する可能性がある
  - インテル® MKL 関数は「ディスパッチされる」
    - 実行されているプロセッサー/マイクロアーキテクチャーを検出する
    - 検出したマイクロアーキテクチャー向けに最適化されたコードパスを実行する

インテル® MKL の条件付き数値再現性

特定の条件下で繰り返し実行した場合、同一の結果になる

- インテル® TBB バージョンではなく、OpenMP* バージョンを使用する
- スレッド数は同じにする
- OMP_SCHEDULE=static (デフォルト)
- OMP_DYNAMIC=false と MKL_DYNAMIC=false (または未設定)
- 同じ OS とアーキテクチャーを使用する (例: インテル® 64 アーキテクチャー・ベースの Linux*)
- 同じアーキテクチャーを使用するか、最小マイクロアーキテクチャーを指定する
  - `mkl_cbwr_set(MKL_CBWR_AUTO)` 呼び出し (実行ごと、同じプロセッサー)
  - `mkl_cbwr_set(MKL_CBWR_COMPATIBLE)` 呼び出し (すべてのプロセッサー)
  - `mkl_cbwr_set(MKL_CBWR_AVX2)` 呼び出し (インテル® プロセッサー開発コード名 Haswell、Knights Landing、Skylake 上では同一の結果になるが、インテル® プロセッサー開発コード名 Sandy Bridge と Nehalem では結果が異なる)
  - または環境変数 `MKL_CBWR_BRANCH=MKL_CBWR_...` を設定
    - `COMPATIBLE`、`SSE2`、`AVX`、`AVX2`、`AVX512`、`AVX512_MIC`、など

異なるプロセッサー間での再現性

結果は ISA ターゲットに依存する (同じ色 ⇒ 同じ結果)

- ISA がサポートされていない場合はデフォルトで AUTO になる

ISA ターゲット: プロセッサー開発コード名	AUTO	インテル® AVX512- MIC	インテル® AVX512	インテル® AVX2	インテル® AVX	インテル® SSE2 または互換
Knights Landing						
Skylake						
Haswell						
Sandy Bridge						
互換性のあるインテル 以外のプロセッサー						

ターゲット ISA がパフォーマンスに与える影響の推定

プロセッサー間での再現性にはコストが伴う

ターゲット ISA	予測される相対パフォーマンス
MKL_CBWR_AUTO	1.0
MKL_CBWR_AVX512	1.0
MKL_CBWR_AVX2	0.50
MKL_CBWR_AVX	0.27
MKL_CBWR_SSE2	0.12

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。

システム構成: 3 スレッド、4999x4999 行列でインテル® MKL 2018 の DGEMM を使用して、2 x インテル® Xeon® Platinum 8180M プロセッサー @ 2.50GHz、28 コア、32GB RAM、38MB キャッシュの Fedora* 25 システムで測定。

インテル® TBB

並列処理向けの C++ テンプレート・ライブラリー

- ユーザー定義タスクの動的スケジューリング
- parallel_reduce() パターンをサポート
- 繰り返し実行した場合、同一の結果にならない可能性がある

再現性のあるリダクションを実現するには:

- **parallel_deterministic_reduce()** テンプレート関数
- ユーザーが提供する本体が一貫した結果を生成する場合、同じバイナリーを繰り返し実行すると同一の結果になる (コンパイラーのセクションを参照)
  - スレッド数に依存しない
    - シンプルなパーティショナーは常に作業を同じ方法で分割する
  - しかし結果はシリアル・リダクションと異なる場合がある
  - パフォーマンスが低下する可能性がある

インテル® MPI ライブラリーにおける再現性

インテル® MPI ライブラリーの条件付き数値再現性

次の条件を満たす場合、ノード上のランクの分散に関する再現性

- ランク (プロセス) 数は固定
- マイクロアーキテクチャー (プロセッサー・タイプ) の変更なし
- 各ランク内のコンパイル済みコードは再現性がある

集合操作 (例: MPI_Reduce() などのグローバル・リダクション)

- 再現性のある実装を使用する必要がある
- 「トポロジーを意識しない」アルゴリズムでなければならない
  - つまり、ノード間のランクの分散に影響されない

アルゴリズムは環境変数で指定できる

I_MPI_ADJUST_REDUCE、_ALLREDUCE、_REDUCE_SCATTER、_SCAN、_EXSCAN、…

インテル® MPI ライブラリーの MPI_REDUCE() アルゴリズム

	ランク分散アルゴリズム	0246 node1 1357 node2	すべての node1	0145 node1 2367 node2	0123 node1 4567 node2
1	Shumilin				
2	二項				
3	トポロジーを意識した Shumilin				
4	トポロジーを意識した二項				
5	Rabenseifner				
6	トポロジーを意識した Rabenseifner				
7	K 項				

同じ色は同じ結果

インテル® Core™ i5-4670T プロセッサー @ 2.30GHz、4 コア、8GB メモリーベースの 2 つのノード。1 つは Red Hat* EL 6.5、もう一方は Ubuntu* 16.04 を実行。『The Parallel Universe 21 号』の「インテル® MPI ライブラリーの条件付き再現性」のサンプルコードを使用 (記事の最後にある「参考文献」を参照)。

「再現性」の定義は人それぞれ

浮動小数点演算結果を再現するための条件はツールによって異なる

- インテル® コンパイラー
  - 同じまたは異なるバイナリー、プロセッサー・タイプ、最適化レベル
  - 同じスレッド数、静的スケジューリング (OpenMP*)
- インテル® MKL
  - 同じバイナリー、同じスレッド数、静的スケジューリング、異なるプロセッサー・タイプ
- インテル® TBB
  - 同じバイナリー、異なるスレッド数、異なるプロセッサー・タイプ
- インテル® MPI ライブラリー
  - 同じバイナリー、同じランク数、同じプロセッサー・タイプとランタイム環境、異なるノード間のランク分散

関連情報

「インテル® コンパイラーの浮動小数点演算における結果の一貫性」
https://www.isus.jp/products/c-compilers/consistency-of-floating-point-results/

『インテル® MKL デベロッパー・ガイド』の「Obtaining Numerically Reproducible Results (数値再現性のある結果を得る)」セクション
https://software.intel.com/mkl-linux-developer-guide (英語)

『The Parallel Universe 21 号』の「インテル® MPI ライブラリーの条件付き再現性」
http://www.xlsoft.com/jp/products/intel/tech/documents.html?tab=2#pu21

『インテル® MPI ライブラリーのチューニング: 基本的な手法』の「Tuning for Numerical Stability (数値安定性のためのチューニング)」セクション
https://software.intel.com/articles/tuning-the-intel-mpi-library-basic-techniques (英語)

『インテル® C++/Fortran コンパイラー・デベロッパー・ガイドおよびリファレンス』の「浮動小数点演算」セクション
https://software.intel.com/fortran-compiler-developer-guide-and-reference (英語)
https://software.intel.com/cpp-compiler-developer-guide-and-reference (英語)

法務上の注意書きと最適化に関する注意事項

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。

インテルは、明示されているか否かにかかわらず、いかなる保証もいたしません。ここにいう保証には、商品適格性、特定目的への適合性、および非侵害性の黙示の保証、ならびに履行の過程、取引の過程、または取引での使用から生じるあらゆる保証を含みますが、これらに限定されるわけではありません。

本資料には、開発中の製品、サービスおよびプロセスについての情報が含まれています。本資料に含まれる情報は予告なく変更されることがあります。最新の予測、スケジュール、仕様、ロードマップについては、インテルの担当者までお問い合わせください。

本資料で説明されている製品およびサービスには、エラッタと呼ばれる不具合が含まれている可能性があり、公表されている仕様とは異なる動作をする場合があります。現在確認済みのエラッタについては、インテルまでお問い合わせください。**絶対的なセキュリティーを提供できる製品またはコンポーネントはありません。**

Intel、インテル、Intel ロゴ、Intel Core、Xeon 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 またはその子会社の商標です。

* 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。

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

intel
Software