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) とヘテロジニアス・コンピューティング

Jackson Marusarz
開発製品部門

内容

- インテル® TBB の概要
- ヘテロジニアスの課題とそれらに対応するための概念
- 課題に対応するためのインテル® TBB の進化

インテル® TBB

threadingbuildingblocks.org

並列アルゴリズムとデータ構造
スレッドと同期
メモリー割り当てとタスク・スケジュール

汎用並列アルゴリズム

ゼロから始めることなく、マルチコアの能力を活かす効率的でスケーラブルな方法を提供

フローグラフ

並列処理を計算の依存性やデータフロー・グラフとして表すためのクラスのセット

コンカレント・コンテナー

同時アクセスを提供
外部ロックを使用するシリアルコンテナーのスケーラブルな代替手段

同期プリミティブ

アトミック操作、さまざまな特性の mutex、条件変数

タスク・スケジューラー

並列アルゴリズムとフローグラフを強化する洗練されたワーク・スケジュール・エンジン

スレッド・ローカル・ストレージ

無制限のスレッドローカル変数

スレッド

OS API ラッパー

その他

スレッドセーフなタイマーと例外クラス

メモリー割り当て

スケーラブルなメモリー・マネージャーとフォルス・シェアリングのないアロケーター

インテル® TBB

threadingbuildingblocks.org

並列アルゴリズムとデータ構造
スレッドと同期
メモリー割り当てとタスク・スケジュール

汎用並列アルゴリズム

ゼロから始めることなく、マルチコアの能力を活かす効率的でスケーラブルな方法を提供

フローグラフ

並列処理を計算の依存性やデータフロー・グラフとして表すためのクラスのセット

コンカレント・コンテナー

同時アクセスを提供
外部ロックを使用するシリアルコンテナーのスケーラブルな代替手段

同期プリミティブ

アトミック操作、さまざまな特性の mutex、条件変数

タスク・スケジューラー

並列アルゴリズムとフローグラフを強化する洗練されたワーク・スケジュール・エンジン

スレッド・ローカル・ストレージ

無制限のスレッドローカル変数

スレッド

OS API ラッパー

その他

スレッドセーフなタイマーと例外クラス

メモリー割り当て

スケーラブルなメモリー・マネージャーとフォルス・シェアリングのないアロケーター

汎用並列アルゴリズム

ループの並列化

parallel_for

parallel_reduce

- ロードバランスの良い並列実行
- 独立した反復数 (固定)

parallel_scan

- 並列プリフィクスを計算

  y[i] = y[i-1] op x[i]

並列ソート

parallel_sort

並列関数呼び出し

parallel_invoke

- 複数のユーザー定義関数を並列実行

ストリーム向け並列アルゴリズム

parallel_do

- 非構造化ストリームまたは大量のワークに使用
- 実行中にワークを追加可能

parallel_for_each

- ワークを追加できないことを除き parallel_do と同じ

pipeline / parallel_pipeline

- ステージの線形パイプライン
- 各ステージは並列、シリアル・インオーダー、またはシリアル・アウトオブオーダー
- キャッシュを効率良く使用

計算グラフ

flow::graph

- ノード間の依存性を実装
- ノード間でメッセージを引き渡し

例: parallel_for

C++11 (λ) 構文の利点

```
void sum(const int* in, const int* in2,
         std::size_t size, int* out)
{
  tbb::parallel_for(std::size_t(0), size,
                    [=](std::size_t i) {
                      out[i] = in[i] + in2[i];
                    } );
}
```

インテル® TBB

threadingbuildingblocks.org

並列アルゴリズムとデータ構造
スレッドと同期
メモリー割り当てとタスク・スケジュール

汎用並列アルゴリズム

ゼロから始めることなく、マルチコアの能力を活かす効率的でスケーラブルな方法を提供

フローグラフ

並列処理を計算の依存性やデータフロー・グラフとして表すためのクラスのセット

コンカレント・コンテナー

同時アクセスを提供
外部ロックを使用するシリアルコンテナーのスケーラブルな代替手段

同期プリミティブ

アトミック操作、さまざまな特性の mutex、条件変数

タスク・スケジューラー

並列アルゴリズムとフローグラフを強化する洗練されたワーク・スケジュール・エンジン

スレッド・ローカル・ストレージ

無制限のスレッドローカル変数

スレッド

OS API ラッパー

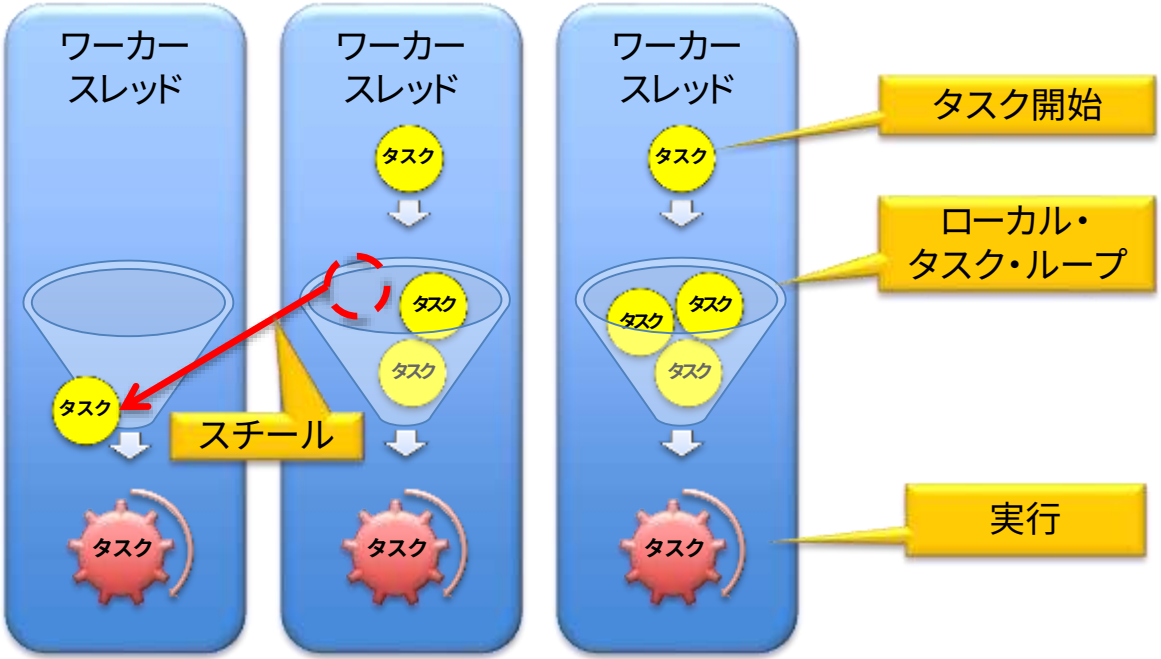
その他

スレッドセーフなタイマーと例外クラス

メモリー割り当て

スケーラブルなメモリー・マネージャーとフォルス・シェアリングのないアロケーター

インテル® TBB のタスク実行



(スケジューラーの簡略版)

インテル® TBB

threadingbuildingblocks.org

並列アルゴリズムとデータ構造
スレッドと同期
メモリー割り当てとタスク・スケジュール

汎用並列アルゴリズム

ゼロから始めることなく、マルチコアの能力を活かす効率的でスケーラブルな方法を提供

フローグラフ

並列処理を計算の依存性やデータフロー・グラフとして表すためのクラスのセット

コンカレント・コンテナー

同時アクセスを提供
外部ロックを使用するシリアルコンテナーのスケーラブルな代替手段

同期プリミティブ

アトミック操作、さまざまな特性の mutex、条件変数

タスク・スケジューラー

並列アルゴリズムとフローグラフを強化する洗練されたワーク・スケジュール・エンジン

スレッド・ローカル・ストレージ

無制限のスレッドローカル変数

スレッド

OS API ラッパー

その他

スレッドセーフなタイマーと例外クラス

メモリー割り当て

スケーラブルなメモリー・マネージャーとフォルス・シェアリングのないアロケーター

スケーラブルなメモリー割り当て

問題

- 並列環境ではメモリー割り当てがボトルネックになる
  - グローバルヒープからメモリーを割り当て/解放するため、スレッドはグローバルロックを取得する必要がある

解決策

- インテル® TBB はスレッドごとのメモリー管理に基づいた、検証済み、チューニング済みのスケーラブルなメモリー割り当てを提供
- Windows* および Linux* では、malloc_proxy ライブラリーを使用して、メモリー割り当てを対応するインテル® TBB 関数呼び出しへ自動的に置換可能
  - Linux*: libtbbmalloc_proxy.so.2 と libtbbmalloc_proxy_debug.so.2
  - Windows*: tbbmalloc_proxy.dll と tbbmalloc_debug_proxy.dll

インテル® TBB

threadingbuildingblocks.org

並列アルゴリズムとデータ構造
スレッドと同期
メモリー割り当てとタスク・スケジュール

汎用並列アルゴリズム

ゼロから始めることなく、マルチコアの能力を活かす効率的でスケーラブルな方法を提供

フローグラフ

並列処理を計算の依存性やデータフロー・グラフとして表すためのクラスのセット

コンカレント・コンテナー

同時アクセスを提供
外部ロックを使用するシリアルコンテナーのスケーラブルな代替手段

同期プリミティブ

アトミック操作、さまざまな特性の mutex、条件変数

タスク・スケジューラー

並列アルゴリズムとフローグラフを強化する洗練されたワーク・スケジュール・エンジン

スレッド・ローカル・ストレージ

無制限のスレッドローカル変数

スレッド

OS API ラッパー

その他

スレッドセーフなタイマーと例外クラス

メモリー割り当て

スケーラブルなメモリー・マネージャーとフォルス・シェアリングのないアロケーター

コンカレント・コンテナーはスレッドセーフで スケーラブルな STL の代替手段

STL は並列処理に対応していない
例えば、2 つのスレッドがそれぞれ次の処理を実行する場合:

```
extern std::queue q;
if(!q.empty()) {
   item=q.front();
   q.pop();
}
```

ここでもう 1 つのスレッドが最後の要素を pop する可能性がある

解決策: concurrent_queue の pop_if_present()

このほかにも concurrent_vector、concurrent_hash_map、concurrent_unordered_map がある

インテル® TBB

threadingbuildingblocks.org

並列アルゴリズムとデータ構造
スレッドと同期
メモリー割り当てとタスク・スケジュール

汎用並列アルゴリズム

ゼロから始めることなく、マルチコアの能力を活かす効率的でスケーラブルな方法を提供

フローグラフ

並列処理を計算の依存性やデータフロー・グラフとして表すためのクラスのセット

コンカレント・コンテナー

同時アクセスを提供
外部ロックを使用するシリアルコンテナーのスケーラブルな代替手段

同期プリミティブ

アトミック操作、さまざまな特性の mutex、条件変数

タスク・スケジューラー

並列アルゴリズムとフローグラフを強化する洗練されたワーク・スケジュール・エンジン

スレッド・ローカル・ストレージ

無制限のスレッドローカル変数

スレッド

OS API ラッパー

その他

スレッドセーフなタイマーと例外クラス

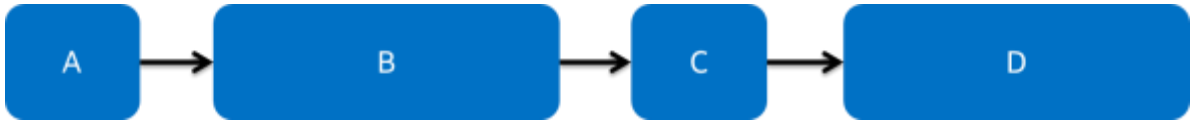
メモリー割り当て

スケーラブルなメモリー・マネージャーとフォルス・シェアリングのないアロケーター

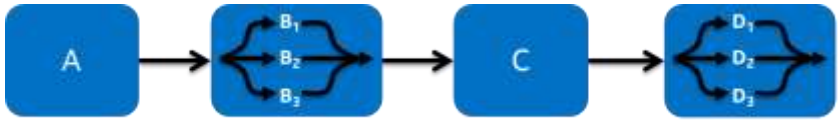
データフローとグラフ並列処理を行う理由

```
x = A();
y = B(x);
z = C(x);
D(y, z);
```

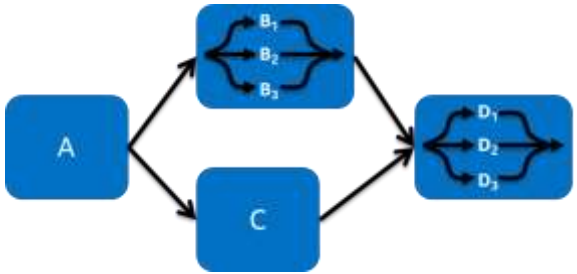
シリアル実装 (ベクトル化されている可能性あり)



ループ並列実装



ループおよびグラフ並列実装

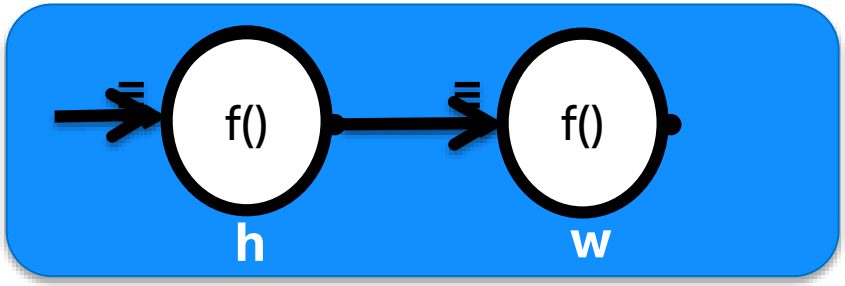


フローグラフの Hello World の例

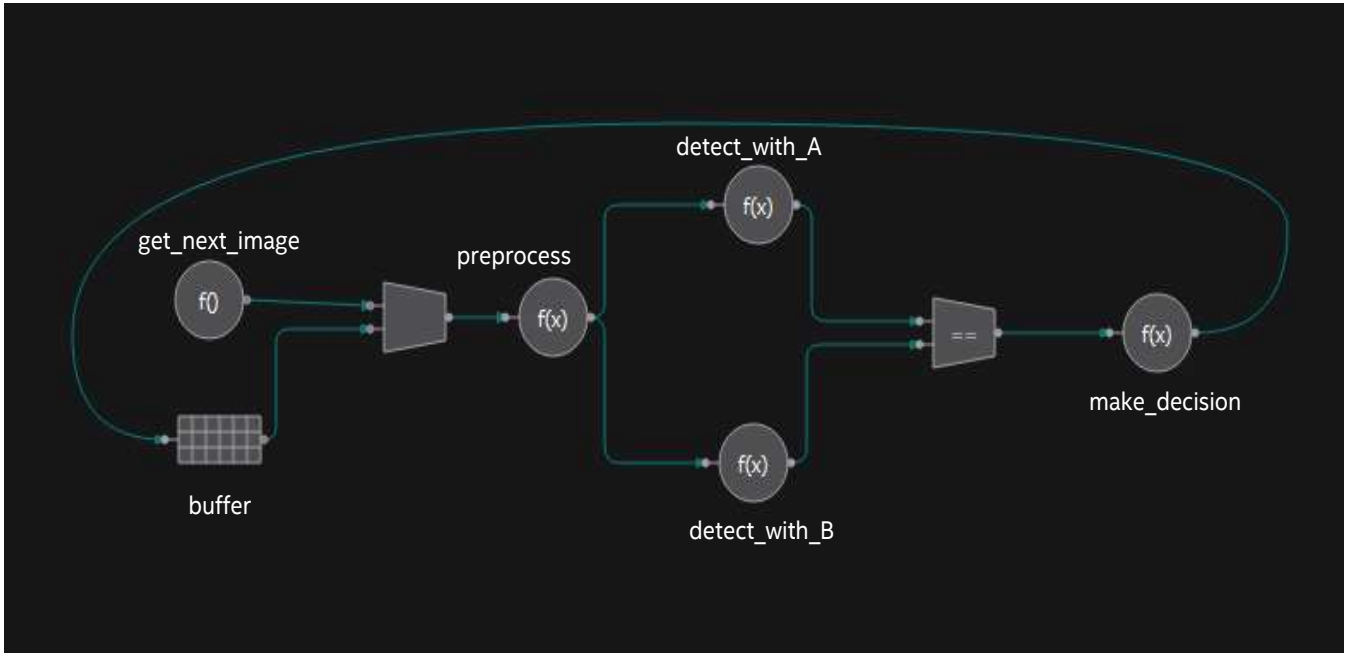
ノードとエッジを作成し、グラフを操作して、処理が完了するのを待機

```
tbb::flow::graph g;
tbb::flow::continue_node< tbb::flow::continue_msg >
  h( g, []( const continue_msg & ) { std::cout << "Hello "; } );
tbb::flow::continue_node< tbb::flow::continue_msg >
  w( g, []( const continue_msg & ) { std::cout << "World¥n"; } );
tbb::flow::make_edge( h, w );
h.try_put(continue_msg());
g.wait_for_all();
```

Hello World

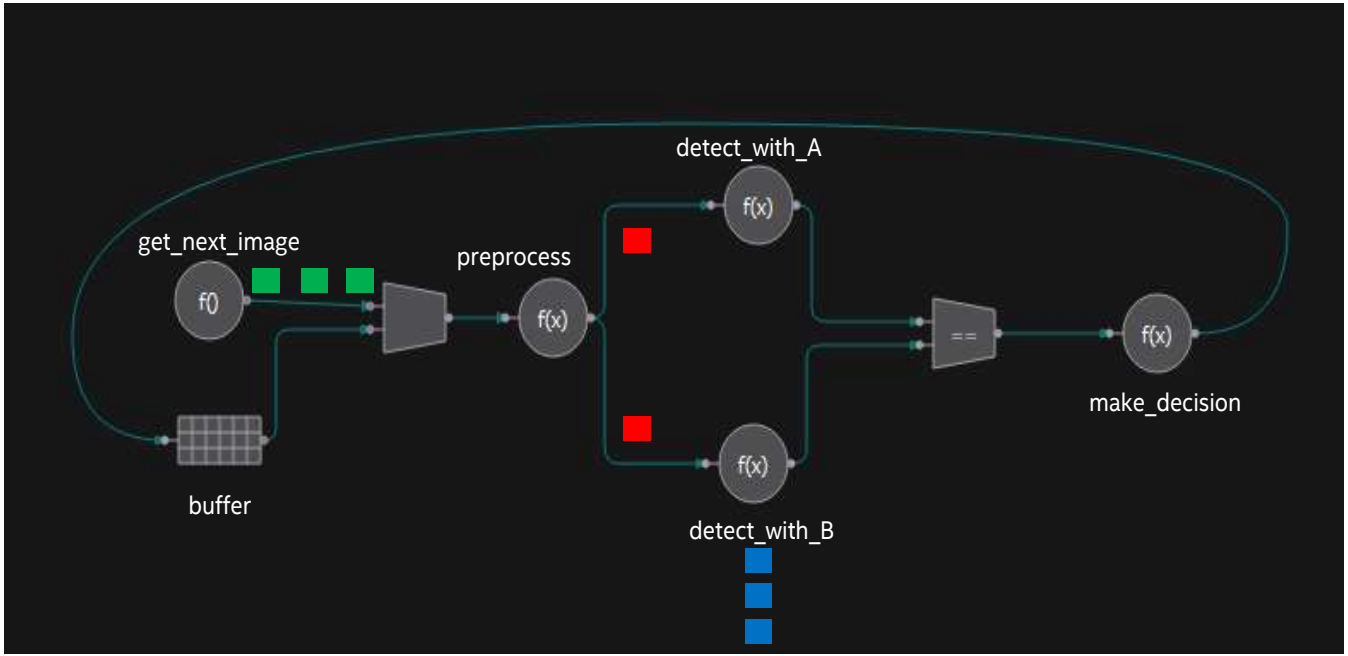


特徴検出アルゴリズムの例



パイプライン、タスク並列処理、データ並列処理を表現可能

特徴検出アルゴリズムの例



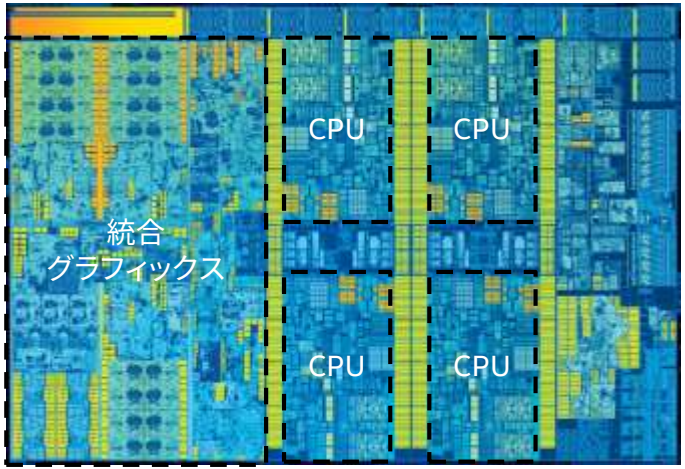
パイプライン、タスク並列処理、データ並列処理を表現可能
インテル® TBB、OpenMP*、インテル® Cilk™ Plus、
インテル® MKL などによる入れ子の並列処理をサポート

内容

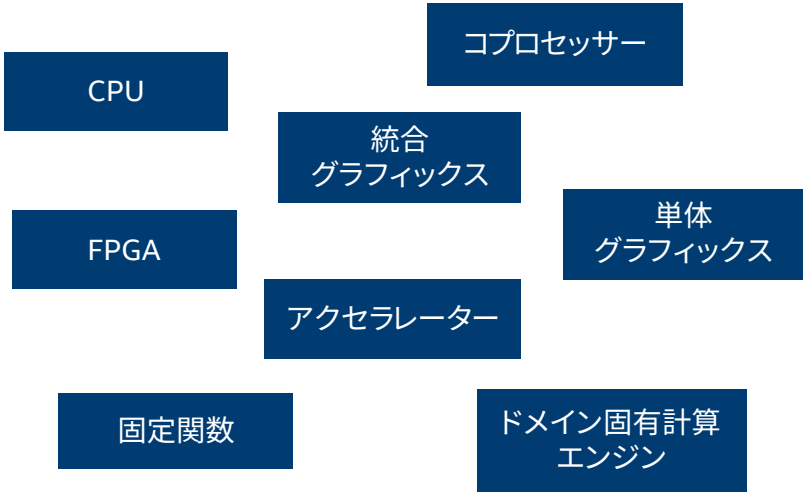
- インテル® TBB の概要
- ヘテロジニアスの課題とそれらに対応するための概念
- 課題に対応するためのインテル® TBB の進化

インテル® TBB は元々マルチコアおよびメニーコア共有メモリーシステム向けに開発されたが...

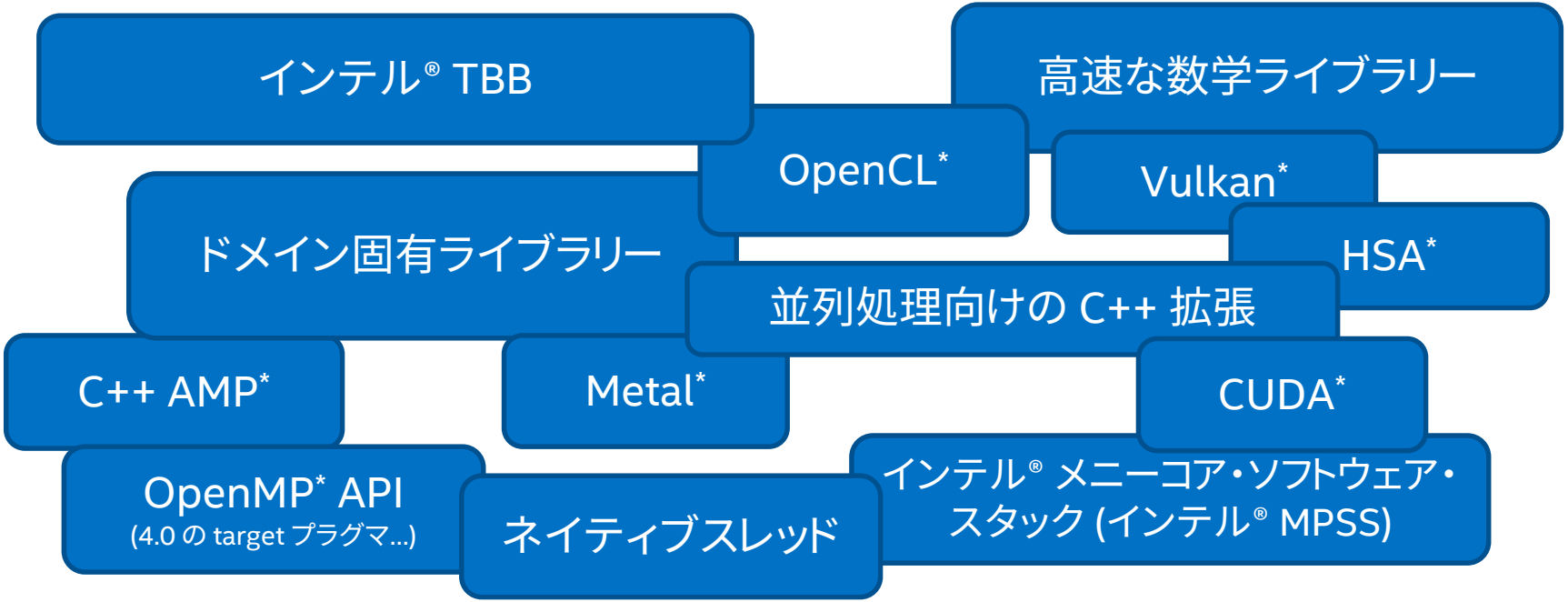
複数の計算リソースが利用可能になり環境がヘテロジニアス化



CPU に加えて、グラフィックス、メディア、計算向けの統合ユニット



ソフトウェアはさまざまなコンポーネント、モデル、ライブラリーを組み合わせて構築されている

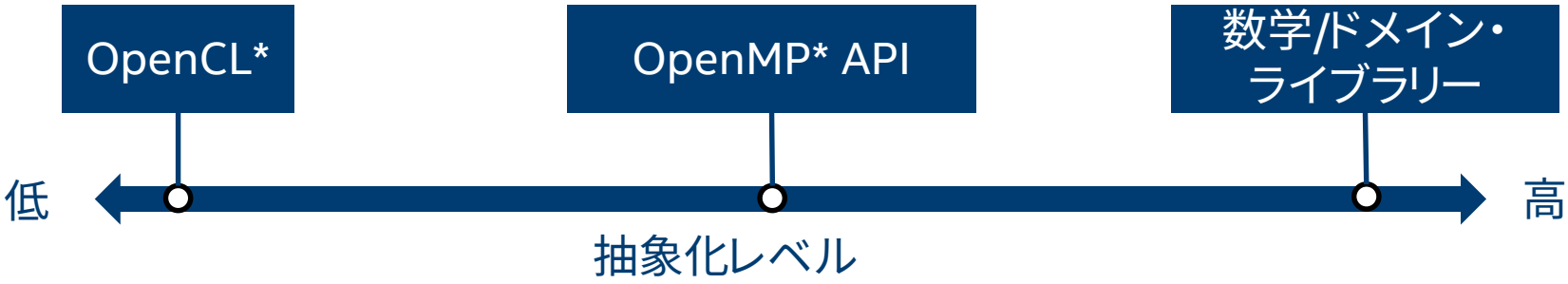


複雑なアプリケーションはこれらのアプローチを組み合わせたり、可能な場合はライブラリーを再利用したり、必要に応じて低水準のアプローチを使用して構築される

インテル® TBB はこのような複雑なアプリケーションに役立つか?

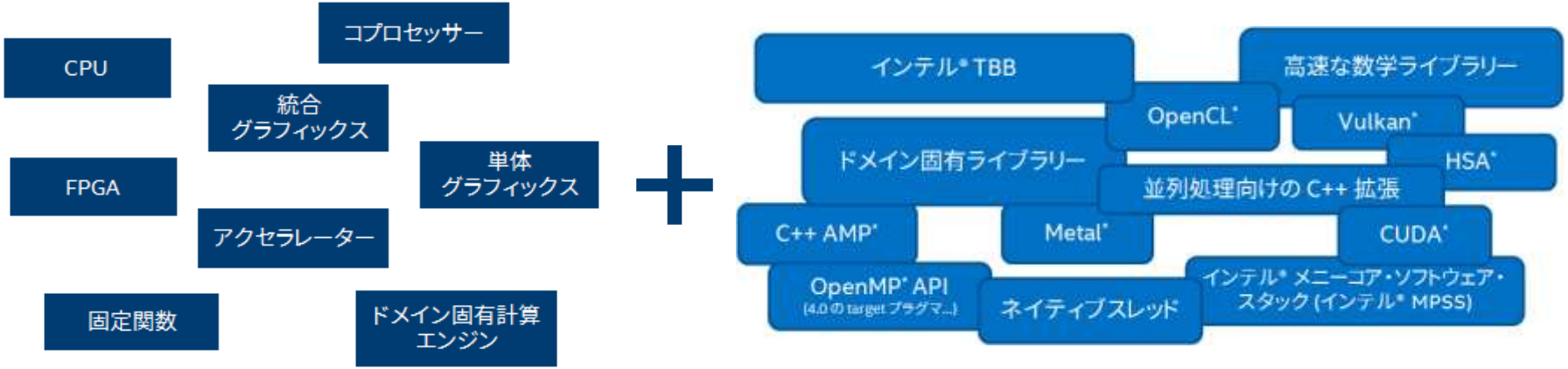
ヘテロジニアス・アプリケーションの共通点

1. デバイス向けの計算カーネルを表現/作成
2. データ依存性が尊重されるように実行順序を強制
3. パフォーマンスを最適化するようにカーネルをスケジュール
   (負荷分散、局所性の最大化、通信の最小化など...)



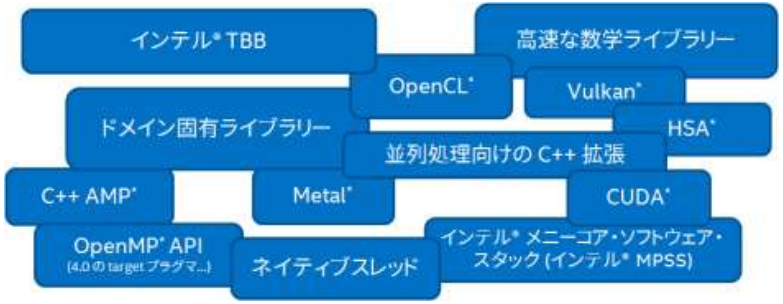
インテル® TBB のビジョン

柔軟性、最適化、既存モジュールとの連携を提供するヘテロジニアス調整レイヤー



1. 計算カーネルを表現/作成

- 計算カーネルとは、数学/ドメイン固有ライブラリーの呼び出し、スレッドを使用する C++ コード、OpenCL* カーネルなど



- カーネルは、システム上のそれらのモデルに対する既存サポートに依存

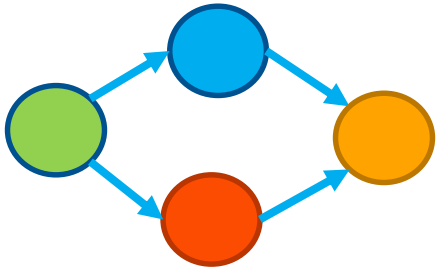
スレッドを使用する C++ コード

コプロセッサーへのオフロード

OpenCL* カーネル

数学ライブラリーの呼び出し

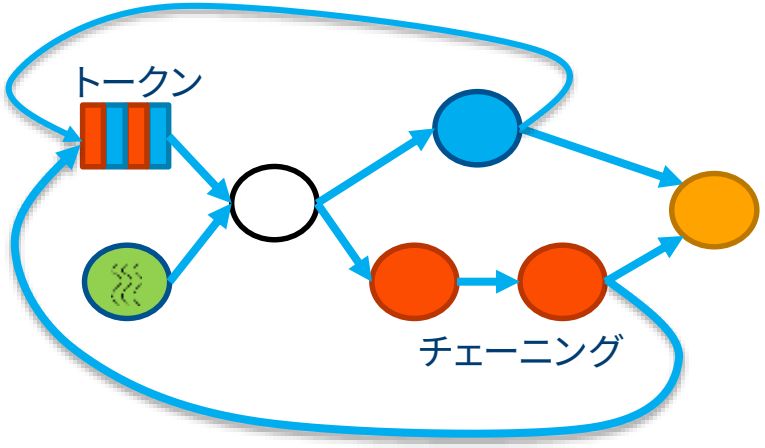
2. 依存性が尊重されるように順序を強制



- 計算カーネル間のデータフローを表現
- グラフにより半順序を指定
  - 正当性が保証されるように実行スケジュールを強制
  - **モジュールとライブラリーを含むカーネル間の並列性を明確にする**
- グラフによりデータ依存性を明示
  - **メモリーとモデルの境界を明確にする**
- モデルが混在する環境では、容易さからシリアル構成が使用されやすい
- グラフにより並列構成が可能

3. パフォーマンスを最適化するようにカーネルをスケジュール

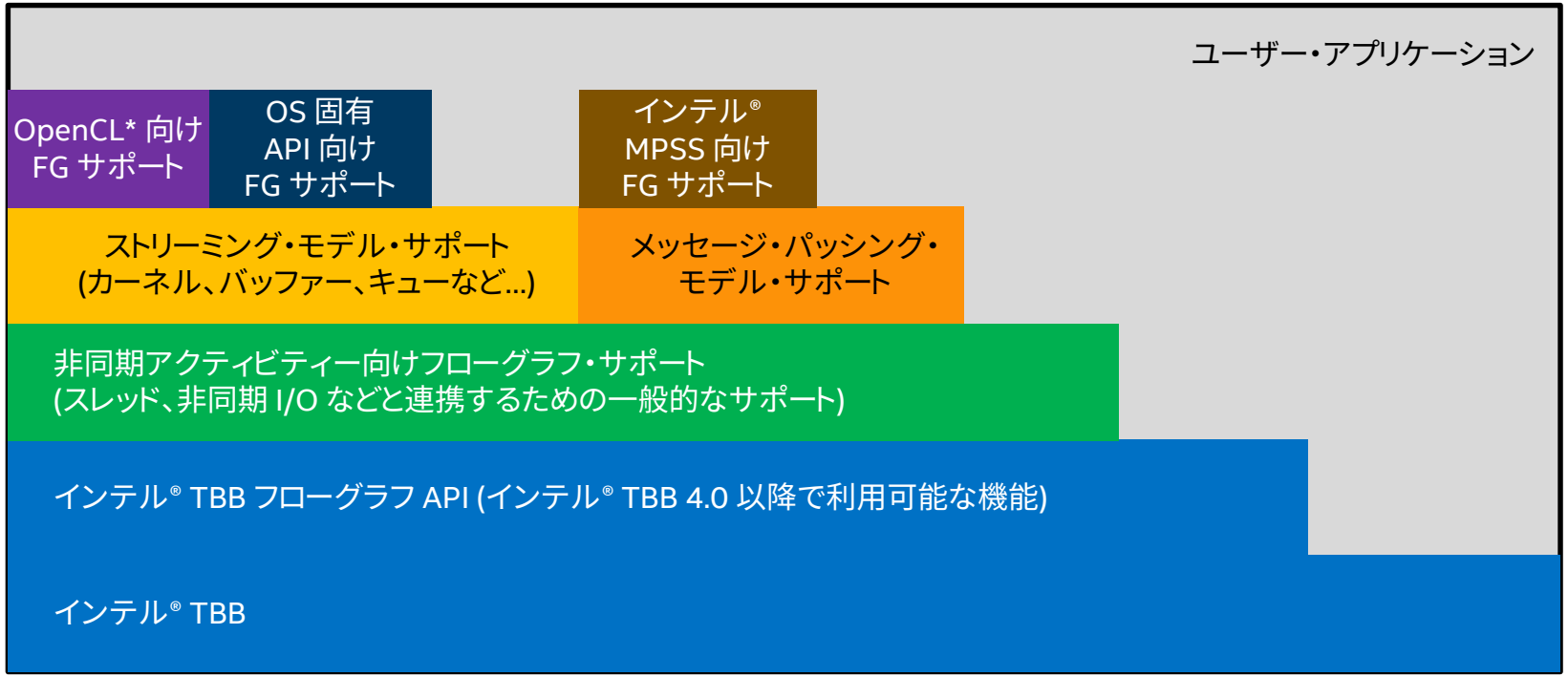
- ランタイムシステムに柔軟性を提供
- 配置オプション
  - 明示的/静的ユーザー制御
  - ドライバー/ライブラリーに任せる
  - トークンベースの配置
- モデルごと/デバイスごとの最適化を維持
  - 同じモデルやデバイスを使用してノードのサブグラフを最適化
  - メッセージによりノードをチェーニング



内容

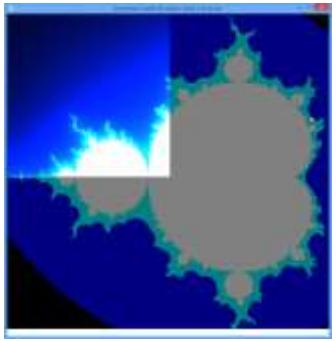
- インテル® TBB の概要
- ヘテロジニアスの課題とそれらに対応するための概念
- 課題に対応するためのインテル® TBB の進化

ヘテロジニアス向けインテル® TBB フローグラフ・アーキテクチャー

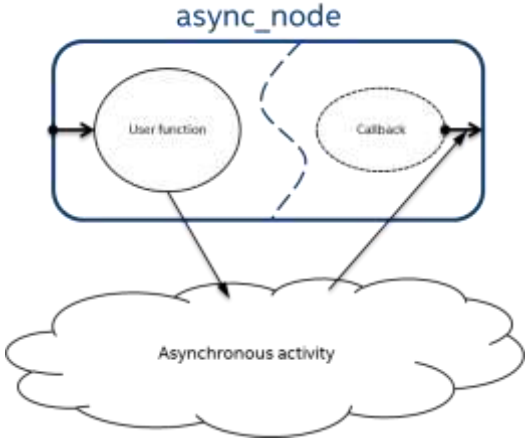


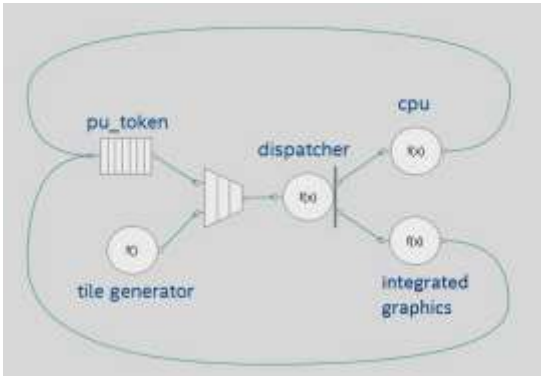
注: 未リリース機能は段階的にリリースされる予定ですが、変更される可能性があります。

例: async_node



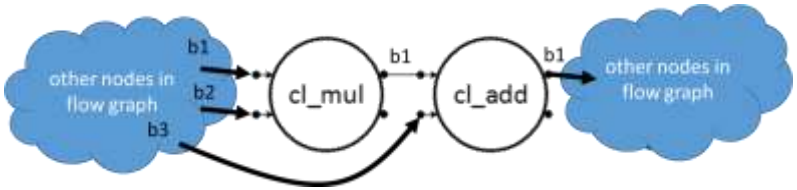
- データフロー・グラフによりデータを非同期アクティビティーへオフロードし、結果を受け取って CPU 上で継続を実行する
- トークンと組み合わせてデバイス間のロードバランスを取ることができる





async_node によりモデルを簡単に
効率良く利用可能

例: opencl_node



- サポートされる OpenCL* デバイスで実行可能な OpenCL* プログラムまたは SPIR バイナリーを受け取る 1 次ノードタイプを提供
- OpenCL* をサポートするデバイスには、CPU、統合/単体グラフィックス、FPGA などが含まれる
- https://software.intel.com/en-us/blogs/2015/12/09/opencl-node-overview

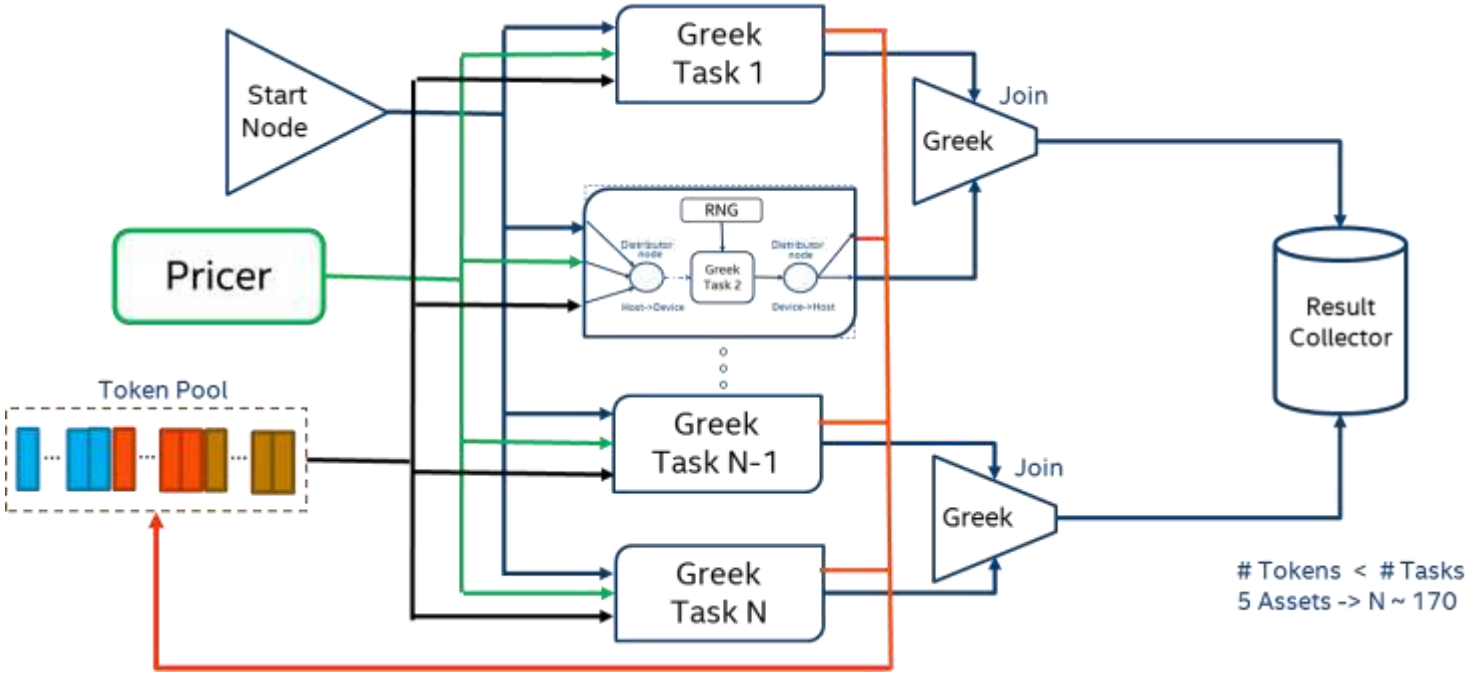
```
01 #define TBB_PREVIEW_FLOW_GRAPH_NODES 1
02 #include "tbb/flow_graph_opencl_node.h"
03
04 #include <algorithm>
05
06 int main() {
07     using namespace tbb::flow;
08
09     opencl_graph g;
10     opencl_node<tuple<opencl_buffer<cl_char>>>
11         clPrint( g, "hello_world.cl", "print" );
12
13     const char str[] = "Hello, World!";
14     opencl_buffer<cl_char> b( g, sizeof(str) );
15     std::copy_n( str, sizeof(str), b.begin() );
16
17     clPrint.set_ndranges( { 1 } );
18     input_port<0>(clPrint).try_put( b );
19
20     g.wait_for_all();
21
22     return 0;
23 }
```

distributor_node を利用した STAC-A2 実装

- 価格設定とリスク管理に関連した計算負荷が高い解析ワークロード
- インテル® TBB のフローグラフと並列アルゴリズム、OpenMP* のベクトル化を利用して実装
- フローグラフの非同期サポートにより “distributor_node” を作成し、インテル® Xeon Phi™ コプロセッサーへオフロード
- トークンベースのシステムにより CPU とコプロセッサー間の動的なロードバランスが可能
- 利用可能なリソースに応じて処理を実行するリソースを動的に決定

https://stacresearch.com/news/2015/11/03/stac-report-stac-a2-system-dual-xeon-phi-cards

STAC-A2 のインテル® TBB フローグラフ実装



https://stacresearch.com/news/2015/11/03/stac-report-stac-a2-system-dual-xeon-phi-cards

まとめ

- インテル® TBB は並列処理に広く利用されている C++ テンプレート・ライブラリー
- ヘテロジニアス環境と分散環境への対応を進めている
- 現在、async_node と opencl_node はプレビュー機能として利用可能
- その他のヘテロジニアス・サポートは現在開発中で、今後のリリースで利用可能になる予定 (低水準のビルディング・ブロックと高水準のサポート)
- インテル® TBB のビジョンは、ライブラリー、プログラミング・モデル、デバイスを連携させるための並列ソフトウェア・プラットフォームを提供すること

関連情報

製品 Web サイト

- インテル® TBB
  - http://www.isus.jp/intel-tbb/
  - https://www.threadingbuildingblocks.org/ (英語)
- Flow Graph Analyzer
  - https://software.intel.com/articles/flow-graph-designer (英語)
- インテル® Advisor XE
  - http://www.isus.jp/intel-advisor-xe/

Code Modernization (コードの現代化) 関連ページ

- インテル® Modern Code
  - http://www.isus.jp/article/idz/modern-code/
- インテル® Code Modernization Enablement Program
  - software.intel.com/code-modernization-enablement (英語)
- インテル® Parallel Computing Centers
  - software.intel.com/ipcc (英語)
- テクニカル Webinar シリーズの登録
  - http://bit.ly/spring16-tech-webinars (英語)
- インテル® Parallel Universe Magazine
  - http://www.isus.jp/products/psxe/issue/

法務上の注意書きと最適化に関する注意事項

本資料の情報は、現状のまま提供され、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。

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。

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固有の最適化は、インテル製マイクロプロセッサーでの使用を目的としています。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
experience
what's inside™