インテル® oneAPI ツールキット 2023 の新機能と強化された SYCL* 2020 への対応について

ハイパフォーマンス・ソフトウェア・カンファレンス 2023

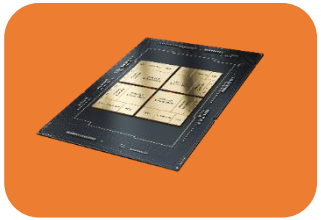
エクセルソフト株式会社

目次

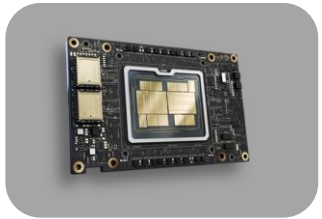
- インテル® oneAPI 2023 新機能について
- oneAPI の SYCL* への対応
  - SYCL* の有用な機能について

高い処理能力が求められる現代のアプリケーション

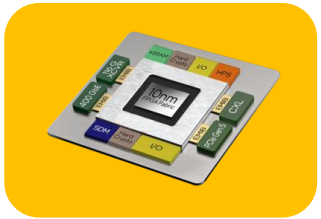
- 今日の性能要求に応えるためには、多様なアクセラレーターに対する開発が必要



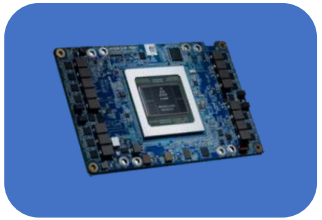
CPU



GPU



FPGA



その他
アクセラレーター

- 開発者の 48% が複数のプロセッサー/コアを使用する、ヘテロジニアス・システムをターゲットにしている[1]

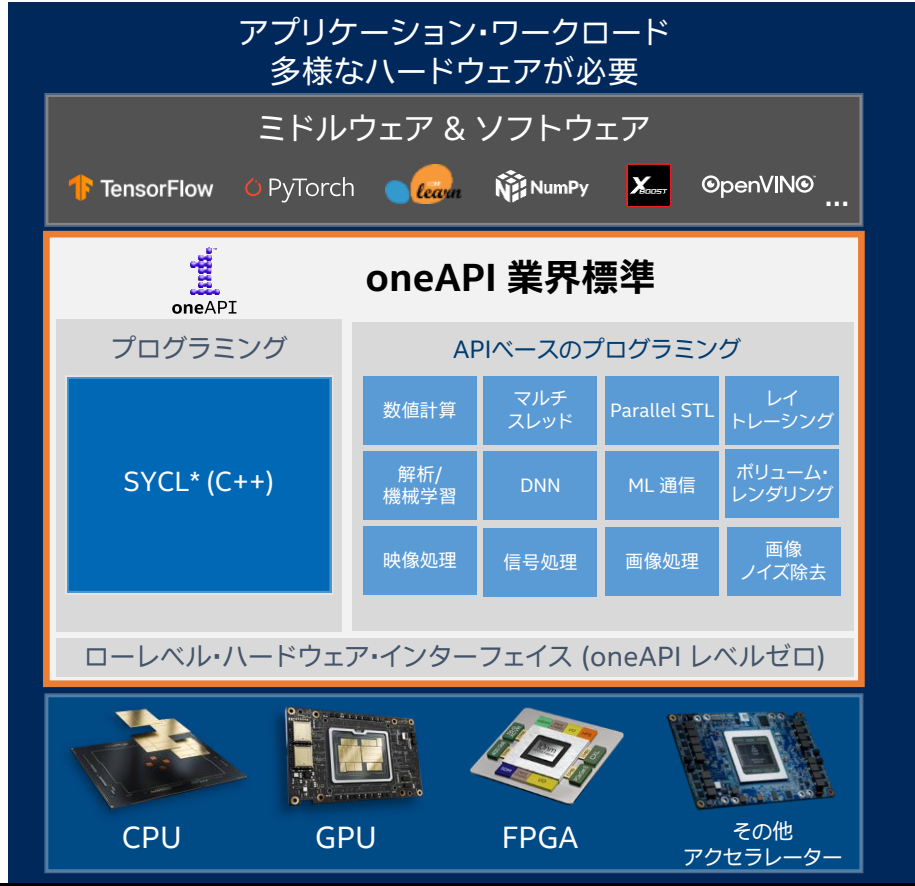
開発者の課題: 複数のアーキテクチャー、ベンダー、プログラミング・モデル

1. Evans Data Global Development Survey Report 22.1, June 2022

oneAPI

複数のアーキテクチャー、
1 つのプログラミング･モデル

- 自由な選択
  - 1 つのプログラミング･モデルで、複数のアーキテクチャーやベンダーに対応
- ハードウェアの実力を引き出す
  - CPU、GPU、FPGA、その他のアクセラレーターで優れたパフォーマンスを実現
- 安心してソフトウェアを開発･導入
  - C、C++ with SYCL*、Python*、OpenMP*、Fortran、MPI など既存の言語やプログラミング･モデルとの互換性あり
  - オープンな業界標準



インテル® oneAPI 2023 新機能ハイライト

- インテル® oneAPI DPC++/C++ コンパイラー
  - SYCL* サポートの改良
  - CPU、GPU オフロード性能の向上
- インテル® Fortran コンパイラー
  - Fortran 2018 までの言語標準をサポート
- 最新のインテル® アーキテクチャーへの対応

- インテル® DPC++ 互換性ツール
  - オープンソースの SYCLomatic プロジェクトがベース
  - 簡単に CUDA* コードから移行

インテル® oneAPI DPC++/C++ コンパイラー

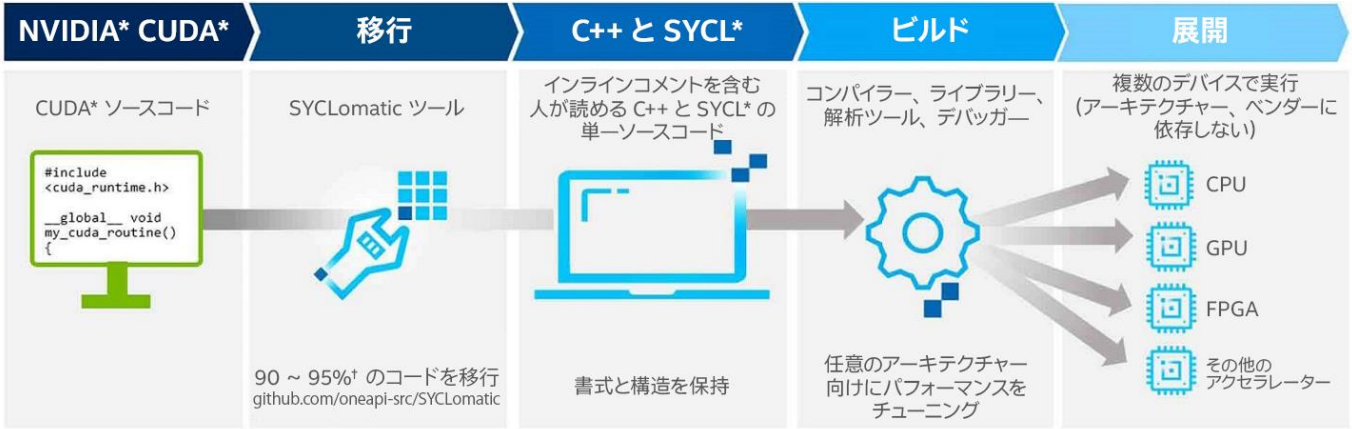
- C++17 がデフォルト言語へ
- SYCL* への対応強化
  - SYCL* 2020 機能の実装
- 最新のプロセッサーへの対応
  - インテル® Xeon® スケーラブル・プロセッサー (開発コード名 Sapphire Rapids)
  - インテル® データセンター GPU マックス・シリーズ (開発コード名 Ponte Vecchio)
- いくつかの非推奨通知
  - インテル® C++ コンパイラー・クラシック (icc) は非推奨であり、2023年後半リリースで削除される予定です

インテル® oneAPI DPC++/C++ コンパイラーへの移行

- icc/icpc、icl から icx/icpx への移行を推奨
  - インテル® C++ コンパイラー・クラシック (icc) は非推奨
  - 2023年後半リリースで削除される予定
  - インテル® oneAPI ポーティング・ガイド (dpcpp/icx) 日本語

インテル® DPC++ 互換性ツール

CUDA* から SYCL* への移行を簡単に



†2021 年 9 月現在のインテルによる推定。Rodinia、SHOC、PENNANT など、70 種類の HPC ベンチマークとサンプルの測定結果に基づいています。結果は異なることがあります。

‡その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。SYCL は Khronos Group の商標です。

- CUDA* で記述されたコードを C++ と SYCL* コードに変換
  - 通常 90～95% のコードが自動的に移行
- アーキテクチャー/ベンダーへの依存からの解放

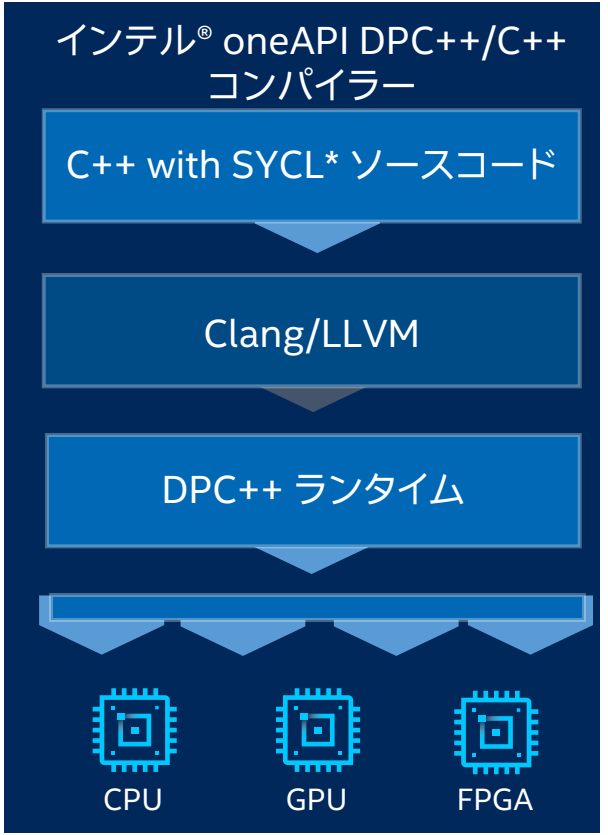
インテル® Fortran コンパイラー

- Fortran 2018 までの言語標準をサポート
  - Co-Array を含む
- DO CONCURRENT GPU オフロードをサポート
- OpenMP* 5.0、5.1 への準拠の強化、新機能が利用可能
- 最新のプロセッサーへの対応
  - インテル® Xeon® スケーラブル・プロセッサー (開発コード名 Sapphire Rapids)
  - インテル® データセンター GPU マックス・シリーズ (開発コード名 Ponte Vecchio)

oneAPI の SYCL* への対応

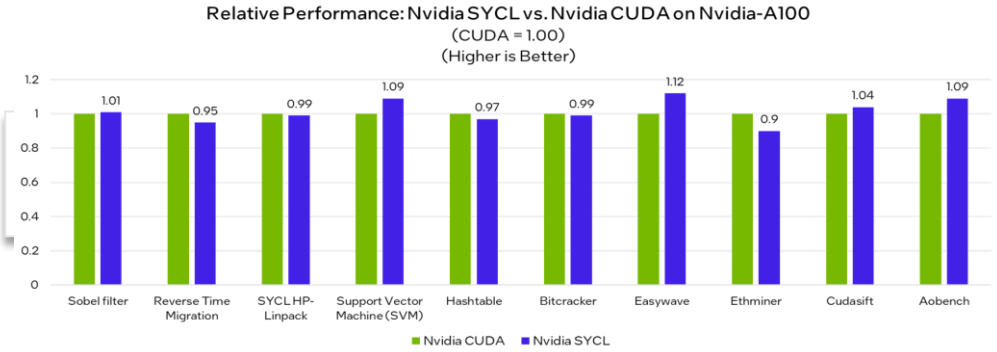
インテル® oneAPI DPC++/C++ コンパイラー

- CPU やアクセラレーターを問わず、妥協のない並列プログラミングの生産性とパフォーマンス
  - 特定のアクセラレーターのためのカスタム・チューニングを可能にしながら、ハードウェア・ターゲット間でコードを再利用可能
  - 単一アーキテクチャーの独自言語に代わる、オープンな業界横断型言語
- SYCL* 規格
  - 一般的で使い慣れた C および C++ 構造を使用
  - データ並列処理とヘテロジニアス・プログラミングをサポートするために Khronos Group によって作成
- インテルの数十年にわたるアーキテクチャーと高性能コンパイラーの経験を基に構築された



SYCL* という選択

- オープンで標準をベース
- マルチアーキテクチャー
- ベンダーロックインからの解放
- ネイティブ CUDA* に匹敵するパフォーマンス
- 広く利用されている C++ の拡張
- SYCLomatic やインテル製ツールを使用することで、コードの移行を高速化



Testing Date: Performance results are **based on testing by Intel as of August 15, 2022** and may not reflect all publicly available updates.

Configuration Details and Workload Setup: Intel® Xeon® Platinum 8360Y CPU @ 2.4GHz, 2 socket, Hyper Thread On, Turbo On, 256GB Hynix DDR4-3200, ucode 0x000363. GPU: Nvidia A100 PCIe 80GB GPU memory. Software: SYCL open source/CLANG 15.0.0, CUDA SDK 11.7 with NVIDIA-NVCC 11.7.64, cuMath 11.7, cuDNN 11.7, Ubuntu 22.04.1. SYCL open source/CLANG compiler switches: -fsycl-targets=nvptx64-nvidia-cuda, NVIDIA NVCC compiler switches: -O3 -gencode arch=compute_80, code=sm_80. Represented workloads with Intel optimizations.

Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See configuration disclosure for details. No product or component can be absolutely secure.

Performance varies by use, configuration, and other factors. Learn more at www.Intel.com/PerformanceIndex. Your costs and results may vary.

アーキテクチャー	Intel | NVIDIA | AMD CPU/GPU | RISC-V* | ARM Mali* | PowerVR* | Xilinx*

oneAPI の SYCL* への対応

- DPC++ は SYCL* 標準とその拡張を実装する LLVM プロジェクト
  - SYCL* 2020 以前より多くの機能の実装に取り組んできた
- SYCL* 2020 では、複数の DPC++ 拡張に基づく機能が採用

インテル® oneAPI DPC++/C++ コンパイラーで対応している SYCL* の仕様

- インテル® oneAPI 2023 で対応したもの
  - 新しいデバイス・セレクター・インターフェイス、テンプレート buffer_allocator、アトミック…

- SYCL* 2020 Specification Features (intel.com) (英語)
- インテル® oneAPI DPC++/C++ コンパイラー (dpcpp) でサポートされている SYCL* 2020 仕様の機能と DPC++ 言語拡張 | iSUS

インテル® oneAPI 2023 での変更

- SYCL* 2020 の対応により従来のデバイスセレクターは非推奨に
- SYCL* 1.2.1 で使われていたデバイスセレクターはコンパイル時に警告を出力

```
// SYCL* 1.2.1 のセレクター（非推奨）
output_dev_info( device{ host_selector{}}, "host_selector" );
output_dev_info( device{ gpu_selector{}}, "gpu_selector" );
// SYCL* 2020
output_dev_info( device{ host_selector_v}, "host_selector" );
output_dev_info( device{ gpu_selector_v}, "gpu_selector" );
```

- dpcpp コンパイラー・ドライバーが非推奨に

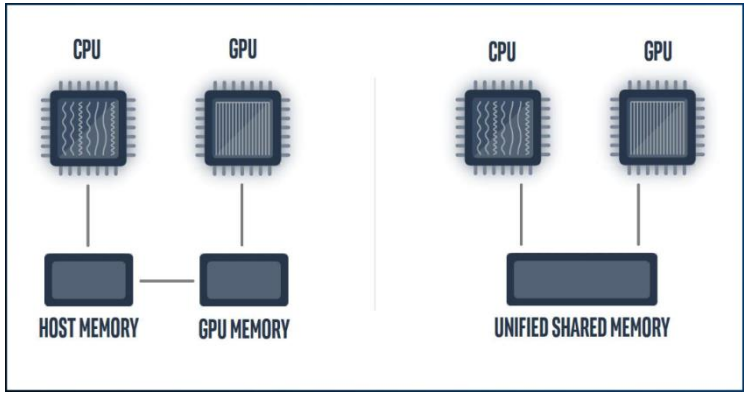
```
C:\Program Files (x86)\Intel\oneAPI>dpcpp test.cpp
icpx: warning: use of 'dpcpp' is deprecated and will be removed in a future release.
Use 'icpx -fsycl' [-Wdeprecated]
```

SYCL* 2020 の有用な機能

- 統合共有メモリー (USM)
- アトミック操作
- リダクション操作

USM (統合共有メモリー)

- ホストとデバイスで、単一のアドレス空間を提供する
  - ポインターはデバイス間で一貫
- バッファーとアクセサー
  - SYCL* 1.2.1 からのアクセスモデル
  - デバイス側でメモリーを管理する場合、アクセサーが必要
- USM (統合共有メモリー)
  - SYCL* 2020 で追加されたアクセスモデル
  - ホスト側とデバイス側が同じデータ・オブジェクトを利用する



USM (統合共有メモリー)

- 同じポインター・オブジェクトを参照するため、ホストとデバイス間のデータ移動は暗黙的に処理される

```
#include <iostream>
#include <sycl/sycl.hpp>
using namespace sycl;

int main() {
  // デフォルトキューの宣言
  queue myQueue;
  // USM の宣言
  int* data = sycl::malloc_shared<int>(1024, myQueue);
  // 0 から始まるランク番号でそれぞれの配列を初期化
  myQueue.parallel_for(1024, [=](id<1> idx) {
    data[idx] = idx;
  }).wait();

  // 結果の表示…
```

USM (統合共有メモリー)

- USM 割り当ての種類
  - sycl::malloc_host
    - ホストメモリー空間を割り当て、デバイスからアクセス可能
  - sycl::malloc_device
    - デバイスメモリー空間を割り当て、ホストからアクセス不可
    - デバイス上での処理を高速化するために使われる
  - sycl::malloc_shared
    - デバイスとホスト、両方からアクセス可能な空間を割り当てる

アトミック操作

- 複数のスレッドから、同時にアクセスされる共有メモリーについて、競合状態を回避するための手法
- インテル® oneAPI 2023 にて SYCL* 2020 アトミック操作に対応
- C++20 における std::atomic_ref を採用し、拡張した sycl::atomic_ref

```
// 処理するデータを割り当てます
int* data = sycl::malloc_shared<int>(2, myQueue);
// 初期化省略…
// 並列処理
myQueue.parallel_for(8, [=](id<1> idx){
   // data[0]、data[1] を繰り返し指定
   int index = idx % 2;
   // 競合が発生する
   data[index] += 1;
```

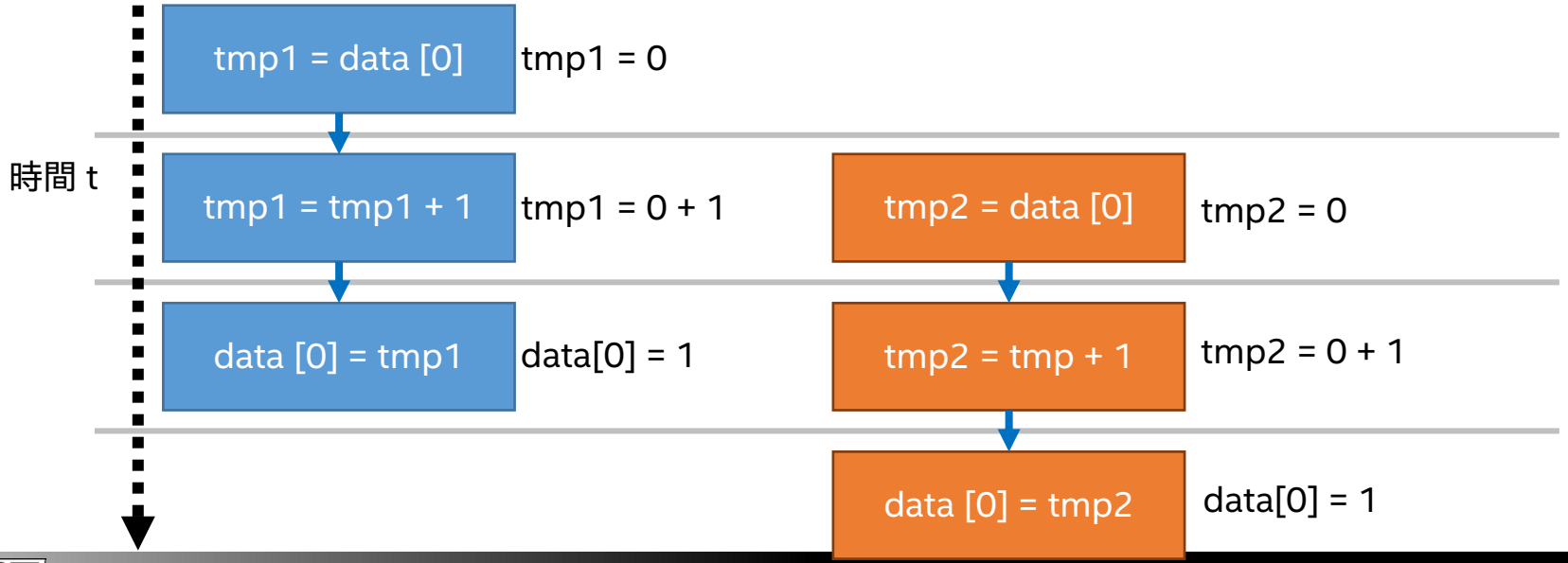
並列処理内で、data[0] と data[1] に対して、4 回ずつ +=1 を実行する

結果

data[0] = 1
data[1] = 1

アトミック参照

- 配列 data[0] += 1 について考える
  - data[0] の初期値: 0



アトミック操作

```
// 処理するデータを割り当てます
int* data = sycl::malloc_shared<int>(2, myQueue);
// 初期化省略…
// 並列処理
myQueue.parallel_for(8, [=](id<1> idx){
    // data[0]、data[1] を繰り返し指定
   int index = idx % 2;
   // アトミック操作で競合を回避する
   auto v = atomic_ref<int, memory_order::relaxed,
        memory_scope::device,
        access::address_space::generic_space>(data[index]);
   // +1
   v.fetch_add(1);
```

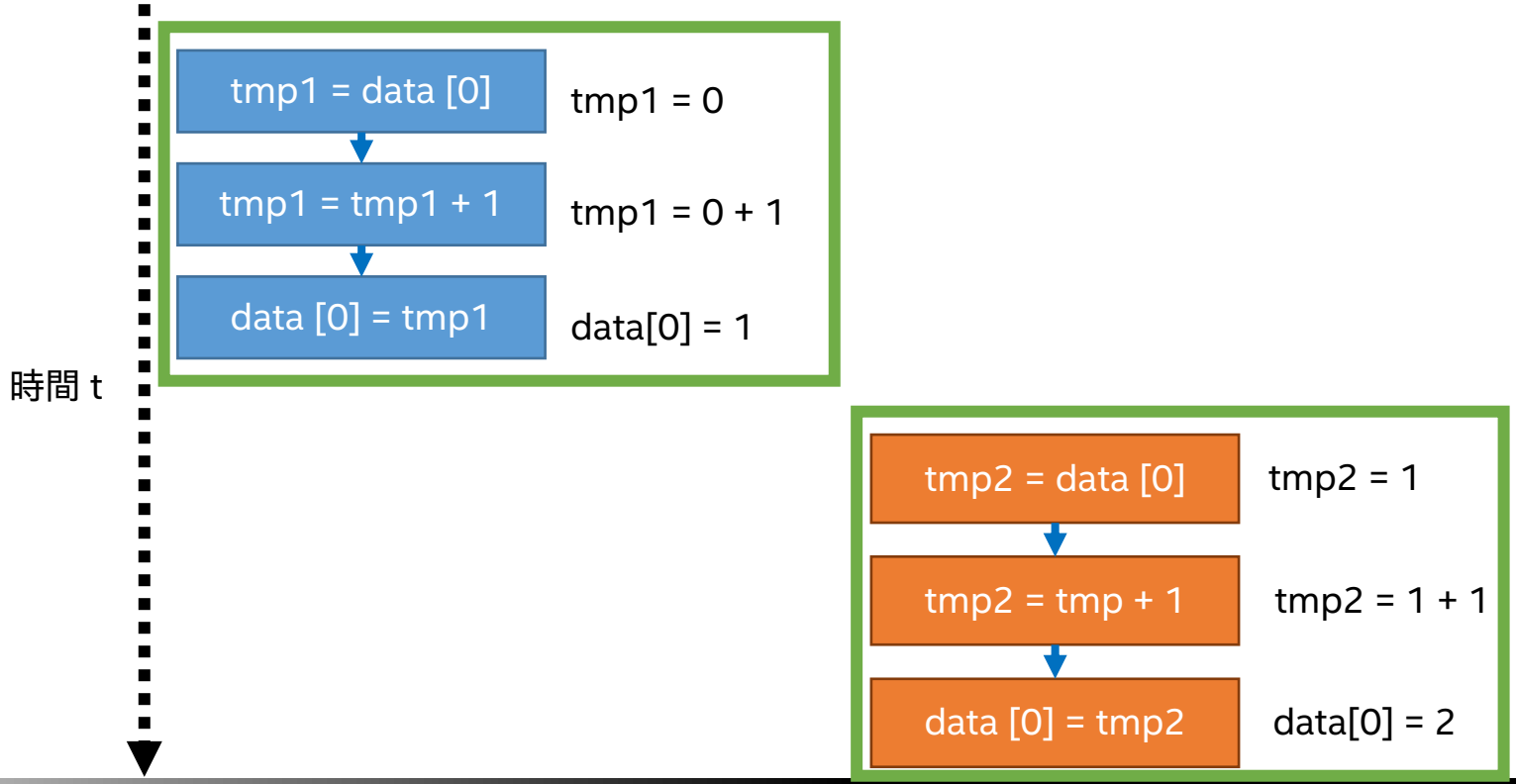
並列処理内で、data[0] と data[1] に対して、4 回ずつ +=1 を実行する

結果

data[0] = 4

data[1] = 4

アトミック参照


時間 t
tmp1 = data [0]
tmp1 = 0
tmp1 = tmp1 + 1
tmp1 = 0 + 1
data [0] = tmp1
data[0] = 1
tmp2 = data [0]
tmp2 = 1
tmp2 = tmp + 1
tmp2 = 1 + 1
data [0] = tmp2
data[0] = 2

リダクション操作

- 配列やコレクションなどの要素を単一の値にまとめる操作
- Reduction 関数と reducer クラスを使用することで、リダクションの記述が簡素化される

```
// リダクションの結果を取得するバッファー
int maxResult = 0;
buffer<int> maxBuf{ &maxResult,1 };

myQueue.submit([&](handler& cgh) {
    // リダクションの入力値は標準アクセサー
    auto inputValues = valueBuf.get_access<access_mode::read>(cgh);
    // リダクションのセマンティクスで変数を記述する一時オブジェクトを作成
    // 今回は MAXIMUM (最大値) を取得
    auto maxReduction = reduction(maxBuf, cgh, maximum<>());
    // リダクション操作を実施
    cgh.parallel_for(range<1> {N}, maxReduction, [=](id<1> idx, auto& max) {
        max.combine(inputValues[idx]);
    });
});
std::cout << "MAX:" << maxBuf.get_host_access()[0] << std::endl;
```

補足

インテル® Parallel Studio XE からの移行

- インテル® oneAPI 2023 リリースに伴い、インテル® Parallel Studio XE のすべての製品はインテル社のサポート対象外に
  - インテル社は最新バージョン + 過去 2 世代をサポート対象とするため
  - 現在はインテル® oneAPI 2023 (最新)、2022、2021 がサポート対象
- インテル® oneAPI へのアップグレードを推奨
  - 現在構築されているインテル® Parallel Studio XE 環境は引き続きご利用可能
  - ただし、インテル社からのサポート/アップデートはありません

インテル® oneAPI ライセンス形態について

- インテルによる優先サポートを得るための有償製品
- ユーザーに対して、サポートを提供させていただくライセンス
  - サポートを利用いただく人数や、サポートを利用されるシステムの規模により変動します

インテル® Parallel Studio XE	インテル® oneAPI
特定ユーザーライセンス (旧称 シングル・ユーザー・ライセンス)	特定ユーザー
フローティング・ライセンス 2-pack	ワークグループ (開発者 10 人サポート)
フローティング・ライセンス 5-pack	デパートメント (開発者 25 人サポート)

インテル® oneAPI 脆弱性

- 2023年1月に脆弱性がアナウンスされています
  - インテル® oneAPI DPC++/C++ コンパイラー
  - インテル® oneAPI C++コンパイラー･クラシック

- 権限昇格の脆弱性
- インテル® oneAPI 2022.3.1 以上へのアップグレードが推奨されております

お問い合わせはこちらまで

https://www.xlsoft.com/jp/qa