インテル® oneAPI ベース & HPC ツールキット

CUDA* から DPC++ へ移行して エッジの計算パフォーマンスを最適化

資料番号: 641591JA

インテル コーポレーション
テクニカル・コンサルティング・エンジニア
Jon Kim

インテル コーポレーション
テクニカル・コンサルティング・エンジニア
Sofea Azrina Azizan

intel®

内容

- oneAPI について
- データ並列 C++ (DPC++) の概要
- インテル® DPC++ 互換性ツール (インテル® DPCT) の概要
- インテル® DPCT の使用例
- インテル® DPCT のデモ
- まとめ

oneAPI について

プログラミングの課題

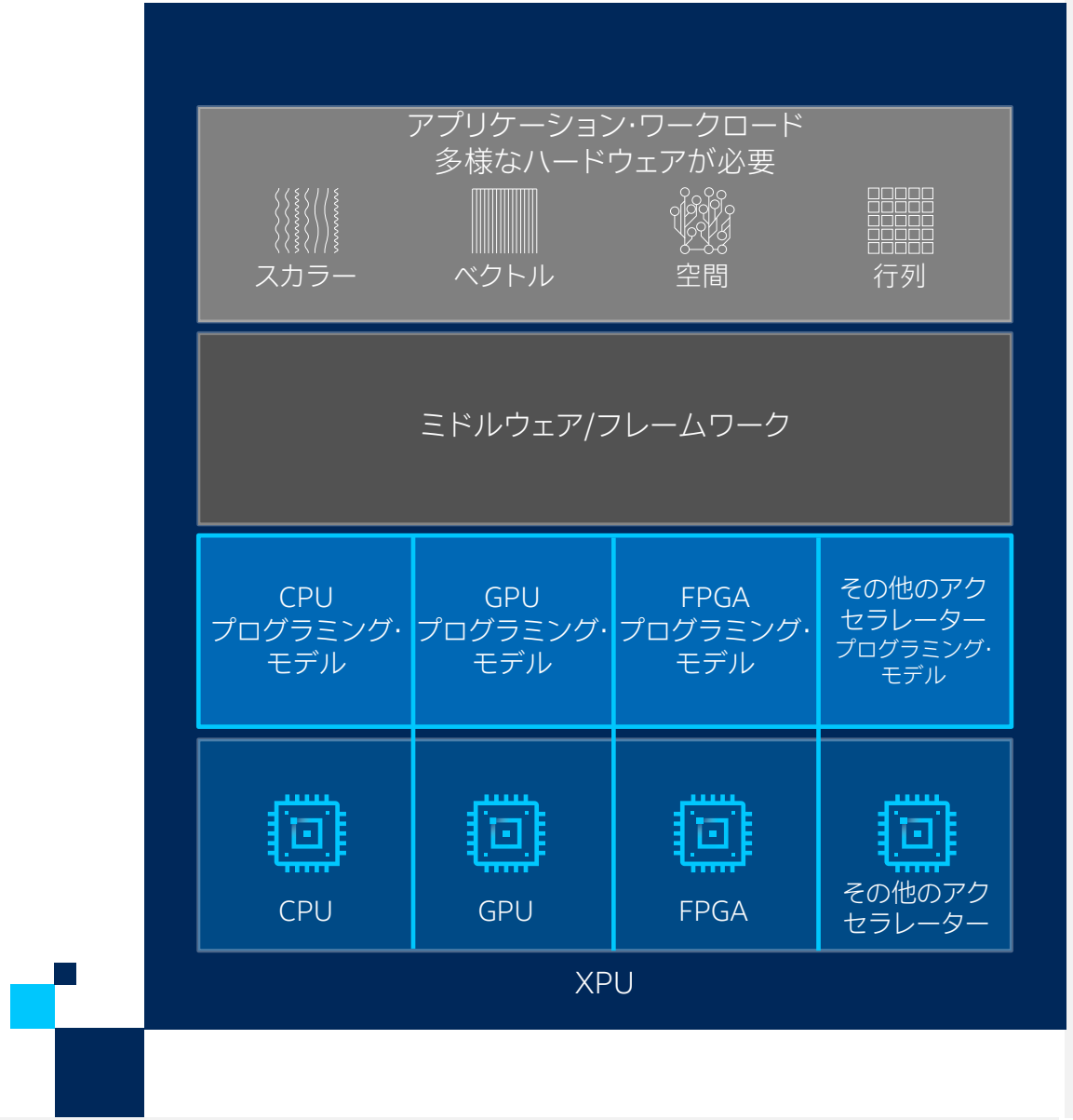
複数のアーキテクチャー

専門化したワークロードの増加

各種データセントリック・ハードウェアへの対応が必要

現在は各アーキテクチャー向けに個別のプログラミング・モデルとツールチェーンが必要

ソフトウェア開発の複雑さはアーキテクチャー選択の自由を制限する



インテル® oneAPI ツールキット

CPU から XPU までカバーする優れた開発者ツールの完全なセット

oneAPI

インテル® oneAPI ベース・ツールキット

ネイティブコード開発者

C++、データ並列 C++ アプリケーションおよび oneAPI ライブラリー・ベースのアプリケーションを構築する
ハイパフォーマンス・ツールの基本セット

ドメイン固有の アドオン・ツール キット

特殊なワークロード

インテル® oneAPI HPC ツールキット

スケーラブルで高速な Fortran、OpenMP* および MPI アプリケーションを開発

インテル® oneAPI IoT ツールキット

ネットワークのエッジで実行する、効率的な、信頼性の高いソリューションを構築

インテル® oneAPI AI アナリティクス・ツールキット

最適化された DL フレームワークとハイパフォーマンスの Python* ライブラリーでマシンラーニングとデータ・サイエンス・パイプラインを高速化

インテル® oneAPI レンダリング・ツールキット

ハイパフォーマンスで高忠実度のビジュアライゼーション・アプリケーションを作成

oneAPI 対応の ツールキット

データ・サイエンティスト/AI 開発者

OpenVINO™

OpenVINO™ ツールキット

エッジからクラウドまでハイパフォーマンスな推論とアプリケーションをデプロイ

インテル® oneAPI ベース & IoT ツールキット

概要

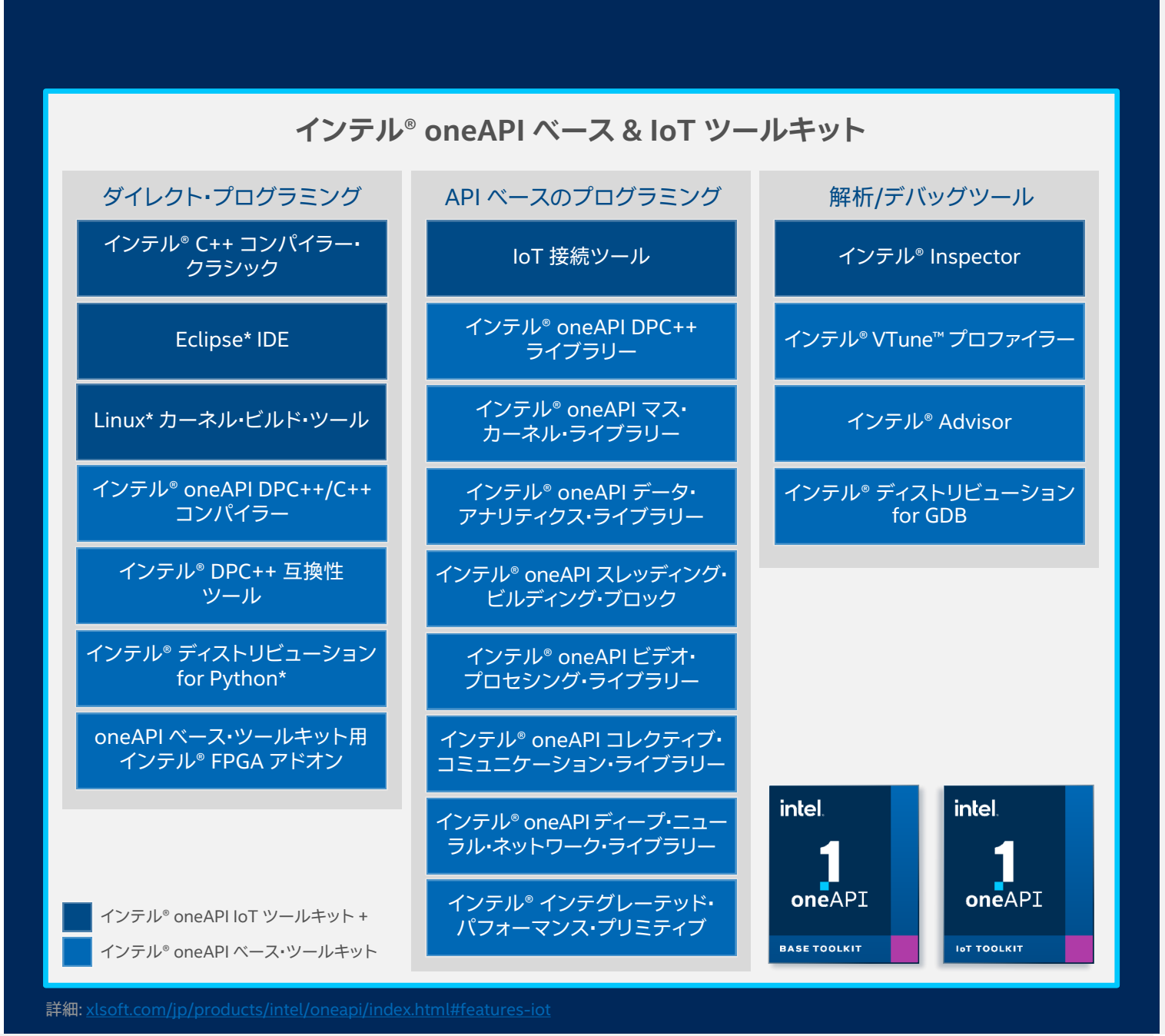
ネットワーク・エッジで動作するスマート・コネクテッド・デバイス向けの IoT アプリケーションの開発を高速化するために必要なものを提供する、インテル® oneAPI ベース・ツールキットのアドオン・ツールキット

対象ユーザー

- OEM、ODM、SI、ISV
- DPC++、C、C++、OpenMP*、Python* 開発者

ツールキットの重要性

- 最適化されたコンパイラーとライブラリーを使用してインテル・アーキテクチャー・ベースのプラットフォームの多くのコアとビルトイン・テクノロジーを活用
- IoT 接続ツールを利用してセンサーとデバイス、デバイスとクラウドを簡単に接続
- Yocto Project* プラットフォーム・プロジェクトの開発と保守を高速化
- スレッド化、メモリー、オフロードの最適化の可能性を見つける強力な解析ツールを使用して信頼できる開発が可能
- CUDA* で記述された既存のコードからの移行を支援するインテル® DPC++ 互換性ツール



詳細: xlsoft.com/jp/products/intel/oneapi/index.html#features-iot

oneAPI エコシステムのサポート

これらの組織は、クロスアーキテクチャー開発のための単一の統合プログラミング・モデルとして oneAPI イニシアチブのコンセプトをサポートしています。
インテルの製品の購入または使用に関する契約を示すものではありません。* 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。

データ並列 C++ (DPC++) の概要

インテル® oneAPI DPC++/C++ コンパイラー

並列プログラミングの生産性とパフォーマンスを向上

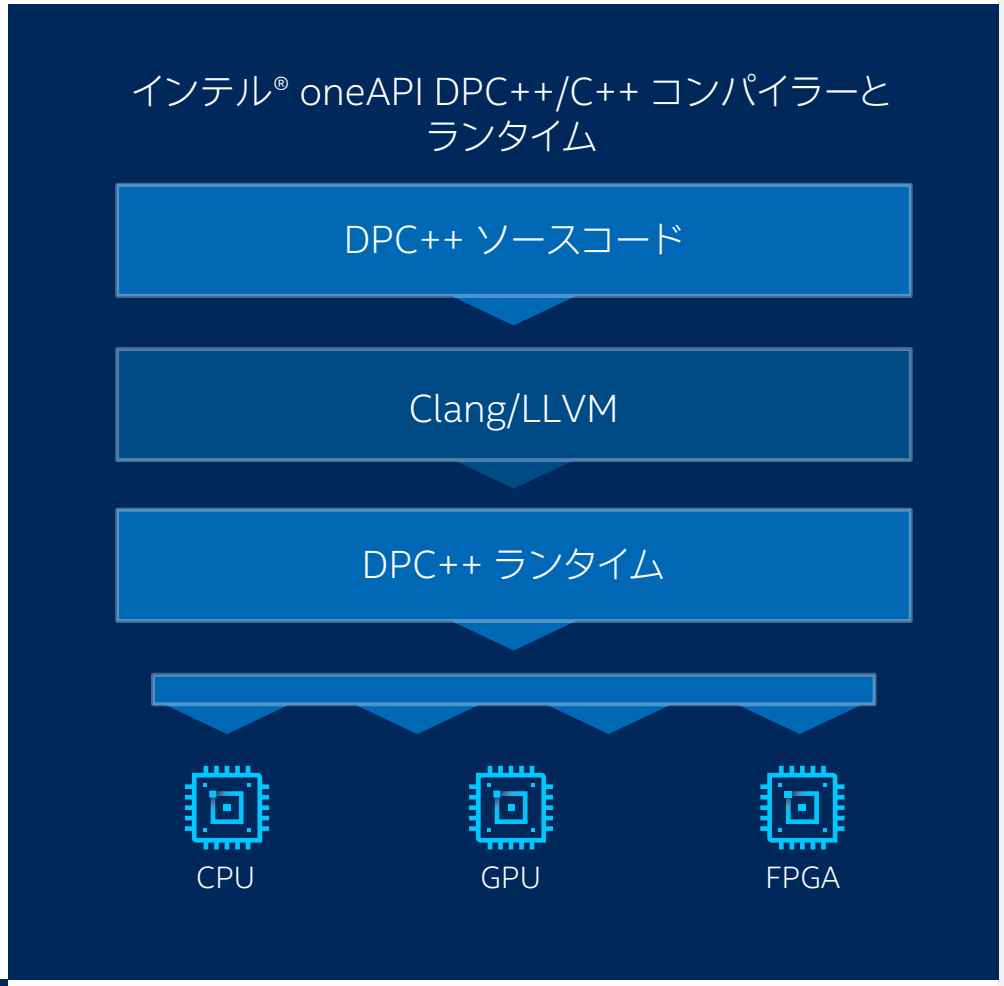
CPU とアクセラレーターに妥協のない並列プログラミングの生産性とパフォーマンスを提供するコンパイラー

- ターゲット・ハードウェア間でコードの再利用が可能、特定のアクセラレーター向けのカスタム・チューニングを行うことが可能
- 単一アーキテクチャー専用の言語に代わる、オープンな業界全体の代替手段

DPC++ は ISO C++ および Khronos SYCL* ベース

- C++ の生産性の利点を提供し、一般的で使い慣れた C および C++ 構造を使用
- The Khronos Group の SYCL* を継承し、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

インテルの数十年にわたるアーキテクチャーとハイパフォーマンス・コンパイラーの経験を基に構築



DPC++ コードの構造

```
void dpcpp_code(int* a, int* b, int* c) {
  // DPC++ デバイスキューを設定
  queue q;
  // 入力と出力ベクトル用のバッファーを設定
  buffer<int,1> buf_a(a, range<1>(N));
  buffer<int,1> buf_b(b, range<1>(N));
  buffer<int,1> buf_c(c, range<1>(N));
  // コマンドグループ関数オブジェクトをキューに送信
  q.submit([&](handler &h){
  // グローバルメモリーに割り当てられたバッファーへのデバイスアクセサーを作成
  auto A = buf_a.get_access<access::mode::read>(h);
  auto B = buf_b.get_access<access::mode::read>(h);
  auto C = buf_c.get_access<access::mode::write>(h);
  // デバイスカーネル本体をラムダ関数として指定
  h.parallel_for(range<1>(N), [=](item<1> i){
    C[i] = A[i] + B[i];
    });
  });
}
```

カーネル呼び出しは並列に実行

カーネルは範囲の各要素に対して呼び出される

カーネル呼び出しは呼び出し ID にアクセス可能

完了!
結果は `buf_c` バッファーを破棄する際にベクトル `c` にコピーバックされる

ステップ 1: デバイスキューの作成 (開発者はデバイスセレクターでデバイスを指定するか、デフォルトのセレクターを使用)

ステップ 2: バッファーの作成 (ホストとデバイスそれぞれのメモリー用)

ステップ 3: (非同期) 実行コマンドの送信

ステップ 4: デバイス上のバッファーデータにアクセスするバッファーアクセサーの作成

ステップ 5: 実行するカーネル (ラムダ関数) の送信

ステップ 6: カーネルの記述

SYCL* 統合共有メモリーのインテル拡張の例

統合共有メモリーではホスト側とデバイス側のメモリーに同一のポインター参照でアクセス可能

統合共有メモリーを設定
ホストで初期化
デバイスで変更
ホストで出力

```
queue q;
auto data =  malloc_shared<int>(N, q);
for(int i=0;i<N;i++) data[i] = 10;
q.parallel_for(N, [=](auto i){
        data[i] += 1;
}).wait();
for(int i=0;i<N;i++) std::cout << data[i] << " ";
free(data, q);
```

カーネル実行のグラフ

```
int main() {
  auto R = range<1>{ num };
  buffer<int> A{ R }, B{ R };
  queue Q;

  Q.submit([&](handler& h) {
    auto out = A.get_access<access::mode::write>(h);
    h.parallel_for(R, [=](id<1> idx) {
      out[idx] = idx[0]; }); });

  Q.submit([&](handler& h) {
    auto out = A.get_access<access::mode::write>(h);
    h.parallel_for(R, [=](id<1> idx) {
      out[idx] = idx[0]; }); });

  Q.submit([&](handler& h) {
    auto out = B.get_access<access::mode::write>(h);
    h.parallel_for(R, [=](id<1> idx) {
      out[idx] = idx[0]; }); });

  Q.submit([&](handler& h) {
  auto in = A.get_access<access::mode::read>(h);
  auto inout =
    B.get_access<access::mode::read_write>(h);
  h.parallel_for(R, [=](id<1> idx) {
    inout[idx] *= in[idx]; }); });
}
```

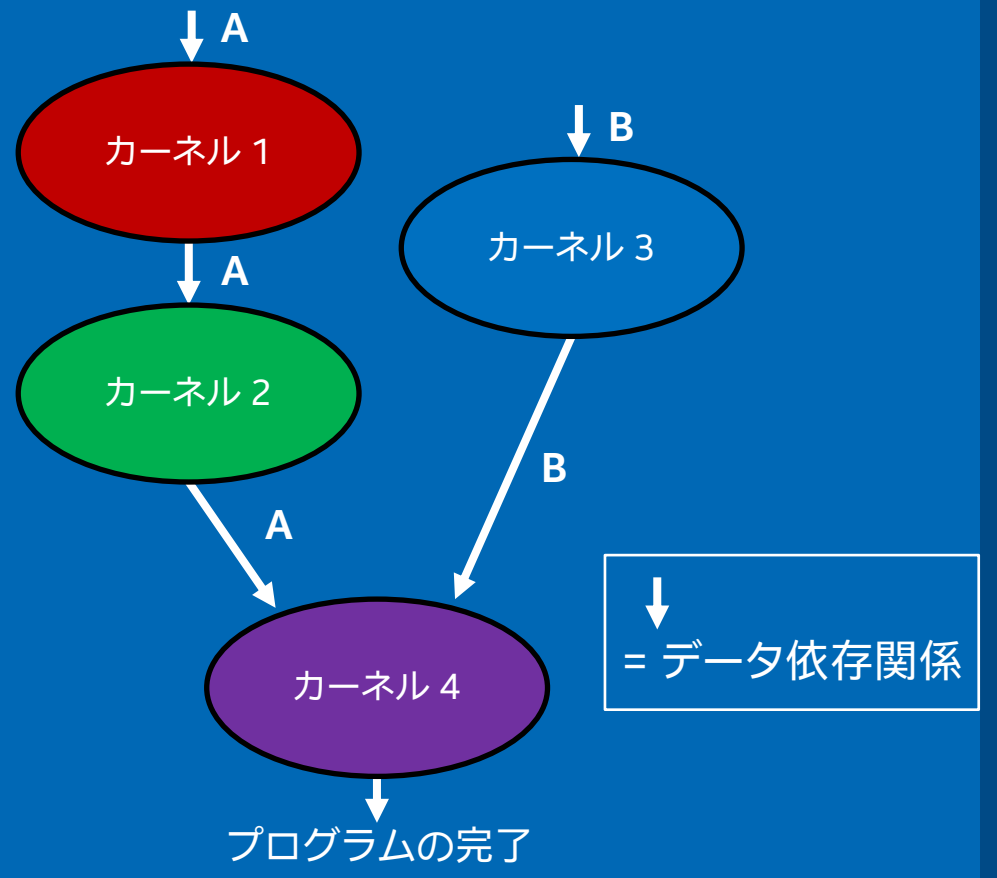
カーネル 1

カーネル 2

カーネル 3

カーネル 4

データと制御の依存関係を
自動的に解決!



インテル® DPC++ 互換性ツール (インテル® DPCT) の概要

インテル® DPC++ 互換性ツール (インテル® DPCT)

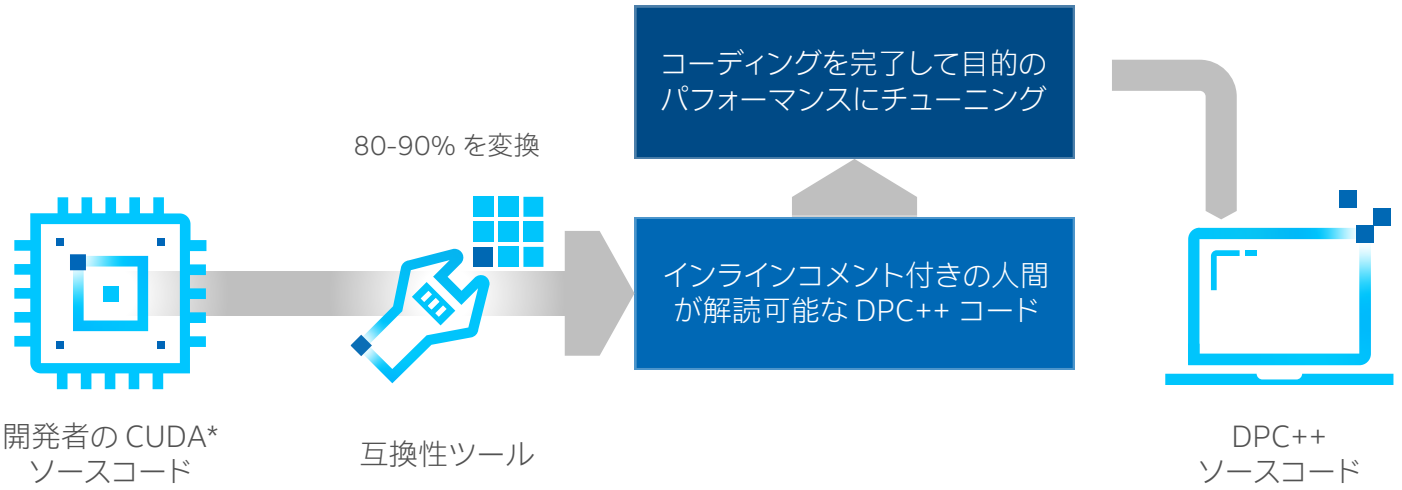
コードの移行時間を最小化

すでに CUDA* で記述されているコードを DPC++ に移行する開発者を支援し、可能な場合は**人間が解読可能な**コードを生成

通常はコードの 80-90% を自動的に移行

開発者がアプリケーションの移植を完了できるように支援するインラインコメントを提供

インテル® DPC++ 互換性ツールの使用フロー



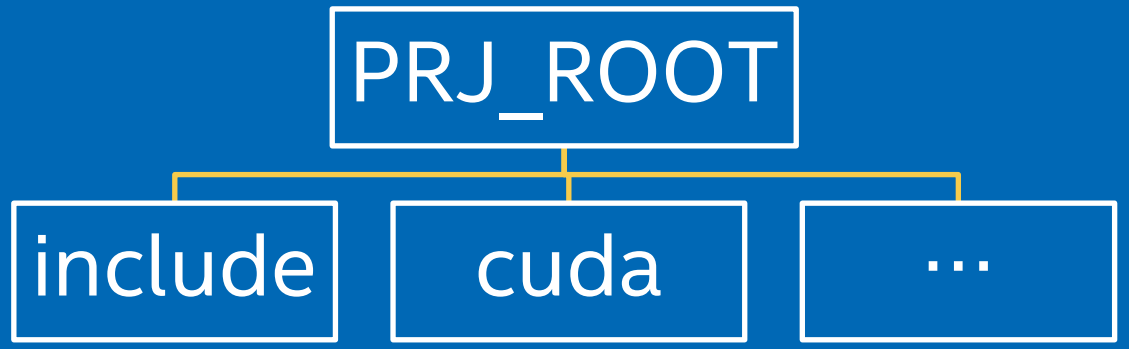
移行ワークフローの概要

- 移行前に CUDA* プロジェクトが nvcc でビルドできることを確認する
- インテル® DPCT は CUDA* ヘッダー (CUDA 8.0 ~ 11.1 でサポート) を探す
- 単純なプロジェクトでは移行するソースコードに対して直接 dpct コマンドを使用する
- 複雑なプロジェクトでは Visual Studio* プロジェクト・ファイルを移行するか、Make/Cmake でコンパイル・データベースをビルドして完全なプロジェクトを移行する
- 出力ファイルには移行できなかった残りのコードを移行するためのヒントやコメントが含まれる。コメントを確認して、移行されたコードが論理的に同等であることを確認する
- インテル® oneAPI DPC++/C++ コンパイラーでコンパイルする

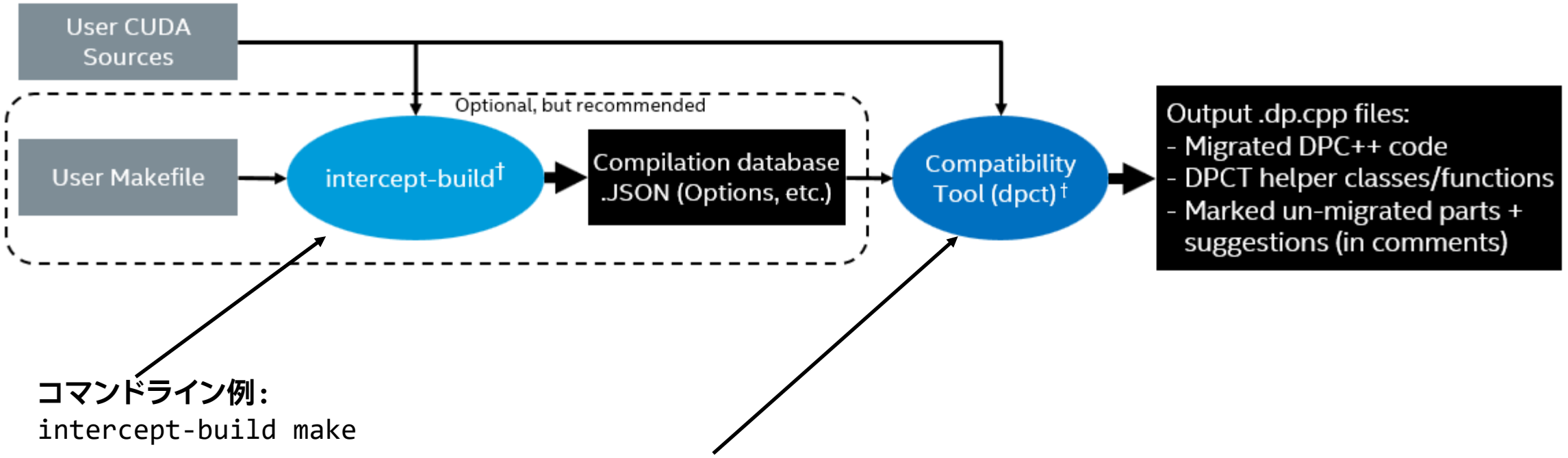
例: 単一ファイルの移行

インテル® DPCT で移行

```
cd PRJ_ROOT
dpct --in-root=./ --out-root=/path/to/output cuda/sample.cu
--extra-arg="-I./include" --keep-original-code --process-all
```



例: Makefile プロジェクトの移行データフロー



コマンドライン例:

```
intercept-build make
```

コマンドライン例:

```
dpct -p=<path to location of compilation database file> --in-root=. --out-root=migration
```

インテル® DPCT を使用した移行のベスト・プラクティス

基本的な役立つオプション

DPCT Basic Options	
--in-root	Path to the root of the source tree to be migrated
--out-root	Path to root of generated files.
-p	Path to compile database JSON file
--process-all	Migraters/copies all files from --in-root directory to the --out-root directory, eliminating need to specify .cu files one by one
--extra-arg	Specify more Clang compiler options. e.g. dpct --extra-arg="-std=c++14" –extra-arg="-I…"
--format-style	Sets formatting style for output files. e.g. =llvm, =google, =custom (Uses .clang-format file)
--format-range	Code formatting applied to no code (=none), migrated code (=migrated), or all code (=all)

インテル® DPCT を使用した移行のベスト・プラクティス

移行を容易にするオプション

DPCT Options that Ease Migration/Debug	
--keep-original-code	Keep original CUDA code in the comments of generated DPC++ file. Allows easy comparison of original CUDA code to generated DPC++ code.
--comments	Insert comments explaining the generated code
---always-use-async-handler	Always create cl::sycl::queue with the async exception handler

統合共有メモリーに関するオプション

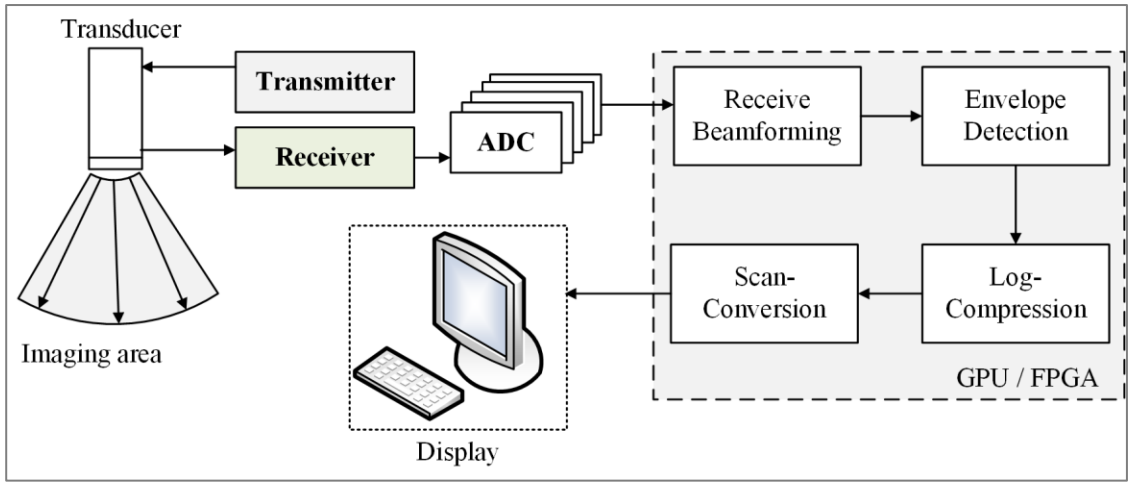
DPCT USM Option	
--usm-level	Sets Unified Shared Memory (USM) level. =Restricted: Use USM (default) =none: Uses helper functions and SYCL buffers

使用例:
oneAPI を使用した GPU および CPU 向け医療用画像アプリケーションの開発

背景

SUPRA とは?

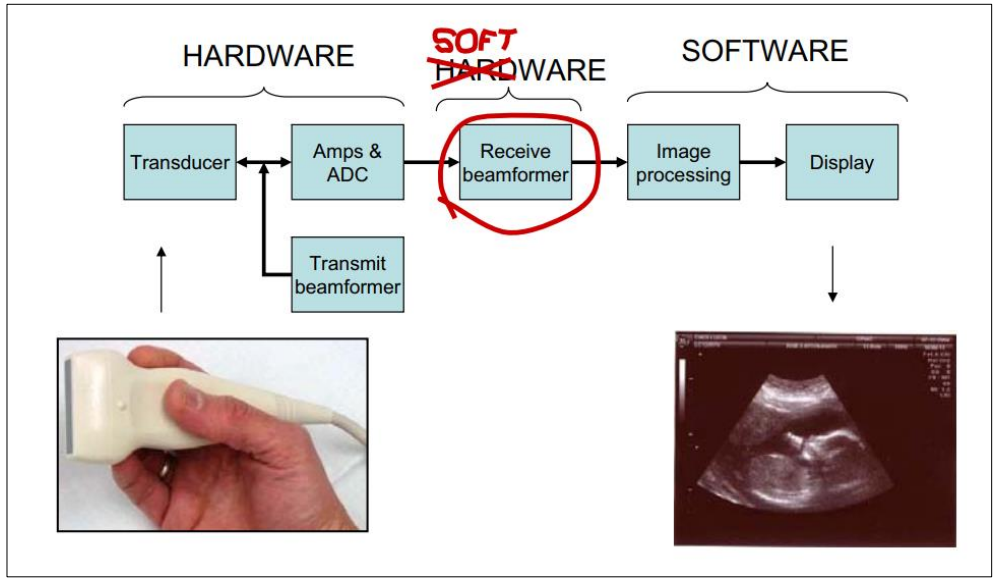
- SUPRA は完全にソフトウェアで定義されたオープンソースの超音波処理パイプライン
  - https://github.com/IFL-CAMP/supra (英語)



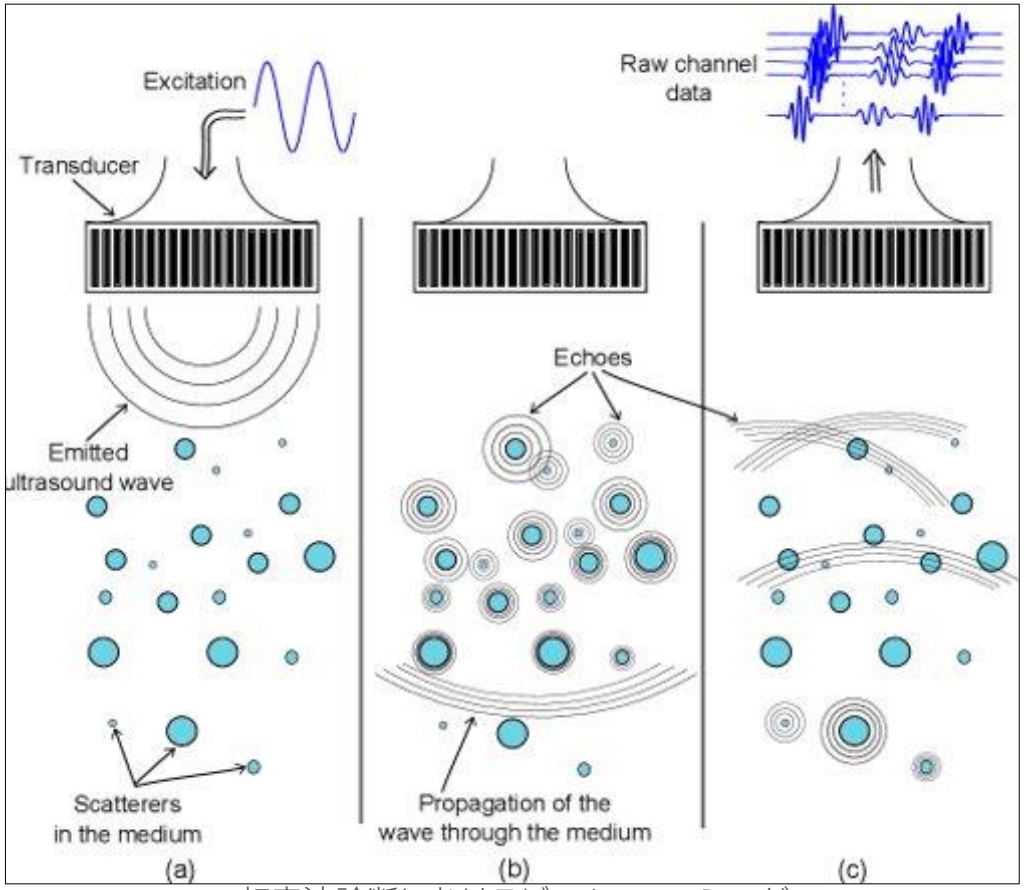
背景 (続き)

ソフトウェア・ビームフォーミングとは?

- SUPRA には標準的な超音波診断ソフトウェア・ビームフォーミング・アルゴリズムが搭載されている



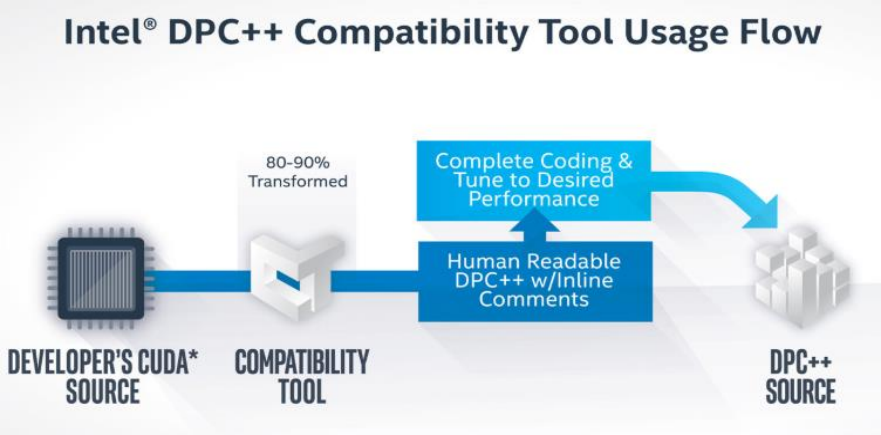
ソフトウェア・ビームフォーミングの図 (出典: Lars Grønvold)



超音波診断におけるビームフォーミング

コードの移行

移行ワークフロー



移行フロー

- インテル® DPC++ 互換性ツールは、既存の CUDA* コードを DPC++ コードに移行するのに役立つ
- DPC++ は、データ並列プログラミングを容易にするため SYCL* とコミュニティー拡張を取り入れた ISO C++
- DPC++ コードの記述と調整を容易にするインラインコメント

oneAPI バージョン	移行個所の合計数	移行済み	要変更	移行率
Golden	84	75	9	89%

SUPRA 移行サマリー

ファイルタイプ	*.cpp	*.cu	*.h
ファイル数	1	4	23

移行ファイル数

移行コマンド: `dpct --in-root=./ --out-root=./oneapi --extra-arg=-Isrc/SupraLib --extra-arg=-Isrc/SupraLib/Beamformer --extra-arg=-Isrc/SupraLib/utilities --extra-arg=-std=c++11 --extra-arg=-Wno-c++11-narrowing --extra-arg=-DHAVE_CUDA ./src/SupraLib/Beamformer/ScanConverter.cu ./src/SupraLib/Beamformer/HilbertFirEnvelope.cu ./src/SupraLib/Beamformer/LogCompressor.cu ./src/SupraLib/Beamformer/RxBeamformerCuda.cu ./src/SupraLib/ContainerFactory.cpp`

手動移行の例

CUDA*	`#ifdef HAVE_CUDA`
	`  cudaEvent_t m_creationEvent;`
	`#endif`

移行	`#ifdef HAVE_CUDA`
	`  sycl::event  m_creationEvent;`
	`  std::chrono::time_point<std::chrono::steady_clock> m_creationEvent_ct1;`
	`#endif`

変更	移行された sycl::event オブジェクトと std::chrono オブジェクトを削除する

CUDA*	`cudaSafeCall(cudaStreamCreateWithFlags(&(sm_streams[k]), cudaStreamNonBlocking));`
移行	`//cudaSafeCall(((sm_streams[k]) = dpct::get_current_device().create_queue()));`
変更	`sm_streams[k] = new sycl::queue(dpct::get_default_queue_wait().get_context(), dpct::get_default_queue_wait().get_device(), property_list);`

手動移行の例

```
thrustLogcompress<InputType, OutputType, WorkType> c(pow(10, (dynamicRange / 20)), static_cast<InputType>(inMax), outMax, scale);
thrust::transform(thrust::cuda::par.on(inImageData->getStream()), inImageData->get(), inImageData->get() + (width*height*depth),
    pComprGpu->get(), c);
cudaSafeCall(cudaPeekAtLastError());

return pComprGpu;
```

手動移行

```
thrustLogcompress<InputType, OutputType, WorkType> c(
    sycl::pow((float)10, (float)(dynamicRange / 20)), static_cast<InputType>(inMax), outMax, scale);

auto in_data = inImageData->get();
auto out_data = pComprGpu->get();
inImageData->getStream()->wait();

sycl::event e_log = inImageData->getStream()->submit([&](sycl::handler &cgh) {
    cgh.parallel_for<>(sycl::range<1>(width * height * depth), [=](sycl::id<1> idx) {
        out_data[idx] = c(in_data[idx]);
    });
});
```

移行の成功例

```
cudaSafeCall(cudaMalloc((void**)&buffer, numBytes));
```

```
cudaSafeCall((buffer = (uint8_t*)sycl::malloc_device(numBytes, dpct::get_current_device(), dpct::get_default_context()), 0));
```

```
cudaSafeCall(cudaMallocManaged((void**)&buffer, numBytes));
```

```
cudaSafeCall((buffer = (uint8_t*)sycl::malloc_shared(numBytes, dpct::get_current_device(), dpct::get_default_context()), 0));
```

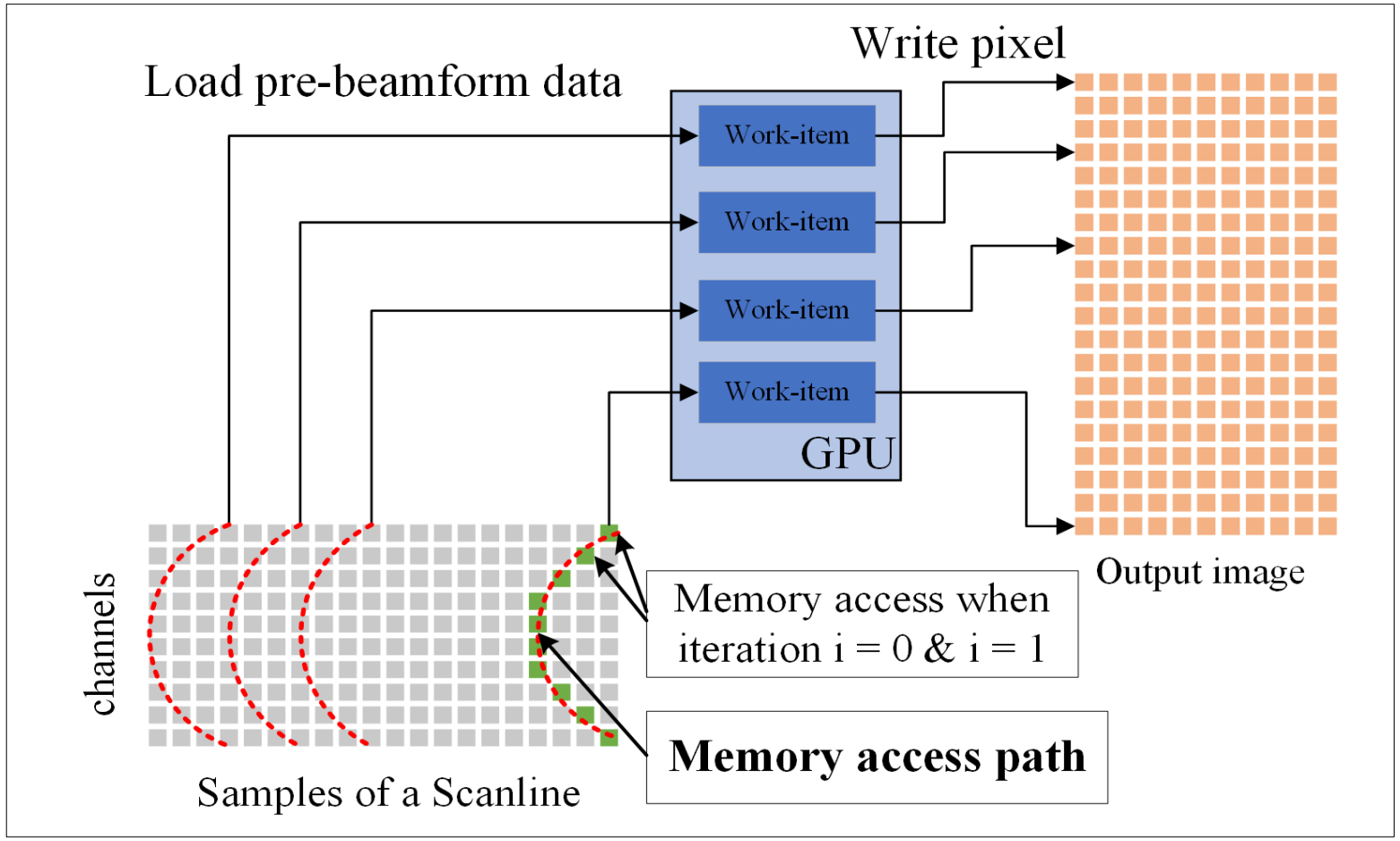
```
cudaSafeCall(cudaMallocHost((void**)&buffer, numBytes));
```

```
cudaSafeCall((buffer = (uint8_t*)sycl::malloc_host(numBytes, dpct::get_default_context()), 0));
```

メモリー割り当て関連関数の移行に成功

GPU 上でのビームフォーミング最適化

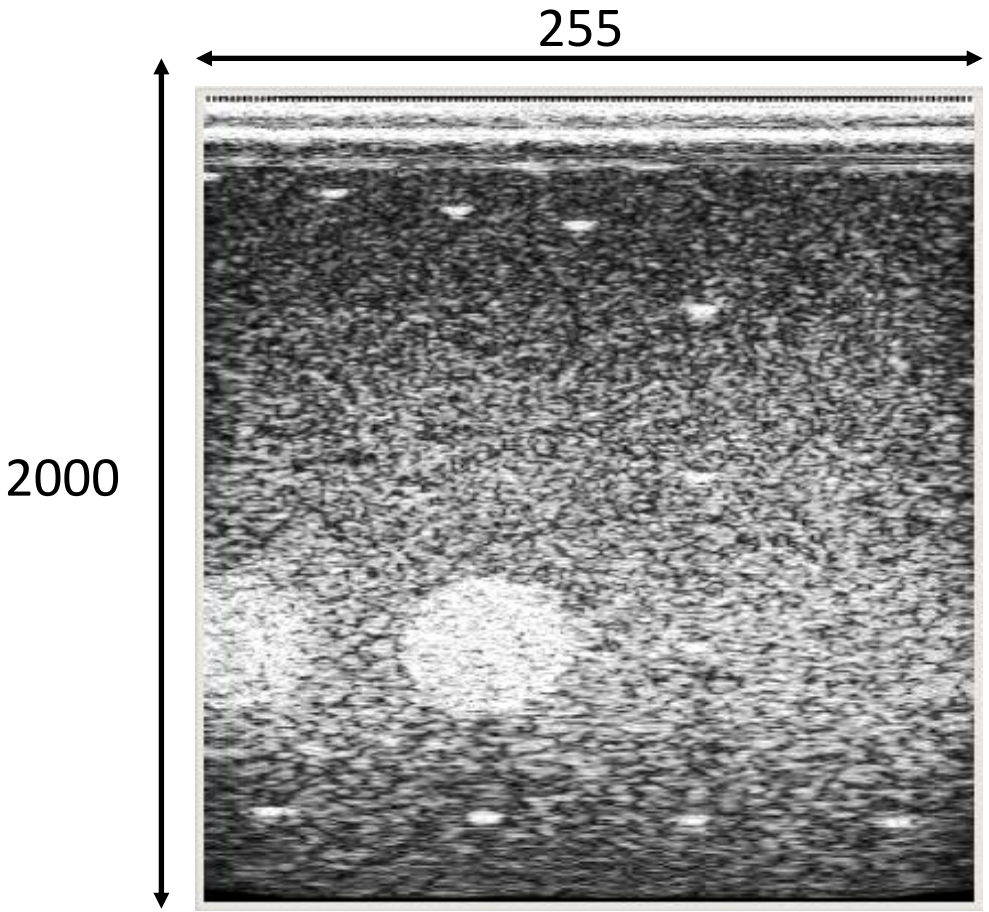
GPU 上でのビームフォーミングの並列実装



GPU 上でのビームフォーミングのオリジナル実装

最適化 #1

RxBeamformerCuda.dp.cpp ファイルと RxSampleBeamformerDelayAndSum.h ファイルの rxBeamformingDTSPACEKernel 関数と sampleBeamfor2D 関数を最適化



最適化の内容:

CUDA*: 各スレッドは 1 つのポイントを計算し、各ポイントは 64 回反復する

oneAPI: 各スレッドは垂直方向の 2 つのポイントをロードし、8 回反復する

最適化 #1

RxBeamformerCuda.cu では
関数が呼び出され、戻り値は **float**

```
#pragma unroll
for (int i = 0; i < row_size; i++) {
    LocationType invMaxElementDistance = 1 / sycl::min(aDT[i], maxElementDistance);
    sInterp[i] = SampleBeamformer::template vec_sampleBeamform2D<interpolateRFlines, RFType, float, LocationType>(txParams, RF, numTransducerElements,
    numReceivedChannels, numTimesteps, x_elemsDT, scanline_x, dirX, dirY, dirZ, aDT[i], d[i], invMaxElementDistance , speedOfSound, dt, additionalOffset,
    windowFunction, mdataGpu);
}
```

sampleBeamform2D 関数では
呼び出しごとに 1 つのポイントを
計算し、for ループは少なくとも
64 回反復する

```
template <bool interpolateRFlines, typename RFType, typename ResultType, typename LocationType>
static ResultType vec_sampleBeamform2D( ScanlineRxParameters3D::TransmitParameters txParams, const RFType* RF, uint32_t numTransducerElements, uint32_t numReceivedChannels,
    uint32_t numTimesteps, const LocationType* x_elemsDT, LocationType scanline_x, LocationType dirX, LocationType dirY, LocationType dirZ, LocationType aDT,
    LocationType depth, LocationType invMaxElementDistance, LocationType speedOfSound, LocationType dt, int32_t additionalOffset,
    const WindowFunctionGpu* __restrict__ windowFunction,const float* mdataGpu
)
{
    const int VEC_SIZE = 8;
    float sampleAcum = 0.0f;
    float weightAcum = 0.0f;
    int numAdds = 0;
    LocationType initialDelay = txParams.initialDelay;
    uint32_t txScanlineIdx = txParams.txScanlineIdx;

    for (int32_t elemIdxX = txParams.firstActiveElementIndex.x; elemIdxX < txParams.lastActiveElementIndex.x; elemIdxX += VEC_SIZE)
    {
        sycl::vec<int, VEC_SIZE> channelIdx;
        sycl::vec<LocationType, VEC_SIZE> x_elem;

        #pragma unroll
        for (int i = 0; i < VEC_SIZE; i +=2) {
            channelIdx[i] = (elemIdxX + i) % numReceivedChannels;
            channelIdx[i+1] = (elemIdxX + i + 1) % numReceivedChannels;
            x_elem[i] = x_elemsDT[elemIdxX + i];
            x_elem[i + 1] = x_elemsDT[elemIdxX + i + 1];
        }
        sycl::vec<float, VEC_SIZE> sample;
        sycl::vec<int, VEC_SIZE> mask = (sycl::fabs(x_elem - scanline_x) <= aDT);
        /*sycl spec1.2.1 mentioned: true return -1, false return 0*/
        mask *= -1;
        numAdds += utils<int, VEC_SIZE>::add_vec(mask);
```

sycl::vec<float, 8>

Use mask

ソースコード: supra/src/SupraLib/Beamformer/ RxSampleBeamformerDelayAndSum.h

最適化 #2

BeamformingNode のもう 1 つの最適化は、関数呼び出しを使用する代わりに、データをカーネル関数に直接移動させる

```
sycl::vec<float, VEC_SIZE> weight = windowFunction->get_vec((x_elem - scanline_x) * invMaxElementDistance);
```

```
inline sycl::vec<ElementType, VEC_SIZE> get_vec(sycl::vec<float, VEC_SIZE> relativeIndex) const
{
    sycl::vec<float,VEC_SIZE> relativeIndexClamped =
        sycl::min(sycl::max(relativeIndex, -1.0f), 1.0f);
    sycl::vec<float,VEC_SIZE> absoluteIndex =
        m_scale * (relativeIndexClamped + 1.0f);
    sycl::vec<int, VEC_SIZE> int_absoluteIndex = absoluteIndex.convert<int, sycl::rounding_mode::automatic>();

    sycl::vec<float, VEC_SIZE> v(0);
    #pragma unroll
    for(int i = 0 ; i < VEC_SIZE; i++) {
        int index = int_absoluteIndex[i];
        v[i] = m_data[index];
    }
    return v;
}
```

m_data は WindowFunctionGpu クラスのプライベート・メンバー

ソースコード: supra/src/SupraLib/Beamformer/WindowFunction.cpp

最適化する前に windowFcuntion->m_data からデータをフェッチする

最適化 #2

```
// use gRawData->getStream copy data to GPU
auto mdataGpu = std::make_shared<Container<float>>(ContainerLocation::LocationGpu, gRawData->getStream(), m_windowFunction->m_data);
```

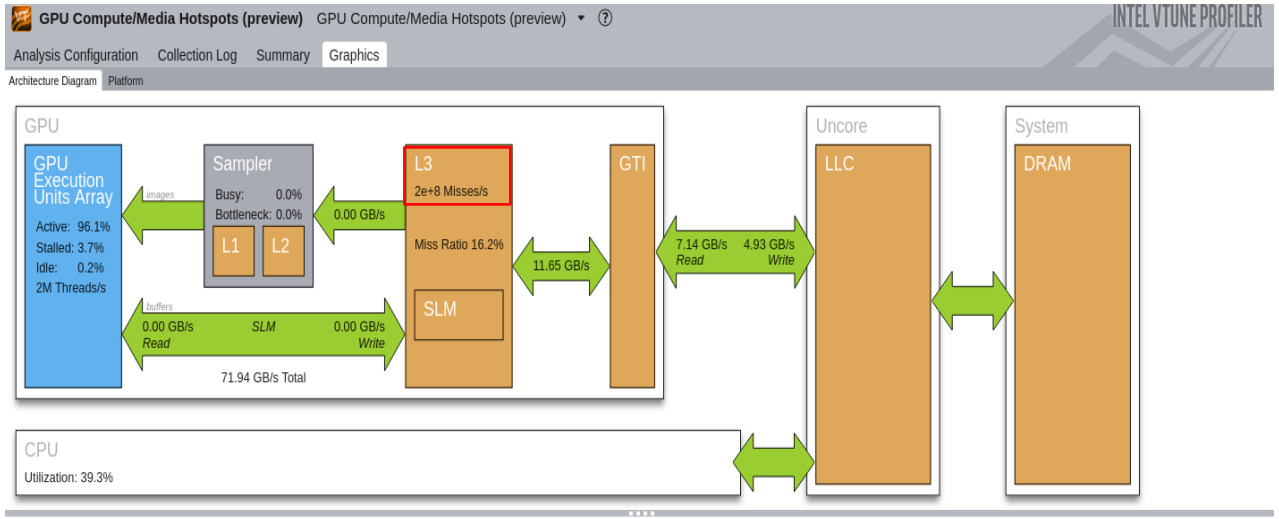
```
sycl::vec<float, VEC_SIZE> relativeIndex = (x_elem - scanline_x) * invMaxElementDistance;
sycl::vec<float, VEC_SIZE> relativeIndexClamped = sycl::min(sycl::max(relativeIndex, -1.0f), 1.0f);
sycl::vec<float, VEC_SIZE> absoluteIndex = windowFunction->m_scale * (relativeIndexClamped + 1.0f);
sycl::vec<int, VEC_SIZE> absoluteIndex_int = absoluteIndex.convert<int, sycl::rounding_mode::automatic>();
sycl::vec<float, VEC_SIZE> weight;
#pragma unroll
for (int i = 0; i < VEC_SIZE; i += 2 ) {
    weight[i] = mdataGpu[absoluteIndex_int[i]];
    weight[i + 1] = mdataGpu[absoluteIndex_int[i + 1]];
}

//weightAcum += weight;
//numAdds++;
weight *= mask.convert<float, sycl::rounding_mode::automatic>();
weightAcum += utils<float, VEC_SIZE>::add_vec(weight);
```

m_data は queue->submit() 呼び出しの前に mdataGpu にコピーされ、mdataGpu はカーネル関数に直接渡されるデータをコピーするため WindowFunctionGpu の m_data を public に変更する

ソースコード: supra/src/SupraLib/Beamformer/RxSampleBeamformerDelayAndSum.h
最適化後、mdataGpu からデータをフェッチする。mdataGpu はカーネル関数に直接渡される

最適化 #2

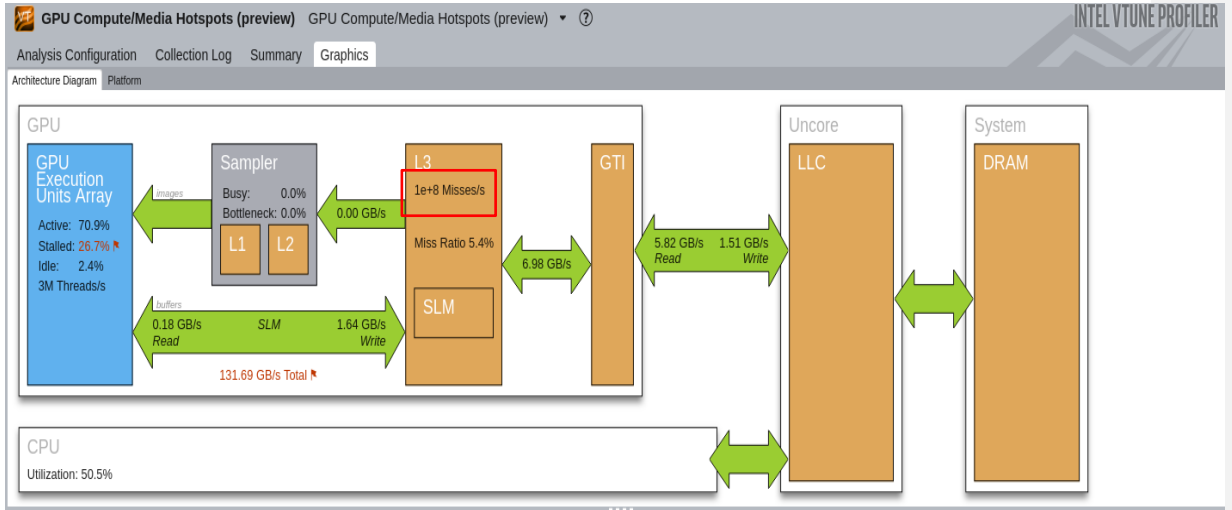


Grouping: Computing Task

Computing Task	Work Size		Computing Task					Data Transferred		EU Array	
	Global ▼	Local	Total Time	Average Time	Instance Count	SIMD Width	SVM Usage Type	Size	Total, GB/sec	Active	Stalled
_ZTSZZN5supra13LogCompressor8compressIffEESt1	510000		0.070s	0.000s	370	32		0 B	0.000	50.4%	25.9%
_ZTSZZN5supra18HilbertFirEnvelope10demodulateIfff	256 x 2000 x 1	16 x 8 x 1	0.855s	0.002s	370	16		0 B	0.000	66.7%	32.4%
_ZTSZZN5supra24rxBeamformingDTspaceCudaINS_2	255 x 2048 x 1	1 x 256 x 1	6.995s	0.019s	370	16		0 B	0.000	96.1%	3.7%
_ZTSZZN5supra13ScanConverter7convertIffEESt10sh	1680 x 2000 x 1	16 x 8 x 1	1.063s	0.003s	368	32		0 B	0.000	17.6%	81.5%
clEnqueueMapBuffer			0.025s	0.000s	370			179 MB	7.485	0.0%	0.0%
[Outside any task]			0s							0.4%	1.6%

最適化前

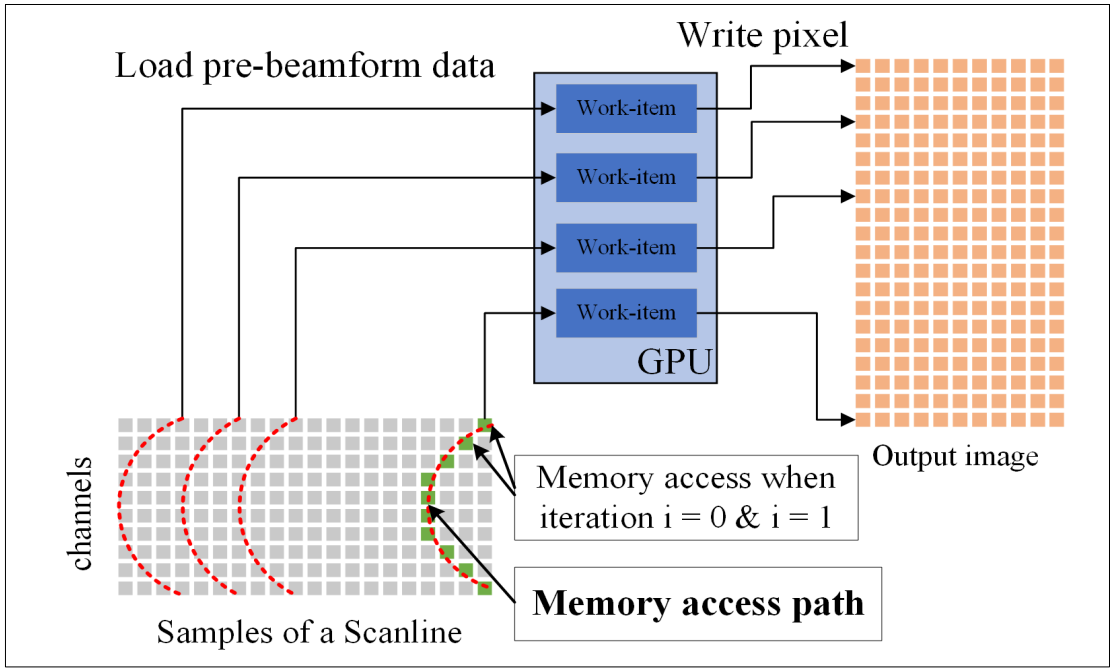
最適化後



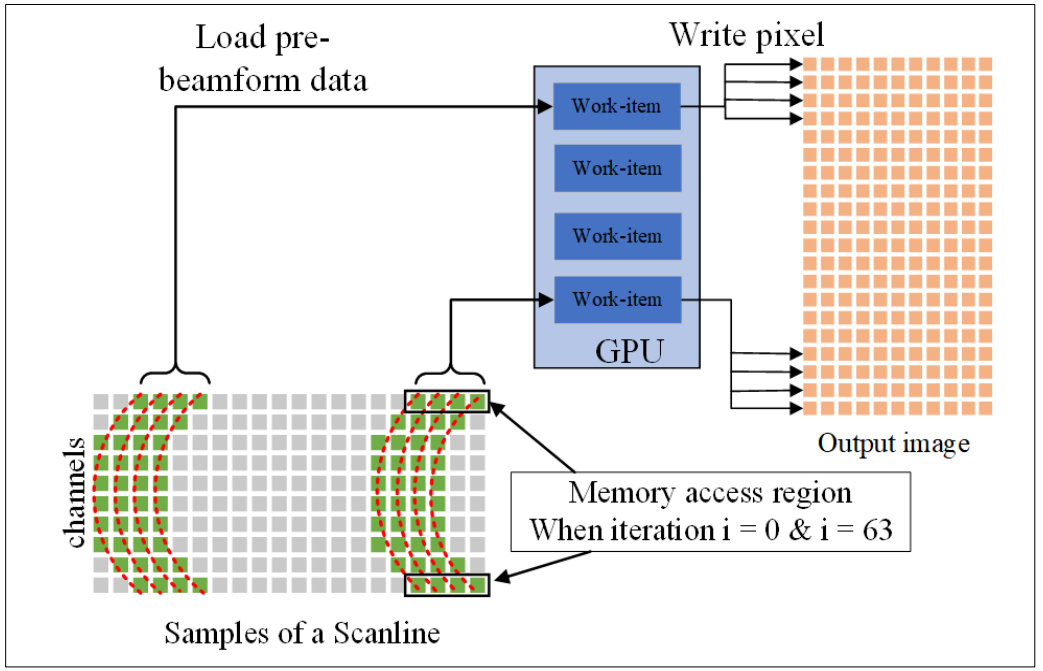
Grouping: Computing Task

Computing Task	Work Size		Computing Task					Data Transferred		EU Array	
	Global ▼	Local	Total Time	Average Time	Instance Count	SIMD Width	SVM Usage Type	Size	Total, GB/sec	Active	Stalled
_ZTSZZN5supra13LogCompressor8compressIffEESt1	510000		0.055s	0.000s	249	32		0 B	0.000	53.3%	26.0%
_ZTSZZN5supra18HilbertFirEnvelope10demodulateIfff	256 x 2000 x 1	16 x 8 x 1	0.702s	0.003s	249	16		0 B	0.000	68.4%	30.6%
_ZTSZZN5supra24rxBeamformingDTspaceCudaINS_2	255 x 1024 x 1	1 x 256 x 1	2.767s	0.011s	250	8		0 B	0.000	70.9%	26.7%
_ZTSZZN5supra13ScanConverter7convertIffEESt10sh	1680 x 2000 x 1	16 x 8 x 1	0.717s	0.003s	248	32		0 B	0.000	19.2%	79.8%
clEnqueueMapBuffer			0.017s	0.000s	250			121 MB	7.540	0.0%	0.0%
[Outside any task]			0s							0.4%	1.5%

ESIMD を使用したビームフォーミングの最適化



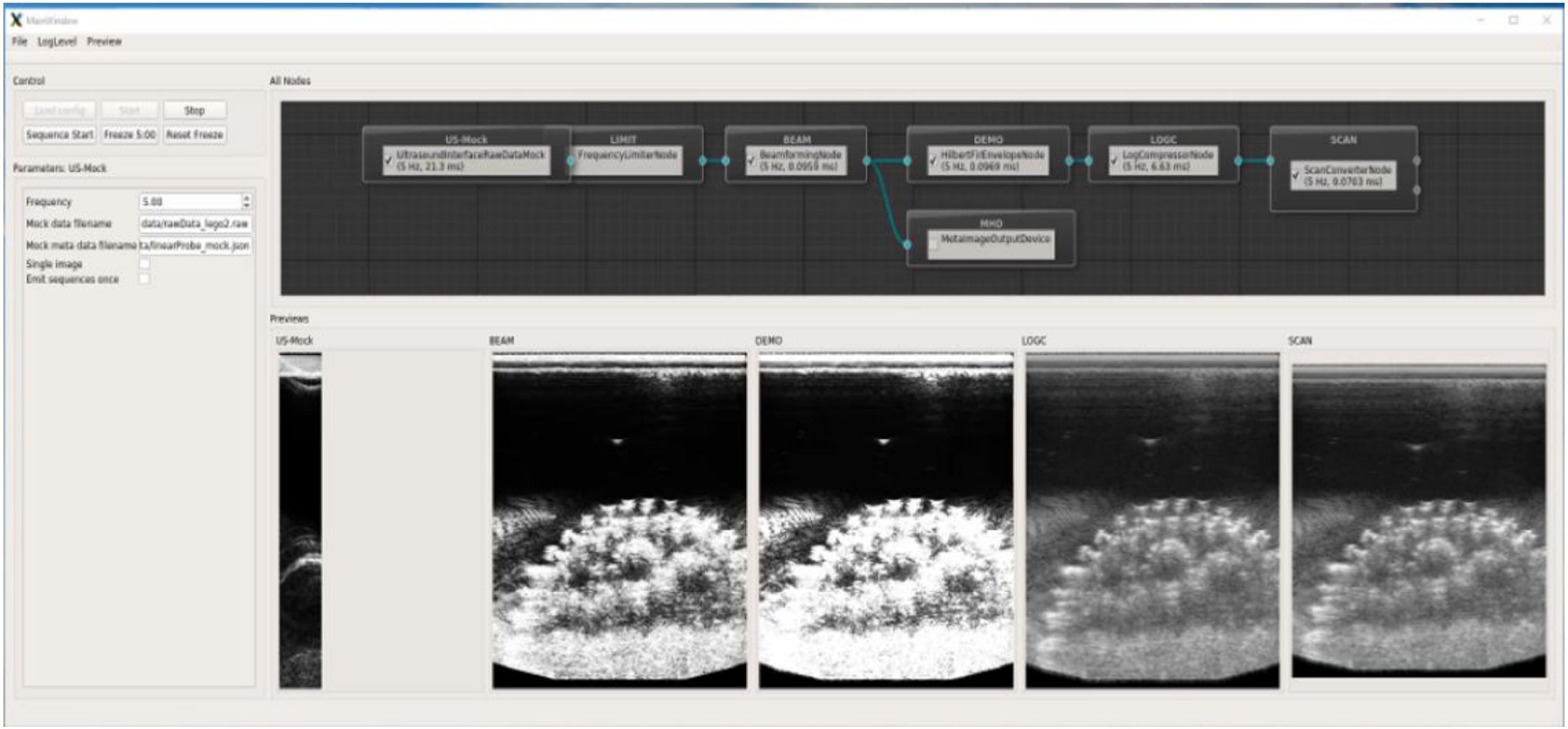
GPU 上でのビームフォーミングのオリジナル実装



GPU 上でのビームフォーミングの最適化された実装

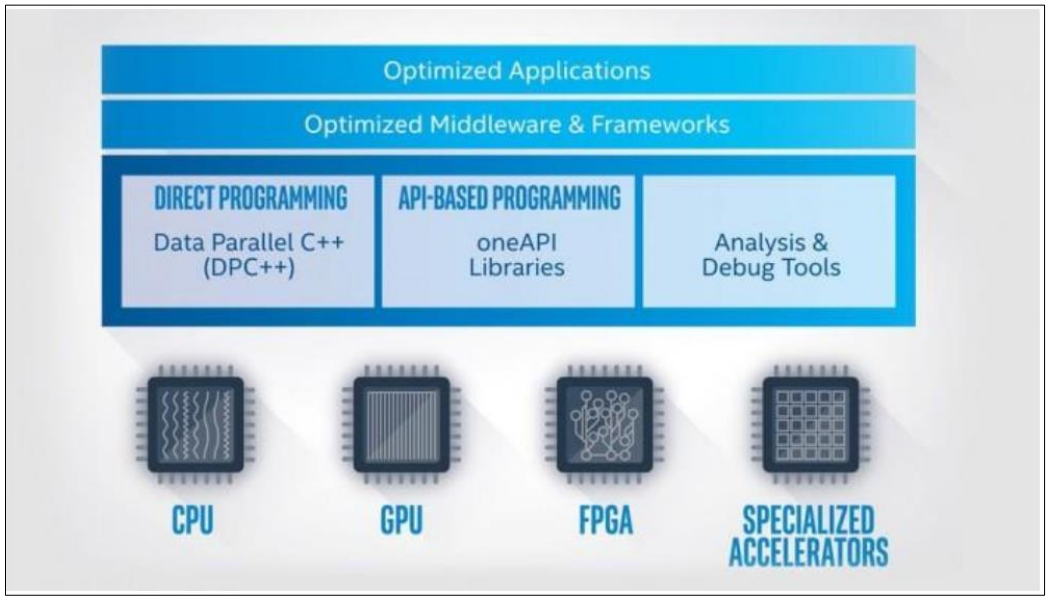
結果

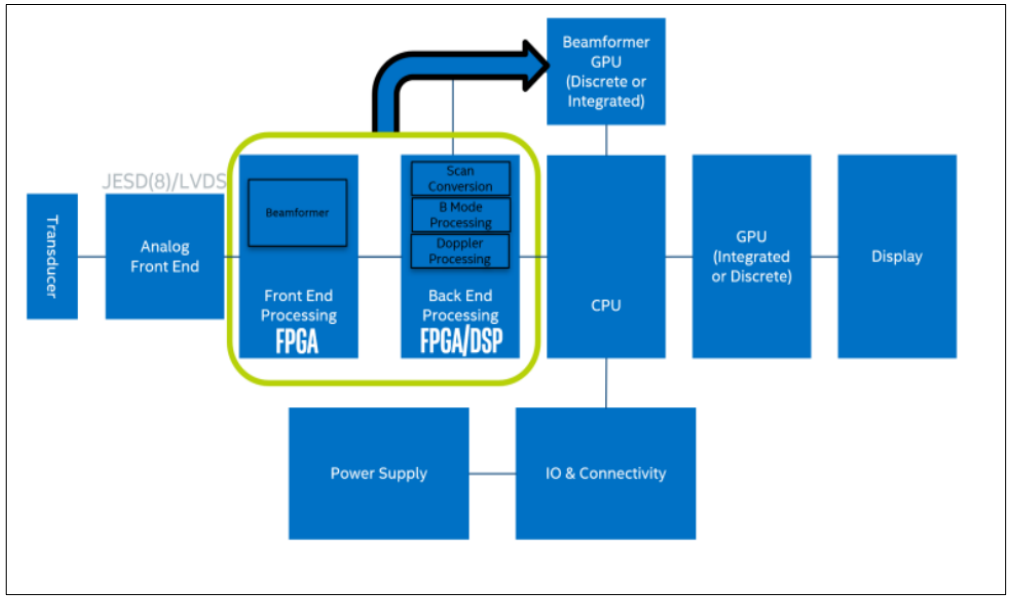
SUPRA GUI


File LogLevel Preview
Control
Stop
Sequence Start
Freeze 5.00
Reset Freeze
Parameters: US-Mock
Frequency
5.00
Mock data filename
data/rawData_lago2.raw
Single image
Emit sequences once
All Nodes
US-Mock
UltrasoundInterfaceRawDataMock
(5 Hz, 21.3 ms)
LIMIT
FrequencyLimiterNode
BEAM
BeamformingNode
(5 Hz, 0.0959 ms)
DEMO
HilbertFirEnvelopeNode
(5 Hz, 0.0969 ms)
LOGC
LogCompressorNode
(5 Hz, 6.63 ms)
SCAN
ScanConverterNode
(5 Hz, 0.0703 ms)
MHD
MetaImageOutputDevice
Previews
US-Mock
BEAM
DEMO
LOGC
SCAN

結果

- インテル・ハードウェア上で医療アルゴリズムの高速化を実装する統一されたプログラミング・フレームワーク/言語
- インテル xGPU 上で超音波ビームフォーミングを実装するサンプル
- ヘテロジニアス計算システム上でアルゴリズムの高速化と AI 推論を統合可能 (oneAPI と OpenVINO™ ツールキット)
- 将来のインテル・アクセラレーション・ハードウェア (XPU) をサポート





まとめ

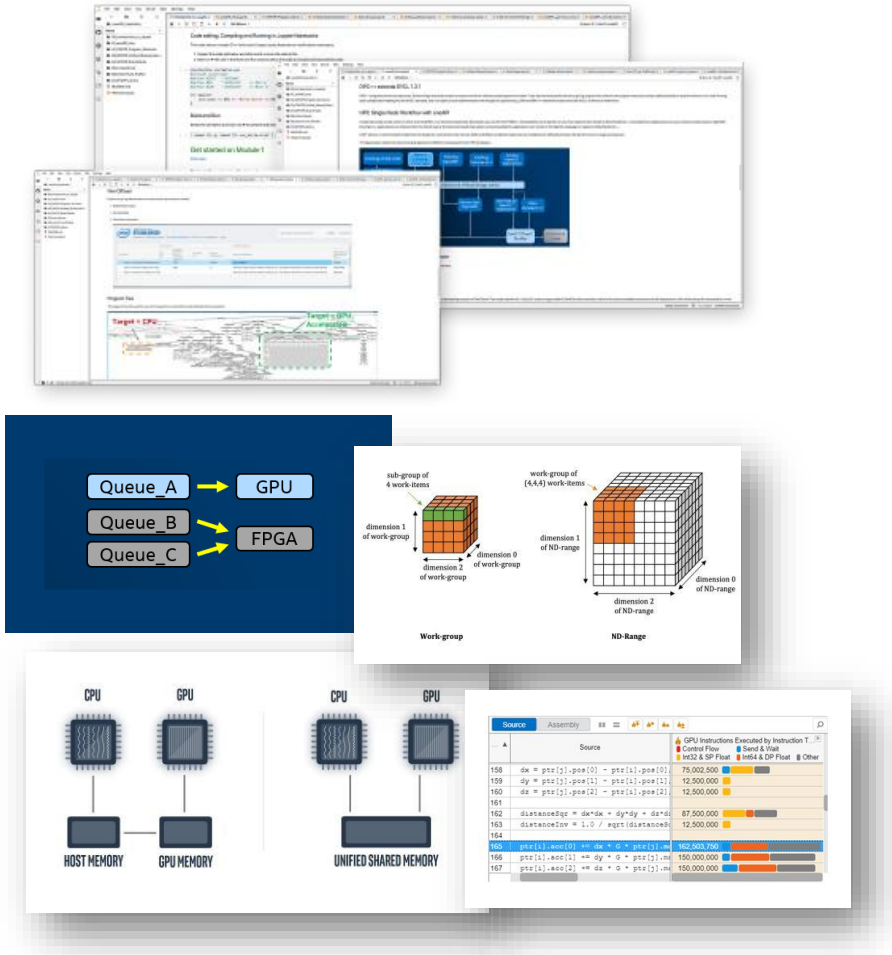
まとめ

- **oneAPI** 業界イニシアチブ - 標準ベースのオープン仕様
- CPU、GPU、FPGA、AI アクセラレーターを含む、さまざまなハードウェアでネイティブコードのパフォーマンスを実現
- **インテル® DPC++ 互換性ツール**は、すでに CUDA* で記述されているコードを DPC++ に移行する開発者を支援し、可能な場合は人間が解読可能なコードを生成

関連情報

- **インテル® oneAPI ベース & IoT ツールキット**をお試しください!
  - xlsoft.com/jp/products/intel/oneapi/index.html#features-iot に今すぐアクセス
- **oneAPI 仕様** - 業界を超えたオープンな標準ベースの統合プログラミング・モデル – 詳細 (英語)
- **データ並列 C++ の要点** - データ並列処理およびヘテロジニアス計算向けに設計されたデータ並列 C++ 言語の基本情報 – 詳細 (英語)
- コンパイラー・ポーティング・ガイド – 詳細

データ並列 C++ (DPC++) に必要なもの



DPC++ の習得

インテル® DevCloud 上の Jupyter* Notebook でサンプルコードを使った実践的な演習を利用可能

DPC++ 開発の学習モジュール

- DPC++ プログラム構造
- DPC++ 統合共有メモリー
- DPC++ サブグループ
- インテル® Advisor のデモ
- インテル® VTune™ プロファイラー

software.intel.com/content/www/us/en/develop/tools/oneapi/training/dpc-essentials.html (英語)

法務上の注意書きと最適化に関する注意事項

インテルのテクノロジーを使用するには、対応したハードウェア、ソフトウェア、またはサービスの有効化が必要となる場合があります。詳細については、OEM または販売店にお問い合わせいただくか、http://www.intel.co.jp/ を参照してください。

実際の費用と結果は異なる場合があります。

インテルは、サードパーティーのデータについて管理や監査を行っていません。ほかの情報も参考にして、正確かどうかを評価してください。

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
https://software.intel.com/en-us/articles/optimization-notice#opt-jp

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。構成の詳細は、補足資料を参照してください。性能やベンチマーク結果について、さらに詳しい情報をお知りになりたい場合は、http://www.intel.com/benchmarks (英語) を参照してください。

性能の測定結果はシステム構成の日付時点のテストに基づいています。また、現在公開中のすべてのセキュリティー・アップデートが適用されているとは限りません。詳細については、公開されている構成情報を参照してください。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。

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。

© Intel Corporation. Intel、インテル、Intel ロゴ、その他のインテルの名称やロゴは、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。

デモの補足資料 – CUDA* コード

```
//==============================================================
// Copyright © 2019 Intel Corporation
//
// SPDX-License-Identifier: MIT
// =============================================================

#include <cuda.h>
#include <stdio.h>
#include <stdlib.h>
#define VECTOR_SIZE 256

__global__ void VectorAddKernel(float* A, float* B, float* C)
{
    A[threadIdx.x] = threadIdx.x + 1.0f;
    B[threadIdx.x] = threadIdx.x + 1.0f;
    C[threadIdx.x] = A[threadIdx.x] + B[threadIdx.x];
}

int main()
{
    float *d_A, *d_B, *d_C;
    cudaError_t status;

    cudaMalloc(&d_A, VECTOR_SIZE*sizeof(float));
    cudaMalloc(&d_B, VECTOR_SIZE*sizeof(float));
    cudaMalloc(&d_C, VECTOR_SIZE*sizeof(float));

    VectorAddKernel<<<1, VECTOR_SIZE>>>(d_A, d_B, d_C);

    float Result[VECTOR_SIZE] = { };

    status = cudaMemcpy(Result, d_C, VECTOR_SIZE*sizeof(float), cudaMemcpyDeviceToHost);
    if (status != cudaSuccess) {
        printf("Could not copy result to host\n");
        exit(EXIT_FAILURE);
    }

    cudaFree(d_A);
    cudaFree(d_B);
    cudaFree(d_C);

    for (int i = 0; i < VECTOR_SIZE; i++) {
        if (i % 16 == 0) {
            printf("\n");
        }
        printf("%3.0f ", Result[i]);
```

デモの補足資料 – インテル® DPCT で移行後のコード

vector_add.dp.cpp

vector-add_dpcpp (Global Scope)

```
1   //==============================================================
2   // Copyright © 2019 Intel Corporation
3   //
4   // SPDX-License-Identifier: MIT
5   // =============================================================
6
7    /* DPCT_ORIG #include <cuda.h>*/
8   #include <CL/sycl.hpp>
9    #include <dpct/dpct.hpp>
10   #include <stdio.h>
11   #include <stdlib.h>
12   #define VECTOR_SIZE 256
13
14   /* DPCT_ORIG __global__ void VectorAddKernel(float* A, float* B, float* C)*/
15  void VectorAddKernel(float *A, float *B, float *C, sycl::nd_item<3> item_ct1)
16   {
17   /* DPCT_ORIG     A[threadIdx.x] = threadIdx.x + 1.0f;*/
18      A[item_ct1.get_local_id(2)] = item_ct1.get_local_id(2) + 1.0f;
19   /* DPCT_ORIG     B[threadIdx.x] = threadIdx.x + 1.0f;*/
20      B[item_ct1.get_local_id(2)] = item_ct1.get_local_id(2) + 1.0f;
21   /* DPCT_ORIG     C[threadIdx.x] = A[threadIdx.x] + B[threadIdx.x];*/
22      C[item_ct1.get_local_id(2)] =
23          A[item_ct1.get_local_id(2)] + B[item_ct1.get_local_id(2)];
24   }
25
26  int main() try {
27      dpct::device_ext &dev_ct1 = dpct::get_current_device();
28      sycl::queue &q_ct1 = dev_ct1.default_queue();
29      float *d_A, *d_B, *d_C;
30   /* DPCT_ORIG     cudaError_t status;*/
31      int status;
32
33   /* DPCT_ORIG     cudaMalloc(&d_A, VECTOR_SIZE*sizeof(float));*/
34      d_A = sycl::malloc_device<float>(VECTOR_SIZE, q_ct1);
35   /* DPCT_ORIG     cudaMalloc(&d_B, VECTOR_SIZE*sizeof(float));*/
36      d_B = sycl::malloc_device<float>(VECTOR_SIZE, q_ct1);
37   /* DPCT_ORIG     cudaMalloc(&d_C, VECTOR_SIZE*sizeof(float));*/
38      d_C = sycl::malloc_device<float>(VECTOR_SIZE, q_ct1);
39
40   /* DPCT_ORIG     VectorAddKernel<<<1, VECTOR_SIZE>>>(d_A, d_B, d_C);*/
41      q_ct1.submit([&](sycl::handler &cgh) {
42          cgh.parallel_for(sycl::nd_range<3>(sycl::range<3>(1, 1, VECTOR_SIZE),
43                                             sycl::range<3>(1, 1, VECTOR_SIZE)),
44                           [=](sycl::nd_item<3> item_ct1) {
45                               VectorAddKernel(d_A, d_B, d_C, item_ct1);
```

```
49      float Result[VECTOR_SIZE] = { };
50
51  /* DPCT_ORIG     status = cudaMemcpy(Result, d_C, VECTOR_SIZE*sizeof(float),
52   * cudaMemcpyDeviceToHost);*/
53      /*
54      DPCT1003:0: Migrated API does not return error code. (*, 0) is inserted. You
55      may need to rewrite this code.
56      */
57      status = (q_ct1.memcpy(Result, d_C, VECTOR_SIZE * sizeof(float)).wait(), 0);
58  /* DPCT_ORIG     if (status != cudaSuccess) {
59          printf("Could not copy result to host\n");
60          exit(EXIT_FAILURE);
61      }*/
62
63   /* DPCT_ORIG     cudaFree(d_A);*/
64      sycl::free(d_A, q_ct1);
65   /* DPCT_ORIG     cudaFree(d_B);*/
66      sycl::free(d_B, q_ct1);
67   /* DPCT_ORIG     cudaFree(d_C);*/
68      sycl::free(d_C, q_ct1);
69
70      for (int i = 0; i < VECTOR_SIZE; i++) {
71          if (i % 16 == 0) {
72              printf("\n");
73          }
74          printf("%3.0f ", Result[i]);
75      }
76      printf("\n");
77
78      return 0;
79  }
80  catch (sycl::exception const &exc) {
81    std::cerr << exc.what() << "Exception caught at file:" << __FILE__
82              << ", line:" << __LINE__ << std::endl;
83    std::exit(1);
84  }
85
```

インテル® DPCT の出力