インテル® コンパイラーを使用した OpenMP* による並列プログラミング

インテル® C++/Fortran コンパイラーのバージョン 19.1 を使用して GPU オフロードに備えましょう

セッション 6: oneAPI への第一歩

IA Software User Society (iSUS)
編集長　すがわら　きよふみ

このセッションの目的

明示的な並列プログラミング手法として注目されてきた OpenMP* による並列プログラミングに加え、インテル® コンパイラーがサポートする OpenMP* 4.0 と 4.5 の機能を使用したベクトル・プログラミングとオフロード・プログラミングの概要をリフレッシュし、インテル® コンパイラー V19.1 でサポートされる OpenMP* 5.0 の機能と実装を紹介します。さらに新たなアクセラレーター・デバイスへのオフロードについて考えます

セッションの対象者

すでに OpenMP* でマルチスレッド・プログラミングを開発し、4.0 以降でサポートされる新たなベクトル化とオフロードを導入し、アプリケーションのパフォーマンス向上を計画する開発者

セッションリスト

セッション	説明
はじめに (13:30 – 14:30)	異なるバージョンのインテル® コンパイラーや異なるコンパイラー間で OpenMP* を使用する注意点や制限について説明します
OpenMP* のタスク機能 (14:30 – 15:30)	OpenMP* 3.1 で追加されたタスク機能が 4.0 から 4.5 でどのように進化したかを例を使用して説明し、最新の OpenMP* 5.0 で強化された新機能を紹介します
OpenMP* の SIMD 機能 (13:40 – 14:30)	OpenMP* のスレッド化機能を使用してプログラマーがマルチスレッドの動作をプログラミングしたように、OpenMP* 4.0 からは omp simd を使用してプログラマーが明示的にベクトル化もできるようになりました。OpenMP* simd に関連する機能を 4.0 から 5.0 までの進化を追って紹介します
OpenMP* のオフロード機能 (14:30 – 15:30)	OpenMP* 4.0 で追加されたオフロード機能を利用することで、これまで共有メモリー型並列処理に加え分散メモリー型の並列処理を表現できるようになりました。このセッションでは、注目されるヘテロジニアス・プログラミング環境での OpenMP* オフロード機能について説明します
OpenMP* 5.0 の注目する機能 (13:30 – 14:15)	セッション2、3、4 でカバーされなかった OpenMP* 5.0 のそのほかの機能について紹介します
インテル® C++/Fortran コンパイラーのバージョン 19.1 を使用して GPU オフロードに備えましょう (14:15 – 15:30)	oneAPI 向けのデータ並列 C++ (DPC++) へ移行する前に、現行のインテル® C++/Fortran コンパイラー V19.1 やインテル® oneAPI HPC ツールキットに含まれるベータ版インテル® C++/Fortran コンパイラー 2021 を使用して簡単にインテル® グラフィックスへのオフロードを行うソフトウェアを開発および検証方法を紹介します

内容

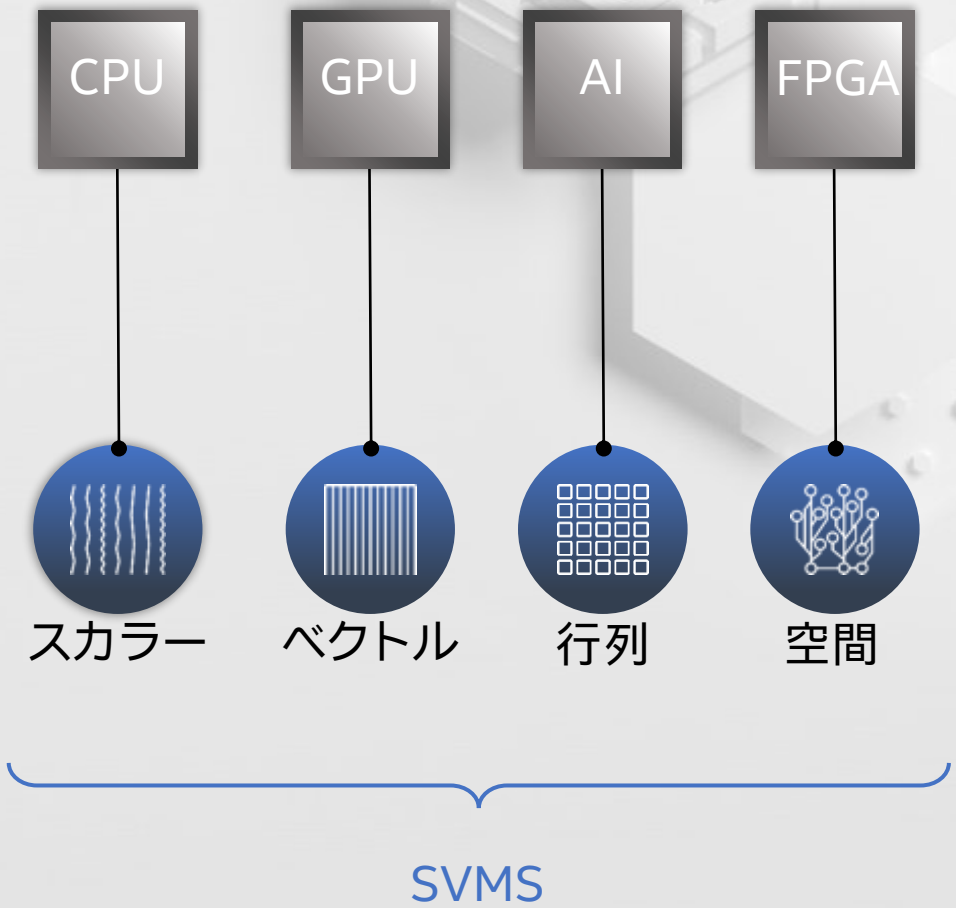
- はじめに (OpenMP* が必要とされる背景) と概要 (OpenMP* とは、歴史、各バージョンの機能概要)
- OpenMP* の各バージョンの機能 (4.0、4.5 および 5.0 の注目される新機能)
- 次世代インテル® コンパイラー (nextgen) の機能

OpenMP* 5.0 API シンタックス・クイック・リファレンス・カードの日本語訳を公開しました:

https://www.isus.jp/products/c-compilers/openmp-ref-5-0-0519-released/

プログラミングの課題

- さまざまなデータ構造を使用するハードウェアのセットが存在します
- 残念ながら、それらに対する共通のプログラミング言語や API はありません
- また、それぞれのプラットフォームにおいてツールのサポートに一貫性もありません
- プラットフォームごとに個別のソフトウェア投資/習得が必要です



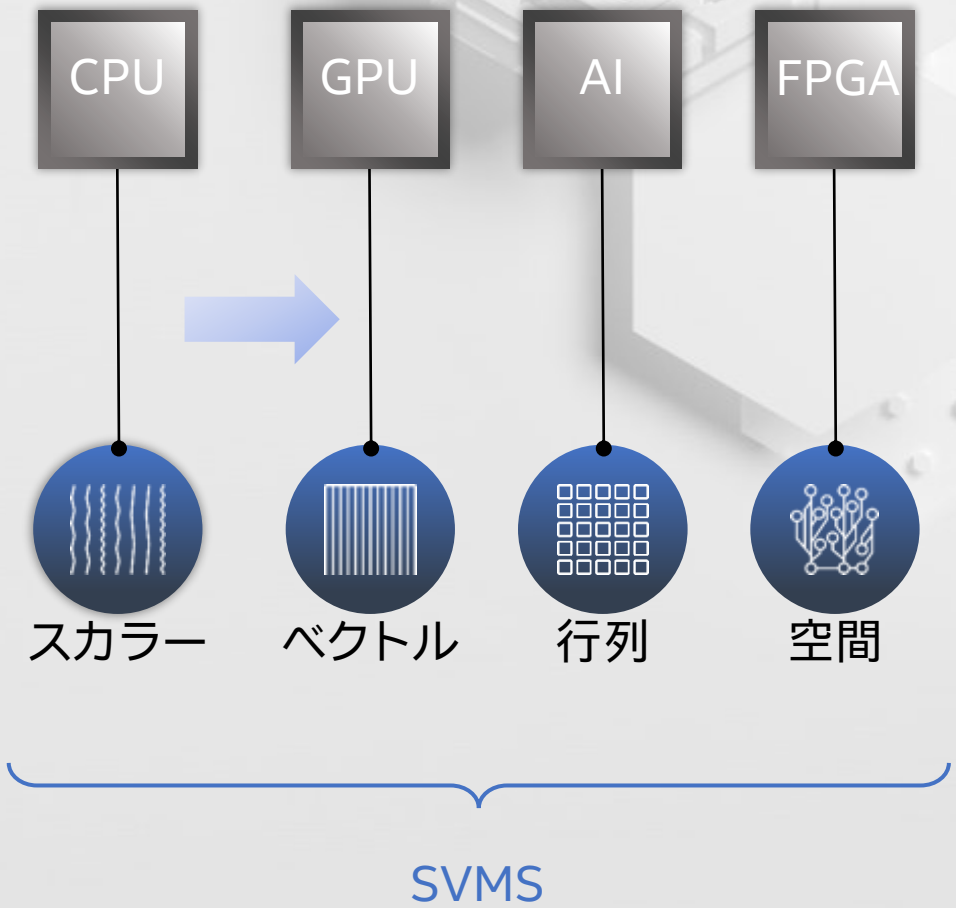
プログラミングの挑戦

```
#pragma omp target data device(0) map(alloc:tmp[:N])
  map(to:input[:N]) map(from:res)
{
#pragma omp target device(0)
#pragma omp parallel for
    for (i=0; i<N; i++)
       tmp[i] = some_computation(input[i], i);

update_input_array_on_the_host(input);

#pragma omp target update device(0) to(input[:N])
#pragma omp target device(0)
#pragma omp parallel for reduction(+:res)
    for (i=0; i<N; i++)
       res += final_computation(input[i], tmp[i], i)
}
```

OpenMP*
OpenCL*
OpenACC*
…



準備するもの (ここでは Windows* を使用)

コンパイラー:

- インテル® C++ または Fortran コンパイラー・バージョン 19.1 Update2

  または

- インテル® oneAPI ベース・ツールキット beta08 以降
- インテル® oneAPI HPC ツールキット beta08 以降

グラフィックス・ドライバー:

- インテル® oneAPI ベース・ツールキット用グラフィックス・ドライバー

  https://software.intel.com/content/www/us/en/develop/articles/intel-oneapi-graphics-driver-now-available.html (英語)

次世代インテル® コンパイラーとは

- ICC Classic と ICC NextGen (icx) は、各コンパイラー・ドライバー (icc、icpc、icl) でも同じ呼び方をします
- ICC NextGen は、コア C++ コンパイラーであるとともに、インテル® DPC++ コンパイラーと新しい dpcpp ドライバーのベース・コンパイラーでもあります
- ICC NextGen は、インテルの「次世代」コンパイラーです。ICC NextGen は ICC Classic の単なる代替品ではないことに注意してください
- ICC Classic から ICC NextGen への移行を可能な限りスムーズにするため、かなり工夫されていますが、開発者が既存のアプリケーションを ICC Classic から ICC NextGen へ移行してチューニングするには、ある程度の労力が必要になります

インテル® C++ コンパイラーの次世代コード・ジェネレーターのドキュメント:
https://www.isus.jp/products/c-compilers/early-documentation-for-c-compiler-based-on-llvm/

この資料では Nextgen コンパイラーの OpenMP* 機能に注目します

ICC NextGen (icx) の OpenMP* オプション:

- /Qiopenmp / -fiopenmp は、OpenMP* parallel 構造と SIMD プラグマ/ディレクティブを認識してコンパイルし、インテルの OpenMP* ランタイム・ライブラリーを使用します
- OpenMP* 4.5/5.0 の target プラグマ/ディレクティブを使用する場合、/Qopenmp-targets=spir64 / -fopenmp-targets=spir64 を指定する必要があります
- OpenMP* 4.5/5.0 の target ディレクティブは、インテル® oneAPI HPC ツールキットに含まれる ICC NextGen コンパイラーによってのみ認識されます
  上記の 2 つのコンパイラー・オプションを同時に指定します:
  - icc -qnextgen -fiopenmp -fopenmp-targets=spir64
  - icl /Qnextgen /Qiopenmp /Qopenmp-targets=spir64

インテル® コンパイラー V19.1 の Nextgen 機能はプレビュー機能です

Intel Compiler 19.1 Update 1 Intel(R) 64 Visual Studio 2019

```
> icl version.c /Qopenmp
インテル(R) 64 対応インテル(R) C++ コンパイラー (インテル(R) 64 対応アプリケーション用) バージョン 19.1.1.216 ビルド 20200306
(C) 1985-2020 Intel Corporation.  無断での引用、転載を禁じます。

version.c
Microsoft (R) Incremental Linker Version 14.26.28806.0
Copyright (C) Microsoft Corporation.  All rights reserved.

-out:version.exe
-defaultlib:libiomp5md.lib
-nodefaultlib:vcomp.lib
-nodefaultlib:vcompd.lib
version.obj

> version
OpenMP version 201611

>
```

Intel Compiler 19.1 Update 1 Intel(R) 64 Visual Studio 2019

```
> icl version.c /Qopenmp /Qnextgen
Intel(R) C++ Compiler for applications running on Intel(R) 64, Version 2021.1 NextGen Build 20200304
Copyright (C) 1985-2020 Intel Corporation. All rights reserved.

version.c
Microsoft (R) Incremental Linker Version 14.26.28806.0
Copyright (C) Microsoft Corporation.  All rights reserved.

-out:version.exe
-defaultlib:libiomp5md.lib
-nodefaultlib:vcomp.lib
-nodefaultlib:vcompd.lib
version.obj

> version
OpenMP version 201511

>
```

_OPENMP マクロが返す値が異なります

GPU オフロードについて (旧バージョンの機能)

```
bool Sobel::execute_offload()
{
            int w = COLOR_CHANNEL_NUM * image_width;
            float *outp = this->output;
            float *img = this->image;
            int iw = image_width;
            int ih = image_height;
#pragma omp target map(to: ih, iw, w) ¥
                        map(tofrom: img[0:iw*ih*COLOR_CHANNEL_NUM], ¥
                                                outp[0:iw*ih*COLOR_CHANNEL_NUM])
#pragma omp parallel for collapse(2)
     for (int i = 1; I < ih - 1; i++) {
          for (int k = COLOR_CHANNEL_NUM; k < (iw - 1) * COLOR_CHANNEL_NUM; k++) {
                        float gx = 1 * img[k + (i - 1) * w -1 * 4]
                                    + 2 * img[k + (i - 1) * w +0 * 4]
                                    + 1 * img[k + (i - 1) * w +1 * 4]
                                    - 1 * img[k + (i + 1) * w -1 * 4]
                                    - 2 * img[k + (i + 1) * w +0 * 4]
                                    - 1 * img[k + (i + 1) * w +1 * 4];
                        float gy = 1 * img[k + (i - 1) * w -1 * 4]
                                    - 1 * img[k + (i - 1) * w +1 * 4]
                                    + 2 * img[k + (i + 0) * w -1 * 4]
                                    - 2 * img[k + (i + 0) * w +1 * 4]
                                    + 1 * img[k + (i + 1) * w -1 * 4]
                                    - 1 * img[k + (i + 1) * w +1 * 4];
                        outp[i * w + k] = sqrtf(gx * gx + gy * gy) / 2.0;
               }
     }
     return true;
}
```

- インテル® C++/Fortran コンパイラーの V18.0 までは、インテル® グラフィックスへのオフロードがサポートされていました
- /Qopenmp-offload:gfx (Windows*)、-qopenmp-offload=gfx (Linux*) オプションでターゲットデバイスが指定できました

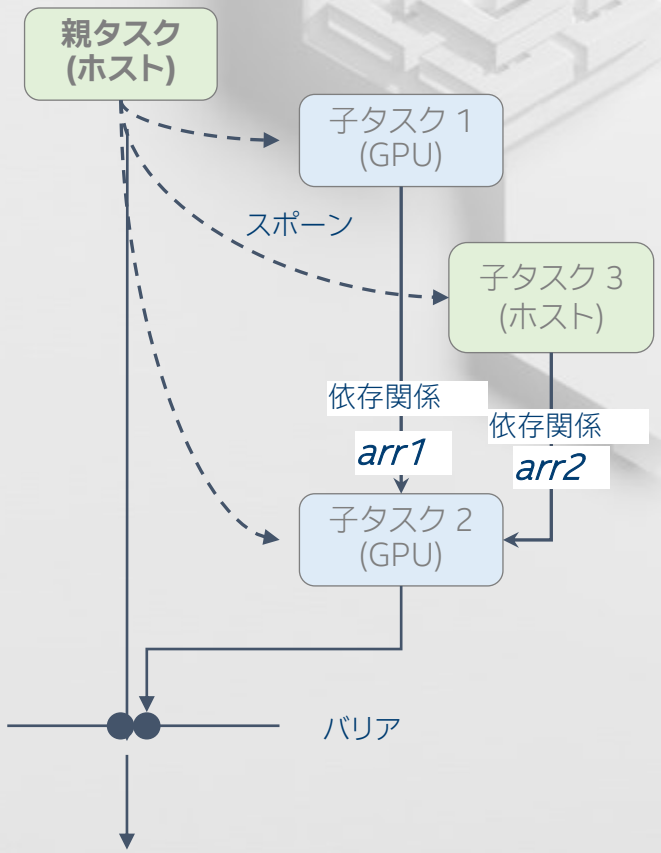
target ディレクティブで depend 節を使用して GPU へ非同期オフロードする例

```
// arr1 を初期化 - ターゲットへオフロード
#pragma omp target map(from: arr1[0:SIZE]) depend(out:arr1) nowait
#pragma omp parallel for
    for (int i = 0; i < SIZE; i++) { arr1[i] +=  i; }

// arr2 の初期化
#pragma omp task depend(out:arr2)
#pragma omp parallel for
    for (int i = 0; i < SIZE; i++) { arr2[i] += -i; }

// ターゲット上で中間結果を計算
#pragma omp target ¥
  map(to: arr1[0:SIZE], arr2[0:SIZE]) ¥
  map(from:arr3[0:SIZE]) ¥
  nowait depend(in:arr1, arr2)
#pragma omp parallel for
    for (int i = 0; i < SIZE; i++) { arr3[i] = arr1[i] + arr2[i]; }

#pragma omp taskwait
#pragma omp parallel for
    for (int i = 0; i < SIZE; i++) { res[i] += arr3[i]; }
```



Classic コンパイラーで並列化

```
#pragma omp parallel for
  for (int i = 0; i < MAX; i++)
    for (int k = 0; k < MAX; k++)
      for (int j = 0; j < MAX; j++)
        C[i][j] += A[i][k] * B[k][j];
```

Intel Compiler 19.1 Update 1 Intel(R) 64 Visual Studio 2019

```
> icl matmul_cpu.cpp /Qopenmp /nologo
matmul_cpu.cpp

> matmul_cpu
PASSED in 0.033000 sec (serial time was 0.217000 sec)

>
```

GPU オフロード向けにコードを変更

```
#pragma omp target teams distribute parallel for ¥
      map(to: A, B) map(tofrom: C) thread_limit(128)
  for (int i = 0; i < MAX; i++)
    for (int k = 0; k < MAX; k++)
      for (int j = 0; j < MAX; j++)
         C[i][j] += A[i][k] * B[k][j];
```

Intel Compiler 19.1 Update 1 Intel(R) 64 Visual Studio 2019

```
> icl matmul_cpu.cpp /Qopenmp /nologo
matmul_cpu.cpp
icc: 警告 #10274: プラットフォーム 'C:¥PROGRA~2¥INTELS~1¥COMPIL~1.216¥linux¥bin¥
intel64¥x86_64-linux.env' の構成が見つかりません。
icc: エラー #10417: インテル(R) コンパイラーのコンパイル環境の設定に問題があります。'install path' 設定 ('gcc.exe' から取得) が必要です。
icl: エラー #10340: ターゲットコンパイルの実行中に問題が発生しました。
```

Classic コンパイラーは、このソースをコンパイルできません

Nextgen モードでコンパイル

Intel Compiler 19.1 Update 1 Intel(R) 64 Visual Studio 2019

```
> icl matmul_cpu.cpp /Qopenmp /nologo /Qnextgen
matmul_cpu.cpp

> matmul_cpu
PASSED in 0.038000 sec (serial time was 0.253000 sec)
```

これは、どこで実行されているのか?

ホットスポット　CPU 利用率によるホットスポット

解析設定　収集ログ　サマリー　ボトムアップ　呼び出し

経過時間: 0.710s

CPU 時間:	0.806s
有効な時間:	0.518s
スピン時間:	0.102s
オーバーヘッド時間:	0.186s
合計スレッド数:	8
ポーズ時間:	0s

ホットスポット　CPU 利用率によるホットスポット

解析設定　収集ログ　サマリー　ボトムアップ　呼び出し元/呼び出し先　トップダウン・ツリー

0ミリ秒

スレッド

OMP Master Thread #0 (TID...
OMP Worker Thread #4 (TI...
OMP Worker Thread #5 (TI...
OMP Worker Thread #2 (TI...
OMP Worker Thread #1 (TI...
OMP Worker Thread #6 (TI...
OMP Worker Thread #7 (TI...
OMP Worker Thread #3 (TI...

スレッド
Running
CPU 時間
CPU 使用率
CPU 時間

Nextgen モードで /Qiopenmp コンパイル

Intel Compiler 19.1 Update 1 Intel(R) 64 Visual Studio 2019

```
> icl matmul_cpu.cpp /Qiopenmp /nologo /Qnextgen
matmul_cpu.cpp

> matmul_cpu
PASSED in 0.035000 sec (serial time was 0.252000 sec)
```

これは、どこで実行されているのか?

ホットスポット　CPU 利用率によるホットスポット

解析設定　収集ログ　サマリー　ボトムアップ　呼び出し

経過時間: 0.647s

CPU 時間:	0.734s
有効な時間:	0.520s
スピン時間:	0.067s
オーバーヘッド時間:	0.148s
合計スレッド数:	8
ポーズ時間:	0s

ホットスポット　CPU 利用率によるホットスポット

解析設定　収集ログ　サマリー　ボトムアップ　呼び出し元/呼び出し先　トップダウン・ツリー

0ミリ秒　500ミリ秒

スレッド

OMP Master Thread #0 (TID...
OMP Worker Thread #2 (TI...
OMP Worker Thread #4 (TI...
OMP Worker Thread #7 (TI...
OMP Worker Thread #6 (TI...
OMP Worker Thread #1 (TI...
OMP Worker Thread #5 (TI...
OMP Worker Thread #3 (TI...

スレッド
Running
CPU 時間
CPU 使用率
CPU 時間

/Qopenmp-targets:spir64 を追加

Intel Compiler 19.1 Update 1 Intel(R) 64 Visual Studio 2019

```
> icl matmul_cpu.cpp /Qiopenmp /
matmul_cpu.cpp
2057224.bc
matmul_cpu.cpp
LINK : fatal error LNK1104: ファ
```

このパスは標準のビルド環境向

PC › Win10 (C:) › Program Files (x86) › IntelSWTools › compilers

名前

- clang
- libomptarget-opencl.bc
- libomptarget-opencl.spv
- omptarget.lib

Intel Compiler 19.1 Update 1 Intel(R) 64 Visual Studio 2019

```
Microsoft(R) Program Maintenance Utility Version 14.26.28806.0
Copyright (C) Microsoft Corporation.  All rights reserved.

        icl /Zi /O2 /Qnextgen /Qiopenmp /Qopenmp-targets=spir64 matmul_cpu.cpp "
C:¥Program Files (x86)¥IntelSWTools¥compilers_and_libraries_2020.1.216¥windows¥l
ib¥omptarget.lib" -o matmul-fat
Intel(R) C++ Compiler for applications running on Intel(R) 64, Version 2021.1 Ne
xtGen Build 20200304
Copyright (C) 1985-2020 Intel Corporation. All rights reserved.

matmul_cpu.cpp
1906824.bc
matmul_cpu.cpp
Microsoft (R) Incremental Linker Version 14.26.28806.0
Copyright (C) Microsoft Corporation.  All rights reserved.

-out:matmul-fat.exe
-debug
-pdb:matmul-fat.pdb
-defaultlib:libiomp5md.lib
-nodefaultlib:vcomp.lib
-nodefaultlib:vcompd.lib
C:/Users/kiyo/AppData/Local/Temp/19068146.obj
C:/Users/kiyo/AppData/Local/Temp/1906853.obj
"C:/Program Files (x86)/IntelSWTools/compilers_and_libraries_2020.1.216/windows/
lib/omptarget.lib"
```

ビルドには 2 つのオプションが必要

- /Qiopenmp (Windows*)、-qiopenmp (Linux*)
- /Qopenmp-targets:spir64 (Windows*)、-qopenmp-targets=spir64 (Linux*)
- omptarget.lib へのパス設定が必要

Intel Compiler 19.1 Update 1 Intel(R) 64 Visual Studio 2019

```
> dir *.exe
 ドライブ C のボリューム ラベルは Win10 です
 ボリューム シリアル番号は BC63-5932 です

 C:¥Users¥kiyo¥Desktop¥oneAPI¥code¥matmul のディレクトリ

2020/07/12  05:15         1,150,464 matmul-fat.exe
2020/07/12  05:17           122,368 matmul_cpu.exe
               2 個のファイル           1,272,832 バイト
               0 個のディレクトリ  144,176,197,632 バイトの空き領域
```

生成されるバイナリーは、CPU バイナリーと GPU バイナリーが結合された FAT バイナリーとなります

GPU で実行するには

OMP_TARGET_OFFLOAD={"MANDATORY" | "DISABLED" | "DEFAULT" }

MANDATORY: GPU またはアクセラレーターでターゲット領域のコードを実行
DISABLED: CPU でターゲット領域のコードを実行
DEFAULT: デバイスが利用可能な場合に GPU で実行され、利用できない場合は CPU にターゲット領域のコードをフォールバックして実行

Intel Compiler 19.1 Update 1 Intel(R) 64 Visual Studio 2019

```
> set OMP_TARGET_OFFLOAD=DISABLED

> matmul-fat
PASSED in 0.033000 sec (serial time was 0.257000 sec)
```

これは!?!?

Intel Compiler 19.1 Update 1 Intel(R) 64 Visual Studio 2019

```
> set OMP_TARGET_OFFLOAD=MANDATORY

> matmul-fat
PASSED in 2.773000 sec (serial time was 0.260000 sec)
```

他のデバイスで実行するには (オプション)

- LIBOMPTARGET_DEVICETYPE に cpu、gpu、accelerator を指定できますが、他のデバイスを使用するにはデバイスをサポートするライブラリーが必要になります

- OMP_TARGET_OFFLOAD とも関連します

Intel(r) oneAPI Tools

```
> set LIBOMPTARGET_DEVICETYPE=accelerator

> matmul-fat
Target LEVEL0 RTL --> Warning: Invalid LIBOMPTARGET_DEVICETYPE=accelerator
Target OPENCL RTL --> Warning: Invalid or unsupported LIBOMPTARGET_DEVICETYPE=ac
celerator
Start to Calc
OpenMP Offload PASSED in 3.302000 sec (serial time was 0.257000 sec)

>
```

オフロードのプロファイル

- オフロードされたかテストするには、LIBOMPTARGET_PROFILE 環境変数を設定

  set LIBOMPTARGET_PROFILE=T

- この出力から、GPU へのデータの送信時間 (Data-Write) と GPU からのデータの受信時間 (Data-Read) が分かります。オフロード TARGET 領域 (EXEC) — main() の 56 行目があり、このカーネルで費やされた時間が示されています

Intel(r) oneAPI Tools

```
> set LIBOMPTARGET_PROFILE=T

> matmul-fat
Start to Calc
OpenMP Offload PASSED in 1.557000 sec (serial time was 0.259000 sec)
LIBOMPTARGET_PROFILE for OMP DEVICE(0) Intel(R) Gen9, Thread 8308
-- Name: Host Time (msec)
-- DataAlloc: 0.162
-- DataRead: 0.241
-- DataWrite: 2.893
-- Kernel#__omp_offloading_bc635932_aa51b__Z4main_l56: 1272.504
-- ModuleBuild: 223.446
-- Total: 1499.245

> _
```

オフロードのデバッグ

- 前述のプロファイルが表示されない場合、OpenMP* ランタイムが GPU ドライバーデバイスを検出できなかったことを意味します
- さらに詳しくデバッグするには、環境変数 LIBOMPTARGET_DEBUG=5 を設定してアプリケーションを再度実行します

Intel(r) oneAPI Tools

```
> set LIBOMPTARGET_DEBUG=5

> matmul-fat  2> debug.txt
Start to Calc
OpenMP Offload PASSED in 1.

>
```

```
Libomptarget --> TargetOffloadPolicy = DEFAULT
Libomptarget --> Initialized OMPT
Libomptarget --> Loading RTLs...
Libomptarget --> Loading library 'omptarget.rtl.level0.dll'...
Target LEVEL0 RTL --> Target device type is set to GPU
Target LEVEL0 RTL --> omp_get_thread_limit() returned 2147483647
Target LEVEL0 RTL --> omp_get_max_teams() returned 0
Libomptarget --> Successfully loaded library 'omptarget.rtl.level0.dll'!
Libomptarget --> Optional interface: __tgt_rtl_data_submit_nowait
Libomptarget --> Optional interface: __tgt_rtl_data_retrieve_nowait
Libomptarget --> Optional interface: __tgt_rtl_manifest_data_for_region
Libomptarget --> Optional interface: __tgt_rtl_data_alloc_base
Libomptarget --> Optional interface: __tgt_rtl_create_buffer
Libomptarget --> Optional interface: __tgt_rtl_release_buffer
Libomptarget --> Optional interface: __tgt_rtl_run_target_team_nd_region
Libomptarget --> Optional interface: __tgt_rtl_run_target_region_nowait
Libomptarget --> Optional interface: __tgt_rtl_run_target_team_region_nowait
Libomptarget --> Optional interface: __tgt_rtl_run_target_team_nd_region_nowait
Libomptarget --> Optional interface: __tgt_rtl_create_offload_queue
Libomptarget --> Optional interface: __tgt_rtl_release_offload_queue
```

インテル® VTune™ プロファイラー 2020 で確認

解析の設定　インテル VTUNE プロファイラー

どこを

ローカルホスト

何を

アプリケーションを起動

解析ターゲットの指定と設定: 実行するアプリケーションまたはスクリプト

アプリケーション:

C:\Users\kiyo\Desktop\oneAPI\code\matmul\matmul-fat.exe

アプリケーションの引数:

☑ アプリケーション・ディレクトリーを作業ディレクトリーとして使用

高度

どのように

GPU オフロード (プレビュー)

プラットフォーム上の各種 CPU と GPU コアのコード実行を
フロードによる影響を推測して、アプリケーションが CPU
かを特定します。 関連情報

⚠ ピーク帯域幅の測定はデフォルトで有効になります。正確
帯域幅は詳細な解析を行う上で重要ですが、小さなベンチ
るため収集時間が増加します。測定したくない場合、カス
機能を無効にしてください。

☑ GPU プログラミング API をトレース

☑ メモリー帯域幅を解析

☐ CPU 側のスタックを収集

☑ 低速な GPU パフォーマンスの詳細

詳細

GPU オフロード (プレビュー)　GPU オフロード (プレビュー)　インテル VTUNE プロファイラー

解析設定　収集ログ　サマリー　グラフィックス　プラットフォーム

経過時間: 5.160s

GPU 利用率: 33.9%

このセクションを使用して、GPU が適切に使用されているか、またどのエンジンが使用されているかを理解します。GPU 利用率のギャップ量を調査し、追加の負荷を与えられる可能性を特定します。このメトリックは、少なくとも 1 つのワークがスケジュールされているエンジン向けに計算されます。

GPU 利用率

GPU エンジンとワークタイプ別の GPU 利用率の内訳。

GPU エンジン / パケットタイプ	CPU 時間	GPU 利用率
レンダーと GPGPU	1.748s	33.9%
不明	1.748s	33.9%
エンジン 0	0.255s	4.9%
不明	0.255s	4.9%
エンジン 12	0.050s	1.0%
不明	0.050s	1.0%
エンジン 8	0.010s	0.2%
不明	0.010s	0.2%
エンジン 6	0.002s	0.0%
不明	0.002s	0.0%

*N/A は、集計されないメトリックに適用されます。

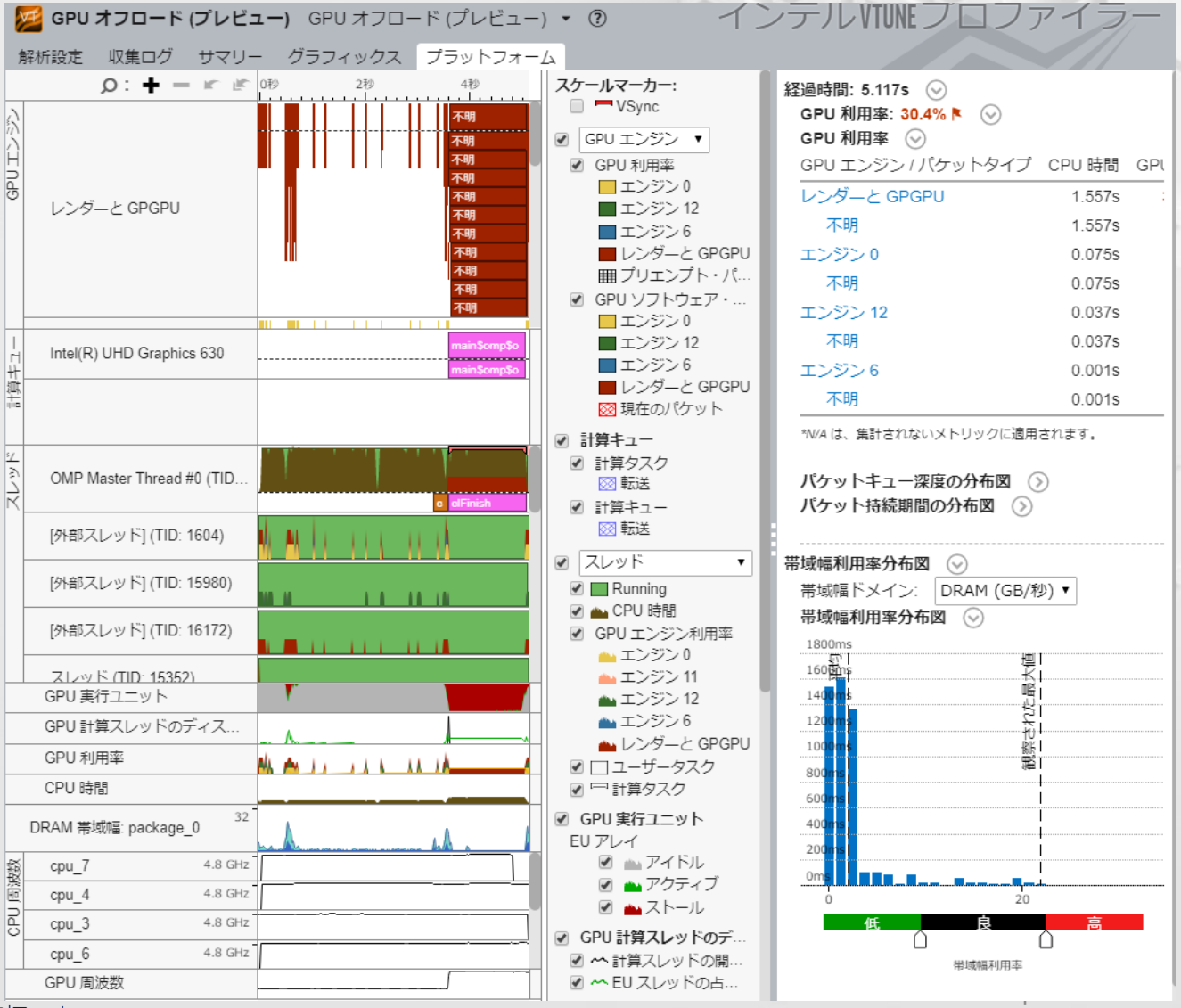
パケットキュー深度の分布図

パケット持続期間の分布図

インテル® VTune™ プロファイラー 2020 の GPU オフロード解析 (プレビュー)

- ここで説明した手順は、インテル® oneAPI HPC ツールキットに含まれるベータ版インテル® C++/Fortran コンパイラー 2021 でも利用できます

ここでの手順は動作を確認したのみであり、実際にはより高速に動作させるため最適化が必要です



簡単な最適化 (1)

```
#pragma omp data map(to: A, B) map(tofrom: C)
 {
  ............
#pragma omp target teams distribute parallel for thread_limit(128)
  for (int i = 0; i < MAX; i++)
    for (int k = 0; k < MAX; k++)
      for (int j = 0; j < MAX; j++)
         C[i][j] += A[i][k] * B[k][j];
 }
```

```
        set OMP_TARGET_OFFLOAD=DISABLED
        matmul-dmap.exe
Start to Calc
OpenMP Offload PASSED in 0.038000 sec (serial time was 0.260000 sec)
        set OMP_TARGET_OFFLOAD=MANDATORY
        matmul-dmap.exe
Start to Calc
OpenMP Offload PASSED in 1.253000 sec (serial time was 0.258000 sec)
```

簡単な最適化 (2)

```
#pragma omp data map(to: A, B) map(tofrom: C)
 {
  ............
#pragma omp target teams distribute thread_limit(128)
  for (int i = 0; i < MAX; i++)
    for (int k = 0; k < MAX; k++)
#pragma omp parallel for
      for (int j = 0; j < MAX; j++)
         C[i][j] += A[i][k] * B[k][j];
 }
```

```
        set OMP_TARGET_OFFLOAD=DISABLED
        matmul-opt1.exe
Start to Calc
OpenMP Offload PASSED in 0.884000 sec (serial time was 0.261000 sec)
        set OMP_TARGET_OFFLOAD=MANDATORY
        matmul-opt1.exe
Start to Calc
OpenMP Offload PASSED in 0.389000 sec (serial time was 0.255000 sec)
```

DPC++ への移行

oneAPI 環境への移行ステップ
インテル® oneAPI ベース・ツールキット

インテル® oneAPI ベース・ツールキットを使用するアクセラレーター向けのプログラミングでは、DPC++ コンパイラーと新しい SYCL* ベースのコードを使用する必要があります

- Fortran ユーザーは?
- OpenMP* ユーザーは?

インテル® oneAPI ベース・ツールキット

ダイレクト・プログラミング

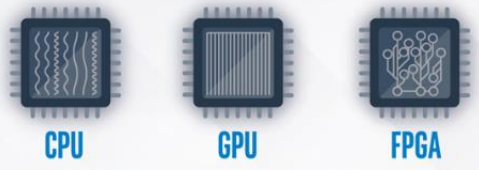
- インテル® oneAPI DPC++ コンパイラー
- インテル® DPC++ 互換性ツール
- Python* 向けインテル® ディストリビューション
- [オプション] oneAPI ベース・ツールキット用インテル® FPGA アドオン

API ベースのプログラミング

- インテル® oneAPI DPC++ ライブラリー
- インテル® oneAPI マス・カーネル・ライブラリー
- インテル® oneAPI データ・アナリティクス・ライブラリー
- インテル® oneAPI スレッディング・ビルディング・ブロック
- インテル® oneAPI ビデオ・プロセシング・ライブラリー
- インテル® oneAPI コレクティブ・コミュニケーション・ライブラリー
- インテル® oneAPI ディープ・ニューラル・ネットワーク・ライブラリー
- インテル® インテグレーテッド・パフォーマンス・プリミティブ

解析ツール

- インテル® VTune™ プロファイラー
- インテル® Advisor
- GDB*



最適化されたアプリケーション

インテル® oneAPI HPC ツールキット

ダイレクト・プログラミング

- インテル® oneAPI DPC++ コンパイラー
- インテル® DPC++ 互換性ツール
- Python* 向けインテル® ディストリビューション
- [オプション] oneAPI ベース・ツールキット用インテル® FPGA アドオン
- インテル® C++ コンパイラーと OpenMP*
- インテル® Fortran コンパイラーと OpenMP*

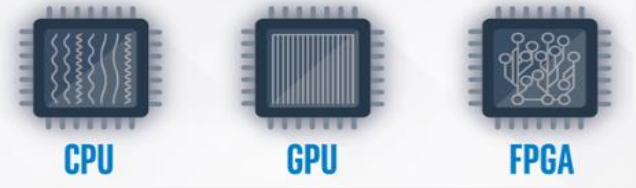
API ベースのプログラミング

- インテル® oneAPI DPC++ ライブラリー
- インテル® oneAPI マス・カーネル・ライブラリー
- インテル® oneAPI データ・アナリティクス・ライブラリー
- インテル® oneAPI スレッディング・ビルディング・ブロック
- インテル® oneAPI ビデオ・プロセシング・ライブラリー
- インテル® oneAPI コレクティブ・コミュニケーション・ライブラリー
- インテル® oneAPI ディープ・ニューラル・ネットワーク・ライブラリー
- インテル® インテグレーテッド・パフォーマンス・プリミティブ
- インテル® MPI ライブラリー

解析ツール

- インテル® VTune™ プロファイラー
- インテル® Advisor
- GDB*
- インテル® Inspector
- インテル® Trace Analyzer & Collector
- インテル® Cluster Checker

インテル® oneAPI ベース・ツールキット



データ並列 C++ (DPC++) の例

データ並列 C++
標準ベースのクロスアーキテクチャー言語

さまざまな 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言語

- 同じデータ並列プログラミング･モデルですべての SVMS アーキテクチャーをサポートする機能と抽象化を提供

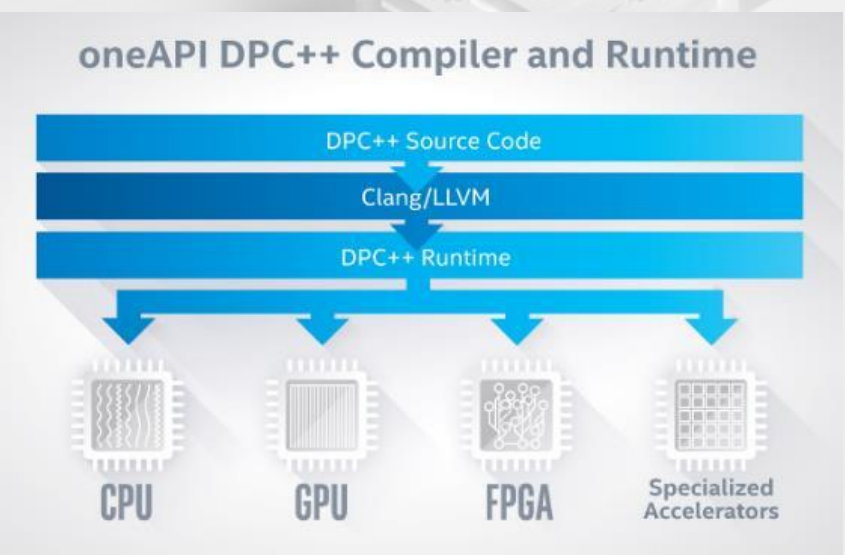
C++ ベース

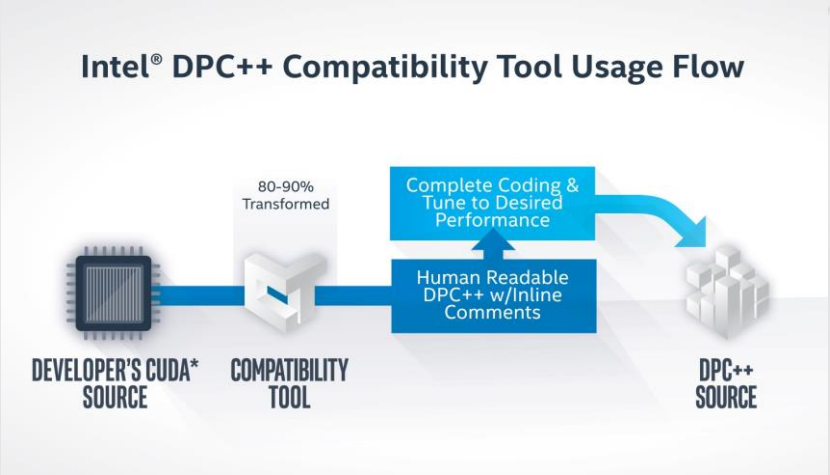
- 使い慣れた C/C++ 構文を使用できるため C++ の生産性が得られる
- Khronos Group の SYCL* を組み込むことでデータ並列処理とヘテロジニアス･プログラミングをサポート

コミュニティー･プロジェクトを通して推進される言語拡張

- データ並列プログラミングを容易にする拡張
- 継続的な進化のためのオープンな共同開発

インテルのアーキテクチャーとコンパイラーの長年の経験に基づいて構築





DPC++ プログラムの例

単一ソース

- ホストコードとヘテロジニアス･アクセラレーター･カーネルを同じソースファイルに混在して記述

使い慣れた C++

- ライブラリー構造は、次の機能を提供:

コンストラクト	目的
queue	ワークを投入
buffer	データ管理
parallel_for	並列処理

ホストコード

アクセラレーター･コード

ホストコード

```
#include <CL/sycl.hpp>
#include <iostream>
constexpr int num=16;
using namespace cl::sycl;

int main() {
  auto R = range<1>{ num };
  buffer<int> A{ R };

  queue{}.submit([&](handler& h) {
    auto out =
      A.get_access<access::mode::write>(h);
    h.parallel_for(R, [=](id<1> idx) {
      out[idx] = idx[0];
    });
  });
  auto result =
    A.get_access<access::mode::read>();
  for (int i=0; i<num; ++i)
    std::cout << result[i] << "¥n";

  return 0;
}
```

OpenMP* から データ並列 C++ へ

```
#pragma omp target teams distribute parallel for ¥
    map(to: A, B) map(tofrom: C) thread_limit(128)
  for (int i = 0; i < MAX; i++)
    for (int k = 0; k < MAX; k++)
      for (int j = 0; j < MAX; j++)
        C[i][j] += A[i][k] * B[k][j];
```

```
cl::sycl::gpu_selector device;
cl::sycl::queue deviceQueue(device);
cl::sycl::range<2> matrix_range(MAX, MAX);

cl::sycl::buffer<TYPE, 2> bufferA((TYPE*) A, matrix_range);
cl::sycl::buffer<TYPE, 2> bufferB((TYPE*) B, matrix_range);
cl::sycl::buffer<TYPE, 2> bufferC((TYPE*) C, matrix_range);

deviceQueue.submit([&](cl::sycl::handler& cgh){
    auto accessorA = bufferA.template get_access<sycl::access::mode::read>(cgh);
    auto accessorB = bufferB.template get_access<sycl::access::mode::read>(cgh);
    auto accessorC = bufferC.template get_access<sycl::access::mode::read_write>(cgh);

    cgh.parallel_for(matrix_range,
        [=](cl::sycl::id<2> ind){
        int k;
        for(k=0; k<MAX; k++){
            accessorC[ind[0]][ind[1]] += accessorA[ind[0]][k] * accessorB[k][ind[1]];
        }
    });
});
```

C / C++ コードの典型的な最適化を実装

```
cl::sycl::accessor<TYPE, 2, cl::sycl::access::mode::read_write, cl::sycl::access::target::local>
   aTile(cl::sycl::range<2>(MATRIX_TILE_SIZE, MATRIX_TILE_SIZE), cgh);
cl::sycl::accessor<TYPE, 2, cl::sycl::access::mode::read_write, cl::sycl::access::target::local>
   bTile(cl::sycl::range<2>(MATRIX_TILE_SIZE, MATRIX_TILE_SIZE), cgh);

cgh.parallel_for(cl::sycl::nd_range<2>(matrix_range, tile_range),
        [=](cl::sycl::nd_item<2> it){
        int k;
           const int numTiles = MAX/MATRIX_TILE_SIZE;
           const int row = it.get_local_id(0);
           const int col = it.get_local_id(1);
           const int globalRow = MATRIX_TILE_SIZE * it.get_group(0) + row;
           const int globalCol = MATRIX_TILE_SIZE * it.get_group(1) + col;
           TYPE acc = 0.0;
           for (int t=0; t<numTiles; t++){
               const int tiledRow = MATRIX_TILE_SIZE * t + row;
               const int tiledCol = MATRIX_TILE_SIZE * t + col;
               aTile[row][col] = accessorA[globalRow][tiledCol];
               bTile[row][col] = accessorB[tiledRow][globalCol];
               it.barrier(cl::sycl::access::fence_space::local_space);
               for (k = 0; k < MATRIX_TILE_SIZE; k++){
                    acc += aTile[row][k] * bTile[k][col];
               }
               it.barrier(cl::sycl::access::fence_space::local_space);
           }
           accessorC[globalRow][globalCol] = acc;
});
```

16 x 16 のタイルを定義して
ブロック化を実装

パフォーマンスは …

インテル® Core™ i7-9600K (インテル® UHD グラフィックス 630) で実行

```
Intel(r) oneAPI Tools

> nmake run

Microsoft(R) Program Maintenance Utility Version 14.26.28806.0
Copyright (C) Microsoft Corporation.  All rights reserved.

        set OMP_TARGET_OFFLOAD=DISABLED
        matmul-fat.exe
Start to Calc
OpenMP Offload PASSED in 0.034000 sec (serial time was 0.251000 sec)
        set OMP_TARGET_OFFLOAD=MANDATORY
        matmul-fat.exe
Start to Calc
OpenMP Offload PASSED in 1.231000 sec (serial time was 0.252000 sec)
        matmul-dpc.exe
Start to calc
Data Parallel C++ PASSED in 0.861000 sec (serial time was 0.356000 sec)
        matmul-dpcopt.exe
Start to calc
Data Parallel C++ PASSED in 0.638000 sec (serial time was 0.354000 sec)

>
```

インテル® Core™ i7-1065G (インテル® Iris® Plus グラフィックス) で実行

```
Intel(r) oneAPI Tools

> nmake run

Microsoft(R) Program Maintenance Utility Version 14.26.28806.0
Copyright (C) Microsoft Corporation.  All rights reserved.

        set OMP_TARGET_OFFLOAD=DISABLED
        matmul-fat.exe
Start to Calc
OpenMP Offload PASSED in 0.075000 sec (serial time was 0.263000 sec)
        set OMP_TARGET_OFFLOAD=MANDATORY
        matmul-fat.exe
Start to Calc
OpenMP Offload PASSED in 0.846000 sec (serial time was 0.262000 sec)
        matmul-dpc.exe
Start to calc
Data Parallel C++ PASSED in 0.907000 sec (serial time was 0.553000 sec)
        matmul-dpcopt.exe
Start to calc
Data Parallel C++ PASSED in 0.637000 sec (serial time was 0.529000 sec)

>
```

パフォーマンスは開発中の DPC++ およびライブラリーを使用して測定したものであり、今後変わる可能性があります

参考資料

- この資料の詳細は iSUS で公開中の

  ベータ版インテル® C++ コンパイラーおよびベータ版インテル® Fortran コンパイラー向けの GPU への OpenMP* オフロード導入

  https://www.isus.jp/products/c-compilers/get-started-with-cpp-fortran-compiler-openmp/

  を参照してください

まとめ

oneAPI 向けのデータ並列 C++ (DPC++) へ移行する前に、
インテル® C++/Fortran コンパイラー V19.1 やインテル® oneAPI HPC ツールキット
に含まれるベータ版インテル® C++/Fortran コンパイラー 2021 を使用して簡単に
インテル® グラフィックスへのオフロードを行うソフトウェアを開発および検証できます

インテル® oneAPI HPC ツールキット Beta09 の追記 (1)

インテル® oneAPI HPC ツールキットに同梱されるインテル® コンパイラーで OpenMP* 5.0 が正式にサポートされました

Intel(r) oneAPI Tools

```
> icl omp_oct2019.c /Qopenmp /nologo /WO
omp_oct2019.c

> omp_oct2019
Find Openmp Version by iSUS - Oct'2019
 OpenMP Version: 5.0 TR4 (201611)

> icx omp_oct2019.c /Qopenmp /nologo /WO

> omp_oct2019
Find Openmp Version by iSUS - Oct'2019
 OpenMP Version: 5.0 (201811)
```

クラシックモード

Nextgen モード

インテル® コンパイラーの Nextgen モードでは、クラシックモードでサポートされていた、いくつかのオプションが未サポートとなっています。詳細は /Qnextgen-diag オプションで確認してください

インテル® oneAPI HPC ツールキット Beta09 の追記 (2)

インテル® oneAPI HPC ツールキットに同梱されるインテル® コンパイラーで OpenMP* 5.0 が正式にサポートされたことにより、前回のセッションで紹介したいくつかの OpenMP* 5.0 の機能が実装されています

```
#pragma omp parallel loop private(x) reduction(+:sum)
        for (i=0; i<num_steps; i++)
        {
                x = (i + .5)*step;
                sum += 4.0/(1.+ x*x);
        }
```

```
Intel(r) oneAPI Tools

> icl pi.c /Qopenmp /DLOOP
インテル(R) 64 対応インテル(R) C++ コンパイラー (インテル(R) 64 対応アプリケーシ
ョン用) バージョン 2021.1 ベータ ビルド 20200827
(C) 1985-2020 Intel Corporation.  無断での引用、転載を禁じます。

pi.c
pi.c(20): エラー: omp 句に構文エラーがあります。
  #pragma omp parallel loop private(x) reduction(+:sum)
                       ^
```

Classic

```
Intel(r) oneAPI Tools

> icx pi.c /Qopenmp /DLOOP
Intel(R) oneAPI DPC++/C++ Compiler Pro for applications running on Intel(R) 64,
Version 2021.1 Beta Build 20200827
Copyright (C) 1985-2020 Intel Corporation. All rights reserved.


> pi
The value of PI is  3.1415927
The time to calculate PI was 0.152000 seconds
```

Nextgen

OpenMP* 5.1 の補足と 5.0 対応の例題

- OpenMP* 5.1 TR9 のドラフト仕様が公開されています:

  https://www.openmp.org/wp-content/uploads/openmp-TR9.pdf
- OpenMP* Application Programming Interface Examples version 5.0.1 が公開されています:

  https://www.openmp.org/openmp-updates/openmp-examples-updated-with-5-0-features/

iSUS

ソフトウェア・セミナー