インテル® oneAPI ツールキットを使用した

はじめてのヘテロジニアス・プログラミング

iSUS 編集長　すがわら　きよふみ

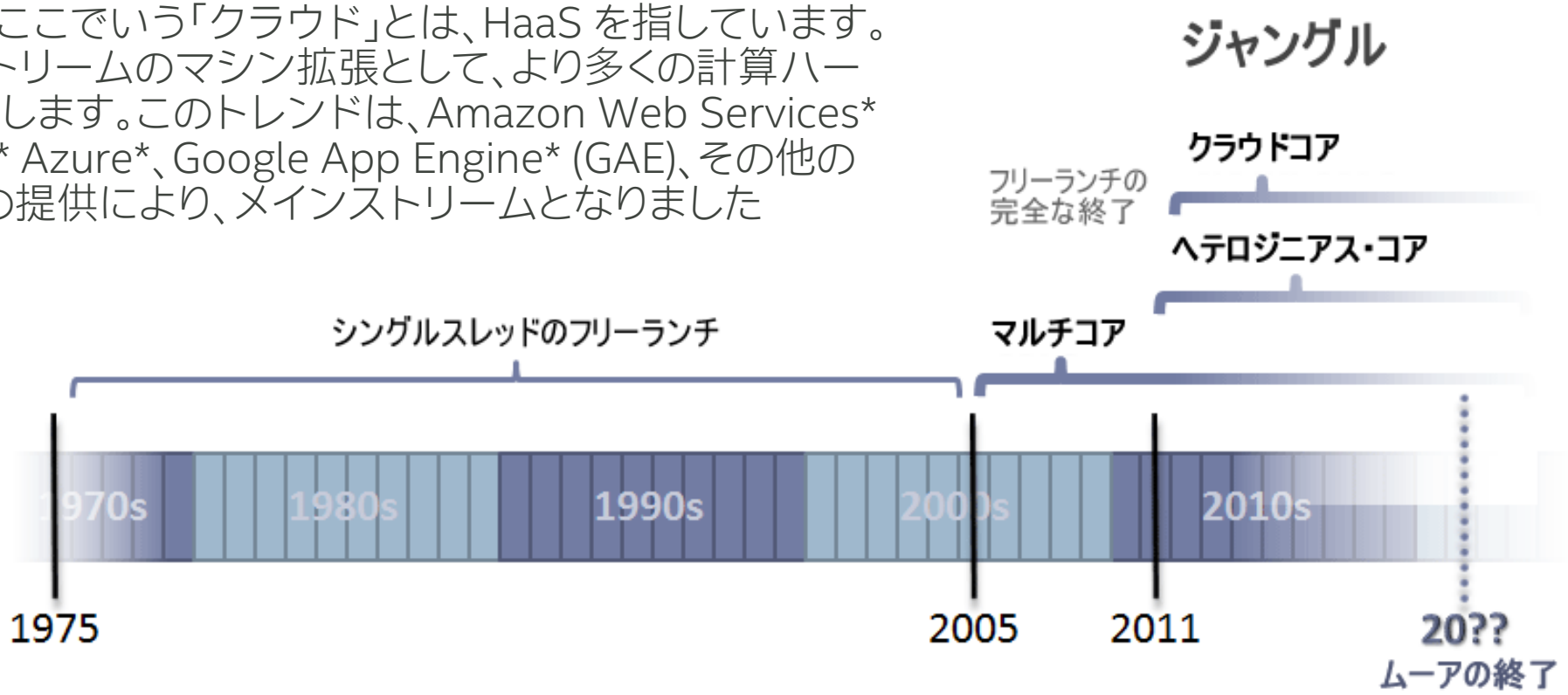
内容

- パラレルジャングルへの 3 つの段階
- ヘテロジニアス・コンピューティングの課題
- OpenMP* と SYCL* のオフロードの例

パラレルジャングルへの 3 つの段階

ヘテロジニアス・コア: メインストリームのラップトップ、コンソール、タブレットに CPU と計算処理が可能な GPU の両方が搭載されているように、1 台のコンピューターに 2 種類以上のプロセッサー・コアが搭載されていることは当たり前になっています

計算クラウドコア: ここでいう「クラウド」とは、HaaS を指しています。HaaS は、メインストリームのマシン拡張として、より多くの計算ハードウェアにアクセスします。このトレンドは、Amazon Web Services* (AWS*)、Microsoft* Azure*、Google App Engine* (GAE)、その他の商用計算クラウドの提供により、メインストリームとなりました

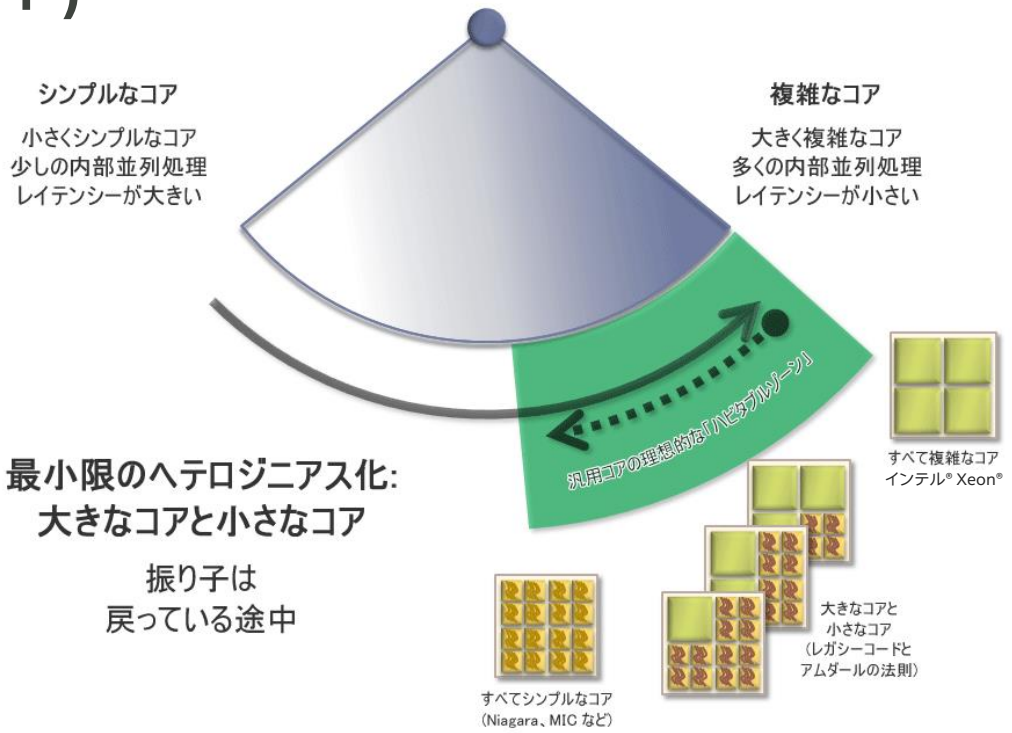


https://www.isus.jp/hpc/welcome-to-parallel-jungle/

ムーアの法則の採掘

- 単一コア「フリーランチ」(1975-2005)
- フェーズ II、第 2 の鉱脈 = ホモジニアス・マルチコア (2005-)
- フェーズ III、第 3 の鉱脈 = ヘテロジニアス・コア (2011-)
  - 大きく高速なコア vs 小さく低速なコア
  - 汎用コア vs 専用コア

GPGPU は**十分に利用されていない**ハードウェアを活用するという点で興味深いものです



プログラマーの視点

1. アプリケーションを少なくとも大規模に並列化して、非ローカルコアやヘテロジニアス・コアを最大限に使用できるようにする必要がある

2. 効率とパフォーマンスの最適化がより重要になる

3. プログラミング言語とシステムは、ヘテロジニアスな分散型並列処理に対処することが求められる

   - 異なるパフォーマンスの計算コア (大きく高速、小さく低速) をサポートしてプロセッサーの下部のセクションに取り組む
   - 完全にサポートしていないメインストリームの言語機能を含む異なる機能をコアで利用できるように、言語のサブセットをサポートしてプロセッサーの上部のセクションに取り組む
   - ローカルにスケーリングするだけでなく計算クラウドにもスケーリングする分散型アルゴリズムを提供して、計算のメモリーに取り組む
   - 多くのノードにわたって分散される分散型データコンテナを提供してデータのメモリーに取り組む
   - 同じソースコードでグラフ全体を扱うことができるユニファイド・プログラミング・モデルを可能にする

ソフトウェアにおける 6 つのトレンド

- それぞれ 1958年から 1973年の間に誕生し、1990年代/2000年代に台頭し始めて、5年強をかけて成熟したツール/言語/フレームワーク/ランタイム体系が構築されたました

ソフトウェア全体に与える影響の比較

	GUI	オブジェクト	GC	ジェネリック	ウェブ	並列化	ヘテロジニアス
アプリケーション・プログラミング・モデル	●●●	●●●	●	●	●	●●●	●●●
ライブラリーとフレームワーク	●●●	●●●		●●	●●●	●●●	●●●
言語とコンパイラー	●●	●●●	●	●●		●●	●●●
ランタイムと OS	●●		●●		●	●●●	●●
ツール（設計、測定、テスト）	●●●	●	●		●●	●●	●●●

影響の範囲と重要性

● = 多少。1 つの主要な製品リリース

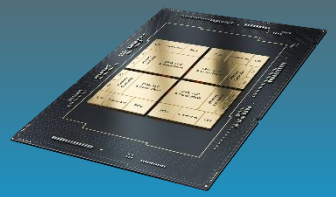
●● = 重要。1 つ以上の製品リリース

●●● = 新しい考え方/必須。複数の主要リリース

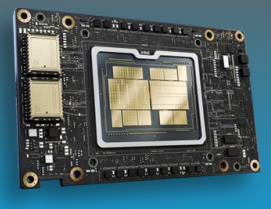
高い処理能力が求められる現代のアプリケーション

今日のパフォーマンス要件を満たすためには多様なアクセラレーターが必要
開発者の 48% が 2 種類以上のプロセッサーやコアを使用する
ヘテロジニアス・システムをターゲットにしている[1]

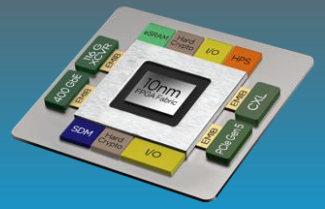
CPU



GPU



FPGA



その他のアクセラレーター

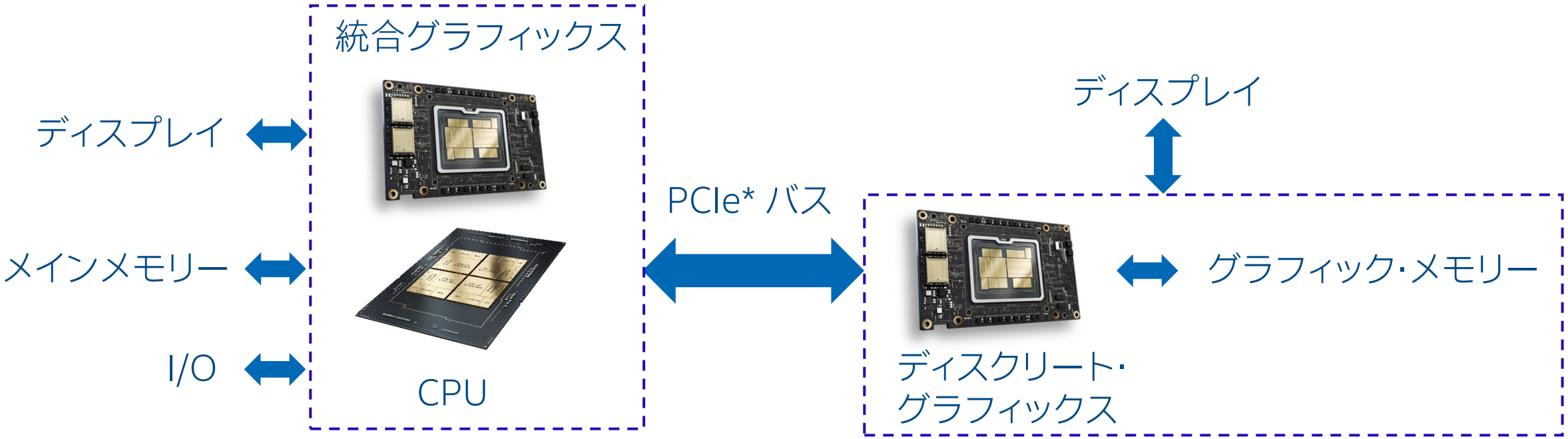
開発者の課題: 複数のアーキテクチャー、ベンダー、プログラミング・モデル

oneAPI

オープンな標準ベースのマルチアーキテクチャー・プログラミング

1. 2022年6月に Evans Data が実施した「Global Development Survey Report 22.1」

ヘテロジニアス・コンピューティングの課題



1. OS やライブラリーを含むほとんどのサービスはホスト側で実行される
2. 利用するデータはメインメモリーに存在する
3. PCIe* バスはメモリーアクセスに比べると低速である

マルチスレッド化の例を思い出してみましょう


```
#pragma omp parallel for private(loop)
for (int i = 0; i < VHIGH; i++){
    float cy = (float)((i - Y_CT) / Y_CT * scale);
        for(int j = 0; j < HWIDE; j++) {
            float cx = (float)((j - X_CT) / X_CT * scale + ox);
            float x = 0.0f; float y = 0.0f;
            for(loop = 0; loop < ITER; loop++) {
               float new_x, new_y;
               new_x = x * x - y * y + cx;
               new_y = (float)2.0 * x * y + cy;
               x = new_x;
               y = new_y;
               if((x * x + y * y)>5.5) break;
            }
#pragma omp critical
            {
        if( loop >= ITER ) SetPixel(hdc, j, i, ColorTable[ 0 ] );
        else SetPixel(hdc, j, i, ColorTable[ loop % __COLOR__ + 1 ] );
            }
        }
}
```

このコードのままでは、マルチスレッド化の恩恵は全く得られません

マルチスレッド化のための改善


```
#pragma omp parallel for private(loop)
for (int i = 0; i < VHIGH; i++){
    float cy = (float)((i - Y_CT) / Y_CT * scale);
        for(int j = 0; j < HWIDE; j++) {
            float cx = (float)((j - X_CT) / X_CT * scale + ox);
            float x = 0,0f; float y = y=0.0f;
            for(loop = 0; loop < ITER; loop++) {
               float new_x, new_y;
               new_x = x * x - y * y + cx;
               new_y = (float)2.0 * x * y + cy;
               x = new_x;
               y = new_y;
               if((x * x + y * y)>5.5) break;
            }
#pragma omp critical
            {
             Draw_table[i][j]  = loop;
            }
        }
}
```

このループでは各ポイントのマンデルブロ値を計算し配列に格納します
描画は別関数またはスレッドで行います

GPU へのオフロードは可能か？


```
#pragma omp target teams distribute parallel for private(loop) \
                                                 map (from:Draw_table)
for (int i = 0; i < VHIGH; i++){
    float cy = (float)((i - Y_CT) / Y_CT * scale);
        for(int j = 0; j < HWIDE; j++) {
             float cx = (float)((j - X_CT) / X_CT * scale + ox);
             float x = 0.0f; float y = 0.0f;
             for(loop = 0; loop < ITER; loop++) {
                float new_x, new_y;
                new_x = x * x - y * y + cx;
                new_y = (float)2.0 * x * y + cy;
                x = new_x;
                y = new_y;
                if((x * x + y * y)>5.5) break;
             }
#pragma omp critical
             {
              Draw_table[i][j]  = loop;
             }
        }
}
```

スレッドセーフなコードはデバイスへのオフロードの対象にできます

ただし…

```
for (unsigned int y=startLine; y < endLine; y++){
   c.imag = p->pMandelSpec->min.imag + y*imagScale;
   int scanLineOffset = y*p->pImageSpec->pitch/sizeof(DWORD);
   for (unsigned int x=0; x < p->pImageSpec->width; x++){
      c.real = minreal + x*realScale;
      color = CalculatePixel(c);
      pBitmap[scanLineOffset+x] = (color << 16) + (color << 8) + ((color & 0x0F) << 4);
   }
   PostMessage(p->hWndProgressBar, PBM_STEPIT, 0, 0);
}
```

アーキテクチャーに依存するインライン・アセンブラーや組込み関数を使用する場合、ターゲットは限定されます

```
Inline DWORD CalculatePixel(complex c){
   _asm {
   mov eax, 0              // count
   fld1                                  // 1
   fadd st(0), st(0) // 2
   fadd st(0), st(0) // 4
   fld c.real  // c.real, 4
   fld c.imag  // c.imag, c.real, 4
   fldz        // z.real, c.imag, c.real, 4
   fldz        // z.imag, z.real, c.imag, c.real, 4
   fldz        // z.real^2, z.imag, z.real, c.imag, c.real, 4
   fldz        // z.imag^2, z.real^2, z.imag, z.real, c.imag, c.real, 4
   ……
}
```

OpenMP* によるオフロードのサポート

インテル® DPC++/C++ コンパイラーでは次のオプションでターゲットデバイスへのオフロードをサポートします

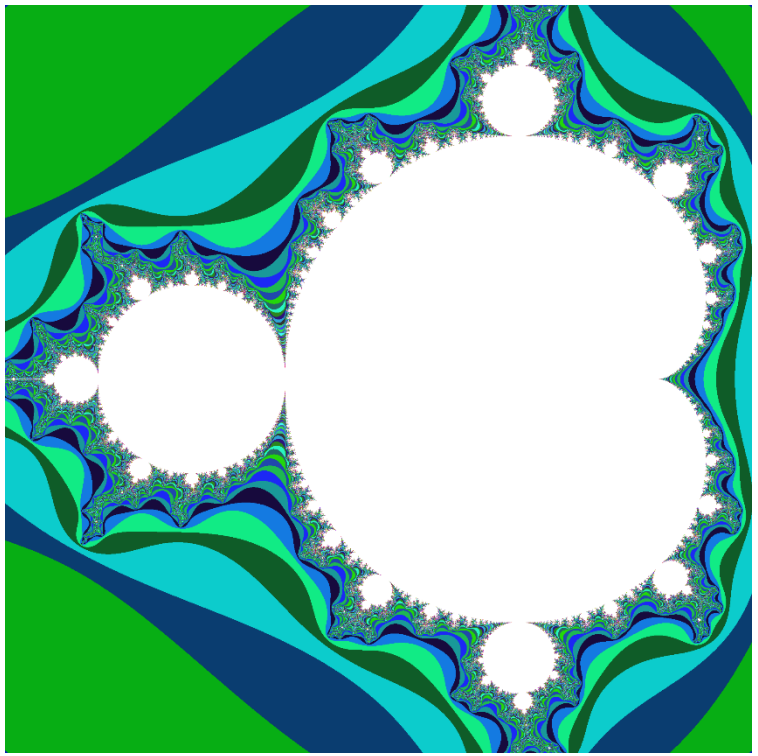
-qopenmp –qopenmp-targets=spir64

ただし生成されたバイナリーを実行できるのは、インテル® プロセッサーとインテル® グラフィックス・デバイスのみです

NVIDIA* または AMD* グラフィックス・デバイスへのオフロード機能は、正式にはサポートされていません

SYCL* の利用を検討してみましょう!!!

SYCL* で stb ライブラリーを使用して結果をファイルへ出力



```
q.submit([&](handler &h) {
      // バッファーへアクセスするアクセサーを定義
      auto b = data_buf.get_access(h,write_only);

      // イメージ全体を反復して各ピクセルのマンデルブロを計算
      h.parallel_for(range<2>(rows, cols), [=](auto index) {
        int i = int(index[0]);
        int j = int(index[1]);
        auto c = MandelParameters::ComplexF(p.ScaleRow(i), p.ScaleCol(j));
        b[index] = p.Point(c);
      });
});
```

MandelParameters::ComplexF での各ポイントの計算は前述の例と同等です

サンプルコードは次からダウンロードできます: https://github.com/oneapi-src/oneAPI-samples (英語)

SYCL* を利用したソースのコンパイル

SYCL* を使用したソースファイルには次のコマンドを使用します

```
$ icpx -fsycl mandel.cpp -o mandel.exe
```

生成したバイナリーは、インテル® プロセッサーとインテル® グラフィックス・デバイスで実行できます

NVIDIA* と AMD* のグラフィックス・デバイスで実行するには …

CodePlay 社の oneAPI for NVIDIA*/AMD* GPU プラグインを使用します

このプラグインは、現在 Linux* 環境でのみサポートされます

oneAPI for NVIDIA*/AMD* GPU プラグインを使用する

- oneAPI for NVIDIA* GPU プラグインを使用するには CUDA* ツールキットが必要です
- oneAPI for AMD* GPUプラグインを使用するには ROCm* ソフトウェア・スタックが必要です

プラグインをインストールして GPU が利用できることを確認したら次のコマンドでソースファイルをコンパイルします

```
$ icpx -fsycl -fsycl-targets=nvptx64-nvidia-cuda sycl-app.cpp -o sycl-app

$ icpx -fsycl -fsycl-targets=amdgcn-amd-amdhsa -Xsycl-target-backend=amdgcn-amd-amdhsa --offload-arch=gfx1030 sycl-app.cpp -o sycl-app
```

https://www.isus.jp/products/oneapi/oneapi-for-nvidia-gpu-get-started/

https://www.isus.jp/products/oneapi/oneapi-for-amd-gpu-get-started/

複数のターゲットデバイスで実行可能な SYCL* アプリケーションを作成

- -fsycl-targets に複数のターゲットを指定することで、単一バイナリーで複数のターゲットデバイスで実行可能なファイルを作成できます

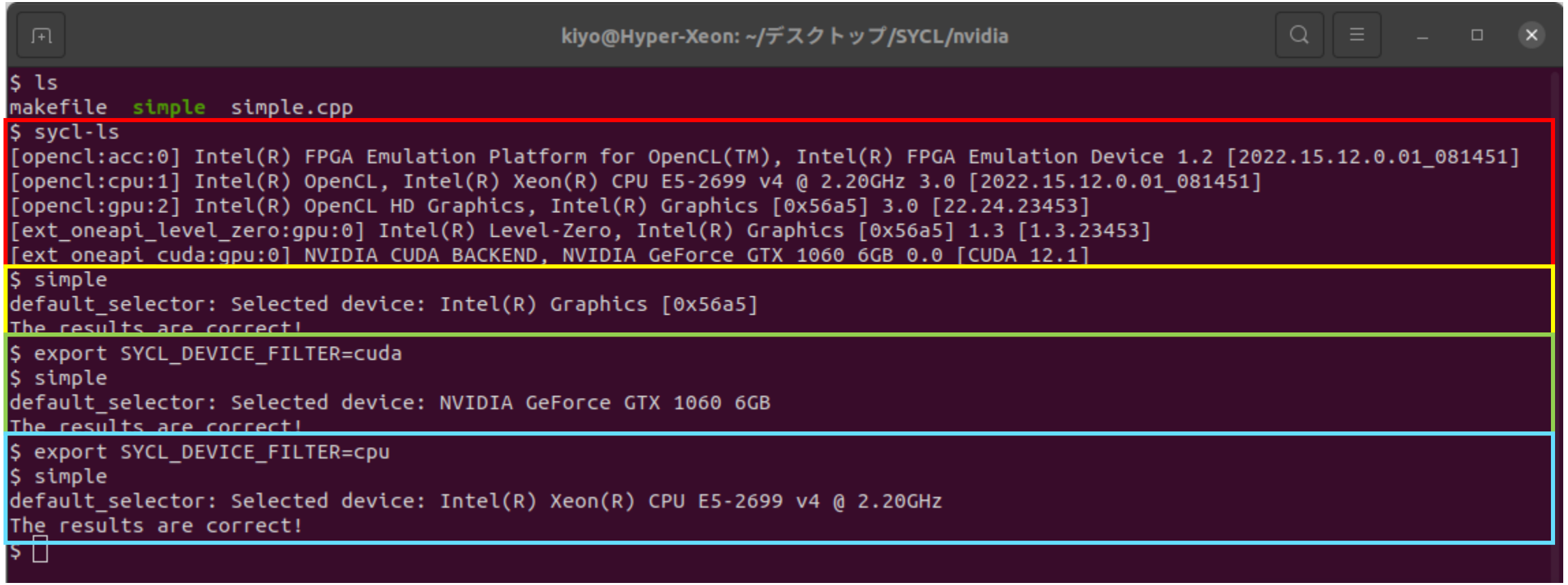
```
$ icpx -fsycl -fsycl-targets=nvptx64-nvidia-cuda,spir64 sycl-app.cpp -o sycl-app
```

`-fsycl-targets=nvptx64-nvidia-cuda,spir64` オプションでは、NVIDIA* GPU と SPIR64 (インテル® GPU など) で実行可能な SYCL* アプリケーションを生成できます

AMD* GPU もターゲットにする場合、`amdgcn-amd-amdhsa` と `-Xsycl-target-backend=amdgcn-amd-amdhsa --offload-arch=gfx1030` を追加します

実行例

Ubuntu*


kiyo@Hyper-Xeon: ~/デスクトップ/SYCL/nvidia
$ ls
makefile simple simple.cpp
$ sycl-ls
[opencl:acc:0] Intel(R) FPGA Emulation Platform for OpenCL(TM), Intel(R) FPGA Emulation Device 1.2 [2022.15.12.0.01_081451]
[opencl:cpu:1] Intel(R) OpenCL, Intel(R) Xeon(R) CPU E5-2699 v4 @ 2.20GHz 3.0 [2022.15.12.0.01_081451]
[opencl:gpu:2] Intel(R) OpenCL HD Graphics, Intel(R) Graphics [0x56a5] 3.0 [22.24.23453]
[ext_oneapi_level_zero:gpu:0] Intel(R) Level-Zero, Intel(R) Graphics [0x56a5] 1.3 [1.3.23453]
[ext_oneapi_cuda:gpu:0] NVIDIA CUDA BACKEND, NVIDIA GeForce GTX 1060 6GB 0.0 [CUDA 12.1]
$ simple
default_selector: Selected device: Intel(R) Graphics [0x56a5]
The results are correct!
$ export SYCL_DEVICE_FILTER=cuda
$ simple
default_selector: Selected device: NVIDIA GeForce GTX 1060 6GB
The results are correct!
$ export SYCL_DEVICE_FILTER=cpu
$ simple
default_selector: Selected device: Intel(R) Xeon(R) CPU E5-2699 v4 @ 2.20GHz
The results are correct!
$

kiyo@Xeon-E5: ~/デスクトップ/SYCL/mandel

```
$ sycl-ls
[opencl:acc:0] Intel(R) FPGA Emulation Platform for OpenCL(TM), Intel(R) FPGA Emulation Device 1.2 [2023.15.3.0.20_160000]
[opencl:cpu:1] Intel(R) OpenCL, Intel(R) Xeon(R) CPU E5-2699 v4 @ 2.20GHz 3.0 [2023.15.3.0.20_160000]
[opencl:gpu:2] Intel(R) OpenCL HD Graphics, Intel(
[ext_oneapi_level_zero:gpu:0] Intel(R) Level-Zero
[ext_oneapi_cuda:gpu:0] NVIDIA CUDA BACKEND, NVID
$ export ONEAPI_DEVICE_SELECTOR="opencl:2"
$ mandel
    Platform Name: Intel(R) OpenCL HD Graphics
  Platform Version: OpenCL 3.0
      Device Name: Intel(R) Arc(TM) A380 Graphic
   Max Work Group: 1024
 Max Compute Units: 128

Parallel Mandelbrot set using buffers.
 Rendered image output to file: mandelbrot.png (o
      Serial time: 0.563607s
    Parallel time: 0.0109801s
Successfully computed Mandelbrot set.
$
```

kiyo@Xeon-E5: ~/デスクトップ/SYCL/mandel

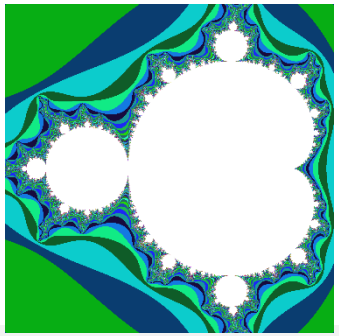
```
$ sycl-ls
[opencl:acc:0] Intel(R) FPGA Emulation Platform for OpenCL(TM), Intel(R) FPGA Emulation Device 1.2 [2023.15.3.0.20_160000]
[opencl:cpu:1] Intel(R) OpenCL, Intel(R) Xeon
[opencl:gpu:2] Intel(R) OpenCL HD Graphics, I
[ext_oneapi_level_zero:gpu:0] Intel(R) Level-
[ext_oneapi_cuda:gpu:0] NVIDIA CUDA BACKEND,
$ export ONEAPI_DEVICE_SELECTOR="opencl:1"
$ mandel
    Platform Name: Intel(R) OpenCL
  Platform Version: OpenCL 3.0 LINUX
      Device Name: Intel(R) Xeon(R) CPU E5-2
   Max Work Group: 8192
 Max Compute Units: 44

Parallel Mandelbrot set using buffers.
 Rendered image output to file: mandelbrot.pn
      Serial time: 0.563242s
    Parallel time: 0.0045227s
Successfully computed Mandelbrot set.
$
```

kiyo@Xeon-E5: ~/デスクトップ/SYCL/mandel

```
$ sycl-ls
[opencl:acc:0] Intel(R) FPGA Emulation Platform for OpenCL(TM), Intel(R) FPGA Emulation Device 1.2 [2023.15.
[opencl:cpu:1] Intel(R) OpenCL, Intel(R) Xeon(R) CPU E5-2699 v4 @ 2.20GHz 3.0 [2023.15.3.0.20_160000]
[opencl:gpu:2] Intel(R) OpenCL HD Graphics, Intel(R) Arc(TM) A380 Graphics 3.0 [23.05.25593.18]
[ext_oneapi_level_zero:gpu:0] Intel(R) Level-Zero, Intel(R) Arc(TM) A380 Graphics 1.3 [1.3.25593]
[ext_oneapi_cuda:gpu:0] NVIDIA CUDA BACKEND, NVIDIA GeForce RTX 3060 0.0 [CUDA 12.1]
$ export ONEAPI_DEVICE_SELECTOR="cuda:*"
$ mandel
    Platform Name: NVIDIA CUDA BACKEND
  Platform Version: CUDA 12.1
      Device Name: NVIDIA GeForce RTX 3060
   Max Work Group: 1024
 Max Compute Units: 28

Parallel Mandelbrot set using buffers.
 Rendered image output to file: mandelbrot.png (output too large to display in text)
      Serial time: 0.557785s
    Parallel time: 0.00533454s
Successfully computed Mandelbrot set.
$
```



ディスクリート・グラフィックス利用時の注意点

インテル® ARC™ A380/750/770 などのディスクリート・グラフィックス・カードを
システムに搭載すると、「BIOS で Re-Size BAR が有効になっていない」という
警告がインテル® Arc™ コントロールから出力されることがあります

```
    Platform Name: Intel(R) OpenCL HD Graphics
  Platform Version: OpenCL 3.0
       Device Name: Intel(R) Arc(TM) A380 Graphics
    Max Work Group: 1024
 Max Compute Units: 128

Parallel Mandelbrot set using buffers.
 Rendered image output to file: mandelbrot.png (output too large to display in text)
      Serial time: 0.332653s
    Parallel time: 0.00508133s
Successfully computed Mandelbrot set.
```

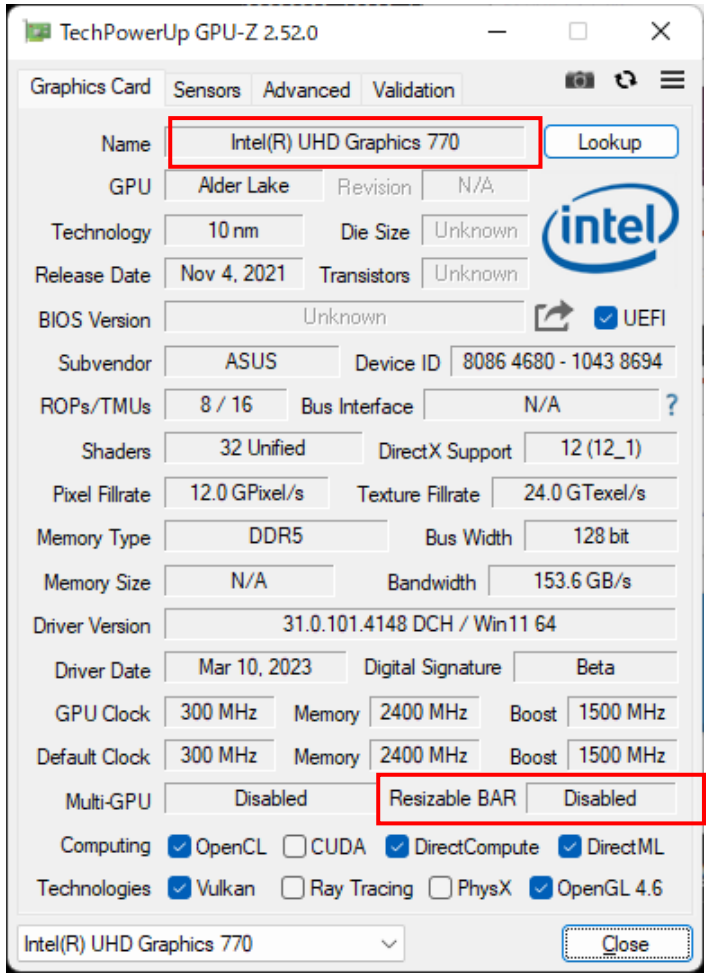
BAR 有効

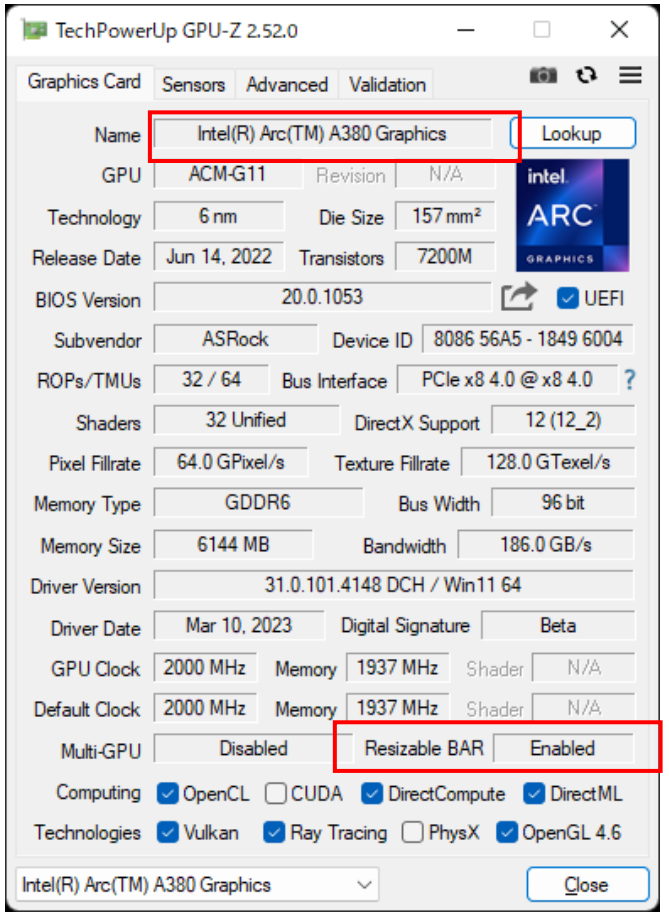
BAR 無効

```
    Platform Name: Intel(R) OpenCL HD Graphics
  Platform Version: OpenCL 3.0
       Device Name: Intel(R) Arc(TM) A380 Graphics
    Max Work Group: 1024
 Max Compute Units: 128

Parallel Mandelbrot set using buffers.
 Rendered image output to file: mandelbrot.png (output too large to display in text)
      Serial time: 0.337012s
    Parallel time: 0.00928928s
Successfully computed Mandelbrot set.
```

Re-Size BAR の設定を確認





統合グラフィックスでは、
この設定は関連しません

複数のグラフィックス・
デバイスを搭載する場合、
ディスクリート・グラフィッ
クス・デバイスであることを
確認してください

まとめ

- すでにマルチスレッド化の経験があれば、ヘテロジニアス・コンピューティングの導入は容易です
- 利用可能なデバイスがすでに皆さんのシステムには搭載されているかもしれません
- 開発ツールはすでに用意されています

オンデマンド・ウェビナーの紹介

- インテル® oneAPI ツールキットのアップデート ~ バージョン 2023 の注目機能と他社製 GPU 向けプラグインの紹介 ~
- OpenMP* を使用した GPU オフロード方法
- GPU 向けのインテル® VTune™ プロファイラーの機能と GPU 最適化
- インテル® DPC++ 互換性ツールの紹介

https://www.isus.jp/products/oneapi/webinar/

参考資料

- oneAPI for NVIDIA* GPU 2023.0.0 ガイド
- ベータ版 oneAPI for AMD* GPU 2023.0.0 ガイド
- oneAPI で NVIDIA と AMD のサポートを実現
- oneAPI DPC++ 導入ガイド
- oneAPI GPU 最適化ガイド日本語版
- oneAPI DPC++ コンパイラーとランタイム・アーキテクチャーの設計 (英語)