HPC コードにおける SYCL* の利用事例: OpenFDTD_SYCL

CUDA* から SYCL* への移行

エクセルソフト株式会社

2025/09/11

このセッションの内容

1. SYCL* の利用事例: OpenFDTD_SYCL

2. CUDA* から SYCL* への移行

- CUDA* [1]
  NVIDIA による、GPU を用いて計算を加速させるための開発環境およびツールキット
- SYCL* [2]
  Khronos Group が標準仕様を管理し公表している、GPU などの
  アクセラレーター・デバイスを利用するプログラムのための C++ API

参考文献スライドへのリンク

前提: インテル® ソフトウェア開発ツール

- インテル® oneAPI ベース・ツールキット、バージョン 2025.2 以降
  - インテル® oneAPI DPC++/C++ コンパイラー (icpx)
  - インテル® DPC++ 互換性ツール (dpct)

Windows* 版は、Microsoft* Visual Studio* による
「C++ によるデスクトップ開発」ワークロードのインストールが事前要件

Linux* 版は、ユーザー権限でもインストール可能 (${HOME}/intel/oneapi 以下に配置)

製品紹介ページ: エクセルソフト株式会社
https://www.xlsoft.com/jp/products/intel/oneapi/index.html?tab=1

1. SYCL* の利用事例: OpenFDTD_SYCL

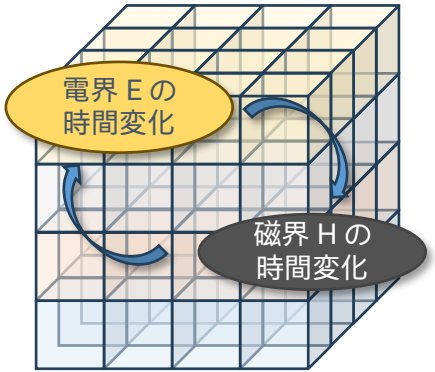
- 電磁界シミュレーション･プログラム OpenFDTD [3] を基に、異なる著者によって独自の変更が施されたプログラム

- 変更点: CUDA* による並列化部分を SYCL* で同様に記述し、インテルの GPU および他の対応デバイスでも実行可能とした
  - 著者による、本作業に関する発表 → 参考文献 [4][5][6][7]
  - ソースコードの公開ページ:
    https://github.com/mitsuboh/OpenFDTD_SYCL (英語)

OpenFDTD の処理内容

※ 詳細は参考文献 [1] を参照

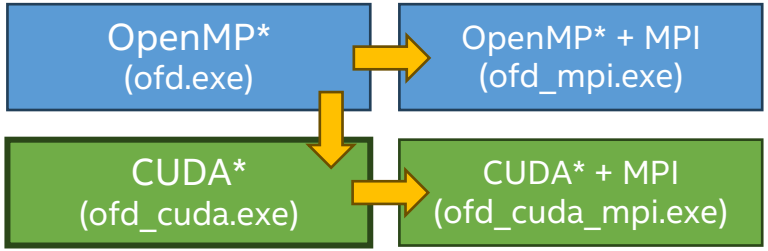
- 主な計算



空間を小さなセルの集合とみなし、各セル位置で
1. 時間ごとに x、y、z 成分の電界と磁界の値を更新
2. 時間ごとに各計算値のフーリエ変換を積算

→ 十分な数の時間ステップを計算することで周波数特性を得る

- 高速化対応



いくつかの入力パラメーター (実行の前提条件) をベンチマークとして、CPU 向け、GPU 向けそれぞれの並列処理による高速化を施したコードを含む

→ 入力パラメーターは data/benchmark 内の各ファイルで表現

OpenFDTD_SYCL のビルドから実行まで (1)

- Git コマンドでソースコードを取得

```
git clone https://github.com/mitsuboh/OpenFDTD_SYCL
cd OpenFDTD_SYCL
git checkout noshm
```

- NVIDIA の GPU で実行させるためには、Makefile 内のコンパイラー・オプション指定に追記が必要 (詳細は後述)

  sol_sycl_dm/Makefile_icpx_win または sol_sycl_dm/Makefile_icpx_linux について、-fsycl の指定を次のように置き換え

```
-fsycl -fsycl-targets=spir64,nvidia_gpu_sm_75
```

OpenFDTD_SYCL のビルドから実行まで (2)

Windows*、CMD

```
:: 環境変数の設定
cmd /k "C:¥Program Files (x86)¥Intel¥oneAPI¥setvars.bat"

:: プログラムのビルド
pushd sol_sycl_dm
nmake -f Makefile_icpx_win clean
nmake -f Makefile_icpx_win
popd

:: 計算の実行
.¥ofd_sycl.exe .¥data¥benchmark¥benchmark100.ofd

:: ofd.log, ofd.out, feed.log ファイルが生成される
:: - ofd.log: 入力設定値や実行中のメッセージなど
:: - ofd.out: ポスト処理に提供するデータ（計算結果）
```

Linux*、bash

```
# 環境変数の設定
source /opt/intel/oneapi/setvars.sh
# source $HOME/intel/oneapi/setvars.sh

# プログラムのビルド
make -C sol_sycl_dm -f Makefile_icpx_linux clean
make -C sol_sycl_dm -f Makefile_icpx_linux


# 計算の実行
./ofd_sycl ./data/benchmark/benchmark100.ofd

# ofd.log, ofd.out, feed.log ファイルが生成される
# - ofd.log: 入力設定値や実行中のメッセージなど
# - ofd.out: ポスト処理に提供するデータ（計算結果）
```

SYCL* のデバイス指定

- SYCL* の仕様では、使用されるアクセラレーター・デバイスは実行時に有効なデバイスの中から適当に選択される
  - 基本的に GPU が優先、次点で CPU
    - プログラム中で sycl::aspect (特徴) に基づいて選択できる
  - ONEAPI_DEVICE_SELECTOR 環境変数で有効なデバイスの範囲を特定できる

ONEAPI_DEVICE_SELECTOR 環境変数は、現状では intel/llvm の実装に固有の機能
https://github.com/intel/llvm/blob/v6.2.0/sycl/doc/EnvironmentVariables.md#oneapi_device_selector (英語)

ONEAPI_DEVICE_SELECTOR 設定例

Windows*、CMD

```
:: デバイスは NVIDIA の GPU、無ければいずれかの CPU
set ONEAPI_DEVICE_SELECTOR=*:cpu;cuda:gpu
.¥ofd_sycl.exe .¥data¥benchmark¥benchmark100.ofd

:: デバイスはインテルの GPU、無ければいずれかの CPU
set set ONEAPI_DEVICE_SELECTOR=*:cpu;level_zero:gpu
.¥ofd_sycl.exe .¥data¥benchmark¥benchmark100.ofd

:: デバイスは OpenCL* ランタイムに基づく CPU のみ
set ONEAPI_DEVICE_SELECTOR=opencl:cpu
.¥ofd_sycl.exe .¥data¥benchmark¥benchmark100.ofd
```

Linux*、bash

```
# デバイスは NVIDIA の GPU、無ければいずれかの CPU
export ONEAPI_DEVICE_SELECTOR="*:cpu;cuda:gpu"
./ofd_sycl ./data/benchmark/benchmark100.ofd

# デバイスはインテルの GPU、無ければいずれかの CPU
export ONEAPI_DEVICE_SELECTOR="*:cpu;level_zero:gpu"
./ofd_sycl ./data/benchmark/benchmark100.ofd

# デバイスは OpenCL* ランタイムに基づく CPU のみ
export ONEAPI_DEVICE_SELECTOR=opencl:cpu
./ofd_sycl ./data/benchmark/benchmark100.ofd
```

NVIDIA の GPU 向け動作要件 (1)

1. コンパイル時のターゲットに追加

実行させる NVIDIA の GPU の世代
(コンピュート・ケイパビリティー) に合わせる

```
icpx -fsycl -fsycl-targets=spir64,nvptx64-nvidia-cuda ...
       -Xsycl-target-backend=nvptx64-nvidia-cuda --cuda-gpu-arch=sm_50 ...
```

```
icpx -fsycl -fsycl-targets=spir64,nvidia_gpu_sm_50 ...
```

省略形

-fsycl-targets オプションの引数リスト (インテル、NVIDIA、AMD のすべてパターン)
https://github.com/intel/llvm/blob/sycl-rel-6_2/sycl/doc/UsersManual.md (英語)

NVIDIA の GPU 向け動作要件 (2)

2. Codeplay によるプラグイン (アダプター・ライブラリー)
   - インテル® oneAPI DPC++/C++ コンパイラー (icpx) の
     ランタイム・ライブラリーに追加する必要がある
     - ur_adapter_cuda.dll (Windows*)、libur_adapter_cuda.so (Linux*)

配布ページ (Codeplay) へのリンク (英語)

3. CUDA* プログラムの実行要件である
   NVIDIA の GPU 用グラフィックス・ドライバー

配布ページ (NVIDIA) へのリンク

他のコンピューター上での実行要件 (Windows*)

- インテル® oneAPI DPC++/C++ コンパイラー (icpx) による SYCL* のランタイム・ライブラリー

配布ページ (インテル) へのリンク (英語)

以下の場所にインストールされ、システム環境変数 (PATH) に追加される
C:¥Program Files (x86)¥Common Files¥intel¥Shared Libraries¥bin

バージョン 2025.x の時点では libumf のコピーも必要となる
C:¥Program Files (x86)¥Intel¥oneAPI¥umf¥latest¥bin¥umf.dll

- Microsoft* Visual C++* 再頒布可能パッケージ (x64)

配布ページ (Microsoft) へのリンク

- インテルの GPU で実行させる場合は、インテルの GPU 用グラフィックス・ドライバー

配布ページ (インテル) へのリンク

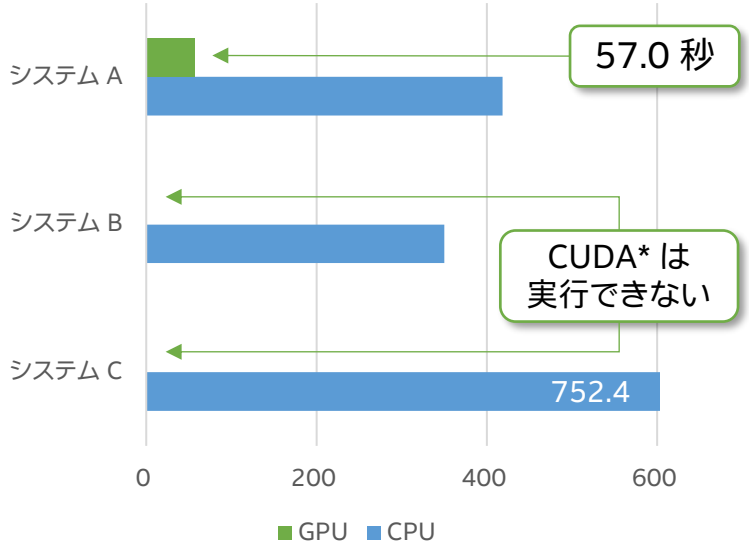
実行するシステム

	システム A	システム B	システム C
GPU	NVIDIA* GeForce* GTX 1660 Ti (拡張カード、専有メモリー 6GB)	インテル® Arc™ グラフィックス (プロセッサーに統合)	インテル® Arc™ 140V GPU (プロセッサーに統合)
GPU ドライバー	32.0.15.8097 (2025/08/07) Studio ドライバー 580.97	32.0.101.7026 (2025/08/19)	32.0.101.6881 (2025/06/04)
CPU	インテル® Core™ i5-12600K プロセッサー	インテル® Core™ Ultra 7 プロセッサー 155H	インテル® Core™ Ultra 7 プロセッサー 258V
CPU コア数	10 コア、16 スレッド	16 コア、22 スレッド	8 コア、8 スレッド
メインメモリー	DDR4 3200MT/s、32GB (16GB x2)	DDR5 6400MT/s、32GB (16GB x2)	LPDDR5X 8533MT/s、32GB
OS	Windows* 11 24H2		
システム形態	デスクトップ PC	ノートブック PC	ハンドヘルド PC

実行結果 (benchmark300.ofd、単位: 秒)

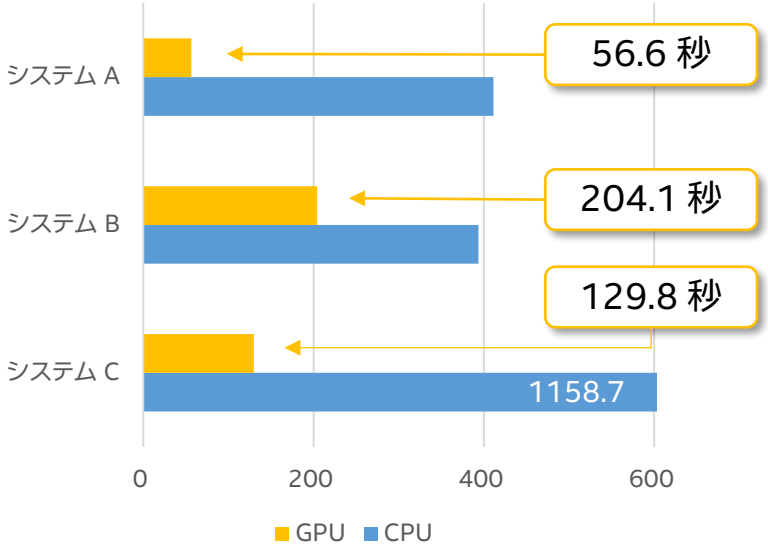
メモリー使用量は
2GB ほど

- OpenFDTD Version 4.3.0 (2025/03/29)



※ 各 CPU はコア数に等しいスレッド数 (-n オプション) で実行

- OpenFDTD_SYCL



※ 各 CPU は SYCL* で CPU デバイスを選択して実行

2. CUDA* から SYCL* への移行

- なぜ SYCL* を使うか
  - CUDA* は NVIDIA の GPU に基づくシステムに専用の言語とツール

参照: SYCL* を使用すべき理由 (2022年)

- CUDA* を基にする理由
  - アクセラレーター・プログラミングにおける事実上の標準手法
    - CUDA* ツールキットの初版公開は 2007年 [8]
    - 現行の SYCL* 仕様とそのコンパイラー実装は 2021年から [9]

参考文献スライドへのリンク

インテル® DPC++ 互換性ツール (dpct)

- ソースコード変換ツール
  - 入力: CUDA* C++ ソースコード (*.cu および *.cuh)
  - 出力: 対応する SYCL* への置き換え、および関連するコメントを加えた C++ ソースコード (デフォルトは *.dp.cpp および *.dp.hpp)

ファイルの拡張子としては単に *.cpp や *.hpp

- 変換コマンド例 (結果は次のスライド)

```
dpct --out-root c2s_output --no-dpcpp-extensions=all sample.cu
```

サンプルコードの出典 (英語)
https://www.intel.com/content/www/us/en/developer/tools/oneapi/dpc-compatibility-tool.html

sample.cu > main()

変換前 (*.cu)

```cuda
#include <cuda.h>
#include <stdio.h>

const int vector_size = 256;

__global__ void SimpleAddKernel(float *A, int offset)
{
    A[threadIdx.x] = threadIdx.x + offset;
}

int main()
{
    float *d_A;
    int offset = 10000;

    cudaMalloc(&d_A, vector_size * sizeof(float));
    SimpleAddKernel<<<1, vector_size>>>(d_A, offset);

    float result[vector_size] = {};
    cudaMemcpy(result, d_A, vector_size * sizeof(float)
        , cudaMemcpyDeviceToHost);

    cudaFree(d_A);

    for (int i = 0; i < vector_size; ++i) {
        if (i % 8 == 0) printf("\n");
        printf("%.1f ", result[i]);
    }

    return 0;
}
```

コードエディター: Visual Studio Code

c2s_output > sample.dp.cpp > main()

変換後 (*.dp.cpp)

```sycl
#include <sycl/sycl.hpp>
#include <dpct/dpct.hpp>
#include <stdio.h>

const int vector_size = 256;

void SimpleAddKernel(float *A, int offset, const sycl::nd_item<3> &item_ct1)
{
    A[item_ct1.get_local_id(2)] = item_ct1.get_local_id(2) + offset;
}

int main()
{
    dpct::device_ext &dev_ct1 = dpct::get_current_device();
    sycl::queue &q_ct1 = dev_ct1.in_order_queue();
    float *d_A;
    int offset = 10000;

    d_A = sycl::malloc_device<float>(vector_size, q_ct1);
    q_ct1.parallel_for(sycl::nd_range<3>(sycl::range<3>(1, 1, vector_size),
                                         sycl::range<3>(1, 1, vector_size)),
                       [=](sycl::nd_item<3> item_ct1) {
                           SimpleAddKernel(d_A, offset, item_ct1);
                       });

    float result[vector_size] = {};
    q_ct1.memcpy(result, d_A, vector_size * sizeof(float)).wait();

    dpct::dpct_free(d_A, q_ct1);

    for (int i = 0; i < vector_size; ++i) {
        if (i % 8 == 0) printf("\n");
        printf("%.1f ", result[i]);
    }

    return 0;
}
```

dpct の動作

1. 与えられた CUDA* C++ コードのコンパイルを試みる
   - コンパイル時の条件 (マクロ定義など) が必要
   - CUDA* ツールキットによるヘッダーファイルが必要

2. コンパイルの中間結果を基に、ソースコードの記述を置き換える
   - CUDA* 特有の記述を、SYCL* による記述へと直接、
     またはサポート・ライブラリーを用いて置き換え
   - 置き換えできない場合は注釈 (コメント) を記述

複数ファイルやプロジェクトへの適用

- Windows*
  - Microsoft* Visual Studio* のプロジェクトから一括で変換処理
- Linux*
  - Makefile (GNU* make)、CMake*、Python* の setuptools といったビルドシステムのログからコンパイル・データベースを生成し利用
- ディレクトリー単位 (共通)

```
dpct --in-root sources --process-all --out-root dpct_output --extra-args="..."
```

sources ディレクトリー以下の *.cu ファイルが対象
(その他のファイルは単にコピーされる)

マクロ定義 (-D) やインクルード・パス (-I) などの
コンパイラー・オプションを伝える

サポート・ライブラリー dpct.hpp

- ヘッダーファイルのみのラッパー
  - SYCL* 仕様のみで記述すると冗長になるコードを簡潔に表現
    - 変換時のオプション指定により、プロジェクト内にコピーさせることも可能

```
dpct --out-root dpct_output --gen-helper-function sample.cu
```

→ dpct_output/include/dpct/dpct.hpp に配置される

- インストールされているパスは次のコマンドで表示

```
dpct --helper-function-dir
```

Windows*	C:¥Program Files (x86)¥Intel¥oneAPI¥dpcpp-ct¥2025.2¥include
Linux*	/opt/intel/oneapi/dpcpp-ct/2025.2/include

dpct.hpp の自動反映例

```cpp
#include <cuda.h>

    float *d_A;
    auto cuda_error = cudaMallocManaged(&d_A, vector_size * sizeof(float));
```

CUDA* の実行時エラーは
エラーコード (数値) が戻る

```cpp
#include <sycl/sycl.hpp>
#include <dpct/dpct.hpp>

    dpct::device_ext &dev_ct1 = dpct::get_current_device();
    sycl::queue &q_ct1 = dev_ct1.in_order_queue();

    float *d_A;
    auto cuda_error =
        DPCT_CHECK_ERROR(d_A = sycl::malloc_shared<float>(vector_size, q_ct1));
```

SYCL* デバイスおよびキューのインスタンスを
グローバルなスレッドセーフ・シングルトンで管理

SYCL* の実行時エラーは
C++ 例外処理 (try-catch)
→ マクロで包む

cudaMallocManaged と sycl::malloc_shared は、意味としては同等 (ホストとデバイスの
どちらからもアクセス可能なメモリー領域の確保) であるが、性能面でふるまいが異なる、参考文献 [10]

SYCL* 拡張

- SYCL* 2020 仕様はすべての CUDA* 機能と一対一に対応しない
  - 処理系ごとの拡張仕様を導入する方法を定めている
- dpct は、変換にあたり拡張仕様を採用するか指定できる

拡張を使用しない	`dpct --out-root c2s_output --no-dpcpp-extensions=all sample.cu`
拡張を採用する (デフォルト)	`dpct --out-root dpct_output sample.cu`

  - 実験的な機能や SYCL* 拡張の使用も許可する場合

```
dpct --out-root dpct_output --use-experimental-features=all sample.cu
```

dpct による変換時の SYCL* 拡張

```
__global__ void SimpleAddKernel(float *A, int offset)
{
    A[threadIdx.x] = threadIdx.x + offset;
}
```

CUDA* のカーネル関数
インデックス指定は
固有のグローバル変数

SYCL* 標準内の方法では
引数の追加が必要

```
void SimpleAddKernel(float *A, int offset, const sycl::nd_item<3> &item_ct1)
{
    A[item_ct1.get_local_id(2)] = item_ct1.get_local_id(2) + offset;
}
```

SYCL* 拡張を採用

```
void SimpleAddKernel(float *A, int offset)
{
    auto item_ct1 = sycl::ext::oneapi::this_work_item::get_nd_item<3>();
    A[item_ct1.get_local_id(2)] = item_ct1.get_local_id(2) + offset;
}
```

機能識別マクロ: SYCL_EXT_ONEAPI_FREE_FUNCTION_QUERIES、この拡張仕様のドキュメント (英語)

CUDA* 関連ライブラリーとの対応

NVIDIA* CUDA* ツールキット	インテル® ソフトウェア開発ツール (インテル® DPC++ 互換性ツールによる置き換え先)
CUDA* Core Compute Libraries (Thrust、CUB、libcudacxx/libcu++)	インテル® oneAPI DPC++ ライブラリー (oneDPL)
cuBLAS、cuFFT、cuRAND、 cuSOLVER、cuSPARSE	インテル® oneAPI マス・カーネル・ライブラリー (oneMKL)
cuDNN	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 (oneDNN)
NCCL	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 (oneCCL)

※ ライブラリーごとに相互運用性は異なる

dpct の関連ドキュメント

- インテル® DPC++ 互換性ツール (インテル® DPCT) 2025.0.0 デベロッパー・ガイドおよびリファレンス (iSUS による日本語参考訳)

  https://www.isus.jp/wp-content/uploads/dpct/2025/user_guide/index.html

  原文 (英語)
- インテル® DPC++ 互換性ツール 2025.2 リリースノート

  https://www.xlsoft.com/jp/products/intel/tech/release-notes/intel-dpcpp-compatibility-tool-release-notes-2025.2.html

- oneAPI for NVIDIA* GPU 2025.2.0 ガイド (iSUS による日本語参考訳)

  https://www.isus.jp/products/oneapi/oneapi-for-nvidia-gpu-get-started/
- oneAPI GPU 最適化ガイド 2025.2 (iSUS による日本語参考訳)

  https://www.isus.jp/wp-content/uploads/pdf/oneapi-gpu-optimization-guide_JA.pdf

まとめ

1. SYCL* の利用事例: OpenFDTD_SYCL
   - CUDA* による並列化の、SYCL* による並列化への移行を確認
   - 同じソースコードから NVIDIA とインテル、それぞれの GPU を利用可能

2. CUDA* から SYCL* への移行
   - インテル® DPC++ 互換性ツールによって、移行の手作業を削減可能

補足: オープンソース・プロジェクトとの関係

各プロジェクトは github.com で公開されている

Clang などを提供

llvm/llvm-project LLVM コンパイラー・インフラストラクチャー

定期的に更新点を反映

intel/llvm
インテル、Codeplay による SYCL* 対応
コンパイラーとランタイムのプロジェクト

→ インテルが製品版としてビルド、サポート

インテル® oneAPI DPC++/C++ コンパイラー

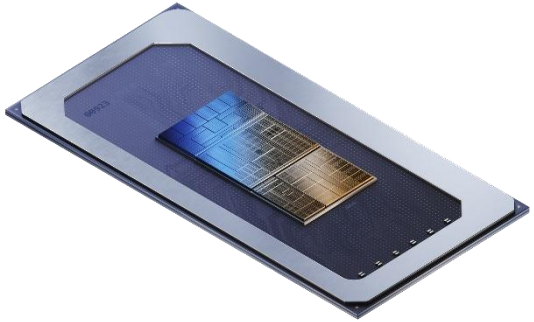
定期的に更新点を反映

oneapi-src/SYCLomatic
SYCL* への変換ツールのプロジェクト

→ インテルが製品版としてビルド、サポート

インテル® DPC++ 互換性ツール

補足: インテルの GPU 製品 (XPU)



プロセッサー統合グラフィックス

インテル® Core™ Ultra プロセッサー (シリーズ 1)
100H (開発コード名 Meteor Lake)

インテル® Core™ Ultra モバイル・プロセッサー (シリーズ 2)
200V (開発コード名 Luner Lake)
200H (開発コード名 Arrow Lake-H)

単体グラフィックス (拡張カード)

インテル® データセンター GPU マックス・シリーズ

インテル® Arc™ A シリーズ・グラフィックス

インテル® Arc™ B シリーズ・グラフィックス

参考文献

- [1] NVIDIA, "CUDA Toolkit", https://developer.nvidia.com/cuda-toolkit/
- [2] Khronos Group, "SYCL", https://www.khronos.org/sycl/
- [3] "OpenFDTD 時間領域差分法電磁界シミュレーター", https://ss023804.stars.ne.jp/OpenFDTD
- [4] 池井 満, "FDTD法電磁界シミュミレータのSYCLによるGPUオフロードの検討", 情報処理学会 研究報告ハイパフォーマンスコンピューティング(HPC), 2023-HPC-188, 15, p. 1-9, 2023-03-09, ISSN, 2188-8841, https://ipsj.ixsq.nii.ac.jp/records/225159 (情報処理学会電子図書館)
- [5] 池井 満, "OpenFDTDのSYCL によるGPUオフロードの検討", 情報処理学会 研究報告ハイパフォーマンスコンピューティング(HPC), 2023-HPC-192, 33, p. 1-7, 2023-11-28, ISSN, 2188-8841, https://ipsj.ixsq.nii.ac.jp/records/231111 (情報処理学会電子図書館)
- [6] 池井 満, "科学技術計算の SYCL* + MPI を用いたダイレクト・プログラミング手法", ハイパフォーマンス・ソフトウェア・カンファレンス 2024, 技術セッション 2, 2024-06-13, HEIPSxkPa_I, https://youtu.be/HEIPSxkPa_I?si=iri8E8nviYrmHkns (YouTube、オンデマンド配信動画)
- [7] 池井 満, "OpenFDTDのSYCLによるオフロード", 2025-03-17, https://github.com/mitsuboh/OpenFDTD_SYCL/wiki (GitHub)
- [8] NVIDIA, "CUDA Toolkit Archive", https://developer.nvidia.com/cuda-toolkit-archive
- [9] Khronos Group, "Khronos Releases SYCL 2020 Specification", 2021-02-09, https://www.khronos.org/news/press/khronos-releases-sycl-2020-final-specification
- [10] Codeplay, " Shared (managed) USM in the CUDA backend", oneAPI for NVIDIA* GPUs 2025.2.0 Guides, https://developer.codeplay.com/products/oneapi/nvidia/2025.2.0/guides/performance/shared-managed-usm

お問い合わせはこちらまで
https://www.xlsoft.com/jp/qa