HAIRDESC
GPUプログラミング教材の紹介

下川辺 隆史
(東京大学)

GPUプログラミング教材の目的と特徴

教材の目的

- 「標準的GPU化」コースの向けた教材作成
- GPUプログラミングの基礎から、実装、プロファイラによる性能チューニングまで、実践的な知識と手法を身につけることができるプログラミング教育教材

教材の特徴

- GPUの仕組み、GPUプログラミングの概念、具体的なプログラミング方法と言語（OpenACC, OpenMP targetなど）をレベル分けし、系統的に整備
  - 一元管理さらた資料で効率的で実践的なGPUプログラミング教育
- 中核機関（筑波大、東大、科学大）、ベンダーの資料を中心に新たに整理
  - 協力機関の講習会・講義で利用している資料も参照
  - 既存の資料の単なる寄せ集めやリンク集ではなく、WGを設置し、複数人で作成・レビュー
- 具体的なコード例とソースコードの提供
- NVIDIA, AMD 両ベンダーのGPUに対応し、かつ、C/C++とFortran言語について提供、特定のGPUベンダーによらない教材作成

WGメンバー

HAIRDESC

- HAIRDESC：朴泰祐，下司雅章

中核機関

- 筑波大学：額田彰，高橋大介，藤田典久
- 東京大学：下川辺隆史，三木洋平，山崎一哉
- 東京科学大学：横田理央，遠藤敏夫，小林諒平

GPUベンダー

- AMD：石橋史康
- NVIDIA：古家真之介

想定ユーザーと全体構成

想定ユーザー

- OpenMPを用いたマルチコアCPU向けのスレッド並列について理解していることが前提
  - 将来的に、GPUプログラミングから始める方を対象とした教材追加予定

全体構成（3/25版）

初級編（初級レベル）
- GPUプログラミングの概要
- 単一GPUプログラミング (OpenACC, OpenMP target)

中級編（中級レベル）
- マルチGPUプログラミング (OpenACC, OpenMP target)

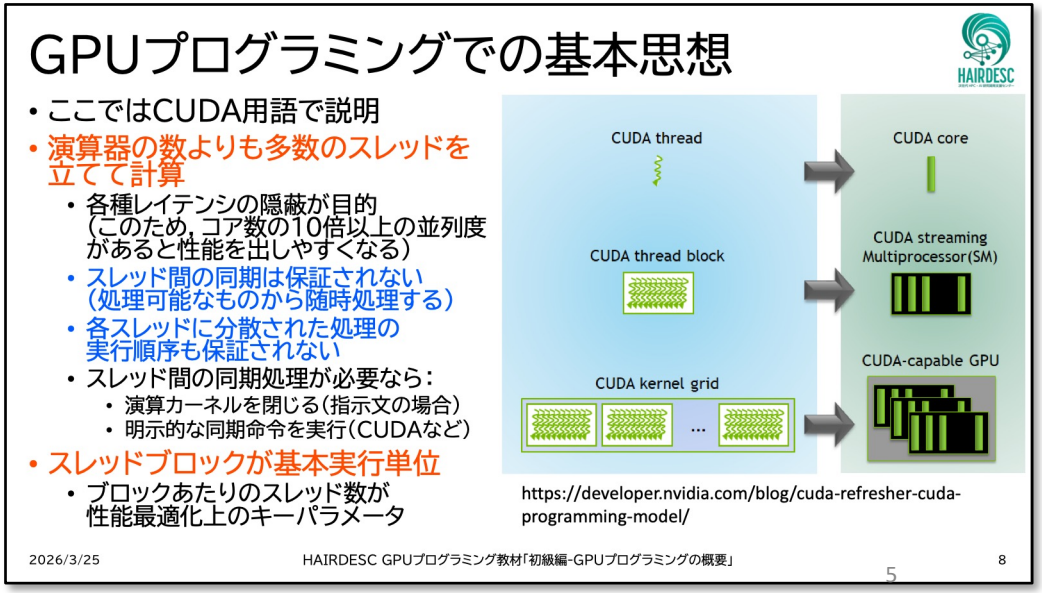
パフォーマンスチューニング編（初級レベル）
- プロファイリング（単一GPU）

初級編

「GPUプログラミングの概要」

GPUプログラミングに取り組むにあたり、GPUやGPUプログラミング手法に関する基本的な知識と、どういったアプリケーションがGPU化で高速化しやすいかを学ぶ

- GPUとは
- GPUプログラミングの基礎
  - 基本思想
  - GPU化で高速化しやすいアプリ例
- GPUプログラミング手法
  - GPUプログラミング手法の比較表



GPU化で高速化しやすいアプリ例①（C）

教材からの抜粋

- 拡散方程式コード（C言語）の抜粋
  - メモリ律速な問題の代表例

```
int main(int argc, char *argv[])
    const int nx = 512;
    const int ny = 512;
    const int nz = 512;

    float *f  = (float *)malloc(sizeof(float) * ln);
    float *fn = (float *)malloc(sizeof(float) * ln);

    for (; (icnt < nt) && (time + 0.5 * dt < 0.1); icnt++) {
        diffusion3d(nx, ny, nz, mgn, dx, dy, dz, dt, kappa, f, fn);

        swap(&f, &fn);
        time += dt;
    }

    return 0;
}
```

GPUのコア数よりも十分に大きな問題サイズ

演算をGPUに閉じ込められるので，こまめなCPU-GPU間データ転送は不要

```
void diffusion3d(int nx, int ny, int nz, int mgn, float dx, float
dy, float dz, float dt, float kappa, const float *f, float *fn)
{
#pragma omp parallel for
    for(int k = 0; k < nz; k++) {
        for (int j = 0; j < ny; j++) {
            for (int i = 0; i < nx; i++) {
                const int ix = nx * ny * (k + mgn) + nx * j + i;
                const int ip = i == nx - 1 ? ix : ix + 1;
                const int im = i == 0      ? ix : ix - 1;
                const int jp = j == ny - 1 ? ix : ix + nx;
                const int jm = j == 0      ? ix : ix - nx;
                const int kp = (k == nz - 1) ? ix : ix + nx*ny;
                const int km = (k == 0     ) ? ix : ix - nx*ny;

                fn[ix] = cc*f[ix] + ce*f[ip] + cw*f[im] +
cn*f[jp] + cs*f[jm] + ct*f[kp] + cb*f[km];
            }
        }
    }
}
```

各スレッドが共通の処理を実行，依存関係もない

教材からの抜粋

GPUプログラミング手法の比較表

- Fortran の「OK」は動作するという意味. 性能については別問題

手法	ベース言語など	強み	弱み	Fortran対応	対象GPUベンダー
CUDA C++	C++	詳細な最適化可能 最新機能が使える	記述量が多い GPU専用コード	CUDA Fortran	NVIDIA限定
HIP	C++	詳細な最適化可能 ほぼCUDA C++	記述量が多い GPU専用コード	hipfort	AMD, NVIDIA (Intel: chipStar?)
SYCL	C++	詳細な最適化可能 ラムダ式	記述量が多い ラムダ式	N/A	Intel, NVIDIA, AMD
OpenACC	指示文	移植コスト低 CPUコードと共通化可能	CUDAより遅い (メモリ律速ならそれなり?)	OK	ほぼNVIDIA限定 (HPE Cray コンパイラはAMDも対応)
OpenMP (target指示文)	指示文	移植コスト低 CPUコードと共通化可能	CUDAより遅い (メモリ律速ならそれなり?)	OK	NVIDIA, AMD, Intel
言語の標準規格	C++17以降 Fortran 2008	CPUでも同じコードが動く	多くの制約あり CUDAより遅い	Fortran 2008	NVIDIA, AMD, Intel

初級編

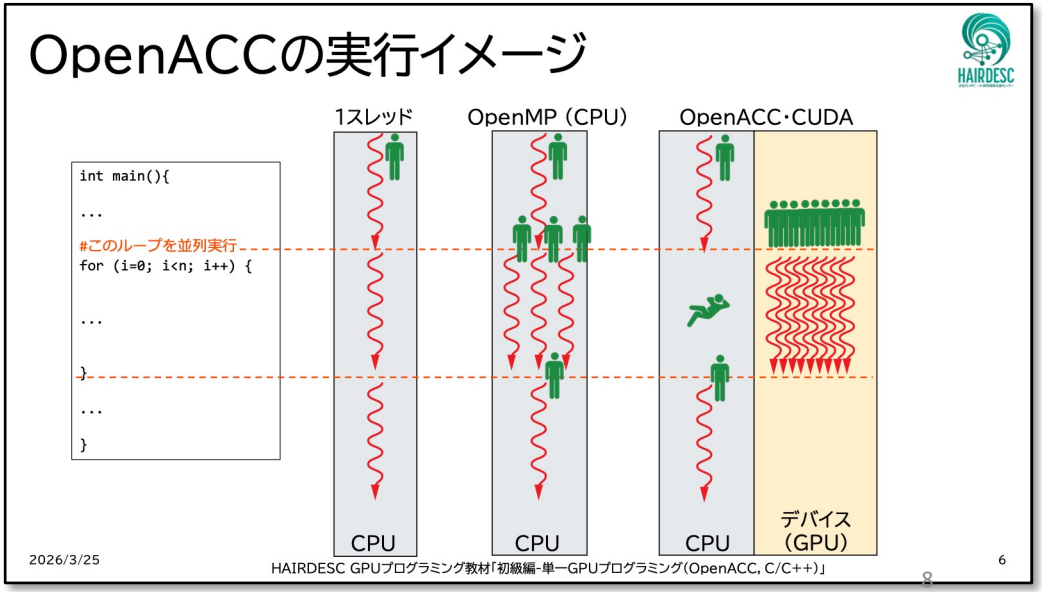
「単一GPUプログラミング」

GPU向けの指示文（OpenACCまたはOpenMP target）を用いてアプリケーションのGPU化に取り組むための基本的な知識と、GPU化方針やコンパイル方法について学ぶ

- 実行イメージ
- GPU化方針・実装方法
- コンパイル方法
- GPU移植例

教材が提供する
指示文・言語の組み合わせ

指示文	C/C++	Fortran
OpenACC	○	○
OpenMP target	○	○



教材からの抜粋

コンパイルメッセージ例(-Minfo=accel,opt)

- GPUによる高速化に成功した例

```
nvc -O3 -acc=gpu -Minfo=accel,opt -gpu=cc90 -c diffusion.c
diffusion3d:
     12, Generating present(fn[:],f[:])
     16, Loop is parallelizable
     18, Loop is parallelizable
     20, Loop is parallelizable
         Generating NVIDIA GPU code
         16, #pragma acc loop gang, vector(128) collapse(3) /*
blockIdx.x threadIdx.x */
         18,   /* blockIdx.x threadIdx.x auto-collapsed */
         20,   /* blockIdx.x threadIdx.x auto-collapsed */
```

  - 並列化可能なループであると判定
    - Loop is parallelizable が出力
  - GPU上で並列化している
    - Generating NVIDIA GPU code 以降の行が, 具体的にどうGPUにマッピングしたかの報告

- GPUによる高速化に失敗した例

```
nvc -O3 -acc=gpu -Minfo=accel,opt -gpu=cc90 -c diffusion.c
diffusion3d:
     12, Generating present(f[:])
     15, Loop carried dependence due to exposed use of f[:]
prevents parallelization
         Accelerator serial kernel generated
         Generating NVIDIA GPU code
         15, #pragma acc loop seq
         16, #pragma acc loop seq
         17, #pragma acc loop seq
     16, Loop carried dependence due to exposed use of f[:]
prevents parallelization
     17, Complex loop carried dependence of f-> prevents
parallelization
         Loop carried dependence due to exposed use of f[:]
prevents parallelization
```

  - 並列化できないループであると検知
    - Loop carried dependence…
  - GPU内で並列化できていない
    - serial kernel generated
    - #pragma acc loop seq

教材からの抜粋

N体計算のOpenACC実装例

- 演算カーネル実装の抜粋

```
static inline void calc_acc(const int32_t Ni, const float4
*const ipos, float4 *__restrict iacc, const int32_t Nj,
const float4 *const jpos, co

#pragma omp parallel for
  for (int32_t i = 0; i < Ni; i++) {
    const auto pi = ipos[i];
    float4 ai = {0.0F, 0.0F, 0.0F, 0.0F};
    for (int32_t j = 0; j < Nj; j++) {
      const auto pj = jpos[j];
      // particle-particle interaction
      const auto dx = pj.x - pi.x;
      const auto dy = pj.y - pi.y;
      const auto dz = pj.z - pi.z;
      const auto r2 = eps2 + dx * dx + dy * dy + dz * dz;
      const auto r_inv = 1.0F / std::sqrt(r2);
      const auto r2_inv = r_inv * r_inv;
      const auto alp = pj.w * r_inv * r2_inv;
      // accumulation
      ai.x += alp * dx;
      ai.y += alp * dy;
      ai.z += alp * dz;
      ai.w += alp * r2;
    }
    iacc[i] = ai;
  }
}
```

#pragma omp parallel for を置き換え

元のCPU実装

```
static inline void calc_acc(const int32_t Ni, const float4 *const ipos,
float4 *__restrict iacc, const int32_t Nj, const float4 *const jpos,
const float eps2) {
#pragma acc parallel
#pragma acc loop independent
  for (int32_t i = 0; i < Ni; i++) {
    const auto pi = ipos[i];
    float4 ai = {0.0F, 0.0F, 0.0F, 0.0F};
    for (int32_t j = 0; j < Nj; j++) {
      const auto pj = jpos[j];
      // particle-particle interaction
      const auto dx = pj.x - pi.x;
      const auto dy = pj.y - pi.y;
      const auto dz = pj.z - pi.z;
      const auto r2 = eps2 + dx * dx + dy * dy + dz * dz;
      const auto r_inv = 1.0F / std::sqrt(r2);
      const auto r2_inv = r_inv * r_inv;
      const auto alp = pj.w * r_inv * r2_inv;
      // accumulation
      ai.x += alp * dx;
      ai.y += alp * dy;
      ai.z += alp * dz;
      ai.w += alp * r2;
    }
    iacc[i] = ai;
  }
}
```

OpenACC実装

中級編

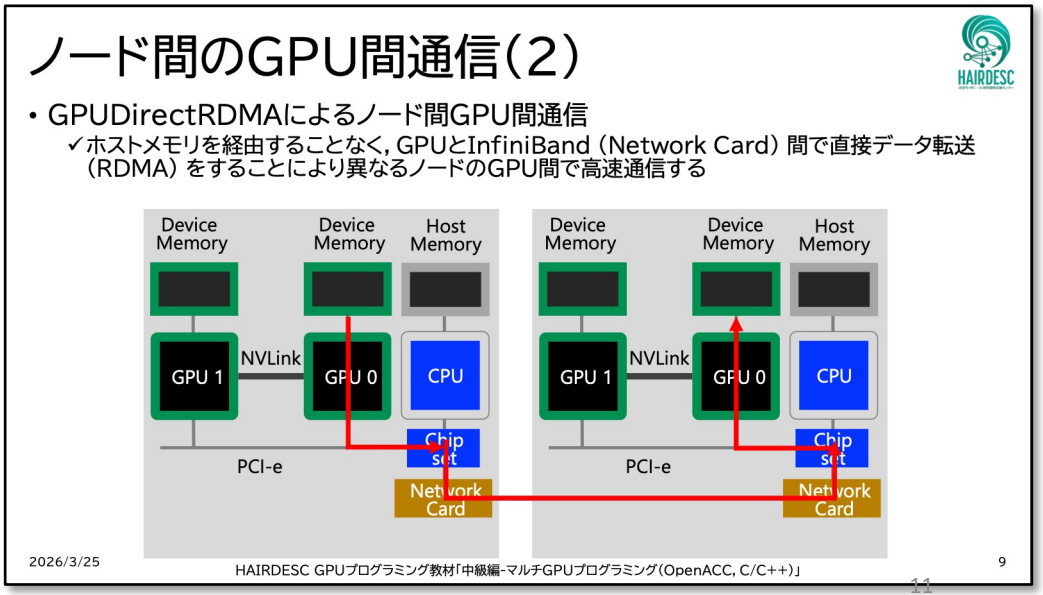
「マルチGPUプログラミング」

GPU向けの指示文（OpenACCまたはOpenMP target）を用いて単体GPU化されたアプリケーションを、MPIを用いてマルチGPU化するための実装手法について学ぶ

- 複数GPUの使用イメージ
- マルチGPU化方針・実装方法
- コンパイル・実行方法
- マルチGPU化例

教材が提供する
指示文・言語の組み合わせ

指示文	C/C++	Fortran
OpenACC	○	○
OpenMP target	○	○



教材からの抜粋

自分でCPU-GPU間のデータ転送を記述

- GPU-Aware MPI未提供の環境でも動作する方法
  - （ただしGPUスパコン上では，GPU-Aware MPIを使う方が楽で高性能）
  - 実装方法は単体GPUのときのCPU-GPU間データ転送を追加するだけ
    - MPI通信については全てCPUにお任せし，全データをCPU経由で送受信する方式

```
!$omp target enter data map(alloc: a, b)
 !$omp target teams distribute parallel do simd collapse(2)
 do j = 1, ny
    do i = 1, nx
       a(i,j) = 3.0 * rank * ny
       b(i,j) = 0.0
    end do
 end do
 dst_rank = mod((rank + 1),nprocs)
 tag      = 0
 if (rank == 0) then
    call MPI_Recv(b, w * nx, MPI_FLOAT, dst_rank, tag, MPI_COMM_WORLD, istat, ierr)
    !$omp target update to(b(:,1:w))
 else
    !$omp target update from(a(:,1:w))
    call MPI_Send(a, w * nx, MPI_FLOAT, dst_rank, tag, MPI_COMM_WORLD, ierr)
 end if
 ! 以降の実装
 !$omp target exit data map(delete: a, b)
```

MPI(H)
H→D

D→H
MPI(H)

MPI通信前後に挿入

パフォーマンスチューニング編

「プロファイリング」

性能チューニングの足がかりとなる基本的な知識と，アプリケーションが「どの程度の性能が得られている」のか、「どこまでの性能が達成可能か」を見積もるための手法を学ぶ

- 性能に関するモデル
- ベンダーが提供するGPUプロファイラツールについて
- GPUカーネルの実行時間や性能測定
- CPU-GPU間データ転送の測定
- スーパーコンピュータにおけるGPUプロファイラの使い方
- NVIDIA（公開中）, AMD（まもなく公開）

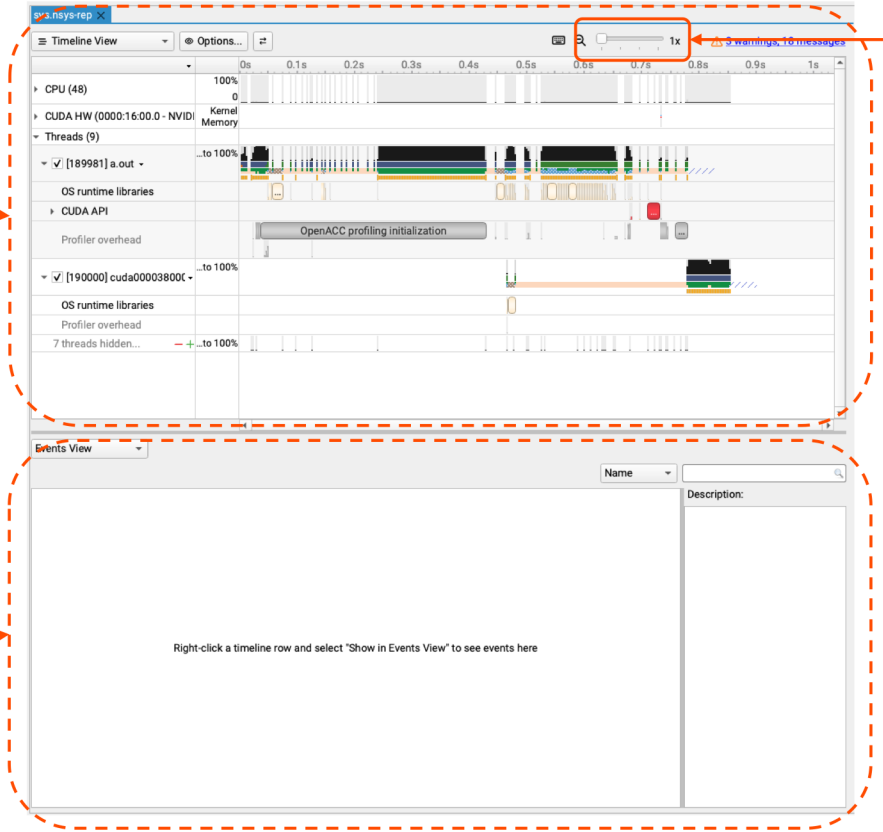
Nsight Systemsの画面例

教材からの抜粋

拡大率設定

タイムラインビュー

詳細ビュー

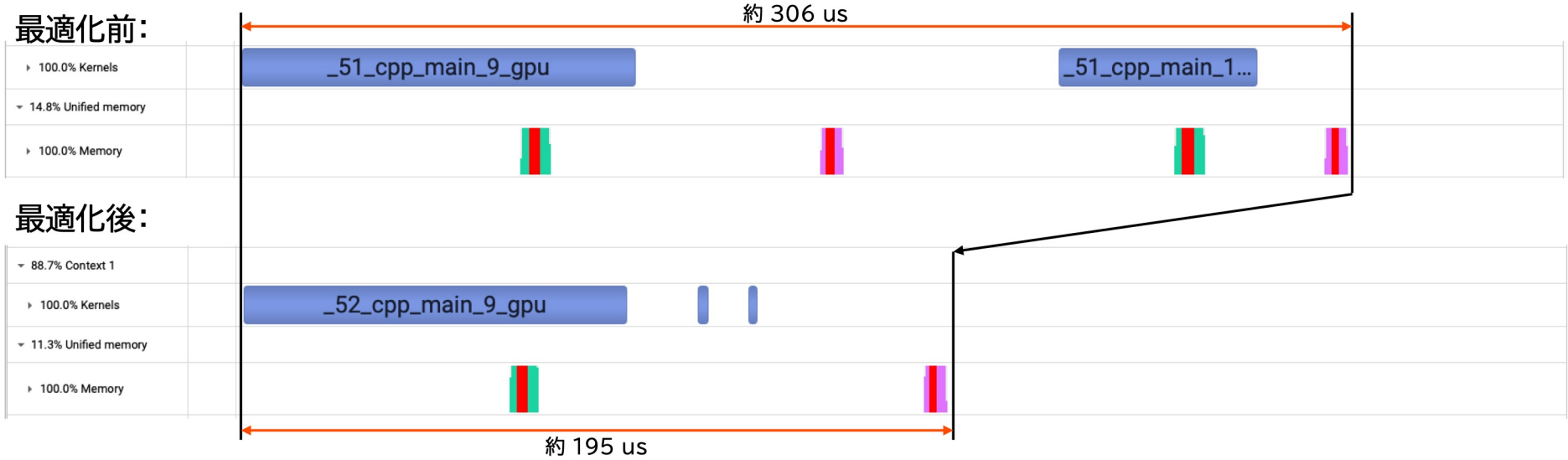


- CPUとGPUの動作がタイムラインとして表示される
- APIの呼び出し，カーネルの実行，データ転送の様子が表示される
- 拡大／縮小が可能
- 動作のおおまかな流れを把握するのに適する
- 詳細な値を確認したい時は，画面下部のビューを切り替えて見る

- 記録されたデータを主に表形式で表示
- タイムライン表示とは異なり，具体的な値を確認できる
- 左上のメニューを操作することで表示するデータを変更できる
- GPUの動作トレース，カーネルの実行時間，データ転送量や時間などの表示

計算時間の比較

教材からの抜粋



計算2がGPUで高速になったことと，データ転送が減少したことにより，計算時間が高速化された

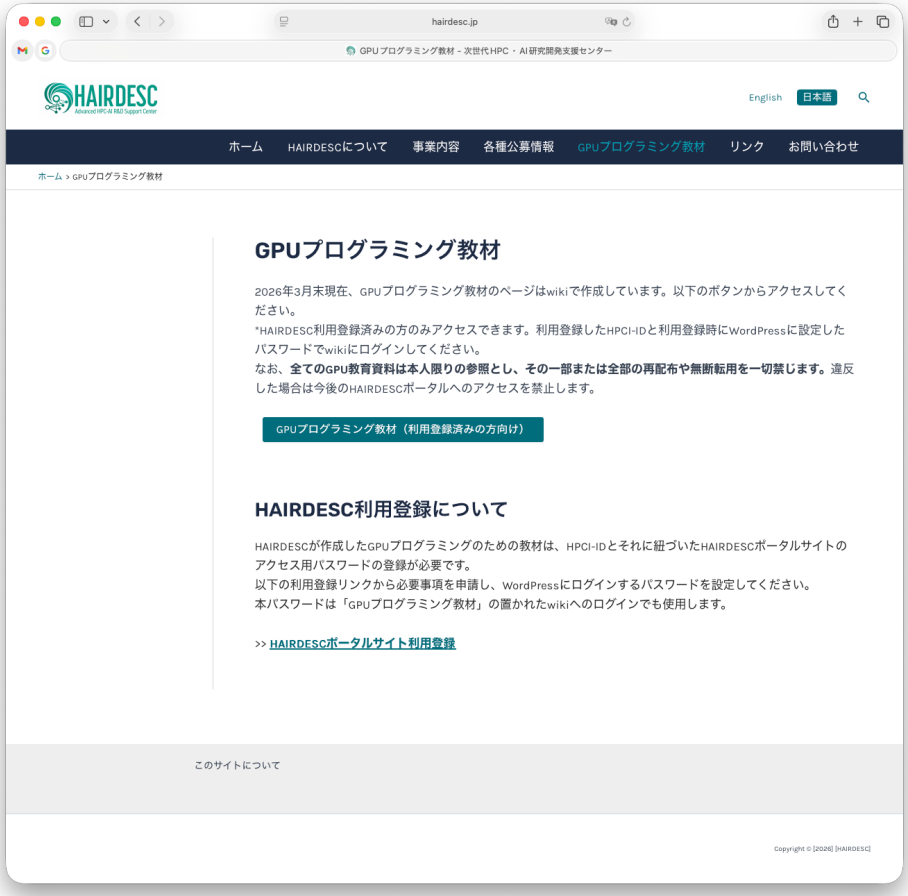
教材へのアクセス方法

教材の公開と注意事項

- 教材はHairdescサイト内Wikiで公開
- HPCI-IDが必要
- 本人限りの参照（再配布禁止）

教材へのアクセス手順

- https://hairdesc.jp/ の「GPUプログラミング教材」ページ
- HPCI-IDをHAIRDESCへ登録
- 登録完了メールに従い、パスワードを設定後に、アクセス可能



https://hairdesc.jp/gpumaterial/

来年度以降の展望

GPUプログラミング教材の充実

- 「パフォーマンスチューニング編 - プロファイリング」の完成
- ライブラリ編の作成予定
  - BLAS, FFT など
- サンプルプログラムの提供
  - 構造格子、非構造格子、N体問題
  - OpenACC, OpenMP target, 言語標準並列化、Kokkosなど
- ビデオ教材の提供
- 英語版（AI自動翻訳）の提供

GPU教育ハッカソンやチュートリアルなど実施