2024/2/5
PCCC AI/HPC OSS活用ワークショップ

GPU移行における可搬性向上に向けて

東京大学情報基盤センター /
最先端共同HPC基盤施設

塙　敏博
三木　洋平

背景

- スパコンへのGPU導入は電力当たり性能の観点で必然になりつつある
  - 最先端共同HPC基盤施設 (JCAHPC)の次期システム OFP-II:
    主力は GH200搭載 1,120ノード

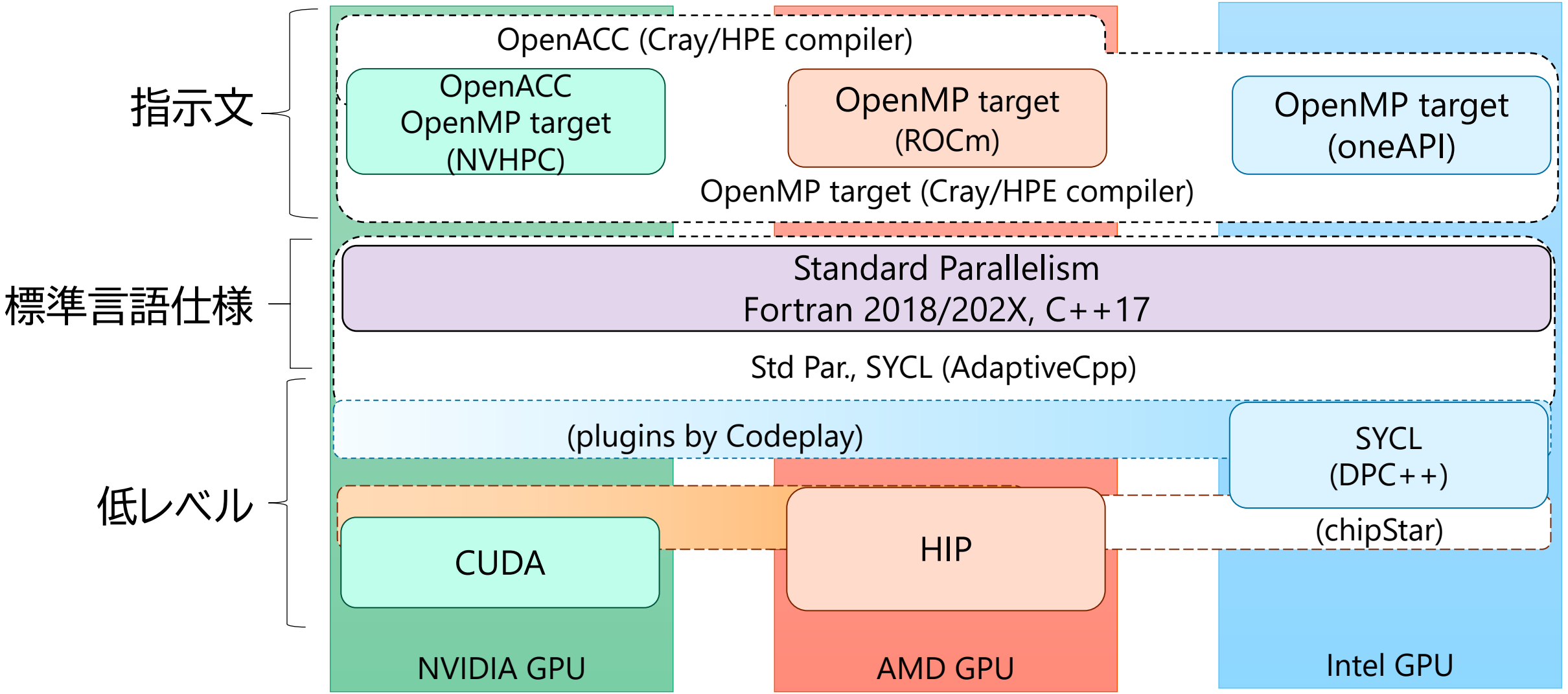
NVIDIA GH200

AMD MI250X

Intel Data Center GPU Max 1550

- 問題点
  - GPUベンダによってプログラミング環境が大きく異なる
  - 既存のMPI+OpenMPコードからの移植
    - 特にFortran
    - ファイルIOはオフロードできない、かえってオーバヘッドが増える
      - 時間発展計算において、スナップショット出力がボトルネックになりがち
      - AI処理のデータセット入力

各GPUにおける標準プログラミング環境



指示文ベースでGPU化したい場合

- OpenACC
  - GPU向けのメジャーな指示文
  - PGIがNVIDIAに買収された結果, NVIDIA色が強くなった
  - AMD, Intelは(きっと)サポートしない
    - HPE Crayコンパイラであれば, AMD GPU向けのOpenACCもサポート
- OpenMPのtarget指示文
  - OpenMP 4.5以降でアクセラレータへのオフロードがサポート
  - OpenMP 5.0で loop 指示節が追加, OpenACC的実装も可能に
  - NVIDIA, AMD, Intel 全てのGPU向けにサポートされる
  - 現時点ではOpenACCの全ての機能に対応できてはいない
- 開発者は使う指示文についても選択する必要がある

指示文を使ってGPU化するということは

- 具体的にどうGPU化するかはコンパイラ任せ
  - 細部まで指定したい人は CUDA/HIP/SYCL を使う
  - ブラックボックス的であってもGPU化されていればOK
- CUDA/HIP/SYCL で全力最適化した場合よりも性能が低いことは受け入れているユーザ層
  - 最大性能よりも, CPUコードとの互換性や移植工数削減の方を優先
  - OpenACC と OpenMP targetの性能差は CUDA/HIP/SYCL からの性能差に比べると十分に小さいはず
- つまり, OpenACC or OpenMP target？ は本来不要な選択肢
  - 何か指示文的なものを書いたらGPU化してくれれば〇, 高性能なら◎
  - ただし, 新しい指示文を作る, 新しいコンパイラを作るのはNG

解決策：プリプロセッサマクロの活用

- バックエンドで OpenACC or OpenMP target に展開
- ユーザ的にはオーバーオールのフラグ制御だけで OpenACC or OpenMP target の切り替えが可能
  - NVIDIA GPU 上では OpenACC で，AMD/Intel GPU 上では OpenMP target で動かす，ということが可能になる
- コンパイラではないので，各ベンダー製のコンパイラ性能をそのまま利用できる
  - HIP の指示文版とイメージすると良いかも
- 開発者が更新をさぼっても，自分でマクロを付け足せる
  - 新コンパイラ/新指示文の実装であれば，ユーザ側では手出しできない（開発中止・提供終了により，自分のコードが動かなくなるリスク）

実際の見え方

- OpenACCとOpenMP両方に対応した指示文の追加例(右側)
  - 場合分け(ACC for GPU, OMP for CPUなど)がかなり煩雑
  - `$ nvc++ -acc=multicore -mp=gpu …` も(見かけないが)実は可能
- マクロ版実装のコード(左側)

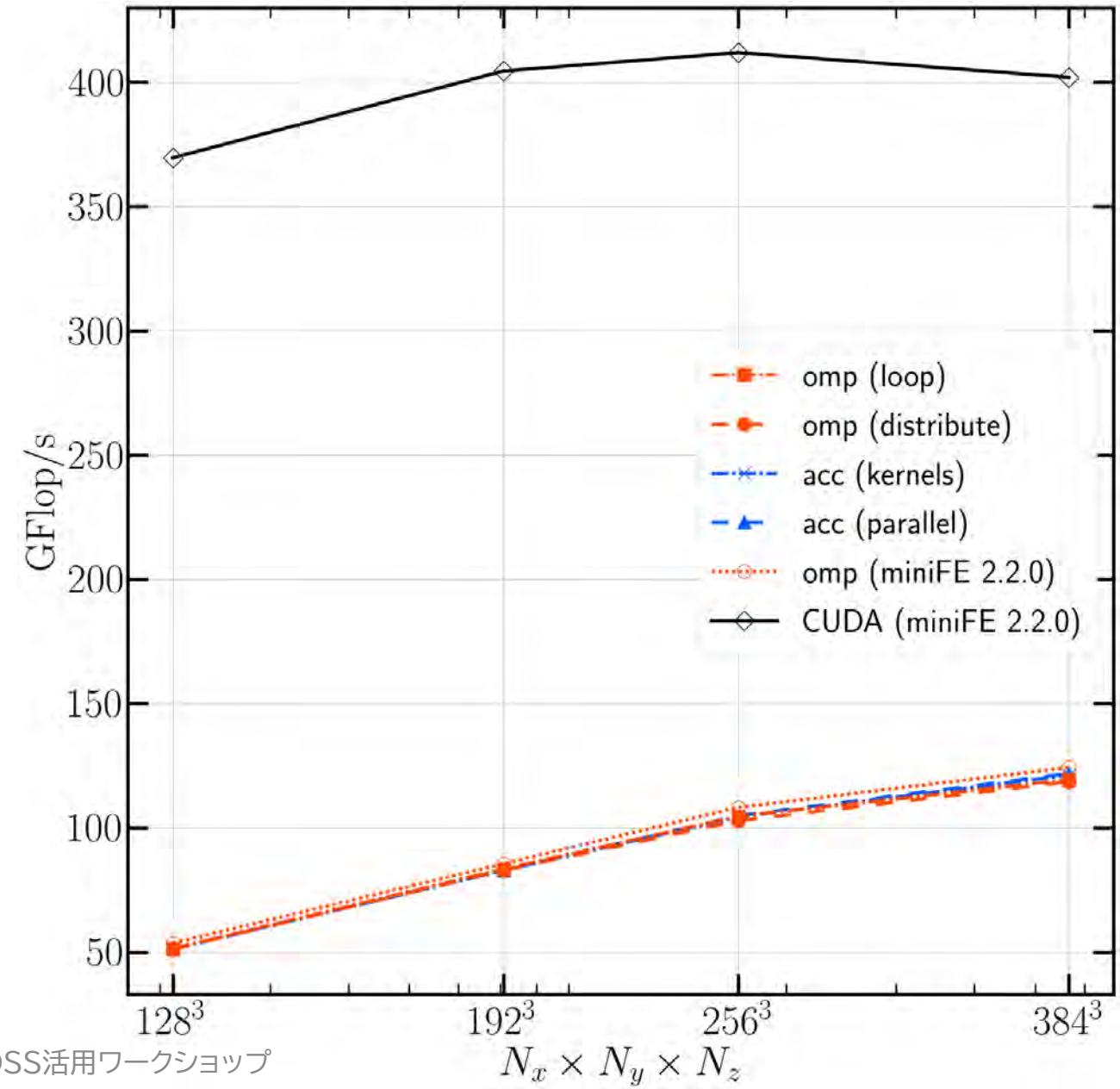
```
OFFLOAD(OPT_SIMD, OPT_NUM(NTHREADS))
for (int32_t i = 0; i < N; i++) {
```

  - 注:↑は現在開発中のマクロ名を(スライド用に)簡略化して表示
  - C/C++限定(Fortran未対応)
  - `reduction`, `collapse` も対応済
    - 現在漏れがないかチェック中
  - 見た目がかなりすっきりするため,コードの見通しも良くなる

```
#ifdef USE_OPENACC_FOR_OFFLOADING
#pragma acc kernels vector_length(NTHREADS)
#pragma acc loop independent
#endif  // USE_OPENACC_FOR_OFFLOADING
#ifdef USE_OPENMP_TARGET_FOR_OFFLOADING
#ifdef USE_OMP_LOOP
#pragma omp target teams loop simd
thread_limit(NTHREADS)
#else  // USE_OMP_LOOP
#pragma omp target teams distribute parallel
for simd thread_limit(NTHREADS)
#endif  // USE_OMP_LOOP
#endif  // USE_OPENMP_TARGET_FOR_OFFLOADING
  for (int32_t i = 0; i < N; i++) {
```

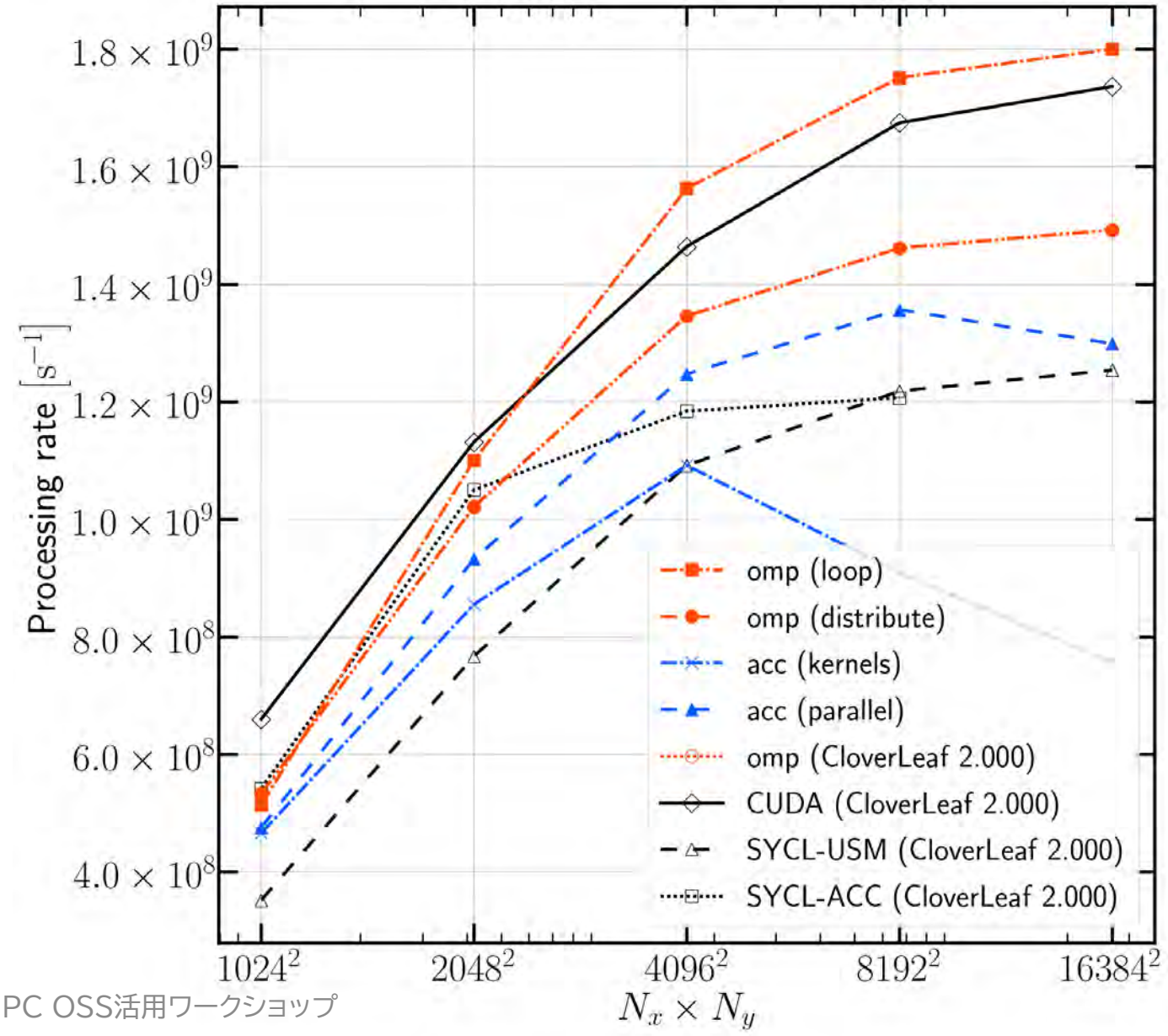
性能評価：miniFE

- ECP Proxy Application
  - https://github.com/Mantevo/miniFE
- CG法の性能をプロット
  - メモリ律速
- H100 SXM 80GB (Wisteria-Mercury)
  - CUDA 12.3, NVHPC 23.7
- OpenACC/OpenMP の差はほぼなし
  - CUDA からの性能差に比べると無視できる程度の性能差
  - `parallel/kernels`, `loop/distribute` の差も微小



性能評価：CloverLeaf(のC++版)

- 2次元圧縮性流体のミニアプリ
  - 本家は Fortran コード
    - http://uk-mac.github.io/CloverLeaf/
  - https://github.com/UoB-HPC/CloverLeaf/tree/main
    - CUDA, HIP, SYCL, Kokkos, OpenMP 4.5, ...
- H100 SXM 80GB (Wisteria-Mercury)
  - CUDA 12.3, NVHPC 23.7
  - icpx 2024.0 (for SYCL)
- OpenMP (loop) がベスト
  - CUDA より速いが, 単にCUDAの最適化が不十分なため？
  - なぜか OpenACC が遅い, ヘタる (まだ理由を調べられていない)

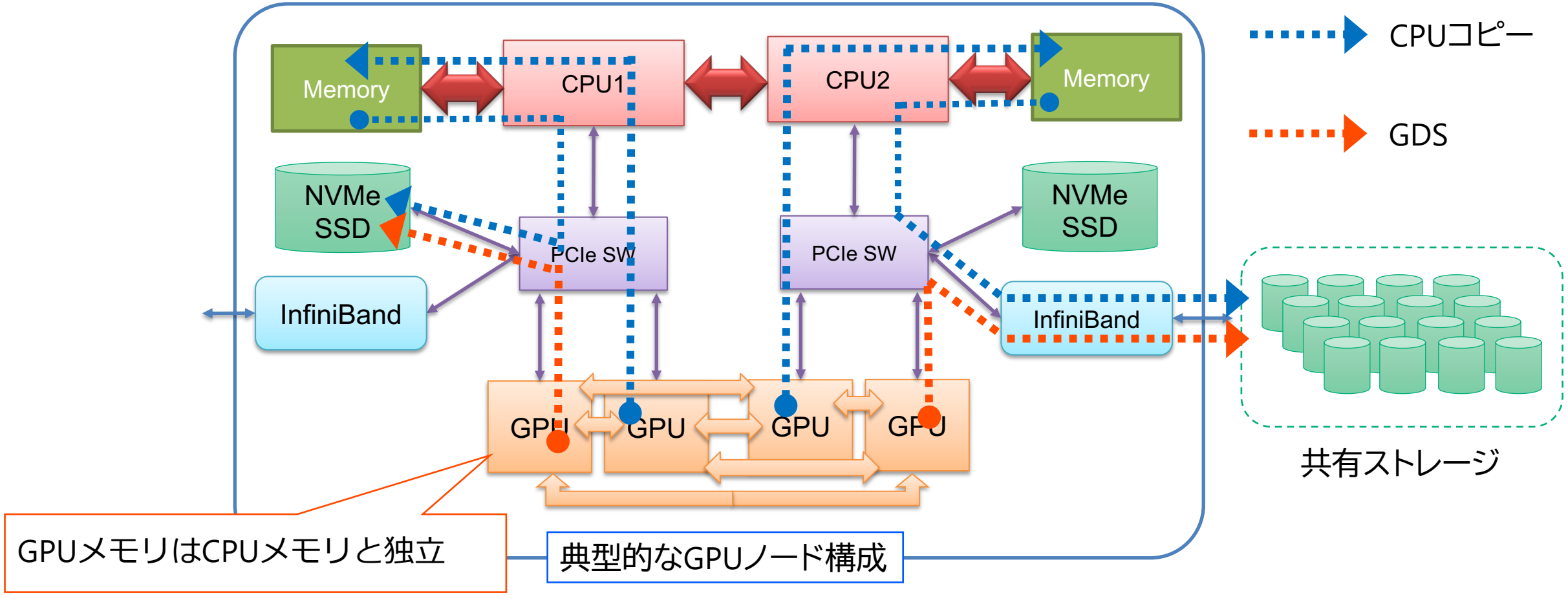


小まとめ:指示文統合マクロ

- 指示文統合マクロを開発・試験中
  - NVIDIA GPU上ではOpenACCを, AMD/Intel GPU上ではOpenMP target を選択, ということができる
  - OpenACC と OpenMP target の性能比較も簡単にできる
  - 完成後には(GitHubで？)公開する予定
- 性能評価も実施中
  - 単一ソースコードを OpenACC/OpenMP targetに切り替えて実行
  - miniFE では OpenACC/OpenMP に性能差はほぼなし
  - CloverLeaf では OpenMP (loop) がベスト
  - どちらも配布されている OpenMP target 実装をマクロ実装に書き替えたものであり, OpenACC に不利な性能評価をしているかもしれない
  - 今後AMD/Intel GPU上でのOpenMP target性能評価も実施予定

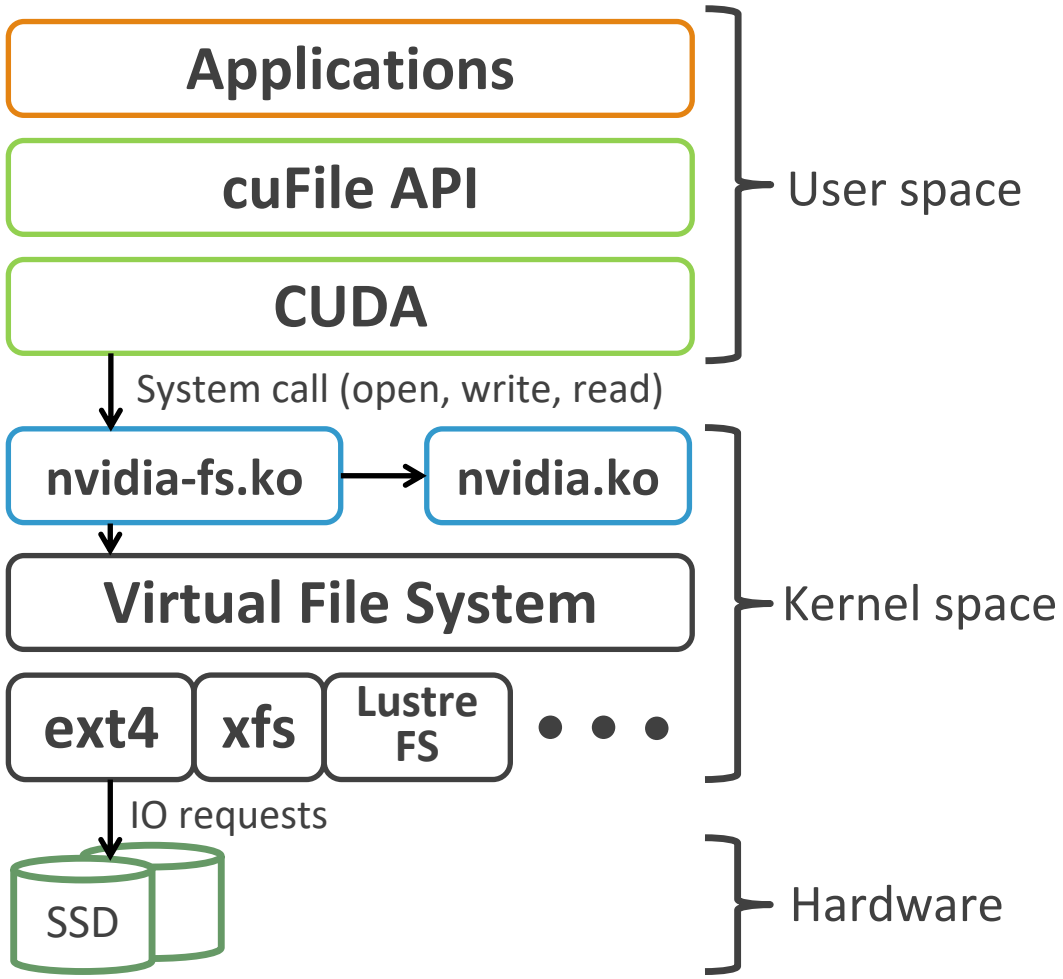
GPUにおけるファイルIO

- 従来: CPU経由のファイルIO
  ➔ GPUからの直接IO: GPU Direct Storage (**GDS**, magnum IO)
  - NVIDIAにより開発, 他社は??



GDSのソフトウェア

- **GDSが必要とするカーネルモジュール**
  - `nvidia.ko` ：NVIDIA カーネルドライバ (OSS版に限る)
  - `nvidia-fs.ko`: FS向けドライバ
- **利用可能なファイルシステム**
  - ローカルなNVMe SSD：ext4, xfs
  - リモートFS：Lustre FS, NFS…
- cuFile API
  - GDS：`cuFileWrite/cuFileRead…`
  - 互換モード：POSIX APIを使用



GDS ソフトウェアスタック

資料提供：当研究室（電気系工学専攻） 富永瑞己

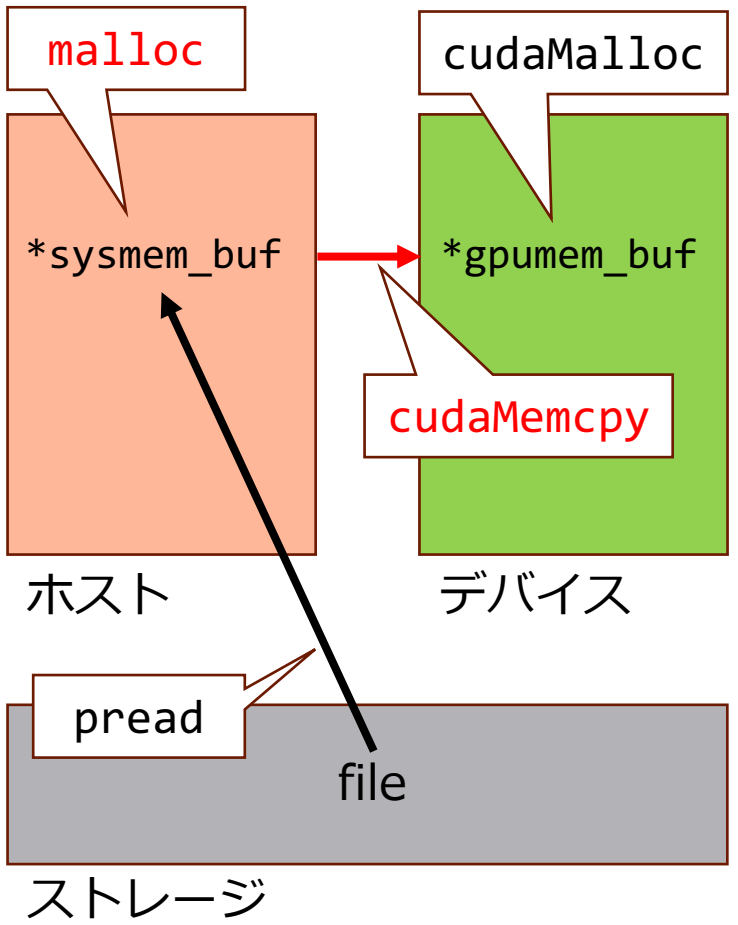
CPUコピー／GDSによるプログラム記述例

```
1 int fd = open(...)
2 void *sysmem_buf, *gpumem_buf;
3 sysmem_buf = malloc(buf_size);
4 cudaMalloc(gpumem_buf, buf_size);
5 pread(fd, sysmem_buf, buf_size);
6 cudaMemcpy(gpumem_buf, sysmem_buf, buf_size, H2D);
```

CPUコピー：POSIX APIによる記述例

```
1  CUFileHandle_t *fh;
2  CUFileDescr_t desc;
3  void *gpumem_buf;
4  int fd = open(file_name, O_DIRECT, ...)
5  desc.type = CU_FILE_HANDLE_TYPE_OPAQUE_FD;
6  desc.handle.fd = fd;
7  cuFileHandleRegister(&fh, &desc);
8  cudaMalloc(&gpumem_buf, buf_size);
9  cuFileRead(&fh, gpumem_buf, buf_size, ...);
```

GDS：cuFile APIによる記述例



資料提供：当研究室（電気系工学専攻） 富永瑞己

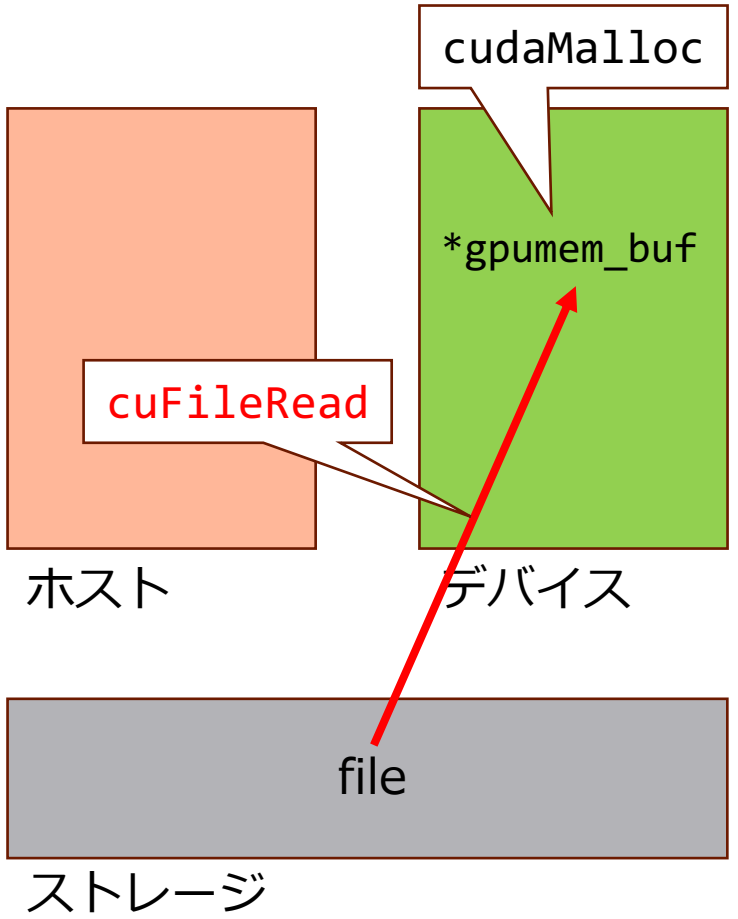
CPUコピー／GDSによるプログラム記述例

```
1 int fd = open(...)
2 void *sysmem_buf, *gpumem_buf;
3 sysmem_buf = malloc(buf_size);
4 cudaMalloc(gpumem_buf, buf_size);
5 pread(fd, sysmem_buf, buf_size);
6 cudaMemcpy(gpumem_buf, sysmem_buf, buf_size, H2D);
```

CPUコピー：POSIX APIによる記述例

```
1  CUFileHandle_t *fh;
2  CUFileDescr_t desc;
3  void *gpumem_buf;
4  int fd = open(file_name, O_DIRECT, ...)
5  desc.type = CU_FILE_HANDLE_TYPE_OPAQUE_FD;
6  desc.handle.fd = fd;
7  cuFileHandleRegister(&fh, &desc);
8  cudaMalloc(&gpumem_buf, buf_size);
9  cuFileRead(&fh, gpumem_buf, buf_size, ...);
```

GDS：cuFile APIによる記述例



資料提供：当研究室（電気系工学専攻） 富永瑞己

実アプリの中でGDSを使うには？

- HDF5（や分野によってはnetCDF）を使っていたりする
  - ただのバイナリファイルよりも後処理がやりやすい（h5py, VisIt, ParaView, …）
- cuFile の API を自分で叩いてファイル入出力
  - HDF5 を使うようにコードを書き換えた人にとってみれば, バイナルファイルへ逆戻り（= 機能が減る, コード改変量もかなり増える）
    ➔ アプリユーザ的には嬉しくないので, この方針は取りたくない
- “GPU-aware HDF5”
  - HDF5向けのVFDがOSSとして配布されている：
    https://github.com/hpc-io/vfd-gds
  - ファイル open 時に追加引数を渡せば, H5Dwrite() などにGPU上のアドレスを指定するだけで, （裏側で）GDSが使われる
    ➔コードの実装コストは無視できる程度であり, こちらが望ましい

実際の書き換えコストは？

- 通常のHDF5でのファイルアクセスと違う部分だけを抜粋

```
#include <H5FDgds.h>  // VFD for GDS

auto fapl = H5Pcreate(H5P_FILE_ACCESS);
H5Pset_fapl_gds(fapl, MBOUNDARY_DEF, FBSIZE_DEF, CBSIZE_DEF);

auto target = H5Fcreate(filename, H5F_ACC_TRUNC, H5P_DEFAULT, fapl);
```

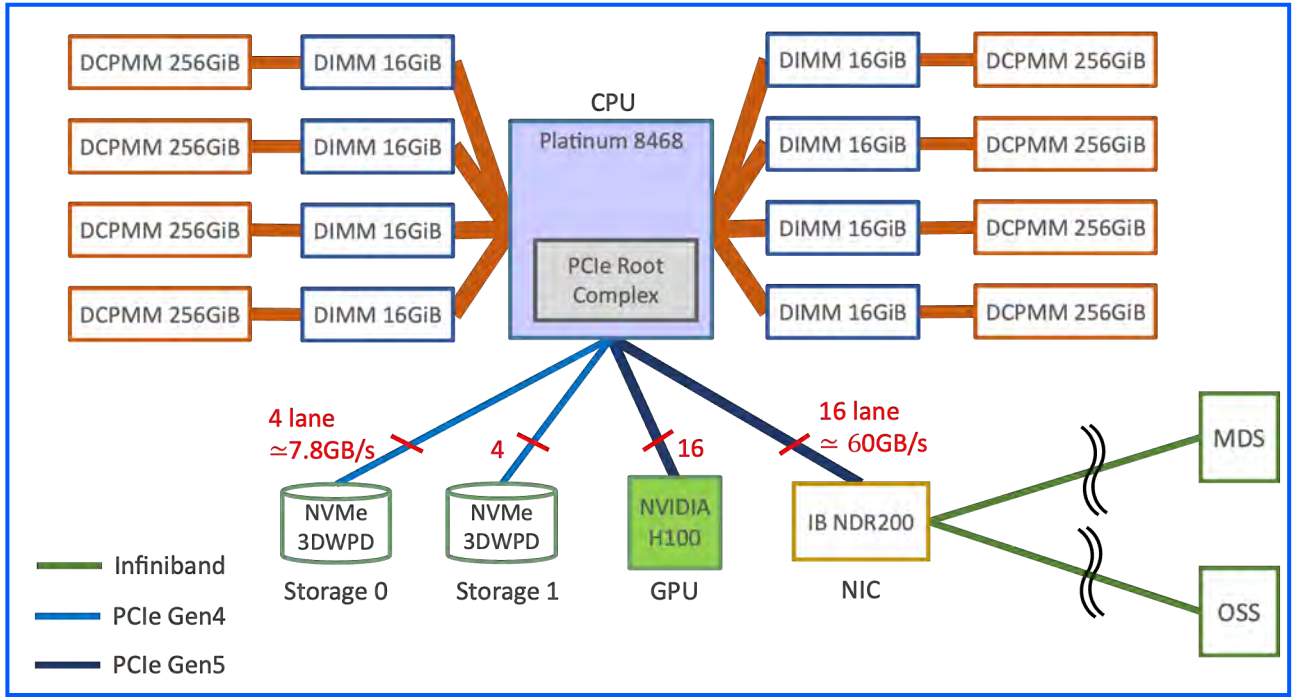
1. H5FDgds.h をインクルード
2. ファイルアクセス用のプロパティを作成, パラメータを指定
3. ファイルオープン時に2. で作ったプロパティも渡す

- GPU-aware MPI よりは処理が増えるとはいえ, 実際のファイルアクセス関連の処理には一切手を付ける必要がない

測定環境：筑波大CCS Pegasus

- 他にAMD EPYC Milan環境、mdx (VM), Wisteria-Mercury (Intel SPRx2+H100x4)でも確認中
  - 特性は異なるが、概ね同様の傾向

Category	Specification	Description
Hardware	CPU	Intel(R) Xeon(R) Platinum 8468
	RAM	DDR5-4800 DIMM DDR5-4800 Persistent Memory
	Storage	Local: NVMe 3.2TB 3DWPD (RAID 0) Remote: HDD 18TB NL-SAS 7200rpm
	NIC	InfiniBand NDR200 HCA
	GPU	NVIDIA H100
	PCIe	CPU-GPU: PCIe Gen5.0 (16 lane) CPU-NIC: PCIe Gen5.0 (16 lane) CPU-Storage: PCIe Gen4.0 (4 lane)
	Interconnect	NVIDIA Quantum-2 InfiniBand
Software	CUDA	12.1
	GPU Driver	535.54.03
	GDS Version	1.6.0.25
	Linux Kernel	5.4.0-167-generic
	File system	Local: xfs Remote: Lustre
	Distribution	bullseye/sid

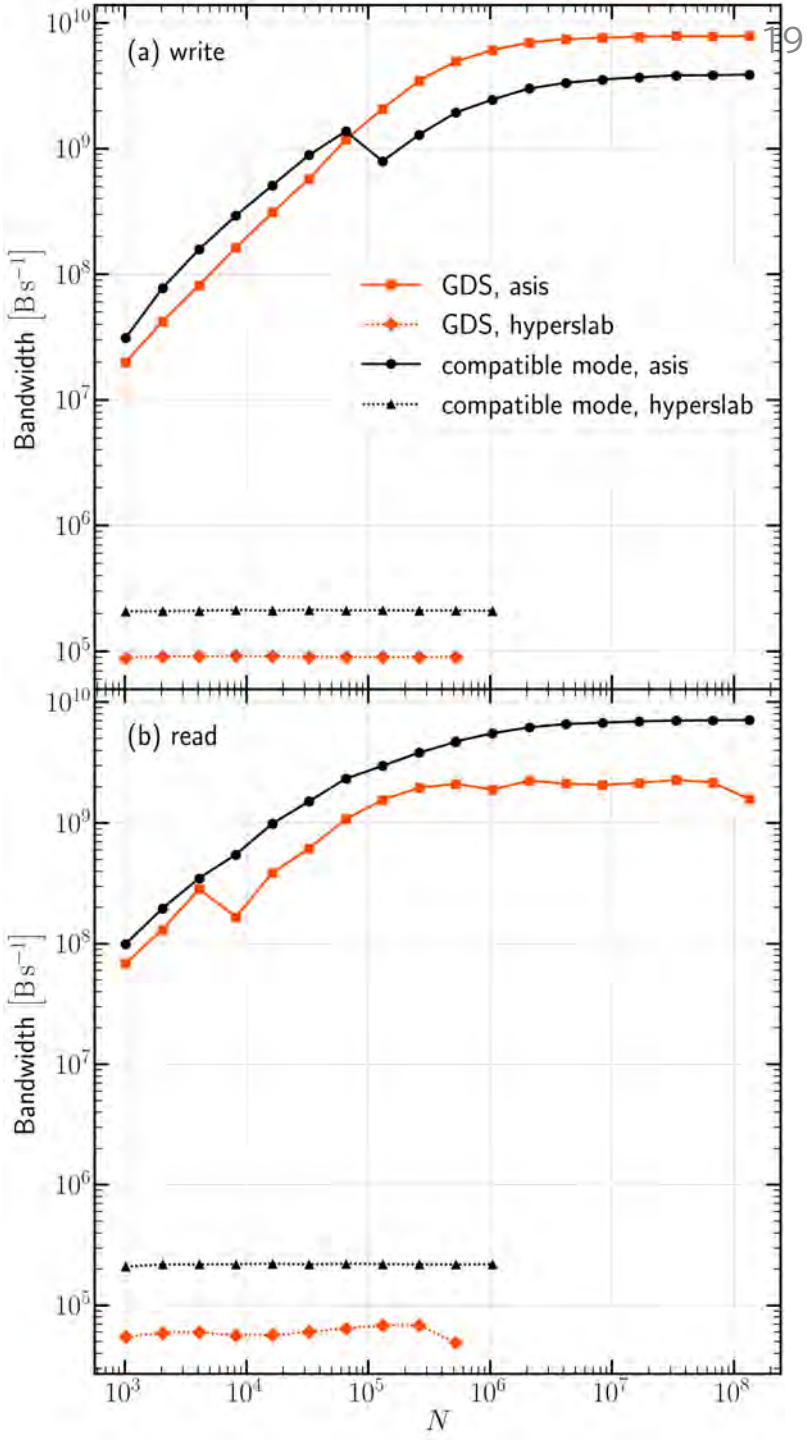


Pegasus: ブロックダイアグラム

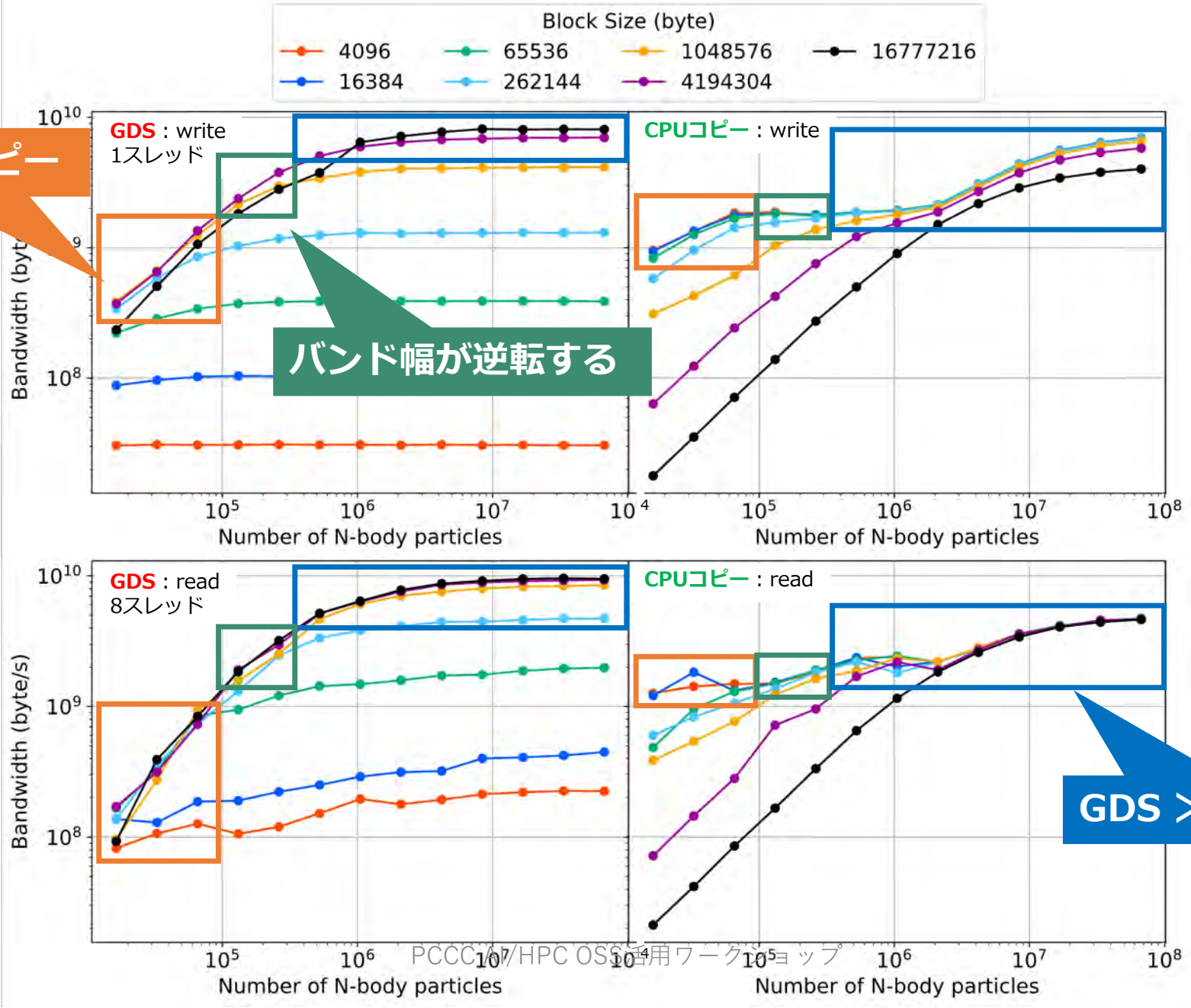
詳細情報：
https://www.ccs.tsukuba.ac.jp/wp-content/uploads/sites/14/Pegasus.pdf

ベンチマーク結果

- N体計算でよくあるデータ構造を想定し，GPU上の配列データをファイルアクセス
  - 粒子位置＋質量: `float4`
  - 粒子速度: `float2(x, y)` + `float(z)`
  - 粒子ID: `uint64_t`
- 筑波大CCSのPegasus上で測定
  - アクセス先は計算ノード内のNVMe SSD
  - VFD用の各種パラメータはデフォルトのまま
- $N \gtrsim 10^5$のwrite, as-is では GDS が高速
- Hyperslab（MPIの派生データ型的機能）を使用した時の性能は悲惨



ローカルNVMe-SSDの測定結果（一部）



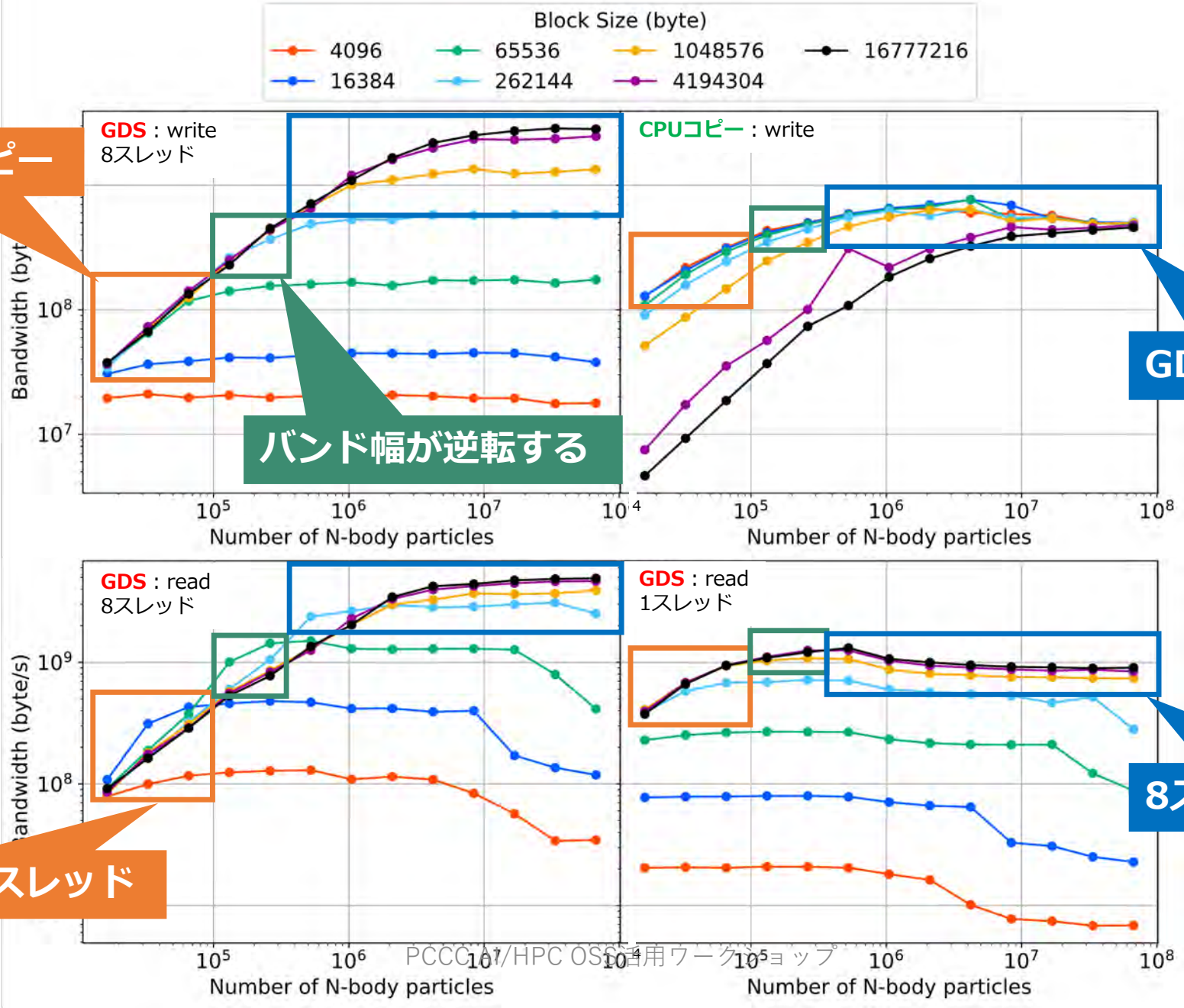
GDS ＜ CPUコピー

バンド幅が逆転する

GDS ＞ CPUコピー

資料提供:
当研究室(電気系工学専攻) 富永瑞己

Lustre FSの測定結果（一部）



GDS ＜ CPUコピー

バンド幅が逆転する

GDS ＞ CPUコピー

8スレッド ＜ 1スレッド

8スレッド ＞ 1スレッド

資料提供:
当研究室(電気系工学専攻)　富永瑞己

アプリケーションベンチマークの結果:まとめ

- バンド幅性能は、以下のパラメータによってbestな転送方式が異なる
  - ファイルサイズ
  - ファイルシステムの種類 (ローカルSSD / LustreFS)
  - Read / Write
  - スレッド数
  - (ほかにもあるかも)

資料提供:
当研究室(電気系工学専攻) 富永瑞己

最もバンド幅が高い転送方式

		ファイルサイズ		
		x < 3MB	3MB ≤ x < 10MB	10MB ≤ x
ローカルSSD	write: read:	CPUコピー CPUコピー	GDS GDS	GDS GDS
Lustre FS	write: read:	CPUコピー GDS	CPUコピー GDS	GDS GDS

ユーザが与えたヒントをwrite/read関数に渡す

```
H5Pset_fapl_gds(fapl_id, MBOUNDARY_DEF, FBSIZE_DEF, CBSIZE_DEF,
FILESIZE_DEF, FILESYSTEM_DEF)
```

変更したH5Pset_fapl_gds関数

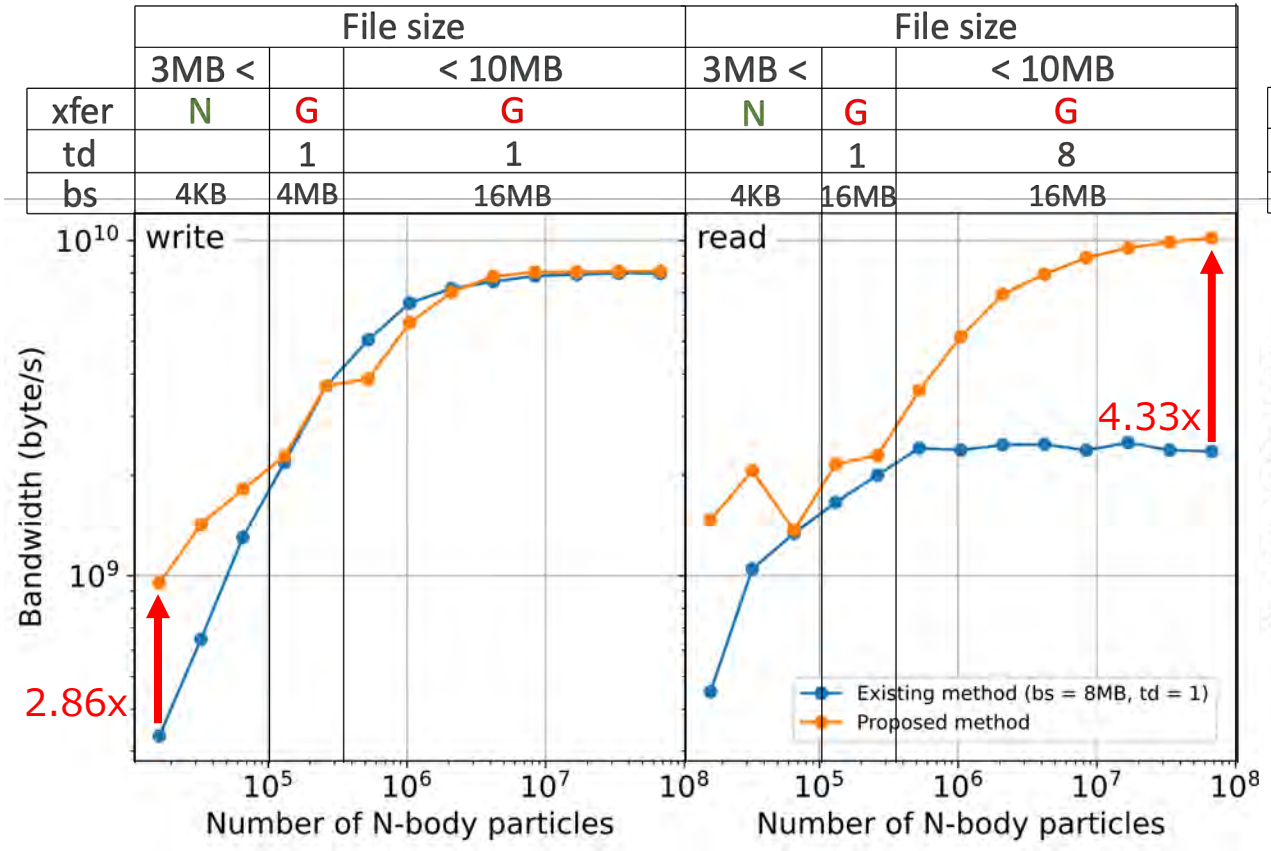
資料提供:
当研究室(電気系工学専攻) 富永瑞己

- **第5引数:ファイルサイズ**
  - LESS_THAN_3MB:ファイルサイズが3MB未満のときに指定
  - BETWEEN_3MB_AND_10MB:ファイルサイズが3MB以上10MB未満のときに指定
  - MORE_THAN_10MB:ファイルサイズが10MB以上のときに指定(デフォルト値)

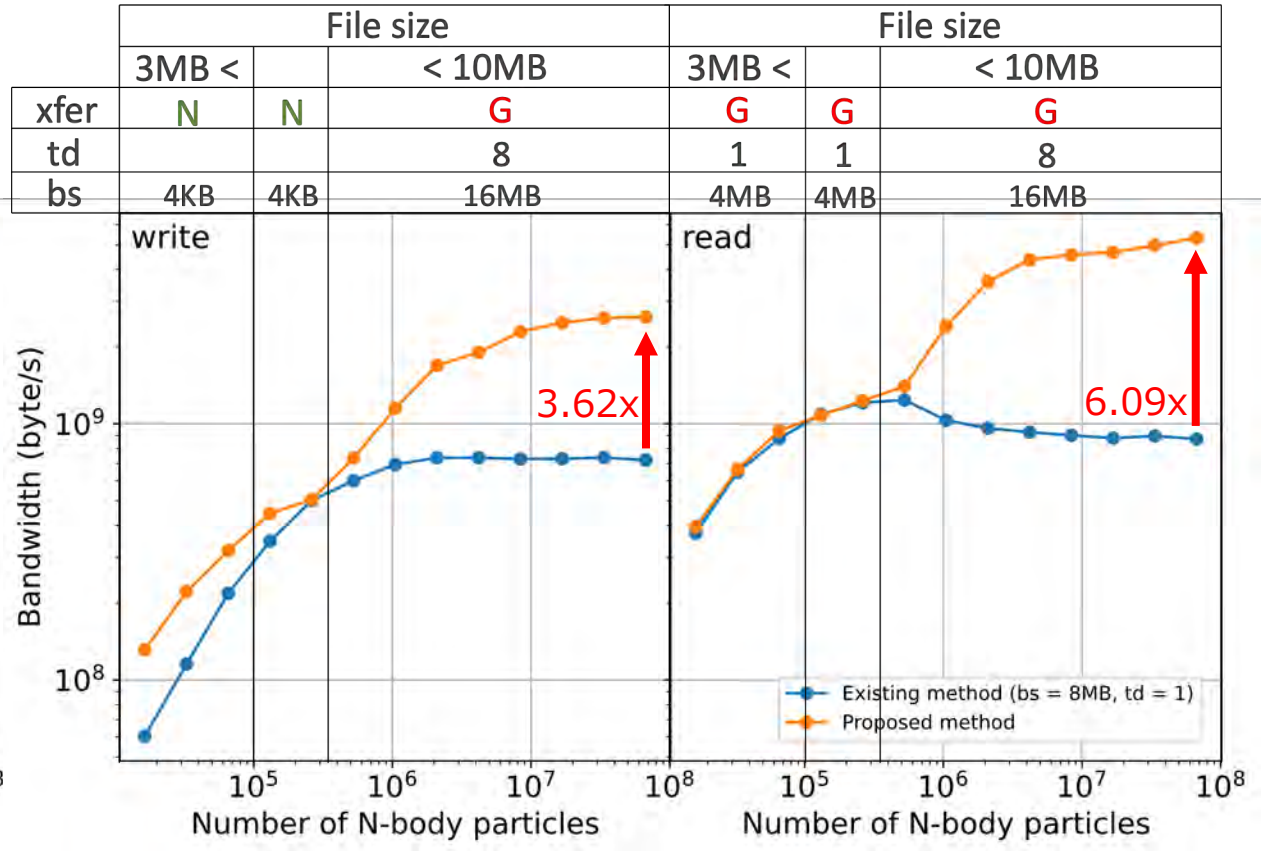
- **第6引数:ファイルシステム**
  - LOCAL:xfsまたはext4のファイルに対してファイルIOを行うときに指定(デフォルト値)
  - REMOTE:Lustreのファイルに対してファイルIOを行うときに指定

既存のGDS VFDと提案手法の性能比較

凡例
G: GDS, N: CPUコピー
td: スレッド数, bs: 転送長



(a) Pegasus：ローカルSSD



(b) Pegasus：Lustre FS

- ファイルサイズが小さいとき：**CPUコピー**を使用することにより，**GDS**より高速化
- ファイルサイズが大きいとき：同じ**GDS**でも，スレッド数を増やしたことにより高速化

資料提供：
当研究室(電気系工学専攻)　富永瑞己

小まとめ:GDS

- HDF5 VFD for GDSの改善については一定の成果
  - ローカルSSD: 最大 4.33倍、Lustre: 最大 6.09倍

問題点

- 現状の実装については改善の余地あり
  - CPU実装のマルチスレッド対応
  - 非同期の利用
    - CPUコピー：CUDAストリーム (cudaMemcpyAsync)
    - GDS: cuFileストリーム (cuFileWriteAsync/cuFileReadAsync)
  - メモリアロケーションの最適化、特にCPUコピー
- マニュアルでのattribute指定
  - 自動で判別できそうな気もする

まとめ

- 既存のMPI+OpenMPコードからGPUクラスタに向けた移植に伴う問題について検討
- 指示文ベースでGPU化したい場合に向けた指示文統合マクロを開発・試験中
  - マクロからOpenACC, OpenMP targetを生成
  - コード管理が容易に、性能の問題もない
- GPUDirect Storage (GDS)によるHDF5のファイルIO高速化、VFD for GDSの改良
  - CPUコピーとGDSとを、転送サイズ等のパラメータで切り替え
  - オリジナルに比べ性能向上、しかし改善の余地あり