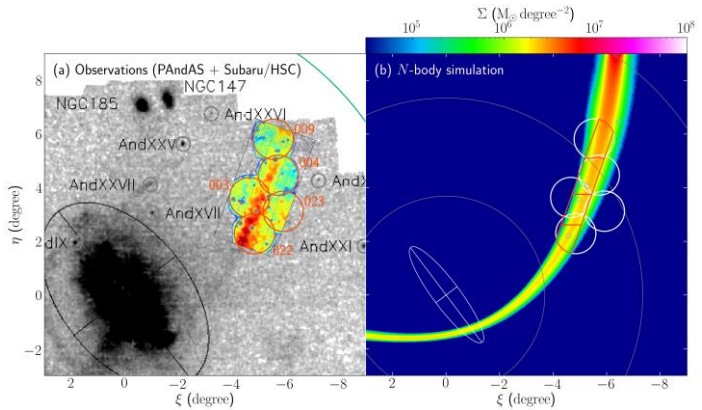
JCAHPCの新スパコンMiyabiの紹介とGPU移植

三木 洋平
（東京大学 情報基盤センター / 最先端共同HPC基盤施設（JCAHPC））

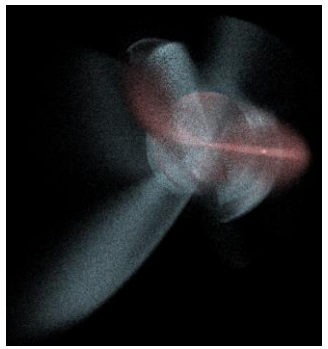
第24回HPC-Phys勉強会

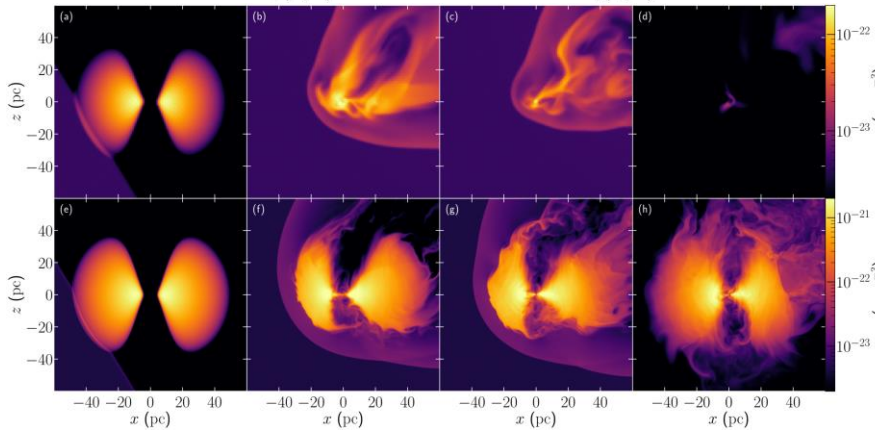
自己紹介: 宇宙物理学の研究のためにコードも書く人

2005-2009	筑波大学 第一学群 自然学類
2009-2011	筑波大学大学院 数理物質科学研究科 物理学専攻(博士前期課程)
2011-2014	筑波大学大学院 数理物質科学研究科 物理学専攻(博士後期課程)
2011-2013	筑波大学大学院 システム情報工学研究科 コンピュータサイエンス専攻(博士前期課程)
2014-2017	筑波大学 計算科学研究センター 研究員
2017-2024	東京大学 情報基盤センター 助教
2024-	東京大学 情報基盤センター 准教授



- 専門分野:宇宙物理学, 高性能計算
  - 銀河の形成・進化(銀河考古学, 銀河衝突)
  - 銀河と中心ブラックホールの共進化
  - GPUを用いた演算加速
  - 重力多体計算, 数値流体力学
  - (準)力学平衡状態生成コードの作成





Contents

- JCAHPCの新スパコンMiyabiの紹介
  - 導入経緯
  - Miyabiの概要
  - NVIDIA GH200の特性
  - Miyabi活用に向けてのGPU移植支援
- GPU向けのプログラミング環境
  - 指示文を使いたい場合
    - OpenACC or OpenMP target
    - GPU向け指示文統合マクロSolomon
  - 低レベルな開発環境を使いたい場合
    - CUDA C++, HIP C++, SYCL
    - 性能可搬フレームワークKokkos

Contents

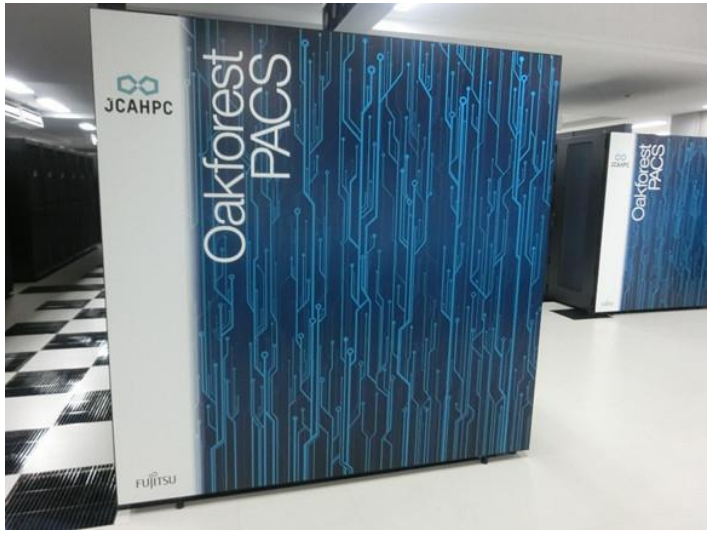
- JCAHPCの新スパコンMiyabiの紹介
  - 導入経緯
  - Miyabiの概要
  - NVIDIA GH200の特性
  - Miyabi活用に向けてのGPU移植支援
- GPU向けのプログラミング環境
  - 指示文を使いたい場合
    - OpenACC or OpenMP target
    - GPU向け指示文統合マクロSolomon
  - 低レベルな開発環境を使いたい場合
    - CUDA C++, HIP C++, SYCL
    - 性能可搬フレームワークKokkos

最先端共同HPC基盤施設

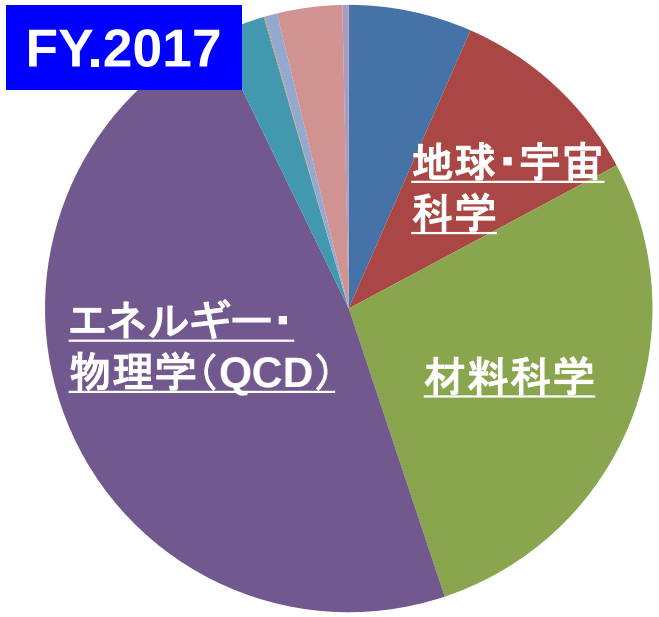
筑波大学 University of Tsukuba　東京大学 THE UNIVERSITY OF TOKYO

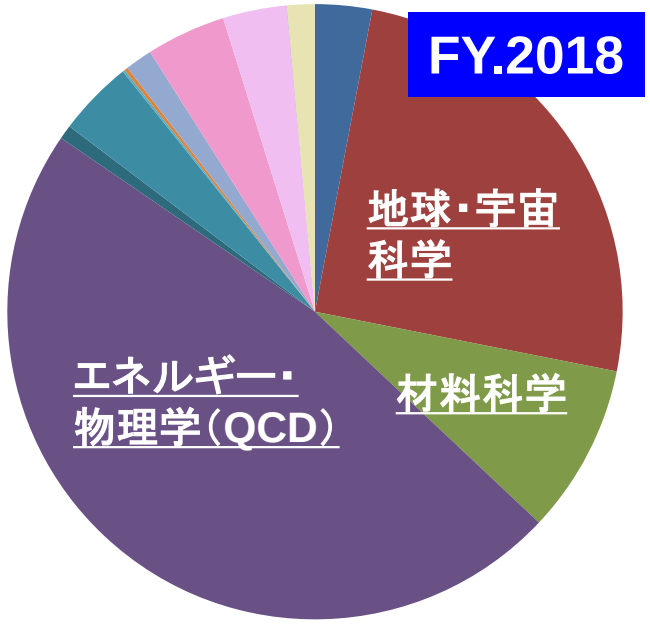
JCAHPC

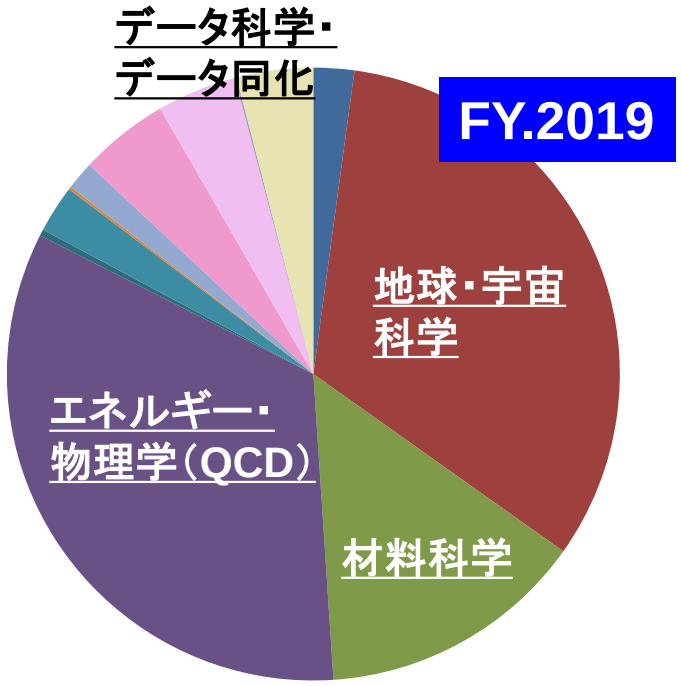
- https://www.jcahpc.jp
- JCAHPC (Joint Center for Advanced High Performance Computing)
  - 2013年創立
  - 筑波大学計算科学研究センター・東京大学情報基盤センター
  - 最先端計算科学の推進
  - 大規模システムの設計・導入・運用を共同で実施する我が国でも初の試み：より大規模なシステムを効率的に導入可能
- Oakforest-PACS(OFP)：JCAHPC第一号機
  - Intel Xeon Phi 8,208ノード, 25PF(富士通)
  - 2016年11月時点でTop500の6位(国内1位)
    - 2022年3月末で退役
  - 「京」の退役後,「富岳」登場までの2019・2020年度は事実上の「National Flagship System」としての役割を担う
- OFP-II(OFP後継機)へ向けた試み

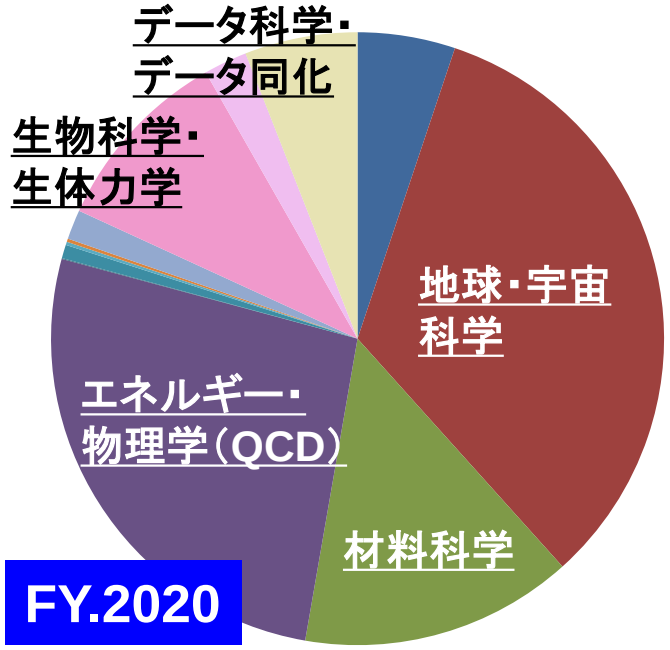


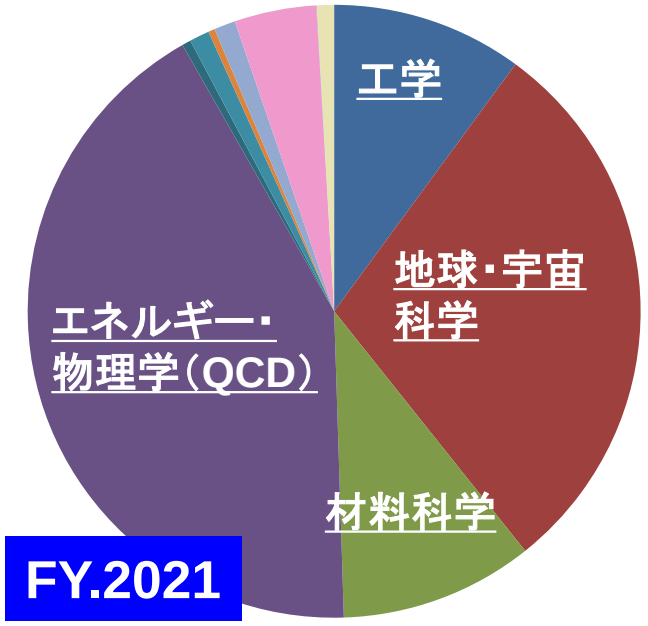
研究分野別CPU時間割合：OFP



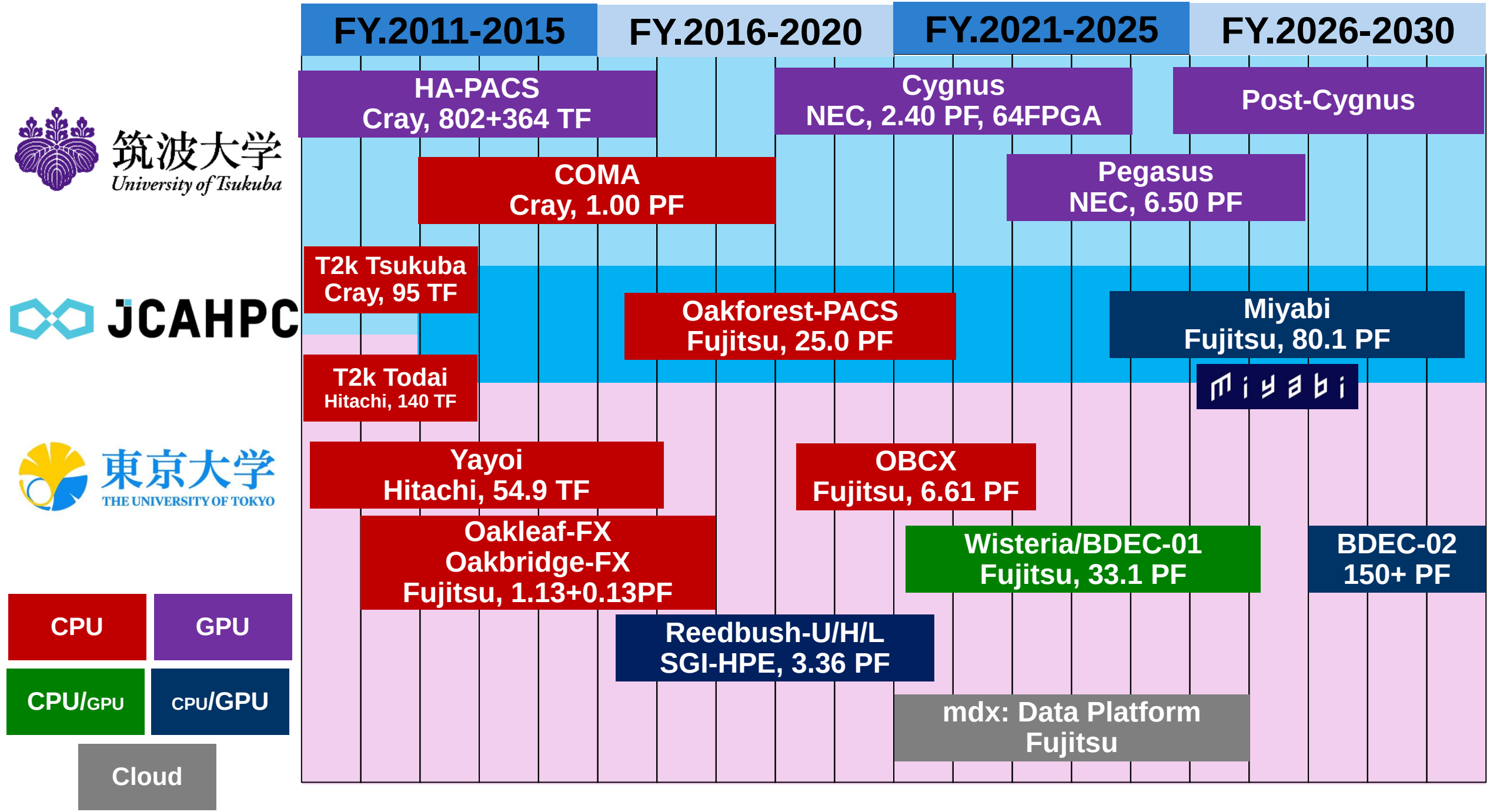








- 工学・ものつくり
- 地球科学・宇宙科学
- 材料科学
- エネルギー・物理学
- 情報科学：システム
- 情報科学：アルゴリズム
- 情報科学：AI
- 教育
- 産業利用
- 生物科学・生体力学
- バイオインフォマティクス
- 社会科学・経済
- データ科学・データ同化

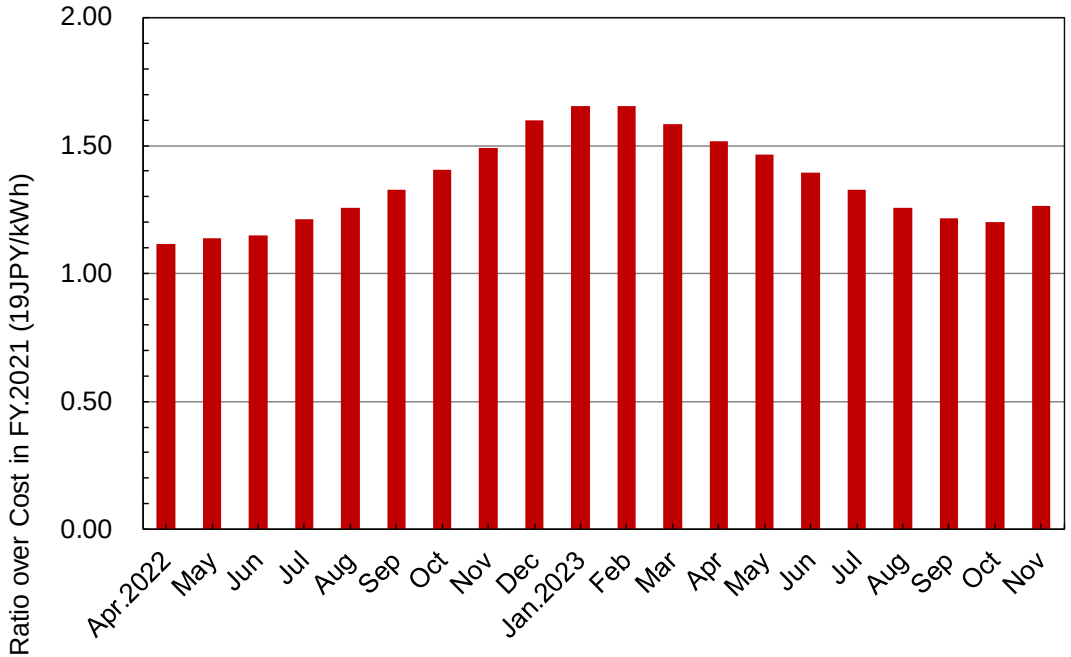

FY.2011-2015
FY.2016-2020
FY.2021-2025
FY.2026-2030
筑波大学
University of Tsukuba
JCAHPC
東京大学
THE UNIVERSITY OF TOKYO
HA-PACS
Cray, 802+364 TF
Cygnus
NEC, 2.40 PF, 64FPGA
Post-Cygnus
COMA
Cray, 1.00 PF
Pegasus
NEC, 6.50 PF
T2k Tsukuba
Cray, 95 TF
Oakforest-PACS
Fujitsu, 25.0 PF
Miyabi
Fujitsu, 80.1 PF
T2k Todai
Hitachi, 140 TF
Yayoi
Hitachi, 54.9 TF
OBCX
Fujitsu, 6.61 PF
Oakleaf-FX
Oakbridge-FX
Fujitsu, 1.13+0.13PF
Wisteria/BDEC-01
Fujitsu, 33.1 PF
BDEC-02
150+ PF
Reedbush-U/H/L
SGI-HPE, 3.36 PF
mdx: Data Platform
Fujitsu
CPU
GPU
CPU/GPU
CPU/GPU
Cloud

Miyabi (OFP-II)導入への道(1/3)

JCAHPC
筑波大学 University of Tsukuba
東京大学 THE UNIVERSITY OF TOKYO

- 2019年11月頃: JCAHPCとして2代目も継続して設計・運用することを確認
- 2021年2月:システム名を “Oakforest-PACS II (OFP-II)” とすることに決定
- スパコンへの性能要求, 省電力, 脱炭素化
  ⇒演算加速器搭載は不可避(電気代も高騰)
  - 2021年秋には方針決定

System (Top/Green 500)	HW	GF/W
JEDI (222,1)	NVIDIA GH200	72.733
Adastra 2 (440,3)	AMD MI300A	69.098
Capella (18,6)	NVIDIA H100	68.053
FX1000 (109,81)	A64FX	16.575
Carpenter (104,101)	AMD EPYC 9654	10.561

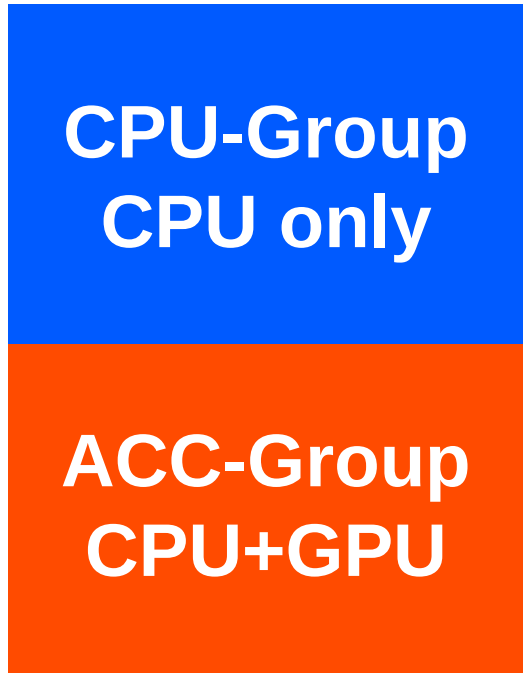


電力単価推移(対2021年度比)

Miyabi (OFP-II)導入への道(2/3)

JCAHPC
筑波大学 University of Tsukuba
東京大学 THE UNIVERSITY OF TOKYO

- ヘテロな構成
  - 汎用CPUクラスタ(CPU-Group)＋GPUクラスタ(ACC-Group)
  - 「計算・データ・学習」融合路線は継続
- 先行してGPUを選定
  - OFPユーザー(3,000人以上)のGPUへの移行には18-30ヶ月必要
  - GPUへの移行用の環境を事前に導入,
    OFP-IIと同アーキテクチャでないと意味がない
    ➔ Wisteria-Mercury導入
  - 2021年10月　GPUの選定を先行して行うことを決定
  - 2022年2~3月　各社にプリベンチマークを打診, 配布
  - 2022年6月　プリベンチマーク結果の報告を元にGPUアーキテクチャ決定



Miyabi (OFP-II)導入への道(3/3)

JCAHPC
筑波大学 University of Tsukuba
東京大学 THE UNIVERSITY OF TOKYO

- GPU検討のためのベンチマークを各社に依頼
  - 7種類, 計算科学系
    - OpenMP+MPI or GPU化済み
    - それぞれについて右表のような対応
    - ベンチマークは次ページ

➔NVIDIA H100またはその後継に決定

A	CPU版のコードに対してホストCPUでの性能を評価	A-1	提供コードをそのまま実行（As-Is）
		A-2	最適化したコードを使用
B	CPU版のコードをGPU化	B-1	OpenACC, OpenMP, Standard Language等の手法でGPU化
		B-2	当該GPUで最大限の性能が出せるようチューニング
C	GPU化済みのコードを最適化	C-1	提供コードまたは既存コードを実行
		C-2	当該GPUで最大限の性能が出せるようチューニング

- 決め手
  - 性能そのもの
  - Fortranで記述されたアプリケーションのポータビリティ
  - OpenACC/StdPar(Standard Parallelism)によるGPU化は比較的簡単, OpenMP/MPIハイブリッドによって並列化されたプログラムに適している

Seven benchmarks

JCAHPC
筑波大学 University of Tsukuba
東京大学 THE UNIVERSITY OF TOKYO

A: Benchmark for General CPU ➔ GPU
B: Already GPU-enabled

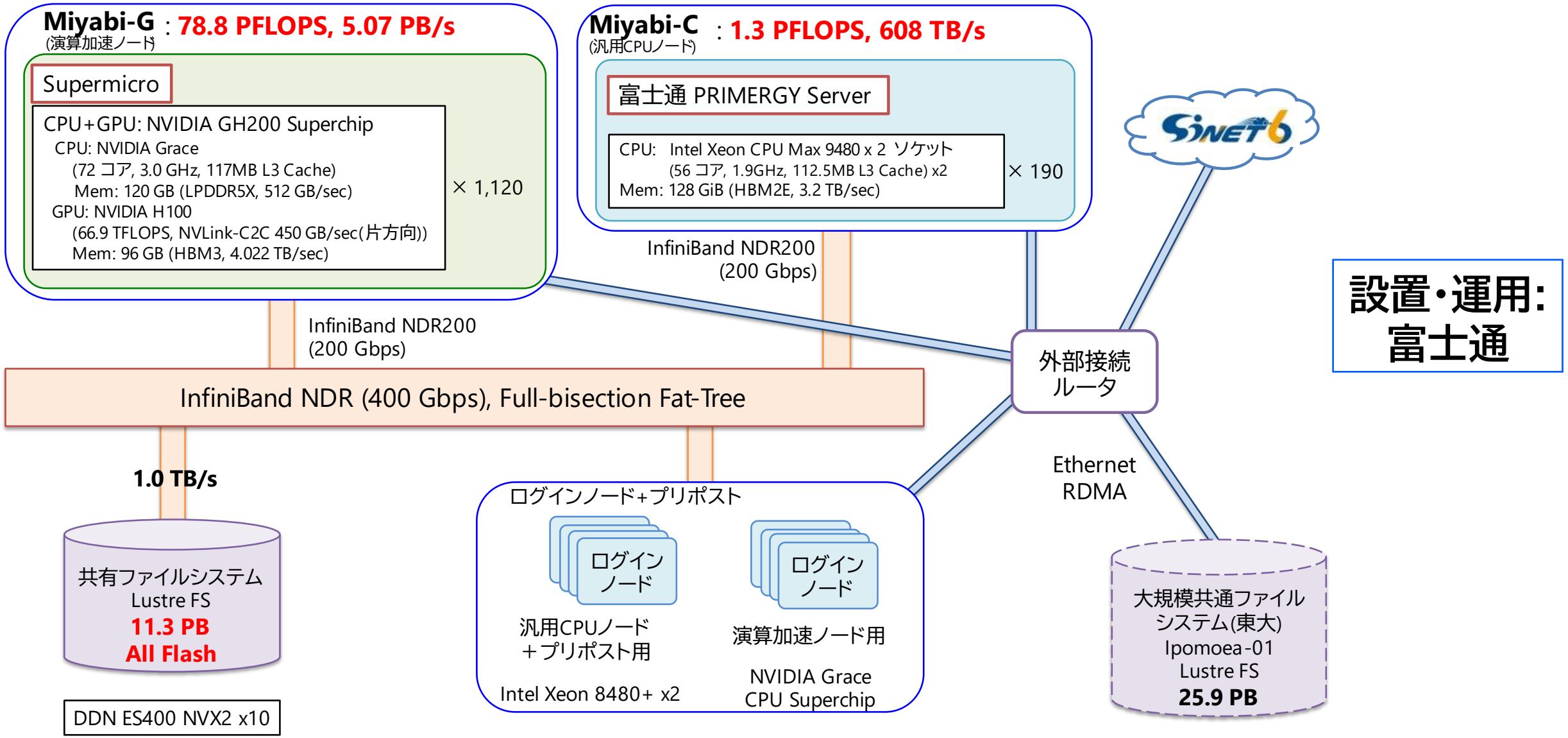
Code	Description	Lang.	Parallelization	GPU	Category
P3D	3-D Poisson's Equation by Finite Volume Method	C	OpenMP	N/A	A
GeoFEM/ICCG	Finite Element Method	Fortran	OpenMP, MPI	N/A	
H-Matrix	Hierarchical-Matrix calculation	Fortran	OpenMP, MPI	N/A	
QCD	Quantum-Chromo Dynamics simulation	Fortran	OpenMP, MPI	CUDA	B
N-Body	N-Body simulation using FDPS	C++	OpenMP, MPI	CUDA	
GROMACS	Molecular Dynamics simulation	C++	OpenMP, MPI	CUDA, HIP, SYCL	
SALMON	Ab-initio quantum-mechanical simulator for optics and nanoscience	Fortran	OpenMP, MPI	(OpenACC)	A

Contents

- JCAHPCの新スパコンMiyabiの紹介
  - 導入経緯
  - Miyabiの概要
  - NVIDIA GH200の特性
  - Miyabi活用に向けてのGPU移植支援
- GPU向けのプログラミング環境
  - 指示文を使いたい場合
    - OpenACC or OpenMP target
    - GPU向け指示文統合マクロSolomon
  - 低レベルな開発環境を使いたい場合
    - CUDA C++, HIP C++, SYCL
    - 性能可搬フレームワークKokkos

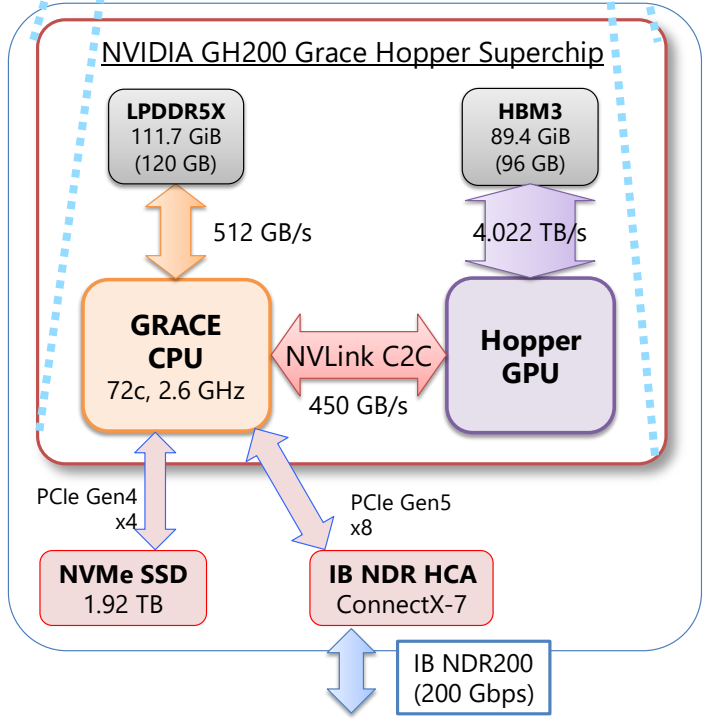
Miyabiの概要(1/3)

- 2025年1月運用開始, 80.1 PFLOPS



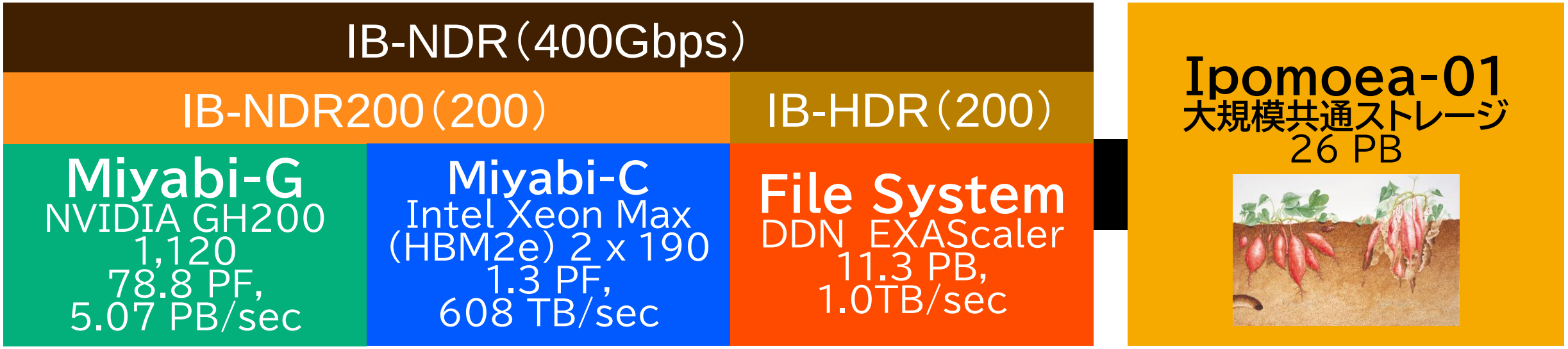
Miyabiの概要(2/3)

- **Miyabi-G: 演算加速ノード: NVIDIA GH200**
  - 計算ノード: NVIDIA GH200 Grace-Hopper Superchip
    - Grace: 72c, 3.45 TF, 120 GB, 512 GB/sec (LPDDR5X)
    - H100: 66.9 TF DP-Tensor Core, 96 GB, 4,022 GB/sec (HBM3)
      - CPU-GPU間はキャッシュコヒーレント
    - NVMe SSD for each GPU: 1.9TB, 8.0GB/sec, GPUDirect Storage
  - 合計 (CPU+GPUの合計値)
    - 1,120 ノード, 78.8 PF, 5.07 PB/sec, IB-NDR 200
- **Miyabi-C: 汎用CPUノード: Intel Xeon Max 9480 (SPR)**
  - 計算ノード:
    - Intel Xeon Max 9480 (1.9 GHz, 56c) x 2
    - 6.8 TF, 128 GiB, 3,200 GB/sec (HBM2e only)
  - 合計
    - 190 ノード, 1.3 PF, IB-NDR 200
    - 372 TB/sec for STREAM Triad (Peak: 608 TB/sec)



Miyabiの概要(3/3)

- ファイルシステム: DDN EXAScalar, Lustre FS
  - 11.3 PB (NVMe SSD) 1.0TB/sec
  - “Ipomoea-01” (26 PB) も利用可能
- Miyabi-G/C の全ノードはフルバイセクションバンド幅Fat Treeで接続
  - (400Gbps/8)×(32×20+16×1) = 32.8 TB/sec
- 2025年1月運用開始, Miyabi-G/C間の通信は h3-Open-SYS/WaitIO により実現



miyabi

64th TOP500 List (Nov, 2024)

R_{max}: Performance of Linpack (TFLOPS)
R_{peak}: Peak Performance (TFLOPS), Power: kW

https://www.top500.org/

	Site	Computer/Year Vendor	Cores	R_{max} (PFLOPS)	R_{peak} (PFLOPS)	GFLOPS/W	Power (kW)
1	**El Capitan, 2024, USA** DOE/NNSA/LLNL	HPE Cray EX255a, AMD 4th Gen EPYC 24C 1.8GHz, AMD Instinct MI300A, Slingshot-11, TOSS	11,039,616	1,742.00 (=1.742 EF)	2,746.38 63.4 %	58.99	29,581
2	**Frontier, 2021, USA** DOE/SC/Oak Ridge National Laboratory	HPE Cray EX235a, AMD Optimized 3rd Gen. EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11	9,066,176	1.353.00	2,055.72 65.8 %	54.98	24,607
3	**Aurora, 2023, USA** DOE/SC/Argonne National Laboratory	HPE Cray EX - Intel Exascale Compute Blade, Xeon CPU Max 9470 52C 2.4GHz, Intel Data Center GPU Max, Slingshot-11, Intel	9,264,128	1,012.00	1,980.01 51.1 %	26.15	38,698
4	**Eagle, 2023, USA** Microsoft	Microsoft NDv5, Xeon Platinum 8480C 48C 2GHz, NVIDIA H100, NVIDIA Infiniband NDR	2,073,600	561.20	846.84 66.3 %		
5	**HPC 6, 2024, Italy** Eni S.p.A.	HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, RHEL 8.9	3,143,520	477.90	606.97 66.3 %	56.48	8,461
6	**Fugaku, 2020, Japan** R-CCS, RIKEN	Fujitsu PRIMEHPC FX1000, Fujitsu A64FX 48C 2.2GHz, Tofu-D	7,630,848	442.01	537.21 82.3 %	14.78	29,899
7	**Alps, 2024, Switzerland** Swiss Natl. SC Centre (CSCS)	HPE Cray EX254n, NVIDIA Grace 72C 3.1GHz, NVIDIA GH200 Superchip, Slingshot-11	2,121,600	434.90	574.84 75.7 %	61.05	7,124
8	**LUMI, 2023, Finland** EuroHPC/CSC	HPE Cray EX235a, AMD Optimized 3rd Gen. EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11	2,752,704	379.70	531.51 71.4 %	53.43	7,107
9	**Leonard, 2023, Italy** EuroHPC/Cineca	BullSequana XH2000, Xeon Platinum 8358 32C 2.6GHz, NVIDIA A100 SXM4 64GB, Quad-rail NVIDIA HDR100	1,824,768	241.20	306.31 78.7 %	32.19	7,494
10	**Tuolumne, 2024, USA** DOE/NNSA/LLNL	HPE Cray EX255a, AMD 4th Gen EPYC 24C 1.8GHz, AMD Instinct MI300A, Slingshot-11, TOSS	1,161,216	208.10	288.88 72.0 %	61.45	3,387
13	**Venado, 2024, USA** DOE/NNSA/LANL	HPE Cray EX254n, NVIDIA Grace 72C 3.1GHz, NVIDIA GH200 Superchip, Slingshot-11	481,440	98.51	130.44 75.5 %	59.29	1,662
16	**CHIE-3, 2024, Japan** SoftBank, Corp.	NVIDIA DGX H100, Xeon Platinum 8480C 56C 2GHz, NVIDIA H100, Infiniband NDR400, Ubuntu 22.04.4 LTS	163.200	91.94	138.32 66.5 %		
17	**CHIE-2, 2024, Japan** SoftBank, Corp.	NVIDIA DGX H100, Xeon Platinum 8480C 56C 2GHz, NVIDIA H100, Infiniband NDR400, Ubuntu 22.04.4 LTS	163.200	89.78	138.32 64.9 %		
18	**JETI, 2024, Germany** EuroHPC/FZJ	BullSequana XH3000, Grace Hopper Superchip 72C 3GHz, NVIDIA GH200 Superchip, Quad-Rail NVIDIA InfiniBand NDR200, RedHat Linux, Modular OS	391,680	83.14	94.00 88.4 %	63.43	1,311
22	**CEA-HE, 2024, France** CEA	BullSequana XH3000, Grace Hopper Superchip 72C 3GHz, NVIDIA GH200 Superchip, Quad-Rail BXI v2, EVIDEN	389,232	64.32	103.48 62.2 %	52.17	1,233
28	**Miyabi-G, 2024, Japan** JCAHPC	Fujitsu, Supermicro ARS 111GL DNHR LCC, Grace Hopper Superchip 72C 3GHz, Infiniband NDR200, Rocky Linux	80,640	46.80	72.80 64.3 %	47.59	983
36	**TSUBAME 4.0, 2024, Japan** Institute of Science Tokyo	HPE Cray XD665, AMD EPYC 9654 96C 2.4GHz, NVIDIA H100 SXM5 94 GB, Infiniband NDR200	172,800	39.62	61.60 64.3 %	48.55	816
58	**Wisteria/BDEC-01 (Odyssey), 2021, Japan** U.Tokyo	Fujitsu PRIMEHPC FX1000, A64FX 48C 2.2GHz, Tofu D	368,640	22.12	25.95 85.2 %	15.07	1,468

Green 500 Ranking (Nov, 2024)

https://www.top500.org/

	TOP 500	System	Accelerator	Cores	HPL Rmax (Pflop/s)	Power (kW)	GFLOPS/W	Level
1	224	JEDI, EuroHPC/Julich, Germany	NVIDIA GH200	19,584	4.50	67	72.733	1
2	122	ROMEO-2025, ROMEO HPC Center - Champagne-Ardenne, France	NVIDIA GH200	47,328	9.86	160	70.912	1
3	442	Adastra2, GENCI-CINES, France	AMD Instrinct MI300A	16,128	2.53	37	69.098	1
4	155	Isambard-AI phase1, U. Bristol, UK	NVIDIA GH200	34,272	7.42	117	68.835	1
5	51	Capella, TU Dresden ZIH, Germany	NVIDIA H100 94GB	85,248	24.06	445	68.053	3
6	18	JETI, EuroHPC/Julich, Germany	NVIDIA GH200	391,680	83.14	1,311	67.963	1
7	69	Helios GPU, Cyfronet, Poland	NVIDIA GH200	89,760	19.14	317	66.948	2
8	371	Henri, Flatiron Institute, USA	NVIDIA H100 80GB	8,288	2.88	44	65.396	?
9	340	HoreKa-Teal, KIT, Germany	NVIDIA H100 94GB SXM5	13,616	3.12	50	62.964	1
10	49	rzAdams, DoE LLNL, US	AMD Instrinct MI300A	129,024	24.38	388	62.803	2
16	13	Venado, DoE LANL, US	NVIDIA GH200	481,440	98.51	1,662	59.287	1
30	36	TSUBAME 4.0, Science Tokyo	NVIDIA H100 94GB SXM5	172,800	39.62	816	48.565	3
33	28	**Miyabi-G, JCAHPC**	**NVIDIA GH200**	**221,952**	**46.80**	**983**	**47.588**	**3**
48	230	Pegasus, University of Tsukuba, Japan	NVIDIA H100 80GB	27,000	4.34	130	41.123	2
49	191	Wisteria/BDEC-01 (Aquarius), The University of Tokyo, Japan	NVIDIA A100 40GB	42,120	4.425	183	24.06	2

HPCG Ranking (Nov, 2024)

	Computer	Cores	HPL Rmax (Pflop/s)	TOP500 Rank	HPCG (Pflop/s)
1	Fugaku	7,630,848	442.01	4	16.00
2	Frontier	8,699,904	1,206.00	1	14.05
3	Aurora	9,264,128	1,012.00	2	5.61
4	LUMI	2,752,704	379.70	5	4.59
5	Alps	2,121,600	434.90	7	3.67
6	Leonardo	1,824,768	241.20	9	3.11
7	Perlmutter	888,832	79.23	19	1.91
8	Sierra	1,572,480	94.64	14	1.80
9	Selene	555,520	63.46	23	1.62
10	JUWELS Booster	449,228	44.12	33	1.28
12	AOBA-S	64,512	17.22	76	1.09
17	Wisteria/Odyssey	348,640	22.12	58	0.818
19	ES4-SX-Aurora Tsubasa	43,776	9.99	119	0.748
21	Miyabi-G	221,952	46.80	28	0.645

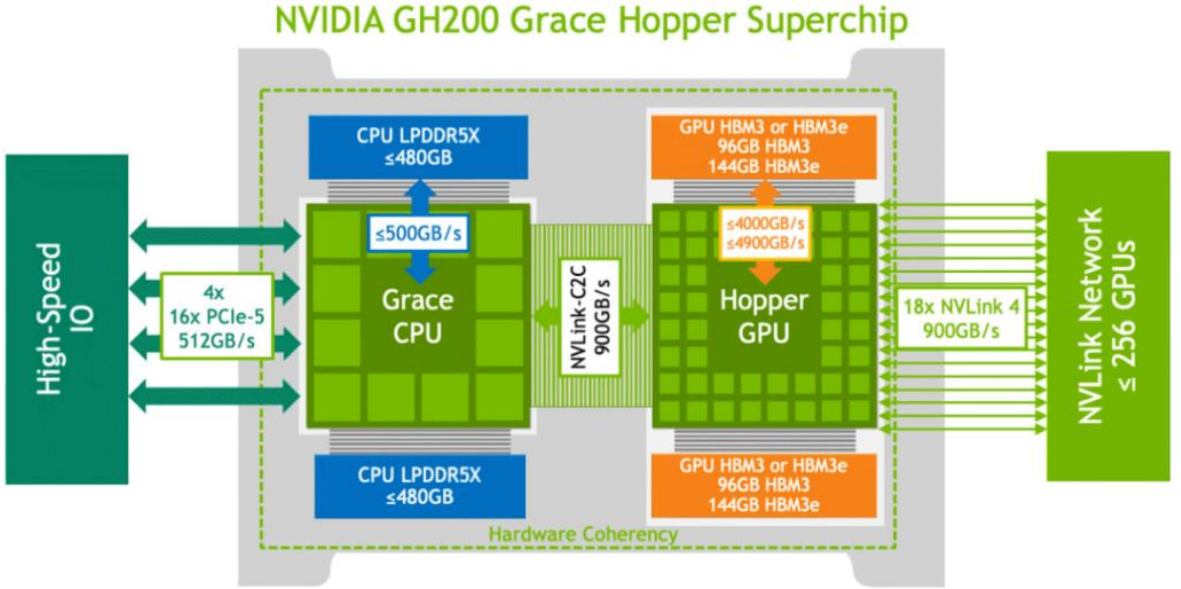
https://www.hpcg-benchmark.org/

Contents

- JCAHPCの新スパコンMiyabiの紹介
  - 導入経緯
  - Miyabiの概要
  - NVIDIA GH200の特性
  - Miyabi活用に向けてのGPU移植支援
- GPU向けのプログラミング環境
  - 指示文を使いたい場合
    - OpenACC or OpenMP target
    - GPU向け指示文統合マクロSolomon
  - 低レベルな開発環境を使いたい場合
    - CUDA C++, HIP C++, SYCL
    - 性能可搬フレームワークKokkos

NVIDIA GH200の特性

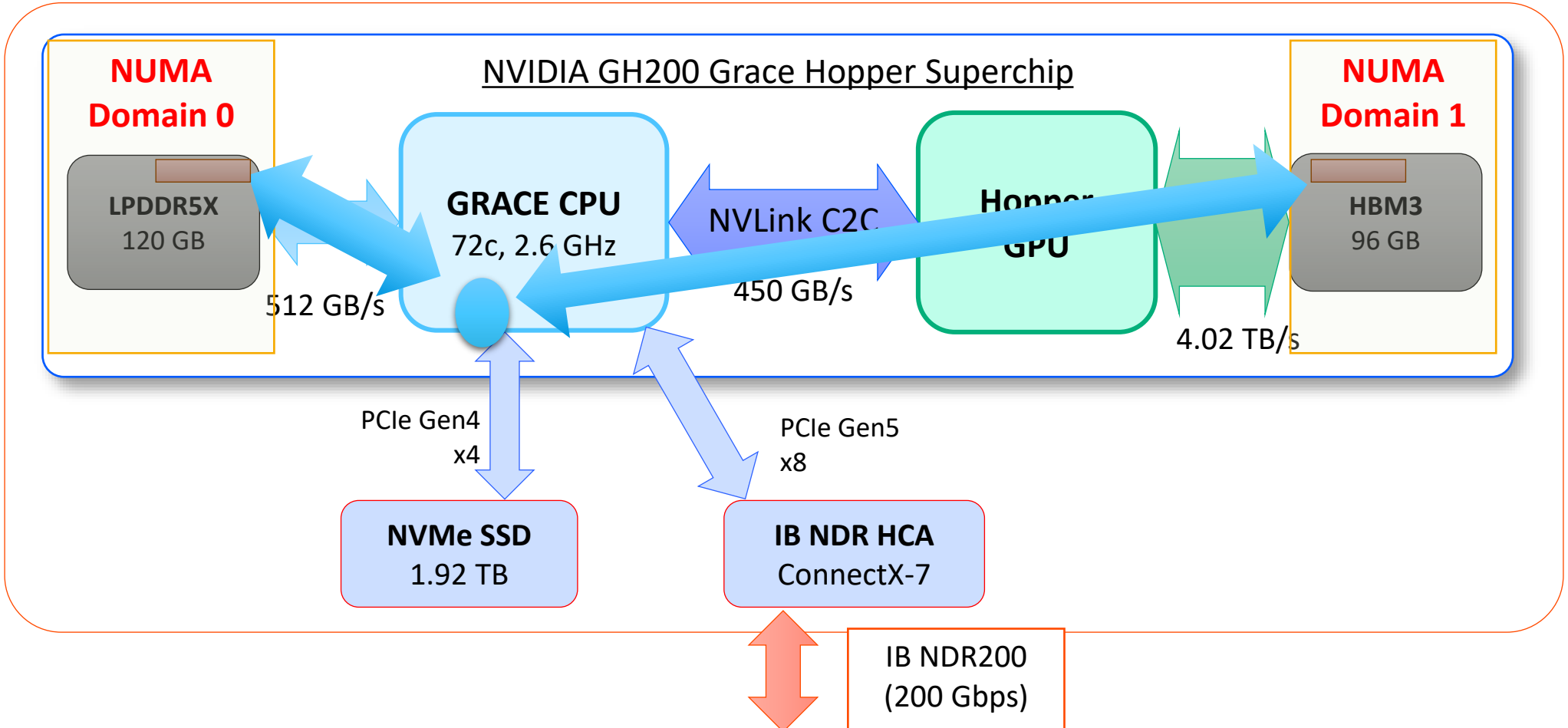
- Grace-Hopper相互のメモリ空間を直接参照可能, NUMA的な扱い
- CPU-GPU間: コヒーレントインタフェース(NVLink-C2C)
  - PCIe Gen 5の7倍以上の帯域(450GB/sec/dir)
  - CPU･GPUの効率的使い分けも可能
    - 従来はデータ転送がボトルネック
    - プログラミングも用意
    - AMD MI300Aも同じ方向性



- 小規模問題, GPUが不得意な計算をCPUが柔軟に処理することも可能

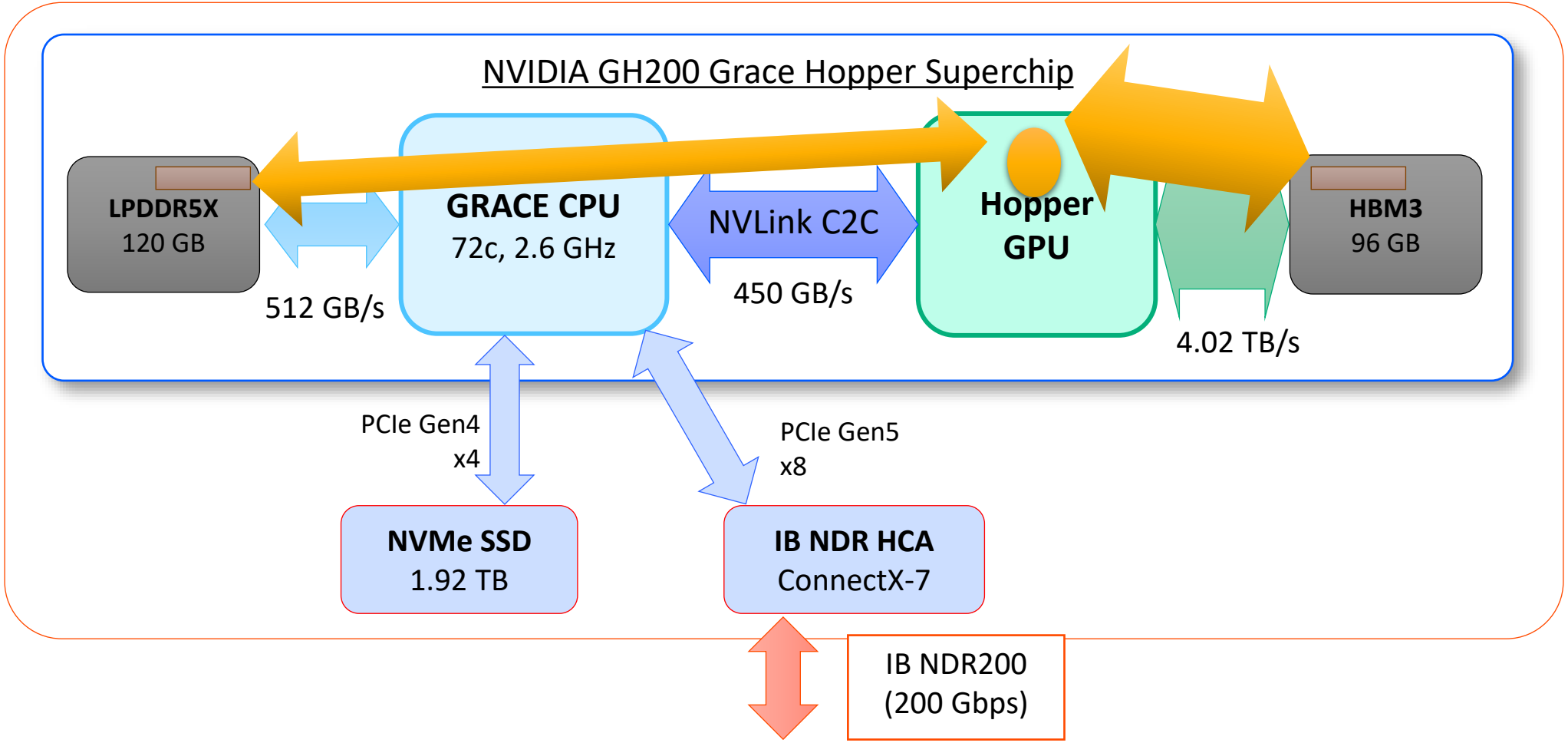
Graceからのメモリビュー

- NUMAとして見える
  - `malloc()`ではFirst Touchが大事
- 従来のCUDAのメモリモデルも使えて普通に動く(コード改変不要)＋転送が速い
  - `cudaMalloc() + cudaMemcpy()`
  - `cudaMalloc()`した領域はGraceからアクセス不可



Hopperからのメモリビュー

- Graceのアドレスを使って直接アクセス可能
- 従来のCUDAのコードはH100とほぼ挙動が変わらない（メモリバンド幅比相当の性能向上）



NVIDIA GH200のメモリアクセス

- CUDA

メモリ	メモリモード	確保される場所	Access-based Migration	CPUからアクセス	GPUからアクセス
System-allocated (malloc, new)	Unified相当	First-touch (GPU または CPU)	○	○	○
CUDA managed (cudaMallocManaged)	Managed相当	First-touch (GPU または CPU)	○	○	○
CUDA device memory (cudaMalloc)	Separate相当 (Device)	GPU			○
CUDA host memory (cudaMallocHost)		CPU		○	○

- OpenACC, OpenMP target, stdpar

メモリモード	コンパイルフラグ	デフォルトとなる環境	
Separate	`-gpu=mem:separate`	OpenACC OpenMP target	GPU上のデータはGPUからのみアクセス可能 GPU-CPU間の明示的なデータ移動が必要
Managed	`-gpu=mem:managed`	stdpar (Managed Memory のみの環境)	動的メモリ確保されたデータはGPU, CPU どちらからもアクセス可能
Unified	`-gpu=mem:unified`	stdpar (Unified Memory 対応の環境)	全てのデータはGPU, CPU どちらからもアクセス可能

MIG（Multi-Instance GPU）

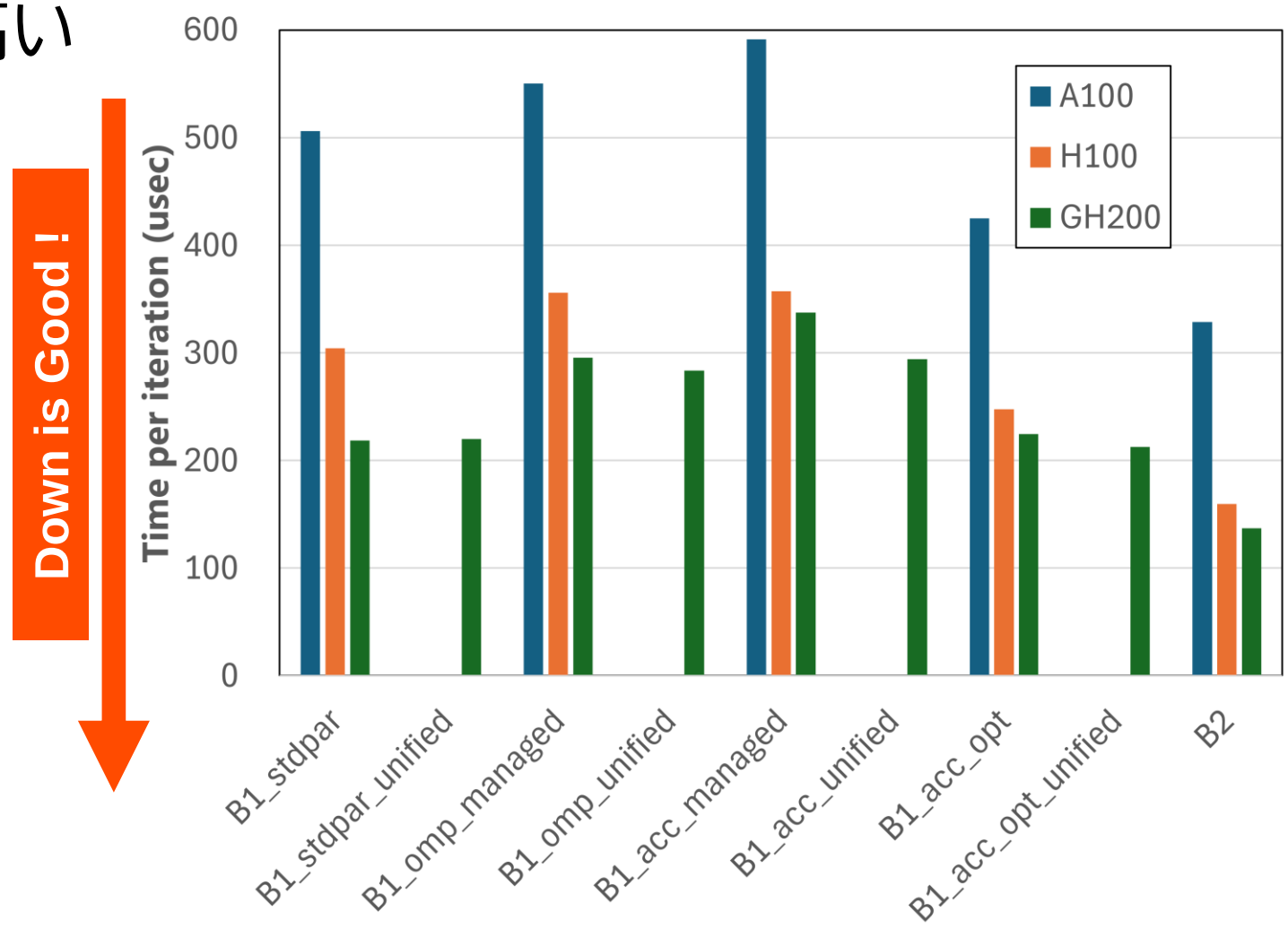
- GPUリソース（SM（演算コア），メモリ）を複数のインスタンスに分割する機能
  - https://www.nvidia.com/ja-jp/technologies/multi-instance-gpu/
- 想定される活用シーン
  - 1基のNVIDIA GH200の演算リソースを使い切れない場合
    ➔ 分割されたGPUを使えば，トークン消費量を抑えられる
  - 実装・開発中のコードのデバッグ・動作テスト・機能テスト
    ➔（見かけの）GPU数が増えるので，ジョブ投入後の待ち時間が短縮される
  - 教育利用（主にGPUプログラミングの初心者・初級者向け）
    ➔（見かけの）GPU数が増えるので，多数の受講者のジョブが同時に実行される
- Miyabi-GでのMIG利用形態（NVIDIA GH200を4分割）
  - MIG利用キュー: debug-mig, short-mig, regular-mig
    - トークン消費量は通常のノード占有キュー（debug-g, short-g, regular-g など）の 1/4
    - GPUリソース（SM数）の少ないMIG#3を回避するジョブ投入オプションはない

	MIG#0	MIG#1	MIG#2	MIG#3
GPUリソース	32 SMs, 24 GB	32 SMs, 24 GB	32 SMs, 24 GB	26 SMs, 24 GB
CPUリソース	18コア, 25 GiB	18コア, 25 GiB	18コア, 25 GiB	18コア, 25 GiB

Poisson 3D: small (128x128x128)

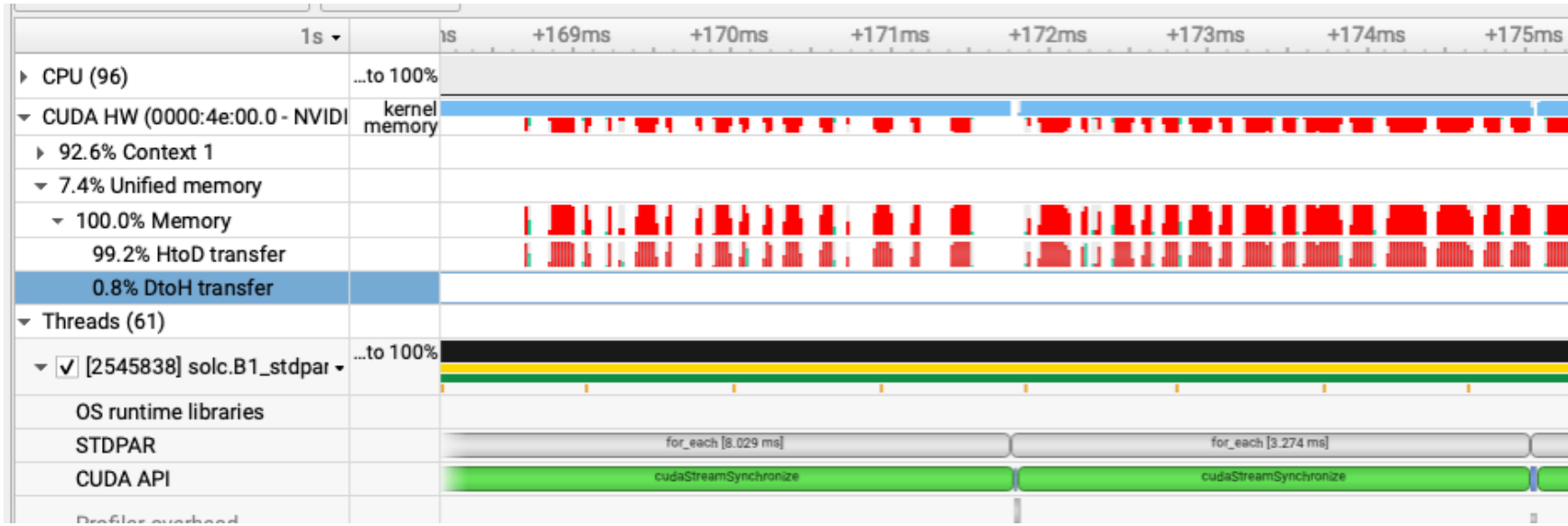
- [塙 他, 第195回HPC研究会, 2024]
- 小サイズでは, managed ➔ unified の効果が高い
- stdpar が GH200で性能が高い
  - H100比で約1.5倍
- B2 は CUDA

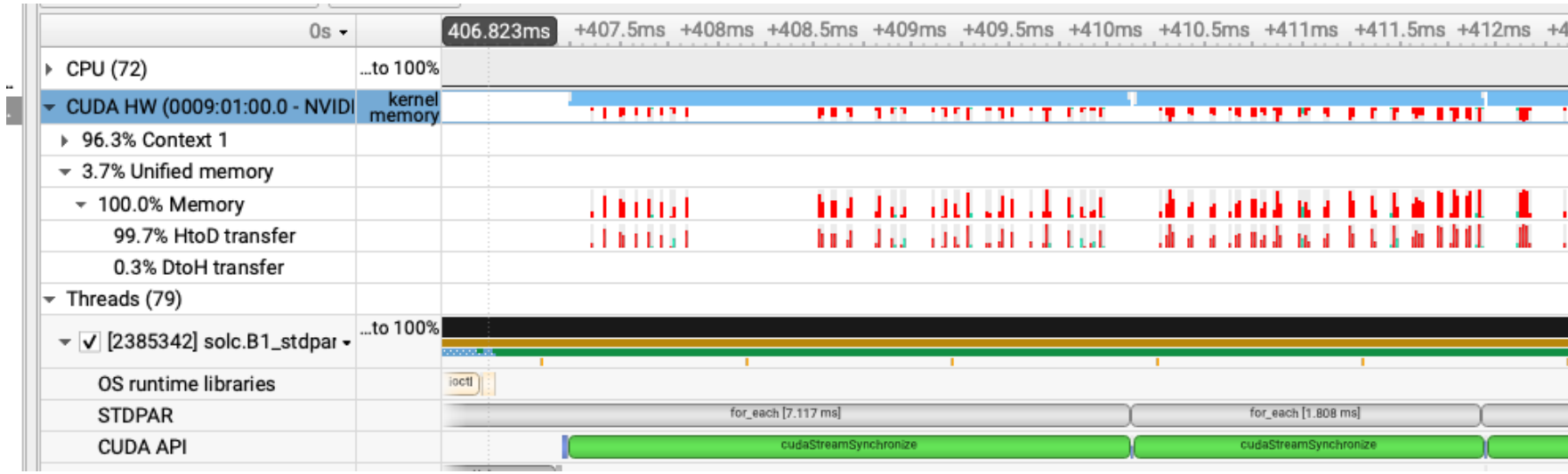
GPU	メモリバンド幅 [GB/sec]	FP64 [GFlop/s]
A100	1,555（1.00）	9,700
H100	3,352（2.16）	33,500
GH200	4,022（2.59）	33,500



Poisson 3D small B1_stdpar のプロファイル

- 上: H100 SXM5
  下: GH200
  - 赤: データ転送
- データ転送の総量はさほど変わらない
- NVLink-C2C vs PCIeは約7~10倍の性能
- 64Kページ@GH200 の効果
  - ページフォルトは1/4（ページフォルトをトリガにしてデータ転送が起こる）





Contents

- JCAHPCの新スパコンMiyabiの紹介
  - 導入経緯
  - Miyabiの概要
  - NVIDIA GH200の特性
  - Miyabi活用に向けてのGPU移植支援
- GPU向けのプログラミング環境
  - 指示文を使いたい場合
    - OpenACC or OpenMP target
    - GPU向け指示文統合マクロSolomon
  - 低レベルな開発環境を使いたい場合
    - CUDA C++, HIP C++, SYCL
    - 性能可搬フレームワークKokkos

GPU移行プラットフォームとしてのMiyabi

- 日本国内でも, GPU搭載スパコンが続々と増えている
  - TSUBAME4.0@科学大, 玄界@九大, Miyabi@JCAHPC, ABCI 3.0@AIST, …
  - 科学技術計算用のGPUスパコンとしてはMiyabiが国内最大のシステム
- GPU初心者にとって移行が一番簡単なシステムはMiyabiだったりする
  - GH200では, GPU初心者がはまりやすい罠が大幅に軽減されている
    - CPUからもGPUからも相互にメモリ空間が参照できる
    - CPU-GPU間のデータ転送が(x86-Hopperに比べて)性能ボトルネックになりづらい
  - コードを部分的にGPU移植しながら動作テストするのも比較的容易
    - CPU-GPU間のデータ転送コストが(通常のPCIe接続に比べて)大幅に軽減されている
      (コードを移植途中に「遅くなった」と思って熱が冷めるリスクが減る)
  - GPU移植しづらい部分をCPUに置き去りにしても, 性能面で問題になりづらい
- 東大情報基盤センターとしてGPUを主体とするシステムはMiyabiが初
  - したがって, ユーザコードの移植支援にも力を入れている状態
  - GPU関係の講習会, GPU移行相談会, ポータルサイトでの情報提供など
    - ユーザアカウントを持っていなくてもOK, すべて無料

GPU移植・移行の計画

- NVIDIA Japanの協力
- 3,000人以上のOFP利用者：2つの形態
- 「自己移植（Self Porting）」：様々なオプション
  - 8日間のハッカソン（ミニキャンプ），年2-3回，オンライン・ハイブリッド，Slack併用
  - 毎月開催される「相談会」（Zoom，非ユーザーも自由に参加できる）
  - 移行ポータルサイト，各種講習会
    - https://jcahpc.github.io/gpu_porting/
- 「サポート移植（Supported Porting）」，2022年10月開始
  - 多くのユーザーを有するコミュニティコード（19種類，次頁），OpenFOAM（NVIDIA）
  - 外注のための予算も確保（落札ベンダーが担当する予定）
  - 「サポート移植」グループメンバー（主に若手）はハッカソン・相談会にも積極的に参加
- 基本的にOpenACC/StdPar（Standard Parallelism）推奨

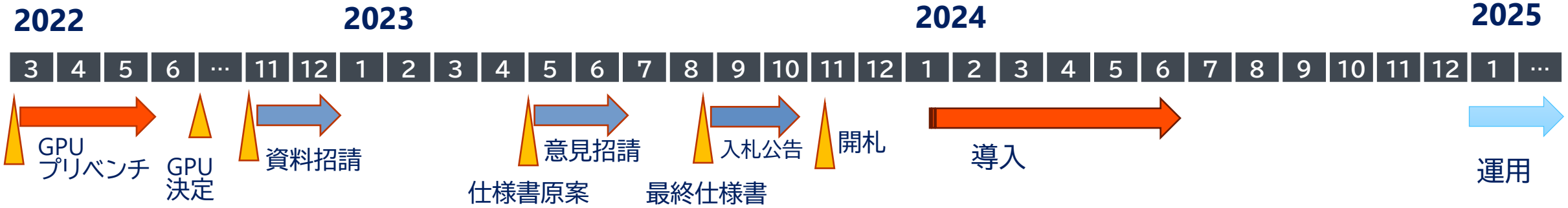
Category	Name (Organizations)	Target, Method etc.	Language
Engineering (5)	FrontISTR (U.Tokyo)	Solid Mechanics, FEM	Fortran
	FrontFlow/blue (FFB) (U.Tokyo)	CFD, FEM	Fortran
	FrontFlow/red (AFFr) (Advanced Soft)	CFD, FVM	Fortran
	FFX (U.Tokyo)	CFD, Lattice Boltzmann Method (LBM)	Fortran
	CUBE (Kobe U./RIKEN)	CFD, Hierarchical Cartesian Grid	Fortran
Biophysics (3)	ABINIT-MP (Rikkyo U.)	Drug Discovery etc., FMO	Fortran
	UT-Heart (UT Heart, U.Tokyo)	Heart Simulation, FEM etc.	Fortran, C
	Lynx (Simula, U.Tokyo)	Cardiac Electrophysiology, FVM	C
Physics (3)	MUTSU/iHallMHD3D (NIFS)	Turbulent MHD, FFT	Fortran
	Nucl_TDDFT (Tokyo Tech)	Nuclear Physics, Time Dependent DFT	Fortran
	Athena++ (Tohoku U. etc.)	Astrophysics/MHD, FVM/AMR	C++
Climate/ Weather/ Ocean (4)	SCALE (RIKEN)	Climate/Weather, FVM	Fortran
	NICAM (U.Tokyo, RIKEN, NIES)	Global Climate, FVM	Fortran
	MIROC-GCM (AORI/U.Tokyo)	Atmospheric Science, FFT etc.	Fortran77
	Kinaco (AORI/U.Tokyo)	Ocean Science, FDM	Fortran
Earthquake (4)	OpenSWPC (ERI/U.Tokyo)	Earthquake Wave Propagation, FDM	Fortran
	SPECFEM3D (Kyoto U.)	Earthquake Simulations, Spectral FEM	Fortran
	hbi_hacapk (JAMSTEC, U.Tokyo)	Earthquake Simulations, H-Matrix	Fortran
	sse_3d (NIED)	Earthquake Science, BEM (CUDA Fortran)	Fortran

GPUプログラミングに関する資料

- GPU移行に関するポータルサイト
  - https://jcahpc.github.io/gpu_porting/
- 「UTokyo N-Ways to GPU Programming Bootcamp」資料
  - https://www.cc.u-tokyo.ac.jp/events/lectures/207/
  - ISO標準言語, OpenACC, CUDA
- 「GPUプログラミング入門」資料
  - https://www.cc.u-tokyo.ac.jp/events/lectures/226/
  - OpenACC
- 「OpenMPで並列化されたC++プログラムのGPU移植手法」資料
  - https://www.cc.u-tokyo.ac.jp/events/lectures/241/
  - OpenACC, OpenMP target, ISO標準言語, CUDA C++
- 「MPI+OpenMPで並列化されたFortranプログラムのGPU移植手法」資料
  - https://www.cc.u-tokyo.ac.jp/events/lectures/230/
  - OpenACC, OpenMP target, ISO標準言語, CUDA Fortran
- 「OpenACCとMPIによるマルチGPUプログラミング入門」資料
  - https://www.cc.u-tokyo.ac.jp/events/lectures/228/
  - OpenACC+MPI

小まとめ

- JCAHPCの新スパコンMiyabi(OFP-II)が2025年1月に運用開始
  - 設置場所:東京大学 柏キャンパス(OFPと同じ部屋, 接地面積はかなり小さくなった)
  - Miyabi-G: 演算加速ノード, NVIDIA GH200 Superchip, ローカルSSD
    - 合計: 1120ノード, 78.8 PFlop/s, 5.06 PB/s
  - Miyabi-C: 汎用CPUノード, Intel Xeon Max 9480
    - 合計: 190ノード, 1.3 PFlop/s, 608 TB/s
  - ストレージ: DDN Lustre 11.3 PB, All Flash
- Miyabiの合計性能 80.1 PFlop/s: OFP (25 PFlop/s)の約3.2倍
  - HPL性能 46.8 PFlop/s (Miyabi-G): OFP (13.55 PFLop/s)の3.46倍
  - 日本のアカデミック用スパコンで2位(日本全体で4位) @ TOP500 (Nov. 2024)
  - GH200を採用する国内初のオープンシステム, 大学の運用するシステムとしては最大



Contents

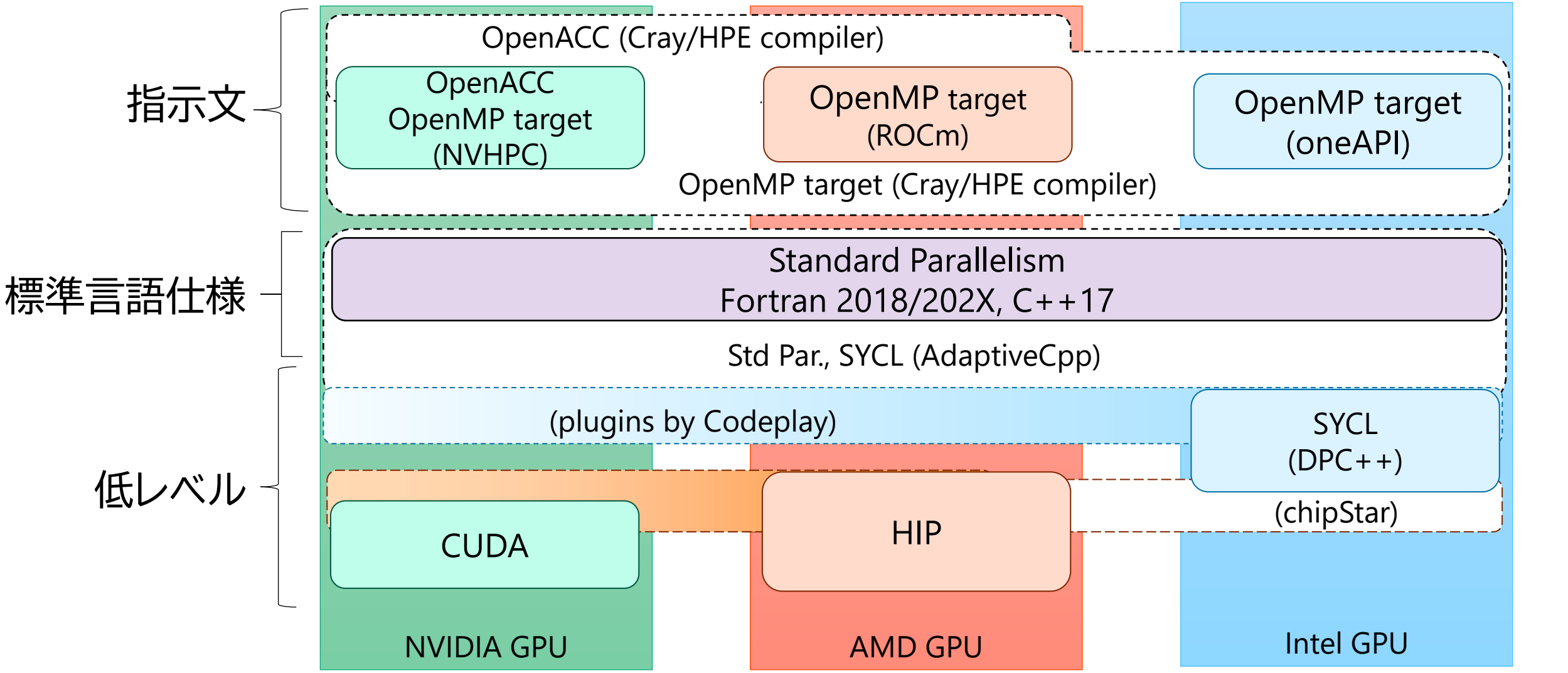
- JCAHPCの新スパコンMiyabiの紹介
  - 導入経緯
  - Miyabiの概要
  - NVIDIA GH200の特性
  - Miyabi活用に向けてのGPU移植支援
- GPU向けのプログラミング環境
  - 指示文を使いたい場合
    - OpenACC or OpenMP target
    - GPU向け指示文統合マクロSolomon
  - 低レベルな開発環境を使いたい場合
    - CUDA C++, HIP C++, SYCL
    - 性能可搬フレームワークKokkos

GPUスパコンの近況

- 「GPU = NVIDIAのGPU」と言って良いぐらいの独占状態
  - 国内では, HPCIに資源提供されている全てのGPUスパコンはNVIDIA製
    - 今年度稼働開始のTSUBAME4.0, 玄界, Miyabi も全てNVIDIA製GPUを搭載
    - (HPCIではないが)QST/NIFSが来年度導入するシステムはAMD MI300Aを搭載
  - 海外のハイエンドスパコンではAMD, Intel製GPUも採用
    - Frontier などAMD製GPUを搭載したシステム, AuroraなどIntel製GPUを搭載したシステム
    - TOP500上位にAMD, Intel, NVIDIA製GPUがランクイン
- GPUベンダー間の競争が活性化
  - NVIDIA製GPUはP100, V100, A100の間は各世代$\sqrt{2}$倍の性能向上だったが, H100では3倍強の性能向上
  - 発散するプログラミング環境への対応も必要に
- ポスト富岳は演算加速器を搭載
  - どのベンダー製の演算加速器なのか, そもそもGPUなのか, が不明な状態
  - それでもコード移植には時間がかかるので, できるだけ早くに着手したい
  - ➔ ベンダーニュートラルな実装手法に舵を切りたい, 性能を出せるかの検証も必要

GPU向けのプログラミング環境

2月のPCCC AI/HPC OSS活用WSでの講演資料
(https://www.pccluster.org/ja/event/data/240205_pccc_wsAI-HPC-OSS_06_hanawa-miki.pdf)



GPUプログラミング手法の比較表

- 独断と偏見に基づく不完全な比較表であることに注意
  - 特に Fortran はケアできていない（Fortranを読み書きできない人が作った資料）
    - Fortran対応で「OK」というのは動作するという意味

手法	ベース言語など	強み	弱み	Fortran対応	対象GPU
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	GPUFORT？ （開発停滞中？）	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, Intel （AMD: roc-stdpar?）

Fortranユーザへのメッセージ（危機感の共有）

- プロセッサベンダがFortranコンパイラを提供しなくなる時代が来るかも？
  - AMD, IntelはCUDA Fortran相当のコンパイラを提供していないというのが現状
  - 言語の設計，書きやすさ以前の問題として，コンパイラがなければどうしようもない
  - FortranサポートがNVIDIA GPU一択とすると？➔調達価格が高止まり（ベンダー間競争があれば，同予算・同一機種でもより大規模システムの導入可能性）
- 今どきのGPU向けプログラミング環境は基本的にC++ベース
  - CUDA C++, HIP C++, SYCL, Kokkos, etc.
  - 演算加速器向けにコードを書き換える「ついでに」言語も乗り換えるチャンス
    - データ構造を大幅にいじらないと性能が出ないという場合はあり，コードの大改修をする場合も
    - 大昔に設計したコードの設計見直し，アルゴリズムの改造などのタイミングでも同様
- 自分ではC++への移行はできないという方は？
  - LLMを使ってC++に変換という手法も模索されはじめている
    - ChatHPC, Code-Scribe, Fortran2CPP, CodeRosettaなど（Fortran to CUDA C++自動変換）
    - ただし，変換後のC++コードを解読できるようにはならないといけない
  - 学生さんがプログラミングを始める際に（Fortranではなく）C++を習得してもらい，協力してコード移行するということは可能では？
  - 短期間で一気に言語移行するのは困難なので，長期計画で（当然，早く着手すべきではある）
  - 新しいFortranユーザを増やすのはもうやめにしませんか？（できればコードも）
- 東大情報基盤センターとしても重要な課題と認識しているので，各種情報を収集しつつFortranからの移植をどうサポートすれば良いかを検討中
  - 現在GPU移植支援に力を入れているのと同様に，将来的にはFortranからの離脱支援も

Contents

- JCAHPCの新スパコンMiyabiの紹介
  - 導入経緯
  - Miyabiの概要
  - NVIDIA GH200の特性
  - Miyabi活用に向けてのGPU移植支援
- GPU向けのプログラミング環境
  - 指示文を使いたい場合
    - OpenACC or OpenMP target
    - GPU向け指示文統合マクロSolomon
  - 低レベルな開発環境を使いたい場合
    - CUDA C++, HIP C++, SYCL
    - 性能可搬フレームワークKokkos

指示文ベースでGPU化したい場合の選択肢

- 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色が強くなってしまった
  - 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
    - IntelはOpenACCからOpenMP targetへの変換ツールを開発中
    - ➔GPUベンダー(AMD, Intel)による直接支援が受けられない
- OpenMPのtarget指示文
  - OpenMP 4.0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いない
    - 非同期実行の(細やかな)制御など
- 実装時には, 使う指示文についても選択する必要がある

マクロを用いた指示文のブラックボックス化

- Solomon(Simple Off-LOading Macros Orchestrating multiple Notations)を実装
  - Miki & Hanawa (2024, IEEE Access)
- バックエンドで OpenACC or OpenMP target に展開
  - Fallback mode (マルチコアCPU向けのOpenMPに展開)も実装済み
- 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
  - 最適化レベルを揃えた上で OpenACC と OpenMP target の性能比較
- コンパイラではないのでベンダー製のコンパイラ性能をそのまま利用できる
  - (GPU向けプログラミングに詳しい人は)HIP の指示文版とイメージすると良い
  - 自作コンパイラの場合には, 最新機能への追随のためのコストが継続的に生じる
- 開発者が更新をさぼっても, 自分でマクロを付け足すことも簡単
  - 新コンパイラ/新指示文の実装であれば, 一般ユーザはほぼ手出しできない

Solomonを用いた実装例

- 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
- プリプロセッサマクロを用いてインターフェースを統合するライブラリを開発
  - https://github.com/ymiki-repo/solomon で公開
  - Miki & Hanawa (2024, IEEE Access)
  - 手品の種: `_Pragma()`形式で指示文を記述
  - 対応しているバックエンド:
    - OpenACC, OpenMP target, OpenMP
- どちらの手法でコードを実装しますか?
  - 通常の(煩雑な)実装方法➡
  - ⬇ Solomon を用いて簡易化した手法

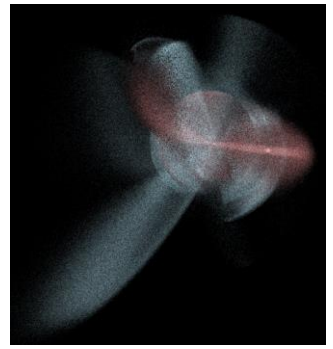
```
OFFLOAD(AS_INDEPENDENT, NUM_THREADS(NTHREADS))
for (int32_t i = 0; i < N; i++) {
```

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

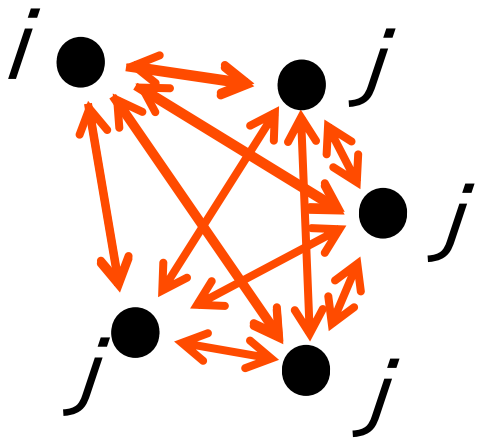
N体計算(重力多体計算)

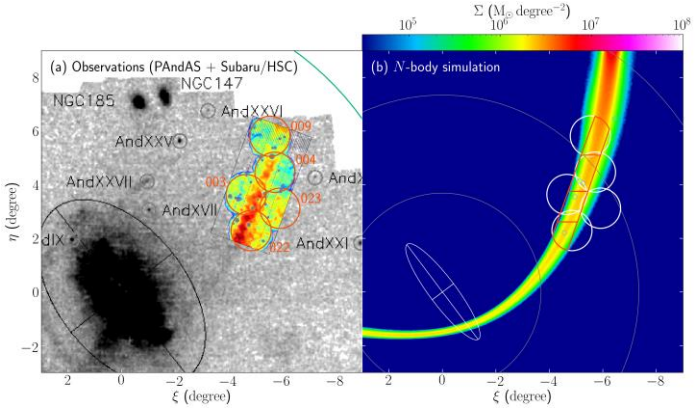
- 粒子どうしに働く自己重力による系の時間進化を，運動方程式に基づいて計算
  - データ量: $\mathcal{O}(N)$
  - 重力計算: $\mathcal{O}(N^2)$
  - 時間積分: $\mathcal{O}(N)$

$$\boldsymbol{a}_i = \sum_{\substack{j=0 \\ j\neq i}}^{N-1} \frac{G m_j \left(\boldsymbol{x}_j - \boldsymbol{x}_i\right)}{\left(\left|\boldsymbol{x}_j - \boldsymbol{x}_i\right|^2 + \epsilon^2\right)^{3/2}}$$



- N体業界的によく使う用語
  - i-粒子: 重力を受ける粒子
  - j-粒子: 重力を及ぼす粒子





- 直接法のソルバーはツリーコードのバックエンドとして使えるので，高速化しておく価値が高い(また，衝突系N体計算であれば直接法を用いる)

実装例(N体計算, 簡易記法)

- CPU上で初期条件を生成, GPUに粒子データ転送後に重力計算

```
set_uniform_sphere(num, pos, vel, Mtot, rad, virial, newton);
MEMCPY_H2D(pos [0:num], vel [0:num])
calc_acc(num, pos, acc, num, pos, eps);
```

  - MEMCPY_H2D( ) でデータ転送

- 重力計算関数(ほぼ省略版)

```
void calc_acc(…) {
  OFFLOAD(AS_INDEPENDENT, NUM_THREADS(NTHREADS))
  for (std::remove_const_t<decltype(Ni)> i = 0; i < Ni; i++) {
    // 初期化（省略）
    PRAGMA_ACC_LOOP(ACC_CLAUSE_SEQ)
    for (std::remove_const_t<decltype(Nj)> j = 0; j < Nj; j++) {
      // ループ内は省略
    }
    iacc[i] = ai;
  }
}
```

- *i*-ループを並列化
  - スレッド数はNTHREADS
- *j*-ループが並列化されると性能低下の要因となるため, 並列化を抑止(OpenACC)

実装例(N体計算, OpenACC的記法)

- CPU上で初期条件を生成, GPUに粒子データ転送後に重力計算

```
set_uniform_sphere(num, pos, vel, Mtot, rad, virial, newton);
PRAGMA_ACC_UPDATE_DEVICE(pos [0:num], vel [0:num])
calc_acc(num, pos, acc, num, pos, eps);
```

  - PRAGMA_ACC_UPDATE_DEVICE() でデータ転送

- 重力計算関数(ほぼ省略版)

```
void calc_acc(…) {
  PRAGMA_ACC_KERNELS_LOOP(ACC_CLAUSE_INDEPENDENT, ACC_CLAUSE_VECTOR_LENGTH(NTHREADS))
  for (std::remove_const_t<decltype(Ni)> i = 0; i < Ni; i++) {
    // 初期化（省略）
    PRAGMA_ACC_LOOP(ACC_CLAUSE_SEQ)
    for (std::remove_const_t<decltype(Nj)> j = 0; j < Nj; j++) {
      // ループ内は省略
    }
    iacc[i] = ai;
  }
}
```

実装例(N体計算, OpenMP的記法)

- CPU上で初期条件を生成, GPUに粒子データ転送後に重力計算

```
set_uniform_sphere(num, pos, vel, Mtot, rad, virial, newton);
PRAGMA_OMP_TARGET_UPDATE_TO(pos [0:num], vel [0:num])
calc_acc(num, pos, acc, num, pos, eps);
```

  - PRAGMA_OMP_TARGET_UPDATE_TO() でデータ転送

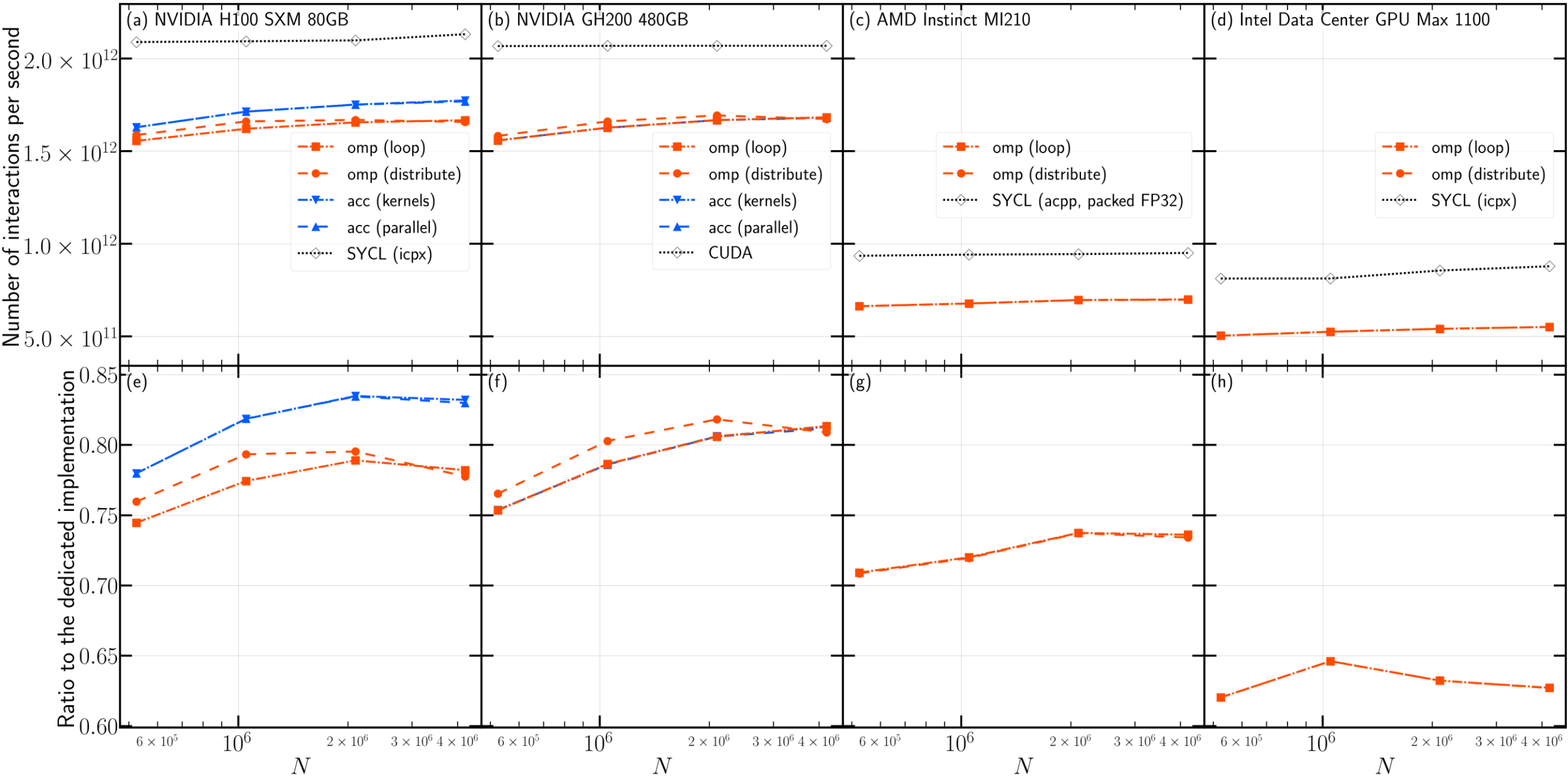
- 重力計算関数(ほぼ省略版)

```
void calc_acc(…) {
  PRAGMA_OMP_TARGET_TEAMS_LOOP(OMP_TARGET_CLAUSE_SIMD, OMP_TARGET_CLAUSE_THREAD_LIMIT(NTHREADS))
  for (std::remove_const_t<decltype(Ni)> i = 0; i < Ni; i++) {
    // 初期化（省略）
    PRAGMA_ACC_LOOP(ACC_CLAUSE_SEQ)
    for (std::remove_const_t<decltype(Nj)> j = 0; j < Nj; j++) {
      // ループ内は省略
    }
    iacc[i] = ai;
  }
}
```

計算機環境

	NVIDIA H100 SXM 80GB	NVIDIA GH200 480GB	AMD Instinct MI210	Intel Data Center GPU Max 1100
FP32性能	66.9 TFlop/s	66.9 TFlop/s	22.6 TFlop/s	22.2 TFlop/s
並列度(FP32)	16896	16896	6656	7168
動作周波数	1980 MHz	1980 MHz	1700 MHz	1550 MHz
メモリ容量	HBM3 80GB	HBM3 96GB	HBM2e 64GB	HBM2e 48GB
メモリバンド幅	3.36 TB/s	4.02 TB/s	1.64 TB/s	1.23 TB/s
TDP/TBP	700 W	1000 W (total)	300 W	300 W
ホストCPU	Intel Xeon Platinum 8468	NVIDIA Grace	AMD EPYC 7713	Intel Xeon Platinum 8468
	48 cores × 2 sockets	72 cores	64 cores × 2 sockets	48 cores × 2 sockets
	2.1 GHz	3.0 GHz	2.0 GHz	2.1 GHz
コンパイラ環境	CUDA 12.3	CUDA 12.4	ROCm 6.0.2	Intel oneAPI 2024.1.0
	NVIDIA HPC SDK 24.3-0	NVIDIA HPC SDK 24.5-1	AdaptiveCpp 24.02.0	
	Intel oneAPI 2024.1.0		LLVM 18.1.7	

N体計算の性能測定結果



実装例(3次元拡散方程式)

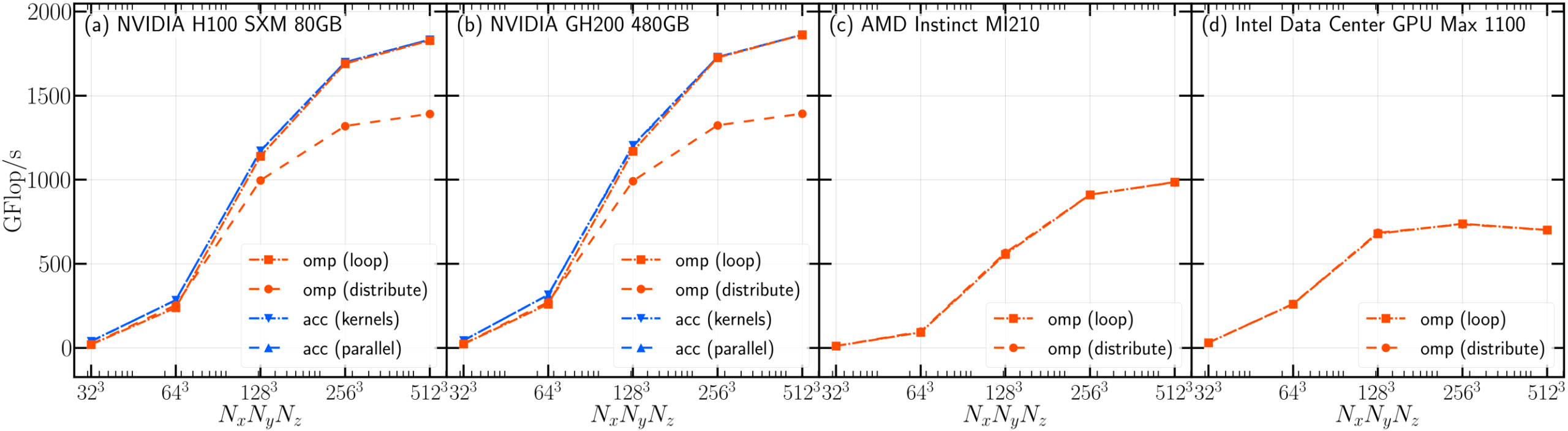
- OpenACCコードをSolomon化
  - 元コードは星野さん(名古屋大)が作ったOpenACCのサンプルコード

```
init(nx, ny, nz, dx, dy, dz, f);
PRAGMA_ACC_DATA(ACC_CLAUSE_COPY(f [0:n]), ACC_CLAUSE_CREATE(fn [0:n])) {
  for (; icnt < nt && time + 0.5 * dt < 0.1; icnt++) {
    flop += diffusion3d(nx, ny, nz, dx, dy, dz, dt, kappa, f, fn);
    swap(&f, &fn);
    time += dt;
  }
}
```

```
OFFLOAD(AS_INDEPENDENT, COLLAPSE(3), ACC_CLAUSE_PRESENT(f, fn))
for (int i = 0; i < nx; i++) {
  for (int j = 0; j < ny; j++) {
    for (int k = 0; k < nz; k++) {
      const int ix = INDEX(nx, ny, nz, i, j, k);
      const int ip = INDEX(nx, ny, nz, IMIN(i + 1, nx - 1), j, k);
      const int im = INDEX(nx, ny, nz, IMAX(i - 1, 0), j, k);
      const int jp = INDEX(nx, ny, nz, i, IMIN(j + 1, ny - 1), k);
      const int jm = INDEX(nx, ny, nz, i, IMAX(j - 1, 0), k);
      const int kp = INDEX(nx, ny, nz, i, j, IMIN(k + 1, nz - 1));
      const int km = INDEX(nx, ny, nz, i, j, IMAX(k - 1, 0));
      fn[ix] = cc * f[ix] + ce * f[ip] + cw * f[im] + cn * f[jp] + cs * f[jm] + ct * f[kp] + cb * f[km];
    }}}
```

3次元拡散方程式

- NVIDIA GPU上では, OpenMP（distribute）のみ遅い
- AMD/Intel GPUs上では, loop と distribute に性能差なし
- B/F=2.5 であり, メモリ律速なアプリケーション(キャッシュ律速)
  - NVIDIA: 4.51 TB/s (H100), 4.58 TB/s (GH200)
  - AMD MI210: 2.43 TB/s
  - Intel Data Center GPU Max 1100: 1.82 TB/s



小まとめ

- GPU向け指示文は複数あり, ユーザーはどれか1つを選択する必要がある（諸悪の根源は各社の政治的な思惑？）
  - OpenACC: 機能・資料が充実しているが, ほぼNVIDIA GPU向け
  - OpenMP target: NVIDIA/AMD/Intel 全社に対応, 機能はキャッチアップ中
- GPU向け指示文統合マクロSolomonを開発（Miki & Hanawa 2024）
  - Simple Off-LOading Macros Orchestrating multiple Notations
  - プリプロセッサマクロ経由で指示文を記載するためのマクロ集
  - 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
  - 簡易記法, OpenACC的記法, OpenMP的記法があるため, 学習コストを低減
  - GPU提供ベンダー製のコンパイラをそのまま使える
  - OpenACC と OpenMP target の性能比較も簡単にできる
- GitHub で公開: https://github.com/ymiki-repo/solomon
  - 今は C/C++ のみ対応. Fortran向けインタフェースを作ってもらう計画

Contents

- JCAHPCの新スパコンMiyabiの紹介
  - 導入経緯
  - Miyabiの概要
  - NVIDIA GH200の特性
  - Miyabi活用に向けてのGPU移植支援
- GPU向けのプログラミング環境
  - 指示文を使いたい場合
    - OpenACC or OpenMP target
    - GPU向け指示文統合マクロSolomon
  - 低レベルな開発環境を使いたい場合
    - CUDA C++, HIP C++, SYCL
    - 性能可搬フレームワークKokkos

指示文では記述力・性能が不十分という人は

- 十分な記述力・限界性能を追求したい場合: 低レベルな開発環境を使用
  - CUDA C++: NVIDIA製GPU専用の開発環境
  - HIP C++: NVIDIA, AMD（chipStar を使えば Intel 対応も可能?）
  - SYCL: NVIDIA, AMD, Intel（Intel oneAPI + Codeplay, AdaptiveCpp）
- 他の手段は？
  - 性能可搬フレームワークKokkos, RAJAなど
- 低レベルな開発環境はほぼC++ベースで提供されている
  - 例外はCUDA Fortran
    - これももともとPGIが(NVIDIAと共同で)開発したもの
    - PGIはNVIDIAに買収された
  - Fortran縛りの場合には, 指示文＋ライブラリ活用でカバーできないときにはC++ベースの開発環境との組み合わせが必要になってくる
    - どうせC++が混ざってしまうならば, 全体をC++化してしまうことを検討したくなったりする？（C と C++ を混ぜてもメモリレイアウトなどは共通だが, FortranとC系を混ぜるとループの回し方をかなり気をつけないといけない）

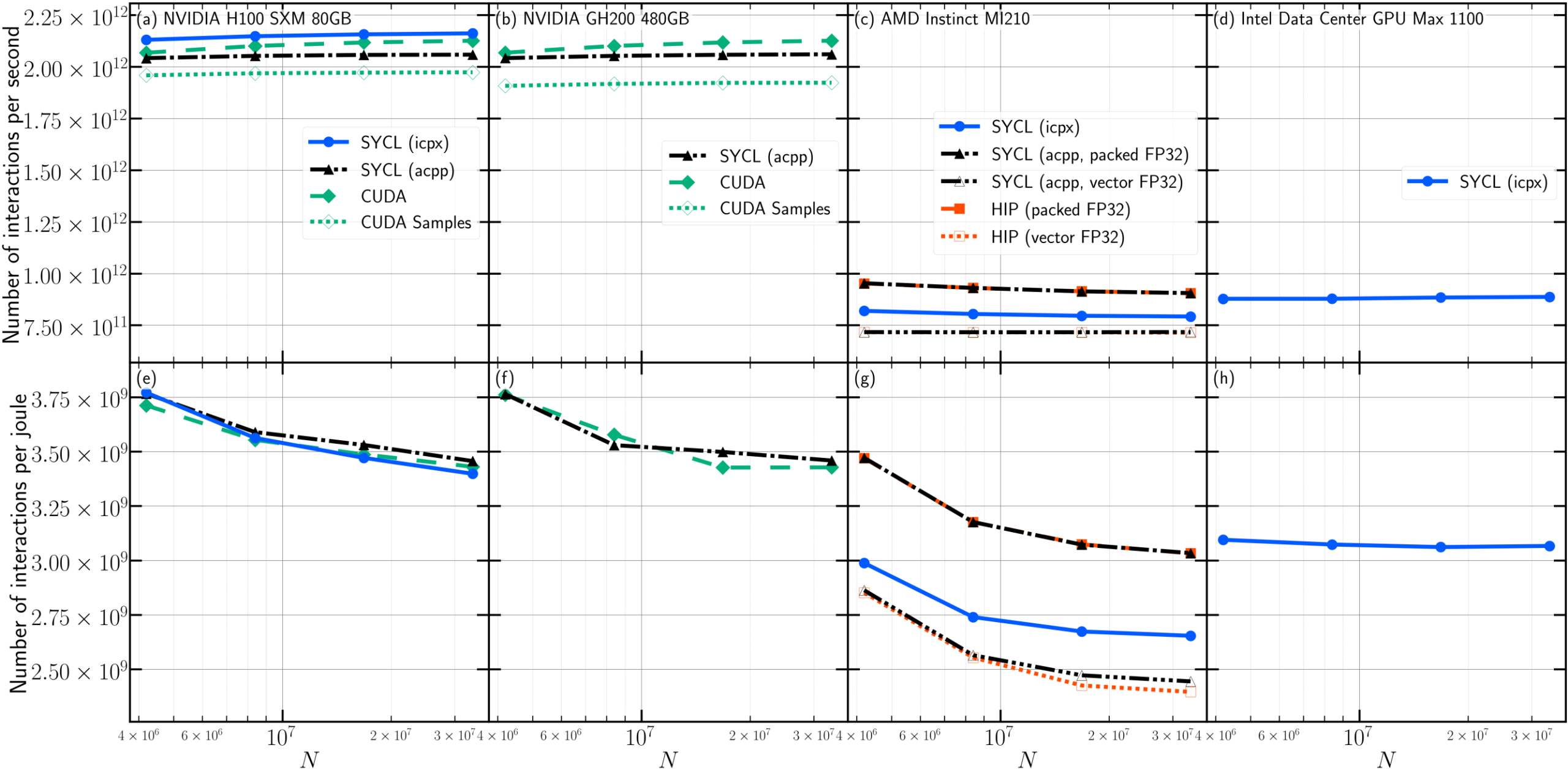
コードの実装手順・勉強手順（CUDAは起点にもなる）

- CUDA C++版
  - 簡単なのでスクラッチから実装（注：このスライドを作った人はCUDA歴10年以上）
- HIP C++版
  - CUDA版のうちいくつか簡単なものを hipify-clang を用いて変換
    （シェアードメモリの使い方など，ドキュメントを眺めるよりも簡単に使い方が分かる）
  - 感触をつかんだ後は，普通にスクラッチから実装できるようになる
- SYCL版
  - CUDA版のうちいくつか簡単なものを dpct（今はSYCLomatic）を用いて変換
  - デフォルトでは各queueが Out-of-Order 実行されるので適宜 `wait()` をかける
  - `sycl::property::queue::in_order` を指定して queue を作成すると，
    CUDAのdefault streamを使ったときと同じ振る舞い
    - CUDAに慣れている人にとってはこちらの使い方のほうが馴染みやすいと思われる
  - 感触をつかんだ後は，普通にスクラッチから実装できるようになる
- 注：「簡単」，「普通に」などはあくまでも個人の感想なので保証はしません

SYCL環境の整備方法

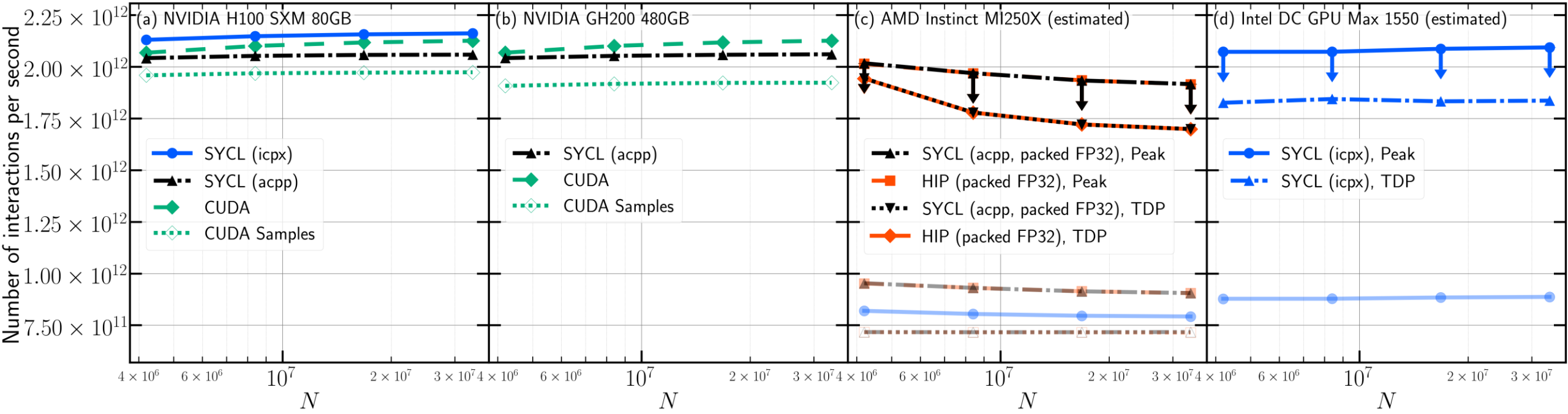
- Intel oneAPI（コンパイラは `icpx`）
  - Codeplay 社提供のプラグインをインストール（for NVIDIA/AMD GPUs）（https://codeplay.com/solutions/oneapi/plugins/）
- AdaptiveCpp（コンパイラは `acpp`）（https://github.com/AdaptiveCpp/AdaptiveCpp）
  - ドキュメントにしたがってインストールすれば特に苦労なく使えた
    - Intel GPU 向けのドキュメントが見当たらなかったので, NVIDIA/AMD GPUs 限定の話
    - NVIDIA GH200 向けにもインストールできた（Arm CPU なので, oneAPI が使えない）
      - ただし, `nvclang`をバックエンドに取ることはできなかった（`llvm-tblgen`が不足）
  - いくつかのインストール方法が提示されているが, 試したのは以下の手順
    1. LLVM を手でインストール（NVPTX or AMDGPU を有効にしておく）
       - ドキュメントでは LLVM は dnf install することを推奨されていたが, 管理者権限なしでインストールできる方法を取ることにした（スパコンに一般ユーザとしてインストールして使うことを想定した予行演習）
    2. AdaptiveCpp のソースを取ってきて, cmake してコンパイル
  - レジスタを大量に消費する場合に計算がこける場合がある
    - スレッド数が512以上かつILP数8以上というかなり極端な条件なので, 普通は出くわさないはず

NVIDIA/AMD/Intel製GPUの性能比較



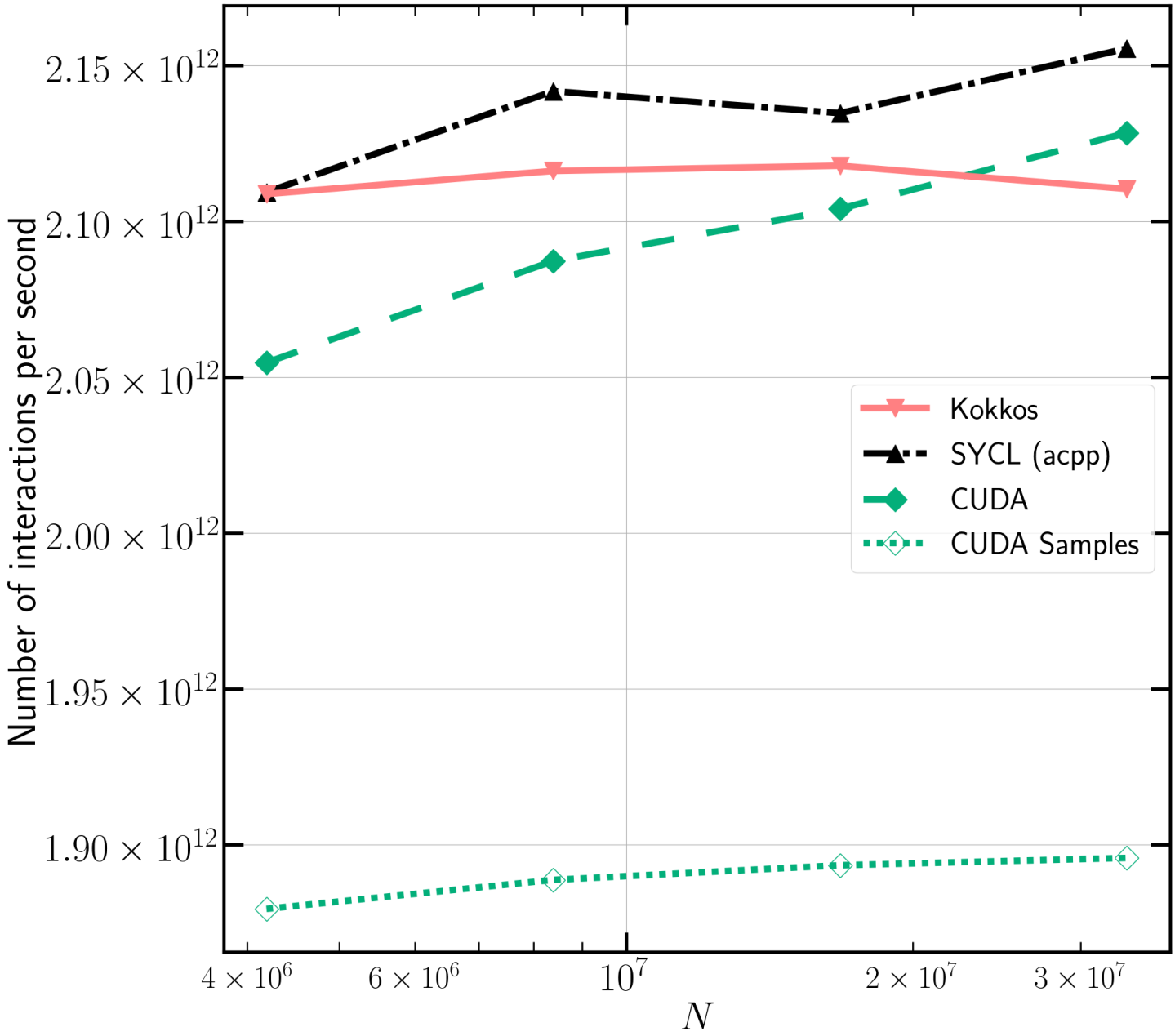
各世代の最上位製品どうしの比較

- AMD MI210, Intel Data Center GPU Max 1100 は最上位製品ではないので, 最上位製品を用いた際の性能を推定
  - 理論ピーク性能に基づく推定：実際には供給電力が不足するので上限値（下矢印つき）
    - AMD MI250X: MI210の2.12倍の理論ピーク性能, 1.87倍のTDP（注：MI210でも不足）
    - Intel Data Center GPU Max 1550: 1100の2.36倍の理論ピーク性能, 2倍のTDP
  - 消費電力に基づく推定：電力性能が変わらなければこちらの方が正解に近いはず
    - 電力性能は動作周波数に依存するため, （動作周波数が下がると）推定値より上がることもある



Kokkos との性能比較(preliminary)

- 欧米では性能可搬フレームワークKokkosがよく使われている
  - CMakeの使い方に一癖あり,(CMakeにはそれなりに慣れていたが)CMakeまわりの調整に一番時間がかかった
- Miyabi-G 上での測定
  - NVIDIA GH200 120GB
  - CUDA 12.6
  - AdaptiveCpp 24.10.0
    - LLVM 19.1.7
  - Kokkos 4.5.99
    - `nvcc_wrapper` は手直しした
- (なぜか)SYCL (acpp) が一番速い
  - Kokkos も良い勝負
  - Intel oneAPI が使えれば……



小まとめ

- CUDA/HIP/SYCL を用いてN体計算コードを実装・最適化し, NVIDIA/AMD/Intel製GPU上での性能を比較した
  - Kokkos についてもMiyabi-Gで実験開始. 今のところ良好な成功を確認
- SYCL実装の性能が良好
  - 全ベンダー製GPUに対応でき, なおかつきちんと性能を出せる
    - Intel oneAPI と AdaptiveCpp を使い分けられることも大きい
    - NVIDIA GH200 上では AdaptiveCpp が使える
  - NVIDIA H100, Miyabi-G 上では CUDA よりも速かった(!!)
  - AMD MI210 上では HIP とほぼ同性能
- NVIDIA Hopper, AMD CDNA 2, Intel PVC 世代においては NVIDIA H100 (SYCL, icpx) が最高性能となった
  - 電力性能についても同様(CPU, 冷却などの寄与は入っていない点に注意)
  - 使用電力は600 Wを越えており, こちらも最大
    (電力性能が一番高く, かつ供給電力も最大なので, 性能も最高値となる)
- 次(or 最新)の世代ではどうなる?(AMD MI300, NVIDIA B200, …)

まとめ

- JCAHPCの新スパコンMiyabi(OFP-II)が2025年1月に運用開始
  - Miyabi-G: 演算加速ノード, NVIDIA GH200 Superchip, ローカルSSD
  - Miyabi-C: 汎用CPUノード, Intel Xeon Max 9480
- Miyabiの合計性能 80.1 PFlop/s: OFP (25 PFlop/s)の約3.2倍
  - 日本のアカデミック用スパコンで2位(日本全体で4位) @ TOP500 (Nov. 2024)
  - GH200を採用する国内初のオープンシステム, 大学の運用するシステムとしては最大
  - 皆様のサイエンスを加速するためのプラットフォームとしてご活用ください
- NVIDIA GH200はGPU初心者にも優しい環境
  - ポスト富岳など将来を見据えてのGPU移植プラットフォームにもご活用ください
- (ベンダーニュートラル視点も取り入れた)GPU向けプログラミング環境
  - GPU向け指示文統合マクロSolomonを開発(Miki & Hanawa 2024)
    - プリプロセッサマクロ経由で指示文を記載するためのマクロ集
    - 簡易記法, OpenACC的記法, OpenMP的記法があるため, 学習コストを低減
    - GitHub で公開: https://github.com/ymiki-repo/solomon
  - SYCLは(direct N-bodyでは)NVIDIA/AMD/Intel GPU上で高い性能を発揮
    - Kokkos についても評価を開始. Miyabi-G(NVIDIA GH200)上では良好な性能を発揮