N体シミュレーションコードの GPU 実装 --これまでとこれから--

三木 洋平
（東京大学 情報基盤センター）

GRAPEには大変お世話になりました

- 卒論発表会のスライド→
（当時は筑波大宇宙理論）
- 初めての研究でGRAPE使用
- FIRSTの64ノードキューを常時使っていたはず
- 良いものを作っていただいて＋使わせていただいてありがとうございました
- GRAPE慣れしていたおかげでGPU実装にすんなり移れた
  - 使い方には類似点あり
  - パラメータ設定の方向性は逆

宇宙シミュレータ ”FIRST”

- FIRSTクラスタ
  - 全256ノード
    （ 496 CPU ＋240 Blade-GRAPE）
  - PCクラスタ 約3.1TFLOPS ,
    Blade-GRAPE 約33TFLOPS
- Blade-GRAPE
  - 組み込み型重力計算専用ボード
  - 理論ピーク性能　136.8 GFlops

GPU(Graphics Processing Units)

- 元々は画像処理を行うために特化していた専用プロセッサを汎用計算にも使えるように拡張
  - 最近のGPUには(主に深層学習用に)行列積向けユニットも
- 高い演算性能, 太いメモリバンド幅, 消費電力あたり性能も高い
  - 大昔は安かったが, 最近はとてもとても高価
- 多数のコア(最新GPUは1万越え)を搭載した並列計算機
  - NVIDIA A100 (SXM, PCIe): $64 \times 108\ \text{SMs} = 6912\ \text{cores}$
  - NVIDIA H100 (SXM): $128 \times 132\ \text{SMs} = 16896\ \text{cores}$
  - NVIDIA H100 (PCIe): $128 \times 114\ \text{SMs} = 14592\ \text{cores}$
  - AMD MI250X: $64 \times 110\ \text{CUs} \times 2\ \text{GCDs} = 14080\ \text{cores}$
  - AMD MI250: $64 \times 104\ \text{CUs} \times 2\ \text{GCDs} = 13312\ \text{cores}$
- スレッド並列(e.g., OpenMP)的考えが分かっていると良い

(NVIDIAの)GPUの構成(1/2)

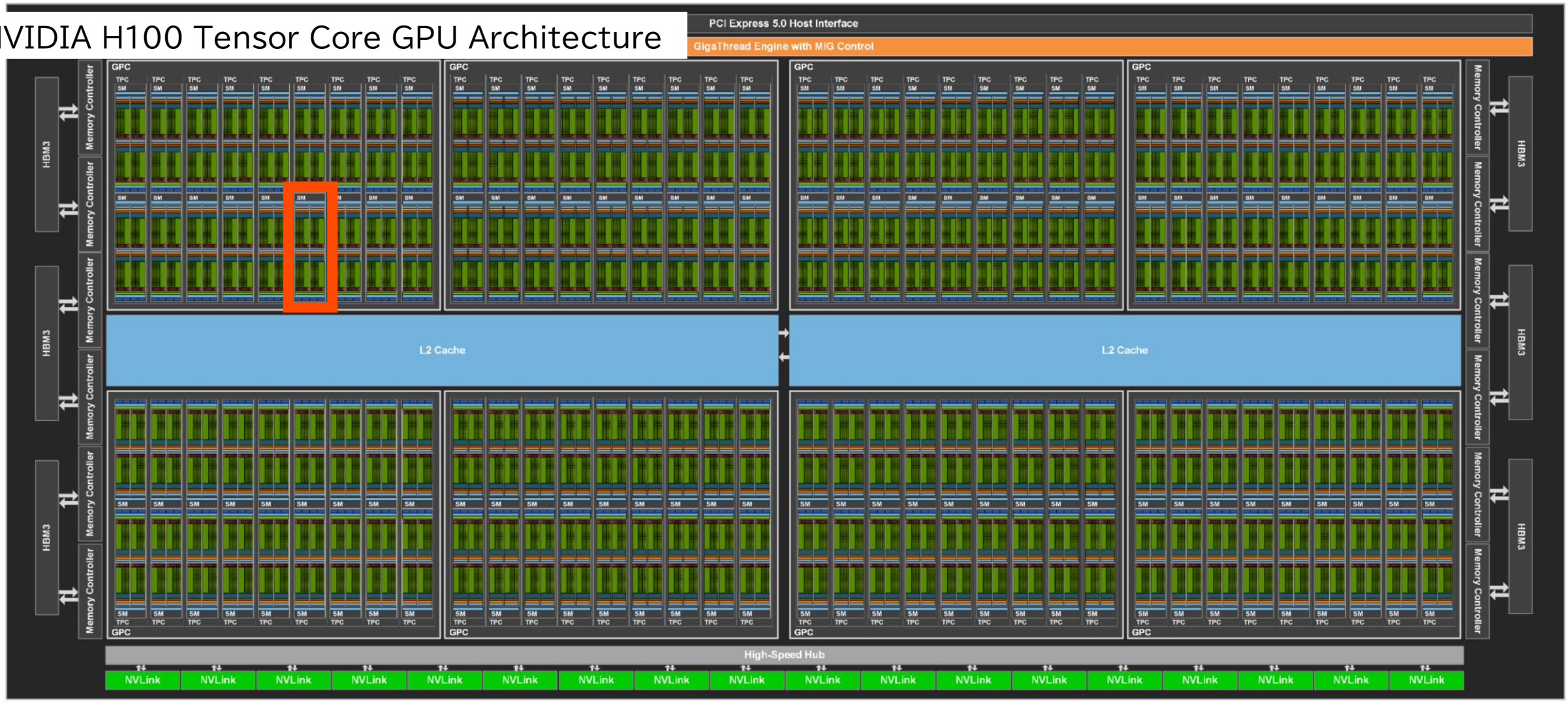
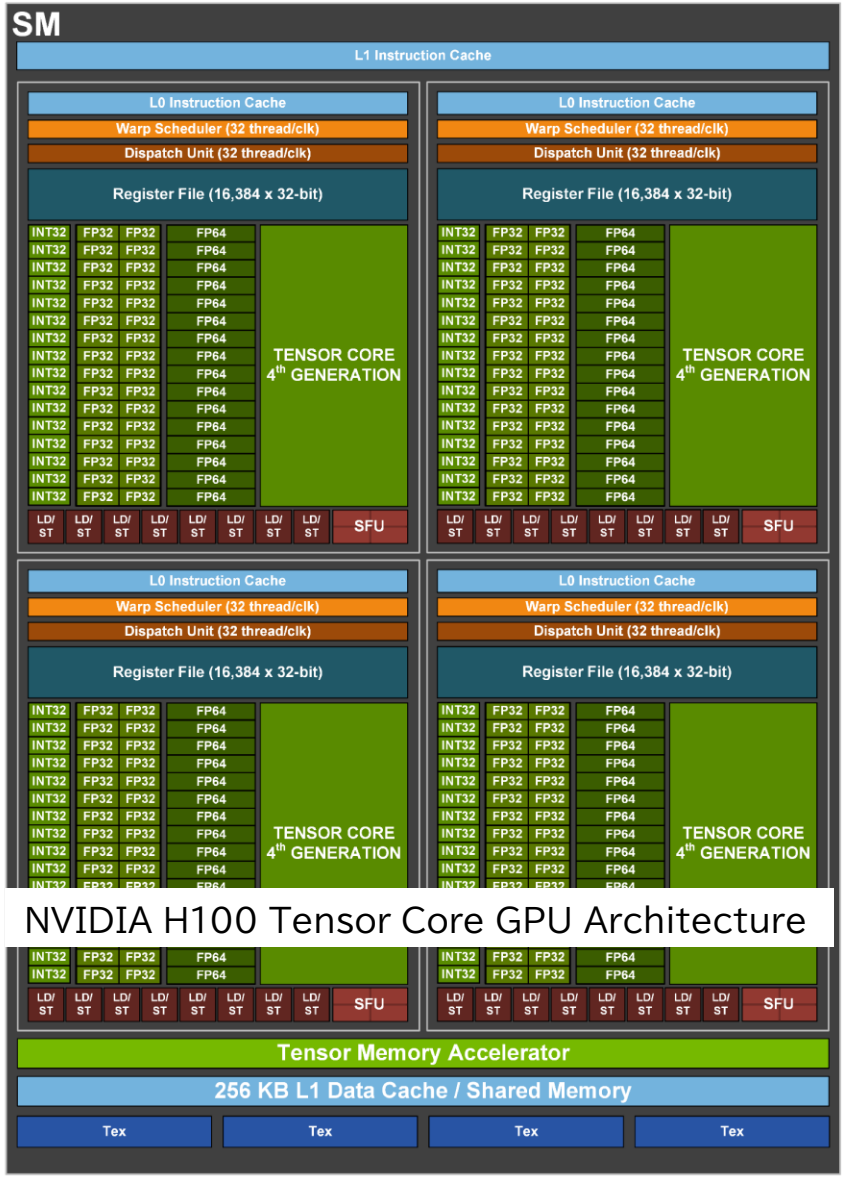


Figure 6. GH100 Full GPU with 144 SMs

(NVIDIAの)GPUの構成(2/2)

- SM: Streaming Multiprocessor
- SM内の構造は気にしなくても性能を出せる
- コア数よりも多くのスレッドを立てて計算
  - 各種レイテンシを隠蔽するため
  - 目安:コア数の10倍以上のスレッド数
- スレッドブロックが基本単位
  - ブロックあたりのスレッド数は32の倍数が標準
    - 32スレッド単位のグループ(ワープ)で動作
    - ワープ内の処理が分裂しないように実装
  - スレッド数を128以上にしておくのが推奨
  - SMに複数ブロックを割り当てることがほとんど



NVIDIA H100 Tensor Core GPU Architecture

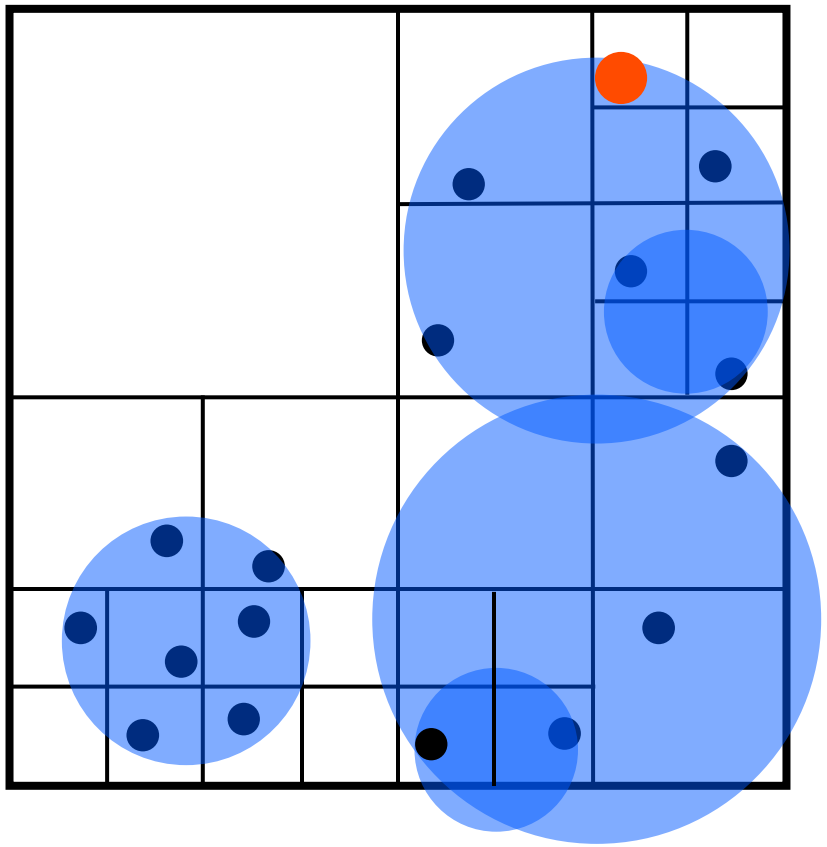
Figure 7. GH100 Streaming Multiprocessor (SM)

N体計算の高速化

- Tree法(Barnes & Hut 1986)がよく使われる
  - 多重極展開を用い, 実効的なj-粒子数を減らす
  - $\mathcal{O}(N_i N_j)$ ➔ $\mathcal{O}(N_i \log N_j)$
  - GADGET criterionを採用 (Springel 2005)

    $$\frac{Gm_J}{d_{iJ}^2}\left(\frac{b_J}{d_{iJ}}\right)^2 \leq \Delta_{\mathrm{acc}}\left|\boldsymbol{a}_i^{\mathrm{old}}\right|$$

    (重力多体系は負の質量がないので, 誤差のleading termは四重極モーメント)
  - GPU上での高速化も可能 Nakasato12, Ogiya+13, Bedorf+12, 14 etc.

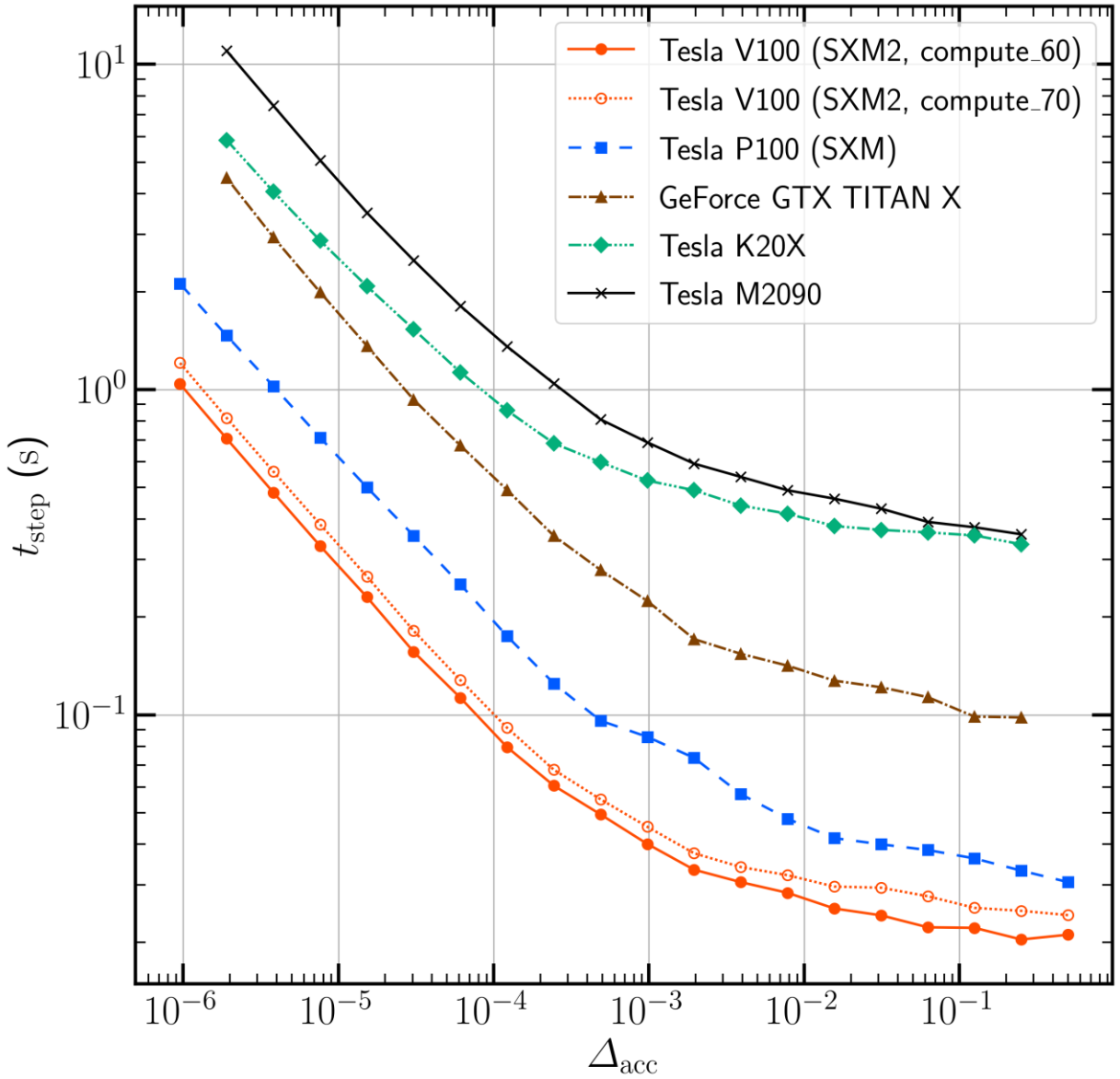


GPU向けツリーコードの実装方針

- 全ての主要な関数をGPU上で動作させる
  - CPU-GPU間のデータ転送は高コスト：CPUが何もしなければ転送不要
  - ツリー構造の構築を並列実行する都合上, 幅優先探索を採用
  - 必然的に, ハードウェアにかなり寄り添った実装
  - （寄り添いすぎたせいで, CUDAの仕様変更まわりで苦労したことも..: Volta世代で導入されたIndependent thread scheduling）
- 「ワープ分裂」を起こさないように重力計算部分を実装
  - MAC判定時に重力計算 or もう一段潜るという判断がスレッド毎に違う
  - シェアードメモリ上に相互作用リストとツリー判定待機リストを作り, 両リストを並列更新
  - 内部的にはscanを大量に発行する実装とし, ワープ分裂を最小化
  - 結果的に, 整数演算がかなり増加

重力ツリーコードGOTHIC

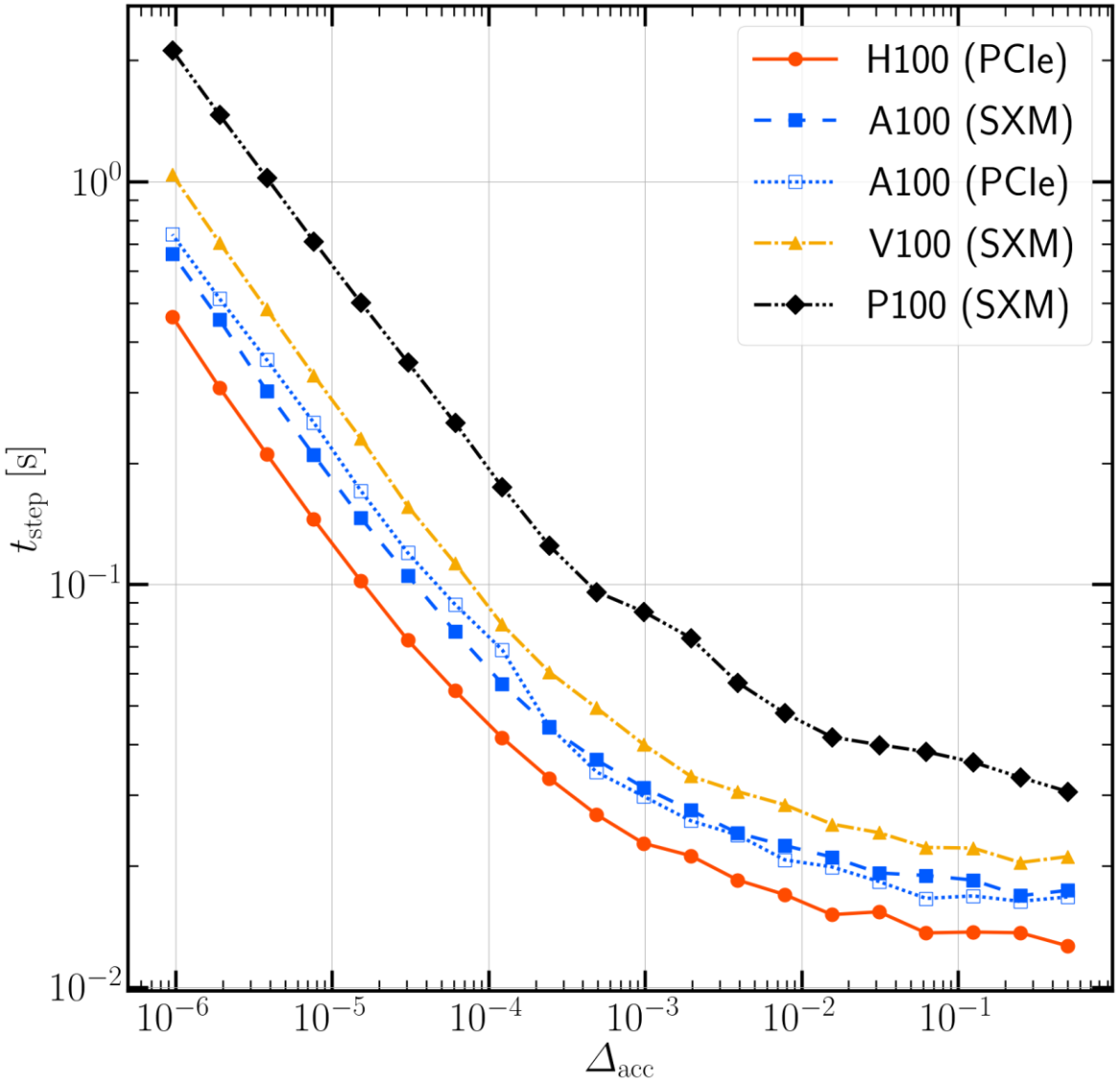
- Gravitational Oct-Tree code accelerated by HIerarchical time step Contolling
  - YM & Umemura (2017)
  - YM (2019)
- Block time stepを採用
- 幅優先探索を採用
- 動的な最適化を実装
- (V100までは) GPUの世代が進むごとにおおむね2倍程度ずつ高速化
  - 右図:M31 model, N = 8M
    - 初期条件生成用に MAGI (YM & Umemura 2018) を作った

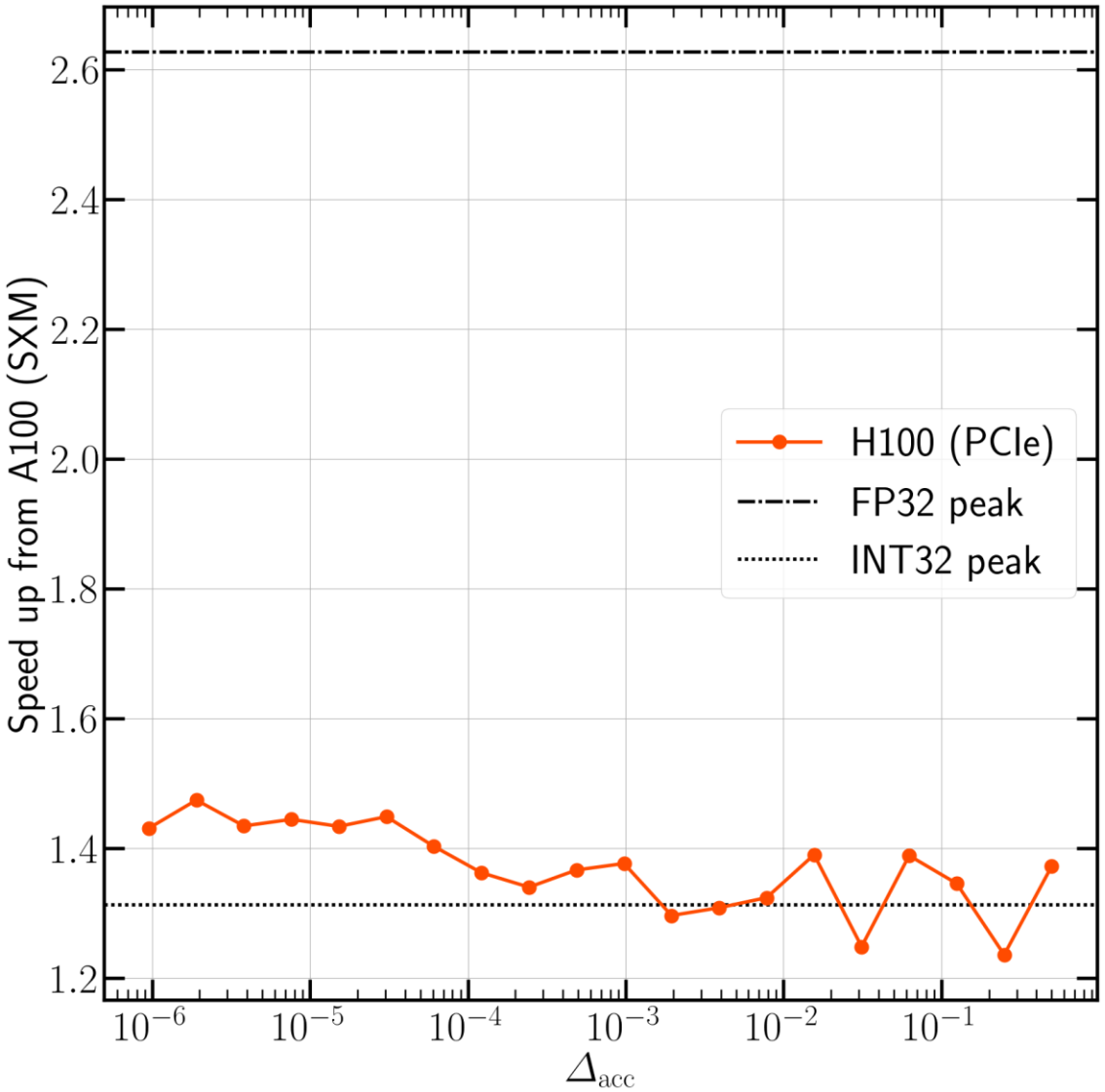


整数演算が多い実装になった影響

- P100 ➔ V100では理論ピーク性能比より高い性能向上
  - 最大2.2倍速くなったが, 理論ピーク性能比は1.5倍
  - NVIDIA Pascal世代以前: CUDAコアが浮動小数点演算と整数演算両方を処理していた
  - NVIDIA Volta世代以降: FP32コアとINTコアに分離されたため, 浮動小数点演算と整数演算の同時実行による高速化が起こりうる
- A100 ➔ H100 PCIeでは理論ピーク性能比より低い性能向上
  - 理論ピーク性能比2.6倍に対して, 速度向上率は1.4倍程度にとどまる
  - NVIDIA Ampere世代: SM内のFP32コアとINTコアの数比は1:1
  - NVIDIA Hopper世代: SM内のFP32コアとINTコアの数比は2:1
  - H100では相対的に整数演算性能が下がっている

H100 PCIe(Pegasus@筑波大CCS)





GPU搭載型スパコンの近況

- GPUは最近のスパコンでのメジャーな演算加速器
  - GPUは内部に多数のコアを搭載した並列計算機
- 「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
  - 欧米でもNVIDIA製GPUを搭載したシステムが多数派
    - Frontier などAMD製GPUを搭載したシステムの台頭
    - Intel製GPU搭載システムもそろそろ？（Auroraは物理構築完了とのこと）
- 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√2倍の性能向上だったが, H100では3倍の性能向上
  - （価格は下がってほしいところだが..）
  - 新GPU発表時には, 性能向上だけではなく, 新機能が提供されることも
  - ➔それぞれのGPU向けに, どう最適化すれば良いのかを調べる必要

AMD GPU向けの実装方法は？

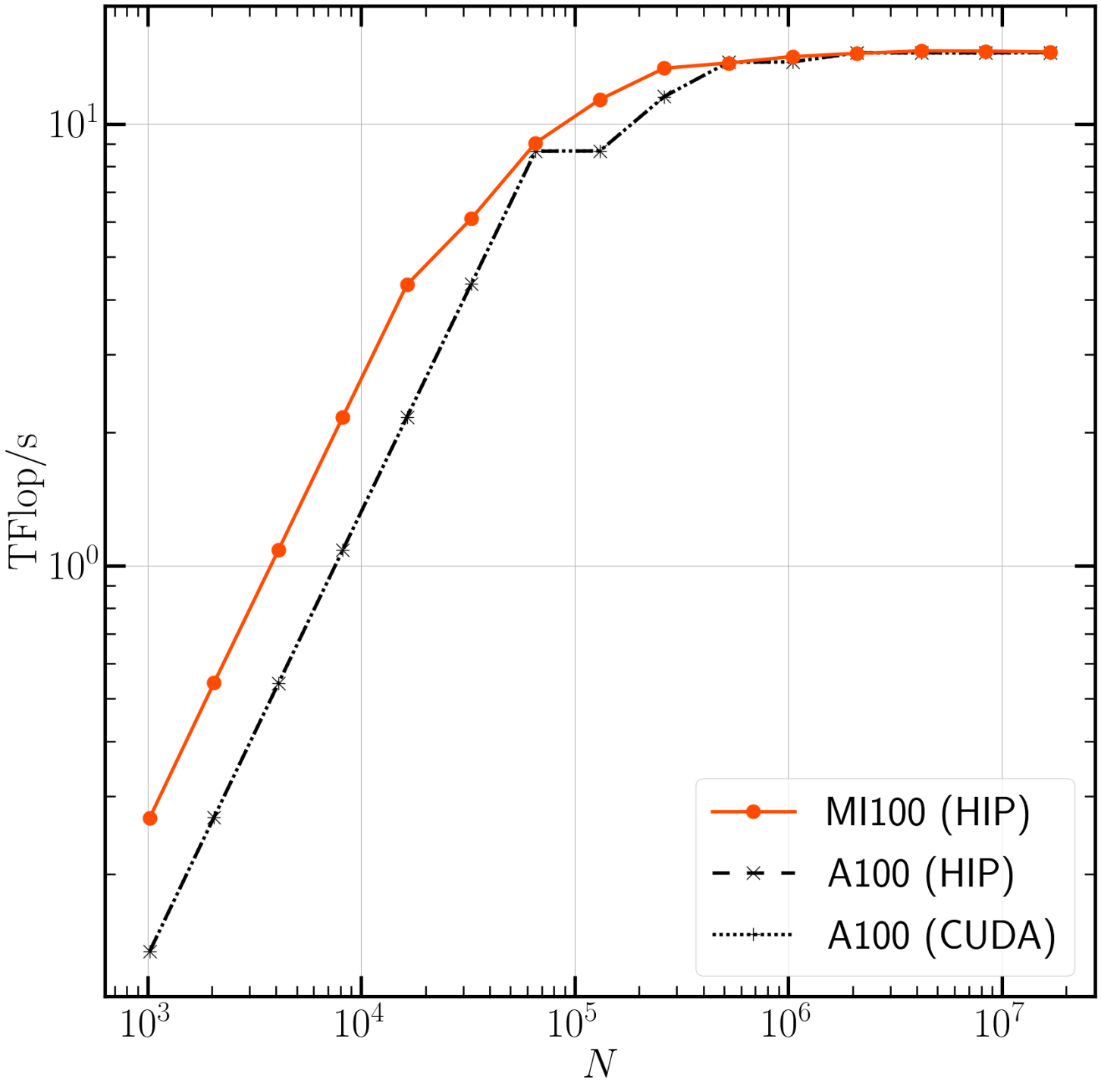
- HIP(Heterogeneous-Compute Interface for Portability)が提供されている
- HIP C++はCUDA C++に非常によく似ている
  - ほぼCUDAを書いている感覚(特にデバイス関数実装時)
- NVIDIAのGPUでもAMDのGPUでも走るコードが実装可能
  - NVIDIA GPU上では, バックエンドでCUDAを使うのでほぼ同じ性能
  - Intel GPU上でも動いて欲しい(HIPLZ, HIPCLの今後に期待？)
- CUDAからの移植ツール(HIPIFY)も提供されている
  - Hipify-clang(コンパイラ的ツール) が推奨とのこと
  - Hipify-perl (簡易ツール)もある

逆数平方根の計算命令(HIP実装版)

- 以下の3通りで計算速度を比較(direct N-bodyの性能)
  1. `1.0F / sqrtf(r2);` 5.3 TF @ MI100, 8.3 TF @ A100
  2. `rsqrtf(r2);` 11.3 TF @ MI100, 13.5 TF @ A100
  3. `__frsqrt_rn(r2);` 14.4 TF @ MI100, 8.7 TF @ A100
     - 組み込み関数, round-to-nearest-even mode (IEEE-compliant)
  - 注:相互作用当たり22 Flopsを仮定, 演算は全て単精度, N = 4M
- A100では`rsqrtf()`が良い(CUDAでの実装でも同じ)
  - SFUで計算されるので, 想定通りの結果
- MI100では`__frsqrt_rn()`が良い
- NVIDIA・AMD向けでコードを切り替える方法
  - `#ifdef __HIP_PLATFORM_NVIDIA__`
  - `#ifdef __HIP_PLATFORM_AMD__`

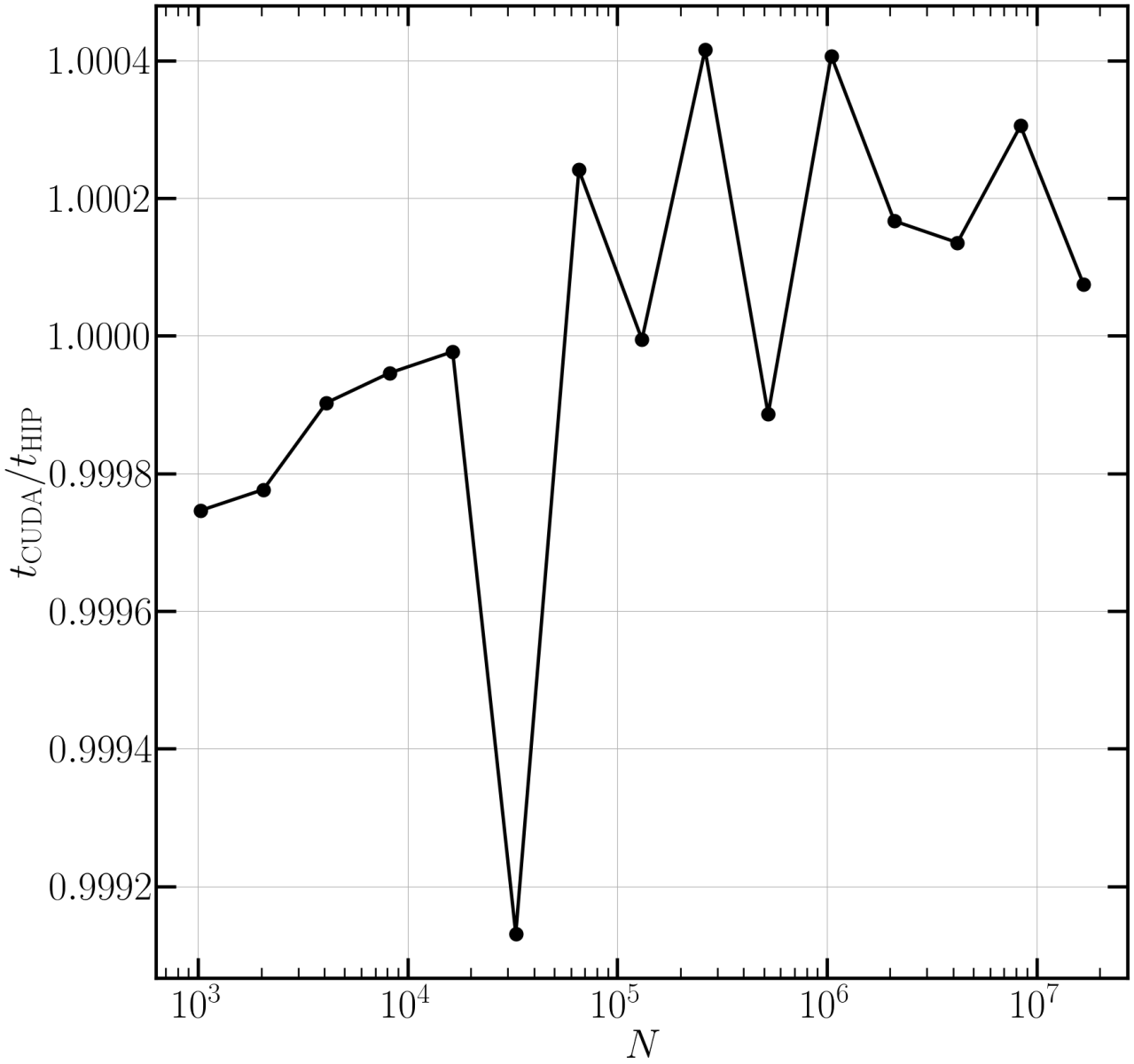
A100 vs MI100

- Direct N-bodyの性能比較
- MI100とA100で, ほぼ同じ性能が得られた
  - 全て単精度で, ~14.5 TF
  - 相互作用あたり22演算を仮定（38だと理論ピーク越え）
- 粒子数が少ない領域でMI100の方が速いのは, ブロックあたりのスレッド数が少ないため（256 vs 1024）
  - より多くのコアを使いやすい設定になっている
  - 粒子数ごとに最適化すれば, 結果は変わる



HIPとCUDAで速度差はあるか？

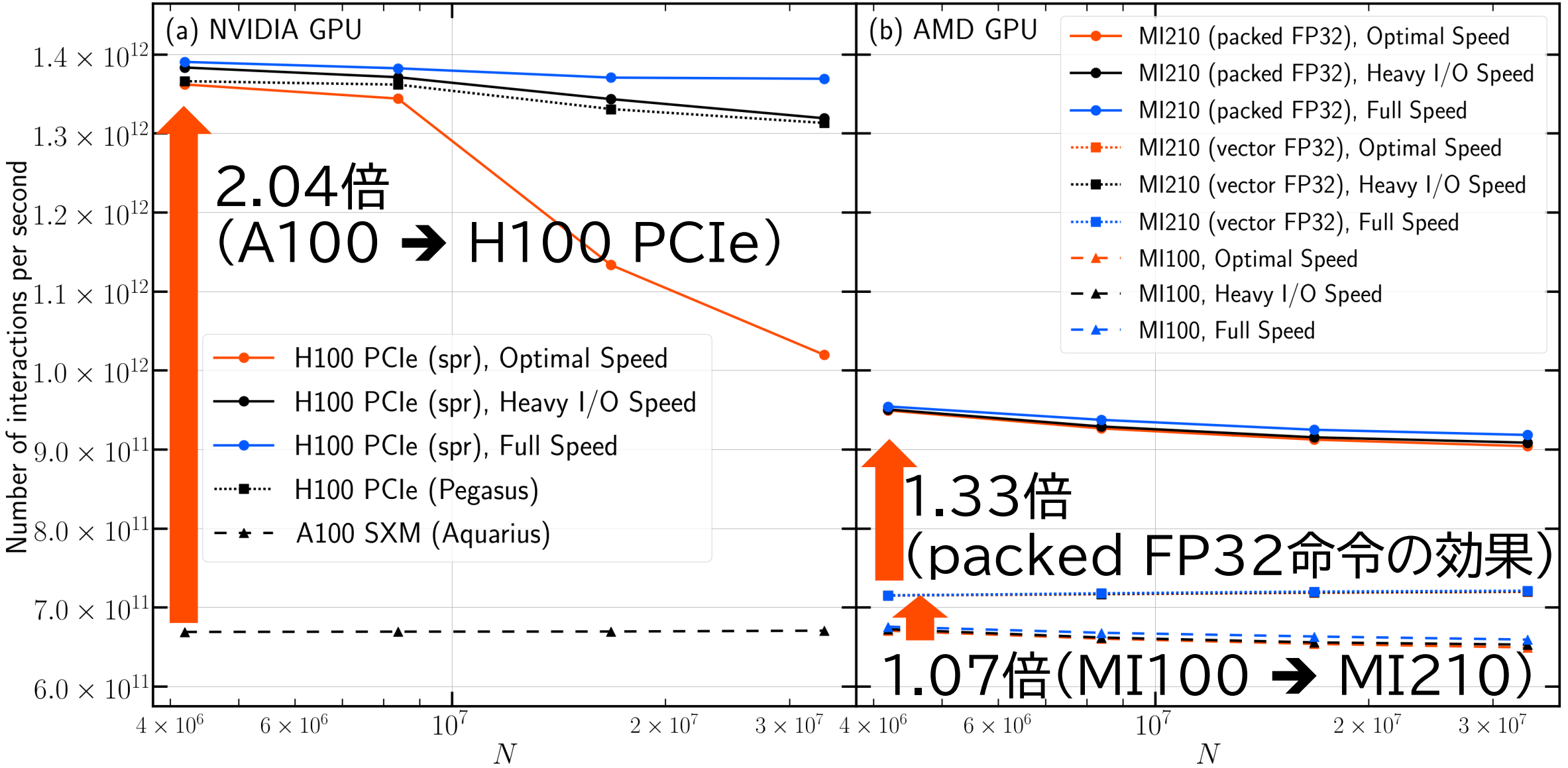
- A100上での測定結果を比較
  - 100回測定したmedianの比
  - 1より大きければ, HIPの方が速い
- 両者の実行時間は非常によく一致
- HIP版にオーバーヘッドは見えない
  - バックエンドではCUDAでコンパイル
  - （保守的なコメントとしては）
    Direct N-body級に重たい処理であれば, オーバーヘッドは見えない
  - そもそもオーバーヘッドが存在しない, ということもありえる
    （GPUの立ち上げの時はあるかも）



最新GPU上でのdirect N-body

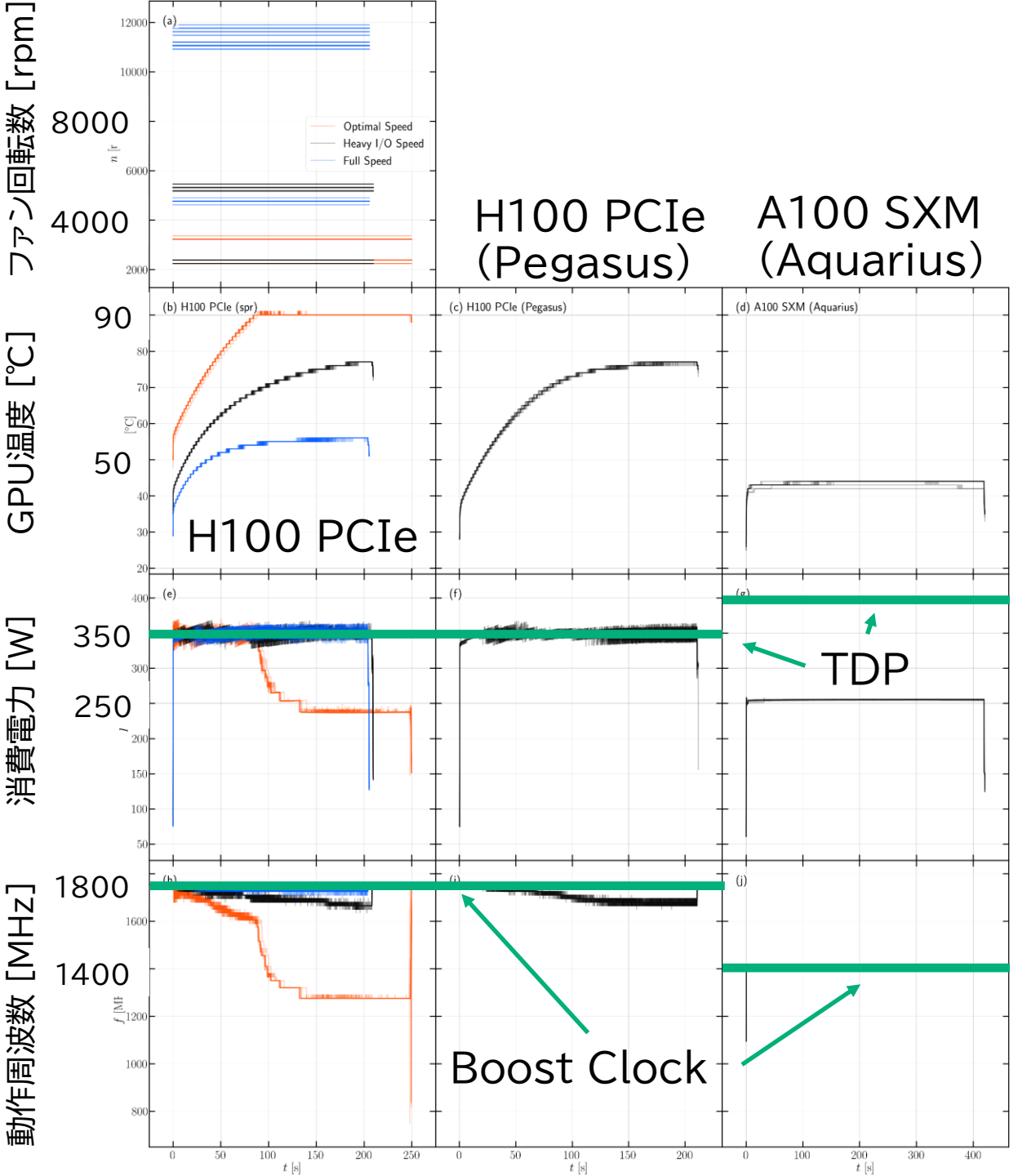
- 詳細は三木&塙(2023, SWoPP)を参照してください
- 最大性能に注目したので, N=4M, 8M, 16M, 32M
  - 普通direct N-bodyでここまで粒子数を増やすことはありませんが..
- GPUの温度, 動作周波数, 消費電力についても監視
  - 前回実行時の余熱を冷却するために, 毎回10分間のcooling time
  - IPMIが触れる環境においては, ファン回転数についても変化させた
- NVIDIA H100 PCIe
  - SMあたりのFPコア数が2倍になった効果が最大
  - SMあたりのINTコア, SFU数は据え置き
- AMD MI210
  - Vector FP32性能については, MI100の方が高い
  - Packed FP32命令(加算, 乗算, 積和算)が導入された

Direct N-bodyの性能測定結果



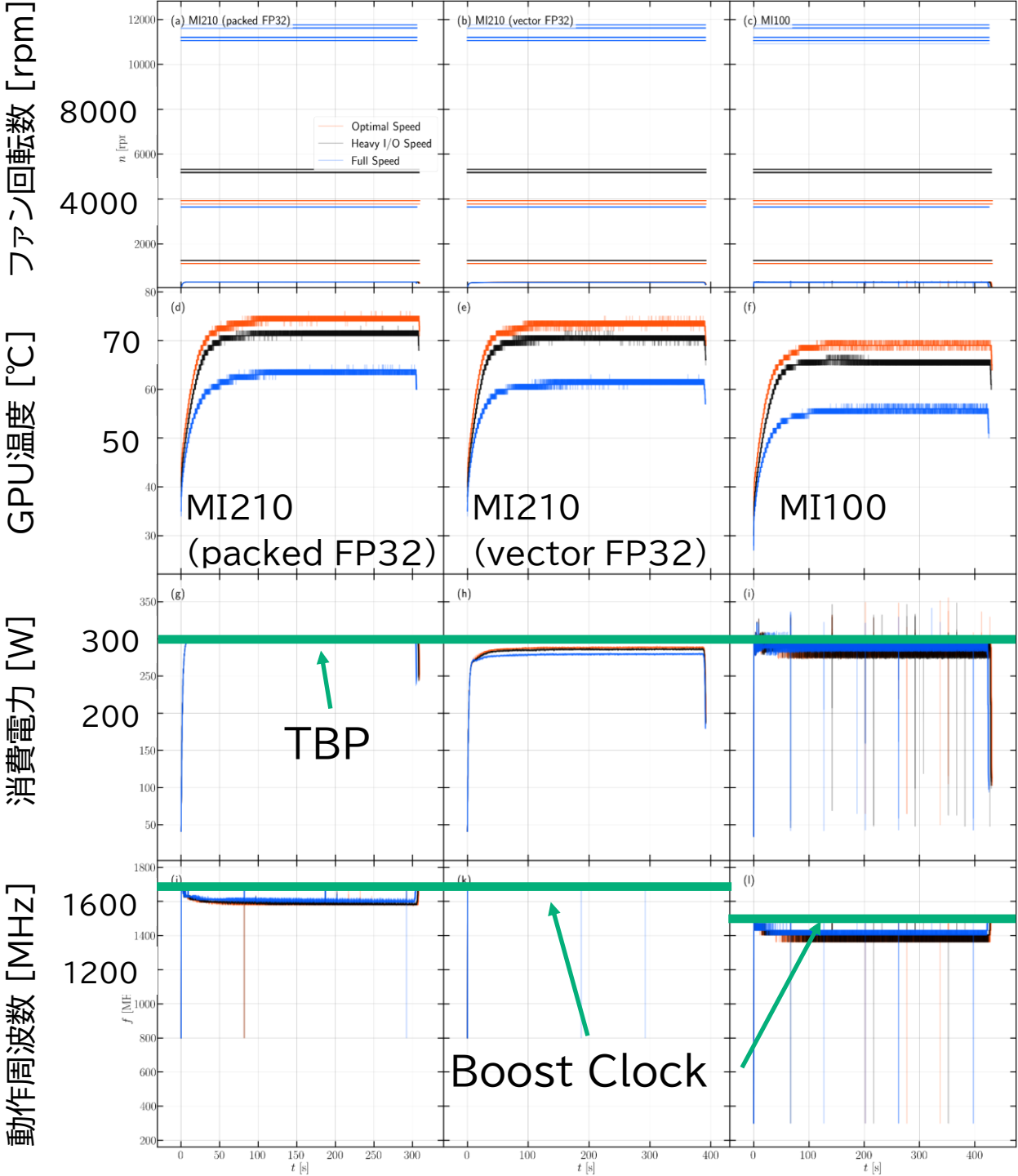
GPU状態の時間変化

- N=16Mの10回分の測定結果
- A100 SXMは水冷
  - 常時Boost Clockで動作
- H100 PCIeは空冷
  - 温度上昇に伴い動作周波数が低下
- H100搭載サーバのファン回転数を最低にした時
  - GPU温度が90度まで上昇
  - 動作周波数も1.3 GHz以下に低下
    - 実行時間増大 = 性能低下の主要因
  - 消費電力は240 W程度に低下

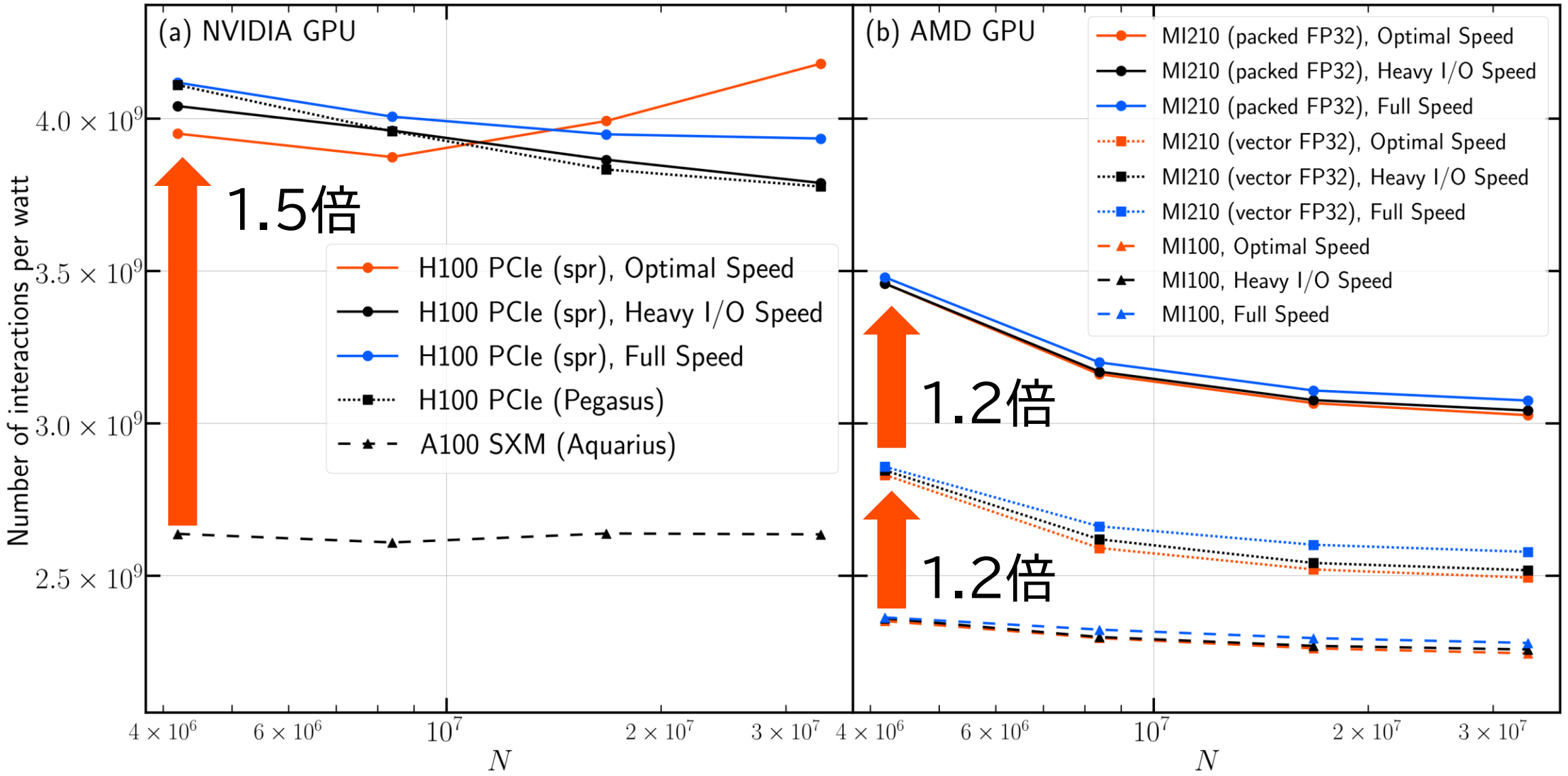


GPU状態の時間変化

- N=16Mの10回分の測定結果
- MI210（packed FP32）とMI100の結果は類似
  - 消費電力がTBPに達するのとほぼ同タイミングで動作周波数が低下
  - 温度上昇とは同期していない
  - 実行時間が伸びるほど動作周波数低下の影響をより長時間受ける
- MI210（vector FP32）は常時Boost Clockで動作
  - Nを増やすと性能向上という，通常見られる振る舞い
  - 消費電力はGPU温度にも依存



電力性能の比較


(a) NVIDIA GPU
(b) AMD GPU
Number of interactions per watt
4.0×10^9
3.5×10^9
3.0×10^9
2.5×10^9
1.5倍
1.2倍
1.2倍
H100 PCIe (spr), Optimal Speed
H100 PCIe (spr), Heavy I/O Speed
H100 PCIe (spr), Full Speed
H100 PCIe (Pegasus)
A100 SXM (Aquarius)
MI210 (packed FP32), Optimal Speed
MI210 (packed FP32), Heavy I/O Speed
MI210 (packed FP32), Full Speed
MI210 (vector FP32), Optimal Speed
MI210 (vector FP32), Heavy I/O Speed
MI210 (vector FP32), Full Speed
MI100, Optimal Speed
MI100, Heavy I/O Speed
MI100, Full Speed
4×10^6
6×10^6
10^7
2×10^7
3×10^7
N

最上位モデルでの性能を予測

- NVIDIA H100 SXMの理論ピーク性能は, PCIe版の1.3倍
  - SM数, 動作周波数が増加
  - TDPは2倍の700 Wになるため, 供給電力不足の心配もない
  - $1.81 \times 10^{12}\ s^{-1}$（Pegasus上での最高性能値からの推定）
    - 理論ピーク性能比, H100 PCIe上で低下していた動作周波数の補正を考慮
- AMD MI250Xの理論ピーク性能は, MI210の2.1倍
  - MI250は, MI210を2つ積んだようなGPU（CU数: 104 ➔ 208）
  - MI250XはさらにCU数を増やしたモデル（CU数: 220）
  - TBPは1.9倍の560 Wなので, 供給電力不足は深刻化？
  - 推定値は$2.01 \times 10^{12}\ s^{-1}$
    - 理論ピーク性能比分の性能向上だけを考慮し, 動作周波数補正は含めていない
    - 実際には供給電力不足による動作周波数低下が起こるはず

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
- 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
- 使う指示文についても選択する必要がある

指示文統合マクロを開発中

- 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
- 同じことを何度も書きたくないので, マクロ化してみた(左側)

```
OFFLOAD_LOOP(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
- 興味のある人(人柱)募集中
  - 指示文ベースの人の役には立つはず
  - 自分自身では使うつもりがないです

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

まとめ

- NVIDIA GPU向けのツリーコードとしてGOTHICを実装した
  - GPUの特性から逆算してアルゴリズムを設計し, CUDAで実装
  - 整数演算の比率が増えたため
    - P100➔V100では大幅に性能向上
    - A100➔H100ではあまり性能上がらず
- HIP C++を用い, NVIDIA・AMD両対応GPUコードを実装中
  - AMD製GPU上でも, (正しく最適化すれば)NVIDIA製GPUと同等の性能が得られる(direct N-bodyの場合)
  - ベストな逆数平方根命令, シェアードメモリの使い方など注意点は多い
  - CDNA 2世代では, packed FP32命令の使用は必須
  - 最上位GPUどうしであればNVIDIA・AMDでほぼ同性能
- OpenACC/OpenMP統合マクロを開発中
  - 興味ある方, ご連絡ください