

Volta世代のGPUにおける 重力ツリーコードの 性能評価

三木 洋平（東京大学・情報基盤センター）

天体形成研究会@筑波大

Contents

- Volta世代のGPU
- Volta世代のGPUへのコード移植
- 重力ツリーコードGOTHICの概要
- GOTHICのTesla V100での性能評価
 - P100からの理論ピーク比以上の速度向上率の起源
 - V100使用時の動作モードに起因する速度向上率
- まとめ

GPGPU

- General Purpose computing on Graphics Processing Unit
 - 画像処理を行うために特化した専用プロセッサを汎用計算にも使えるように拡張
 - 高い演算性能，太いバンド幅，安価，高い電力効率
 - 多数のコアを搭載した並列計算機
 - NVIDIA Tesla P100 では 3584 個のCUDAコア
 - NVIDIA Tesla V100 では 5120 個のCUDAコア
- HPC向けの最新世代はVolta世代
 - 今年8月に発表されたTuring世代 (CC 7.5) は主にグラフィックス向けのGPU (Voltaとの違いは INT4 & INT8 on Tensor coreとRT coreの追加)

Volta世代のGPU

- 最近稼働を開始した大規模システムに搭載
→これから多くのユーザが使うGPU
 - Cygnus @ 筑波大CCSに搭載
 - Summit @ 米国オークリッジ国立研究所



<https://www.nvidia.com/ja-jp/data-center/tesla-v100/>

V100の構成

- <https://devblogs.nvidia.com/parallelforall/inside-volta/>
- INT unit の追加で整数演算が強化
 - 別ユニットなので、浮動小数点演算と同時実行が可能
 - ツリー法にも恩恵?
- Warp Scheduler あたり1つの Dispatch Unit



Contents

- Volta世代のGPU
- Volta世代のGPUへのコード移植
- 重力ツリーコードGOTHICの概要
- GOTHICのTesla V100での性能評価
- 議論
 - V100使用時の動作モードに起因する速度向上率
 - P100からの理論ピーク比以上の速度向上率の起源
- まとめ

ワープの動作パターンの変更

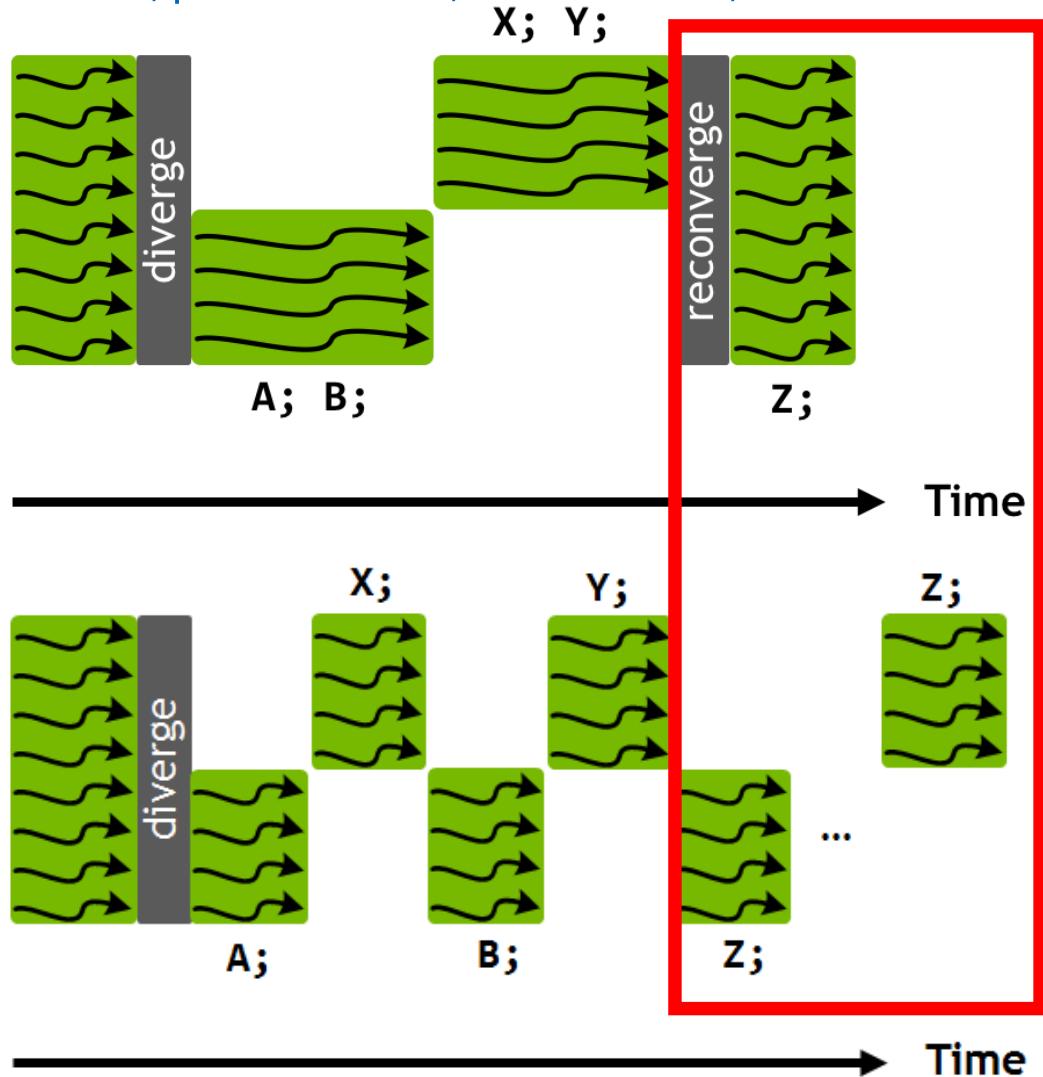
<https://devblogs.nvidia.com/parallelforall/inside-volta/> より

```
if (threadIdx.x < 4) {  
    A;  
    B;  
} else {  
    X;  
    Y;  
}  
Z;
```

Pascal 以前

```
if (threadIdx.x < 4) {  
    A;  
    B;  
} else {  
    X;  
    Y;  
}  
Z;
```

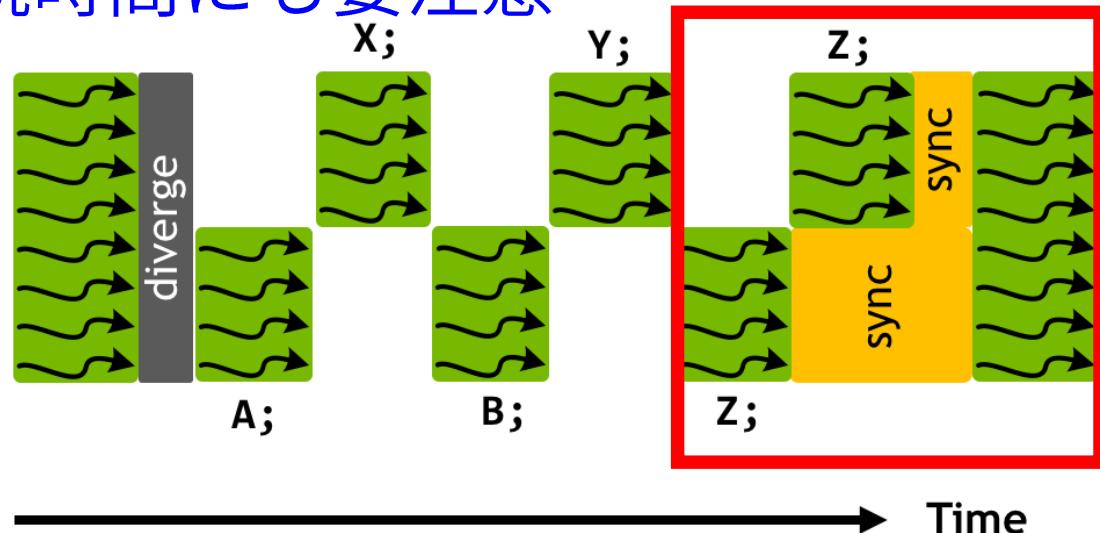
Volta 世代



ワープ内の暗黙の同期は??

- CPUコードからの移植性を上げるためにこと
- 暗黙の同期を仮定したコードは書き換えが必要
(`__syncwarp` などによる明示的な同期)
 - Reduction, scan などなど
- ワープ分裂の持続時間にも要注意

```
if (threadIdx.x < 4) {  
    A;  
    B;  
} else {  
    X;  
    Y;  
}  
Z;  
__syncwarp()
```



ワープシャッフル命令の更新

- CUDA 9 からは, `__shfl*_sync()` が推奨
 - 今まで使っていた`__shfl()`などは非推奨となった
- 追加の引数として mask が導入された
 - ワープシャッフルに関与させる lane が1, 残りを 0 とする (32 threads = 1 warp なので, uint で指定)
 - <https://devblogs.nvidia.com/using-cuda-warp-level-primitives/> を参照
- 16 threads * 1 group : mask = 0xffff
16 threads * 2 groups : mask = 0xffffffff
- データ依存性がある場合には`__activemask()` 命令を使い, 同時に入ってきた lane の情報を取得して対処
- マスクの値が事前に分かっていれば, べた書きを推奨

Volta世代向けのコード改修

- CUDA C Programming guide 9.0 からの抜粋
- Volta世代以降では、コードの書き換えが必要
 1. ワープシャッフル命令などに __sync を追加
 2. 暗黙の同期点に __syncwarp() を追加
 3. __syncthreads() の挿入位置を再検討
- 書き換えが大変な人
 - Pascal世代以前のモードで動かす方法がある
 - コンパイル時: -arch=compute_60 -code=sm_70 を指定
 - 以降では、「Pascalモード」と呼ぶこととする
- 今後はワープ内の同期にも注意を払う必要あり
 - 暗黙の同期を使っていなくても、if() 文の直後に __syncwarp()を入れないと性能面のペナルティが発生

TIPS for Tesla V100

- Volta世代以降のGPUをVoltaモードで動作させる場合限定で `__syncwarp()` を発行するコード
 - 32 threads = 1 warp が同時に if() 文に入る場合

```
if( hoge ){
    fuga;
}
else{
    piyo;
}
#    if __CUDA_ARCH__ >= 700
    __syncwarp();
#endif//__CUDA_ARCH__ >= 700
```

32 スレッド未満に対する同期

- Cooperative Groupsを使って同期をとる
 - __syncwarp(), __syncthreads() をはった場合には、計算がハングアップする

```
#include <cooperative_groups.h>
using namespace cooperative_groups;

__global__ void func(){
    #if __CUDA_ARCH__ >= 700
        thread_block_tile<16> tile = tiled_partition<16>(this_thread_block());
    #endif

    #if __CUDA_ARCH__ >= 700
        tile.sync();
    #endif
}
```

シェアードメモリの容量指定

- 今まで
 - (SM, L1) = (48KB, 16KB), (16KB, 48KB) を切り替え
 - 64 KB on P100
- Volta 世代
 - 128 KBをSMとL1に割り当てる
 - 0, 8, 16, 32, 64, 96 KB のうちどれかをSMに割り当てる
 - 使いたい容量が“最大値の何%か”を指定する
 - 最終的にはCUDAが良きに計らう
 - 64 KB にしたければ66 (= 66% of 96 KB)を指定する

```
cudaFuncSetAttribute(f0, cudaFuncAttributePreferredSharedMemoryCarveout, 100);  
cudaFuncSetAttribute(f1, cudaFuncAttributePreferredSharedMemoryCarveout, 0);
```

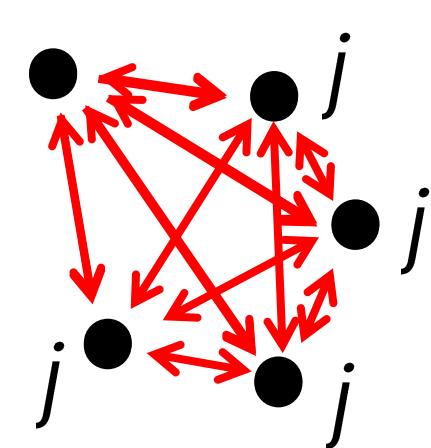
Contents

- Volta世代のGPU
- Volta世代のGPUへのコード移植
- 重力ツリーコードGOTHICの概要
- GOTHICのTesla V100での性能評価
- 議論
 - P100からの理論ピーク比以上の速度向上率の起源
 - V100使用時の動作モードに起因する速度向上率
- まとめ

N体計算（重力多体計算）

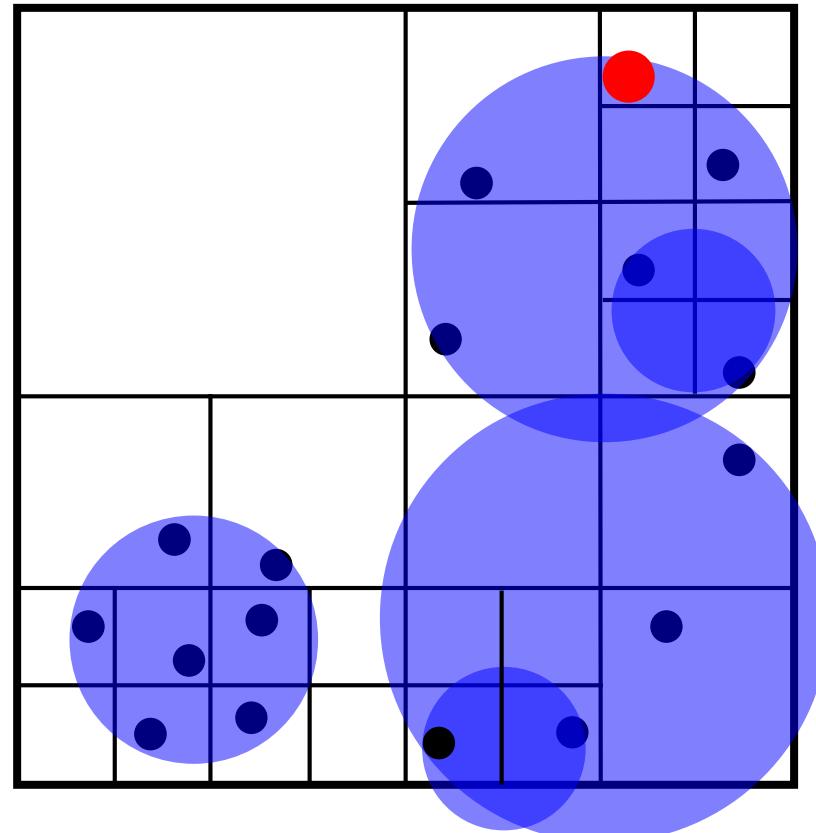
- 粒子どうしに働く自己重力による系の時間進化を，運動方程式に基づいて計算
 - データ量: $\mathcal{O}(N)$
 - 重力計算: $\mathcal{O}(N^2)$
 - 時間積分: $\mathcal{O}(N)$
- 以降で使う用語の整理
 - i-粒子: 重力を感じる粒子
 - j-粒子: 重力を及ぼす粒子

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



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
 - $\mathcal{O}(N_i N_j) \rightarrow \mathcal{O}(N_i \log N_j)$
 - GADGET criterionを採用
(Springel 2005)
$$\frac{Gm_J}{d_{ij}^2} \left(\frac{b_J}{d_{ij}} \right)^2 \leq \Delta_{\text{acc}} |a_i^{\text{old}}|$$
 - GPU上での高速化も可能
Nakasato12, Ogiya+13,
Bedorf+12, 14 etc.



自動最適化: 木構造の更新頻度

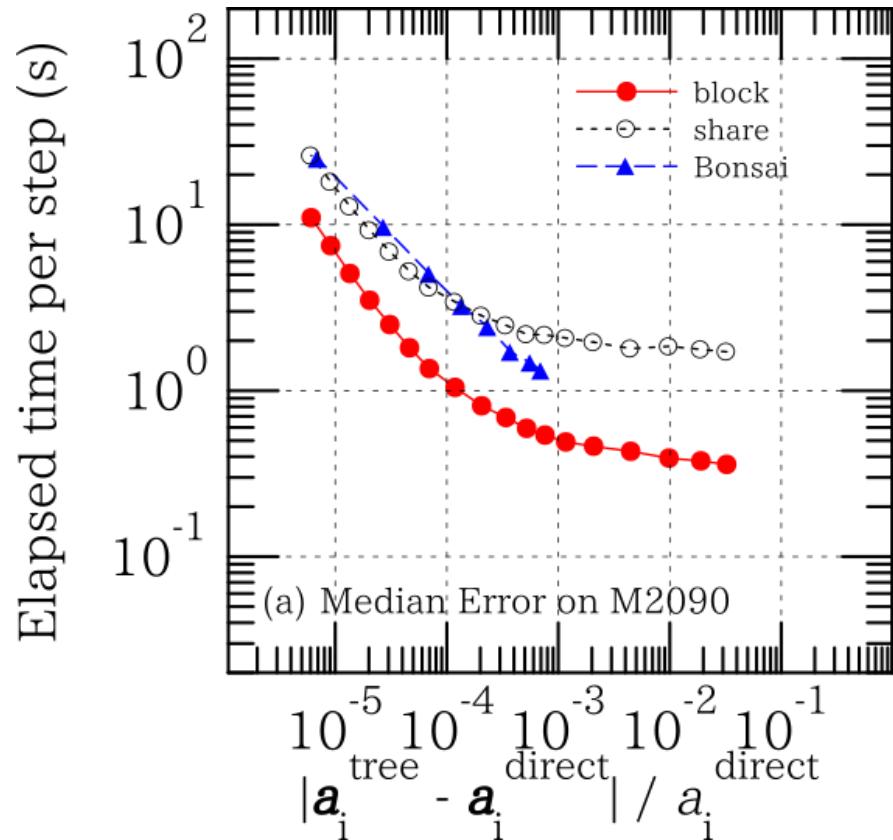
- 木構造を毎ステップ作り直すのは無駄
 - 注: ツリーノードの重心座標などは毎ステップ更新
- 古い木構造を用いて重力計算をすると…
 - 木構造の再構築コストを削減
 - 論理構造と実際の粒子分布のズレが大きくなると、重力計算のコストが増大
- 両者の和を最小化する更新頻度を動的に設定
 - 実行時間をモデル化し、最適な更新頻度を予測

$$t_{\text{tot}} = t_{\text{make}} + n(t_1 - \Delta t) + \frac{n(n+1)}{2} \Delta t \quad n^2 = \frac{2}{\Delta t} t_{\text{make}}$$

GOTHIC: Gravitational Oct-Tree code accelerated by Hierarchical time step Controlling

- GPU上で高速に動作する tree codeを開発
 - 幅優先探索を採用
 - Block time stepを採用
 - 動的な最適化を実装
- Block time stepによって 3-5倍程度の高速化
- Bonsai (Bédorf+12, 14) より 高速であることを確認
- プロダクトランにも使用
 - Kirihsara, YM et al. (2017a, b)

Miki & Umemura (2017),
New Astronomy, 52, 65

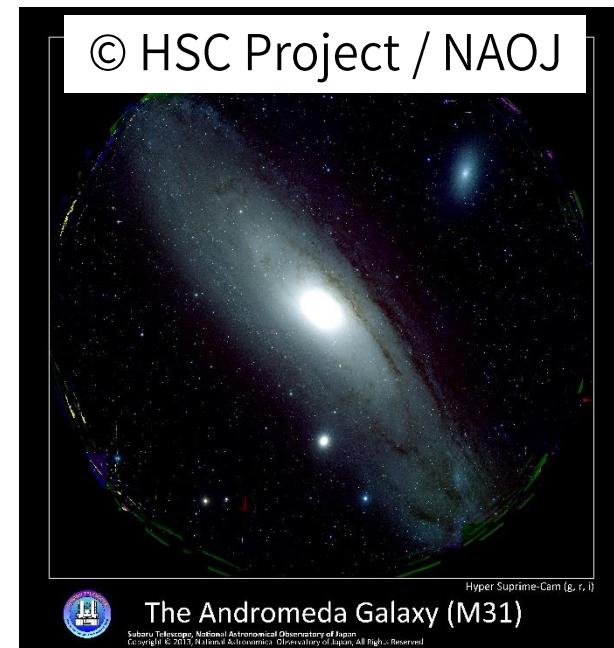


Contents

- Volta世代のGPU
- Volta世代のGPUへのコード移植
- 重力ツリーコードGOTHICの概要
- GOTHICのTesla V100での性能評価
- 議論
 - P100からの理論ピーク比以上の速度向上率の起源
 - V100使用時の動作モードに起因する速度向上率
- まとめ

性能測定

- 4096ステップの計算のwall clock timeを測定
 - Block time step, 自動最適化の効果を反映
 - ファイルの入出力, メモリ確保などの時間も含む
(これらは無視できる)
- 粒子モデル: アンドロメダ銀河
 - Geehan+06, Fardal+07
 - ダークマターハロー + 恒星ハロー + バルジ + 銀河円盤
 - 粒子分布はMAGI (YM & Umemura 2018) を用いて生成
 - 粒子数は $N = 2^{23} = 8,388,608$



評価環境

- Pascal: TSUBAME 3.0 @ 東京工業大学GSIC
- Volta: V100を搭載したPOWER9サーバ @ 東大

CPU	IBM POWER9 (8335-GTG) 16 cores, 2.0–3.1 GHz	Intel Xeon E5-2680 v4 14 cores, 2.4 GHz
GPU	Tesla V100 (SXM2) 5120 cores, 1.530 GHz HBM2 16 GB	Tesla P100 (SXM2) 3584 cores, 1.480 GHz HBM2 16 GB
Compiler	gcc 4.8.5 CUDA 9.2.88	icc 18.0.1.163 CUDA 8.0.61

スペックの比較

Tesla V100 (SXM2)

- 80 SMs (5120 cores)
- 1.530 GHz
- 15.67 TFlop/s (SP)
 - P100 の1.5倍
- 900 GiB/s (理論)
- 750 GiB/s (実測)
 - P100の1.5倍

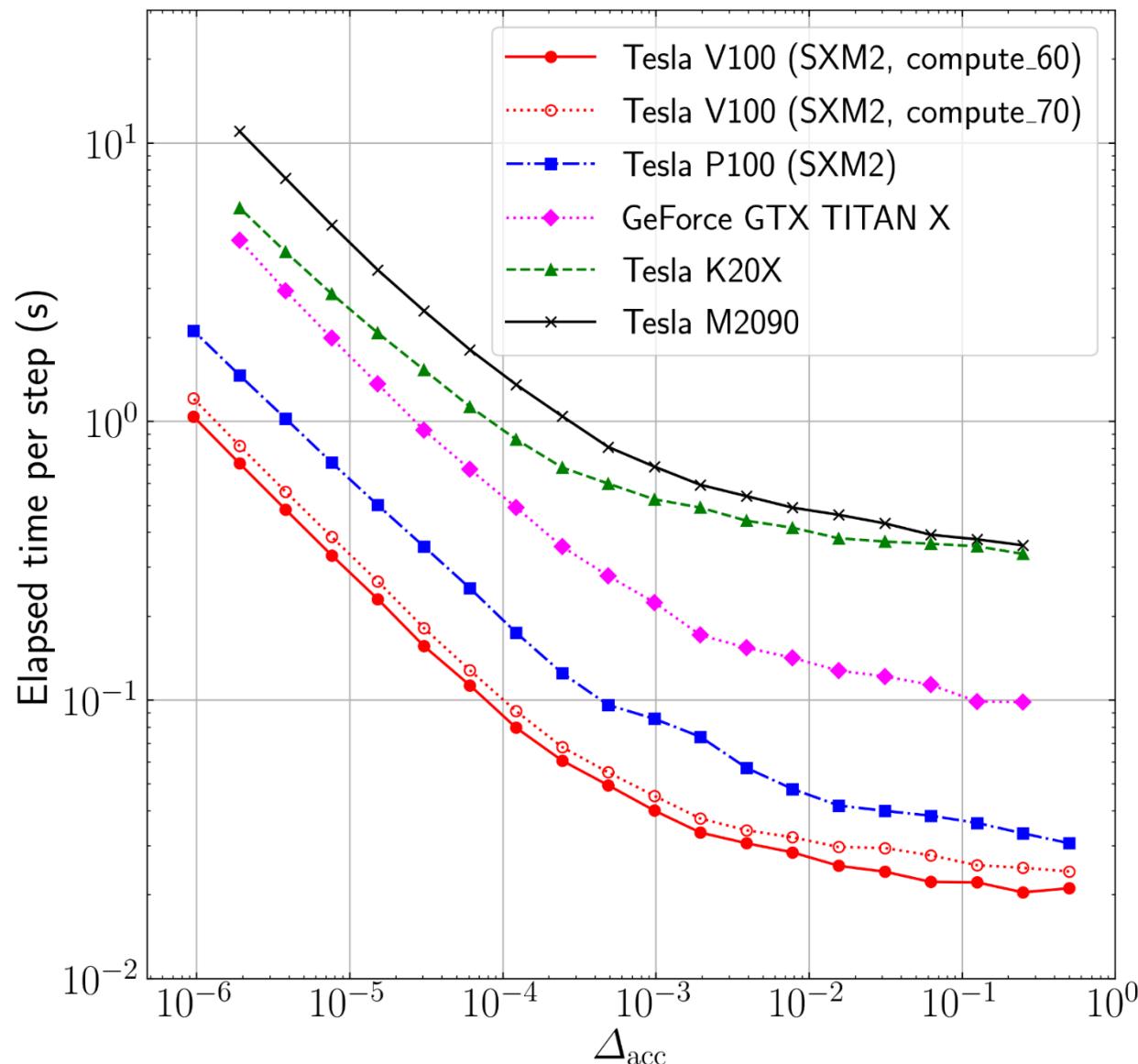
Tesla P100 (SXM2)

- 56 SMs (3584 cores)
- 1.480 GHz
- 10.61 TFlop/s (SP)
- 732 GiB/s (理論)
- 510 GiB/s (実測)

メモリ性能はJia et al. (2018)による測定値 (PCIe版)

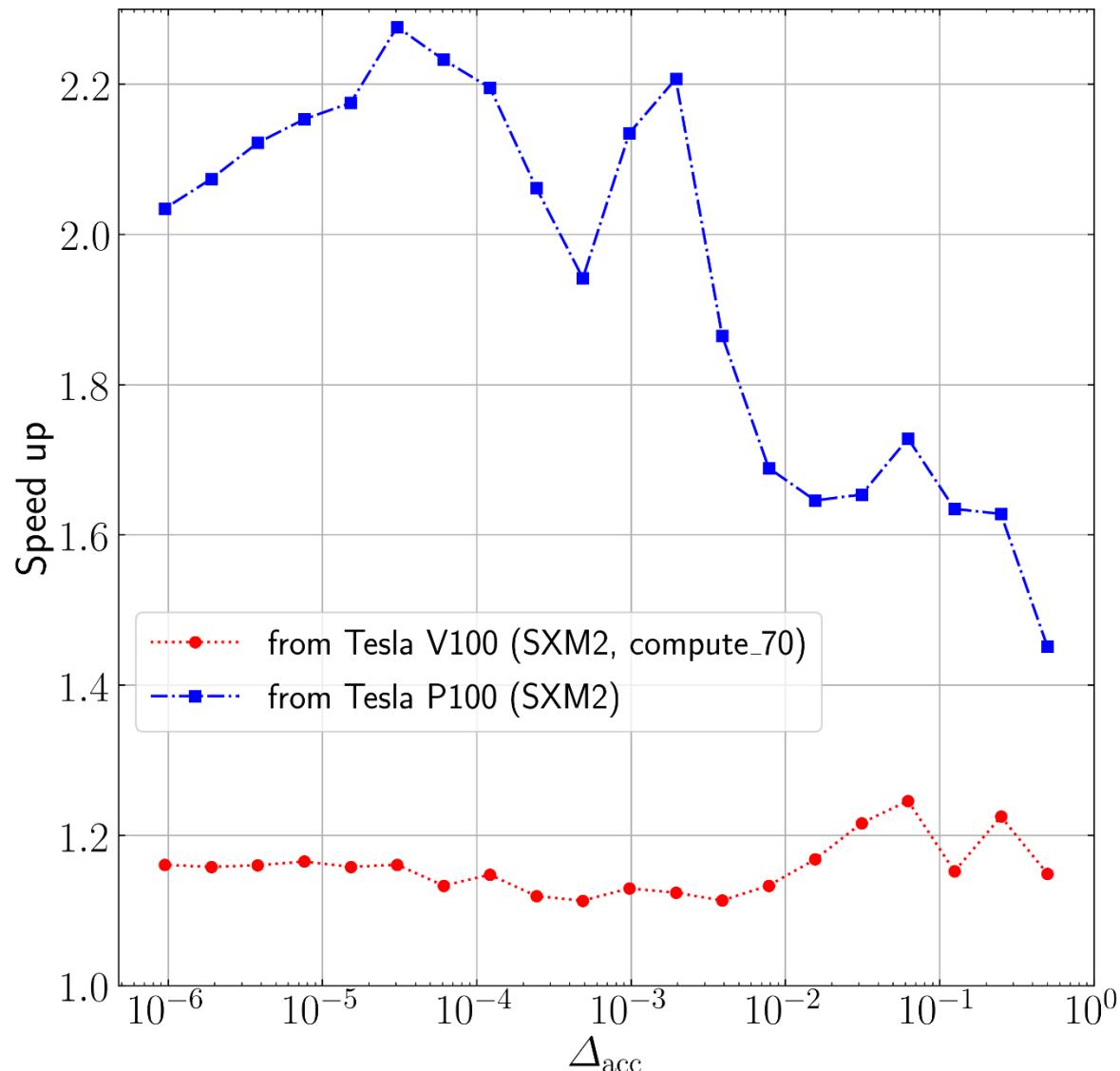
実行時間

- M31 model
 - N=8M
 - 4096 steps
- 一番速いのは
V100を Pascal
モードで動作
させた場合
- Maxwell以前の
データはMiki &
Umemura
(2017) のもの



速度向上率

- P100よりも最大で2.2倍程度高速
 - 理論ピーク演算性能比(1.5倍)よりも高速化
 - $\Delta_{\text{acc}} \lesssim 10^{-3}$ の領域では2倍以上高速
- PascalモードはVoltaモードよりも1.2倍程度高速

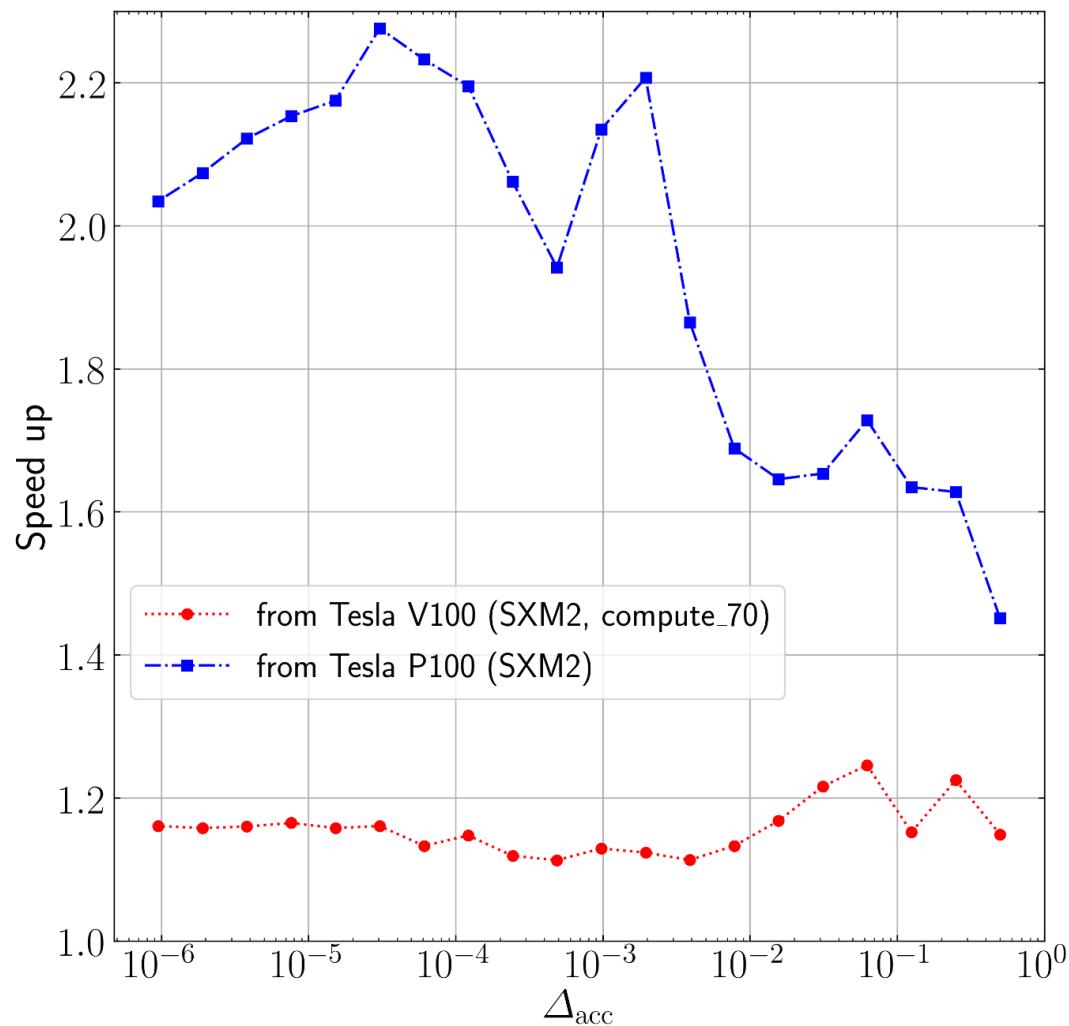


Contents

- Volta世代のGPU
- Volta世代のGPUへのコード移植
- 重力ツリーコードGOTHICの概要
- GOTHICのTesla V100での性能評価
- 議論
 - P100からの理論ピーク比以上の速度向上率の起源
 - V100使用時の動作モードに起因する速度向上率
- まとめ

速度向上率の起源は何か？

- Tesla V100はTesla P100より2.2倍高速
 - 整数演算の実行時間が隠蔽されるため
- PascalモードはVoltaモードより1.2倍高速
 - Voltaモードで必要な`syncwarp()`命令を発行する必要がない

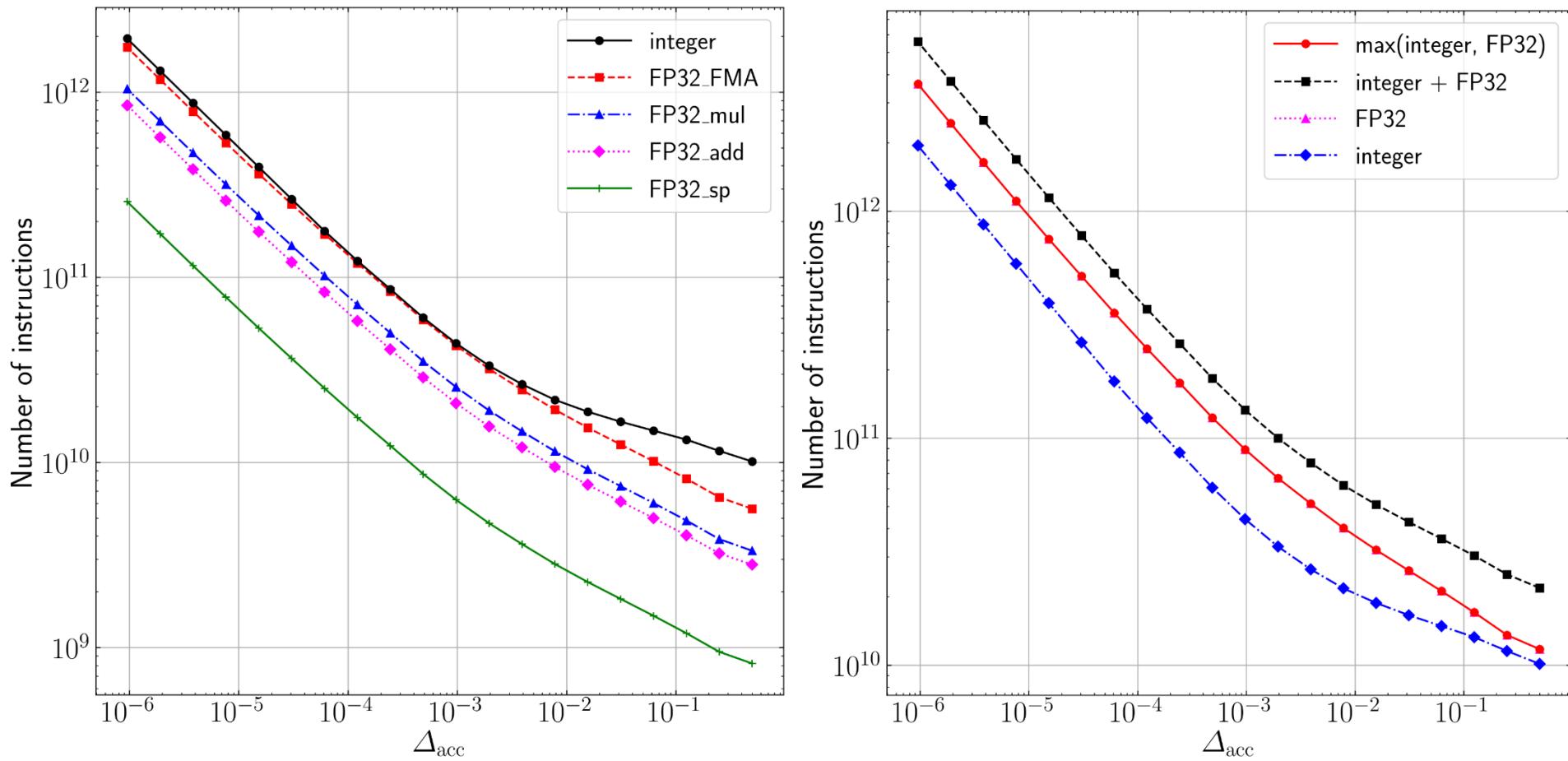


演算数の取得 (nvprof)

- \$ numactl --cpunodebind=0 --localalloc nvprof --kernels “::calcAcc_kernel:” --metrics inst_integer,flop_count_sp_fma,flop_count_sp_add,flop_count_sp_mul,flop_count_sp_special bin/gothic [options]
 - 重力計算カーネルに絞って演算数を取得
 - 倍精度演算は含まれていないので省略
 - つけて損することはないので numactl も追加
 - 計算がシリアル化されて遅くなるため，256ステップ分を測定
 - 自動最適化機能は全て無効化
 - 最適なパラメータは別途測定した値を渡す

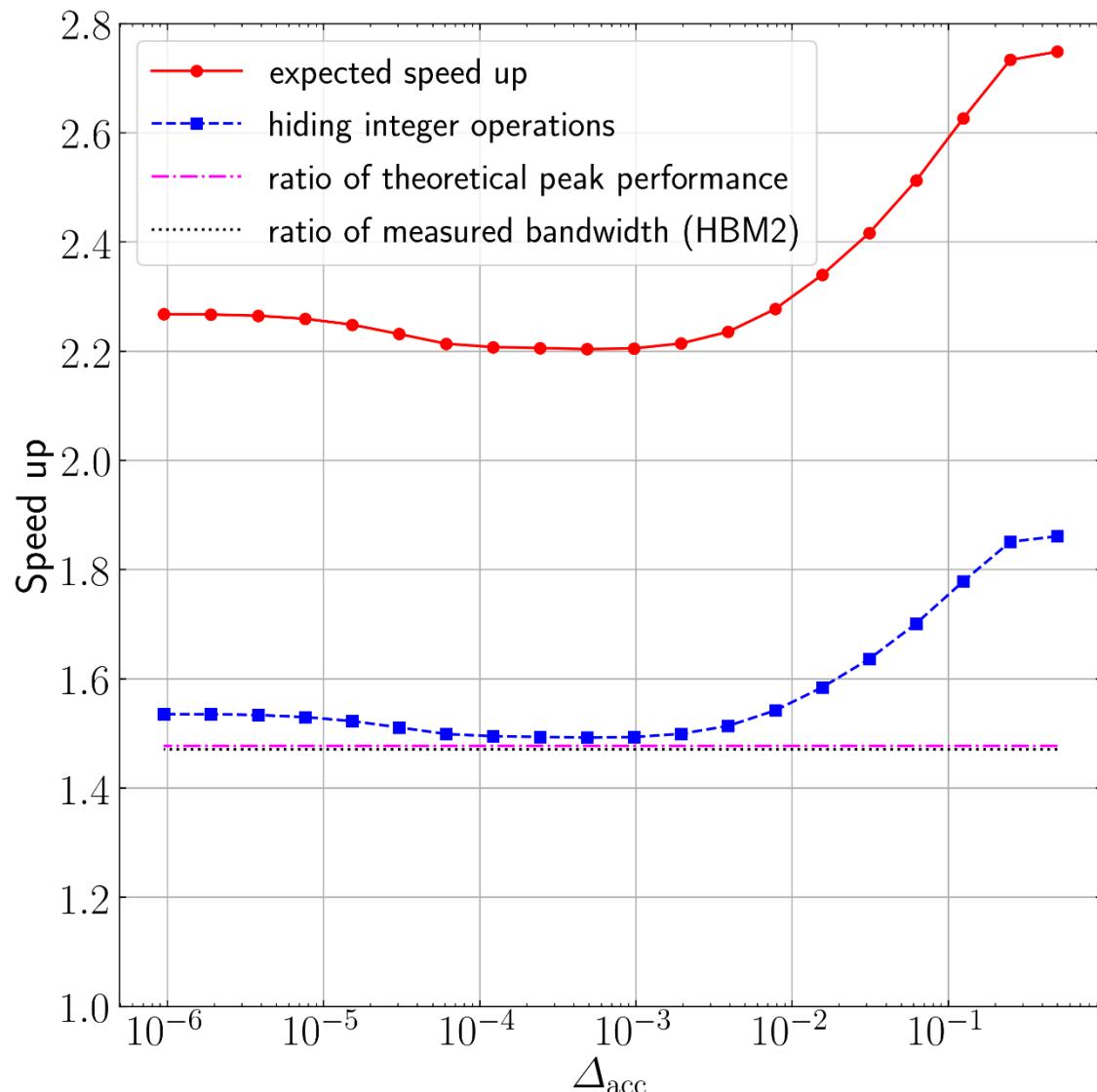
実行された命令数を取得

- 整数演算の実行時間は隠蔽される可能性あり



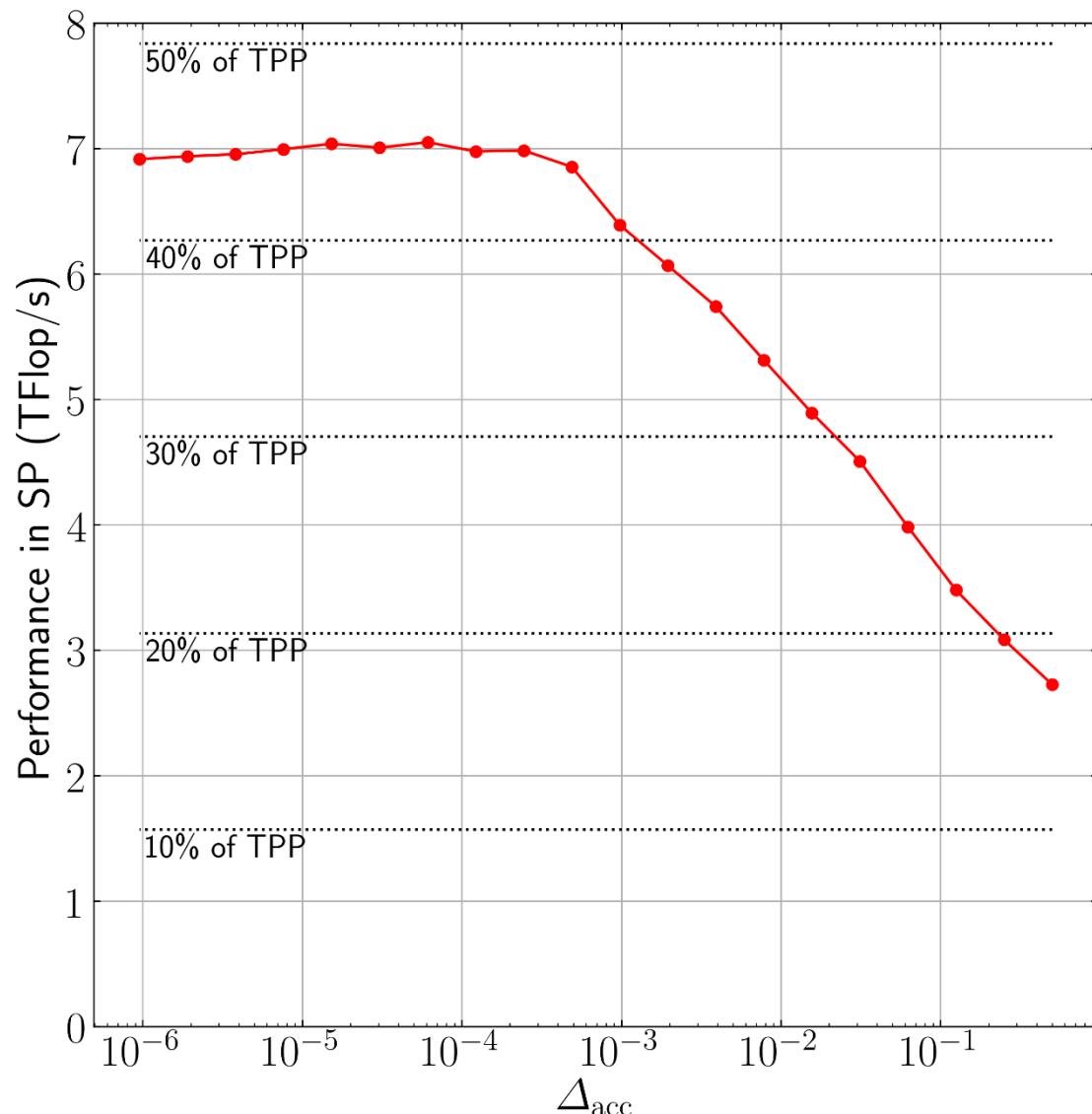
期待される速度向上率

- 理論ピーク性能比、実測メモリバンド幅比はともに約1.5倍
- 整数演算の隠蔽により、1.5倍以上の高速化が可能
- 演算律速であれば、2.2倍以上の高速化が可能
 - P100からの速度向上率を説明できる



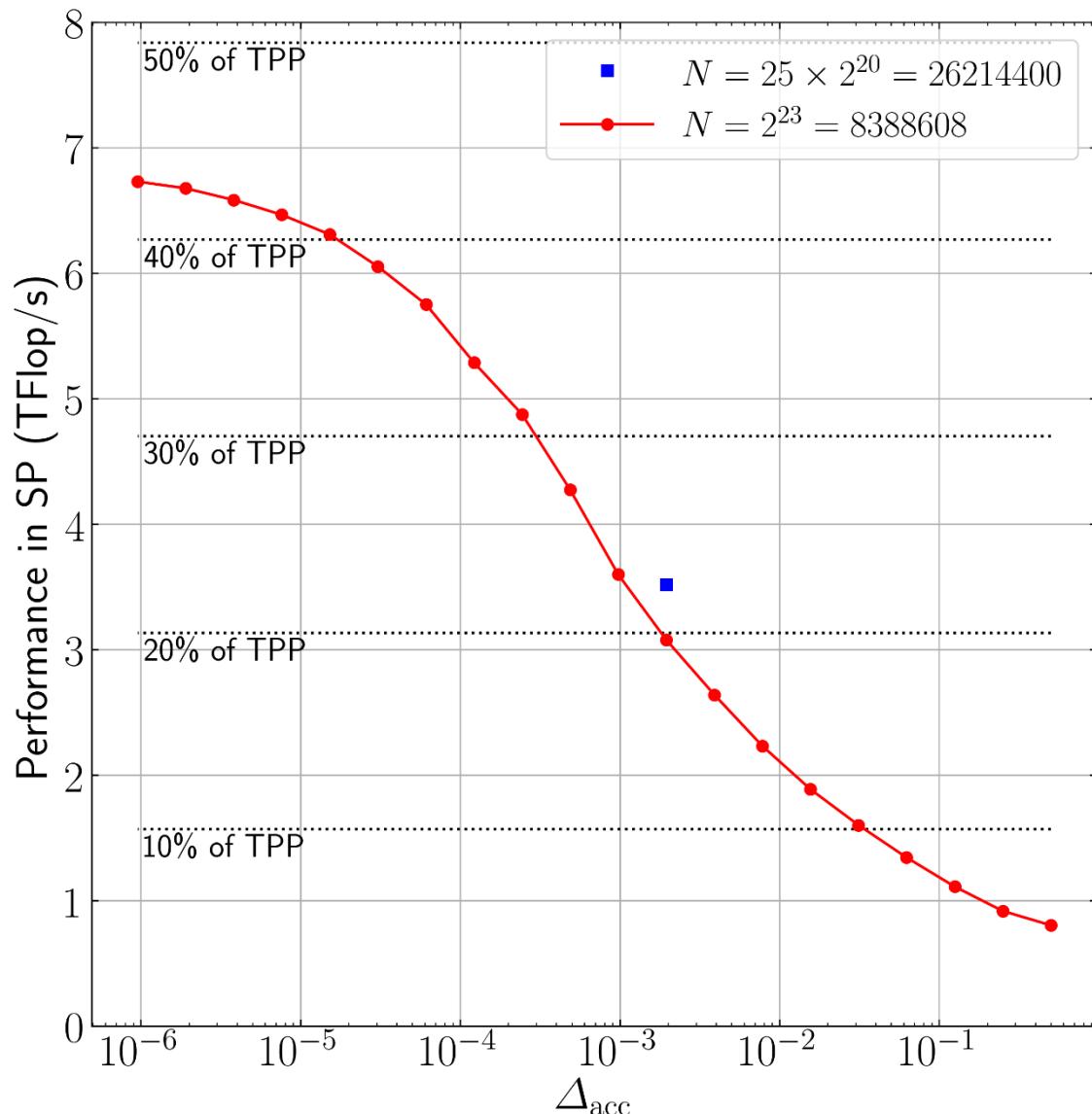
重力計算カーネルの性能

- 重力計算関数の実行時間を独立に測定
- 逆数平方根の浮動小数点演算数を4 Flopsと仮定
 - 加算・乗算とのスループット比に対応
- $\Delta_{\text{acc}} \sim 10^{-3}$ を境に振る舞いが変化
 - 演算量が減り、整数演算の実行時間が隠蔽されづらくなる？
 - P100 からの速度向上率とも符合する



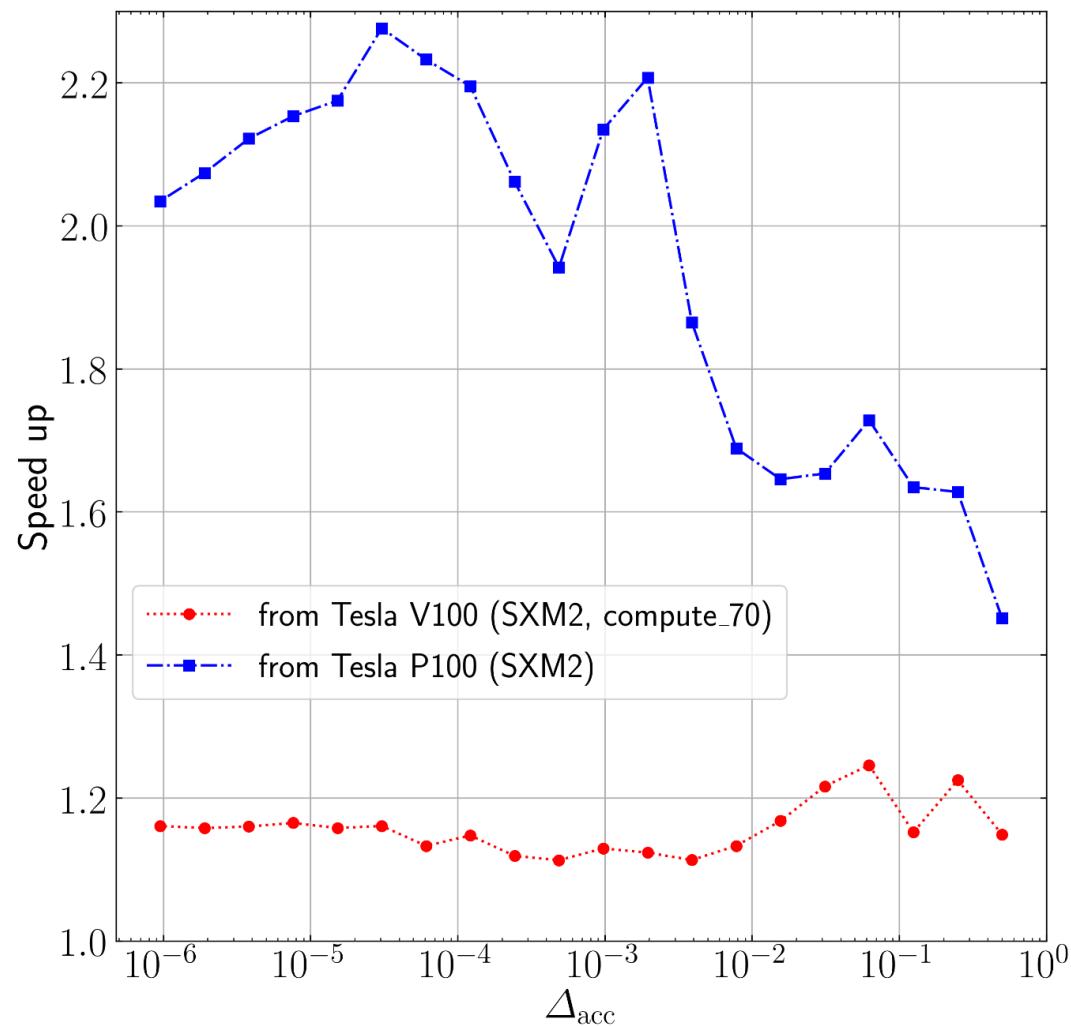
コード全体の演算性能

- `rsqrtf() = 4 Flops` を仮定
- 重力計算の精度を上げると演算性能も単調増加
- 典型的な演算性能では3 TFlop/s 程度
 - 単精度理論ピーク演算性能の2割程度
 - 3.5 TFlop/s ($N=25M$)



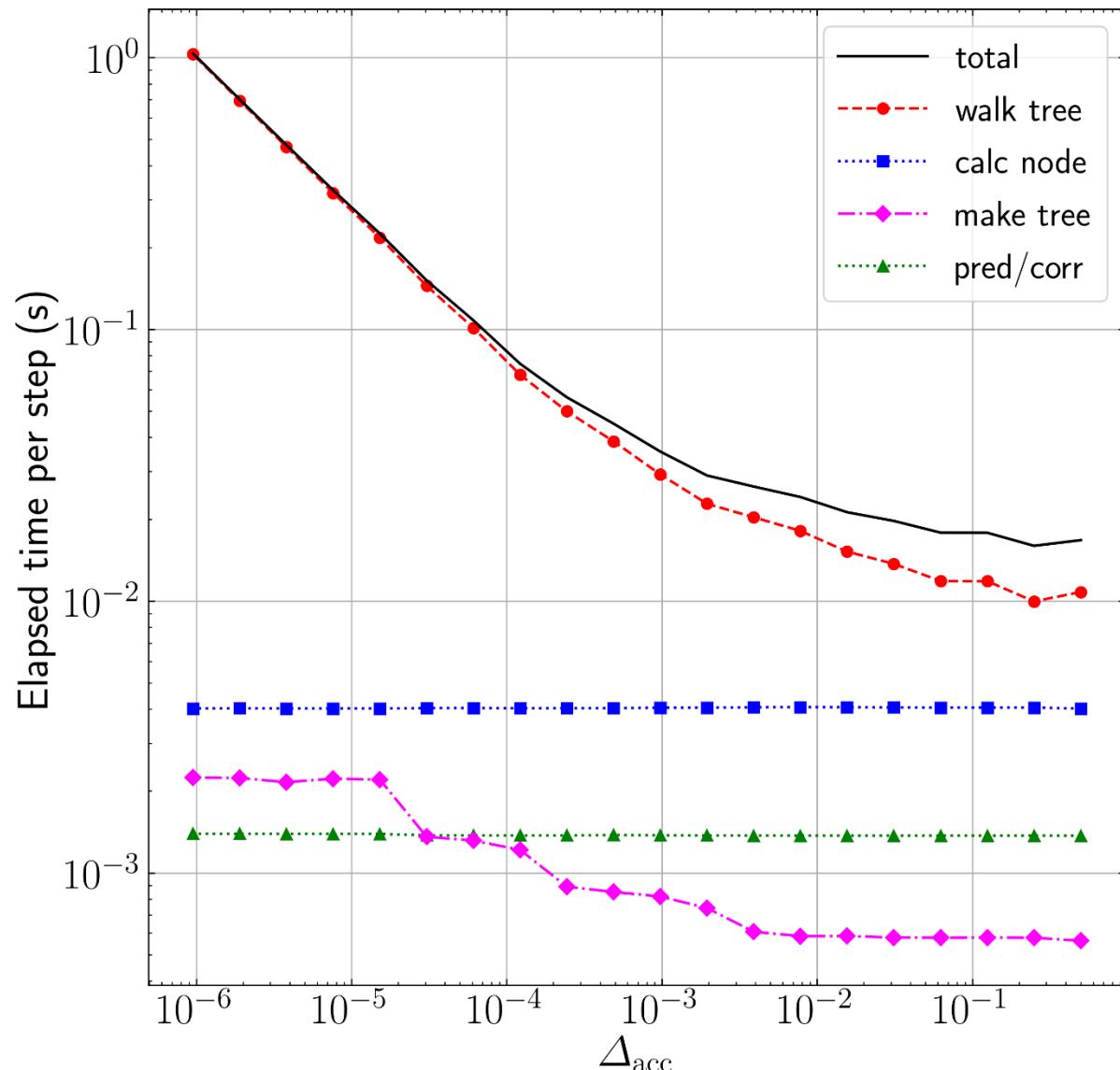
速度向上率の起源は何か？

- Tesla V100はTesla P100より2.2倍高速
 - 整数演算の実行時間が隠蔽されるため
- PascalモードはVoltaモードより1.2倍高速
 - Voltaモードで必要な`syncwarp()`命令を発行する必要がない



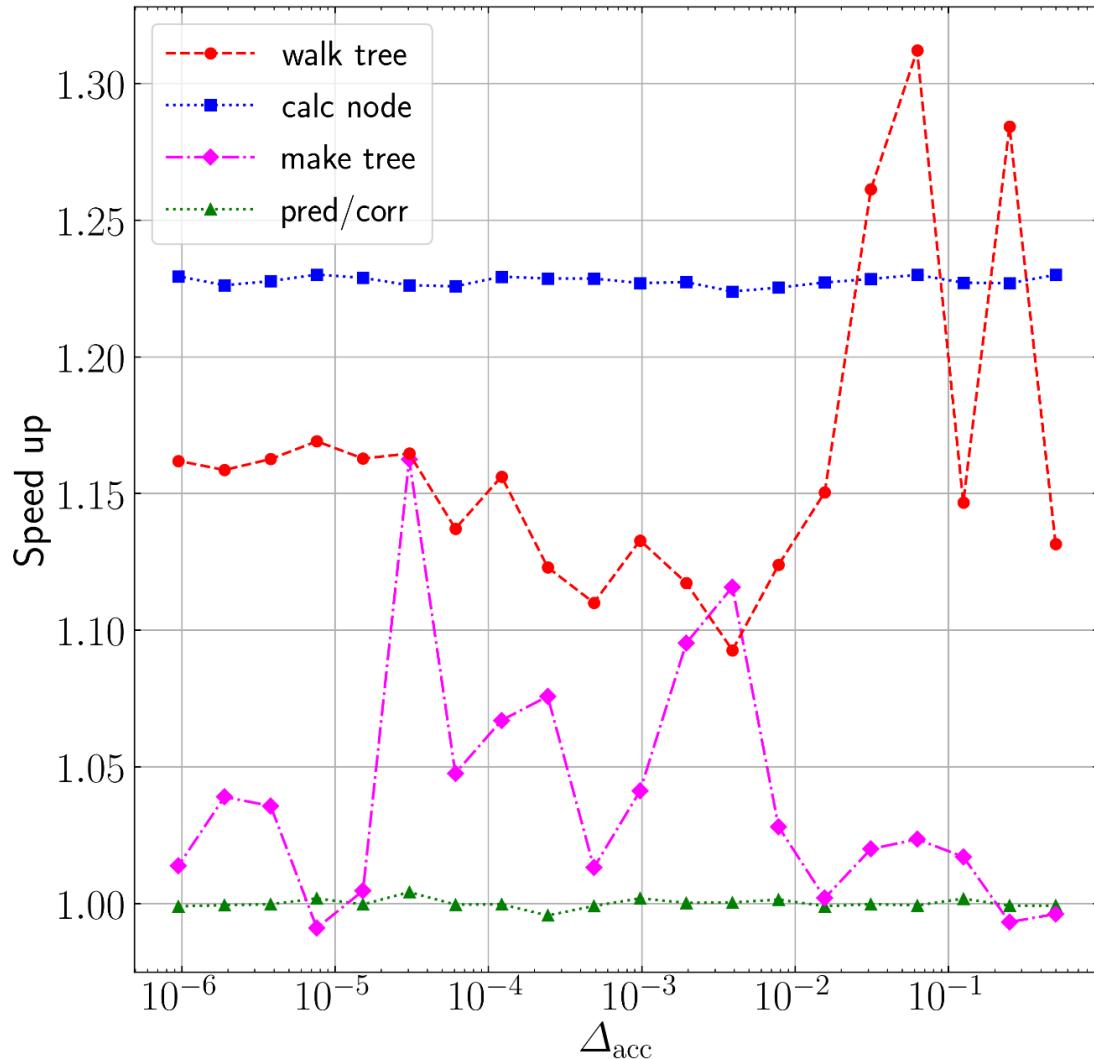
関数ごとの計算時間

- 精度に依らず重力計算が支配的
- 低精度側では多重極の計算の寄与もある
- 木構造の構築間隔は、重力計算コストとの和を最小化するよう自動調整



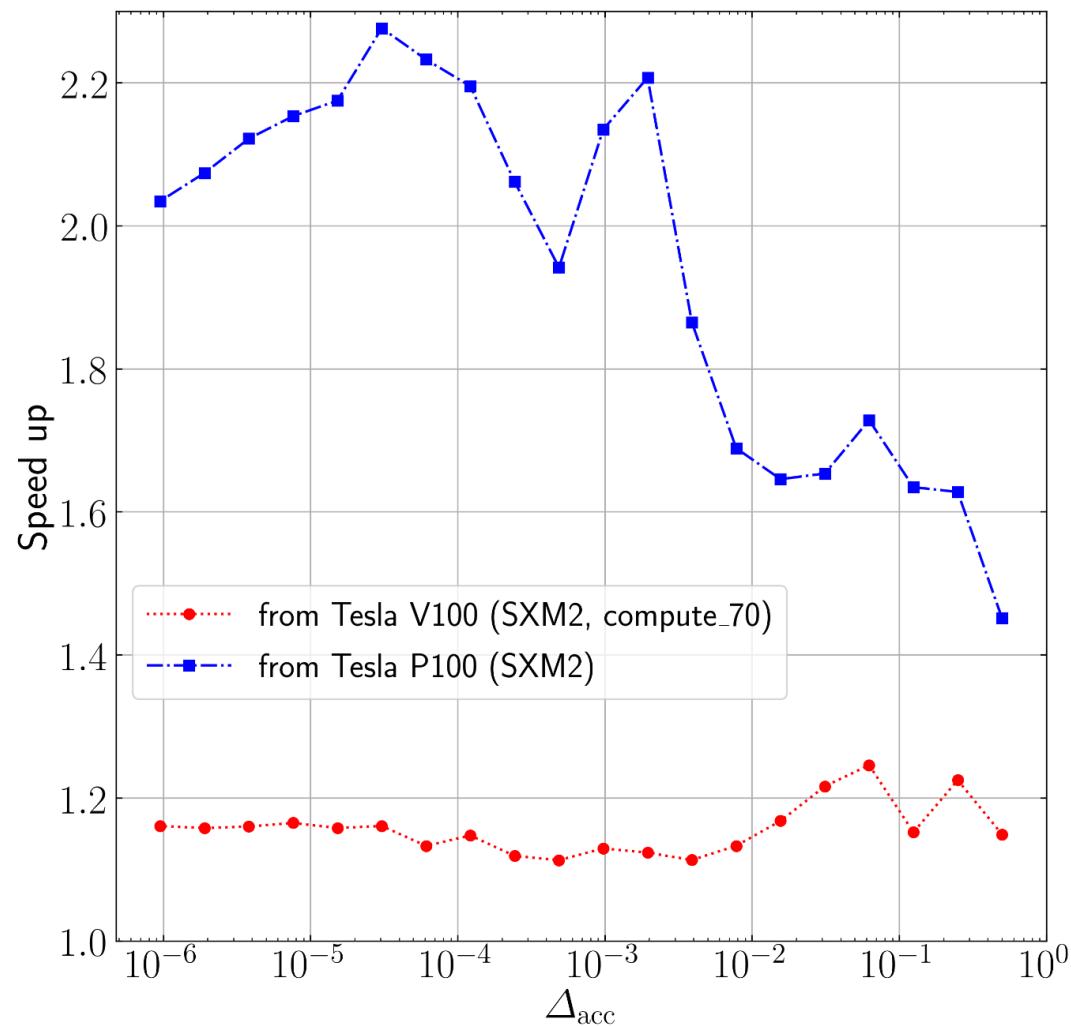
関数ごとのPascalモードとVoltaモードの比較

- Voltaモードの方が速い関数は存在しない
- 影響の大きい関数はツリーノードの処理関数など：
`syncwarp()`の割合が多い
- ツリー構築は、全体同期やソート関数の影響もある



速度向上率の起源は何か？

- Tesla V100はTesla P100より2.2倍高速
 - 整数演算の実行時間が隠蔽されるため
- PascalモードはVoltaモードより1.2倍高速
 - Voltaモードで必要な`syncwarp()`命令を発行する必要がない



まとめ

- 重力ツリーコードGOTHICをTesla V100向けに移植し、性能評価を行った
 - 移植はかつてないほどに大変だったが、実入りは良い
 - 3.3×10^{-2} s per step @ N = 8M (3.1 TFlop/s in SP)
 - 2.0×10^{-1} s per step @ N = 25M (3.5 TFlop/s in SP)
- Tesla P100よりも最大で2.2倍高速化
 - 整数演算ユニットで実行される整数演算の実行時間が隠蔽されたことによる高速化
- -gencode arch=compute_60,code=sm_70を指定してコンパイルすることで1.2倍高速化
 - ワープ内の暗黙の同期を有効化することで、`__syncwarp()`関数のコストを削減