Wisteria/BDEC-01 利用事例（2）

NVIDIA A100 と CUDA 11 の新機能

三木 洋平

東京大学情報基盤センター

1. はじめに

東京大学情報基盤センター（以下，本センター）が運用する「計算・データ・学習」融合スーパーコンピュータシステム（Wisteria/BDEC-01）のデータ・学習ノード群（Aquarius または Wisteria-A）には，NVIDIA 社の最新の GPU である NVIDIA A100 Tensor コア GPU（以下，A100）が全 360 基（ノード当たり 8 基，全 45 ノード）搭載されている[1]．A100 は，2021 年 11 月 30 日に運用を終了した Reedbush-H/L システムに搭載されていた NVIDIA Tesla P100（以下，P100）から性能・機能ともに強化された GPU である．「Wisteria/BDEC-01 利用事例」の第 2 回にあたる本稿では， A100 や CUDA 11 で導入された新機能とその使い方について紹介し，Reedbush-H/L から Aquarius への移行を支援する．

2. P100・V100・A100 の性能比較

本節では，過去の GPU と A100 の仕様を比較することで A100 の特徴を概観する．本センターが運用する GPU スパコンには搭載されなかったが，A100 と P100 の間には NVIDIA V100（以下，V100）があるため，この V100 も交えて表 1 に P100，V100，A100 のハードウェア観点での比較を示す[2， 3， 4]．各世代の GPU ともに複数の製品があるが，P100 および A100 については本センターが運用するシステムに搭載されている製品，V100 についてはこれに対応する型番の GPU の仕様を示した．

表 1：各 GPU の比較

	P100 (SXM)	V100 (SXM)	A100 (SXM, HBM2)
CUDA コア数	3584 (=56×64)	5120 (=80×64)	6912 (=108×64)
GPU Boost Clock	1.48 GHz	1.53 GHz	1.41 GHz
倍精度理論ピーク	5.3 TFlop/s	7.8 TFlop/s	9.7 (19.5) TFlop/s [1]
単精度理論ピーク	10.6 TFlop/s	15.7 TFlop/s	19.5 TFlop/s
半精度理論ピーク	21.2 TFlop/s	125 TFlop/s [2]	312 TFlop/s [3]
メモリ容量	16 GB (HBM2)	16 GB (HBM2)	40 GB (HBM2)
メモリ帯域幅	732 GB/s	900 GB/s	1555 GB/s
L2 キャッシュ容量	4 MB	6 MB	40 MB

[1] カッコ内は Tensor コアを使用した時の理論ピーク性能．
[2] Tensor コアの理論ピーク性能．
[3] Tensor コアの理論ピーク性能．

動作周波数については大きな違いがなく，Streaming Multiprocessor（SM）数が 56，80，108 とおよそ$\sqrt{2}$倍ずつ増えてきたことが性能向上の要因である．また，Volta 世代において導入された Tensor コアはもともと半精度の行列積和算のみを実行する演算機であったが，A100 においては倍精度の行列積和算にも対応したため理論ピーク性能を大きく上げている．さらに Tensor コアは倍精度，半精度に加えて TensorFloat32[4] に対応し，また非ゼロ要素だけを取り出して計算することで最大 2 倍の高速化を実現する Sparsity を導入するなど主に深層学習を意識した強化がなされている．ただし行列の積和算の専用演算機であるため全てのアプリケーションが恩恵を受けられるわけではないこともあり，本稿では Tensor コアについてはこれ以上掘り下げない．

メモリ性能については容量・バンド幅ともに増加してきているが，A100 における増強が大きい．特に L2 キャッシュが 40 MB に増量された恩恵は多くのアプリケーションが受けられるものである．

3. Volta 世代において導入された機能の復習

GPU の構成および CUDA のプログラミングモデルは，V100 および CUDA 9 において大きな変更が加えられ，最新の A100 および CUDA 11 はこの変更点をそのまま引き継いでいる．Reedbush-H/L から Aquarius へと直接移行するユーザはこうした変更点を見落としがちであるので，ここでまとめておく（V100 以降の GPU を活用したことのあるユーザにとっては既知の情報である）．

最大の変更点は“Independent Thread Scheduling”という動作モデルの導入である．P100 に代表される Pascal 世代以前の GPU においては，ワープを構成する 32 スレッドの演算は同時に実

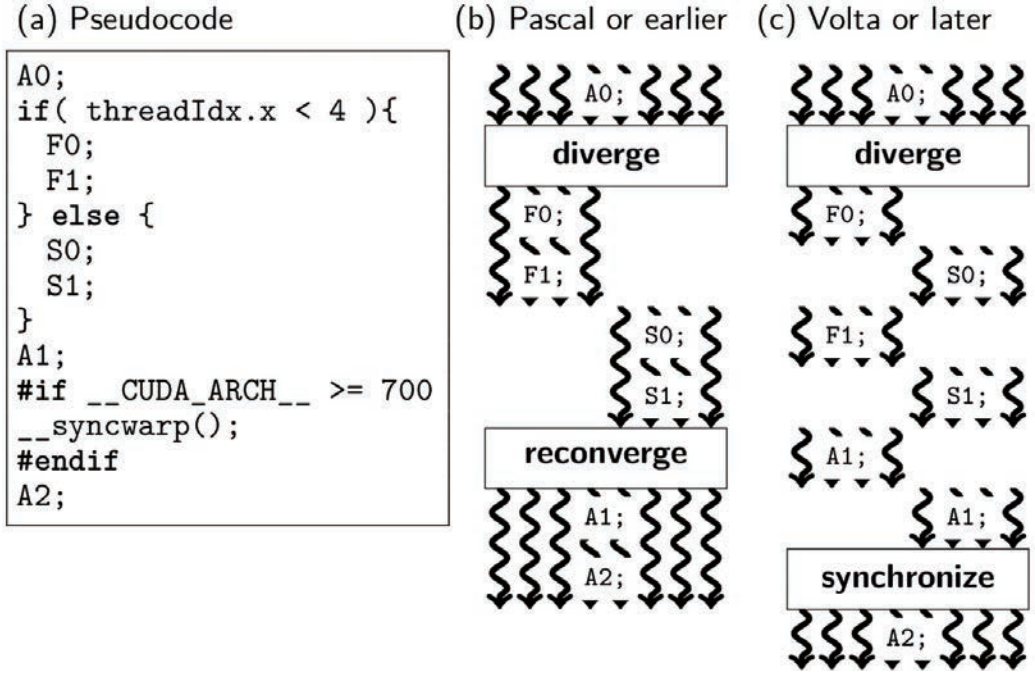

図 1　Independent Thread Scheduling の有無による動作モデルの違い．(a) 条件分岐を含んだコードの例．(b) Pascal 世代以前の GPU 上での動作モデル．(c) Volta 世代以降の GPU 上での動作モデル．

[4] 符合ビット 1，FP16 と同じ仮数部 10 ビットと FP32 と同じ指数部 8 ビットを持つ合計 19 ビットからなるデータ型であり，FP19 と呼ぶ方が実態を表している．

行されていたため，同期が必要な処理をワープ内に閉じ込めるよう実装した際には，明示的な同期命令を発行する必要がなかった．これに対して，Independent Thread Scheduling が導入された V100 や A100 においてはワープ内の 32 スレッドの演算が同時に実行されるという保証はなくなっており，ワープ内の 32 スレッドを同期する__syncwarp()，Cooperative Groups のタイル同期といった明示的な同期処理を行う必要がある（図 1）．特にワープ内の暗黙同期がなくなった影響として，if 文の実行などによってワープ分裂が生じた後には，明示的な同期処理を実行するまではワープが分裂したまま動作し続けるという点に注意が必要である．これは図 1b においては 1 回だけ実行されていた A1 が，図 1c においては 2 回実行されているという動作パターンである．図 1a において__syncwarp()を挿入する位置が A1 の後になってしまっているという実装ミスによって引き起こされる性能低下を回避するにも，慎重なコードの改修が必要となる．一方で，こうしたコードの書き換えが困難なユーザのための回避策も提供されている．通常 A100 向けに CUDA コードをコンパイルする際には「-gencode arch=compute_80,code=sm_80」を指定するが，この際に「-gencode arch=compute_60,code=sm_80」を指定することで Pascal 世代以前の動作モードを使用することができる（ただし，あくまで回避策なので推奨はしないとのことである）．

また P100 までは整数演算は CUDA コアにおいて実行されていたが，V100 からは整数演算用のコアが追加された．これによってインデックス計算などが浮動小数点演算と同時に実行される場合もあり，理論ピーク性能比以上の性能向上が実現されるアプリケーションもある[5]．

4. A100/CUDA 11 で導入された新機能とその使い方

本節では，A100 および CUDA 11 において導入された新機能について紹介する．CUDA 11 は，A100 対応がなされたバージョンの CUDA である[6]．代表的な新機能としては，Asynchronous copy/barrier，L2 cache residency control，Warp-wide reduction があげられる．

CUDA 11 において，グローバルメモリからシェアードメモリへと（レジスタを介さずに）直接データをコピーできる memcpy_async()が導入された．本命令は非同期実行されるため，データ転送と演算を同時に実行することで一方の実行時間を隠蔽できる．また，A100 上ではハードウェア的な加速が効くという利点もある．使い方としては，cuda/pipeline や cuda/barrier をインクルードして libcu++経由で使うこともできるが，本稿では実装が簡単な Cooperative Groups 経由での実装例を紹介しておく（図 2）．Svedin らによってマイクロベンチマークが行われており，pipeline API が barrier API よりも高速であること，メモリ律速な問題では性能が向上すること，演算律速な問題では性能が低下することが報告されている[7]．

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

// デバイス関数内での実装：
  thread_block_tile<TSUB> tile = tiled_partition<TSUB>(this_thread_block());
  // TSUB は，この関数内で memcpy_async に関与するスレッド数（この書き方の場合は32以下）

  cooperative_groups::memcpy_async(tile, &shared_mem, &global_mem, sizeof(uint) * num);
  // 何か裏で回したい処理があれば，ここに書く
  cooperative_groups::wait(tile);// ここでコピーの完了待ちをする
```

図 2　memcpy_async() の実装例．

A100においてはL2キャッシュの容量が40 MBと大幅に増量されたが，グローバルメモリ上の単一・連続なデータ領域をL2キャッシュ上に置き続けることもできるようになった．この際，L2キャッシュ容量の1/16である2.5 MB単位での調整が可能であるが，CUDAストリームごとに1つのデータ領域しか指定できないという制約がある．実装は図3のようになる．

```
cudaStreamAttrValue attr;

attr.accessPolicyWindow.base_ptr = /* beginning of range in global memory */ ;
attr.accessPolicyWindow.num_bytes = /* number of bytes in range */ ;

// hitRatio causes the hardware to select the memory window to designate as per
sistent in the area set-aside in L2
attr.accessPolicyWindow.hitRatio = /* Hint for cache hit ratio: 1.0で良い */

attr.accessPolicyWindow.hitProp = cudaAccessPropertyPersisting;
attr.accessPolicyWindow.missProp = cudaAccessPropertyStreaming;

cudaStreamSetAttribute(stream, cudaStreamAttributeAccessPolicyWindow, &attr);
```

図 3　L2 cache residency control の実装例.

ワープ内のリダクション演算を高速に計算できる__reduce_*_sync()命令が導入された．本命令は「-gencode arch=compute_80,code=sm_80」をつけてコンパイルする必要があるので，暗黙の同期との併用はできない．提供されている演算はadd，min，max，and，or，xorの6種類であり，32ビット整数型にのみ対応している（and，or，xorは符号なし整数型のみに対応）．前述の通り浮動小数点数には対応していないが，minおよびmaxについては大小関係を保存した上で適切な符号なし整数型へと変換することで流用することは可能である[8]．

本節の最後に，A100において顕在化したシェアードメモリ使用時の制限事項をリマインドしておく．デバイス関数内でシェアードメモリを静的に確保する際には，ブロックあたり48 KBが上限の容量である．過去のGPU,例えばV100においてはSMあたり96 KBのシェアードメモリが使用可能であったが，標準的にはSMあたりに複数のブロックを割り当てるため，48 KBの制限は実質的には存在しないと言えた．A100においてはSMあたり164 KB [5]までシェアードメモリを使えるため，例えばSMあたりのブロック数が2であれば静的な確保はできず，この場合には動的な確保が必要となる．

4. 重力ツリーコードGOTHICでの例

今まで紹介してきたA100およびCUDA 11の新機能を実アプリケーションに適用し，A100向けに最適化した事例を紹介しておく．ここでは，宇宙物理学の研究に用いるために開発されたN体シミュレーションコードGOTHIC［5，8，9］を対象とする．

GOTHICにおいては，遠方粒子からの重力を計算する際に多重極展開を用いて演算量を削減するツリー法，軌道進化の遅い粒子に関する重力計算・軌道積分を間引く階層化時間刻み法を採用しており，ほとんどの浮動小数点演算は単精度で実行されている．また，重力計算だけではなくツリー構造の構築などの処理もCUDA C/C++を用いてGPU化されている．

GOTHICのA100向け最適化としては，本稿で紹介した各種機能を使用するかどうかや関連する

[5] 1ブロックあたりでは163 KBまでのシェアードメモリが使用可能である．

パラメータの調整（例えば，L2 キャッシュにデータを固定する際の容量）が必要となる．このためには各パターンでの性能測定を実施し，最も高速なパラメータセットを抽出すれば良い．ただし重力ツリーコードの演算性能には用いる粒子分布に対する依存性もあるため，実際の宇宙物理学の研究で用いられる現実的な状況設定を準備しなければならない．そこで今回はアンドロメダ銀河を模した準力学平衡状態にある粒子分布を初期条件生成コード MAGI [10]によって生成した．また GOTHIC は階層化時間刻み法を採用しており重力計算に要する時間がステップごとに 2 桁以上変動するため，4096 ステップに渡る N 体シミュレーションを実行し，その全体の実行時間を測定することとした．また，この際の粒子数は$N = 2^{23} = 8388608$とした．

A100/CUDA 11 において導入された新機能としては，memcpy_async()については使わない方が高速であった．これは，演算律速な問題においては性能が低下する傾向にあるという Svedin らの報告[7]とも一致する．L2 キャッシュへのデータ固定についてはデータサイズに対する依存性がほぼなかったが，Independent Thread Scheduling 有効時（A100 のデフォルト，compute_80 を指定）においては粒子データを 40 MB 固定した場合，無効化時（compute_60 を指定した時）にはデータ固定を行わない場合が最も速かった．本機能もメモリ律速な問題に効果的と考えられるため，演算律速なアプリケーションである GOTHIC において大きな効果がなかったことは不思議ではない．最後にワープ内のリダクション命令については Independent Thread Scheduling 有効時のみしか使えないが，本命令を使用した方が高速になることが確認できた．また，CUDA のバージョンを変えた実験も行った結果，CUDA 11.2 よりも CUDA 11.1 の方が高速であったため，GOTHIC については CUDA 11.1 環境を用いて動作させることとした．

以下では，Aquarius 上での測定結果を紹介する．図 4 に，重力計算の精度を制御するパラメー

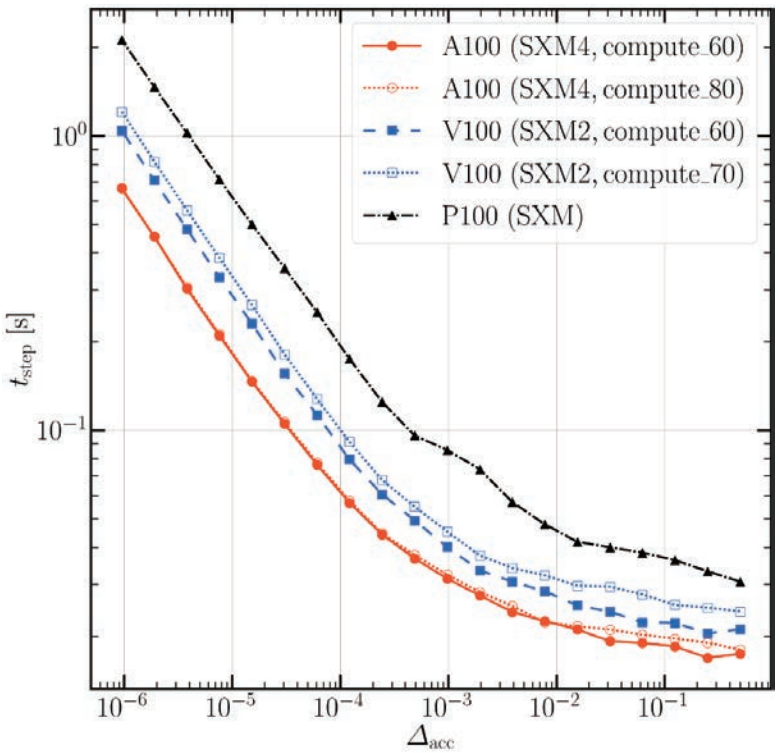

図 4　GOTHIC の実行時間．重力計算の精度を制御するパラメータ$\Delta_{\rm acc}$の関数として，ステップ当たりの実行時間を示した．

タΔ_{acc}の関数としてステップあたりの実行時間を示した．パラメータΔ_{acc}の値が小さいほど重力計算は正確であるので，測定結果が左下方向へ移動するほど高性能であるということを意味する．図 4 からは，赤で示した A100 の測定結果が常に P100 や V100 よりも高速であることが分かる．また，V100 および A100 においては Independent Thread Scheduling の有効時（open symbols）と無効化時（filled symbols）の結果も示した．V100 においては Independent Thread Scheduling を無効化することで 10-20%の高速化が達成されていたが，A100 においては依然無効化時の方が速いものの両者の差は縮まっている．これには，warp-wide reduction による高速化が Independent Thread Scheduling 有効時にしか得られないことが寄与していると考えられる．

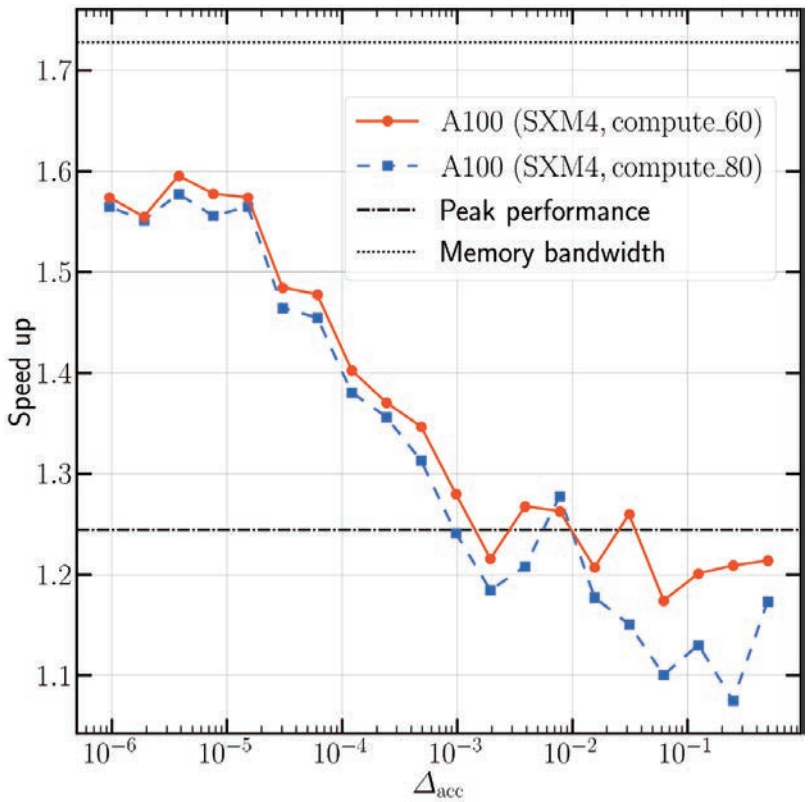

図 5　GOTHIC の速度向上率．

図 5 には，V100 からの速度向上率を示す．一点鎖線は V100 と A100 の単精度理論ピーク性能比，点線はメモリバンド幅比を示している．得られた速度向上率は Independent Thread Scheduling の有無にはあまり依らずにおおむね理論ピーク性能比よりも高い値で推移し，最大で 1.6 倍に達することが分かった．理論ピーク性能比よりも高い速度向上率が得られた理由は分かっていないが，シェアードメモリの容量増加や今回採用したパラメータセットの特性といった要因が考えられる．前者は SM あたりに割り当てられるブロック数の増加につながり，その結果として浮動小数点演算，整数演算，グローバルメモリへのアクセスといった命令の同時実行が容易となり，実行時間の隠ぺいがなされるということである．また後者については，性能に関係するパラメータの設定を行ったマイクロベンチマークは典型的な精度である$\Delta_{acc} = 2^{-9}$を仮定していたため，他の値においては最適な構成とは異なるパラメータセットを使用している可能性があ

る．これは A100・V100 に共通しているため，理論ピーク性能比よりも高い速度向上率の原因となっているかもしれない．

5. まとめ

Wisteria/BDEC-01（Aquarius）に搭載された A100 を使いこなすために，本稿では A100 および CUDA 11 において導入された新機能とその使い方を紹介した．また，活用事例として重力ツリーコード GOTHIC の性能評価結果を紹介した．GOTHIC は一世代前の GPU である V100 上での性能測定結果と比べて最大で 1.6 倍高速であり，演算律速な問題においても理論ピーク性能比よりも高い速度向上率を示す実例があることを示した．

参 考 文 献

[1] Wisteria/BDEC-01: https://www.cc.u-tokyo.ac.jp/supercomputer/wisteria/system.php

[2] NVIDIA Corporation: NVIDIA Tesla P100 (2016)

[3] NVIDIA Corporation: NVIDIA Tesla V100 GPU Architecture (2017)

[4] NVIDIA Corporation: NVIDIA A100 Tensor Core GPU Architecture (2020)

[5] Miki, Y.: Gravitational Octree Code Performance Evaluation on Volta GPU, Proceedings of the 48th International Conference on Parallel Processing, ICPP 2019, New York, NY, USA, ACM, pp. 62:1-62:10 (2019)

[6] NVIDIA Corporation: CUDA C++ Programming Guide Version 11.5 (2021)

[7] Svedin, M., Chien, S. W. D., Chikafa, G., Jansson, N. and Podobas, A.: Benchmarking the Nvidia GPU Lineage: From Early K80 to Modern A100 with Asynchronous Memory Transfers, HEART '21: Proceedings of the 11th International Symposium on Highly Efficient Accelerators and Reconfigurable Technologies, 9:1-9:6 (2021)

[8] 三木洋平： NVIDIA A100 における重力ツリーコードの性能評価，研究報告ハイパフォーマンスコンピューティング（HPC），Vol. 2021-HPC-180, No. 24, pp. 1-7 (2021)

[9] Miki, Y. and Umemura, M.: GOTHIC: Gravitational oct-tree code accelerated by hierarchical time step controlling, New Astronomy, Vol. 52, pp. 65-81 (2017)

[10] Miki, Y. and Umemura, M.: MAGI: many-component galaxy initializer, Monthly Notices of the Royal Astronomical Society, Vol. 475, pp. 2269-2281 (2018)