Vol.2021-HPC-180 No.24
2021/7/21

NVIDIA A100における重力ツリーコードの性能評価

三木 洋平[1,a)]

概要：Volta 世代までの GPU に最適化されていた重力ツリーコード `GOTHIC` を NVIDIA Ampere 世代の GPU である NVIDIA A100 向けに移植し，その性能を評価した．NVIDIA Ampere 世代においても，Volta 世代と同様にコンパイル時に`-gencode arch=compute_60,code=sm_80` を指定してワープ内の暗黙の同期を有効化した方が高速であった．A100（PCIe）を用いて測定した性能は，$N = 2^{23} = 8\,388\,608$ 粒子で表現したアンドロメダ銀河モデルを用いた典型的な精度での計算に要したステップあたりの実行時間が 2.6×10^{-2} s となり，V100（SXM2）上での 3.3×10^{-2} s よりも 1.27 倍高速だった．ここで得られた性能向上は，テンソルコアを除外した場合の A100 と V100 の理論ピーク性能比 1.24 倍よりも大きい．また，GPU のアプリケーションクロックを変化させた測定から，アプリケーションクロックがブーストクロックにより近づくと想定される SXM4 版においては 5 %–10 % 程度の高速化が期待できると示唆する結果を得た．

1. はじめに

重力多体シミュレーション（N 体シミュレーション）は，銀河などの重力多体系の形成・進化過程を研究する上で強力なツールであり，多くの研究で用いられている．N 体シミュレーションでは，粒子間に働く重力を計算し，各 N 体粒子の軌道進化を計算することで系の時間発展を求めている．個々の N 体粒子の加速度は Newton の運動方程式

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

によって計算される．ここで m_i, $\boldsymbol{r}_i$, $\boldsymbol{a}_i$ はそれぞれ i 番目の粒子の質量，位置，加速度であり，また G は重力定数である．N 体シミュレーションで用いる粒子数 N は現実の系の粒子数に比べて桁で少ないため，2 体緩和の効果が現実の系よりも強く入ってしまう．重力ソフトニング ϵ は人工的な 2 体緩和の影響を低減するために導入しており，同時にゼロ割による発散や自己相互作用を取り除くという役割も担う．本研究では，よく用いられる形である Plummer ソフトニングの形を採用する．

（1）式の表式をそのまま用いて重力相互作用を計算する方法（直接法）では計算量が $\mathcal{O}(N^2)$ となり，大粒子数の計算を許容可能な実行時間で終えることは難しい．一方で銀河などの無衝突系の力学進化を調べる際には，個々の粒子間に働く重力相互作用を精密に計算するよりも，系の質量分布をよりなめらかに表現する方が重要である．そこで，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計算量を減らす手法であるツリー法 [1]（演算量は $\mathcal{O}(N \log N)$）などの近似解法がよく用いられている．また，N 体シミュレーションの高速化手法としては GRAPE や GPU などの演算加速器の使用も広く採用されており，GPU を用いたツリー法の高速化も多くの成功を収めてきた [2], [4], [10], [12], [17].

東京大学情報基盤センターは 2021 年 5 月 14 日に Wisteria/BDEC-01 の運用を開始した [24]. Wisteria/BDEC-01 は，シミュレーションノード群 Odyssey とデータ・学習ノード群 Aquarius からなる複合型スーパーコンピュータであり，Aquarius には NVIDIA 社の最新の GPU である NVIDIA A100 がノードあたり 8 枚，全 380 枚搭載されている．NVIDIA A100 の正式名称は「NVIDIA A100 Tensor Core GPU」であり，倍精度理論演算性能 19.5 TFlop/s の半分がテンソルコア由来であるなど，テンソルコアが代表的な構成要素となっている．しかし小行列の積和演算専用ユニットであるテンソルコアは数学関数を用いた計算ができず，全ての科学技術計算がテンソルコアの恩恵を受けられるわけではない．本研究で扱う N 体シミュレーションでは最内カーネルにおいて逆数平方根の計算が必須であり，テンソルコアの活用は不可能である．そこで本研究では，テンソルコアが存在しないものとして議論を進めることとする．

表 1 に，A100 と V100 の比較を示す．A100 は V100 から動作周波数は低下したものの CUDA コア数の増加によっ

[1] 東京大学 情報基盤センター
[a)] ymiki@cc.u-tokyo.ac.jp

表 1　A100 と V100 の比較

	V100（SXM2）	A100	比
単精度理論ピーク性能	15.7 TFlop/s	19.5 TFlop/s	1.24
CUDA コア数	5120	6912	1.35
	(= 80 × 64)	(= 108 × 64)	
GPU Boost Clock	1.53 GHz	1.41 GHz	0.922
グローバルメモリ容量	HBM2 16 GB	HBM2 40 GB	2.5
メモリバンド幅	900 GB/s	1555 GB/s	1.73
L2 キャッシュ容量	6 MB	40 MB	6.67
オンチップメモリ容量	128 KB	192 KB	1.5
熱設計電力（TDP）	300 W	400 W（SXM4）	
		250 W（PCIe）	

て全体性能を向上させており，演算性能の向上よりもメモリ性能の向上の方が大きいといった点が読み取れる．また SXM4 版と PCIe 版では最大動作周波数は共通であるが TDP が異なるため，実際には PCIe 版の方が低いアプリケーションクロックで動作することになると考えられる．

こうした A100 と V100 の違いが実アプリケーションの性能に対して与える影響を評価するため，本研究では重力ツリーコードを NVIDIA Ampere 世代向けに最適化し，PCIe 版の A100 を用いてその性能評価を行う．2 節において対象とした重力ツリーコード GOTHIC の実装概要，3 節において NVIDIA Ampere 世代の GPU へのコード移植について紹介する．その後 4 節において性能評価結果を報告し，5 節では PCIe 版の結果を元に SXM4 版の性能予測について議論する．

2. 重力ツリーコード GOTHIC の概要

著者が開発してきた GOTHIC（Gravitational Oct-Tree code accelerated by HIerarchical time step Controlling）はツリー法と階層化時間刻み法 [9] を採用した重力ツリーコードであり，CUDA C/C++で実装されている [10]．GOTHIC では階層化時間刻み法を採用したため，重力計算関数の計算量はステップごとに数桁にわたって変動する．計算量の削減は重力を受ける粒子の数を実効的に削減したためであり，GPU 内の CUDA コア数を可能な限り動かし続けるためにより並列度の高いアルゴリズムを採用することが望ましい．そこで GOTHIC ではツリー探索を幅優先にすることによって並列度を確保することとした．

また，粒子データの論理構造を表現するツリー構造の作成コストも無視できないため，GOTHIC においては論理構造を複数ステップ使い回すことでツリーの更新コストを低減している．ツリー構造の構築コストと重力計算の計算コストは粒子分布に依存し，なおかつ粒子分布自体が時間発展するため，事前に最適な更新頻度を設定することは不可能である．そこで GOTHIC では，それぞれの関数の実行時間を監視しながら両コストの和が最小化されるようにツリー構造の再構築頻度を自動設定する実装になっている．

重力計算時に多重極展開を許容するかもう一段ツリー構造を潜るか判定する基準としては，重力計算の精度を制御するパラメータ $\Delta_{\rm acc}$ を用いて

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

を満たした時に重力計算を行うという基準 [20], [21] を採用する．ここで，m_J, b_J は遠方粒子群の質量およびサイズ，d_{iJ} は重力を受ける粒子と遠方粒子群の重心との距離であり，$\boldsymbol{a}_i^{\rm old}$ は重力を受ける粒子の 1 ステップ前の時刻における加速度である．この判定基準はよく知られている opening angle を用いた判定に比べて短い計算時間で同じ精度の計算が実行できることが知られている [10], [14]．

ワープ内の各スレッドが独立にツリー判定を実行するとワープ分裂が多発し性能低下の要因となるため，GOTHIC においてはワープ内のスレッドが担当する N 体粒子全てに対して多重極展開が許容される場合にのみ遠方粒子群を相互作用リストに追加することとし，シェアードメモリ上に作成する相互作用リストを共有している．この際，ワープ内のスレッドが担当する N 体粒子群全てを内包する仮想粒子を定義してツリー判定を行うことで，各スレッドは幅優先探索の特性を活かして複数の遠方粒子群に対するツリー判定を同時に実行できる．シェアードメモリ上の相互作用リストの更新時には，ワープ内でのスキャン演算を用いて判定結果を適切に集計し，競合が発生しないように実装している．相互作用リストのサイズが閾値を越えた段階でまとめて重力計算を行い，全ての N 体粒子からの重力を計算し終えるまで相互作用リストの作成・重力計算という手順を繰り返す．

GOTHIC を Volta 世代の GPU である V100 向けに最適化した際には，整数演算ユニットが CUDA コアから独立した結果として，整数演算と浮動小数点演算が同時に実行されたことによる高速化が観測された [12]．Volta 世代においてはワープの動作モデルが変更され，ワープ内の 32 スレッドが暗黙のうちに同期されることがなくなったため，明示的に同期をかける必要が生じた．コンパイル時に`-gencode arch=compute_60,code=sm_70` を指定することで暗黙の同期を有効化でき，以降ではこの動作パターンを Pascal モードと呼ぶこととする．GOTHIC の性能を V100（SXM2）上で測定した際には，Pascal モードは Volta 世代標準の場合に比べて 2 割程度高速であった [12]．

3. NVIDIA A100 へのコード移植

本節では，NVIDIA Ampere 世代におけるワープ内暗黙同期の使用方法（3.1 節），新たに導入された機能やその実装方法（3.2 節）と，マイクロベンチマークに基づいた NVIDIA A100 向けの調整（3.3 節）について紹介する．実アプリケーションにおいて有用であると期待される新機能

については，本研究において採用しなかったものも含めて紹介することとする．

3.1 ワープ内の暗黙同期の復活方法

Volta 世代においてはコンパイル時に`-gencode arch=compute_60,code=sm_70` を指定することで，Pascal 世代以前と同様のワープ内の暗黙同期を使用できた．暗黙同期だけを復活させるコンパイルオプションが新規に提供されることを期待していたが，NVIDIA Ampere 世代においても`-gencode arch=compute_60,code=sm_80` の指定だけが暗黙同期を復活させる唯一の方法である．標準のコンパイル方法である`-gencode arch=compute_80,code=sm_80` 指定時については，以降 Ampere モードと呼ぶこととする．3.2 節において紹介するように`-gencode arch=compute_60,code=sm_80` 指定時には無効化される新機能が複数存在するため，それぞれの機能のメリットを比較した上でどちらのモードを使うか判断する必要がある．

3.2 NVIDIA Ampere 世代・CUDA 11 における新機能

3.2.1 グローバルメモリからシェアードメモリへの非同期データコピー機能

CUDA 11 において，グローバルメモリからシェアードメモリに直接データをコピーできる `memcpy_async()` が導入された [15]．本命令は `async` と入っているように非同期で実行されるため，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，NVIDIA A100 上ではハードウェア的な加速が効く [18] ため，アルゴリズム上非同期実行が不可能であったとしても利用するメリットがある．

本命令を使用する際には，`cooperative_groups/memcpy_async.h` をインクルードして Cooperative Groups 経由で使用する，`cuda/pipeline` あるいは `cuda/barrier` をインクルードして libcu++ 経由で使用するという選択肢がある．ただし`-gencode arch=compute_60,code=sm_80` を指定した Pascal モード時においては，`cuda/pipeline` 経由の場合には `sm_70` が必要である旨のエラーメッセージが出てコンパイルできなかったため，Cooperative Groups 経由で使用することになる．[22] は pipeline API と arrive/wait barrier それぞれを採用したマイクロベンチマーク結果に基づき，pipeline API を用いた実装の方がより高速であると報告している．また，`memcpy_async()` による性能向上には演算密度に対する依存性があり，演算律速な問題においては `memcpy_async()` を使うと性能が劣化する傾向があることが報告されている [22]．N 体シミュレーションは演算律速な問題であるため，本研究ではこの機能を用いないこととした．

3.2.2 L2 キャッシュへのデータ固定

NVIDIA A100 においては L2 キャッシュの容量が 40 MB となり，V100 での 6 MB から大幅に増えた．また，グローバルメモリ上の単一のデータ領域を L2 キャッシュに置き続けることができるようになった [16]．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 [15]．そのため 16 個のデータ領域を指定できても良さそうだが，CUDA ストリームに対して Attribute を追加する形式で指定するため，実際には 1 つのデータ領域しか指定できない．つまり，1 つの CUDA ストリームに対して複数の配列を L2 キャッシュに固定することはできない．

ツリーノードの根に近いデータ領域は多くのスレッドが参照する．幅優先探索を採用している `GOTHIC` においては，このデータ領域がメモリ空間上で連続であるため，L2 キャッシュに固定するメリットが期待される．反対に深さ優先探索の場合には，ツリーノードの根に近いデータ領域はメモリ空間上で離散的に配置されることになるため，L2 キャッシュへの固定機能を使用するのは簡単ではない．ツリーデータについては，重心位置とツリー判定用データを格納した `float4` 型の配列，子セルの情報を格納した `int` 型の配列，ノードの質量を格納した `float` 型の配列という 3 つがある．このうち 1 つしか L2 キャッシュに固定できないため，データ量が一番大きい `float4` 型の配列を固定することとした．

3.2.3 ワープ内でのリダクション処理

NVIDIA Ampere 世代では，ワープ内で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．本命令を使用したコードは`-gencode arch=compute_60,code=sm_80` を指定した Pascal モードではコンパイルできなかったため，Ampere モードでのみ適用できる最適化となる．したがって，本研究で報告する Pascal モードの実装においてはこの機能は用いていない．提供されている演算は `add`, `min`, `max`, `and`, `or`, `xor` の 6 種であり，現段階では整数変数対応となっており浮動小数点演算には対応していない．ただし浮動小数点数についても，大小関係を保存した上で適切な符号無し整数型変数へと変換してやることで最小値・最大値は得られる（実装例は A.1 を参照）．

3.2.4 シェアードメモリ容量の増加

A100 においては，SM あたり最大 164 KB のシェアードメモリが使えるようになった（ブロックあたりの最大は 163 KB）．これによりブロックあたり 48 KB を越えるシェアードメモリを使う場合が現実的になってきた．この場合，シェアードメモリを静的に確保できる上限値が 48 KB であるため，dynamic allocation を用いて確保する必要がある．前世代の V100 では SM あたり最大 96 KB のシェアードメモリが使えたため静的な確保容量の上限を越える場合はありえたが，SM あたりに複数のブロックを割り当てること

表 2　評価環境

CPU	AMD EPYC 7662 64 cores, 2 sockets 2.0 GHz–3.3 GHz
GPU	NVIDIA A100（PCIe）
コンパイラ	GCC 8.3.1 CUDA 11.1.105

が標準的であるため，過去の GPU においては実質的に制限がないと見なせる状態であった．

GOTHIC においては，シェアードメモリにツリー判定用のキュー（格納しきれない分はグローバルメモリに退避させる）と重力計算用の相互作用リストを保持している．ブロックあたりで使用するシェアードメモリの容量は，この 2 つのデータ領域に割り当てる容量によって決定される．ブロックあたりのシェアードメモリ容量が 48 KB 以下である場合には静的確保，これを越える場合には動的確保となるようにマクロを用いて切り替える実装とした．

3.3　パラメータ調整

重力ツリーコードの性能には粒子分布に対する依存性があるため，宇宙物理学の研究で使われる現実的な粒子分布を用いて性能評価する必要がある．そこで本研究では，Fardal らによって構築されたアンドロメダ銀河モデル [3], [5] に恒星ハロー成分 [6], [8] を追加した粒子分布 [12] を採用する．この粒子分布は，ダークマターハロー（Navarro–Frenk–White モデル [13]，$M = 8.11 \times 10^{11}\,\mathrm{M}_\odot$, $r_\mathrm{s} = 7.63\,\mathrm{kpc}$），恒星ハロー（Sérsic モデル [19]，$M = 8 \times 10^{9}\,\mathrm{M}_\odot$, $r_\mathrm{s} = 9\,\mathrm{kpc}$, $n = 2.2$），バルジ（Hernquist モデル [7]，$M = 3.24 \times 10^{10}\,\mathrm{M}_\odot$, $r_\mathrm{s} = 0.61\,\mathrm{kpc}$），銀河円盤（等温・指数関数型 [11]，$M = 3.66 \times 10^{10}\,\mathrm{M}_\odot$, $R_\mathrm{s} = 5.4\,\mathrm{kpc}$, $z_\mathrm{s} = 0.6\,\mathrm{kpc}$, $Q_\mathrm{min} = 1.8$ [23]）からなる．全 N 体粒子の質量が等質量となるように各成分に粒子数を配分し，初期条件生成コード MAGI [11] を用いて力学平衡状態にある粒子分布を生成する．

GOTHIC は階層化時間刻み法を採用しており，最重要部分である重力計算関数においては計算時間がステップごとに数桁に渡って変動するという特性がある．最終目的は time-to-solution を最小化することであるので，性能プロファイラを用いた評価に基づく最適化が困難である．そこで，性能に関係するパラメータを変えながら実行時間を測定し，関数ごとに最適なパラメータを設定するという手順を取る．ツリー構築と重力計算のパラメータ調整は独立に行えるため，ツリー構築に関係したパラメータ設定を先に行い，そこで設定したパラメータを反映した上で重力計算に関連したパラメータを調べる．ここでは $N = 2^{23} = 8\,388\,608$ のアンドロメダ銀河モデルを $\Delta_\mathrm{acc} = 2^{-9} = 1.953125 \times 10^{-3}$ として 4096 ステップだけ計算し，**表 2** に示した評価環境上でその実行時間を測定した．ツリー構築に関係する部分

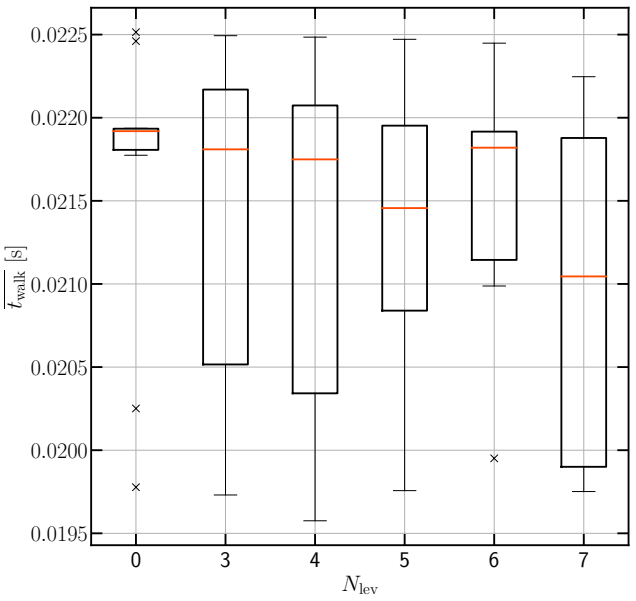

図 1　L2 キャッシュへのデータ固定容量の実行時間への影響．L2 キャッシュに設置したツリーデータの段数 N_lev に対して，1 ステップあたりの重力計算関数の実行時間 $\overline{t_\mathrm{walk}}$ を箱ひげ図を用いて示した．赤線は 10 回の測定値の中央値，箱の上下端は第 1 および第 3 四分位点である．箱の上下端から測定して，箱の長さの 1.5 倍よりも外側の点は外れ値としてバツ印でプロットした．箱の外側の横線は，外れ値を除外した上での最大値または最小値である．

としては Pascal モードと Ampere モードの合計で 192 通りを探索した．この結果として Pascal モードの方が高速であったため，重力計算に関しては Pascal モードのみに絞った上で 8358 通りのパラメータの組み合わせを検証した．

図 1 に，L2 キャッシュに固定するデータ容量を変えながら各パターン 10 回測定した結果を示す．コード内ではツリーデータの段数 N_lev がパラメータとなっており，$N_\mathrm{lev} = 0$ は L2 キャッシュの固定機能を使わない場合，$N_\mathrm{lev} = 1$ はルートのみを固定する場合，$N_\mathrm{lev} = 2$ はルートおよびその子の合計 9 要素を固定する場合に対応する．ツリーノードあたりに `float4` 型変数を 1 つ（16 Byte）用いており，$N_\mathrm{lev} \leq 5$ では容量が最低単位の 2.5 MB に満たないためにこの範囲内では実行時間が一致すると期待される．また $N_\mathrm{lev} = 7$ は容量が 38 MB 程度となり，指定領域全体が L2 キャッシュに収まる最大値である．図 1 からは実行時間のばらつきが大きく，N_lev に対する明確な依存性は読み取れない．これは，N 体シミュレーションが演算律速でありメモリ性能への依存性が弱いためだと考えられる．本研究では，中央値および最悪値が最小となった $N_\mathrm{lev} = 7$ を標準設定とする．

4.　性能測定の結果

GOTHIC の実行時間を，重力計算の精度を制御するパラメータ Δ_acc の関数として測定した（**図 2**）．測定条件

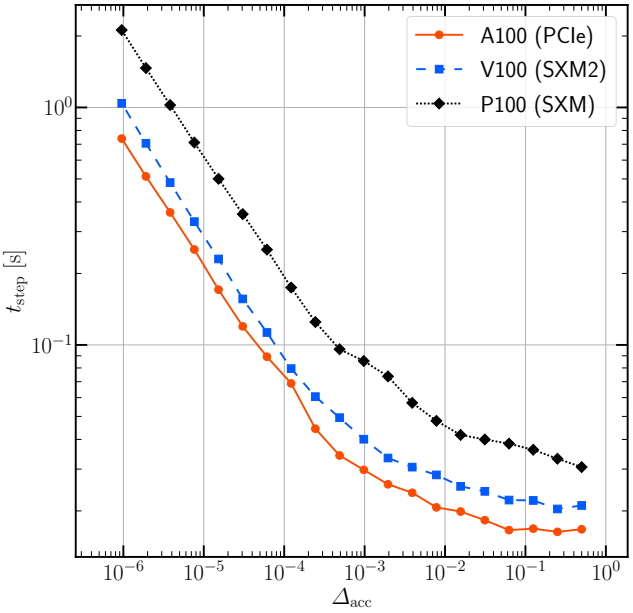

図 2 GOTHIC の実行時間．重力計算の精度を制御するパラメータ Δ_{acc} の関数として，ステップあたりの実行時間を示した（赤丸）．P100（黒の菱形）および V100（青の正方形）を用いた測定結果は [12] において報告済みのデータである．

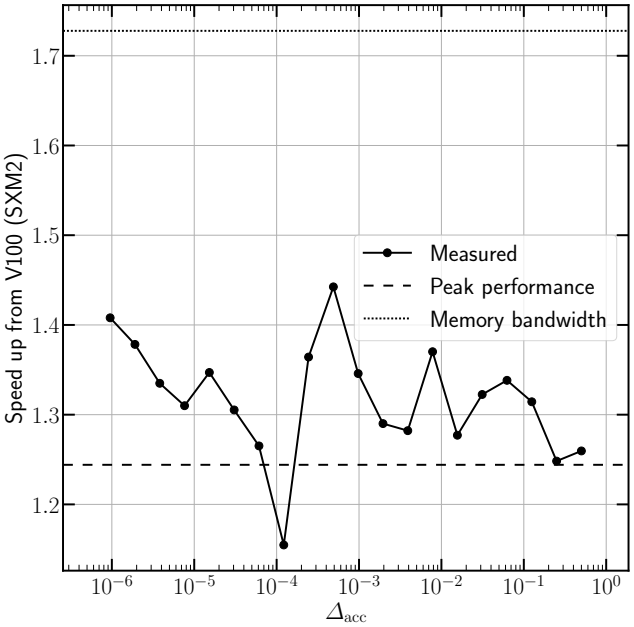

図 3 GOTHIC の速度向上率．A100（PCIe）を用いた際の実行時間を，V100（SXM2）と比較した（黒丸）．得られた速度向上率は，A100 と V100 の理論演算性能比（破線）とメモリバンド幅比（点線）の間に入っている．

は 3.3 節で行ったマイクロベンチマークと同じである（表 2）．また，P100 や V100 といった NVIDIA Ampere 世代よりも前の世代の GPU 上での測定結果については [12] において報告済みのものである．典型的な精度である $\Delta_{\mathrm{acc}} = 2^{-9} = 1.953125 \times 10^{-3}$ におけるステップあたりの実行時間は，P100（SXM）上で 7.4×10^{-2} s，V100（SXM2，Pascal モード）で 3.3×10^{-2} s であったのに対し，A100（PCIe）上では 2.6×10^{-2} s であった．

図 3 に，A100（PCIe）の V100（SXM2）に対する速度向上率を示す．おおむね 1.3 倍程度の高速化が達成されており，これは破線で示した理論演算性能比 1.24 倍よりも大きい．理論演算性能比よりも大きい性能向上は，点線で示したメモリ性能の向上やシェアードメモリ容量の増加といった要因によると考えられる．シェアードメモリ容量の増加は，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，結果としてグローバルメモリからデータを取得する際のアクセスレイテンシの隠蔽に寄与すると考えられる．

5. 議論

NVIDIA A100 の PCIe 版は TDP が 250 W と SXM4 版の TDP である 400 W よりも低く設定されており，ブーストクロックである 1410 MHz よりも低い動作周波数で計算が実行されている可能性がある．N 体シミュレーションは演算律速な問題の典型例であり，その実行時間は動作周波数に反比例すると期待されることを利用して，PCIe 版での測定結果から SXM4 版での測定結果を推測できるか検討する．

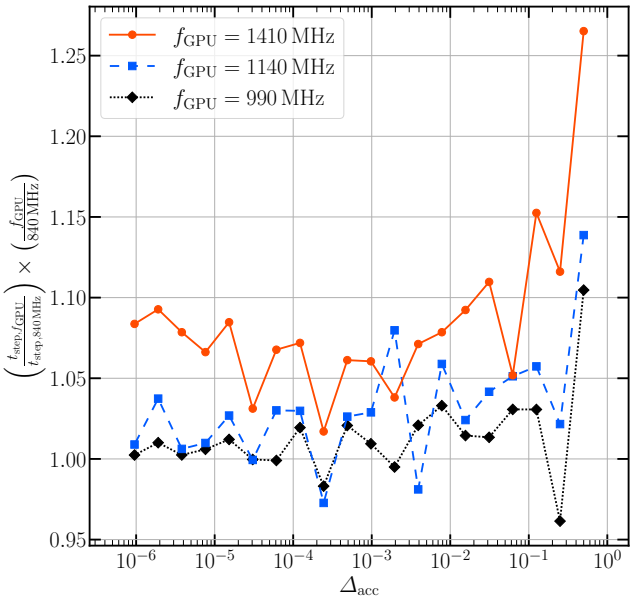

図 4 GOTHIC の実行時間の動作周波数依存性．アプリケーションクロック f_{GPU} を変化させながら測定した結果を，$f_{\mathrm{GPU}} = 840$ MHz の測定結果をベースラインとして示した．縦軸の値は，アプリケーションクロックを指定した際の実行時間と，実行時間が f_{GPU} に反比例するという仮定の下で $f_{\mathrm{GPU}} = 840$ MHz の結果から予想される実行時間の比である．したがって，1 は実行時間が f_{GPU} に反比例すること，1 よりも大きい値は予測よりも実行時間が長いことを意味する．

本研究で用いた NVIDIA A100（PCIe）搭載サーバにはジョブスケジューラとして Slurm が導入されており，Slurm にはジョブ実行時に GPU のアプリケーションクロックをユーザ権限で変更できる機能が実装さ

れている．そこでアプリケーションクロック f_{GPU} を 1410 MHz，1140 MHz，990 MHz, 840 MHz と変えながら実行時間を測定した．**図 4** に，$f_{\mathrm{GPU}} = 840$ MHz の測定結果をベースラインとして実行時間の動作周波数依存性を示す．実行時間が動作周波数に反比例している場合には縦軸の値が 1 となり，$840\,\mathrm{MHz} \leq f_{\mathrm{GPU}} \leq 1140\,\mathrm{MHz}$ の範囲においてはこの関係が成り立っていることが分かる．一方で，$f_{\mathrm{GPU}} = 1410$ MHz の場合には反比例の関係よりも実行時間が長くなっており，$f_{\mathrm{GPU}} = 840$ MHz における測定結果から期待される数値と比較して 5 %–10 % 程度遅いことが分かる．この結果は，使用した GPU が指定したアプリケーションクロックよりも低い周波数で動作している場合に起こると想定され，TDP の制約による周波数低下だと考えられる．したがって，PCIe 版に比べて TDP が大きい SXM4 版は PCIe 版よりも 5 %–10 % 程度速いことが期待される．ただし実際のアプリケーションクロックは GPU の温度に依存するため，GPU の冷却方式やサーバ内のデバイス配置といった要因の影響が大きいと考えられ，環境要因を排除した見積もりをすることは難しい．

6. まとめ

本研究では，Volta 世代までの GPU に最適化されていた重力ツリーコード `GOTHIC` を NVIDIA Ampere 世代の GPU である NVIDIA A100 向けに移植し，その性能を評価した．NVIDIA Ampere 世代においては warp reduce 命令や `cuda/pipeline` 経由での `memcpy_async()` 命令といった新機能が導入されたが，Volta 世代において導入された Individual Thread Scheduling を無効化すると使えなくなる機能も多い．`GOTHIC` においては，新機能の活用による高速化よりも Individual Thread Scheduling の無効化による高速化の方が効果が大きいことが分かった．

A100（PCIe）を用いて測定した性能は，$N = 2^{23} = 8\,388\,608$ 粒子で表現したアンドロメダ銀河モデルを用いた典型的な精度での計算に要したステップあたりの実行時間が 2.6×10^{-2} s となり，V100（SXM2）上での 3.3×10^{-2} s よりも 1.27 倍高速だった．ここで得られた性能向上は，テンソルコアを除外した場合の A100 と V100 の理論ピーク性能比 1.24 倍よりも大きい．また，GPU のアプリケーションクロックを変化させた測定から，アプリケーションクロックがブーストクロックにより近づくと想定される SXM4 版においては 5 %–10 % 程度の高速化が期待できると示唆する結果を得た．

謝辞 本研究は JSPS 科研費 JP20K14517 および JP20H00580 の助成を受けた．

参考文献

[1] Barnes, J. and Hut, P.: A hierarchical O(N log N) force-calculation algorithm, *Nature*, Vol. 324, pp. 446–449 (online), DOI: 10.1038/324446a0 (1986).

[2] Bédorf, J., Gaburov, E., Fujii, M. S., Nitadori, K., Ishiyama, T. and Portegies Zwart, S.: 24.77 Pflops on a Gravitational Tree-Code to Simulate the Milky Way Galaxy with 18600 GPUs, *ArXiv e-prints* (2014).

[3] Fardal, M. A., Guhathakurta, P., Babul, A. and McConnachie, A. W.: Investigating the Andromeda stream - III. A young shell system in M31, *Monthly Notices of the Royal Astronomical Society*, Vol. 380, pp. 15–32 (online), DOI: 10.1111/j.1365-2966.2007.11929.x (2007).

[4] Gaburov, E., Bédorf, J. and Portegies Zwart, S.: Gravitational tree-code on graphics processing units: implementation in CUDA, *Procedia Computer Science, volume 1, p. 1119-1127*, Vol. 1, pp. 1119–1127 (online), DOI: 10.1016/j.procs.2010.04.124 (2010).

[5] Geehan, J. J., Fardal, M. A., Babul, A. and Guhathakurta, P.: Investigating the Andromeda stream - I. Simple analytic bulge-disc-halo model for M31, *Monthly Notices of the Royal Astronomical Society*, Vol. 366, pp. 996–1011 (online), DOI: 10.1111/j.1365-2966.2005.09863.x (2006).

[6] Gilbert, K. M., Guhathakurta, P., Beaton, R. L., Bullock, J., Geha, M. C., Kalirai, J. S., Kirby, E. N., Majewski, S. R., Ostheimer, J. C., Patterson, R. J., Tollerud, E. J., Tanaka, M. and Chiba, M.: Global Properties of M31's Stellar Halo from the SPLASH Survey. I. Surface Brightness Profile, *The Astrophysical Journal*, Vol. 760, p. 76 (online), DOI: 10.1088/0004-637X/760/1/76 (2012).

[7] Hernquist, L.: An analytical model for spherical galaxies and bulges, *The Astrophysical Journal*, Vol. 356, pp. 359–364 (online), DOI: 10.1086/168845 (1990).

[8] Ibata, R. A., Lewis, G. F., McConnachie, A. W., Martin, N. F., Irwin, M. J., Ferguson, A. M. N., Babul, A., Bernard, E. J., Chapman, S. C., Collins, M., Fardal, M., Mackey, A. D., Navarro, J., Peñarrubia, J., Rich, R. M., Tanvir, N. and Widrow, L.: The Large-scale Structure of the Halo of the Andromeda Galaxy. I. Global Stellar Density, Morphology and Metallicity Properties, *The Astrophysical Journal*, Vol. 780, p. 128 (online), DOI: 10.1088/0004-637X/780/2/128 (2014).

[9] McMillan, S. L. W.: The Vectorization of Small-N Integrators, *The Use of Supercomputers in Stellar Dynamics* (Hut, P. and McMillan, S. L. W., eds.), Lecture Notes in Physics, Berlin Springer Verlag, Vol. 267, p. 156 (online), DOI: 10.1007/BFb0116406 (1986).

[10] Miki, Y. and Umemura, M.: GOTHIC: Gravitational oct-tree code accelerated by hierarchical time step controlling, *New Astronomy*, Vol. 52, pp. 65–81 (online), DOI: 10.1016/j.newast.2016.10.007 (2017).

[11] Miki, Y. and Umemura, M.: MAGI: many-component galaxy initializer, *Monthly Notices of the Royal Astronomical Society*, Vol. 475, pp. 2269–2281 (online), DOI: 10.1093/mnras/stx3327 (2018).

[12] 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 (online), DOI: 10.1145/3337821.3337845 (2019).

[13] Navarro, J. F., Frenk, C. S. and White, S. D. M.: Simulations of X-ray clusters, *Monthly Notices of the Royal Astronomical Society*, Vol. 275, pp. 720–740 (online), DOI: 10.1093/mnras/275.3.720 (1995).

[14] Nelson, A. F., Wetzstein, M. and Naab, T.: Vine–A Numerical Code for Simulating Astrophysical Systems

Using Particles. II. Implementation and Performance Characteristics, *The Astrophysical Journal Supplement*, Vol. 184, pp. 326–360 (online), DOI: 10.1088/0067-0049/184/2/326 (2009).
[15] NVIDIA: *NVIDIA A100 Tensor Core GPU Architecture* (2020).
[16] NVIDIA: *CUDA C++ Best Practices Guide Version 11.3* (2021).
[17] Ogiya, G., Mori, M., Miki, Y., Boku, T. and Nakasato, N.: Studying the core-cusp problem in cold dark matter halos using N-body simulations on GPU clusters, *Journal of Physics Conference Series*, Vol. 454, No. 1, p. 012014 (online), DOI: 10.1088/1742-6596/454/1/012014 (2013).
[18] Ramarao, P.: CUDA 11 Features Revealed (2020).
[19] Sérsic, J. L.: Influence of the atmospheric and instrumental dispersion on the brightness distribution in a galaxy, *Boletin de la Asociacion Argentina de Astronomia La Plata Argentina*, Vol. 6, p. 41 (1963).
[20] Springel, V.: The cosmological simulation code GADGET-2, *Monthly Notices of the Royal Astronomical Society*, Vol. 364, pp. 1105–1134 (online), DOI: 10.1111/j.1365-2966.2005.09655.x (2005).
[21] Springel, V., Yoshida, N. and White, S. D. M.: GADGET: a code for collisionless and gasdynamical cosmological simulations, *New Astronomy*, Vol. 6, pp. 79–117 (online), DOI: 10.1016/S1384-1076(01)00042-2 (2001).
[22] Svedin, M., Chien, S. W. D., Chikafa, G., Jansson, N. and Podobas, A.: Benchmarking the Nvidia GPU Lineage (2021).
[23] Tenjes, P., Tuvikene, T., Tamm, A., Kipper, R. and Tempel, E.: Spiral arms and disc stability in the Andromeda galaxy, *Astronomy and Astrophysics*, Vol. 600, p. A34 (online), DOI: 10.1051/0004-6361/201629991 (2017).
[24] 中島研吾, 塙　敏博, 下川辺隆史, 伊田明弘, 芝　隼人, 三木洋平, 星野哲也, 有間英志, 河合直聡, 坂本龍一, 近藤正章, 岩下武史, 八代　尚, 長尾大道, 松葉浩也, 荻田武史, 片桐孝洋, 古村孝志, 鶴岡　弘, 市村　強, 藤田航平: 「計算・データ・学習」融合スーパーコンピュータシステム「Wisteris/BDEC-01」の概要, 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-179, No. 1 (2021).

付　　録

A.1 ワープ内リダクション命令を用いた浮動小数点数の最大値・最小値取得方法

ワープ内での最大値・最小値を高速に計算できる`__reduce_max_sync()`と`__reduce_min_sync()`命令であるが，対象となるデータ型は`int`型あるいは`unsigned int`型のみであり，`float`型を直接渡すことはできない．ただし radix sort の実装で用いられるような適切なビットフリップ[*1]を施すことで，順序を保存した整数型データと見せかけることはできるので，最大値・最小値の取得は可能となる．具体的には，**図 A·1** に示した前処理・後処理を施せば良い．

```
typedef unsigned int uint;
__device__ __forceinline__
uint flip(const uint src){uint mask = -int(src
     >> 31)   | 0x80000000;  return (src ^
    mask);}
__device__ __forceinline__
uint undo(const uint src){uint mask = ((src >>
     31) - 1) | 0x80000000;  return (src ^
    mask);}

__device__ __forceinline__
float get_min(float val, const uint mask){
    union {uint u; float f;} tmp;
    tmp.f = val;
    tmp.u = undo(__reduce_min_sync(flip(tmp.u)
        , mask));
    return (tmp.f);
}
```

図 A·1　浮動小数点数の最小値を取得する実装例.

[*1] http://stereopsis.com/radix.html