超並列重力多体問題シミュレーションコードの性能測定

石 山 智 明

筑波大学計算科学研究センター

概要

本研究の目的はスーパーコンピューター「京」で最適化された重力多体問題シミュレーションコード、”GreeM” を後継機である FX10 上で性能評価し、「京」上で実施してきた最適化が FX10 上でも有効かどうかを検証することである。また多スレッド多並列環境においての問題点を抽出し、きたるエクサスケールに備えた最適化とアルゴリズム改良のための準備を行うことである。シミュレーションコードはすでに「京」上で最大限チューニングされており、フルシステム(82944 ノード)を用いて 55%程度の対ピーク性能と良好なスケーラビリティが得られている。

ダークマター粒子数 4096^3 の宇宙論的 N 体シミュレーションを 4800 ノードで行った場合、「京」では対ピーク性能 51%の実行効率が得られたが、FX10 でも 45%とそれほど遜色のない性能が得られた。パート毎に見ると、ノードあたりのネットワーク性能が同じであるにも関わらず、全対通信、隣接通信ともに「京」が速いことがわかった。また並列性が自明な部分は、「京」、FX10 双方でピーク性能に比例するような性能が得られた。一方、自明でない、または難しい部分は、予想通り並列度が増えた恩恵をさほど受けられないばかりか、性能が逆転するケースもあった。

1. ダークマター構造形成

最新の宇宙の構造形成理論では、普段われわれが目にする原子や分子のような**バリオン**と呼ばれる物質は、宇宙の物質全体の 15%程度の割合で存在し、残りは**ダークマター**[1]と呼ばれる、重力を通してのみ相互作用する物質であると考えられている。ダークマターは宇宙初期にはほとんど一様に存在したが、ごくわずかな密度の揺らぎが存在した。この揺らぎは重力により成長して、宇宙の至るところにダークマターハローとよばれるさまざまなサイズの高密度なダークマター天体を形成した。はじめに小さいハローができて、それらが繰り返し合体することで、より大きなハローができていった。そしてハローの中でバリオンがダークマター重力によって集まり、現在の宇宙で観測される銀河や銀河団のような多種多様な構造が生まれてきた。この意味で、ダークマターは天体形成に必要不可欠な存在である。またハローの構造形成過程や分布は、銀河や銀河団の形成進化や分布をも決定づける。

こういった宇宙の構造形成過程は、重力の非線形性が本質的に重要な役割を果たしているため、そのダイナミクスを研究するには数値シミュレーションが非常に有用である。

---

[1] さまざまな観測から間接的に存在が確かめられているが、ダークマターの素粒子としての正体そのものはよくわかっていない。ダークマター粒子が「冷たい」、つまり速度分散が小さいモデルは現在の宇宙を非常に良く再現する。冷たいダークマターに基づいた構造形成理論は CDM (cold dark matter) 理論と呼ばれる。

図 1: さまざまなスケールで存在するダークマターハローとそこでできる天体

宇宙ではじめにできるダークマターハロー（最小ハロー）は地球質量（10^{-6} 太陽質量）程度であると考えられている。初代星はおよそ 10^{5-6} 太陽質量のハローの中でバリオンが集まってできたと考えられている（画像は初代星ではなく太陽である。初代星は太陽の数十倍の質量であると考えられている）。銀河は 10^{12} 太陽質量、銀河団は 10^{15} 太陽質量程度のハローの中に存在している。

ダークマターハローは最小のものでは地球質量程度（10^{-6} 太陽質量）[2]、最大のものでは銀河団質量程度（10^{15} 太陽質量）の 20 桁以上にもおよぶ非常に広い質量幅に渡って存在すると考えられており、それぞれの中で初代星や銀河などの対応する天体が形成する（図 1）。宇宙の構造形成過程は、こういったマルチスケールな天体が宇宙約 140 億年という長い時間において相互作用するという極めて複雑な過程である。したがって、考慮すべき質量・空間・時間スケールが広大なため、大規模シミュレーションが必要不可欠である。また宇宙空間でみられる多くの物理現象は地上で実験して検証することが不可能であるため、シミュレーションでのみ扱うことのできる対象も数多い。時にシミュレーションは「理論の望遠鏡」とも呼ばれ、宇宙の研究において非常に重要な位置を占めている。

[2] ダークマター粒子として質量が 100GeV 程度の、「超対称性粒子」、「ニュートラリーノ」を仮定した場合。最小ハローのスケールは、ダークマター粒子による無衝突減衰によって密度揺らぎがならされるスケールに対応している。他のダークマターの候補として「アクシオン」が挙げられる。

2. シミュレーション

本研究で行うシミュレーションの対象は、宇宙の重力的な構造形成に主要な役割を担ってきたダークマターである。シミュレーション手法は**重力多体シミュレーション**、いわゆる N 体シミュレーションである。重力多体シミュレーションでは、N 個の粒子間にはたらく重力相互作用を計算し、粒子の時間発展を追う。ダークマターの他に、星団や銀河、ブラックホールなど適用範囲は非常に幅広い[3]。

ダークマターシミュレーションで、宇宙初期の密度揺らぎを持ったダークマターの分布を離散化して粒子として表現し、重力相互作用による時間発展を追うものを、特に**宇宙論的 N 体シミュレーション**と呼ぶ。無限遠方からの重力を取り入れるために周期境界条件が用いられる。初期条件は宇宙背景放射の観測などから非常に良く制限されている。対象とする天体によって、扱う領域はさまざまである。銀河の空間分布を調べたい場合は、観測できる宇宙の全領域にせまる領域を扱い、銀河の微細構造を調べる時は一つのハローだけを扱う。また初代星形成や初代ハロー形成を追う場合は、宇宙初期のごく限られた狭い領域を取り扱う。図 2 に広い領域を扱った宇宙論的 N 体シミュレーションの一例を示す。ダークマター粒子数 2048^3 を用いた宇宙の大規模構造形成シミュレーションであり、はじめは小さかった密度揺らぎが重力不安定で成長していき、現在の宇宙まで至る様子がわかる。これは観測された宇宙の大規模構造と非常に良く似ており、二点相関関数などの手法を用いて統計的に比較することで、構造形成過程そのものの研究を行うことが可能となる。

宇宙論的 N 体シミュレーションでは、前述のように大規模化が必要不可欠であるため、大規模並列コードの開発が世界のいくつかのグループで進められてきた。我々のグループでも C++ による OpenMP+MPI 並列シミュレーションコード、”GreeM” を開発してきた（Ishiyama et al. 2009）。もともとは国立天文台のスーパーコンピューター Cray-XT4（Opteron 2.2 GHz, 740 ノード、2960 コア）上で開発を進めていたが、さらに理化学研究所計算科学研究機構の「京」上で、フルシステムである 8 万ノードでも効率良く動作するように最適化を行った。その結果良好なスケーラビリティと、「京」のフルシステムを用いた場合、最大 5.8Pflops の実効性能（55% の対ピーク性能比）を達成した。この成果が認められ、「ハイ・パフォーマンス・コンピューティングに関する国際会議 SC12」（2012 年 11 月、米国ソルトレークシティー開催）において、**ゴードン・ベル賞**を単独受賞した。

ゴードン・ベル賞ファイナリストには、ピーク性能が 20 ペタフロップス（「京」の約 2 倍）の「セコイア」（米国ローレンス・リバモア国立研究所）を使い、同様のダークマターシミュレーションで 14 ペタフロップスを達成した米国のグループがあった。ところがわれわれのグループのアプリケーションが実際の計算速度で上回り、1 粒子あたり 2.4 倍の速さでシミュレーションをすることが可能であった（米国と同じ計算機を用いた場合は 5 倍近く速い）。こういった点が評価され、ゴードン・ベル賞の受賞につながった。

---

[3] ダークマター構造形成や銀河は二体緩和時間が対象とする時間スケールより十分長い。このような系は無衝突系と呼ばれ、高精度な重力計算や時間積分よりも、粒子数をできるだけ大きくすることが本質的に重要である。一方、星団やブラックホールは二体緩和時間が対象とする時間スケールより短く、粒子の近接遭遇が系全体の進化に影響を与える。このような系は衝突系と呼ばれ、高精度な重力計算や時間積分が要求される。無衝突系ではツリー法のような近似を用いて重力を計算し、2 次程度の時間積分公式が用いられることが多いが、衝突系では式(1)を直接解き、4 次や 6 次の時間積分公式が用いられることが多い。

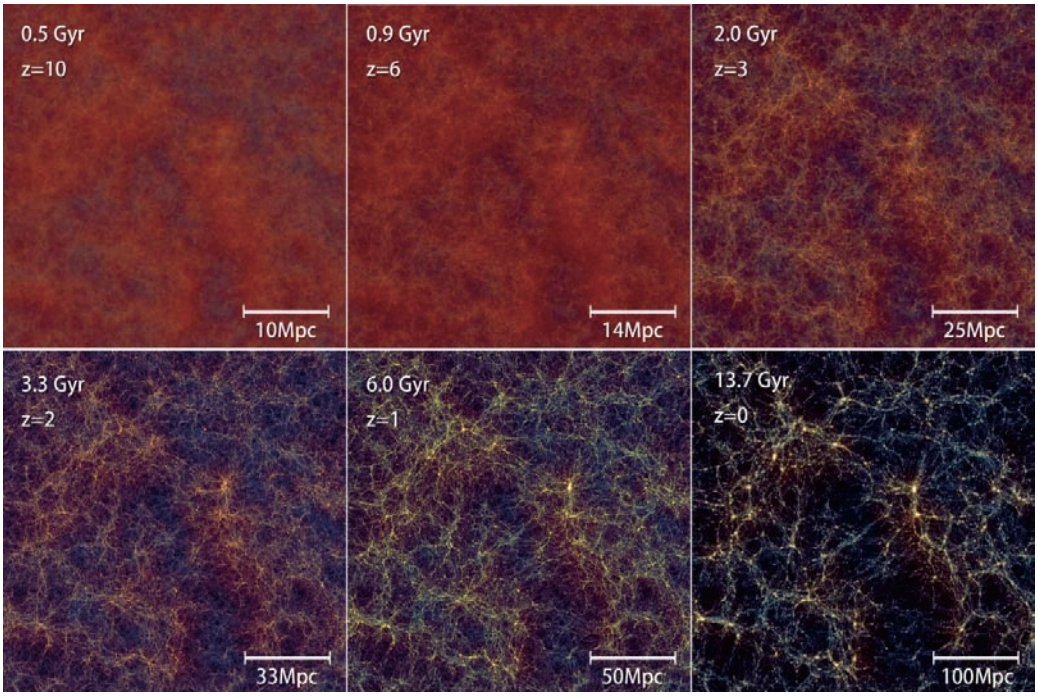

図 2: 宇宙論的 N 体シミュレーションによる、宇宙の大規模構造形成シミュレーション。

各パネルは左上→右上→左下→右下の順に時間進化を表す。左上の数値は宇宙年齢で z=0 は現在の宇宙(宇宙誕生からおよそ 137 億年後)に対応する。右下の数値は物理的な空間スケールを表しており、pc(パーセク)は約 3.26 光年である。

以下 GreeM で採用している重力相互作用計算アルゴリズム、TreePM 法や、並列化手法について述べる。

2.1 TreePM 法

粒子間に働く重力加速度 **f** は通常次のように表される。

$$\mathbf{f}(\mathbf{r}') = \frac{m(\mathbf{r}-\mathbf{r}')}{|\mathbf{r}-\mathbf{r}'|^3}. \tag{1}$$

ここで、**r'**、**r**、m はそれぞれ力を受ける粒子の座標、力を及ぼす粒子の座標、質量である。万有引力定数 G は省略してある。全粒子の相互作用をこのまま直接計算すると、相互作用数はペア数に比例するので、粒子数 N に対して演算量は $O(N^2)$ となり非常に大変である。そこで開発されたのが、ツリー法 (Barnes and Hut, 1986) や PM (Particle-Mesh) 法 (Hockney and Eastwood, 1981) などの近似法である。

ツリー法では、近傍の粒子からの力は直接計算するが、十分遠方の粒子群からの力はまとめて多重極展開を使って計算する[4]。このため 8 分木構造が良く使われる(図 3)。ツリー法では計算量を $O(N \log N)$ に落とすことが可能である。

[4] monopole (重心)まで計算することが多いが、コードによっては、quadrupole、octopole、hexadecapole まで計算することもある。

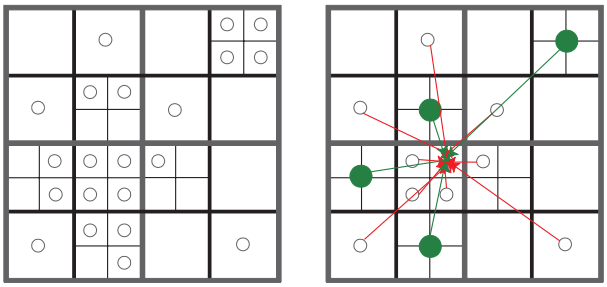

図 3: ツリー法の概念図。

(左)8 分木構造。(右)ある粒子が受ける力。緑線は多重極展開として力を計算していることを表す。

PM 法では、まず粒子の位置と質量から一様格子状の密度場を計算し、FFT とポアソン方程式から一様格子状のポテンシャルを計算する。そして差分と補間から、各粒子位置への加速度を計算する。PM 法では計算量を O(MlogM) (M はメッシュ数)に落とすことができ、宇宙論的 N 体シミュレーションに必要な周期境界条件を自然に組み込むことができるという点で非常に強力であるが、分解能がメッシュ数によって制限される[5]。粒子分布が一様に近い場合はともかく、構造が発達した箇所はより細かいメッシュが必要となり、粒子数 N に対して M>>N が要求されるため、PM 法だけでは計算量の観点からも、メモリ使用量の観点からも不都合である。

そこで考え出されたのが P^3M 法と TreePM 法である[6] (図 4)。まず力を近距離力 $\mathbf{f}_{sr}$ と遠距離力 $\mathbf{f}_{lr}$ に分割する。すなわち

$$\mathbf{f}(\mathbf{r}') = \mathbf{f}_{\rm sr}(\mathbf{r}') + \mathbf{f}_{\rm lr}(\mathbf{r}'), \tag{2}$$

とし、近距離力は式(1)に遠距離で 0 になるようなカットオフがついたカーネル g_{p3m} を掛けたものを用いる。

$$\mathbf{f}_{\rm sr}(\mathbf{r}') = \frac{m(\mathbf{r}-\mathbf{r}')}{|\mathbf{r}-\mathbf{r}'|^3} g_{\rm P3M}(|\mathbf{r}-\mathbf{r}'|/\eta), \tag{3}$$

ここ η はスケール長で、粒子間の距離を規格化する。カーネルはいくつか流儀があるが、ここでは以下の多項式型のものを用いる。

$$g_{\rm P3M}(R) = \begin{cases} 1 - \frac{1}{140}(224R^3 - 224R^5 + 70R^6 \\ +48R^7 - 21R^8) \quad \text{for} \quad 0 \le R < 1 \\ \\ 1 - \frac{1}{140}(12 - 224R^2 + 896R^3 - 840R^4 + 224R^5 \\ +70R^6 - 48R^7 + 7R^8) \quad \text{for} \quad 1 \le R < 2 \\ \\ 0 \quad \text{for} \quad 2 \le R. \end{cases} \tag{4}$$

このカーネルでは 2η がカットオフ長となる。近距離力を直接解いて、遠距離力を PM 法で計算

[5] PM を再帰的に適応して、高分解能を実現するアルゴリズムも考案されており、たとえば AMR 法 (Adaptive Mesh Refinement) などがある。AMR も宇宙論的 N 体シミュレーションに良く用いられる。

[6] PM の代わりに分子動力学で良く用いられる Ewald 法が使われることもある。

するのが P^3M (particle-particle paricle-mesh)法である。構造が発達し、いたるところに高密度な構造ができると、直接計算での計算量が非常に大きくなるので、P^3M 法ではやはり大変である。そこで近距離力をツリー法で解くことでそれを防ぐのが TreePM 法であり、現在宇宙論的 N 体シミュレーションで最も良く用いられているアルゴリズムである。比較的新しいアルゴリズムであり、原案が提唱されたのは1995年である (Xu 1995)。広く使われるようになったのは 21 世紀に入ってからで、いくつかの並列コードが開発されている (GOTPM: Dubinski et al. 2004、GADGET-2: Springel 2005、GreeM: Ishiyama et al. 2009 など)。

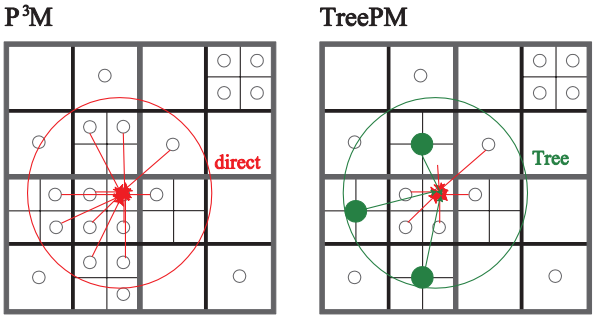

図 4: P^3M 法と TreePM 法の概念図。
(左)P^3M 法。(右)TreePM 法。円は近距離力のカットオフを表し、この外からの力は PM の寄与のみとなる。

最適な誤差と計算量の観点から、カットオフ半径は PM のメッシュ間距離の 3 倍程度にすることが多い。また PM のメッシュ数を全粒子数と等しくするか、全粒子数の 1/8 (メッシュ間距離が平均粒子間距離の 2 倍)とすることが多い。TreePM 法の最大のホットスポットは、ツリー法で重力相互作用を計算する部分である。つまり全粒子の加速度を式(3)、(4)を用いて計算する相互作用カーネル部であり、我々のコード GreeM では、全計算の 6〜7 割を占める。歴史的には、この部分は重力多体問題専用計算機、GRAPE (GRAvity PiPE: Sugimoto et al. 1990 など多数)や、GPU (Hamada et al. 2009 など)を用いて加速されてきた。一方、汎用計算機上でも、SSE や AVX などの SIMD 拡張命令を用いて高速に計算することを可能とするライブラリ、Phantom-GRAPE (Nitadori et al. 2006, Tanikawa et al. 2012, 2013) が開発され、WEB 上で公開されている[7]。「京」と FX10 でのシミュレーションでは、共同研究者の似鳥啓吾氏(2013 年 1 月現在、理化学研究所計算科学研究機構粒子系シミュレータ研究チーム研究員)が開発した HPC-ACE に対応したものを用いている。式(3)、(4)を 1 相互作用あたり 51 演算で計算することができ、FMA 部と非 FMA 部の割合が 1:1 (つまり 17 命令ずつ)であるので、理論ピーク性能は、CPU のピーク性能の 75%となる。カーネルだけの単純なベンチマークテストでは、72%程度の実行効率(理論ピークの 96%)を達成した。

2.2 並列化

並列化は領域分割に基づく。シミュレーション空間をノード数で分割し、各ノードは領域内にある粒子が受ける力を計算する。我々のコードでは、再帰的多段分割法(Makino 2004)に基づいた 3 次元領域分割を行う。この領域分割では使用するノード数が、任意の 3 つの数の積で表されれば何でも良い。またネットワークトポロジーに合致するように空間分割を設定すること

[7] http://code.google.com/p/phantom-grape/
AVX に対応したものや、精度が異なるいくつかのバリエーションが存在する。

で通信を最適化することが可能である[8]。各ステップにおいて、近距離力の計算に必要なカットオフ半径内のツリー情報は隣接ノードからもらい、遠距離力の計算に必要な自領域をカバーするローカルメッシュ上のポテンシャルは、全対通信が含まれる FFT により計算される。

宇宙論的 N 体シミュレーションでは、いたるところに高密度な構造が発達する。したがってできる構造に応じた空間分割や、動的分割が必要不可欠である。プログラム中最大のホットスポットはツリー法による重力相互作用計算部であるが、構造が発達した箇所は計算量も多くなるので、ツリー部分の相互作用の計算量がいっそう大きくなる。したがって粒子数が均等になるような領域分割では、構造が発達するとロードバランスが非常に悪化する。これを解決するために、各ノードが計算する相互作用の総数が均等になるように領域分割をする方法が多用される(Dubinski et al. 2004、 Springel 2005 など)。この方法では、構造が発達した箇所は相互作用数が多くなるので、粒子数を減らして相互作用数を均等にしようとする。したがって粒子数の不均一が大きくなる傾向にある。これはツリー構築や PM 部分で若干のロードバランスの低下を招く。

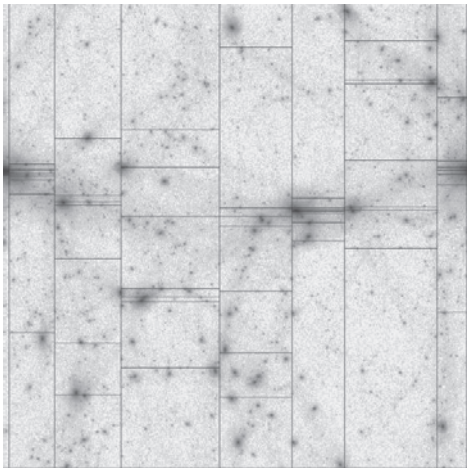

図 5: 領域分割の例。

8x8 分割されている。

そこで GreeM では完全に相互作用の総数が均等になるようにするのではなく、それに若干の補正を加え全体のロードバランスの低下を抑えている。それにはまずツリー法と PM 法の計算全体にわたってプログラム中でかかった時間を計測する。そして領域分割の際にこの時間が均等になるように分割を行っている。単純にホットスポットの情報のみを用いるのではなく、他の部分の情報を用いることでロードバランスの低下を最大限防いでいる。

領域分割の例を図 5 に示す。構造ができる箇所で細分化されているようすがわかる。

なお宇宙論的 N 体シミュレーションのために、世界で最も良く使われている公開コードは GADGET-2(Springel 2005)[9]である。このコードは我々と同じ TreePM 法を用いているが、領域分割に peano-hilbert-ordering を用い、ロードバランスの調整に相互作用数を用いている。このコードと我々のコードの並列化効率の違いを図 6 に示す。なお Phantom-GRAPE を用いて、相

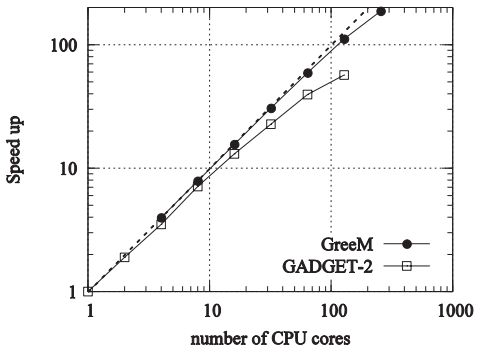

図 6: 我々のコード GreeM と GADGET-2 の並列化効率の比較。

使用した CPU コア数に対する加速率を表す。

[8] 「京」や FX10 では各ノードに 1〜3 次元の仮想トーラス座標が設定される。たとえば、「京」のフルシステムでは、全 82944 ノードを 48x54x32 の 3 次元の仮想トーラスとして割り当てることが可能である。

[9] http://www.mpa-garching.mpg.de/gadget/

互作用計算部をSIMD化していることにより、そもそも我々のコードの方が数倍早い。一般的に遅いコードは並列化効率を出しやすいが、我々のコードはGADGET-2と比較して圧倒的に良い並列化効率を示していることがわかる。

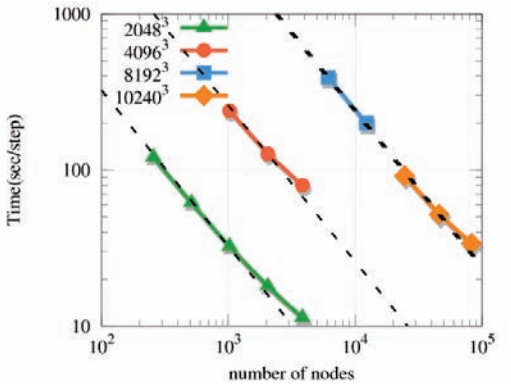

図7:「京」でのストロングスケーリング。使用したノード数に対する計算時間

図7は「京」でのストロングスケーリングを表したものである。粒子数が 2048^3～10240^3 のシミュレーション結果で、非常に良いスケーラビリティを示していることがわかる。我々のコードはスケーラビリティだけでなく実行効率も高く、たとえば2兆粒子のシミュレーションを「京」のフルシステムで実行した時の実行性能は5.8Pflops（55%の対ピーク性能）であった。

3. 「京」とFX10での性能評価

		京	FX10
ノード	CPU	SPARC64™ VIIIfx	SPARC64™ IXfx
	コア数	8	16
	クロック	2.0 GHz	1.848 GHz
	ピーク性能	128 GFlops	236.5 GFlops
	共有L2キャッシュ	6 MB	12 MB
	メモリ容量	16 GB	32 GB
	メモリ帯域	64 GB/s	85 GB/s
	インターコネクト	Tofu	
システム全体	総ノード数	82944	4800
	ピーク性能	10.62 PFlops	1.135 PFlops

表1:「京」とFX10

「京」とFX10でのコードの性能評価に移る。本研究では、「京」で最適化されたコードをそのままFX10で動かしたときに、どれくらい性能が異なるかを評価する。その前に両者の違いを表1にまとめた。FX10のCPUは「京」で用いられているものの後継であり、クロックが若干低く、1CPUあたりのコア数が16と2倍になっているのが大きな違いである。したがって「京」と比べて、スレッド並列に向かずシングルスレッドで処理せざるをえない部分による性能低下がより顕著になる。またスレッド間でのロードバランスを良好に保つプログラミング技術が要求される。さらにノードあたりのネットワーク性能が「京」とFX10では同じであり、スレッド

数の違いによる通信性能への影響も懸念材料の一つである。

そこで本研究では主に主要演算部であるPMとツリー部分の、

(i)　並列度が自明な部分の性能

(ii)　並列度が自明でない部分の性能

(iii)　隣接通信・全対通信の性能

に着目して性能の比較を行う。

3.1. シミュレーション

「京」とFX10ではノードあたりのピーク性能とメモリ帯域が異なる一方、ネットワーク性能が同じと、単純に同じシミュレーションから両者の性能を議論することは難しい。とりあえず同じ規模で同じノード数のシミュレーションで、「京」とFX10の比較を行う。ダークマター粒子数 4096^3 のシミュレーションを、4800 ノード、2048^3PM メッシュで計算したときに各パートにかかった時間を表2にまとめた。10ステップ目から数えて20ステップ分で平均してある。ノード間はMPI並列で、ノード内はOpenMPのハイブリッド並列である。

まず全体の実行性能から述べる。計算した重力相互作用数は1ステップ、1ノードあたりおよそ 3.8×10^{10} であった。これに51(ひとつの重力相互作用計算あたりに必要な演算数)をかけて、1ステップにかかった計算時間で割ると、おおよその実行性能値が計算できる。結果は

- FX10: 507Tflops、45%の対ピーク性能比
- 「京」: 314Tflops、51%の対ピーク性能比

であった。FX10は対ピーク性能比が若干落ちているものの、FX10の性能は「京」と比べ遜色の無いものと言える。

以下各パートについて詳しく述べる。

3.1.1. PM

表2にある演算1はPM部で並列性が自明な箇所を合計したものである。差分、補間などが含まれる。演算2は並列性が自明でない箇所の合計で、送信リスト作成などが含まれる。FFTは

項目(演算1は並列性が自明な部分、演算2はそれ以外)		計算時間 sec/step	
		京	FX10
PM	全対通信	0.15	0.93
	FFT	0.66	0.82
	演算1	0.57	0.30
	演算2	0.092	0.13
ツリー	隣接通信	0.56	1.01
	演算1	24.72	13.76
	演算2	1.23	0.96

表2: 「京」とFX10で、各パートにかかった時間。

両者ともに 4096^3 粒子、4800ノードでのシミュレーションにおいて、1ステップあたりに要した時間である。演算1は並列性が自明な部分、演算2は自明でない部分を合計したものである。仮に演算速度が完全にCPUのクロックに比例するならば、「京」の数値の 128/236.5=0.54倍がFX10での理想値である。

ここではFFTW3ライブラリ[10]を用いた。スレッド並列をしない、1次元の分散並列に対応したものを用いている。いったん各ノードが3次元分割された自領域を覆うローカルメッシュ上の密度場を計算した後に、1 次元分割になるように密度データを全対通信している。このシミュレーションでは、各ノードが1x2048x2048のメッシュデータを持つように通信を行う。厳密に言うと部分対部分通信であるが、極めて非一様な通信であるためMPI_Alltoallvで実装してある[11]。また今回、FFTはノードあたり1コアを使用するため、FFTの計算の際の並列度は「京」とFX10は同じで、2048ノード、2048コアである。

まず全対通信であるが、FX10 は「京」よりも大きく時間を費やしていることがわかる。FFTもFX10が少し遅い。FFTはメモリ帯域リミットであると考えられるため、シングルスレッド処理をしているFFTは、ノードあたりのメモリ帯域が大きいFX10のほうが良い性能が出そうであるが、逆の結果が得られた。全対通信の性能値とあわせて考えると、「京」のほうがネットワークまわりのライブラリのチューニングが進んでいる可能性がある。

演算1に関しては、ほぼ理想的にピーク性能に比例する結果となった。演算2はピーク性能に対して逆の結果となっている。ここは主にループ長が100程度、またはそれ以下と少ないループに対してスレッド並列を行っている。さらにスレッド毎で処理するデータ量が大きく異なるため、ロードバランスの調整が難しく、同期処理に時間がかかったためだと考えられる。

3.1.2. ツリー

次にツリー計算部について述べる。PM部の時と同様に並列性が自明な箇所は演算 1 として、自明でないところは演算2としてまとめてある。前者はある粒子群に対してツリー構造をたどり、相互作用を及ぼす粒子リストを作成するところと、作成した粒子リストから力を計算するところが含まれている。後者はツリー構造の生成である。より具体的には、まず粒子ごとにmorton key を作成し、ソートする。そしてそれに基づき構造体の配列となっているツリー構造を生成していく。各ノードに含まれる粒子全てに対する、グローバルなツリー構造が必要であること、より深い階層のツリー構造を予測することが難しいことから、スレッド並列で性能を得るにはきわめて困難な部分である。我々のコードでは、はじめにあるひとつのスレッドでルートを含む数階層分のツリー構造を生成した後に、それより深い階層のツリー生成をスレッド並列で行っている。

隣接通信が発生するのは、近傍のノードから近距離力の計算に必要なツリー情報をもらうところである。各ノードの領域形状から、そのノードに送るべきツリー構造が生成できる。我々のコードではこの通信と通信リスト作成をオーバーラップさせているが、ここはシングルスレッドで処理をしている。

隣接通信の性能であるが、シングルスレッドで処理、つまりノードあたりのネットワーク性能は同じにも関わらず、FX10の方が2倍近く遅い結果が得られた。本コードのホットスポット

---

[10] http://www.fftw.org/
「京」では富士通製の3次元分散並列FFTを使うことも可能であるが、ここでFFTW3を用いた。

[11] MPI_IsendやMPI_Irecvのような非同期通信が良いと思うかもしれないが、メッシュ形状が板状で、領域分割は3次元形状なので、ある板メッシュを担当するノードは数百～数千のノードから密度データを受け取る必要がある。したがって非同期通信は向かない。また数万ノードではこの部分がボトルネックになり得るため、通信を階層化している。詳しくは Ishiyama et al. 2012を参照。

である演算 1 はほぼピーク性能に比例した理想的な結果が得られた。そのおかげで、「京」と FX10 での実行性能はそれほど遜色の無いものとなっている。演算 2、ツリー構築はやはり原理的な難しさからか、FX10 では 1.3 倍程度の加速であった。

4. まとめ

「京」でチューニングされた超並列重力多体問題シミュレーションコードをそのまま FX10 上で動かした時に、どれくらい性能が異なるかを調べた。ダークマター粒子数 4096^3 の宇宙論的 N 体シミュレーションを 4800 ノードで行った場合、「京」では対ピーク性能 51%の実行効率が得られたが、FX10 でも 45%とそれほど遜色のない性能が得られた。これは「京」上で実施してきた最適化が FX10 上でも有効であることを意味している。つまり FX10 は「京」の後継として優れたシステムであり、プログラムが相互に容易に移植できる利用性の高い環境であるといえる。

その一方、パート毎に見ると、両者で優位な性能差を示す部分が存在することがわかった。まず通信性能であるが、ノードあたりのネットワーク性能が同じであるにも関わらず、全対通信、隣接通信ともに「京」が優位に速いことがわかった。「京」のほうがネットワークまわりのライブラリのチューニングが進んでいる可能性がある。

また並列性が自明な部分は、スレッド並列数が 8(「京」) → 16(FX10)になっても、ほぼ理想的にピーク性能に比例するような性能が得られた。一方、自明でない、または難しい部分は、予想通り並列度が増えた恩恵をさほど受けられないばかりか、性能が逆転するケースもあった。今回は全体の性能にさほど悪影響を与えなかったが、よりノードあたりのコア数やピーク性能が増大した場合は問題になる可能性がある。

今回は時間の都合により表面的な比較しかできなかったが、機会があれば FX10 でノードあたりのスレッド数を変化させた時の性能や、並列性が自明でないところについて、「京」と FX10 でスレッド数を変えた時の振舞いなどを調べていきたいと考えている。

参 考 文 献

(1) J. Barnes and P. Hut, “A hierarchical O(N log N) force-calculation algorithm,” Nature, vol. 324, pp. 446-449, Dec. 1986.

(2) J. Dubinski, J. Kim, C. Park, and R. Humble, “GOTPM: a parallel hybrid particle-mesh treecode,” New Astronomy, vol. 9, pp. 111-126, Feb. 2004.

(3) T. Hamada, T. Narumi, R. Yokota, K. Yasuoka, K. Nitadori, and M. Taiji, “42 Tflops hierarchical n-body simulations on gpus with applications in both astrophysics and turbulence,” in Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, ser. SC ’09. New York, NY, USA: ACM, 2009

(4) R. W. Hockney and J. W. Eastwood, Computer Simulation Using Particles (New York: McGraw-Hill), J. W. Hockney, R. W. & Eastwood, Ed., 1981

(5) T. Ishiyama, K. Nitadori, J. Makino, “4.45 Pflops Astrophysical N-Body Simulation on K computer - The Gravitational Trillion-Body Problem”, Proceedings of the 2012 ACM/IEEE conference on Supercomputing (SC ’12), Saltlake, Utah, USA, Nov 2012 (ACM Gordon Bell Prize Winner)

(6) T. Ishiyama, T. Fukushige, J. Makino, GreeM：Massively Parallel TreePM Code for Large Cosmological N-body Simulations, 2009, PASJ, 61, 1319
(7) J. Makino, “A Fast Parallel Treecode with GRAPE,” Publ. of the Astron. Society of Japan, vol. 56, pp. 521-531, Jun. 2004
(8) K. Nitadori, J. Makino, and P. Hut, “Performance tuning of N-body codes on modern microprocessors: I. Direct integration with a hermite scheme on x86 64 architecture,” New Astronomy, vol. 12, pp. 169-181, Dec. 2006.
(9) A. Tanikawa, K. Yoshikawa, T. Okamoto, and K. Nitadori, “N-body simulation for self-gravitating collisional systems with a new SIMD instruction set extension to the x86 architecture, Advanced Vector eXtensions,” New Astronomy, vol. 17, pp. 82-92, Feb. 2012.
(10) A. Tanikawa, K. Yoshikawa, K. Nitadori, and T. Okamoto, “Phantom-GRAPE: numerical software library to accelerate collisionless N-body simulation with SIMD instruction set on x86 architecture,” New Astronomy, vol. 19, pp.74-88, Feb. 2013
(11) V. Springel, “The cosmological simulation code GADGET-2,” Mon. Not.R. Astron. Soc., vol. 364, pp. 1105-1134, Dec. 2005.
(12) D. Sugimoto, Y. Chikada, J. Makino, T. Ito, T. Ebisuzaki, and M. Umemura, “A special-purpose computer for gravitational many-body problems,” Nature, vol. 345, pp. 33-35, May 1990.
(13) G. Xu, “A New Parallel N-Body Gravity Solver: TPM,” Astrophys. J.Supp., vol. 98, pp. 355-+, May 1995