

各種アプリケーションにおける GPGPU対Many Core Processorの性能比較

高橋 航平^{1,†1,a)} 塙 敏博^{2,4,b)} 朴 泰祐^{2,4,c)} 児玉 祐悦^{2,4,d)} 扇谷 豪^{3,5,e)} 佐藤 三久^{2,4,f)}

概要: 演算加速装置として、これまで広く使われてきた GPGPU に加えて、Many Core Processor が利用可能になってきている。本稿では、GPGPU として NVIDIA Tesla M2050, K20 と Many Core Processor として Intel MIC アーキテクチャ Xeon Phi を用いて、計算の割り当てが静的な姫野ベンチマークと、動的に変化するツリー法による N 体シミュレーションを用いて性能評価を行った。その結果、姫野ベンチマークにおいて、K20 では 60.5 GFlops だったのに対し、Xeon Phi では 43.6 GFlops という性能にとどまった。一方、N 体シミュレーションにおいては、16Mi 粒子のときに K20 が 4.0 秒に対して MIC では 3.1 秒と約 30%高い性能が得られた。理論ピーク性能に近い K20, MIC の両プロセッサにおいて、実効性能がアプリケーションの特性（特にダイナミズム）に応じて変化することが確認された。

1. はじめに

近年、演算加速装置（アクセラレータ）の高い演算性能とメモリバンド幅に着目し、様々な HPC アプリケーションへの利用が進められている。加えて、優れた電力性能比に着目し、大規模 HPC システムにも演算加速装置が多く用いられるようになってきている。

これまで、演算加速装置として GPU (Graphics Processing Unit) を用いる、GPGPU (General Purpose computing on GPU) が盛んに行われてきた。

GPU は限られた機能を持った小型のプロセッサコアを高密度に集積させ、高い並列性とコストパフォーマンス、電力性能比を実現している。一方、アーキテクチャ上の制約から、アプリケーションによってはその性能を十分に発揮することができない場合もある。各 GPU ベンダは専用のプラットフォームを提供したり、OpenCL や OpenACC 等の共通フレームワークに対応することで GPGPU への対応を行なっているが、いずれの場合においても専用の言語

を用いてコードを記述する必要がある。

一方 Intel 社は、汎用 CPU で用いられてきた IA-32 アーキテクチャをベースに多数のプロセッサコアを集積した Intel MIC (Many Integrated Core) アーキテクチャ（以下 MIC と略す）、および MIC を採用した製品である Xeon Phi コプロセッサ [1] を発表した。MIC は IA-32 をベースとしているため、既存の CPU 向けコードに若干の修正を加えることで実行が可能になっている。GPU が比較的単純なコアを多数用意し電力性能比を高めることで性能向上を行なってきたのに対し、MIC は汎用マルチコア CPU の特性を保ったまま性能向上を図っている。

本稿では、性質の異なるアプリケーションコードを用いて GPGPU と MIC という異なるアーキテクチャの演算加速装置の性能比較を行う。アプリケーションには、各反復毎にメモリアクセスと計算のパターンに変化のない、静的なアプリケーションとして姫野ベンチマークを用いる。一方、メモリアクセスパターンや計算が動的に変化するアプリケーションとして、ツリー法を用いた N 体シミュレーションを用いる。

2. Intel MIC アーキテクチャ

2.1 概要

MIC は P5 マイクロアーキテクチャを基に開発された、IA-32 in-order プロセッサであり、汎用 x86 CPU に搭載されている、SSE (Streaming SIMD Extensions) 命令や AVX (Advanced Vector eXtensions) 命令とは異なる、専用の 512bit 長 SIMD (Single Instruction Multiple Data)

¹ 筑波大学情報学群情報科学類
² 筑波大学システム情報系
³ 筑波大学大学院システム情報工学研究科
⁴ 筑波大学計算科学研究センター
⁵ 筑波大学大学院数理物質科学研究科
^{†1} 現在、株式会社 Aiming
^{a)} kohei@hpcs.cs.tsukuba.ac.jp
^{b)} hanawa@ccs.tsukuba.ac.jp
^{c)} taisuke@cs.tsukuba.ac.jp
^{d)} kodama@cs.tsukuba.ac.jp
^{e)} ogiya@ccs.tsukuba.ac.jp
^{f)} msato@cs.tsukuba.ac.jp

表 1 ある Workload に於ける CPI の例 [4]

HW スレッド数	スレッド当たりの CPI	コア当たりの CPI
1	5.24	5.24
2	8.80	4.40
3	11.18	3.73
4	13.74	3.43

命令を備える [2]. 搭載しているコア数は、製品である Intel Xeon Phi 5110P[3] では 60 基にも達し、各コアは最大で 4 ハードウェアスレッドを動作させることができる。ただし各コアの逐次実行性能は低く、OpenMP や CilkPlus といった並列処理フレームワークとの組み合わせや、SIMD 命令の積極的な活用が前提となる。

このハードウェアスレッドは各コア内で、ラウンドロビンによって順にスケジューリングされており、実行するハードウェアスレッドを増やせばそれに応じて CPI (Clock per Instruction) が増加していく。Intel Labs による、ある Workload における CPI を表 1 に示す [4].

MIC が dual-issue をサポートしていることもあり、スレッド数を増やす毎にコア当たりの CPI は減少している。一方、コアを跨ぐキャッシュへのアクセスやメモリへのアクセスが多いアプリケーションの場合には CPI が大きくなってしまいうため、プロファイリング等を用い、最適な実行スレッド数を決定する必要がある。

MIC を利用する上での特徴として、既存の CPU コードの構造を大きく変化させることなく MIC に適用することができる点が挙げられる。若干の制限があるものの、ソースコード中に指示文を挿入するだけで対応可能となっている。SIMD 命令の利用についても、コンパイラの自動ベクトル化や適切な attribute の利用で対応できるため、データの配置方法を工夫することでプロセッサを効率的に利用することができる。

MIC が持つ専用の 512bit SIMD 命令が、従来の汎用 x86 プロセッサに搭載されていた SSE 命令や AVX 命令と大きく異なる点として、AOS (Array of Structure) への対応が挙げられる。一般に、シミュレーション等で座標や速度といったデータを扱う際には、1つの構造体に 1 要素分のデータをまとめ、その配列としてメモリ上に配置することが多い。これはプログラミングを行う上で、管理しやすくなる利点がある。しかし、要素間の相互作用等の演算に SIMD 命令を使用し、並列に複数要素を演算するには、一度各要素の各データが連続になるように展開する必要がある。SIMD 命令を適用するのが困難であったり性能に影響が出ることになる。もちろん、SIMD 命令が効率的に実行できるように、最初から構造体を使用しないで各データ毎に配列として持つ (SOA, Structure of Array) こともできるが、データの管理に手間がかかる。MIC では命令レベルで AOS への対応が行われており、ストライド幅に応じて

メモリアクセス数が増加するものの、データの並べ替えや一時領域の確保が不要なため、AOS 対応時のオーバーヘッドを低減することができる。

2.2 プログラミングモデル

MIC には、複数のプログラミングモデルが提供されており、ユーザは自由に使い分けることができる。

Native モデル MIC は PCI Express を介してホストに接続されているが、MIC 上では、micro-OS と呼ぶ専用の Linux が動作しており、単体のノードとして扱うことができる。ユーザは、このノードにログインして直接 MIC 用にコンパイルされたアプリケーションを実行することができる。これを Native モデルと呼ぶ。MIC ノードはデバイスを持たないため、疑似デバイスドライバにより PCI Express 経由でホストにファイル入出力を行ったり、ホスト上のデバイスを介して他ノードと通信を行う。OpenMP や MPI で書かれたアプリケーションをそのまま MIC 用にコンパイルし実行することができる。

Offload モデル GPGPU と同様に、アプリケーション全体はホスト上で実行し、その一部を MIC 上にオフローディングして実行する。これを Offload モデルと呼ぶ。Offload モデルの場合は、OpenMP などで並列実行部分を記述し、さらにオフローディングを指示するための専用の指示文が必要になる。さらに、ホストと MIC とでは異なるメモリ空間を持つため、データ転送にも考慮する必要がある。明示的に変数のデータ転送を指定する必要がある Explicit offload モデルと、暗黙に行われる Implicit offload モデルがある。但し、Implicit offload モデルでは、専用の API を使う必要があり、ソースコードの変更量は大きくなる。

本稿では、GPGPU との比較を行うため、以降は Explicit offload モデルのみを対象とする。

3. NVIDIA Tesla 対 Intel MIC の性能比較

本稿では、GPGPU と Many Core Processor の比較に NVIDIA 社の Tesla GPU と Intel 社の MIC を用いる。実際に用いた評価環境として、NVIDIA Tesla GPU 環境を表 2 に、Intel MIC 環境を表 3 に示す。

Tesla 環境では、Fermi アーキテクチャ 2 種、Kepler アーキテクチャ 1 種の合計 3 種を選択し、プログラミング環境には CUDA5.0 を用いた。

一方、MIC 環境は Intel 社より貸与されたものであり、製品版の Xeon Phi とは一部異なる箇所がある。前述の通り、プログラミング環境には Offload モデルを用いた。また、今回使用した MIC は物理コアを 61 コア搭載しているが、Offload モデルにおいて実際に計算に利用できるのは 60 コアであり、実行可能スレッド数は最大で 240 スレッド

表 2 GPGPU の評価環境

アーキテクチャ	Fermi	Kepler
種類	M2050	K20
コア数 (SM × コア)*1	14×32	13×192
実行単位	32 (= 1 Warp)	
L2 キャッシュ	768KB	1280KB
クロック周波数 (コア)	1.15GHz	706MHz
周辺	575MHz	706GHz
ピーク性能 (単精度)	1.03TF	3.52TF
ピーク性能 (倍精度)	515GF	1175GF
メモリ	GDDR5	
メモリ容量	3GB	5GB
メモリバンド幅	148GB/s	208GB/s
コンパイラ	CUDA 5.0	

表 3 Intel MIC の評価環境

コア数	61 (Offload では 60 コア使用可能)
HW スレッド数	4
実行単位	1
L2 キャッシュ	512KB × コア数
クロック周波数	1.1GHz
メモリ	GDDR5
メモリ容量	8GB
メモリバンド幅	352GB/s
ピーク性能 (単精度)	2.15TFlops
ピーク性能 (倍精度)	1074GFlops
コンパイラ	Intel Composer XE Ver. 13.0.1.117
MPSS*2	2.1.4346-16

となる。

4. 姫野ベンチマーク

姫野ベンチマークは非圧縮流体解析コードの性能評価のために考案されたベンチマークコードで、ポアソン方程式をヤコビの反復法を用いて解くものである [5]。

姫野ベンチマークは 3 次元空間に対しステンシル計算を行うため、各ステップの演算量が静的に決定される。しかし 1 ステップの間に複数の配列を参照する上、各配列の中でも i, j 方向に比較的大きな空間を跨いでアクセスするため、キャッシュミスが発生しやすいと考えられる。

問題サイズには LARGE (512 × 256 × 256) を使用した。また姫野ベンチマークでは単精度が用いられている。MIC では、OpenMP を用いて並列化を行い、スレッドの割り当て方法として、各物理コアにラウンドロビンで割り当てを行う、scatter を用いた。GPGPU では、ヤコビ法の反復内部を CUDA カーネルとして記述し、グリッドサイズ = (512, 256, 0)、スレッドサイズ 256 として割り当てている。

姫野ベンチマークでの測定結果を図 1 に示す。横軸

*1 Fermi アーキテクチャでは SM (Streaming Multiprocessor), Kepler アーキテクチャでは SMX (SM eXtreme) が構成単位となっている。SMX を SM と記述する場合もある。

*2 Intel MIC Platform Software Stack

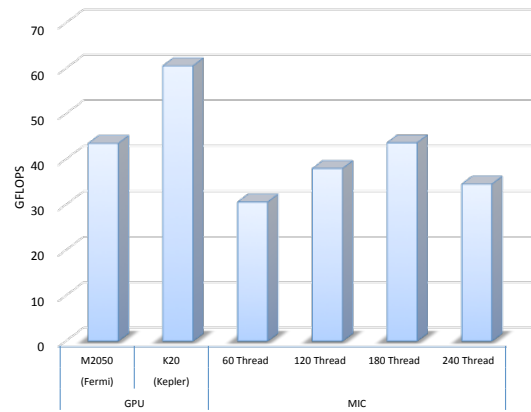


図 1 GPGPU および MIC を用いた姫野ベンチマークの結果

には用いたアクセラレータを、縦軸には得られた結果を GFLOPS で示した。

GPGPU においては、M2050 では 43.5 GFLOPS, K20 では 60.5 GFLOPS となったが、MIC は最大で 43.6 GFLOPS にとどまった。

GPU の様に複数のコアが SIMD 的に動作するアーキテクチャでは、メモリアクセスの集約 (coalesced access) が可能であるため、チップ外への実際のメモリアクセスの回数を減らすことができる。更に、アルゴリズム的に条件分岐が発生しないので、コアの負荷が均一となり高い性能が得られていると考えられる。一方で MIC は CPU アーキテクチャに近く、各コアが独立した命令を実行できる上、レイテンシが不均一なキャッシュや dual-issue といった要因で、キャッシュミス等が発生しメモリアクセス量が増える傾向にあるためだと考えられる。姫野ベンチマークの場合、1 ステップの演算で参照する 3 次元配列が多くある上、各配列内ではメモリ領域を大きく跨ぐアクセスが行われるため、キャッシュが有効に使われずに性能が低下したのではないかと推測できる。

また、MIC で 240 スレッド (各コア 4 スレッド) で実行した場合より、180 スレッド (各コア 3 スレッド) で実行した場合の方が性能が良くなった。これは、各コアに 3 スレッド割り当てた時点でメモリ帯域を使い切っており、ロードストア待ちをするスレッドが増えてしまい、性能が低下したためだと考えられる。

ただし、今回いずれの場合においても計算カーネルにほとんど最適化は行っていない。また、今回用いた GPU より前の世代の GPU を用いて、最適化を行った結果 60 GFLOPS を超える性能が得られたという報告もある [6]。このように、各スレッド及びコアが規則的に SIMD の様な動作をし、かつ大きな並列性が存在する姫野ベンチマークのような問題では、GPU の様なアーキテクチャが有利であると考えられる。

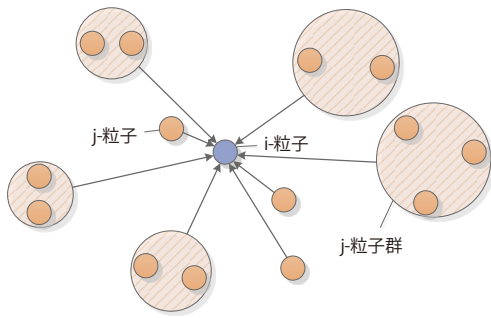


図2 ツリー法の概念図。計算対象となる i -粒子と作用を及ぼす j -粒子、また i -粒子から十分に遠く、1粒子として計算を行う j -粒子群

5. ツリー法によるN体シミュレーション

本節では、N体シミュレーションを用いてMICの性能を評価する。本稿で扱うN体シミュレーションはツリー法[7]と呼ばれ、全ての粒子の組み合わせを計算する直接法とは異なる。ツリー法は宇宙物理の分野でも銀河の形成や進化といった、計算精度よりも粒子数を重要とする分野で利用されるアルゴリズムである。この評価では、扇谷らによりGPU向けに最適化されたツリー法コード[8], [9]を元としている。直接法は全ての組み合わせを計算するため計算精度は良いが、粒子数 n としたとき、粒子数が増えると計算量が $O(n^2)$ に従って爆発的に増加し、現実的な時間での計算が難しくなる。一方ツリー法では、現在の計算対象となっている粒子 (i -粒子) と、 i -粒子に重力作用を及ぼす粒子 (j -粒子) のうち、十分に離れた j -粒子群を、1つの重い粒子として計算することで計算量を $O(n \cdot \log(n))$ に削減している。図2にツリー法における i -粒子と j -粒子の関係を示す。従って、計算の内容は i -粒子毎に動的に変化する。また、粒子の軌道進化によりその分布が変動する影響に応じて計算内容は変化する。

扇谷らのコードは、ベクトル化とグループ化という手法によって高速化が行われており、どちらも複数の i -粒子をまとめて処理することで Warp divergence を減らすために用いられている。CUDAでのグループ化はWarp内ではスレッドが同時に実行されることを利用し、隣り合ったスレッドでデータの交換を行なっているが、MICで提供されている並列フレームワークでは、オーバーヘッド無しにこれを実現するのは難しく、MIC上ではグループ化は行わなかった。

5.1 ベクトル化

CUDAでの実装では、Warp divergence を減らす目的に加えて、キャッシュヒット率を向上させるためにベクトル化を行っていた。一方で、MICでは、各コアがMIMD

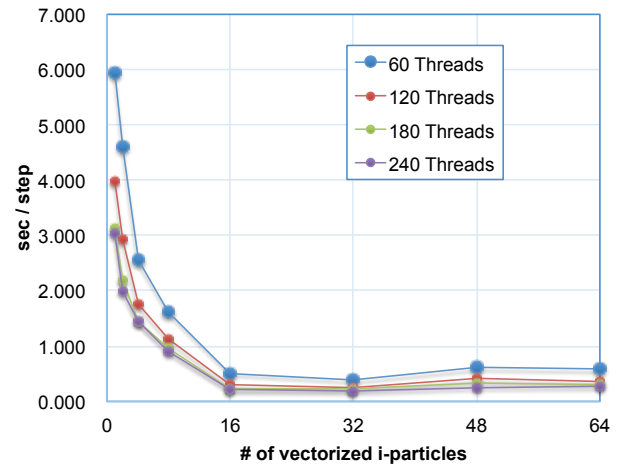


図3 ベクトル化での i -粒子数を变化させた時の性能の推移

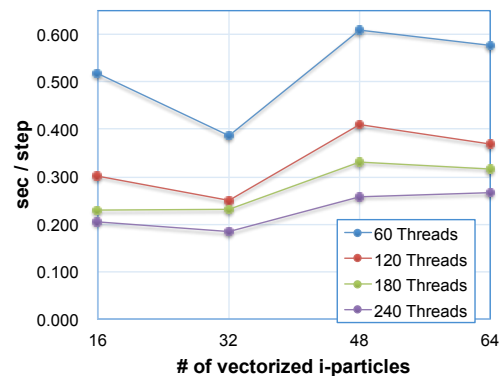


図4 図3の16粒子～64粒子までを拡大

で動作するため、Warp divergence のようなことを考慮する必要はないが、512bit SIMD 命令を効率的に利用する目的でベクトル化を行う。512bit SIMD 命令では、単精度浮動小数点数ならば16要素をまとめて計算することができる。SIMD 命令はintrinsic関数によって明示的に配置することも、コンパイラに判断させSIMD 命令を生成させる自動ベクトル化を利用することもできる。本稿ではコンパイラによる自動ベクトル化が有効に働くように、ローカルデータの配置を工夫することで性能向上を図った。

ベクトル化ではツリーをより深く走査するかの判定を個別に行わず、現在参照している j -粒子群と最も近い i -粒子を用いて判定する。これによりベクトル化されている i -粒子群はより j -粒子に近いと判定されやすくなり、結果的に削減されるはずの計算量が元に戻ってしまう可能性がある。そのため、ベクトル化にはSIMD 命令の利用率と計算量との間でトレードオフが存在する。

問題サイズを16Mi粒子に固定し、最も効率的にベクトル化できる i -粒子数を調べた。図3及び図4に、ベクトル化する粒子数を变化させた際のステップあたりの経過時間を示す。

各コアのハードウェアスレッド数が何れの場合であって

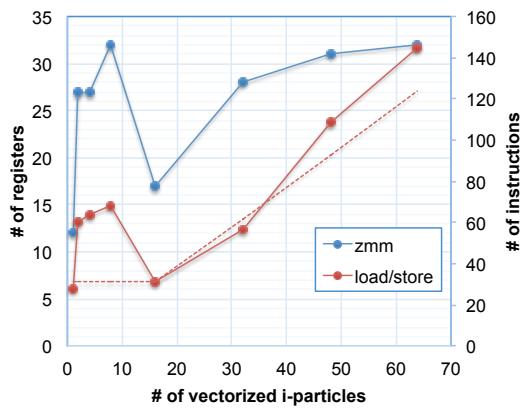


図 5 ベクトル化での i 粒子数を変化させた時の、zmm レジスタ使用数とロードストアに関与している命令数の推移。赤点線は 16 粒子時を基準とした時の期待されるロードストアの数

も、32 粒子でベクトル化した場合に最も良い性能が得られた。特に、ハードウェアスレッド数を全て使用した場合が、1 ステップあたり 0.18 秒と最も性能が高かった。

そこでコンパイラが生成したアセンブラについて、使用している zmm レジスタ数とロードストアに関与している命令数を調べた。なお、zmm レジスタは 512bit 長を持つレジスタで、512bit SIMD 演算の際に用いる。MIC ではコア当たり 32 本が利用可能である。図 5 に結果を示す。この結果から、16 粒子未満の場合には、レジスタをより多く消費する上、ロードストアも多く発行していることがわかる。一方 16 粒子の場合では使用レジスタ数 17 本、ロードストア 31 回と低く抑えられており、最も効率が高いことがわかる。

最も性能が良かった 32 粒子の場合には、単純に計算してロードストアが 16 粒子時の 2 倍発行されると予想される(図 5 における赤点線)はずだが、57 回と約 1.8 倍に抑えられている。同様に 48 粒子、64 粒子について 16 粒子との比較を行うと、こちらはそれぞれ 109 回(約 3.5 倍)、145 回(約 4.7 倍)と単純に予想されるロードストア数を超える。これは 16 粒子の場合には一部の zmm レジスタの要素が全て埋まっていなかった状態であり、32 粒子の場合に zmm レジスタが効率よく割り当てられたと考えられる。その一方で、48 粒子と 64 粒子時には溢れてしまい、レジスタの値を入れ替えるためのロードストアが発生したという事が考えられる。

先にも述べた通り、32 粒子時は 16 粒子時よりも計算量が増えてしまい性能が悪くなることも予想されるが、実際には zmm レジスタを埋めて SIMD 命令の利用率を上げたり、ロードストア数を減らす方が MIC で良い性能を得るために重要であることがわかる。

5.2 OpenMP スレッドのスタックサイズ調節

次に OpenMP の各スレッドが確保するスタックフレー

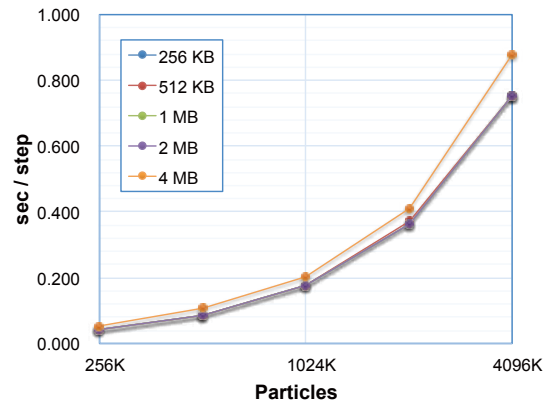


図 6 OpenMP スレッドのスタックフレームサイズを変更した場合の性能の推移

ムサイズを調節した。MIC での OpenMP は、デフォルトで各スレッドに対して 4 MB ずつスタックフレームを確保する。つまり 240 スレッド起動しハードウェアスレッドを完全に埋めた場合では、スタックフレームだけで 960 MB 消費されることになる。結果として非常に大きな空間をスキャンすることとなり、性能を低下させる原因となることが考えられる為、スタックフレームを削減し最適なサイズを調査した。

図 6 に結果を示す。スタックフレームサイズは 2 MB に変更すると性能が向上したが、それ以下のサイズでは特に変化は見られなかった。MIC ではメモリ管理に 4 KB と 2 MB のページを用いており、OpenMP ランタイムは常にスタックフレームには 2 MB のページを割り当てている可能性がある。そのためスタックフレームを 2 MB 以下に抑えても TLB 上では 2 MB 単位での扱いとなり、性能が変化しなかったのではないかと予想される。

5.3 GPU との比較・評価

以上を踏まえて MIC 向けに最適化したツリー法のコードを実行し、GPU での結果と比較・評価を行った。

図 7 に Tesla M2050, Tesla K20 及び MIC における重力計算部の計算時間の推移及び、それぞれの GPU と比較した MIC の性能比を示す。M2050 では他と比べて搭載メモリ容量が少ないため、8Mi 粒子までしか実行できなかった。

この結果から、MIC について、M2050 に比べて 80~90%、K20 に比べて 30~40% 高い性能が得られていることがわかる。また、粒子数を増加させても性能比はほとんど変化しないことから、キャッシュミス等によるオーバーヘッドよりも、コアを有効に使い切れているかどうかの方が性能に表れていると考えられる。このことから、ツリー法のように、メモリアクセスパターンや計算の内容が動的に変化する演算においては、MIC は良好な性能を得ることができると考えられる。

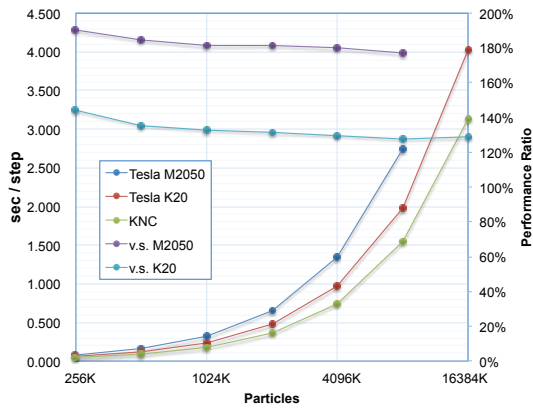


図 7 M2050, K20 および MIC における重力計算部のステップあたりの計算時間と、各 GPU との性能比。紫の線が Tesla M2050 との性能比を示し、水色の線が Tesla K20 との性能比を示す

6. 考察

表 1 におけるハードウェアスレッド数と CPI の関係から、起動するハードウェアスレッドが多くなることでコアを効率良く利用できると考えられた。実際に、ツリー法を用いた N 体シミュレーションの場合にはハードウェアスレッドを全て埋めた場合が最良の結果となった。しかし、姫野ベンチマークのようなメモリアクセスの割合が多く、メモリバンド幅を要求するコードにおいては、ハードウェアスレッド数を減らすことで性能が良くなるという結果であった。これについては、汎用マルチコアプロセッサにおいても、メモリバンド幅を要求するアプリケーションでは同様の結果を示すことが知られている [10]。

また N 体シミュレーションでの性能評価では、SIMD レジスタの利用率とロードストア数に関して重要な知見が得られた。MIC の 512bit SIMD で用いる zmm レジスタは SandyBridge プロセッサなどに搭載されている AVX における SIMD レジスタの倍の長さであり、レジスタの本数も倍となっている。従って、より多くのデータを SIMD 命令で処理出来るようになっているが、適切なデータ量になるように調整する必要がある。扱うデータ量を増やすぎると、過剰にロードストアが発生し、性能を落とす原因となることがわかった。

7. まとめ

まず、規則的に演算やメモリアクセスを行う問題として、姫野ベンチマークを用いて性能評価を行った。MIC は 43.6 GFLOPS であったのに対し、K20 では 60.5 GFLOPS の性能が得られた。このような問題においては、GPU のような複数のコアが SIMD 的に動作するアーキテクチャの方が有利であるという結果となった。

次に、メモリアクセスパターン等が動的に変化する問題として、ツリー法による N 体シミュレーションを用いて

性能評価を行った。N 体シミュレーションでは、MIC は 1 ステップ辺り 3.1 秒となり、K20 に対し最大で 40% 高い性能が得られた。これにより、動的にカーネルの負荷が変化するコードでは、MIC の方が有利であるという結果となった。

以上のことから、アプリケーションの特性によってプロセッサの性能は変化するため、高速化を行うアプリケーションの特性を調査し、最適なアクセラレータを検討する必要がある。現在 GPU では記述することが困難なアプリケーションであっても、MIC のハードウェア特性を理解し、コンパイラのサポートを活用することで、指示文を用いることで比較的 low コストに高速化が得られると期待される。

謝辞 本研究を行う上で、筑波大学計算科学研究センター 中尾 昌広 研究員、並びにインテル株式会社 池井 満氏には、MIC 検証環境のメンテナンスや MIC に関する有益な情報をご提供頂きました。この場をお借りして御礼を申し上げます。

参考文献

- [1] Intel®Xeon Phi™. 入手先 <http://www.intel.com/content/www/us/en/processors/xeon/xeon-phi-detail.html>
- [2] George Chrysos, "Knights Corner, Intel's first Many Integrated Core (MIC) Architecture Product," A Symposium on High Performance Chips (Hot Chips 24), Aug. 2012. 入手先 http://www.hotchips.org/wp-content/uploads/hc_archives/hc24/Hc24-3-ManyCore/Hc24.28.335-XeonPhi-Chrysos-Intel.pdf
- [3] ARK — Intel®Xeon Phi™Coprorocessor 5110P. 入手先 <http://ark.intel.com/products/71992/Intel-Xeon-Phi-Coprorocessor-5110P-8GB-1.053-GHz-60-core>
- [4] Optimization and Performance Tuning for Intel®Xeon Phi™Coproprocessors, Part 2: Understanding and Using Hardware Even. 入手先 <http://software.intel.com/articles/optimization-and-performance-tuning-for-intel-xeon-phi-coproprocessors-part-2-understanding>
- [5] 姫野ベンチマーク/理化学研究所 情報基盤センター. 入手先 <http://accr.riken.jp/2145.htm>
- [6] 成瀬 彰, 住元 真司, 久門 耕一. GPGPU 上での流体アプリケーションの高速化手法 ~1GPU で姫野ベンチマーク 60 GFLOPS 超~. Vol. 2008-HPC-117, No. 99, pp. 49-54, 2008 年 10 月.
- [7] J. Barnes and P. Hut. A hierarchical $O(N \log N)$ force-calculation algorithm. Nature 324(4), pp. 446-449, Dec. 1986.
- [8] 扇谷 豪, 三木 洋平, 朴 泰祐, 森 正夫, 中里 直人. 重力多体系用 Tree Code の並列 GPU 化. 情報処理学会研究報告 (ハイパフォーマンスコンピューティング), Vol. 2012-HPC-135, No. 14, pp. 1-9, 2012 年 8 月.
- [9] 扇谷 豪, 三木 洋平, 朴 泰祐, 森 正夫, 中里 直人. 重力多体系用 Tree Code の並列 GPU 化による計算加速. ハイパフォーマンスコンピューティングと計算科学シンポジウム論文集, pp. 146-155, 2013 年 1 月.
- [10] 三木 康次, 原田 知徳, 朴 泰祐, 堀 敏博, 三浦 信一. メモリバンド幅に着目したマルチコアノード上のアプリケーション

ン最適化. 情報処理学会研究報告, 2010-HPC-124(4), pp.
1-7, 2010 年 2 月.