Vol.2015-HPC-148 No.19
2015/3/2

実時間実空間密度汎関数理論による電子動力学シミュレーションのXeon Phiクラスタ向け最適化

廣川 祐太[1,a)]　朴 泰祐[1,3]　佐藤 駿丞[2]　矢花 一浩[2,3]

概要：近年，Intel Xeon Phi プロセッサを搭載した PC クラスタが注目されているが，同プロセッサを多数用いた並列実応用問題の事例はあまり多くない．本研究では，RTRSDFT (実時間実空間密度汎関数理論) に基づく電子動力学シミュレータ ARTED (Ab-initio Real-Time Electron Dynamics simulator) を Xeon Phi クラスタへポーティングし，最適化と性能評価を行った．まず，ステンシル計算カーネルに対しベクトル化や最適化オプションの付与，OpenMP の実行環境などの最適化を行い，Intel E5-2670v2 (Ivy-Bridge，10 コア) と同等の性能を得られた．時間発展部全体では，CPU と Xeon Phi 間で協調計算を行う Symmetric 実行により，CPU のみを利用する場合に比べて 32 ノード時に約 1.68 倍の性能が得られた．プロセス数で比較すると，Symmetric 実行は CPU のみの実行に対して約 90%の相対性能が得られている．この結果は，従来 CPU のみの実行で 32 ノード必要であった計算が，多少の性能低下はあるものの Symmetric 実行によって 16 ノードで済むようになると考えることができ，計算機の全リソースを有効活用できるかという視点で重要な評価である．

1. はじめに

Intel Xeon Phi (以下，Xeon Phi) は MIC (Many Integrated Cores) アーキテクチャに基づくアクセラレータで，現在のプロダクトは KNC (Knights Corner) と呼ばれる．Xeon Phi は GPU (Graphics Processing Units) と同様に PCI-Express で接続されるが，Xeon Phi 上ではマイクロ Linux カーネルが動作しており，従来開発してきたアプリケーションをソースコードの変更無く Xeon Phi 上で実行することが可能である．しかしながら，Xeon Phi は従来の CPU とはその特性が大きく異なるため単純なプログラムの移植だけでは実効性能を上げることができず，その性能チューニングは難しいとされている．

Xeon Phi では，3 種類の実行または使用方法が用意されている．まず 1 つ目は，GPU と同様の使い方ができる Offload 実行，2 つ目は CPU で開発したプログラムを Xeon Phi のみの上で動作させる Native 実行，3 つ目は CPU と Xeon Phi の両者で MPI (Message Passing Interface) を用いて協調計算を行う Symmetric 実行である．Offload 実行は GPU にホスト CPU からのデータ移動が必要で，Native 実行は従来開発したアプリケーションをそのまま実行できるものの，並列性の高いアルゴリズムでなければ性能を引き出すのが難しい．Symmetric 実行は Native 実行の問題に加え，CPU と Xeon Phi というヘテロジニアス環境での実行となり処理のロードバランスが大きな問題となる．しかしながら，Symmetric 実行では計算機に搭載されている全てのリソースを使い切ることが理論的には可能である．来年以降にリリースが予定されている次世代 Xeon Phi プロダクトである KNL (Knights Landing) では現在の PCI-Express で接続するカードタイプに加え CPU と同様なソケットタイプの製品も予定されており，KNL を見据えた性能評価としては，Native 実行と Symmetric 実行での性能評価は重要であると考えられる．従って，本研究では Native 実行と Symmetric 実行に焦点を当て，実アプリケーションの実装と評価を行う．

これまでに，Xeon Phi に対してはステンシル計算や疎行列ベクトル積等の主要カーネルの性能評価や最適化，ベンチマークによる評価が行われてきた [1], [2], [3] が，実アプリケーションへの適用事例，さらに大規模並列環境での実装例は未だ少ない．実アプリケーションを Xeon Phi クラスタにポーティングし，CPU に加えて Xeon Phi を用いることが可能か，また Xeon Phi のみでどれだけの性能が出せるのかは同システムを有効利用する上で重要な課題である．そこで，本研究では RTRSDFT (実時間実空間密度汎関数理論) に基づく電子動力学シミュレータである

[1] 筑波大学大学院 システム情報工学研究科
[2] 筑波大学大学院 数理物質科学研究科
[3] 筑波大学 計算科学研究センター
[a)] hirokawa@hpcs.cs.tsukuba.ac.jp

ⓒ 2015 Information Processing Society of Japan

ARTED を Xeon Phi クラスタへポーティングし，最適化を行うことで，実アプリケーションを用いた Xeon Phi クラスタの性能評価を行う．

2. 電子動力学シミュレータ ARTED

ARTED (Ab-initio Real-Time Electron Dynamics simulator) は，筑波大学計算科学研究センターで開発中の RTRSDFT (Real-Time Real-Space Density Functional Theory, 実時間実空間密度汎関数理論) に基づく電子動力学のマルチスケールシミュレータである [4]．RTRSDFT は RSDFT (Real-Space Density Functional Theory, 実空間密度汎関数理論) が基底状態，電子が動かない状態をシミュレーションするのに対して，外部から力が加えられることにより電子が時間とともに変化する様子をシミュレーションする．そのため，RTRSDFT は RSDFT を用いて基底状態を求めた後，時間変化を求めるために時間ステップで計算を進める．その際の場の計算は一般的なステンシル計算である．RSDFT はナノ構造を対象とし 10 万原子以上の大規模な系が計算対象となっているが [5]，ARTED は結晶中の光電子ダイナミクスを計算対象とし，10 から 100 程度の比較的少数の原子を含む単位セルの計算を，非常に多くの個数，並列して行うことが必要になる．従って，ARTED で行われる基底状態の計算は非常に小さなものとなる．電子の波動関数の時間発展はテイラー展開法が用いられ，波動関数にハミルトニアンを演算する際にステンシル計算を行う．時間発展部の計算は 1 万から 10 万ステップ程度の大きさとなるため，ARTED では時間発展での計算が支配的となり，ステンシル計算の最適化が重要であると考えられる．

近年の主な成果では，ARTED を利用したシミュレーションによりアト秒光科学実験で見られるシリコンのバンドギャップの超高速変化の特徴を再現し，電子の励起プロセスが量子トンネル効果により起きていることが明らかにしたものがある [6]．また，ARTED は京コンピュータにおける性能評価も進められている．11520 ノードによる強スケーリングの評価であっても並列化効率は 90%を達成しており，アプリケーションとしての並列最適化は十分行われていると言える．

3. ARTED の計算フローと計算量

ARTED では，電子の波動関数を表現するために k 点，バンド番号，空間格子点を設定する．k 点の数を NK，バンドの数を NB，x, y, z 方向の空間格子点数を (NLx, NLy, NLz) とすると，波動関数は (NK, NB, NLx, NLy, NLz) の配列で表現される．ただし，空間格子点は一次元で確保されるため実際には (NK, NB, NL) の配列となる．今回，評価に使用する SiO_2 では (4^3, 48, 36000 = (20, 36, 50)) となり，ダイアモンド構造の Si では (24^3, 32, 4096 = (16, 16, 16)) となる．この時，MPI によって k 点が分散され，バンドは

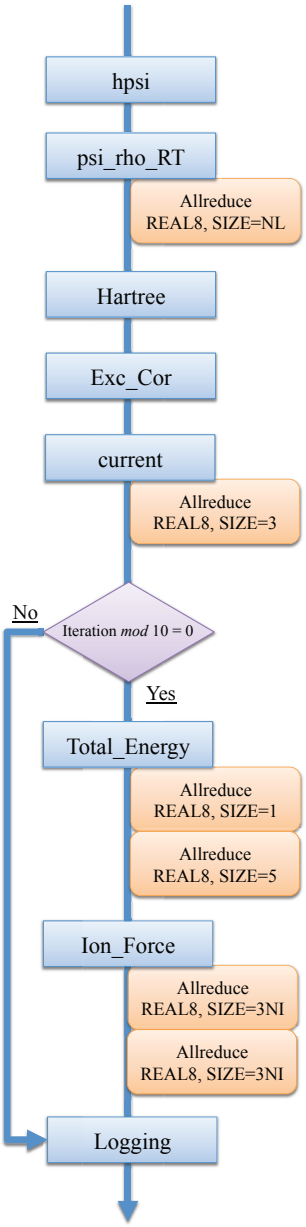


図 1 ARTED の時間発展部の計算フロー

OpenMP を用いて並列計算を行う．即ち，MPI の各プロセスは (NK / # of process, NB, NL) の計算領域を持つことになる．従って，ARTED の評価は全て強スケーリングで行う．

ARTED はまず RSDFT と同様の方法を用いて電子の基底状態を求めるが，前述の通り基底状態の計算は全体から見れば小さなものであり本研究では基底状態の計算について評価を行わない．基底状態を求めた後，時間発展でステンシル計算を行う．各 OpenMP スレッドが空間格子点 (NL) に対しステンシル計算を並列に行うため，並列数は (NK / # of process) × NB である．従って，k 点とバンド数がより多くなればステンシル計算の並列性が高い．

時間発展部の計算フローを**図 1** に示す．時間発展部では，最大で 6 回の Allreduce が発生する．そのうち 1 つは

ⓒ 2015 Information Processing Society of Japan

表 1　評価環境

Number of Node	32
CPU	Intel E5-2670v2 ×2
Number of Cores	20 (10 cores ×2 sockets)/Node
Memory	64GB
Xeon Phi	7110P ×2/Node
Infiniband	FDR Connect-X3 (56 Gbit/s)
Compiler	Intel 15.0.0
MPI	Intel MPI 5.0.1
Intel MKL	11.1.2
OS	Red Hat Enterprise Linux Server 6.4
MPSS	3.4.1
OFED	1.5.4-1

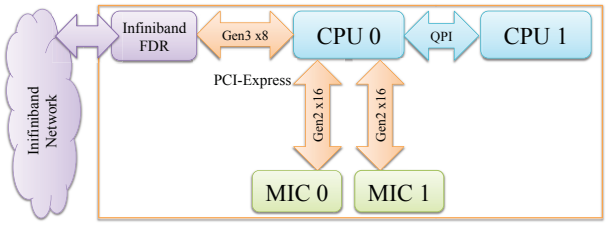


図 2　COMA のノード構成

空間格子点全体の値を参照する Allreduce で，サイズが NL の倍精度浮動小数点数ベクトルに対して総和を計算する．ここで，NI は単位セルに含まれる原子数を表し，SiO_2 の場合 18，ダイアモンド構造の Si では 8 となる．従って，特に Allreduce が発生するカーネルの計算時間に配慮する必要がある．図 1 の中で特に計算量が多いのは hpsi で，前述したステンシル計算を行うカーネルである．計算領域の空間格子点は倍精度複素数で表現され，周期境界条件による 25 点ステンシル計算が行われる．コードは後述するが，隣接点に対する係数が全てキャッシュに保存されていると仮定すると，Byte/Flop は各次元で 20 となりメモリバンド幅に非常に律速される．current も同様の計算を行うが，hpsi では同じ空間格子点のステンシル計算を 4 回連続で行う必要があるためより多くの計算時間を要する．

以上より，本研究ではまず 25 点ステンシル計算カーネル hpsi の最適化を行い，どの程度の性能を得られるかを測定する．その後，特に Allreduce を呼び出しているカーネルについて最適化を行い，最後に CPU のみでの実行，Native 実行と Symmetric 実行による時間発展部の性能評価を行う．

4. 評価環境

本研究では，筑波大学計算科学研究センターの Xeon Phi クラスタ COMA の 32 ノードを用いる [7]．COMA は 2 台の Xeon Phi を搭載したノード 393 台で構成され，ノード単体の理論ピーク演算性能は約 2.5TFLOPS，システム全体で約 1PFLOPS である．評価環境を**表 1** に示し，ノード構成を**図 2** に示す．COMA では，InfiniBand および Xeon Phi は全て CPU0 に接続されている．一般的な構成では，各ソケットに対し同じ台数の Xeon Phi を接続すると考えられるが，COMA では QPI を経由した際の Xeon CPU 内蔵の PCI-Express スイッチにおけるデバイス間通信性能低下問題を考慮し，InfiniBand も合わせて同一ソケットへ接続されている．NVIDIA GPU での実験ではあるが，QPI を経由したノード間通信は，経由しない場合に比べて通信バンド幅に倍以上の性能差があることが分かっている [8]．

ノード内のプロセス割り当ては，CPU の場合ソケットあたり 1 プロセスを割り当て，OpenMP は 10 スレッドで計算する．Xeon Phi の場合は 1 台に対して 1 プロセスを割り当て，240 スレッドで計算する．以上のプロセス割り当て方針は，Symmetric 実行を考慮したためで，この構成で Symmetric 実行を行うと 2 台の CPU と 2 台の Xeon Phi に各 1 プロセスが割り当てられるため，ノードあたり 4 プロセスが割り当てられる．もし 2 台の CPU が計 20 コアのフラット環境として 1 プロセスのみを割り当てると，Symmetric 実行ではノードあたり 3 プロセスと奇数プロセスになり，計算領域の割り当てが難しくなる．

5. Xeon Phi による実装

5.1 ステンシル計算カーネル hpsi のチューニング

Xeon Phi では，512bit 幅の SIMD 命令を用いなければ，高い性能を得ることは難しい．Intrinsics を用いて手動でベクトル化を行うこともできるが，Intel コンパイラではベクトル化可能性を検出し自動でベクトル化を行えるため，コンパイラによるベクトル化を支援する形でコードを修正する，という方針の方が実装効率は高い．この方針では，対象のコードを検証し，コンパイラが十分にベクトル化が行えるようにコードを適宜修正する必要があるが，十分にベクトル化されている場合であっても，キャッシュメモリやレジスタの利用方法といったコンパイラの最適化方針によっては，期待した性能が得られない場合がある．Intel コンパイラでは，最適化に関する方針や，Xeon Phi 専用の最適化オプションが用意されているため，これらのオプションを用いることで性能向上が期待できる．また，Xeon Phi では最大で 240 個のスレッドを利用するためスケジューリング方法や，アフィニティといった OpenMP 実行環境の調整によって，性能向上の可能性がある．従って，チューニングではまずコードの修正，最適化を行い十分なベクトル化が行われているかを検証し，最適化オプションの付与や OpenMP 実行環境の調整によって更なる性能の改善を行う．

チューニングには，SiO_2 のデータを利用する，SiO_2 は 4^3 の k 点，48 のバンド，$20 \times 36 \times 50$ の空間格子点を持つ．しかしながら，ステンシル計算の並列数は前述の通り k 点およびバンド数で決まるため，最大でも 3072 並列しか

ⓒ 2015 Information Processing Society of Japan

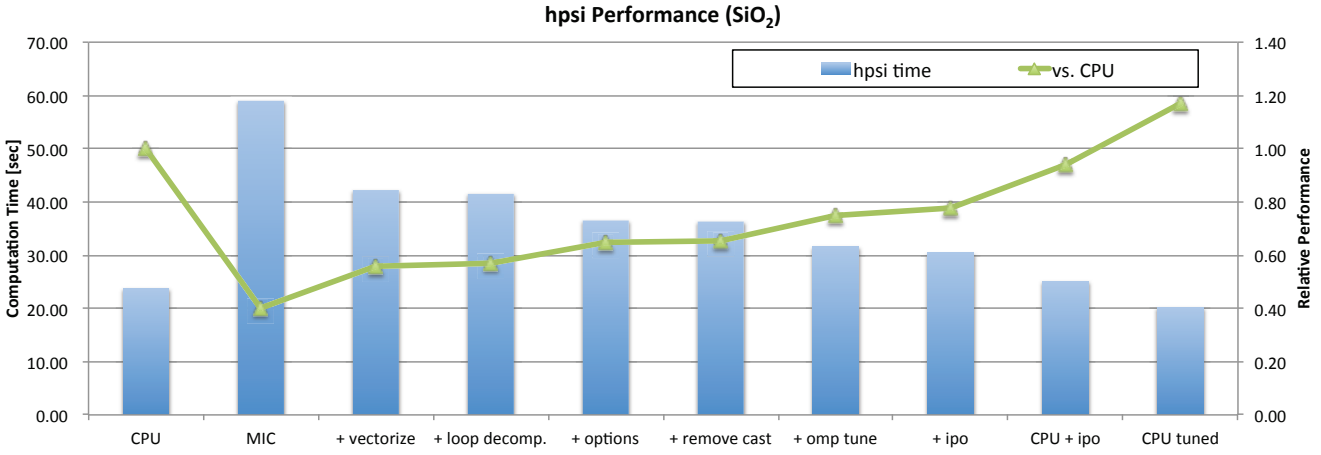


図 3 hpsi の Xeon Phi 向け最適化と CPU との性能比較

表 2 SiO_2 を用いた場合のステンシル計算の並列数

# of Process	Loop Size	Field Size/Process
1	3072	110592000
2	1536	55296000
4	768	27648000
8	384	13824000
16	192	6912000
32	96	3456000
64	48	1728000

ない．ここで，プロセス数とステンシル計算の並列数の対応表を**表 2** に示す．表 2 の通り，16 プロセスで並列数が Xeon Phi が同時に並列計算可能な 240 スレッドよりも少なくなり，Xeon Phi で計算する場合に並列数が全く足りていない．そのため，本研究では 240 スレッドが同時に計算できるように 8 プロセスでカーネルの最適化を行った．

時間発展部の計算を 100 ステップ行ったときのステンシル計算の実行時間と最適化後の結果を**図 3** に示す．まず，特に最適化を行わずに CPU と Xeon Phi で実行した場合，Xeon Phi の性能は CPU の約 40%しかない．これは計算カーネルが Xeon Phi で十分にベクトル化されていないのが原因と考えられる．ARTED ではステンシル計算に用いられる係数が，各次元で別の配列として確保されているため，まずこれをまとめた．また隣接点の計算も各次元で別々に行われており，効率的なベクトル化が行われていなかったため，計算をまとめてベクトル化しやすいように変更を行った．その結果，CPU に対する性能は 56%まで改善した．次にループが NL に対するループになっていたため，これを (NLx, NLy, NLz) の 3 重ループに分解しキャッシュヒット率を上げようとしたが，大きな性能向上には繋がらなかった．チューニング前後のステンシル計算のメインコードを**図 4** および**図 5** に示す．`unroll` ディレクティブを挿入することで，内部の j ループは完全アンロールされ，コンパイラによって最適化が行われる．

次に Intel コンパイラの最適化オプションを試し，性能改善を図った．最適化オプションの詳細については，Intel 社のドキュメントを参照されたい [9]．使用したオプションは以下の通り．

- (共通) -ipo -fp-model fast=2 -complex-limited-range -no-vec-guard-write -qopt-ra-region-strategy=block
- (Xeon Phi のみ) -qopt-threads-per-core=4 -qopt-gather-scatter-unroll=4 -qopt-assume-safe-padding -opt-streaming-cache-evict=0 -qopt-streaming-stores always

特に性能向上に寄与していたオプションは，パディングをコンパイラが行うための -opt-assume-safe-padding と，コアあたりのスレッド数を明示して最適化を行う -qopt-threads-per-core であった．どちらのオプションも Xeon Phi 専用である．-qopt-threads-per-core=4 と指定すると，コアあたり 4 スレッドが動作するとしてコンパイラに最適化を指示する．Xeon Phi のみに付与したオプションの中で，CPU でも効果があるオプションは -qopt-streaming-stores のみである．-qopt-streaming-stores オプションはキャッシュ階層の汚染を最小限にするように，ストアされるデータをメモリに直接書込む命令を発行させる．これに関連し，-opt-streaming-cache-evict はキャッシュ退避命令の挙動を指定するオプションで，0 を指定するとキャッシュ退避命令を発行しない．最適化オプションによって，相対性能はさらに 65%まで改善した．次に，暗黙の型変換によって係数が倍精度浮動小数点数から倍精度複素数へ変換されていたのを修正したが，改善は見られなかった．

また，ここまで OpenMP のスレッド数は COMA に搭載された Xeon Phi で使用できる 240 スレッドを使うように指定していたが，真に最適な値かどうかを調査した．結果として，SiO_2 の場合は 120 スレッドが最適な性能であった．原因としては，並列数が 384 であったため 240 スレッドで計算すると OpenMP のオーバー

ⓒ 2015 Information Processing Society of Japan

```
real(8)     :: cef
complex(8)  :: zI,sx,sy,sz,tx,ty,tz
real(8)     :: lapx(4),lapy(4),lapz(4)
real(8)     :: nabx(4),naby(4),nabz(4)
integer     :: idx(4,NL),idy(4,NL),idz(4,NL)
complex(8)  :: u(NL),s(NL)

do i=1,NL
  sx=lapx(1)*(u(idx(1,i))+u(idx(-1,i)))&
   &+lapx(2)*(u(idx(2,i))+u(idx(-2,i)))&
   &+lapx(3)*(u(idx(3,i))+u(idx(-3,i)))&
   &+lapx(4)*(u(idx(4,i))+u(idx(-4,i)))

  tx=nabx(1)*(u(idx(1,i))-u(idx(-1,i)))&
   &+nabx(2)*(u(idx(2,i))-u(idx(-2,i)))&
   &+nabx(3)*(u(idx(3,i))-u(idx(-3,i)))&
   &+nabx(4)*(u(idx(4,i))-u(idx(-4,i)))

  ! 以下y, z次元についても同様に計算

  s(i)=cef*u(i)-0.5d0*(sx+sy+sz)-zI*(tx+ty+tz)
end do
```

図 4 ステンシル計算のコード

```
real(8)     :: cef
complex(8)  :: zI,st,tt
complex(8)  :: lapt(12),nabt(12)
integer     :: idpt(12,NL),idmt(12,NL)
complex(8)  :: u(NL),s(NL)

!dir$ vector aligned
do iz=1,NLz
do iy=1,NLy
do ix=1,NLx
  i=((iz-1)*NLy*NLx)+((iy-1)*NLx)+ix

  st=0.d0
  tt=0.d0
  !dir$ unroll(12)
  do j=1,12
    st=st+lapt(j)*(u(idpt(j,i))+u(idmt(j,i)))
    tt=tt+nabt(j)*(u(idpt(j,i))+u(idmt(j,i)))
  end do
  s(i)=cef*u(i)-0.5d0*st-zI*tt
end do
end do
end do
```

図 5 チューニング後のステンシル計算のコード

ヘッドが無視できないレベルになったことが予想される．また，スケジューリングのデフォルトはブロック分割 (OMP_SCHEDULE=static) だが，この計算ではサイクリック分割 (OMP_SCHEDULE=static,1) または動的サイクリック分割 (OMP_SCHEDULE=dynamic,1) の方がより高い性能を得られた．またスレッドアフィニティ (KMP_AFFINITY) も隣接スレッドが同じコアに配置される compact や balanced ではなく，隣接スレッドが別々のコアに配置される scatter が最適であった．計算が 25 点の倍精度複素数のステンシルでありメモリバンド幅律速のため，キャッシュをスレッド間で共有するよりも，メモリアクセスを分散させ，アクセスレイテンシを隠して計算を行った方がよりパフォーマンスに寄与できるためと考えられる．OpenMP の実行環境を変更することで，相対性能は 75%と大きく改善した．最後に，IPO (Interprocedual Optimization) を有効にし，最終的な相対性能は 0.78 となった．

しかしながら，これらの最適化を同様に施すことにより，当然 CPU の性能も改善していると考えられる．ところが，ここまでの最適化を施すと CPU は元よりも性能が低下し，94%の性能しか得られなかった．性能が下がった原因は，-qopt-streaming-stores オプションが CPU では悪影響を与えていることが分かったため，同オプションを外すことでオリジナルコードに対する相対性能は 117%と向上した．チューニング後の CPU と比べるとチューニング後の Xeon Phi の相対性能は 66%となる．

表 3 Si (diamond) を用いた場合のステンシル計算の並列数

# of Process	Loop Size	Field Size/Process
1	221184	1769472000
2	110592	884736000
4	55296	442368000
8	27648	221184000
16	13824	110592000
32	6912	55296000
64	3456	27648000
128	1728	13824000
256	864	6912000
512	432	3456000

5.2 Si での hpsi の性能

5.1 節で用いた SiO_2 では，表 2 の通り並列数が十分とは言えない．そこで，より高い並列性を持つ問題として，ダイアモンド構造を持つ Si を用いて hpsi の性能を測定した．Si の場合の並列数を**表 3** に示す．Si では，k 点が 24^3，バンド数が 32，空間格子点が 16^3 となる．ステンシル計算の並列数は SiO_2 に対して 72 倍になり，8 プロセス時には 27648 であるため，Xeon Phi で十分な並列数が確保できると考えられる．しかしながら，空間格子点は SiO_2 に対して 8 分の 1 となり，サイズは 4096 となるため 1 回のステンシル計算の計算量は減っている．問題サイズが影響し，8 プロセス以下では Xeon Phi のメモリ不足が発生し実行が困難となったため 16 プロセス以上での測定を行った．

5.1 節の最適化を施したコードで，時間発展部を 100 ステップ計算したときの 16 プロセスから 64 プロセスでの hpsi の計算時間を**図 6** に示す．ただし，並列数の関係で Xeon Phi の OpenMP スレッドは SiO_2 では 120 が最適であったのに対し Si では 240 が最適であったため，240 ス

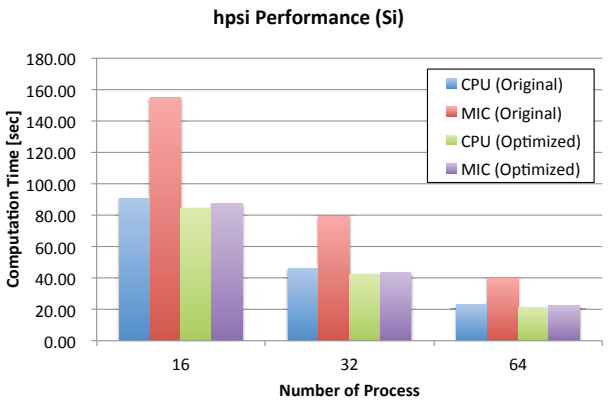


図 6 Si での hpsi の計算時間

レッドで計算を行っている．オリジナルコードの実行では Xeon Phi は CPU に対し 2 倍近くの計算時間を要していたが，チューニングによって CPU 1 ソケットに相当する性能が得られている．この結果は非常に重要であり，これにより CPU 部と Xeon Phi 部の性能がほぼ均等であることから，問題の MPI プロセス分割が自然に行え，ヘテロジニアス環境であるにもかかわらず負荷分散問題に悩まされずに済む．

5.3 時間発展部全体の最適化

current も hpsi と同様なステンシル計算を行うため，hpsi と同様の最適化を施す．特に Allreduce 通信が含まれる psi_rho_RT の計算は，オリジナルコードでは hpsi の次に計算時間を要していた．次章で実行時間内訳を示す．

hpsi を除いたカーネルの中で，CPU に比べて Xeon Phi の性能が出ていないケースの多くはループの並列化が行われていないか，並列化が行われていても並列数が 10 から 20 程度で Xeon Phi の並列数を活かしきれていない状態であった．CPU では無視できるレベルであったものが，Xeon Phi では大きな影響があり，特に並列化が行われていない場合 Xeon Phi は十分な性能を出すことができない．また並列化が行われていても，並列数が Xeon Phi に対して最適な値となっているかも調べる必要がある．

6. 性能評価

本章では，前章で行ったチューニングを踏まえ，ARTED の時間発展部の性能について評価を行う．評価は CPU のみでの実行，Native 実行および Symmetric 実行の 3 種類を用いて，CPU と Xeon Phi の個別の性能，Symmetric 実行によって性能向上が可能かを検証する．評価には，ダイアモンド構造の Si のデータを使用し，時間発展部を 100 ステップ計算し計算時間を比較する．

OpenMP のスケジューリングは，ステンシル計算の最適化を行った際に動的サイクリック分割が最適であったが，全体を通してはブロック分割の性能が高いため，omp do 句に schedule(dynamic,1) を記述してステンシル計算 (hpsi, current) のみ動的サイクリック分割とした．ステンシル計算がメモリバンド幅に非常に律速されることは前述したが，ステンシル計算以外の箇所ではブロック分割によってキャッシュの有効利用を図った方が良いと考えられる．

6.1 CPU と Xeon Phi の計算時間内訳

8 ノードから 32 ノードまでの CPU と Xeon Phi の計算時間内訳を**図 7** に示す．この図では，図 1 に示した計算フローの Allreduce による通信時間も各カーネルの時間に含んで計測している．図のように，時間発展部の計算はステンシル計算を行う hpsi および current，そして psi_rho_RT の 3 つのカーネルが大部分を占める．前章で説明したように，Xeon Phi は特にステンシル計算カーネル hpsi の性能が向上し，CPU と同等になっている．また全てのノード，プロセスで同じ傾向を示し，Xeon Phi の計算時間は CPU に比べて多少多い．これは current や psi_rho_RT の性能に起因しており，どちらも CPU に比べて性能が悪くなっている．

psi_rho_RT は，CPU と Xeon Phi の両者で特に性能改善が見られる．これは元々の並列化方法に起因しており，同カーネルでは 3 重ループの計算が行われ，最内のループが並列化できるような計算となっている．オリジナルコードでは，最内ループに対し `omp parallel do` が付けられ，各反復で OpenMP スレッドの初期化と終了に関するオーバーヘッドがかかっていた．そのため，最外ループに `omp parallel` を付け，最内ループには `omp do nowait` を付け最外ループでスレッドを初期化するようにした．この変更のみで，特に CPU は大幅な改善が見られた．

6.2 時間発展部の計算時間

時間発展部のノード数ごとの計算時間を**図 8** に示し，プロセス数ごとの計算時間を**図 9** に示す．CPU ではオリジナルコードに比べて性能が改善し，32 ノードのとき約 1.7 倍の性能が得られた．Xeon Phi は，CPU に対して約 90%の相対性能となった．ステンシル計算の性能はほぼ同等となったが，ステンシル計算以外の処理は CPU 並みにチューニングできていないため，コード全体では性能が悪くなったと考えられる．結果として，Symmetric 実行は Xeon Phi の性能に律速され，32 ノード実行時に CPU のみの場合に対して約 1.68 倍の相対性能が得られた．32 ノードのとき，CPU のみの場合では 64 プロセスで計算が行われるが，Symmetric 実行では CPU と Xeon Phi の両方で 2 プロセス立ち上がり 128 プロセスで計算が行われる．128 プロセスの場合，表 3 の通り並列数が 1728 となるため Xeon Phi で並列数が不足しつつあると考えられる．本研究は 32 ノードまでの評価だが，上述の結果を踏まえると 256 プロセスまたは 512 プロセスまではある程度の性能

ⓒ 2015 Information Processing Society of Japan

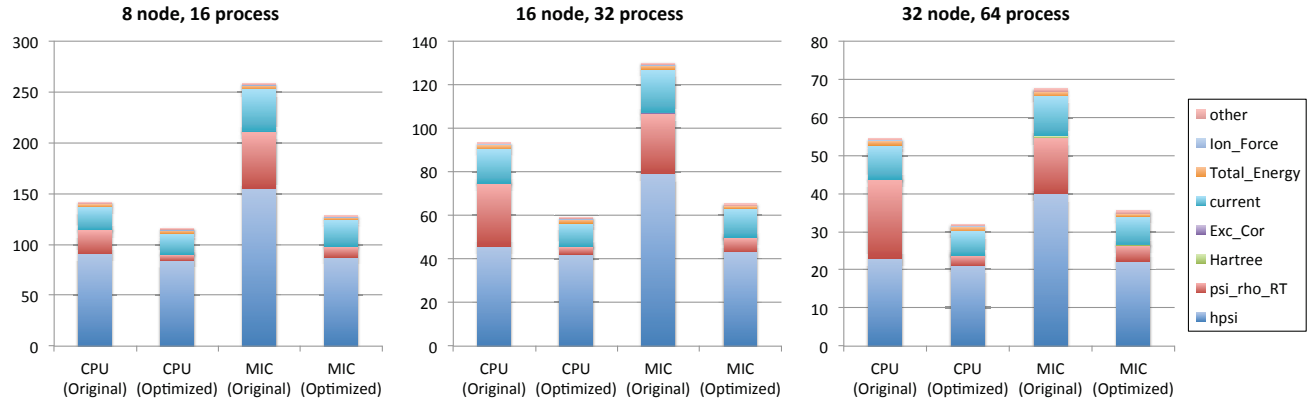


図 7 各ノード，プロセスでの計算時間内訳

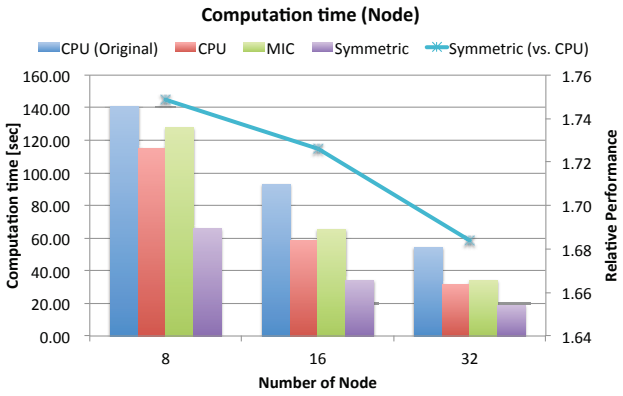


図 8 時間発展部の計算時間 (ノード)

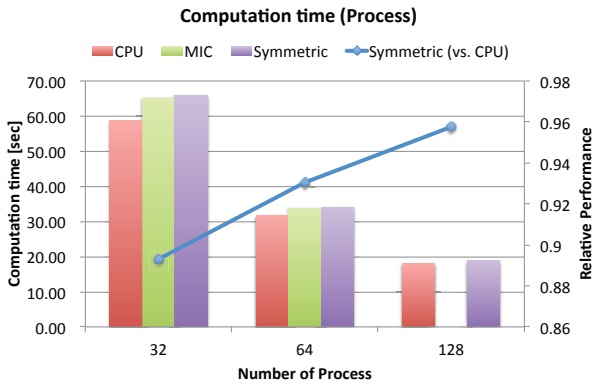


図 9 時間発展部の計算時間 (プロセス)

を期待できる．また別の考え方としては，CPU のみで 32 ノードを要する計算を，Xeon Phi との Symmetric 実行で半分の 16 ノードで実行でき，その計算効率が約 90%である，と考えることもできる．これは，ノード上の全リソースをフルに利用できているという観点での評価である．

理想的には，CPU ソケット 1 つに対して Xeon Phi 1 台の性能が同等となれば，COMA の構成の場合ノードが 2 倍になったと捉えることができるので Symmetric 実行で 2 倍の性能が得られるが，Xeon Phi で性能の出にくい計算も CPU と同じ粒度で行っており，それがボトルネックとなってしまっている．また Symmetric 実行では計算するプロセス数が増加するため，より高い強スケーリング性が要求される．しかしながら，Xeon Phi はこれまでの CPU に比べると 10 倍以上の規模の OpenMP スレッドを準備する必要があり，それに付随する形で，ある程度の計算量や計算粒度がなければ Xeon Phi で高い性能を獲得するのは困難である．よって，Symmetric 実行では強スケーリングがより難しいと考えられる．

7. まとめ

本研究では，実時間実空間密度汎関数理論に基づく電子動力学シミュレータである ARTED を Xeon Phi クラスタへポーティングし，その性能評価を行った．

結果として，ステンシル計算部分のみでは CPU とほぼ同等の性能を得ることができ，これにより Symmetric 実行で良い性能を得ることが期待できた．しかしながら，幾つかのカーネルで CPU に比べて多く計算時間を要し，ARTED の時間発展部の計算全体で見ると，CPU 1 ソケットと Xeon Phi 1 台では CPU に対して約 90%の相対性能となった．この影響で，Symmetric 実行は Xeon Phi の処理性能に律速されてしまい，同一プロセス数で比べると CPU よりも性能は低下した．だが，同一ノード数で比べた場合 Symmetric 実行は CPU のみの実行に比べて 32 ノード実行時に約 1.68 倍の性能が得られ，チューニングにより Symmetric 実行が有効であることが分かった．

今後は，CPU と Xeon Phi の性能が一致するように更なる最適化や，より大規模環境での性能調査が必要であると考えられる．

謝辞 本研究の評価環境は，筑波大学計算科学研究センターの平成 26 年度学際共同利用プログラム課題「時間依存密度汎関数理論によるパルス光と物質の相互作用」による．本研究の性能チューニング，評価にあたり筑波大学大学院システム情報工学研究科の児玉祐悦教授，東京大学情報基盤センターの塙敏博准教授には様々なご助言を頂きました．ここに感謝申し上げます．

ⓒ 2015 Information Processing Society of Japan

参考文献

[1] 松田 元彦，丸山 直也，滝沢 真一郎：Xeon Phi (Knights Corner) の性能特性とステンシル計算の評価，情報処理学会研究報告， Vol. 2014-HPC-143, No. 32 (2014).
[2] 前田 宏志，高橋大介：GPU/MIC クラスタにおける疎行列ベクトル積の性能評価，情報処理学会研究報告，Vol. 2014-HPC-144, No. 4 (2014).
[3] 大島 聡史，金子 勇：メニーコアプロセッサ Xeon Phi の性能評価，情報処理学会研究報告， Vol. 2013-HPC-139, No. 20 (2013).
[4] Shunsuke A. Sato and Kazuhiro Yabana: Maxwell + TDDFT multi-scale simulation for laser-matter interactions, *J. Adv. Simulat. Sci. Eng.*, Vol. 1, No. 1, pp. 98–110 (2014).
[5] Hasegawa, Y., Iwata, J.-I., Tsuji, M., Takahashi, D., Oshiyama, A., Minami, K., Boku, T., Shoji, F., Uno, A., Kurokawa, M., Inoue, H., Miyoshi, I. and Yokokawa, M.: First-principles Calculations of Electron States of a Silicon Nanowire with 100,000 Atoms on the K Computer, *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11, ACM, (online), DOI: 10.1145/2063384.2063386 (2011).
[6] Schultze, M., Ramasesha, K., Pemmaraju, C., Sato, S., Whitmore, D., Gandman, A., Prell, J. S., Borja, L. J., Prendergast, D., Yabana, K., Neumark, D. M. and Leone, S. R.: Attosecond band-gap dynamics in silicon, *Science 12 December 2014*, Vol. 346, No. 6215, pp. 1348–1352 (online), DOI: 10.1126/science.1260311.
[7] 筑波大学計算科学研究センター：スーパーコンピュータ COMA (PACS-IX) について, `http://www.ccs.tsukuba.ac.jp/files/coma-general/coma_outline.pdf`.
[8] 小田嶋 哲哉，塙 敏博，児玉 祐悦，朴 泰祐，村井 均，中尾 昌広，佐藤三久：HA-PACS/TCA における TCA および InfiniBand ハイブリッド通信，情報処理学会研究報告，Vol. 2014-HPC-147, No. 32 (2014).
[9] Intel: User and Reference Guide for the Intel Fortran Compiler 15.0, `https://software.intel.com/en-us/compiler_15.0_ug_f`.

ⓒ 2015 Information Processing Society of Japan