Vol.2020-HPC-173 No.4
2020/3/16

電子動力学アプリケーションSALMONのスーパーコンピュータ「富岳」に向けた最適化と性能評価

廣川 祐太[1,a)]　野田 真史[1]　山田 俊介[1]　山田 篤志[1]　朴 泰祐[1]　矢花 一浩[1]

概要：我々は電子動力学アプリケーション “SALMON” の開発と最適化を進め，Intel Xeon Phi (Knights Landing, KNL) クラスタ “Oakforest-PACS” での全系実行・性能評価を行ってきた．現在，製造と設置が進められているスーパーコンピュータ「富岳」に搭載される “A64FX” プロセッサはメニーコア，高バンド幅メモリ，512 bit SIMD と KNL に非常に似た特徴を持ち，KNL プロセッサへの最適化と性能評価は富岳の活用に強く貢献することが期待される．本研究では我々が特に KNL に対し最適化を進めてきた SALMON を A64FX に移植，さらなる最適化を行い，富岳共用前評価環境にて実施した最大 1152 ノードまでの性能評価から，SALMON のメニーコア型超並列計算機環境下での課題について論ずる．

1. はじめに

これまで, 我々は電子動力学アプリケーション “SALMON (Scalable Ab-initio Light-Matter simulator for Optics and Nanoscience)” [1] の開発を進め，メニーコアプロセッサ，特に Intel Xeon Phi を対象として最適化を行ってきた．[2] では，筑波大学と東京大学が共同設置する JCAHPC (Joint Center for Advanced HPC, 最先端共同 HPC 基盤施設) にて稼働している Knights Landing (KNL) クラスタ “Oakforest-PACS” [3] での全系実行と性能評価を報告し，大規模メニーコアクラスタにおける良好な性能を示した．現在，我々はスーパーコンピュータ「富岳」 (ポスト「京」コンピュータ) を次のターゲットシステムとして，同システムによる大規模電子動力学シミュレーションの実現を目指している．

富岳は，Arm v8.2-a アーキテクチャベースの “A64FX” プロセッサ [4] を採用したメニーコア型システムで、(1) 48 コア (12×4) のメニーコア構成，(2) 合計 1024 GB/sec の高バンド幅メモリ HBM2 を 32 GiB 搭載，(3) Arm Scalable Vector Extension (SVE) [5] をサポートし 512 bit 幅の SIMD 演算を提供する．A64FX プロセッサは KNL と非常に近い構成を持っており，KNL へのアプリケーション最適化は富岳の活用に向けて非常に有効である．これまでに，核となるステンシル計算について AVX-512 intrinsics を用いた手動ベクトル化実装から，ACLE (Arm C Language Extension for SVE) intrinsics を用いた 512-bit SVE への書き換えを論じた [6], [7]．本研究では，富岳共用前評価環境と KNL クラスタの Oakforest-PACS で性能評価を行い，富岳における SALMON の実性能とボトルネック，課題について論ずる．

本研究の構成は以下の通りである．2 節では，光科学シミュレータ SALMON の概要と主たる計算と通信を説明し，3 節で大規模シミュレーションのターゲットシステムである富岳と，比較評価した KNL クラスタ Oakforest-PACS について述べる．4 節では特に主たる計算であるステンシル計算の最適化と，集団通信の最適化について述べる．5 節と 6 節ではステンシル計算と実時間発展の性能評価と大規模計算での課題について考察し，7 節で本研究をまとめる．

2. SALMON: 光科学シミュレータ

2.1 概要

“SALMON (Scalable Ab-initio Light-Matter simulator for Optics and Nanoscience)” は，主に筑波大学計算科学研究センターと自然科学研究機構分子科学研究所にて開発されている，光と物質の相互作用の第一原理計算を目的とした光科学シミュレータである [1]．SALMON は，我々がこれまでにメニーコアプロセッサへ向けた最適化を行ってきた ARTED [8] と，ARTED から派生した GCEED [9] を統合し，それぞれ異なるスケールでのシミュレーションを 1 つのアプリケーションとして利用可能としている．SALMON のベースとなっている 2 つのアプリケーションは，対象とする問題 (方程式) の規模，並列化における分割方法などが異なるが，計算内容はほぼ同一で，[2] で最適化

[1] 筑波大学 計算科学研究センター
[a)] hirokawa@ccs.tsukuba.ac.jp

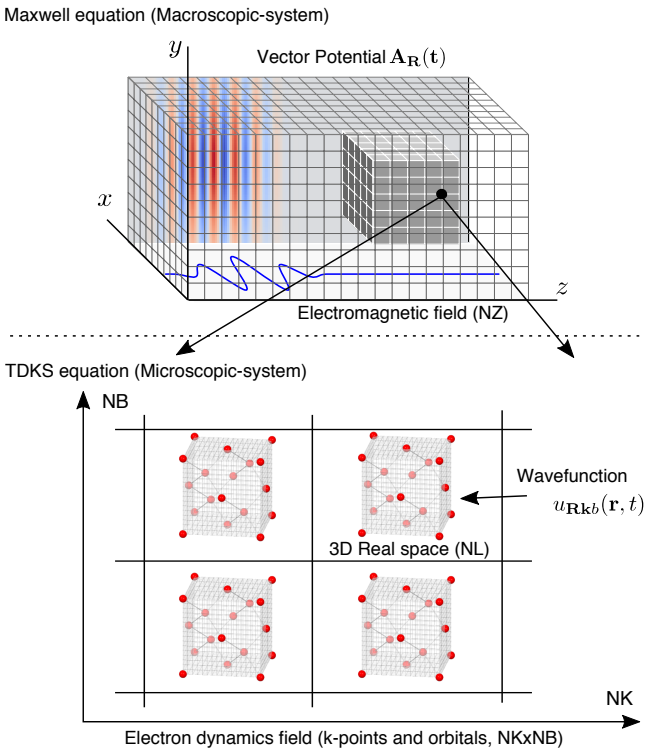

図 1 TDKS+Maxwell マルチスケールシミュレーション

してきたコードをベースとできる．現在までに SALMON version 1.2.1 をリリースし，現在は最適化とアルゴリズムの整理を行い version 2 のリリースを準備している．本研究の実装は，整備中の version 2 を用いる．

SALMON は，TDKS (Time-Dependent Kohn-Sham) 方程式において，実時間・実空間法により電子の波動関数の記述および求解を行っている．また，電子動力学と電磁気学を組合せたマルチスケール計算が可能で，電磁気学の方程式には Maxwell 方程式を FDTD (時間領域差分，Finite-Difference Time-Domain) 法により解き，TDKS 方程式と Maxwell 方程式を電流項で結合する．**図 1** に，TDKS 方程式と 3-D Maxwell 方程式を組合せたマルチスケール計算の際の空間格子について示す．マルチスケール計算時の TDKS 方程式はプロセッサ数台に収まる程度に小さく，TDKS 方程式を FDTD 法のグリッド点数分，サブコミュニケータで閉じて計算する．サブコミュニケータ間は一般的な差分法と同様にグリッド点について近傍点通信を行うため，FDTD 法を解く場合の計算および通信コストは TDKS 方程式の計算に比べて極めて小さく，大部分を TDKS 方程式の求解に費やす．我々はこれまで，マルチスケール計算を大規模シミュレーションのターゲットとして最適化を行ってきた．

本研究では，GCEED が対象としていた TDKS 方程式のみのシミュレーションの性能評価を行う．同計算をマルチスケール計算と対比して，シングルスケール TDDFT (Time-Dependent Density Functional Theory) 計算と呼ぶ．我々は，シングルスケール TDDFT 計算を富岳を用いた大規模シミュレーションの目標と位置づけている．シングルスケール TDDFT 計算は，より複雑な並列性を持つ大規模系の TDKS 方程式を解く．マルチスケール計算における TDKS 方程式は，実空間グリッドが 16^3 程度と小さく，波数空間が最大 $24^3 \times 16$ 程度と非常に大きいため，袖通信が不要な波数空間を MPI で並列計算し on-cache なステンシル計算を非常に多数解く必要があった．シングルスケール TDDFT 計算は，DFT (Density Functional Theory) 計算を行う RSDFT と同様の並列化方法 [10] で，実空間グリッドが最大 256^3 程度，電子軌道 (波数空間) は最大 10^4 のオーダとなり，実空間と波数空間両方に MPI 並列化を行うことで大規模並列性を確保する．

表 1 主な計算と通信のパターン

	Computation	Communication
Hamiltonian	Stencil	Halo
Density	Vector summation	Allreduce
Hartree	FFT	All-to-All
Current	Stencil+Summation	Halo+Allreduce

表 2 各通信のメッセージサイズ，P_n は MPI プロセス数を示す

	Message size / process [Byte]
Halo	$8\frac{L_x}{P_{dx}}\frac{L_y}{P_{dy}} \times 8\frac{L_y}{P_{dy}}\frac{L_z}{P_{dz}} \times 8\frac{L_x}{P_{dx}}\frac{L_z}{P_{dz}} \times \frac{N_{orbital}}{P_{orbital}} \times 16$
Density	$\frac{L_x}{P_{dx}} \times \frac{L_y}{P_{dy}} \times \frac{L_z}{P_{dz}} \times 8$
Hartree	$3 \times (P_{all}-1) \times \frac{L_{gx}\times\frac{L_y}{P_{gy}}\times\frac{L_z}{P_{gz}}}{{P_{all}}^2} \times 16$
Current	Halo + 3×8

我々が目標とする富岳の全系計算は，1–3 万原子系の電子動力学シミュレーションである．静電子状態計算では，RSDFT に代表されるように 10 万原子以上の大規模計算が京コンピュータの規模で報告されている [10]．しかしながら，SALMON のような電子動力学シミュレーションは LLNL の BlueGene/Q 全系や ORNL の Summit の一部を用いた計算などが報告されているが，1000-5000 原子系の規模にとどまっている [11], [12]．数万原子系の大規模電子動力学シミュレーションを富岳を用いて実現することで，ナノ粒子系の光と物質の相互作用における初期状態過程を計算可能となり，太陽電池や光デバイスなど多くの科学技術に対し基礎科学上のブレークスルーを与えることが期待される．

2.2 計算内容

本研究では TDKS 方程式を $\psi(L_x, L_y, L_z, N_{orbital})$ と表す．$L_{x,y,z}$ はそれぞれ各軸の空間格子点数，$N_{orbital}$ は電子軌道数，すなわち波動関数の数を示す．TDKS 方程式の求解における，時間発展計算の主な計算と通信のパターンを**表 1** に示す．時間発展計算は，主に下記の計算で構成される．

(1) ψ のハミルトニアン
(2) 電子密度 $n(L_x, L_y, L_z) = \sum_{i=1}^{N_{orbital}} |\psi(L_x, L_y, L_z, i)|^2$
(3) 電子密度より Hartree ポテンシャルを求める (Poisson 方程式の求解)
(4) ψ について電流項を計算

ハミルトニアンと電流項の計算はほぼ同じ計算が行われ，ψ を更新するか，電流成分として全 MPI プロセスの MPI_Allreduce を行う．計算の半分以上をステンシル計算が占めており，ステンシル計算と通信の最適化が大規模システムでの SALMON の活用において極めて重要である．

表 2 に，通信における各 MPI プロセスのメッセージサイズについて示す．ハミルトニアン計算は，実空間格子に対するステンシル計算と袖通信が中心で，各 MPI プロセスは周期境界条件で 6 方向に一対一通信を行う．電子密度は ψ の畳み込み計算で，全ての波動関数の値をまとめるために Allreduce を必要とする．Hartree ポテンシャルは Poisson 方程式を陰的に解く必要があり，SALMON は Six-step FFT を用いる FFTE version 6.0 を組み込んで使用しているため，行列転値のために All-to-All 通信が必要となる [13]．電流項はハミルトニアンと同様にステンシル計算が行われるが，ハミルトニアンと異なり電流成分の畳み込みが発生する．

表中の $P_{dx,dy,dz,gx,gy,gz,orbital,all}$ は，それぞれ各次元と電子軌道の MPI プロセス分割数と全 MPI プロセス数である．Hartree ポテンシャルの並列化は，電子密度のサイズが実空間格子点数しかないため全 MPI プロセスで Six-step FFT を並列計算し，実装の関係で X 次元は分割されない．MPI プロセス分割数の関係は以下の通り．

$$
\begin{aligned}
P_{all} &= P_{dx} \times P_{dy} \times P_{dz} \times P_{orbital} \\
&= P_{gx} \times P_{gy} \times P_{gz} \\
P_{gx} \bmod P_{dx} &= 0 \\
P_{gy} \bmod P_{dy} &= 0 \\
P_{gz} \bmod P_{dz} &= 0
\end{aligned}
$$

Hartree ポテンシャルの計算サイズは $product(L_{x,y,z})$ のため，プロセス数が増加した場合に通信コストが無視できない可能性が高い．電子密度を計算した時点で，各波動関数を計算するプロセス群で閉じて Hartree ポテンシャルを冗長計算することも可能である．

ハミルトニアンでの袖領域交換は，ステンシル計算とオーバーラップして処理可能である．ただし，ステンシル計算の袖が各次元で ± 4 必要になるため，各プロセスの計算量が少なくなったときにオーバーラップによる処理コストが無視できなくなる可能性がある．本研究では，Idomura らの手法を用いて袖交換に影響しない内点部分の通信と通信オーバーラップ処理した [14].

表 3 本研究の評価環境

	富岳共用前評価環境	Oakforest-PACS
Processor	A64FX	Xeon Phi 7250
# of core	48+2/4 with 2 GHz	68 with 1.4 GHz
Peak perf.	3072 GFLOPS	3046.4 GFLOPS
Memory	32 GiB HBM2	16 GiB MCDRAM
Bandwidth	1024 GiB/s	490+ GiB/s
Network	Tofu-D 6-D mesh-torus 40.8 GiB/s (injection)	Omni-Path Full-bisect. Fat-tree 12.5 GiB/s
OS	RHEL/CentOS 8	RHEL/CentOS 7
Software	Fujitsu compiler/MPI version 4.1.0	Intel compiler/MPI version 2019 u1
SIMD inst.	512-bit SVE	AVX-512

3. 評価環境

本研究の性能評価環境について，**表 3** に示す．富岳は鋭意製造と設置が進められているが，一部が共用前評価環境として提供されている．本研究では同共用前評価環境について，説明を簡便にするため富岳と呼ぶ．特に断りがなければ、本研究で述べる富岳の性能測定結果はすべてラージページ機能を有効にした性能である。

共用前評価環境の A64FX プロセッサは，48 の計算コアが 2 GHz で動作し，各コアで 2 個の FMA ユニットを有するため 32 FLOP/Cycle，理論ピーク性能は 3072 GFLOPS である．計算コアの他に 2 または 4 のアシスタントコアが提供され，アシスタントコアは OS や通信などの割り込みが発生する処理を引き受ける．A64FX は，12 の計算コアと 8 GiB の HBM2 メモリを CMG (Core Memory Group) と呼び，4 つの CMG がチップ内に実装される [4]．4 つの CMG はチップ内インターコネクトで接続され，4 NUMA ノード構成を取る．上述の構成からメモリアクセスを CMG 内で閉じるように，CMG に 1 個以上の MPI プロセスを割り当てる方式が最適と考えられる．富岳は「京」コンピュータに採用された Tofu の改良版である Tofu-D をノード間インターコネクトとして提供する [15]．Tofu-D はリンクあたりのバンド幅は低下したが同時通信リンク数が 4 から 6 に増加し，各リンク 6.8 GiB/s，合計で 40.8 GiB/s のネットワークバンド幅を提供する．

本研究では，富岳との性能比較のために JCAHPC の Intel Xeon Phi (Knights Landing) クラスタ Oakforest-PACS (OFP) を用いる．OFP は富岳と同様のメニーコア型スーパーコンピュータで，搭載されている KNL プロセッサは特徴・特性が A64FX に類似しており，富岳との比較に最適なシステムのひとつといえる．富岳と異なる点は，

- 計算コア数は約 1.3 倍以上だが，各計算コアの動作周波数が定格 1.4 GHz と低い
- 高帯域幅メモリの 16 GiB MCDRAM と，大容量メモ

```
do ib = 1,Norbital
do iz = 1,Lz; do iy = 1,Ly

! initialize
do ix = 1,Lx
  htpsi(ix,iy,iz,ib,ik) = &
    (V_local(ix,iy,iz,ib,ik) + lap0(ik)) &
    * tpsi(ix,iy,iz,ib,ik)
end do

! compute X-difference
do ix = 1,Lx
#define DX(dt) tpsi(idx(ix+(dt)),iy,iz)
  t8=DX(-4); t7=DX(-3); t6=DX(-2); t5=DX(-1)
  t1=DX( 1); t2=DX( 2); t3=DX( 3); t4=DX( 4)

  v=(lapt(1)*(t1+t5) + lapt(2)*(t2+t6) &
    +lapt(3)*(t3+t7) + lapt(4)*(t4+t8))

  w=(nabt(1)*(t1-t5) + nabt(2)*(t2-t6) &
    +nabt(3)*(t3-t7) + nabt(4)*(t4-t8))

  htpsi(ix,iy,iz) = htpsi(ix,iy,iz) &
                  - 0.5d0 * v - zI * w
end do
! compute Y-difference
do ix = 1,Lx ; ... ; end do
! compute Z-difference
do ix = 1,Lx ; ... ; end do

end do; end do; end do
```

図 2 富士通コンパイラに最適化したステンシル計算の Fortran 実装

リの 96 GiB DDR4-2400 DRAM を備える，ただし本研究では MCDRAM しか使用しない

- Intel Omni-Path を用いた Full bisection fat-tree のネットワークを提供する

プロセッサの理論ピーク性能は A64FX とほぼ等価で，本研究ではコアの動作周波数やネットワークトポロジの違いなどの影響について考察を行う．

4. SALMON の最適化

4.1 コンパイラによる SIMD 化の促進

SALMON のステンシル計算は倍精度複素数型で，各次元で ±4 の近傍点が必要かつ周期境界条件で計算される．ステンシル計算の Byte/FLOP は約 2.68 B/F で，非常にメモリバンド幅律速な計算である．我々はこれまでに，京コンピュータや Knights Corner，Knights Landing に同計算を最適化し，高速化を達成してきた [2], [16].

富士通コンパイラの SIMD 化を促進したステンシル計算の Fortran コードを**図 2** に示す．一般的に，ステンシル計算のコードはキャッシュやレジスタの活用を考えると全差分計算を一回のループに収めるのが効率的と考えられるが，富士通コンパイラはレジスタメモリを分割してパイプライン的に処理・最適化するソフトウェアパイプライニング機能がある．同機能は、レジスタ内のデータをパイプライン的に処理するためにひとつのループ中で使用されるレジスタ数を可能な限り少なくしなければならない．すなわち，ループボディはできる限り簡潔かつ短く書かれている必要がある．

```
svfloat64_t _mm512_set4_pd_sve(d, c, b, a) {
  svfloat64_t x = svdupq_f64(a, c);
  svfloat64_t y = svdupq_f64(b, d);
  return svzip1(x, y);
}
```

図 3 _mm_512_set4_pd の SVE 実装

SALMON のステンシル計算は各次元が 4 次差分と袖が深く，一般的な実装ではループボディが長すぎるためにソフトウェアパイプライニングを適用できなかった．したがって，メモリ上連続な最内の X 次元のループを 4 つに分割することで，全てのループでソフトウェアパイプライニングを有効化した．図 2 の実装では，書き込み先配列を一時的なバッファとして利用するため，キャッシュメモリを消費してしまうが，A64FX は Intel CPU のようなキャッシュメモリを経由せず直接メインメモリに書き込む (streaming store) [17] 機能を有しないため，デメリットは生じない．本研究の実装では，ソフトウェアパイプライニングが有効になるループ長は 24–48 以上となり，対象の実空間格子点数に適合する．

4.2 AVX-512 実装の 512-bit SVE への変換

我々は [2] で，Intel AVX-512 intrinsics を用いた同計算の最適化を論じた．A64FX がサポートする SVE (Scalable Vector Extension) 命令 [5] は，ハードウェアの SIMD 幅を 128-bit から 2048-bit の中で実装者が任意に決定できる．A64FX がハードウェアで実装する SIMD 幅は AVX-512 と同じく 512-bit であり，AVX-512 実装を 512-bit SVE に変換することで，手動 SIMD 最適化の実装コストを下げつつ高速化できることが期待される．

AVX-512 から SVE への変換は，SVE intrinsics が AVX-512 の masked intrinsics であると考えれば多くの場合で可換な intrinsic が存在する．しかし，可換でない場合は SVE intrinsics を組み合わせて実装する必要があり，AVX-512 intrinsics の SVE intrisics を用いたエミュレーションと言える．変換に関する詳細は，[7] に詳しい．

非可換命令の例として，`_mm512_set4_pd` を SVE intrinsics でどのように実装するかを考える．ベクトルの要素型が `int32_t` など 128 bit で 4 要素を表現できる場合，`svdupq` で容易に実現できるが，64 bit 整数や浮動小数点数の場合，4 要素は 256 bit 必要となるため，既存命令では処理できない．`_mm512_set4_pd` の SVE 実装を，**図 3** に示す．

```
! zdot(pseudo_pt, psi)
uVpsi = 0d0
do ib=1,Norbital; do ip=1,Nlma
  ia = get_atom_number(ip)
  il = get_projector_number(ip)
  ir = get_num_rgrids(il)

  uVpsi(ia,ib) = sum(conjg(pseudo_pt(1:ir)) &
                     * psi(ipos(1:ir)))
end do; end do

! summation
call MPI_Allreduce(MPI_IN_PLACE, uVpsi, &
                   Natom*Norbital, &
                   MPI_DOUBLE_COMPLEX, MPI_SUM, &
                   icomm_dxyz, ierr)

! update wave function
do ib=1,Norbital; do ip=1,Nlma
  ia = get_atom_number(ip)
  il = get_projector_number(ip)
  ir = get_num_rgrids(ip)

  psi(ipos(1:ir)) = psi(ipos(1:ir)) &
                  + uVpsi(ia,ib) * pseudo_pt(1:ir)
end do; end do
```

図 4 非局所ポテンシャル計算の疑似 Fortran コード

svdupq_f64 で (a, c, a, c, ...) および (b, d, b, d, ...) と値をコピーしたベクトル x, y を生成する．ベクトル x, y を svzip1(x, y) で interleave copy し，ベクトル x, y の最下位ビットから数えて半分の要素を 1 要素ずつ交互にコピーしたベクトルを生成する．図 3 で 4 つのスカラ値で埋めたベクトルを生成できるが，128 bit SVE で実行した場合は a, b しか参照されない．以上のように，ステンシル計算で利用されるような AVX-512 命令を SVE 命令の組み合わせで実現することは容易である．

4.3 原子の非局所ポテンシャルの通信局所化

図 1 に示すとおり，各波動関数は実空間内に複数の原子を持つ．第一原理計算は計算コストの観点から，原子のポテンシャルについて直接計算を行わず任意のポテンシャル関数に置き換えて計算するのが一般的である．これを擬ポテンシャル (Pseudo-potential) と呼ぶ [18] が，擬ポテンシャルによるシミュレーション精度上の問題点について本研究では議論しない．

擬ポテンシャル関数を用いた波動関数の更新を，非局所ポテンシャル計算と呼ぶ．非局所ポテンシャル計算は各原子について，擬ポテンシャル関数と原子相当部分の実空間格子点の内積和を求める必要があるため，ある原子が複数の MPI プロセスによって分割計算されている場合に通信が必要となる．内積和を求めたあと，波動関数をその値にしたがって更新する．

```
! ... nonlocal potential
n = 0
do ia=1,Natom
  if (is_atom_referred(ia)) then
    n = n + 1
    call MPI_Iallreduce(MPI_IN_PLACE,
                        uVpsi_r(1,ia), &
                        Norbital, &
                        MPI_DOUBLE_COMPLEX, &
                        MPI_SUM, &
                        icomm_atom(ia), &
                        ireq(n), ierr)
  end if
end do
call MPI_Waitall(n,reqs,MPI_STATUSES_IGNORE,ierr)

do ia=1,Natom
  uVpsi(ia,:) = uVpsi_r(:,ia)
end do
```

図 5 MPI_Iallreduce を用いた非局所ポテンシャルの通信最適化

図 4 に，非局所ポテンシャル計算の擬似コードを示す．擬ポテンシャル関数は計算コストを削減するために，すべて初期化時に求めて配列 (pseudo_pt) に保存し，計算時は配列の読み取りだけを行う．MD 計算を行わなければ，擬ポテンシャル関数は一回の初期化で全時間ステップにおいて利用できる．各 MPI プロセスのローカルな擬ポテンシャル関数と実空間格子点の内積和 (uVpsi) を求めた後，MPI_Allreduce を用いて uVpsi の全要素の和を取る．

ある uVpsi(ia) に着目すると，ia 番目の原子の更新に必要な uVpsi(ia) は，ia の原子を含む実空間格子点を計算する MPI プロセス間だけで通信すれば良い．最も簡単な実装は uVpsi(ia) を計算する MPI プロセスのリストを作り，MPI_Group API を用いて新たなコミュニケータを作成し MPI_Allreduce を呼び出す方法である．しかしながら，原子の数に比例し MPI_Allreduce の呼び出し回数は急激に増加するため，MPI version 3.0 で加えられた Nonblocking collective の MPI_Iallreduce を用いる．**図 5** に，MPI_Iallreduce を用いた通信最適化実装を示す．MPI_Iallreduce を最大 natom 回呼び出し，MPI_Waitall で通信を起動させることで非局所ポテンシャルの内積和の通信量を最小かつ効率よく処理できると考えられる．[19] のように，パイプライン化して通信隠蔽を図ることも可能と考えられるため，実装とパイプライン化の効果は今後の課題とする．

5. 性能評価と考察

5.1 計算問題とプロセス分割方法

本研究では，シングルスケール TDDFT 計算として $Au_{54}H_{48}Si_{144}$ のダングリングボンドのシミュレーションを取り上げ，実時間発展の計算時間について評価を行う．

表 4 Weak scaling の測定に用いた問題サイズ

# of node	W_{ob}	$L_{x,y,z}$
72	648	(128, 64, 81)
288	1296	(128, 128, 81)
1152	2592	(128, 128, 162)

表 5 Weak scaling の MPI プロセスマッピング

# of node	$P_{orbital}$	$P_{dx,dy,dz}$	$P_{gx,gy,gz}$
72	48	(1, 2, 3)	(8, 4, 9)
288	96	(1, 4, 3)	(16, 8, 9)
1152	192	(1, 4, 6)	(16, 16, 18)

表 6 Strong scaling の MPI プロセスマッピング

# of node	$P_{orbital}$	$P_{dx,dy,dz}$	$P_{gx,gy,gz}$
256	64	(1, 4, 4)	(8, 8, 16)
512	128	(1, 4, 4)	(8, 16, 16)
1024	256	(1, 4, 4)	(16, 16, 16)

ダングリングボンドの計算は格子欠陥や物質表面を扱うシミュレーションにあたるため，スーパーセル法が用いられる．スーパーセル法は，シミュレーションの構成について見ると周期境界条件を持つユニットセルのレプリカを大量に並べることで巨大なひとつのセルとして計算する手法である．2 倍のユニットセルを配置するためには軌道と実空間格子点を同時に 2 倍にする必要があるため，問題サイズは 4 倍になる．したがって，計算ノード数を 4 倍刻みにして測定しなければ weak scaling を正しく評価できない．

Weak scaling 計測時の問題サイズについて，**表 4** に示す．軌道および実空間格子点を同時に倍にする必要があるため，72 ノードから測定し 4 倍刻みで性能測定を行う．各ノードは 4 MPI プロセス (1 MPI proc/CMG) を割り当て，合計 MPI プロセス数はノード数の 4 倍である．このとき L_x は固定で，$L_{y,z}$ 方向にユニットセルを配置する．MPI プロセスのマッピング方法は**表 5** に示すとおりで，全プロセスが同サイズの $L_{x,y,z}$ を持つが，L_x は分割しない．

Strong scaling 計測では，表 4 から 288 ノードのサイズを取り出し，各プロセスが持つ実空間格子点数が等価かつ FFTE の問題サイズ制約から L_z を 96 に変更したものを用いた．MPI プロセス分割数は**表 6** に示すように，$P_{orbital}$ のみを増加させる形とした．袖通信とステンシル計算のオーバーラップは，MPI のプロセス分割数が多くなった場合，前述の通り袖が深いために袖領域に依存しない内点の計算量が少なくなるため通信コストが増加してしまう．しかし，軌道分割の場合は分割数に依存せず同じサイズの畳み込み通信 (MPI_Allreduce) を用いるため，相対的に通信コストを低く抑えられる．

5.2 ステンシル計算

OpenMP 並列化されたステンシル計算部のみを抜き出したミニベンチマークを作成し，性能評価を富岳共用前

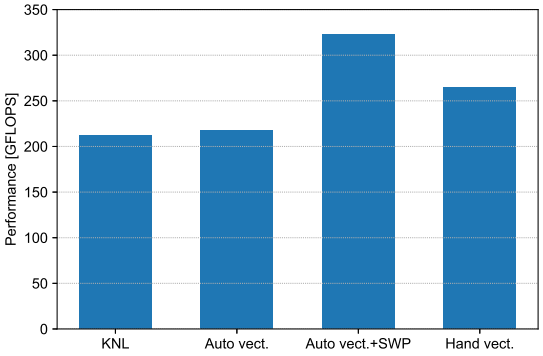

図 6 ステンシル計算のミニベンチマーク測定結果

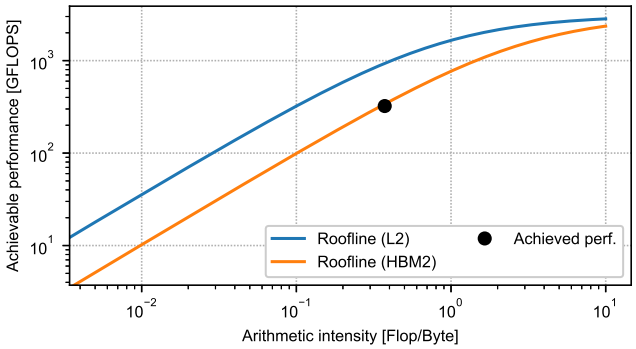

図 7 ステンシル計算の達成可能な性能と達成した性能

評価環境ではなく，富岳試作機を用いて行った．富岳試作機の A64FX プロセッサと，KNL の性能を**図 6** に示す．性能測定は 128^3 の実空間格子点を用いて，A64FX は 48 OpenMP スレッド，KNL (Xeon Phi 7250) は 64 OpenMP スレッドで実施した．Xeon Phi 7250 は 68 コアを搭載しているが，OFP では 2 コアないし 4 コアが OS の処理をできるような実行を推奨しているため，実計算と同様に 64 コアでの測定が適していると判断した．

“Auto” はコンパイラによる SIMD 最適化を促進したもの，“Hand” は KNL の AVX-512 SIMD 命令で実装した手書きの SIMD 化コードを A64FX の 512-bit SVE 命令に書き換えたもの，“SWP” はソフトウェアパイプライニングによる最適化が有効になっていることを示している．単純な SIMD 最適化だけであれば，SIMD 命令を Intrinsic レベルで記述することでコンパイラよりも高い性能が得られる可能性がある．しかし，SIMD 最適化に加えソフトウェアパイプライニングのような，実装者に比べコンパイラが得意と考えられる最適化を組み合わせることで，ハンドコーディングによる SIMD 化は不要なことがある．本研究のハンドコーディング実装では，ループ形状が富士通コンパイラにとって解析できないレベルで複雑になってしまい，ソフトウェアパイプライニングが有効にできなかった．ソフトウェアパイプライニング相当処理を記述することができればさらなる高速化を行える可能性があるが，SIMD intrinsics+ソフトウェアパイプライニング相当処理の記述はコードが複雑になりすぎてしまい，コンパイラによって

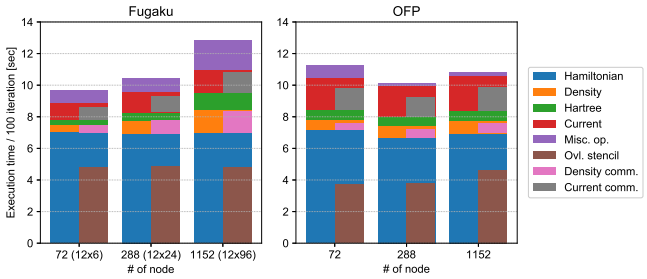

図 8 Weak scaling 測定結果 (実行時間内訳)

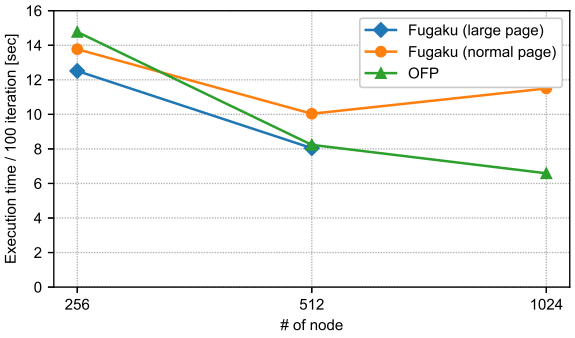

図 9 Strong scaling 測定結果

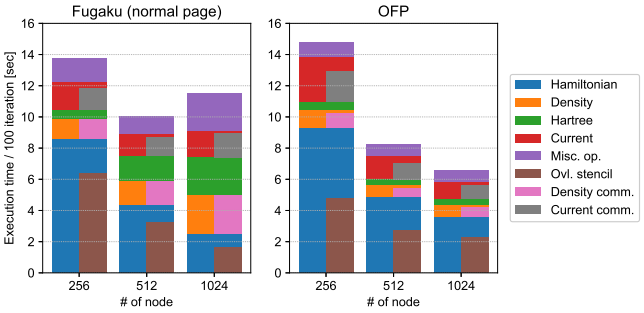

図 10 Strong scaling 測定結果 (実行時間内訳)

行われる他の最適化を妨害または止めてしまうため，現在まで性能向上を確認できていない．

本研究の実装は最高 322.72 GFLOPS を達成し，約 2.68 Byte/FLOP から，ステンシル計算の実効メモリバンド幅は約 866 GiB/s である．A64FX のメモリバンド幅は 1024 GiB/s なので，約 84.5%の実メモリバンド幅を獲得しており，よく最適化されている．ここで，L2 キャッシュおよび HBM2 のメモリバンド幅からルーフラインモデル [20] を**図 7** に作成し，本研究の達成性能をプロットした．L2 キャッシュバンド幅は 3600 GiB/s 以上であるとされているため，グラフでは 3600 GiB/s でルーフラインモデルを作成している．ステンシル計算は計算サイズが L2 キャッシュサイズを超え HBM2 のメモリバンド幅に律速されるため，A64FX で達成可能な最低水準はクリアできている．ただし，キャッシュメモリの影響を考慮すると，真に達成可能な最高性能は L2 キャッシュと HBM2 のルーフラインの間にあると考えられる．その仮定に従えば，ステンシル計算の性能は今だ不十分で，さらなる最適化の余地について検討しなければならない．

5.3 Weak/Strong scaling

富岳および OFP での weak scaling の実時間発展計算全体の測定結果を，**図 8** に示す．Hamiltonian, Density, Hartree, Current およびそれ以外の計算時間の内訳と，通信と計算をオーバーラップしたステンシル計算の実行時間，Density と Current の通信時間について示している．Hartree は FFTE の利用方法の問題で通信時間測定ができていなかったが，All-to-All 通信が支配的であると考えられる．

現在の富岳の環境では weak scaling を達成できていないが，OFP は 72 ノードに対して 1152 ノードは約 96%の実行時間で weak scaling をほぼ完全に達成している．計算や通信量は同一であることを考えると，ネットワークトポロジの違いによるものと考えられる．OFP はどの計算ノードとの通信も等価コストで処理できる Full-bisection fat-tree であるのに対し，富岳はユーザには 3 次元のメッシュトーラスとして見える Tofu-D である．評価時の環境では 1152 ノード (3 ラック) を 4x6x48 の Tohu-D ネットワークで接続しているが，この形状では Tofu-D ネットワークに対し表 5 に示すような SALMON にとって適切なプロセスマッピングができない．特に、全 MPI プロセスで分割計算している Hartree は All-to-All 通信を使用する FFTE が利用されるため，最もネットワークトポロジの影響を受ける．また，他の全体通信も適切なマッピングになっていない結果，遠距離のノードと通信する必要があり，図 8 に示す通り明らかに通信に律速されている．しかしこの問題はネットワーク形状に依存するため，ラックが増加し X と Y 方向のノード数が増えれば，Tofu-D に対して適切にマッピングできるようになり，Weak scaling の改善が期待される．

図 9 に，それぞれ strong scaling の結果を示す．当初 1024 ノードまでの測定を予定していたが，試行期間途中でラージページを利用できなくなったため 256 と 512 ノードと完全に同一の環境で測定が困難となった．したがって，同グラフではラージページを利用した場合は 512 ノードまで，ノーマルページで全て動作させた場合は 1024 ノードまでの結果を示している．両システムともに 1024 ノードですでにスケーリング性能が悪化しているが，富岳にいたっては性能低下を起こしている．しかしながら悪化の理由は異なり，**図 10** に示すように OFP の場合はステンシル計算で構成された hamiltonian の性能に飽和が見られるのに対し，富岳は Hartree や Current の実行時間が増加していることが原因である．しかしハミルトニアン計算に着目すると，通信隠蔽が行われているステンシル計算は OFP と異なりほぼ完璧にスケールしているように読み取れる．

以上から，一対一通信を含む計算性能は OFP に対して

優位性があるものの，全体通信の時間に律速されているような挙動を示しており，ネットワークトポロジへのマッピングの問題が極めて重大な影響を与えていることが推察される．

6. 大規模計算にむけた課題

6.1 通信ボトルネックの克服

図 8 などに示したとおり，プロセスマッピングの問題から通信ボトルネックな挙動を示している．しかしながら，ネットワークへのプロセスマッピングの問題が解消したとしても，そもそものネットワーク帯域幅が足りない可能性も考えられる．富岳は injection network bandwidth は広帯域と言えるが，各リンクは 6.8 GiB/s と InfiniBand FDR 相当のネットワーク帯域しか持たないのに対し，OFP は Full-bisection fat-tree で 12.5 GiB/s のリンクで，単リンクのネットワーク性能は OFP に優位性がある．また，富士通 MPI は Tofu ネットワークに最適化した collective のアルゴリズムを提供しているが，特定の条件を満たした場合に呼び出されるため，高速なアルゴリズムが選択されていない可能性もある．通信ボトルネックを克服するための直接的な手段として，全体通信のパイプライン化による通信隠蔽も有効である．

以上から，通信性能に関する改善手段は以下のことが考えられる．

- 適切な MPI プロセスとネットワークのマッピング
- Tofu-friendly な collective アルゴリズムが適用されているかの確認，または適用できるように実装する
- Nonblocking collective を用いた通信隠蔽を実装する

6.2 Hartree ポテンシャルの冗長計算

SALMON 固有の問題として，スーパーセル計算では完全な weak scaling を行う場合には軌道と実空間をそれぞれ 2 倍にする必要があり，ノード数 4 倍で推移させなければならない．このとき $product(P_{gx,gy,gz})$ は 4 倍になるのに対し、計算対象となる $product(L_{x,y,z})$ は 2 倍にしかならない．つまり計算の特性上，波動関数に対して weak scaling はできるが，Hartree ポテンシャルの計算は自動的に strong scaling になり，一定のノード数に達するとバランスが崩れる可能性がある．任意に Hartree ポテンシャルの計算を冗長化し，例えば同一の軌道を計算する $P_{dx,dy,dz}$ で閉じた計算に変形する必要があると考えられる．

$P_{dx,dy,dz}$ をサブコミュニケータとすると，このサブコミュニケータは Density で計算した電子密度を分割して持っているため，$product(P_{dx,dy,dz})$ のプロセスで閉じて Hartree ポテンシャルを解くことができる．このとき，サブコミュニケータでの Hartree ポテンシャルの計算量は $\frac{product(P_{gx,gy,gz})}{product(P_{dx,dy,dz})}$ 倍になるため，従来手法とのプロトコルスイッチが必要と考えられる．

7. まとめ

本研究では，電子動力学アプリケーション SALMON について，ステンシル計算の最適化と通信局所化，富岳共用前評価環境を用いた 1152 ノードまでの性能評価と KNL クラスタ Oakforest-PACS との性能比較を行った．ステンシル計算はミニベンチマークレベルで約 322 GFLOPS，ピーク性能比 10%を達成し，ルーフラインモデルの評価によって HBM2 に律速された場合の性能とほぼ等価な性能を達成していることがわかった．しかしながら，真に達成可能な性能は L2 キャッシュバンド幅と HBM2 バンド幅のルーフラインの間に存在すると考えられる．

$Au_{54}H_{48}Si_{144}$ のダングリングボンドシミュレーションによる実時間発展の計算性能評価では，アプリケーションに通信ボトルネックが発生していることがわかった．OFP では Weak scaling をほぼ完璧に達成していたことから，SALMON が要求するプロセスマッピングと Tohu-D ネットワークの形状がマッチしなかったために通信性能が最適化されていないのではないかと考察した．この問題を解決するためには，Nonblocking collective を用いた通信隠蔽を実装する必要があると考えられる．

SALMON の富岳における性能は，ハミルトニアン計算についてのみ着目すると十分な Weak/Strong scaling を達成していることがわかる．しかし時間発展計算全体を比較した場合，袖領域交換が支配的なハミルトニアンに比べ全体通信が支配的な他の計算によって全体性能が律速されてしまっている．富岳の計算性能は OFP に対し優位性があるため，通信性能を改善することによって OFP より優れた実行性能を獲得することが期待される．

謝辞 本研究の一部は，筑波大学計算科学研究センターの学際共同利用プログラムによる．本研究の一部は，JST-CREST 研究課題「光・電子融合第一原理計算ソフトウェアの開発と応用 (課題番号: JPMJCR16N5)」により支援された．

免責事項

本研究の「富岳」に関係する評価結果は，スーパーコンピュータ「富岳」試作機および共用前評価環境により得られた．これらは，スーパーコンピュータ「富岳」の共用開始時の性能・電力等の結果を保証するものではない．

評価で使用したコンパイラ等のソフトウェアは開発中のものであり，スーパーコンピュータ「富岳」共用開始時の性能と異なる可能性がある．利用したソフトウェアの版は以下のとおり．

試作機

富士通コンパイラ&MPI version 4.0.0 20190701

共用前評価環境

富士通コンパイラ&MPI version 4.1.0 20191219

参考文献

[1] M. Noda, Shunsuke A. Sato, Y. Hirokawa and *et al.*: SALMON: Scalable Ab-initio Light-Matter simulator for Optics and Nanoscience, *arXiv: Computational Physics*, arXiv:1804.01404 (2018).

[2] Y. Hirokawa, T. Boku, M. Uemoto, S. A. Sato and K. Yabana: Performance Optimization and Evaluation of Scalable Optoelectronics Application on Large Scale KNL Cluster, *ISC High Performance 2018: High Performance Computing*, pp. 205–225 (2018).

[3] 最先端共同 HPC 基盤施設: `http://jcahpc.jp/`.

[4] T. Yoshida: Fujitsu High Performance CPU for the Post-K Computer, *Hot Chips 30* (2018).

[5] Arm Scalable Vector Extension User Guide Version 6.11: `https://developer.arm.com/docs/100891/0611`.

[6] Arm C Language Extensions for SVE: `https://developer.arm.com/docs/100987/latest/arm-c-language-extensions-for-sve`.

[7] 廣川 祐太，朴 泰祐，矢花 一浩: AVX-512 Intrinsics で実装されたステンシル計算の Scalable Vector Extension への展開，第 168 回 HPC 研究会， Vol. 2019-HPC-168, No. 4 (2019).

[8] S. A. Sato and K. Yabana: Maxwell + TDDFT multiscale simulation for laser-matter interactions, *J. Adv. Simulat. Sci. Eng.*, Vol. 1, No. 1, pp. 98–110 (2014).

[9] M. Noda, K. Ishimura, K. Nobusada and *et al.*: Massively-parallel electron dynamics calculations in real-time and real-space: Toward applications to nanostructures of more than ten-nanometers in size, *Journal of Computational Physics*, Vol. 265, No. 14, pp. 145–155 (2014).

[10] Y. Hasegawa, J. Iwata, M. Tsuji and *et al.*: 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).

[11] E. W. Draeger, X. Andrade, J. A. Gunnels and *et al.*: Massively parallel first-principles simulation of electron dynamics in materials, *2016 IEEE International Parallel and Distributed Processing Symposium* (2016).

[12] Weile Jia, Lin-Wang Wang and Lin Lin: Parallel Transport Time-Dependent Density Functional Theory Calculations with Hybrid Functional on Summit, *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '19, ACM, (online), DOI: 10.1145/3295500.3356144 (2019).

[13] Daisuke Takahashi: FFTE: A Fast Fourier Transform Package, `http://www.ffte.jp/`.

[14] Yasuhiro Idomura, Motoki Nakata, Susumu Yamada and *et al.*: Communication-overlap techniques for improved strong scaling of gyrokinetic Eulerian code beyond 100k cores on the K-computer, *The International Journal of High Performance Computing Applications*, Vol. 28, No. 1, pp. 73–86 (online), DOI: 10.1177/1094342013490973 (2014).

[15] Y. Ajima, T. Kawashima, T. Okamoto and *et al.*: The Tofu Interconnect D, *2018 IEEE International Conference on Cluster Computing (CLUSTER)*, pp. 646–654 (online), DOI: 10.1109/CLUSTER.2018.00090 (2018).

[16] 廣川 祐太，朴 泰祐，佐藤 駿丞，矢花一浩: 電子動力学シミュレーションのステンシル計算最適化とメニーコアプロセッサへの実装，情報処理学会論文誌コンピューティングシステム (ACS)， Vol. 9, No. 4, pp. 1–14 (2016).

[17] R. Krishnaiyer, E. Kultursay, P. Chawla, S. Preis, A. Zvezdin and H. Saito: Compiler-Based Data Prefetching and Streaming Non-temporal Store Generation for the Intel(R) Xeon Phi(TM) Coprocessor, *IPDPSW 2013*, pp. 1575–1586 (2013).

[18] N. Troullier and J. L. Martins: Efficient pseudopotentials for plane-wave calculations, *Physical Review B*, Vol. 43 (1991).

[19] Daisuke Takahashi: Automatic Tuning of Computation-Communication Overlap for Parallel 1-D FFT, *2016 IEEE Intl Conference on Computational Science and Engineering (CSE) and IEEE Intl Conference on Embedded and Ubiquitous Computing (EUC) and 15th Intl Symposium on Distributed Computing and Applications for Business Engineering (DCABES)*, pp. 253–256 (online), DOI: 10.1109/CSE-EUC-DCABES.2016.193 (2016).

[20] S. Williams, A. Waterman and D. Patterson: Roofline: An Insightful Visual Performance Model for Multicore Architectures, *Commun. ACM*, Vol. 52, No. 4, pp. 65–76 (online), DOI: 10.1145/1498765.1498785 (2009).