メニーコア型スーパーコンピュータにおける
大規模電子動力学シミュレーションの実現

廣川　祐太、矢花　一浩、山田　篤史

野田　真史(*)、山田　俊介、朴　泰佑

筑波大学計算科学研究センター
(*)現在、株式会社アカデメイア

１．はじめに

筑波大学計算科学研究センターを主な実施機関として、我々は光科学シミュレータ SALMON (Scalable Ab-initio Light-Matter simulator for Optics and Nanoscience) の研究開発を行っている [1]。SALMON は古典的電磁気学 (Maxwell 方程式) と、第一原理電子状態計算 (Time-Dependent Kohn-Sham, TDKS 方程式) を組み合わせた異なるスケールの計算を同時に解く電子動力学シミュレーション (以下、マルチスケール計算) を実現する世界初のアプリケーションである。我々はこれまでに、マルチスケール計算について Oakforest-PACS 全系を用いた性能評価を実施し、良好なスケーリング性能を示した [2]。

マルチスケール計算は巨視的・微視的挙動を同時に解くことで従来法では表現不可能な電子と物質の相互作用をシミュレート可能となっただけでなく、HPC 分野においては、袖領域通信や大規模 IO 処理といった多くの実アプリケーションが大規模実行時の課題として挙げる諸問題を解決可能なスケーラブルなアプリケーションを実装できる。現在、我々は大規模 TDKS 方程式と Maxwell 方程式を組み合わせた電子動力学シミュレーション (以下、シングルスケール計算) の実現を目指している。シングルスケール計算では、先に掲げた諸問題の解決が極めて大きな課題となる。

我々は JCAHPC のご協力の下、シングルスケール計算を対象とした電子動力学シミュレーションの性能評価を大規模 HPC チャレンジにて遂行したが、Oakforest-PACS の 50%程度のノード規模になって初めて下記の複数の問題が発覚し、全系でのシミュレーションを達成することができなかった。

1. 初期波動関数生成時に NaN が発生
   - 初期値生成に使ったガウス関数と使用した擬似乱数の偏りによって、グラムシュミットによる直交化でゼロ除算が発生
2. メモリ消費量の課題
   - 全系計算であっても、MCDRAM (16 GiB) に問題なく収まると考えていたが、メモリ不足が多発した
3. 波動関数の IO 処理コストの爆発的増加
   - 計算対象となる波動関数配列が巨大化するにつれ、IO 処理時間が支配的となった

1 は当時の実装では数学的に問題があったこと、2 は問題規模に対し線形増加で確保されてしまう配列が複数あることが後日の調査で判明し、改修することができた。また、特に 3 はスーパー

コンピュータ「富岳」での実行を見据えた場合、数万ノード規模での IO 処理が必要となるため、大幅な高速化が必要であると考えた。

本稿では、シングルスケール計算におけるアプリケーション固有の課題である大規模 IO 処理の高速化とスーパーコンピュータ「富岳」のネットワークに起因する最適な MPI プロセスマッピングについて、それぞれの実装を紹介する。それらを踏まえ、我々がスーパーコンピュータ「富岳」の試行的利用（富岳共用前評価環境）にて実施した 27,648 ノードまでの弱スケーリング性能を紹介し、現在の課題を述べ結びとする。

２．光科学シミュレータ SALMON

SALMON（Scalable Ab-initio Light-Matter simulator for Optics and Nanoscience）は、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。SALMON は、前進としてマルチスケール計算を実現する ARTED［3］と派生ソフトウェア GCEED［4］を統合し、異なる複数の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計算・実現可能である。2 つは TDKS 方程式の規模、それにともない並列分散方法が異なるが、計算内容はほぼ同一で、［2］で最適化したコードを活用できる。本稿で紹介する最適化も、すべて SALMON の最新版に取り込まれている[1]。

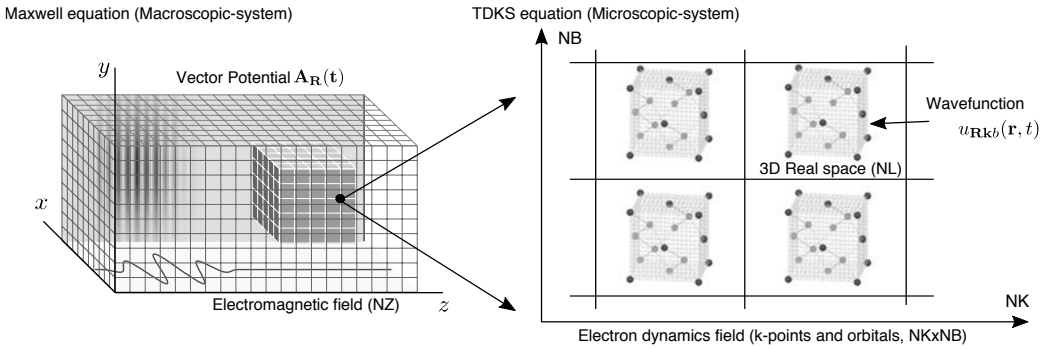

図 1: TDKS+Maxwell マルチスケールシミュレーションのイメージ図

3 次元 Maxwell 方程式で示された巨視的空間上に物質があり、超短波光を照射したときに物質（差分格子）内で発生する微視的現象を TDKS 方程式で解く。

電子動力学計算は、時刻ゼロの状態を決定するために電子の波動関数の基底状態を求める必要がある。これを基底状態計算と呼ぶ。同計算は、岩田らによる電子状態計算ソフトウェア RSDFT と同様のアルゴリズムを採用している［5］。同計算も第一原理計算に基づくため、計算コストが極めて高く、SALMON が対象とする問題規模では基底状態を求めるだけで数時間を要する場合がある。ただし、基底状態計算はグラムシュミットの正規直交化や部分対角化といった $O(N^3)$ の計算が支配的であるのに対し、電子動力学計算はステンシル計算や FFT などメモリ・ネットワークバンド幅に律速されるシミュレーションで、SALMON はどちらも高速に計算可能でなければならない。

[1] http://salmon-tddft.jp

第一原理計算は極めて多くの計算量が必要であることが知られており、マルチスケール計算も例に漏れず TDKS 方程式の求解時間が支配的となる。しかしながら、一般的な TDKS 方程式に対し実空間格子の規模は 16^3 といった一般的な CPU の L2 キャッシュメモリである 512 KiB 未満で、それぞれを独立に並列計算可能な波数空間（バンド計算、k 点計算）が大規模となるような、**MPI による分散並列化が容易に行える**シミュレーションが中心であった。加えて、Maxwell 方程式の差分計算（Finite-Difference Time-Domain method, FDTD 法）と各式の電流項にて結合することにより、FDTD 法の求解で必要とする格子点数の数だけ、TDKS 方程式を解く必要がある。このとき、1 つの TDKS 方程式の問題サイズは比較的少ないプロセッサ（16 台程度まで）で計算可能な規模にとどまり、TDKS 方程式は MPI のサブコミュニケータに閉じて計算が行われる。サブコミュニケータ**内**は TDKS 方程式の通信（波数空間を束ねるための Collective 通信）、サブコミュニケータ**間**は Maxwell 方程式（FDTD 法における格子点の袖交換）の通信に該当する。Maxwell 方程式の格子点数は TDKS 方程式の規模とのバランスから、Oakforest-PACS を用いても 32^3 程度と小さく、Maxwell 方程式の計算コストは無視できるレベルに小さい。

対してシングルスケール計算は Maxwell 方程式を TDKS 方程式と同じスケール、電子間相互作用を記述する。この場合の TDKS 方程式の実空間格子点は 256^3 程度と一般的な差分計算で採用される水準となるため、実空間の分散並列化が必要かつマルチスケール計算に比べて Maxwell 方程式の計算コストが高い。そこで、我々は TDKS 方程式の大規模化に伴う IO データ量の増加と、局所的通信の最適化を行う必要がある。

３．大規模電子動力学計算における大規模 IO 処理の効率化

SALMON では基底状態計算と電子動力学計算をそれぞれ別のバッチジョブとして実行可能なように、2 つの計算をバイナリデータファイルの受け渡しにより接続している。例えば気象・海洋学のアプリケーションでは netCDF や HDF5 が用いられるが、物性物理学では一般的なフォーマットが定義されておらず、また様々なシステムでの動作・利用を目的とする SALMON の性質上、依存するソフトウェアパッケージを削減するため、MPI-IO ネイティブ、および Fortran I/O ルーチン群で実装している。

バイナリデータファイルはほぼすべて電子の波動関数で、大規模実行においては**数 TB から数百 TB 程度のバイナリデータの IO 処理が必要**となる。加えて両計算はどちらも同水準に計算コストが高いため、システムエラーやソフトウェアのクラッシュなどの障害から復帰可能なようにチェックポイントの保存も必要である。以上の要求を満足するために、十分に高速な IO 処理を実現する必要がある。Oakforest-PACS 全系で計算可能な波動関数の規模を見積もると全体で約 4 TB で、同システムが提供するファイルシステムの理論データ転送速度（500 GB/s on Lustre, 1560 GB/s on BurstBuffer）では数秒で読み書き可能なサイズではあるが、MPI-IO を用いた全プロセスで 1 個のファイルとして読み書きする方式（以下、SSF: Single Shared File）ではこの性能を達成することは極めて困難である。

並列ファイルシステムの性能を最大限活用するためには各 MPI プロセスが独立に自身の担当領域を読み書きする方式（以下、FPP: File Per Process）が最適と考えられるが、SALMON では各計算において最適な性能を実現できる MPI プロセス割当ルールが異なる。したがって、計算ごとにプロセス割当を容易に変更できる柔軟性を持ち、かつ高速な IO 処理を提供可能なファイル構造が要求されている。

また、並列ファイルシステムの輻輳対策も必要となる。Lustre 上のファイルにアクセスする場合、実際にデータが保存されているオブジェクトストレージサーバ（OSS）のアドレスやデータサイズといったメタデータが保存されているメタデータサーバ（MDS）へアクセスし、メタデータを参照して対象ファイルが保存されている OSS へアクセス、IO 処理を行う。MDS および OSS は複数サーバによって構成されるが、対象ファイルのメタデータ保存先である MDS のアドレスは通常アクセス先のディレクトリによって決定される。つまり、複数のプロセスが同一ディレクトリ上の単一または複数ファイルへ同時にアクセスすると、特定の MDS へ問い合わせが集中し輻輳が発生、IO 性能の低下だけでなくファイルシステムの障害を引き起こしかねないため、ディレクトリを分散することで各 MDS の負荷を軽減しなければならない。FPP の場合、各 MPI プロセス専用のディレクトリを準備することで、全 MDS への負荷を均等にすることが可能であると考えられる。

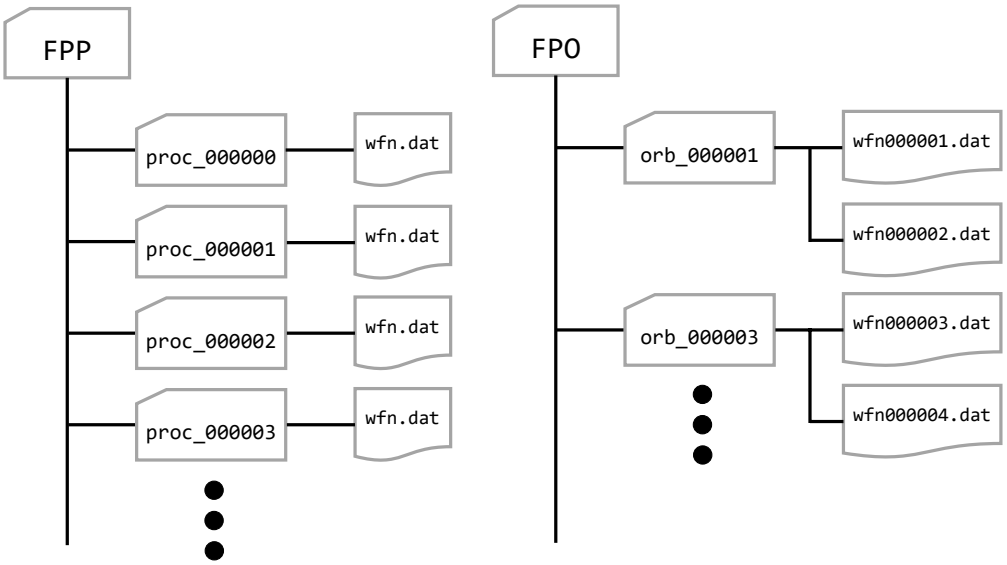

図 2: FPP および FPO の書き込みパターン

FPP（File Per Process）および FPO（File Per Orbital）の書き込みパターン、FPP では各プロセスが独立に専用のディレクトリにファイルの読み書きを行うのに対し、FPO は各軌道関数を計算するプロセス群が MPI-IO で読み書きを行う。その際のディレクトリ内の軌道関数ファイルの数は任意に決定可能。図の場合、2 を設定している。

アプリケーション特有の情報として、電子の波動関数として定義される配列は（X, Y, Z）の 3 次元実空間と、バンド理論によるバンド数および k 点数の 5 次元空間として定義される。このうち k 点は、非専門分野では各バンドのレプリカデータと解釈して良い。バンドそれぞれのデータについて、一般的に「軌道関数（Orbital Function)」と定義される。軌道関数単位でファイルを保存することで、MPI-IO 通信の局所化、ファイルシステムの輻輳軽減、バンド幅の有効活用を両立できるものと考える（以下、FPO: File Per Orbital)。軌道関数単位で取り回すことで、データの読み書きそのものもかなり単純化され、軌道関数配列の MPI_Datatype の定義、データタイプに基づいた Collective IO（MPI_File_read_all/MPI_File_write_all）通信を実行、

Collective IO を必要な軌道関数分だけ行うだけで良い。図 2 に FPP および FPO の模式図を示す。SALMON では FPP は障害復帰用のチェックポイント作成手段として実装し、通常利用時には SSF、大規模実行向けに FPO を実装している。FPO は、実空間の分割プロセス数によりアクセス負荷が変わることから、各ディレクトリ内に保存可能な軌道関数の数を任意に設定できる。ただし、我々の実装はディレクトリ生成に POSIX 定義の関数を利用しているため、POSIX を利用できない環境では動作しない。

この機能の実装により、富岳共用前評価環境上において、最高 12.3 TB の読み書き処理を 10 分以内に完了できることを確認した[2]。

４．Tofu-D ネットワークにおける MPI プロセス割当の最適化

Oakforest-PACS や筑波大学の Cygnus などは Intel Omni-Path や Mellanox InfiniBand を用いた Full-bisection Fat-tree ネットワークで構築されているため、ノードあたり 1 MPI プロセスを割り当てる場合、各 MPI プロセスが物理的な計算ノード群のどの位置にあり、通信相手となる計算ノードの距離を考慮することなく通信が可能であった。しかし富岳や FX1000 などの超大規模スーパーコンピュータでは Fat-tree の構築に必要なネットワークスイッチやケーブルの数といった実装コストの観点から、多次元メッシュトーラスネットワークが採用される。そのため、適切な MPI プロセスマッピングを考慮せずに実行すると、長い距離（ホップ数）となる通信が多発し、大幅な性能低下を招く可能性がある。富岳が採用する Tofu-D ネットワークは 6 次元メッシュトーラス形状で、一般ユーザからは 3 次元のノード集合として扱うことができる。

SALMON は、主に以下の 2 つの通信パターンに分けられる。

(1) 軌道関数**間**の通信: Collective (MPI_Bcast, MPI_Allreduce)

(2) 軌道関数**内**の通信: 袖領域交換、FFT 計算時の MPI_Alltoall

ここで重要なのは、基底状態計算では (1) が、実時間発展計算では (2) の通信が支配的で、他方の通信コストは negligible になる点である。SALMON は波動関数配列のすべての次元を MPI でプロセス分割できるため、支配的な通信を行うプロセス群が可能な限り連続した塊（以下、プロセスブロック）として計算ノードに割り当てられるようにすることで通信性能を最大化する必要がある。また富岳に搭載されている A64FX プロセッサは、12 コアを 1 NUMA 構成として 4 NUMA 構成を取り、1 NUMA ノードあたり 1 MPI プロセスでの実行が推奨されている。Tofu-D インターコネクトに比べて高速な NUMA ノード間通信を活用するためにも、プロセスの割当方法を工夫する必要がある。このとき k 点はレプリカのため、波動関数配列は 3 次元実空間 + 1 次元波数空間（バンド + k 点、電子軌道と呼ぶ）の 4 次元空間配列 (X, Y, Z, W) と解釈できる。対して、Tofu-D で構成されたネットワークは 3 次元 MPI プロセス空間として扱われるため、4 次元配列を 3 次元 MPI プロセス空間にマッピングする必要がある。(1) の場合は W 方向に対する通信が、(2) の場合は (X, Y, Z) 領域内で発生する通信コストが最小となるようにマッピングしなければならない。

[2] 我々が実験した際の富岳共用前評価環境において、並列ファイルシステムの整備状況に起因し、富岳が本来提供する計算ノードあたりの IO のバンド幅に比べ大幅に低い性能しか利用できなかったため、この結果でも良好と考えられる。

表 1: 各変数の説明

変数名	説明
$T_{x,y,z}$	Tofu-D ネットワーク上の各方向のノード数
P_{all}	全 MPI プロセス数
$P_{rx,ry,rz}$	実空間方向の MPI プロセス数
$P_{orbital}$	波数空間方向の MPI プロセス数
$P_{ox,oy,oz}$	$P_{orbital}$の因数
P_{node}	ノードあたりプロセス数（4 で固定）
$L_{x,y,z}$	実空間格子点数
N_{orb}	電子軌道数

ここで、(2) の場合のマッピング方法について考える。(2) の場合、(X, Y, Z) 領域を計算する 3 次元 MPI プロセスブロックを、合計で W 領域の MPI 並列数となるように 3 次元的に並べることにより、**擬似的な 4 次元 MPI プロセス空間を表現**し、(X, Y, Z) 領域内における通信コストを最小化する。以下、表 1 に示す各変数を用いて詳細を説明するが、各変数はそれぞれ以下の関係性を持つ。

$$(T_x \times P_{node}) \times T_y \times T_z = P_{all}$$
$$(T_x \times P_{node}) \div P_{rx} = P_{ox}$$
$$T_y \div P_{ry} = P_{oy}$$
$$T_z \div P_{rz} = P_{oz}$$
$$product(P_{ox}, P_{oy}, P_{oz}) = P_{orbital}$$
$$(T_x \times P_{node})\ mod\ P_{rx} = 0$$
$$T_y\ mod\ P_{ry} = 0$$
$$T_z\ mod\ P_{rz} = 0$$

また、一部の計算で FFTE[3]を利用し (X, Y, Z) 領域内で閉じた MPI 版 3 次元 FFT を利用するため、下記の条件も満たす必要がある。

$$L_x\ mod\ P_{ry} = 0$$
$$L_y\ mod\ P_{ry} = 0$$
$$L_y\ mod\ P_{rz} = 0$$
$$L_z\ mod\ P_{rz} = 0$$

まず、プロセス分割数を決定するユーザは、$P_{rx,ry,rz,orbital}$の 4 つを決定しなければならない。これにより必要な$T_{x,y,z}$および$P_{ox,oy,oz}$を上記式より決定できるが、$max(P_{ox,oy,oz})$が W 方向における最大の通信ホップ数となるため、これを最小に設定することが望ましい。ただしここで注意したいのは、導入されたシステムによって提供される 3 次元ノード形状の制約を受けるため、適切なパラメータの設定はシステムや問題規模に強く影響される。また$P_{rx} < P_{node}$の場合は、P_{ry}をノード内で連続に割り当てるといった設定も可能である。

[3] http://www.ffte.jp/

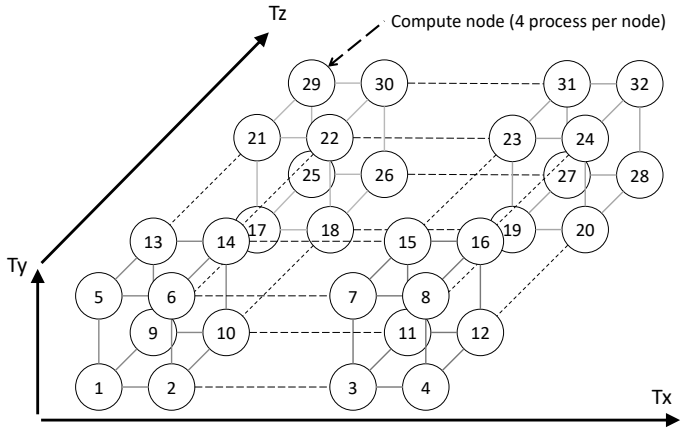

図 3: 3 次元ノード空間に対する 4 次元 MPI プロセスのマッピング

同図では合計 32 ノード 128 MPI プロセスにおけるマッピング例を示している。このとき、同色線で接続された 8 ノードが同じ（X, Y, Z）領域を計算する 3 次元ノードブロックで、同ブロックが 4 つ配置されている。すなわち、W 領域は 4 MPI プロセスで、（X, Y, Z）領域は 32 MPI プロセスで並列化されている。

ここで、マッピングイメージを図 3 に示すが、このとき、以下通り各変数が設定されている。実際にどのように並列化されているかは図 3 の説明に示す通り。

$$(T_x, T_y, T_z) = (4,2,4)$$
$$(P_{rx}, P_{ry}, P_{rz}) = (8,2,2)$$
$$(P_{ox}, P_{oy}, P_{oz}) = (2,1,2)$$
$$P_{orbital} = 4$$

以上の通り 4 次元 MPI プロセス空間を擬似的に作成することで、通信コストを最小化することが可能となる。今回、本節の冒頭で説明した（2）におけるマッピング方法を紹介したが、（1）の場合は条件式のP_{rx}と$P_{orbital}$を入れ替えた式に変形することで、最適な MPI プロセスマッピングを実現できる。

５．スーパーコンピュータ「富岳」での性能評価と現在の課題

我々の現在のシミュレーション目標は、電子動力学計算の時間ステップあたりの実計算時間を 1 秒以内（1 second/iteration）で実行することである。この目標値は、電子動力学計算で一般的に必要な時間ステップから、1 実験あたり 24 時間程度でシミュレーションを完了するために必要な計算速度として算出している。本節では弱スケーリング性能を紹介し、現在の課題について述べる。

表 2: 弱スケーリングにおける問題設定

# of node	$T_{x,y,z}$	$P_{rx,ry,rz,orbital}$	$L_{x,y,z}, N_{orb}$
432	(3, 6, 24)	(2, 2, 12, 36)	(108, 96, 480, 5328)
1,728	(6, 12, 24)	(2, 4, 12, 72)	(108, 192, 480, 10656)
6,912	(24, 12, 24)	(4, 4, 12, 144)	(216, 192, 480, 21312)
27,648	(48, 12, 48)	(4, 8, 12, 288)	(216, 384, 480, 42624)

今回、我々が対象とした問題はスーパーセル計算という複数のシングルセル（レプリカ）を結合した物理シミュレーションとなっている。同計算は、対象となる波動関数の実空間を 2 倍にしたとき、同時に電子軌道も 2 倍にする必要があり、Weak scaling では 4 倍の計算リソースでスケールしなければ正当な評価ができない。表 2 に、27,648 ノードまでの弱スケーリングにおける問題サイズ、3 次元ノード形状およびプロセス分割ルールを示す。Z 方向から長短パルスが入力されるため、Z 次元の格子点数は問題の性質上固定される。

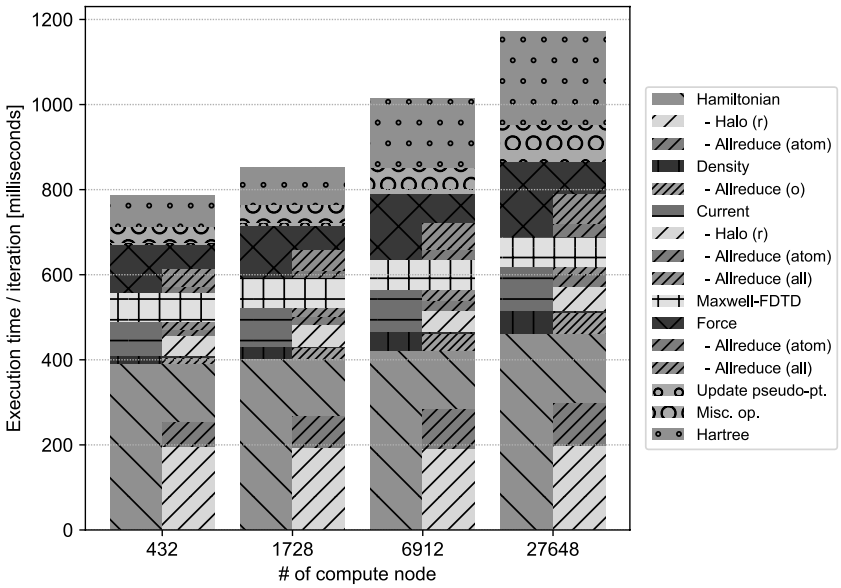

図 4: 弱スケーリング性能評価結果

縦軸は時間ステップあたりの計算時間［ミリ秒］で、横軸は計算ノード数である。図中の幅が半分になっている棒グラフは、各実行時間の中での通信種別および通信時間を示している。すなわち、通常幅の棒グラフから通信時間を差し引くと正味の計算時間と解釈できる。

図 4 に富岳共用前評価環境における弱スケーリングの結果を示す。この中で最下段グラフは支配的な計算である波動関数のハミルトニアンの実行時間で、通信時間が半分以上を占めているが MPI_Allreduce のみ増加が見られる。これは 3 次元空間上に存在する全原子に対する通信だが、原子数は弱スケーリング時に各ステップで 2 倍に増加し、通信時間も増加する傾向にある。対してハミルトニアンの実行時間の半分を占める袖領域交換は、ほぼ完全な状態でスケールしており、本稿で紹介した MPI プロセスマッピング手法が効果的に動作したことを示している。

ここで注目したいのは最上段グラフで、これは Hartree potential を 3 次元実空間から得た電子密度から計算する工程である。Hartree potential は陰的に解く必要があるが、SALMON では 3 次元実空間内での局所的 3 次元 FFT で解を求めている。FFT は MPI_Alltoall が必要な計算パターンのため、富岳のようなネットワークバンド幅が相対的に低いシステムでは性能を出しにくい欠点がある。ただし、SALMON における FFT 計算は、表 2 を参照すると $product\left(P_{rx,ry,rz}\right)/P_{node}$ が FFT の計算で使われるノード数となり、27,648 ノード構成であっても 96 ノードといった小規模

ノード群で閉じた並列計算である。もしこの性能劣化特性がキープされるとするならば、全系計算を行った場合、432 ノード実行に対して約 55%の実行効率、約 1.5 seconds/iteration となり当初の目標値を達成できない。したがって、FFT の代わりに前処理付き CG 法で解くなど、別の手法が必要と考えられる。

６．おわりに

本稿では、我々が開発する SALMON を用いた大規模電子動力学シミュレーションの実現のために、IO 処理の高速化と安定化、メッシュネットワークに対する多次元 MPI プロセスマッピング手法について紹介し、スーパーコンピュータ「富岳」の 1/6 に相当する 27,648 ノードまでの計算性能を紹介した。

計算性能については、Oakforest-PACS（Intel Xeon Phi）プロセッサにおける知見、最適化の成果を十分に活用でき、また Oakforest-PACS 全系を用いた実験により発覚した諸問題の解決、富岳共用前評価環境にて IO 性能と MPI プロセスマッピングを最適化することにより、富岳全系を用いた大規模電子動力学シミュレーションの実現に大幅に近づけたと考えられる。

今後の課題は、大規模化につれ局所的な FFT の MPI_Alltoall 通信が全体性能を押し下げているため FFT フリー計算の実現が挙げられる。そして、富岳の供用開始後にはその計算リソースを十分に活かし光科学の発展に寄与できるものと期待される。

謝　辞

本研究の内容の一部は、JCAHPC の運用する Oakforest-PACS を用いた大規模 HPC チャレンジの結果から得られた。同チャレンジがなければ富岳共用前評価環境における成果を得ることは困難であり、今回の機会をくださった JCAHPC に感謝する。

免　責　事　項

本研究の富岳共用前評価環境の利用における性能評価は、2020 年 4 月末までの利用結果であり、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。

参　考　文　献

[1] M. Noda, *et.al*: SALMON: Scalable Ab-initio Light-Matter simulator for Optics and Nanoscience, Computer Physics Communications. Volume 235, 356 (2019). https://doi.org/10.1016/j.cpc.2018.09.018

[2] Y. Hirokawa, *et.al*: Performance Optimization and Evaluation of Scalable Optoelectronics Application on Large Scale KNL Cluster, Proceedings of ISC2018 (2018). https://doi.org/10.1007/978-3-319-92040-5_11

[3] 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). https://doi.org/10.15748/jasse.1.98

[4] M. Noda, *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). https://doi.org/10.1016/j.jcp.2014.02.006

[5] Y. Hasegawa, *et.al*: Firstprinciples Calculations of Electron States of a Silicon Nanowire with 100,000 Atoms on the K Computer, Proceedings of SC11 (2011). https://doi.org/10.1145/2063384.2063386