［大学 ICT 推進協議会 2016 年度年次大会論文集より］

『銅酸化物の有効モデルに対する揺らぎ交換近似』コードの SX-ACE 向け最適化

山下 毅 [1)], 山崎 国人 [2)], 江川 隆輔 [3,4)],
吉岡 匠哉 [2)], 土浦 宏紀 [2)], 小林 広明 [4,3)], 曽根 秀昭 [3,4)]

1) 東北大学 情報部情報基盤課, 2) 東北大学 大学院工学研究科,
3) 東北大学 サイバーサイエンスセンター, 4) 東北大学 大学院情報科学研究科

yamacta@tohoku.ac.jp

Performance Optimization of "Fluctuation exchange approximation for an effective model for cuprates" code for SX-ACE .

Takeshi Yamashita[1)], Kunito Yamazaki[2)], Ryusuke Egawa[3,4)],
Takuya Yoshioka[2)],Hiroki Tsuchiura[2)],Hiroaki Kobayashi[4,3)], Hideaki Sone[3,4)]

1) Information Infrastructure Division, Information Department, Tohoku Univ.
2) Graduate School of Engineering, Tohoku Univ.
3) Cyberscience Center, Tohoku Univ.
4) Graduate School of Information Sciences, Tohoku Univ.

概要

FFT を用いた数値演算アプリケーションでは、Windows や Linux システム上で実行出来る FFTW ライブラリが広く利用されている。従来 SX システムでは標準で FFTW ライブラリ、もしくはそのインターフェースは提供されておらず、このようなアプリケーションを SX-ACE 向け実行オブジェクトとするためには、ASL ライブラリへのコード修正、もしくは FFTW のソースコードを SX-ACE 向けにコンパイルしてライブラリ化することが必要である。本稿では FFTW ライブラリを利用したアプリケーションを ASL Wrapper を用いて SX-ACE 向けにコンパイルする方法と、SX-ACE における高性能演算の実現を目的とした最適化手法について示す。

1 はじめに

東北大学サイバーサイエンスセンター（以下、本センター）の大規模科学計算システムは、日本電気株式会社（以下、NEC）製ベクトル型スーパーコンピュータ SX-ACE を主力計算機とし、汎用アプリケーションの実行環境として NEC 製スカラ型の並列コンピュータ LX406-Re2 の運用をしている。異なる特性を有する二種類の計算機の運用により、利用者の幅広いニーズに応えるサービスを提供している。表 1 にこれらシステムの諸元を示す。

また本センターでは 1999 年より、ユーザアプリケーションの高精度化、大規模化の支援を目的とした共同研究制度を実施している。利用者、計算機科学を専門とするセンター教員、技術職員、およびベンダー技術者が連携してアプリケーションの高速化に取り組んでいる。図 1 に 1999 年から本センターで取り組んでいるセンター独自の共同研究、学際大規模情報基盤共同利用・共同研究拠点（JHPCN）課題および革新的ハイパフォーマンス・コンピューティング・インフラ（HPCI）課題採択数の推移を示す。本センター独自の共同研究は恒常的に年 10 課題ほど実施されていることに加え、近年では JHPCN、HPCI を介した共同研究数が増加している。これは、センターの共同研究を通してユーザアプリケーションが高度化・大規模化し、JHPCN、HPCI 採択課題へとステップアップしており、我々の継続的な高速化支援活動が一定の成果を上げていることが分かる。

以下では高速化支援活動の一例として、『銅酸化物の有効モデルに対する揺らぎ交換近似』コードの、SX-ACE 向け最適化事例について報告する。

表 1　サイバーサイエンスセンター大規模科学計算システム諸元

	性能	SX-ACE	LX406Re-2
CPU 性能	名称	NEC ベクトルプロセッサ	Intel(R) Xeon(R) E5-2695v2
	コア数	4 個	12 個
	理論最大演算性能	276GFLOPS	230GFLOPS
	最大ベクトル演算性能	256GFLOPS	-
	メモリバンド幅	256GB/sec	4.9GB/sec
	ADB / Cache	1MB/コア	30MB (L3)
ノード性能	CPU 数	1 個	2 個
	理論最大演算性能	276GFLOPS	460GFLOPS
	最大ベクトル演算性能	256GFLOPS	-
	メモリ容量	64GB	128GB
	メモリバンド幅	256GB/sec	9.9GB/sec
	ノード間通信速度	8GB/sec	7GB/sec
システム性能	CPU 数	2,560 個	136 個
	理論最大演算性能	706.6TFLOPS	31.3TFLOPS
	最大ベクトル演算性能	655.4TFLOPS	-
	メモリ容量	160TB	8.5TB

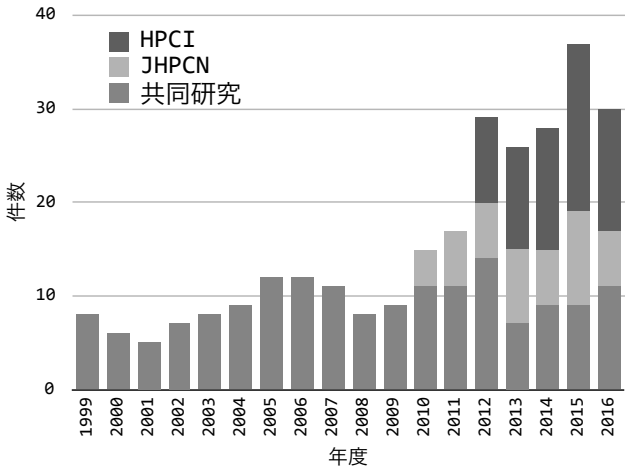

図 1　共同研究課題採択数

2　アプリケーションの概要

今回 SX-ACE での最適化の対象としたアプリケーションコードは、東北大学大学院工学研究科応用物理学専攻で開発中の『銅酸化物の有効モデルに対する揺らぎ交換近似』コードである。本コードは Fortran90 で記述され、コメント行を含み約 950 行からなり、特有のシステム向けの最適化は施されていない。

2.1　揺らぎ交換近似手法

今回最適化の対象とするコードは、銅酸化物超伝導体のうち特に電子ドープ型と呼ばれる物質の電子状態を記述するために提案された有効 2 バンド模型において、超伝導および磁気秩序転移を解析するアプリケーションである。有効 2 バンド模型に含まれる電子間相互作用を揺らぎ交換近似（fluctuation-exchange approximation）を用いて取り扱い、その結果得られた線型化 Elisashberg 方程式を解くことによって超伝導転移温度等が求められる。このアプリケーションにより、電子ドープ型銅酸化物超伝導体のアンダードープ領域で新たに超伝導状態が出現し得るという近年の実験的報告 [1] について、理論的観点からの解析を与えることができる。

2.2　コード概要

揺らぎ交換近似を用いた線形化 Eliashberg 方程式を解く際に、畳み込み積分を行う必要がある。コードのアルゴリズムにおいては、$N_x \times N_y \times N_\omega$ の 3 次元配列に対して 3 次元 FFT を行うことでこの畳み込み積分を高速化する。N_x, N_y, N_ω はそれぞれ 2 次元実空間および虚時間軸における解像度を定めるパラメータである。超伝導状態の研究において重要である低温での解析を行うためには、より高いエネルギー解像度が求められるため、これらのパラメータをより大きくとる必要がある。また、温度変化を解析するために、200K〜50K まで 10K 刻みに計 16 回の反復実行が行われる。3 次元 FFT の実行には、FFTW ライブラリを、また固有値の求解には、Lapack ライブラリを使用する。

現状のモデルサイズは $N_x = N_y = 64, N_\omega = 2,048$ であり、研究室のサーバ（Intel® Xeon® CPU E5-2630 v2）では計算に約 7 時間を要している。今後は低温領域における秩序状態を詳細に解析するために、各次元のサイズを N_x, N_y については 30 倍以上、N_ω については 10 倍程度拡張する必要があり、より大容量の物理メモリと計算時間の短縮が必要となる。

2.3 使用ライブラリについて

FFTW ライブラリ [2] は、任意の基数の実数および複素数配列の FFT を行うライブラリで、GNU ライセンスに基づくフリーソフトウェアである。FFTW ライブラリは商用アプリケーションにも採用されるほど幅広く利用されている FFT ライブラリである。また開発も継続的に行われており、年に数度のマイナーアップデートが行われている。また SIMD 命令のサポートなど、Intel® 系 CPU および Intel® コンパイラ向けの最適化がソースコードになされている。

LX406-Re2 では Intel®CPU に最適化された Intel®MKL ライブラリから FFTW と Lapack が利用できるため、コンパイル時に FFTW のインクルードファイルの場所と MKL ライブラリをリンクする指定を行うのみで実行オブジェクトの作成が可能である。

一方 SX-ACE では、FFTW ライブラリやそのインターフェースは標準では提供されていないため、FFTW ライブラリを利用できない。SX-ACE で高性能演算を実現するためには、ベクトルチューニングされた NEC 製の ASL（Advanced Scientific Library）を FFT ライブラリとして利用する必要がある。そこで後述の ASL Wrapper を導入し、コード中の FFTW ライブラリをコールする箇所を修正することなく ASL の FFT ライブラリを利用することとした。

3 ASL Wrapper

3.1 概要

ASL Wrapper は FFTW や Intel(R) MKL の FFT ライブラリを用いたプログラムを、SX 上で利用するためのインターフェースである。ASL Wrapper は Xevolver プロジェクト [3] で開発され、現在 v1.0r1.2 が公開されている。ASL Wrapper を用いることで、FFTW あるいは Intel(R) MKL の FFT ライブラリを使用するアプリケーションを SX-ACE へ移植することが容易になるとともに、SX 向けに高度にチューニングされた ASL ライブラリを利用することで大きな性能向上が期待できる。

ASL ライブラリでは複素引数型の 3 次元複素フーリエ変換を行う場合、サブルーチン ZFC3FB を利用する。このサブルーチンも FFTW と同じく任意基数に対応した FFT で、データ数を調整できる場合には、配列の各サイズを 2,3,5 の倍数となるように設定した方が効率よい計算を行える。また SX では主記憶のバンクコンフリクトを避けるために、配列の整合寸法を奇数に設定するのが望ましいが、現在の ASL Wrapper では整合寸法を別途指定する方法はないので、配列サイズの 1 次元目を奇数となるよう N のサイズを 65 に設定しバンクコンフリクトの回避を試行した。

3.2 使用方法

ASL Wrapper を利用するには、開発元のウェブページ [4] で提供される tar ファイルを入手して利用者のホームディレクトリ配下に展開し、アプリケーションのコンパイル時にインクルードファイルのパス指定、およびリンク時にアーカイブファイルのパス指定とライブラリ名の指定を行う。Fortran コードのコンパイル時には、-Ep オプションにより C プリプロセッサを起動する必要がある。

4 SX-ACE への移植と性能比較

4.1 コードの移植方法

ASL.V1.0_r1.2.tar をダウンロードした後、ホームディレクトリ配下 ($HOME/ASL_Wrapper/) に展開し、SX-ACE 用に移植を行うアプリケーションのコンパイルとリンクを行う。今回のアプリケーションの場合は Lapack ライブラリも必要であるため、リスト 1 に示すコマンドでコンパイルとリンクを行い、実行オブジェクトの作成を行った。

リスト 1 ASL Wrapper のリンク方法

```
1 sxf90 -Ep -I$HOME/ASL_Wrapper/FFTW/include \
2    source.f90 -L$HOME/ASL_Wrapper/FFTW/lib \
3    -lfftw2asl -lasl -llapack -lblas
```

4.2 FFT ライブラリの性能比較

本センターの LX406Re-2 と SX-ACE でベクトルチューニングを行う前のアプリケーションコード（以下、初期コード）の演算時間の比較を行った。それぞれ 1 コアでの逐次実行である。なお性能比較を行うコードは演算時間短縮のため、単一の温度（iTm=200）のみの計算とした。

LX406Re-2 では Intell®MKL ライブラリを用いて FFTW ライブラリおよび Lapack ライブラリのリンクを行った。SX-ACE では ASL Wrapper を用いた場合と、FFTW ライブラリのソースコードを FORTRAN90/SX コンパイラによりコンパイルして作成したライブラリを用いた場合との比較を行った。Lapack ライブラリは SX 向けに最適化された Math-Keisan ライブラリを用いた。それぞれの演算時間を表 2 に示す。ASL Wrapper を用いて SX-ACE で実行した場合が最も演算時間が短く、FFTW ライブラリのソースコードをコンパイルした用いた場合の約 2.5 倍の性能となった。SX 向けに最適化された ASL ライ

表 2　初期コードの演算時間比較

FFT ライブラリ・計算機	演算時間 [sec]
FFTW (Inlel MKL) on LX406Re-2	1,036
FFTW using ASL Wrapper on SX-ACE	707
FFTW compiled by FORTRAN90/SX on SX-ACE	1,830

ブラリの効果が見られ、ASL Wrapper を用いて ASL ライブラリを使用する有用性が確認できた。

次節では、本コードの SX-ACE 向け最適化について説明する。

5　ベクトルチューニングと性能比較

5.1　簡易性能解析機能によるコスト調査

コード実行時に性能解析情報の取得を行う簡易性能解析（FTRACE）機能を利用して、演算時間のコスト分析を行う。FTRACE 機能を利用するにはソースコードのコンパイル時とリンク時に-ftrace を指定する。FTRACE 機能の一つであるユーザー指定リージョンを利用することで、コードの任意の範囲で性能解析情報の取得が可能である。ASL 等の外部ライブラリは FTRACE 機能のみでは性能解析情報に表示されないが、ライブラリ呼び出し前後でユーザー指定リージョンを利用することで、外部ライブラリの性能解析情報も得ることができる。

5.2　配列宣言方法の変更によるループ融合

リスト 2 に初期コードの編集リストの一部を示す。ソースコードのコンパイル時の編集リストを出力するためにはコンパイル時に-R5 を指定する。ユーザ指定リージョン REGION A を 2 箇所の 3 重ループの演算が行われる箇所に設定した。

内側の 2 重ループの演算（4、16 行目）は、DO ループにより 1 次元目、2 次元目とも定義された配列サイズの最初から最後まで参照されるため、コンパイラが自動的に 2 重ループの演算を 1 重ループの演算に一重化するはずであるが、編集リストでは最内ループ（5 行目および 17 行目）のみベクトル化されたことが分かる（V マークがベクトル化されたループを示す）。これはリスト 3 に示すように、演算で用いられる配列の宣言が complex 文と pointer 文により行われるため、コンパイラが配列サイズと参照範囲が同一である判断ができないためである。

そこでリスト 4 に示すように、配列の宣言を通常の complex 文のみで行い、また実行中に配列サイズに変更がないことを確認し、静的なサイズで配列の確保を行った。

配列宣言の変更後の編集リストをリスト 5 に示す。最内ループに*記号と 2 番目のループに W 記号が付き、コンパイラにより 2 重ループが一重化されたことが分かる。コンパイラにより一重化された箇所は、REGION A の 2 箇所を含む計 30 箇所あった。

リスト 2　REGION A（初期コード）

```
1  |||  CALL FTRACE_REGION_BEGIN('REGION A')
2  |||+---> do iw=0,Wm-1
3  ||||         w=dble(2*iw+1)*pi*Tm
4  ||||+--->    do iky=0,N-1
5  |||||V->       do ikx=0,N-1
6  ||||||  A        Gdd(ikx,iky,iw)=(im*w-Ep(ikx,iky))&
7  ||||||            /((im*w-Ed-selfs3(ikx,iky,iw)) &
8  ||||||            *(im*w-Ep(ikx,iky))-V(ikx,iky))&
9  ||||||            -1.0d0/(im*w-gza)
10 |||||V-        end do
11 ||||+--      end do
12 |||+---  end do
13 |||
14 |||+---> do iw=0,Wm-1
15 ||||         w=dble(2*iw+1)*pi*Tm
16 ||||+--->    do iky=0,N-1
17 |||||V->       do ikx=0,N-1
18 ||||||  A        Gpp(ikx,iky,iw)=(im*w-Ed &
19 ||||||            -selfs3(ikx,iky,iw)) &
20 ||||||            /((im*w-Ed-selfs3(ikx,iky,iw)) &
21 ||||||            *(im*w-Ep(ikx,iky))-V(ikx,iky))&
22 ||||||            -1.0d0/(im*w-gza)
23 |||||V-        end do
24 ||||+--      end do
25 |||+---  end do
26 |||  CALL FTRACE_REGION_END('REGION A')
```

リスト 3　complex，pointer 文による配列宣言

```
1  integer,parameter ::N=64,Wm=2048
2  complex(kind(0d0)),pointer::Gdd(:,:,:),Gpp(:,:,:)
3  complex(kind(0d0)),pointer::selfs3(:,:,:)
4  allocate(Gdd(0:N-1,0:N-1,0:Wm-1))
5  allocate(Gpp(0:N-1,0:N-1,0:Wm-1))
6  allocate(selfs3(0:N-1,0:N-1,0:Wm-1))
```

リスト 4　complex 文のみによる配列宣言

```
1  integer,parameter ::N=64,Wm=2048
2  complex(kind(0d0))::Gdd(0:N-1,0:N-1,0:Wm-1)
3  complex(kind(0d0))::Gpp(0:N-1,0:N-1,0:Wm-1)
4  complex(kind(0d0))::selfs3(0:N-1,0:N-1,0:Wm-1)
```

リスト 5　REGION A（ループ一重化後）

```
1  |||  CALL FTRACE_REGION_BEGIN('REGION A')
2  |||+---> do iw=0,Wm-1
3  ||||         w=dble(2*iw+1)*pi*Tm
4  ||||W--->    do iky=0,N-1
5  |||||*->       do ikx=0,N-1
6  ||||||  A        Gdd(ikx,iky,iw)=(im*w-Ep(ikx,iky))&
7  ||||||            /((im*w-Ed-selfs3(ikx,iky,iw)) &
8  ||||||            *(im*w-Ep(ikx,iky))-V(ikx,iky))&
9  ||||||            -1.0d0/(im*w-gza)
10 |||||*-        end do
11 ||||W--      end do
12 |||+---  end do
13 |||
14 |||+---> do iw=0,Wm-1
15 ||||         w=dble(2*iw+1)*pi*Tm
16 ||||W--->    do iky=0,N-1
17 |||||*->       do ikx=0,N-1
18 ||||||  A        Gpp(ikx,iky,iw)=(im*w-Ed &
19 ||||||            -selfs3(ikx,iky,iw)) &
20 ||||||            /((im*w-Ed-selfs3(ikx,iky,iw)) &
21 ||||||            *(im*w-Ep(ikx,iky))-V(ikx,iky))&
22 ||||||            -1.0d0/(im*w-gza)
23 |||||*-        end do
24 ||||W--      end do
25 |||+---  end do
26 |||  CALL FTRACE_REGION_END('REGION A')
```

5.3 コード修正によるループ融合

REGION A で示した 2 箇所の 3 重ループの演算箇所は、3 次元目のループ（2 行目および 14 行目）が同範囲の演算を行い、3 行目および 15 行目のスカラ演算も共通の演算を行うので、3 次元目のループを融合することで、ベクトル演算命令のオーバヘッドを削減することが可能である。コンパイラオプションで最適化レベルを最高の-Chopt、もしくは明示的にループ融合を行うオプション-pvctl loopfusion を指定することができるが、この箇所はコンパイルオプションではループ融合されなかったため、ソースコードの一部をコメントアウトすることでループ融合を促進した。リスト 6 にソースコードの修正により 3 次元目のループが融合された編集リストを示す。

リスト 6　REGION A（ループ融合後）

```
1  |||  CALL FTRACE_REGION_BEGIN('REGION A')
2  |||+---> do iw=0,Wm-1
3  ||||         w=dble(2*iw+1)*pi*Tm
4  ||||W-->     do iky=0,N-1
5  |||||*->       do ikx=0,N-1
6  ||||||  A        Gdd(ikx,iky,iw)=(im*w-Ep(ikx,iky))&
7  ||||||            /((im*w-Ed-selfs3(ikx,iky,iw)) &
8  ||||||            *(im*w-Ep(ikx,iky))-V(ikx,iky))&
9  ||||||            -1.0d0/(im*w-gza)
10 |||||| !       end do
11 |||||| !     end do
12 |||||| ! end do
13 ||||||
14 |||||| ! do iw=0,Wm-1
15 |||||| !     w=dble(2*iw+1)*pi*Tm
16 |||||| !     do iky=0,N-1
17 |||||| !       do ikx=0,N-1
18 ||||||  A        Gpp(ikx,iky,iw)=(im*w-Ed &
19 ||||||            -selfs3(ikx,iky,iw)) &
20 ||||||            /((im*w-Ed-selfs3(ikx,iky,iw)) &
21 ||||||            *(im*w-Ep(ikx,iky))-V(ikx,iky))&
22 ||||||            -1.0d0/(im*w-gza)
23 |||||*-        end do
24 ||||W--      end do
25 |||+---   end do
26 |||  CALL FTRACE_REGION_END('REGION A')
```

2 箇所の 3 重ループの演算部分が 3 次元目のループで融合され、2 行目から 25 行目が 1 つのループとなったことが分かる。

以上 2 種類のベクトルチューニングを行ったコードと、初期コードを実行した際の FTRACE 情報を表 3 に示す。上段からそれぞれ、初期コード（A-1)、ループ一重化後（A-2)、さらにループ融合後（A-3）の REGION A の性能解析情報である。

ループ一重化後（A-2）は、初期コード（A-1）と比較して FLOPS 値が約 2.8 倍向上した。また平均ベクトル長が 64 から 256 に増加した。これは初期コードでは最内ループの繰り返し数が N=64 であったものが、2 重ループの一重化により 64 × 64=4,096 となり最大ベクトル長の 256 よりも十分長くなったことの結果である。また、バンクコンフリクト（NETWORK)の値は約 32 秒から 2 秒に減少している。これは、ベクトルプロセッサではベクトル長が長いほど同じ演算量を処理する際のメモリアクセスが効率化されるためである。結果として FLOPS 値は約 2.8 倍向上した。

ループ融合後（A-3）はベクトル演算命令のオーバヘッド削減により、さらに FLOPS 値が 4% 程度向上した。

5.4 基数サイズの変更によるバンクコンフリクトの削減

前述の通り ASL ライブラリの FFT では、基数のサイズが偶数（特に 2 の冪乗）のときにバンクコンフリクトが大きくなるので、N のサイズを 64 から 65 に変更し、リスト 7 に示した FFT 演算部分の REGION B の範囲について性能解析を行った。

2 行目から 5 行目において FFTW ライブラリをコールする箇所では、ASL Wrapper により実際は ASL ライブラリがコールされる。

リスト 7　REGION B

```
1 || CALL FTRACE_REGION_BEGIN('REGION B')
2 || call dfftw_plan_dft_3d(p,N,N,Wm,Gdd,Gdd, &
3 ||           FFTW_FORWARD,FFTW_ESTIMATE)
4 || call dfftw_execute(p)
5 || call dfftw_destroy_plan(p)
6 || CALL FTRACE_REGION_END('REGION B')
```

表 4 に N=64 で実行したとき（B-1）と N=65 で実行したとき（B-2）の FTRACE 情報を示す。基数のサイズが $64 = 2^6$ の場合はバンクコンフリクトの値が大きく、実行効率は 1% 程度であるが、基数のサイズが 65 の場合はバンクコンフリクトの値が約 1/30 に短縮され、実行効率は 28% まで向上している。FFT による演算箇所は REGION B の箇所を含む計 9 箇所あり、この高速化の効果は大きい。

5.5 初期コードとチューニング後コードの性能比較

初期コードと 2 種類のベクトルチューニングおよび N = 65 で実行した際の性能比較を表 5 に示す。各項目の値は、Program Information からの抜粋である。

オリジナルコードでは実行効率は 4% 程度であったが、ベクトルチューニングと基数サイズの変更によるバンクコンフリクトの削減で、実行効率を約 30% まで改善することができた。特に FFT ライブラリを使用する際の基数サイズの最適化は、ベクトルプロセッサの性能に大きく影響することが分かる。

次にベクトルチューニング後のコードを表 2 と同じ環境で実行した結果を表 6 に示す。速度向上比は表 2 の各演算時間に対する比である。ASL Wrapper を使用して SX-ACE で実行した場合の速度向上比は約 7 倍であった。研究室のサーバでの実行時間と比較し、約 6.5% の実行時間であった。

表 3　FTRACE REGION A の性能比較

PROC.NAME	FREQUENCY	EXCLUSIVE TIME[sec](%)	AVER.TIME [msec]	MOPS	MFLOPS	V.OP RATIO	AVER. V.LEN	VECTOR TIME	I-CACHE MISS	O-CACHE MISS	BANK CONFLICT CPU PORT	BANK CONFLICT NETWORK	ADB HIT ELEM.%
REGION A-1	2893	81.907(11.6)	28.312	22048.5	13630.6	99.44	64.0	81.907	0.001	0.001	0.000	31.853	54.92
REGION A-2	2893	29.389(4.7)	10.159	61210.9	37988.7	99.83	256.0	29.388	0.000	0.000	0.000	1.707	49.99
REGION A-3	2893	20.193(3.3)	6.980	67385.5	39661.6	99.87	256.0	20.193	0.000	0.000	0.000	0.854	49.95

表 4　FTRACE REGION B の性能比較

PROC.NAME	FREQUENCY	EXCLUSIVE TIME[sec](%)	AVER.TIME [msec]	MOPS	MFLOPS	V.OP RATIO	AVER. V.LEN	VECTOR TIME	I-CACHE MISS	O-CACHE MISS	BANK CONFLICT CPU PORT	BANK CONFLICT NETWORK	ADB HIT ELEM.%
REGION B-1	142	138.112(22.4)	972.621	1408.9	659.4	98.56	213.6	137.976	0.001	0.051	2.425	112.758	33.33
REGION B-2	142	7.998(8.1)	56.725	43620.8	17917.9	98.37	230.1	7.895	0.001	0.038	0.177	2.123	70.10

表 5　初期コードとチューニング後コードの性能比較

Program Information より抜粋	演算時間 [sec]	GFLOPS	実行効率 [%]	平均ベクトル長	平均ベクトル化率 [%]	バンクコンフリクト (CPU PORT) [sec]	バンクコンフリクト (NETWORK) [sec]
オリジナルコード	707.18	2.69	4.20	77.24	98.68	9.99	495.47
ベクトルチューニング後	614.98	2.55	3.99	194.56	99.26	9.71	459.71
N=65に変更後	99.01	18.66	29.16	192.30	98.95	1.24	14.77

表 6　チューニング後コードの演算時間比較 (N=65)

FFT ライブラリ ・計算機	演算時間 [sec]	速度向上比 [倍]
FFTW (Inlel MKL) on LX406Re-2	986	1.05
FFTW using ASL Wrapper on SX-ACE	99.6	7.14
FFTW compiled by FORTRAN90/SX on SX-ACE	1,660	1.10

また、LX406Re-2 での実行においても、初期コードとほぼ同程度の性能が担保され、SX-ACE 向けベクトルチューニングによる性能低下は見られなかった。

5.6　目標モデルサイズでの実行環境推定

SX-ACE において $N_x = N_y = 65, N_\omega = 2,048$ のパラメータ値で実行した際のメモリ使用量は約 2.4GB であった。コード中で使用されている配列サイズから算出し、今後のシミュレーションにおける目標モデルサイズとなる $N_x = N_y = 2,000, N_\omega = 20,480$ でのメモリ使用量は 15TB と推定される。SX-ACE では 256 ノードの利用でこのメモリ量を使用可能であるので、今後は MPI ライブラリを用いたコードの並列化を実施し、引き続き大規模並列での実行を支援する予定である。

6　まとめ

本稿では『銅酸化物の有効モデルに対する揺らぎ交換近似』コードの SX-ACE 向け最適化手法について述べた。ASL Wrapper を用いることで、FFTW ライブラリの記述部分を変更することなく SX-ACE でのコンパイルと実行が可能であり、また ASL の利用により高い実行効率を得ることができた。今回実施した最適化により、SX-ACE において約 30% の実行効率での演算を可能とした。今後は将来の問題規模拡大に対応するべく、大規模並列での実行を見据えて MPI 並列化の支援を行う予定である。

サイバーサイエンスセンターの大規模科学計算システムが、利用者の研究発展の一助になるために、今後も高速化推進研究活動に取り組む所存である。

参考文献

[1] 足立匡、小池洋二、Nd2CuO4 構造 (T' 構造) を有する電子型銅酸化物における還元処理による電子状態の変化とノンドープ超伝導のメカニズム、固体物理 vol.49、No.5、333-344、2014.
[2] FFTW、http://www.fftw.org
[3] Xevolver、http://xev.arch.is.tohoku.ac.jp/
[4] ASL Wrapper、http://xev.arch.is.tohoku.ac.jp/software/