Vol.2013-HPC-140 No.24
2013/8/1

マルチプラットフォームにおける最適化手法の効果に関する一検討

小松 一彦[1,4,a)]　佐々木 俊英[2]　江川 隆輔[1,4]　滝沢 寛之[3]　小林 広明[1]

概要：近年，HPC システムの多様化が進んでおり，特徴の異なる複数種類の HPC システムにおいて高い性能を引き出すことができる，性能可搬性の高い HPC コードの開発が強く求められている．本研究では，各種 HPC システム向けの最適化手法が HPC コードの性能に与える効果を詳細に解析し，その知見に基づいて性能可搬性の高い HPC コードを開発することを目的としている．本報告では，異なる手動最適化同士や自動最適化を組み合わせた場合の HPC コードの性能可搬性を解析する．HPC システムごとに，それぞれの手動最適化同士や自動最適化の組み合わせによる相乗効果を評価し，性能可搬性の低下を引き起こす可能性のある最適化について議論する．

キーワード：性能可搬性, 最適化手法

1. はじめに

近年，ベクトルプロセッサ，スカラプロセッサ，アクセラレータなど，特徴の異なる様々なプロセッサを搭載する大規模計算システム (HPC システム) が開発されている [1], [2], [3], [4] ．各 HPC システムで得意とする計算が異なるため，HPC コードの持つ特徴に依存して実効性能が HPC システム間で大きく異なる．HPC コードは，対象とする計算自体に由来する特徴を持つだけでなく，性能改善のために施された各種最適化手法からも特徴付けられる．ある特定の HPC システムを念頭に置いて行われた最適化が，他の HPC システムにおける性能も改善するとは限らず，逆に性能を低下させる恐れすらある．

このような背景の中，近年，様々な HPC システムにおいて，高い性能を引き出すことができる性質である性能可搬性が注目されている [5], [6]．性能可搬性の高い HPC コードを開発するためには，まず特定の HPC システムの性能のみを改善し，性能可搬性を低下させてしまう手動での最適化を回避する必要がある．本研究グループでは，性能可搬性の高い HPC コードの開発を目的として，様々な HPC システムを用いた評価を通じて，避けるべき最適化などのデータベースを経験則に基づき作成している．

本報告では，複数の手動最適化同士や自動最適化を組み合わせた場合の HPC コードの性能可搬性を明らかにする．HPC システムごとに，それぞれの最適化と組み合わせによる相乗効果を評価し，性能可搬性の低下を引き起こす可能性のある最適化について調査する．

ループ交換，多重ループの一重化，ループ内不変量コードの移動，マスクによるベクトル・SIMD 化の 4 種類の最適化 [7], [8] を取り上げ，まず，複数の手動最適化を組み合わせた場合の相乗効果について調査する．次に，複数の最適化とコンパイラによる自動最適化を組み合わせた場合の相乗効果について調査を行う．

2. 実アプリケーションに用いられる最適化の性能可搬性

我々の研究グループは，個別の最適化が HPC コードの性能可搬性に与える効果について，これまで調査してきた [9]．東北大学サイバーサイエンスセンターのスーパーコンピュータシステム上で開発が進められている海洋シミュレーション，飛行機周りの流体解析シミュレーション，ナノプラズマの生成シミュレーション，地震波の伝搬シミュレーションの 4 つの HPC コードに対して，一時変数の利用，ループ分散，未定義変数の利用，条件文のループ外への移動，ループ内不変量コードの移動といった最適化手法

[1] 東北大学サイバーサイエンスセンター
Cyberscience Center, Tohoku University
[2] 東北大学工学部
Graduate School of Information Sciences, Tohoku University
[3] 東北大学大学院情報科学研究科
Graduate School of Information Sciences, Tohoku University
[4] 科学技術振興機構戦略的創造研究推進事業
Japan Science and Technology Agency, Core Research for Evolutional Science and Technology
[a)] komatsu@isc.tohoku.ac.jp

表 1 HPC システムのノード諸元
Table 1 Node specifications of multiple HPC systems

HPC System	Sockets	Cores/socket	On-chip Memory	Compiler and options
NEC SX-9	16	1	256 KB ADB	sxf90 -Popenmp -ftrace (Rev.460)
Intel Nehalem EX	4	8	24 MB shared L3	ifort -openmp (12.1.0 20110811)
Fujitsu FX1	1	4	6 MB shared L2	frt -KOMP (Version 8.1)
Fujitsu FX10	1	16	12 MB shared L2	frtpx -Kopenmp (Version 1.2.1)
Hitachi SR16000 M1	4	8	32 MB shared L3	f90 -64 -model=M1 -omp (Hitachi)

の効果が異なる HPC システムで評価し，個別の最適化が HPC コードの性能可搬性に与える影響について，分析・評価を行っている [9].

条件文のループ外への移動とループ内不変量コードの移動による最適化では，条件分岐や冗長な演算を削減することにより，いずれの HPC システムにおいても実行時間を大きく削減し，性能可搬性を高めることが示されている．また，一時変数の利用や未定義変数の削除による最適化は，一部のシステムにおいて実行時間を短縮するが，その他のシステムでは実行時間がほとんど変化しない．これらの最適化によって，実効性能の低下する HPC システムが見られないことから，これらは性能可搬性への影響が少ない最適化手法であるといえる．

ループ分散による最適化は，一部の HPC システムでは実行時間の短縮が得られるが，逆に実行時間が増加する HPC システムも見られる．これは HPC システムに応じてループを分割する場所が異なるため，特定の HPC システムに合わせた最適化が，他のシステムに適しているとは限らないからである．このように，一部の HPC システムで実効性能を高めるが，他の HPC システムにおいて実効性能を低下させる最適化は，性能可搬性を低下させうる最適化であることが示されている．

以上のように，これまで典型的なループ最適化技法を個別に適用した場合の性能可搬性への影響を調査してきた．しかし，個々の最適化手法は性能可搬性を低下させない場合でも，それらを組み合わせた場合に性能可搬性を大きく低下させる事例がある．このため本報告では，HPC システムのアーキテクチャの違いや最適化手法によるコードの変更を考慮し，それぞれの最適化や複数の最適化の組み合わせの効果について明らかにする．

3. マルチプラットフォームにおける最適化の効果

3.1 評価環境

複数の最適化を組み合わせによる最適化の相乗効果や，他の最適化を組み合わせなければ適用できない最適化の効果を調べるため，単一の最適化手法を施したコードのみでなく，それらを最適化を組み合わせたコードについて検討する．対象とするコードは，水素燃焼反応流シミュレーションのコードで，3 次元圧縮性 Reynolds 平均 Navier-Stokes 方程式に乱流エネルギーの輸送方程式と化学種保存式を加えた支配方程式を解いている [10]．ソースコード 1 に何も最適化を施していないコードの例を示す．この HPC コードに，ループ交換，多重ループの一重化，ループ内不変量コードの移動，マスクによるベクトル・SIMD 化の最適化を手動で施す．HPC コードに適用する最適化同士の依存関係を考慮し，以下に示す 6 通りの最適化を施した HPC コードを作成して評価を行う．

ソースコード 1 評価に用いたコードの例

```
do k=2,z-1
do j=2,y-1
do l=1,w
do n=1,w
do i=2,x-1
   A(i,j,k,n,l)=dsqrt(( B(i,j,k,n) / B(i,j,k,l)
  &             * (C(l)/C(n))**0.25 )**2 )
  &             / dsqrt( C(n)/C(l) )

 . . .

end do
end do
end do
end do
end do
```

- ループ交換
- ループ内不変量コードの移動
- ループ交換，ループ内不変量コードの移動
- ループ交換，三重ループの一重化
- ループ交換，ループ内不変量コードの移動，三重ループの一重化
- ループ交換，ループ内不変量コードの移動，三重ループの一重化，マスクによるベクトル・SIMD 化

表 1 に示す国内の情報基盤センター群でサービス提供している主な HPC システムを評価に用いた．ベクトル型スーパーコンピュータ NEC SX-9 システムはベクトルプロセッサを 1 ノードに 16 基搭載している．スカラ型スーパーコンピュータである Fujitsu FX1, Fujitsu FX10, Hitachi SR16000M1, Intel Nehalem EX クラスタは，それ

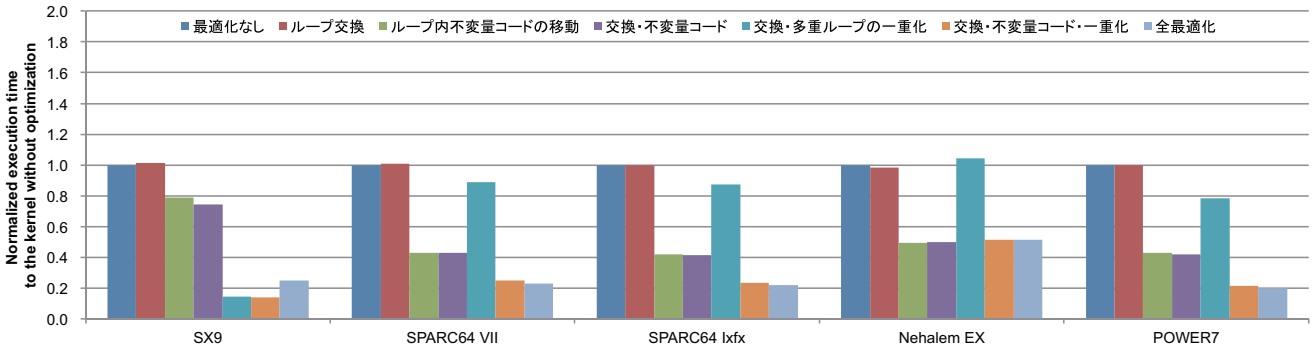

図 1 自動最適化を行わない場合のシングルスレッドにおける最適化手法の効果
Fig. 1 Effects of the optimization methods using single thread without compiler automatic optimizations

ぞれ Fujitsu SPARC64 VII, Fujitsu SPARC64 IXfx, IBM Power7, および Intel Nehalem EX というスカラプロセッサを搭載している．

評価指標として，それぞれの最適化を施したコードの実行時間を，最適化を全く施していない実行時間で正規化した値を用いる．

$$NormalizedTime_{system}^{optimized} = \frac{Time_{system}^{optimized}}{Time_{system}^{original}} \qquad (1)$$

ここで，$NormalizedTime$ は正規化実行時間，$system$ はコードを実行した HPC システム，$optimized$ は施した最適化，$original$ は最適化なしをそれぞれ表し，$Time_{system}^{optimized}$ は HPC システム $system$ における最適化 $optimized$ を施した場合の実行時間，$Time_{system}^{original}$ は HPC システム $system$ における何も最適化を施していない場合の実行時間をそれぞれ表す．

最適化を施していない実行時間で正規化することで，各システムにおけるそれぞれの最適化の効果を定量化することができる．すべてのシステムで性能が向上している場合，施されている最適化は性能可搬性が低下させる恐れのない最適化である．一方，ただ 1 つのシステムでも性能低下が見られる場合，その最適化は性能可搬性を低下させる恐れのある最適化であると判断することが出来る．

評価では，これらの最適化が施されたコードと最適化が全く施されていないコードの実行時間をそれぞれの HPC システムで計測する．計測は 10 回行い，その平均を用いる．

3.2 シングルスレッドにおける最適手法の効果

まず，HPC システムに搭載されているプロセッサで，シングルスレッド実行を行った場合の性能を評価する．**図 1** にコンパイラによる自動最適化を行わない場合の結果を示す．SX-9 では Cvsafe，他のシステムでは O0 の最適化オプションを用いる．縦軸は式 1 で定義される正規化実行時間を，横軸はそれぞれの HPC システムを示す．

ループ交換に着目すると，スカラ型スーパコンピュータではほとんど性能の変化が見られないが，SX-9 では若干実行時間が増加していることがわかる．この要因としては，ループの順番を変えたことによって，配列アクセスの順番が変わり，バンクコンフリクトが増加したためである．

次に，ループ内不変量コードの移動に着目する．ループ内不変量コードの移動が適用されている組み合わせにおいて，すべての HPC システムの実行時間が大きく減少していることがわかる．ループ内不変量コードの移動では，ループ間で値が変わらない余分な計算を明示的に削除しているため，効果が大きい．また，SX-9 において，ループ内不変量コードの移動と速度低下を招いたループ交換を組み合わせた方が実行時間が短いことがわかる．これはループ交換と組み合わせることによって，ループ内不変量コードがより外側のループに移動し，多くの冗長な演算を削減できたためである．

次に，ループ交換と多重ループの一重化の組み合わせに着目する．今回対象とするコードでは，ループ交換を施し，i，j，k の内側の三重ループを一重化する．一重化を施すにあたり，袖領域も内側領域と同じ計算を一旦行い，ループ終了後に袖領域の本来の計算結果で上書きしている．一重化の効果を見ると，Nehalem EX を除くすべての HPC システムにおいて，実行時間が短縮しているのがわかる．特に SX-9 においてはその効果が非常に大きく，約 85.5%もの実行時間が短縮されている．SPARC64 VII，SPARC64 IXfx，Power7 を搭載する 3 つのスカラ型の HPC システムでは，ループの一重化によるループオーバヘッドの削減により性能が向上したと考えられる．また，SX-9 における性能向上の要因は，ループ間で独立して処理が可能なループ長が増えたため，効率的なベクトル処理が可能になるためである．コードを解析すると，ループの一重化を施す前のループ長は 139 で，SX-9 が同時にベクトル演算可能な 256 要素に足りていない．一重化によりループ長は 221149 に増え，SX-9 において効率的なベクトル演算が可能になり，大幅な速度向上に繋がった．Nehalem EX において若

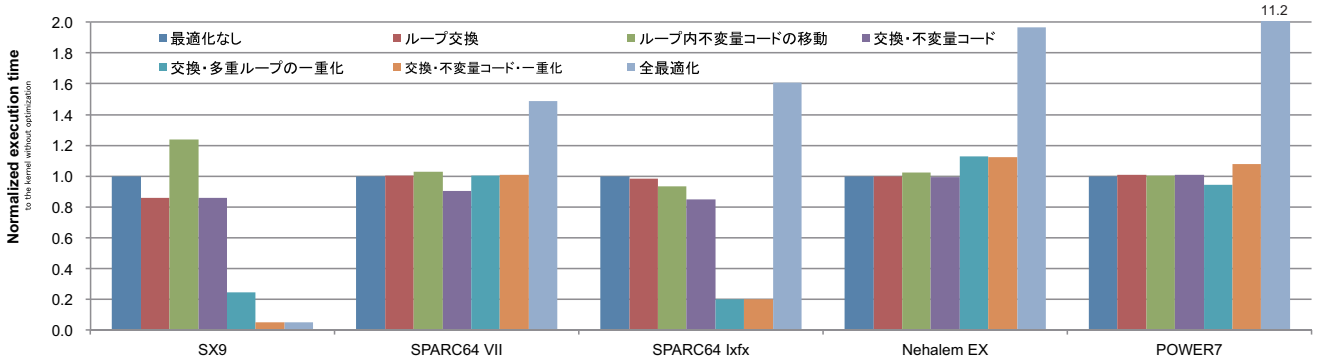

図 2 自動最適化を行った場合のシングルスレッドにおける最適化手法の効果
Fig. 2 Effects of the optimization methods using single thread with compiler automatic optimizations

干実行時間が増加している要因としては，演算量の増加が考えられる．一重化により袖領域では不要な計算を行うため，演算数自体が増えてしまう．一重化の利点よりも演算数増加の影響で若干の速度低下を招いたと考えられる．

すべての最適化を組み合わせた結果を見ると，スカラ型スーパーコンピュータにおいては，実行時間がほぼ変化がないことがわかる．コンパイラ自動最適化を行わない場合，SIMD 化が行われないため，実行時間の削減が見られない．一方，SX-9 では実行時間が長くなっている．本来マスク処理はベクトル処理に適した最適化であるが，コンパイラ自動最適化を行わない場合，実効性能が低下してしまう．これはコンパイラの自動最適化レベルが低い場合には，ベクトル拡張命令であるベクトルマスク演算が無効であるためである．この最適化手法は，コンパイラに対してベクトルマスク命令を利用したベクトル処理を促す最適化であるが，ベクトルマスク命令が利用できないため，通常の条件分岐として実行されてしまう．その結果，実行時間が増加している．

図 1 の結果より，いくつかの例外を除いてほぼすべての最適化手法が各 HPC システムの性能を改善しており，性能可搬性を大きく損なう組み合わせも見られない．しかしながら，一般的な HPC コードはコンパイラ自動最適化機能を有効にしてコンパイルされるため，各最適化手法とコンパライ自動最適化との組み合わせが性能に与える影響についても議論する必要があり，次節でこのことを取り扱う．

3.3 シングルスレッドにおける手動最適化と自動最適化と組み合わせた場合の効果

コンパイラによる最大限の自動最適化を適用した場合において，手動で行った最適化がシングルスレッド実行の性能に与える影響を調査した結果を**図 2** に示す．最適化レベルは SX-9，SPARC64 VII，SPARC64 IXFx，Nehalem EX，Power7 において，それぞれ，Chopt，O5，O3，O3，Oss である．

ループ交換の効果を見ると，スカラ型スーパーコンピュータにおいては実行時間の変化がほとんど見られない．この要因としては，コンパイラ自動最適化にループ交換が含まれていることが挙げられる．一方，SX-9 においては，自動最適化を行わない場合と傾向が異なり，実行時間が短縮している．SX-9 の解析機能を用いて実行結果を分析したところ，バンクコンフリクトは増加しているが，ベクトル長及びベクトル化率が向上していることが分かった．また，ベクトル演算中の再利用性のあるデータを自動的にオンチップメモリに保存していることが分かった．コンパイラ自動最適化を有効にすることにより，オンチップメモリの活用やループ交換による効率的なベクトル計算が促進が行われたため，実行時間の短縮につながっている．

次に，ループ内不変量コードの移動に着目すると，図 1 の結果とは異なり，コンパイラ自動最適化を有効にすることによって，手動によるループ内不変量コードの移動はほとんど性能を変化させていないことがわかる．これは，コンパイラが自動的にループ内不変量の移動を行ったためである．また，SX-9 においては，実行時間が約 24%も増加している．SX-9 のコンパイラから出力される中間コードを用いて解析すると，最適化を何も施さないオリジナルのコードでは最も内側の 2 つのループがベクトル演算の対象となっている．一方，最適化を施すと，冗長な計算が内側から 2 つ目のループ中に移動したため，最も内側のループのみしかベクトル演算の対象となっていない．ループ内不変量コードの移動を適用することにより，コンパイラによる自動ベクトル化が阻害されてしまい，実行時間が増加したと考えられる．

一方で，ループ内不変量コードの移動とループ交換を組み合わせることで，SX-9 や SPARC64 VII, SPARC64 IXfx において，実行時間が約 10%から 15%短縮している．図 1 で示した結果と同様に，ループ交換を組み合わせることにより，ループの変数に依存関係のないループ内不変量コードをより外側のループへ移動できる．これにより，冗

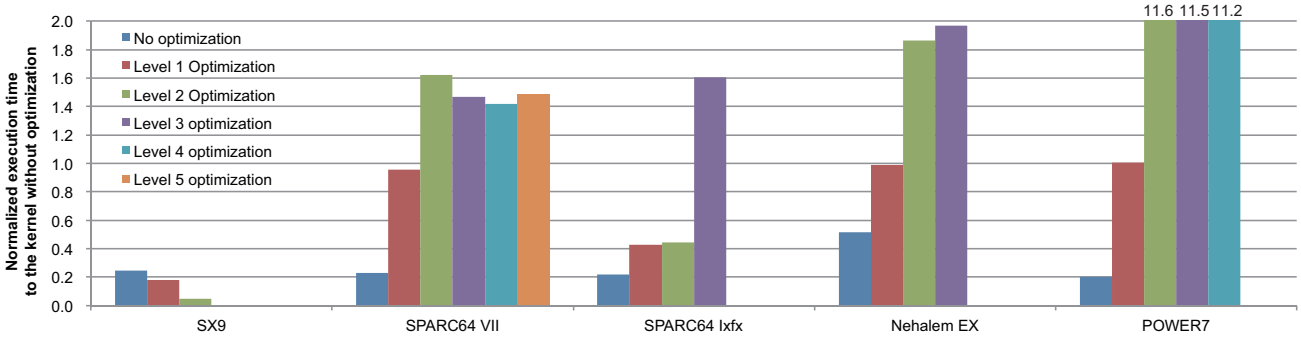

図 3 自動最適化レベルを変更した場合のマスクによる最適化手法の効果
Fig. 3 Performance differences of the automatic optimization levels

長な計算を更に削減できるため，その効果が高まり，速度向上につながった．また，SX-9 においては，ループ内不変量コードがより外側へ移動されたため，自動ベクトル化を阻害することはなくなり，適切なベクトル処理が行われていることが確認できた．

次に，三重ループの一重化について着目すると，SX-9 と SPARC64 IXfx において実行時間が大きく減少しているのがわかる．これは，コンパイラ自動最適化を行わない場合と同様に，効率的なベクトル・SIMD 処理やループオーバーヘッドの削減によって実行時間が短縮されたためであると考えられる．SX-9 における解析機能を用いて実行結果の分析を行ったところ，ベクトル処理の平均の長さがほぼ最大ベクトル長となり，効率的なベクトル処理が実現できることを確認できた．

一方，Nehalem EX や SPARC64 VII，Power7 においては，三重ループの一重化によって実行時間が増加している．Nehalem EX においては，他の最適化との組み合わせた場合にも実行時間の増加を招いている．アセンブリコードを解析したところ，三重ループの一重化を施さない場合と比較して，ループ展開や SIMD 演算が利用されていない場合があることが分かった．また，SPARC64 VII と Power7 においては，三重ループの一重化とループ内不変量コードの移動の組み合わせで性能低下を引き起こしている．自動最適化を行わない場合には三重ループの一重化の効果が見られているため，三重ループの一重化が自動最適化と競合していると考えられる．コンパイラによる自動最適化のうち，どの最適化と競合して性能低下を招いているのか，今後さらなる調査が必要である．

以上の結果から，多重ループの一重化は一部の HPC システムにおいて大きな性能向上をもたらす一方で，他の HPC システムにおける性能低下を引き起こす要因になりうることがわかる．すなわち，多重ループの一重化はシステム依存性の高い最適化手法であるといえる．そのため，性能可搬性の観点からは避けるべき最適化手法であり，性能改善が期待できる HPC システム向けにはコンパイラによる自動最適化が求められる．

マスクによるベクトル・SIMD 化の促進の効果を見てみると，スカラ型スーパーコンピュータにおいて著しく実行時間が増加していることがわかる．この原因として，マスクによる SIMD 化の促進するための記述が複雑でコンパイラが解釈出来ず，SIMD 化や自動最適化の阻害要因となったことが考えられる．Nehalem EX 向けのアセンブリコードを解析したところ，本最適化を行わないオリジナルのコードには，コンパイラ自動最適化により，積極的にループ展開が行われている．また，SIMD 命令が多用されていることが分かった．一方で，マスクによる SIMD 化を行った場合，ループ展開などの自動最適化はほとんど行われていない．

図 3 にコンパイラによる自動最適化レベルを変更した場合の最適化の効果を示す．これを見ると SX-9 においては自動最適化レベルを上げるにつれて実行時間が短くなっている．しかしながら，スカラ型スーパーコンピュータにおいては，自動最適化レベルを上げるにつれて，実行時間が長くなる傾向がある．これは，マスクによるベクトル・SIMD 化がコンパイラ自動最適化を阻害していることを示している．さらに，最適化を行わない場合に利用されていた SIMD 命令がスカラ命令に置き換わっており，SIMD 命令がほとんど利用されていないだけでなく，ループ中に性能低下を引き起こす恐れのある条件分岐が増えていることが分かった．本最適化によりコードが複雑になってしまい，コンパイラが認識することができず，SIMD 化や自動並列化を阻害している．また，SX-9 においても，マスクを用いた場合の実行時間はほぼ同等で，ベクトル化がそれほど促進されないことが分かった．

以上より，本節では，自動最適化と手動最適化を組み合わせることで，最適化の効果が大きく変化することが分かった．特に，スカラ型スーパーコンピュータにおける多重ループの一重化やマスクを用いたベクトル・SIMD 化では性能低下が見られており，性能可搬性を低下させる要因であることが分かった．そのため，性能可搬性の高いコードを開発する際には，使用するコンパイラの差異やその自動最適化機能との組み合わせも考慮して最適化を検討する

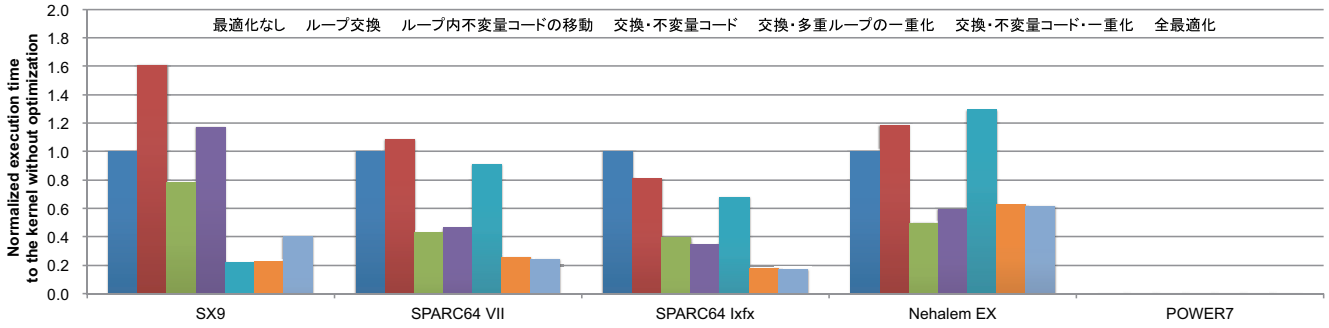

図 4 自動最適化を行わない場合のマルチスレッドにおける最適化手法の効果
Fig. 4 Effects of the optimization methods using multi-thread without compiler automatic optimizations

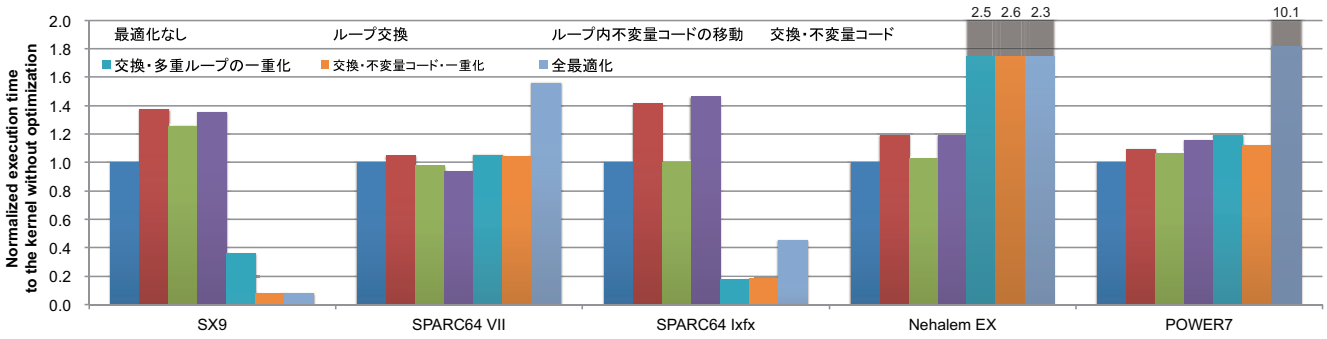

図 5 自動最適化を行った場合のマルチスレッドにおける最適化手法の効果
Fig. 5 Effects of the optimization methods using multi-thread with compiler automatic optimizations

必要がある．

3.4 マルチスレッドにおける最適手法の効果

複数のスレッドを用いた場合の最適化手法の効果を調べるために，HPC システムの 1 ノードのプロセッサ数と同数のマルチスレッド実行によって，3.2 節と同様の評価を行った．スレッド数は，SX-9，SPARC64 VII，SPARC64 IXFx，Nehalem EX，Power7 において，それぞれ，16，4，16，32，32 である．なお，Power7 において，マルチスレッド実行を行うためにはコンパイラ自動最適化が必須となるため，自動最適化を行わないマルチスレッド実行は評価できていない．

図 4 にコンパイラによる自動最適化を行わない場合の結果を，**図 5** にコンパイラによる最大限の自動最適化を行った場合の結果をそれぞれ示す．図 4 と図 5 の結果を見ると，多くの最適化とその組み合わせにおいて，シングルスレッド実行における評価結果とほぼ同様の最適化の効果が得られていることがわかる．図 4 では，ループ内不変量コードの移動や多重ループの 1 重化，マスクによるベクトル・SIMD 化の促進については，シングルスレッド実行における最適化の効果と同様の傾向である．これは，これらの最適化が冗長な演算の削減やベクトル・SIMD 処理の効率化など，プロセッサに搭載されるレジスタやオンチップメモリなどの共有資源への影響が小さい最適化手法であるためだと考えられる．

一方，複数スレッドによる並列処理によって，メモリのアクセス順番や共有資源であるオンチップメモリの利用へ影響を与えうるループ交換やその組み合わせの最適化については，実行時間が増加する傾向にある．これは，すべてのハードウェア資源を専有できるシングルスレッド実行における最適化の効果と異なり，スレッド間でリソースの競合が発生し，実行時間の増加につながったためであると考えられる．

図 5 に示すコンパイラによる最大限の自動最適化を行う場合においても図 4 と同様に，シングルスレッドにおける場合と比べ，ハードウェア資源の競合が発生する可能性のあるループ交換においては実行時間の増加が見られる．

図 6 にスレッド数を変化させた時の各 HPC システムにおけるループ交換の効果を示す．スレッド数が 2 の場合がどのシステムにおいても最も実行時間が短く，さらにスレッド数が増えるにつれて実行時間が増えることがわかる．これは，スレッド数が少ない場合はリソースの競合が起こらず，スレッド数が増えるにつれてリソースの競合が高まったためだと考えられる．また，Nehalem EX や Power7 ではスレッド数が 32 の場合に 16 に比べ実行時間が短くなっている．これは，他のスレッドが利用したデー

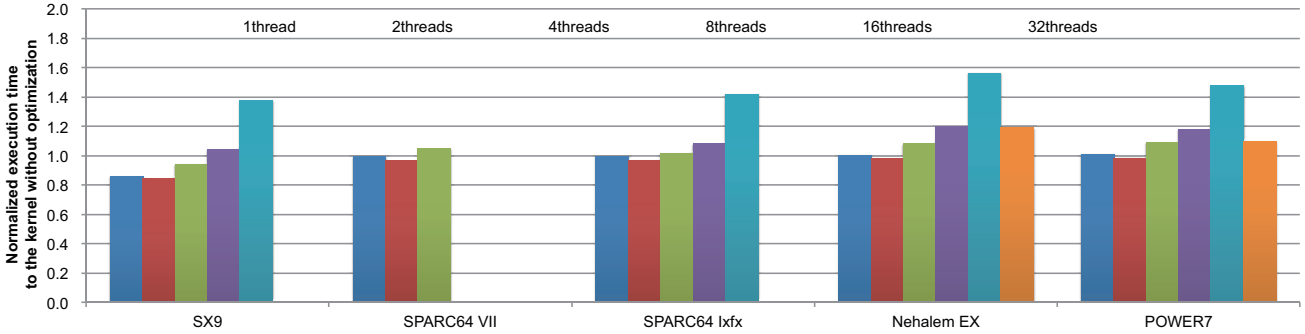

図 6 スレッド数を変更した場合のループ交換の効果

Fig. 6 Performance differences of loop exchanges on various numbers of threads

タを他のスレッドでも利用できるようになり，競合が緩和したためだと考えられる．

以上より，マルチスレッドで実行する場合の最適化の効果は，自動最適化を行う場合も行わない場合も，基本的にはシングルスレッドと同様である．しかしながら，複数のスレッドで同一のリソースを利用するようなループ交換などの最適化では，リソースの競合により実行時間が増加し，性能可搬性が低下することが分かった．

4. おわりに

本報告では，ループ交換，多重ループの一重化，ループ内不変量コードの移動，マスクによるベクトル・SIMD 化の 4 種類の最適化を取り上げ，それぞれの最適化や複数の手動最適化，コンパイラによる自動最適化を組み合わせた場合の効果について調査した．その結果，シングルスレッドでの実行においては多重ループの一重化やマスクによるベクトル・SIMD 化が，コンパイラ自動最適化と競合を起こし，性能可搬性を下げることが分かった．また，複数スレッドでの実行においては，ハードウェア資源の競合を引き起こす可能性があるループ交換などの最適化が性能可搬性に影響を及ぼすことが分かった．

今後の課題として，手動最適化と競合する自動最適化の詳細な調査や，スカラ型スーパーコンピュータ同士での効果の違いの解析，最適化手法の対象を増やすことが挙げられる．

5. 謝辞

本研究は，北海道大学情報基盤センター，東北大学サイバーサイエンスセンター，東京大学情報基盤センター，名古屋大学情報基盤センターのスーパーコンピュータを利用することで実現することができた．本研究の一部は，文部科学省科研費研究 (S)(21226018) と科学技術振興機構 (JST) 戦略的創造研究推進事業 (CREST) 研究領域「ポストペタスケール高性能計算に資するシステムソフトウェア技術の創出」研究課題「進化的アプローチによる超並列複合システム向け開発環境の創出」の助成を受けている．

参考文献

[1] Soga, T., Musa, A., Shimomura, Y., Egawa, R., Itakura, K., Takizawa, H., Okabe, K. and Kobayashi, H.: Performance evaluation of NEC SX-9 using real science and engineering applications, *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, SC '09, pp. 28:1–28:12 (2009).

[2] Dunigan Jr., T. H., Vetter, J. S., White III, J. B. and Worley, P. H.: Performance Evaluation of the Cray X1 Distributed Shared-Memory Architecture, *IEEE Micro*, Vol. 25, No. 1, pp. 30–40 (online), DOI: 10.1109/MM.2005.20 (2005).

[3] 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, pp. 1:1–1:11 (online), DOI: 10.1145/2063384.2063386 (2011).

[4] Rahimian, A., Lashuk, I., Veerapaneni, S., Chandramowlishwaran, A., Malhotra, D., Moon, L., Sampath, R., Shringarpure, A., Vetter, J., Vuduc, R., Zorin, D. and Biros, G.: Petascale Direct Numerical Simulation of Blood Flow on 200K Cores and Heterogeneous Architectures, *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '10, pp. 1–11 (online), DOI: 10.1109/SC.2010.42 (2010).

[5] Sidelnik, A., Maleki, S., Chamberlain, B. L., Garzaran, M. J. and Padua, D. A.: Performance Portability with the Chapel Language., *IPDPS*, IEEE Computer Society, pp. 582–594 (2012).

[6] F. Kjolstad, D. D. and Snir, M.: Bringing the HPC Programmer's IDE into the 21st Century through Refactoring, *SPLASH 2010 Workshop on Concurrency for the Application Programmer (CAP'10)* (2010).

[7] 中田育男：コンパイラの構成と最適化, 朝倉書店, 2 edition (2009).

[8] 東北大学サイバーサイエンスセンター：高速化推進研究活動報告 (2011).

[9] 小松一彦，江川隆輔，安田一平，撫佐昭裕，松岡浩司，小林広明：HPC アプリケーションの性能可搬性に関する一検討，第 136 回 HPC 研究会 (2012).

[10] 滝田謙一：炭化水素系燃料の超音速乱流燃焼の数値シミュレーション，学際大規模情報基盤共同利用・共同研究拠点平成 23 年度共同研究 最終報告書 (2012).