Vol.2016-HPC-155 No.7
2016/8/8

最適化手法を自動化するXevolverフレームワーク用定義ファイルの実装

五味 歩武[1,a)]　高橋 大介[2,b)]

概要：古いプログラム資産の最新アーキテクチャへの移植や最適化によるソースコードの複雑化の抑止，ソースコードの保守性と複数のアーキテクチャへの対応の両立といった諸問題の解決は現代の HPC 分野において重要なテーマであり，これらの解決策として最適化手法の自動的な適用が考えられる．本研究では， Xevolver と呼ばれるプログラミングフレームワークを用いて，OpenMP の`!$OMP DO` ディレクティブ への `COLLAPSE(2)` 節の追加およびキャッシュブロッキング， `!$OMP DO` ディレクティブ への `SCHEDULE(DYNAMIC)` 節の追加という 3 種類の最適化手法を自動化する．また，これらを実際のプログラムに対して適用し，手動で最適化した場合との比較および性能の測定を行い，上記の問題に対する解決可能性について検討する．

Implementation of Definition Files of the Xevolver Framework that Automatize Optimization Techniques

1. はじめに

HPC 分野には最新のアーキテクチャで効率よく実行できない古いプログラム資産が多数存在するが，古い資産を現在のアーキテクチャ上で高速に実行できるよう手動で性能最適化を行うのは手間がかかる．また，HPC では様々なアーキテクチャが使用され，それぞれで有効な最適化手法が異なるため，同一のプログラムのソースコードをアーキテクチャ毎に分けて管理しなければならず，修正を行うたびに全てのアーキテクチャ向けコードにその修正を反映させなければならない．そのため，単一のソースコードからアーキテクチャ毎に最適化されたコードを出力したいという要求が存在する．こうした問題を解決する手段として，最適化手法の適用を自動化することが考えられる．

本研究では，Xevolver[1] と呼ばれるプログラミングフレームワークを用いて，OpenMP の`!$OMP DO` ディレクティブ への `COLLAPSE(2)` 節の追加およびキャッシュブロッキング，`!$OMP DO` ディレクティブ への `SCHEDULE(DYNAMIC)` 節の追加という 3 種類の最適化手法を自動化することにより，上記の問題に対する解決可能性を検討する．

1 筑波大学大学院 システム情報工学研究科
2 筑波大学 計算科学研究センター
a) agomi@hpcs.cs.tsukuba.ac.jp
b) daisuke@cs.tsukuba.ac.jp

2. 関連研究

Source-to-source 変換によるコンパイル前のプログラム最適化の試みはさまざまなものが行われてきており，Xevolver が使用しているのと同じ ROSE[2] コンパイラ基盤を用いたものだけでも，Quinlan らによる構文木操作フレームワーク [3] や，Liao らによる OpenMP 化を自動化するもの [4] などが行われてきている．しかし，これらは変換ルールが実装内部に隠蔽されてしまっており，ユーザが新たなルールを追加したり，ルールを柔軟に組み合わせたりすることができない．

ユーザがプログラム可能な最適化フレームワークとその実例としては，CHiLL フレームワーク [5] を用いた Hall らによるもの [6] や Tiwari らによるもの [7] がある．また，POET 言語 [8] ではユーザがスクリプト言語を記述することによって Source-to-source 変換を行っている．

Xevolver を用いた研究としては，山田ら [9] がデータのメモリ配置を Array of Structure 形式から Structure of Array 形式に変換するルールの実装を行っているほか，小松らの論文 [10] ではレガシープログラムの GPU への移植を目的とし，Xevolver を用いて気象シミュレーションプロ

グラムに OpenACC のディレクティブを自動的に付与し，その性能を測定している．

3. 最適化手法

本研究では，自動化を行う対象として，以下の 3 種類の最適化手法を選択した．

3.1 !$OMP DO ディレクティブ への COLLAPSE(2) 節の追加

perfectly nested loop と呼ばれる多重ループの特殊なケースに対しては，`!$OMP DO` ディレクティブに `COLLAPSE` 節を追加することにより，多重ループを一つのループに統合し，ループ一つあたりの繰り返し回数を増やすことにより，スケーラビリティを向上させることができることが知られている．しかし，OpenMP を用いた既存のプログラムには`!$OMP DO` ディレクティブの付いた `DO` 文が多数出現するため，それら全てについて perfectly nested loop となっているかを判定し， `COLLAPSE` 節を追加するのは大変な労力を要する．

そこで，指定したファイルまたはブロック内に出現するすべての`!$OMP DO` ディレクティブ付き `DO` 文について上記の処理を自動的に行うことで，OpenMP を用いた既存のプログラムの多重ループを一重化することができ，少ない労力でスケーラビリティを向上できるものと考えられる．

本研究では，そのなかでも最も高い効果が見込まれる，`COLLAPSE(2)` 節の追加のみを実装している．

3.2 キャッシュブロッキング

キャッシュブロッキングとは，ソースコード上の多重ループにおいて，用いられるデータをキャッシュ上になるべく長く留めるようデータのアクセス順序を変更することによりメモリアクセスを低減させる，比較的よく知られた最適化手法である．

この手法を手動で適用した場合，ループのネスト数が増大するなどソースコードの複雑性が増加し，使用したアルゴリズムなどといったプログラマの意図がソースコードから読み取ることが困難になってしまう．そこで，最適化対象のループ文を書き換える代わりに，直前にディレクティブを一行挿入するに留め，最適化自体は自動化することにより，ソースコードを読んだ際の理解しやすさを向上させることができると考えられる．

3.3 !$OMP DO ディレクティブ への SCHEDULE(DYNAMIC) 節の追加

本研究で取り扱うプロセッサである Xeon Phi 5110P ではアウトオブオーダ実行をサポートしておらず，OpenMP によってループを並列化した場合，より一般的なスケジューリング方式である static では十分な性能を発揮できない恐れがある．そこで，`!$OMP DO` ディレクティブに対し，`SCHEDULE(DYNAMIC)` を追加することでスケジューリング方式を変更し，ループ処理を繰り返すごとに空いているスレッドに処理を動的に割り当てるようにすることにより，高速化を図ることができるのではないかと考えられる．現在の多くのアーキテクチャ上で `SCHEDULE(DYNAMIC)` を追加すると，スケジューリング機構のコスト増大により速度が低下してしまうために，一般的に高速化手法としては用いられないが，MIC（Many Integrated Core）アーキテクチャ上でこの手法を用いて高速化を図った事例として，伊奈ら [11] のものが挙げられる．

4. Xevolver

Xevolver はソフトウェアチューニングを行うために開発されているコード変換フレームワークであり，C または Fortran コードを外部ファイルに記述した規則に基づいて変更することができる．

変換規則ファイルはレシピと呼ばれ，XML ドキュメントを変換する規則の集合を記述するためのファイル形式である XSLT で記述される． Xevolver のうち，`src2xml` と呼ばれるプログラムは，ソースコードを入力として受け取るとそれらをパースし，抽象構文木を表現する XML ドキュメントを生成する．生成された XML ドキュメントは `xsltexec` と呼ばれる　XSLT 処理系によってレシピに記述された規則に従って変換を行い，その結果を再度 XML ドキュメントとして出力する．そして `xml2src` と呼ばれるプログラムに変換後の XML ドキュメントを入力することにより，最終的に元の言語で書かれたソースコードに変換することができる．

5. 対象としたプログラムおよび測定環境

本研究では，作成したレシピの実用性の検証を目的とし，著名なベンチマークプログラムである姫野ベンチマーク [12] に対して各レシピを適用し，変換後のコードを MIC アーキテクチャと汎用 x86 アーキテクチャの 2 種類の環境で性能を測定した．

また，測定に用いた各環境の概要を以下に示す．

5.1 MIC

本研究で用いたモデルである Xeon Phi 5110P は，ハードウェアスレッドを 4 スレッド実行可能なコアを 60 コア搭載しており，合計で 240 スレッドを実行することができる．本研究ではスレッド数として，1 から 128 までの 2 の累乗のほか，物理コア数の倍数を用いて測定した．詳細な仕様を**表 1** に示す．

5.2 Xeon

本研究では上記の MIC アーキテクチャとの比較を目的

表 1　MIC 環境の詳細仕様
Table 1　The specification of MIC environment.

コプロセッサ	Xeon Phi 5110P （60 コア，240 スレッド）
メモリ	8GB
Intel MPSS	3.6
コンパイラ	ifort 16.0.2 20160204
コンパイルオプション	`-O3 -qopenmp -mmic`
KMP_AFFINITY	`granularity=fine,compact`
測定スレッド数	1,2,4,8,16,32,60,64,120,128,180,240

```
!$OMP DO
  do k=2,kmax-1
     do j=2,jmax-1
        do i=2,imax-1
```

図 1　姫野ベンチマークのオリジナルコード
Fig. 1　The original code of the Himeno benchmark.

```
!$OMP DO COLLAPSE(2)
DO k = 2, kmax - 1
DO j = 2, jmax - 1
DO i = 2, imax - 1
```

図 2　COLLAPSE(2) 節追加レシピを適用したコード
Fig. 2　The code transformed by the COLLAPSE(2) insertion recipe.

とし，従来型の x86 CPU である Intel 社の Xeon プロセッサでも性能測定を行った．詳細な仕様を**表 2** に示す．

6. レシピの概要および評価

次に，レシピによる変換と手動での変換の比較，およびレシピによる変換での速度向上の有無の確認を目的とし，各レシピの実装方針および姫野ベンチマークに対して適用した場合の変換後コード，性能測定の結果について述べる．また変換後コードの例として，姫野ベンチマークの中で全てのレシピの影響を受ける箇所である，himenoBMTxp_omp.f90 の 298 行目から 301 行目を使用する．この箇所のオリジナルコードを**図 1** に示す．

6.1 !$OMP DO ディレクティブ への COLLAPSE(2) 節の追加

この最適化手法をレシピとして実装するにあたり，まず全てのディレクティブについて，そのディレクティブが `!$OMP DO` やそれと同等のものであること，そのディレクティブが `DO` 文についていること，その `DO` 文が perfectly nested loop になっていることを検査し，これらの要件を満たすディレクティブにのみ `COLLAPSE(2)` 節を追加する，という方針をとった．姫野ベンチマークには`!$OMP DO` ディレクティブが二箇所存在するが，そのなかで `COLLAPSE(2)` 節を追加可能であるのは一箇所しかない．本研究で実装したレシピでは，追加可能か否かを正しく判別し，人の手による追加と同じ箇所にのみ追加していることを確認した．**図 2** にレシピ適用後のコードを示す．

また，このレシピを適用した姫野ベンチマークの，サイズ L（グリッドサイズ：512 × 256 × 256）での実行時間について，MIC 環境での結果を**図 3**，Xeon 環境での結果を**図 4** に示す．なお，各条件について 20 回ずつ実行し，姫野ベンチマークの出力する実行時間を平均したものが縦軸，

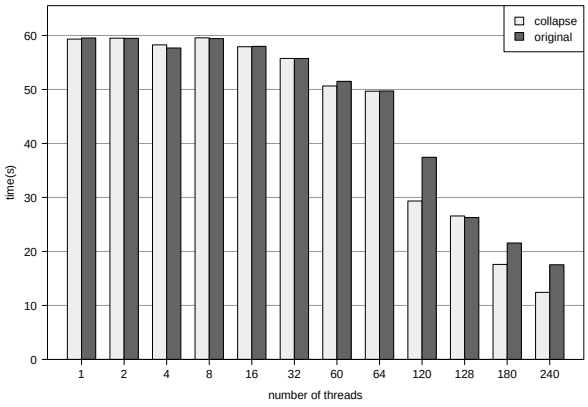

図 3　COLLAPSE(2) 節追加レシピ適用前後の MIC 環境での実行時間
Fig. 3　Execution time of the benchmark before and after the COLLAPSE(2) insertion recipe applied on the MIC environment.

実行したスレッド数が横軸となっている．

図 3，図 4 から，両アーキテクチャともに全体的に速度向上がみられ，本研究により開発したレシピによって，期待した通りの結果が得られたことがわかる．

6.2 キャッシュブロッキング

次に，キャッシュブロッキングを行うレシピの実装について述べる．キャッシュブロッキングは適用するループを手動で選択する必要があり，またブロックの幅を容易に変更できることが望ましいため，**図 5** に示すように，キャッシュブロッキングを適用したいループに`!$XEV CACHE_BLOCKING` という本レシピ用に定めたディレクティブを一行挿入し，その引数としてブロックの幅を記述するという方法をとった．この方法では，ユーザがソースコードに目を通した上でキャッシュブロッキングが有効であ

表 2　Xeon 環境の詳細仕様
Table 2　The specification of Xeon environment.

プロセッサ	Xeon E5-2670 v3（12 コア，24 スレッド，2 ソケット）
メモリ	16GB
OS	CentOS 7.0.1406
コンパイラ	ifort 16.0.2 20160204
コンパイルオプション	`-O3 -qopenmp -march=core-avx2 -xCORE-AVX2`
KMP_AFFINITY	`granularity=fine,compact`
測定スレッド数	1,2,4,8,12,16,24,32,36,48

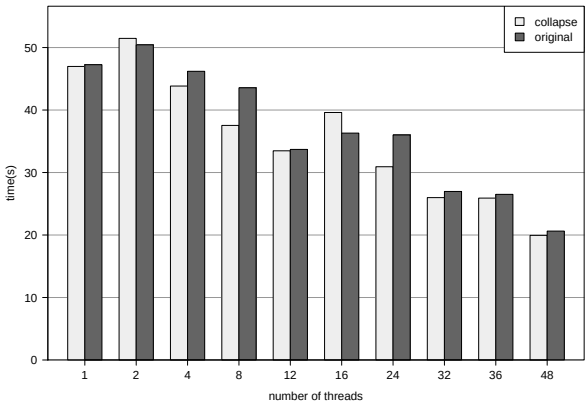

図 4　COLLAPSE(2) 節追加レシピ適用前後の Xeon 環境での実行時間
Fig. 4　Execution time of the benchmark before and after the COLLAPSE(2) insertion recipe applied on the Xeon environment.

```
!$OMP DO
!$XEV CACHE_BLOCKING 16
  do k=2,kmax-1
     do j=2,jmax-1
        do i=2,imax-1
```

図 5　キャッシュブロッキングレシピのためのディレクティブを追加したコード
Fig. 5　The code with a directive for the chache blocking recipe.

るか否かを判断し，ディレクティブを挿入しなければならないため，COLLAPSE(2) 追加レシピなどに比べてユーザの負担が大きくなってしまっているが，ユーザが手作業でキャッシュブロッキングを適用する場合に比べると負担が少なく，また追加変更を要するコード量が少ないため，コードの複雑化の抑止が図れるものと考えられる．

実装方針としては，まずユーザが挿入したディレクティブを検出し，引数として与えられたブロック幅，対象ループのループ回数や変数名，ループ本体などを取得したのち，それらを元に新たなループ文を生成し挿入するという方針をとった．

図 5 に対して本レシピを適用した結果を**図 6** に示す．

また，姫野ベンチマークの当該箇所（図 1）に対し手作業でキャッシュブロッキングを施したコード（**図 7**）と比較しても，大文字小文字やインデントの有無，定数同士の足し算が既に行われているか否かなどの差異が存在するものの，等価なコードになっていることがわかる．

```
!$OMP DO
DO kk = 2, kmax - 1, 17
DO jj = 2, jmax - 1, 17
DO k = kk, MIN(kmax - 1,kk + 16)
DO j = jj, MIN(jmax - 1,jj + 16)
DO i = 2, imax - 1
```

図 6　キャッシュブロッキングレシピを適用したコード
Fig. 6　The code transformed by the cache blocking recipe.

```
!$OMP DO
  do kk=2,kmax-1,16+1
     do jj=2,jmax-1,16+1
        do k=kk,min(kmax-1,kk+16)
           do j=jj,min(jmax-1,jj+16)
              do i=2,imax-1
```

図 7　キャッシュブロッキングを手動で適用したコード
Fig. 7　The code applied cache blocking by hand.

本最適化を施したコードに対しても性能測定を行ったが，速度向上がみられるブロック幅を見出すことはできなかった．これは，姫野ベンチマークがステンシル計算を行っているため，キャッシュブロッキングが有効でなかったことが原因であると考えられる．しかし，本レシピ本来の意図である，コードの意図の読み取りやすさを極力損なわないまま最適化手法を適用するという目標は達成できているものと考えられる．

6.3 !$OMP DO ディレクティブへの SCHEDULE(DYNAMIC) 節の追加

次に，!$OMP DO ディレクティブへ SCHEDULE(DYNAMIC) 節を追加するレシピの実装について述べる．SCHEDULE(DYNAMIC) 節を追加する !$OMP DO ディレクティブに，COLLAPSE(2) 追加レシピにあるような制約がないため，本レシピは対象コード上に存在する全ての!$OMP DO ディレクティブに対し，SCHEDULE(DYNAMIC) 節を追加するという実装になっている．**図 8** に変換後のコードを示す．これらのコードについても，手作業で最適化するのと同じく，全ての!$OMP DO ディレクティブに対して

```
!$OMP DO SCHEDULE(DYNAMIC)
DO k = 2, kmax - 1
DO j = 2, jmax - 1
DO i = 2, imax - 1
```

図 8 SCHEDULE(DYNAMIC) 節追加レシピを適用したコード
Fig. 8 The code transformed by the SCHEDULE(DYNAMIC) insertion recipe.

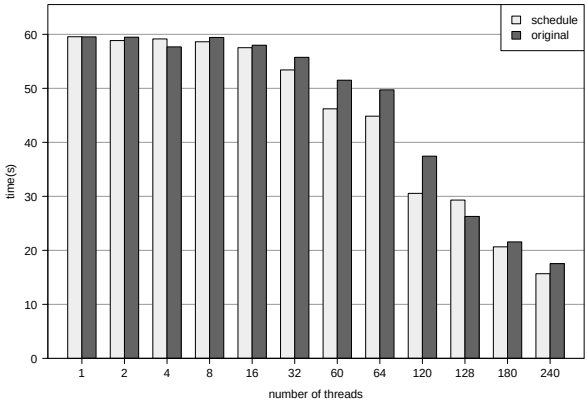

図 9 SCHEDULE(DYNAMIC) 節追加レシピ適用前後の MIC 環境での実行時間
Fig. 9 Execution time of the benchmark before and after the SCHEDULE(DYNAMIC) insertion recipe applied on the MIC environment.

SCHEDULE(DYNAMIC) 節を追加できていることを確認した．

また，姫野ベンチマークサイズ L でのレシピ適用前後の実行時間について，MIC 環境での結果を**図 9** に，Xeon 環境での結果を**図 10** に示す．これらのグラフについても，各条件について 20 回ずつ実行し，姫野ベンチマークの出力する実行時間を平均したものが縦軸，実行したスレッド数が横軸となっている．

MIC 環境では全体的に性能向上がみられるが，Xeon 環境では性能向上がほぼみられず，スレッド数の大きな場合では性能が大幅に悪化している．こうしたケースでは，従来であれば元のソースコードをアーキテクチャごとに複製し，それぞれに対してチューニングを施していたが，本レシピのように最適化手法を自動化することで，単一のソースコードのみから複数のアーキテクチャで高速に実行できるコードを容易に生成できるようになるものと考えられる．

7. まとめおよび今後の課題

本研究では，OpenMP の !$OMP DO ディレクティブへの COLLAPSE(2) 節の追加およびキャッシュブロッキング，!$OMP DO ディレクティブ への SCHEDULE(DYNAMIC) 節の追加という 3 種類の最適化手法を Xevlover レシピとして新規に実装した．さらに，これらを姫野ベンチマークに適用した上で，出力されたソースコードが手動で最適化を行ったソースコードと相違ないことを確認し，MIC アーキテクチャマシンおよび従来型の x86 アーキテクチャマシンでの性能測定を行った．その結果，HPC 分野の諸課題に対する最適化手法の自動化という解決策について，コードの理解しやすさを維持した最適化や，複数アーキテクチャ対応の省力化といった分野での知見を得ることができた．

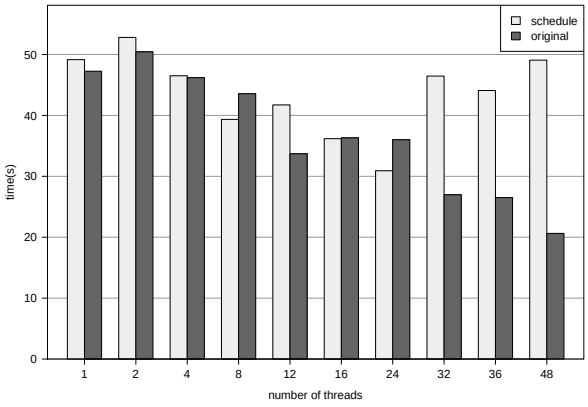

図 10 SCHEDULE(DYNAMIC) 節追加レシピ適用前後の Xeon 環境での実行時間
Fig. 10 Execution time of the benchmark before and after the SCHEDULE(DYNAMIC) insertion recipe applied on the Xeon environment.

一方，本研究はいくつかの課題を残している．COLLAPSE(2) 追加レシピでは，対象となる DO 文にステップ数の指定があった場合に，レシピ適用前後で計算結果が異なってしまうという問題が確認されているため，ステップ数の指定がある DO 文を避けるような改良を予定している．また，キャッシュブロッキングレシピでは，ループの生成時にループ変数の変数名を自動的に生成しているが，これらが対象ソースコード内に既に存在しているかをチェックしていないため，既に存在している場合に問題となる．このため，生成する変数名にランダムな文字列を加える，生成した変数名が既に存在していないことをチェックするなどの対策を施す必要がある．また，現状では二重ループのキャッシュブロッキングまでしか対応していないため，よりネスト数の多いループに対してもキャッシュブロッキングを行えるよう改良を予定している．

参考文献

[1] Takizawa, H., Hirasawa, S., Hayashi, Y., Egawa, R. and Kobayashi, H.: Xevolver: An XML-based code translation framework for supporting HPC application migration, *Proc. 21st IEEE International Conference on High Performance Computing (HiPC 2014)*, pp. 1–11 (2014).
[2] Quinlan, D.: ROSE: Compiler Support for Object-Oriented Frameworks., *Parallel Processing Letters*, Vol. 10, pp. 215–226 (2000).
[3] Quinlan, D., Schordan, M., Philip, B. and Kowarschik, M.: The Specification of Source-To-Source Transformations for the Compile-Time Optimization of Parallel Object-Oriented Scientific Applications, *Proc. 14th In-*

ternational Workshop on Languages and Compilers for Parallel Computing (LCPC'01), Lecture Notes in Computer Science, Vol. 2624, pp. 149–162 (2003).

[4] Liao, C. and Quinlan, D.: Automatic parallelization using OpenMP based on STL semantics, *International Journal of Parallel Programming*, Vol. 38, pp. 361-378 (2010).

[5] Chen, C., Chame, J. and Hall, M.: CHiLL: A framework for composing high-level loop transformations, Technical Report 08–897, University of Southern California (2008).

[6] Hall, M., Chame, J., Chen, C., Shin, J., Rudy, G. and Khan, M.: Loop Transformation Recipes for Code Generation and Auto-Tuning, *Proc. 22nd International Conference on Languages and Compilers for Parallel Computing (LCPC'09), Lecture Notes in Computer Science*, Vol. 5898, pp. 50–64 (2010).

[7] Tiwari, A., Chen, C., Chame, J., Hall, M. and Hollingsworth, J. K.: A scalable auto-tuning framework for compiler optimization, *Proc. 2009 IEEE International Parallel and Distributed Processing Symposium (IPDPS 2009)*, pp. 1–12 (2009).

[8] Yi, Q.: POET: a scripting language for applying parameterized source-to-source program transformations, *Software: Practice and Experience*, Vol. 42, pp. 675–706 (2012).

[9] Yamada, T., Hirasawa, S., Takizawa, H. and Kobayashi, H.: A Case Study of User-Defined Code Transformations for Data Layout Optimizations, *2015 Third International Symposium on Computing and Networking (CANDAR'15)*, pp. 535–541 (2015).

[10] Komatsu, K., Egawa, R. and Hirasawa, S.: Migration of an Atmospheric Simulation Code to an OpenACC Platform Using the Xevolver Framework, *2015 Third International Symposium on Computing and Networking (CANDAR'15)*, pp. 515–520 (2015).

[11] 伊奈拓也，朝比祐一，井戸村泰宏：テラフロップス級メニーコアアーキテクチャにおけるステンシル計算の最適化手法の開発，情報処理学会研究報告ハイパフォーマンスコンピューティング（HPC），Vol. 2015-HPC-152, No. 10, pp. 1–6 (2015).

[12] 姫野ベンチマーク — 理化学研究所情報基盤センター（所内および所外向け）（http://accc.riken.jp/supercom/himenobmt/）