社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2007－ARC－172 (12)
2007－HPC－109 (12)
2007／3／1

iPat/OMPでのソースコードレベル最適化における試行錯誤支援ツール

永野悠介[†] 石原誠[†]
平澤将一[†] 本多弘樹[†]

開発者が直接細やかな調整を行うことができるソースコードレベルでの性能最適化は並列プログラミングにおいて重要である．しかし，従来の開発環境は，最適化のためのコード変換後にコードの可読性が低下する，最適化における試行錯誤の作業量が多いといった問題を持っているため，最適化における試行錯誤に適していない．本研究では，ソースコードレベル最適化における新しい試行錯誤支援手法を提案する．提案手法ではユーザとの対話に基づいて指示文を用いたコード変換・復元を行う．提案手法を対話型並列化支援ツールiPat/OMP上のプログラムリストラクチャリング機能に適用し，実装を行った．適用実験の結果，性能最適化における試行錯誤を可読性を維持しつつ少ない作業量で行えることを確認した．

Assistance Tool for Trial-and-Error in Source-Code-Level Optimization on iPat/OMP

Yusuke Nagano,[†] Makoto Ishihara,[†] Shoichi Hirasawa[†] and Hiroki Honda[†]

Source-code-level performance optimization is important in parallel programming. However, previous development environments have problems, such as decrease of readability of codes and large amount of work for the trial-and-error in source-code-level optimization. In this paper, we propose a new method for trial-and-error in source-code-level optimization. The optimization is performed by directive-based interactive code transformation and undo in our method. We apply the proposed method to program restructuring function of iPat/OMP, which is an interactive parallelization assistance tool for OpenMP. The results of application experiment show our method solves the above-mentioned problems in source-code-level optimization.

1. はじめに

近年コンパイラに自動性能最適化機能が搭載され，一般的に利用されているが，性能低下を招くことがあった．このため，現在においても，開発者が直接細やかな調整を行うことができるソースコードレベルでの最適化が広く行われており，並列プログラミングにおいても有効であることが知られている．この作業を手作業で行うことがあるが，熟練者であっても手間がかかり，ミスを含みやすいため，これまで様々な開発環境が提案されてきた[1)2)3)]．

しかし，従来の開発環境では，ソースコードレベルでの性能最適化が元来持つ，コードの可読性の低下という問題を回避できない．そして，一度可読性が低下してしまうと，さらなる作業が困難になる．また，特定の実行環境でのみ性能最適なコードとなってしまい，環境を移行する際には再度作業する必要があった．これらは開発者にとって負担となっており，プログラムの処理の本質的な部分への集中を妨げていた．

本研究では，性能最適化のための試行錯誤における新しい支援手法を提案し，対話型並列化支援ツールiPat/OMP[4)]上のプログラムリストラクチャリング機能に適用する．

iPat/OMPは汎用エディタ上に実装されており，プログラム編集と逐次プログラムの並列化を同時に行うことができる．これにより，ユーザは作業を中断することなく，並列プログラムをスムーズに作成することが可能となっている．

本研究では，新たに指示文によるコードの変換と元のコードの復元をiPat/OMP上で実現することで，ユーザが変換後のコードと可読性の高い元のコードを同時に確認しつつ開発することを可能にする．これにより，プログラム編集，並列化，最適化における試行錯誤をスムーズに行う開発環境を実現し，ユーザがプログラムの処理の本質的な部分に集中できるようにすることを研究目的とする．

† 電気通信大学大学院情報システム学研究科
Graduate School of Information Systems, The University of Electro-Communications

2. ソースコードレベル最適化

2.1 最適化における試行錯誤

性能最適化には様々な手法があり，有効な最適化手法やそのパラメタは実行環境に依存する．そのため，最適なコードを得るためには，手法やパラメタを変化させコード変換し，性能を計測するという最適化における試行錯誤が不可欠である．また，最適化における試行錯誤は，コード変換し性能計測した後，元のコードを再度他の手法やパラメタでコード変換し性能計測するという手順を踏むため，スムーズな試行錯誤を行うには元のコードを復元する手段が重要である．

なお，本稿では以降，コード変換されていない元のコードを元コード，コード変換されたコードを変換後のコードと呼ぶ．

2.2 従来の開発環境の問題点

ソースコードレベル最適化という観点に立つと，エディタ上での手作業や並列化支援ツール[1)3)2)]付属のコード変換機能の利用，またこれらとバージョン管理システムの併用といった従来の開発環境では，最適化における試行錯誤を阻害する次のような問題がある．

(1) コード変換後にコードの可読性が低下する．
- (a) 記述が繁雑になり，可読性が低下する．
- (b) 元コードと変換後のコードとの対応が不明確であり，可読性が低い．

(2) 最適化における試行錯誤の作業量が多い．
- (a) コード変換の作業量が多い．
- (b) 元コードの復元の作業量が多い．
- (c) ソースコードを変更しなければ変換後のコードを確認できないため，その際のコード変換・復元の作業が必要となる．
- (d) 最適化手法におけるパラメタを変更するための作業量が多い．

3. ソースコードレベル最適化における試行錯誤支援手法

本章では，従来の開発環境の問題を解決するための，ソースコードレベル最適化における試行錯誤支援手法を提案する．

提案手法では，ユーザとの対話に基づいて指示文を用いたコード変換と元コードの復元を提供することで，性能最適化における試行錯誤をスムーズに行うことを可能にし，ユーザがプログラムの処理の本質的な部分に集中することを促進する．

3.1 指示文を用いたコード変換・復元

提案手法では，指示文によりコード変換や元コードの復元を行うこととする．具体的には，コード変換したい場合，ソースコード内に図1のように記述したコード変換指示文をコード変換機能により処理することで，変換後のコードを生成し，後述する変換後のコード個別挿入機能によってソースコードに挿入する．その際，図2のようにコード変換指示文と元コードはコメントとして保存する．また復元したい場合，図2の復元指示文を元コード検出機能により処理することで，元コードを探索・検出し，後述する元コード個別復元機能によってソースコードに復元する．

```
#OPT_PRAGMA(PRAGMA_ID, ...)
(元コード)
```

図 1 コード変換指示文

```
(コメントアウトされたコード変換指示文・元コード)
#undo(PRAGMA_ID)
(変換後のコード)
```

図 2 復元指示文

ここで図1の *OPT_PRAGMA* は最適化手法，第1引数は固有の指示文ID番号，第2引数以降は各最適化手法におけるパラメタ，図2の第1引数は対応するコード変換指示文の指示文ID番号である．

3.2 ユーザとの対話

提案手法では指示文を用いたコード変換・復元を効果的に利用するため，ユーザとの対話を行うこととする．

3.2.1 支援情報の事前確認・個別適用

提案手法は，コード変換・復元処理中に得られた支援情報をソースコードに適用する前にユーザに提示し，確認させ，個別に適用させるための，変換後のコード個別挿入機能，元コード個別復元機能を備える．

これにより，処理結果に対する誤解や操作ミスを防ぐことができる，個別にコード変換の有無を切り替えることができ性能向上に寄与する部分を発見することが容易になるといった利点がある．

3.2.2 コード変換指示文作成

提案手法は指示文の記述を補助する仕組みを備える．また，指示文を作成する際に，各最適化手法を選択肢として提示し，必要なパラメタをユーザに問い合わせることとする．

これにより，コード変換指示文を書く手間が軽減される，最適化手法についての知識がない非熟練ユーザの補助となるといった利点がある．

前述の指示文ID番号 *PRAGMA_ID* はこの段階で自動的に生成されるものとする．

3.3 提案手法の利点

これらの機能が連携して動作し，2.2節で述べた問題点は次のように解消する．

(1) コード変換後も可読性が維持される．
- (a) コード変換後も元コードのみを確認すれば処理内容を判断でき，可読性が維持される．
- (b) 指示文ID番号により，元コードに施したコード変換と変換後のコードとの対応が明確になり，可読性が高い．

(2) 最適化における試行錯誤の作業量が少ない.
(a) コマンド入力によりコード変換されるため，コード変換の作業量が少ない.
(b) コマンド入力により元コードが復元されるため，元コードの復元の作業量が少ない.
(c) ソースコードを変更せずに変換後のコードを確認できるため，その際のコード変換・復元の作業が不要となる.
(d) コード変換指示文中のパラメタのみを変更できるため，最適化手法におけるパラメタを変更するための作業量が少ない.

4. 実　　装

提案手法を汎用エディタ上で動作する対話型OpenMPプログラム並列化支援ツールiPat/OMPのプログラムリストラクチャリング機能に適用することで，プログラム編集，並列化，性能最適化における試行錯誤をスムーズに行う開発環境を実現した．本章ではその実装方法について述べる.

4.1 実装環境

提案ツールを以下のソフトウェアを用いて実装した.

4.1.1 Omni OpenMP Compiler

iPat/OMPはバックエンド内でOmni OpenMP Compier[5]を使用している.

C言語用フロントエンドC-frontはC言語プログラムを可読性の高い抽象構文木構造の中間表現Xobject Codeへ変換する機能と，Xobject CodeをC言語プログラムへ変換する機能を持つ．変換されたXobject Codeは付属のJavaクラスライブラリ群Exc tool kitにより，解析や変換を行うことができる．iPat/OMPではExc tool kitを用いて，後述する並列化支援ライブラリを実現している.

4.1.2 iPat/OMP

iPat/OMPは汎用エディタGNU Emacs[6]上で動作する対話型OpenMPプログラム並列化支援ツールであり，逐次プログラムをOpenMPを用いた並列プログラムへ変更するための支援を行う．並列化支援機能群と，それらを対話的に利用するためのユーザインターフェースを備える.

iPat/OMPは，汎用エディタGNU Emacs，その機能を拡張するEmacs Lisp関数群，並列化支援の核となるバックエンドからなる．バックエンドは，既存Cコンパイラのプリプロセッサ，Omni OpenMP Compiler，支援ライブラリ，既存XSLTプロセッサで構成され，これらの処理をShell Scriptで統合・制御している.

iPat/OMP動作時，GNU Emacsのフレームは上下2ウインドウに分割されており，上部ウインドウはソースコード編集環境を提供し，下部ウインドウはバックエンドからの支援情報の表示を行う．本稿では以降，上部ウインドウをメインウインドウ，下部ウインドウをサブウインドウと呼ぶ.

ユーザはメインウインドウのプログラムに対し，Emacs Lisp関数群を通じて，バックエンドの並列性解析機能，OpenMP指示文作成機能，実行時間解析機能といった支援機能群を使用し，サブウインドウに表示された支援情報に基づいて並列化を行う.

またiPat/OMPでは，並列性解析の結果，並列性阻害要因が発見された場合，ユーザが知るプログラム特性を指示文により導入することで阻害要因を除去できる．つまり指示文によってユーザとの対話を行う機構を備えている．本研究ではこの機構を拡張し，指示文によるコード変換機能を実現する.

4.2 提案ツールの概要

提案ツールの実装概要を図3に示す．図の反転部はiPat/OMPに対して新たに実装した部分とその結果提示される支援情報，または拡張した部分である.

ユーザはコード変換指示文作成機能を実行後，性能に満足するまで，コード変換・個別挿入と元コード検出・個別復元を繰り返し実行する.

以降，各機能の詳細について述べる.

4.3 最適化支援機能群の実装

4.3.1 コード変換機能

コード変換指示文に対応する変換後のコードを生成する機能を実装した.

並列化が求められるような数値計算プログラムでは，実行時間の大半がループに費やされている．そこで今回はループ除去（loop expansion），ループ展開（loop unrolling），ループ交換（loop interchange），ループ分割（loop distribution）といったループリストラクチャリング機能を実装した.

ユーザが図4のようなコード変換指示文を対象ブロックの直前にコード変換指示文作成機能を用いて記述し，コード変換機能を実行すると，コード変換機能はExc tool kitを利用しXobject Code Fileを解析，指示文に対応する変換後のコードを生成し，変換後のコードと指示文ID番号と行番号を出力する.

ここで，*OPT_PRAGMA*は最適化手法，第1引数はその指示文固有のID番号，第2引数以下は各最適化手法におけるパラメタである.

4.3.2 元コード検出機能

変換後のコードの挿入後に，復元指示文の指示文ID番号に対応する元コードを検出する機能を実装した.

ユーザが変換後のコードを挿入すると，図5のような復元指示文が自動的に生成される．ユーザが元コード検出機能を実行すると，元コード検出機能がExc tool kitを利用しXobject Code Fileを解析，元コードの指示文ID番号と行番号を出力する.

ここで，第1引数は対応するコード変換指示文の指示文ID番号である.

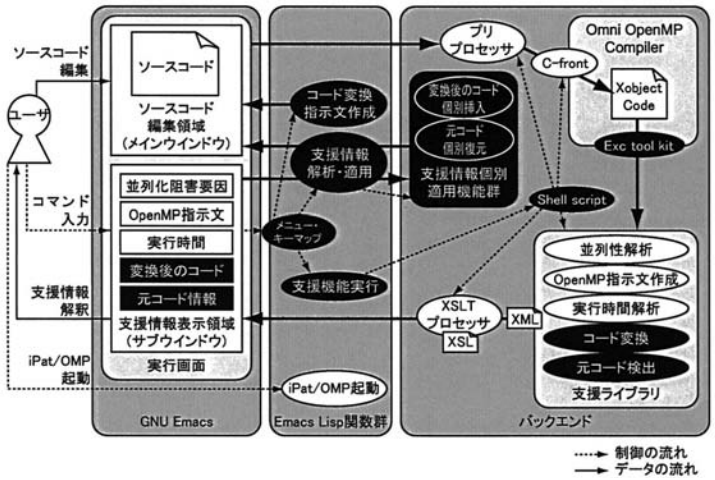

図 3　提案ツールの実装概要

```
#pragma ipat restructure \
OPT_PRAGMA(PRAGMA_ID, ...)
(元コード)
```

図 4　コード変換指示文

```
#pragma ipat restructure \
undo(PRAGMA_ID)
(変換後のコード)
```

図 5　復元指示文

```
/* original code ID:PRAGMA_ID
(元コード)
original code ID:PRAGMA_ID */
```

図 6　元コードのコメント形式

```
/* generated code ID:PRAGMA_ID
(変換後のコード)
generated code ID:PRAGMA_ID */
```

図 7　変換後のコードのコメント形式

4.4　支援情報個別適用機能群の実装

提案ツールにおいてユーザとの対話を行うために，最適化支援機能群から得られた支援情報を個別に適用する機能を実装した．

4.4.1　変換後のコード個別挿入機能

コード変換機能によって生成された変換後のコードをユーザに提示し，挿入の可否判断を仰ぎ，ユーザの可否判断後，ソースコードに挿入する機能を実装した．

コード変換機能により生成された変換後のコードと行番号と指示文 ID 番号がサブウインドウに提示され，ユーザのキー入力を待つ．キー入力があれば，個別の変換後のコードが一時ファイルに保存された後，変換後のコード挿入ルーチンによりソースコード内に挿入される．このとき元コードは図 6 のような形式でソースコード内にコメントとして保存される．

4.4.2　元コード個別復元機能

元コード検出機能によって検出された元コードの情報をユーザに提示，復元の可否判断を仰ぎ，ユーザの判断後，元コードをソースコードに復元する機能を実装した．

元コード検出機能により検出された元コードの行番号と指示文 ID 番号がサブウインドウに提示され，ユーザのキー入力を待つ．キー入力があれば，元コードの情報が一時ファイルに保存された後，元コード復元ルーチンにより元コードが元の位置に復元される．このとき変換後のコードは図 7 のような形式でソースコード内にコメントとして保存される．

4.5　Emacs Lisp 関数群の拡張

提案ツールにおいてユーザとの対話を行うため，Emacs Lisp 関数群を拡張した．

4.5.1　コード変換指示文作成機能

最適化手法の選択肢を提示し，選択された手法に対応するコード変換指示文を作成する機能を実装した．

最適化手法を選択すると，作成する指示文固有の ID 番号を自動生成し，その最適化手法におけるパラメタの入力をユーザに求め，コード変換指示文を作成する．

4.5.2　機能メニュー・キーマップ

各機能を GNU Emacs のメニューバーへ登録し，キーマップを作成した．

これにより，機能の周知と作業効率の向上を図る．

5.　評　　価

本章では，提案ツールの評価について述べる．

5.1　手作業と提案ツールとの作業量の比較

最適化における試行錯誤は，コード変換・元コードの復元の反復である．よってコード変換・元コードの復元の各々の作業量は最適化における試行錯誤の作業量に影響する．そこで，コード変換の作業量，元コードの復元の作業量を手作業と提案ツールとで比較する．

5.1.1　コード変換の作業量

Omni OpenMP Compiler に添付されている laplace 方程式の差分解法プログラムである lap.c を用いる．図 8 にこのプログラムのソースコードの一部を示した．このコードを 6 段ループ展開する際の作業量を手作業と提案ツールとで比較する．

図 9 に提案ツールでの実行手順を示した．

なお，提案ツールでは C 言語プログラムを中間表現 Xobject Code に変換する際に定数が数値に置換されるため，図 8 の定義に従って，定数 YSIZE=1000 として変換後のコードが生成されている．

- 手作業

(1) ループ展開の端数用に対象ループを複製．
(2) 上記ループのループ制御変数の初期値を変更．
(3) ループ制御変数の増分を展開段数を考慮した値に変更．

```
#define XSIZE 1000
#define YSIZE XSIZE

for(x = 1; x <= XSIZE; x++){
  for(y = 1; y <= YSIZE; y++){
    u[x][y] = (uu[x-1][y] + uu[x+1][y] + uu[x][y-1] + uu[x][y+1])/4.0;
  }
}
```

図 8 ループ展開の対象コード（lap.c）

(4) ループ内の各ステートメントを展開段数分複製.
(5) ループ内の各ステートメントに含まれるループ制御変数を段数に見合った値に変更. 作業終了.

- 提案ツール（図 9）

(1) コード変換指示文作成機能を実行し，状態 1 を得る.
(2) 最適化手法を選択，コード変換機能を実行し，状態 2 を得る.
(3) 生成された変換後のコードを確認し，挿入を指示し，状態 3 を得る. 作業終了.

よって手作業と比較し，提案ツールでは複雑な手順を踏まず，少ない作業量で変換後のコードを得られることが確認できた.

また，手作業では展開段数とステートメントに含まれるループ制御変数の量に比例して作業量が増加するが，提案ツールでは指示文のパラメタを変更するだけでよいため，作業量は一定である.

5.1.2 元コードの復元の作業量

次に，より最適なコードを模索するために，一度コード変換されたコードを破棄し，元コードの復元を行う際の作業量を手作業と提案ツールとで比較する.

- 手作業かつバージョン管理システムを併用する手法

(1) あらかじめ登録名やリリース名，ログなどを入力し，リポジトリに登録.
(2) リポジトリから元コードが記述されたファイルを得る（現在の変換後のコードは失われる）.

- 手作業かつ元コードをコメントアウトする手法

(1) コメント接頭辞を削除.
(2) コメント接尾辞を削除.
(3) 変換後のコードをコメントアウト.

- 提案ツール

(1) 元コード検出機能を実行.
(2) 元コードを確認，復元を指示. 作業終了.

よって手作業と比較し，提案ツールでは複雑な手順を踏まず，少ない作業量で元コードを復元できることが確認できた.

前項と本項により，提案ツールを用いるとソースコードレベルの最適化における試行錯誤を少ない作業量で行うことができることが示せた.

5.2 可 読 性

最適化における試行錯誤を行う際に，可読性が高い方が効率よく作業を行うことができる. そこで，提案

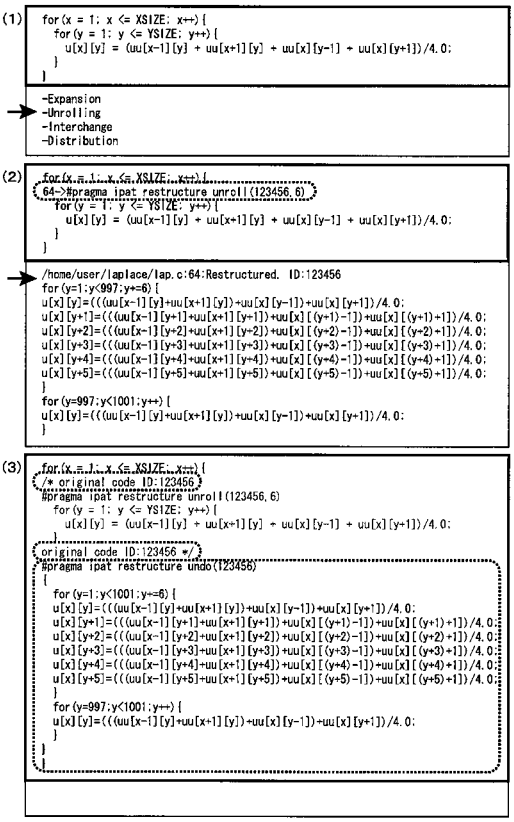

図 9 提案ツールでのループ展開機能の実行手順（lap.c）

表 1 プログラマが確認すべき行数

手作業	提案ツール
11	4

ツールにおけるコードの可読性を評価する.

5.2.1 元コードの参照

まず，元コードの参照の手間を評価する.

提案ツールでは，コード変換指示文と元コードをソースコード内に保持しているため，バックアップファイルを用いる方法やバージョン管理システムを用いる方法と比較し，容易に元コードを参照することができ，一覧性が高く，保持されているコード変換指示文から手法やパラメタの情報を即座に得ることができる.

5.2.2 プログラマが確認すべき行数

次に，前節と同様のソースコード（lap.c）を対象に，プログラマが確認すべき行数を比較する. 比較結果を表 1 に示した.

手作業でコード変換した場合，プログラマは自身が記述した変換後のコード全てを確認する必要がある.

```
for (i = 1; i < 10; i++) {
  a[i] = (a[i-1] + a[i]) / 2;
  b[i] = b[i] * 2 + 1;
}
```

図 10 iPat/OMP と提案ツールの連携効果の評価対象コード

一方，提案ツールを使用した場合（図 9），プログラマはコード変換指示文と元コードのみを確認すればよい．つまり，確認すべき行数の少なさに加え，ループ展開における端数の記述等，コード変換における処理の整合性を保つための記述を確認せずに済むという点で，可読性を維持できていると言える．

ここで，手作業かつ元コードをコメントアウトしつつコード変換した場合を考える．この場合，コメントアウトした元コードを参照しても，提案ツールと同様の効果を得ることはできない．なぜなら，元コードと変換後のコードとの対応関係が明確でなく，また施したコード変換の内容が不明であるためである．これに対し，提案ツールでは元コードと変換後のコードが固有の指示文 ID 番号で紐付けられおり，対応関係が明確になっているため，前述の効果が得られている．

5.3 iPat/OMP の並列化支援機能との連携

iPat/OMP の並列化支援機能と提案ツールの連携による効果について，図 10 のソースコードを対象として評価する．

従来，このようなループを iPat/OMP で自動的に並列化することはできなかった．配列 a への代入文にイタレーション間の依存が存在するためである．しかし，提案ツールを用いてループ分割すれば，一方は並列化することができる．つまり，ループ内の 2 つの代入文間には依存が存在しないため，このループを各々の代入文を含むループに分割できる．その結果，配列 b への代入文を含むループにはイタレーション間の依存が存在しなくなり，並列化が可能になる．

提案ツールにより，従来 iPat/OMP で並列化できなかったソースコードを並列化することが可能になった．

6. おわりに

本稿では，ソースコードレベル最適化における新しい試行錯誤支援手法を提案し，対話型並列化支援ツール iPat/OMP 上のプログラムリストラクチャリング機能に適用し，実装を行った．

提案手法は，ユーザとの対話に基づいて指示文を用いたコード変換・復元機能を提供することで，性能最適化における試行錯誤をスムーズに行うことを可能にし，ユーザがプログラムの処理の本質的な部分に集中することを促進するものである．この手法を iPat/OMP 上で実現することで，プログラム編集，並列化，性能最適化における試行錯誤をスムーズに行う開発環境の実現を目指した．そこで，コード変換機能，元コード検出機能，変換後のコード個別挿入機能，元コード個別復元機能，コード変換指示文作成機能を実装した．

提案手法の仕様による効果を評価した結果，コード変換後も元コードを参照すれば処理内容を把握することができ，コードの可読性を維持できていることを確認した．また，ループ展開を例にとり，従来の開発環境と作業量の比較を行った．その結果，提案ツールを用いるとコード変換と元コードの復元の双方を少ない作業量で行うことができ，それらの反復である最適化における試行錯誤を少ない作業量で行えることを確認した．以上より，最適化における試行錯誤において，提案手法が有効であることが示せた．

本研究の今後の課題は以下の通りである．

(1) さらなる可読性の向上
提案手法では元コードのみを参照すればよいことを前項で示した．これに加え，変換後のコードを適宜隠蔽するための機構を実装すれば，さらに可読性を向上させることができると考えられる．

(2) 最適化手法の選択における補助の必要性
提案ツールでは最適化手法の選択をユーザ自身が行うことにより，無意味なコード変換を防止している．しかし，プログラム解析，実行時間解析，実行環境情報，ユーザの選択履歴などを利用すれば，適用すべき手法の候補を絞り込むことができる．これを利用した最適化手法推薦機能を提案ツールに付加すれば，より効率的に最適化作業を行うことができるようになると考えられる．

謝辞 本研究を進めるに当たり，ご助言をいただいた筑波大学 佐藤三久教授に心より感謝いたします．

参考文献

1) Irigoin, F., Jouvelot, P. and Triolet, R.: Semantical interprocedural parallelization: an overview of the PIPS project, *ICS '91*, ACM Press, pp.244–251 (1991).
2) Evans, E. W., Johnson, S. P., Leggett, P. F. and Cross, M.: Automatic and effective multi-dimensional parallelisation of structured mesh based codes., *Parallel Computing*, Vol.26, No.6, pp.677–703 (2000).
3) ParaWise: `http://www.parallelsp.com/`.
4) Ishihara, M., Honda, H. and Sato, M.: Development and Implementation of an Interactive Parallelization Assistance Tool for OpenMP: iPat/OMP, *IEICE Transactions on Information and Systems*, Vol.E89-D, No.2, pp.399–407 (2006).
5) Sato, M., Satoh, S., Kusano, K. and Tanaka, Y.: Design of OpenMP Compiler for an SMP Cluster, *EWOMP '99*, pp.32–39 (1999).
6) GNU Emacs: `http://www.gnu.org/`.