ソフトウェア制御オンチップメモリ向け自動最適化コンパイラの提案

藤田元信[†,††] 近藤正章[††,†] 中村宏[†]

近年のプロセッサとメモリの性能差の問題に対応するため，従来のキャッシュメモリに加えソフトウェア制御可能なオンチップメモリを搭載するアーキテクチャが提案されている．しかし，ソフトウェア制御可能なオンチップメモリを利用した高性能化では従来のキャッシュでは自動的に行われていたデータ配置や入替え，転送のスケジューリングをプログラマが行わなければならず，アプリケーションの最適化にともなうユーザの負荷が問題となる．そこで本論文では自動最適化コンパイラの 1 つのアプローチとして，“ヒント情報に基づく自動最適化コンパイラ” を提案する．本コンパイラでは，ソフトウェア制御メモリの制御方法やパラメータなどのアーキテクチャをユーザが意識することなく，配列データの再利用性の有無といったアプリケーションに関する情報のみをヒント情報として与えるだけで最適化を可能にすることを目標としている．本コンパイラを実装しいくつかのアプリケーションに適用した結果，提案手法によりユーザの負荷を大幅に減らせること，および性能評価を通じて従来の方法で最適化したものと同等の性能が得られることが分かった．

Automatic Compilation for Software-controlled On-chip Memory

Motonobu Fujita,[†,††] Masaaki Kondo[††,†] and Hiroshi Nakamura[†]

In order to overcome performance degradation caused by performance disparity between processor and main memory, there have been proposed several new VLSI architectures which have software controlled on-chip memory in addition to the conventional cache. However, users must specify data allocation/replacement on software controlled on-chip memory and data transfer between the on-chip and off-chip memories to achieve higher performance by utilizing on-chip memory. Because such properties are automatically controlled by hardware in conventional caches, a cost of optimization for a program becomes a matter that should be considered. In this paper, we propose an automatic optimizing compiler based on “Optimization Hint Informations”. Using proposed compiler, users can optimize programs only providing hint informations for data reusability without any knowledge of architecture details. We evaluate the performance and cost of programming for our compiler using two applications. The results reveal that the proposed compiler can drastically reduce the programmers' burden and achieve high performance.

1. はじめに

近年，マイクロプロセッサの性能は飛躍的に向上しており，サイクル時間が数 ns あるいはそれ以下のものも登場している．一方，主記憶として広く用いられている DRAM は集積密度の向上こそ認められるものの，アクセス時間に関しては低い改善率にとどまっており，現在広く使われている Direct Rambus DRAM[1] においても 40 ns 程度である．このように，プロセッサと主記憶の性能差は相対的に拡大する方向にある．

この問題に対応するため，データの空間的・時間的局所性に基づいたデータの再利用を可能にするキャッシュメモリが広く用いられている．しかし，科学技術計算をはじめとして多くの主記憶アクセスをともなうアプリケーションでは，データセットサイズがキャッシュサイズに比べて大きく時間的局所性が活用できない，あるいは複数のデータがキャッシュ上の同一ブロックにマッピングされてしまうことによりお互いをキャッシュ上から追い出すなどキャッシュが有効に機能しない場合があることが知られている．キャッシュブロッキング[2] などデータアクセスの時間的局所性を向上さ

† 東京大学先端科学技術研究センター
Research Center for Advanced Science and Technology, The University of Tokyo
†† 独立行政法人科学技術振興機構
Japan Science and Technology Agency

せる手法，キャッシュ上の競合を防ぐためにデータ配置を変更する手法[3] をはじめとして，キャッシュの利用効率を改善する手法も提案されているが，データの配置および入替えがすべてハードウェアによって自動的に決定されるキャッシュでは，このようなソフトウェア的な手法により完全に最適化することは難しい．

そこで，キャッシュメモリの動的な再構成を行い，その一部をソフトウェアにより入替え制御可能なオンチップ RAM として利用できるアーキテクチャ[4] や，上記機能に加え，従来のキャッシュの一部をロックすることでハードウェアによる自動的な入替えを抑制し，ロックされたデータの再利用性を活かす機能を備えたアーキテクチャ[5] が提案されている．また，我々もソフトウェアによりアドレス指定可能なメモリをチップ上に搭載するアーキテクチャ（*Software Controlled Integrated Memory Architecture*）[6] の提案を行っている．SCIMA では，チップ上のソフトウェア制御メモリを頻繁に使われるデータの一時的な格納領域として使うだけではなく，これを用いてオフチップメモリとのデータ転送を最適化することを目的としている．

しかし，ソフトウェア制御オンチップメモリを用いた高性能化を達成するためには，プログラマがソフトウェア制御可能メモリのデータ配置や入替え，データ転送のスケジューリングなども行う必要がある．これらは従来のキャッシュではハードウェアにより自動的に行われていたものであるため，アプリケーションの最適化にともなうユーザの負担が問題となる．

そこで，ソフトウェア制御オンチップメモリを用いた高性能化をより多くのアプリケーションが享受するためにはソフトウェア制御可能なメモリと主記憶間のデータ転送を自動的に最適化する自動最適化コンパイラが重要となる．

本論文では自動最適化コンパイラの 1 つのアプローチとして，ソフトウェア制御オンチップメモリの最適化に必要なアプリケーションに関する情報をプログラマが “最適化ヒント情報” として記述するだけで最適化を行う，“最適化ヒント情報に基づく自動最適化コンパイラ” を提案する．本手法では，ソフトウェア制御オンチップメモリの制御方法はユーザから隠蔽され，プログラマは従来のキャッシュ向け最適化においても考慮してきたデータの再利用性の有無のみをヒント情報ディレクティブとしてソースコードに挿入するだけで最適化を行うことができる．

本コンパイラは，ソフトウェア制御可能なメモリを搭載するアーキテクチャの 1 つである SCIMA を対象として作成したが，再利用性のあるデータの最適化手法については他のソフトウェア制御可能メモリを持つアーキテクチャに対しても適用できるものと考えられる．

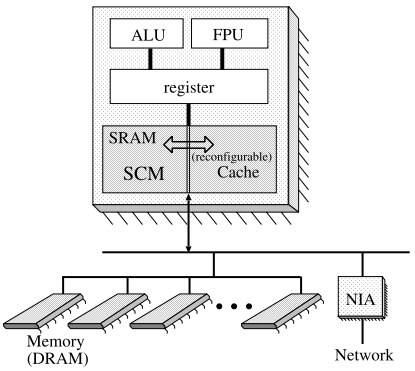

図 1 SCIMA の構成
Fig. 1 SCIMA.

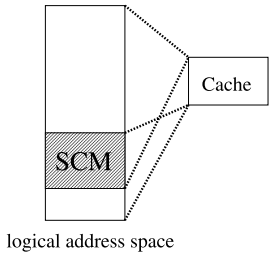

図 2 SCIMA におけるアドレス空間
Fig. 2 Address space.

2. SCIMA

2.1 SCIMA のアーキテクチャ

SCIMA は，チップ上に従来のキャッシュに加えソフトウェア制御可能なメモリ *SCM*（*Software Controlled on-chip Memory*）を搭載する（**図 1**）．SCM は論理アドレス空間の一部の連続した領域を占めており，キャッシュと SCM の間にアドレス空間の包含関係はない（**図 2**）．キャッシュと SCM はハードウェアとしての SRAM 自体は共有し，アプリケーションの性質に応じ，その容量比を動的に変更させることも可能である[7]．

キャッシュと SCM では，データアロケーション，リプレースメントの方式が異なる．キャッシュはハードウェア制御により自動的にデータ配置，置き換えが行われるのに対し，SCM では，ソフトウェアから明示的にデータ配置，置き換えを行う．

2.2 SCIMA の拡張命令

SCIMA では，SCM と主記憶間のデータ転送を行う *page-load/page-store* 命令を備え，SCM のデータ

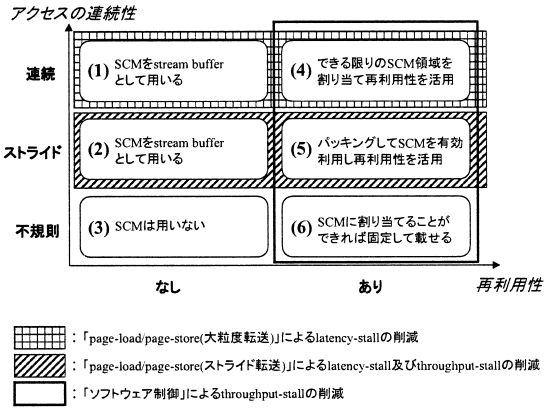

図 3 配列の特徴に対する SCM の利用戦略
Fig. 3 Strategy for using SCM.

配置，置き換えはこの命令で行う．

本命令は，データ転送元の開始番地，データ転送先の開始番地，転送サイズ，ブロック幅，ストライド幅，の 5 オペランドをとる．SCM 領域は数 KB の *page* と呼ばれる単位に分割して管理を行い，*page-load/page-store* 命令はこの *page* を最大サイズとした大粒度転送を可能にする．さらに，本命令はブロックストライド転送機能を持つ．これにより，主記憶バンド幅を無駄にすることなく必要なデータのみを SCM 上に転送することができる．

2.3 SCIMA 向け最適化戦略

過去に行われた SCIMA の有効性に関する評価において，特に配列に対し SCM を用いてアクセスすると性能向上が期待できることが分かった．そこで我々はアクセスの連続性とデータの再利用性に配列の特徴を整理し，その分類に基づいた SCM の利用戦略を提案している[7]．具体的には**図 3** のように SCM を用いることで性能向上を図る．以下に，主に再利用性の観点から説明を行う．

再利用性のないデータに関しては，SCM の一部をストリームバッファとして割り当て，バッファを単位とした転送を行うことで，レイテンシの削減効果ならびにチップ上記憶領域の効率的な利用が期待できる．なお，*page-load* 命令によって一度に転送できる最大サイズは *page* であるため，1 つのバッファに対し確保する SCM 領域サイズは *page* サイズ以下が妥当である．一方，再利用性のあるデータに関しては，SCM 上に大きなワーキング領域を割り当て，従来キャッシュ上で発生していた競合を防ぐことで，再利用性をより確実に引き出すことができる．

ここで，再利用性のないデータに対しキャッシュは無力であるため，再利用性のない配列への最適化を優先して行うことが効果的である．

```
integer N
double precision x(N), y(N), sum
integer i

sum = 0.0d0

do i = 1, N
  sum = sum + x(i) * y(i)
enddo
```

図 4 ベクトル内積計算
Fig. 4 Inner product of two vectors.

2.4 SCIMA ディレクティブベースコンパイラ

前節で述べた最適化戦略を実現するために，SCM と主記憶間のデータ転送を表現する SCIMA ディレクティブと，それを解釈するディレクティブベースコンパイラを開発した[8]．ディレクティブベースコンパイラは，*Omni OpemMP Compiler*[9] の持つディレクティブ解釈の枠組みをベースに開発したもので，SCIMA ディレクティブを，*page-load/page-store* に相当する関数呼び出しに変換する機能を持つ．

SCIMA ディレクティブでは，SCM 上の領域確保および SCM と主記憶のデータ転送をいくつかの引数を用いて指定する（引数の詳細については文献 8) を参照）．主記憶（キャッシュ）の参照から SCM 領域の参照への変換にともなうアドレスの変換は自動的に行われるため，SCM 領域内のアドレス管理はユーザ自身が行わなくてもよい．

図 4 に示すベクトル内積計算を，SCIMA ディレクティブを用いて前節の最適化戦略に従い最適化を行う場合を考える．図 4 の配列 $x(N)$，$y(N)$ はともにデータの再利用性がなく連続アクセスを行う配列である．そこで，図 3 の最適化戦略に従い，SCM 上にストリームバッファ領域を設け，このバッファを単位として主記憶へのアクセスを行い，主記憶のレイテンシに起因するストール時間の削減を図る．ここで，バッファのサイズは *page* 以下になるように設定する．

これを実現するため，オリジナルのコードに対しユーザが以下のことを行う．

i *page* サイズを考慮に入れ，ストリームバッファサイズを設定する（**図 5** 中の *SBUF*）．
ii 配列 x，y のアクセスに用いるストリームバッファとして，サイズ SBUF の SCM 領域をそれぞれに対して確保するための SCIMA ディレクティブをループ直前に挿入する．
iii SBUF を単位とした SCM アクセスを行うためにループ細分を行う．
iv 細分化後の最内ループの計算に必要なデータを

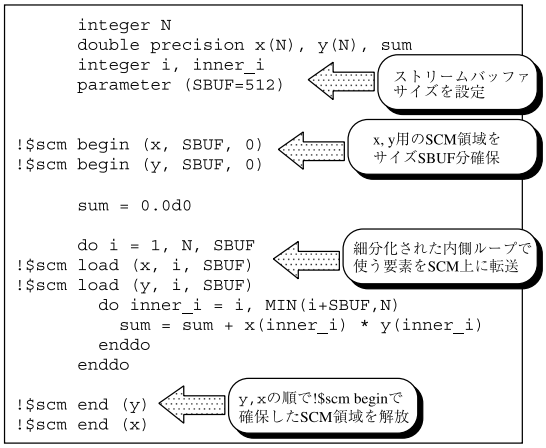

図 5 SCIMA ディレクティブを用いたプログラム例
Fig. 5 Optimized code with SCIMA directives.

SCM 上に転送するディレクティブを最内ループの直前に挿入する．

v 計算を終了した後は配列 x，y のために確保したストリームバッファは必要なくなるため，ループ終了直後に確保した SCM 領域を解放するディレクティブを挿入する．

以上の過程を経て最適化されたコードを図 5 に示す．

3. ヒント情報に基づく最適化

3.1 ディレクティブベースコンパイラの問題点

SCIMA ディレクティブを用いることで，元のソースコードのセマンティクスを変更することなく SCM を用いたプログラミングが可能となった．しかしながら，前章の例に見られるように SCIMA ディレクティブを用いて最適化にはいくつかの問題点がある．

- ユーザがつねに利用可能な SCM 容量を把握しておく必要がある．
- ユーザがあらかじめ SCM の *page* サイズを把握しており，これを単位とした計算を行うためループ細分を行わなければならない．
- ユーザがループ中でアクセスされる範囲を把握し，これらをディレクティブの引数として記述しなければならない．
- ユーザがデータアクセススケジューリングを考慮し，最適な SCIMA ディレクティブの挿入位置を決定しなければならない．

このように，SCIMA 向け最適化では，SCIMA 特有の知識・情報をもとにしたプログラミングが必要であり，キャッシュ向けの最適化で必要なデータアクセスパターンなど，アプリケーション自身に関する情報以外にも多くの知識を必要とする．また，プログラム自体が SCM 容量に依存しており，性能に可搬性がない．多くのユーザの立場を考えた場合，このような SCIMA に特有の情報を意識することなくプログラミングを行えることが望ましい．そこで，SCM の容量や主記憶間とのデータ転送をユーザに意識させることなく，SCM を利用し高性能化を達成するコードを生成する最適化コンパイラが必要となる．

3.2 ヒント情報

3.2.1 ヒント情報の利用方針

前節の問題点を解決するため，SCIMA のハードウェア構成に特有な情報を与えることなくプログラムを作成する際に，ユーザが意識しているであろうアプリケーションのみから分かる情報を “ヒント情報” としてユーザがコンパイラに与えることとし，そのヒント情報に基づいて SCM を用いた性能最適化を行う “ヒント情報に基づく自動最適化コンパイラ” を提案する．

最適化方針の決定の基準となる配列の再利用性についての判断は入力データセットサイズによって容易に変化しうるため，最適化対象コードの解析のみに基づきコンパイラが判断することは難しいと考えられる．しかし，データの再利用性についてはキャッシュ向け最適化としてユーザがこれまでも考慮してきたことであり，ユーザがこれをコンパイラの入力として与えることは新たな負担とはならず，十分可能であると考えられる．

一方，最適化範囲ループの指定および最適化対象配列の指定に関しては，プログラム全体にわたる解析を行い，最適化対象となる配列へのアクセスを行うループ部分を選択する必要があるが，HPC 分野のアプリケーションでは実行時間の多くをコード中の限られた部分が占めていることが多く，従来の最適化でもその部分を中心として最適化を行うため，これをユーザが発見することは難しくないと考えられる．

提案する自動最適化コンパイラのための，ヒント情報の仕様を**図 6** に示す．本ヒント情報は，アプリケーションの性質に関する情報として配列の再利用性に関する情報のみをディレクティブの引数として持ち，SCM の存在を意識することなく記述できるように設計している．

3.2.2 ヒント情報を利用したプログラム例

従来の SCIMA ディレクティブを用いた最適化と，今回提案する “最適化ヒント情報” を用いた最適化とを比較するため，前章の図 4 で示したベクトル内積計算を “最適化ヒント情報” を用いて最適化を行ったものを**図 7** に示す．

図 7 のヒント情報を用いたプログラムでは，ユーザ

!$scm opt_notreuseable**(<配列名>) / **!$scm opt_end(<配列名>)
意味：本ディレクティブで囲まれた範囲において指定された配列を再利用性のない配列と見なし，最適化を行う．

!$scm opt_reuseable (<配列名>) / **!$scm opt_end**(<配列名>)
意味：本ディレクティブで囲まれた範囲において指定された配列を再利用性のある配列と見なし，最適化を行う．

!$scm element
意味：ユーザがブロッキングを行ったループに関して，エレメントループとブロッキングループの境界を示す．

図 6 SCIMA 自動最適化ヒント情報
Fig. 6 Hint directives for automatic optimization.

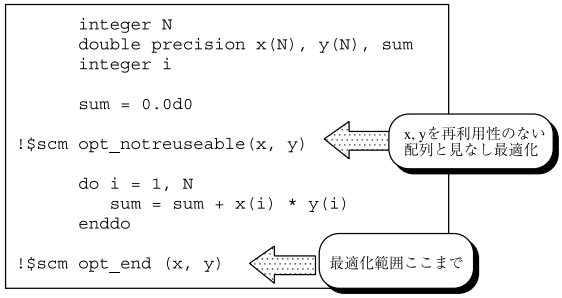

図 7 ヒント情報によるプログラム例
Fig. 7 Optimized code with Hint Infomation.

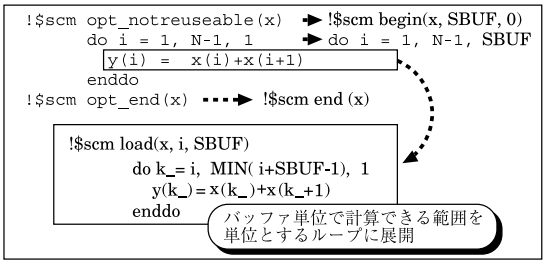

図 8 再利用性のない配列の最適化の例
Fig. 8 Optimization for non-reusable arrays.

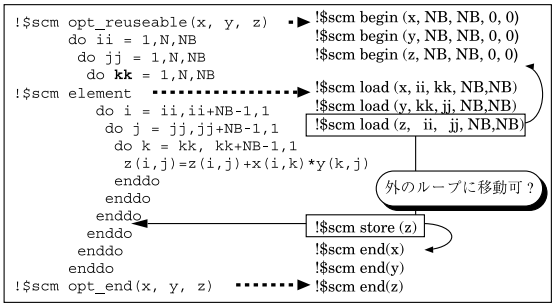

図 9 再利用性のある配列の最適化の例
Fig. 9 Optimization for reusable arrays.

は最適化範囲を指定し，最適化を行う対象配列とユーザの判断による配列の再利用性の有無の情報を挿入するのみである．引数の中に SCIMA 特有の情報は含まれていない．

従来の SCIMA ディレクティブを用いて最適化を行ったコード（図 5）とヒント情報を用いて最適化を行ったコード（図 7）を比較すると明らかなように，ヒント情報による SCIMA 最適化プログラミングではオリジナルのコードに対してヒント情報として 2 行のディレクティブを追加するのみであり，ユーザの負担は大きく減少する．したがって，提案するヒント情報による最適化は SCM を利用した高性能化を広く一般のプログラムに適用させるために，大変重要であると考えられる．

3.3 最適化アルゴリズム

本節では，提案コンパイラの最適化アルゴリズムについて述べる．本アルゴリズムは，プログラムの静的な解析およびプログラム中に挿入されたヒント情報に基づきコード変形および SCIMA ディレクティブの挿入を行う．図 3 に示した SCIMA 向け最適化戦略のうち，配列データの再利用性に関してはヒント情報で与えられ，アクセスの規則性についてはコンパイラがコードの静的な解析に基づいて判断し，最適化方針を決定する．

また，本アルゴリズムにおける最適化は，再利用性のない配列への最適化（**図 8**），再利用性のある配列への最適化（**図 9**）の順で行う．それぞれの最適化の内容については以下に詳述する．

3.3.1 再利用性のない配列に対する最適化

ユーザが再利用性なしと判断した配列に関しては，SCM 上に小容量のバッファ領域を設け，このバッファサイズを単位として大粒度でのオフチップメモリアクセスを行い，オフチップレイテンシの影響削減を図る．

ステップ 1：SCM 割当てサイズの決定 *page-load* 命令により一度にバースト転送できる最大サイズは SCM 領域の管理の単位でもある *page* である．したがって，主記憶上での連続方向に *page* サイズ以上のバッファを確保したとしても性能向上は見込めない．そこで，確保するバッファのサイズは主記憶上での連続方向の次元については *page* サイズに設定し，これを最内ループにおける主記憶上の不連続方向の次元のアクセス範囲の上限から下限を引いた数だけ用意する．

なお，各配列に対するバッファ領域はヒント情報

として宣言された順に確保するが，途中で SCM 容量が不足した場合には，以降の配列に SCM 領域を割り当てることを諦め，これらの配列はキャッシュを経由してアクセスする．

ステップ 2：ループ細分 多くのコードでは，ループ中でアクセスする範囲は先に設定したバッファサイズよりも大きい．そこで，バッファサイズを単位とした計算を行うため，最内ループの細分を行う．

ステップ 3：ディレクティブの挿入 次に，細分化後の最内ループの直前に当該ループ中で必要とされるデータを SCM 上に転送するディレクティブの挿入を行う．読み込みだけを行う配列については値を主記憶に書き戻す必要がないため，主記憶から SCM へのデータ転送を行うディレクティブのみを細分化後の最内ループの直前に挿入する．このとき，ディレクティブの引数は最適化対象範囲に含まれるループの静的な解析に基づき決定する．同様に，主記憶へのデータ書き戻しだけを行う配列については値を主記憶から読み込む必要がないため，SCM から主記憶へのデータ転送を行うディレクティブのみを細分化後の最内ループの直後に挿入する．読み込みと書き戻しの両方を行う配列については両方のディレクティブの挿入を行う．最後に，ヒント情報の挿入位置でもある最適化範囲の開始点と終点にそれぞれバッファ領域確保および解放のためのディレクティブを挿入する．この場合もディレクティブの引数は最適化対象範囲に含まれるループの静的な解析に基づき決定する．

3.3.2 再利用性のある配列に対する最適化

再利用性がある配列を含むループはユーザによりブロックサイズを変数（図 9 中の NB）としてブロッキングが行われていることを前提とする．ブロックサイズはループレベルによらず共通である．ブロッキング自体はキャッシュ向け最適化として広く使われている手法であるため，この前提は妥当と考えられる．

この前提のもとで，データの再利用性ありとユーザが判断した配列を SCM 上に載せることで競合を防止しつつ再利用性の活用を図る．

ステップ 1：SCM 割当てサイズの決定 入力されるコードでは，ブロックサイズが変数（仮に NB とする）として与えられる．まず，ユーザが再利用性ありと判断した各配列について，ブロッキングループとエレメントループの境界（ヒント情報 `!$scm element` で与えられる）より内側のループでアクセスされる領域サイズを NB を用いて表す．次に，これらの合計と，利用可能な SCM の容量の関係から，NB の値を決定する．ここで，利用可能な SCM の容量とは全 SCM 容量から再利用性のない配列の最適化で使われた SCM 容量を差し引いた容量である．仮に，この最適化より先に行われる再利用性のない配列の最適化で SCM 容量を使いきってしまった場合には，再利用性のある配列はキャッシュを用いたアクセスに切り替え，以降のディレクティブ挿入は行わず利用可能なキャッシュ容量をもとに NB の値の決定のみを行う．

ステップ 2：ディレクティブの挿入 次に，上記で決定されたブロッキングサイズをもとに，エレメントループ中で必要とされるデータを SCM 上に転送するディレクティブおよび SCM 上から主記憶にデータを書き戻すディレクティブをエレメントループ直前に挿入する．これらのディレクティブの引数はコードの静的な解析および前のステップで計算されたブロック数から決定する．最後に，ヒント情報の挿入位置でもある最適化範囲の開始点と終点にそれぞれ領域確保および解放のためのディレクティブを挿入する．この場合もディレクティブの引数は最適化対象範囲に含まれるループの静的な解析に基づき決定する．

ステップ 3：ディレクティブの挿入位置変更 SCM と主記憶間のデータ転送を行うディレクティブをすべてブロッキングループとエレメントループ境界に挿入すると転送データの重複が発生してしまいオフチップトラフィックが増加し性能が低下してしまう．そこで，データ転送の重複を防ぐため先に挿入したデータ用転送ディレクティブのうち，同じ要素を SCM 上に転送しているディレクティブの挿入を 1 つ外のループレベルに移動させる．

4. 実　　装

以上に示した最適化アルゴリズムの実装を行った．提案コンパイラによる処理の流れを**図 10** に示す．

本コンパイラは，最適化ヒント情報が挿入された Fotran77 のソースコードを入力として受け取る．まず，フロントエンドにより入力ソースコードを *Omni OpemMP Compiler*[9] でも使われている中間言語表現 Xobject Code に変換する．次に，先に紹介した最適化アルゴリズムによりコードの変形およびヒント情報の SCIMA ディレクティブへの展開を行う．

続いて，すでに完成している SCIMA ディレクティブベースコンパイラ[8] の機能を利用し，挿入された SCIMA ディレクティブを SCIMA の *page-load/page-*

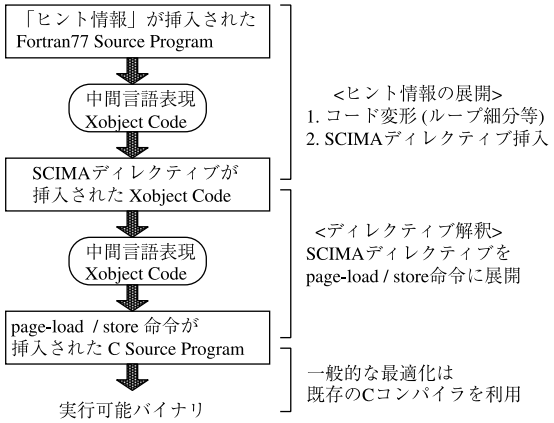

図 10 SCIMA 最適化処理の流れ
Fig. 10 Optimization flow.

store に対応する関数呼び出しに変換する．最後に，C ソースコードへの変換を行う．

以上の過程により得られた C ソースコード中には，SCM と主記憶間のデータ転送のための関数呼び出しが挿入されている．この C ソースコードをコンパイルすると，SCM を使用した最適化が行われた実行可能バイナリが得られる．

5. 評　　価

5.1 評価環境

提案するコンパイラの有効性を確認するため，再利用性のない配列に対する評価対象として，SPEC 171.swim の一部，再利用性のある配列に対する評価対象として行列積（倍精度 200×200）を用いて性能評価を行った．

評価にはサイクルレベルシミュレータを用いた．従来のキャッシュ向けに最適化を行ったコード（Cache）をもとに，SCIMA ディレクティブを挿入しディレクティブベースコンパイラを用いてコンパイルしたコード（SCIMA_directive）と，最適化ヒント情報のみを挿入し提案する自動最適化コンパイラを用いてコンパイルしたコード（SCIMA_HintInfo）を準備した．

Cache については，ラインサイズの変更による影響を評価するため 2 通りのラインサイズについて評価を行った．SCIMA_directive，SCIMA_HintInfo についてもスカラ変数などキャッシュを利用するデータもあることから，Cache と同様に 2 通りのラインサイズについて評価を行った．ただし，SCIMA 向け最適化コードで挿入するディレクティブはキャッシュのラインサイズに依存せず同じである．

評価における共通のパラメータを**表 1** に示す．それぞれのコードに対応する SCM とキャッシュメモリの構成について**表 2** に示す．いずれのコードに関しても，ブロッキングはユーザが行うことを想定しており，行列積の評価コードではあらかじめキャッシュブロッキングおよびレジスタブロッキングを行っている．

表 1 評価に用いたパラメータ
Table 1 Evaluation parameters.

キャッシュラインサイズ	32 B, 128 B
キャッシュway 数	4-way
オフチップメモリスループット	2 B/cycle
オフチップメモリレイテンシ	80 cycle

表 2 キャッシュと SCM の構成
Table 2 Configuration of cache and SCM.

	キャッシュサイズ	SCM サイズ
Cache	64 KB (4-way)	0
SCIMA_directive	16 KB (1-way)	48 KB
SCIMA_HintInfo	16 KB (1-way)	48 KB

本論文では，プログラムの実行時間を CPU-busy time（T_b），Latency-stall（T_l），Throughput-stall（T_t）の 3 つに分類する．CPU-busy time はプロセッサが計算処理を行っている時間であり，Latency-stall は主記憶のアクセスレイテンシがもたらすストール時間を，Throughput-stall はオフチップメモリのスループット不足に起因するストール時間を表す．

ここで，プログラムの総実行時間を T，オフチップメモリスループットを無限大と仮定した場合の実行時間を T_∞，オフチップメモリスループットが無限大かつオフチップメモリレイテンシが 0 とした場合の実行時間 T_p とする．この T，T_∞，T_p を用い，本論文では T_b，T_l，T_t を以下のように定義する．

$$T_b = T_p$$
$$T_l = T_\infty - T_p$$
$$T_t = T - T_\infty$$

評価を公平に行うために，ストリームバッファ領域を複数用いてデータ転送と演算をオーバラップさせる最適化は行っていない．また，プリフェッチは行っていないが，今回使用したシミュレータはリザベーションステーションを用いた out-of-order 実行機構をサポートしていることから，プリフェッチと同様の効果があるものと考えられる．

5.2 評価結果

評価結果を**図 11** と**図 12** に示す．まず，従来のキャッシュ向けに最適化された Cache とディレクティブ挿入と手動のコード変形により SCIMA 向けに最適化された SCIMA_directive を比較すると，両アプリケーションともに SCIMA_directive は Cache に比べて高い性

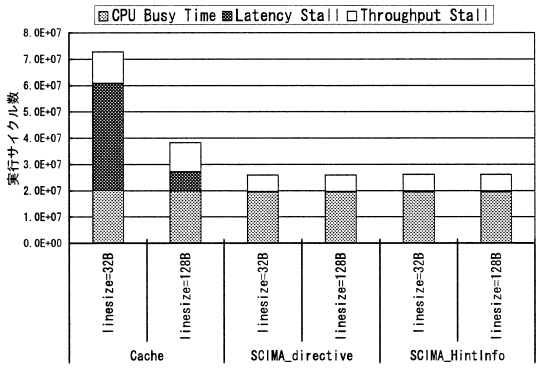

図 11 評価結果（SPEC 171.swim CALC1）
Fig. 11 Evaluation result (SPEC 171.swim CALC1).

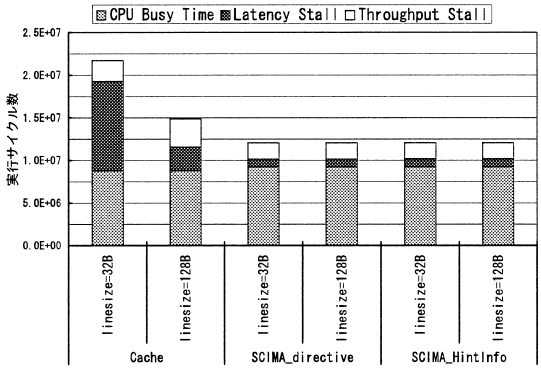

図 12 評価結果（行列積 倍精度 200 × 200）
Fig. 12 Evaluation result (matrix multiplication 200 × 200 in double precision).

能を示していることが分かる．これは，大粒度転送によりオフチップメモリレイテンシに起因するストール時間が減少したことや，SCM を用いることで再利用性を最大限活用できた効果である．

SCIMA 向けに最適化されたコードについて，Latency-Stall の割合は行列積の方が多くなっているが，これは行列積では二次元のブロッキングを行っており，連続に転送できる領域が 171.swim と比較すると小さくなっていることに起因すると考えられる．

また，行列積では，SCIMA_directive および SCIMA_HintInfo で CPU Busy Time がおよそ 5.5% 増加している．これは SCIMA ディレクティブの仕様により，二次元のブロッキングにより生じた小行列を 1 命令で転送できず，複数の *page-load* に展開し命令数が増加したことに起因すると考えられる．

それぞれのアプリケーションに対する SCM 容量の利用状況について述べる．171.swim では再利用性のない配列の最適化に用いるバッファとして 48 KB のうちおよそ 40 KB を利用している．また，行列積については，3 つの行列それぞれを二次元にブロッキングして各ブロックをすべて SCM に割り当てており，48 KB のうちおよそ 46 KB を利用している．

表 3 記述量に関する比較［行数］（171.swim）
Table 3 Comparison of the number of modified line for the optimization.

	変更	追加	挿入ディレクティブ
SCIMA_directive	8	9	21
SCIMA_HintInfo	0	0	2

次に，SCIMA_directive と，最適化ヒント情報のみを挿入し提案するコンパイラによって最適化を行った SCIMA_HintInfo を比較する．SCIMA_HintInfo は SCIMA_directive とほぼ同等の性能を示している．この結果より，提案するコンパイラを用いることで，最適化ヒント情報のみで実際に高性能を達成できることが分かる．次節では，SCIMA_directive と SCIMA_HintInfo に関し，最適化プログラミングにおけるユーザ負荷について議論する．

5.3 考　　察

本節では，記述容易性の観点から “最適化ヒント情報” を用い自動最適化コンパイラにより最適化されたコード（SCIMA_HintInfo）と従来の手動で最適化されたコード（SCIMA_directive）の比較を行う．

例として，**表 3** に SPEC 171.swim の最適化にともなう記述量の比較を示す．表より SCIMA_HintInfo では SCIMA_directive に比べて非常に多くの記述量を削減できることが分かる．従来の最適化では，SCIMA ディレクティブの挿入に加え，ループ細分などを手動で行う必要があり，元のコードに対し手を加える必要がある．一方，提案手法である “最適化ヒント情報” を用いた SCIMA 向け最適化ではオリジナルコードに対し，アプリケーションに依存する情報のみに基づきヒント情報ディレクティブを挿入するだけでよい．また，再利用性のある配列を含むプログラムの例としてあげた行列積に関しても同様に，記述量は提案手法の方が少ない（図 9）．

以上より，本論文で提案するコンパイラを用いることで，アプリケーションの最適化にともなうプログラミングのコストを大幅に減少させつつ，SCIMA ディレクティブを用いた従来の最適化手法と同等の性能を得ることが可能であると考えられる．

6. 関 連 研 究

組み込み分野においては，記憶容量の制約が厳しい，リアルタイム制約条件に合致しなければならない，消費電力の制約が厳しいなどの理由からスクラッチパッドメモリ（Scratch-Pad Memory，SPM）をはじめと

したソフトウェア制御可能メモリを備えたプロセッサやキャッシュの一部を別の目的に使う機能を備えたプロセッサが存在し，これらの記憶領域のデータの利用方法に関して様々な手法が提案されている．

Panda ら[10] は，組み込みアプリケーションを対象として，キャッシュ上の競合ミスを低減するため，プログラムの静的な解析に基づきスカラおよび配列変数をスクラッチパッドメモリまたはオフチップメモリに割り当てる手法を提案している．配列の再利用性に注目し，頻繁にアクセスされる配列をスクラッチパッドメモリに割当てデータの再利用性の有効利用を図るという点で提案手法と類似しているが，提案手法ではそれに加えて再利用性のない配列データについても，SCM 上にバッファ領域を設け，拡張命令による大粒度データ転送を行うことでレイテンシ削減を図りさらなる高速化を目標とする．

Chiou ら[11] は，スクラッチパッドメモリのような固定メモリではなく，従来のセットアソシアティブキャッシュの一部のウェイをハードウェアによるデータ入替え対象外とし，特定のメモリ領域と 1 対 1 対応とすることでスクラッチパッドメモリの役割をエミュレートする手法を提案している．この手法では同時に必要となる複数データのそれぞれに対して，あらかじめ割り当てる領域を排他的に決定しておくことにより，データがキャッシュ上で競合することを防ぐことを目的としている．これに対し提案コンパイラでは，ソフトウェア制御メモリへデータ割当てを行い競合を防ぐだけではなく，データ転送スケジューリングを考慮に入れデータ転送命令の挿入を行う．

星ら[12] は，ソフトウェアにより外付けキャッシュ上の特定領域を保護し，完全にソフトウェア制御下に置いてベクトルレジスタ領域として使用することで高性能化を達成するアーキテクチャを提案している．コンパイラはこの領域と主記憶間のデータ転送を行うため既存のベクトル計算機と同様の命令列の書き換えを行う．この処理は本論文中の提案手法のうち，“再利用性のない配列の最適化” と類似していると考えられる．しかしベクトル計算機と同様の多バンク化，パイプライン化された主記憶システムを前提にスカラプロセッサのベクトル向き処理を効率化することを主眼としており，チップ上記憶を利用したデータの再利用に関しては検討が行われていない点が本研究とは異なる．

キャッシュ向けのレイテンシ削減手法としてソフトウェア・プリフェッチ技術は広く研究が行われている[13]．Mowry ら[14] は密行列の演算を対象として，コンパイラによるプリフェッチ命令の自動挿入手法を提案している．この手法ではあらかじめキャッシュミスを誘発する可能性があるメモリアクセスを予測し，それらのメモリアクセスに対しプリフェッチの発行を行う．プリフェッチは他の演算と重複できるようソフトウェアパイプライン的に行われる．ソフトウェア・プリフェッチはデータ転送先を明示的に指定できないためキャッシュ上で競合を生じる可能性があるが，SCM 上では明示的にデータ転送先を指定することで競合を抑えられる．したがって，提案手法に対し本手法のようなソフトウェア・プリフェッチ挿入技術を組み合わせることで，その有効性をさらに増大させることが可能と考えられる．

7. まとめと今後の課題

プロセッサと主記憶の性能差の問題に対処するため，ソフトウェア制御可能メモリを利用した高性能化を図るアーキテクチャが提案されている．しかし，ソフトウェア制御メモリを用いた高性能化を達成するためには，プログラマがソフトウェア制御可能メモリのデータ配置とデータ転送のスケジューリングを行う必要もあり，ユーザの負担が問題となる．

そこで本論文では，ソフトウェア制御可能なメモリのデータ配置や転送を自動的に最適化する自動最適化コンパイラについて検討を行い，自動最適化コンパイラへの 1 つのアプローチとして，ソフトウェア制御メモリの最適化に必要なアプリケーションに関する情報を “最適化ヒント情報” として記述するだけで最適化を行う “最適化ヒント情報に基づく自動最適化コンパイラ” を提案した．最適化ヒント情報を用いたプログラミングは従来手法である SCIMA ディレクティブを用いた最適化プログラミングと比較するとソースコードの記述が容易であるという利点を持つ．これはソフトウェア制御オンチップメモリ向けの最適化をより多くのアプリケーションに適用させるうえで重要であると考えられる．また，HPC 分野のアプリケーションではコードの一部分が実行時間の多くを占める場合が多く，大規模なプログラムにおいても実行プロファイルなどを用いることで最適化対象を容易に発見することができるため，提案手法はより大規模なプログラムに対しても有効であると期待できる．さらに，最適化ヒント情報はハードウェアパラメータに依存しないため，別のマシン上で再コンパイルする際にコードを書き換える作業は不要であり，この利点は対象コードが大規模化するに従いいっそう増大するものと考えられる．

シミュレーションによる評価を通じ，ヒント情報を本コンパイラによって解釈して得られたコードは従来

どおり手動でループ変形やディレクティブ挿入を行ったコードと同等の性能を達成できることが分かった．以上より，提案手法の有効性を確認することができた．

今後の課題として，今回用いた単純なカーネルループではなく，より実アプリケーションに近い評価プログラムを利用した評価を行うことがあげられる．今回の評価ではプログラム実行中にはSCM/キャッシュ容量比の変更は行っていないが，実アプリケーションではループの処理の重さが各コード部分に応じて変化する場合がある．したがって，ループの処理の重さに応じてSCM/キャッシュ容量比を動的に変更することについて検討することは有用であると考えられる．一方で，プログラム全体の解析に基づきデータの再利用性などを判断しプロセッサ近接のメモリに対しデータの配置を行う手法[15]も提案されており，今後このようなことについても検討を行いたい．

謝辞 本研究の一部は，文部科学省科学研究費補助金（基盤研究（B）No.14380136）によるものである．

参 考 文 献

1) Crisp, R.: Direct Rambus Technology: The mew main memory standard, *IEEE Micro*, Vol.17, No.6, pp.18–28 (1997).
2) Lam, M., Rothberg, E. and Wolf, M.: The cache performance and optimizations of Blocked Algorithms, *Proc. ASPLOS-IV*, pp.63–74 (1991).
3) Panda, P., Nakamura, H., Dutt, N. and Nicolau, A.: Augmenting Loop Tiling with Data Alignment for Improved Cache Performance, *IEEE Trans. Computers*, Vol.48, No.2, pp.142–149 (1999).
4) 日立製作所：*SuperH*^{TM} RISC Engine SH-4プログラミングマニュアル (1998).
5) intel Corporation: *XScale Core Developer's Manual* (2000).
6) 中村 宏，近藤正章，大河原英喜，朴 泰祐：ハイパフォーマンスコンピューティング向けアーキテクチャSCIMA，情報処理学会論文誌：ハイパフォーマンスコンピューティングシステム，Vol.41, No.SIG5(HPS1), pp.15–27 (2000).
7) 近藤正章，中村 宏，朴 泰祐：SCIMAにおける性能最適化手法の検討，情報処理学会論文誌：ハイパフォーマンスコンピューティングシステム，Vol.42, No.SIG12(HPS4), pp.37–48 (2001).
8) 藤田元信，近藤正章，中村 宏，千葉 滋，佐藤三久：ソフトウエア制御オンチップメモリのための最適化コンパイラの構想，情報処理研究報告，Vol.ARC-146, pp.31–36 (2002).
9) Sato, M., Satoh, S., Kusano, K. and Tanaka, Y.: Design of OpenMP Compiler for an SMP Cluster, *Proc. European Workshop on OpenMP EWOMP '99*, pp.32–39 (1999).
10) Panda, P., Dutt, N. and Nicolau, A.: Efficient Utilization of Scratch-Pad Memory in Embedded Processor Applications, *Proc. 1997 European Design and Test Conference* (*ED&TC'97*), pp.7–11 (1997).
11) Chiou, D., Jain, P., Rudolph, L. and Devadas, S.: Application-Specific Memory Management in Embedded Systems Using Software-Controlled Caches, *Proc. 37th Design Automation Conference* (2000).
12) 星 宗王，細見岳生：スカラプロセッサのベクトル向き処理を効率化するVector Register On-Cache機構，ハイパフォーマンスコンピューティングと計算科学シンポジウム（HPCS2003），pp.39–46 (2003).
13) Vanderwiel, S. and Lilja, D.: Data Prefetch Mechanisms, *ACM Computing Surveys*, Vol.32, No.2 (2000).
14) Mowry, T., Lam, M. and Gupta, A.: Design and Evaluation of a Compiler Algorithm for Prefetching, *Proc. 5th International Conference on Architectural Support for Programming Languages and Operating Systems*, pp.62–73 (1992).
15) 中野啓史，小高 剛，木村啓二，笠原博徳：チップマルチプロセッサ上での粗粒度タスク並列処理によるデータローカライゼーション，情報処理研究報告，Vol.ARC-151, pp.13–18 (2003).

(平成15年5月13日受付)
(平成15年8月27日採録)

藤田 元信（学生会員）

昭和54年生．平成13年東京大学工学部計数工学科卒業．平成15年同大学大学院情報理工学系研究科修士課程修了．現在，同博士課程に在籍．計算機アーキテクチャ，最適化コンパイラの研究に従事．

近藤 正章（正会員）

平成 10 年筑波大学第三学群情報学類卒業．平成 12 年同大学大学院工学研究科博士前期課程修了．平成 15 年東京大学大学院工学系研究科先端学際工学専攻修了．工学博士．現在，独立行政法人科学技術振興機構戦略的創造研究推進事業 CREST 研究員．計算機アーキテクチャ，ハイパフォーマンスコンピューティング，ディペンダブルコンピューティングの研究に従事．電子情報通信学会会員．

中村 宏（正会員）

昭和 60 年東京大学工学部電子工学科卒業．平成 2 年同大学大学院工学系研究科電気工学専攻博士課程修了．工学博士．同年筑波大学電子・情報工学系助手．同講師，同助教授を経て，平成 8 年より東京大学先端科学技術研究センター助教授．この間，平成 8 年〜9 年カリフォルニア大学アーバイン校客員助教授．高性能・低消費電力プロセッサのアーキテクチャ，ハイパフォーマンスコンピューティング，ディペンダブルコンピューティング，ディジタルシステムの設計支援の研究に従事．情報処理学会より論文賞（平成 5 年度），山下記念研究賞（平成 6 年度），坂井記念特別賞（平成 13 年度），各受賞．IEICE，IEEE，ACM 各会員．