計算機アーキテクチャ 151－8
（２００３．１．３１）

ソフトウエア制御オンチップメモリ向け自動最適化アルゴリズムとその初期評価

藤田元信[†] 近藤正章[†,††]
家田正孝[†] 中村宏[†]

近年深刻化しているプロセッサとメモリの性能格差の問題に対応するため，チップ上にソフトウエア制御可能なメモリを搭載することで性能向上を図るメモリアーキテクチャが提案されている．本稿では，そのようなメモリアーキテクチャの一つである SCIMA (Software Controlled Integrated Memory Architecture) において，コンパイラによってオンチップメモリを自動的に用いて最適化を行うアルゴリズムについて検討し，初期評価を行った．

An Adaptive Compilation Algorithm for Software-Controlled On-Chip Memory and its Preliminary Evaluation

Motonobu FUJITA,[†] Masaaki KONDO,[†,††] Masataka IEDA[†] and Hiroshi NAKAMURA[†]

Recently, performance disparity between processor and main memory is increasing. To remedy performance degradation caused by this problem, a new memory architecture called SCIMA which has software controllable integrated memory has been proposed. SCIMA improves performance using software-contorllable memory. In this paper, we propose an algorithm of automatic optimizing compiler for SCIMA, and present preliminary evaluation of the proposed algorithm.

1. はじめに

近年，半導体集積技術の進歩によるプロセスの微細化，命令レベル並列性の活用などによりプロセッサの性能は飛躍的な向上を見せた．その一方で，主記憶として多く用いられている DRAM の速度向上は緩やかで，年率数パーセント程度に留まっている．このため，計算システムの性能は主にメモリの性能によって制限されてしまっている．

この問題に対応するため，従来からキャッシュメモリが用いられてきた．しかし，科学技術計算をはじめとするハイパフォーマンスコンピューティング分野の多くのアプリケーションでは，キャッシュメモリが有効に機能しない場合がある．多くの配列アクセスを伴うアプリケーションでは特に，データセットサイズがキャッシュメモリのサイズに比べて大きく，データが本来持っている局所性がうまく活用されない，配列のサイズにより配列の要素自身がキャッシュ上の同一のブロックを奪い合う，また複数の配列がキャッシュ上の同一の領域にマッピングされ，互いをキャッシュ上から追い出し合う等の現象が発生し，性能が大きく低下する場合がある．

この問題を克服するために，ブロッキング[1] 等アクセスパターンを変更し局所性を改善する手法，配列内の干渉を抑えるため配列サイズを変更する手法[2] 等ソフトウエアによる最適化手法が提案されてきた．しかしながら複数の配列が，キャッシュ上で同一ブロックにマッピングされ発生する競合をはじめ，ソフトウエア的な最適化手法だけでは対処が困難な場合がある．

また，キャッシュメモリでは，オフチップメモリとのデータ転送は常にラインと呼ばれる固定された単位で行われる．主記憶上で連続しているデータを扱う場合はできるだけ大きな粒度で転送した方が主記憶のレイテンシによる影響を減らすことができる．しかしラインサイズを大きくすると，非連続なデータを扱った場合に無駄な転送が増加する，あるいはキャッシュ内のエントリ数減少に伴う競合が多発してしまい，必ずしも性能向上に貢献しない．

そこで従来のキャッシュメモリに加え，ソフトウエアによりアドレス指定可能なメモリをチップ上に追加して搭載し，これを用いて主記憶に対する柔軟なデー

† 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
†† 科学技術振興事業団 JST

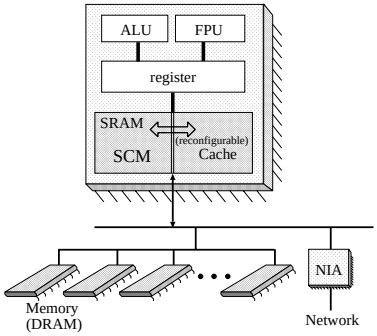

図 1 SCIMA の構成

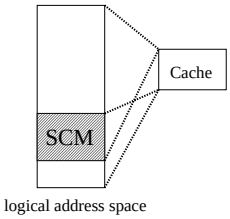

図 2 SCIMA におけるアドレス空間

タ転送を可能にすることを可能にするメモリアーキテクチャ SCIMA (Software Controlled Integrated Memory Architectute) が提案されている[3]．これまでの SCIMA の性能評価の過程において，メモリアクセスパターンや再利用性によって SCIMA のソフトウエア制御メモリの利用法が整理されている．そこで，SCIMA 向けの最適化をアルゴリズムとして体系化し，コンパイラとして実現することを計画している．

本稿では，そのアルゴリズムについて検討し， 評価を行った．

2. SCIMA

2.1 SCIMA のアーキテクチャ

SCIMA のアーキテクチャを**図 1** に示す．SCIMA では，チップ上に従来のキャッシュに加えソフトウエア制御可能なメモリ (Software Controlled on-chip Memory, SCM) を搭載する．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，同じハードウエア機構を共有しているが，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エア制御により自動的にデータ配置，置き換えが行われる．一方 SCM では，ソフトウエアから明示的にデータ配置，置き換えを行うことが可能である．

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，ソフトウエアによる SCM のデータ配置，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 *page* と呼ばれる単位に分割され，この *page* を単位としてメモリアクセス順序保証などを行う．*page-load/page-store* 命令により一度に転送できる最大サイズはこの *page* であり，*page* のサイズは数 KB 程度を想定している．

表 1 配列のアクセスの特徴の分類

アクセスの特徴	分類
再利用性	あり / なし
連続性	連続 / ストライド / 不規則

2.3 SCIMA ディレクティブベースコンパイラ

これまでに行われた SCIMA に関する性能評価の多くでは，SCM と主記憶間のデータ転送は *page-load/page-store* に直接対応する関数をソースプログラムに直接挿入することで表現していた．この関数による表現では，データ転送先アドレスの指定を直接指定できるなど，高い自由度を持つ．しかし一方でユーザ自身が SCM 領域の管理を行わなければならないので，この関数を用いることはユーザに大きな負担となっていた．

そこで，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 SCIMA ディレクティブベースコンパイラを提案している[7]．SCIMA ディレクティブベースコンパイラは，*Omni OpemMP Compiler*[5] 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．代表的な SCIMA ディレクティブの仕様とその意味を**図 3** に示す．

SCIMA ディレクティブは，SCM 上の領域確保，SCM と主記憶のデータ転送を少ない引数で簡単に指定できる．さらにオフチップメモリへの参照から SCM 領域の参照への変換は自動的に行われるため，SCM の管理をユーザ自身が行わなくてもよい，という利点を持っている．

3. SCIMA 向け自動最適化アルゴリズム

3.1 SCIMA 向け最適化

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わかった．そこで我々はアクセスの連続性とデータの再利用性の観点から配列の特徴を整理し (**表 1**)，その分類に基づいた SCM の利用法を提案している[6]．具体的には**図 4** のように SCM を用いることで性能向上を図る．

まず，(1)(2) に分類される再利用性のない配列に対

!$scm begin (<配列名>,<size_1>,<size_2>,...,<size_n>,<ストライド幅_1>,<ストライド幅_2>,...,<ストライド幅_n>)

意味：指定された配列用のオンチップメモリ領域を確保する。対応する**!$scm end**との間にある指定された配列への参照はオフチップメモリ 参照からオンチップメモリ参照に自動的に置き換えられる。<size_i> は各次元ごとの大きさを表し、<ストライド幅_i>は各次元のストライド幅を表す。ストライド幅が0のときはストライド機能を使わない。

!$scm end (<配列名>)

意味：対応する**!$scm begin**との間で指定された配列のオンチップ領域を解放する。

!$scm load (<配列名>,<index_1>,<index_2>,...,<index_n>,<size_1>,<size_2>,...,<size_n>)

意味：指定された配列の<index_1>,<index_2>,...,<index_n>を基点とし、<size_1>,<size_2>,...,<size_n>の大きさの領域をオンチップメモリに転送する。このとき転送サイズはbeginで確保した大きさを越えてはならない。

!$scm store (<配列名>)

意味： 指定された配列のオンチップメモリの領域をオフチップメモリ領域へ転送する。基点や大きさは直前のloadで指定された値が用いられる。

SCIMA ディレクティブの仕様 (一部抜粋)

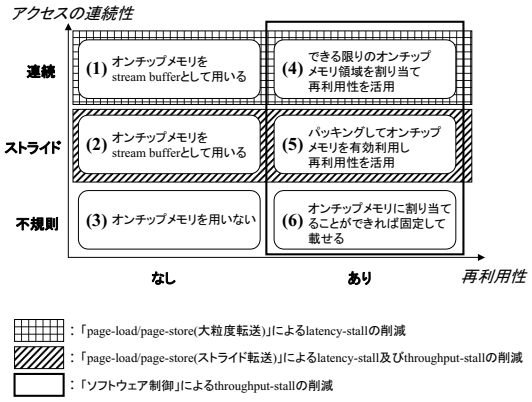

図 4 配列の特徴に対するオンチップメモリの利用法

しては，SCM 上に *page* サイズのバッファ☆を確保し，その領域をストリームバッファとして用いながらアクセスする．(1) に分類される配列では *page-load/page-store* 命令による大粒度転送よって転送回数が減り，オフチップメモリレイテンシの影響が軽減されることが期待できる．(2) に分類される配列では，*page-load/page-store* 命令のストライド転送機能により無駄な転送が減ることが期待できる．

次に，(4)(5) に分類される再利用性がある配列に対しては，ブロッキング手法を用いるなど SCM 上に搭載可能なサイズまでワーキングセットを縮小し，ブロック単位で SCM 上に転送し，他のデータとの干渉を防ぎつつ再利用性を最大限活用する．

(6) に分類されるアクセスに規則性がない配列については，アクセス範囲があらかじめ分かり，かつ SCM に載る程度の大きさであれば SCM 領域に固定して載せ再利用性を活用する．SCM に載らない場合はキャッシュ経由でアクセスする．また，(3) のようにアクセスの規則性がなく，再利用性もない配列については SCM を用いたとしても性能向上が期待できないので，SCM を用いない．

☆ 仮にこれ以上のサイズのバッファ領域を確保しても，オフチップメモリ一度に転送できる最大サイズが *page* であるため性能向上は期待できない

3.2 SCIMA 向け自動最適化

SCIMA ディレクティブベースコンパイラにより，SCM を用いた最適化プログラミングは以前と比較すると比較的容易になった．しかしながら，SCIMA ディレクティブは関数を別の形で表現したに過ぎず，SCIMA ディレクティブを用いて性能最適化を達成するためには，ユーザは最適化対象の選択，最適化対象となる配列のアクセスパターンやストライド幅といったアプリケーション，プログラムに関する情報に加え，必要となる SCM の容量，一回あたりの転送量など，キャッシュ向けの性能最適化の際には必要のなかった SCIMA に特有の多くの知識を要する．また，SCM とのデータ転送命令をユーザが記述するため，SCM のアクセススケジューリングおよび SCM 領域の管理もユーザ自身が行わなければならない．

多くのユーザの立場を考慮すると，これら SCIMA に特有の情報の入力から，SCM のアクセススケジューリングなどをすべてユーザに負担させることは望ましくない．そこで現在，ユーザにこれらを負担させることなく SCM を用いた性能最適化を行うことのできるアルゴリズムについて検討を行っている．

最終的な段階では，ユーザが SCM の存在すら意識することなく，

- 最適化対象の選択
- 局所性改善のためのコード変形
- SCM と主記憶間のデータ転送命令の挿入

が自動的に行われ SCM を用いた性能向上が達成されるコンパイラを目標としている．

それぞれの段階のうち，最適化対象の選択や局所性改善のためのコード変形については，既に多くの研究がなされており，既存技術の延長として SCIMA 向けの最適化にも応用が可能であると考えられる．そこで本稿では最適化対象が選択され，ブロッキング等局所性改善のためのコード変形が行われた後の段階に着目し，入力コードに対して施される SCIMA 特有の処理を対象として最適化アルゴリズムについて検討を行う．

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

      sum = 0.0d0

c    ***** x, yを再利用性のない配列と見なし最適化 *****
!$scm opt_notreuseable(x)
!$scm opt_notreuseable(y)

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

c    ***** 最適化範囲ここまで *****
!$scm optend (y)
!$scm optend (x)
```

図 6 ベクトル内積計算 (ヒント情報による最適化)

なお，*page* サイズ，SCM サイズといった SCIMA 固有の値については，ユーザが与えるのではなく，システムから取得するものとする．

3.3 ヒント情報

最適化対象の選択，局所性改善のためのコード変形は本稿では扱わなかったため，これらはあらかじめ別の手段によって完了していることを前提とした．そこで，ユーザがあらかじめ入力コードのブロッキングを行い，最適化対象となる配列の名前と，その配列の再利用性に関する情報をヒント情報として与える．具体的な最適化ヒント情報の仕様を**図 5** に示す．

ヒント情報は，ユーザの手によってソースコード中にディレクティブとして与えられる．これらはアプリケーション，プログラムの性質に由来するものであり，SCM の存在を意識することなく記述できる．ユーザ自身がソースコードにブロッキングを施すことから，再利用性や，ブロッキングループとエレメントループの境界をユーザが判断してヒント情報を与えることは困難ではないと考えられる．

このヒント情報を用いてベクトル内積計算を SCIMA 向けに自動的に最適化する例を示す．ユーザは**図 6** のように，最適化対象配列となる配列を引数として再利用性に関する指示を伴うヒント情報ディレクティブにより，最適化対象範囲を指定する．

3.4 最適化処理の流れ

このヒント情報による最適化処理の流れを**図 7** に示す．

あらかじめユーザの手でブロッキングされた Fortran77 のソースコードと，最適化対象とする配列と再利用性に関するヒント情報ディレクティブを入力として受け取る．これはフロントエンドにより Fortran77 から Xobject Code と呼ばれる中間言語表現に変換される．次に，本稿で紹介するアルゴリズムによりヒント情報を解釈し，Xobject Code 上においてループの変形，SCIMA ディレクティブ☆の挿入を行う．

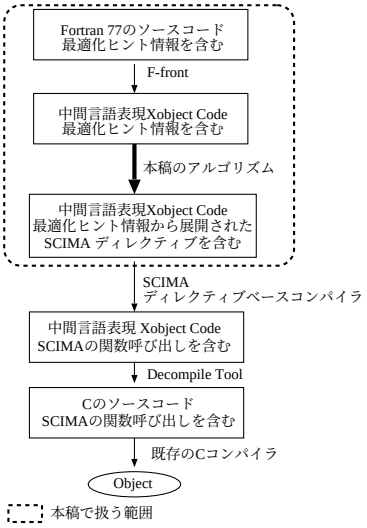

図 7 SCIMA 最適化処理の流れ

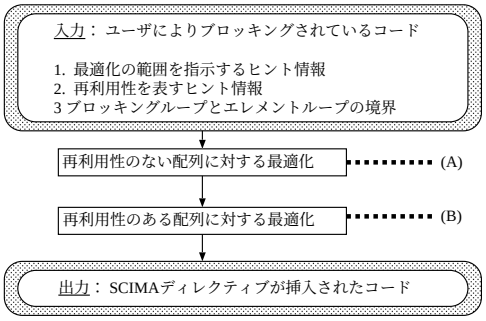

図 8 自動最適化アルゴリズムの全容

以降は，既に存在する SCIMA ディレクティブベースコンパイラの機能を利用することができる．SCIMA ディレクティブベースコンパイラの機能により，Xobject Code 中に挿入した SCIMA ディレクティブを関数呼び出しの形に変換する．さらに，Decompile Tool と呼ばれる変換ツールにより，関数呼び出しを含む Xobject Code から C のソースコードに変換する．この C のソースコードには，SCM と主記憶間のデータ転送を行う関数呼び出しが含まれている．最後に，既存の C コンパイラによりオブジェクトを生成することができる．

3.5 自動最適化アルゴリズム

今回作成した図 7 の破線で囲まれた部分について説明する．ここではユーザの与えた最適化ヒント情報のうち再利用性に関する情報を元に，入力コードに対し再利用性のない配列への最適化 (**図 8**-(A))，再利用性のある配列への最適化 (図 8-(B)) の順で SCM を利用するためのコード変形および SCIMA ディレクティブ

☆ 正確には SCIMA ディレクティブに相当する Xobject Code である

!$scm opt_notreuseable(<配列名>)
意味：このディレクティブと、対応する!$scm opt_end()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。

!$scm opt_reuseable (<配列名>)
意味：このディレクティブと、対応する!$scm opt_end()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。

!$scm element
意味：!$scm opt_reuseable()と!$scm opt_end()の間で用いられ、ブロッキングが施された入力コードにおける、エレメントループとブロッキングループの境界となるループレベルを示す。

!$scm opt_end(<配列名>)
意味：!$scm opt_notreuseable()または !$scm opt_reuseable()に対応し、最適化の対象となる範囲の終点を示す。

図 5 SCIMA 自動最適化ヒント情報

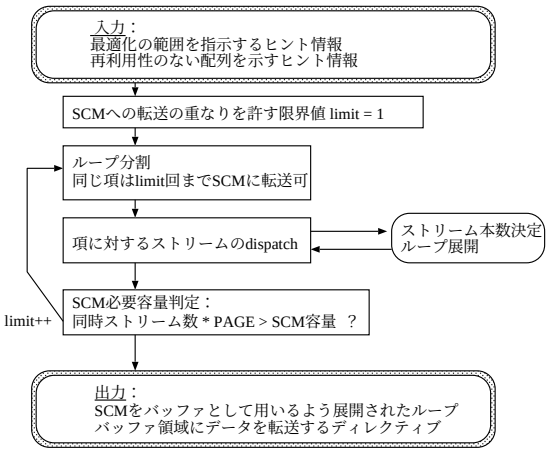

図 9 再利用性のない配列の最適化

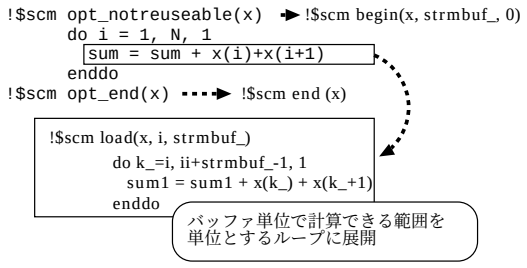

図 10 再利用性のない配列の最適化の例

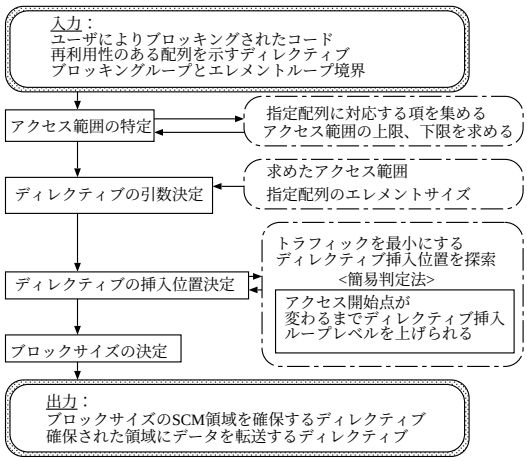

図 11 再利用性のある配列の最適化

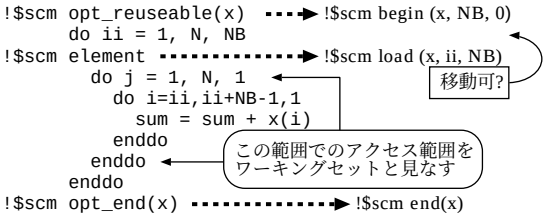

図 12 再利用性のある配列の最適化の例

の挿入を行う．

ここで，図 8-(A) の再利用性のない配列への最適化は図 4 の (1)(2) に対応し，図 8-(B) の再利用性のある配列への最適化は図 4 の (4)(5) に対応していると考えることができる．これらはそれぞれ独立な最適化として実施可能であるが，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．それぞれの内容については以下に示す．

(A) 再利用性のない配列に対する最適化 再利用性のない配列への最適化では，同時に必要になるストリームバッファ領域の総量が SCM サイズを越えないようにするため，まずループ分割を行う．次に分割されたループに含まれる各項に対して SCM 領域を割り当てる．このとき，同じ配列にアクセスする複数の項のアクセス開始点がメモリ上で *page* サイズ以上に離れている場合は別々のバッファに割り当て，それ以下の場合は同じバッファ領域に割り当て，そのバッファサイズを単位としてメモリアクセスを行うようにループを展開する (**図 10**)．最後に，算出された SCM の必要容量が利用可能な SCM 容量を超過していないかどうかを確認し，超過していなければこのまま次の最適化に進み，もし超過しているようならば同じ項を複数回 SCM に転送することを許しループをさらに分割し同様の処理を行う．ただし，この場合ループ中のステートメント間のデータ再利用が阻害される可能性がある．

(B) 再利用性のある配列に対する最適化 再利用性のある配列への最適化ではヒント情報として与えられるブロッキングループとエレメントループ

表 2　評価環境

	キャッシュサイズ	SCM サイズ
Original	64KB(4-way)	0
directive_byalgorithm	16KB(1-way)	48KB

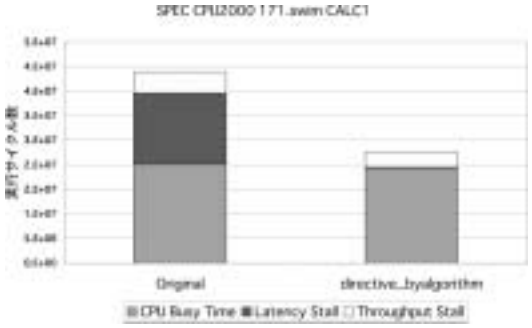

図 13　実行サイクル数

の境界を基準として，それより内側のループにおいてアクセスされる範囲をワーキングセットと見なす．各ループにおけるアクセス範囲の下限，上限が判ればディレクティブを挿入することができる．ループ開始点に領域確保のためのディレクティブを，ブロッキングループとエレメントループの境界に SCM と主記憶間のデータ転送ディレクティブを挿入する．さらに，同一の要素が何度も SCM に転送されることを防ぐため，SCM と主記憶間のデータ転送ディレクティブの挿入レベルを上げていく．**図 12** にこの最適化の例を示す．

4. アルゴリズムの評価

本稿で紹介したアルゴリズム初期評価として，SPEC CPU2000 171.swim に含まれるサブルーチン CALC1 について，クロックレベルシミュレータを用いて以下の実行サイクル数を比較した．

- オリジナルのコード (Original)
- 今回のアルゴリズムによって出力されたコード (directive_byalgorithm)

SCM とキャッシュメモリの容量については**表 2** とした．評価結果を**図 13** に示す．対象アプリケーションは，再利用性のない配列のみを含むコードであるが，ヒント情報をもとに本稿のアルゴリズムにより最適化したものは，オリジナルのコードに比べてスループット不足に起因するストール，オフチップメモリレイテンシに起因するストールともに減少し，性能が向上することがわかった．これは，再利用性のない配列に対し，無駄のない転送が行われていることに加え，*page-load/page-store* 命令を用いた大粒度転送により転送回数が減少したことに起因すると考えられる．

5. まとめと今後の課題

SCIMA 向けの自動最適化アルゴリズムについて検討し，その初期評価を行った．本稿のアルゴリズムでは，ユーザがブロッキングしたコードとプログラムのデータ再利用性等に関する最適化ヒント情報をもとに，自動的に SCIMA のソフトウエア制御メモリを用いた最適化が行われる．なお，この過程においてユーザは SCIMA のソフトウエア制御メモリの制御の仕方や，ハードウエア構成についての知識を必要としない．

ただし，今回は SCIMA 特有の，ソフトウエア制御メモリとオフチップメモリ間のデータ転送命令を挿入する処理のアルゴリズム化を優先したため，入力コードのブロッキング，最適化対象の選択，ヒント情報の入力についてはユーザが行うことを前提とした．今後はユーザの手を介することなくそれらの処理を行う方法についても検討する予定である．

謝辞　本研究を行うにあたり，御助言，御討論頂いた筑波大学 佐藤 三久教授， 朴 泰祐教授，東京工業大学 千葉 滋講師，並びに南谷・中村研究室の皆様に感謝致します．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) M. Lam, E. Rothberg and M. Wolf, “ The cache performance and optimizations of Blocked Algorithms ”, Proc. ASPLOS-IV, pp.63–74, 1991
2) P.R.Panda, H.Nakamura, N.D.Dutt, and A.Nicolau, “ Augmenting Loop Tiling with Data Alignment for Improved Cache Performance ”,IEEE Transactions on Computers, Vol.48, No.2, pp.142-149, Febrary, 1999.
3) 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, Vol.41, No.SIG5(HPS1), pp.15-27,2000 .
4) 大根田拓, 近藤正章, 中村宏.“ SCIMA におけるメモリアクセス機構の検討 ”. 情処研報,ARC-144, Vol.99, No.76, pp.165-170, 2001
5) M.Sato, S.Satoh, K.Kusano, and Y.Tanaka. “ Design of OpenMP Compiler for an SMP Cluster ”, Proc. European Workshop on OpenMP EWOMP ’99, pp.32-39, 1999.
6) 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,Vol.42, No.SIG12(HPS4), pp.37-48, 2001.
7) 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 ”, 情処研報,ARC-146, pp.31-36, 2002