SR8000におけるデータプリロード処理

西山博泰† 久島伊知郎† 菊池純男†

大規模なデータを扱う科学技術計算などでは，データ再利用を前提とした通常のキャッシュ機構が有効に機能しない場合が多い．このような，大規模データを扱うアプリケーションの性能を向上を目的として，SR8000ではデータのプリロード機構とプリフェッチ機構を実現している．SR8000では，これらの機構を利用して，データ転送と演算実行を並行して行うことにより，メモリ参照レイテンシを隠蔽する．本論文では，SR8000の特徴である，プリロード機構を対象としたコンパイラ最適化について述べる．SR8000コンパイラでは，依存の除去によりプリロードの適用効果を高めるプログラム変換，多重ループ向けのピーリングによるループスタートオーバヘッドの削減，条件分岐を含むループを適用対象とするためのpartial maskingなどのコンパイラ最適化を実現している．ベンチマークによる性能評価の結果，データプリロード機構とプリフェッチ機構の組合せにより，高いレイテンシ隠蔽効果が得られることを確認した．

Data Preloading on the SR8000

Hiroyasu Nishiyama,† Ichiro Kyushima† and Sumio Kikuchi†

Cache memory may not work effectively for applications with large data sets such as those for scientific computations. To improve performance of these applications, the SR8000 incorporates data preloading and data prefetching. The SR8000 hides memory latency using these mechanisms by executing instructions in parallel with data transfer from memory. In this paper, we describe compiler algolithms for data preloading for the SR8000. These include program transformations that enhances applicability of preloading by eliminating dependence, peeling for nested loops that reduces loop startup overhead, and partial masking for loops with conditional constructs. Our experimental results using benchmark programs show that data preloading combined with prefetching effectively hides memory latency.

1. はじめに

近年，消費電力や価格といった点から，マイクロプロセッサを用いた並列システムが，科学技術計算などに広範に利用されるようになっている．特に，動作周波数の向上や，スーパスカラ，VLIWといった方式による命令レベル並列性の向上などにより，マイクロプロセッサの性能は年々向上を続けており，マイクロプロセッサを用いた並列システムのピーク性能は，ベクトルプロセッサシステムの性能を凌駕するようになっている．

このような，マイクロプロセッサの著しい速度向上と比較すると，メモリシステムの速度向上は低く抑えられており，メモリとプロセッサの動作速度の乖離が進んでいる．通常，マイクロプロセッサのメモリシステムは，キャッシュヒット率の高いアプリケーションを前提として設計されており，non-blockingキャッシュやout-of-order実行による動的なレイテンシ隠蔽方式のみでは，隠蔽可能なメモリレイテンシに限界がある．

一方で，ワーキングセットの大きな科学技術計算では，データの参照局所性が期待できない．このため，従来のマイクロプロセッサをベースとしたシステムでは，キャッシュミスが頻発し，性能が著しく低下してしまうという問題をかかえている．ブロッキングなどのプログラム変換によってキャッシュ向けに局所性を向上する最適化[16]は研究されているが，こういったプログラム変換が適用可能な場合は限られている．

このような問題に対処するために，SR8000では，(1) 大容量浮動小数点レジスタ，および，キャッシュメモリ，(2) 高スループットのメモリシステムによる，主記憶から浮動小数点レジスタ，および，キャッシュメモリへのデータ供給（データプリロード，および，データプリフェッチ），(3) スカラ演算のパイプライン実行，および，out-of-order実行による複数の演算の並列実行，を実現することによって，ベクトルプロ

† 株式会社日立製作所システム開発研究所
Systems Development Laboratory, HITACHI, Ltd.

セッサと同様な高速計算を実現している．

データプリフェッチ[2]は，将来の処理で必要となるデータの属するキャッシュブロックを，演算などの命令実行と並行して，主記憶からキャッシュへ転送する機構である．SR8000では，ソフトウェア制御によるデータプリフェッチにより，最大16個のプリフェッチリクエストを，同時に処理することが可能である．ただし，プリフェッチではキャッシュブロック単位でデータの転送を行うので，離散的なデータを参照する場合には，キャッシュへの無駄なデータ転送のために，バスの使用効率が低下してしまう．また，フルアソシアティブ方式でない場合には，キャッシュ競合などによっても，性能が低下する可能性を持つ．

一方，データプリロードでは，他の命令の実行と並行して，主記憶からレジスタへ直接データ転送を行うプリロード命令を使用して，メモリレイテンシを隠蔽する．一般に，メモリレイテンシは通常の演算レイテンシと比較して非常に大きな（>100 cycle）ため，レジスタへデータを直接ロードすることによってメモリレイテンシを隠蔽するためには，多数のレジスタを必要とし，必要なコード量も増大する．SR8000のプリロード機構では，多数のレジスタを用意し，これを論理レジスタ番号で管理して，物理レジスタと論理レジスタ番号のマッピングをスライドさせながら使用することで，コード量の増大を抑えながらメモリレイテンシを隠蔽する[11]．

プリロード命令では，主記憶とプロセッサ間で必要なデータのみがやりとりされるので，離散的な参照を行った場合に，キャッシュを介した場合と比較して，性能を向上することができる．ただし，キャッシュ容量に比較すると，レジスタの容量は小さなため，プリロードで隠蔽可能なレイテンシには限界がある．

SR8000では，連続参照に対してはデータプリフェッチを，離散データ参照に対してはデータプリロードを使用することにより，メモリレイテンシを隠蔽する方式をとっている．特に，データプリロードは通常のマイクロプロセッサの苦手とする離散データ参照性能を向上するのに有効である．

プリロード命令を対象としたコンパイル処理では，命令スケジューリング，レジスタ割当てに加えて，

- 効果的にプリロード命令を発行するための依存除去処理，
- ループ起動オーバヘッドの削減，
- 条件分岐を含むループに対する適用，

などの最適化が重要となる．本論文では，SR8000におけるプリロード向け最適化について述べる．

以下，本論文では，2章においてSR8000のアーキテクチャおよびプリロード処理について説明し，3章ではプリロード命令を利用するためのソフトウェアパイプライニング方式，4章では条件分岐を含むループに対する最適化処理について述べる．また，5章では性能評価の結果を示す．

2. SR8000アーキテクチャ

本章では，SR8000アーキテクチャの概要について説明する．

SR8000は，複数のRISCマイクロプロセッサから構成されるSMP（Symmetric Multi Processing）ノードを，高速ノード間ネットワークで結合した並列計算機システム[14]である．各ノードは，IPと呼ばれる8台の演算用RISCマイクロプロセッサ，SPと呼ばれる1台のシステム制御用RISCマイクロプロセッサ，および，これらのマイクロプロセッサ間で参照される共有メモリから構成される．

IPおよびSPを構成するRISCマイクロプロセッサは，PowerPC☆をベースとして，大規模計算を高速実行するための独自拡張を付与したアーキテクチャであり，スーパスカラ実行により1サイクルあたり最大4つの浮動小数点演算を実行可能である[7]．SR8000は，高スループットの主記憶・プロセッサ間結合により，演算器に連続的にデータを供給することができる．

2.1 PowerPCに対するアーキテクチャ拡張

SR8000におけるアーキテクチャの拡張点のうち，本論文に関連するものを以下に示す．

(1) 大容量浮動小数点スライドレジスタ

PowerPCで定義されている32本の浮動小数点レジスタ（FR0〜FR31）に加えて，128本の拡張レジスタ（FR32〜FR159）を定義している．FR0からFR31は固定部，FR32からFR159はスライド部と呼ばれる．

スライド部のレジスタ番号は，スライド命令によるソフトウェア制御により，リネームすることができる[11]．スライド命令を実行することにより，レジスタ番号を一定値だけずらしたレジスタが同じ物理レジスタを指すようになる．これを，レジスタのスライドと呼び，スライド命令によるレジスタ番号の変移の数をスライドピッチと呼ぶ．スライドピッチを P とすると，スライド命令の実行後のレジスタ番号 R_a と実行前のレジスタ番号 R_b には，次式の関係が成り立つ．

☆ PowerPC is a trademark of IBM Corp.

$$R_b = ((R_a - 32 + P)\ mod\ 128) + 32 \quad (1)$$

論理的には，スライド部は環状の構成を持ち，同一レジスタ番号の指示するレジスタの内容が，スライド命令の実行によって一定数分移動すると考えることができる．

なお，汎用レジスタに関しては，ベースとなる PowerPC と同じく 32 本である．

(2) データプリロード/ポストストア命令

スライド部を対象として，データプリロード命令が定義されている．データプリロード命令は，他の命令実行と並行して，主記憶からスライド部に浮動小数点データを転送する命令である．

データプリロード命令では，データのキャッシングを行わず，主記憶からレジスタへ直接データを転送する．このため，離散参照の場合においても，不要なデータの転送は生じない．

プリロード命令をメモリレイテンシを隠蔽できるよう，十分前に発行するためには，ソフトウェアパイプライニングの一手法である，モジュロスケジューリングを使用する．これに関しては 3 章で説明する．プリロード命令と同様に，スライド部からメモリへのストアを行うポストストア命令が定義されている．

(3) 拡張浮動小数点演算命令

浮動小数点レジスタの固定部とスライド部を対象として，浮動小数点演算を行う命令が定義されている．なお，命令空間の制約により，浮動小数点レジスタのスライド部については，最初の 96 本のみ拡張浮動小数点演算命令のオペランドとして指定可能である．すなわち，拡張浮動小数点演算命令で指定可能なレジスタは FR0〜FR127 であり，FR128〜FR159 は演算で参照することはできない．

(4) 条件付き命令実行命令

SR8000 では，分岐ペナルティの削減と，分岐を含むループに対してソフトウェアパイプライニングを適用することを目的として，限定的な条件付き命令実行機構を導入している．これに関しては，4 章で説明する．

2.2 プリロード処理

図 1 (a) のプログラムに対するプリロード化コードの例を，図 1 (b) に示す．図 1 (b) のコードでは，最初に，配列要素 `A[0]` から `A[4*M]` の値を，浮動小数点レジスタのスライド部にプリロードしている．ループ内の処理では，まず，(7) において最初の配列要素 `A[0]` を変数 `S` に加算する．次に，(8) で配列要素 `A[5*M]` をレジスタ `FR42` にプリロードする．(9) の `slide 2` はスライド部を 2 だけスライドさせる命令である．この命令を実行した結果，レジスタのリネームが行われ，リネーム前のレジスタ `FR34`〜`FR42` が，それぞれ，`FR32`〜`FR40` で参照できるようになる．したがって，次のループ繰返しでは，`FR32` が (2) で読み込んだ配列要素 `A[M]` を保持していることとなる．(6)〜(10) のループの処理が終了した時点で，すでに読み込み済みの配列要素が `FR32`〜`FR40` に保存されているので，(11)〜(15) で加算を行う．ここで，(1)〜(5) の部分はプロローグ部，(6)〜(10) はカーネル部，(10)〜(15) はエピローグ部と呼ばれる．

```
for(i=0; i < N; i+=M) {
   s += A[i];
}
```

```
 1: FR32 = PLD A[0]
 2: FR34 = PLD A[M]
 3: FR36 = PLD A[2*M]
 4: FR38 = PLD A[3*M]
 5: FR40 = PLD A[4*M]
 6: for(i=0; i < N-5; i+=M) {
 7:     s += FR32;
 8:     FR42 = PLD A[(i+5)*M];
 9:     slide 2;
10: }
11: s += FR32;
12: s += FR34;
13: s += FR36;
14: s += FR38;
15: s += FR40;
```

(a) **ソースプログラム** (b) **プリロード化コード**

図 1 プリロード処理の例
Fig. 1 Example of preloading.

このように，レジスタのスライド機構とプリロード命令を組み合わせることにより，ループ中の命令数を増加することなく，効果的にメモリレイテンシを隠蔽することが可能となる．

3. プリロード向け最適化

SR8000 向けソフトウェアパイプライニングは，プリロード向けに拡張したモジュロスケジューリングを，スケジューリングアルゴリズムとして使用している．**図 2** にその概要を示し，以下，各々の処理に関して説明する．

3.1 依存の除去

プリロード命令のレイテンシは通常の演算レイテンシと比較すると数十倍以上となるため，アドレス計算などに使用する汎用レジスタのライブレンジを派生的に増大させる可能性を持つ．また，プリロード命令がリカレンスを構成している場合には，プリロード命令を先行的に発行することが困難となる．そこで，プログラム変換により依存を除去する．

3.1.1 依存分割

モジュロスケジューリングでは，繰返し間にフロー依存を持つループに対して，フロー依存を解消する

```
依存除去();
L := 想定メモリレイテンシ;
II := 最小II;
repeat
  repeat
    Success := カーネル生成(II, L);
    II := II+n                    // 0 < n
  until II ≧ MaxII or Success
  if Success then
    Success := スライドレジスタ割り当て();
    if !Success then
      L := L*α;                   // 0 < α < 1
      if L < MinLatency then
        // プリロード対象のメモリ参照を、
        // プリフェッチ適用対象に変更
until II ≧ MaxII or Success;
コード生成();
```

図 2 プリロード化アルゴリズムの概要
Fig. 2 Overview of preloading algorithm.

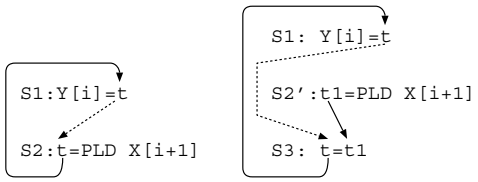


図 3 依存分割
Fig. 3 Dependency splitting.

ようなスケジュールを得ることができない．これは，ループ内の逆依存とループ間のフロー依存によって，循環した依存が構成されてしまうことによる．

プリロード命令のように，長いレイテンシを持つ命令が繰返し間のフロー依存を持つ場合，プリロード命令の実行を，他の命令実行と並行して行うことができないため，性能が低下してしまう．そこで，プリロード命令による繰返し間フロー依存については，テンポラリ変数を導入して，レイテンシの長い繰返し内フロー依存と，レイテンシの短い繰返し間フロー依存に分割する．このように変換することにより，レイテンシの長い命令が循環した依存サイクルから分離されるため，ループの開始間隔（Initiation Interval: II）を小さくすることができる．導入したコピー文については，レジスタ割当て処理により両辺に同じレジスタが割り当てられれば削除される．

たとえば，命令 S のレイテンシを $L(S)$ とすると，**図 3** (a) の依存グラフでは II の最小値が $L(S1)+L(S2)$ となる．この場合，命令 $S1$ と $S2$ で循環した依存サイクルが構成されるため，プリロード命令 $S2$ を先行的に発行することができず，プリロード命令 $S2$ のレイテンシにより，II も大きな値となる．これに対して，図 3 (b) のように繰返し間の依存を分割すること

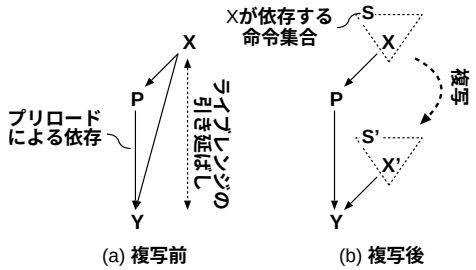


図 4 命令複写
Fig. 4 Instruction duplication.

で，II の最小値はより小さな値（$L(S1)+L(S3)$）となる．この場合には，プリロード命令 $S2'$ が依存サイクルから分離されるため，$S2'$ を先行的に発行することが可能となる．

3.1.2 命令複写

プリロード命令 P と，P にフロー依存する命令 Y を考える．ここで，命令 P および Y が，命令 X に共通にフロー依存する場合，命令 X に定義を持ち，命令 P と命令 Y に使用を持つような，変数 V のライブレンジは，命令 P と Y の間の依存により引き延ばされることとなる（**図 4** (a)）．

SR8000 では汎用レジスタの拡張は行われていないため，変数 V が整数型変数である場合，モジュロ変数展開[8] によって，ループのコード量増加や，必要となる汎用レジスタ数の増加によるレジスタスピルを生じるおそれがある．

そこで，ループ中の命令を複写することにより，ライブレンジの増大を抑止することを試みる（図 4 (b)）．まず，プリロード命令 P に対して，直接あるいは間接的にフロー依存する命令 Y がある場合，命令 P および Y が，整数型変数 V を介して，共通にフロー依存する命令 X を求める．次に，変数 V の値の計算に必要となる命令の集合 S を求める．集合 S は，$S=\{X\}\cup\{p|s\in S$，かつ，s が p に整数型変数を介してフロー依存$\}$，により定義される．続いて，集合 S に含まれる命令を複写し，命令 Y の前に挿入する．ここで，複写した命令と複写対象の命令間での変数の競合を避けるため，複写した命令群および命令 Y に対して，変数リネームを適用する．これにより，集合 S に含まれる命令が，命令 P と命令 Y の間で再実行されるようにコードが変換される．この変換の結果，必要命令数は増加するものの，変数 V のライブレンジは分割され縮小される．

リストベクトルを利用したループでは，整数型変数のライブレンジの増大が生じることが多い．例として，

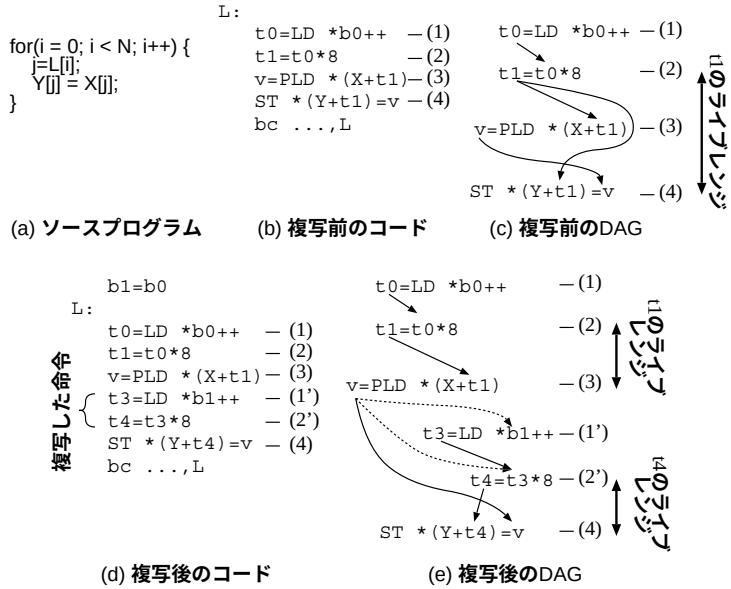


図 5 命令複写の例
Fig. 5 Example of instruction duplication.

図 5 (a) に示すプログラムを考える．このプログラムに対する疑似コード（図 5 (b)）において，プリロード命令（3）は乗算命令（2）にフロー依存し，同様にストア命令（4）も乗算命令（2）にフロー依存している．ここで，ストア命令（4）がプリロード命令（3）に依存する場合を考えると，ループを構成する命令の依存グラフ（DAG）は図 5 (c) に示すようになる．この DAG において，乗算命令（2）で定義される変数 t1 は，プリロード命令（3）とストア命令（4）で使用されるため，命令（3），（4）間の依存によって，変数 t1 のライブレンジが引き延ばされる結果となる．

図 5 (b) の例を図 4 に即して考えると，P =（3），X =（2），Y =（4）となる．また，プリロード命令（3）とこれに依存する命令（4）は，命令（2）に依存するため，S = {1, 2} となる．集合 S の命令を複写し，命令（4）の前に挿入すると，図 5 (d) のようなコードが得られる．ここで，複写対象の命令と複写後の命令間で干渉が生じないように，変数のリネームを行う．この結果，変換後の依存グラフは図 5 (e) に示すようになる．ここで，プリロード命令（3）から複写した命令（1'）および（2'）へは，点線で示す仮想的なエッジを作成して，プリロード命令（3）を越えて上方へ移動しないようにする．

変換後のコードでは，元の変数 t1 のライブレンジが，変数 t1 と t2 に分割され，ライブレンジが短縮されていることが分かる．

3.2 モジュロスケジューリング

前項で示した依存の除去処理の適用後，メモリレイテンシ L と，最小 II を想定して，カーネル生成とスライドレジスタ割付けを試みる．

カーネル生成は基本的に文献 12) の方式と同様であり，II の値をリカレンス依存の関係および資源制約から得られる最小値 MinII から，リストスケジューリングを適用した場合の最大値 MaxII まで増加しながら，カーネルの構成を行う．

カーネルが得られると，次に浮動小数点レジスタのスライド部に対するレジスタ割当てを行う．スライドレジスタの割当てアルゴリズムは，基本的に文献 11), 17) に示されている方式をベースとしており，次のような処理からなる．(1) 浮動小数点データに関してライブレンジを求める．ここで，ライブレンジがループ繰返し全体をカバーするような変数については，割当て対象のレジスタを固定部とし，スライド部の割当て対象からは除く．(2) 各ライブレンジを，ソフトウェアパイプライニングのステージごとに分割する．分割によってできた（元々1 つのライブレンジを構成していた）ライブレンジの集合を，スライドグループと呼ぶ．(3) 分割したライブレンジに関して，干渉グラフを生成し，グラフ彩色法によりレジスタを割り当てる．スライドグループ中の i 番目のノードを G_i，スライドピッチを P とし，ノード G_0 にレジスタ番号 n のレジスタが割り当てられたとすると，各ステージ間で

各々 P だけレジスタのスライドが行われることから，ノード G_i のレジスタ番号は次式で与えられる．

$$G_i = (n - 32 + P \times i) \ mod \ 128 + 32$$

そこで，スライドグループのノードに対してレジスタを割り当てる際には，この制約を満たすようにする．

SR8000 では，FR0〜FR127 を演算で参照できるため，文献 11) と異なり，より少ない制約でレジスタを割り当てることができる．

また，スライドグループの割当てができなかった場合には，スライドレジスタ割当て失敗とし，想定したメモリレイテンシ L を減少して再スケジューリングを試みる．さらに，L の値が一定値以下に減少した場合は，プリロードによるメモリレイテンシ隠蔽効果が期待できないので，メモリレイテンシの隠蔽方式を，プリロードからプリフェッチに変更するという点が従来と異なっている．

3.3 コード生成

命令スケジュールとレジスタ割当てが得られると，従来のソフトウェアパイプライニング処理と同様に，プロローグ，カーネル，エピローグなどからなるループ構造への変換を行う．この際，ステージ間にスライド命令を挿入する．

3.3.1 スライド命令の削減

ソフトウェアパイプライニングの適用によって生成されるプロローグ部およびエピローグ部には，各々「ステージ数 − 1」個のスライド命令が生成される．プロローグ部およびエピローグ部では，カーネル部と比較して演算数が減少するため，ロード命令や演算命令など実質的な計算を行う命令と異なり，直接的に計算に寄与しないスライド命令の割合が増加してしまう．

そこで，生成されたプロローグ部およびエピローグ部に対して，スライド命令削減処理を適用し，基本ブロック中におけるスライド命令の割合を減少する．

スライドピッチ P に対して，スライド命令の実行後のレジスタ番号 R_a と実行前のレジスタ番号 R_b には，2.1 節の式 (1) で示した関係が成り立つ．この関係が満たされるように，レジスタ番号を書き換えれば，プログラムの意味を保持したままで，スライド命令を移動することができる．

そこで，レジスタ割当てを適用したプロローグ部に対して，スライド命令を上方に，エピローグ部に対してスライド命令を下方に，レジスタ番号を書き換えながら移動する．たとえば，**図 6** (a) のコードに対して，スライド命令の移動を行った結果，図 6 (b) に示すように，プロローグ部ではプロローグの開始前，エピローグ部ではエピローグの終了後にスライド命令が集まることになる．プロローグとその前のコード，および，エピローグとその後のコードの間には，レジスタのスライド関係はないため，これらのスライド命令は削除することができる．

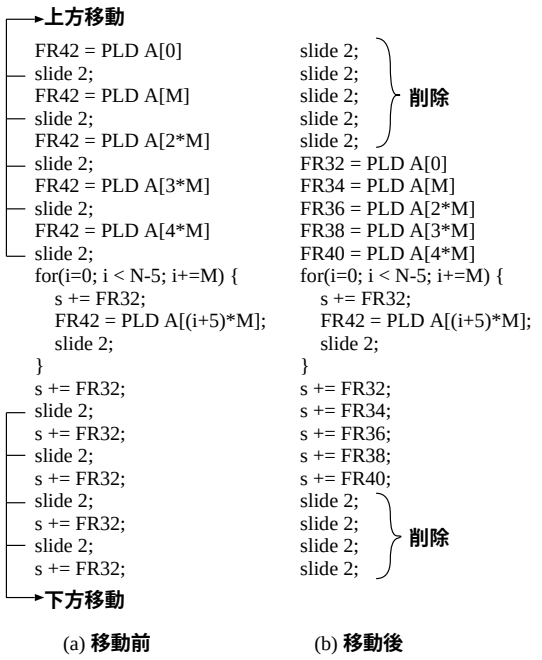
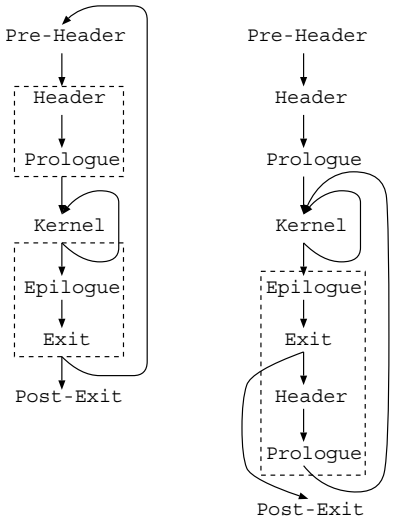


図 6 スライド命令の削除
Fig. 6 Eliminating slide instructions.

3.3.2 多重ループ向けピーリング

主記憶からのプリロードに要するレイテンシは演算レイテンシに比べ大きなため，プリロード命令を使用した場合に，プロローグおよびエピローグ部に多数の命令が生成される．ここで，プリロード対象ループが多重ループの内側にある場合，プリロード適用対象ループのプロローグおよびエピローグ部は，スケジュール可能な隣接基本ブロックを持たないため，非循環な大域命令スケジューリングの対象とならないという問題があった．

たとえば，多重ループを構成する最内側ループに対してソフトウェアパイプライニングを適用した場合，**図 7** (a) に示すようなコードが得られる．ここで，大域命令スケジューリングは非循環なパスに沿って行われるので，破線で囲んだ 2 つのパスがスケジュール対象となる．しかしながら，破線で囲んだパスのみでは，十分な並列度が得られない可能性が高い．特に，プロローグ部はレイテンシの長いプリロード命令を多く含むため，プロローグ部のプリロード命令をカーネル部の入口よりできるだけ前に移動しなければ，ループ開始時の性能の低下を生じる．



(a) **ピーリング前** (b) **ピーリング後**

図 7 多重ループ向けピーリング
Fig. 7 Peeling for nested loops.

そこで，外側ループに関して，ループピーリングを適用し，ヘッダ，および，ソフトウェアパイプライン化したループのプロローグ部を，ループ外に複写して，図 7 (b) に示すようにプログラムを変形する．このように変形することによって，破線で囲んだパスが大域命令スケジューリングの対象となり，プロローグ・エピローグ間の命令移動が可能となる．

4. 条件分岐を含むループへの適用

モジュロスケジューリングは，分岐を含むループに対して適用することが難しいという問題を持ち，これに対して様々な解決法が提案されている[8),15)]．

SR8000 では，分岐を含むループに対してソフトウェアパイプライニングを適用するために，限定的な条件付き命令実行機構を導入している．SR8000 向けコンパイラは，partial masking と呼ばれる最適化により，ループ中の条件分岐を削除することで，分岐を含むループに対してソフトウェアパイプライニングを適用することを可能とする．

Partial masking では，条件付きで行われるスカラ変数の定義を制御条件に従って値の選択を行う命令に，条件付きで行われるメモリへのストアを制御条件に従って条件付きでストアを行う命令に，それぞれ変換することによって，ループ内の条件分岐を削除する．

4.1 アーキテクチャサポート

SR8000 における条件付き命令実行のサポートを，以下に示す．

(1) プレディケートレジスタ
プレディケートレジスタは，分岐を除去する前のプログラムの，動的な制御フローに対応した真偽値を保持するために使用する．SR8000 では，1 bit × 32 個のプレディケートレジスタを定義している．

(2) プレディケート比較命令
プレディケート比較命令は，汎用レジスタ，または，浮動小数点レジスタの値の比較を行い，比較結果をプレディケートレジスタに書き込む命令である．PowerPC[6)] で定義されている比較命令と同様に，比較結果は 4 bit 単位で生成される．

なお，以下の説明では，プレディケートレジスタを $\mathtt{pr}_n$，比較命令で生成された 4 bit の条件コードのうちの 1 bit を $\mathtt{pr}_m\mathtt{.GE}$ のように表すものとする．

(3) プレディケート間演算命令
プレディケート間演算命令は，AND，OR などの 1 bit 単位の論理演算を，プレディケートレジスタ間で行うための命令である．

(4) プレディケートセレクト命令
プレディケートセレクト命令は，指定されたプレディケートレジスタの値の真偽に従って，2 つのオペランドレジスタから一方の値を選択して，ターゲットレジスタに格納する．

条件付きで定義されるスカラ変数については，変数の定義が行われるか否かを制御する分岐条件をプレディケートレジスタに計算し，セレクト命令によって定義する値を選択するように変換することで，条件分岐を削除することができる．

(5) 条件付きストア命令
条件付きストア命令は，指定されたプレディケートレジスタの値が，真であればメモリへのストアを行い，偽であればストア処理を抑止するようなストア命令である．

配列などへのストアに関して，分岐によって選択的にメモリへのストアが行われる場合，値のストアが行われる分岐条件をプレディケートレジスタに計算し，ストアを条件付きストアに変換することによって，条件分岐を削除する．

(6) 投機ロード命令
投機ロード命令は，ロード対象のアドレスが不正なアドレスであっても，プログラムの実行を停止させるような例外（致命的な例外）を生じないロード命令である．

Partial masking を適用したコードでは，ループ内のロード命令は無条件で実行されるように変換される．この結果，変換後のコードを実行する際に，

元のコードでは発生しなかった致命的な例外が生じる可能性がある．

そこで，分岐を除去した結果，致命的な例外を生じる可能性を持つロード命令については，投機ロード命令に変換し，致命的な例外の発生を抑止する．

なお，ある例外が致命的かどうかの判断はOSが行っているため，ハードウェア的には特殊な機構は用意せず，OSでロード命令によるメモリ参照例外を検出した場合に，致命的な例外を発生した命令が投機ロード命令であれば，例外を無視するようにしている．

4.2 Partial Maskingによる条件文の変換

分岐の除去処理では，(a) ループ中の各基本ブロックに関して，その制御条件をプレディケートレジスタに求め，(b) 条件付きで定義されるスカラ変数に関しては，SSA（Static Single Assignment）形式[3]に類似の構造へ変換し，(c) メモリに対するストアに関しては，制御プレディケートを付加した条件付き実行命令形式[1),4)]に変換する．これにより，ループ中から分岐を取り除く．また，分岐の除去により，プログラムの実行時に致命的例外が発生するようになることを抑止するため，致命的例外を生じる可能性を持つロード命令は投機ロード命令に変換する．

(1) 変換領域の決定

最初に適用対象領域を選択する．対象となる領域は，単一入口，単一出口のハンモック型の領域とする．領域の選択処理では，まず，各手続きの入口ブロックから開始して，DFN（Depth First Number）の最も近い制御等価な基本ブロックを求める．次に，その基本ブロック対によって支配される制御フローグラフ上の領域に対して，変換の適用可能性を判定する．ここでは，手続き呼び出しの有無，領域を構成する基本ブロックの数などを基に，適用の可否を判定する．適用可であれば，次に近いDFNを持つ制御等価な基本ブロックを求め，領域の拡大を継続する．拡大した領域が適用不可の場合は，領域を確定し，確定した領域の最終ブロックの後続基本ブロックを先頭ブロックとして，次の領域の選択を行う．

(2) 制御プレディケートの決定

適用対象の領域を構成する各基本ブロックに対して，従来の条件付き命令への変換方式[1),4)]と同様に，その制御プレディケートを求める．

ある基本ブロック BB の制御プレディケートの値は，BB の先行基本ブロックを BB_i $(i = 1 \ldots n)$，BB_i の制御プレディケートを P_i，BB から BB_i への分岐条件を C_i とすると，

$$(C_1 \wedge P_1) \vee \ldots \vee (C_n \wedge P_n)$$

により求められる．

(3) セレクト命令の挿入と変数リネーム

領域内で定義される各変数に対して，SSA形式への変換アルゴリズム[3]により，ϕ 項の挿入位置を求める．次に，各 ϕ 項に対して，到達する複数の定義から1つの値を選択するため，制御プレディケートの選択を行う．ここで，ϕ 項 $V_0 = \phi(V_1, V_2)$ において，ϕ 項での変数選択のための制御プレディケートは，V_1，V_2 の定義の所属する基本ブロックのうち，DFNが大きな方の基本ブロックの制御プレディケートとする．

求めた位置にセレクト命令を挿入するとともに，セレクト命令に到達する定義と，その使用に対して，定義が1つのみとなるように変数リネームを行う．このリネーム処理は，データフロー解析による到達定義解析を利用して，定義・使用の組ごとに一意な名前を割り振ることによって行う．

(4) 命令の変換と移動

プレディケート計算式に対して，論理式簡略化を適用した後，プレディケート間演算などを利用して，各プレディケート値の計算のために必要となる命令の追加を行う．続いて，元の制御フロー順を満たす順番で，命令を領域の先頭に移動する．この際，ロード命令の投機ロード命令への変換と，ストア命令の条件付きストア命令への変換を行う．ストア命令については，元々所属していた基本ブロックの制御プレディケートを付加して，条件付きストア命令とする．最後に，不要な基本ブロックを削除する．ここでは，単一入口，単一出口の領域のみを候補とするため，すべての命令は領域の入口基本ブロックへ移動することになる．

4.3 変 換 例

図 8 に partial masking による変換例を示す．図8 (a) に示すソースプログラムに対する，中間語の制御フローグラフは図8 (b) に示すようになる．ここで，領域選択処理によってBB0〜BB4が対象領域となったものとする．このフローグラフにおいて，条件付きで実行される基本ブロックBB1とBB3の制御プレディケートの値はそれぞれ，「$a > 0$」，「$b < 0$」となる．

プレディケート値の計算コードの挿入後，条件付きで定義される変数に対して，セレクト命令の挿入と変数リネームを行う．この結果を，図8 (c) に示す．ここでは，条件付きで定義される変数 `s` に対して，基本ブロックBB2へのセレクト命令の挿入と，変数 `s0`，

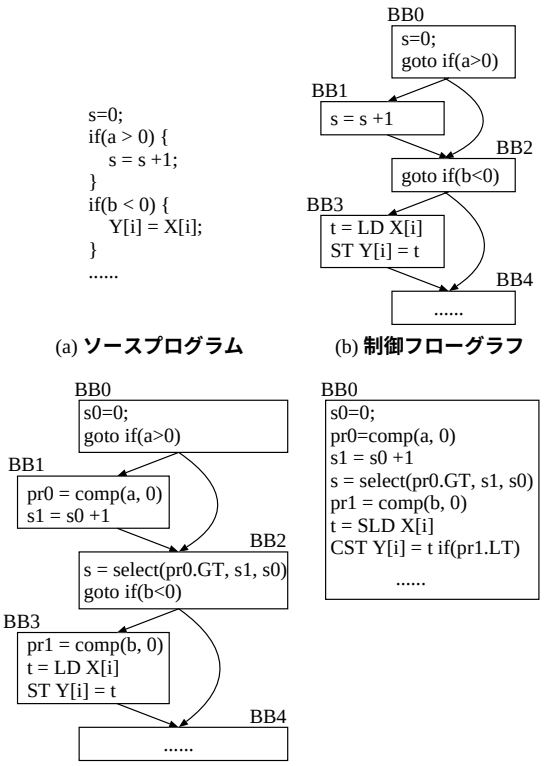


図 8 Partial masking の例
Fig. 8 Example of partial masking.

s1 へのリネームを行っている．

最後に，基本ブロックの制御フロー順を満たす順番で，分岐命令を除く各命令を，領域の入口基本ブロック BB0 に移動し，不要な基本ブロックを除去する．この際，ストア命令は条件付きストア命令（CST）に，ロード命令は投機ロード命令（SLD）に変換する．これにより，図 8 (d) に示すような，分岐を除去したコードが得られる．

元のコードにおいて，配列要素 X[i] の参照は $b < 0$ の場合にしか生じない．したがって，配列 X の添字 i が範囲外であっても，$b < 0$ でなければ，不正メモリ参照が生じることはない．一方，変換後のコードでは，配列要素 X[i] の参照はつねに行われる．このため，X[i] の参照を行うロード命令は，投機ロード命令に変換し，不正メモリ参照が生じても致命的例外が生じないようにしている．

5. 評　　価

本章では，SR8000 におけるデータプリロード処理の評価結果を示す．ここでは，まず，前章までに示したコンパイル方式の効果を示し，次に，ストライド参照におけるメモリレイテンシ隠蔽機構の基本性能を示す．最後に，SPEC95 浮動小数点ベンチマーク[13] を対象とした，性能向上効果を示す．

なお，以下の性能測定は，ノード内自動並列化を抑止して，SR8000 の 1IP のみを使用して行った．実行時間などの測定は，ハードウェアパフォーマンスモニタにより計測している．

5.1 コンパイラ最適化の効果

SPEC95 の浮動小数点系ベンチマークを対象として，前章までに述べたプリロード向け最適化の評価を行った．この結果を**図 9** に示す．評価は，上記の最適化を適用しなかった場合と，適用した場合，それぞれについて，ソフトウェアパイプライニングの適用対象となったループ数と，各ループのモジュロ展開数の平均値を求めることにより行った．図 9 において，「applied loops」は，最適化を適用した場合における，ソフトウェアパイプライニングの適用対象ループの増加率，「modulo expansion」はモジュロ変数展開数の減少率を表している．

この図から，依存分割による依存の除去や，partial masking による分岐除去などの最適化を適用することにより，ソフトウェアパイプライニングの適用ループが，最大 2.7 倍に増加していることが分かる．また，命令複写などの最適化によって，汎用レジスタへ割り当てられる整数型変数のライブレンジを縮小することにより，モジュロ変数展開数に関しても，最大で 1.7 倍の削減効果が得られている．

特に，hydro2d では，最適化によってソフトウェアパイプライニングの適用率が著しく向上し，turb3d ではモジュロ変数展開数が減少している．なお，hydro2d など一部のベンチマークでは，モジュロ変数展開数が増加しているものもある．これは，ソフトウェアパイプライニングの適用対象全体が増加したことによる．

5.2 基本ループ性能

メモリ系の基本性能を示すため，DAXPY（$Y_i = A * X_i + Y_i$）ループについて，ストライド値を変化させた場合の性能測定結果を**図 10** に示す．この図は，プリフェッチおよびプリロードを使用しない場合（LD），プリフェッチのみを使用した場合（PF），プリロードのみを使用した場合（PLD），それぞれについて，ストライドを 1 から 19 まで変化させた場合の相対性能を示している．ここでは，プリフェッチおよびプリロードを使用しない場合で，ストライド値 1 のケースの性能を 1.0 としている．評価対象ループのループ長は，1,000,000 とした．

図 10 の結果から分かるように，プリフェッチおよ

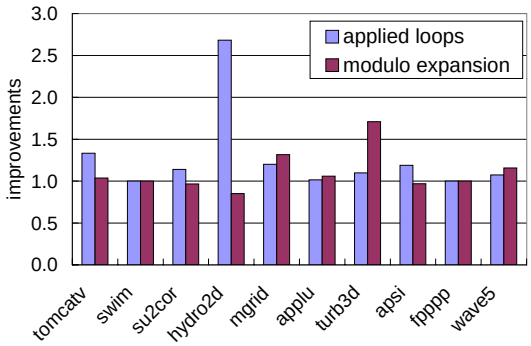


図 9 プリロード向け最適化の効果
Fig. 9 Effects of preload optimization.

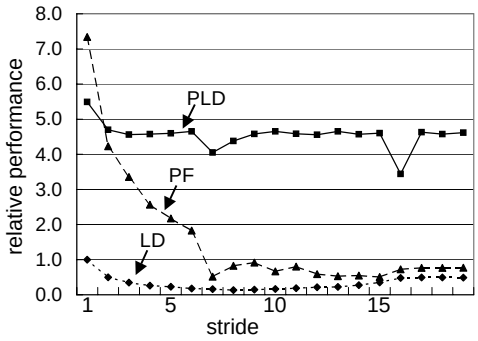


図 10 DAXPY ループの性能
Fig. 10 Performance of DAXPY loop.

びプリロードを使用しない場合は，ストライドが小さな連続参照の場合においても，低い性能しか得られていない．

これに対して，プリロードを利用した場合は，ストライドが大きくなっても性能低下は低く抑えられており，連続参照のロードの 4〜5 倍の性能を維持している．

一方，プリフェッチを利用した場合を見ると，連続参照のケースでは，プリロードよりも高い性能を得ているが，ストライド値が増加するにつれ性能が急激に低下しており，連続参照のロードの性能と同程度まで，性能が低下している．

以上から，SR8000 においては連続参照能力はプリフェッチが勝るが，大ストライドやランダム参照に対する耐性は，プリロードを用いた方が高いことが分かる．

5.3 SPEC95 ベンチマーク

次に，SPEC95 の浮動小数点系ベンチマークを対象として，性能評価を行った結果を，**図 11** に示す．この図では，プリフェッチおよびプリロードを利用しない場合（LD）の性能を 1.0 として，プリロードのみを使用した場合（PLD），プリフェッチのみを使用した場合（PF），プリロードとプリフェッチを併用した場合（PLD+PF）の相対性能を示している．ベンチマークの入力データには，SPEC95 ベンチマーク付属の 3 つの入力データのうち，reference データを使用した．

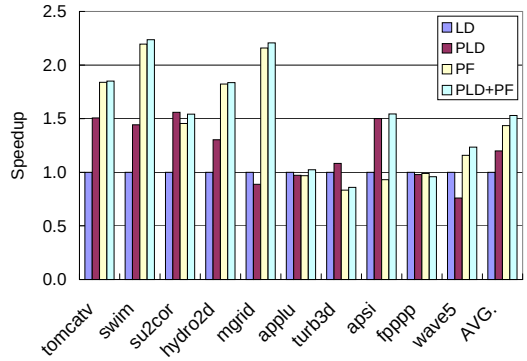


図 11 SPEC ベンチマークの性能
Fig. 11 Performance of SPEC benchmark.

この結果から分かるように，ロードのみを利用した場合の性能に対して，プリロードのみでは最大 1.56 倍（平均 1.20 倍），プリフェッチのみでは最大 2.20 倍（平均 1.44 倍），両者を組み合わせて選択的に利用することによって最大 2.24 倍（平均 1.53 倍）の性能が得られている．

su2cor や turb3d など一部のベンチマークでは，プリロードのみを使用した場合の方が，プリフェッチのみを使用した場合よりも性能向上している．これはキャッシュ競合によるスラッシングの影響などによるものと予想される．

一方，mgrid や applu などは，プリロードのみを利用した場合に，性能があまり向上していない．ソフトウェアパイプライニングを適用したループを実行するためには，一定値以上のループ長を必要とする．このため，ループ長が一定値以下の場合には，ロード命令のみを使用したループが実行されることとなる．また，配列間の依存関係が不明な場合には，ストア・ロード間の依存により，プリロードを先行的に発行することができない場合がある☆．こういった理由により，これらのプログラムではプリロードが有効に動作していない．

単一の方式を利用した場合と比較して，プリロード

☆ ディレクティブやオプションを利用することによって，ユーザが配列間の依存の有無を与えることは可能．

とプリフェッチを併用したケースでは，各々の方式の特徴が活かされており，ほとんどのプログラムで最も良い性能を得ている．

6. 関連研究

メモリレイテンシを隠蔽するための手法としては，non-blocking キャッシュを用いる方法[5] と，データプリフェッチ[10] による方法が広く利用されている．Non-blocking キャッシュは，ロード命令がミスした時点でストールせず，後続する命令の実行を継続する機構である．一方，データプリフェッチは，主記憶からキャッシュへのデータ転送を先行的に行う仕組みである．SR8000 では，プリロード，プリフェッチに加えて，non-blocking キャッシュを実装しており，比較的小さなメモリレイテンシを隠蔽し，メモリ参照による性能低下を軽減するのに有効に利用している．ただし，non-blocking キャッシュは，メモリレイテンシをある程度低減するためには有効であるが，メモリレイテンシを隠蔽することは，ハードウェア量から現実的ではない．これに対して，データプリフェッチでは，十分前にプリフェッチを発行することにより，メモリレイテンシを隠蔽することが可能である．ただし，データプリフェッチでは，ストライドの大きな参照や，リストベクトルの参照など，非連続なデータ参照を行った場合に，演算に利用しないデータもキャッシュへの転送対象となるため，バスの使用効率が低下してしまう．これに対して，SR8000 では，キャッシュ再利用性の期待できる連続参照に対してはデータプリフェッチ，非連続参照に対してはデータプリロードを使い分けることにより，メモリレイテンシを効果的に隠蔽している．

SR8000 のスライドレジスタ機構は，SR2201 の疑似ベクトル機構[11] を拡張したものであり，演算などで参照可能なレジスタの拡大，条件分岐を含むループのための partial masking のサポートといった点が拡張されている．本論文で示したように，プリロードを利用してメモリレイテンシを隠蔽するためには，ソフトウェアパイプライニングの適用が不可欠である．参照可能なレジスタの拡大と partial masking のサポートは，ソフトウェアパイプライニングの適用条件を緩和するのに有効である．

SR8000 に見られるような，部分的な条件付き実行機構を持つアーキテクチャに対するコンパイル方式としては，Mahlke らによる partial if-変換[9] がある．この方式では，一度すべての命令にプレディケートを付加した形式に変換する．このコードに対して，predicate promotion と呼ばれる手法を適用することで，プレディケートを付加しなくても結果に影響のない命令から，プレディケートを除去する．次に，条件付きの命令を条件付き move などを使用した命令列に置き換える．この置き換えは，パターンマッチングにより行われ，無駄な命令を多く生成するため，partial if-変換ではこの冗長コードを peephole 最適化で除去することを提案している．しかしながら，peephole 最適化は，局所的なコードパターンを検査することによって，冗長コードを削除する最適化であり，広範囲で分岐の除去を行った場合には，冗長コードが残ってしまう可能性が高く，生性コードの質が低下する可能性がある．大域的な共通式削除や冗長性削除など，より強力な最適化を利用すれば，冗長コードを削除することは可能であるが，中間的なコードの増大は，コンパイル処理時間の増大を招く．これに対して，本論文で示した方式では SSA 形式への変換手法を適用することで部分的な条件付き実行形式への変換を行っていることから，partial if-変換と比較して冗長なコードが生成される可能性は低く，中間的なコードの増大も生じない．

7. おわりに

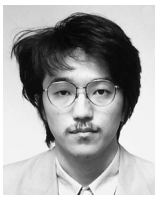
科学技術計算など大規模計算では，メモリレイテンシの隠蔽が重要な課題となる．SR8000 では，データプリロードとデータプリフェッチを利用して，データの移動をソフトウェア的に制御することにより，メモリレイテンシを効果的に隠蔽することができる．本論文では，SR8000 のデータプリロード機構を効果的に利用するための，コンパイラ最適化手法について説明した．また，実機上で評価を行うことにより，SR8000 に対する最適化手法が，メモリレイテンシの隠蔽に高い効果を持つことを確認した．

参考文献

1) Allen, J., Kennedy, K., Porterfield, C. and Warren, J.: Conversion of Control Dependence to Data Dependence, *Proc. 11th Annual ACM Symposium on Principles of Programming Languages*, pp.177–189 (1983).
2) Callahan, D., Kennedy, K. and Portfield, A., Software Prefetching, *Proc. 4th International Conference on Architectural Support for Programming Languages and Operating Systems* (1991).
3) Cytron, R., Ferrante, J., Rosen, B., Wegman, M. and Zadeck, F.: An Efficient Method of Computing Static Single Assignment Form,

Proc. 16th Annual ACM Symposium on Principles of Programming Languages, pp.23–35 (1989).
4) Dehnert, J. and Towle, R.: Compiling for the Cydra 5, *Journal of Supercomputing*, Vol.7, No.1/2, pp.181–227 (1993).
5) Farkas, K. and Jouppi, N.: Complexity/ Performance Tradeoffs with Non-Blocking Loads, *Proc. 21st Annual International Symposium on Computer Architecture*, pp.211–222 (1994).
6) IBM Microelectronics and MOTOROLA: *PowerPC Microprocessor Family: The Programming Environments* (1994).
7) Kurihara, T., Shimada, K., Kamada, E. and Shimizu, T.: A RISC Processor for SR8000: Accelerating Large Scale Scientific Computing with SMP, *Hot Chips 11* (1999).
8) Lam, M.: Software Pipelining: An Effective Scheduling Technique for VLIW Machines, *Proc. ACM SIGPLAN '88 Conference on Programming Languages Design and Implementation*, pp.317–327 (1988).
9) Mahlke, S., Hank, R., McCormick, J., August, D. and Hwu, W.: A Comparison of Full and Partial Predicated Execution Support for ILP Processors, *Proc. 22nd Annual International Symposium on Computer Architecture*, pp.138–150 (1995).
10) 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.62–75 (1992).
11) Nakamura, H., Imori, H., Nakazawa, K., Boku, T., Nakata, I., Yamashita, Y., Wada, H. and Inagami, Y.: A Scalar Architecture for Pseudo Vector Processing based on Slide-Windowed Registers, *IEEE Proc. Supercomputing '93*, pp.298–307 (1993).
12) Rau, B.: Iterative Modulo Scheduling: An Algorithm For Software Pipelining Loops, *Proc. 27th Annual International Symposium on Microarchitecture*, pp.63–74 (1994).
13) Standard Performance Evaluation Corporation: SPEC CPU95 Benchmarks, http://www.spec.org/osg/cpu95/.
14) Tamaki, Y., Sukegawa, N., Ito, M., Tanaka, Y., Fukagawa, M., Sumimoto, T. and Ioki, N.: Node Architecture and Performance Evaluation of the Hitachi Super Technical Server SR8000, *Int'l Conference on Parallel and Distributed Computing and Systems*, pp.487–493 (1999).
15) Warter, N., Bockhaus, J., Haab, G. and Subramanian, K.: Enhanced Modulo Scheduling for Loops with Conditional Branches, *Proc. 25th Annual International Symposium on Microarchitecture*, pp.170–179 (1992).
16) Wolfe, M.: *High Performance Compilers for Parallel Computing*, Addison-Wesley (1996).
17) 萩川友宏，添野元秀，山下義行，中田育男：スライドウィンドウを考慮したレジスタ割付，情報処理学会論文誌，Vol.39, No.9, pp.2684–2694 (1998).

(平成 11 年 9 月 8 日受付)
(平成 12 年 6 月 1 日採録)



西山　博泰（正会員）

1965 年生．1993 年筑波大学大学院博士課程工学研究科修了．同年（株）日立製作所中央研究所入社．現在，同社システム開発研究所にてコンパイラ最適化の研究開発に従事．ACM 会員．博士（工学）．

久島伊知郎（正会員）

1986 年東京大学理学部情報科学科卒業．同年（株）日立製作所に入社．以来同社システム開発研究所．研究テーマ：プログラミング言語，コンパイラ．

菊池　純男（正会員）

1952 年生．1978 年東京工業大学大学院理工学研究科電気工学専攻修了．同年（株）日立製作所中央研究所入社．現在，同社システム開発研究所企画室長．コンパイラ最適化技術，並列化技術の研究開発に従事．コンピュータアーキテクチャにも興味を持つ．ACM，IEEE 各会員．