解　説

RISC 系の細粒度並列マシン†

斎藤光男††　井上　淳††

1. はじめに

RISC (Reduced Instruction Set Computer) アーキテクチャをもつマイクロプロセサが，市民権を得てからすでに 10 年近く経ている．その間の性能向上はめざましく，単一の CPU としては，最も高性能なものになろうとしている．その理由としては，半導体技術の進歩とともに，アーキテクチャ上の進歩も見逃せない．ここでは，RISC 登場以来その高速化のために払われた，アーキテクチャ上の工夫，特にその細粒度並列化への試みについて解説する．

マイクロプロセサは，1971 年に発表された Intel 社製の 4004 以来一貫して高速化への道を歩んできている．そのなかで，最も特筆すべき出来事は，現在事実上のパーソナルコンピュータ用の標準になっている，8086 系のプロセサの出現と，RISC の登場であろう．

RISC は，もともと IBM 社の 801 プロジェクト[1]でその有効性が確認されていたが，UC Berkeley の Patterson らによる，RISC-I, RISC-II[2] で話題になり，RISC という名前が定着した．その後実用にされたのは，IBM 社の PC-RT であったが，性能が低かったため，あまり受け入れられなかった．本当に世の中に衝撃を与えたのは，SUN 社のワークステーション SUN-4 に使われた SPARC プロセサである．このマイクロプロセサは，大部分がゲートアレイで作られていたにも関わらず，その当時，最も高速なマイクロプロセサの 3 倍近い能力をもっていたために，プログラムの互換性の壁をやすやすと乗り越え，世の中に受け入れられていった．

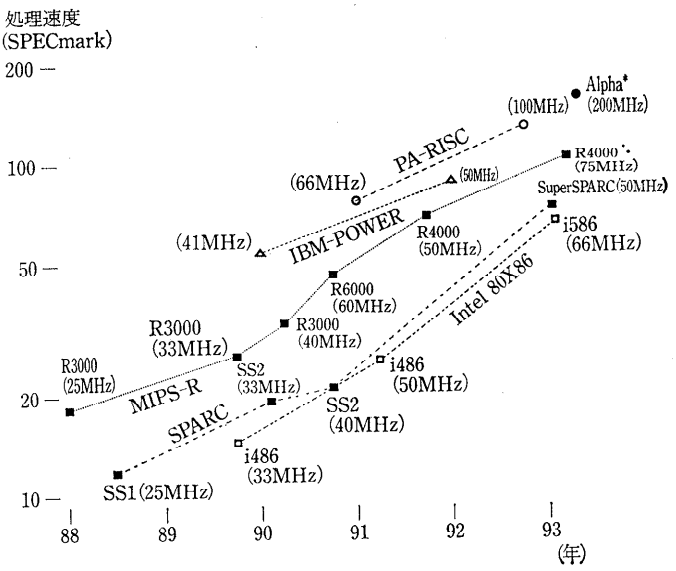


図-1　RISC 系プロセサの発展過程

この当時は，SUN, MIPS といったベンチャ企業が関わっていたが，その成功に刺激され，Intel, Motorola などの半導体メーカ，IBM, DEC, HP などのコンピュータメーカも参入又は再参入し，混戦模様になっている．

このような状況のなかで，当初ゲートアレイで作られていた RISC は，それ以前のマイクロプロセサと同様のフルカスタムの手法で作られるようになっていき，さらに VLIW, Super Pipeline, Super Scalar などの細粒度並列処理を取り入れることにより，急速な発展を遂げてきた（図-1）．

ここでは，RISC の発展過程のなかで取り入れられてきた，細粒度並列処理の原理，実現方法，コンパイラ技術などについて述べ，将来動向を占う．

† RISC Derived Fine Grain Parallel Machines by Mitsuo SAITO and Atsushi INOUE (Toshiba Research & Development Center, Information and Communication Research Laboratory).

†† (株)東芝研究開発センター情報通信システム研究所

2. RISC 系の細粒度並列アーキテクチャ

RISC アーキテクチャはそれまでのプロセサの複雑化の歩みを否定し，単純化することによる高速化を目指したもので，その定義は，Patterson によれば以下のとおりである．

1. 簡単な命令のみによる1クロック1命令実行
2. すべての命令を同一サイズにしデコードを単純化する
3. ロード，ストア命令のみによるメモリアクセス，それ以外の命令は，すべてレジスタ間の演算のみにする
4. アセンブリ言語ではなく高級言語とコンパイラの使用を前提とする

これらの簡単化によって，クロック周波数を上げ，かつ命令を原則として1クロックで終了させることにより高速化を図ろうとするものであった．

このアイデアは，32ビットアーキテクチャで初めて意味をもつもので，しかもその当時の半導体技術が，その1チップ化に適合するものであったため，効果的であることが実際に証明された．そのためこれらのアイデアをさらに発展させることによって高速化を図ろうとする動きが生まれてきた．その着目点によって大きく3種類に分類される．

2.1 VLIW (Very Long Instruction Word) アーキテクチャ

細粒度並列のアイデアとしては最も古く現れたもので，RISC が垂直型のマイクロプログラムをそのまま命令セットにしたようなものであるのに対して，VLIW は水平型のマイクロプログラムを命令セットにしたと考えることができる．このアイデアの原型は，京都大学の，QA-1, QA-2 にみられるが[3]，VLIW という名称は，1983 年ごろ，Fischer によって提唱され，当初は，RISC の 10 から 30 倍の性能が得られるものと楽観的に考えられていた[4]．図-2 は，典型的な VLIW 計算機の構成図である．この計算機は，通常 256 ビットから 512 ビットの命令語長をもち，数個の整数演算器と，浮動小数点演算器をもっている．これらの演算器はすべて同時に動作し，コンパイラは，これらのハードウェア構成を完全に意識し，タイミングまで考慮したオブジェクトを生成する．

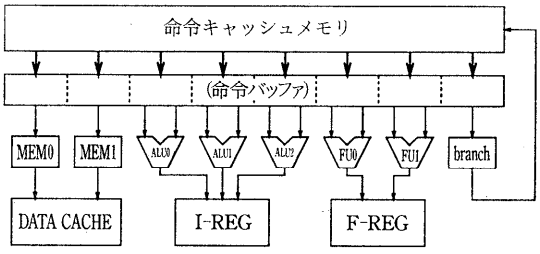


図-2 VLIW 計算機の構成例

しかしこの方式は，Trace Scheduling 方式を始めとするさまざまな最適化技法が提案されたにも関わらず[4),5)]，並列度は，一般のプログラムではそれほど高くならず，思うように性能が上がらなかった．そのため，命令語長が長いことによるオブジェクトコードのサイズが大きいという欠点が目立ってしまった．さらに致命的だったのは，ハードウェアにきわめて近いアーキテクチャを命令セットとしているため，ハードウェアの改良のたびに，再コンパイルする必要があり，オブジェクトレベルの互換性を保つのがきわめて難しかったことである．そのため，オブジェクトを基本とする商業的な流通ソフトウェアが育たず，特殊な用途でしかメリットが生かされなかったため，一般的にはならなかった．しかし，コンパイラの最適化技術の向上にはたした役割は大きく，その技術の多くは，スーパスケーラ型のプロセサ用のコンパイラに生かされている．これについては章を改めて説明する．

また最近では，VLIW をベースにして，互換性を取る試みも行われており，実用的なアーキテクチャとして復活する動きもある．

2.2 スーパパイプライン (Super Pipelined) アーキテクチャ

この方式は，RISC の単純なパイプラインに着目した方式である．RISC では，4，5段の単純なパイプラインを採用している．そのため，このパイプラインをさらに分割して，パイプ1段当たりの処理を少なくすることにより，より多くの命令を処理することができる．ここに MIPS 社の R4000 のパイプラインをあげる (図-3)[6]．これを同じ MIPS 社の R3000 と比較すると，違いがよく分かる (図-3)．R3000 は，もともとクロックの半周期をうまく使って，命令用とデータ用のアドレスとデータピンを共用するとともに，ブランチによるペナルティを1個のデレイド命令を採用

することにより，完全に防いでいる．一方 R 4000 では，すべてのパイプステージをクロックの半周期で流すことにより，2倍の命令を処理できるようにしている．そのなかで特徴的なのは，演算器の回路構成を工夫し，演算ステージを半周期で行っていることである．これを行わないと，大半のプログラムでは，連続する命令間に依存性があるために，ハザードが発生し，事実上1クロックに1命令しか実行できなくなってしまう．ただし，これを行っても性能が2倍になるわけではなく，ブランチのペナルティ，メモリアクセスのレーテンシが増加するために，性能向上は，1.5 倍程度にとどまる．またパイプのピッチが小さくなるために，クロックスキューの影響を受けやすくなり，クロック周波数を落とさないためには，慎重な設計を必要とする．しかし，うまく設計すれば，ハード量の増加は少なく効果的な方法と言える．そのため最近は，次に述べるスーパスケーラと組み合わせることにより，高性能化する動きがある．DEC 社のアルファプロセサも明確にはうたっていないが，この方向と考えられる[7]．

2.3 スーパスケーラ (Super Scalar) アーキテクチャ

スーパスケーラ（スーパスカラともいう）アーキテクチャは，VLIW の改良として現れた．その原理は，**図-4** のように，複数の独立した命令を同時にキャッシュメモリからフェッチしてきて，それらの命令を同時に解析して，必要な演算器に渡すものである．VLIW との違いは，独立した命令列を同時にフェッチするため，命令の並び方に制限がなく，演算器が不足した場合や，依存関係があり，同時に実行できない場合は，複数クロックかけて実行するようにハードウェアで制御するものである．このようにすることにより，ハードウェアは複雑になるが，既存の命令セットと互換性を保つことができ，新しい命令セットを定義する際も，ハードウェア独立とすることができる．このアーキテクチャでは VLIW と同様に周波数を上げることなく，性能を上げることができるためその複雑さにも関わらず最近発表されているプロセサは，このアーキテクチャをとるものが多い．**図-5** に最近発表された RISC 系のプロセサの一覧表をあげる．また CISC 系のプロセサでも INTEL 社製の Pentium (旧称 80586) ではこのアーキテクチャを採用するとのことで，ますます広がる傾向にある．次章で，実際の設計上の問題点について解説する．

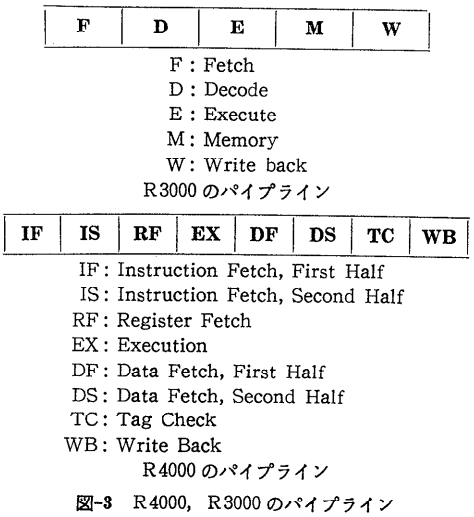


図-3 R4000, R3000 のパイプライン

3. スーパスケーラプロセサのデザイン

スーパスケーラプロセサの並列度は，原理的に VLIW と同じであるが，ハード的に競合を解決するために，さまざまな工夫を必要とする．また既存のプロセサとアーキテクチャ的な互換性を保とうとすると，さらに問題が複雑化する．ここでは，実際のスーパスケーラプロセサのデザイン上のキーポイントについて解説する．

3.1 命令供給

スーパスケーラプロセサにとって，複数の命令を適当な演算ユニットに供給するのは，RISC または，VLIW に比べてはるかに大変な作業である．これは，それぞれの演算ユニットの数は，通

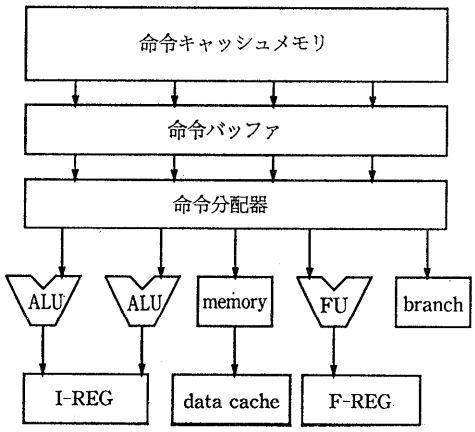


図-4 スーパスケーラ計算機の構成例

常同時に発行できる命令より少なく，同時に発行できる命令の種類に制限がないためである．デコードフェーズでは，命令のデコード以外に，どの演算ユニットへ供給すべきか，また可能かどうかも判断する必要がある．そのため，デコードフェーズ以外に，命令分配のために1ステージもうける場合もある[8]．しかし，このようにすると，分岐のペナルティが大きくなるため，以下に述べるプリデコードが用いられる場合がある[9]．この原理はメインメモリからキャッシュメモリに命令をロードする際に，デコードを行い，使用する演算器の種類や，供給可能性などをあらかじめ調べておき，その情報をキャッシュメモリに蓄えることにより，デコードフェーズの負担を減らすものである．

必要な命令を演算器に渡すには，素直に考えると命令バッファから命令を演算器に供給するためにマルチプレクサを演算器の個数分だけ用意すればいいわけであるが，同時供給命令数が多い場合は配線量が膨大になり，非現実的になってしまう．そのために複数の命令を複数の演算器に一度に振り分けるために特別につくられたクロスバースイッチを使うことが多い．プリデコードとクロスバースイッチの組合せにより，デコード，演算器への命令供給，レジスタの読み出しを1サイクルで行うことができる．これにより命令供給に関するオーバヘッドを RISC 並に抑えることができる．

3.2 分岐予測

スーパスケーラプロセサでは，複数の命令を同時に実行するために，分岐によるペナルティは，通常の RISC に比べて同時に実行可能な命令数分だけ大きくなる．そのためもはや RISC で採用されたデレイドブランチはそのデレイドスロットが多くなるため，ほとんど効果がなくなる．そのためスーパスケーラ専用に設計されたアーキテクチャではデレイド命令は採用していない．また既存のプロセサと互換性をとるためには，デレイド命令を1個に制限するか，またはなくさなければならないそのため，並列度の大きなスーパスケーラプロセサでは，分岐予測器 (Branch Target Buffer) をもつようになっていくものと思われる．分岐予測器は，一度分岐が発生すると，その分岐先をテーブルに登録しておくことにより，2回目からは分岐先からフェッチを行うことにより，分岐によるペナルティを軽減することができる．この概念は古くからあったが，特殊な例外をのぞきスーパスケーラプロセサで意味をもつようになってきたものである．また最近は，この分岐テーブルを命令キャッシュと共用する提案がなされ，今後それを使ったものが増えるものと思われる[10]．

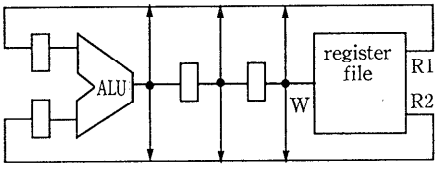
3.3 バイパス

通常のプロセサでは，実際に結果をレジスタに書き込むまでには時間がかかる．等価的に次のサイクルで，書き込まれたように見せかけるために，演算器からの出力をただちに次の計算のための入力に使えるようにしている．それをバイパス (Bypass) と呼んでいる．典型的な RISC におけるバイパスは**図-6**(a)のとおりである．これがスーパスケーラだと**図-6**(b)のようになり，きわめて

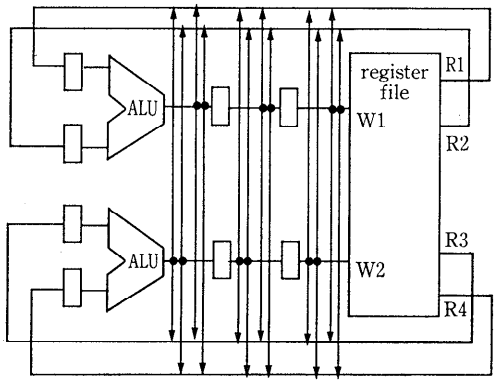
開発元	DEC	HP	IBM/Motorola	SUN/TI	MIPS
製品名	Alpha	PA-RISC	Power PC 601	SuperSPARC	R4400
最大動作周波数	275 MHz	100 MHz	80 MHz	60 MHz	200 MHz
演算ユニット数	IU1, FPU1	IU1, FPU1	IU1, FPU1	IU3, FPU1	IU1, FPU1
最大並列実行命令数	2	2	2	3	1
内蔵キャッシュ構成	I: 8K, D: 8K	な し	I/D 共用 8K	I: 20K, D: 16K	I: 8K, D: 8K
チップ寸法 (mm)	16.8×13.9	14.0×14.0	15.2×14.6	15.98×15.98	15.2×12.6
Tr 数	168万	85万	120万	310万	120万
パッケージ	431 ピン PGA	504 ピン PGA	218 ピン PGA	293 ピン PGA	447 ピン PGA
プロセス	0.8 μmCMOS	0.8 μmCMOS	0.8 μmCMOS	0.8 μmBi-CMOS	0.8 μmCMOS
電源電圧	3.3 V	5.0 V	3.6 V	5.0 V	5.0 V
消費電力	30 W	20 W	4 W	8 W	15 W

図-5 RISC 系プロセサ一覧

複雑になる．すなわち同時に発生する出力は，すべての入力に接続されなければならないために，おのおののゲートの入力容量，及び出力容量は馬鹿にならない．これによる遅れ時間は，適切に設計されたALUの遅れ時間に近い値を示す．このままでは，サイクルタイムはALUの速度の半分になってしまうため，それを防ぐために，ALUの出力や，データキャッシュメモリの出力のみを別扱いするなどの工夫がなされる．しかしこれを行っても，マルチプレクサが入ってしまうために，大幅な性能向上は難しい．ALUの速度がクリチカルパスになるような設計の場合（本来これが望ましいが）単純なRISCに比べて20%程度のサイクルタイムの低下は免れない．このため整数演算中心のプログラムでは，**5.**で述べるようにALUを2個もつことによる性能向上は，せいぜい30%程度であるので，ALUを2個もってもメリットがないことになってしまう．しかし実際には，クリチカルパスはデータメモリや，浮動小数点演算であることも多く，浮動小数点演算中心の問題では，プログラムの流れが比較的単純で，しかも整数演算系の命令が複数個実行できるメリットが大きい場合が多いので，どのような処理に重点を置くかにより決定すべきである．



（a） 典型的なRISCのバイパス



（b） スーパスケーラ（2スケーラ）のバイパスの例
各ステージごとのバイパスの本数は4倍になる

図-6 バイパスの例

3.4 トラップ，ストール

スーパスケーラプロセサではトラップや，キャッシュミスなどの際に全体を待たせるためのストール機能も複雑化する．スーパスケーラアーキテクチャでは，トラップの原因となる命令と同時に実行されている命令が存在しうる．しかし同時に実行されていても意味的にトラップの原因となった命令より前に実行されるはずの命令は，完了しなければならないし，後のものは完了させてはならない．これを完全に行うためには，同時に実行された命令間の順序をトラップ発生時に判定する回路が必要になってくる．この回路は，かなり複雑になる．これをなくすためには，たとえば，同時に実行された命令はすべてキャンセルし，実行を再開すべき命令の番地と，トラップの原因となった命令の番地を区別してOSに通知するなどの方法が必要になってくる．たとえば**図-7**のようにある時間（$t1$）に3命令が同時に実行されていたとし，101番地の命令がトラップを起こした場合，正しくは100番地のadd命令は完了させ，102番地のsub命令はキャンセルしなければならない．その替わりに3命令すべてキャンセルし，OSが101番地の命令のトラップ原因を取り除いた後，100番地から実行を再開してもよい．

またキャッシュミスなどによる停止のためには，回路量が多くなった分，一つの信号で大量（数千）のゲートに信号を供給する必要がでてくる．このための時間遅れは，場合によっては，1クロック近くになってしまい，たとえクロックのはじめのほうで，原因が分かっても，そのクロック内で停止させることが困難になり，クロック周波数の低下を招く．これを防ぐためには，その原因別に，停止させる部分を限定し，それ以外の部分は無効命令を実行し続けるような，パイプライ

```
(time)                                   (address)
        load    r1,$a
t0      mul     r1,r1,r4
        store   r4,$b
        add     r1,r2,r3                 100
t1      load    r0,$b                    101
        sub     r3,r4,r5                 102
        store   r5,$c
t2      load    r2,$d
        sub     r2,r3,r4
```

図-7 トラップ処理の例

ンステージごとの非同期実行形式をとることが行われる．この考え方は，ミクロレベルでデータフローマシンの考え方を取り入れているといえる．

3.5 メモリシステム

メモリシステムは，CPU が高速になるに従って重要性を増してきている．特にスーパスケーラプロセサでは，複数の演算が同時に行われるため，そのバンド幅が問題になりつつある．従来は，キャッシュメモリをチップ内に内蔵することで問題が解決できると考えられてきたが，実際は内蔵できるキャッシュメモリが小さいことと，結局結果はメインメモリまたは，ディスクに書かないと意味がないため，一定のバンド幅を確保することの重要さが認識され始めている．

そこで最近の動きとしては，内蔵キャッシュメモリは比較的小さくし，外部キャッシュメモリを高速に動作させることにより，性能を上げる方向が考えられている．そのための工夫として，いわゆるノンブロッキングキャッシュ方式がある．この方式は，通常のキャッシュメモリではミスが発生すると，CPU は停止してしまうのに対し，ただちにはCPUを停止させず，ロードの場合はロードされてきたデータにアクセスして初めてCPU を停止させるとともに，リフィル中でも次のリフィルを開始できるようにするものである[11]．これは図-8 のようにリフィル動作をパイプライン化することにより達成できる．この技術と，ロード命令をできる限り速く出すというコンパイラの最適化により，内部キャッシュメモリがミスしても性能の低下を防ぐことができる．この種の最適化は特に問題が深刻な浮動小数点演算に関してはかなり効果的であることが知られている．

また，述べてきたような細粒度並列だけではその並列度に限界があるため，マルチプロセサをサポートするメモリシステムが，一般的になってきている．このための機構としては，キャッシュメモリの一貫性を保つのを効率良くするための，ウイークメモリモデルの採用や，メモリの読出し要求と，読み出したデータの転送を独立させるスプリットバスや，さらにアドレスとデータバスを分離し非同期に動作させることにより，広いバンド幅を確保するようになってきている．

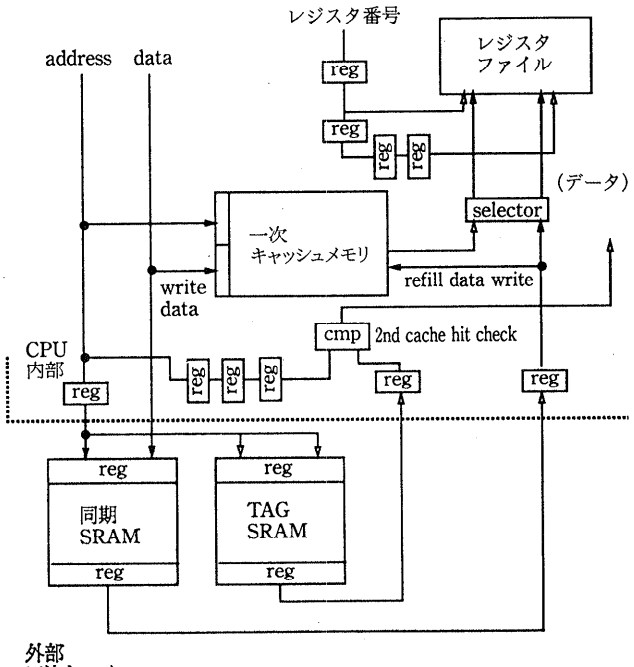
4. コンパイラの最適化技術

細粒度並列マシンのためのコンパイラ技術としては，従来 RISC 用に開発された最適化技術と，マシンの並列処理能力を活用するために新たに開発された最適化手法を組み合わせて使用することが多い[12]．

4.1 RISC 用最適化技法の適用

RISC 用の最適化技法としては，マシン独立なプログラム変換手法と，マシンのアーキテクチャに依存した最適化手法がある．たとえば，手続きの inline 展開，ループ不変式の抽出，共通部分式の抽出などはマシン独立な手法であり，細粒度並列マシンにおいても有効である．ただし，以下では RISC の命令実行方式に依存した最適化手法に着目し，その細粒度並列マシンにおける適用方法を述べる．

リストスケジューリング パイプライン化された演算ユニットで単純な命令列を実行する RISC マシンでは，命令の依存関係によるロックの発生



本構成例では，一次キャッシュメモリにヒットしない場合は4クロック，データロードが遅れるが，データを使用しない限り CPU は停止しない

図-8 ノンブロッキングキャッシュメモリの例

が実行性能を著しく低下させてしまう．リストスケジューリングはRISCマシン用の最も基本的な最適化手法で，コード生成時にパイプラインの停止（ストール）を回避するために基本ブロック内の命令を並べ替える処理であり，いくつかの発見的アルゴリズムが提案されている[13)~17)]．細粒度並列マシンに適用する場合，通常RISCで行う命令依存グラフ上でのクリティカルパス解析によるスケジューリングに加え，複数の演算ユニット使用の競合，マルチサイクル実行するユニットの動作条件などを考慮したアルゴリズムを適用することが必要である．

ループアンローリング　ループアンローリングはループ内演算を複数個展開し，ループ反復数を減らすプログラム変換で，分岐回数を減らすことでループ実行効率を高める最適化である．ループアンローリング自体はマシン独立な最適化手法であるが，RISCで前述のリストスケジューリングと組み合わせて適用すると並べ替えの対象となる命令数が増加するのでスケジューリングがより効果的になる．細粒度並列マシンでは1クロック当たりの命令実行スロットが多いので，この効果はさらに著しい．

4命令同時実行可能なスーパスケーラプロセサでLivermore loopに対してアンローリング段数を変えた場合の性能比を調べたものが**図-9**である[18)]．一般にRISCマシンでは比較的小さな展開段数（2または4）でアンローリングの効果は飽和してしまうが，図-9から分かるように細粒度並列マシンでは処理ループの演算内容に応じて最適な展開段数を判定してアンローリングを行うことが必要である．またコードサイズ増加による命令キャッシュミスの効果も考慮して処理することも必要である．

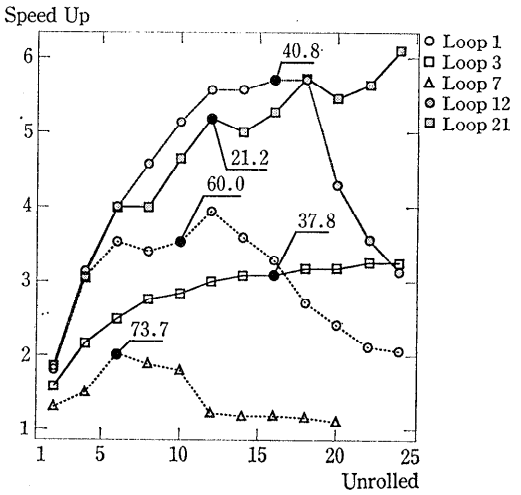


図-9　Livermoreループのアンローリング効果

4.2　細粒度並列マシンの最適化技法

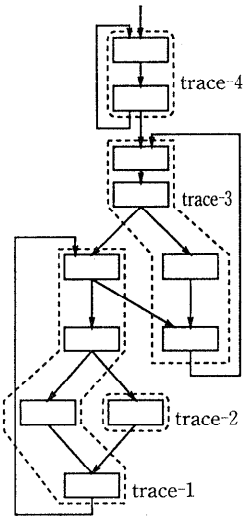
以下では，細粒度並列マシン用に新たに開発されたコンパイラ最適化技法の概略とその効果について述べる．

トレース・スケジューリング　リストスケジューリングは分岐命令が頻繁に出現するコードでは処理対象となる命令数自体が少ないため十分な性能向上が得られない欠点がある．並列度をさらに向上するには，より広範囲の命令群を対象に命令スケジューリングを行う（広域コードスケジューリング）ことが必要である．

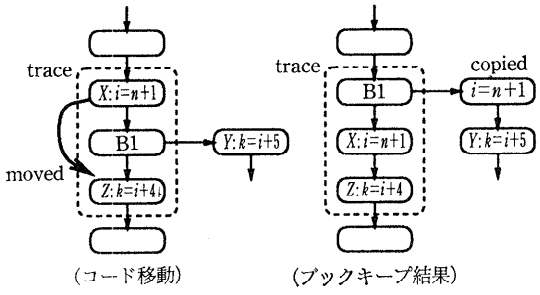
トレース・スケジューリング[19)~22)]は，プログラムの分岐予測を元に複数の（優先度の高い）基本ブロック群（トレース）を取り出して優先的にスケジューリングし，後処理でトレース部分と他の部分との意味的正しさを保証するコード補正を行うことで，意味的に正しく並列性の高い最適化プログラムを生成する処理である．

トレース選択は，各分岐命令の分岐予測を基に行う．予測実行回数最大の基本ブロックを核にその前後で実行回数の最大のブロックをトレースに加えていく．これを全プログラムに対して行い，予測実行回数の大きい順に優先度が付けられた複数のトレースが形成される（**図-10**（a））．このように得られた各トレースに対してリストスケジューリングを行う．トレース上では分岐命令を渡る命令移動も行われるので，トレース外の部分との整合をとるブックキープという後処理を行う．**図-10**（b），（c）にトレースからの分離，合流の場合のブックキープ処理を示す．いずれの場合も分岐点，合流点を越えて移動する命令（文 X）の作用がトレース外でも現れるように文 X のコピーを追加する．

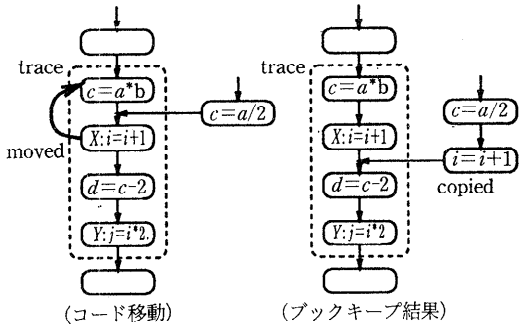
トレース・スケジューリングによる最適化効果を文献[20)]の評価結果より抜粋したものが**図-11**である．ここでは8個の演算ユニットをもつVLIW計算機ELIをターゲットに，各プログラムの理想性能比（無限個の演算ユニットをもつ場合の1CPUマシンに対する性能比）と実際にELI上で実行した場合の性能比を示している．ここであげ



(a) トレースの選択例



(b) トレースからの分離の例



(c) トレースへの合流の例

図-10 トレース・スケジューリング

たような科学技術計算では一般に分岐予測が容易であり，マシン規模に応じた性能比が得られているが，分岐予測がうまく働かないプログラムでは極端に性能が低下してしまうのがトレーススケジューリングの欠点と言える．またブックキープ時の命令コピーのためコードサイズが膨大になってしまうことも問題である．

パーコレーション・スケジューリング　トレース・スケジューリングの分岐予測の問題を回避するため考案されたのがパーコレーション・スケジューリングである[12),22),23)]．制御フローグラフ上のブロックの分岐，合流部分で(1)命令の上流ブロックへの移動，(2)分岐の上流ブロックへの移動，(3)同一命令の単一化，(4)空ブロックの削除，という4種類の基本操作を適用してプログラムの意味を保持しつつコードを上流ブロックに持ち上げていく (percolate) ことで最終的に上流ブロックの並列性を高める方法である．トレース・スケジューリングと異なり，実行時の分岐方向に一定の仮定を設定しない点で一般的といえるが，やはりコピー命令によりコードサイズの増大は避けられない．またループ部分の並列度に限界があるという欠点もある．

ソフトウェア・パイプライニング　ソフトウェア・パイプライニング[5)]はループ・アンローリングにおける展開段数を無限にしたときの実行状態を実現するようにループ構造を変換する手法と言える．ループの本体を独立な部分に分割し(**図-12**(1))，N 段のアンローリングを行ったとする．分割した各部分間に依存がなければ，隣接反復の処理をオーバラップして実行できる(**図-12**(2))．図-12(2)の '…' 部分では，A_i, B_{i-1}, C_{i-2} の同時実行が反復されているので，これをループ構造として構成したものが**図-12**(3)のループである．再構成されたループ本体を loop body，前後の処理を prologue, epilogue と呼ぶ．実際には，各 A_i, B_{i-1}, C_{i-2} は複数の命令からなる命令列であり，loop body はそれらを並列に実行するように命令スケジュールされている．loop body 内ではルー

プログラム	理想性能比	実性能比
行列乗算	25.5	7.4
FFT	48.3	6.9
LU 分解法	18.9	6.2
固有値分解	16.2	5.4
三重対角求解1	2.7	0.9
三重対角求解2	3.8	1.2
三重対角求解4	33.3	7.0
状態方程式	8.3	2.3

図-11 トレース・スケジューリングの性能評価 (文献19)より)

プの異なる反復が並列実行されるので，ループ内の依存関係をあらかじめ調べておくことが必要である．またレジスタ割当ても並列実行部分を考慮して必要ならば renaming などの処理を行う必要がある．

5. 並列性の性能に与える影響

スーパスケーラ計算機を想定し，その並列実行機構が実プログラム性能に与える影響を評価する．可変構造ハードウェアシミュレータ[24]上で最適化コンパイラ[18),24)]により生成したプログラムを実行する．本シミュレータは，所定の命令数を(in-order で) 同時発行するスーパスケーラプロセッサを仮定し，各ユニットのパイプライン処理を忠実にシミュレーションする．演算ユニット数，命令レイテンシなどを自由に設定して異なるアーキテクチャ下での実行性能を容易に評価できる．コンパイラは基本ブロックごとのリストスケジューリング，ループアンローリング，ソフトウェアパイプライニングを与えられたプロセサ構成を考慮して実行する．

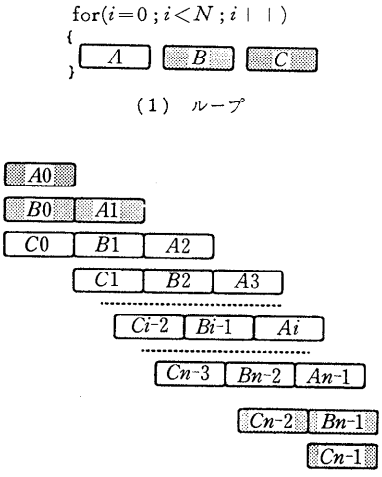


図-12 ソフトウェア・パイプライニングの変換過程

5.1 アーキテクチャの効果

スーパスケーラ・アーキテクチャによる並列化効果を調べるため，次の3種類のプロセサ構成についてシミュレータ上での実行速度を調べた．

2命令モデル：最大2命令同時発行，ALU 数 2，メモリポート数 1，FP 加算器 1，FP 乗算器 1

4命令モデル：最大4命令同時発行，ALU 数 4，メモリポート数 2，FP 加算器 2，FP 乗算器 2

8命令モデル：最大8命令同時発行，ALU 数 8，メモリポート数 4，FP 加算器 2，FP 乗算器 2

ここで，メモリポート数は1サイクルに同時実行できる load/store 命令数の上限を示す．FP 演算命令は，各プロセサがもつ演算器数の制限を越えない範囲で任意の組合せで同時使用できる．

各プログラムの実行結果を**図-13** に示す．各モデルの性能を1命令モデル（通常の RISC）に対する相対性能で示している．整数系プログラムの場合，2命令モデルと4命令モデルの間では一定の性能向上がみられるが8命令モデルにしても性能向上は得られず，1.15〜1.6 倍程度の高速化にとどまっている．これはプログラムの基本ブロッ

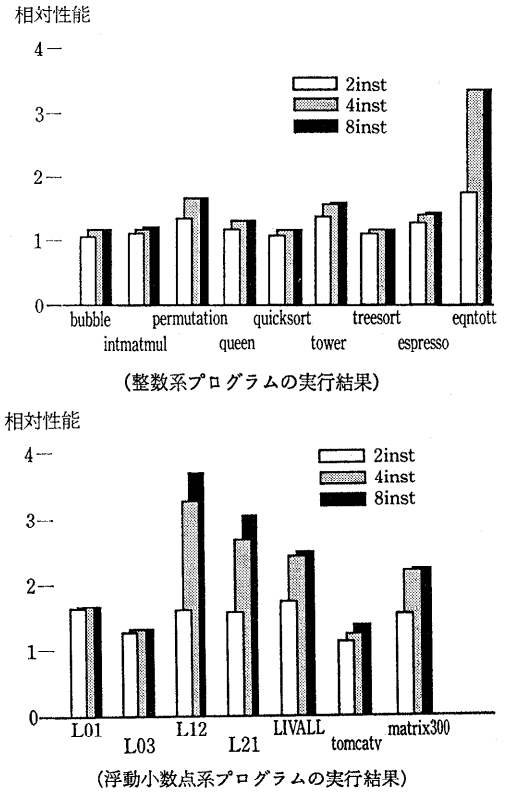


図-13 評価結果

ク長が短く，多くのハードウェアを十分に使用するコードが生成できないためと考えられる．したがって，整数系プログラムを局所命令スケジューリングのみで高速化する場合の発行命令数は4で十分であることが言える．浮動小数点系プログラムの場合，処理内容によって命令発行数を増やしても高速化できないもの (L 01, L 03) と命令発行数の効果が顕著に現れるもの (L 12, L 21) に大きく別れることが分かる．大規模なプログラムでは，このような異なる性質のループが複数存在して，それらの効果が加重された形で並列化効果が現れると考えられる (LIVALL, fSPEC の場合)．

整数系プログラムについて，整数ユニットの構成を変えた場合の実行結果を**図-14** に示す．命令発行数は4に固定し，ALU 数，メモリポート数を 1〜3 と変えた場合の 1 ALU-1 メモリポートモデルに対する相対性能を示す．この結果をみると，大部分のプログラムは 2 ALU，2 メモリポートで性能がほぼ飽和しており，それ以上のハードウェアをもつ効果は小さいと言える（メモリポートを2個もつ効果も，(3, 1) と (2, 2) の差をみると，プログラムによりばらつきがある)．また ALU を 2〜3 個もった最大性能でもせいぜい 30% 程度の性能向上しか得られないことも分かる．

浮動小数点プログラムの場合，演算命令，FP ロード命令などの演算レイテンシが大きく性能に影響する．この様子を Livermore ループについて調べたものが**図-15** である．図-15 (a) は FP 加算，乗算命令の実行サイクル数を，図-15 (b) は FP ロード命令のサイクル数を変化させた場合のループ全体の性能低下をプロットしたものである（太線は理論的最悪値を示す)．両者を比較すると十分な最適化（アンローリング，スケジューリング）を施した状態では，FP 演算命令のレイテンシ増大の影響は大きく，一方，ロード命令のレ

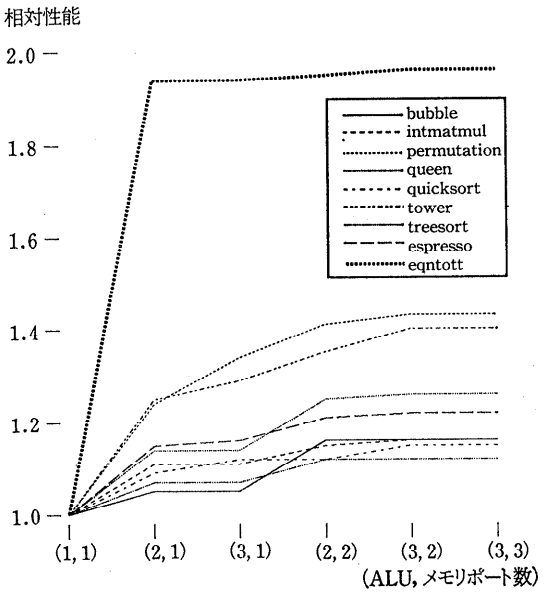
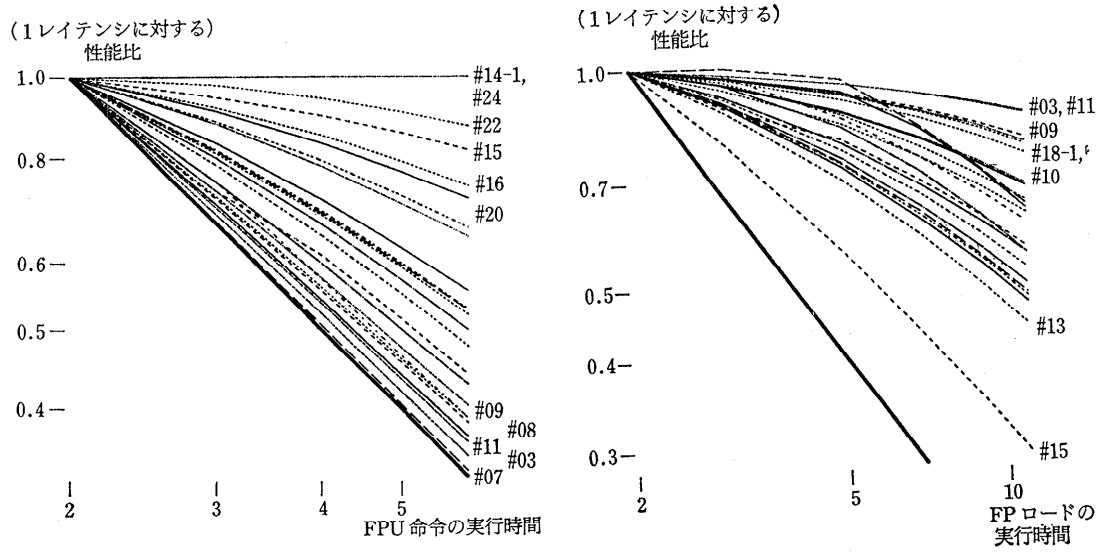


図-14 整数ユニット構成の効果



(a) FP 演算命令レイテンシの効果　　(b) FP ロード命令レイテンシの効果

図-15 FP 命令レイテンシの効果

イテンシはスケジューリングにより空きスロットを埋めることができるのでそれほどループ全体の性能には効かないと言える.

5.2 コンパイラ最適化の効果

コンパイラ最適化の実性能への効果を調べるため, 4命令発行, ALU2個, メモリポート2, FP演算器1個, FP演算レイテンシ2, FPロード命令レイテンシ2のマシンを対象に, 次の4種類の方法で生成したコードの実行性能を比較する.

No optimize: 通常のRISC用コンパイラ(スーパスカラ用の最適化を行わない)で生成したコード

Scheduled: 基本ブロック単位のリストスケジューリングのみを施したコード

Unrolled: ループに対してアンローリングを施したコード

Pipelined: ループに対して前処理でアンローリングを行い, さらにソフトウェアパイプライニングを施したコード

各プログラムの実行結果を**図-16**に示す. 各実行結果をNo optimize実行に対する相対性能で示す. Livermoreループではループ最適化の効果が非常に大きく, アンローリングで3～5倍, パイプライニングで3～7倍の高速化ができる. 逆にスケジューリングのみでは最大でも2倍程度しか高速化できない. またパイプライニングの効果はループによりばらついている. L01は加算, 乗算のバランスがパイプライニングにより改善できるため, L12は演算命令とロード・ストア命令の多重化の効果によりパイプライニングの効果が大きく現れている. 一方, それ以外のループ(L03, L07, L11)は, 命令間依存のためパイプライニングの効果は小さい. 2種類のSPECプログラムは, ループ部分以外の比率が大きいため, スケジューリングの効果が大きくなっている(4～5倍). ループ最適化はmatrix300では効果が大きいがtomcatvでは小さい.

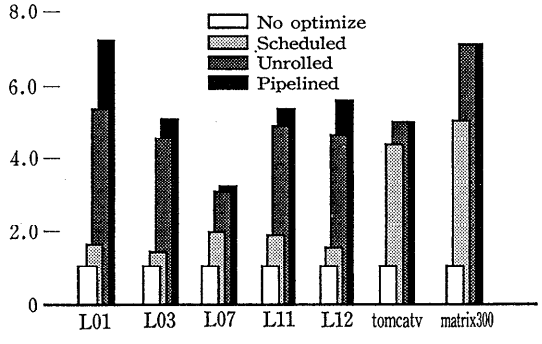


図-16 コンパイラ最適化の効果

以上, ループ部分が支配的なプログラムと比較的大規模なプログラムの双方についてコンパイル最適化効果をみたが, いずれの場合も通常のRISC用コンパイラの生成コードをそのまま実行する場合に対し3～7倍の高速化が得られ, 最適化の重要性を示している. また, プログラムの規模, 性質によって有効な最適化手法が変化するので, 命令スケジューリング, ループ最適化の両方を順次適用することが広範囲のプログラムで高性能を達成するためには必要である.

6. おわりに

いままで述べてきたように, マイクロプロセサの高速化のために, 細粒度並列処理が取り入れられ始めている. しかしその並列度は特殊な例をのぞき, それほど大きくならないことも明らかになってきている. しかも, 並列に動かすための機構はその並列度が上がるに従って大きくなり, クロック周波数の向上を抑える方向に働く. このことから現実的な並列度としては4程度が限度であり, それ以上上げても意味がない. このハードウェアの複雑さは現在, あるいはここ2～3年の半導体技術に見合った値であるが, その後の半導体技術の伸びには対応できない. その解決策もいくつか提案されており, 粗粒度並列を取り入れる, あるいは超並列的な考え方を導入するなどの方向が考えられるが, いずれも一般的な処理を高速化するというよりは, ある特定の処理の高速化に効果があるものである. 今後は, 目的とする処理の内容に適した方式が取り入れられていき, プロセサの多様化が進むものと思われる.

参考文献

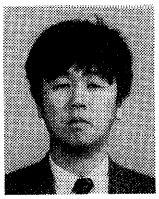
1) Radin, G.: The 801 Minicomputer, Proc. of the symposium on Architectual Support for Programing Languages and Operating Systems, pp. 39-47 (1982).
2) Patterson, D. A. and Sequin, C. H.: A VLSI RISC, Computer, 15 (9), p. 8 (1982).
3) 小柳 滋他: マイクロプログラム制御計算機QA-1のハードウエア構成, 電子通信学会論文誌 Trans. IECE '78/1, Vol. 61-D, No. 1 (1978. 1).
4) Fisher, J. A.: The VLIW Machine: A Multi-

processor for Compiling Scientific Code, Computer, 17 (7), p. 45 (1984).
5) Lam, M.: Software Pipelining: An Effective Scheduling Technique for VLIW Machines, ACM SIGPLAN '88 Conference on Programming Language Design and Implementation, pp. 318-328 (1988).
6) MIPS R 4000 User's Manual, MIPS Corp.
7) Alpha Architecture Handbook, Digital Equipment Corporation.
8) Grohoski, G. F. et al.: Branch and Fixed-Point Instruction Execution Units, IBM RISC System/6000 Technology, IBM Corporation (1990).
9) Minagawa, K., Saito, M. and Aikawa, T.: Pre-Decoding Mechanism for Superscalar Architecture, Proc. of IEEE Pacific Rim Conferece on Communications, Computers and Signal Processing, pp. 21-24 (May 1991).
10) Bray, B. K. and Flynn, M. J.: Strategies for Branch Target Buffers, Technical Report No. CSL-TR-91-480, Stanford University (June 1991).
11) 中島雅逸他：スーパースカラ・マイクロプロセッサ OHMEGA における動的ハザード解消機構と高速化手法，情報処理学会計算機アーキテクチャ研究報告 91-ARC-89-5 (1991. 7).
12) 村上和彰：スーパースカラ・プロセッサの性能を最大限に引き出すコンパイラ技術，日経エレクトロニクス No. 521, pp. 165-185 (1991. 3).
13) Gibbons, P. B. and Muchnick, S. S.: Efficient Instruction Scheduling for a Pipelined Architecture, Proc. of SIGPLAN '86 Symp. on Compiler Construction, pp. 11-16 (June 1986).
14) Hennecy, J. and Gross, T.: Postpass Code Optimization of Pipelined Constraints, ACM Trans. Programming Language and System, Vol. 5, No. 3, pp. 422-448 (July 1988).
15) Krishnamurthy, S. M.: A Brief Survey of Papers on Scheduling for Pipelined Processors ACM SIGPLAN notices, Vol. 25, No. 7, pp. 318-328 (July 1990).
16) Bernstein, D. and Gertner, I.: Scheduling Expressions on a Pipelined Processor with a Maximal Delay of One Cycle, ACM Trans. Programming Language and System, Vol. 11, No. 1, pp. 57-66 (Jan. 1989).
17) Coffman, E. G., Jr. (Ed): Computer and Job-Shop Scheduling Theory, John Wiley & Sons (1976).
18) 白川健治他：スーパースカラプロセッサにおけるループ並列化の検討，情報処理学会計算機アーキテクチャ研究報告 91-ARC-91-3 (1991. 11).
19) Fisher, J. A.: Trace Scheduling: A Technique for Global Microcode Compaction, IEEE Trans. Comput., Vol. C-30, No. 7, pp. 478-490 (July 1981).
20) Ellis, J. R.: Bulldog: A Compiler for VLIW Architecture, MIT Press (1986).
21) Touzeau, R. F.: A Fortran Compiler for the FPS-164 Scientific Computer, Proc. SIGPLAN '84 Symp. on Compiler Construction, pp. 48-57 (June 1984).
22) 中谷登志男：VLIW 計算機のためのコンパイラ技術，情報処理，Vol. 31, No. 6, pp. 763-772 (June 1990).
23) Aiken, A. and Nicolau, A.: A Development Environment for Horizontal Microcode, IEEE Trans. Softw. Eng., Vol. 14, No. 5, pp. 584-594 (May 1988).
24) 斎藤光男他：スーパスケーラプロセサの構成と，その性能評価，情報処理学会計算機アーキテクチャ研究報告 92-ARC-94-1 (1992. 6).

(平成 5 年 10 月 29 日受付)

斎藤 光男（正会員）

1950 年生．1974 年東京大学工学系研究科修士課程修了．同年(株)東芝入社．現在東芝情報通信システム技術研究所主幹．この間，計算機の日本語化，ヒューマンインタフェースの研究などを経て，マイクロプロセッサおよびそれを使った計算機システムの開発に従事．電子情報通信学会，人工知能学会各会員．



井上 淳（正会員）

1962 年生．1985 年東京大学工学部計数工学科卒業．同年(株)東芝入社．以来研究開発センター情報・通信システム研究所において，マイクロプロセッサ，コンパイラ，性能評価技術の研究開発に従事．ACM 会員．