九州大学学術情報リポジトリ
Kyushu University Institutional Repository

RISCを用いた実時間システムの処理能力の評価手法に関する研究

杉村, 康

https://doi.org/10.11501/3159132

出版情報：九州大学, 1999, 博士（工学）, 論文博士
バージョン：
権利関係：

第４章　二次キャッシュの競合ミスの回避

上記の第３章では，R4000 の処理能力を実RTS のトレースデータに基づいて評価した結果，その処理能力がメモリの割当て状況の影響を受けることを，定量的に示した．その影響を回避するためには，メモリの割当て状況を最適化する手法が必要である．

本章では，4.1 で，新最適化手法の概要とその前提条件について説明し，4.2 で，最適化手法の詳細について提案し，4.3 で，第２章のシミュレーションの応用によって上記最適化手法を評価する手法と，その評価手法による評価結果について述べる．

４．１　前提条件と新手法の概要

上記研究の位置付けで述べたように，評価対象のR4000[1][2][3]の二次キャッシュは巨大であり，かつ，ダイレクトマッピングであり，その場合，メモリの割当ての最適化において，競合ミス[4] を最低にすることが，必要である．また，処理能力を優先した実RTS では，TSS とは違って，以下の特徴がある．

（A）オンラインのサービス中に使用されるプログラムとデータは，メモリ内に常駐する．

（B）オンラインのサービス中に動作するプログラムは，定型的なトランザクション処理[5] のみであり，それらのロジックは，システム全体のロジックの一部分に限定されている．

（C）オンラインのサービス中に使用されるプログラムの論理アドレスは固定であり，あらかじめシステム生成時までにアドレス解決済となっており，オンラインのサービス中に変更されることはない．

それらを前提とした場合，下記（1)～(3）の手順により，オンラインのサービス中の二次キャッシュのミスの発生を最小にすることが，可能である．

（1）実RTS のサービス中のトレースデータから得た第２章でのRISCアクセス状態データより，論理ページや物理ページの情報だけでなく，二次キャッシュの行の使用の情報を含んだ「メモリ割当ての最適化用の情報」を，あらかじめ収集する．

（2）それらの情報をオフラインで計算することにより，最適な論理ページと物理ページの関係を求める（最適化を行う）．

（3）それらの論理ページと物理ページの最適な関係を，オンラインのサービスを開始する直前（オンラインのサービスの初期設定完了時）に指定することにより，OSの新規機能が，メモリの割当てを最適状態に一括して変更する．

上記の最適化を，次の二つの最適化手法により，実現する．

（A）物理ページと論理ページの関係が固定であるエリア（以下，固定エリアと略記）の最適化（以下，最適化Ｘと略記)：　固定エリアは，TLB でマッピング*1されないエリアであり，R4000 の場合，カーネルエリアである．すべてのエリアがTLB でマッピング可能なCPU も存在する[6] が，その場合は，下記の(B）の最適化のみでよい．当最適化は，固定エリアの論理空間のすべてを，二次キャッシュの大きさに等しい「群」という単位に分割する．そして，「二次キャッシュのミスが発生している行を含むページの一覧表」と「参照・更新（以下，参照と略記）されているページの一覧表」と「群の一覧表」を，RISCアクセス状態データを元に作成し，ミスが発生しているページを含む群の間に，できるだけ少ない「数ページのダミーページ」を試行的に挿入し，ミスが目標値以下となる挿入ダミーページ数と挿入位置を求めることにより，最適化を行う．その後，それらの最適化結果に基づき，アドレスを解決するとき（システム生成時）に，群の間に数ページのダミーページを挿入することにより，最適化結果を反映する．それらの詳細については，4.2.1 で述べる．

マッピング*1：複数の情報の関係を定めること．　TLBの場合は，論理ページと物理ページの関係を定めること．

（B）物理ページと論理ページの関係が変更可能であるエリア（浮動エリア）の最適化（以下，最適化Lと略記）: 浮動エリアは，TLB でマッピングされるエリアである．固定エリアの論理ページと物理ページの関係は変更できない｛上記(A)により最適化された結果を含む｝ことを前提として，浮動エリアの論理ページと物理ページの関係を最適化する．

その最適化では，

（a）二次キャッシュ内の相対ページ番号（以下，二次キャッシュ相対ページ番号と略記）毎に，ミスを起こしている論理ページの一覧表を作成して，それらのページを最適化対象と定義し，

（b）該一覧表内に固定エリアの論理ページがあれば，それらを最適化対象より削除し，(b')固定エリアの論理ページを含まない場合は，ミスを起こしている行の数が一番多い論理ページを，最適化対象から外し，

（c）最適化対象のそれぞれの論理ページ対して，二次キャッシュ内相対物理ページ番号を，順次，二次キャッシュのページの数の分，仮に割当てて，ミスを起こす論理ページの数が一番少なくなる二次キャッシュ相対ページ番号を探索し，その相対ページ番号を該論理ページに割当てて，該論理ページを最適化対象より外す．上記の探索の際には，「最適化対象の論理ページ」を除いたすべての論理ページの行の参照の状態を反映する．

（d）それらの最適化の結果をファイルに出力し，

（e）その出力を，新規に作成したOSのシステムコールにより，システムのサービスの初期設定の直後にOSに通知し，

（f）OSは，それらを満足するように，物理アドレスの再配置を，一括して行う．

また上記(A)の最適化Xと合わせて，上記の最適化Lの結果の二次キャッシュ相対ページ番号に，上記のOSの物理アドレスの再配置の実施と同様に，最適化実施プログラムが物理ページ番号を割当て，その結果をファイルに出力し，そのファイルを入力してシミュレーションを再度行うことにより，最適化後のシステムの処理能力を算出する．それらの詳細については，4.2.2 で述べる．

また，第2章で述べた実RTS のRISCアクセス状態データを使用して，上

表4．1　当評価でのR4000 での主な前提条件

No	分類	前提条件
1	ｷｬｯｼｭ関連	WB方式，一次ｷｬｯｼｭと二次ｷｬｯｼｭ 有，一次ｷｬｯｼｭは命令/ﾃﾞｰﾀ独立，二次ｷｬｯｼｭ は命令とﾃﾞｰﾀ で共用，一次ｷｬｯｼｭ/二次 ｷｬｯｼｭのﾊﾞｽ幅=16ﾊﾞｲﾄ，一次命令・ﾃﾞｰﾀｷｬｯｼｭ各8KB，二次ｷｬｯｼｭ1MB，一次ｷｬｯｼｭ 行長16ﾊﾞｲﾄ，二次ｷｬｯｼｭ 行長32ﾊﾞｲﾄ，一次・二次ｷｬｯｼｭ のﾘﾌｨﾙは行単位，ﾒﾓﾘ は内部にﾘｰﾄﾞﾊﾞｯﾌｧとﾗｲﾄﾊﾞｯﾌｧ を各1 個所有，CPU/ﾒﾓﾘﾊﾞｯﾌｧ間の転送時間はﾊﾞｽ幅とﾃﾞｰﾀ 長に依存，一次ｷｬｯｼｭ 論理ｱﾄﾞﾚｽﾀﾞｲﾚｸﾄﾏｯﾋﾟﾝｸﾞ，二次ｷｬｯｼｭ 物理ｱﾄﾞﾚｽﾀﾞｲﾚｸﾄﾏｯﾋﾟﾝｸﾞ，32 ﾋﾞｯﾄｱﾄﾞﾚｽ．
2	その他	CPUのｸﾛｯｸの周波数は75MHz(ｽﾃｰｼﾞの周波数は，150MHz)，ﾒﾓﾘ のﾌﾞﾛｯｸｱｸｾｽ時間は100nsのｵｰﾀﾞｰ，二次ｷｬｯｼｭ のﾌﾞﾛｯｸｱｸｾｽ時間は10nsのｵｰﾀﾞｰの各1点に固定．

記の最適化を行わなかった場合と，行った場合の処理能力を比較することにより，上記の最適化の効果を評価し，それらの最適化による処理能力の改善効果が，およそ30～100[%]であること，を明らかにする．

それらの最適化結果を評価するシミューレータは，第2章の2.3.2 のシミューレータに，物理アドレスをすり替えるロジックを追加しただけである．従って，それらの最適化結果の評価精度は，第3章の精度検証の結果より，非常に高いと考えてよい．

なお，後述の評価結果は，R4000 を用いた小型RTS を前提としており，そこでの上記(1) ～(3) 以外の前提条件は，表4.1 に示すとおりである．

また，上記最適化においては，TLB を使用するエリアの論理アドレスを，一切変更しないので，TLB ミスの影響は，評価より割愛する．

4．2　最適化手法と評価手法

4．2．1　固定エリアの最適化（最適化X）

【原理】

固定エリアにおいては，物理ページと論理ページの関係は，ハードウェアによって固定されている．従って，固定エリア内の複数のページの相互間の競合ミスを減少させるためには，論理ページと物理ページの両者を移動するしか方法はない．

一方，固定エリアを有するシステムでは，一般に，固定エリアはカーネルエリアであり，そこには，カーネルの多数の機能のためのプログラムやデータが存在し，それらの大きさの合計は4～16MBに達する．それらのうち，オンラインのサービス中に走行する部分は，一般に，ほんの一部である．また，カーネルエリアは，多数の機能を含むが故に，各プログラム間にパッチエリアと称する補修用のエリアや，将来の拡張用のダミーエリアを多数含有する．従って，あるまとまった単位で，論理空間を移動させることは，一般に，容易である．

当手法では，その移動の単位を「群」とよび，群を移動することにより，固定エリア内のキャッシュの競合ミスを回避する．

例えば，今，話を簡単にするために，図4.1のように，キャッシュがｇｚ行からなり，固定エリアのメモリが，ｇａ行に相当する大きさからなり，1ページは1行であり，それらのうち，オンラインのサービス中に，同図中の①，②，③の行のみが使用されていると仮定する．そこでは，巨大な二次キャッシュの中の行は，行1と行ｇｚの二つしか使用されず，残りはすべて未使用である．しかし，メモリ上の行1と行ｇｚ＋1は，二次キャッシュ上で，競合ミスを引き起こす．

当手法では，同図(b)の網かけ部分のように，ダミーページを，1ページ挿入してみる．そうすると同図(a)の③は，同図(b)の③’の位置になり，二次キャッシュの行1ではなく行2を使用するようになる．それによって，①と②と③は，すべて，競合ミスを発生しなくなり，最適化は完了

する．もし，1ページの挿入で，競合ミスが解消しない場合は，順次挿入ページを増やして，最適な挿入ページ数を探す必要がある．

当手法では，群の大きさの最小単位は，二次キャッシュの大きさである．それは，競合ミスを群の移動により回避するために，必要である．なぜならば，図4.1において，メモリ内の行ｇｃ1と行ｇｃ2が競合ミスを起こしている場合を考え，それらの「二次キャッシュ内の行の番号」を，それぞれｇｃ1ｚ，ｇｃ2ｚとすると，下式(4.1)～(4.3)が成立ち，ｇｃ1番目とｇｃ2番目の行の競合ミスをページの移動によって回避するために

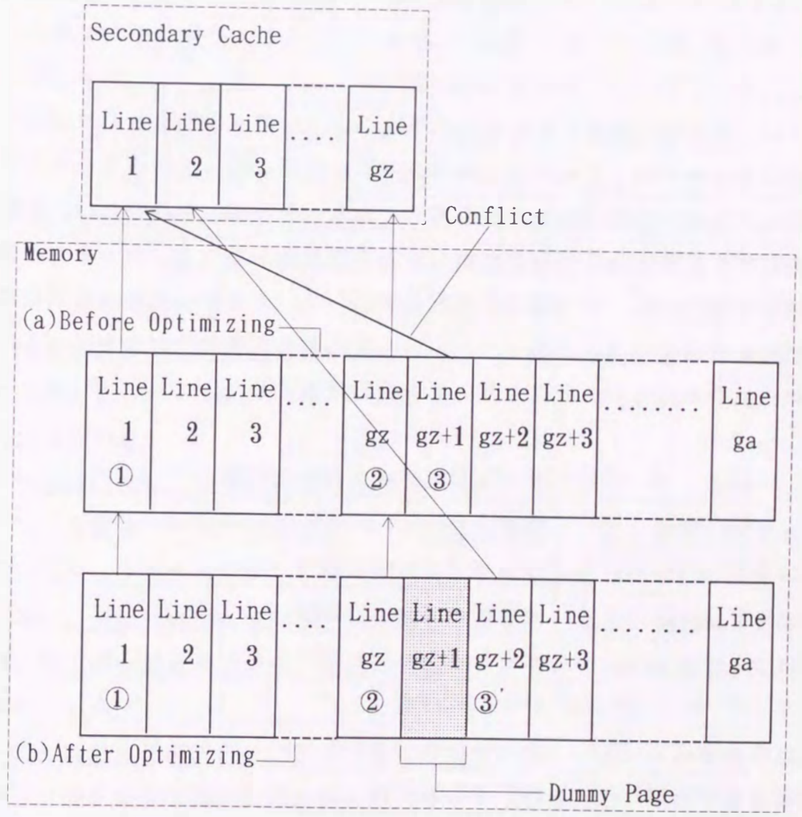


図４．１　　最適化Ｘの概念図

は，少なくともｇｚ行以上離れたものを移動させないと効果はないからである．

$$gc1z = gc1 \ \% \ gz \quad (4.1)$$

$$gc2z = gc2 \ \% \ gz \quad (4.2)$$

$$gc1z = gc2z \quad (4.3)$$

（注：％は，左辺を右辺で割った剰余を求めるための演算子）

言い替えれば，ｇｚ行以上離れていないものを移動することは，二次キャッシュの参照に新たな無駄な擾乱を発生させるだけで，最適化にメリットを与えない．そのために，群の大きさの最小単位を二次キャッシュの大きさと定める．なお，群の大きさは，ロードモジュール等より十分大きいことが必要であるが，ここでは，二次キャッシュの大きさは巨大であることを前提としており，その点も問題はない．

一方，最適化の方式としては，当手法のように群を移動するのではなく，実在するロードモジュールをページの境界に区切って，区切られたものを自由に入替えて最適化する方法も考えられる．しかし，その場合は，最適化の打切り基準の設定が非常に難しいと予想されることと，カーネルの論理空間の割当ては，その規模の増大を予想して，あらかじめ製造元単位に割当てて管理する場合が多いので，自由に入換える手法はなじみにくく，受け入れられにくいことにより，その検討は割愛した．

表４．２　RISCアクセス状態データの主な内容

No	項目	説明
1	論理ｱﾄﾞﾚｽ	命令/ﾃﾞｰﾀ ﾌｪｯﾁ の論理ｱﾄﾞﾚｽ
2	物理ｱﾄﾞﾚｽ	命令/ﾃﾞｰﾀ ﾌｪｯﾁ の物理ｱﾄﾞﾚｽ
3	ﾌｪｯﾁ種別	命令ﾌｪｯﾁかﾃﾞｰﾀ ﾌｪｯﾁかを示す．
4	命令ﾆｰﾓﾆｯｸ名	命令ﾆｰﾓﾆｯｸ名
5	ﾘｰﾄﾞﾗｲﾄ種別	ﾃﾞｰﾀ ﾘｰﾄﾞ ﾌｪｯﾁかﾃﾞｰﾀ ﾗｲﾄ ﾌｪｯﾁかを示す．
6	ﾌｪｯﾁ長	ﾃﾞｰﾀ のﾌｪｯﾁの長
7	ﾚｼﾞｽﾀ情報	ﾚｼﾞｽﾀ番号毎の参照の有無を示す．
8	ﾀｽｸID	R4000 のｱﾄﾞﾚｽ空間ID（ASID）に対応．

表４．３　各論理ページの情報

No	項目	説明
1	論理ﾍﾟｰｼﾞｱﾄﾞﾚｽ	参照ｱﾄﾞﾚｽの下12ﾋﾞｯﾄ を零にしたもの
2	ﾀｽｸ ID	ﾕｰｻﾞ 空間の場合のみ
3	参照行数	当ﾍﾟｰｼﾞ で参照されている相異なる行の数
4	旧物理ﾍﾟｰｼﾞｱﾄﾞﾚｽ	ｱｸｾｽ状態ﾃﾞｰﾀ 内で割当てられている物理ﾍﾟｰｼﾞｱﾄﾞﾚｽ(ｱﾄﾞﾚｽの下12ﾋﾞｯﾄ を零にしたもの)
5-132	行参照有無情報	各行毎の参照の有無の表示，参照無＝０，(No － 5)が行の番号．
133	群番号	当ﾍﾟｰｼﾞ が属する群の番号
134	競合ﾐｽ状態	行の競合ﾐｽの有無の表示
135	ﾍﾟｰｼﾞ 属性	最適化不要，最適化中，最適化済の表示
136	相対ﾍﾟｰｼﾞ 番号	最適化後の二次ｷｬｯｼｭ 内相対ﾍﾟｰｼﾞ 番号
137	新物理ﾍﾟｰｼﾞｱﾄﾞﾚｽ	最適化後にｼﾐｭﾚｰﾀが割当てた物理ﾍﾟｰｼﾞｱﾄﾞﾚｽ

（注）参照されているﾍﾟｰｼﾞ に対してのみ作成される．

当手法の最適化の打切りのための手法は，比較的単純であり，下記(4)で詳説する．

【具体的手順】

当最適化Ｘは，以下の手順で行う．

（1）参照されている各論理ページの情報の作成：　表4.2 のRISCアクセス状態データをすべて入力して，固定エリア内で参照されている論理ページ毎に，表4.3 のNo１～No132 の情報を作成する．なお，固定エリアの論理空間の最小値と最大値は，最適化実施プログラムの内部情報として用意されている．

（2）群の情報の作成：固定エリアの空間を，表4.4 のいずれかの群に分類し，表4.5 の群毎の情報を作成し，その後，群番号を，対応する各論理ページの情報である表4.3 のNo133 に設定する．

（3）全ページの競合数の特定：

（A）表4.3 の各論理ページの情報をすべて使用して，二次キャッシュの中の各ページ毎の競合ミスの情報である表4.6 の中のNo1 ～No280 を，まず設定する．

（B）その後，該表のNo1 のページ重複数が２以上の二次キャッシュ内のページについて，行毎の参照回数（No3-130)が２以上のものの有無をチェックし，

（a）あれば，競合ミスが発生しているので，該行を参照している表4.3 のページを，表4.6 のNo1 のページ重複数とNo131 ～No(130+ ページ重複数）にある添字一覧より探し出し，該行を参照している（競合している）もののみの添字を，表4.6 のNo282 以降に設定し，No281 のページ競合数をカウントアップする．

（C）二次キャッシュ内の全ページについて，ページ競合数の設定が完了したら，各ページ競合数の合計を，全ページの競合数（以下，全ページ競合数と略記）とする．

（4）最適化の実施：全ページ競合数(W) が零でない場合，最適化が必要である．最適化は，二次キャッシュ内の各ページのうち，表4.6 のNo281 のページ競合数が零でないページに対して行う．該ページが複数ある場合は，その中の一つに対してのみ最適化を行い，その結果必要であれば｛下記(D) でW'<=Q なら｝，再度，二次キャッシュ内の別のページに対して，最適化を行うことを繰返す．

（A）まず，二次キャッシュ内のあるページの「ページ競合数（表4.6 のNo281)」がY(>=2)であるとすると，最適化後の目標競合数を

Q = W - Y

とする．

（B）次に，表4.6 のNo282 以降の情報より，競合を起こしている表4.3 の論理ページの情報を得，その群番号より，その論理ページが属する群のクラスを表4.5 より得，競合を起こしている複数の論理ページの群のクラスのパターンが，表4.7 のどのパターンになるかを判定する．

（C）表4.3 と表4.5 と表4.6 のコピーを作成し，

（a）上記(B) の判定結果に従って，直前または直後にダミーページを挿

表４．４　群の種類

No	名称	説明
1	先頭使用群	最も小さい群番号*1を持つ使用群*2(ﾊｰﾄﾞｳｪｱ 的にｱﾄﾞﾚｽを移動できないﾍﾞｸﾀｰﾃｰﾌﾞﾙ等を含むので，最適化では移動不可)．
2	前半使用群	使用群*2のうち，犠牲未使用群より小さい群番号を持ち，かつ先頭使用群でない群（最適化では，ｱﾄﾞﾚｽを大きい方に移動可能)．
3	前半未使用群	未使用群*3のうち，犠牲未使用群より小さい群番号を持つ未使用群（最適化ではｱﾄﾞﾚｽを大きい方に移動可能)．
4	犠牲未使用群	最も大きい未使用群*3［最適化によって，その大きさが変化する（小さくなる）唯一の群]．
5	後半使用群	使用群*2のうち，犠牲未使用群より大きい群番号を持つ群（最適化では，ｱﾄﾞﾚｽを小さい方に移動可能)．
6	後半未使用群	未使用群*3のうち，犠牲未使用群より大きい群番号を持つ未使用群（最適化ではｱﾄﾞﾚｽを小さい方に移動可能)．

（注）群番号*1：１から始まる群のｼｰｹﾝｼｬﾙ 番号．　論理ｱﾄﾞﾚｽ（符号なし整数）が小さい群が，小さい群番号を持つ．

使用群*2：二次ｷｬｯｼｭ の大きさをZ ﾍﾟｰｼﾞ としたとき，連続するZ ﾍﾟｰｼﾞ 内に，一つ以上の参照されているﾍﾟｰｼﾞ がある場合，その連続するZ ﾍﾟｰｼﾞ を一つの使用群とよぶ．

未使用群*3：二次ｷｬｯｼｭ の大きさをZ ﾍﾟｰｼﾞ としたとき，連続するZ ×m (m≧1)ﾍﾟｰｼﾞ 内に一つも参照されているﾍﾟｰｼﾞ がなく，その直後の群が使用群である（または固定ｴﾘｱ でない）場合，その連続するZ ×m ﾍﾟｰｼﾞ を，一つの未使用群とよぶ．

表４．５　群毎の情報

No	名称	説明
1	群番号	「当表を構造体とした構造体の配列の添字」+1が群番号である．当情報はどこにも格納されていない．最大値が内部情報に保持されているだけ．
2	群ｸﾗｽ	当群のｸﾗｽ（1〜6;表4.4 のNoに対応）．
3	旧先頭ｱﾄﾞﾚｽ	最適化前の群の先頭論理ﾍﾟｰｼﾞ ｱﾄﾞﾚｽ．
4	現先頭ｱﾄﾞﾚｽ	最適化後の群の先頭論理ﾍﾟｰｼﾞ ｱﾄﾞﾚｽ．
5	直前挿入ﾍﾟｰｼﾞ 数	最適化によって直前に挿入されたﾍﾟｰｼﾞ 数．群ｸﾗｽ=前半使用群にのみ設定．
6	直後挿入ﾍﾟｰｼﾞ 数	最適化によって直後に挿入されたﾍﾟｰｼﾞ 数．群ｸﾗｽ=後半使用群にのみ設定．
7	旧最終ｱﾄﾞﾚｽ	最適化前の群の最終論理ﾍﾟｰｼﾞ ｱﾄﾞﾚｽ．群ｸﾗｽ=犠牲未使用群のみ設定．
8	現最終ｱﾄﾞﾚｽ	最適化後の群の最終論理ﾍﾟｰｼﾞ ｱﾄﾞﾚｽ．群ｸﾗｽ=犠牲未使用群のみ設定．

入して群毎の情報を更新し，

（b）群の直前または直後に挿入したダミーページの分，アドレスを後ろまたは前にずらしたものを，表4.3 に反映し，

（c）それらの結果を使用して，表4.6 の二次キャッシュの各ページの競合の情報を，再度，作成し直して，

（d）上記(3）と同様の処理を行って，新しい全ページ競合数(W')を求め，

（e）その値が，目標競合数(Q）以下となるかどうかをチェックする．

なお，表4.4 の群の種類では，No2 の前半使用群は，アドレスを大きい方に移動可能，No5 の後半使用群は，アドレスを小さい方に移動可能としている．これは，表4.4 の先頭使用群が，通常，カーネル空間の先頭であり，アドレスを小さい方に移動は不可であることと，メモリの割当ての設

計において，カーネルが使用する論理空間の総量が空間管理表というドキュメントによってあらかじめ規定されていることがある，ことに基づいている．従って，上記のような規定がないシステムにおいては，上記の後半使用群で，アドレスを大きい方に移動させても支障はないし，その場合でも，後述の最適化中止の判定手法の変更は，不要である．

（D）W' <= Q であれば，最適化が成功したので，上記(C）で作成した表4.3 と表4.5 と表4.6 のコピーを，コピー元に複写し，W = W'として，W = 0 ならば最適化を終了し，W > 0 なら，上記（4)に戻って，二次キャッシュ内の別のページの最適化を行う．

（E）W' > Qならば，最適化は失敗したので，挿入するダミーページ数を変えて，上記(C）以降を繰返す．但し，各挿入ページ数が，それぞれ二次キャッシュのページ数(Z）に達した場合は，最適化を中止する．

表４．６　二次キャッシュの中の各ページの競合の情報

No	項目	説明
1	ﾍﾟｰｼﾞ 重複数	二次ｷｬｯｼｭ 内の当ﾍﾟｰｼﾞ を参照しているﾍﾟｰｼﾞ（表4.3)の数（行の競合を起こしていないﾍﾟｰｼﾞ を含む)．
2	使用行数	当ﾍﾟｰｼﾞ 内で参照されている相異なる行の数（最大128)．
3-130	行毎の参照回数	(No - 3)が行の番号．当行を参照しているﾍﾟｰｼﾞ（表4.3)の数(2以上なら競合ｽ有)．
131 - 280	参照ﾍﾟｰｼﾞ の添字一覧	当ﾍﾟｰｼﾞ を参照しているﾍﾟｰｼﾞ の表4.3 の構造体の配列の添字の一覧表（No1 のﾍﾟｰｼﾞ 重複数の分だけ格納される)．
281	ﾍﾟｰｼﾞ 競合数	当ﾍﾟｰｼﾞ で，行の競合を起こしているﾍﾟｰｼﾞ（表4.3)の数．
282 - 430	競合ﾍﾟｰｼﾞ の添字一覧	当ﾍﾟｰｼﾞ で，行の競合を起こしているﾍﾟｰｼﾞ（表4.3)の構造体の配列の添字の一覧表(No281のﾍﾟｰｼﾞ 競合数の分だけ格納される)．

上記(A)～(E)の最適化は，例えば，W=3,Y=3 で，競合ミスを起こしている論理ページを含む群（以下，競合群と略記）のパターンが表4.7 のNo1であり，競合群１，２，３の先頭アドレスが，以下の(a),(b),(c) である場合，次のように進行する．

（a）　競合群１の先頭アドレス= 0x00000000

（b）　競合群２の先頭アドレス= 0x00200000

（c）　競合群３の先頭アドレス= 0x00300000

まず(a) を，表4.7 のNo1 の最適化の方法に従って，最適化対象から外す．

このとき，Q = 3(=W) - 3(=Y) = 0 である．

【１回目の最適化】

競合群２と競合群３の直前に以下のように，それぞれ，1 ページを挿入する．

（ア）競合群２の直前に1 ページを挿入することにより，競合群２と３の先頭アドレスは，以下になる．

（b'）　競合群２の先頭アドレス= 0x00201000

（c'）　競合群３の先頭アドレス= 0x00301000

４．７　競合群のパターンと最適化方法の関係

No	ﾊﾟﾀｰﾝ	最適化の方法
1	先頭使用群または前半使用群だけの場合	群番号が最小のﾍﾟｰｼﾞ を最適化対象外とし，残りの各ﾍﾟｰｼﾞ の群の直前に，それぞれ，数ﾍﾟｰｼﾞ を埋込む．
2	後半使用群だけの場合	群番号が最大のﾍﾟｰｼﾞ を最適化対象外とし，残りの各ﾍﾟｰｼﾞ の群の直後に，それぞれ，数ﾍﾟｰｼﾞ を埋込む．
3	上記以外の場合	群番号が最小のﾍﾟｰｼﾞ を最適化対象外とし，残りの各ﾍﾟｰｼﾞ の内，前半使用群はその直前，後半使用群はその直後に，それぞれ，数ﾍﾟｰｼﾞ を埋込む．

（イ）競合群３の直前に1 ページを挿入することにより，競合群２と３の先頭アドレスは以下になる．

（b'）　競合群２の先頭アドレス= 0x00201000

（c''）競合群３の先頭アドレス= 0x00302000

この状態で，表4.3 と表4.5 と表4.6 を作成し直して，W'を求める．

【２回目の最適化】

W' > Q(=0)である（最適化失敗）なら，次に，競合群２の直前に1 ページ，競合群３の直前に2 ページを，以下のように挿入する．

（ウ）競合群２の直前に1 ページを挿入することにより，競合群２と３の先頭アドレスは以下になる．

（b'）　競合群２の先頭アドレス= 0x00201000

（c'）　競合群３の先頭アドレス= 0x00301000

（エ）競合群３の直前に２ページを挿入することにより，競合群２と３の先頭アドレスは以下になる．

（b'）　競合群２の先頭アドレス= 0x00201000

（c''）競合群３の先頭アドレス= 0x00303000

この状態で，W' を求める．

【３回目以降の最適化】

W' > Qであるなら，競合群３の直前に挿入するページをZ-1 まで順次増していき，それでも失敗するなら，競合群２の直前の挿入するページを２～(Z-1) まで変更し，それぞれについて，競合群３の直前に挿入するページを1 ～(Z-1) まで変化させる．

競合群２，３のすべてについて，それぞれ1 ～(Z-1) に変化させたにもかかわらず，W'<=Q を満足できない場合は，最適化を中止する．この場合は，最適化が不能である．

なぜならば，挿入ページをＺすると，移動した競合パターンは，最適化を行う前，すなわち挿入ページが０の状態に戻ってしまうからである．これは，式4.2 にｇｃ２を代入したときのｇｃ２ｚと，それを「Ｚページ（すなわちｇｚ行）移動したときの（ｇｃ２＋ｇｚ）」を代入したときのｇｃ２ｚ'とが，全く同じ行になることからも明らかである．同様に，挿入

ページをZ+1 にした場合の競合パターンは，挿入ページが１のときと同じである．言い替えれば，挿入ページを，それぞれ，１～(Z-1) 変化させて得たすべてのパターンが，当最適化手法におけるありえるすべてのパターンであり，それ以上移動させても新たな解を得ることはできない．

(5) 最適化の結果の出力: 各群の旧先頭論理アドレス，前後に挿入したダミーページ数の一覧表をファイルへ出力する．

なお，後述評価においては，固定エリアの論理アドレスと物理アドレスは一対一に固定であるが，R4000 のトレースデータを用いる場合は，(KSEG0とKSEG1[2]の）二対一に固定である．その場合，KSEG1は，キャッシュを使用しないので，無視できる．また，ブートモード[2] 時のベクターテーブル等は，オンラインのサービス中には使用されない特殊なエリアであるので，最適化の範囲外である．

４．２．２　浮動エリアの最適化（最適化Ｌ）

【原理】

固定エリアの論理ページと物理ページの関係は変更できない(上記4.2.1により既に変更されている場合を含む）ことを前提として，浮動エリアの論理ページと物理ページの関係を最適化する．

浮動エリアにおいては，論理ページと物理ページの関係を，OSが自由に設定できる．それらの対応関係を，OSは，TLB やPTで管理する．従って，当最適化では，物理ページの配置のみを変更し，論理ページの変更は行わない．

今，話を簡単にするために，図4.2 のように，キャッシュがｇｚ（＝６）行からなり，浮動エリアのメモリがｇａ（＝36）行の大きさからなり，１ページは３行であり，それらのうち，オンラインのサービス中に同図の①～⑥の行のみが使用されていると仮定すると，③は①と，⑤は②と，⑥は④と，それぞれ二次キャッシュ内の同一の行を使用するので，競合ミスを引き起こす．

当最適化では，競合ミスを起こしている上記の③，⑤，⑥を含むページ

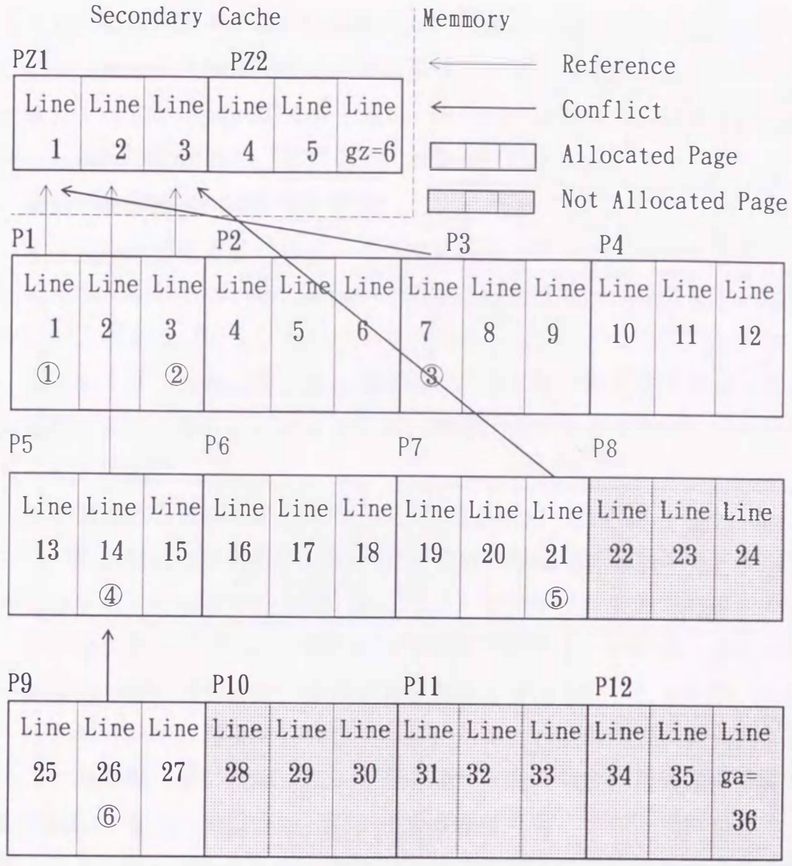


図４．２　最適化Ｌの概念図（最適化前）

（図中P3, P7, P9）を最適化対象として抽出し，それらの最適化対象のページのそれぞれについて，一つずつ，二次キャッシュ内の別のページへの参照に変更した場合の競合ミスの発生状況を，二次キャッシュ内の全ページについて計算し，それらの中で最も競合ミスが少ない二次キャッシュ相対ページ番号を割当てる．

例えば，図中③については，まず，最適化対象である③と⑤と⑥を除いた(①と②と④だけの)ものを反映した二次キャッシュのアクセスの状況を

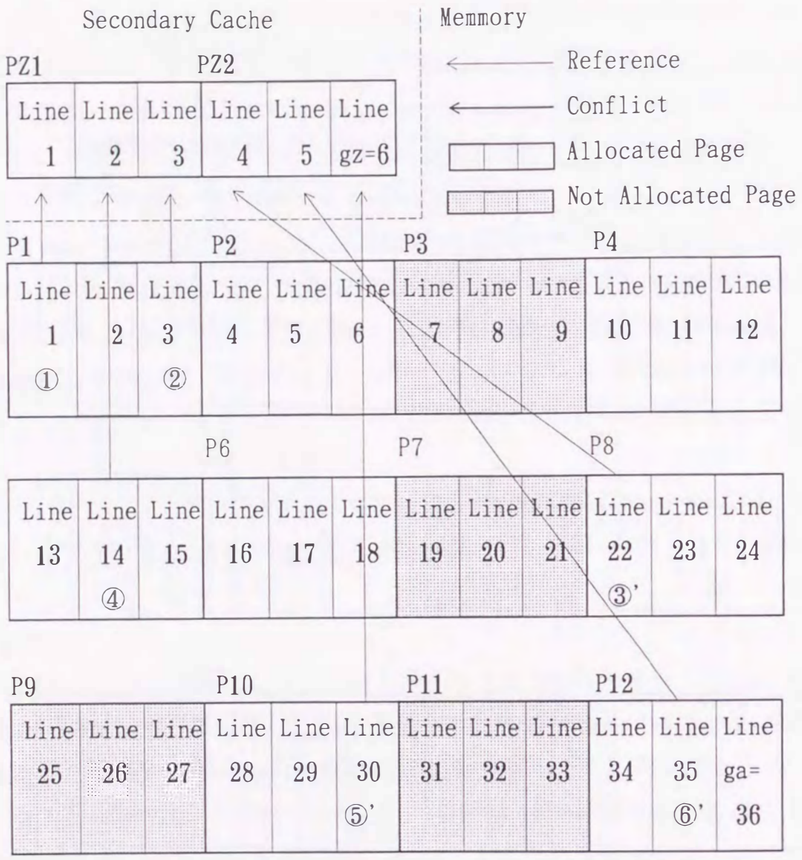


図４．３　　最適化Ｌの概念図（最適化後）

表す表4.6 を作成し，図中の二次キャッシュのページPZ1 に③を割当てたときの競合ミス数を計算し（この場合１となる),次にPZ2 に割当てたときの競合ミス数を計算する（この場合０となる）．この時点で，二次キャッシュ内のすべてのページに対する競合ミス数を計算し終わっているので，それらのうち，最も競合ミス数が少ない「二次キャッシュ相対ページ番号（この場合PZ2)」を，③の最適化結果とし，③を最適化対象より除去する．

次に，⑤については，最適化対象である⑤と⑥を除いた｛①と②と③（

最適化後の参照を反映）と④だけの｝ものを反映して，二次キャッシュのアクセスの状況を表す表4.6 を作成し，同様に，⑤をPZ1 とPZ2 に割当てたときの競合ミス数を計算し（この場合１と０となる),それらのうち，最も競合ミス数が少ない二次キャッシュ相対ページ番号（この場合PZ2)を，⑤の最適化の結果とし，⑤を最適化対象より除去する．

同様の処理を⑥についても行う．

すべての最適化対象に対する最適化が終了したら，最適化の結果の二次キャッシュ相対ページ番号を満たす未使用の物理ページを探して，物理ページを入替える．

以上により，最適化前に存在した図4.2 の太線矢印で示す競合ミスは，最適化により，図4.3 に示すように，競合ミスが０となる．

【具体的手順】

当最適化は，以下の手順で行う．

（1）参照されている各論理ページの情報の作成：　固定エリアと浮動エリアのすべてに対して，上記4.2.1 の(1）と同様に，参照されている論理ページ毎に表4.3 のNo１～No132 の情報を作成する．その際，上記4.2.1 で最適化を行った場合は，その結果を継承する．

（2）全ページ競合数と空き二次キャッシュページ数の特定：上記4.2.1 の(3）と同様に，二次キャッシュ内の各ページの競合の情報である表4.6 を作成し，全ページ競合数（W)を求める．その後，表4.6 の情報をすべてチェックして，表4.6 のNo1 のページ重複数が０であるページ（全く使用されていない二次キャッシュ内のページ）の数を求めて，その数を空き二次キャッシュページ数(K）とする．

（3）最適化対象の特定：二次キャッシュ内の各ページのうち，表4.6 のNo281 のページ競合数が零でないページに対して以下を行う．

（A）表4.6 のNo281 のページ競合数とNo282 以降の情報より，競合を起こしている表4.3 のNo1 の論理ページアドレスをチェックして，固定エリアのページが含まれているかどうかをチェックする．

（B）固定エリアのページが含まれていれば，該ページは論理ページと物理ページの関係を変更しない（最適化対象外）とし，固定エリア以外のす

べての競合を起こしているページの表4.3 の構造体の配列の添字を，シミュレータの内部情報である最適化対象キューに挿入する．

（C）固定エリアのページが含まれていなければ，競合を起こしているページのうち，表4.3 のNo3 の参照行数が一番多いページを最適化対象外とし，そのページを除いた「競合を起こしているページ」の表4.3 の構造体の配列の添字を，最適化対象キューに挿入する．

（D）上記(A) ～(C) を二次キャッシュ内のすべてのページに対して行ない，それが完了したときにおける「最適化対象キューに挿入されているページの数」を，最適化対象数(J) とする．

（E）J <= Kであれば，空き二次キャッシュページだけで，競合ミスを最小限にすることが可能であるので，最適化対象キューにある論理ページに，各空き二次キャッシュページの二次キャッシュ相対ページ番号を順次割当てて，最適化は終了する．

（F）J > K であれば，空き二次キャッシュページだけで競合を最小限にすることはできないので，最適化対象キューにある論理ページを，表4.3 のNo3 の参照行数が少ない順に並べ変えて，参照行数が多いものからK ページ分に，空き二次キャッシュページの二次キャッシュ相対ページ番号を順次割当てて，最適化対象キューより削除し，

最適化対象数J = J - K

とする．

（4）重み計算用テーブルの作成: 表4.3 の各論理ページの情報のうち，最適化対象キュー内にないページの情報のみを使用して，表4.6 の二次キャッシュの各ページの競合の情報を再度作成する．その作成した情報を，重み計算用テーブルとよぶ．

（5）最適化の実施: 最適化対象キューより，最も参照行数が多い表4.3 の論理ページの情報を取出して，そのページに対して，以下により最適化を行う．

（A）「表4.3 のNo5-132 の行参照有無情報」と，「二次キャッシュ内相対ページ番号m= 0～Z-1 のそれぞれの表4.6 の二次キャッシュの各ページの重み計算用テーブルのNo3 ～130 」との論理積と論理和をそれぞれとり，

その結果の「零でない行の数」を，それぞれ，論理積競合数(m) と論理和競合数(m) とする．

（B）論理積競合数(m) が最小であって，かつ論理和競合数(m) が最小である二次キャッシュ相対ページ番号mを，最適化を行っている論理ページの表4.3 のNo.136に設定して，No135 を最適化済に設定することにより，当論理ページの最適化を完了する．

（C）「二次キャッシュ相対ページ番号mの重み計算用テーブル（表4.6）のNo3-130 」と，「上記の最適化が完了した論理ページ（表4.3)のNo5-132 の行参照有無情報」との論理和を取った結果を，二次キャッシュ相対ページ番号mの重み計算用テーブル（表4.6)のNo3-130 に格納することにより，次の最適化のための重み計算用テーブルを作成する．

（D）最適化対象キューに論理ページの情報がまだあるなら,(5)に戻って,最適化対象キューに情報がなくなるまで，上記の最適化を繰返す．

（6）最適化結果の出力: 上記(5) の最適化がすべて終わった時点で，それらの結果をOSに反映するために，表4.3 の各論理ページの情報のうち，浮動エリアのすべてのページの情報について，表4.8 の①の情報を，OS用結果ファイルに出力する．①の情報のうち，No3 の新二次キャッシュ相対ページ番号には，以下を設定する．

（A）表4.3 のNo135 のページ属性が最適化済のものは，表4.3 のNo136

表４．８　最適化Ｌの結果情報

No	項目	①OS用	②ｼﾐｭﾚｰﾀ用
1	論理ﾍﾟｰｼﾞ ｱﾄﾞﾚｽ	○	○
2	ﾀｽｸ ID（ﾕｰｻﾞ 空間のみ）	○	○
3	新二次ｷｬｯｼｭ 相対ﾍﾟｰｼﾞ 番号	○	×
4	新物理ﾍﾟｰｼﾞ ｱﾄﾞﾚｽ	×	○

（注)ﾀｽｸ ID の情報は，ｼｽﾃﾑを再立上げすると変化する場合があるので，最適化結果をOSに与える場合は，最適化時のｺｱﾀﾞﾝﾌﾟ を元に文献[7] で述べた手法でﾌﾟﾛｸﾞﾗﾑ 名に変換し，ﾀｽｸ IDの情報と合わせて，OSに与える．

の相対ページ番号.

(B) 該ページ属性が最適化不要のものは，表4.3 のNo4 の旧物理ページアドレスから求めた二次キャッシュ相対ページ番号.

また，それらの最適化結果の効果をシミュレータに計算させるために，表4.3 の各論理ページの情報のうち，「No135 に最適化済が表示されている浮動ページの情報」についてのみ，物理ページを疑似的に割当てを行い，上記4.2.1 で最適化Xを行ったページも含めて，表4.8 の②の情報を，シミュレーション用結果ファイルに出力する.

4．2．3　最適化結果の反映手法

シミュレータは，上記のシミュレーション用結果ファイルを入力して，表4.2 のRISCアクセス状態データのNo2 の物理アドレスをすり替えて，再度，シミュレーションを行うことにより，最適化後のシステムの処理能力等を明らかにする.

一方，OSは，下記 (A)～(C) の新規機能を実現するシステムコールを提供し，上記のOS用の結果ファイルを入力し，サービス初期設定の最後の部分で，入力した最適化に関する情報をパラメータとして，該新規機能のシステムコールを発行することにより，最適化Lの結果を反映する.

(A) パラメータで指定されている最適化に関する情報がカーネル空間の場合は，「指定された論理アドレス」，ユーザ空間の場合は「指定された論理アドレス」と「タスクIDに対応するプログラム名」(以下，指定された論理アドレス等と略記) が，物理ページを使用していることを確認する.

(B) パラメータで指定された論理アドレス等に対する物理アドレスを得[7]，指定されている新二次キャッシュ相対ページ番号と一致するかどうかをチェックする.

(a) 一致すれば，その論理アドレスについては，物理ページの変更は不要である.

(b) 一致しなければ，下記(C) 以降を実施する.

(C) 下記の順に，指定された二次キャッシュ相対ページ番号を持つページを，OS内から探して，物理ページの入替え（配置替え）を行う.

(a) 空き物理ページキュー: 使用されていない物理ページのキュー. 通常は，これだけで十分賄えるが，万一満たされない場合は，下記(b) も探す.

(b) 使用実績がない連続ページ内のページ: 「物理ページを使用している連続した {二次キャッシュの大きさZ×(n=0,1,..)} ページであって，システムコールのパラメータとして指定されたすべての論理アドレスを含まない」空間を，あらかじめすべて求めておき，その中のページ.

4．3　評価結果

4．3．1　前提条件

当評価においては，第2章で使用した実RTS のトレース情報より作成したRISCアクセス状態データを使用して，上記の最適化を行わなかった場合の処理能力と，最適化後の処理能力とを求め，それらを比較することにより，最適化の効果を明らかにする.

上記の4.1 で述べた前提条件以外の評価の前提条件は，以下のとおりである.

(1) スタート状態: 下記の評価では，実測時により近い評価を行うために，ウォームスタートでの評価を行う.

(2) 最大構成を考慮した二次キャッシュの大きさ: 当評価で使用するトレースデータは，「ユーザタスク数が上り電文用に1タスク，下り電文用に1タスク」の構成で取得したものである. 一方，実在する商用マシンでは，最大構成で各8タスクが存在しえる. 従って，最大構成での二次キャッシュの負荷状態は，該トレースデータに比べて，プログラム部分では，共用されるために，負荷は増えないが，データ部分では，タスクが増えた分，負荷が重くなると思われる. よって，トレースデータのアクセス状態に比べて，最大構成の商用マシンでは，二次キャッシュの負荷が数倍となることもありえる.

しかしながら，実在する商用マシン上で追加データを取得することは，試験期間が完了した現在では不可能である．それを疑似する方法として，当評価では，二次キャッシュの大きさを，実在マシン(1 MB)の1/2(512 KB)，1/4(256 KB)，および1/8(128 KB) としたときの特性について述べる．実在する最大構成の商用マシンにおいては，当然ながら二次キャッシュは２～４MBを実装するであろうから，そのときの特性は，「二次キャッシュが１MB～256 KB付近の当評価結果の特性」に近いものになると考えてよい．

４．３．２　　最適化Ｌの効果

表4.9 に，二次キャッシュの大きさを１MB，512 KB，256 KBおよび128 KBとしたときで，最適化Ｘを行っていないときの「最適化Ｌの実施前と実施後の処理能力等」を示す．

同表中，①，②，③，および④は，最適化Ｌ実施前の特性であり，①’，②’，③’，および④’ は最適化Ｌ実施後の特性である．また，同表のNo1 ～No7 は，第２章で述べたシミュレータと同様の「動的なシミュレーション」の結果であり，No8 ～No11は，当最適化手法における静的統計情報である．

同表のNo1 とNo2 より，最適化Ｌの実施前に比べて，実施後では，処理能力がおよそ20～60[%] 向上することがわかる．

また，同表No3 とNo4 より，最適化Ｌの実施により，二次キャッシュのミス率がおよそ31～68[%] 減少していることがわかる．しかし，二次キャッシュの大きさが1 MBと512 KBにおいては，最適化Ｌの実施後のミス率が1.2 [%] および2.3[%]であり，かつ同表No9 の①’ と②’ より，それらのミス率は，すべて，固定エリア相互間の競合ミスにより生じていることがわかる．従って，それらのミス率を零にできると仮定するなら，ミス率の低下と処理能力の向上の関係より，それぞれ，更に12[%] {= 20.45(No2 ①’)÷[3.2(No3①)-1.2(No3 ①’)] ×1.2 (No3①’)} ，および25[%] {= 44.01(No2 ②’)÷[7.3(No 3 ②)-2.3(No3 ②’)] ×2.3(No3 ②’)} の処理能力の向上が得られると予想できる．

従って，最適化Ｘが必要である．

表４．９　最適化Ｌの実施前後の処理能力等

No	項目＼二次ｷｬｯｼｭ／最適化Ｌ		1 MB		512 KB		256 KB		128 KB	
			①前	①’後	②前	②’後	③前	③’後	④前	④’後
1	MIPS		31.0	37.3	23.2	33.5	16.5	26.3	10.9	14.2
2	同α^{*1}[%]		-	20.5	-	44.0	-	59.7	-	29.5
3	二	ﾐｽ率	3.2	1.2	7.3	2.3	14.3	5.6	26.1	18.1
4	次	同α^{*1}[%]	-	-62.5	-	-68.5	-	-60.8	-	-30.7
5	ｷｬｯ	ﾐｽ回数	899	326	2072	664	4094	1596	7575	5251
6	ｼｭ	同α^{*1}[%]	-	-63.7	-	-68.0	-	-61.0	-	-30.7
7	行使用率[%]		13.9	14.3	26.0	28.5	45.4	54.3	67.8	84.5
8	静的競合ﾍﾟｰｼﾞ数		69	4	116	11	169	103	204	203
9	（同固定ｴﾘｱ分）		-	(4)	-	(11)	-	(38)	-	(59)
10	静的競合行数		292	28	877	89	1877	515	3321	2332
11	最適化ﾍﾟｰｼﾞ数		-	36	-	66	-	107	-	141

α^{*1}: 最適化前（①，②，③，④）に比した最適化後（①’，②’，③’，④’）のMIPS，二次ｷｬｯｼｭのﾐｽ率，二次ｷｬｯｼｭのﾐｽ回数の改善率．

表４．１０　最適化Ｘの実施前後の処理能力等

No	項目＼二次ｷｬｯｼｭ／最適化		1 MB		512 KB		256 KB		128 KB	
			L+X①”	X分①”’	L+X②”	X分②”’	L+X③”	X分③”’	L+X④”	X分④”’
1	MIPS		41.9	4.6	41.9	8.4	33.5	10.0	13.9	-0.1
2	同α^{*1}[%]		35.2	14.8	80.2	36.2	103.4	43.7	28.2	-1.3
3	二	ﾐｽ率	0.0	-1.2	0.0	-2.3	2.4	-3.2	18.1	0.0
4	次	同α^{*1}[%]	-100	-37.5	-100	-32.1	-83.2	-22.4	-30.7	0.0
5	ｷｬｯ	ﾐｽ回数	0	-326	0	-664	672	-924	5319	68
6	ｼｭ	同α^{*1}[%]	-100	-36.3	-100	-32.1	-83.6	-22.6	-29.8	0.9
7	行使用率[%]		14.4	0.1	28.9	0.4	55.3	1.0	85.4	0.9
8	静的競合ﾍﾟｰｼﾞ数		0	-4	0	-11	113	10	210	7
9	（同固定ｴﾘｱ分）		(0)	(-4)	(0)	(-11)	-	-	-	-
10	静的競合行数		0	-28	0	-89	362	-153	2305	-27
11	最適化Ｌﾍﾟｰｼﾞ数		37	1	72	6	110	3	137	-4
12	最適化Ｘ実行		完了		完了		中止		中止	
13	ﾀﾞﾐｰ 挿入箇所数		2		4		4		3	
14	挿入合計ﾍﾟｰｼﾞ数		4		23		34		11	

α^{*1}: 最適化前（表4.9 ①，②，③，④）に比した，MIPS，二次ｷｬｯｼｭ のﾐｽ率，二次ｷｬｯｼｭ のﾐｽ回数の改善率．但し，①”’，②”’，③”’，④”’，は，①”，②”，③”，④”と①’，②’，③’，④’ との差分である．

4．3．3　最適化Xの効果

表4.10の①",②",③",および④" に，それぞれ，二次キャッシュの大きさを1 MB,512 KB,256 KBおよび128 KBとしたときの，最適化Lと最適化Xの両者の実施後の処理能力等を，①"'，②"'，③"'，および④"'に，最適化Xの分の処理能力等を示す．最適化Xの分の処理能力等は，No12～14以外は，表4.10の①",②",③",④" から，それぞれ，表4.9 の①',②',③',④' を差し引いた差分である．No12～14は，最適化Xの統計情報である．表4.10と表4.9 より，以下のことがわかる．

（1）表4.10のNo2 の①",②",③",および④" より，最適化Lと最適化Xの両者の実施により，処理能力が28～103[%]向上している．

（2）表4.10のNo2 の①"'，②"'，③"'，④"'と表4.9 のNo7 の①，②，③，④より，行の使用率がおよそ67[%] のときには，最適化Xは処理能力向上に寄与していない．しかし，行の使用率が14～45[%] の間においては，最適化Xは，およそ15～44[%] の処理能力の向上をもたらす．

（3）表4.10のNo14等より，最適化Xによる15～44[%] の処理能力の向上は，わずか２～４か所に，合計で4 ～34ページだけのダミーページの挿入により実現されている．従って，実OSでのシステム生成時での対処は，十分に容易である．

以上のように，最適化Xには適用領域が存在するが，最適化Xと最適化Lの両者による処理能力の向上効果は，最低で28[%],最大で103[%]もある．従って，それらの効果は大きいと言える．

4．4　効果

上記のように上記手法の実現等により下記を明らかにすることができた．

（1）　サーヒス中にメモリの動的確保・開放を行わないRTS であってR4000 プロセッサの使用を前提とした場合，実RTS のトレースデータに基づいてシミュレートした結果は，二次キャッシュの容量を128 KB～1MB に変化させたとき，上記最適化Xと最適化Lの総合の処理能力の向上率が，

28～103 [%] であることを示しており，それらの効果は大きい．

（2）上記のうち，最適化Lの処理能力の向上率は，31～68[%] である．

（3）上記のうち，最適化Xの処理能力の向上率は，最適化前の二次キャッシュの行の使用率が50[%] 未満では，14～45[%] であるが，同使用率が67[%] では，処理能力の向上をもたらさない．

第５章　結論

前章までに，RTS における処理能力の評価手法について述べた．本章では，全体のまとめを行うと共に，残された課題について言及する．

第２章では，CISCのトレースデータを入力しRISCの処理能力要因等を解析する新手法を提案・実現し，実際の商用RTS の特性に基づいて以下の(2-1）～(2-5）等を，RISCTのRTS のサービスインの1 年以上前に明らかにすることにより，今後のRTS でのシステム設計や評価での目安を提供した．

(2-1）TSS 用のAPのRISC/CISC 実行命令数比は，実RTS のユーザ空間で走行するプログラムの命令であって，「命令長が4 バイトを超え多数バイトの処理を特徴とする複合命令」を除いた単純命令のみのRISC/CISC 実行命令数比とほぼ一致するが，実RTS のシステム全体のRISC/CISC 実行命令数比より，およそ30[%] 小さい．

(2-2）TSS 用のベンチマークによる処理能力の評価結果は，RTS 全体に対するものより2 倍以上良い（危険側の）評価結果となるので，RTS においては，当研究のようにシステム全体の特性を反映した評価が重要である．

(2-3）「キャッシュへの書込み時にメモリにも書込むことを特徴とするWT方式」を採用しているR3000 では，当研究対象のRTS のように，ストア命令による書込み頻度が高い場合，それらによる処理能力の低下が大きいので，当研究対象の実RTS への導入には適していない．

(2-4）「キャッシュミス時にのみにメモリに書込むことを特徴とするWB方式」を採用しているR4000 の処理能力は，二次キャッシュの速度に大きく依存するので，R4000 を導入する場合，高速な二次キャッシュを使用することが望ましい．

(2-5）R4000 では，二次キャッシュは，命令とデータとを独立にするより，共用にした方が処理能力が高く，一次キャッシュの行の大きさは，32バイトより16バイトの方がよく，二次キャッシュの行の大きさは，32バイトでも64バイトでも，ほとんど変わらない．

今後の課題として，以下が考えられる．

(2-A）タスク数の影響の検討: 第２章では，AP～カーネルまでのRTS 全体の評価を実現している．しかし，その入力は，既存トレースデータであり，限定された業務タスク数の下でのデータである．商用のRTS では，そのシステムの規模によっては，もっとタスク数が多いものもありえる．タスク数を増加させた場合，データの数が増えることによる処理能力の低下と，タスクがプログラムを共用することによる処理能力の上昇の両者の影響が考えられる．今後，それらの影響の評価が必要である．

(2-B）定期走行タイマー処理の影響の検討: 文献[1] で述べられている定期走行タイマー処理は，過去のトレースデータの取得上の制約より，今回の研究で使用したトレースデータに含まれていない．このことは，該トレースデータを取得したシステムにおいて，タイマー処理が主に異常処理ルートおよび低負荷状態でしか動作しないため，タイマーを停止しても，高負荷時の処理能力評価に余り影響を与えないことに起因している．しかしながら，最近盛んになってきたマルチメディア処理を行うRTS[2][3] においては，高負荷時において，タイマーの動作が不可欠である．従って，参考特許にもあるように，タイマーの動作を含めたトレースデータを取得できるものを，つい最近開発した．今後，上記の処理の影響を含めた評価が必要である．

なお，本論文では，2.3.1 のRISCアクセス状態データ生成手法については，実測データとの比較を示すことにより，精度が良好であることを述べた．しかし，2.3.2 のRISC処理能力評価手法については，綿密な机上検討に基づくシミュレーションのみであり，実測結果との比較に基づく精度の検証が必要であった．これに関しては，第３章で検証した．

第３章では，RISCの処理能力の解析の精度検証について述べた．そこでは，TLB ミスのオーバヘッド測定用のベンチマーク，および，実RTS 全体のキャッシュのヒット率を反映した疑似RTS ベンチマークを実現し，それらのシミュレーションと実測結果等を得た．

それらの結果により，

(3-1）ドライストーンベンチマークでのシミュレーションと実測との誤差は0.1[%]未満であり，実RTS のキャッシュのヒット率等を疑似した疑似RTS ベンチマークでの誤差は7.7[%]よりかなり下回るので，第２章で述べ

たシミュレーションの精度はかなり高いこと，

（3-2）実RTS のTLB ミス１回当りのオーバヘッドは，その頻度に余り関係なく一定で，文献[4] での使用値のおよそ６割であること，および

（3-3）第２章で述べた実RTS でのTLB ミスによるオーバヘッドは，システム全体のCPU 使用量のおよそ5.4[%]であること，

を明らかにした．

今後の課題としては，以下が考えられる．

当章では，実測環境上の制限等より，割込み等によるのキャッシュの擾乱要因の正確な分析ができなかった．より高精度の評価のためには，割込み等の擾乱要因の更なる分析が必要である．

また，当章では，メモリの割当て状態を正確に反映した結果について述べ，その状態の相違により処理能力が変動することを定量的に示した．その状態の相違は，ベンチマーク走行を各走行毎に利用者タスクの生成により実現したことにより発生している．しかしながら，処理能力の最適化が完了した高性能なRTS では，ユーザタスクをサービスの初期設定時に生成し，以降サービスの終了時まで該タスクを終了しないように設計する[5][6][7]．その場合，タスクのメモリの割当て状況は，サービス中にほとんど変化しない．従って，サービスの初期設定時のメモリの割当て状況によっては，処理能力がサービス中にずっと低い値に停滞する場合があり得る．その影響を回避するための何らかの手法の実現が必要であった．これに関しては，第４章で対処した．

第４章では，二次キャッシュの競合ミスの回避と，その評価結果について述べた．そこでは，物理ページと論理ページの関係が固定であるエリアに対する新たな最適化手法（最適化Ｘ）と，それらのページの関係が浮動であるエリアに対する新たな最適化手法（最適化Ｌ）を提案した．また，第２章で実現した新手法の応用として，実RTS のトレースデータに基づいて，R4000 でのそれらの効果を評価することにより，

（4-1）二次キャッシュの容量を128 KB～１MBに変化させたとき，上記の両最適化手法の総合の処理能力の向上率は，28～103[%]であること，

（4-2）上記のうち，最適化Ｌの処理能力の向上率は，31～68[%] であること，および

（4-3）上記のうち，最適化Ｘの処理能力の向上率は，二次キャッシュの行の使用率が 50[%]未満では，14～45[%] であるが，該使用率が67[%] では，処理能力の向上をもたらさないこと，

を明らかにした．

今後の課題には，以下が考えられる．

（4-A）当評価のシミュレータの精度は，第３章の評価結果等より，十分に高いと考えられる．しかし，提案した手法のうち，最適化の結果のOS上での反映手法については，その実装には至っていない．今後，その実装と実測により，検証の強化が必要である．

（4-B）当評価で使用した実RTS のトレースデータは，文献[1] で述べたように，既存の為替交換・中継を行うトランザクション処理用小型RTS によるものである．該実RTS は，広く知られているオンライントランザクション処理用ベンチマーク:TPC-A [8]に似ており，またAP～カーネルに到る全階層を含んでおり，何かに偏心したような特殊なシステムではない．また，「商用の最大構成を考慮したキャッシュの負荷」の補充のために，二次キャッシュの大きさを実在するマシンより小さくすることにより，擬似的にキャッシュの負荷を上昇させて評価した．更に該システムは，小型であり，かつ座席予約や銀行業務等にみられる定型処理[8] であるが故に，限られたロジックを繰返し実行する傾向が大である．従って，上記の評価におけるキャッシュの負荷等に不足はないし，一般性に欠ける部分は少ないと考えられる．しかしながら「トレースデータをどこまで取れば十分であるか」等の実測評価の実現には至っていないし，もっと大型の異なるモデルの場合どうなるかの評価には至っていない．上記効果の実証等のためには，今後，それらの実現が必要である．

（4-C）また，上記実RTS においては，入出力頻度は一電文当り数回であり，データ転送の総量は，一電文当りたかだか１KBに過ぎないので，それらの影響の評価を省略している．しかしながら，例えば，最近のマルチメディア処理[2][3]等を考えると，そこでは，画像データを入出力するために，その転送量がかなり大きなものとなる．その場合，入出力の影響は無

視できないものとなるであろう．特に，入出力機器がIBM PC-AT 互換機のISA バスを使用する場合は，CPU がデータ転送を行う[9] ので，当論文で使用したトレースデータによる解析の手法で解析が可能であるが，最近増えてきたバスマスター方式の入出力機器の場合では，データの転送を入出力機器側で行う[9] ので，それらの転送によるCPU とバス*1との競合によるCPU の処理能力の低下の影響を反映する新たな手法の確立が，今後必要である．

なお，マルチメディア処理を行うRTS は，新しい技術であるが故に，現在発展途上にあると考えられ，文献[2][3]に見られるように，いまだ，処理能力が向上している最中にあり，メモリの割当ても，いまだ，動的に行われている．しかしながら，過去の処理能力評価の経験からは，メモリの動的割当ては，処理能力低下に確実につながる．従って，高性能に向けての最適化が進むにつれて，今後は，当評価における実RTS のように，あらかじめ必要量を割付けることにより，最高の処理能力を実現する方向へ進むであろう．なぜならば，それを行わない限り，処理能力の改善の余地が残されたままであるといっても過言ではないからである．

以上のような今後の課題があるとはいえ，一般に，処理能力の最適化が完了した高性能なRTS においては，システムの処理能力のボトルネックがCPU となるようにシステム設計を行う．（それが実現できなかった場合，高性能RTS としては失敗作であるといっても過言ではないであろう．） その場合，CPU の処理能力を事前に正確に予測することが今後も非常に重要である．当研究の開始は1993年であり，その最初の外部発表は1996年２月に行った[10]．それらと前後して，CPU の処理能力の評価を促進するために，"Performance Analysis and its Impact on Design (PAID)" というワークショップ*2が，IEEEで1995年に発足しており[11]，評価手法として当研究で述べたシミュレーションやエミュレーションの手法が，益々盛んになりつつある．

そのことは，ベンチマークが不要になるということを意味しない．なぜなら，シミュレーション等にはモデルが必要であり，そのモデルとしての役割の一端をベンチマークが担っているからである．

最近最も人気が高いベンチマークは，SPECベンチマーク[12]であると言われている[13]．SPECベンチマークには，現在，整数の処理能力を測定するCINTxxと，浮動小数点の処理能力を測定するCFPxx の二種類がある[14]が，当研究で使用した実RTS のトレースデータには浮動小数点命令が含まれていないので，該実RTS は，CINTxxの方に近い．

一方，第２章での評価で参照したUNIX用のベンチマーク[15]は，CINTxxで使用されているプログラム[14]のうち，espressoとcompressを含んでいる．これらのうち，compressというプログラムは，上記のUNIX用ベンチマークの中では一番「VAX に比したMIPS値」(SPEC レシオ) が低いプログラムである．また「最近のSPECベンチマークの結果(SPECint95のR10000に対する結果)[16] の中で最もSPECレシオが低いプログラム(vortex)」と上記のcompressとのSPECレシオの相違は，20[%] 程度しかない．しかしながら，第２章で述べたように，実RTS のトレースデータに基づいたSPECレシオは，compressより，かなり低い．従って，RTS においては，今後も，実RTS の特性に基づいた評価をすることが非常に重要である．

更に，シミュレーションによる評価の観点からSPECベンチマークを考察すると，もう一つの問題点が存在する．それは，R10000に関する文献[16]で，SPECベンチマークの１走行に，最低でも146 秒を要しているという点である．これは，シミュレーションを行うには余りにも長すぎる．なぜならば，146 秒という走行時間は，そのときの処理能力が13 MIPS[16]であるので，少なくとも1.9 ギガステップ以上の走行に相当する．そのステップ数のシミュレーションを行うには，これまでの経験から言って，現在において最高速のワークステーションを使用したとしても，恐らく半年から１年のシミュレーション時間が必要であろう．

また，SPECベンチマークには，1992年制定のSPEC92と，1995年制定のSPEC95とがあるが，近年，実行時間をより長くする傾向にある[17]．その原

バス*1: CPU やメモリや入出力コントローラ間でデータのやり取りを行うために，それらの間で共通に使用される信号線群．

ワークショップ*2: 研究会．

因は，『CPU の処理能力が上がり，測定時間が短くなったため，その測定誤差を小さくするために長くしている』と言われている[17]．しかし，測定時間は，単にループさせて時間を延ばすことにより延長できる．従って，該文献での説明は，説得力に欠ける．

シミュレーションを行うという観点からは，少なくとも，SPECベンチマークは，そのプログラムのロジックの長さを，現状の20分の１未満に変更することが望ましい．測定誤差が問題であれば，１ロジックを20回ループするように変更すればよい．それらの変更が実現されるなら，当研究で実現したシミュレーションの手法は，それらのうちの２回のループの解析によって，なぜそのような処理能力となるかという疑問に対する詳細な答えを出すことが可能であろう．

しかしながら，SPECの運営の資金は米国の計算機の製造元によって提供されており，第三者がその運営の決定に影響を及ぼす仕組にはなっていない[14]．従って，それらの実現は，多分今後も非常に困難であろう．とはいえ，MIPS社の創立者の一人であるJohn L. Hennessyも文献[18]で述べているように，将来は，性能測定用ベンチマークに関する決定は，中立グループによって行われるか，または最低限中立のグループがメンバーに入るべきであろう．

当研究の過程において，シミュレーション時間を何とか短縮するために，シミュレータをスーパーコンピュータ(CRAY2) 上に移植して，どの程度速くなるかを検討したことがある．該スーパーコンピュータはバイトマシン*1ではなく，ワードマシン*1であるため，その移植にはかなりの工数を必要としたが，移植後に得られた該マシンの処理能力は，市販のワークステーションと何ら変わらなかった．要するに，スーパーコンピュータは，浮動小数点命令等の並列実行ではメリットがあるかもしれないが，Ｃ言語記述の整数演算中心のプログラムの単なる移植ではメリットはないというこ

バイトマシン，ワードマシン*1：メモリへのアクセスをバイト単位でアドレス指定可能な計算機をバイトマシンといい，(バイトより大きい) ワードでしか指定できない計算機をワードマシンという[20]．

とであろう．

また，シミュレーション用のシステムの処理能力の向上が今後ありえるとしても，その向上は，とりもなおさず評価対象の処理能力の向上にもつながるであろう．従って，SPECベンチマークのように，その向上に合わせて走行時間を増加するなら，シミュレーション用のシステムの処理能力の向上は，今後も，上記の問題点を解決し得ないと思われる．それらを勘案すれば，SPECベンチマークより，もっとキャッシュの負荷が重く，かつ走行時間が非常に少ないベンチマークが，今後も必要である．

当研究の中心であるR4000 は，文献[19]にもあるように，「チップのレベルでの互換を保証した上で，複数の製造元により供給が保証されている」点で，文献[13]で述べられているような「チップレベルの互換性が保証されないSPARC プロセッサ」とは異なる．また，R4000 シリーズは，方式的には，パイプラインを１本しか持たないことを特徴とするスーパーパイプライン方式[21][22]である．しかし該CPU を開発したミップス社は，その後スーパースカラ方式のR5000 およびR10000等の新しいプロセッサを発表した．また，それらのチップの製造は，日本でも行われている[23][24][25][26][27]．

それらとR4000 との主な相違点は，以下のとおりである．

(5-1) R5000 やR10000では，下記の命令の機能拡張が追加されている．

(A) 浮動小数点命令関連の命令

(B) 条件付転送命令

(C) プリフェッチ用の命令　　(R10000 のみ)

(D) キャッシュ関連の命令

(5-2) R10000では，TLB が48->64エントリに変更．

(5-3) 一次キャッシュが命令・データ共に，２ウェイ*1 32 KB．

(5-4) R5000 は，２ウェイ・スーパースカラ方式*2．　ステージの周波

２ウェイ*1：キャッシュの一つの行の番号に対して，二つの内容を格納できる方式．(連想性=2．)

ｎウェイ・スーパースカラ方式*2：パイプラインがｎ個ある方式．

数は，最大200MHz．R10000 は，5ウェイ・スーパースカラ方式．ステージの周波数は，最大300MHz．

(5-5) ブランチ予測と命令実行順序変更(out of order)．(R10000のみ)

上記のうち，(5-2) のTLB の変更に関しては，当研究での実RTS においては動作環境が小さいので，ほとんど影響はないであろう．(5-1) の命令の機能の追加については，RTSのOSで，それらが今後どのように活用されるかによって当研究結果への影響が異なる．従って，導入されたOSの状態を明確にした上で，必要に応じて，再検討が必要である．(5-3) ～(5-5) については，キャッシュとパイプライン等の方式上の大幅な変更である．従って，それらが当研究結果へどの程度影響するかは，今後，別途検討が必要である．

更に，今後のCPU の処理能力の評価に関して，今後の展望について述べる．当論文における第2章～第4章の処理能力の評価は，CISCにおけるシングルステップ割込みを利用したトレースデータを一貫して用い，第3章においては，RISC用のロードモジュールを入力としたエミュレーションを併用して，それらの補完を行った．それらを行わなければならなかった理由は，RISC上で初めてOSを開発し，その評価をOSの完成前に行う必要があったからである．すなわちRISCにおけるトレースデータが存在しなかっただけでなく，RISCにおけるトレースデータを取るための仕組みさえも存在しなかった．

そのトレースデータを取るための手法に関しては，Pentium の例に見られように，現在でも，CISCにおいてはシングルステップ割込み機能がサポートされているが，RISCにおいては該機能を有するCPU が存在しないという大きな問題点が存在する．

その問題点を克服する手法として，一つには，(1) 第3章で用いたエミュレーションによってトレースデータを得る手法がある．しかし，該手法は，第3章でもカーネル部分を含まないドライストーンベンチマークに限定したように，カーネル部分をエミュレートすることが非常に難しい．その困難さは，システムコールの動作を疑似するだけでなく，入出力機器等のハードウェアの動作のエミュレーションに非常に工数がかかることに起

因する．既に世に発表されている処理能力評価において，エミュレーションの手法だけを用いてユーザプログラムに閉じた評価を行っているものが多数存在する．その場合は，その評価精度は決して高くないであろう．従って，エミュレーションの手法は，その中にカーネルのエミュレーションを含む精巧な仕掛けを内在しない限り，評価精度の面から，RISCの評価の本命になるとは，今後も考えにくい．

また，第3章で述べたように，(2) ハードウェアモニタによる方法もある．しかし，最近のCPU は，一次キャッシュの信号線がチップ上にピンとして出ていない．よってキャッシュを停止しなければならず，そのためにはOSに手を入れなければならない．かつ，ハードウェアモニタによって大量の情報を得るためには，非常に高価な大規模なモニタを必要とする．従って，該手法は，今後も，資金力が豊富な一部のCPU の製造元等の利用に限られるであろう．

一方，最近，(3) 命令付加 (Instrumentation)方式[28][29]というものが普及しつつある．この手法は，ブランチ命令の次やブランチ先，および，メモリの参照・更新命令の前後に，情報を取るための命令を埋込む方式である．該手法は，APにおいては，上記の命令の埋込みが自動化されているものが殆どである[28]が，カーネルにおいては，制御命令等の技術的難しさのためか，該埋込みを手作業で行っている[29]．また，該手法では，埋込みによるアドレスのずれ等を後で補正しているが，命令の埋込みによってプログラムの大きさが，およそ2倍になってしまうため，カーネルを含めた全システムに導入すると，使用メモリ量が倍増してしまうという欠点がある．更に，該手法は，カーネルへの命令の埋込みと，その正常性の確認等に多大の技術力と工数必要とするため，OSの開発元等が行わない限り，かなり難しいのではないかと思われる．しかし，ソフトウェアだけを用いた実現であるので，一旦，高品質のものができ上がれば，急速に普及が進む可能性がある．

以上を総括すると，RISCにおいては，上記(3) の方向が最も有望であると思われるが，OSの製造元が，トレース機能を含むOSを公開するかどうかは，命令埋込みの自動化の達成度合や，企業戦略に依存すると思われる．

なぜならば，OSの製造元が「OSの中身がわかってしまうような機能」を利用者に開放している例は，ほとんどないからである．従って，RISCにおいては，今後も，OS等の製造元に所属しない研究者がCPU の処理能力評価を，正確に行うことは難しいことであるだろう．

反面，CISCでは，Pentium [30]のように，現在もシングルステップ割込みをサポートしているものがある．該機能がないものでも，マイクロ命令がサポートされているものが多いので，文献[31]のようなマイクロ命令によるトレースの実現が可能かもしれない．筆者は，Pentium のシングルステップ割込みを利用した，カーネルの動作を含めた全命令トレーサを，フリーウェア*1であり実時間用OSであるRT-Mach[32] 上で，最近，試作を完了した．該トレーサはマルチメディア拡張(MMX) 命令や浮動小数点命令等の処理を省略した限定版であるが，機会があれば，上記OSの一部分としての公開も可能である．

上記のような理由から，今後のOSに関連する研究は，有償で提供されて，ソースプログラムが非公開であるOSにおける場合よりも，フリーウェアであって，ソースプログラムが公開されているOSにおいて，より盛んになるのではないかと筆者は予想している．その意味で，米国において『ソフトウェアの製造元が，ソースプログラムが公開されているフリーウェアのOSにも注目し始めた』という報道があった[33]ことは，大変喜ばしいことであろう．

いずれにしろ，処理能力を改善するためには，なぜそうなるのかを正確に分析することが必要である．その点に関して，当研究で述べたシミュレーション等の手法は，今後ますます使用されることになるであろう．なぜならば，それらの手法は，取得する分析用の情報を必要に応じて自在に拡張でき，"なぜ ?" という疑問に必ず答えることが可能であるからである．

フリーウェア*1: 無償で提供されているソフトウェア[34].

謝　辞

本論文をまとめるに当り，終始，御指導と御高配を賜った九州大学大学院システム情報科学研究科・研究科長 牛島和夫教授 に厚く御礼申し上げます．また，本論文について貴重な御助言を賜った九州大学大学院システム情報科学研究科 安浦寛人教授および 牧之内顕文教授，ならびに，論文の細部にわたり討論を賜った同研究科 谷口秀夫助教授に，深謝申し上げます．

本研究は，筆者が日本電信電話株式会社・情報通信処理研究所に在職中に行ったものであり，この間，多数の先輩，同僚の方々から，御厚情を頂きました．特に，NTT情報流通プラットフォーム研究所 伊土誠一所長ならびに NTTソフトウェア株式会社 中川豊部長には，共同研究と共に，温かい御支援を頂きました．またNTT 情報通信処理研究所 鶴保征城元部長(現在NTTソフトウェア株式会社社長)，石野福彌元所長(現在一橋大学教授)，中村行宏元部長(現在京大教授)，和佐野哲夫元所長，東田正信元研究企画部長および歴代のリータの皆様方からは，温かい御支援と励ましを頂きました．

ここに，心から御礼申し上げます．

文献

第１章

[1] 鉄道総合研究所, "NTシステムでの応答時間悪化を試行錯誤で解決," 日経コンピュータ, no. 413, pp. 184-186, March 1997.

[2] 全国信用情報センター連合会, "大量の照会要求を処理しきれずチューニングとホスト増強で乗り切る," 日経コンピュータ, no. 435, pp. 141-143, January 1998.

[3] 京都大学医学部付属病院, "稼動直後からレスポンス悪化, 主記憶増強やDB調整で回復," 日経コンピュータ, no. 441, pp. 179-181, April 1998.

[4] AT&T ベル研究所編/長谷部紀元 清水謙多郎訳, "UNIX原典," pp. 1-393, 1986.

[5] 社団法人日本電子工業振興協会, "コンピュータの市場動向: 平成七年度出荷実績全体の概要," コンピュータガイドブック1996, pp. 9-13, October 1996.

[6] 喜連川優, 伏見信也, 田中茂, 丸山光行, "データベース処理におけるベンチマーク," 情報処理, vol. 31, no. 3, pp. 175-205, 1994.

[7] 社団法人トロン協会編, "原典CTRON大系:CTRON概説," pp. 1-180, December 1988.

[8] 杉村康, "マルチプロセッサソフトオーバヘッド評価におけるロックモデル解析の一手法," 電子情報通信学会論文誌(D-1), vol. J75-D-I, no. 10, pp. 917-925, October 1992.

[9] コンピュータ用語辞典編集委員会, "英和: コンピュータ用語大辞典," 日外アソシエーツ, p. 828, November 1989.

[10] コンピュータ用語辞典編集委員会, "英和: コンピュータ用語大辞典," 日外アソシエーツ, p. 650, November 1989.

[11] 竹垣盛一, "ハードリアルタイムシステムの応用例," 情報処理, vol. 35, no. 1, pp. 34-40, 1994.

[12] 徳田英幸, "分散リアルタイムシステムのためのOSアーキテクチャ," 情報処理, vol. 35, no. 1, pp. 18-25, 1994.

[13] 杉村康, "RTS のメモリ使用の分散的分析手法と実施例," 情報処理学会研究報告, システムソフトウェアとオペレーティングシステム, 95-OS-70-11, pp. 81-88, August 1995.

[14] 大久保利一, 香西省治, 二神新, 大南正人, "リアルタイム性を考慮した通信処理用OSインタフェース," 電子情報通信学会論文誌(D-I), vol. J78-D-I, no. 8, pp. 687-698, August 1995.

[15] 芳西崇, 小林伸幸, 中村仁之輔, 田中豪, "リアルタイムDBMS構築技術," NTT R&D, vol. 45, no. 1, pp. 27-32, 1996.

[16] 西尾信彦, 徳田英幸, " 連続メディア処理のためのミドルウェアの性能評価," 情報処理学会研究報告, システムソフトウェアとオペレーティングシステム, 95-OS-69-10, pp. 55-60, June 1995.

[17] 横田英史, "次世代RISC, 並列処理を導入しCMOSで100MIPS ねらう," NIKKEI ELECTRONICS, no. 487, pp. 191-200, November 1989.

[18] 横田英史, "RISC 形の32ビット・マイクロプロセッサ, CISC型との棲み分け進む," NIKKEI ELECTRONICS, no. 440, pp. 145-150, February 1988.

[19] NEC日本電気株式会社, "VR3000TM32ビット・マイクロプロセッサ アーキテクチャ編," pp. 1-191, 1989.

[20] NEC日本電気株式会社, "VR3000TM32ビット・マイクロプロセッサ ハードウェア編," pp. 1-133, 1989.

[21] MIPS Computer Systems, Inc., "MIPS R4000 Microprocessor Introduction," pp. 1-64, 1991.

[22] MIPS Computer Systems, Inc., "MIPS R4000 Microprocessor User's Manual," pp. 1-593, 1991.

[23] NEC日本電気株式会社, "VR4400TM64ビット・マイクロプロセッサ アーキテクチャ編," pp. 1-400, 1993.

[24] NEC日本電気株式会社, "VR4400TM64ビット・マイクロプロセッサ ハードウェア編," pp. 1-134, 1993.

[25] M. H. MacDougall, "Instruction-level program and processor modeling," IEEE Computer, vol. 17, pp. 14-24, 1984.

[26] H. J. Curnow and B. A. Wichmann, "A synthetic Benchmark: Whetstone," The Computer J., vol. 19, no. 1, pp. 43-49, 1976.

[27] R. P. Weicker, "Dhrystone benchmark: A synthetic systems programming benchmark," Commun. ACM, vol. 27, no. 10, pp. 1013-1030, October 1984.

[28] R. P. Weicker, "Dhrystone benchmark: rationale for version 2 and measurement rules," SIGPlan Notices, vol. 23, no. 8, pp. 49-62, August 1988.

[29] Systems Performance Evaluation Cooperative, "SPEC Benchmark Release 1.0," SPEC Newsletter, vol. 1, no. 1, Fall 1989.

[30] T. Horikawa, "TOPAZ: Hardware-tracer based computer performance measurement and evaluation system," NEC Res. & Develop., vol 33, no. 4, pp. 638-647, October 1992.

[31] A. J. Smith, "Cache memories," Comput. Surv., vol. 14, no. 3, pp. 473-530, September 1982.

[32] D. W. Clark and J. S. Emer, "Performance of the VAX-11/780 translation buffer : simulation and measurement," ACM Trans. Comput. Syst., vol. 3, no. 1, pp. 31-62, February 1985.

[33] 財団法人日本規格協会情報技術標準化研究センター, "システム性能評価標準化調査研究委員会報告書," pp. 1-57, March 1995.

[34] ラインホルト P. ワイカー, "ほころびが見えはじめたベンチマークテスト," NIKKEI BYTE, pp. 245, May 1991.

[35] Williams T., "RISC is simple, but benchmarking isn't," Comput Des, vol. 31, no. 2, pp. 67-76, 1992.

[36] R. E. Kessler and M. D. Hill, "Page placement algorithms for large real-indexed caches," ACM Trans. Comput. Syst., vol. 10, no. 4, November 1992.
[37] B. N. Bershad, D. Lee, T. H. Romer and J. B. Chen, "Avoiding conflict misses dynamically in large direct mapped caches," Proc. of the sixth Int. Conf. on Architectual Support for Programming Languages and Operating Systems, pp. 158-170, 1994.
[38] Clifford Barney Tom Manuel, "論争を巻き起こしつつ商用化が始まったRISCアーキテクチャ," NIKKEI ELECTRONICS, no. 402, pp. 252-258, August 1986.
[39] Fujitsu, "A RISC Tutorial," FUJITUSU S Family Manual, pp. 1-11, 1990.
[40] 板垣克彦, "高機能化進むRISC," 日経産業新聞, 7月23日, 1992.
[41] Hewlett Packard, "HP3000 and HP9000 Series Computers, Precision Architecture & Instruction Set Reference Manual," 1989.
[42] NTT電気通信研究所, "VLSIを用いたDIPS小型システム," 通研月報, vol. 36, no. 10, 1983.
[43] 飯島順一, 町田元, "アセンブラプログラミング詳説; IBMシステム/360,370; 上巻," 近代科学社, pp. 1-274, 1979.
[44] 飯島順一, 町田元, "アセンブラプログラミング詳説; IBMシステム/360,370; 下巻," 近代科学社, pp. 275-523, 1979.
[45] 日本デジタルイクイプメント株式会社教育部, "VAXアーキテクチャハンドブック," 共立出版株式会社, 1984.
[46] 加藤雅浩, " 分散環境で使う計算サーバ, RISCチップのマルチ構成と並列化コンパイラ, UNIXが鍵に," NIKKEI ELECTRONICS, no. 502, pp. 169-174, June 1990.
[47] Walter J. Price, "A Benchmark Tutorial," IEEE MICRO, vol. 9, no. 5, pp. 28-43, October 1989.
[48] Walter J. Price, "各種ベンチマーク・テストでシステム性能を比較する," NIKKEI ELECTRONICS, no. 497, pp. 189-208, April 1990.
[49] Reinhold P. Weicker, "An Overview of Common Benchmarks," IEEE COMPUTER, vol. 23, no. 12, pp. 65-75, December 1990.
[50] Ashis Khan, "64ビット・アーキテクチャのR4000 を開発," NIKKEI ELECTRONICS, no. 538, pp. 171-185, October 1991.
[51] 門田浩, "SPEC ベンチマーク, コンピュータをシステムレベルで評価," NIKKEI ELECTRONICS, no. 498, pp. 277-287, April 1990.
[52] 戸室隆彦, "SPEC ベンチマーク," 情報処理, vol. 37, no. 8, pp. 718-724, August 1996.
[53] SGI, "O2 R10000: SPECCint95," http://open.specbench.org/osg/cpu95/results/res98q1/cpu95-980206-02411.html, September 1998.
[54] 松岡正晴, 谷口秀夫, "RISC 性能評価の一手法:CISC との比較," 情報処理学会全国大会第41回, 1P-1, p. 6-73, 平成2年後期

[55] D. Bhandarkar and D. W. Clark, "Performance from Architecture Comparing a RISC and CISC with Similar Hardware Organization," Proc. of Symp. Architectual Support for Programming Languages and Operating Systems IV, pp. 310-319, 1991.
[56] R. Uhlig, D. Nagle, T. Stanley, T. Mudge, S. Sechrest and R. Brown, "Design Tradeoffs for Software-Managed TLBs," ACM Trans. Comput. Syst., vol. 12, no. 3, pp. 175-205, 1994.
[57] J. B. Chen, A. Borg and N. P. Jouppi, "A Simulation Based Study of TLB Performance," The 19th Annual Symposium on Comput. Architecture, IEEE, pp. 114-123, 1992.
[58] Brian A. Malloy, "Trace-Driven and Program-Driven Simulation: A Comparison," 5th procee. of International Workshop on Modeling and Simulation of Com. and Telecomm. Syst., pp. 395-396, January 1994.
[59] David A. Sykes and Brian A. Malloy, "The Design of Efficient Simulator for the Pentium Pro Processor," Winter Simulation Conference, December 1996.
[60] Pradip Bose and Thomas M. Conte, "Performance Analysis and Its Impact on Design," COMPUTER, vol. 31, no. 5, pp. 41-49, May 1998.
[61] Integrated Device Technology, "R3000 FAMILY ASSEMBLY PROGRAMMER'S GUIDE," IDT, August 1988.
[62] M. D. Hill and A. J. Smith, "Evaluation associativity in CPU caches," IEEE Trans. Comput., vol. 38, no. 12, pp. 1612-1630, December 1989.
[63] 迫川修一, 徳田英夫, "ユーザレベル実時間スレッドの高速化技法," 電子情報通信学会論文誌(D-1), vol. J79-D-I, no. 11, pp. 946-953, November 1996.
[64] 石井善昭, "計算機の基本方式," 共立出版株式会社, pp. 1-298, 1973.
[65] 広瀬健, 高橋延匡, 土居範久, "コンピュータソフトウェア辞典," 丸善株式会社, p. 521, 1990.
[66] 社団法人トロン協会編, "原典CTRON大系: カーネルインタフェース," pp. 1-441, December 1988.
[67] 徳田英幸, "Real-time mach:実時間マイクロカーネル," 情報処理, vol. 37, no. 12, pp. 1117-1124, 1996.
[68] Hiroyuki Tomiyama and Hiroto Yasuura, "Code Placement Techniques for Cache Miss Rate Reduction," ACM Trans. Design Automation of Electronic Syst., vol. 2, no. 4, pp. 410-429, October 1997.

第2章

[1] NEC 日本電気株式会社, "VR3000TM32ビット・マイクロプロセッサ アーキテクチャ編," pp. 1-191, 1989.
[2] NEC 日本電気株式会社, "VR3000TM32ビット・マイクロプロセッサ ハードウェア編," pp. 1-133, 1989.

[3] MIPS Computer Systems, Inc., "MIPS R4000 Microprocessor Introduction," pp. 1-64, 1991.
[4] MIPS Computer Systems, Inc., "MIPS R4000 Microprocessor User's Manual," pp. 1-593, 1991.
[5] NEC 日本電気株式会社, "VR4400TM64ビット・マイクロプロセッサ アーキテクチャ編," pp. 1-400, 1993.
[6] NEC 日本電気株式会社, "VR4400TM64ビット・マイクロプロセッサ ハードウェア編," pp. 1-134, 1993.
[7] D. W. Clark and J. S. Emer, "Performance of the VAX-11/780 translation buffer : simulation and measurement," ACM Trans. Comput. Syst., vol. 3, no. 1, pp. 31-62, February 1985.
[8] パターソン&ヘネシー,"コンピュータの構成と設計 :上巻," 日経BP社, p. 57, 1996.
[9] 杉村康, "マルチプロセッサソフトオーバヘッド評価におけるロックモデル解析の一手法," 電子情報通信学会論文誌(D-1), vol. J75-D-I, no. 10, pp. 917-925, October 1992.
[10] A. J. Smith, "Cache memories," Comput. Surv., vol. 14, no. 3, pp. 473-530, September 1982.
[11] NTT電気通信研究所, "VLSIを用いたDIPS小型システム," 通研月報, vol. 36, no. 10, 1983.
[12] 飯島順一, 町田元, "アセンブラプログラミング詳説; IBMシステム/360, 370; 上巻, 下巻," 近代科学社, 1979.
[13] 社団法人トロン協会編, "原典CTRON大系: カーネルインタフェース," pp. 1-441, December 1988.
[14] 日本デジタルイクイプメント株式会社教育部, "VAXアーキテクチャハンドブック," 共立出版株式会社, 1984.
[15] Hewlett Packard, "HP3000 and HP9000 Series Computers, Precision Architecture & Instruction Set Reference Manual," 1989.
[16] 坂村健, 榎本龍弥, "TRON 仕様に基づく32ビットマイクロプロセッサ," 情報処理, vol. 30, no. 5, pp. 565-573, 1989.
[17] 田島豊久, "トランザクション処理用のベンチマーク," 情報処理, vol. 37, no. 8, pp. 731-737, 1996.
[18] Jim Gray, "データベース・ベンチマーキング," 日経BP社, PP. 38-78, 1992.
[19] 喜連川優, 伏見信也, 田中茂, 丸山光行, "データベース処理におけるベンチマーク,"情報処理, vol. 31, no. 3, pp. 175-205, 1994.
[20] パターソン&ヘネシー,"コンピュータの構成と設計 :下巻," 日経BP社, p. 749, 1996.
[21] D. Bhandarkar and D. W. Clark, "Performance from Architecture Comparing a RISC and CISC with Similar Hardware Organization," Proc. of Symp. Architectual Support for Programming Languages and Operating Systems IV, pp. 310-319, 1991.
[22] Ashis Khan, "64ビット・アーキテクチャのR4000 を開発," NIKKEI ELECTRONICS, no. 538, pp. 171-185, October 1991.

[23] R. P. Weicker, "Dhrystone benchmark: rationale for version 2 and measurement rules," SIGPlan Notices, vol. 23, no. 8, pp. 49-62, August 1988.
[24] Integrated Device Technology, "R3000 FAMILY ASSEMBLY PROGRAMMER'S GUIDE," IDT, August 1988.
[25] R. P. Weicker, "Dhrystone benchmark: A synthetic systems programming benchmark," Commun. ACM, vol. 27, no. 10, pp. 1013-1030, October 1984.
[26] Walter J. Price, "各種ベンチマーク・テストでシステム性能を比較する," NIKKEI ELECTRONICS, no. 497, pp. 189-208, April 1990.

第3章

[1] インテルジャパン株式会社, "Pentium ファミリーディベロッパーズマニュアル下巻アーキテクチャとプログラミングマニュアル," CQ 出版社, 1995.
[2] NTT 電気通信研究所, "VLSIを用いたDIPS小型システム," 通研月報, vol. 36, no. 10, 1983.
[3] T. Horikawa, "TOPAZ: Hardware-tracer based computer performance measurement and evaluation system," NEC Res. & Develop., vol 33, no. 4, pp. 638-647, October 1992.
[4] 東洋テクニカ, "マルチロジックアナライザCLAS4000ユーザーマニュアル," pp. 1-225, 1993.
[5] Biomation Corporation, "R3000/R2000 MICROPROCESSOR ANALYSIS PACKAGE USER'S MANUAL," March 1991.
[6] NEC 日本電気株式会社, "VR4400TM64ビット・マイクロプロセッサ アーキテクチャ編," pp. 1-400, 1993.
[7] Integrated Device Technology, "R3000 FAMILY ASSEMBLY PROGRAMMER'S GUIDE," IDT, August 1988.
[8] Hewlett Packard, "a.out(4)," HP-UX Reference, vol. 3, section 4, August 1992.
[9] R. P. Weicker, "Dhrystone benchmark: A synthetic systems programming benchmark," Commun. ACM, vol. 27, no. 10, pp. 1013-1030, October 1984.
[10] R. P. Weicker, "Dhrystone benchmark: rationale for version 2 and measurement rules," SIGPlan Notices, vol. 23, no. 8, pp. 49-62, August 1988.
[11] J. B. Chen, A. Borg and N. P. Jouppi, "A Simulation Based Study of TLB Performance," The 19th Annual Symposium on Comput. Architecture, IEEE, pp. 114-123, 1992.
[12] R. Uhlig, D. Nagle, T. Stanley, T. Mudge, S. Sechrest and R. Brown, "Design Tradeoffs for Software-Managed TLBs," ACM Trans. Comput. Syst., vol. 12, no. 3, pp. 175-205, 1994.
[13] A. J. Smith, "Cache memories," Comput. Surv., vol. 14, no. 3, pp. 473-530, September 1982.

[14] 杉村康, "マルチプロセッサソフトオーバヘッド評価におけるロックモデル解析の一手法," 電子情報通信学会論文誌(D-1), vol. J75-D-I, no. 10, pp. 917-925, October 1992.
[15] NEC日本電気株式会社, "VR4400TM64ビット・マイクロプロセッサ アーキテクチャ編," pp. 1-420, 1995.
[16] Walter J. Price, "各種ベンチマーク・テストでシステム性能を比較する," NIKKEI ELECTRONICS, no. 497, pp. 189-208, April 1990.
[17] 杉村康, "RTS のメモリ使用の分散的分析手法と実施例," 情報処理学会研究報告, システムソフトウェアとオペレーティングシステム, 95-OS-70-11, pp. 81-88, August 1995.
[18] 大久保利一, 香西省治, 二神新, 大南正人, "リアルタイム性を考慮した通信処理用OSインタフェース," 電子情報通信学会論文誌(D-I), vol. J78-D-I, no. 8, pp. 687-698, August 1995.
[19] 芳西崇, 小林伸幸, 中村仁之輔, 田中豪, "リアルタイムDBMS構築技術," NTT R&D, vol. 45, no. 1, pp. 27-32, 1996.

第４章

[1] MIPS Computer Systems, Inc., "MIPS R4000 Microprocessor User's Manual," pp. 1-593, 1991.
[2] NEC 日本電気株式会社, "VR4400TM64ビット・マイクロプロセッサ アーキテクチャ編," pp. 1-420, 1995.
[3] NEC 日本電気株式会社, "VR4400TM64ビット・マイクロプロセッサ ハードウェア編," pp. 1-134, 1993.
[4] M. D. Hill and A. J. Smith, "Evaluation associativity in CPU caches," IEEE Trans. Comput., vol. 38, no. 12, pp. 1612-1630, December 1989.
[5] 喜連川優, 伏見信也, 田中茂, 丸山光行, "データベース処理におけるベンチマーク," 情報処理, vol. 31, no. 3, pp. 175-205, 1994.
[6] NTT 電気通信研究所, "VLSIを用いたDIPS小型システム," 通研月報, vol. 36, no. 10, 1983.
[7] 杉村康, "RTS のメモリ使用の分散的分析手法と実施例," 情報処理学会研究報告, システムソフトウェアとオペレーティングシステム, 95-OS-70-11, pp. 81-88, August 1995.

第５章

[1] 杉村康, "マルチプロセッサソフトオーバヘッド評価におけるロックモデル解析の一手法," 電子情報通信学会論文誌(D-1), vol. J75-D-I, no. 10, pp. 917-925, October 1992.

[2] 追川修一, 徳田英夫, "ユーザレベル実時間スレッドの高速化技法," 電子情報通信学会論文誌(D-1), vol. J79-D-I, no. 11, pp. 946-953, November 1996.
[3] 西尾信彦, 徳田英幸, "連続メディア処理のためのミドルウェアの性能評価," 情報処理学会研究報告, システムソフトウェアとオペレーティングシステム, 95-OS-69-10, pp. 55-60, June 1995.
[4] J. B. Chen, A. Borg and N. P. Jouppi, "A Simulation Based Study of TLB Performance," The 19th Annual Symposium on Comput. Architecture, IEEE, pp. 114-123, 1992.
[5] 杉村康, "RTS のメモリ使用の分散的分析手法と実施例," 情報処理学会研究報告, システムソフトウェアとオペレーティングシステム, 95-OS-70-11, pp. 81-88, August 1995.
[6] 大久保利一, 香西省治, 二神新, 大南正人, "リアルタイム性を考慮した通信処理用OSインタフェース," 電子情報通信学会論文誌(D-I), vol. J78-D-I, no. 8, pp. 687-698, August 1995.
[7] 芳西崇, 小林伸幸, 中村仁之輔, 田中豪, "リアルタイムDBMS構築技術," NTT R&D, vol. 45, no. 1, pp. 27-32, 1996.
[8] 喜連川優, 伏見信也, 田中茂, 丸山光行, "データベース処理におけるベンチマーク," 情報処理, vol. 31, no. 3, pp. 175-205, 1994.
[9] 日経BP社, "日経パソコン新語辞典 96 年版," p. 554, September 1995.
[10] 杉村康, "リアルタイムシステムにおけるRISC性能の解析," 電子情報通信学会論文誌(D-1), vol. J79-D-I, no. 2, pp. 88-100, February 1996.
[11] Pradip Bose and Thomas M. Conte, "Performance Analysis and Its Impact on Design," COMPUTER, vol. 31, no. 5, pp. 41-49, May 1998.
[12] Systems Performance Evaluation Cooperative, "SPEC Benchmark Release 1.0," SPEC Newsletter, vol. 1, no. 1, Fall 1989.
[13] Williams T., "RISC is simple, but benchmarking isn't," Comput Des, vol. 31, no. 2, pp. 67-76, 1992.
[14] 門田浩, "SPEC ベンチマーク, コンピュータをシステムレベルで評価," NIKKEI ELECTRONICS, no. 498, pp. 277-287, April 1990.
[15] Ashis Khan, "64ビット・アーキテクチャのR4000 を開発," NIKKEI ELECTRONICS, no. 538, pp. 171-185, October 1991.
[16] SGI, "O2 R10000: SPECCint95,", http://open.specbench.org/osg/cpu95/results/res98q1/cpu95-980206-02411.html, 1998.
[17] 戸室隆彦, "SPEC ベンチマーク," 情報処理, vol. 37, no. 8, pp. 718-724, August 1996.
[18] パターソン&ヘネシー, "コンピュータの構成と設計 :上巻," 日経BP社, p. 75, 1996.
[19] MIPS Computer Systems, Inc., "MIPS R4000 Microprocessor Introduction," pp. 1-64, 1991.
[20] 丸善株式会社, "情報・通信・マイクロコンピュータ辞典," p. 302, p. 446, 1986.
[21] 板垣克彦, "高機能化進むRISC," 日経産業新聞, 7月23日, 1992.

[22] 日経エレクトロニクス，日経バイト，"スーパーチップ最前線，" 日経BP出版センター，pp. 1-270, June 1995.
[23] NEC, "VR5000TM 64 ビット・マイクロプロセッサ: データ・シート," pp. 1-63, April 1997.
[24] NEC, "VR5000TM, VR10000 64 ビット・マイクロプロセッサ: データ・シート," pp. 1-67, May 1998.
[25] NEC, "VR5000TM 64 ビット・マイクロプロセッサ: ユーザズマニュアル," pp. 1-313, November 1997.
[26] NEC, "VR5000TM, VR10000TM 64 ビット・マイクロプロセッサ: 命令編," pp. 1-367, June 1998.
[27] NEC, "VR10000TM, VR12000TM 64-bit Microprocessor: User's Manual," pp. 1-394, May 1998.
[28] T. M. Conte and C. E. Gimarc, eds., "Fast Simulation of Computer Architectures," Kluwer Academic Publishers, pp. 5-85, Boston, 1995.
[29] J. Bradley Chen, David W. Wall, Anita Borg, "Software Methods for System Address Tracing: Implementation and Validation," DEC WRL Research Report 95/6, http://www.research.digital.com/wrl/techreports/abstracts/89.14.html , September 1994.
[30] インテルジャパン株式会社, "Pentium ファミリーディベロッパーズマニュアル下巻アーキテクチャとプログラミングマニュアル," CQ 出版社, 1995.
[31] Anant Agarwal, Richard L. Sites and Mark Horowitz, "ATUM: A New Technique for Capturing Address Traces Using Microcode," Proceedings of the 13th International Symp. on Computer Arch. (ISCA-14), pp. 119-127, June 1986.
[32] Real-Time and Multimedia Laboratory Home Page, http://www.cs.cmu.edu/afs/cs.cmu.edu/project/art-6/www/rtmach.html
[33] John Edwards, "The Changing Face of Freeware," COMPUTER, vol. 31, no. 10, pp. 11-13, October 1998.
[34] 横井秀明, "現代用語の基礎知識 1999," 自由国民社, p. 399, January 1999.

参 考 特 許

[1] 杉村康, "平均命令実行時間の計算方法," 特許第2087306号.
[2] 杉村康, "ロック区間の自動解析方法と平均命令実行時間の計算方法," 特許第2570018号.
[3] 杉村康, "トレース情報解析方法," 特願平5-241770号.
[4] 杉村康, 倉田正博, 古川知幸, "情報処理装置の制御方法," 特許第1475806号.
[5] 杉村康, "浮動式フットスイッチ付キーボード入力装置, 特許第1925315 号.
[6] 杉村康, "密結合マルチプロセッサにおける制御表排他制御の統計情報収集方法," 特許第2607671号.
[7] 杉村康, "補助機能の選択制御方法," 特許第2591818号.
[8] 杉村康, "マルチウィンドウ編集管理方式," 特許第2540940号.
[9] 杉村康, "システムCPU 時間の測定方法," 特願平5-320533号.
[10] 杉村康, "時間測定の補正方法," 特願平5-320534号.
[11] 杉村康, "トリッガ判定用アドレス割当方法およびトリガ判定用アドレス割当装置," 特願平7-52905号.
[12] 杉村康, "固定エリアのキャッシュ競合ミスの最適化方法," 特願平8-284201号.
[13] 杉村康, "浮動エリアのキャッシュ競合ミスの最適化方法," 特願平8-285361号.
[14] 杉村康, "エミュレータにおける全命令トレースの制御方法," 特願平10-239249 号.
[15] 杉村康, "全命令トレース時のタイマー制御方法," 特願平10-239253号.
[16] 杉村康, "全命令トレースデータの２次記憶装置への格納競合防止方法," 特願平10-239256 号.

索引

【 あ 】

空き二次キャッシュページ数 93
空き物理ページキュー 97
アーギュメント 59
アクセス 8
アドレス解決済 6
アドレス変換バッファ 5
一次キャッシュ 6
ウォームスタート 29
エミュレーション 6
エミュレーションの正常性 51
エントリ 55
エントリポイント 56
応答時間 1
応用プログラム 1
オーバヘッド 6
オフセット 57
重み計算用テーブル 94
オンデマンドページング 15

【 か 】

カウントレジスタ 65
カーネル 4
カーネルエリア 16
為替交換・中継システム 1
機械語命令 2
疑似RTS ベンチマーク 13
机上処理能力試算手法 3
犠牲未使用群 85
競合群 88
競合ミス 14
キャッシュ 3
行 3
行参照有無情報 83
空間 24
下り電文 31
群 77

群クラス 86
群番号 83
言語バインド用ルーチン 58
高速バッファ 3, 5
後半使用群 85
後半未使用群 85
固定エリア 77
コールドスタート 29

【 さ 】

サイクル 19
最適化失敗 89
最適化対象 78
最適化対象数 94
最適化不要 83
最適化L 78
最適化X 77
サービスイン 1
サービスのタイプ 1
サン社 3
参照 77
参照回数 84
参照行数 83
システム空間 24
実RTS 4
実行駆動 13
実時間サービス 1
実時間システム 2
シミュレーション 3
シミュレーション用結果ファイル 96
縮小命令セットコンピュータ 2
使用群 85
使用タイマー 65
処理トランザクション数 19
スタックエリア 57
スタックポインタ 57
スタート状態 64
ステージ 3, 18
ストール 18
スーパースカラ方式 3

スーパーパイプライン方式 3
セグメント番号 27
先頭使用群 85
前半使用群 85
前半未使用群 85
全ページ競合数 84
測定ループ 50
ソフトRTS 2

【 た 】
ダイナミックリンク 15
タイムシェアリングサービス 1
タイムシェアリングシステム 2
ダイレクトマッピング 14
タスクID 23
タスクディスパッチャ 24
単純変換 24
単純命令 9, 32
遅延スロット 47
チューニング 4
定期走行タイマー 65
定型処理 1
定量評価 5
ディスパッチ 59
デッドライン制御 2
添字一覧 87
特権命令 33
ドライストーンベンチマーク 9
トランザクション処理 76
トレースデータ 3

【 な 】
ナリファイ 8
二次キャッシュ 6
二次キャッシュ相対ページ番号 78
日本規格協会 4
上り電文 31

【 は 】
ハードウェアモニタ 49

ハードRTS 2
配置替え 97
パイプライン 3
バス 2, 106
バスマスター方式 106
光ループ 65
ヒット 3
ヒット率 3
非定型処理 1
フェッチ 23
フェッチ種別 23
複合命令 9, 33
複合命令セットコンピュータ 2
複合変換 26
物理アドレス 5
物理空間 15
物理ページ 15
浮動エリア 78
ブートモード 90
ブラックボックス 50
フリーウェア 112
プログラム駆動 13
ベクターテーブル 59
ページ 15
ページ競合数 84
ページ重複数 87
ページ属性 83
ページフォルト 15
ページ割当てアリゴリズム 15
ペナルティ 18
ベンチマーク 3
ボトルネック 2

【 ま 】
マイクロ命令 2
マシンサイクル 8
マッピング 77
マルチメディア 1
マルチメディア拡張命令 112
未使用群 85

ミス 6
ミップス社 3
無効状態 29
無効ミス 14
命令付加方式 111
メモリプール機能 15
目標競合数 84
モデル 4
モデルの妥当性 50
モデルの同一性 51

【 や 】
ユーザ空間 24
容量性ミス 14

【 ら 】
ライト 21
リード 21
リフィル 21
レジスタウィンドウ 8
連想性 14
ρ_{CPU} 19
ロード遅延 21
ロードモジュール 6
論理アドレス 5
論理空間 15
論理積競合数 95
論理ページ 15
論理和競合数 95

【 わ 】
ワイヤードレジスタ 55
ワークステーション 1
ワールドワイドウェブ 1
割込み等による擾乱 66

【 A 】
AP 1
ASID 57

【 B 】
B 23
BLKRDL 21
BR 26
BRANDL 21

【 C 】
CFWRDL 21
CFWWDL 21
CISC 2
CMIPS 19
CML バッファ 16
CPU 2
CPU 使用率 19
CTRON 1
C1TrEI 19

【 D 】
DIPS 9, 26
DPCH 24
DSB 63

【 E 】
EAMP 19
$EAMP_{bm-sr}$ 53
$EAMPPT_{m}$ 53
$EAMP1T_{r}$ 53
EI_{bm-sr} 53
EI_{cm} 53

【 F 】

【 G 】
g_c 19
g_r 46
GR 26

【 H 】

【 I 】

ISA バス 106

【 J 】
J 94
JAL 58

【 K 】
K 93
KSEG0 20
KSEG1 90

【 L 】
LOADDL 21

【 M 】
MEMRDL 21
MIPS値 18
MMX 命令 112
MUDIDL 22
M68020 10

【 N 】
nウェイ・スーパースカラ方式 109
NOP 命令 8
NT_m 53

【 O 】
ODP 15
OS 1
OS用結果ファイル 95
out of order 110
$OVFT_m$ 53

【 P 】
PAID 106
Pentium 111
printf 59
PSTF 18
PT 59
PTE 60

PTLBミス 59
PWWRDL 21

【 Q 】

【 R 】
RCEIR 18
RCPFS 19
REA1DL 21
REA2DL 21
RISC 2
RISCアクセス状態データ 5, 19
RISC/CISC実行命令数比 5
RT-Mach 112
RTS 2
RTSアドレス 56
RUNRDL 21
R10000 107
R2000 10
R3000 3
R4000 3
R5000 109

【 S 】
SP 19
SPECベンチマーク 9
SPECレシオ 107
System/370 9

【 T 】
TLB 5
TLB 不一致例外 59
TLB 変更例外 59
TLB ミス 49
TLB ミス測定用ベンチマーク 52
TLB 無効例外 59
TLB-A 62
TLB-B 62
TLB-C 62
TPS_c 19

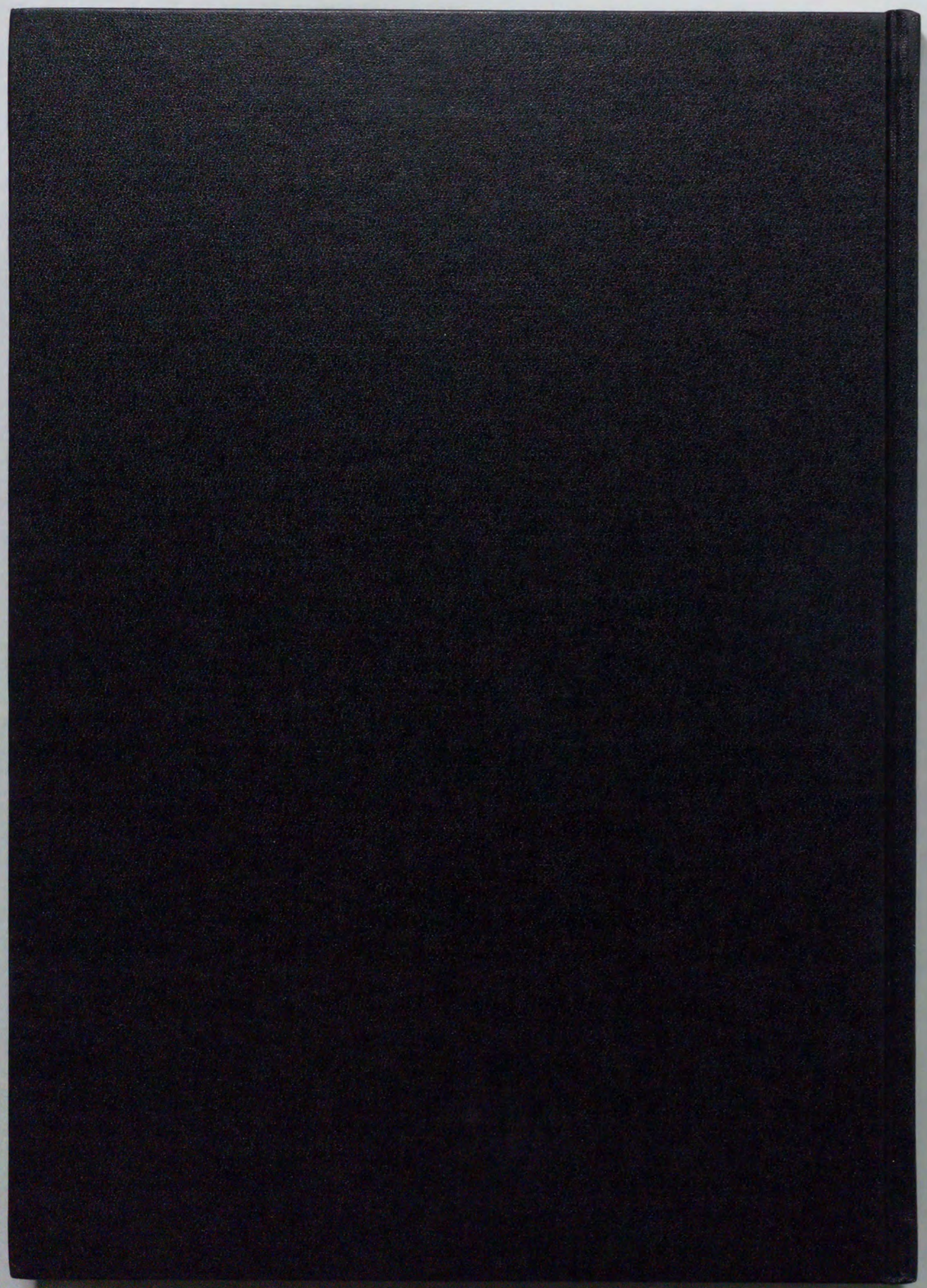
TPS_R 19
T_{real-i} 52
TR_{bm-sr} 52
TSS 2
TSS 用APベンチマーク 9
TSS-RISC/CISC 実行命令数比 5
T_{sym-i} 52
T_{sym-sr} 52
T_{tlb-i} 52

【 U 】
UNIX 1
UNIXアドレス 56
UTLBミス 59

【 V 】
VAX-MIPS 45
VAX-11 26
VAX-11/780 47

【 W 】
W 84
WB方式 20
wired 15
WRI1DL 21
WRI2DL 21
WSYIDL 22
WT方式 19

【 X 】

【 Y 】

【 Z 】
Z 85