計算機アーキテクチャ 142−17
ハイパフォーマンス
コンピューティング 85−17
（2001．3．8）

プログラム実行時間のキャッシュ容量とアクセス範囲による挙動の計測
—動的コード書換えと高精度時間計測による効率化—

永 松 礼 夫† 星 野 剛 史††

キャッシュ容量などシステム構成のパラメータを実際のプログラム実行時間より推定する場合，アクセス範囲などを変えた一連の実験のスケーラビリティから求める方式が知られている．測定用プログラムに動的コード書換えの技法を適用することと時間をナノ秒単位の精度で計測することで，測定サイクルを大幅に短縮する方式を提案する．この手法を用いることで，既存の手法と比較すると総計測時間の速度向上比が約 90 になり，測定用プログラム以外のプロセスによる実験データの変動が減少した．

Measuring Behavior of Program Execution Time effected by Installed Cache Size and Accessed Memory Range — Speedup by Dynamic Code Rewriting and High-resolution Timing Measurement —

LEO NAGAMATSU† and TAKESHI HOSHINO††

On estimating system configuration parameters such as cache size, from program execution time, it is known to use scalability behavior of program executions on various access ranges of the memory. In this paper, we propose a method to reduce cycle of such measuring by dynamic code rewriting and high (nano-second) resolution timing measurement. By this scheme, the measurement speed becomes about 90 times faster than previous one,and reduces the fluctuation of experiment results caused by other system activities.

1. はじめに

近年，プロセッサは製造技術の向上にともない，プロセスルールが格段に細線化し，プロセッサのダイサイズはそのままで，2 次キャッシュまでもがプロセッサ内に格納されるようになり，プロセッサの性能は格段に向上した．

しかし，いくらプロセッサの性能が向上したとしても，プロセッサの特徴をつかみ，各プロセッサごとにプログラムのコーディングをするのは基本的に人間の仕事である．いくら高性能なプロセッサを積んだコンピュータシステムでも，そのシステムに合致したプログラミングを行なわなければ，その処理能力は低下することは避けられない．プログラミング技法のなかでも，命令に関するキャッシュは特にプロセッサの処理能力を左右することがわかっている．[1] 特にシステムの 1 次キャッシュや 2 次キャッシュの容量をプログラム内で考慮しなければ，プログラムの処理速度は低下し，期待した性能は得られない．

本研究では，プロセッサの性能を引き出すために命令に関するキャッシュの容量をプログラムから推定する実験サイクルを動的コード書換えと高精度時間計測により効率化することを目的とする．プロセッサのカタログからだけではなく，実験からデータを得ることにより，実際のプログラム作成時の環境でプロセッサの性能を引き出すことができる．だが，このような実験はキャッシュ容量が大きくなると測定ポイントが多くなり，測定時間が大幅に増加する．本研究で適用する動的コード書換えと高精度時間計測で，このキャッシュ容量推定プログラムの実験サイクルを効率化することで，実用的なレベルでの測定時間で容量測定が行なえるとともに，このようなプログラムの細部を変更していくようなサイクルをもつ実験を効率化する手法の提案を行う．

† (nag@u-aizu.ac.jp) 会津大学 情報センター，Information Systems and Technology Center, The University of AIZU,Aizu-Wakamatsu,Fukushima,965-8580 JAPAN
†† (m5032105@u-aizu.ac.jp) 会津大学 大学院，Graduate School of Computer Science and Engineering,The University of AIZU

2. プログラム実行時間のキャッシュ容量とアクセス範囲による挙動の計測

2.1 命令に関するキャッシュ容量の測定

命令に対して行なわれるキャッシュに関するキャッシュミスは，あるブロックに対する最初の参照時に発生する**初期参照ミス**，プログラムの実行に必要な全てのブロックを収容するだけの十分な容量がキャッシュにない場合にいくつかのブロックはキャッシュから追い出される**容量性ミス**，1つのセットに対応づけられているブロック数が，そのセットの物理ブロック数を越えている場合にキャッシュから追い出される**競合性ミス**が原因となる.[2)]

本研究では，命令に関するキャッシュ容量をプログラム実行時間の変化により推定するため，容量性キャッシュミスを利用する方法を用いた.

これは**図1**に示すように，測定用プログラムのサイズを変更することで，キャッシュ内に十分格納できる場合は容量性ミスがなくなるためプログラム実行時間が短く，命令キャッシュに格納しきれない場合は容量性ミスが増加し，プログラム実行時間が増加する現象[3)]を利用してキャッシュ容量を推定する.

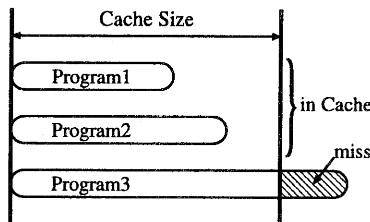

図1 命令に関するキャッシュの容量性ミス

2.2 キャッシュ容量推定プログラムの作成と実験

キャッシュ容量推定プログラムのサイズを変化させるために，スタックポインタの退避と関数からのリターン，スタックポインタの復元のみのダミー関数を用意する．**図2**に示すように，ダミー関数のリターンの直前にアセンブラ命令NOPを書き加えてコンパイルし，ダミー関数の実行時間を測定するサイクルを繰り返すシェルスクリプトを作成した．このプログラムサイズを増加させ，コンパイルを行なっていく手法（以後，再コンパイルと呼ぶ）で，命令に関するキャッシュ容量の推定を行なった.[4)]

実験には，**表1**の会津大学コンピュータ演習室のUNIXワークステーションを使用した．表1から，UltraSPARC-IIi[5)]の1次命令キャッシュ容量は

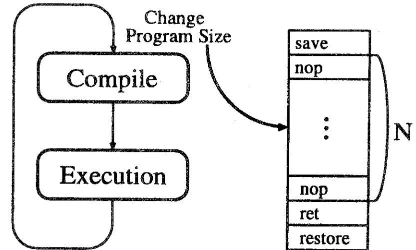

図2 再コンパイルによるプログラムサイズの変更

16Kbyte，2次キャッシュ容量は2Mbyteである.

表1 実験で使用したマシン環境

Machine	Sun Ultra 10 Workstation
CPU	Sun 64bit UltraSPARC-IIi 440 MHz
1次キャッシュ	32Kbyte (harvard) Instruction-Cache 16Kbyte Data-Cache 16Kbyte
2次キャッシュ	2Mbyte (unified)
Memory	256MB
OS	Solaris7
命令キャッシュ	
キャッシュサイズ	16Kbyte
連想方式	2way set-associative
ブロックサイズ	32byte
ライン数	256 × 2
2次キャッシュ	
キャッシュサイズ	2Mbyte
キャッシュ構成	unified
連想方式	direct map
ラインサイズ	64byte

実験では，アセンブラ命令NOP(4byte命令)を100づつ挿入することで，プログラムサイズを400byteずつ増加させている．また，キャッシュ容量推定プログラムの実行時間の測定精度がマイクロ秒単位であるため，ダミー関数を50回実行しマイクロ秒で測定できるようにした．実行結果はダミー関数を50回実行させた実行時間である.

図3に示すように，プログラムサイズを増加させていくと，サイズが16Kbyteを越えると実行速度が大幅に上昇することがわかった．これより，命令キャッシュは16Kbyteであると推測できる.

NFS(Network File System)上で実行した場合では，**図4**に示すように，実行時間にかなりの変動が発生している．ローカルな環境で実行した場合では，**図5**に示すように，実行時間に変動は少ない．これは，プログラムサイズを更新する際にNFSへのファイルの更新で遅延が生じているためである.

図5に示すように，プログラムサイズが2Mbyteを

越えると実行速度が大幅に上昇することがわかった．これより，2次キャッシュは2Mbyteであると推測できる．この実験結果より，プログラム実行に伴うプロセス(メモリへのマップ)などとのプロセス切り替えが行なわれ，実行時間に影響を与えていることがわかった．

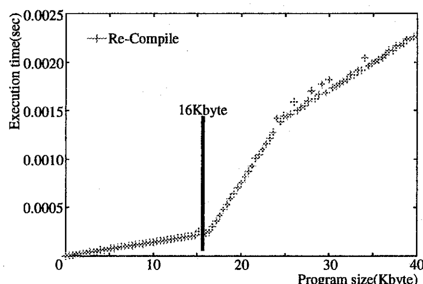

図3 再コンパイルを用いたプログラムサイズ変更による実行時間の変化（0→40Kbyte，100ポイント測定）

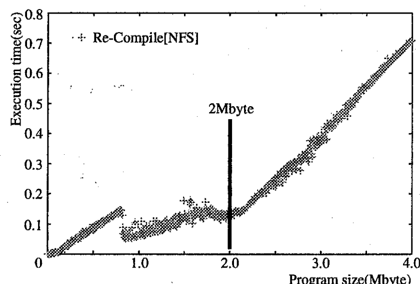

図4 再コンパイルを用いたプログラムサイズ変更による実行時間の変化（0→4Mbyte，10000ポイント測定，NFS）

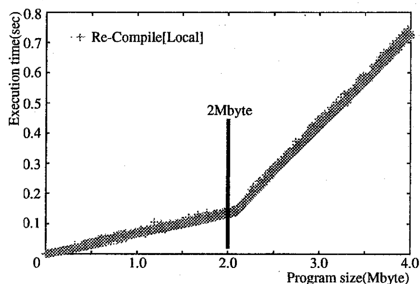

図5 再コンパイルを用いたプログラムサイズ変更による実行時間の変化（0→4Mbyte，10000ポイント測定，Local）

この再コンパイルを用いた手法方法では，NOPを書き加えプログラムサイズを変更するのにかかる時間とプログラムサイズを変更するたびにコンパイルを行なう必要があるため，**表2**に示すように一連の計測サイクルでの総計測時間が増加していく．特に4Mbyteまでプログラムサイズを増加させる実験は終了までに約30時間かかり，測定サイクルの効率化が必要であることがわかった．

表2 再コンパイルによるキャッシュ容量推定の計測サイクル時間

サイズ (byte)	0→40K	0→400K	0→4M
計測時間 (秒)	37.6	1782.0	104989.5

3. 動的コード書換えと高精度時間計測による効率化

再コンパイルを用いた手法では，測定にかかる時間が非常に長くなることと，プロセス切り替えによる遅延が実行時間へ影響を与えることが2章の実験結果からわかった．

本研究で提案している動的コード書換えでプログラムサイズの変更とコンパイルにかかる時間を短縮し，高精度時間計測で測定サイクルを短縮することでプロセス切り替えが実行時間に与える影響が少なくなり，キャッシュ容量推定プログラムによる実験が効率化すると検討した．

3.1 メモリ空間でのプログラムサイズの変更

SolarisのDynamic Linking[6]は，実行中のプログラムから共有ライブラリをプロセス空間にマップし，ライブラリ中の参照する関数のアドレスを提供し，そのアドレスを元に関数を呼び出す機能である．この機能により，メインプログラムでは実行時に動的に共有ライブラリを読み込み，共有ライブラリ内の関数を実行する．また共有ライブラリ中のデータへのアクセスも可能である．

本研究では，SolarisのDynamic Linkingを参考にしてキャッシュ容量推定プログラムのメモリ空間でのサイズの変更を行なった．Dynamic Linkingはシステムコール *mmap*()[7] を用いて実装されている．

mmap()では，メモリ空間にマップする開始アドレス，サイズ，属性などを指定し，**図6**に示すように共有ライブラリをメモリ空間上にマップを行なう．プログラム・ヘッダ・テーブル[8]から，使用する関数のアドレスへの共有ライブラリ先頭からのオフセットを取得する．そのオフセットアドレスとマップを行なう開始アドレスの合計が実行中のプロセスから共有ライブラリ内の関数のポインタとなる．

本研究では，メモリ空間での動的コード書換えの手法を用いるので，必要なメモリサイズを確保しておき，

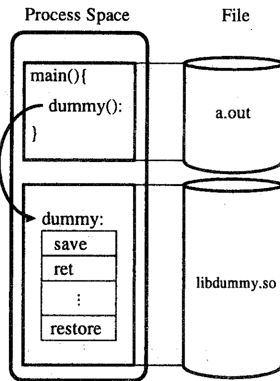

図 6　プログラムのメモリ空間へのマップ

プログラムサイズを変更しながら実験を行なう．

3.2　動的コード書換え

今回の実験では，システムコール *mmap*() を用いて，あらかじめ必要なメモリサイズを確保し，スタックポインタの退避と関数からのリターン，スタックポインタの復元のみからなるダミー関数を持つ共有ライブラリを確保したメモリ空間に書き込む初期化作業を行なう．図 7 のように，アセンブラ命令 NOP をダミー関数のリターン直前に書き加えていくことでプログラムサイズを変更を行なう．

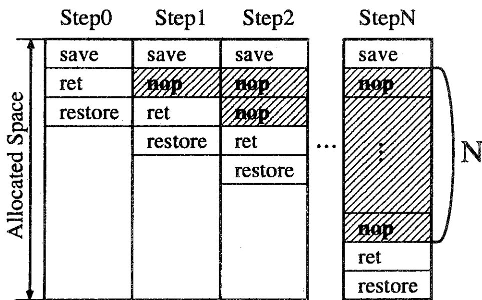

図 7　プログラムの動的コード書換え

この動的コード書換えを用いた手法では実行コードをメモリ空間で書き換えるので，再コンパイルを用いた手法でソースコードを書き換え，コンパイルを行なうことと比較して，ファイルへの書き込みとコンパイル時間を削減することができる．

また，アセンブラ命令 NOP を直接メモリ空間に書き込むので，ファイル操作によるファイルバッファのフラッシュが起こらず，プログラム実行時間への影響がないと考えられる．動的コード書換えを用いた手法では，プログラムを実行するのは最初の一回のみなので，プログラムの実行に伴うプロセス（プロセスをメモリ空間にマップを行なう等）によるプログラム実行時間への影響を受けない．

3.3　命令キャッシュのフラッシュ

最近のマイクロプロセッサと異なり，SPARC の命令キャッシュは，キャッシュされている命令がメモリ空間の命令と一貫性を保持しているかどうか検証しない．これは，メモリ空間のプログラムを書き換えても，キャッシュされている命令は変更されないことを意味する．メモリとキャッシュの一貫性が損なわれた状態で命令を実行すると，プログラムは期待した動作を行なうことが出来ない．

本研究でメモリ空間のプログラムを動的に書き換えた場合も，自動的に命令キャッシュは更新されることはない．キャッシュとの整合性を保つため，SPARC のアセンブラ命令 FLUSH を用いて命令キャッシュをフラッシュする必要がある．

3.4　高精度時間計測

2 章の再コンパイルを用いた手法での実験から，プログラムの実行時間が 10 ミリ秒以上になると実行時間に荒れが発生する現象が確認できる．実験で使用した Solaris では 10 ミリ秒単位でプロセス切り替えが行なわれている．このプロセス切り替えにより，キャッシュ容量推定プログラムとは別のプロセスとの切り替えによるプログラムの実行時間の増加が，実行時間に変動を発生させる原因である．

従来の計測精度では，測定に必要な実行時間を確保するとプロセス切り替えが発生する．本研究では，高精度時間計測を用いて測定に必要な実行時間の短縮を行なう．システムコール *gethrtime*() を用いることでナノ秒単位の高精度時間計測を行ない，測定に必要な実行時間の短縮を行なう．プログラム実行時間の短縮により，プロセス切り替えをまたぐことが少なくなるため，図 8 に示すようなプロセス切り替えによる遅延からの計測時間への影響は減少すると検討した．

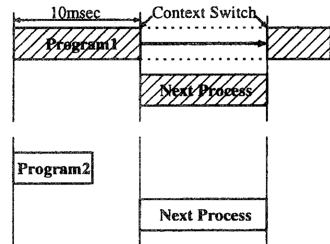

図 8　実行時間短縮によるプロセス切り替えの影響の低減

4. 動的コード書換えと高精度時間計測を適用した実験と評価

4.1 動的コード書換えと高精度時間計測を適用したキャッシュ容量推定プログラムの動作確認

動的コード書換えと高精度時間計測を用いた方法で，キャッシュ容量の推定が出来るのか動作確認を行なった．

実験ではプログラムサイズを 400byte(NOP100 個) ずつ増加させている．また，ダミー関数の前後で実行する時間測定命令がキャッシュされることで測定に影響を与えるため，ダミー関数を 5 回実行することで測定への影響を減少させている．実行結果はダミー関数を 5 回実行させた実行時間である．

図 9 に示すように，プログラムサイズが 16Kbyte でプログラム実行時間が急激に上昇しているため，1 次命令キャッシュは 16Kbyte であると推定できる．**図 10** に示すように，プログラムサイズが 2Mbyte で実行時間が急激に上昇しているため，2 次キャッシュは 2Mbyte であると推定できる．この実験結果より，動的コード書換えを用いた手法でもキャッシュ容量の推定が行なえることを確認できた．

図 10 に示すように，動的コード書換えと高精度時間計測を用いた手法では，プログラム実行時間に比較的荒れが少ない．高精度時間計測を用いた手法でも，OS による必要最低限のプロセス (割り込みなど) の動作があるために実行時間が 10 ミリ秒を超えると荒れが発生する．しかし，動的コード書換えによる手法では，プログラムを実行するのは最初の一回のみなので，NFS 上でもローカル環境でもプログラムの実行に伴うプロセスがバックグラウンドで動作する影響を受けず，再コンパイルを用いた手法のように多くの変動が発生することはない．

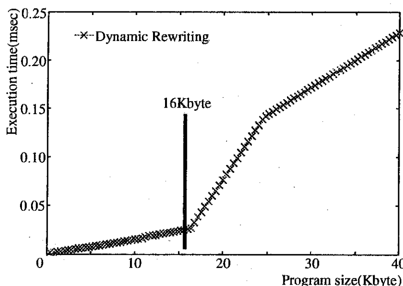

図 9 動的コード書換えを用いたプログラムサイズ変更による実行時間の変化（0→40Kbyte，100 ポイント測定）

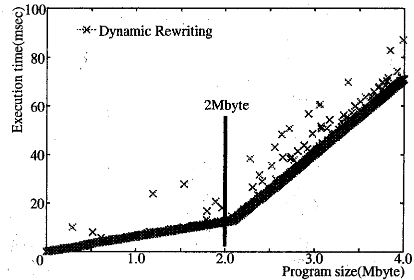

図 10 動的コード書換えを用いたプログラムサイズ変更による実行時間の変化（0→4Mbyte，10000 ポイント測定）

4.2 動的コード書換えと高精度時間計測を適用したキャッシュ容量推定プログラムの評価

動的コード書換えを用いた手法と再コンパイルによる手法とのキャッシュ容量推定プログラムの総計測時間の比較を行なった．

再コンパイルによる手法にも高精度時間計測を用い，プログラムサイズを 400byte(NOP100 個) ずつ増加させ，ダミー関数を 5 回実行した．実行結果はダミー関数を 5 回実行させた実行時間である．

表 3 に示すように，動的コード書換えと再コンパイルによる手法の総計測時間を比較した場合，4Mbyte までプログラムサイズを増加させた場合の実験終了までの時間は，再コンパイルを用いた手法は約 30 時間であったが動的コード書換えを用いた手法では約 20 分であり，速度向上比が 91.2 となった．同様に 40Kbyte までサイズを増やした場合で速度向上比が 192.0，400Kbyte で 138.1 となった．

表 3 キャッシュ容量推定プログラムの総計測時間 (秒)

変更サイズ (byte)	0→40K	0→400K	0→4M
再コンパイル	37.6	1782.0	104989.5
動的コード書換え	0.2	12.9	1150.9
速度向上比	192.0	138.1	91.2

次に，動的コード書換えによる速度向上を純粋に比較するために，測定点 40Kbyte，400Kbyte，4Mbyte において計測し，動的コード書換えによるプログラムサイズ変更時間の速度向上を調べた．**表 4** に示すように，動的コード書換えを用いた手法では，あらかじめ必要なメモリサイズを確保し，ダミー関数を書き込む初期化と命令キャッシュのフラッシュが必要であるが，それは再コンパイルでのプログラムサイズの変更時間と比べて非常に小さく，一連の計測サイクルでの計測時間に影響を与えるレベルではなかった．再コンパイルによる手法でのプログラムサイズの変更とコンパ

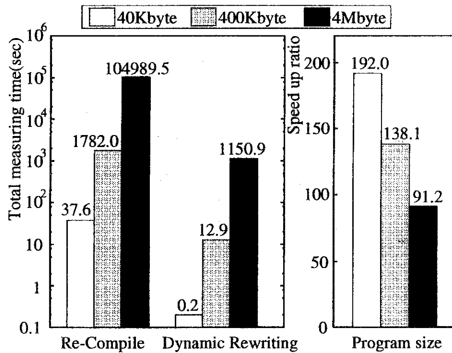

図 11 動的コード書換えと高精度時間計測による効率化
(キャッシュ容量推定プログラムの総計測時間:秒)

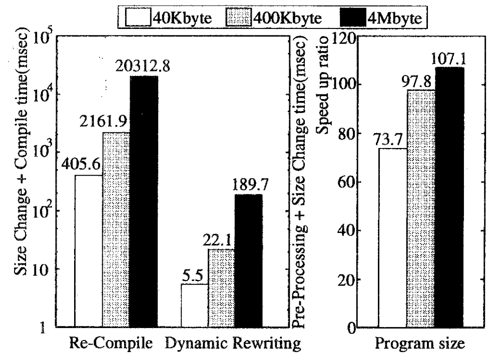

図 12 動的コード書換えと高精度時間計測による効率化
(キャッシュ容量推定プログラムのサイズ変更時間:ミリ秒)

イルにかかる時間と，動的コード書換えを用いた手法で初期化とプログラムサイズの変更と命令キャッシュのフラッシュにかかる時間を比較した場合，40Kbyteで速度向上比が 73.7，400Kbyte で 97.8，4Mbyte で 107.1 となった．この実験結果より，一連の計測サイクルでの総計測時間の速度向上比が低くなるのは，測定ポイントが多くなるとプログラムの開始と終了にかかる時間が与える影響が低くくなり，本来の速度向上比に近づくためであることがわかった．

表 4 測定点 40Kbyte，400Kbyte，4Mbyte での
プログラムサイズ変更時間と実行時間（ミリ秒）

プログラムサイズ (byte)	40K	400K	4M
再コンパイル			
プログラムサイズ変更時間	405.6	2161.9	20312.8
実行時間	0.2	2.3	71.2
動的コード書換え			
初期化時間	2.4	2.4	2.4
プログラムサイズ変更時間	2.3	18.9	186.5
キャッシュフラッシュ時間	0.8	0.8	0.8
実行時間	0.2	2.3	71.2
速度向上比 (サイズ変更)	73.7	97.8	107.1

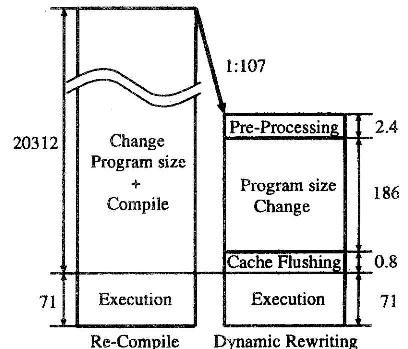

図 13 動的コード書換えと高精度時間計測による効率化
(測定点 4Mbyte でのサイズ変更時間:ミリ秒)

5. まとめ

本論文では，測定用プログラムに動的コード書換えと高精度時間計測を適用する手法を提案した．この手法を用いることにより，再コンパイルを適用した方法と比較して総計測時間の速度向上比が 90 から 190 になり，プログラムサイズの変更にかかる時間の速度向上比は 70 から 100 となった．また，プロセス切り替えによる実行時間への影響を低減させることができた．この手法を用いることで，プログラムを最適に動作させることに必要なシステム構成のパラメータを，プロセスを稼働させながら短時間に取得できる．

参考文献

1) J. Torrellas, C. Xia, and R. Daigle: Optimizing the Instruction Cache Performance of the Operating System, IEEE Transactions on Computers,vol.47,no.12,pp.1363-1381,1998.
2) D.A. Patterson, J.L. Hennessy: Computer Organization & Design The Hardware/Software Interface,Morgan Kaufmann, 1986.
3) J.J. Dongarra and A.R. Hinds: Unrolling Loops in FORTRAN, Software-Practice and Experience, vol.9,pp.219-226,1979.
4) 永松礼夫, 古市実裕, 出口光一郎: 倍の計算は倍の時間で済むか, 情報処理学会 第 37 回プログラムシンポジウム報告書,pp.91-96.1996.
5) Sun Microsystems: UltraSPARC IIi,Technical White Paper,1998.
6) SunSoft Developer Engineering: Solaris Porting Guide Second Edition,Prentice Hall,1995.
7) W.R. Stevens: Advanced Programming in the UNIX Environment,Addison-Wesley,1992.
8) S.D. Pate: UNIX Internals A Practical Approach,Addison-Wesley,1996.