計算機アーキテクチャ 123－12
（1997. 3. 7）

インスツルメンテーションに基づく性能評価環境の高精度化

久保田 和人† 板倉 憲一††
佐藤 三久† 朴 泰祐††

インスツルメンテーションによってプログラムの挙動を解析するツール ExCit(ExC instrument tool) を構築している。インスツルメンテーションによる解析は、シミュレータによる解析に較べ高速であるが精度は劣る。本稿では、ExCit による実行時間予測の精度について評価する。精度を上げるためにソースコードを基本ブロックに分け、その基本ブロックを静的に解析した結果を利用している。linpack ベンチマークでは、ほぼ正しく実行時間を予測できたが、specfp92 ベンチマークでは問題によっては精度が上がらなかった。これらの理由について考察し、今後の課題を述べる。

Accurate performance analysis based on code instrumentation

KAZUTO KUBOTA ,† KEN'ICHI ITAKURA ,†† MITSUHISA SATO † and TAISUKE BOKU††

ExCit is a code instrumention tool for program behavior analysis. Code instrumentation tool works faster than simulator, but the measurement result is less accurate. So we analyze basic blocks of the target program in order to increase the accuracy of the execution time prediction. We report an experimental results of execution time prediction of ExCit using linpack and specfp92 benchmarks. ExCit can generate reasonable results in linpack, but can not predict exection time accurately in some benchmarks in specfp92. We discuss the reasons and present the future works.

1. まえがき

プログラムの挙動の正確な理解は、プログラムのチューニングやコンパイラや OS の開発、計算機アーキテクチャの設計をする上で必要不可欠である。通常はシミュレータを利用した解析が行われることが多いが、処理速度に問題があるため大規模なプログラムに適用するのは困難である。我々は、コードインスツルメントを行ってプログラム実行時に情報を集め、プログラムの性能評価を行う方法に着目している。この方法は、シミュレータに比べて高速であるが精度が問題となる。一般に、精度と実行時間はトレードオフの関係にある。本稿では、インスツルメントに基づく性能評価ツールの精度について評価し、その実用性を明らかにする。

現在、我々はアセンブラレベルでコードをインスツルメントするツール ExCit を開発中である。評価項目としては、プログラム実行時間の予測精度をとりあげた。ExCit では、インストラクションのアドレストレースやデータ領域のメモリアクセスパタンといった実行時間予測のための基本的なデータをプログラム実行時に得ることができる。実行時間の予測にあたっては、精度を高めるためにソースコードを基本ブロックに分け、その基本ブロックを静的に解析した結果を利用している。対象プログラムには、linpack[12]ベンチマークと specfp92[11]ベンチマークを使用した。

本稿は以下の構成をとる。第 2 節では、インスツルメントツールについて述べる。第 3 節では、ExCit の構成と実行時間予測の方法について述べる。第 4 節では、ベンチマークプログラムを用いた実験およびその結果の解析を行う。第 5 節では、考察を行う。第 6 節では、関連研究について述べる。第 7 節では、まとめと今後の課題について述べる。

2. インスツルメンテーションツール

インスツルメンテーションツールは、ソースコードに付加的に情報取得の為のコードを埋め込んで、プロ

† 新情報処理開発機構 超並列パフォーマンス研究室
Massively Parallel Computing Laboratory, Real World Computing Partnership
†† 筑波大学 電子・情報工学系
Institute of Information Sciences and Electronics, University of Tsukuba

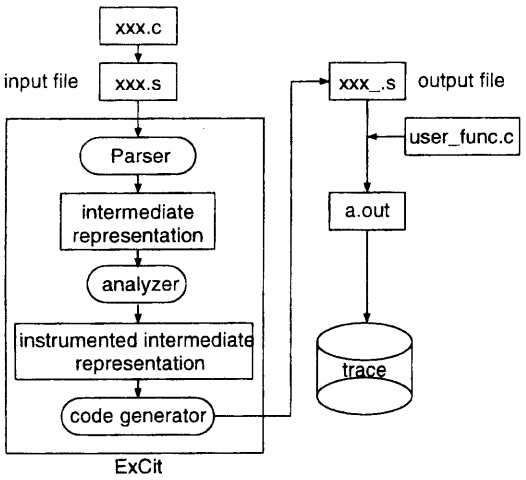

図1　ExCit の基本構成

グラム実行時にデータを収集するツールである。コード挿入によるプローブ効果のため精度はシミュレータに劣るが、オブジェクトコードがダイレクトに実行されるので高速である。よって、プログラム全体にわたる情報収集を行うことが可能となる。

インスツルメンテーションツールでは様々な情報が収集できる。

- メモリアクセスの情報収集により、キャッシュのヒット率を知ることができる。
- インストラクションのアドレストレースから、gprof などで実現されているレベルよりも、さらに細かいレベルでのプログラムの実行状況を知ることができる。
- オリジナルコードの解析情報を利用すれば、実行時間の予測も可能となる。

これら収集した情報は、プログラムのデバッグやチューニングだけでなく、コンパイラやアーキテクチャ設計にも利用できる。

この他、プログラムを細分化し、個々の部分の正確な処理時間を把握する技術はリアルタイム処理へと応用することが可能である[3]。

なお、プログラムの実行時の情報を収集する方法は、この他に、実行時にコードを書換えトラップを利用する方法[7]や、タイマ割り込みを利用する方法がある。

3. ExCit による実行時間予測

3.1 ExCit の構成

ExCit は、パーザー部と解析部、コード生成部からなる。図 1は、C 言語で記述された `xxx.c` というファイルに対して、ExCit によるコードの解析およびインスツルメントが行われ、出力ファイル `xxx_.s` が生成される様子を示している。`xxx_.s` は、ユーザ関数 `user_func.c` とリンクされ、実行されることでトレースファイルを生成する。

パーザー部は、SparcV8[1]のコードを入力し中間構造を生成する。これは、SparcV8 のシンタックスとは独立した構造を持つ。解析部は、中間構造を解析しインスツルメントのための情報を挿入する。現在行っている解析は、load/store 命令および基本ブロックの抽出である。

コード生成部は中間構造に挿入された情報を元に、情報収集のためのコードを挿入した SparcV8 コードを生成する。生成されたコードには、情報収集地点においてユーザ関数を呼び出す命令が埋め込まれている。load/store 命令に対してはメモリのアドレスが、基本ブロックに関しては基本ブロックに対して付けられた固有の番号が、ユーザ関数の引数となる。

ユーザ関数は、採取されたデータを処理するための関数である。通常は、引数を外部ファイル出力するプログラムが入っておりトレースデータが得られる。トレースデータが膨大な量になる場合は、ユーザがここでデータ処理を行うことでトレースファイルを作らずにすむ。第 4 節に挙げる例では、キャッシュシミュレーションと実行時間の計算をユーザ関数内で行っている。基本ブロックの番号はデータベースで管理されており、複数のファイル間で同一の基本ブロック番号が付けられるのを防いでいる。

3.2 実行時間予測

プログラムの実行時間は、個々の基本ブロックの処理に要する時間を積算することで算出する。ある基本ブロックの処理時間は、データの内容やキャッシュの状態によって毎回異なってくる。命令がキャッシュから外れていた場合は、そのペナルティが処理時間に影響を与える。load 命令は、キャッシュの hit/miss hit によってサイクル数が変化し、また浮動小数点演算命令はデータの値そのものによってサイクル数が変化する。ある i 番目の基本ブロックの典型的な処理時間 tb_typ_i を以下の場合の処理時間と定める。

- 命令キャッシュは全て hit。
- データキャッシュは全て hit。
- 浮動小数点演算命令は、全て典型的な場合のサイクル数で計算が終了するものとする。
- 例外処理は起こらないものとする。

全体の処理時間 T は、以下の式で近似する。

$$T = \sum_i (tb\_typ_i * num_i) + \sum_{all} tdl_{miss}$$

ここで、num_i は i 番目の基本ブロックを通過した回数であり、tdl_{miss} は、load 命令においてデータキャッシュがミスとなった場合のペナルティである。すなわち、処理時間の合計はすべての基本ブロックが典型的な時間で処理を終えた場合に、データキャッシュの

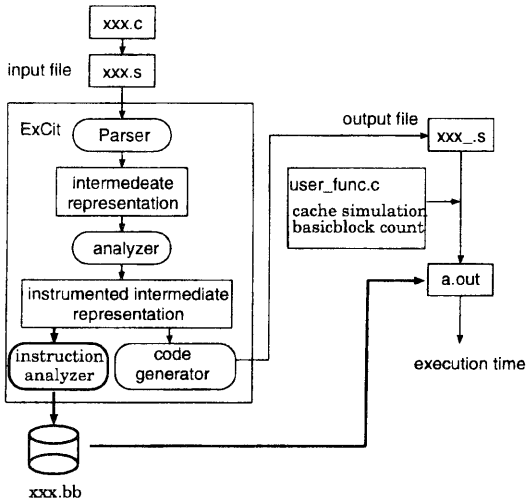

図2　ExCitを用いた時間予測

ミスのペナルティの合計を加えたものであるとする。num_iは、インスツルメントされたコードを実行することによって得られる基本ブロックの通過情報を集計することで得られる。tdl_{miss}は、load/storeに関する情報をキャッシュシミュレータにかけることで得られる。tb_typ_iは、インストラクションを静的に解析することで求める。この方法は、プログラムを実行開始から終了までシミュレートする場合と較べて、短い時間で全体の処理時間を算出することができる。

ExCitを用いた実行時間予測の様子を図2に示す。ExCitの部分には、インストラクションアナライザーが追加されている。これは、各基本ブロックの所用サイクル数を計算し出力ファイル **xxx.bb** に出力する。ユーザ関数内には、基本ブロック集計ルーチンとキャッシュシミュレータが用意されている。実行モジュール **a.out** は、**xxx.bb** のデータとインスツルメントされたコードによって生成される基本ブロックの通過状況から基本ブロックの処理に要する時間を算出する。またload/storeがアクセスするメモリアドレスの情報はキャッシュシミュレータにかけられ、loadのキャッシュミスのペナルティが算出される。最後にこれらを集計することで実行時間の予測値が算出される。

基本ブロックの処理に要する時間を一意に決定することは難しい。図3のコードは、最後の命令が **faddd** で終わっている。この命令のレイテンシは、通常5クロックである。後続命令が浮動小数点演算命令ならば、**faddd** は、5クロック要すると考えてよい。一方、後続の4命令が整数命令ならば、これらの命令は依存関係なく実行されるので **faddd** は、実質上1クロックで動作するものとみなせる。同様なことが先行命令に関しても言える。結局のところ、基本ブロックの処理サイクル数は、前後の基本ブロックの状況がわからないと決定できない。ここでは、二つの方法で計算を行うことにする。前提として、計算に先立ち全てのリソースは解放されているものとする。第一の方法は、最後のインストラクションがフェッチされるまでのサイクル数を基本ブロックの処理サイクル数とする方法である。第二の方法は、最後のインストラクションの実行が終わるまでのサイクル数を基本ブロックの処理サイクル数とする方法である。次節の評価では、これら二つについて予測を行っている。

```
add %o1,3,%o1
sub %o2,3,%o2
bne L13
faddd %f0,%f2,%f4
```

図3　予測が困難な命令列

4. ベンチマークによる評価

linpack[12]ベンチマークおよびspecfp92[11]ベンチマークプログラムの実行時間を予測した結果を報告する。対象計算機はSun Sparc Station 5である。仕様を表4に示す。

表1　Sparc Station 5の仕様

CPU	clock	L1 D-cache	L2 D-cache
MicroSparc-II	85MHz	8Kbyte	なし

4.1 linpack

linpackプログラムから一部のルーチンを抜きだし、実行時間の実測とExCitによる予測を行った。結果を表2に示す。使用したコンパイラは、**gcc** で、コンパイルオプションは**-msupersparc -O** である。浮動小数点演算の乗算および除算は、そのままSparcV8のインストラクションが出力されるのでライブラリのコールはない。トレースを取ったと仮定すると、そのサイズは約14Mbyteである。予測値1は第3節で述べた第一の方法であり、基本ブロックの処理時間を最後のインストラクションのフェッチが終了した時点のサイクル数とした場合である。予測値2は第3節の述べた第二の方法であり、最後のインストラクションが終了した時点のサイクル数とした場合である。total timeに幅があるのはキャッシュミスのペナルティの与え方による。外部RAMへのアクセスがページモードで行われた場合、ペナルティは7サイクルとなる。一方、ノンページモードで行われた場合は、ペナルティは14サイクルとなる。7と14は実測して求めた。実測時間は精度を上げるために、OS上でハードウェアインタラプトを止めてからプログラムを実行している。これは、プログラムそのものをデバイスドライバとして組み込み、カーネルモードで実行することによって実

表2　ExCit による linpack の予測時間

	L1 miss	hit ratio	instructions	tick	total time(msec)
実測値	–	–	–	–	110.976
予測値1	203554	75.64%	4150495	6762620	96.32～113.08
予測値2	↑	↑	↑	7323603	102.92～119.68

現している。タイマは、0.5μsec の分解能を持つハードウェアタイマを使用している。これによって実測値のデータの精度は高い物となっている。結果を比較すると、実測時間は予測値1、予測値2とも予測範囲の中に入っているので、linpack に関してはほぼ正確な予測が行われているといえる。

予測値1と予測値2では、予測結果が約5%違う。これは以下の理由による。表3は、各基本ブロックの要した処理時間を大きい順に並べたものである。113番の基本ブロックの処理が全体の64%を占めている事がわかる。この基本ブロックの処理時間の見積もりが1サイクル違うと、全体の処理時間の見積もりが数%異ってくる。繰り返し集中して実行される基本ブロックについては、前後の基本ブロックを考慮しつつ正確な予測値を立てることが重要である。

4.2 specfp92

specfp92 ベンチマークの処理時間と予測時間を表4に示す。また、実測時間に対する予測時間の割合を図4に示す。予測時間は、基本ブロックのサイクル数の見積もり方、キャッシュミスペナルティの与え方によって4通りの値が出る。ここでは、その中の最小のものと最大のものを示してある。accuracy は、予測時間を実測時間で割ったものである。使用したコンパイラは、Sparc Fortran 3.0.1 および acc 3.0.1 であり、コンパイルオプションは、それぞれ`-fast -O4 -cg92 -sun4`、`-fast -O4 -cg92 -target sun4`である。浮動小数点の乗算、除算はそのまま SparcV8 のコードが出るが、exp や sin といった浮動小数点演算は標準ライブラリのコールとなる。この部分にはインスツルメントが行えないので、プロファイラで計測した値を使用した。実測値は/bin/time を用いて計測した。

実験結果によると、予測値の範囲が広い物がある。これは二つの理由による。第一の理由は、キャッシュミスのペナルティの付け方によるものである。013.spice2g6 や 094.nasa7 がこれにあたる。これらのプログラムは、実行時間に占めるキャッシュミスの処理時間が大きい。両者ともに、全体の処理時間の 40%から 60%がキャッシュミスによるストール時間である。したがって、キャッシュミスのストール時間を7サイクルで見積もるか、14サイクルで見積もるのかで、全体の処理時間の予測値が大きく変わって来る。この部分になんらかの指針を立てる必要がある。

第二の理由は、基本ブロックの処理サイクル数の予測の差によるものである。048.ora がこれにあたる。このプログラムの特徴は、処理が集中する基本ブロックの最後にレイテンシの大きな浮動小数点演算が含まれているということである。通過回数が最大である基本ブロックの一つは、`fsqrtd` 命令で終わっているため予測値の差が 64clock である。したがって、このような基本ブロックに対しては、前後の基本ブロックを考慮したサイクル数の算出が必要である。

全体を見ると、ほぼ実行時間を予測できたものから実測値の半分程度の時間に予測してしまったものがある。これには、いくつかの要因が考えられる。

- インストラクションのキャッシュミスが起きている。
- 割り込みによってデータキャッシュの中身が破壊されている。
- TLB ミスが起きている。
- レジスタウインドウのオーバフローが起きている。

今後は、これらの影響を考慮に入れた時間予測を行う予定である。

5. 考　　察

基本ブロックの解析結果から全体の処理時間を算出する方法は、高速であるという利点を持つ反面、様々な問題点がある。

まず、基本ブロック間の影響を考えられないということがあげられる。基本ブロックの処理時間は、先行する基本ブロックの命令によって影響を受け、また後続の基本ブロックの処理時間に影響を与える。ここで、ある基本ブロックから見れば、先行する基本ブロックの種類は高々2個であり、後続の基本ブロックの種類も2個である。したがって、前後の基本ブロックまで含めた解析を行い、インスツルメントによって得られる基本ブロックの通過状況を考慮しつつ基本ブロックの処理時間を決めていけば、現在より高い精度で実行時間を予測できると考えられる。

次に、プロセッサアーキテクチャが異った場合に予測精度が下がる可能性があるという問題点がある。現在の実行時間予測は、load 命令がキャッシュミスを起こすとパイプラインがストールし後続命令は実行されないという仮定を置いている。MicroSparc-II[2]ではこの仮定は成り立つが、他の CPU では成り立たない。他の CPU では、メモリアクセス命令がキャッシュミスを起こしても他の命令が実行されミスペナルティが隠される場合もある。メモリアクセスの情報を基本ブロック処理時間に何等かの形で反映させなければ、他の CPU で処理時間の予測は困難である。ペナルティが隠れる CPU を用いて、この部分の誤差がどのくら

表3　ExCit による linpack の予測時間

基本ブロック番号	予測サイクル数1	予測サイクル数2	通過回数	処理サイクル数1	処理サイクル数2	比率
-	-	-	-	6762620	7323603	-
113	13	14	338160	4396080	4734240	64%
36	58	60	20000	1200000	1160000	18%
34	21	21	20000	420000	420000	6%
48	11	12	20000	240000	220000	3%

表4　ExCit による specfp92 の予測時間

	excit(a)(sec)	libm(b)(sec)	(a)+(b)(sec)	実測時間 (sec)	accuracy(%)	L1-hit(%)
013.spice2g6	361.02～499.50	30.27	391.29～529.78	612.6	63.87～86.48	67
015.doduc	23.67～30.08	8.75	32.42～38.83	50.8	63.81～76.43	90
034.mdljdp2	70.54～97.91	0.01	70.55～97.92	122.3	57.68～80.06	85
039.wave5	62.80～76.54	8.42	71.22～84.97	123.0	57.90～69.08	88
047.tomcatv	31.13～38.86	0.00	31.13～38.86	48.6	64.05～79.95	75
048.ora	45.08～88.64	0.01	45.09～88.65	93.8	48.07～94.50	99
052.alvinn	56.75～78.14	0.00	56.75～78.14	77.6	73.13～100.69	80
056.ear	234.39～261.52	0.00	234.39～261.52	335.9	69.77～77.85	96
077.mdljsp2	60.43～84.72	0.03	60.45～84.75	80.3	75.28～10.554	92
078.swm256	261.02～298.94	0.00	261.02～298.94	312.9	83.41～95.53	84
089.su2cor	119.99～166.54	26.44	146.43～192.99	220.9	66.28～87.36	52
090.hydro2d	150.63～227.09	0.07	150.70～227.16	259.6	58.05～87.50	72
093.nasa7	194.01～279.11	19.22	213.23～298.34	311.7	68.40～95.71	52
094.fpppp	86.54～103.13	5.91	92.45～109.04	193.7	47.72～56.29	90

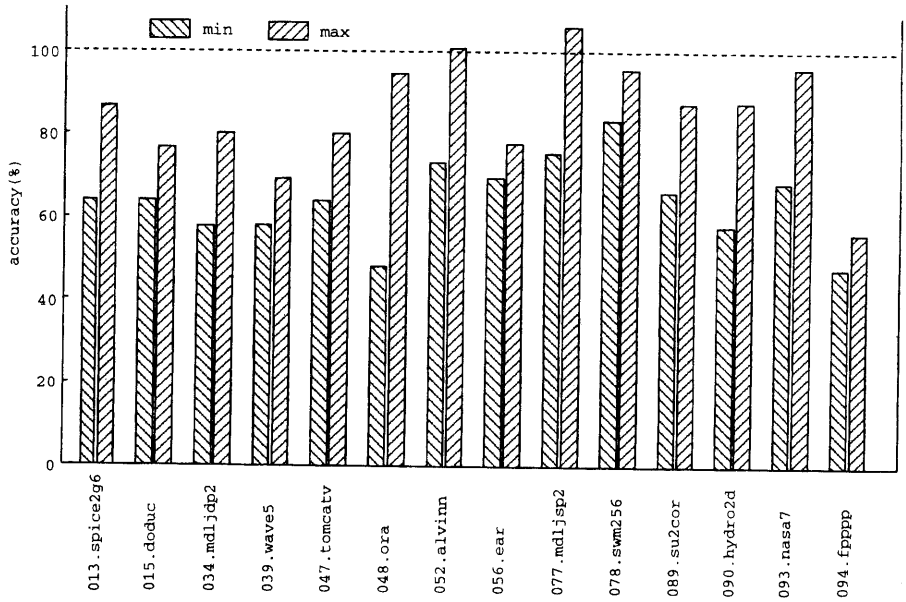

図4　予測値と実測値の割合

い予測時間に現れるのかを調べる必要がある。

また、現在のシステムはソースコードがないとインスツルメントを行うことが出来ない。specfp92 ベンチマークでは、様々な浮動小数点算術関数について情報を得ることができなかった。オブジェクトファイルレベルから情報を得られるシステムに拡張することが望ましい。

予測に要した時間は、Sun Sparc Station20 を利用したところ実測時間の 7.2 倍～38 倍であった。Sun Sparc Station5 を利用した場合は、予測にかかる時間

はさらに大きいものとなる。現在、予測時間を短縮する工夫は特別に行っていないので、将来的には予測に必要な時間は短縮できると考えている。

6. 関連研究

これまで様々なインスツルメントツールが開発されている。Spa[7]は、プログラムの性能評価を行うためのツールであり、トラップを利用したデータ収集を行っている。バイナリの実行ファイルを入力すると、インストラクションのカウントが得られる。Tango Lite[4]は、共有メモリマルチプロセッサの性能評価および、プログラムボトルネックの発見を目的としたツールである。Talisman[9]は、Meerkatというシステムのシミュレータであり、台数を増やした時の挙動を知るためのツールである。ユーザモードに加えてシステムモードの処理に関しても挙動を知ることができる。例外にも対応している。予測結果と実行結果を較べ、モデルをチューニングすることで精度を上げている。SimICS[6]は、プロファイリング、メモリシミュレーション、デバッグを行えるツールである。キャッシュミスだけではなく、TLBミスまで調べることができる。EEL[5]は、インスツルメント用のコードを挿入する際に、スーパースカラの利点を生かし、なるべくサイクル数が増加しないようにするという方式に特徴がある。実行ファイルを直接ディスアセンブルしている。Shade[8]は、プログラムのシミュレーションやメモリアドレストレースの取得、デバッグ、マイクロプロセッサのパイプラインの性能を知るといった多用途のツールである。ExCitは、インスツルメントによって様々なデータを取得し、解析するためのツールである。高速性と精度の高さのトレードオフを狙っている。

7. あとがき

本稿では、インスツルメントツールを用いて実行時間の予測を行った場合の精度および問題点について述べた。ベンチマークプログラムを用いた実験では、いくつかの問題に関しては精度よく実行時間が予測できたが、精度が上がらなかったものもあった。これらについては、その理由について考察した。

今後は、まずspecfp92で予測がうまくいかなかった問題に関して、その原因を突き止め改良を図る予定である。また、複雑なCPUアーキテクチャやメモリアーキテクチャを持つマシンについて本方式を適用し、どの程度の精度が得られるのか調べる予定である。ExCitでは、基本ブロックやload/storeだけでなく、様々なプログラムの情報を集めることが可能である。これらの情報からプログラムの特徴抽出を行い、プログラムのチューニングやデバッグ、さらに、コンパイラへのフィードバックについても考えていきたい。

現在、我々は大規模並列プログラムのシミュレーションシステムの構築を計画している。これは、インスツルメントツールを利用して自分自身の実行時間を計算しながら動作する要素プログラム間を、ネットワークシミュレータ(INSPIRE[10])で結合しようというものである。

謝　辞

本研究に対して議論に参加頂き，多くのアドバイスを頂きました新情報処理開発機構超並列ソフトウェア研究室、同超並列パフォーマンス研究室の皆様に感謝致します.

参考文献

1) SPARC International, Inc.,"SPARC アーキテクチャマニュアル バージョン 8", トッパン, 1992.
2) "STP1012PGA microSPARC-II User's Manual rev1.1,"Sun Microelectronics, 1994.
3) Chang Yun Park and Alan C. Shaw, "Experiments with a Program Timing Tool Based on Source-Level Timing Schema," IEEE Coumputer, pp. 48-57, May 1991.
4) Stephen Alan Herrod, "Tango Lite: A Multiprocessor Simulation Environment, Introduction and User's Guide," http://www-flash.stanford.edu/ herrod/docs/tango-lite.ps.
5) Eric Schnarr and James R.Larus, "Instruction Scheduling and Executable Editing,"ftp://ftp.cs.wisc.edu/wwt/pldi95_eel.ps.
6) http://www.sics.se/simics/
7) http://www.cs.washington.edu:80/homes/pardo/sim.d/index1.d/Index.html#Tool-Spa
8) Robert F. Cmelik, and David Keppel, "Shade: A Fast Instruction-Set Simulator for Execution Profiling," Proceedings of the 1994 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, pp. 128-137, 1994.
9) Robert C. Bedichek,"Talisman: Fast and Accurate Multicomputer Simulation," Proceedings of the 1995 ACM SIGMETRICS Conference on Modeling and Measurement of Computer Systems",pp. 14-24,1995.
10) 原田, 曽根, 朴, 中村, 中澤, "並列処理ネットワークのための性能評価用シミュレータ生成系INSPIRE", 情処研報, 95-ARC-113, pp. 65-72, 1995.
11) http://www.spec.org
12) J. J. Dongarra, et. al., "LINPACK User's Guide," SIAM, Philadelphia, PA, 1979.