計算機アーキテクチャ 146−12
（２００２．２．１）

0/1 の局所性を利用したデータ値予測機構のハードウエア量削減

佐藤 寿倫[1,2]　　有田 五次郎[1]

[1] 九州工業大学 情報工学部 知能情報工学科

[2] 九州工業大学 マイクロ化総合技術センター

近年値予測を用いたデータ依存の投機的実行が注目されているが、値予測のためのハードウエア量が問題となっている。本稿では、値予測機構のハードウエア量を削減することを検討している。具体的には、頻繁な値の局所性に着目し、予測値を０と１だけに限定している。SPECint95 ベンチマークではレジスタに書き込まれる値の平均で20%以上が０と１で占められるので、予測値を制限しても有意義なパフォーマンスが得られると予想される。シミュレーションの結果、提案する予測器は最終値型予測器よりもハードウエアの利用効率が良いことが確認された.

Reducing Hardware Budget of Data Value Predictors by Exploiting Locality on 1-bit Values

TOSHINORI SATO　　ITSUJIRO ARITA

KYUSHU INSTITUTE OF TECHNOLOGY

Recently, the practice of speculation in resolving data dependences has been studied as a means of extracting more instruction level parallelism. One of the serious hurdles for realizing data speculation is the huge hardware budget of the predictors. In this paper, we propose techniques that reduce the budget significantly by exploiting frequent value locality. Based on the proposals, we evaluate a value predictor, named *0/1-value predictor*. The 0/1-value predictor generates only values 0 and 1. Simulation results show that the proposed predictor has better performance than the last-value predictor which is twice as large in hardware budget as is the predictors. Therefore, the 0/1-value predictors is one of the promising candidates for cost-effective and practical value predictors which can be implemented in real microprocessors.

1. はじめに

現在主流のマイクロプロセッサは性能向上を命令レベルの並列性 (ILP: instruction level parallelism) の抽出に頼っている。しかしながら、ILP は命令間の制御依存，資源競合，データ依存関係によって制限される。このうち、近年データ依存を投機的に解消する方式が注目されている。値予測に基づいたデータ投機実行[7),8)]は、命令間の依存関係を緩和し命令レベルの並列性を抽出する新たなパラダイムとして注目されている。ある命令の結果は値予測機構を用いて予測され、この命令と依存関係にある命令を同時に実行でき、ILP を積極的に抽出できるわけである。ここで注意しなければならないのは、間違った投機によるペナルティを被ること無く値予測を有効に利用するためには、高い予測精度が必要になるということである。ハイブリッド値予測機構[15)]やコンテキスト型値予測機構[13)]は非常に高い予測精度を達成できるが、ハードウエアコストも非常に大きくなってしまう。近年ハードウエア量削減に対する興味が高まりいくつかの研究がなされている[3)~6),9),10),14)]。われわれもこれまでに

様々な検討を行ってきた[1),11),12)]が、本稿ではそれらに続く検討結果を報告する。

2. 評価方法

本節では、シミュレーションに用いたプロセッサのモデルとベンチマークプログラムを紹介し、提案手法の評価環境について説明する。

2.1 プロセッサモデル

性能評価に用いられるプロセッサモデルはレジスタ更新ユニット (RUU: register update unit) に基づいて動的命令スケジューリングを行なうスーパースカラプロセッサであり、以下の構成となっている。RUU の容量は128エントリである。命令フェッチ幅と命令発行幅は8命令であり、8つの機能ユニットは完全に対称であらゆるタイプの命令を実行できる。各演算のレイテンシは乗算(4サイクル)と除算(12サイクル)を除いて全て1サイクルである。データキャッシュは4ポートのノンブロッキングキャッシュで、容量128KBでブロックサイズ32Bの2ウエイセットアソシアティブで構成される。データキャッシュのレイテンシはデータアドレス計算の他に、ヒット時1サイクル、ミス時6サイクルである。また全てのメモリアクセス命令は、先行するストア命令が完了していなければ実行できない。命令キャッシュは容量128KBで、ブロックサイズ32Bの2ウエイセットアソシアティブキャッシュである。2次キャッシュはデータと命令で共有されており、容量8MBでブロックサイズ64Bのダイレクトマップキャッシュである。2次キャッシュミス時のレイテンシは、最初のワードが得られるまでが18サイクルで、後続のワードのアクセスにはそれぞれ2サイクルを必要とする。分岐先アドレスの予測にはエントリ数1Kで4ウエイセットアソシアティブの分岐先バッファとエントリ数8のリターンアドレススタックを、分岐方向の予測にはエントリ数4Kのgshareタイプ2レベル適応型分岐予測器を用いた。これらの分岐予測器の更新は命令のリタイア時に行なわれる。

2.2 評価環境

サイクルレベルのタイミングシミュレータは SimpleScalar ツールセット (ver.3.0a)[2)] を用いて作成されている。SPECint95 ベンチマークプログラムを用いて評価する。それぞれのプログラムの入力セットは現実的な時間でシミュレーションが終了するように調整してある。プログラムと入力セットを**表1**にまとめる．これらのプログラムは MIPS アーキテクチャを拡張した SimpleScalar/PISA アーキテクチャ[2)] をターゲットとしてコンパイルされている。132.ijpeg は GNU GCC(version 2.6.3) を用いて最適化オプション-O3 でコンパイルした。残りのプログラムはウイスコンシン大学が配付しているコンパイル済みのバイナリを利用した。表1には各ベンチマークプログラムの実行命令数と値予測の対象となる命令の割合も示してある。つまり、分岐命令やストア命令などのデータ値予測の対象とならない命令を除いた割合である。命令は nop 命令を除いたリタイアした命令だけを数えている。

表1 ベンチマークプログラム

プログラム	入力セット	実行命令数	(%) 対象
099.go	9 9	133M	77.69
124.m88ksim	dcrand.big	241M	68.50
126.gcc	genrecg.i	117M	65.96
129.compress	14000 e 2231	48M	66.25
130.li	queens 7	202M	58.76
132.ijpeg	vigo.ppm	29M	73.14
134.perl	primes.in	10M	62.04
147.vortex	persons.250	101M	59.94

3. 頻繁な値の局所性を利用した値予測器

頻繁な値の局所性 (frequent value locality)[16)] は新しい局所性であり、従来の値の局所性[8)] とは以下の点で異なる。すなわち、後者があるひとつの命令において定義されるのに対して、前者は複数の命令間で観測される。本節では、頻繁な値の局所性がどのように観測させるかを示し、これに基づいたハードウエア削減方式を提案する。

3.1 頻繁な値の局所性

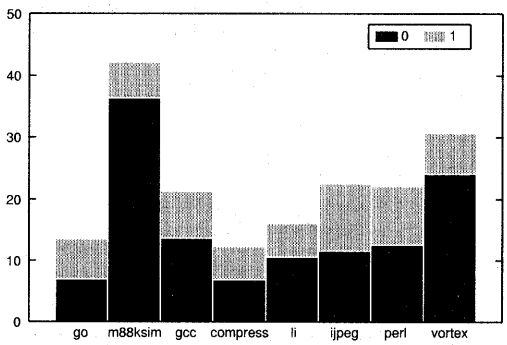


図1 (%) 頻繁な値の局所性

Zhang ら[16)] によると、プログラム実行中のある時点においては、メモリアクセスで参照されるデータの大部分が、少数の値の集合に集中していることが観測されている。SPECint95 ベンチマークにおいては，中でも0と1が上位を占める場合が多い。われわれは彼らの研究を更に進め、レジスタに値を書き込む全ての命令について頻繁な値の局所性を調査した。シミュレーション結果を**図1**にまとめる。各棒グラフは二つの部分から成っており、下部(黒)は実行結果が0であった命令の割合であり、上部(灰)は実行結果が1であった命令の割合である。平均して約20%の命令が0あるいは1を生成していることがわかる。これはメモリアクセスの場合の局所性と比較するとかなり小さな値であるが、それでも考慮するには

表 2　0 を頻繁に生成する命令

program	inst.	%occu.	%freq.	program	inst.	%occu.	%freq.
099.go	lw	34.79	9.07	130.li	lw	49.86	11.58
	slt	27.05	51.92		addu	23.23	15.99
	addu	18.15	5.42		andi	12.04	63.56
	sll	5.70	2.93		lbu	7.61	26.76
	alti	5.46	37.62		addiu	1.81	0.85
124.m88ksim	lw	16.71	29.69	132.ijpeg	lw	22.64	9.84
	addu	16.67	45.55		addu	15.86	5.64
	sll	13.83	98.82		slt	15.15	57.81
	and	11.07	56.89		sll	11.22	11.33
	andi	8.36	37.48		sltu	10.72	47.78
126.gcc	lw	32.93	14.30	134.perl	lw	30.98	11.02
	addu	23.79	19.27		addu	18.41	12.75
	sltiu	6.23	41.83		slt	8.48	74.71
	sll	6.23	14.96		andi	6.79	50.39
	slt	5.21	45.91		sltu	5.17	25.32
129.compress	lw	22.20	5.84	147.vortex	lw	53.04	27.54
	slti	18.10	45.61		addu	37.06	41.26
	slt	17.46	31.19		sltu	2.58	37.42
	sltu	7.54	42.00		andi	1.39	16.19
	and	7.23	31.55		lbu	1.13	41.54

表 3　1 を頻繁に生成する命令

program	inst.	%occu.	%freq.	program	inst.	%occu.	%freq.
099.go	lw	30.57	7.56	130.li	sltu	22.18	90.51
	slt	26.40	48.08		sltiu	16.63	95.05
	addiu	25.04	8.76		lbu	16.62	30.36
	slti	9.54	62.38		slti	16.13	100.0
	addu	3.20	0.91		andi	8.56	23.48
124.m88ksim	sltu	33.32	66.15	132.ijpeg	slti	26.17	87.34
	slti	33.27	99.93		lw	19.94	10.86
	addiu	33.18	16.34		addiu	19.65	12.78
	sra	0.07	0.39		sltu	9.35	52.22
	lw	0.03	0.01		slt	8.82	42.19
126.gcc	addiu	20.93	7.17	134.perl	sltiu	20.01	88.54
	lw	18.69	4.54		sltu	19.08	74.68
	sltiu	15.50	58.17		addiu	18.84	9.30
	slti	15.15	71.08		lw	10.69	3.04
	slt	10.99	54.09		lbu	6.59	31.95
129.compress	slt	45.45	68.81	147.vortex	addiu	52.04	18.03
	slti	25.47	54.39		sltu	15.49	62.58
	sltu	12.30	58.00		lw	11.26	1.61
	addiu	4.95	1.14		lhu	6.51	35.90
	sra	2.57	8.61		addu	5.50	1.71

十分な割合である。

表 2 と**表 3** に、0 と 1 を頻繁に生成する命令についてまとめた。第 1 列はプログラム名である。第 2 列には、0 あるいは 1 を頻繁に生成する上位 5 命令がまとめられている。次列は、0 あるいは 1 を生成する全実行のうちでそれらの命令がどの程度占めるかを表している。最後の列は各命令の 0 あるいは 1 生成頻度である。例えば 099.go の場合、0 を生成する全命令のうちで 34.79% が `lw` である。さらに、全 `lw` 命令の 9.07% が 0 を生成していることが判る。他のプログラムを考慮すると、全体として `lw` 命令が最も 0 を生成する傾向が大きく、次いで addu 命令である。減算命令が全く現れていない点が興味深い。また論理演算命令の出現頻度も比較的小さい。対照的に、論理演算命令は 1 を頻繁に生成する傾向が見られる。

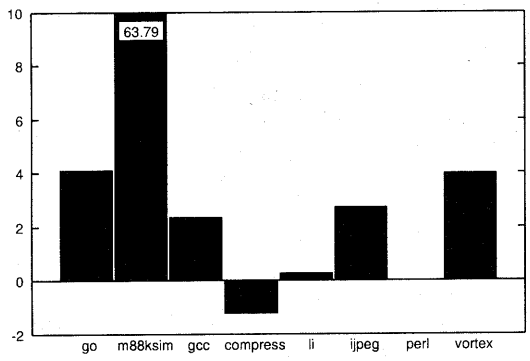
以上の考察から本稿では、この 1 ビットで表現できる頻繁な値の局所性を利用して、データ値予測器のハードウエア量削減を検討する。

3.2　0 値予測器

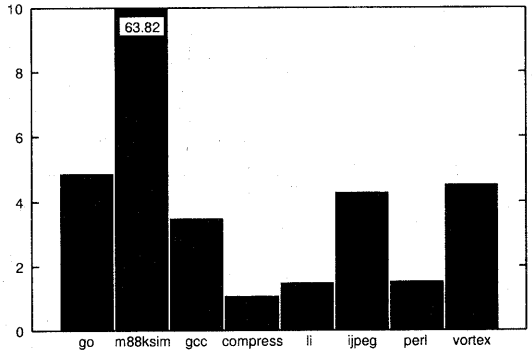
0 が最もよく生成される値であるのならば、0 だけを予測する値予測器であっても意味のあるパフォーマンスが得られる可能性がある。われわれはそのような、0 だけを予測する 0 値予測器 (zero-value predictor) を提案する。0 値予測器は最終値型値予測器の値予測表 (VHT:Value History Table) からデータフィールドを取り除くことで実現でき、ハードウエア規模が大幅に削減できる。

3.3　0/1 値予測器

図 1 から 1 も頻繁に出現することがわかる。したがって、0/1 値予測器 (0/1-value predictor) を構築し 1 の局



(i) 1 サイクル比較遅延



(ii) 比較遅延無し

図 2 (%) 性能向上率 (0 値)

所性も利用できれば、予測可能性が大いに向上すると期待できる。0 と 1 とを区別するために、0/1 値予測器の VHT には 1 ビットの Data Value フィールドが必要である。加えて、その入れ替え方式を考慮する必要がある。ある命令が0 または1 を生成した時には、対応する VHT のエントリにはその値を保持すればよい。0 でも 1 でもない時には、入れ替えには自由度がある。本稿では単純な方式を採用した。すなわち、その命令が 1 を生成した時に限り VHT のエントリを更新し、そうでない場合には VHT には 0 を保持しつづけることとした。言い替えれば、1 よりも 0 を優先させるわけである。図 1 より 0 の局所性の方が大きいので、妥当な選択と言える。

4. 評　　価

本節でシミュレーション結果を紹介する。信頼性の機構には閾値 2 の 2 ビット飽和型カウンタを用いる。タグは使用しない。また容量は 8K エントリとした。これらの信頼性機構と容量の決定については文献1) を参照されたい。

まず、8K-エントリのタグを持たない0 値予測器と0/1 値予測器が、プロセッサ性能にどの程度貢献できるかを調べる。性能の評価には1 サイクル当たりの完了命令数 (IPC:committed instructions per cycle) を用いた。カウントされる命令には nop 命令を含んでいない。プロセッサの性能向上は、増加した IPC をデータ予測を行なわない基本モデルの IPC で割った増加率で示す。続いて、これらの予測器を最終値型予測器と比較する。値予測に失敗した場合には、正しい実行を保証するために、間違った予測値を用いて投機的に実行された命令を、正しいオペランドで再実行する必要がある。このための回復機構には、投機に失敗した命令とデータ依存の関係にある命令だけを選択的に再実行する方法を選択した。再実行する時には、予測値と正しい値とを比較するために必要な 1 サイクルの比較ステージがペナルティとなる。

図 2(i) に、8K エントリの 0 値予測器を用いた時のプロセッサ性能向上率をまとめる。129.compress を除いて性能向上が確認できる。特に124.m88ksim では63.8%もの向上が得られている。これらの性能向上は、図 1 に示されている値の局所性と非常に似た傾向を示している。このように、0 値予測器は 0 しか予測しないにも関わらず、データ投機実行により性能向上が得られる。したがって、0 値予測器は ILP を抽出するためのコストパフォーマンス比の高い方式の一つだと期待できる。

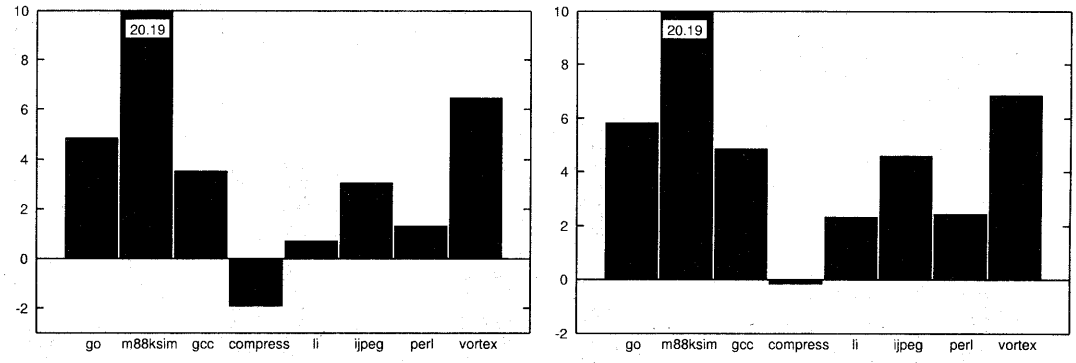
0 値予測器は 0 しか予測しないので、予測値 (常に 0) と正しい値を比較するための回路は簡単に出来る。そのため再実行時に被る 1 サイクルのペナルティを取り除くことも可能である。このような仮定の下で性能向上率を調べた結果が図 2(ii) である、平均して 4.5% の改善となる。

0/1 値予測器を採用した場合の結果は**図 3** にまとめた。結果の絶対値は異なるが、おおまかな傾向は 0 値予測器の場合に近いことが判る。

表 4　予測器のハードウエア規模

predictor type	capacity (entries)	cost (bytes)
zero-value	8K	2K
0/1-value	4K	1.5K
	8K	3K
last-value	512	2K
	1K	4K
	2K	8K

つづいて、0 値および 0/1 値予測器を最終値型予測器と比較する。最終値型予測器はこれまで考案された予測器の中で、最もハードウエアコストの小さい予測器の一つである。**表 4** にまとめられた 6 つの場合を評価する。この表には各予測器のハードウエア量もまとめられている。タグは使用されていないことを思い出していただきたい。この表より、8K エントリの 0 値予測器は 512 エントリの最終値型予測器とハードウエア量において同じ



(i) 1 サイクル比較遅延　　(ii) 比較遅延無し

図 3 (%) 性能向上率 (0/1 値)

であることがわかる。また4K エントリの0/1 値予測器は512 エントリの最終値型予測器よりもハードウエア量において小さいことが判る。

プロセッサの性能向上率は**図4** のとおりである。各プログラムには6 本の棒グラフがあり、左から順に、8K エントリ 0 値、4K エントリ 0/1 値、8K エントリ 0/1 値、512 エントリ最終値型、1K エントリ最終値型、そして2K エントリ最終値型予測器を用いた際の結果である。最終値型予測器には比較に要する1 サイクルの遅延が存在するが、0 値および0/1 値予測器にはその遅延が無いことに注意されたい。まず第一に、0 値予測器は1K エントリの最終値型予測器よりもプロセッサ性能向上における貢献が大きい。1K エントリの最終値型予測器は8K エントリの0 値予測器の倍のハードウエア量が必要であることを思い出すと、非常に有意義な結果と言える。さらに、多くのプログラムにおいて0 値予測器は2K エントリの最終値型予測器と匹敵した効果を誇っている。特に `124.m88ksim` においては0 値予測器は最終値型予測器を大きく凌駕している。これは0 の値の局所性を利用する場合の効果を示す良い例であろう。2K エントリの最終値型予測器には、0 の値の局所性を利用できるに充分な容量が無く値の入れ替えが生じてしまうが、8K エントリの0 値予測器には充分な容量があると考えられる。以上により、0 値予測器のコストパフォーマンスの高さが確認できた。

0/1 値予測器は0 値予測器とは異った振舞いを見せている。`099.go`、`126.gcc`、`134.perl`、そして`147.vortex` では、4K エントリであっても0/1 値予測器は2K エントリの最終値型予測器と匹敵した効果を表している。この時、ハードウエア量には5 倍の違いがある。`132.ijpeg` と `124.m88ksim` では、両予測器の効果の違いは小くなっている。これらの観察により、0/1 値予測器は0 値予測器と同様に、最終値型予測器と比較して、コストパフォーマンスの高い方法だと言える。しかし `129.compress` と `130.li` では、違いは顕著である。つづいて0/1 値予測器と0 値予測器とを比較すると、`124.m88ksim` の場合に興味深い結果が得られている。この時には0/1 値予測器と比べて0 値予測器の方が3 倍の性能向上率を示している。これは、0 と1 の間の入れ替えが破壊的競合を生じているためだと考えられる。しかしながら全体としては、0/1 値予測器がコストパフォーマンスの高い方法であることに変わりは無い。

5. ま と め

本稿では値予測器のハードウエア規模を削減するために、1 ビットの値の局所性を利用した0 値および0/1 値予測器を提案し、詳細なシミュレーションにより評価した。その結果、これらの値予測器のコストパフォーマンスの高さが確認できた。

謝　辞

本研究の一部は、科学研究費補助金 奨励研究(A) 課題番号12780273、基盤研究 B(2) 展開 課題番号13558030 の援助によるものです。

参 考 文 献

1) 佐藤寿倫, 有田五次郎: 頻繁な値の局所性を考慮したデータ値予測機構のハードウエア量削減, 信学技報 *CPSY2000-62*, 2000.
2) D.Burger, T.M.Austin, "The SimpleScalar tool set, version 2.0", *ACM SIGARCH Computer Architecture News*, vol.25, no.3, 1997.
3) M.Burtscher, B.G.Zorn, "Hybridizing and coalescing load value predictor", *Technical Report CU-CS-903-00*, Department of Computer Science, University of Colorado at Boulder, 2000.
4) B.Calder, G.Reinman, D.M.Tullsen, "Selective value prediction", *26th International Symposium on Computer Architecture*, 1999.
5) S.Del Pino, L.Pinuel, R.A.Moreno, F.Tirado, "Value prediction as a cost-effective solution to

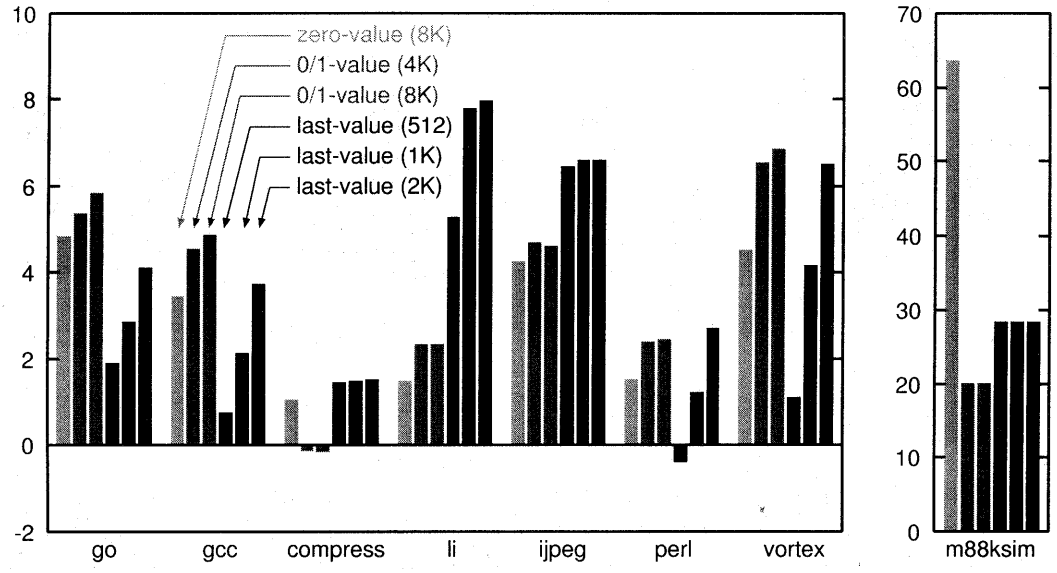


図4 (%) 最終値型との比較

improve embedded processor performance", *3rd International Meeting on Vector and Parallel Processing*, 2000.

6) C-y. Fu, M.D. Jennings, S.Y. Larin, T.M. Conte, "Software-only value speculation scheduling", *Technical Report*, Department of Electrical and Computer Engineering, North Carolina State University, 1998.

7) F.Gabbay, "Speculative execution based on value prediction", *Technical Report #1080*, Department of Electrical Engineering, Technion, 1996.

8) M.H.Lipasti, C.B.Wilkerson, J.P.Shen, "Value locality and load value prediction", *International Conference on Architectural Support for Programming Languages and Operation Systems VII*, 1996.

9) E.Morancho, J.M.Llaberia, A.Olive, "Split last-address predictor", *International Conference on Parallel Architectures and Compilation Techniques*, 1998.

10) B. Rychlik, J.W. Faistl, B.P. Krug, A.Y. Kurland, J.J. Sung, M.N.Velev, J.P.Shen, "Efficient and accurate value prediction using dynamic classification", *Technical Report CMuART-98-01*, Department of Electrical Computer Engineering, Carnegie Mellon University, 1998.

11) T.Sato, I.Arita, "Table size reduction for data value predictors by exploiting narrow width values", *14th International Conference on Supercomputing*, 2000.

12) T.Sato, I.Arita, "Partial resolution in data value predictors", *29th International Conference on Parallel Processing*, 2000.

13) Y.Sazeides, J.E.Smith, "Implementations of context based value predictors", *Technical Report TR-ECE-97-8*, Department of Electrical Computer Engineering, University of Wisconsin-Madison, 1997.

14) D.M.Tullsen, J.S.Seng, "Strageless value prediction using prior register values", *26th International Symposium on Computer Architecture*, 1999.

15) K.Wang, M.Franklin, "Highly accurate data value prediction using hybrid predictors", *30th International Symposium on Microarchitecture*, 1997.

16) Y.Zhang, J.Yang, R.Gupta, "Frequent value locality and value-centric data cache design", *Interbational Conference on Architectural Support for Programming Languages and Operation Systems IX*, 2000.