計算機アーキテクチャ 150−18
（２００２．１１．２８）

値予測を用いた命令流分割によるエネルギー消費量削減

神代　剛典[†]　　佐藤　寿倫[†,††]　　有田　五次郎[†]

エネルギー消費量の削減には供給電圧を下げることが効果的であるが、性能低下の問題も発生する。性能低下を防ぐためには、プログラムの実行速度に影響しない部分のみ低速、低電圧で実行させることが考えられる。我々は、トレースレベル値予測を用いて命令流を分割し、マルチスレッド実行するコントレイルアーキテクチャを検討している。実行されるアプリケーションプログラムを値予測を適用したメインの処理を行う投機流と、値予測の検証を行う検証流に分割する。投機流での命令数の減少と、検証流の低速実行でエネルギー消費量を削減する。本論文では、コントレイルアーキテクチャのエネルギー消費量と実行サイクルへの影響を調査する。

Energy Reduction by Dividing Stream Using Value Prediction

Takenori KOSHIRO,[†] Toshinori SATO[†,††] and Itsujiro ARITA[†]

While lower supply voltage is effective for reducing energy consumption, it suffers the problem of performance loss. In order to mitigate the performance loss, we propose to execute only the part, which does not have any influence on execution speed, with low-speed and low-voltage. We are investigating a multithreaded execution, named Contrail Architecture, which divides an instruction stream into two streams using trace-level value prediction. One is the speculation stream, which is the main part of an application program and is applied value predictions, and the other is the verification stream which verifies the value predictions. The energy consumption is reduced by the decrease in the number of instructions in the speculation stream and by the low-speed execution in the verification stream. In this paper, we evalute the energy consumption and performance of Contrail Architecture.

1. はじめに

近年、携帯電話や PDA などの携帯情報端末の需要が高まっている。最近では、携帯電話で Java が実行可能になるなど、組込用マイクロプロセッサも高性能化が求められている。いずれ現在の汎用プロセッサ並の性能が要求されてくると予想される。実際、既に GHz 超[5)6)8)12)] や SMT 化[5)] された組込用プロセッサの開発計画が公表されている。プロセッサの性能が向上すると問題となってくるのが、エネルギー消費量である。携帯端末では常に電源供給が受けられるとは限らず、内蔵されるバッテリーが大きな制約になっている。バッテリーは、エネルギー消費量によって寿命が決定される。そのため、マイクロプロセッサのエネルギー消費量を削減することが重要となってくる。

† 九州工業大学 情報工学部 知能情報工学科
Department of Artificial Intelligence, Kyushu Institute of Technology
†† 九州工業大学 マイクロ化総合技術センター
Center for Microelectronic Systems, Kyushu Institute of Technology

マイクロプロセッサのエネルギー消費量は、プロセッサで消費される電力とプログラム実行時間の積で求めることができる。したがって、エネルギー消費量を削減するには、どちらか一方または両方を削減できればよい。CMOS トランジスタの消費電力は式 (1) で表せる。消費電力 P_{active} は、クロック周波数 f、負荷容量 C_{load}、供給電圧 V_{dd} によって求まる。また、式 (2) は、供給電圧とゲート遅延 t_{pd} の関係を示している。V_{th} はしきい値、α はキャリアの飽和速度の依存する係数である。

$$P_{active} = fC_{load}V_{dd}^2 \quad (1)$$

$$t_{pd} \propto \frac{V_{dd}}{(V_{dd} - V_{th})^{\alpha}} \quad (2)$$

式 (1) から、供給電圧を下げることが消費電力を削減することに特に有効であることが分かる。しかし式 (2) から明らかなように、供給電圧を下げるとゲート遅延が増加する。ゲート遅延の増加は、マイクロプロセッサのクロック周波数を低下させる要因となる。以上のことから、供給電圧を下げると消費電力を削減できるが、性能低下を引き起こすことが分かる。

この性能低下を補うために、並列性を利用すること

ができる。元の回路の半分の周波数で動作する回路を2つ用意すれば、元のスループットを維持できる。動作周波数を下げるので供給電圧も下げることができる。このケースでは並列度の増加より消費電力削減の効果が大きいことが知られている。我々は、別の並列性を利用したコントレイルアーキテクチャ[9]を検討している。

2. コントレイルアーキテクチャ

コントレイルアーキテクチャは、元のプログラムの命令流を図1の様に分割してマルチスレッド実行をおこなうアーキテクチャである。分割した命令流は、一方を投機流と呼び、他方は検証流と呼んでいる。投機流は、プログラムのメインの部分を実行するが、トレースレベル値予測を用いて実行される命令は少なくなる。実行される命令が減少するためエネルギー消費量は減少する。

一方検証流では、投機流で値予測の成否を判定するために、値予測した命令の実行を行う。ここで重要なのは、検証処理は予測が正しい限り直接プログラムの実行時間に影響を及ぼさないことである。検証される命令の実行結果は、値予測によって既に分かっている。これは、検証操作は特に急いで実行する必要性が無いことを意味する。

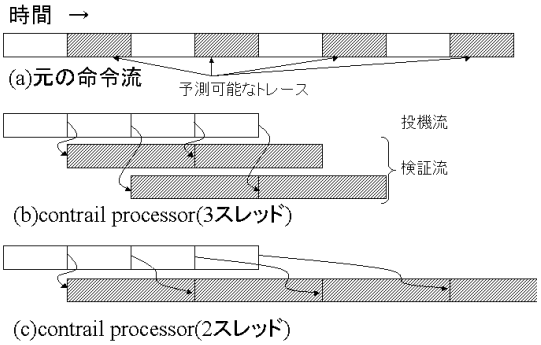

図 1 命令流の分割

各命令流は、SMT プロセッサ[4]や CMP プロセッサ[11]上で独立したスレッドとして実行される。SMT ベースのコントレイルアーキテクチャでのプログラムの実行例を図1に示す。命令列は、図1(a)で示すように値予測が適用できる命令とそうでない命令がある。ここでは、全体の半分の命令が予測可能な命令で、それらは一様に分散していると仮定している。図1(b),(c)は、マルチスレッド機構をもったプロセッサ上のスレッドに各命令流を割り付けた構成例である。(b)はスレッド数に制限が無いのに対し、(c)は、スレッド数を2以下という制約をおいている。この例では、検証流が実行されるスレッドは、供給電圧とクロック周波数が投機流が実行されるスレッドに対し、半分で動作するものとする。

図1の例におけるエネルギー消費量の概算を表1に示す。図1(a)が実行される回路の供給電圧と実行時間を基準として1とする。この場合、消費電力は式1から1と求められ、エネルギー消費量も1となる。コントレイルアーキテクチャの投機流では、供給電圧は変わらないが実行命令数が半分になるため、エネルギー消費量は $\frac{1}{2}$ になる。検証流では、供給電圧とクロック周波数が $\frac{1}{2}$ になるため、消費電力が $\frac{1}{8}$ になる。また、実行命令数は値予測が行われた命令のみで $\frac{1}{2}$ であるが、1つの命令を処理するのにかかる時間は倍になっているので検証流全体でかかる時間は1になる。よって、検証流のエネルギー消費量は、$\frac{1}{8}$ になる。以上のことから、投機流と検証流を合わせたエネルギー消費量は、$\frac{5}{8}$ と求められる。

表 1 エネルギー消費量

		消費電力	実行時間	エネルギー消費量	
オリジナル		1	1	1	
コントレイルアーキテクチャ	投機流	1	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{5}{8}$
	検証流	$\frac{1}{8}$	1	$\frac{1}{8}$	

3. 値 予 測

ここでは、コントレイルアーキテクチャで利用する値予測器について説明を行う。まず、値予測とは過去の履歴を利用して命令の結果を予測するものである。プログラムの実行において、ある命令が過去の命令の実行結果に依存する場合が有る。これは、真の依存関係と呼ばれ、値予測はこの依存を解消するために用いられる。また、命令は依存する命令の実行を待たずして実行可能となる。

$$R3 := R3 \text{ op } R5 \quad (1)$$
$$R4 := R3 + 1 \quad (2)$$
$$R3 := R5 + 1 \quad (3)$$
$$R7 := R3 \text{ op } R4 \quad (4)$$

上記のプログラムでは、命令(2)はレジスタR3の値を利用する。しかし命令(2)がR3に実行結果を出力するために、命令(2)は命令(1)が結果を出力する

まで実行できない。値予測を行うと、命令 (2) は命令 (1) が結果を出力する前に、実行が可能になる。

値予測には、履歴の使い方によりいくつかの予測方式がある。最終値予測、ストライド値予測、コンテキストベース値予測、ハイブリッド値予測といった予測方式が提案されている[7]。これらの予測器は、値の履歴を記録する VHT(Value History Table) や値の出現パターンを記録する PHT(Pattern History Table) を用いて構成される。

3.1 トレースレベル値予測機構

上記までの値予測の方式は、すべて命令単位で値予測を行う。コントレイルアーキテクチャでは、実行はスレッドレベルで分割される。このため命令レベルでの値予測の実行は、スレッド実行の粒度としては細かすぎる。よって、値予測機構としてトレースレベル値予測[7] を利用する。トレースとは、動的な連続する命令の集まりのことである。トレースを用いる手法としてトレースレベル値予測の他に、トレースの再利用 (Trace Reuse)[2)3)] といった手法も提案されている。

従来提案されている手法では、VHT を改良することでトレースレベル値予測を行う。命令単位の値予測では、VHT の 1 つのエントリには 1 つの命令の値の履歴が記録されている。トレースレベル値予測のために改良された VHT では、1 つのエントリに複数の命令の値の履歴が保存される。1 つのエントリで k 個の命令の値の履歴を保持する VHT の例を図 2 に示す。

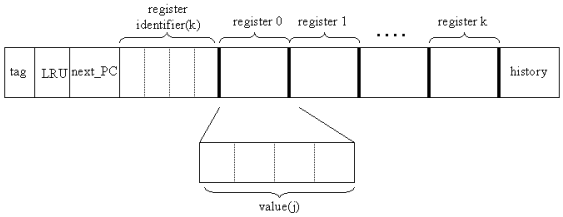

図 2 VHT の拡張によるトレースレベル値予測

3.2 トレースレベル値予測器の改良

従来のトレースレベル値予測器では、VHT に複数の命令の値の履歴が記録されている。個々の命令の予測方式にコンテキストベースやハイブリッド予測といった複数の値を記録する方式を用いると、VHT のサイズは膨大となる。我々は、トレースの情報を値予測機構から切り離したトレースレベル値予測機構を検討している[10]。トレース情報を記録するテーブルをトレーステーブルと呼ぶ。図 3 に、トレーステーブルを用いたトレースレベル値予測器の構成を示す。

トレーステーブルには、予測値を記録せず値予測器のインデックスを保持する。予測は、インデックスを使って値予測器から予測値を取得することで行われる。トレーステーブルでは予測値を記録しないため、予測に用いるハードウェア量は従来の方式より少なくなる。

ハイブリッド値予測器を用いるとき、ハードウェア量の削減率は最も高くなる。過去 k 回の実行履歴を保持するハイブリッド値予測器を用いるとすると従来の方式の場合、記録するレジスタ数が増えるごとに VHT のエントリサイズは、k 個のデータ分増加することになる。しかし、トレーステーブルを利用する場合、VHT のエントリサイズは変化が無くトレーステーブル中のインデックスが増加することになる。

トレーステーブルの各エントリは、以下のフィールドで構成される。

- tag
  トレースを識別するタグ
- 2bc
  2 ビット飽和型アップダウンカウンタ
- nextPC
  トレース後の次の命令のアドレス
- register identifier
  出力レジスタのレジスタ識別子
- predict index
  予測する命令に対応するインデックスを記録する。値予測器は、インデックスとして命令のアドレスを使用するので、実際には命令のアドレスを記録する。

2bc は、トレースの予測に関する信頼性を表すカウンタである。トレーステーブルには、予測値を記録せず値予測器のインデックスを保持する。予測は、インデックスを使って値予測器から予測値を取得することで行われる。以下に予測手順を示す。

(1) 命令のアドレスをインデックスとして、トレーステーブルを検索する。
(2) 対応するエントリが存在する場合、エントリ内のカウンタ値を取得する。
(3) カウンタ値がしきい値以上であれば、予測をおこなう。
(4) 予測器へのインデックスを利用し、レジスタ数だけ並列に値予測を行う。予測器側で VHT のキャッシュミスが起こった場合、予測ミスとして扱う。
(5) 予測器から予測値が出力されたら、レジスタ識別子を判別し該当する場所に書き込む。

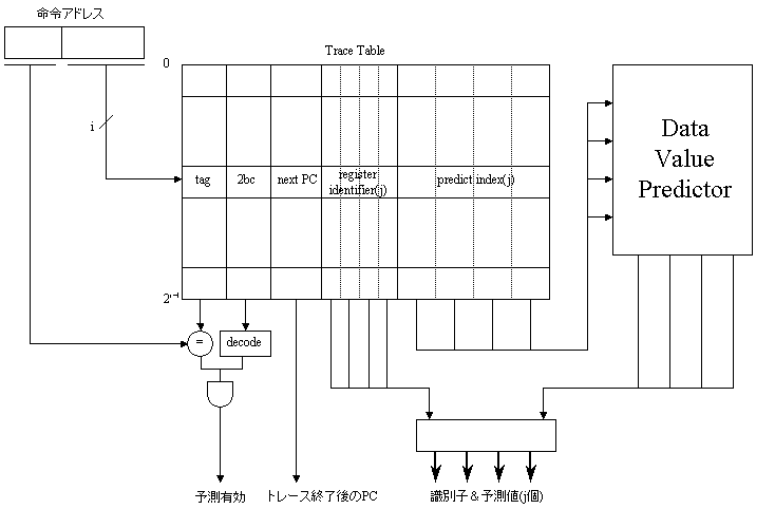

図 3 トレースレベル値予測器

4. 評　　価

4.1 評価環境

性能の評価には、プロセッサシミュレータ SimpleScalar/PISA ツールセット (ver 3.0)[1] に、値予測器を組み込んだものを用いた。テストプログラムは、SPECInt2000 ベンチマークの 5 本のプログラムを用いる。評価は、先頭 10 億命令をスキップして以降の 1 億命令に対して行った。プログラムと入力ファイルを表 2 に示す。

表 2 ベンチマークプログラム

プログラム	入力ファイル
164.gzip	input.compressed
175.vpr	net.in arch.in
176.gcc	cccp.i
197.parser	2.1.dict
255.vortex	lendian.raw

今回の評価は、スーパースカラプロセッサモデルでプログラムの実行を行った。評価に用いたモデルを表 3 に示す。コントレイルアーキテクチャを評価するためのモデルは、整数演算用の演算器 IntALU と、その 2 倍の実行レイテンシを持つ IntALU_S の動作速度の違う演算器を持っていると仮定する。プログラムの実行時に値予測を行い、予測を行った命令は IntALU_S で実行される。現状では、このシミュレータは予測ミス時のペナルティを正しく評価していない。値予測が失敗した場合、以降の命令を全て破棄して再実行するのが本来の処理であるが、このシミュレータは、予測ミスした命令を予測しなかったものとみなして、予測値を利用すること無く、遅い演算器で実行される。

さらに、基準となるモデルとして値予測を行わず、演算器が IntALU のみ 6 個のモデルと、IntALU_S のみ 6 個のモデルを用意する。

表 3 評価モデル

	FU:IntALU	FU:IntALU_S
値予測有り	3	3
値予測無し	6	0
値予測無し	0	6

値予測機構は、命令レベルのハイブリッド型予測器を利用する。予測に用いる VHT と PHT は、共に 4096 エントリである。1 つのエントリに 4 個の値を保持することができる。

4.1.1 消費電力のモデル

表 1 では、エネルギー消費量を求めるために検証流が実行される回路の電圧を、元の半分と仮定して消費電力を計算した。以降の評価には、ARM アーキテクチャのモデルを用いる。表 4 に、ARM プロセッサの消費電力を示す[5]。表中の Intel Xscale 800MHz と 400MHz の消費電力を評価に用いる。投機流が 800MHz、検証流が 400MHz で動作すると仮定すると、検証流の消費電力は投機流の $\frac{1}{5}$ である。

4.2 予測器の評価

まず、図 4 に値予測を行った命令の割合を示す。値予測が多く実行されたプログラムは 197.parser で、約 60%の命令に対し値予測を実行している。逆に、175.vpr や 255.vortex は予測された命令の割合が少ないが、それでも 40%近くの命令に値予測が実行されて

表 4　ARM プロセッサの消費電力

	ARM1020E	Intel XScale
400MHz	200mW @1.1V	180mW@1.0V
600MHz	None	450mW@1.3V
700MHz	None	800mW@1.3V
800MHz	None	900mW@1.65V

いる。平均すると約 45%の命令に対し値予測が実行されたことになり、今回テストしたプログラム中には十分予測対象命令があるといえる。

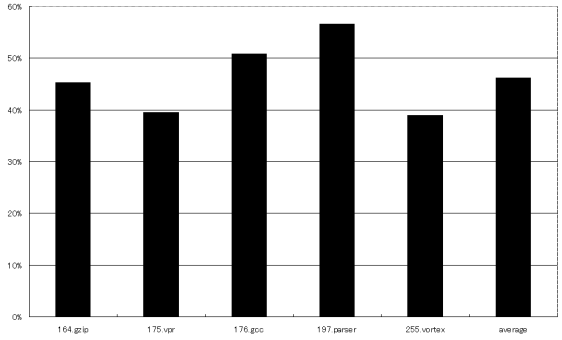

図 4　値予測対象の割合

次に、予測された命令の成功率を図 5 に示す。最も予測の成功率が高いのが 164.gzip で約 80%の命令の値予測が成功する。逆に、175.vpr や 197.parser は、約 60%の予測成功率である。平均すると約 70%で、分岐予測と比べると高い予測成功率とはいえない。

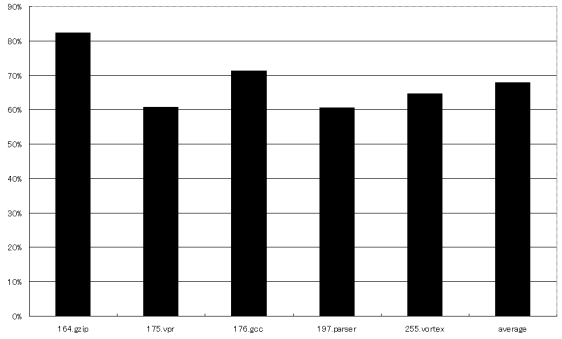

図 5　予測成功率

4.3　コントレイルアーキテクチャの性能評価

図 6 は、実行された命令がどの命令流で実行されたかを表している。平均すると、予測を行わない命令が 55%、予測が成功する命令が 30%、予測が失敗する命令が 15%といった割合になることが分かる。

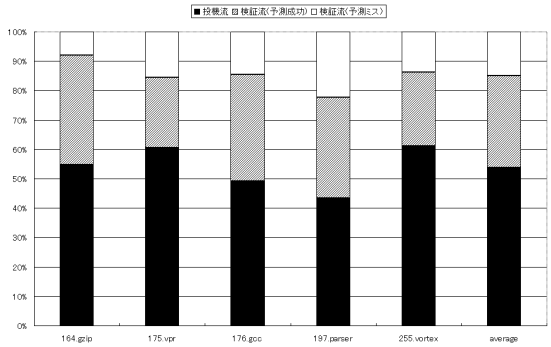

図 6　実行される命令の割合

値予測が性能に与える影響について述べる。図 7 は、IntALU が 6 個で値予測を行わないモデルを基準とした各実行モデルでのプログラムの実行サイクルの比較をおこなった。値予測を行うモデルで実行サイクルが減少したものは、164.gzip と 176.gcc である。この 2 つは、図 5 から分かるように、予測の成功率が高いプログラムである。予測が成功することで、命令がスキップされてプログラムの実行が短縮されるため、予測率が高いものは実行サイクルが短くなる。逆に、実行サイクルが増えたプログラムの中で、一番増加率が高いのが 197.parser である。197.parser は、予測成功率は 175.vpr と大きく変わらないが、図 4 から分かるように予測対象となる命令が多いため、実際の予測ミスした命令は多い。

次に、各モデルでの IPC(Instruction Per Cycle) の変化について述べる。図 8 に、各モデルの IPC を挙げる。予測成功率の高い 164.gzip と 176.gcc は、わずかではあるが IPC が向上している。予測成功率の高くない残り 3 つのプログラムでは、IPC は低下する。

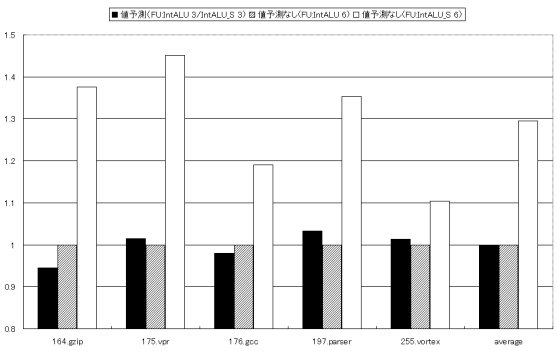

図 7　実行サイクルの比較

図 9 に、4.1.1 で述べた XScale プロセッサの消費電力にしたがって、計算されたコントレイルアーキテク

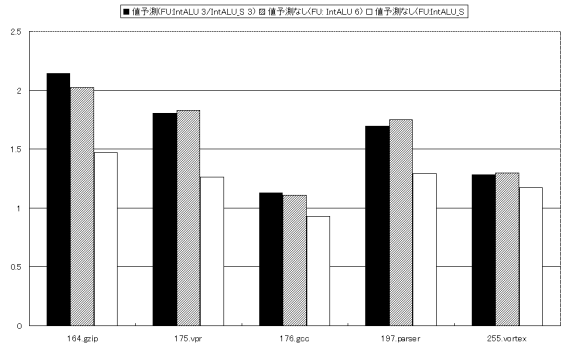

図 8 IPC

チャのエネルギー消費量の削減率を示す。最も削減率が高いのは、197.parser である。これは、IntALU_S の利用率が高いためであるが、値予測成功率は高くないため実行サイクルの増加を伴う。

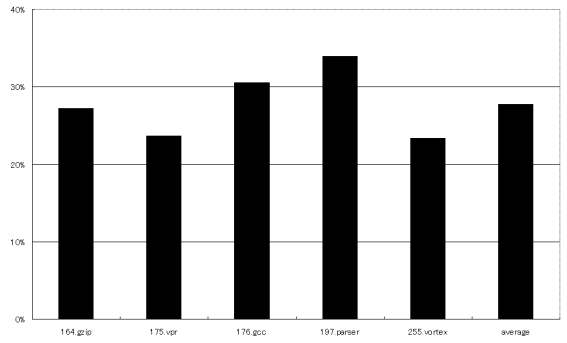

図 9 エネルギー消費量削減率

5. ま と め

本論文では、エネルギー消費削減を目的としたコントレイルアーキテクチャの提案を行い、値予測を用いたこのアーキテクチャの実行時間とエネルギー消費量に与える影響を調査した。命令レベルの値予測器を用いたスーパースカラプロセッサシミュレータ上での評価の結果、平均すると 30%近くのエネルギー消費量削減効果があることが分かった。実行サイクル数に関しては、今回のシミュレーションの結果では大きな増加は見られなかった。しかし、現状では予測ミス時のペナルティが適切でないため、今後正確な調査を行いたい。

謝　　辞

本研究の一部は、科学研究費補助金 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) Doug Burger, Todd M. Austin, “The SimpleScalar Tool Set, Verson 2.0”, Technical Report No.1342, Computer Sciences Department University of Wisconsin-Madison, 1997.
2) Amarildo T. da Costa, Relipe M. G. Franca, Eliseu M. C. Filho, “The Dynamic Trace Momoization Reuse Technique”, International Conternce on Parallel Architectures and Compilation Techniques, 2000.
3) Antonio González, Jordi Tubella and Carlos Molina, “Trace-Level Reuse”, International Conference on Parallel Processing, 1999.
4) Intel Corp., “Hypre-Threading Technology”, Intel Technical Journal, volume 6, issue 1, 2002.
5) Jin Cheon Kim, “1.2GHz と高速動作が可能な ARM 系システム LSI”, NE Electronics Embedded Processor Symposium, 2002.
6) Doug Parse, “2GHz と高速動作が可能な MIPS 系マイクロプロセッサ”, NE Embedded Processor Simposium, 2002.
7) Rahul Sathe, Kai Wang, Manoj Franklin, “Techniques for performing highly accurate data value prediction”, Microprocessors and Microsystems, Vol.22, No.6, 1998.
8) 新井智久, “オリジナル MIPS コアのアーキテクチャと次世代 GHz コア”, NE Embedded Processor Simposium, 2002.
9) 神代剛典, 佐藤寿倫, 有田五次郎, “Contrail Processor Architecture”, 第 5 回システム LSI ワークショップ, 2001.
10) 神代剛典, 佐藤寿倫, 有田五次郎, “トレースレベル値予測におけるハードウェア量削減”, 並列処理シンポジウム, 2002.
11) 鳥居淳, “1 チップ制御並列プロセッサ Merlot のアーキテクチャ”, 並列処理シンポジウム, 2000.
12) 森安俊紀, “ギガヘルツ級 SoC ソリューション実現に向け”, NE Embedded Processor Simposium, 2002.