

COSMOS プロセッサにおける最適化の有効性

山本季之 佐藤 寿倫 有田 五次郎

九州工業大学 情報工学部 知能情報工学科

{yama,tsato,arita}@mickey.ai.kyutech.ac.jp

<http://www.mickey.ai.kyutech.ac.jp/~tsato/cosmos/>

あまし COSMOS プロセッサは空間多重マルチスレッド (SMT: Simultaneous Multi-Threading) を用いて構成されており, 命令レベルの並列性 (ILP: Instruction Level Parallelism) とスレッドレベルの並列性 (TLP: Thread Level Parallelism) に着目している. ILP が低い場合, 複数のスレッドを同時に実行することで TLP を活用し, 余った機能ユニットを用いてアプリケーションの実行時に動的最適化を行うオブティマイザを同時実行することで利用効率を向上させようと考えている. しかし, アプリケーションに最適化する余地がなければ, 利用効率は得られない. そこで, 実行トレースを行う Shade を用いてアドレスの検出を行い, 最適化についての考察を行う.

キーワード 命令レベル並列処理, 動的最適化, 空間多重マルチスレッド

The KIT COSMOS Processor: Preliminary Study on Characterizing Hot Spots

TOSHIYUKI YAMAMOTO TOSHINORI SATO ITSUJIRO ARITA

KYUSHU INSTITUTE OF TECHNOLOGY

{yama,tsato,arita}@mickey.ai.kyutech.ac.jp

<http://www.mickey.ai.kyutech.ac.jp/~tsato/cosmos/>

Abstract In this paper, we perform a preliminary study on characterizing hot spots in order to evaluate a mechanism named CONcurrent Dynamic Optimizer (CONDOR). CONDOR enables an application program to be optimized on-the-fly, based on Simultaneous Multi-Threading (SMT). The hot spots are candidates for dynamic optimization. Thus, it is good for CONDOR that many individual hot spots are observed while executing the application. We use 'Shade' for this evaluation.

key words instruction level parallelism, dynamic optimization, simultaneous multithreading

1. ま え が き

現在における半導体技術の発展は目覚ましく、多量のトランジスタを集積したプロセッサの出現が予想されているが、従来のマイクロプロセッサアーキテクチャでは多量のトランジスタを有効に利用して性能に貢献させることはできない。なぜならば、いくら演算器を追加して同時に実行可能な命令数を増加させようとしても、プログラムが本来持っている命令レベル並列性 (ILP: Instruction Level Parallelism) が小さければ、同時に実行可能な演算は限られ、演算器は有効に使われないまま無駄になるだけであるからである。

プロセッサが処理可能な ILP を増やすために、従来は投機的実行や複数バス実行が行われている。投機的実行は分岐予測に失敗した場合に投機を開始した命令と依存関係にある全ての命令の実行結果を破棄し、それらの命令を再実行しなければならない。複数バス実行は分岐の確定時に選択されなかったバスにある全ての命令の実行結果が破棄されることになる。つまり、この二つの方式は無意味になる命令実行を覚悟の上で高い ILP を抽出しようとする方式である。

そこで、COSMOS プロセッサは無意味な命令ではなく有意義な命令を増やすことで ILP を向上させることを検討している。方法としては、アプリケーションの ILP が低い場合、余っている機能ユニットにオブティマイザを同時実行させ、動的最適化を行うことで高い ILP を得られるようにする方法を用いる。本稿は、アプリケーションに対して実行トレースを行って得られるアドレスの値を採取し、その結果から最適化の有効性について検討する。大規模数値計算アプリケーションの様に少数のループが全実行時間の大部分を占める場合には、立上り時に動的な最適化を施してしまうと以降は最適化の余地が無いとも考えられる。一方非数値計算アプリケーションでは、実行は様々なフェーズから構成されており、各フェーズにおいて動的な最適化を実施できる見込みがある。COSMOS プロセッサではこのような非数値計算アプリケーションの動的最適化を目指しており、特に VLSI 設計のための EDA アプリケーションを最初のターゲットとしている。

以下、2 節で COSMOS プロセッサの概要について説明する。3 節で評価環境について述べ、4 節でアドレスの検出結果に最適化の余地があるかどうかを考察する。5 節で関連する研究をまとめ、6 節で結論とする。

2. COSMOS プロセッサの概要

COSMOS プロセッサでは、無意味な命令ではなく有意義な命令を増やすことで ILP を向上させることを目的にしている。あるアプリケーションプログラムの実行時に余っている機能ユニットを利用して、そのアプリケーションのパフォーマンスを向上させる。すなわち、アプリケーションと、それを動的に最適化するオブティマイザ

とを同時実行させることを考えている。我々はこの方式を CONcurrent Dynamic Optimizer(CONDOR) 方式と呼んでいる。注意して欲しいのは、同時に複数のスレッドが実行されるので、一つのスレッドが実行される場合よりも同時に実行される命令が増えている、すなわち全てのスレッドのトータルで ILP が改善されている、と主張しているわけではない。アプリケーションが動的に最適化されるために、そのパフォーマンスが向上する、すなわちアプリケーションプログラムだけを考慮した時の ILP が向上できると主張している。上述した多重スレッドの同時実行は、空間多重マルチスレッド (SMT: Simultaneous Multi-Threading) 方式^{(5),(6),(15),(18)}が適している。そこで我々は SMT を元にして、CONDOR 方式を利用する COSMOS プロセッサの検討を開始した。

2.1 CONDOR 方式

本節で CONDOR 方式を説明する。簡単に言えば、CONDOR 方式は SMT 上でアプリケーションと動的最適化のためのオブティマイザを同時に実行する方式である。アプリケーション実行時に利用されないで遊んでいる機能ユニットを用いて、動的最適化を行なう。動的最適化の結果、アプリケーションの ILP が向上する。つまり、利用されていない命令スロットに、アプリケーションとは別の有効な命令を投入して、アプリケーションのパフォーマンスを向上させるわけである。アプリケーションに並列性が無ければ、十分機能ユニットは余っているので、オブティマイザを実行させる余裕がある。逆に、並列性が高く機能ユニットが余っていない時は、オブティマイザを実行する必要はない。

図 1 に、従来の動的最適化方式と CONDOR 方式の違いを示す。図 1(a) が従来の方式であり (b) が CONDOR 方式である。時刻は縦方向に上から下に向かっており、命令は四角で表されている。命令 1, 2, 3... がアプリケーションプログラムで、命令 A, B, C... がオブティマイザである。アプリケーションは動的に最適化され、命令 1', 2', 3'... となる。

(a) で説明されている従来の動的最適化では、あるループの繰り返し回数が閾値を越えると、オブティマイザが起動され動的な最適化が開始される。最適化の間はアプリケーションプログラムの実行は停止している。オブティマイザの実行が完了すると、最適化後のアプリケーションが再開される。最適化に必要なサイクルは、アプリケーションの実行にとってはオーバーヘッドとなるので、十分繰り返しの多い場合を選んで最適化する必要がある。つまり、オブティマイザを起動するまでの閾値を高くする必要がある。

一方 (b) で説明される CONDOR 方式では、動的最適化の実行の間もアプリケーションは実行されている。オブティマイザは遊んでいる機能ユニットを利用して最適化を行なう。最適化が完了すると、アプリケーションの実行は最適化後の命令に切り替わる。すなわち、CONDOR

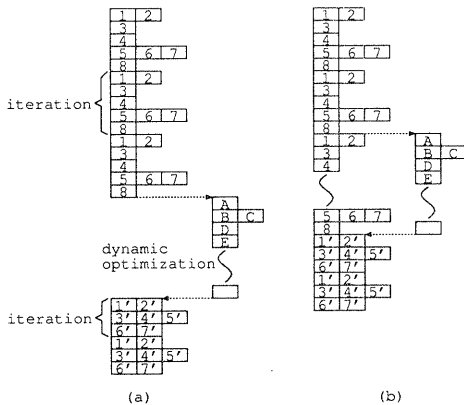


図1 CONDOR方式

方式では最適化に必要なオーバーヘッドを隠蔽可能である。さらに、オーバーヘッドが無いので最適化を開始するまでの閾値を低くできる。ループが検出されれば直ちに最適化を起動することも可能であるし、あるいは最適化実行中にループの繰り返し回数が小さいことがわかれば最適化のスレッドを破棄することも可能である。

以上のように、CONDOR方式は従来の動的最適化方式と比較して、最適化に要するオーバーヘッドを小さくできる。

2.2 プロセッサモデル

現在検討が進行中のプロセッサモデルのブロック図を図2に示す。COSMOSプロセッサはSMTプロセッサと同様に、基本的には大規模なスーパースカラプロセッサであるが、動作速度や設計複雑度を考慮して、以下の様にマイクロアーキテクチャに工夫が施されている。

- フォワーディング遅延を考慮したクラスタ化とデータ値予測の利用。
- 再発行機構とスケジューリング機構を分割した命令ウィンドウ²³⁾。
- 連想検索を排除したスケジューリング機構²⁵⁾。
- 命令バンド幅拡大のためのターボキャッシュ²⁴⁾。

将来のマイクロプロセッサでは、データフォワーディング回路、大容量のレジスタファイル、そして命令発行回路が動作速度を律する要因になると予測されている^{11),17)}。データフォワーディングの遅延を考慮するためには、

- クラスタ化スーパースカラプロセッサ
- オンチップマルチプロセッサ

の選択肢が考えられる。前者は要素プロセッサ (PE: Processing Element) 間で命令供給エンジンが共有され命令実行エンジンは分散されており、現在のスーパースカラプロセッサとの親和性が高い。後者は独立したプロセッサをPEとして扱うので、FIFOキューに基づく並列分岐実行²⁶⁾によるSMTの実現に適している。現在は前者

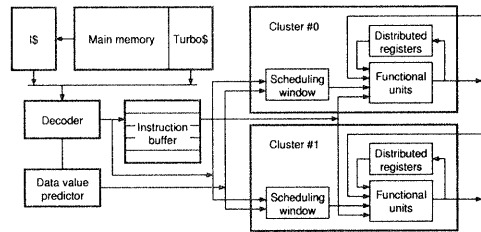


図2 プロセッサのブロック図

での実現を検討しており、後者での実現は将来の検討課題である。図2は二つのクラスタで構成されている例である。自クラスタ内でのフォワーディングは1サイクルで可能である。大容量のレジスタファイルは各クラスタに分散配置される。そのため、他クラスタのレジスタファイルに存在するデータや、他クラスタの演算結果を利用するためには、移動に伴うペナルティを被る。このペナルティを補うためにデータ値予測⁸⁾を利用する。本モデルではクラスタ間通信の可能性があるデータだけを予測対象とするので、予測テーブルの容量を削減できる見込みがある。

データ値予測を利用すると、投機失敗からプロセッサの状態を回復させるための機構が必要になる。そのための代表的な命令再発行機構はプロセッサの性能に悪影響を与える可能性がある。投機実行されている命令とデータ依存の関係にある命令を命令ウィンドウ中に保持し続けなければならないために、命令ウィンドウの実効容量が減少し、その結果命令スケジューリングの自由度が低下してしまう。そこで、命令スケジューリングのための機構と命令再発行のための機構を分けることを提案している²³⁾。命令スケジューリング機構と命令再発行機構を分割すれば、上記の問題は解決できる。各クラスタに分散している小さな命令ウィンドウが命令スケジューリングを担当し、全クラスタで共有される大きな命令バッファが命令再発行を担当する。後者が利用されるのは投機失敗時だけなので、プロセッサ性能に深刻な影響を与えることなくパイプライン化可能である。さらに、各クラスタに分散配置されるスケジューリング機構は連想検索機能を必要としない²⁵⁾ので、高速かつ大容量の命令ウィンドウを実現できる。

最適化後のバイナリはターボキャッシュに保持される²⁴⁾。ターボキャッシュは、ソフトウェアの観点から見ると未使用のメモリマップ領域に割り付けられた空間⁴⁾であり、ハードウェアの観点から見ると大容量のオンチップSRAMである。したがって、ターボキャッシュは動的に生成されたトレースを保持しており、言い換えるとソフトウェア的に構成されるトレースキャッシュ^{13),22)}となっている。オンチップSRAMと動的最適化プログラムが協調することで、トレースキャッシュという特殊なハードウェアを不要にしている。

以上述べた大規模スーパースカラプロセッサ上に、SMTを実現するために複数のコンテキストを保持できるような拡張を施したものが COSMOS プロセッサである。さらに、動的最適化に利用するプロファイル採集をサポートするために、パフォーマンスモニタを活用したハードウェア機構が備わっている。

3. 評価環境

評価に用いるアプリケーションは以下のようにして決定された。大規模数値計算アプリケーションの様に少数のループが全実行時間の大部分を占める場合には、立上り時に動的な最適化を施してしまうと以降は最適化の余地が無いとも考えられる。一方非数値計算アプリケーションでは、実行は様々なフェーズから構成されており、各フェーズにおいて動的な最適化を実施できる見込みがある。したがって、CONDOR ではこのような非数値計算アプリケーションの動的最適化を目指しており、特に VLSI 設計のための EDA アプリケーションを最初のターゲットとしている。EDA アプリケーションをターゲットに選んだのは以下の理由からである。VLSI 設計は性格の異なる様々な工程から構成されており、各工程に対してそれぞれ別のツールが用意されている。これらのツールは単独で用いられる場合もあるが、多くはある一つのフレームワークの中で統一的に使用される場合が多い。これは設計データベースの効率的な管理のために有効である。したがって VLSI 設計においては、設計工程にしたがって EDA アプリケーションの実行は様々なフェーズから構成されていると見做すことが出来る。さらに、特定のアプリケーション、例えば論理合成ツールに注目した場合でも、論理の簡単化、論理の最適化、テクノロジマッピング等々、様々なフェーズから構成されていることがわかる。以上から、EDA アプリケーションは CONDOR の評価に適していると考えられる。

これまでの考察から、アプリケーションとして SPECint92 ベンチマークから、論理関数の簡単化を行う 008.espresso を用いる。EDA アプリケーションとしては espresso は古典的であるが、評価の第 1 段階としては適当な規模であると言える^{*}。入力データは、2bit 入力 1bit 出力の論理関数 OR と 4bit 入力 7bit 出力の論理関数 dc1.in の二つを用意した。dc1.in は SPEC 協会が用意している入力データの中で最も小さなものである。これらの実行トレースを行う Shade³⁾(ver.5) で各命令アドレスの検出を行った。Shade は SPARC アーキテクチャの実行トレースを採取可能なツールであり、実行形式のバイナリを実行時に変換することで高速なトレース採取を可能にしている。Shade は加算や分岐といった収集したい命令と対応する opcode をプログラム中に記すことで、その命令

が検出されるたびに処理を行うことができる。本稿での Shade は全命令を対象にして、各命令ごとにそのアドレスを出力するようにした。

4. 評価結果

本節では、実際のアプリケーションには本当に最適化する余地があるのかを検証する。最適化は頻繁に実行される命令群に対して行われるため、Shade により同一アドレスに複数回のアクセスを行っていることが検出できれば、そのアプリケーションは最適化可能であると言える。espresso のトレースを行った結果を図 3 と図 4 示す。横軸が実行の開始から数えた動的な命令数を表しており、縦軸がそれらの命令のアドレスを示している。容易にわかるように、二つの入力データは図の範囲でほぼ同様の結果を示した。これは、入力データに依存せずにアプリケーションが実行されたことを表している。図に示された約 20 万個の命令は同一アドレスへのアクセスが頻繁に行われていた。このことから、このアプリケーションはイタレーションが大きいループを持つことが分かる。このイタレーションは明らかに閾値を超えることが予想される。これらことから、このアプリケーションは入力データとは無関係に最適化の効果が現れることが分かる。

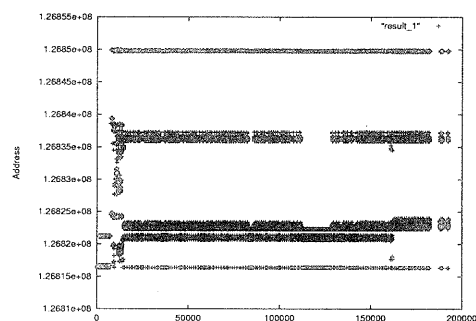


図 3 アドレス検出結果 1 (OR)

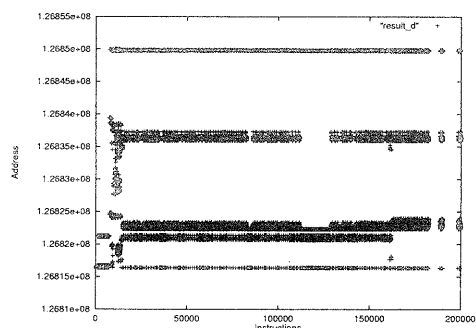


図 4 アドレス検出結果 2 (dc1.in)

^{*} Synopsys 社の DesignCompiler のトレース採取を試みているが、Shade ver.5 に実装されていない機能が必要なこと、DesignCompiler が大規模であること等の理由から、まだ成功していない。

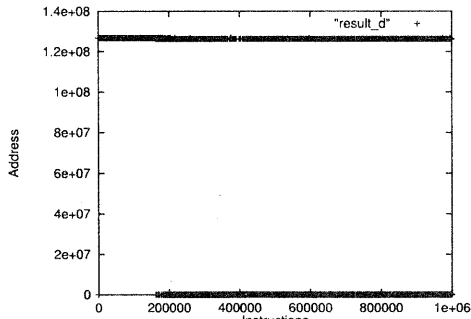


図5 トレース (全体)

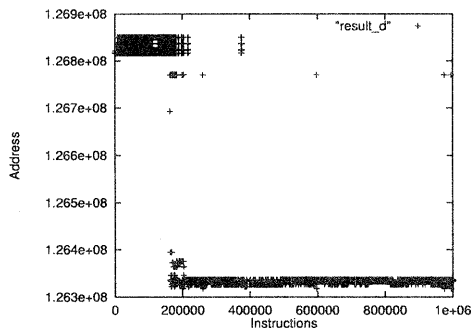


図6 トレース (上位アドレス領域)

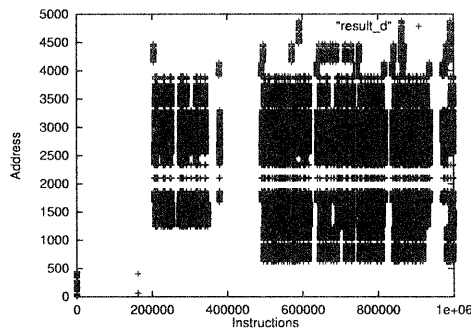


図7 トレース (下位アドレス領域)

図5～図7にdc1.inを入力として与えた場合の100万命令分の命令アドレストレースを示す。図5は全アドレス領域を、図6は上位アドレス領域を、図7は下位アドレス領域を表している。これらからわかるように、小規模なespressoにおいてもアプリケーションの実行にはフェーズがあることがわかる。概してespressoでは二つのフェーズから構成されており、さらに各フェーズで実行される命令領域にはホットスポットが確認できる。

以上をまとめると、espressoのみの評価ではあるが、アプリケーションの実行では特定のアドレス領域の実行が支配的であり、その領域を動的に最適化出来れば実行性能を向上できる可能性がある。また、複数存在するフェーズのそれぞれで動的最適化を実施出来る可能性もある。これらの考察から、CONDORを適用してアプリケーションの実行性能を改善できる可能性があることが期待できる。

5. 関連研究

BalaらのDynamo¹⁾らはソフトウェアによる動的な最適化を提案しているが、最適化時にはアプリケーションの実行は停止しており、そのペナルティを被る恐れがある。Transmeta社のCrusoeプロセッサ^{7),16),21)}は、動的にx86命令を独自のVLIW命令に変換し、さらにプログラムの動的な振舞に応じて最適化を施しているが、Dynamoと同様に最適化に要する時間がペナルティとなる可能性がある。Mertenら^{9),10)}はハードウェアの支援の元に採取されたプロファイルを利用して動的な最適化を検討している。しかし、この場合もSMTを利用して最適化のペナルティを削減することは検討されていない。ChappelらのSSMT²⁾では、予めマイクロRAMに用意されたマイクロスレッドがメインスレッドを最適化する。SMTを利用して最適化を行なう点ではCONDORと共通しているが、CONDORでは最適化プログラムは他のアプリケーションと同様にメインメモリ上のアプリケーションである点で異なる。Patelら¹²⁾はトレースキャッシュ¹³⁾を拡張したrePLayを提案している。rePLayはハードウェアだけに頼った方式であり、CONDORのようなハードウェアとソフトウェアの協調は検討されていない。

小林ら¹⁹⁾やPalacharlaら¹¹⁾のクラスタ化スーパーカブリプロセッサやトレースプロセッサ¹⁴⁾は、プロセッサをクラスタに分割することを検討している。クラスタ化スーパーカブリプロセッサではデータ値予測は利用されていない。また、クラスタ化スーパーカブリプロセッサもトレースプロセッサも、レジスタファイルには複製された部分があり完全に分散されていない。さらに命令ウィンドウの階層化は検討されていない。中村らのSCIMA²⁰⁾はオンチップSRAMの利用を検討しているが、その目的は大量のデータの効率的な配置のためである。一方CONDORではオンチップSRAMであるターボキャッシュは最適化後の命令を配置する目的で使用されている点で異なる。

6. まとめ

CONDOR方式はILPが低い場合に余った機能ユニットにオブティマイザを実行させアプリケーションに動的最適化を行うため、最適化の有効性に性能が大きく左右される。本稿ではCOSMOSプロセッサにおけるアプリケーションに最適化の余地があるかについて考察をおこなった。Shadeを用いたトレース採取により、espressoの実行では特定の領域の実行が支配的であり、動的最適

化の可能性があることが確認された。

最後に今後の課題を述べる。今回は espresso の評価のみであるので、他のアプリケーション、特に SPECint2000 に含まれる 175.vpr と 300.twolf の評価を実施することが第 1 の目標である。さらに、1 節でプログラムが本来持っている ILP が小さければ、同時に実行可能な演算は限られると述べたが、実際の ILP の大きさがどの程度であるのか調べる必要がある。今後は Shade 上に命令レベル並列プロセッサのシミュレータを作成し、様々な種類のアプリケーションについて評価を行う。

本稿は九州工業大学で現在進行中である高性能低消費電力マイクロプロセッサ COSMOS の検討における一つの成果です。Web ページからは追加情報が入手可能です。

謝 辞

本研究の立ち上げ時に、御助言、御討論頂いた九州工業大学マイクロ化総合技術センター田中康一郎助手、鹿児島工専情報工学科堂込一秀助教授に感謝致します。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、福岡県産業・科学技術振興財団 テーマ探索・シーズ発掘事業の援助によるものです。

参 考 文 献

- 1) Bala V., Duesterwald E., and Banerjia S.: Transparent Dynamic Optimization, *Technical Report HPL-1999-77*, HP Laboratories Cambridge (1999).
- 2) Chappel R.S., Stark J., Kim S.P., Reinhardt S.K., and Patt Y.N.: Simultaneous Subordinate Microthreading (SSMT), *Proc. 26th Int'l Symp. on Computer Architecture* (1999).
- 3) Cmelik R.F. and Keppel D.: Shade: A Fast Instruction-Set Simulator for Execution Profiling, *Technical Report UWCS93-06-06*, Dept. of Computer Science and Engineering, Univ. of Washington (1993).
- 4) Ebcioğlu K. and Altman E.: DAISY: Dynamic Compilation for 100% Architecture Compatibility, *Proc. 24th Int'l Symp. on Computer Architecture* (1997).
- 5) Emer J.: Simultaneous Multithreading: Multiplying Alpha's Performance, *Proc. Microprocessor Forum'99 Conference* (1999).
- 6) Gwennap L.: XStream Unit Handles Simultaneous Multithreading, *EE Times*, 10/09/00 (2000).
- 7) Klaiber A.: The Technology behind Crusoe Processors, *White Paper*, Transmeta Corp. (2000).
- 8) Lipasti M.H., Wilkerson C.B., and Shen J.P.: Value Locality and Load Value Prediction, *Proc. 7th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems* (1996).
- 9) Merten M.C., Trick A.R., George C.N., Gyllenhaal J.C., and Hwu W-m.W.: A Hardware-Driven Profiling Scheme for Identifying Program Hot Spots to Support Runtime Optimization, *Proc. 26th Int'l Symp. on Computer Architecture*, (1999).
- 10) Merten M.C., Trick A.R., Nystrom E.M., Barnes R.D., and Hwu W-m.W.: A Hardware Mechanism for Dynamic Extraction and Relayout of Program Hot Spots *Proc. 27th Int'l Symp. on Computer Architecture*, (2000).
- 11) Palacharla S., Jouppi N.P., and Smith J.E.: Complexity-Effective Superscalar Processors, *Proc. 24th Int'l Symp. on Computer Architecture* (1997).
- 12) Patel S.J. and Lumetta S.S., rePLAY: A Hardware Framework for Dynamic Program Optimization, *Technical Report CRHC-99-16*, University of Illinois at Urbana-Champaign, 1999.
- 13) Rotenberg E., Bennet S., and Smith J.: Trace Cache: a Low Latency Approach to High Bandwidth Instruction Fetching, *Proc. 29th Int'l Symp. on Microarchitecture* (1996).
- 14) Rotenberg E., Jacobson Q., Sazeidas Y., and Smith J.: Trace Processors, *Proc. 30th Int'l Symp. on Microarchitecture* (1997).
- 15) Tullsen D.M., Eggers S.J., and Levy H.M.: Simultaneous Multithreading: Maximizing On-Chip Parallelism, *Proc. of 22nd Int'l Symp. on Computer Architecture* (1995).
- 16) Ditzel D.: VLIW 構造で 86 系互換の低消費電力型 MPU を開発, *日経エレクトロニクス*, no.765 (2000).
- 17) 原哲也, 安藤秀樹, 中西知嘉子, 中屋雅夫: 並列性とサイクル時間評価による命令レベル並列マシンの性能比較, 並列処理シンポジウム JSPP'96 (1996).
- 18) 平田博章, 木村浩三, 永峰聡, 西澤貞次, 鷲島敏之: 多重スレッド・多重命令発行を用いる要素プロセッサアーキテクチャ, *情処論*, vol.34, no.4 (1993).
- 19) 小林良太郎, 安藤秀樹, 島田俊夫: データフローグラフの最長パスに着目したクラスタ化スーパースカラ・プロセッサにおける命令発行機構, *情処研報*, 99-ARC-134-31 (1999).
- 20) 中村宏, 近藤正章, 大河原英喜, 朴泰祐: ハイパフォーマンスコンピューティング向けアーキテクチャ SCIMA, *情処論*, vol.41, no.SIG5(HPS1) (2000).
- 21) 野口岳郎: Crusoe とは何か? その真実 ~あるいはソフトウェアの復権, *ASCHI*, vol.24, no.11 (2000).
- 22) 佐藤寿倫: 分岐先バッファを応用した命令フェッチバンド幅の向上, 並列処理シンポジウム JSPP'97 (1997).
- 23) 佐藤寿倫: データ投機実行のための命令再発行機構と命令スケジューリング機構の分割, *情処研報*, 99-ARC-133-4 (1999).
- 24) 佐藤寿倫, 有田五次郎: KIT COSMOS プロセッサ: 背景と着想, *信学技報*, CPSY99-115 (2000).
- 25) 佐藤寿倫, 中村佑介, 有田五次郎: 大規模スーパースカラプロセッサ向け命令発行機構, *信学技報*, ICD2000 (2000).
- 26) 芝浩二郎, 有田五次郎: キュー構造プログラムカウンタによる多重投機実行機構, *情処研報*, 2000-ARC-139-30 (2000).