社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2005－ARC－164 (12)
2005／8／3

GALS型プロセッサにおける動的命令カスケーディング

佐々木 広[†] 近藤正章[†] 中村 宏[†]

プロセッサのチップ全体に単一のクロックを分配する難しさとコストは世代と共に増加しており，既存の同期式設計に変わるプロセッサの構成方式として *Globally-Asynchronous Locally-Synchronous* (*GALS*) 型の構成が広く研究されている．本稿では，GALS 型のプロセッサと相性のよいアーキテクチャ的な改良として，動的命令カスケーディング手法 (*Dynamic Instruction Cascading: DIC*) を提案する．DIC は依存関係にある 2 つの命令を 1 サイクルで実行する手法である．クロック周波数を低下させることにより信号が伝搬する距離は伸び，それゆえ 2 命令を同一サイクル内に実行することが可能になる．ターゲットとなる領域のクロック周波数のみを下げることで適用することが可能になるため，DIC は効果的に GALS 型のプロセッサに用いることができる．また，DIC を用いても効果のないアプリケーションについては DIC をオフにして実行するための判定機構も提案する．評価結果より，DIC が適用可能となるクロック周波数を 80%とした場合において，SPEC CPU2000 の整数ベンチマークと MediaBench において平均で 9%の性能向上を得られることが分かった．

Dynamic Instruction Cascading on GALS Microprocessors

HIROSHI SASAKI,[†] MASAAKI KONDO[†] and HIROSHI NAKAMURA[†]

As difficulty and the costs of distributing a single global clock throughout a processor is growing generation by generation, Globally-Asynchronous Locally-Synchronous (GALS) designs are an alternative approach to the conventional synchronous processors. In this paper, we propose Dynamic Instruction Cascading (DIC) which work out well on GALS processors.. DIC is a technique to execute two dependent instructions in one cycle. Lowering the clock frequency enables the signal to reach farther, thereby executing two instructions in the same cycle becomes possible. DIC is effectively applied to GALS processor because lowering only the clock frequency of the target domain is necessary and high performance (or low power) will be achieved. DIC do not improve the performance of all the applications, so we propose a mechanism to turn off the DIC when the performance is determinated to degrade. The results showed average performance improvement of 9% on SPEC CPU2000 int and MediaBench applications when assuming that DIC is possible by lowering the clock frequency to 80%.

1. はじめに

近年，半導体のプロセスサイズの微細化に伴うチップ上のハードウェア密度の増加，配線遅延の増大などが顕著になってきている．これにより近い将来，既存の同期式設計において単一のクロックを 1 サイクル以内にプロセッサ全体に伝播させることは困難になると予想されている．また，消費電力の観点からも，プロセッサ全体にクロックを撒く従来の同期式手法は効率が悪いと考えられる．このような問題を解決するためにプロセッサの構成方式としてチップを複数の領域に分割し，それぞれを異なる電源電圧・クロック周波数で動作させる *GALS* (*Globally-Asynchronous Locally-Synchronous*) 型のアーキテクチャにおける研究が広く行われている[1)2)4)5)6)]．領域間の通信は非同期になるため新たにインターフェースを追加する必要があり，また通信のオーバーヘッドも生じるが，グローバルクロックを取り除くことができる点や，より狭い領域にクロックを伝搬するので最高クロック周波数が高められるという利点もある．マイクロプロセッサの低消費電力化・低消費エネルギー化がプロセッサの設計における最も重要な課題となってきているが，GALS 型のプロセッサにおいて動的電源電圧制御 (*Dynamic Voltage and Frequency Scaling: DVFS*) をそれぞれの領域ごとに適用することによって，性能の低下を招くことなく消費電力の削減が可能だと報告されている[1)2)4)5)]．先行研究ではプロセッサをフロントエンド，整数演算，浮動小数点演算，ロード/ストアの 4 つのドメインに分割する構成方式が一般的であった．しかし最近では整数演算ドメインとロード/ストアドメ

† 東京大学先端科学技術研究センター
Research Center for Advanced Science and Technology, The University of Tokyo

インの間の非同期通信によるペナルティが性能低下に大きく寄与するため，両者を統合して1つのドメインとする構成が性能的に，また消費電力的にもよいと考えられている[3]．

また，近年のハイエンドマイクロプロセッサはout-of-order実行やsuperscalarといった手法を用い，命令レベル並列性 (*Instruction Level Parallelisim: ILP*) をできる限り抽出し，1サイクルに複数の命令を実行することにより性能向上を図っている．しかし，汎用アプリケーションにおいてはILPが十分でなく，ほとんどの演算資源 (ALUなど) は実行できる命令がないために演算中の大半の時間において使われないでいることが多い．本稿では，このような余剰な演算資源を有効に活用するため，データ依存によって並列に実行できない2命令を同一サイクルに実行することを可能にするアーキテクチャ的手法である，動的命令カスケーディング (*Dynamic Instruction Cascading: DIC*) を提案する．実行するアプリケーションによってはDICが効果的でないこともあるが，そのような場合にはDICをオフにすることによって性能の低下を防ぐ．また，DICとGALS型のプロセッサとの親和性の高さについて言及し，提案手法を通常のプロセッサとGALS型のプロセッサのそれぞれに適用した場合との比較を行う．

本稿の構成は以下のとおりである．次章において提案するDIC手法において説明する．3章では性能評価環境，および評価条件について説明し，4章で評価結果を示す．5章で関連研究を述べ，6章でまとめと今後の課題について述べる．

2. 動的命令カスケーディング

2.1 概　　要

今日，ハイエンドなマイクロプロセッサはプログラムのILPをできる限り抽出し，1サイクルになるべく多くの命令を実行することによってパフォーマンスを向上させるために，out-of-order実行やsuperscalarといった手法を用いている．したがって，同一サイクルに実行できる命令が多ければ多いほど，高い性能が期待できるが，多くの汎用アプリケーションにおいてはILPが命令の発効幅や演算器などの資源に対して十分でない場合がほとんどである．提案するDICは，このような使用頻度の少ない演算資源を有効に活用することを目的としている．メインとなるアイディアは，「依存関係にある2つの命令を1サイクルで実行する」，というものである．ここで，依存関係にある2命令とは，片方の命令の演算結果がもう一方の命令の入力オペランドとなるような命令のペアである．我々は，依存関係にある2命令を同一サイクルに実行するための条件として，プロセッサの電源電圧を高く保ったままクロック周波数を低下させることが必要であると考えている．上記の条件によってサイクルタイムは延長され，信号が伝搬する距離は伸び，その結果2命令を1サイクル内に完了することが可能になる．

例として，命令列が図1のように続いている場合を考える．この場合，Instrction 1の結果がInstrction

```
Instruction 1: add r5 ← r3, r2
Instruction 2: add r4 ← r5, r1
```

図1 命令列

2の入力オペランドとなっているため，これらの2つの命令は並列に実行することはできない．図2の左側

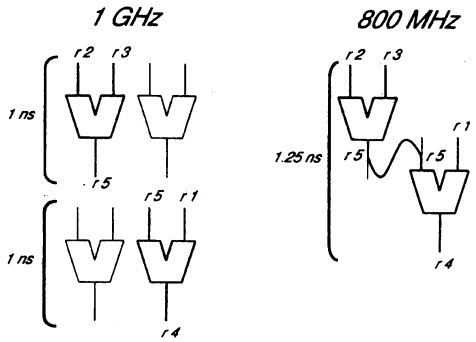

図2 動的命令カスケーディングの例

のように1 GHzで動作しているプロセッサ上のALUでこの2命令を実行したとすると，2サイクル (2 ns) 要する．ここで，DICを適用した場合，Instrction 2の右オペランドr1がレディであればこの2命令は同一サイクル内に実行可能となる．図2の右側に示すように，クロック周波数を800 MHzまで低下させることによってDIC手法が適用可能になるならば，1サイクル (1.25 ns) で2命令を実行可能となる．このように同時発行命令数が増えるため，クロック周波数を低下させても性能が向上する場合がある．整数演算命令においては1サイクルの内，命令の実行に0.5サイクル，結果をバイパスするのに0.5サイクルかかるということがよく知られている．Dynamic Strandsの研究においてはクロック周波数を低下させることなくセルフバイパスモードをもったALUにおいて2命令を1サイクルで完了できると報告されている[11]．このような関連研究から，DICが800 MHzや900 MHz程度のクロック周波数において可能になると主張することは妥当であると考えられる．

本手法を用いることにより，本来ならば次のサイクルまで待たなければ実行することができなかった命令が同じサイクル内に実行可能となるため，同時発行命令数は増え，それゆえ *Committed Instructions Per*

Cycle (*IPC*) は必ず増加する．しかし，適用条件として周波数を低下させなければならない．したがって，提案手法は周波数を低下させることによるサイクルタイムの増加と，IPC の向上による性能の向上とがトレードオフの関係にある．周波数と IPC の積が性能を表すため，IPC の向上率が周波数の低下率を上回る場合には本手法を用いることによって性能が向上する．

2.2 GALS における動的命令カスケーディング

DIC を既存の同期プロセッサに適用する場合，プロセッサ全体のクロック周波数を下げる必要がある．したがって，図 2 の例のように DIC を適用し整数演算命令において IPC が向上したとしても，浮動小数点命令やロード/ストア命令，またフェッチやデコードなどのパイプラインステージにおいて周波数を低下することによる性能の低下は著しく，全体の性能が向上するとは考えにくい．しかし，本稿では GALS プロセッサを仮定しており，DIC を適用する整数演算領域においてのみクロック周波数を低下させることによって不必要な性能の低下を防ぐことができる．

対象とする GALS 型プロセッサの構成は図 3 に示すようにフロントエンド (front-end)，整数演算-ロード/ストア (int-load/store)，浮動小数点 (fp) ドメインの 3 つに分割されており，文献[3] において提案されているものとを基本としている．我々の提案する GALS 型プロセッサはロード/ストアキューと整数命令キューが統合されているという点において異なっている．

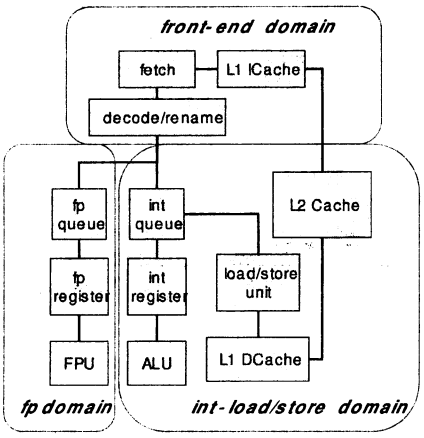

図 3 提案手法における GALS 型プロセッサ

本稿では，2 種類の DIC を提案する．一つは整数演算命令と整数演算命令をカスケードし，もう一方はロード命令と整数演算命令をカスーディングするものである．上記の 2 種類の手法は同時に用いることができる．

(1) **整数命令 - 整数命令**

DIC を適用するに当たって，2.1 で述べたように，演算の実行を 0.5 サイクルで終えることができる整数演算に着目する．図 2 の例に示したように，クロック周波数を低下させることによって依存関係にある 2 つの整数演算命令を 1 サイクル内に完了することが可能となる．命令のリネームステージにおいて命令キュー内の命令間の依存がチェックされる際に，カスケードが可能となる命令の組が選び出される．本手法の利点は，使われていない演算資源を有効に活用し同時発行命令数を増やすことができる点である．したがって，発行ステージにおいてカスケードできる命令の組は優先して発行される．

(2) **ロード命令 - 整数命令**

さらに積極的に DIC を適用するために，整数演算命令と依存関係にあることが多いと思われるロード命令に着目する．多くの商用プロセッサにおいて，レベル 1 データキャッシュのヒット時のレイテンシは 2 サイクルである．内訳はデータアクセスに 1 サイクル，データ転送にさらに 1 サイクルである．そこで，本稿ではプロセッサがラインバアッファを整数-ロード/ストアドメインに有していることを想定する．ラインバッファとは容量の非常に小さい，ヒット時のレイテンシが 1 サイクル未満であるレベル 0 キャッシュである．一般的にその構成はマルチ・ポートでマッピングはフル・アソシアティブ方式でありデータの入れ替えは FIFO 方式となっている[13]．ラインバッファを用意することによって，ラインバッファにヒットするロード命令と，その命令に依存のある整数命令がカスケード可能になると考えられる．

2.3 プログラム適応型動的命令カスケーディング

上記の 2 種類の DIC を提案したが，初期評価の結果この手法を用いても IPC の向上がほとんど見られず，クロック周波数を下げることによって大幅に性能が低下してしまうアプリケーションが存在することが分かった．例えば，あるアプリケーションはまったくカスケードできる命令の組が存在せず，DIC が意味をなさない．また，カスケードできる命令は多く見つかった場合でもメモリバウンドなアプリケーションにおいてはメモリのアクセスレイテンシによって DIC の効果が隠蔽されてしまう場合もあった．このようなプログラムに対して DIC を適用してもいたずらに性能を下げるだけなので，本稿では性能向上が見込めるプログラムかどうかを見分ける機構を設けることを提案する．

プログラムの開始時に DIC を適用せず (ノーマルモード) に一定時間コミットされた命令数を記録し，その後 DIC を適用し同様に記録を行う．この両方の値を比べることによって性能が向上したかどうかを判定し，向上した場合は DIC を適用し続けそうでない

場合はノーマルモードに戻って実行を続ける．上記の比較を行うことにより，DIC を適用した場合に性能が向上するアプリケーションのみを選択することができる．

3. 評　　価

3.1 評価環境

本稿で提案する動的命令カスケーディング手法による性能への影響を調べるため，SimpleScalar Tool Set[8] を用いたシミュレータにより評価を行う．本評価ではプロセッサの内部を分割し，それぞれが異なる電源電圧・クロック周波数で動作できる GALS 型のプロセッサの評価を行うため，サイクルレベルシミュレータである SimpleScalar Tool Set をイベントドリブン型のシミュレータへと変更する．また，メモリ階層を正確にシミュレーションするための拡張がなされた "Simplescalar with Memory Extension"，および消費電力を評価するための改良がなされた "Wattch"[7] の拡張を統合したものを用いる．

評価には，SPEC CPU2000[10] の整数ベンチマークおよび，MediaBench ベンチマークからいくつかのプログラム (adpcm, epic, g721, そして mpeg2, それぞれデコードとエンコード) を用いる．SPEC CPU2000 についてはプログラムの最初の 10 億命令を fast-forward し，200 万命令をシミュレーションした．また，MediaBench についてはプログラムの最初から最後までを評価した．

3.2 評価の仮定

表 1 に，本評価におけるプロセッサの仮定を示す．すべての評価において，プロセッサはラインバッファを搭載しているものとする．ラインバッファのサイズは 4 ライン，32 B である．非同期通信のインターフェースには文献[12] にて提案されている FIFO を実装し，そのオーバーヘッドについても考慮する．また，DIC によって性能が向上するかどうかについて判定するサイクル数は 100%のクロック周波数における 10000 サイクルとした．電源電圧を固定したままクロック周波数のみを変更するのにかかる時間は PLL の安定にかかる時間となり，約 1000 サイクルである．したがって，200 万命令をシミュレーションする本評価においてこの時間は無視できるものとなる．

我々は DIC を GALS 型プロセッサに適用した場合と，既存のプロセッサとの性能の比較を行った．また，提案手法が GALS 型プロセッサにおいて，より効果を発揮することを示すため，同期式プロセッサにこの手法を適用した場合とを比較した．同期式プロセッサに適用した場合は，チップ全体のクロック周波数を低下させることになる．

表 1　評価における仮定

Fetch & Decode width	8
Branch prediction	Combined bimodal (4K-entry)
BTB	1024 sets, 4-way
Mis-prediction penalty	3 cycles
Queue size	
- integer(+load/store)	128
- floating-point	64
Issue width	
- integer	8
- load/store	2
- floating-point	4
Commit width	8
L1 I-cache	32 KB, 32 B line, 2-way 1-cycle latency
L1 D-cache	32 KB, 32 B line, 2-way 2-cycle latency
L2 unified cache	512 KB, 64 B line, 8-way 10-cycle latency
Memory latency	80 cycles
Bus width	16 B
line buffer	4 lines
Clock frequency rate	100%, 90%, 80%, and 70%

4. 評 価 結 果

本稿で提案する DIC 手法が性能へ及ぼす影響とを見るため，図 4 および図 5 に性能の結果を示す．また，章の後半において消費エネルギー削減の効果について議論する．図 4 中，"non-GALS-DIC" は既存のプロセッサに提案手法を，"GALS-DIC" は図 3 に示す GALS 型プロセッサに提案手法を適用した場合を表す．

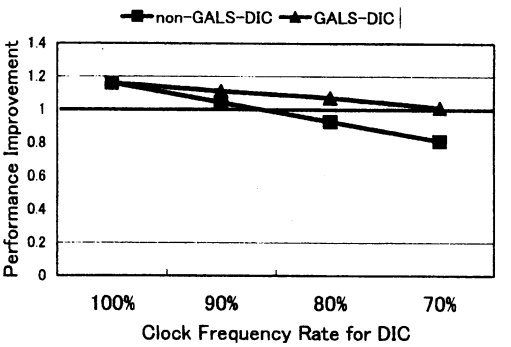

図 4　結果: 全プログラムの平均性能向上率

性能

図 4 は既存の同期式プロセッサと，DIC を用いたプロセッサとの性能比を示している．non-GALS-DIC が既存のプロセッサ，そして GALS-DIC が GALS 型

のプロセッサに提案手法を適用した場合であり，評価に用いたすべてのベンチマークにおける平均の性能を比較している．また，図は既存の同期式プロセッサの性能を 1 として正規化してある．提案手法を適用するためにはクロック周波数を低下させる必要があるため，4 種類のクロック周波数について評価を行った (それぞれ，100%，90%，80%，そして 70%)．既存のプロセッサに DIC を適用した場合，チップ全体のクロック周波数を低下させるためその性能は周波数に比例して低下するが，GALS 型プロセッサに適用した場合は整数-ロード/ストアドメインのみ周波数を低下させるため，性能の低下が抑えられる点に注意する必要がある．

図から分かるように，100%のクロック周波数，つまり周波数を下げることなく DIC が適用可能であるならば，大幅な性能向上が得られる．この場合，既存のプロセッサと GALS 型のプロセッサどちらに適用した場合でも 18%程度性能が向上している．また，クロック周波数を下げていくとそれにしたがって性能は低下していくが，GALS-DIC は，最も性能に対してクリティカルであるフロントエンドドメインの周波数を高く保てるため，既存のプロセッサに適用する場合と比べて高い性能を達成できている．その結果，クロック周波数を 80%まで低下させた場合においてでも，既存のプロセッサと比較して高い性能を達成しており，このときの性能向上率は約 9%である．先行研究などを踏まえると DIC を適用するために必要なクロック周波数は 80%程度で十分現実的であると考えられるため，これ以降は周波数を 80%として評価した場合について議論する．

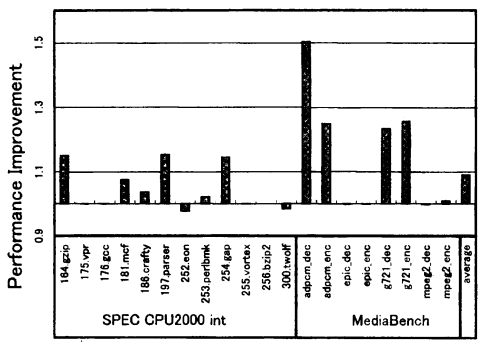

図 5　結果: クロック周波数 80%における性能向上率

図 5 はクロック周波数を 80%としたときの，すべてのベンチマークプログラムにおけるパフォーマンスの向上率を示している．`adpcm`，`164.gzip`，`197.parser`，そして `254.gap` などのプログラムにおいては大幅に性能が向上していることが分かる．反対に，`176.gcc`，`255.vortex`，また `epic` においては性能が全く向上していない．これらは DIC を適用しても同時発行命令数がほとんど増えず，IPC 向上の効果が得られないアプリケーション群である．したがって，2.3 で述べたように DIC を適用しても性能向上が望めないと判断され，ノーマルモードで動作したために，このような結果となっている．このように効果が低いベンチマークも存在するが，提案手法は同時発行命令数を増やし，演算資源を効率的に使うことによって全体として性能を 9%向上させることが示されており，この点において有効であると考えられる．

今後の発展として，プログラムはフェーズ毎に性質が異なるという特徴に着目し，提案手法の効果があるかどうかを，本稿のように一回だけではなくフェーズ毎に判断し，その都度 DIC モードかノーマルモードを選択するという機構を設けることが挙げられる．本手法では，はじめに選択したモードのまま実行を続けるので，DIC によって性能が向上するチャンスを逃している可能性が大いにある．上記によって，さらなる性能向上が期待できる．

消費エネルギー

図 5 に示したように，クロック周波数を 80%に低下させることによって DIC が適用できるとしたときの GALS-DIC の平均性能向上率は 9%である．DIC は電源電圧を低下させないため，演算量が同じならばその消費エネルギーは提案手法を適用しなかった場合と変わらない．つまり前述の結果を言い換えると，DIC によって消費エネルギーを増やすことなく性能向上を達成している．したがって，DIC を適用しなかった場合と同じ性能が得られるようにクロック周波数と電源電圧を低下させることによって消費エネルギーの削減が可能となる．

以下では，具体的な例を挙げて消費エネルギーの削減効果について議論する．表 2 は Intel Pentium M プロセッサ (1.6 GHz 版) において設定可能なクロック周波数とそれに対応する電源電圧の関係を表す．GALS-DIC がこの表の値から補完されるあらゆるクロック周波数と電源電圧の組を選択できるとすると DIC を適用していないプロセッサ (1.6 GHz & 1.484 V) に比べて，最大で 35% (adpcm_dec)，平均して 8.5%の消費エネルギーを性能低下を招くことなく削減できる．

5. 関連研究

本研究では依存関係にある 2 命令を 1 サイクルで実行する手法を提案したが，同じように依存関係にある命令をまとめることによって性能向上を達成したり，ハードウェアを効率的に使用するといったことを目的とした研究がなされている．Intel Pentium 4 プロセッサにおいては，レイテンシの短い ALU において "staggered add" と呼ばれる手法を用いることによって 2 倍のクロック周波数で依存関係にある 2 命令を実行できる[9]．また，文献[11] は，依存関係にある

表 2 Intel Pentium M プロセッサのクロック周波数と電源電圧の関係

Processor Clock	1.6 GHz	1.4 GHz	1.2 GHz	1.0 GHz	800 MHz	600 MHz
FSB Clock	400 MHz	400 MHz	400 MHz	400 MHz	400 MHz	400 MHz
Memory Bus Clock	266 MHz	266 MHz	266 MHz	266 MHz	266 MHz	266 MHz
Processor Core Vdd	1.484 V	1.420 V	1.276 V	1.164 V	1.036 V	0.956 V

命令の列をマクロ命令に置き換えることによって命令発行キューやリオーダー・バッファを効果的に利用する手法を提案している．上記においてはセルフ・バイパス・モードをもつ ALU において 2 命令を 1 サイクルで実行可能であると主張しているが，本手法のようにクロック周波数を低下させることが必要であるとは考えていない．また，この研究では整数演算命令同士のみをまとめており，本研究のようにロード命令と整数演算命令の組も考慮に入れている点において異なっている．

6. まとめと今後の課題

本稿では，チップ内を異なるクロック周波数と電源電圧で駆動させる GALS 型プロセッサの特長を生かし，アプリケーションの実行中に使われていないことが多い演算資源を有効に活用する動的命令カスケーディング (Dynamic Instruction Cascading: DIC) と言う手法を提案した．DIC はクロック周波数を下げることによって依存関係にある 2 つの命令を同一サイクルに実行し，クロック周波数を下げる代わりに IPC を向上させる．また，クロック周波数と電源電圧を領域毎に設定可能である GALS 型プロセッサと提案手法は相性がよいことを示した．

イベントドリブン型のシミュレータにより評価を行い，提案手法を適用するためにクロック周波数を 80%にした場合，平均して 9%の性能向上が得られることが分かった．また，この性能向上分を消費エネルギー削減効果に変換した場合，Intel Pentium M をベースとしたモデルを仮定したときに平均 8.5%の消費エネルギーを性能を低下することなく削減できることを示した．また，今回は DIC をオン・オフするに当たっては非常に簡単な手法を用いたが，これを拡張した上でさらなる評価を行うことが今後の課題である．

謝辞 本研究の一部は (株) 半導体理工学研究センターとの共同研究によるものである．

参 考 文 献

1) Greg Semeraro, Grigorios Magklis, Rajeev Balasubramonian, David H. Albonesi, Sandhya Dwarkadas, and Michael L. Scott, "Energy-Efficient Processor Design Using Multiple Clock Domains with Dynamic Voltage and Frequency Scaling," (HPCA'02), 2002.
2) Greg Semeraro, David H. Albonesi, Steven G. Dropsho, Grigorios Magklis, Sandhya Dwarkadas and Michael L. Scott, "Dynamic Frequency and Voltage Control for a Multiple Clock Domain Microarchitecture," (MICRO35), 2002.
3) YongKang Zhu, David H. Albonesi and Alper Buyuktosunoglu, "A High Performance, Energy Efficient GALS Processor Microarchitecture with Reduced Implementation Complexity," (ISPASS-2005), 2005.
4) Anoop Iyer, Diana Marculescu, "Power Efficiency of voltage Scaling in Multiple Clock, Multiple Voltage Cores," (ICCAD02), 2002.
5) Anoop Iyer, Diana Marculescu, "Power and Performance Evaluation of Globally Asynchronous Locally Synchronous Processors," (ISCA02), 2002.
6) Diana Marculescu, "Application Adaptive Energy Efficient Clustered Architectures," (ISLPED 2004), 2004.
7) David Brooks, Vivek Tiwari, and Margaret Martonosi, "Wattch: A Framework for Architectural-Level Power Analysis and Optimizations," (ISCA00), 2000.
8) Doug Burger, Todd M. Austin, and Steve Bennett, "Evaluating Future Microprocessors: The Simplescalar Tool Set," Technical Report CS-TR-1996-1308, 1996.
9) Glenn Hinton, Dave Sager, Mike Upton, Darrell Boggs, Doug Carmean, Alan Kyker, and Patrice Roussel, "The Microarchitecture Of The Pentium 4 Processor," Intel Technology Journal Q1, 2001.
10) "SPEC CPU2000 Benchmarks," http://www.spec.org.
11) Peter G. Sassone and D. Scott Wills, "Dynamic Strands: Collapsing Speculative Dependence Chains for Reducing Pipeline Communication," (MICRO37), 2004.
12) Tiberiu Chelcea and Steven M. Nowick, "A Low-Latency FIFO for Mixed-Clock Systems," (WVLSI'00), 2001.
13) Kenneth M. Wilson, Kunle Olukotun, and Mendel Rosenblum, "Increasing Cache Port Efficiency for Dynamic Superscalar Microprocessors," (ISCA96), 1996.