2006－ARC－170 (12)
2006／11／30

命令の並列性と逐次性を利用したクラスタ型プロセッサの命令ステアリング方式

佐藤 幸紀[†‡], 鈴木 健一[†], 中村 維男[†]
東北大学大学院情報科学研究科[†]
ファインアーク（株）仙台ソフトウェア開発センター[‡]

クラスタ型プロセッサは、スーパースカラ方式の広域的な構造を複数の局所性をもつPEに分割しクラスタ化した構成であり、将来有望なマイクロアーキテクチャとして注目を集めている。しかしながら、クラスタ化を行うことは、PE間の通信や負荷不均衡を誘発するため、プロセッサの性能を低下させる可能性がある。本論文では、プログラムに内在する命令間のデータ依存関係に着目し、クラスタ化による性能低下を最小限に抑えた命令ステアリング手法を検討する。オペランドの状態とレジスタファンアウトという指標に着目してデータ依存関係のある命令間の距離が短い場合を効果的に処理することを狙った結果、他の手法よりもIPCが改善することが分かった。

Instruction Steering Schemes for Clustered Processors using Parallel and Serial Properties of Instructions in Programs

Yukinori SATO[†‡] and Ken-ichi Suzuki[†] and Tadao NAKAMURA[†]
[†]Graduate School of Information Sciences, Tohoku University
[‡]Sendai Software Development Center, FineArh Inc.

Recently, clustering the complex and centralized structures of superscalar processors into localized PEs (clusters) becomes a popular approach to increase the performance. However, clustered processors induce inter-PE communication and workload imbalance among PEs, which will cause the performance degradation. In this paper, we investigate the features of programs to perform efficient parallel processing with localized PEs, and attempt to apply the general properties of instruction sequences to instruction steering schemes. The result shows our proposed schemes perform better IPC than other schemes so far considered.

1. はじめに

近年、クラスタ型スーパースカラプロセッサは高速処理性と低消費電力性の両立を目指すプロセッサアーキテクチャとして注目を集めている。クラスタ型プロセッサにおいて、スーパースカラ方式のデータパスを構築する単一レジスタファイルや自由度の高い演算資源間ネットワーク等の広域的かつ複雑であった構造は、分割され局所化された単純な処理要素（PE）として編成される。クラスタ化によりデータパスを構築する各要素におけるエントリ数やポート数が減少するため、要素を構成する回路の構造は単純化される。単純化された回路によりプロセッサを構成可能であることは、低消費電力性と高速処理性の両面で有効である。

例えば、低消費電力化については、レジスタファイルに代表されるマルチポート化されたレジスタセルを用いる要素はエントリ数やポート数が増加した場合、消費電力を線形以上に増加させることが知られている。したがって、クラスタ化により各要素のエントリ数やポート数を減少させることは低消費電力化に寄与する。また、クラスタ化は、回路の動作速度の観点からも、ポート数やエントリ数の削減による高速化が期待できる点や、スーパースカラ方式において問題となっていた演算資源間での結果の直接転送を行うバイパッシング回路の遅延を減少する点で有効である [1]。

しかしながら、クラスタ化によりデータパスの分割を行った場合、PE間の通信や負荷の不均衡が発生し、性能低下を引き起こす可能性がある。そこで、クラスタ型プロセッサにおいて高い性能を発揮するためには、プログラム中に内在する処理の局所性を踏まえて、PE間通信が増加しないように命令をステアリングする必要がある。そこで、本論文では、プログラムにおける逐次性の性質を踏まえつつ効率的に並列性を抽出し [2][3]、クラスタ型スーパースカラプロセッサ上において局所性を利用した効果的な処理を行うことを目指す。

2. 実験条件

図1に本論文で想定するクラスタ型スーパースカラプロセッサを示す。ここで、クラスタ型プロセッサの構成はX*Y（X:PE数、Y:PE内の命令発行幅）と表される。図1に示すようにレジスタファイルは

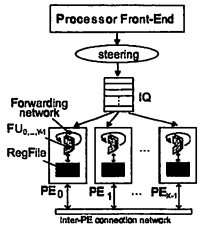

図 1: X*Y 構成のクラスタ型スーパースカラプロセッサ

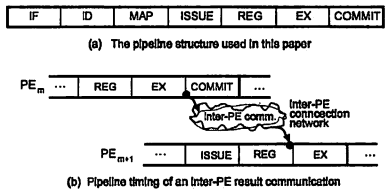

図 2: パイプラインの構造

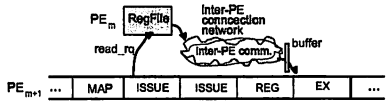

図 3: PE 間レジスタの読み込み

各 PE に一つずつ分割され、各 PE におけるレジスタファイルには個別のレジスタインスタンスが保持される非重複分散レジスタファイル構成とした。また、全ての PE が 1 つの IQ を共有する構成とした。

図 2 (a) に Alpha21264 のパイプラインの構成をベースとした本論文で想定するパイプライン構成を示す。クラスタ化に対応するために、MAP ステージにおいて命令ステアリング機構がそれぞれの命令に対して実行する PE の割付けを行うと同時に、演算結果の出力先のレジスタを割付けると変更した。命令は MAP ステージにおいてステアリングされた後、命令キュー (IQ) に格納され、ISSUE ステージにてオペランドが利用可能であれば命令は wakeup される。wakeup された命令は対応する PE の演算資源が利用可能であれば select され、REG ステージにおいてレジスタが読み込まれた後、EX ステージにおいて実行される。

入力オペランドがまだ利用可能でない場合、同一 PE 内では入力オペランドが利用可能になった直後のサイクルにおいてフォワーディングロジックにより結果を利用できるため、逐次的に処理を進行することが出来る。しかし、異なる PE からの結果を必要とするオペランドを持つ場合は、同一 PE 内でのフォワーディングと比べて図 2 (b) に示すように 2 サイクル余分に必要と想定した。重複分散方式と非重複分散方式のどちらの構成においても、未解決オペランドの結果のフォワーディングに対する遅延は等しくなる。

非重複分散レジスタファイル方式の場合、割付けられた PE 内のレジスタファイルに命令の実行に必要な入力オペランドが存在しない場合があるため、他 PE のレジスタファイルとの通信を必要とする。本論文では、この通信にかかる遅延を 1 サイクルと想定した。図 3 は他 PE からのレジスタ読み込みのための PE 間通信のタイミングを示す。

プログラムの性質やクラスタ型スーパースカラプロセッサを評価する実験は、SimpleScalarV4 ツールセットの sim-alpha[4] をベースとするサイクル精度の実行駆動型シミュレータを用いて行った。本シミュレータの主要なアーキテクチャパラメタを表 1 に示す。残りの構成や、キャッシュ、機能ユニットの遅延は alpha21264 と同様とした。

表 1: 主要なアーキテクチャパラメタ

Fetch and decode	8 instructions per cycle
Branch predictor	Tournament branch predictor
IQ, FQ, LQ; SQ size	64
ROB size	256
The number of total issues	8
Icache	128kB, 2way
Dcache	128kB, 2way

MediaBench より 4 つのベンチマーク (djpeg, cjpeg, rawdaudio, rawcaudio) を、SPEC2000CPUint より 7 つのベンチマークを (gzip, vpr, gcc, mcf, perlbmk, bzip, twolf) を選び実験を行った。全てのベンチマークは Compaq C compiler v6.5 により -O4 -fast -non_shared オプションを用いてコンパイルされた。MediaBench の各プログラムは終了するまで命令の実行を行った。SPEC2000int の各プログラムは 1G 命令フォワードした後の 100M 命令の実行を行った。

3. 命令の並列性と逐次性の解析

命令レベルの並列処理を行っていく場合、命令列の実行サイクル数はクリティカルパスと呼ばれる最も長い命令列によって決定される。図 4 にクリティカルパスの概念図を示す。クラスタ化スーパースカラプロセッサにおいては、命令ステアリングによりいかなる命令間にもリソース競合や PE 間通信が発生してしまう可能性がある。命令ステアリングの結果によってクリティカルパス上にある命令の実行が遅れてしまう場合、クラスタ化を行わなかった場合

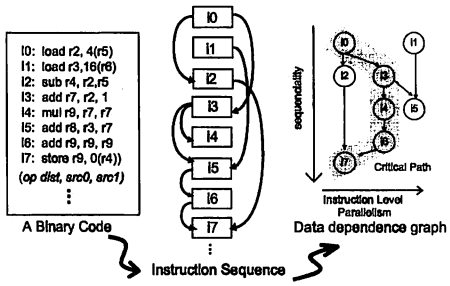

図 4: 命令列における並列性と逐次性の関係

と比べて IPC が低下する。逆に、適切な命令ステアリングを行うことによりクラスタ化による IPC 低下を抑えることが可能となる。

本論文では、適切な命令ステアリング方式を実現するために、プログラムにおける命令の並列性と逐次性の傾向の解析を行う。プログラムにおけるデータ依存関係を定量的に評価するための尺度として、命令中のレジスタを媒介としたデータ依存関係のある命令間の距離がある。プログラムが 1 命令ずつ逐次的に実行されていくとすると、任意の 2 つの命令間の実行の間には時間的な距離が発生する。そこで、データ依存関係のある 2 つの命令間の距離をその間に実行された命令数と定義する。この命令間の距離をデータ依存のある命令間において求めたものが、依存のある命令間の距離である [2]。依存のある命令間距離はレジスタに値が書き込まれてから、そのレジスタが読まれるまでに実行される命令数を測定することにより求めることが可能である。例えば、図 4 において、命令 i2 は命令 i0 に依存しており、距離は 2 である。

スーパースカラプロセッサにおいて、命令間の距離が短い依存関係はバイパッシング機構を用いて処理が行われる。バイパッシング機構により、演算結果を次のサイクルで依存先の命令の入力として利用することができる。先行研究 [5][2] において、オペランドの大部分は生成された直後に使用されるという報告がある。また、SPEC2000 ベンチマークにおいて、平均 66%の命令が演算結果を直後の演算器に転送するバイパスロジックを利用してオペランドを得るという報告もある [6]。一方で、演算結果のバイパッシングに必要となるサイクル数が増加した場合、すなわち演算結果が次のサイクルで利用できなくなった場合、性能が大きく低下することが報告されている [7][8]。従って、命令間の距離が短い依存関係を効率良く実行することが重要である。

クラスタ型スーパースカラプロセッサは、クラスタ化を行うことにより、スーパースカラ方式で問題とされていたバイパッシング機構の遅延の増加の問題は解決された [1]。一方で、クラスタ化した場合、命令ステアリングによっては命令間の距離が短い依存関係の間にも PE 間通信やリソース競合が発生する可能性がある。そこで、クラスタ型スーパースカラプロセッサにおいて処理時間を短縮するためには、命令間の距離が短いデータ依存のある命令から構成される命令列をいかに遅延を発生させないで実行する方式が必要である。

依存のある命令間の距離が短い命令の集合は、オペランドの状態とレジスタファンアウトによりさらに細分化することが可能となる。本章では、プログラムにおいてこれらの 2 つの指標がどのような傾向をもつかを調べる。

オペランドの状態は、アウトオブオーダー実行のためにパイプライン上でオペランドをフェッチする際において常に監視されている状態であり、入力オペランドが利用可能 (ready) か利用可能でない (unready) の 2 種類の状態がある。アウトオブオーダー方式においては、依存のあるオペランドが既に利用可能である場合は即座に実行可能となる。図 4 のデータ依存グラフにおいて、命令 i3 まで実行が完了しているとすると、命令 i4、i5 の入力オペランドは ready 状態であり、命令 i6 の入力オペランドは unready 状態である。

図 5 に依存のある命令間の距離の分布を調べた結果を示す。データ依存関係のある命令間の距離は、レジスタを介した命令間に真依存 (Read After Write dependencies) がある場合のみを計測し、メモリを介した依存は計測が困難なため除外した。縦軸はオペランドの 2 種類 (ready、unready) の状態毎に全実行命令数で正規化した依存のある命令の割合を SPEC2000CINT において算術平均したものであり、横軸は命令間の距離を示す。

結果より、unready のオペランドは依存がある命令との距離が短く、ready のオペランドは依存のある命令との距離が長い傾向があることが分かる。従って、依存のある命令間の距離はオペランドとの状態と強い相関関係があることが推測される。同時に、動的にオペランドの状態の分布を調査した結果、少なくとも 1 つの unready のオペランドを持つ命令は 75%を占め、逆に全てのオペランドが利用可能にある命令は 25%を占めることが分かった。

レジスタファンアウトはあるレジスタに値が書きこまれてから、そのレジスタにある値が使われるま

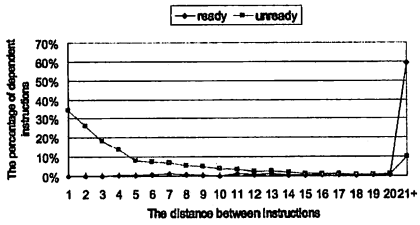

図 5: SPEC2000CINT におけるオペランドの状態毎の依存のある命令間距離の分布

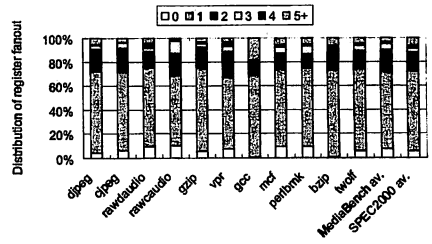

図 6: レジスタファンアウトの分布

での回数と定義される [9]。一度レジスタに書き込まれた値は、同じレジスタに再度書き込みが行われるまで何度でも読み込まれる可能性があり、レジスタファンアウトは、そのレジスタにある値が後続の命令においてどの程度使われるかの尺度となる。図 4 はバイナリコードと対応する命令列における依存のある命令間の距離とレジスタファンアウトの関係を示す。依存のある命令は矢印で結ばれている。例えば、命令 i2 は命令 i0 に依存しており、命令 i3 のレジスタファンアウトは 2 である。図 4 のデータ依存グラフにおいて、依存のない独立な命令は水平方向に展開され、依存のある命令はその命令の垂直方向に展開される。ここで、ある命令のレジスタファンアウトが 1 を越える場合、その後続命令は並列実行が可能となる。

図 6 にレジスタファンアウトの分布を調べた結果を示す。レジスタファンアウトが 1 である命令は最も多く、平均 65%となった。互いに依存のある命令においてレジスタファンアウトが 1 であるという場合、その 2 つの命令は並列実行が不可能であり、逐次的に実行する必要がある。レジスタファンアウトが 2 である命令は平均 15%を占め、また、レジスタファンアウトが 0 となる命令を含めるとレジスタファンアウトが 2 以下の命令は全体の 9 割を占めることが分かった。

4. 未解決オペランドを優先する命令ステアリング

クラスタ型スーパースカラプロセッサにおいて、命令間の距離が短いデータ依存のある命令列を効率良く処理する方式として、!ready(not ready) 命令ステアリング方式を提案する。!ready 方式においては、命令間の距離が短いデータ依存関係のある命令はオペランドの状態が未解決である可能性が高いこと利用する。オペランドの状態を依存のある命令間の距離の尺度とするのは、依存のある命令間の距離を常に観測することは距離を保持するテーブルが巨大になるために現実的ではないためである。!ready 方式は未解決オペランドを持つ命令をクリティカルパス上に存在する命令として優先的に PE 間通信やリソース競合を起こさないようにステアリングする。具体的には、クリティカルな命令である未解決なオペランドを少なくとも一つ持つ命令はそのオペランドが生成される PE に割付け、非クリティカルな命令である解決済オペランドからなる命令は DCOUNT 指標 [10] により推測した負荷最小の PE に割付ける。

比較対象としての dep_based、Advanced_RMBS の 2 つを取り上げる。データ依存に基づく方式 (dep_based) は、ある命令のオペランドに依存がある場合、必ず依存のある先行命令が割付けられている PE のいずれかに命令を割付ける方式である。負荷分散を追加した依存関係に基づく方式 (Advanced_RMBS) [10] は、dep_base 方式の問題点であった負荷集中による性能が低下を回避するため、dep_based 方式に DCOUNT という各 PE にディスパッチされた命令の差分を示すカウンタを用いて負荷の状況を監視し、負荷の集中が起こったとみなした際に負荷が最小の PE に命令を割付ける方式である。

図 7 は 8*1 構成における命令ステアリング方式毎の IPC の結果である。ここで、各 PE は 40 本のレジスタを持つとした。実験結果より、!ready 方式は dep_based 方式や advanced RMBS 方式より IPC が 1 割ほど高いことが分かる。これは、未解決オペランドを持つ命令を優先的に同一 PE 内で処理することにより、プログラムの処理時間を決定するクリティカルパス上の処理をオーバーヘッドなしで実行できたためである。

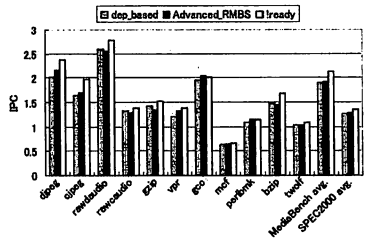

図 7: オペランドの状態に基づく命令ステアリング方式

5. 隣接 PE 間の協調

未解決オペランドを持つ命令を優先的に処理するステアリングは、全てのオペランドが利用可能にある命令をクリティカルパス上にないと判断し、負荷分散を行うことにより IPC が向上することが分かった。しかしながら、オペランドの状態の分布において、全てのオペランドが利用可能にある命令は 25%を占めるにすぎないため、依然として残りの 75%を占める未解決オペランドを持つ命令同士がリソース競合を起こしていることが推測される。

例えば、図 4 において命令 i3 の結果が未解決である場合、Iready 方式では i4 と i5 は同一 PE に割付けられる。しかしながら、クラスタ化により PE 内で同時実行可能な命令数は制限されるため、リソース競合のため i5 の実行が遅れてしまう場合がある。例えば、8*1 構成のクラスタ型プロセッサの場合、PE 内での同時実行可能な命令数は 1 であるため、レジスタファンアウトが 2 以上の命令の後続でリソース競合が発生する。図 6 のようにレジスタファンアウトが 2 以上のものは 30%を占めるため、レジスタファンアウトが 2 以上の場合も効率良く実行可能にすることも負荷分散において重要である。

そこで、同一の未解決オペランドを持つ命令同士のリソース競合を防ぐことを目的として、隣接する PE が協調して処理を行う方式を提案する。この隣接 PE 間の協調は、隣接 PE 間に 1 方向の局所的な通信経路を追加すること、隣接 PE 間協調を支援するように命令をステアリングすることから成り立つ。1 方向の隣接 PE 間の通信経路とした理由は、ハードウェア的なコストの面とプログラムにおける命令列の特性の傾向を踏まえたことによる。隣接 PE 間に 1 方向の局所的な通信経路を追加するコストは、隣接 PE 間の物理的な距離が配線遅延が問題にならない程度に短く複雑度も低いことから、十分に小さいと考えられる。また、レジスタファンアウトが 2 以下の命令が 90%を占めることから、レジスタファンアウトが 2 である命令の転送を想定した 1 方向の通信経路で十分と考えられる。

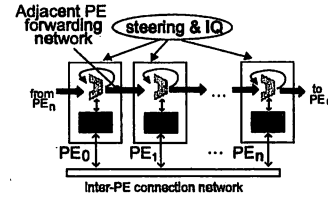

図 8: Adjacent PE Forwarding (AF) 構成

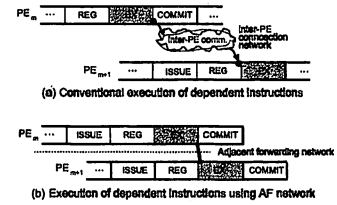

図 9: Executions of dependent instructions.

図 8 は隣接 PE 間の協調を行うための構成を示す。ある PE における演算の結果を隣接 PE に直接バイパスする通信経路がすべての隣接 PE 間に追加され、1 次元配列のループを形成するような配置となる。図 9 は、依存のある命令が隣接する 2 つの PE で実行される様子を示す。従来の構成においては依存のある命令の実行の間に PE 間通信が発生してしまうが、AF 構成においては依存のある 2 つの命令が連続するサイクルで実行できるようになる。

隣接 PE 間に通信経路を有効に活用するために、レジスタファンアウトの数に基づく命令ステアリング手法を提案する。この命令ステアリング手法はレジスタファンアウトが 1 より大きい場合の後続命令を並列に実行することが目的であり、レジスタファンアウトを動的に観測することにより以下のように実現する。未解決オペランドを持つ命令はレジスタファンアウトの値により、未解決オペランドを生成する PE あるいはその右側に隣接する PE に命令を割付ける。未解決オペランドをまったく持たない命令は負荷最小の PE に割付ける。

レジスタファンアウトのスレッショルド値によりレジスタファンアウトが 2 の命令のみ隣接 PE にステアリングする FO2 と、レジスタファンアウトが 2 または 4 のときに隣接 PE にステアリングする FO4 の 2 つの命令ステアリング方式を提案する。FO2 方式は FO4 方式より隣接する PE におけるリソース競

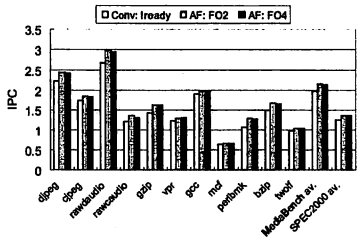

図 10: 隣接 PE 間の協調の効果

合を抑制することが見込まれるが、FO4 方式より多くの PE 間通信によるロスは発生させる。

図 10 は、8*1 構成において隣接 PE 間の協調を評価した結果を示す。各 PE のレジスタ数を 32 とした。図中の conv は従来の構成を示し、AF は隣接 PE 間にフォアーディング機構を追加した構成を示す。また、各構成の後ろに利用したの命令ステアリング方式が示されている。

隣接 PE 間の協調を行うことにより従来の構成と比べて 1 割ほど IPC が向上していることが分かる。また、FO2 ステアリングと FO4 ステアリングを比べた場合、FO2 ステアリングの方が高い IPC を達成できることが分かった。これは、隣接する PE に多くの命令が割付けられると、隣接する PE における本来の処理のために必要なリソースを奪ってしまい、IPC が低下してしまうためと思われる。

隣接 PE 間の協調の効果を議論するために、クラスタ化により発生した命令のストールの原因の内訳を調べた。図 11 に 1 サイクルあたりの命令ストール数の平均を示す。結果より、AF 構造は従来の構造と比べてリソース競合のためのストールのを減らしていることが分かる。また、AF 構造においては FO2 命令ステアリング方式は隣接する PE におけるリソース競合を、FO4 方式は PE 間通信のためのストールをより減らしていることが分かる。しかしながら、隣接 PE のリソース競合のほうが実行のクリティカルパスを延ばす傾向があると思われるため、FO2 方式の方が良い結果となったとなったと考察される。

6. 結論

本論文では、クラスタ型スーパースカラプロセッサの IPC に大きく影響を与える命令間の距離が短い依存関係を効率良く実行するために、オペランドの状態とレジスタファンアウトに着目した命令ステアリングや隣接 PE 間における協調処理方式を提案した。実験の結果、提案手法は従来方式と比べて命令

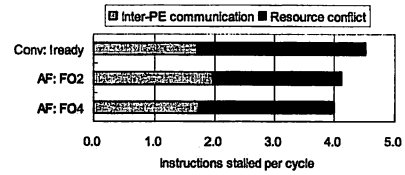

図 11: 1 サイクルあたりの平均命令ストール数

間の距離が短い依存関係を適切に処理するため IPC が向上することが分かった。

参考文献

[1] Subbarao Palacharla, Norman P. Jouppi, and J. E. Smith. Complexity-effective superscalar processors. In *Proceedings of the 24th annual international symposium on Computer architecture*, pp. 206–218, 1997.

[2] 佐藤幸紀, 鈴木健一, 中村維男. プログラムにおける命令の並列性と逐次性について. 情報処理学会研究会報告 2004-ARC-157, pp. 121–132, 2004.

[3] 佐藤寿倫. 命令レベル逐次プロセッサ. 情報処理学会研究報告 2006-ARC-169, pp. 49–54, 2006.

[4] Rajagopalan Desikan, Doug Burger, and Stephen W. Keckler. Measuring experimental error in microprocessor simulation. In *Proceedings of the 28th annual international symposium on Computer architecture*, pp. 266–277, 2001.

[5] Manoj Franklin and Gurindar S. Sohi. Register traffic analysis for streamlining inter-operation communication in fine-grain parallel processors. In *Proceedings of the 25th annual international symposium on Microarchitecture*, pp. 236–245, 1992.

[6] Il Park, Michael D. Powell, and T. N. Vijaykumar. Reducing register ports for higher speed and lower energy. In *Proceedings of the 35th annual ACM/IEEE international symposium on Microarchitecture*, pp. 171–182, 2002.

[7] M. S. Hrishikesh, Doug Burger, Norman P. Jouppi, Stephen W. Keckler, Keith I. Farkas, and Premkishore Shivakumar. The optimal logic depth per pipeline stage is 6 to 8 fo4 inverter delays. In *Proceedings of the 29th annual international symposium on Computer architecture*, pp. 14–24, 2002.

[8] Eric Sprangle and Doug Carmean. Increasing processor performance by implementing deeper pipelines. In *Proceedings of the 29th annual international symposium on Computer architecture*, pp. 25–34, 2002.

[9] K. Sankaralingam, R. Nagarajan, S.W. Keckler, and D.C. Burger. A technology-scalable architecture for fast clocks and high ILP. In *5th Workshop on the Interaction Between Compilers and Computer Architectures*, 2001.

[10] Joan-Manuel Parcerisa and Antonio González. Reducing wire delay penalty through value prediction. In *Proceedings of the 33rd annual international symposium on Microarchitecture*, pp. 317–326, 2000.