

DSN 型スーパースカラ・プロセッサ・プロトタイプ — 分岐およびロード／ストア・アーキテクチャの評価 —

納富 昭† 久我守弘† 村上和彰† 富田眞治†

†: 九州大学 大学院総合理工学研究科

‡: 京都大学 工学部

動的ハザード解消, 静的コード・スケジューリング, 非均質機能ユニットという基本的な性質を有するスーパースカラ・プロセッサ: *DSNS* (*Dynamically-hazard-resolved, Statically-code-scheduled, Nonuniform Superscalar*) プロセッサを現在開発中である。

分岐命令の存在に起因する分岐ハザードの影響(分岐ペナルティ)を軽減するため, 分岐アーキテクチャとして, (1) 静的分岐予測+分岐先バッファ(BTB), (2) 投機的実行, (3) 先行条件決定方式, (4) 早期分岐解消, といった方式を採用している。また, スーパースカラ度に見合った十分なデータ・バンド幅を供給するため, ロード／ストア・アーキテクチャとして, (1) 実行順序指定可能なロード／ストア命令アーキテクチャ, (2) デュアルポート・ノンブロッキング・データキャッシュ, という手法を用いている。

本稿では, これら特徴的なアーキテクチャについて述べるとともに, その性能をシミュレーションにより評価する。

DSNS Processor Prototype

— Evaluation of the Branch and the Load/Store Architectures —

Akira NOUDOMI†, Morihiro KUGA†, Kazuaki MURAKAMI†, and Shinji TOMITA‡

†: Department of Information Systems

Interdisciplinary Graduate School of Engineering Sciences

Kyushu University

6-1 Kasuga-koen, Kasuga-shi, Fukuoka 816 Japan

‡: Kyoto University

E-mail: {noudomi, kuga, murakami}@is.kyushu-u.ac.jp

We are developing *DSNS* (*Dynamically-hazard-resolved, Statically-code-scheduled, Nonuniform Superscalar*) processor prototype.

To alleviate the effect of control hazards due to branches, *DSNS* processor prototype adopts (1) static branch prediction with branch-target-buffer, (2) speculative execution, (3) advanced conditioning, and (4) early branch resolution. To expand data memory bandwidth, it also adopts (1) load/store instruction-set architecture which explicitly specifies the execution order of load/store instructions, and (2) dual-port nonblocking data cache.

This paper presents and evaluates these architectural features.

1 はじめに

命令レベルの並列性を活用して単一プロセッサの性能を向上させるアーキテクチャとしてスーパースカラ方式[7]に着目し、現在試作機を開発中である。本プロセッサは以下のような特長を持つ[1]。

1. 動的ハザード解消 (D: Dynamically-hazard-resolved): 命令間依存関係 (データ依存, 制御依存) および資源競合に起因するハザードを実行時にハードウェアで検出・解消する。これにより、スーパースカラ度の異なるプロセッサ間でのオブジェクトコードの可搬性を保証する。
2. 静的コード・スケジューリング (S: Statically-code-scheduled): 資源競合や命令間依存関係が存在すると、同時に実行できる命令数は減り、命令レベル並列度が減少する。命令がパイプラインを滞りなく流れるためには、命令実行順序の並べ換え (コード・スケジューリング) を行い、同時に実行できる命令数を増やす必要がある。このコード・スケジューリングをコンパイル時に静的に行う。これにより並列度の増加を図ると同時に、動的コード・スケジューリングにより生じるハードウェアの増加を回避する。
3. 非均質機能ユニット (N: Nonuniform): ハードウェア・レベルでは、スーパースカラ度に関するスケラビリティを保証しない。よって、命令フェッチ機構、デコード機構、レジスタ・ポートなどは、スーパースカラの多重度に従って多重化しているが、機能ユニットについては必要なものを多重化している。使用頻度の高いユニットのみを多重化しているので、コスト/パフォーマンスが良い。

上記の特長に基づき、現在開発中のプロセッサを DSNS (Dynamically-hazard-resolved, Statically-code-scheduled, Nonuniform Superscalar) プロセッサと呼んでいる [9]。

プロセッサのハードウェア設計と並行して、ソフトウェア・シミュレーションによる評価を行っている。先に、評価項目として静的コード・スケジューリングとアーキテクチャとの相乗効果に着目して評価を行った結果を報告した [5]。本稿では、DSNS プロセッサの分岐アーキテクチャ [3] およびロード/ストア・アーキテクチャ [4] について、その特徴的な機能を評価する。

本稿では、まず2章で、DSNS プロセッサのハードウェア構成を概観する。次に、3章および4章で、DSNS プロセッサの特徴である分岐アーキテクチャおよびロード/ストア・アーキテクチャをそれぞれ述べる。そして、5章で、評価結果を示し考察を行う。

2 DSNS プロセッサの概要

命令パイプラインは、

- IF: 命令ブロック・フェッチ+プリデコード
- D: コンフリクト・チェック+デコード+オペランド・フェッチ
- E: 実行
- W: 書込み

の4ステージ構成である [9]。パイプライン・サイクルは 60ns を目標にしている。

DSNS プロセッサは、図1に示すように、以下の主要ユニットから構成される [9]。

1. 命令キャッシュ (IC: Instruction Cache)

ポートサイズ 16 バイトで、4 バイト長命令 4 個から成る命令ブロックが一時にフェッチ可能である。

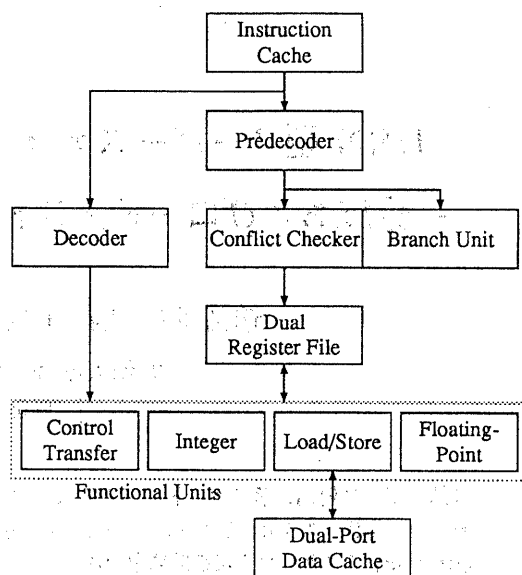


図 1: DSNS プロセッサの構成

ラインサイズ 64 バイト (=4 命令ブロック)、キャッシュサイズ 512K バイトのダイレクト・マッピング方式の仮想アドレス・キャッシュである。

命令ブロックごとに 1 個の予測分岐先アドレスを保持する分岐先バッファ (BTB: Branch Target Buffer) を備える。これにより、分岐命令の有無に関わらず、命令ブロックの連続フェッチが可能となる。BTB は、命令キャッシュとタグアレイを共用する。

2. プリデコーダ (PD: PreDecoder)

フェッチした 4 命令をプリデコードして、コンフリクト・チェックに必要な情報を得る。また、分岐命令実行に関するすべての情報もここで生成する。

3. デコーダ (D: Decoder)

各命令の実行制御に必要な情報、特に機能ユニットの制御情報を ROM から読み出す。

4. コンフリクト・チェッカ (CC: Conflict Checker)

プリデコード結果に基づき、最大 4 命令に対して以下の処理を行う。

- 命令ブロック内の命令間のフロー依存および出力依存の検出
- 先行命令ブロックに対するフロー依存および出力依存の検出
- 機能ユニット競合の検出と調停
- レジスタ・ファイルの読出しポートの割当て

5. 分岐ユニット (BU: Branch Unit)

早期分岐解消を行うための分岐命令実行専用ユニットで、3 段のパイプライン構成となっている。

6. 機能ユニット (FUs: Functional Units)

以下の 4 系統、計 13 個の機能ユニットを備える。最大 4 命令が毎サイクル、任意の機能ユニットの組合せに対して発行可能である。

- (a) 整数系機能ユニット: ALU (×2)、シフタ、乗算器
- (b) 浮動小数点系機能ユニット: ALU、乗算器、除算器、型変換器

- (c) ロード/ストア・ユニット：2つの独立したロード/ストア・ユニット
- (d) 分岐系機能ユニット：再フェッチ・アドレス生成器、先行条件決定方式のための2つのユニット

浮動小数点の除算器を除くすべての機能ユニットはパイプライン化されており、1サイクル毎（ただし、倍精度の浮動小数点演算は2サイクル毎）に実行結果を得ることができる。また、実行結果を次サイクルの演算のオペランドとして使用可能とするために、バイパスを備える。

7. 2重化レジスタ・ファイル (DRFs: Dual Register Files)

以下の3系統のレジスタ・ファイルを備える。3.2節で述べるように、レジスタの各エントリは2重化されている。そのため、下記のレジスタ個数は論理的なものである。

- (a) 汎用レジスタ・ファイル (GPRF: General-Purpose Register File)：32ビット長レジスタ32個から成る。読出しポート8、書込みポート2を備える。各レジスタには、コンディションを格納する4ビットのタグがある。
 - (b) 浮動小数点レジスタ・ファイル (FPRF: Floating-Point Register File)：64ビット長レジスタ32個から成る。読出しポート4、書込みポート2を備える。GPRF同様、各レジスタは4ビットのタグを持つ。
 - (c) TFレジスタ・ファイル (TFRF: True/False Register File)：1ビット長レジスタ32個から成る。読出しポート3、書込みポート1を備える。
- #### 8. デュアルポート・データキャッシュ (DPDC: Dual-Port Data Cache)

4.2節で述べるように、スーパースカラ度に見合ったデータ供給能力を確保するため、デュアルポート化およびノンブロッキング化を行っている。ポートサイズは、最長データである倍精度浮動小数点データに対応して8バイトである。ラインサイズ32バイト、キャッシュサイズ512Kバイトのダイレクト・マッピング方式の仮想アドレス・キャッシュである。主記憶との間の転送データ幅は8バイトであり、コピーバック方式を採用している。

3 分岐アーキテクチャ

分岐命令の存在に起因する分岐ハザードの影響（分岐ペナルティ）を軽減する方法として、DSNSプロセッサでは以下の手法を採用している [3]。

3.1 静的分岐予測十分岐先バッファ

命令フェッチを中断することなく連続して行う方法として、分岐先バッファ (BTB: Branch Target Buffer) [10] を採用した。従来、BTBは動的な分岐予測と組み合わせて用いられていたが、DSNSプロセッサでは静的分岐予測と組み合わせる。コンパイラは静的分岐予測結果に基づいて、個々の分岐命令に対して以下のいずれかの型を指定する。

- BTB登録型：taken予測、かつ、分岐先アドレスが不変であると判断された分岐命令に対して指定する。実行時に、生成した分岐先アドレスをBTBに登録する。

- BTB非登録型：untaken予測、または、taken予測だが分岐先アドレスが変化すると判断された分岐命令に対して指定する。分岐先アドレスはBTBに登録されない。

BTBに分岐先アドレスが登録されると、対応する分岐命令はtakenと予測され、その登録分岐先アドレスが次サイクルの命令フェッチに使用される。

3.2 投機的実行

DSNSプロセッサでは、制御依存関係にある命令の投機的実行 (speculative execution) を許す。すなわち、制御依存の解消を待たずに、当該制御依存関係にある命令の実行を開始する。その具体的な実現方式として、以下の2方式を採用している。

1. 条件付実行モード (conditional mode)：分岐予測によりフェッチされた命令は、対応する分岐命令に起因する制御依存が解消するまで条件付実行モード下に置かれる。条件付実行モード下にある命令は、実行することは可能だが、実行結果でレジスタ内容等を更新することはできない。
2. ブースティング (boosting) [11]：個々の命令に対して、条件付実行モードに置くか否かをコンパイラが決定する。条件付実行モードに置かれた命令をブースト (boosted) 命令と呼び、op-codeによりその指定を行う。ブースト命令はオブジェクト・コード上、対応する分岐命令より前に位置する。すなわち、上記1とは逆に、条件付実行モード下の命令が対応する分岐命令より先にフェッチされ実行を開始する。よって、静的コード・スケジューリング次第では、上記1より投機的実行の効果が期待できる。

上記2方式のハードウェア実現方法は基本的に等価である。すなわち、条件付実行モードのためのハードウェア機構がブースティングにも流用できる。

ブースト命令を含む条件付実行モード下の命令が制御依存の解消を待たずにレジスタ内容等を変更するのを防ぐ手段として、DSNSプロセッサではレジスタ・ファイルが多重化している [3]。ここで、レジスタ・ファイルの多重度は、「条件付実行モードの最大レベル+1」となる。DSNSプロセッサの条件付実行モードの最大レベル数は1であるので、多重度は2となる。このことから、DSNSプロセッサのレジスタ・ファイルを2重化レジスタ・ファイル (DRF: Dual Register File) と呼ぶ。

2重化レジスタ・ファイルにおいては、各レジスタは論理的には1個であるが、物理的には2個の実体（物理レジスタ）を有する。そして、一時には、一方がカレント (current) 状態、他方がオルタネート (alternate) 状態となる。オルタネート状態にはさらに、有効/無効の2状態が存在する。また、カレント状態、オルタネート状態にある物理レジスタを便宜上、それぞれカレント・レジスタ、オルタネート・レジスタと呼ぶ。2重化レジスタ・ファイルの動作は次の通りである。

- レジスタ書込み：無条件実行モード (unconditional mode)：制御依存関係にないこと) 下で終了した命令の実行結果は、カレント・レジスタに書き込む。一方、条件付実行モード下で終了した命令の実行結果はオルタネート・レジスタに書き込み、当該オルタネート・レジスタを有効状態とする。
- レジスタ読出し：無条件実行モード下の命令は、そのソース・オペランドをカレント・レジスタから読み出す。一方、条件付実行モード下にある命令は、そのソース・オペランドを次のようにして得る。
 - － オルタネート・レジスタが有効な場合：オルタネート・レジスタから読み出す。

- － オルタネート・レジスタが無効な場合：カレント・レジスタから読み出す。
- 分岐命令の実行終了：制御依存が解消されると、個々のレジスタについて次の状態遷移が起きる。
 - － 分岐予測が当たった場合：有効なオルタネート・レジスタがカレント状態となり、対を成すカレント・レジスタはオルタネート無効状態になる。それ以外のレジスタには状態の変化がない。
 - － 分岐予測が外れた場合：すべてのオルタネート・レジスタは無効状態となる。カレント・レジスタには状態の変化がない。

3.3 先行条件決定方式

条件分岐方式として、DSNSプロセッサでは先行条件決定方式 (*advanced conditioning*) を採用している [3]。まず、条件分岐は一般に以下の処理から成る。

1. コンディション生成：条件分岐の元になるコンディション (状態) を生成する。
2. 条件決定 (*conditioning*)：上記1で生成したコンディションを分岐条件でテストし、分岐するか否かを決定する。
3. 分岐先アドレス生成：分岐先の命令アドレスをアドレッシング・モードに従って計算する。
4. 分岐処理：上記2で生成した分岐先アドレスをプログラム・カウンタ (PC) に設定する。

先行条件決定方式では、上記1は通常の算術・論理演算命令におけるコンディション・コード (CC: *Condition Code*) 設定で行う。そして、従来の *branch-on-condition* 方式では1つの命令で行っていた上記2, 3, 4の処理をテスト命令 (2を担当) と分岐命令 (3と4を担当) の2命令に分割している。また、上記1と2の処理を一括して行う比較&テスト命令も用意している。

先行条件決定方式により、以下の効果が期待できる。

- テスト命令または比較&テスト命令と分岐命令との間の距離を大きくするようにスケジューリングすることで、条件決定に伴う分岐遅延を隠蔽できる。
- 分岐命令で行う処理が少ないことから次に述べる早期分岐解消が可能となる。その結果、分岐遅延自体を低減できる。

3.4 早期分岐解消

3.1~3.3節で述べた方法に加えて、分岐命令の実行自身の高速化、つまり、分岐の早期解消が重要である。これは、以下の理由による。

- 分岐予測が外れた場合、条件付実行モード下のすべての命令を無効化することで命令パイプラインを復元する (パイプライン・フラッシュ)。したがって、当該分岐命令に起因する制御依存の解消が遅くなるほど、分岐ペナルティが増大する。
- 条件付実行モード下で新たな分岐命令をフェッチした場合、最大1レベルの条件付実行モードでは、当該分岐命令ならびに後続命令の実行は開始できない。これは、条件付実行モードの最大レベルを越えるからである。しかし、条件付実行モードの最大レベル数の増加は、多重化レジスタ・ファイルの多重度の増加を意味し、ハードウェア量の増大に直接結びつく。よって、条件付実行モードの最大レベル数を2以上に上げるのは難しい。

このような背景から、DSNSプロセッサには早期分岐解消 (*early branch resolution*) を採用している。分岐命令以外の一般命令は命令パイプラインのE (実行) ステージで行うのに対して、分岐命令はそれよりも前のD (解説) ステージで実行を開始する。

しかし、一般命令とは異なるステージで分岐命令の実行を行うためには、余分なハードウェアが必要となると問題が生じる。ところが、DSNSプロセッサでは、3.3節で述べたように、分岐命令で行うべき処理が少なく、必要とされるハードウェア量もさほど多くはならない [3]。

4 ロード／ストア・アーキテクチャ

スーパースカラ度4に見合う十分なデータ・バンド幅を確保するために、DSNSプロセッサではロード／ストア・パイプラインを2重化し、以下の手法を用いている [4]。

4.1 ロード／ストア命令セット・アーキテクチャ

DSNSプロセッサでは、データ依存、制御依存、資源競合といったハザードの検出・解消をハードウェアが実行時に行っている。これらのハザードの中でデータ依存は、レジスタ・アクセスに関するデータ依存、および、メモリ・アクセスに関するデータ依存に大別できる。そして、レジスタ・アクセスに関するデータ依存については、レジスタ・スコアボードによりその検出・解消を行っている [2]。しかし、メモリ・アクセスに関するデータ依存への対処をレジスタと同様な方法で処理することは不可能である。

ロード／ストア命令間の依存関係を実行時に検出するためには、これから実行するロード／ストア命令の実効アドレスと、実行を終了していないすべての先行ロード／ストア命令の実効アドレスとを比較しなければならない。これをハードウェアで実現するのは非常に困難である。しかも、DSNSプロセッサではロード／ストア・ユニットを2重化しているため、実行時の依存関係検出はいっそう困難になる。そこで、DSNSプロセッサではハードウェアによる依存関係の検出は一切行わず、コンパイル時の静的な検出にすべてを任せている。コンパイラは個々のロード／ストア命令に実行順序を指定する情報を付加する。この情報により、ロード／ストア命令は、以下の3種類に分類される。

1. SO (*Strongly Ordered*)：完全逐次実行すべき命令である。静的な解析により依存関係の存在を検出できる場合、または、依存関係がないということが判断できないような場合には、そのロード／ストア命令を完全に逐次実行するよう明示する。このように明示されたロード／ストア命令同士のデータへのアクセスは *in-order* に行うことをハードウェアで保証する。
2. WO (*Weakly Ordered*)：同一データ型同士で逐次実行すべき命令である。メモリ・アクセスに関して、2種類のデータ型 (整数および浮動小数点データ) を区別する。一般に、整数データと浮動小数点データはメモリ上では異なる領域に割り当てられるので、通常は整数ロード／ストア命令と浮動小数点ロード／ストア命令の間に依存関係は存在しない。したがって、それぞれ同一データ型同士で逐次実行すべき命令であると明示しておくことにより、同一

データ型同士は逐次実行することを保証する。一方、異なるデータ型に対するロード/ストア命令に関する実行順序は何ら制限しない。

3. UO(Unordered): 完全非逐次実行可能な命令である。他の任意のロード/ストア命令に対して依存関係がないということが静的に解析できる場合には、そのロード/ストア命令は完全非逐次実行可能であると明示する。このように明示されたロード/ストア命令は他のロード/ストア命令に対して out-of-order に実行開始可能であり、実行時の追越しも許される。

DSNS プロセッサの D ステージでは、上記情報にのみ基づいて 2 つのロード/ストア・ユニットへの命令ディスパッチを制御している [4]。

4.2 デュアルポート・データキャッシュ

DSNS プロセッサには 2 本のロード/ストア・ユニットが存在し、各々が独立にデータキャッシュへのアクセスを要求する。複数のアクセス要求を高速かつ並列に処理するため、以下に示す構成を採用している [4]。

4.2.1 マルチポート・キャッシュ

複数のアクセスを同時に処理するデータキャッシュの構成として、マルチポートのメモリチップを使用する方法が考えられる。しかし、高速かつ大容量のマルチポート・メモリチップのコストは非常に高い。DSNS プロセッサを 1 チップ化する場合にも、マルチポート・メモリの面積コストは無視できない。そこで、シングルポート・メモリを用いてマルチポート化するという前提で次の 4 選択肢を検討し、最終的に ID/I 型の構成法を採用した [4]。

- D (Duplicated) 型: データアレイおよびタグアレイの完全なコピーをポート毎に設ける。
- IS (Interleaved, Shared) 型: データアレイを複数バンクに分割し、各バンクを全ポートで共有する。タグアレイについては、完全なコピーをポート毎に設ける。バンク数とポート数は無関係に決まる。
- ID (Interleaved, Dedicated) 型: データアレイを複数バンクに分割し、各バンクをいずれかのポートに専有させる。よって、バンク数とポート数は等しい。タグアレイについては、次の 2 つの選択肢が存在する。
 - ID/D (ID/Duplicated) 型: タグアレイの完全なコピーをポート毎に設ける。
 - ID/I (ID/Interleaved) 型: タグアレイもデータアレイ同様に、複数バンクに分割しポートに専有させる。

ID 型では各ポートが全ロード/ストア・ユニット間で共有されるので、ポート数はロード/ストア・ユニット数とは無関係に決まる。しかし、DSNS プロセッサでは、ロード/ストア・ユニット数と等しく 2 ポートとしている。ID/I 型は、所要ハードウェア量が最も少なく、また、各バンクをそれぞれ独立したキャッシュとして設計することができるので、ハードウェア構成は比較的簡単である。ただし、バンクのインターリーブ幅がラインサイズに等しいため、連続アドレスへのアクセスの場合バンク・コンフリクトが生じやすいという問題点がある (5 章参照)。

4.2.2 ノンブロッキング・キャッシュ

並列アクセスを可能にするとともに、キャッシュ本来の性能向上、つまり、キャッシュヒット率の向上およびキャッシュミス・ペナルティの低減化も図る必要がある。DSNS プロセッサでは、キャッシュミス・ペナルティ低減のために、デュアルポート・データキャッシュをノンブロッキング化している。ノンブロッキング・キャッシュとは、「ミスヒット処理を行っている最中でも、後続のアクセス要求を受け付け得るキャッシュ」であり、これによりミスヒット処理による遅延を隠蔽できる。

ノンブロッキング・キャッシュの実現方式としては、MSHR (Miss information/Status Holding Register) を用いた Kroft の方式がある [8]。Kroft の方式の特徴として、ミスヒット処理によるデータアレイへのアクセス回数を最小限にする、MSHR を有効に利用する、といったきめ細かなミスヒット処理が挙げられる。しかし、Kroft の方式を忠実に実装しようとする、ハードウェア量、ハードウェアの複雑度ともに大幅に増大する。そこで、DSNS プロセッサでは、ハードウェア・コストの低減を主眼として以下の構成を採った。

- MSHR 数: 各ポート当たり 2 個の MSHR を設ける。これにより、各ポート当たり最大 2 個、データキャッシュ全体では最大 4 個のミスヒット処理要求を保持することができる。
- ミスヒット処理の順番: 最大 4 個のミスヒット処理を発生順に逐次的に処理する。

5 評価

DSNS プロセッサの特長である分岐アーキテクチャ (3 章参照) およびロード/ストア・アーキテクチャ (4 章参照) について、ソフトウェア・シミュレータを用いて評価を行った。

5.1 評価方法

本シミュレータは、DSNS プロセッサの動作を機能レベルで忠実にモデル化している。これにより、毎クロック・サイクルのプロセッサ状態を忠実にシミュレーション可能である。ただし、ミスヒット処理時の主記憶アクセスは簡略化しており、一定のクロック・サイクル (=6 + ラインサイズ/8) 後にライン・リプレースが完了するものと仮定している。なお、特に断らない限り、命令キャッシュおよびデータ・キャッシュともに、cold start とした。評価項目は、次の通りである。

- 分岐アーキテクチャの評価
 - 分岐先バッファ (BTB) の効果: 命令ブロック当りの BTB エントリ数を 0 (BTB なし), 1, 2, 4 と変化させる。
- ロード/ストア・アーキテクチャの評価
 - ロード/ストア・パイプラインの性能: ロード/ストア・ユニットおよびマルチポート・データキャッシュのポート (バンク) 数を 1, 2, 4 と変化させる。また、キャッシュの構成/能力自身をえる。
 - ノンブロッキング化の効果: ブロッキング対ノンブロッキング。さらに、ノンブロッキングに関しては、ポート当りの MSHR 数を 1, 2, 3, 4 と変化させる。

ベンチマーク・プログラムとしては、LFK (Livermore Fortran Kernel) 1, 3, 5, 7, 12 の 5 つを用いた。プログラムに内在する命令レベル並列性を引き出すために、

ループ・アンローリング、ソフトウェア・パイプライン
ング、局所コード・スケジューリング、等の手法[6]を用
いた。

評価指標は、次の通りである。

- 分岐アーキテクチャの評価：命令発行率 (issue rate)
IPC (Instructions Per Clock cycle)

$$\text{命令発行率} = \frac{\text{実行命令数}}{\text{プログラム実行所要サイクル数}}$$

- ロード/ストア・アーキテクチャの評価：データ供
給バンド幅 (bandwidth)

$$\text{データ供給バンド幅} = \frac{\text{メモリ・アクセス回数}}{\text{プログラム実行所要サイクル数}}$$

5.2 分岐アーキテクチャの評価

5.2.1 分岐先バッファ(BTB)の効果

図2に、命令ブロック当りのBTBエントリ数と命令
発行率との関係を示す。なお、今回のベンチマーク・プ
ログラムは分岐命令自体の数が少なく、1命令ブロック
(4命令)中に複数の分岐命令が存在することがなかつ
た。そのため、BTBエントリ数1, 2, 4における性能に
差が出ていない。

図2から判るように、BTBを設けても0%~68% (相
乗平均12%)の低い性能向上に留まっている。これは以
下の理由による。BTBなしの場合、分岐命令はすべて
untakenと予測される。予測が外れた(つまり、分岐結
果がtakenだった)場合パイプライン・フラッシュ(条
件付実行モード下の命令の無効化)が行われるが、早期
分岐解消の効果により分岐ペナルティは1~2サイクル
と小さい。ここで、次の事実がシュミレーションにより
確認されている。すなわち、分岐命令に起因する制御依
存が解消した時点ではまだ、条件付実行モード下の命令
(すなわち、当該分岐命令の後続命令)がDステージ以
降に進んでいない場合が多い。これは、当該分岐命令の
先行命令が引き起こしたデータ依存や資源競合により、
条件付実行モード下の命令がIFステージでインターロ
ックされていることを意味する。結局、データ依存や資源
競合に起因するハザードが引き起こす遅延の方が分岐遅
延よりも大きいため、1~2サイクルの分岐ペナルティを
隠蔽していることになる。

以上の考察だけでは断定できないが、分岐命令が早期
に解消できるなら「常にuntakenと予測」するBTBな
しの静的分岐予測だけでも十分と言えるかも知れない。

5.2.2 早期分岐解消の効果

前項で述べたように、命令が条件付実行モード下でD
ステージ以降に進むことは非常に稀であることが確認さ
れた。このことは、2重化レジスタ・ファイル(3.2節参
照)のオルタネート・レジスタが用いられることがほと
んどないことを意味している。すなわち、早期分岐解消
が可能ならBTBのみならず2重化レジスタ・ファイル
も不要であると言えるかも知れない。

早期分岐解消を妨げる要因としては、(1)分岐するか
否かの条件決定の遅れ、および、(2)分岐先アドレス生
成の遅れ、の2つがある。DSNSプロセッサでは、先行
条件決定方式(3.3節参照)により、(1)の条件決定の遅
れは回避できる。一方、(2)の分岐先アドレス生成の遅
れは、アドレッシング・モードとしてレジスタ相対を用
いる場合に起こり得る。なぜなら、当該汎用レジスタに

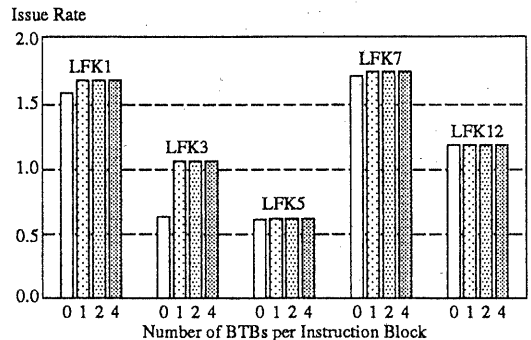


図2: 分岐先バッファ(BTB)の効果

関してデータ依存が生じる可能性があるからである。し
かし、DSNSプロセッサでは、分岐命令の前後256キ
ロ命令への分岐を可能とするPC相対アドレッシング・
モードをサポートしている。システム・コール、ライブラ
リ・コール、テーブル分岐、等を除く通常分岐命令の
大部分は、このPC相対で十分である。しかも、プログ
ラム・カウンタ(PC)に関してはデータ依存は生じ得
ないので、PC相対を用いる限り分岐先アドレス生成の
遅れは起こらない。したがって、大半の分岐命令に対
しては、早期分岐解消が可能ということになる。

このことから、DSNSプロセッサでは早期分岐解消が
極めて有効に機能していることが判る。

5.3 ロード/ストア・アーキテクチャの 評価

5.3.1 ロード/ストア・パイプラインの性能

図3に、ロード/ストア・パイプライン(LSP)の多重
度とデータ供給バンド幅との関係を示す。ここで、多重
度nのロード/ストア・パイプラインとは、ロード/ス
トア・ユニット(LSU)をn個、マルチポート・データ
キャッシュのポート(およびバンク)をn個それぞれ備
えた構成を言う。また、比較のため、以下の4種類の構
成/能力の異なるデータキャッシュをシミュレートした。

- ID/I: DSNSプロセッサで採用した構成(4.2.1項
参照)。バンク・コンフリクトやキャッシュミスが
起こり得る。
- No Conflict: バンク・コンフリクトが生じない構
成(これは、4.2.1項で述べたD型に相当する)。た
だし、キャッシュミスは起こり得る。
- Perfect Hit: キャッシュミスが生じないような完全
キャッシュ。ただし、バンク・コンフリクトは起こ
り得る。
- Ideal (Perfect Hit, No Conflict): キャッシュミス
もバンク・コンフリクトも生じない理想キャッシュ。

上記のIdeal (Perfect Hit, No Conflict)は理想的な
キャッシュであり、このときのデータ供給バンド幅の値
はプログラム自身が多重度nのロード/ストア・パイプ
ラインに要求するデータ・バンド幅に他ならない。そこ
で、この値を要求データ・バンド幅と呼ぶことにする¹。

¹残り3つのキャッシュ構成が要求データ・バンド幅を越え
るような(つまり、要求以上の)データ・バンド幅を提供する
ことはありえない。

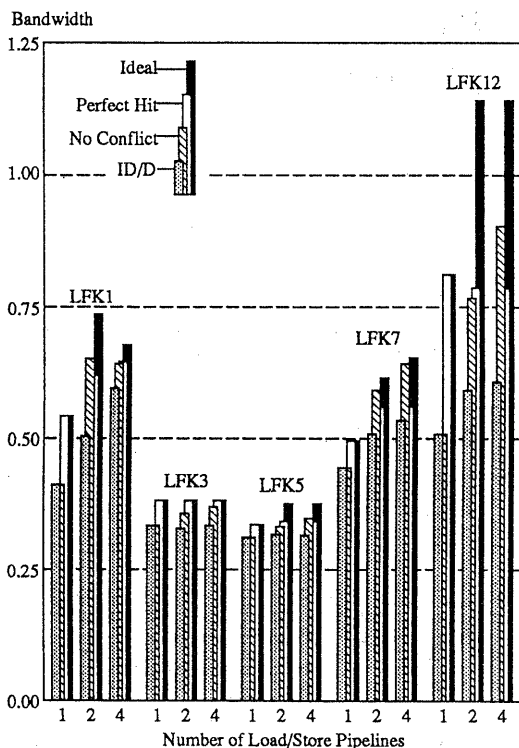


図 3: ロード/ストア・パイプラインの性能

図 3 からまず、要求データ・バンド幅自身がそれほど高くないことが判る。ロード/ストア・パイプラインの多重度が 1, 2, 4 の時の全ベンチマーク・プログラムの要求データ・バンド幅の調和平均は、それぞれ 0.50, 0.57, 0.58 である。各ベンチマークにおけるロード/ストア命令の実行頻度そのものは 28.9%~51.4% (算術平均 33.5%) であるから、命令発行率 4 (= DSNS プロセッサのスーパースカラ度) を維持しようとする下式より計算上 1.16~2.06 (調和平均 1.58) 以上のデータ・バンド幅が要求されるはずである。

$$\text{命令発行率} \leq \frac{\text{データ供給バンド幅}}{\text{ロード/ストア命令実行頻度}}$$

ところが、要求データ・バンド幅の実測値 (調和平均) は計算値 (調和平均) の 1/3 程度である²。このことから、本稿での評価対象であるデータ供給系以外のいずれかに、何らかの性能ボトルネックが存在していることが判る。その 1 つとして、レジスタ・ファイルの書き込みポートを獲得する際に競合が多発していることがシミュレーションにより確認されている。

さて、ロード/ストア・パイプライン (LSP) の実効的なデータ供給能力に注目しよう。図 3 から判るように、

$$\begin{aligned} \text{ID/I(1LSP)} &\leq \text{ID/I(2LSPs)} \leq \text{ID/I(4LSPs)} \\ \text{ID/I} &\leq \text{No Conflict} < \text{Ideal} \\ \text{ID/I} &\leq \text{Perfect Hit} \leq \text{Ideal} \end{aligned}$$

²つまり、命令発行率が平均でスーパースカラ度 4 の 1/3 程度しか出ていない。

という性能上の大小関係がある。つまり、DSNS プロセッサの現在のロード/ストア・アーキテクチャ仕様である ID/I(2LSPs) に対しては、まだ性能改善の余地が残っている。性能改善策には、上式に対応して次の 3 つが考えられる。

1. ロード/ストア・パイプラインの多重化: ロード/ストア・パイプラインを 2 多重の ID/I(2LSPs) から 4 多重の ID/I(4LSPs) に倍増した場合の性能向上率は 0%~18% (相乗平均 2%) と極めて小さい。一方、ハードウェア量は、データキャッシュのデータアレイやタグアレイのサイズは変わらないものの、ポート数、バンク数、ロード/ストア・ユニット数が倍増する。全性能改善策の中で最もコスト対性能比が悪い。
2. バンク・コンフリクトの低減化: ロード/ストア・パイプラインの多重度は 2 のままでバンク・コンフリクトを低減化した場合、ID/I(2LSPs) に対し最大 (すなわち、バンク・コンフリクトが生じない No Conflict: D(2LSPs) の場合) で 5%~30% (相乗平均 18%) 性能が向上する。ただし、D 型構成のハードウェア量は、データ・キャッシュのポート数は変わらないものの、データアレイおよびタグアレイのサイズが倍増する。
3. キャッシュ・ヒット率の向上、および、キャッシュミス・ペナルティの低減化: ロード/ストア・パイプラインの多重度は 2 のままでキャッシュ・ヒット率を 100% にした場合 (Perfect Hit), ID/I(2LSPs) に対し性能が 8%~33% (相乗平均 20%) 向上する。しかしながら、キャッシュ・ヒット率を 100% にすることは不可能であり、しかも、ID/I(2LSPs) においてもキャッシュ・ヒット率は cold start に関わらず 81.3%~90.5% (算術平均 84.4%) に達している。したがって、キャッシュミス・ペナルティの低減が残る有効手段だと思われる (次項参照)。

5.3.2 ノンブロッキング化の効果

キャッシュミス・ペナルティの低減方法の 1 つであるノンブロッキング化 (4.2.2 項参照) の効果を調べた。図 4 に、ポート当りの MSHR 数とデータ供給バンド幅との関係を示す。ここで、比較のため、キャッシュ構成 ID/I(2LSPs) を基本に、以下の 6 種類の能力の異なるデータキャッシュをシミュレートした。

- DPB: デュアルポート・ブロッキング・キャッシュ。キャッシュミスが生じた場合、すべてのポートをブロックする。
- DPNB(n): ポート当り n 個 ($n = 1, 2, 3, 4$) の MSHR を備えるデュアルポート・ノンブロッキング・キャッシュ。キャッシュミスが生じた場合でも、空の MSHR が存在する限りポートをブロックしない。DSNS プロセッサのデータ・キャッシュは、DPNB(2) に相当する。
- Perfect Hit: キャッシュミスが生じないような完全キャッシュ。

図 4 からまず、ブロッキング・キャッシュ DPB は完全キャッシュ Perfect Hit の 60%~84% (相乗平均 73%) 程度の性能しか発揮できていないことが判る。一方、ポート当りの MSHR が 4 個のノンブロッキング・キャッシュ DPNB(4) は、Perfect Hit の 75%~99% (相乗平均 86%) の性能に達しており、DPB に比べて 5%~27% (相乗平均 19%) 性能を向上させている。DSNS プロセッサのデータ・キャッシュに相当する DPNB(2) は、Perfect Hit に対する性能達成率 75%~92% (相乗平均 84%)、DPB に対する性能向上率 3%~24% (相乗平均 15%) で

ある。ポート当りの MSHR を 2 個から 4 個に倍増することでもまだ性能向上が可能であるが、その性能向上率は 0%~9% (相乗平均 3%) と小さい。

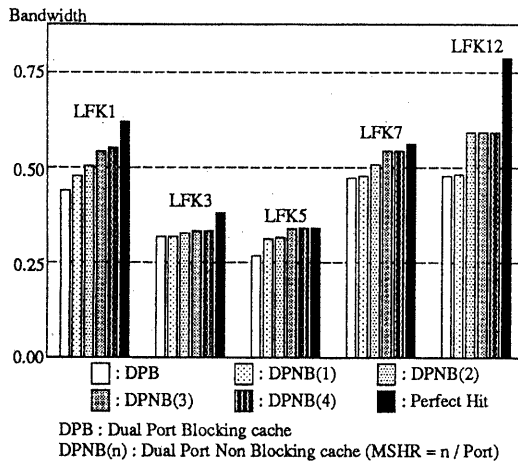


図 4: ノンブロッキング化の効果

6 おわりに

以上、DSNS プロセッサの特徴的なアーキテクチャである分岐アーキテクチャおよびロード/ストア・アーキテクチャに関して、性能評価を行った。その結果、次のことが判明した。

- 分岐アーキテクチャの評価: 分岐先バッファ (BTB) の効果を評価していく過程で、DSNS プロセッサでは早期分岐解消が極めて有効に機能していることが明らかになった (5.2.1 項)。その結果、BTB を用いた静的分岐予測、および、2 重化レジスタ・ファイルを用いた条件付実行モードが性能に対してそれほど貢献していないことが判明した (5.2.2 項)。
- ロード/ストア・アーキテクチャの評価: ロード/ストア・パイプラインの多重化の効果を評価していく過程で、データ供給系以外の箇所での性能ボトルネック (つまり、レジスタ・ファイルへの書き込み権の獲得競争) が判明した (5.3.1 項)。また、データ供給系に関しては、ロード/ストア・パイプラインの多重度を増すよりも、バンク・コンフリクトの低減およびキャッシュミス・ペナルティの低減の方が効果があることが明らかになった。特に、キャッシュミス・ペナルティの低減については、ノンブロッキング化が極めて効果的であることを確認した。

DSNS プロセッサは、SIMP スーパースカラ・プロセッサ「新風」の後継改良機として大幅にコスト対性能比の向上を図った結果 [1] であるが、今回の評価を通して、まだ性能およびコストに改善の余地があることが判明した。今後は、より広範なベンチマーク・プログラムを用いて性能評価を重ねていき、DSNS アーキテクチャをよりコスト対性能比に優れた洗練されたスーパースカラ・アーキテクチャへと完成させていくつもりである。

謝辞

日頃ご討論頂く九州大学大学院総合理工学研究科 安浦寛人教授ならびに安浦研究室の諸氏に感謝致します。

参考文献

- [1] 村上, 久我, 富田, "SIMP (単一命令流/多重命令流パイプライン) 方式に基づくスーパースカラ・プロセッサの改良方針," 信学技報, CPSY-90-54, 1990 年 7 月。
- [2] 原, 納富, 久我, 村上, 富田, "SIMP (単一命令流/多重命令流パイプライン) 方式に基づく改良版スーパースカラ・プロセッサの構成と処理," 信学技報, CPSY-90-55, 1990 年 7 月。
- [3] 原, 久我, 村上, 富田, "DSN 型スーパースカラ・プロセッサ・プロトタイプの方岐パイプライン," 情報研報, ARC-86-3, 1991 年 1 月。
- [4] 納富, 久我, 村上, 富田, "DSN 型スーパースカラ・プロセッサ・プロトタイプの方岐パイプライン," 情報研報, ARC-86-4, 1991 年 1 月。
- [5] 納富, 原, 久我, 村上, 富田, "DSN 型スーパースカラ・プロセッサ・プロトタイプ — アーキテクチャおよび静的コード・スケジューリングに関する総合評価 —," 並列処理シンポジウム JSPP'91 論文集, pp.125-132, 1991 年 5 月。
- [6] 村上和彰, "スーパースカラ・プロセッサの性能を最大限に引き出すコンパイラ技術," 日経エレクトロニクス, no.487, pp.165-185, 1991 年 3 月。
- [7] 村上和彰, "マイクロプロセッサ・アーキテクチャと並列処理技術 — マイクロプロセッサ用並列処理と並列処理用マイクロプロセッサ —," 信学技報, ICD-91-91, 1991 年 9 月。
- [8] Kroft, D., "Lockup-free Instruction Fetch/Prefetch Cache Organization," *Proc. 8th Ann. Symp. Comput. Architect.*, pp.81-87, May 1981.
- [9] Kuga, M., Murakami, K., and Tomita, S., "DSNS (Dynamically-hazard-resolved, Statically-code-scheduled, Nonuniform Superscalar): Yet Another Superscalar Processor Architecture," *ACM SIGARCH Comput. Architect. News*, vol.19, no.4, pp.14-29, June 1991.
- [10] Smith, J. E., "A Study of Branch Prediction Strategies," *Proc. 8th Ann. Symp. Comput. Architect.*, pp.135-148, May 1981.
- [11] Smith, M. D., Lam, M. S., and Horowitz, M. A., "Boosting Beyond Static Scheduling in a Superscalar Processors," *Proc. 17th Ann. Int'l. Symp. Comput. Architect.*, pp.344-354, May 1990.
- [12] Sohi, G. and Franklin, M., "High-Bandwidth Data Memory System for Superscalar Processors," *Computer Sciences Technical Report, #968*, Computer Sciences Department, University of Wisconsin-Madison, Sept. 1990.