Vol.2018-ARC-232 No.4
2018/7/30

アウトオブオーダプロセッサのクリティカルパス解析に基づくボトルネック命令チェーン抽出手法の提案

谷本 輝夫[1,a)]　小野 貴継[1,b)]　井上 弘士[1,c)]

概要：プロセッサ設計時にボトルネックとなりやすい命令チェーン情報があらかじめ分かれば，より高速なプロセッサの設計に有用であると考えられる．本論文では，アウトオブオーダプロセッサにおける命令実行のクリティカルパス解析を用い，ボトルネック命令チェーンを抽出する手法を提案する．

1. はじめに

ITRS (International Technology Roadmap for Semiconductors) により CMOS 半導体製造プロセス微細化限界が予測されており [10]，これまでのようなチップ当たりトランジスタ数の増加や，トランジスタ当たり消費電力削減の恩恵は期待できない．一方，IoT (Internet of Things) やビッグデータ処理などの普及に伴い，計算機器の数や処理したいデータ量は増加し続けると考えられる．したがって，計算機システムにはポストムーア時代においても更なる性能向上と電力当たり性能向上が求められる．そこで，アプリケーション特化型アーキテクチャが注目されている [11]．

より良いアーキテクチャ設計のためには，アプリケーションもしくはアルゴリズムが必要とする性能や，ボトルネックを理解する必要がある．従来から大規模並列計算機などで実行されてきた数値計算は，計算強度のルーフラインモデルによりボトルネックを見積もることが可能である．しかしながら，データベースマネジメントシステム [8], [24] やリクエスト駆動処理 [16] などの不規則な処理ではアプリケーションの詳細な解析が困難である．

アプリケーションの性能特性解析は性能解析可能なプラットフォームで実行することにより行われる．この実行により，どのような処理にどの程度の処理時間を要するか，及びその要因調査を行う．アウトオブオーダプロセッサは一般に広く普及しており，様々なアプリケーションを高速に実行できるよう設計されている．そのため，アーキテクチャ探索のベースラインに適していると考えられる．

[1] 九州大学
Kyushu University
[a)] tteruo@kyudai.jp
[b)] takatsugu.ono@cpc.ait.kyushu-u.ac.jp
[c)] inoue@ait.kyushu-u.ac.jp

しかしながら，アウトオブオーダプロセッサは実行時命令スケジューリングをハードウェアで行うため，命令粒度や，パイプラインステージ粒度での詳細な解析は困難である．そこで，本稿では，アウトオブオーダ命令実行のボトルネックとなる命令チェーンの抽出を目指す．このような情報は，アプリケーション特化型プロセッサを設計する際のアーキテクチャ検討に有効である．

本稿の主な貢献点は以下の 3 つである．

- **依存グラフに基づく重要な命令チェーン検出**

  本稿では，アウトオブオーダ命令実行の依存グラフ表現を用いたクリティカルパス解析を基に依存グラフを詳細に解析することで性能の決定要因となる命令チェーンを抽出する手法を提案する．グラフ探索アルゴリズムにより抽出を行うことにより，ヒューリスティックに依らずに，かつ機械的な抽出を実現できる．

- **命令チェーンのクリティカリティ指標の導入**

  ボトルネックとなりうる命令チェーンの抽出のみならず，それらのチェーンの実行時間への影響度の指標 (*chain tautness*) を導入する．この指標は，当該チェーンの実行に要するレイテンシを短縮することによる実行時間短縮の余地を定量評価したものであり，プロセッサ設計時に優先的にサポートするチェーンの判断に有用である．

- **SPEC CPU 2006 を用いたケーススタディ**

  SPEC CPU 2006 を用いたケーススタディにより 2 つの知見を明らかにする．一つは評価すべきチェーンの長さである．プロセッサやプログラムに依存するが，INT プログラムではチェーン長 100 までのチェーンにより大部分のボトルネック命令チェーンを考慮可能だ

が，FP プログラムではより長いチェーンの考慮が必要であることが分かった．もう一つは典型的なボトルネック命令チェーンとしてストアバンド幅競合，分岐予測ミスが依存するロード命令，分岐予測ミス復帰後の命令フェッチの発見である．これにより，十分なストアバンド幅の確保や，L1 キャッシュヒットレイテンシの実行時間への直接的な影響の回避などの課題が見つかった．

2. アウトオブオーダ命令実行のクリティカルパス解析

2.1 アウトオブオーダ命令実行の依存グラフ表現

アウトオブオーダプロセッサにおける命令実行を依存グラフとして表現する際には，コミットされた動的な命令ごとかつプロセッサ内における状態ごとにノードを作成し，ノード間の順序関係を有向エッジで結ぶ．順序関係は，ある処理が終わった後でないと次の処理が行えないことを表す．順序関係が循環することはないので，このグラフは閉路を持たない有向グラフとなる．エッジは依存を解決するために要するサイクル数を重みとしてもつ．

この依存グラフは，プログラムに内在する命令レベルのコントロールフローとデータフローの情報に加え，マイクロアーキテクチャの命令パイプラインに起因する依存関係を表現する．そのため，依存グラフの作成にはプログラムを対象プロセッサで実行した際の動的情報が必要である．

2.2 依存グラフモデル

プログラムの静的情報および実行時の動的情報をどのように依存グラフとして表現するかを規定するものを依存グラフモデルと呼ぶ．依存グラフモデルはマイクロアーキテクチャの抽象的なモデルであるといえる．

これまでにいくつかの依存グラフモデルが提案されている．Fields らは 1 命令をディスパッチ，実行，コミットの 3 状態（状態数はノードの種類数に対応）で表す依存グラフモデルを提案している [3]．彼らはこれを改良し，レディ（オペランドがそろった状態），コンプリート（演算完了状態）を加えた 5 状態とし，キャッシュラインを共有するロード命令間の依存関係のモデル化を実現している [4], [5]．これらはディスパッチ以降のバックエンド部をモデル化する．また，Lee らは，1 命令を 13 状態を用いて表すモデルを提案し，フロントエンド部も含めたモデル化を提案している [14]．

著者らは現代的なアウトオブオーダプロセッサを高精度にモデル化可能な依存グラフモデルを提案している [19], [20]．このモデルは Fields らの 5 状態モデルをベースとし，分岐予測ミスペナルティの動的な変動やストア命令のライトバックの考慮を加えたものである．本稿ではこのモデルにわずかに変更したもの（第 3.2 節参照）を用いる．

Algorithm 1 Calculate tautness of an edge

Require: G: 依存グラフ，e: G のエッジ，L_{orig}: G の最長経路長
Ensure: T: e の *tautness*
1: **procedure** CalcTautness(G, e, L_{orig})
2: $G' \leftarrow$ (G with e's weight $\leftarrow$ *zero*)
3: $L \leftarrow$ *longestPathlength*(G') ▷ Re-calculation of path length
4: $T \leftarrow L_{orig} - L$
5: return T
6: **end procedure**

2.3 依存グラフ表現とクリティカルパス解析

依存グラフにおいて，先頭命令の始状態のノードから末尾命令の終状態までの最長経路がクリティカルパスである．仮に依存モデルがプロセッサ内の全ての依存を表現できているとすると，この実行パスに要するサイクル数はプログラムの実行サイクル数に一致する．本稿では，始状態をディスパッチ，終状態をコミットとして最長経路探索したものをクリティカルパスとする．

2.4 コンテキストを考慮した命令実行解析の必要性

汎用プロセッサの持つ高い柔軟性を活かしつつ，専用化により性能または電力当たり性能を向上するためには，前後のコンテキストを活用することが極めて重要であると考えられる．この情報は，頻出する命令チェーンを専用命令化したり，分岐予測アルゴリズムや，プリフェッチの精度向上 [13] に有用である．

アウトオブオーダ命令実行の依存グラフ表現を用いた命令レベル粒度での解析として，*slack* [2] や *tautness* [23] が知られている．*slack* はある処理のレイテンシを増加させても全体の実行時間に影響のない最大サイクル数を調べたものであり，*tautness* はある処理のレイテンシを削減したときに全体の実行時間が削減される最大サイクル数を調べたものである．本稿ではボトルネックとなる命令チェーンの抽出を目的とするため，中でも *tautness* に着目する．

tautness は依存グラフ中の各エッジに対して考えることのできる指標である．単位はサイクル数であり，エッジの重み以下の自然数をとる．Algorithm 1 に算出アルゴリズムを示す．調査対象の edge ごとに計算する必要があり，対象 edge のみの重みを 0 にして最長経路長を調べ，元のグラフの最長経路との差分を *tautness* とする．

ちなみに，*tautness* = 0 となるエッジの枝狩りが可能である．具体的には，e を対象エッジ，s, d を e がつなぐノード ($s \rightarrow d$) とし，*weight()*, *dist()*, *r_dist()* がそれぞれエッジの重み，ノードの始点からの距離，ノードの終点からの距離，L が最長経路長を表すとすると，式 (1), (2) のどちらかを満たすエッジの *tautness* は 0 である．なお，これらの条件を満たさなくても *tautness* が 0 となるエッジは存在する．

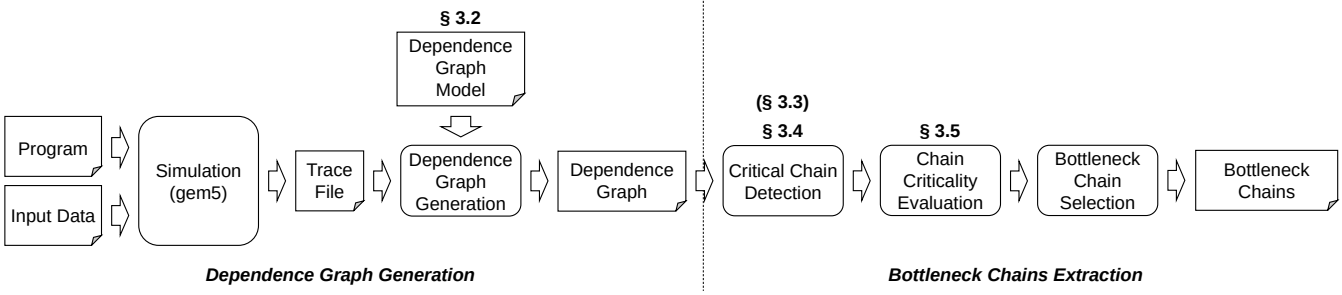

図 1 ボトルネック命令チェーン抽出手法の概要

表 1 依存グラフモデル [20] に加えた変更点

Name	Latency
CS	writeback latency if previous store is already retired when the instruction is committed else 0.
SS	0 if previous store is already retired when the instruction is committed else the number of cycles from the retirement of previous store to the retirement of the store.

$$weight(e) = 0 \quad (1)$$

$$dist(s) + weight(e) + r\_dist(s) < L \quad (2)$$

tautness の算出アルゴリズムからもわかる通り，一度には単一命令の解析しか行えない．そのため，前後の命令列が持つコンテキストを反映した解析が難しい．例えば，分岐予測ミスペナルティやロード命令の実行（ロードアドレスの計算とメモリアクセスの両方を含む）を表現するエッジの *tautness* は他のエッジに比べて大きな値をとる傾向がある．分岐予測ミスやキャッシュミスにはそれ以前の命令列のコンテキストが関係しているが，それらの情報は *tautness* の解析からは分からない．

3. クリティカルパス解析に基づくボトルネック命令チェーン抽出手法

3.1 概要

図 1 に抽出手法の概要を示す．本手法のワークフローは前半の依存グラフ生成部 (*Dependence Graph Generation*) と，後半のボトルネック命令チェーン抽出部 (*Bottleneck Chains extraction*) からなる．

前半部は一般的な依存グラフを用いたクリティカルパス解析におけるフローと同様である．本手法で用いる依存グラフモデルについて第 3.2 節で述べる．

後半部は前半部で生成した依存グラフを解析し，ボトルネック命令チェーン候補の抽出および定量的評価を行い，評価結果に基づき抽出する．ボトルネック命令チェーン候補の抽出法について第 3.3, 第 3.4 節で述べる．また，命令チェーンのクリティカリティの定量的指標について第 3.5 節で述べる．

3.2 ボトルネック命令抽出のための依存グラフモデル

クリティカルパス解析には近年のアウトオブオーダプロセッサのアーキテクチャを反映した依存グラフ [20] に表 1 の変更を加えたものを用いた．これらの変更は，ストア命令がコミットされてからリタイアする（ライトバックバッファからキャッシュへのライトバックが完了する）までの遅延を依存グラフでどのように扱うかに関係する．

従来モデルでは，ストア命令のコミットからリタイアまでの制約を表す *CS* エッジの重みでこの遅延を考慮し，ストアデータのインオーダライトバック (TSO: Total Store Order) に対応する *SS* エッジでの重みは 0 であった．ストア命令がリタイアした直後にコミット済み後続ストア命令はリタイア可能となるため，*SS* エッジの重みを 0 にすることは妥当と考えられる．実際，この定義でも最長経路探索により実行時間を正しく計算できる．

しかしながら，複数のストア命令が同時にライトバックを待つ状況では，待ち時間が複数のエッジの重みに反映される．このように，同じ制約による遅延の複数エッジへの計上はクリティカルパスの通り数を増やすことにつながる．本手法ではクリティカルパスの中で，唯一のパスしかない部分に着目する．そのため，同じ制約による遅延は一つのエッジの重みとして表現することが望ましい．

そのため，本手法では，ストア命令がライトバック待ち状態になった時点で直前のストアがすでにライトバック済みかどうかによりライトバック待ちの遅延を *CS* エッジで考慮するか *SS* エッジで考慮するか決定する．具体的には，すでにライトバック済みであれば *CS* エッジで，まだライトバックされていなければ *SS* エッジに重みづける．

3.3 ボトルネック命令チェーン抽出対象の定義

プログラムの実行時間を決定する実行パスは依存グラフにおいてクリティカルパスとして現れる．したがって，本研究では依存グラフ中のクリティカルパスに着目する．また，クリティカルパス上のオペレーションの中でも実行時間に対する影響の大きいものに焦点を当てる観点では，*tautness* のより大きなエッジに着目するのが自然である．

しかしながら，依存グラフ中のクリティカルパスは膨大な通り数存在することがわかっており [20]，このような

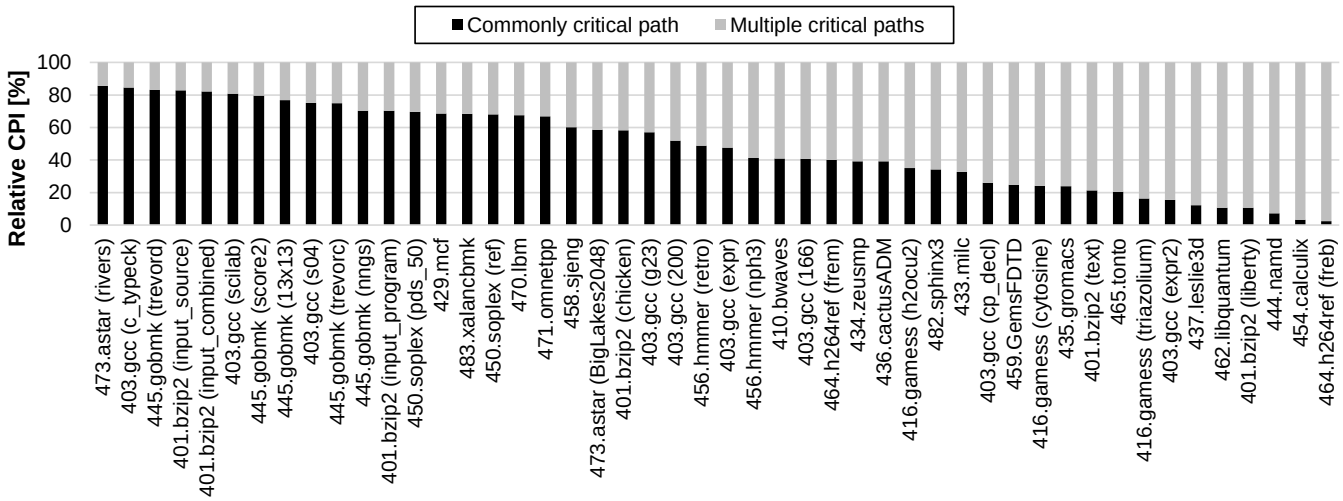

図 2　全クリティカルパスに共通するパスの実行時間に占める割合の評価結果

エッジに対しては *tautness* やそれに類する指標によりボトルネックを検出することができない．なぜなら，クリティカルパス上のあるエッジに注目した場合，そのエッジを通らない別のクリティカルパスが存在する場合，注目したエッジの重みを短縮しても別のクリティカルパスの経路長は変わらないため $tautness = 0$ となるからである．

言い換えれば，$tautness > 0$ であるようなエッジは，全てのクリティカルパスが通るエッジであることがわかる．本研究では，そのようなエッジをボトルネック抽出対象とする．全てのクリティカルパスが通る $tautness > 0$ を満たすエッジは，遅延削減によりプログラムの実行時間を直接的に削減できることがわかっているエッジである．そのため，このようなエッジは最適化の優先度が高いと考えることができる．

この選択の妥当性調査のため，$tautness > 0$ を満たすエッジが実行時間を占める割合の評価を行った．*tautness* は単一命令に対してのみ考慮可能なメトリックであり，命令ごとにこれらの値を加算することができないため，この評価の際には，依存グラフ中の $tautness > 0$ を満たすエッジ全てに対して，同時に重みから *tautness* を減じて得られる依存グラフの最長経路長を調べる．

これにより得られる経路長（実行時間）はクリティカルパス上のエッジのうち，そのエッジを通らないクリティカルパスが存在するようなエッジによるものである．つまり，この実行時間はプログラムの実行において複数のクリティカルパスをもつ部分といえる．元の依存グラフの最長経路長からこの経路長を減じた長さ（実行時間）は全クリティカルパスが共通して通るエッジによるものとみなすことができる．

図 2 に評価結果を示す．“multiple critical paths” はクリティカルパス上のエッジのうち，そのエッジを通らないクリティカルパスが存在するようなエッジによる部分を示し，“commonly critial path” は全クリティカルパスが共通して通るエッジによる部分を示す．用いた評価環境は第 4.2 節で述べる環境と同様である．図中の黒で示す部分が本手法のボトルネック抽出対象である．これは 48 のベンチマークプログラムのうち約半数である 23 のプログラムで 50 % を超えており，抽出対象として十分であると考えられる．

3.4　ボトルネック命令チェーンとその抽出手法

前節において本研究が対象とするボトルネック命令パターンを依存グラフ上の全てのクリティカルパスが共通して通るパスと定義した．このようなエッジはグラフ中で連続しているとは限らず，点在していることも考えられる．より具体的には，依存グラフ上の全てのクリティカルパスが共通して通るパスのうち，エッジの連続（依存チェーン）のできるだけ長いものをボトルネック命令チェーンの候補とする．

依存グラフ (G) が与えられたとき，このような依存チェーンは以下の手順で抽出可能である．

(1)　クリティカルグラフの作成 (G_{crit})

G からクリティカルパス上にあるエッジとそれらの両端のノードからなる部分グラフを作成する．これをクリティカルグラフ (G_{crit}) と呼ぶ．ノード s からノード d へのエッジ e がクリティカルパス上にあることは式 (3) の条件を満たすことと等しい．

$$dist(s) + weight(e) + r\_dist(d) = L \qquad (3)$$

(2)　クリティカルグラフ (G_{crit}) の無向グラフ化 (G')

G_{crit} を無向グラフへ変換したものを G' とする．依存グラフ G は元々弱連結であり，G_{crit} も始点ノードから終点ノードまでのパスが複数集まったものであるから弱連結である．したがって，それを無向グラフへ変換した G' は連結である．

Algorithm 2 Calculate chain criticality

Require: G: 依存グラフ，c: 依存チェーン, L_{orig}:G の最長経路長
Ensure: C: 依存チェーンのクリティカリティ
1: **procedure** CalcChainCriticality(G, c, L_{orig})
2: $G' \leftarrow G$
3: **for** $e \leftarrow Edges(c)$ **do**
4: $G' \leftarrow G'$ with e's weight $\leftarrow$ *zero*
5: **end for**
6: $L \leftarrow longestPathLength(G')$
7: $C \leftarrow Lorg - L$
8: return C
9: **end procedure**

(3) 無向グラフ (G') における橋であるエッジの探索

橋とは，グラフ中のエッジでそのエッジを取り除いた場合に部分連結グラフの数が増えるものを言う．すなわち，全てのクリティカルパスが通るエッジである．橋は深さ優先探索により効率よく発見できることが知られている [21].

(4) 連続する橋の接続

全てのクリティカルパスが共通して通るパスが分岐することはない．したがって，橋とその両端ノードからなる部分無向グラフを考えた際の連結成分を調べる．得られた連結成分をボトルネック命令チェーンの候補とする．

3.5 チェーンのクリティカリティ

依存チェーンのクリティカリティを評価する際には，*tautness* の考え方を依存チェーンに拡張したものを用いる．*tautness* では一つのエッジの重みのみを変更して最大経路長の変化を調べたが，依存チェーンのクリティカリティを調べる際にはアルゴリズム 2 に示すように依存チェーンに属するエッジ全ての重みを同時に変更して最大経路長の変化を調べる．

4. SPEC CPU 2006 を用いたケーススタディ

4.1 ケーススタディの目的

本研究が提案するボトルネック命令チェーン抽出手法の適用実験により，以下の調査を行う．

- **チェーン長とチェーン数及びクリティカリティの関係**
  SPEC CPU 2006 ベンチマークのような挙動のよく知られているプログラムにおいてボトルネック命令チェーンの長さ及びクリティカリティがどのような分布を持っているかを知ることは，実際のアプリケーション解析時の目安となる．
- **ボトルネック命令チェーン抽出による最適化可能性**
  実際に抽出された典型的命令チェーンについて具体的な最適化シナリオの検討を行う．

表 2 評価に用いたプロセッサモデルのパラメタ

Parameter	Value
CPU model	O3CPU
Frequency	2 GHz
ROB/Issue queue	192/60
LQ/SQ entries	72/48
Pipeline width	8 (issue width: 6)
Branch predictor	Tournament predictor
Choice/global/local predictor size	8 K/8 K/2 K entries
Local history table size	2 K entries
L1 ICache/Dcache size	32 KB / 32 KB
L1 Dcache access latency	4 cycles
L2 cache size	256 KB
L2 cache access latency	12 cycles
DRAM	DDR3-1600 11-11-11

4.2 評価環境

評価には，サイクル精度のプロセッサシミュレータである gem5 [1] の SE モードを用い，命令セットは x86 を用いた．表 2 に実験に用いたプロセッサのパラメタを示す．このパラメタは高性能プロセッサを志向したものである．

ベンチマークプログラムには SPEC CPU 2006 [9] から 25 のプログラムを用い，入力には `ref` を用いた．入力違いを含めて 48 のプログラムにより評価を行った．

gem5 によるベンチマーク実行時には，1 G 命令 `fast-forward` した後，100 M 命令 `warmup`，そして，1 M 命令の `detailed` シミュレーションを行った．クリティカルパス解析には，`detailed` シミュレーション区間の中間部を用いた．

本稿の評価におけるケーススタディでは解析対象の命令数を 1 K とした．アプリケーション全体を依存グラフ化し一度に解析するのはメモリ容量や解析に要する時間の観点から困難である．これには SimPoint [18] が利用できると考えているが，今後の課題とする．

4.3 抽出されたチェーン数

表 3 及び 表 4 に INT プログラムおよび FP プログラムのボトルネック命令チェーン抽出結果を示す．これらの表は抽出されたチェーンの最大長 (Max. length) とチェーン数，チェーンのクリティカリティをまとめたものである．ただし，チェーン長は命令数ではなく，エッジの連続数である．また，クリティカリティの集計時にはクリティカリティ（単位はサイクル数）が 5 サイクル以上のものを集計対象にしている．Coverage 欄はチェーン長が 1 または 100 以下のチェーンが占めるチェーン数およびクリティカリティの割合を示している．

チェーンの最大長やチェーン数はプログラムによって大きく異なる．チェーン長に関しては，`403.gcc (200)` の 12,436 や，`464.h264ref (frem)` の 9,984，`437.leslie3d` の 7,253，`434.zeusmp` の 6,775，`459.GemsFDTD` の 6,572，

表 3　INT プログラムのボトルネック命令チェーン抽出結果

Programs (input)	Max. Length	Number of chains			Critical cycles (*criticality* > 5)	
		all	Coverage [%]		Coverage [%]	
			$L^* = 1$	$L \leq 100$	$L = 1$	$L \leq 100$
403.gcc (200)	12,436	1	0.00	0.00	0.00	0.00
464.h264ref (frem)	9,984	1	0.00	0.00	0.00	0.00
456.hmmer (nph3)	2,971	21	23.81	61.90	0.01	0.94
456.hmmer (retro)	2,886	39	15.38	66.67	0.04	0.69
403.gcc (166)	2,561	1,627	10.88	99.63	5.34	22.42
445.gobmk (score2)	1,087	1,902	29.60	95.79	3.46	72.59
429.mcf	1,064	2,357	19.73	98.52	1.07	79.51
403.gcc (scilab)	855	3,159	13.58	99.49	1.72	95.67
483.xalancbmk	545	1,613	21.14	99.01	2.00	90.76
403.gcc (expr2)	523	685	32.41	70.36	3.06	51.40
464.h264ref (freb)	504	547	5.12	52.10	0.20	7.67
445.gobmk (trevorc)	493	2,781	19.20	98.17	2.11	88.90
445.gobmk (nngs)	425	890	21.57	51.57	0.43	12.11
401.bzip2 (input_program)	402	2,580	25.43	99.03	2.92	93.92
401.bzip2 (input_source)	354	2,207	19.71	94.29	1.60	76.27
445.gobmk (13x13)	344	2,701	17.70	98.48	2.05	91.54
445.gobmk (trevord)	310	3,073	16.04	96.97	1.64	78.58
401.bzip2 (liberty)	264	933	44.37	97.11	1.94	87.00
458.sjeng	256	2,392	19.98	97.78	1.56	88.45
473.astar (rivers)	234	2,549	20.01	93.33	1.58	78.33
403.gcc (expr)	231	1,606	35.74	96.58	2.39	75.32
403.gcc (g23)	226	1,365	28.86	93.92	1.84	77.88
403.gcc (s04)	203	2,257	19.23	99.38	2.04	96.04
403.gcc (c_typeck)	167	3,158	12.54	99.56	1.76	97.79
401.bzip2 (input_combined)	165	2,839	14.58	98.38	1.34	92.84
403.gcc (cp_decl)	149	1,470	74.97	99.93	52.66	85.38
401.bzip2 (chicken)	130	2,483	23.24	99.96	1.69	99.59
473.astar (BigLakes2048)	124	1,403	50.68	99.86	14.21	96.72
471.omnetpp	123	2,092	25.53	99.81	4.39	97.69
401.bzip2 (text)	120	1,679	39.49	99.82	2.52	99.19
462.libquantum	21	721	48.13	100.00	10.09	100.00

* L is the lengths of chains.

表 4　Summary of detected critical chains of floating-point programs.

Programs (input)	Max. Length	Number of chains			Critical cycles (*criticality* > 5)	
		all	Coverage [%]		Coverage [%]	
			$L^* = 1$	$L \leq 100$	$L = 1$	$L \leq 100$
437.leslie3d	7,253	36	2.78	66.67	0.02	0.96
434.zeusmp	6,775	308	40.58	98.05	1.47	39.42
459.GemsFDTD	6,572	1	0.00	0.00	0.00	0.00
470.lbm	6,332	2	0.00	50.00	0.00	0.06
416.gamess (h2ocu2)	5,535	382	39.27	98.95	1.82	25.72
416.gamess (triazolium)	2,138	159	50.31	97.48	1.63	29.66
416.gamess (cytosine)	1,490	643	38.26	98.76	3.26	48.28
450.soplex (ref)	1,470	2,278	26.69	97.98	4.07	82.89
465.tonto	1,216	598	51.84	98.66	4.35	26.04
444.namd	587	1,011	9.30	92.38	1.01	60.79
410.bwaves	537	244	22.95	77.87	0.58	18.33
450.soplex (pds_50)	434	2,271	26.95	97.93	5.09	86.03
436.cactusADM	384	272	42.65	80.51	1.47	24.84
433.milc	219	352	65.34	65.91	3.81	3.84
482.sphinx3	149	897	30.77	96.99	3.20	80.91
435.gromacs	116	995	0.10	99.90	0.01	99.35
454.calculix	58	1,042	58.93	100.00	9.01	100.00

* L is lengths of chains.

`470.lbm` の 6,332，`416.gamess (h2ocu2)` の 5,535 などのいくつかのプログラムで非常に長いチェーンが見られる．長いクリティカルなチェーンはプログラムの性能がそのよ

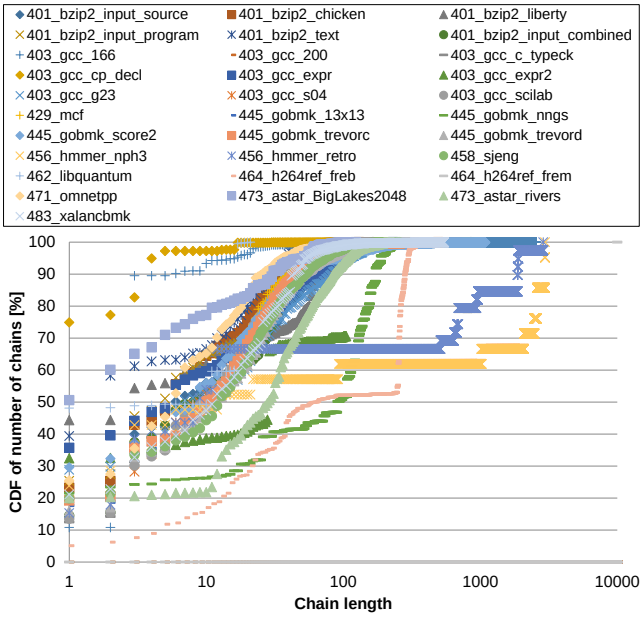

図 3　INT プログラムのチェーン長に対するチェーン数の累積分布関数．

うなチェーンに含まれる処理の系列に律速されていることを意味している．つまり，プロセッサ内における処理の並列度が低く，アウトオブオーダ・スーパースカラプロセッサの処理能力が活用できていない可能性がある．

一方で，`462.libquantum` や `454.calculix` はそれぞれ 21，58 とチェーンの最大長が比較的小さい．チェーンの最大長とチェーン数に有意な相関はなく，チェーンの最大長が短くても必ずしもチェーン数が大きいわけではない．

チェーン数に関しては，多くのプログラムは多数のチェーンが抽出された．31 の INT プログラム中 22，17 の FP プログラムのうち 4 では 1,000 以上のチェーンが抽出された．その一方で，`403.gcc (200)`, や `464.h264ref (frem)`, `459.GemsFDTD`, `470.lbm` は 1 または 2 のチェーンのみが抽出された．これらは長いチェーンを持つ．

4.4　チェーン長とチェーン数の関係

図 3 と 図 4 はそれぞれ INT プログラムと FP プログラムのチェーン長とチェーン数の累積分布関数を示したものである．分布関数の形状はプログラムのより様々ではあるが，全体としては INT プログラムも FP プログラムも同様の傾向であるといえる．いくつかのプログラムは 1 つのエッジからなるチェーンが大きな割合を占めている．これは表 3 及び 表 4 の “Number of chains” の $L = 1$ の列に示されており，74.97% (`403.gcc (cp_decl)`), 50.68% (`473.astar (BigLakes2048)`), 65.34% (`433.milc`), 58.93% (`454.calculix`), 51.84% (`465.tonto`), 50.31% (`416.gamess (triazolium)`) などである．

それ以外のプログラムでは長さ 1 のチェーンのカバー率は 50%を下回っている．また，いくつかのプログラムで

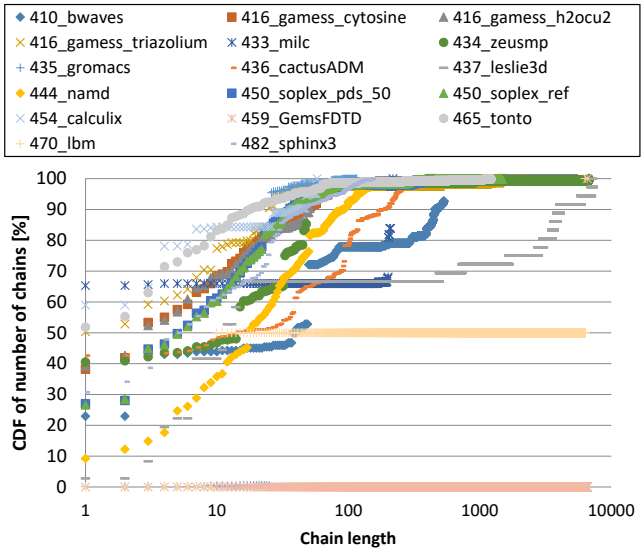

図 4 FP プログラムのチェーン長に対するチェーン数の累積分布関数.

は長さ 1 のチェーンのカバー率はとても小さい．たとえば，`464.h264ref (frem)`, `435.gromacs`, `437.leslie3d`, `444.namd` のカバー率はそれぞれ 5.12%, 0.10%, 2.78%, 9.30% である（`403.gcc (200)`，`464.h264ref (frem)`, `459.GemsFDTD`, `470.lbm` はチェーン数が小さいため除外）．これは単一のエッジだけを対象とした解析では最適化機会を見誤る可能性があることを示しており，単一エッジではなくチェーンを抽出することの重要性を示している．

動的にクリティカルなチェーンの検出をしたい場合など，調査すべきチェーンの長さを知ることは重要である．累積分布関数の形状はアプリケーションによるが，ケーススタディの結果からは長さ 100 までを調べればチェーン数の大部分をカバーできることがわかる．長さ 100 以下のチェーンは 31 の INT プログラムのうち 24，17 の FP プログラムのうち 11 で 90%以上のカバー率となる．カバー率が 90%以下となるのは 51.57% (`445.gobmk (nngs)`), 52.10% (`464.h264ref (freb)`)，61.90% (`456.hmmer (nph3)`), 66.67% (`456.hmmer (retro)`)，65.91% (`433.milc`), 66.67% (`437.leslie3d`) である．プログラムごとのカバー率は表 3 及び 表 4 中 “Number of chains” の $L \leq 100$ 列に示している．これらのプログラムでは，ストアバンド幅の制約を表す *SS* エッジがボトルネック命令チェーンに頻繁に表れ長いチェーンを形成する傾向がある．

4.5 チェーン長とクリティカリティの関係

図 5 と 図 6 はそれぞれ INT プログラムと FP プログラムのチェーン長に対するチェーンのクリティカリティの累積分布関数を示す．前述の通り，クリティカリティが 5 以上のチェーンのみを集計の対象とした．*tautness* は単一のエッジしか考慮しないので複数のエッジの値を加算することができないが，今回算出しているチェーンのクリティカリティは相互作用がない単位でチェーンを作成しているため加算して評価できる．チェーン数の分布と同様，アプリケーションにより分布の形状は異なることがわかる．

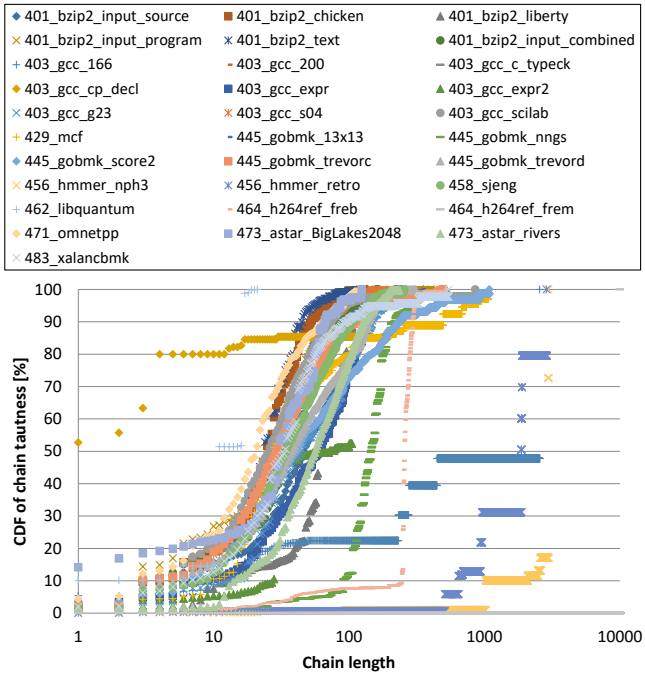

図 5 INT プログラムのチェーン長に対するクリティカリティの累積分布関数.

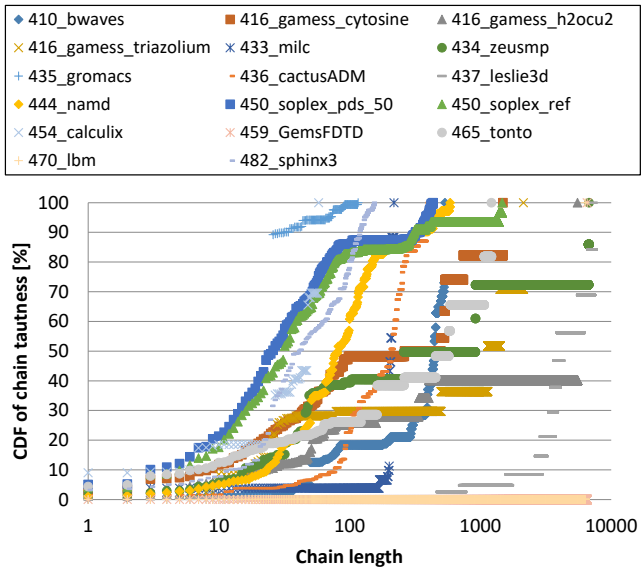

図 6 FP プログラムのチェーン長に対するクリティカリティの累積分布関数.

`403.gcc (cp_decl)` のみはチェーン長 1 のチェーンのカバー率が 50.68%と高い．これは表 3 及び 表 4 の “Critical Cycles” の $L = 1$ の列に示されている．他のプログラムのカバー率は全て 50%未満である．また，`464.h264ref (frem)`, `435.gromacs`, `437.leslie3d`, `444.namd` などのプログラムでは 1 エッジからなるチェーンのカバー率はそれぞれ 5.12%, 0.10%, 2.78%, 9.30%である．このことからも単一エッジではなくチェーンとしてボトルネック命令を抽出することの重要性がわかる．

チェーン数と同様にクリティカリティにおいても調査すべきチェーンの長さを考える．31 の INT プログラムのうち 22 でチェーン長 100 以内のチェーンで 75%のクリティカリティを占めている．カバー率が低いのは，0.69% (456.hmmer (retro)), 0.94% (456.hmmer (nph3)), 7.67% (464.h264ref (freb)), 12.11% (445.gobmk (nngs)) など少数のプログラムに限られる．一方で，FP プログラムでは，75% 以上のカバー率を持つプログラムは 17 のうち 5 つに留まる．これは表 3 及び 表 4 の “Critical Cycles” の $L \leq 100$ の列に示されている．特に，9 つの FP プログラムではカバー率は 30%以下である．

FP プログラムの低いカバー率の理由の一つはレイテンシの長い浮動小数点演算がカーネル関数で使われていることにある．これらの演算を含むチェーンはクリティカリティが大きくなる傾向にあるので結果として長いクリティカルチェーンが形成されやすいと考えられる．また，多くの浮動小数点演算は多くのストア命令を伴うのでストアバンド幅律速になりやすく，*SS* エッジによる長いクリティカルチェーンができやすい．これらを考慮すると，長いチェーンは大きなクリティカリティを持つため，より長いチェーンも抽出対象に含める必要があるといえる．

4.6 典型的なボトルネック命令チェーン

続いて，ボトルネック命令チェーンとして見られる典型的なチェーンについて詳細に調査する．大きなクリティカリティを持つチェーンや，頻繁に抽出されるチェーンはプロセッサ最適化の候補とみなすことができる．

ボトルネック命令チェーンの解析を通して，3 つの典型的なチェーンを発見した．それらはストアバンド幅競合，予測ミス分岐命令が依存するロード命令，そして，分岐ミス復帰後の命令フェッチである．これらのチェーンがクリティカルであること自体は自然なことであるが，解析によりこれらが抽出されたこと及びそれらのクリティカリティが定量的に評価できることは抽出法の有効性を示している．

- **ストアバンド幅競合**

  ストアバンド幅競合は *SS* エッジにより表現されており，今回のプロセッサモデルでは重みが大きくなりやすいため非常に長いチェーンを形成しやすい．図 7 に例を示す．*SS* がクリティカルになりやすいことは，今回ベースとして用いた依存グラフモデルにおいてストアバンド幅に関する修正が精度向上に大きく貢献している [20] ことから見ても妥当であるといえる．アウトオブオーダにとって十分なストアバンド幅を確保することは極めて重要であることがわかる．

- **予測ミス分岐命令が依存するロード命令**

  分岐ミスからの復帰は完了するまで後続の命令をディスパッチできないのでクリティカルになりやすい．条件分岐や関節分岐命令は分岐結果や分岐先を得るためにデータが必要である．したがって，それらのデータを生成するためのデータに対するロード命令が存在する．そして，これらのロード命令のレイテンシもクリティカルに見えることが多い．図 8 の例は 403.gcc (166) で 35 サイクルのクリティカリティを持つ．このチェーンでは，太い実線で示す *EP* と *PD* エッジが大きな重みをもつ．この例では，分岐ミスペナルティを表す *PD* の重みは 30 サイクルであり，ロード命令の実行（アドレス計算とメモリアクセスの両方を含む）のレイテンシは L1 キャッシュ時で 6 サイクルである．

  従来 L1 キャッシュヒット時のアクセスレイテンシの性能への影響はあまり重視されてこなかった．しかしながら，容量を増やしヒット率を向上させるため，アクセスに複数サイクルを要するキャッシュを最上位に配置することが増えている．このレイテンシは，命令ウィンドウに命令が十分溜まっている状態では隠ぺいできるが，分岐予測ミスと組み合わさると他の命令を実行できないので性能に直接的に影響する．このチェーンは，ロード命令がさらに先行するロード命令に依存している場合にはさらに長くなる．このようなチェーンのクリティカリティを低減するためには，ミスしやすい分岐命令が依存するデータを極小容量で高速なバッファに保持するなどが考えられる．

- **分岐予測ミス復帰後の命令フェッチ**

  分岐予測ミス復帰後の命令フェッチもクリティカルになる場合がある．*PD* エッジは分岐ミス検出から次の命令のディスパッチまでを表現するので，次の命令をフェッチするための遅延は *PD* エッジに含まれている．今回使用したプロセッサモデルはフェッチバッファを持っているが，復帰直後はこのバッファ蓄えられる命令数が不十分で，フロントエンドがバックエンドに十分なスループットで命令を供給できていない可能性がある．この場合にも，L1 キャッシュのレイテンシがクリティカルになりうる．このようなチェーンのクリティカリティを低減するためには，より正確な命令プリフェッチが必要である．

5. 関連研究

5.1 命令クリティカリティ検出

いくつかの手法が知られている．特定のマイクロアーキテクチャ要素に着目することで命令クリティカリティを予測する手法 [22] や，ロード命令のクリティカリティを予測する手法 [6], [12], [17] が知られている．これらの手法はいずれもクリティカリティの定義にヒューリスティックを用

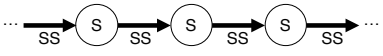

図 7 ストアバンド幅競合の例

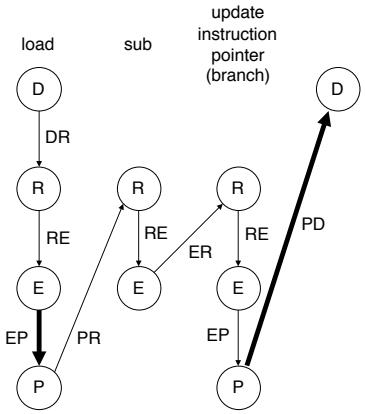

図 8 予測ミス分岐命令が依存するロード命令の例

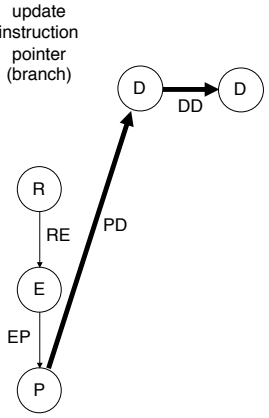

図 9 分岐予測ミス復帰後の命令フェッチの例

いている．

本研究は依存グラフのクリティカルパス解析に立脚しており，クリティカリティはグラフ上の経路長，すなわち実行時間により定義されている．このことは，予断なくアプリケーションおよびアーキテクチャの性能特性を解析する上で重要である．

5.2 アウトオブオーダ命令実行の依存グラフを用いたプロセッサ最適化

RpStacks [14] は依存グラフを用いたクリティカルパス解析により設計空間探索を行っている．グラフ上の探索を用いることでプロセッサシミュレーションによる探索よりも大幅な加速が可能であるとしている．しかしながら，RpStacks により探索可能な設計空間はパラメタ化されたプロセッサの範囲にとどまる．

依存グラフを用いた異なるアーキテクチャをまたいだ性能推定の試みも行われている [15]．この研究では，グラフを変換することで SIMD (Sigle Instruction Multiple Data) や DySER [7] などのアーキテクチャを採用した場合の性能推定を行っている．この手法はあらかじめ用意された特定のアーキテクチャを対象に設計されている．我々の手法はそのような前提を置かず，ボトルネックとなっている命令チェーンを抽出できる点で異なる．

6. おわりに

半導体製造プロセス微細化の鈍化に伴い，アプリケーション特化プロセッサの重要性が相対的に増している．アウトオブオーダプロセッサ上でのソフトウェアの挙動を理解し，プロセッサをソフトウェア向けに最適化するためのボトルネック命令チェーン抽出手法について述べた．アウトオブオーダ命令実行の依存グラフ表現を用いたクリティカルパス解析を応用し，全てのクリティカルパスが共通して通るパスを抽出し，さらにそれらのパスのクリティカリティの定量的な指標を提案した．

SPEC CPU 2006 を用いたケーススタディにより 2 つの知見を得た．一つは評価すべきチェーンの長さである．プロセッサやプログラムに依存するが，INT プログラムではチェーン長 100 までのチェーンを調査することでほとんどのボトルネック命令チェーンを考慮できるが，FP プログラムではより長いチェーンも考慮する必要がある．もう一つは典型的なボトルネック命令チェーンとしてストアバンド幅競合，分岐予測ミスが依存するロード命令，分岐予測ミス復帰後の命令フェッチを発見したことである．この発見により，十分なストアバンド幅の確保や，L1 キャッシュヒットレイテンシの実行時間への直接的な影響の回避などの課題が見つかった．

今後はアプリケーションを専用プロセッサ化していく要求がより一層高まると考えられる．そのため，プログラムの本質的な振る舞いを機械的に抽出し，設計に活用できるツールの重要性が高まると考えている．アプリケーション全体の評価や，これらの知見を活かしたプロセッサの最適化は今後の課題である．

謝辞 本研究は一部，JSPS 科研費 JP16H02796 の助成による．計算機の利用にあたり九州大学情報基盤研究開発センターの協力を得た．

参考文献

[1] Binkert, N., Beckmann, B., Black, G., Reinhardt, S. K., Saidi, A., Basu, A., Hestness, J., Hower, D. R., Krishna, T., Sardashti, S., Sen, R., Sewell, K., Shoaib, M., Vaish, N., Hill, M. D. and Wood, D. A.: The Gem5 simulator, *ACM SIGARCH Computer Architecture News*, Vol. 39, No. 2, pp. 1–7 (online), DOI: 10.1145/2024716.2024718 (2011).

[2] Fields, B., Bodík, R. and Hill, M. D.: Slack: maximizing performance under technological constraints, *Proceedings of the 29th Annual International Symposium on Computer Architecture*, ISCA '02, Washington, DC, USA, pp. 47–58 (online), DOI: 10.1109/ISCA.2002.1003561 (2002).

[3] Fields, B., Rubin, S. and Bodík, R.: Focusing processor policies via critical-path prediction, *Proceedings of the 28th Annual International Symposium on Computer Architecture*, ISCA '01, New York, NY, USA, pp. 74–85 (online), DOI:

10.1145/379240.379253 (2001).

[4] Fields, B. A., Bodík, R., Hill, M. D. and Newburn, C. J.: Using interaction costs for microarchitectural bottleneck analysis, *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 36, Washington, DC, USA, pp. 228–239 (2003).

[5] Fields, B. A., Bodik, R., Hill, M. D. and Newburn, C. J.: Interaction Cost and Shotgun Profiling, *ACM Transactions on Architecture and Code Optimization*, Vol. 1, No. 3, pp. 272–304 (online), DOI: 10.1145/1022969.1022971 (2004).

[6] Ghose, S., Lee, H. and Martínez, J. F.: Improving Memory Scheduling via Processor-side Load Criticality Information, *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ISCA '13, New York, NY, USA, pp. 84–95 (online), DOI: 10.1145/2485922.2485930 (2013).

[7] Govindaraju, V., Ho, C. H., Nowatzki, T., Chhugani, J., Satish, N., Sankaralingam, K. and Kim, C.: DySER: Unifying Functionality and Parallelism Specialization for Energy-Efficient Computing, *IEEE Micro*, Vol. 32, No. 5, pp. 38–51 (online), DOI: 10.1109/MM.2012.51 (2012).

[8] Haas, S., Arnold, O., Scholze, S., Hppner, S., Ellguth, G., Dixius, A., Ungethm, A., Mier, E., Nthen, B., MatÅ, E., Schiefer, S., Cederstroem, L., Pilz, F., Mayr, C., Schffny, R., Lehner, W. and Fettweis, G. P.: A database accelerator for energy-efficient query processing and optimization, *Proceedings of the 2016 IEEE Nordic Circuits and Systems Conference*, NORCAS '16, Copenhagen, Denmark, pp. 1–5 (online), DOI: 10.1109/NORCHIP.2016.7792904 (2016).

[9] Henning, J. L.: SPEC CPU2006 benchmark descriptions, *ACM SIGARCH Computer Architecture News*, Vol. 34, No. 4, pp. 1–17 (online), DOI: 10.1145/1186736.1186737 (2006).

[10] International Technology Roadmap for Semiconductors: International Technology Roadmap for Semiconductors 2.0, `https://www.semiconductors.org/main/2015_international_technology_roadmap_for_semiconductors_itrs/` (2015).

[11] Jouppi, N. P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., Bates, S., Bhatia, S., Boden, N., Borchers, A., Boyle, R., Cantin, P.-l., Chao, C., Clark, C., Coriell, J., Daley, M., Dau, M., Dean, J., Gelb, B., Ghaemmaghami, T. V., Gottipati, R., Gulland, W., Hagmann, R., Ho, C. R., Hogberg, D., Hu, J., Hundt, R., Hurt, D., Ibarz, J., Jaffey, A., Jaworski, A., Kaplan, A., Khaitan, H., Killebrew, D., Koch, A., Kumar, N., Lacy, S., Laudon, J., Law, J., Le, D., Leary, C., Liu, Z., Lucke, K., Lundin, A., MacKean, G., Maggiore, A., Mahony, M., Miller, K., Nagarajan, R., Narayanaswami, R., Ni, R., Nix, K., Norrie, T., Omernick, M., Penukonda, N., Phelps, A., Ross, J., Ross, M., Salek, A., Samadiani, E., Severn, C., Sizikov, G., Snelham, M., Souter, J., Steinberg, D., Swing, A., Tan, M., Thorson, G., Tian, B., Toma, H., Tuttle, E., Vasudevan, V., Walter, R., Wang, W., Wilcox, E. and Yoon, D. H.: In-Datacenter Performance Analysis of a Tensor Processing Unit, *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ISCA '17, New York, NY, USA, pp. 1–12 (online), DOI: 10.1145/3079856.3080246 (2017).

[12] Ju, R. D.-c., Lebeck, A. R. and Wilkerson, C.: Locality vs. Criticality, *Proceedings of the 28th Annual International Symposium on Computer Architecture* (Srinivasan, S. T., ed.), ISCA '01, New York, NY, USA, pp. 132–143 (online), DOI: 10.1145/379240.379258 (2001).

[13] Kim, J., Pugsley, S. H., Gratz, P. V., Reddy, A. L. N., Wilkerson, C. and Chishti, Z.: Path confidence based lookahead prefetching, *The 49th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '16, pp. 1–12 (online), DOI: 10.1109/MICRO.2016.7783763 (2016).

[14] Lee, J., Jang, H. and Kim, J.: RpStacks: Fast and Accurate Processor Design Space Exploration Using Representative Stall-Event Stacks, *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-47, Washington, DC, USA, pp. 255–267 (online), DOI: 10.1109/MICRO.2014.26 (2014).

[15] Nowatzki, T., Govindaraju, V. and Sankaralingam, K.: A Graph-Based Program Representation for Analyzing Hardware Specialization Approaches, *IEEE Computer Architecture Letters*, Vol. 14, No. 2, pp. 94–98 (online), DOI: 10.1109/LCA.2015.2476801 (2015).

[16] Putnam, A., Caulfield, A. M., Chung, E. S., Chiou, D., Constantinides, K., Demme, J., Esmaeilzadeh, H., Fowers, J., Gopal, G. P., Gray, J. et al.: A reconfigurable fabric for accelerating large-scale datacenter services, *Proceedings of the 41st Annual International Symposium on Computer Architecture*, ISCA '14, pp. 13–24 (2014).

[17] Rakvic, R., Black, B., Limaye, D. and Shen, J. P.: Non-vital Loads, *Proceedings of the Eighth IEEE International Symposium on High-Performance Computer Architecture*, HPCA '02, Cambridge, MA, USA, pp. 165–174 (online), available from ⟨http://dl.acm.org/citation.cfm?id=874076.876455⟩ (2002).

[18] Sherwood, T., Perelman, E. and Calder, B.: Basic Block Distribution Analysis to Find Periodic Behavior and Simulation Points in Applications, *Proceedings of the 2001 International Conference on Parallel Architectures and Compilation Techniques*, PACT '01, Washington, DC, USA, pp. 3–14 (online), available from ⟨http://dl.acm.org/citation.cfm?id=645988.674158⟩ (2001).

[19] Tanimoto, T., Ono, T., Inoue, K. and Sasaki, H.: Enhanced Dependence Graph Model for Critical Path Analysis on Modern Out-of-Order Processors, *IEEE Computer Architecture Letters*, Vol. 16, No. 2, pp. 34–37 (online), DOI: 10.1109/LCA.2017.2684813 (2017).

[20] Tanimoto, T., Ono, T. and Inoue, K.: Dependence Graph Model for Accurate Critical Path Analysis on Out-of-Order Processors, *Journal of Information Processing*, Vol. 25, pp. 983–992 (online), DOI: 10.2197/ipsjjip.25.983 (2017).

[21] Tarjan, R. E.: A Note on Finding the Bridges of a Graph., *Elsevier Information Processing Letters*, Vol. 2, No. 6, pp. 160–161 (1974).

[22] Tune, E., Liang, D., Tullsen, D. M. and Calder, B.: Dynamic prediction of critical path instructions, *Proceedings of the Seventh IEEE International Symposium on High-Performance Computer Architecture*, HPCA '01, Washington, DC, USA, pp. 185–195 (2001).

[23] Tune, E., Tullsen, D. M. and Calder, B.: Quantifying instruction criticality, *Proceedings of the 2002 International Conference on Parallel Architectures and Compilation Techniques*, PACT '02, Washington, DC, USA, pp. 104–113 (2002).

[24] Wu, L., Lottarini, A., Paine, T. K., Kim, M. A. and Ross, K. A.: Q100: The Architecture and Design of a Database Processing Unit, *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XIX, New York, NY, USA, pp. 255–268 (online), DOI: 10.1145/2541940.2541961 (2014).