資源リサイジングとクロック周波数ブーストを適応的に切り替えるデュアルターボブースト

山口恭平[†,☆] 塩谷亮太[†] 安藤秀樹[†]

プロセッサの TDP(thermal design power) は，すべてのコアが動作している場合を想定して決められている．しかし，そのような場合は多くなく，電力予算は余っていることが多い．現在では，これを動作コアに割り当て，クロック周波数を増加させ，単一スレッドの実行性能を改善する手法がとられている．しかし，クロック周波数を増加するだけでは，メモリアクセス時間が実行時間の多くを占めるメモリインテンシブなプログラムの実行においては効果が少ない．これに対して，我々はこれまでに，動的に命令ウィンドウの資源を拡大し，メモリレベル並列を利用し性能を改善する資源のリサイジング手法を提案した．しかし，電力をより多く消費する，また，計算インテンシブなプログラムの性能を向上させることはできないという欠点があった．本論文では，アイドルコアが存在するときの余剰電力予算の活用方法として，プログラムの実行フェーズが，メモリインテンシブか計算インテンシブのどちらかであるかを動的に予測し，資源リサイジングかクロック周波数ブーストかを選択することによりそれぞれの利点を活かすデュアルターボブーストを提案する．本手法を用いれば，プログラムがメモリインテンシブか計算インテンシブのどちらであっても適応的に余剰電力予算を性能向上に結びつけることができる．SPEC2000 プログラムを使用して評価を行ったところ，資源リサイジング，クロック周波数ブースト単体を適用した場合に対して，デュアルターボブーストにより，それぞれ，12%，8%性能が向上することを確認した．

Dual Turbo Boost: Adaptively Selecting Resource Resizing and Clock Frequency Boost

Kyohei Yamaguchi,[†] Ryota Shioya[†] and Hideki Ando[†]

The TDP (thermal design power) of a processor is determined under the assumption where all cores are working. However, such a case is not often, and thus power budget is left in many cases. In current microprocessors, a method that boosts the clock frequency by allocating the left budget to working cores is introduced to increase performance. Unfortunately, this method is ineffective in memory-intensive programs, because memory access time dominates the execution time. On the other hand, we previously proposed the resource resizing scheme that improves performance of memory-intensive programs. However, this scheme has a drawback that more power is consumed and it cannot improve performance of computer-intensive programs. This paper proposes a scheme called dual turbo boost that effectively utilizes left power budget when several cores are idle. The scheme selects the resource resizing or clock frequency boost by predicting whether the executing phase is memory-intensive or not, and exploit each advantage. Our evaluation results using SPEC2000 programs show that the dual turbo boost achieves performance improvements over the resource resizing or clock frequency boost solely by 12% or 8%, respectively.

1. はじめに

プロセッサが消費可能な電力は，自身の発生する熱量によって制限されている．現在のマルチコアプロセッサにおいては，全てのコアが動作する場合を想定して消費電力の上限である TDP(thermal design power) が決定されている．しかし，そのようにすべてのコアが動作している場合は多くなく，アイドルコアにより電力予算が余っていることが多い．そこで，現在のプロセッサでは，余剰電力予算を動作中のコアに回し，クロック周波数を上げ，単一スレッドの性能を向上させるターボブースト技術が搭載されている．しかしこの手法は，メモリインテンシブなプログラムに対しては有効ではない．なぜなら，主記憶とプロセッサの間にはメモリウォールと呼ばれる大きな速度ギャップが存在し，主記憶アクセス時間が実行時間の多くを支配

† 名古屋大学大学院工学研究科
Graduate School of Engineering, Nagoya University
☆ 現在，ルネサスエレクトロニクス (株)
Presently with Renesas Electronics Corp.

しているからである．

メモリウォール問題を解決するためのアプローチとして，積極的なアウトオブオーダ実行がある．この手法は，命令ウィンドウを構成する資源であるリオーダバッファ(ROB: reorder buffer)，発行キュー (IQ: issue queue)，ロード/ストアキュー (LSQ: load/store queue) を拡大し，プロセッサがサポートするインフライト命令数を大幅に増加させるものである．これにより，キャッシュミスを起こすロードの早期実行が可能となり，メモリアクセスを並列化する．このタイプの並列をメモリレベル並列 (MLP: memory-level parallelism) と呼ぶ．

このアプローチを単純に実現しようとすると，資源拡大によりクロック速度を低下させてしまうという問題がある．この問題は，資源をパイプライン化すれば解決するが，代わりに，命令レベル並列 (ILP: instruction-level parallelism) の有効利用を阻害してしまう．これは主に，IQ のパイプライン化による．IQ をパイプライン化すると，ウェイクアップ→セレクトのループが 1 サイクルで完結しなくなり，依存関係にある命令を連続したサイクルで発行できなくなる．

このトレードオフを解決する手法として，我々は，プログラムの実行中の利用可能な並列性 (ILP か MLP か) に適応して，命令ウィンドウの資源サイズを細粒度で動的に変化させる**動的資源リサイジング手法**を提案した[11]．この手法では，実行がメモリインテンシブと予測される場合，資源を拡大し，同時にパイプライン化し，MLP 利用によって性能向上をはかる．逆に，計算インテンシブと予測される場合，資源を縮小し，パイプライン段数を減少させ，ILP を利用し，性能向上をはかる．

この資源リサイジング手法の欠点として，以下の 2 点があげられる．1) 資源拡大時にはより多くの電力を消費する．2) 計算インテンシブなプログラム，あるいは，フェーズの性能を向上させることはできない．

そこで本論文では，アイドルコアによる余剰電力予算を利用し，クロック周波数ブーストと資源リサイジングを動的に切り替える**デュアルターボブースト**という手法を提案する．本手法では，実行中のプログラムのフェーズがメモリインテンシブか計算インテンシブかを時間的に粗い粒度で予測し，それぞれに応じて，資源リサイジングまたはクロック周波数ブーストを選択し，いずれのフェーズについても，アイドルコアの余剰電力を利用し性能向上をはかることができる．

本論文の残りの部分は，以下のような構成となっている．2 節では，資源リサイジング手法を説明する．3 節では，デュアルターボブーストについて提案する．4 節では，デュアルターボブーストの性能限界を見積もるための最適なオラクルモード選択パターンの求め方について述べる．5 節で評価環境を説明し，6 節で評価結果を示す．最後に 7 節で本論文をまとめる．

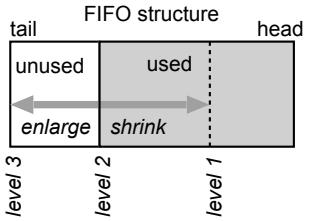


図 1 レベルを変えることによる資源のリサイジング

2. 資源リサイジング手法

本節では，資源リサイジング手法[11] について説明する．積極的なアウトオブオーダ実行により MLP を利用するために必要な資源は，命令ウィンドウを構成する資源である．本研究では，Intel P6[6] タイプのアーキテクチャを仮定している．したがって，命令ウィンドウは ROB，IQ，LSQ からなる．これらの資源は，すべて FIFO で構成される．したがって，ある時点で先頭からあるエントリまで使用していたとするなら，リサイジングとは，**図 1** に示すように，使用領域と不使用領域の境界を移動させることである．この境界には，クロック・ゲート付きのラッチを挿入し，レベルを増加させる場合，ゲートを開き，ラッチをパイプライン・ラッチとして動作させる．レベルを減少させる場合，ゲートを閉じ，境界より上位の領域に信号が伝搬しないようにする．

ここで，リサイズする資源のサイズとそのパイプラインの深さの組の呼称として，**資源レベル**(または，単にレベル) ($level = \{size,\ pipeline\text{-}depth\}$) と呼ぶ用語を定義する．レベルが増加すれば，サイズは増加する．パイプラインの深さは，そのサイズの資源の遅延に応じて決定される．

2.1 資源拡大・縮小の概要

一般に，最終レベルキャッシュミス (以下，文脈に応じて単にキャッシュミスと呼ぶこともある) は，時間についてかたまって生じる傾向があることが知られている．これは，プログラム実行におけるフェーズの変化に応じて，メモリアクセスの局所性が低下する瞬間があるためと考えられる．資源リサイジング手法では，この性質を利用し，一度キャッシュミスが生じたら，続けてしばらくの間ミスが生じると予測し，資源を拡大し，MLP を利用できるようにする．具体的に

```
 1:  cycle = 0;          // current clock cycle
 2:  level = 1;          // resource level
 3:  shrink_timing = -1; // timing of shrink
 4:  do_shrink = 0;      // flag instructing shrink of the resources
 5:
 6:  foreach cycle {
 7:    if (L2_miss) {
 8:      level = min(level + 1, max_level);  // enlarge the resources
 9:      shrink_timing = cycle + memory_latency;
10:      do_shrink = 0;
11:    } else if (cycle == shrink_timing) {
12:      do_shrink = 1;
13:    }
14:    if (level > 1 && do_shrink) {
15:      // check if the regions of ROB, IQ, and LSQ to be removed by shrinking are vacant
16:      if (is_shrinkable(level)) {
17:        level = level - 1;  // shrink the resources
18:        shrink_timing = cycle + memory_latency;
19:        do_shrink = 0;
20:      } else {
21:        stop_alloc();  // stop resource allocation to increase the vacancies in resources
22:      }
23:    }
24:  }
```

図 2 資源リサイジングのアルゴリズム

は，キャッシュミスが生じたら，各資源のレベルを 1 つ増加させる (もし，現在が最大レベルなら，そのまま変化させない)．

一方，最後のキャッシュミスが生じてから主記憶レイテンシが経過したら，今後ミスはしばらく生じないとして，ILP を利用するため資源を縮小する．具体的には，各資源のレベルを 1 つ減少させる (もし，現在が最小レベルなら，そのまま変化させない)．ただし，ROB，IQ，LSQ の削除される領域に命令があるなら，資源割り当てを停止し，その領域に命令がなくなるまで待ち合わせる．

2.2 アルゴリズム

図 2 に，資源リサイジングのアルゴリズムを擬似コードで示す．本研究では，L2 キャッシュを最終レベルキャッシュとしており，それを仮定した記述となっている．

L2 キャッシュミスが生じたサイクルでは，資源を拡大する．すなわち，資源レベルを 1 つ増加させる (現在のレベルがすでに最大レベルなら，そのまま変化させない)(第 8 行)．そして，後に資源を縮小するタイミングを知るため，そのタイミングである `shrink_timing` を，現在のサイクルに主記憶レイテンシを加えたサイクルとする (第 9 行)．加えて，資源の縮小を指示するフラグである `do_shrink` をクリアする (第 10 行)．

L2 キャッシュミスが生じなかったサイクルで，現在のサイクルが以前に定めた資源縮小タイミングに至っていたら，`do_shrink` フラグをセットし，以後，可能なら資源を縮小するよう制御する (第 12 行)．

現在のレベルが 1 より大きく，`do_shrink` フラグがセットされていれば，ROB，IQ，LSQ が同時に縮小可能かをチェックする (第 16 行)．すなわち，縮小により削除する領域に命令が残っていないかチェックする．もし命令が残っていれば，現在サイクルでの縮小は行わず，資源の空きの領域が増加するよう割り当てを停止し，縮小を後のサイクルに延期する (第 21 行)．もし命令が残っていなければ，資源を縮小する．すなわち，資源レベルを 1 つ減少させる (第 17 行)．そして，`shrink_timing` を，次回の縮小のために，現サイクルに主記憶レイテンシを加えたサイクルとし，`do_shrink` フラグをクリアする (第 19 行)．

図 3 に，どのように資源レベルが遷移するかの例を示す．レベルの最大値を 3 と仮定する．時刻 *t0* に，L2 キャッシュミスが起こったとすると，レベルが 1 つ増加される．同様に，時刻 *t1* に，さらに L2 キャッシュミスが起こり，再びレベルは増加され 3 になる．時刻 *t2* に再び L2 キャッシュミスが起こるが，今度はレベルが最大値になっているので，そのまま変化しない．時刻 *t4* で，最後の L2 キャッシュミスから主記憶レイテンシが経過した．そこで，レベルは 1 減少される．さらに主記憶レイテンシが経過する時刻 *t5* では，レベルは再び 1 減少される．ここで，時刻 *t1*〜*t3* の期間，メモリアクセスがオーバラップしており，MLP

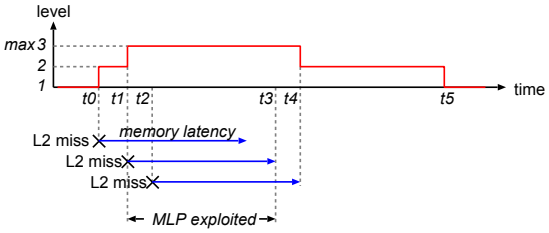


図 3　L2 キャッシュミスによるレベル遷移の様子

が利用される．

3. デュアルターボブースト

本節では，提案手法であるデュアルターボブーストの動作を説明する．デュアルターボブーストは，実行しているプログラムのフェーズが，時間軸上の粗い間隔で，メモリインテンシブか計算インテンシブかどうかを動的に予測する．メモリインテンシブと予測した場合は，MLP 利用による性能の改善を図るために，資源リサイジングモードで動作させる．逆に，計算インテンシブと判断した場合は，クロック周波数ブーストモードで動作させる．

ここで，注意していただきたいことは，モード切り替え時に，プロセッサをしばらく停止する必要があるということである．この停止期間は，クロック周波数と電源電圧を変更するために要する．この時間は，$10\mu s$[7] 程度 (5 節の評価パラメータでは 30k サイクル) と非常に長いので，モード切り替えによる損失を利益が上回るためには，少なくとも，切り替え後，そのモードに長くとどまらなければならない．このため，どちらの性質を持つ期間が現れるかどうかを予測する方法として，資源リサイジングで用いている細粒度な手法は適切でなく，粒度の粗い方法が適している．そこで，比較的長い期間のインターバルを設ける．あるインターバルでメモリインテンシブ，または，計算インテンシブであるなら，次のインターバルも，同様であると予測し，それぞれ，資源リサイジングモード，クロック周波数ブーストモードで動作させる．

資源リサイジングモードのインターバルでは，計算インテンシブな期間が占める割合は少ないが，資源リサイジング手法により，資源を縮小し，通常クロック周波数の下ではあるが，ILP を利用することができる．

メモリインテンシブかどうかの判断には，1k 命令あたりの L2 キャッシュミス回数 (MPKI: misses per kilo instructions) を用いる．あるインターバルの MPKI があらかじめ定めたしきい値以上であれば，次のインターバルはメモリインテンシブと予測し，そうでなければ，計算インテンシブと予測する．ただし，前述したモード切り替えのペナルティを被る頻度を少なくするため，**図 4** に示すように，モード遷移にヒステリシスをもたせる．すなわち，しきい値を高低 2 つ設け，周波数ブーストモードから資源リサイジングモードへの遷移には，MPKI が高しきい値を上回ることを条件とし，逆方向の遷移には，低しきい値を上回ることを条件とする．こうすることにより，モードにとどまる時間が長くなり，モード遷移頻度が低下する．

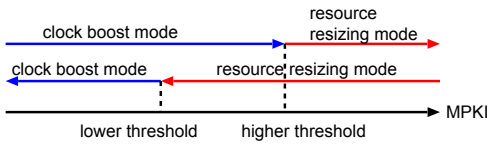


図 4　ヒステリシスをもったモード遷移

4. オラクルな最適モード選択

前述したように，モード切替時には，プロセッサを停止させなければならない．性能を向上させるためには，このペナルティを超える利益がモード切り替え後に得られなければならない．本節では，最初にこの点におけるトレードオフについて説明する．次に，このトレードオフを考慮した最適なモード切り替えパターンを求める方法について説明する．このモード選択は，将来のモード選択に依存するため，実際のアルゴリズムとして適用することはできない．つまり，このモード選択はオラクルである．このモード選択による性能評価を行うことにより，3 節で述べた方式が，最適な選択を行った場合に対してどの程度性能を失っているかを，6 節で評価する．

4.1 モード遷移におけるトレードオフ

3 節で述べた方式では，モード遷移のペナルティを定量的に考慮していないという点で理想的な方式ではない．この点を**図 5** を用いて説明する．

図 5 は，モード遷移の 2 つのパターンを表している．インターバル 1 と 3 はメモリインテンシブであり，インターバル 2 は計算インテンシブとする．上のパターンの場合 (switch case) では，インターバル 2 を計算インテンシブと正しく予測し，資源リサイジングからクロック周波数ブーストにモード遷移している．その後，インターバル 3 で再び資源リサイジングに遷移している☆．2 度の遷移においては，それぞれプロセッ

☆ 3 節で述べた方式では，現在のインターバルの MPKI によって次のインターバルのモードが決まるので，このような遷移は起こらないが，ここでは説明を簡単にするため，遷移パターンを単純化している．

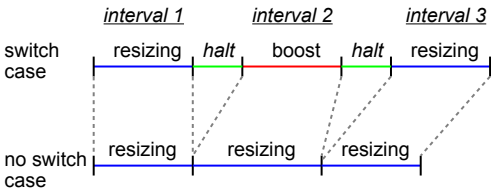


図 5　モード遷移におけるトレードオフ

サは停止し，ペナルティを支払っている．

これに対して，下のパターンの場合 (no switch case) は，一度も遷移しない場合を表している．インターバル 2 は計算インテンシブなので，上の場合に比べて実行時間が長い．しかし，遷移に伴うペナルティを支払うことがなく，インターバル 3 までの総実行時間は，下の遷移しない場合の方が短い．

4.2　オラクルモード選択の求め方

あるインターバルにおいてモード遷移すべきかどうかは，その将来のモード遷移パターンに依存している．そこで，次のようにして最適なモード遷移パターンを求める．

(1) プログラムを全期間資源リサイジングモードで実行し，各インターバルの実行時間を記録して保存する．同様に，全期間クロック周波数ブーストモードで実行し，各インターバルの実行時間を保存する．

(2) $N(\gg 1)$ 個のインターバルをまとめてタームと呼ぶ．ターム毎に (3) を繰り返すことにより，タームにおける最適なモードのパターン p_{opt} を求める．

(3) ターム k において，含まれるインターバルが取りうるモードの組み合わせ p の全て (2^N 通り) に対して，実行時間を以下のようにして計算する．

$$exec\_time(p) = \sum_{i=k\times N}^{(k+1)\times N-1} t_m(i) + n(p) \times halt\_time$$

ここで，$t_m(i)$，$halt\_time$，$n(p)$ は，それぞれ，(1) で保存したインターバル i のモード m での実行時間，モード遷移時のプロセッサ停止時間，モードの組み合わせ p におけるモード遷移の回数である．実行時間 $exec\_time(p)$ が最小となるモードの組み合わせが，ターム k の最適なモードのパターン p_{opt} である．

5. 評価環境

性能の評価には，SimpleScalar Tool Set Version 3.0a[1] をベースに提案手法を実装したシミュレータを用いた．命令セットは DEC Alpha ISA である．ベンチマークプログラムとして，SPEC2000 を使用した．

表 1　ベースプロセッサの構成

Pipeline width	4-instruction wide for each of fetch, decode, issue, and commit
ROB	128 entries
IQ	64 entries
LSQ	64 entries
Function unit	4 iALU, 2 iMULT/DIV, 2Ld/St, 4 fpALU, 2 fpMULT/DIV/SQRT
L1 I-cache	64KB, 2-way, 32B line
L1 D-cache	64KB, 2-way, 32B line, 2ports, 2-cycle hit latency, non-blocking
L2 cache	2MB, 4-way, 64B line, 12-cycle hit latency
Main memory	300-cycle min. latency, 27.2GB/sec bandwidth
Branch prediction	16-bit history 64K-entry PHT gshare, 2K-set 4-way BTB, 10-cycle misprediction penalty
Data prefetcher	stride-based, 4K-entry, 4-way pred. table, 16-data prefetch to L2 cache on miss
Clock freqency	3.0GHz

バイナリは DEC/Compaq コンパイラを用い，-fast -O4 のオプションでコンパイルし作成した．各プログラムについて，以下の 2 つの実行区間を評価した．

- **シビア区間**: 最初の 1G 命令をスキップした次の 5G 命令の内，資源リサイジングモードで実行した場合，レベル 1 とそれより上のレベル間を遷移した回数が最も多い 1G 命令の区間．この区間では，計算インテンシブとメモリインテンシブなフェーズを多く繰り返し，デュアルターボブーストにおいて，最適なモード選択が困難な区間である．
- **SimPoint 区間**: SimPoint[5] によって選んだ 100M 命令の区間．この区間は，SPEC ベンチマークの各プログラムを最も代表する区間である．

評価の基準となるベースプロセッサの構成を**表 1** に示す．

5.1　資源リサイジングに関する仮定

各資源レベルでのサイズは，次のようにして定めた．最初に，レベル 1 の IQ をベースプロセッサの構成 (64 エントリ，1 段パイプライン) とし，その遅延を求めた．その遅延の L 倍の遅延を持つ IQ のサイズを求め，それをレベル L の IQ のサイズとする (パイプライン段数は L とする)．次に，レベル L の他の資源のサイズは，同レベルの IQ のサイズとバランスするように定めた．つまり，レベル L の IQ サイズが，レベル 1 のそれの M 倍なら，その他の資源のレベル L のサイズは，レベル 1 のそれの M 倍とした．

IQ の遅延は，HSPICE により回路シミュレーションを行い得た[9]．このシミュレーションでは，32nm LSI

表 2 各レベルでの資源のサイズとパイプライン段数

resource	parameter	level		
		1	2	3
IQ	entries	64	384	544
	pipeline depth	1	2	3
ROB	entries	128	768	1088
	pipeline depth	1	2	2
LSQ	entries	64	384	544
	pipeline depth	1	2	3

表 3 クロック周波数ブーストにおける仮定

Clock frequency	3.9GHz
Main memory	390-cycle min. latency, 27.2GB/sec bandwidth

プロセスの MOSIS の設計ルール[2] を仮定し，アリゾナ州立大学が開発した予測トランジスタモデル[3),10)] を使用した．

ROB のパイプライン段数については，割り当てとコミットは IPC に影響しないが，レジスタフィールドの読み出しは，分岐予測ミスペナルティに影響を与える．レジスタフィールドの読み出し遅延は，CACTI[8] を使って求め，パイプライン段数を決定した．パイプライン段数の増加分だけ，分岐予測ミスペナルティが増加する．

LSQ のパイプライン段数については，遅延測定が難しく，単純に，IQ のそれと同じとした．

表 2 に，各レベルにおける資源のサイズとパイプライン段数を示す．

2 節で述べたように，資源レベルの切り替えは，FIFO の使用領域と不使用領域の境界のラッチのクロック・ゲートの開閉で行われる．これは非常に単純なので，要する時間は長くないと推測できる．今回のシミュレーションでは，10 サイクルを仮定した．

5.2 クロック周波数ブーストに関する仮定

特に断りのない限り，クロック周波数ブースト時のクロック周波数は，Intel Ivy Bridge のハイエンドのプロセッサ (通常モードで 3.0GHz) のターボブースト時のクロック周波数である 3.9GHz[4] とした．主記憶については，ブースト時も，その速度は変わらないので，バンド幅は変わらない．一方，クロックサイクルを単位とするレイテンシは増加する．**表 3** に，クロック周波数ブースト時におけるパラメータをまとめる．

5.3 デュアルターボブーストに関する仮定

モード遷移においては，クロック周波数と電源電圧を変えなければならない．これは，DVFS(dynamic voltage/frequency scaling) と同じ技術が使われ，10μs 停止するとした[7]．また，予備評価によってインターバル長は 100K 命令とした (紙面の制限上評価結果を示さないが，インターバル長の性能への感度は低い)．加えて，以下断りのない限り，モード遷移の低しきい値と高しきい値を，それぞれ，0.5，2.9 とした．これらの値は，シビア区間で性能のベンチマーク平均が最大となる値である．しきい値に対する性能感度は，6.3 節で評価する．

6. 評　　価

6.1 性　　能

以下の 5 つのモデルを評価した．

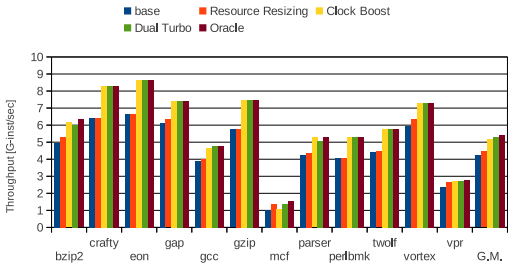
- **ベースモデル:** 資源リサイジングもクロック周波数ブーストも行わないモデル
- **資源リサイジングモデル:** 常に資源リサイジングモードで動作するモデル
- **クロックブーストモデル:** 常にクロック周波数ブーストモードで動作するモデル
- **デュアルターボモデル:** デュアルターボで動作するモデル
- **オラクルモデル:** 4 節で説明した方法で得られた最適なモード選択で動作するデュアルターボモデル

6.1.1 シビア区間での評価結果

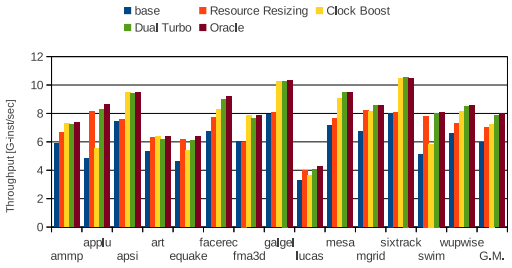
図 6 に，シビア区間での性能 (命令実行のスループット) の測定結果を示す．同図からわかるように，SPECint2000 では，ほとんどのプログラムで資源リサイジングよりクロックブーストの方が高い性能を示している．これは，SPECint2000 の多くのプログラムは計算インテンシブであるからである．これに対し，デュアルターボはそれらのプログラムでクロックブーストとほぼ同等の性能を示している．例外として，*mcf* はクロックブーストよりも資源リサイジングのほうが高い性能を示しているが，これに対してもデュアルターボはほぼ同等の性能を得ている．

SPECfp2000 では，SPECint2000 とは異なり，クロックブーストよりも資源リサイジングのほうが高い性能を示しているプログラムも多くある．これは，SPECfp2000 にはメモリインテンシブなプログラムをより多く含んでいるからである．デュアルターボの性能を見ると，それらのプログラムにおいて，同等の性能を示している．逆に，資源リサイジングよりもクロックブーストのほうが高い性能を示すプログラムについても，デュアルターボは同等の性能を示している．

興味深いプログラムとして，*facerec*，*mesa*，*mgrid*，*wupwise* に着目いただきたい．これらのプログラムでは，デュアルターボは，資源リサイジングとクロックブーストのどちらよりも非常に高い性能を示している．これらのベンチマークではメモリインテンシブと計算

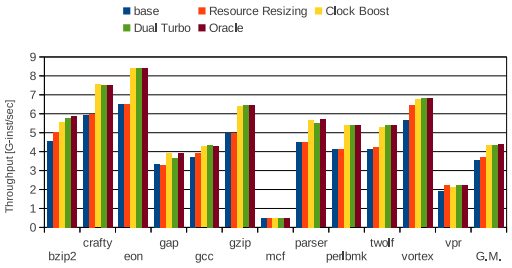


(a) SPECint2000

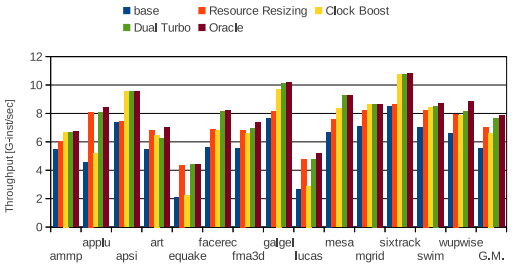


(b) SPECfp2000

図 6 シビア区間でのスループット



(a) SPECint2000



(b) SPECfp2000

図 7 SimPoint 区間でのスループット

インテンシブなフェーズが適度にミックスされており，デュアルターボの特性がよく生かされている．

デュアルターボとオラクルを比較すると，デュアルターボはほぼ同等か，やや劣る程度の性能 (平均で 2%劣るのみ) を達成している．このことより，提案手法は非常に単純であるが，高い適応力を持っていることがわかる．

全てのプログラムの平均で，デュアルターボモデルはベースモデル，資源リサイジングモデル，クロックブーストモデルよりも，それぞれ，29%，16%，6%高い性能を達成している．

6.1.2 SimPoint 区間での評価結果

図 7 に，SimPoint 区間での命令実行のスループットの測定結果を示す．シビア区間での測定結果とほぼ同様の傾向が見られるが，一般的に，シビア区間よりモード遷移のペナルティを被る頻度が小さいため，より高い性能向上を示している．(ベースに対する性能向上率は，前述したように，シビア区間では 29%であるが，SimPoint 区間では 30%である)．全てのプログラムの平均で，デュアルターボモデルは資源リサイジングモデル，クロックブーストモデルよりも，それぞれ，12%，8%高い性能を達成している．

6.2 モード遷移しきい値に対する性能の感度

3 節で述べたように，モード遷移のしきい値として，2 つの値がある．これらに関する性能の感度を調べるために，1) 2 つのしきい値の中央値，2) 最適な中央値 (MPKI=1.7) における 2 つのしきい値の差 (以下，ギャップと呼ぶ)，に対する感度を調査した．

まず，しきい値の中央値に対する感度であるが，極端に小さいか大きくなければ，性能に大きな変化はなかった．

次に，ギャップに対する性能感度を**図 8** に示す．横軸はギャップであり，低しきい値が 0.1 になるまでの範囲で評価した．縦軸は，最適なしきい値でのスループットで正規化したスループットである．測定区間は，SimPoint 区間である．折れ線グラフは 3 本あり，SPECint2000，SPECfp2000，及び，しきい値に敏感な 5 つのプログラム (*bzip2*，*gap*，*art*，*facerec*，*swim*) それぞれの平均である．

図からわかるように，SPECint2000，SPECfp2000 ともに，平均では，中央値と同様ギャップに対する性能の感度は鈍く，最適な値からいくらかはずれた設定を行なっても高い性能を達成できることがわかる．これは，前述したように，それぞれのベンチマーク・スイートで，計算インテンシブあるいはメモリ・インテンシブに偏ったプログラムが多いからである．しかし，しきい値に敏感なベンチマークも存在し (Threshold sensitive programs の折れ線を参照)，ギャップを 0 と

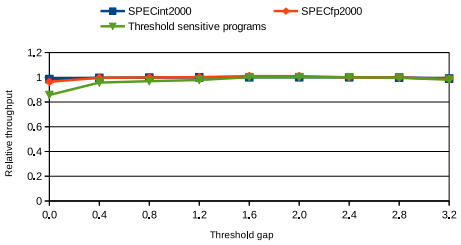


図 8 モード選択しきい値に対する性能の変化

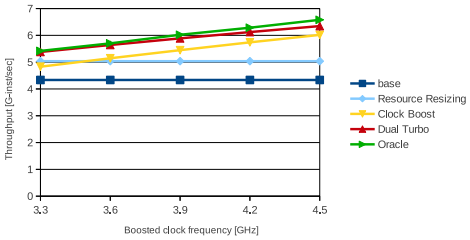


図 9 ブーストクロック周波数に対する性能の変化

すると，大きく (14%) 性能が低下する．これは，モード遷移回数が多くなり，遷移ペナルティを頻繁に被るためである．ペナルティを被る頻度を下げるために，ギャップを設ければ，その増加に応じて性能は緩やかに向上し最大値を取った後，再び緩やかに低下していく．

6.3 クロック周波数ブースト率に対する性能の感度

図 9 に，ブーストクロック周波数を変えた時のベンチマーク平均性能を示す．測定区間は，SimPoint 区間である (シビア区間でも同様の測定結果となる)．図からわかるように，ブーストクロック周波数を上昇させると，当然ながら，クロックブーストモデルの性能は向上する．それに応じて，デュアルターボブーストモデルの性能も向上し，測定範囲の最大ブーストクロック周波数 4.5GHz で，ベース，資源サイジング，クロックブーストに対し，それぞれ，46%，26%，6%の性能向上を達成している．オラクルに対する性能低下は，依然として非常に小さく，ブーストクロック周波数 3.3GHz の時が最大であり，1%である．

7. ま と め

本論文では，マルチコアプロセッサにおけるシングルスレッド性能の向上手法として，クロック周波数ブーストと資源リサイジング手法を組み合わせるデュアルターボブーストと呼ぶ手法を提案した．この手法では，実行フェーズが，計算インテンシブかメモリインテンシブかのどちらかでしか有効でない単一の手法と異なり，どちらのフェーズにおいても，余剰電力予算を性能向上に結びつけることができる．

評価の結果，プログラムがメモリインテンシブか計算インテンシブかに応じて，適切に 2 つの手法を切り替えることができ，その結果，資源リサイジングのみ，クロック周波数ブーストのみの場合に比べ，デュアルターボブーストは，それぞれ，12%，8%高い性能を達成することを確認した．

謝辞

本研究の一部は，日本学術振興会 科学研究費補助金基盤研究 (C)(課題番号 22500045，25330057) および若手研究 (A)(課題番号 24680005) による補助のもとで行われた．また，本研究は東京大学大規模集積システム設計教育研究センターを通し，シノプシス株式会社の協力で行われたものである．

参 考 文 献

1) `http://www.simplescalar.com/`.
2) `http://www.mosis.com/`.
3) `http://www.eas.asu.edu/~ptm/`.
4) http://ark.intel.com/products/71096/Intel-Core-i7-3940XM-Processor-Extreme-Edition-8M-Cache-up-to-3_90-GHz.
5) Hamerly, G., Perelman, E., Lau, J. and Calder, B.: SimPoint 3.0: Faster and More Flexible Program Phase Analysis, *The Journal of Instruction-Level Parallelism*, Vol. 7, pp. 1–28 (2005).
6) Intel: *P6 Family of Processors - Hardware Developer's Manual* (1998).
7) McGregor, J.: x86 Power and Thermal Management, *Microprocessor Report*, Vol. 18, Archive 12, pp. 1–6 (2004).
8) Muralimanohar, N., Balasubramonian, R. and Jouppi, N. P.: CACTI 6.0: A Tool to Model Large Caches, HPL-2009-85, HP Laboratories (2009).
9) Yamaguchi, K., Kora, Y. and Ando, H.: Evaluation of Issue Queue Delay: Banking Tag RAM and Identifying Correct Critical Path, *Proceedings of the 29th International Conference on Computer Design*, pp. 313–319 (2011).
10) Zhao, W. and Cao, Y.: New Generation of Predictive Technology Model for Sub-45nm Design Exploration, *Proceedings of the 7th International Symposium on Quality Electronic Design*, pp. 585–590 (2006).
11) 甲良祐也, 安藤秀樹: MLP に着目したパイプライン化発行キューの動的サイジング, 2011 年先進的計算基盤システムシンポジウム SACSIS 2011, pp. 72–81 (2011 年).