Vol.2012-ARC-201 No.18
2012/8/2

動的な資源のリサイジングを組み合わせたデュアルターボブースト

山口 恭平[1]　塩谷 亮太[1]　安藤 秀樹[1]

概要：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，シングル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という問題がある．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判断し，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，クロック周波数ブーストのみ，あるいは，資源リサイジングのみを使用したときと比較し，デュアルターボブーストはそれぞれ，19%，1.5%の性能向上が得られた．その結果，平均では，28%の性能向上が得られた．

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，すべてのコアが動作している場合は少なく，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，メモリ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，メモリアクセス時間が実行時間の多くを支配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．これはメモリレベル並列性（MLP: memory-level parallelism）を利用し，メモリレイテンシを隠蔽しようとするものである．この積極的なアウトオブオーダ実行のためには，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必要がある．そのためには，この数を規定する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) を拡大しなければならない．しかしこれらの資源の単純な拡大は，クロック速度を低下させてしまうという問題がある．また，メモリインテンシブでない実行フェーズでは，ILP の利用が阻害され，性能が低下してしまうという問題もある．

これらの問題を解決するために，我々は，パイプライン化した大きな資源を用意し，プログラムの実行中に利用可能な並列性 (ILP か MLP か) に適応して，その資源のサイズを動的に変化させるという**資源リサイジング手法**を提案した [11]．この手法では，まず，大きな資源をパイプライン化することによりクロックへの悪影響を除く．その上で，実行がメモリインテンシブかどうかを判別し，もしそうならば資源を拡大し，MLP 利用によって性能向上をはかる．逆に，計算インテンシブならば，資源を縮小し，パイプライン段数を減少させ，ILP(instruction-level parallelism) 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，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

[1] 名古屋大学大学院工学研究科
Graduate School of Engineering, Nagoya University

う点が挙げられる．また，メモリインテンシブなフェーズでの資源拡大では，電力を多く消費するという問題がある．

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予測し，それぞれに応じて，資源リサイジングかクロック周波数ブーストかを選択し，どちらの場合も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紹介する．3 節ではデュアルターボブーストの動作について述べる．4 節で評価環境を説明し，5 節で評価結果を示す．最後に 5 節で本論文をまとめる．

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．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，使用領域と不使用領域の境界を移動することである．

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 1 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 17 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 22 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 18 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 20 行)．

図 2 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 が利用される．

3. デュアルターボブーストの動作

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，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判断する．そしてメモリインテンシブと判断した場合は，MLP 利用による性能の改善を図

```
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 that are to be removed
16:     // by shrink are vacant
17:     if (is_shinkable(level)) {
18:       level = level - 1;  // shrink the resources
19:       shrink_timing = cycle + memory_latency;
20:       do_shrink = 0;
21:     } else {
22:       stop_alloc();
23:     }
24:   }
25: }
```

図 1　資源リサイジングのアルゴリズム

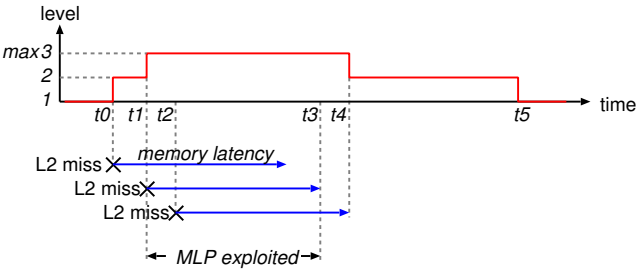


図 2　L2 キャッシュミスによるレベル遷移の様子

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．

以上を実現するために，一定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であるなら，次のインターバルもメモリインテンシブであると予測する．計算インテンシブの場合も同様である．ここで，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．MPKI があらかじめ定めたしきい値以上であれば，メモリインテンシブと判断し，そうでなければ，計算インテンシブと判断する．

具体的な動作を示す．まず，資源リサイジングモードで最初のインターバルを実行する．実行中の L2 キャッシュミス回数を数え，インターバルが終了したら MPKI を求める．これがあらかじめ定めたしきい値を超えていたら，次のインターバルはメモリインテンシブであると予測し，資源リサイジングモードで動作させる．2.2 節で述べたアルゴリズムで，資源レベルを変化せる．逆に，MPKI がしきい値を超えていなければ，次のインターバルは計算インテンシブであると予測し，資源レベルを 1 とし，クロック周波数ブーストモードで動作させる．以上を繰り返す．

モード切替時には，クロック周波数と電源電圧を変更しなければならない．このため，DVFS(dynamic voltage/frequency scaling) と同様，遷移の間プロセッサをしばらく停止する必要がある．

4. 評価環境

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 から整数系プログラムを 12 本，浮動小数点系プログラムを 14 本を使用した．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．ref 入力を用い，SimPoint [5] によって選んだ 100M 命令を実行した．

評価の基準となるベースモデルの構成を表 1 に示す．

表 1　ベースモデルのプロセッサ構成

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, 13.6GB/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	3.4GHz

4.1　資源サイズとパイプライン段数

各資源レベルでのサイズは，次のようにして定めた．まず最初に，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 N 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 N 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 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．なお，レベルの最大値を 3 としたが，これ以上大きくしても性能向上が頭打ちとなるからである．

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 of register field read	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

Main memory	327-cycle min. latency, 13.6GB/sec bandwidth
Clock frequency	3.8GHz

4.2　クロック周波数ブーストに関する仮定

クロック周波数ブースト時のクロック周波数は，Intel Sandy Bridge のハイエンドのプロセッサ (通常モードで 3.4GHz) のターボブースト時のクロック周波数 [4] により決定した．

主記憶のレイテンシについては，ブースト時も，主記憶の速度は変わらないので，クロックサイクルから見たレイテンシは増加する．バンド幅 (GB/sec) は変わらない．

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．

4.3　デュアルターボブーストに関する仮定

3 節で述べたように，モード切替時には，クロック周波数と電源電圧を変更しなければならないため，プロセッサをしばらく停止する必要がある．技術的には，DVFS と同じであり，10μs とした [7]．

モードを切り替えるインターバルは，今回の評価では 8M サイクルとした．

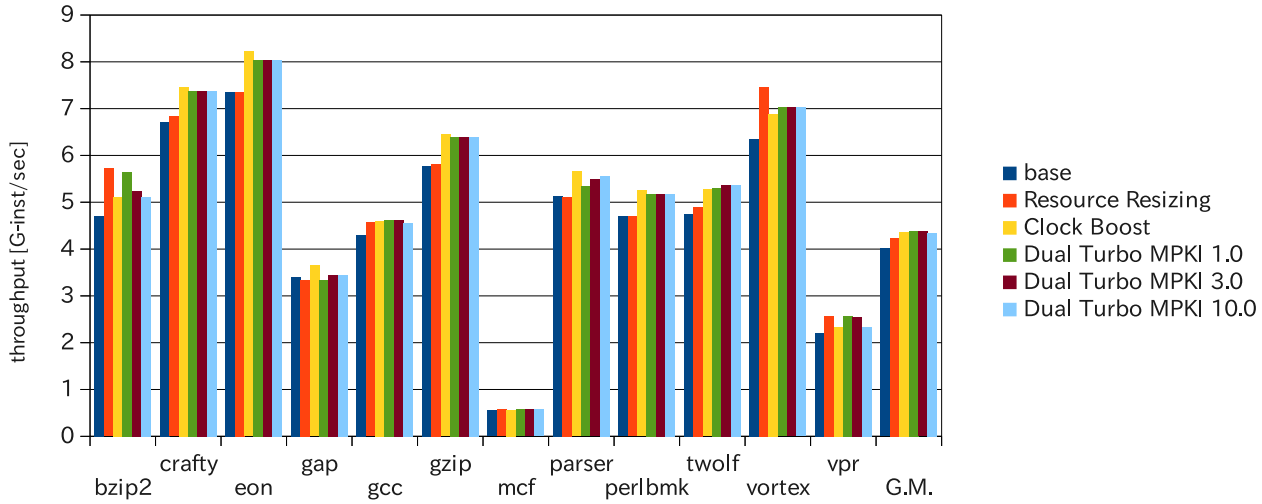
5.　評価

5.1　各モデルのスループット

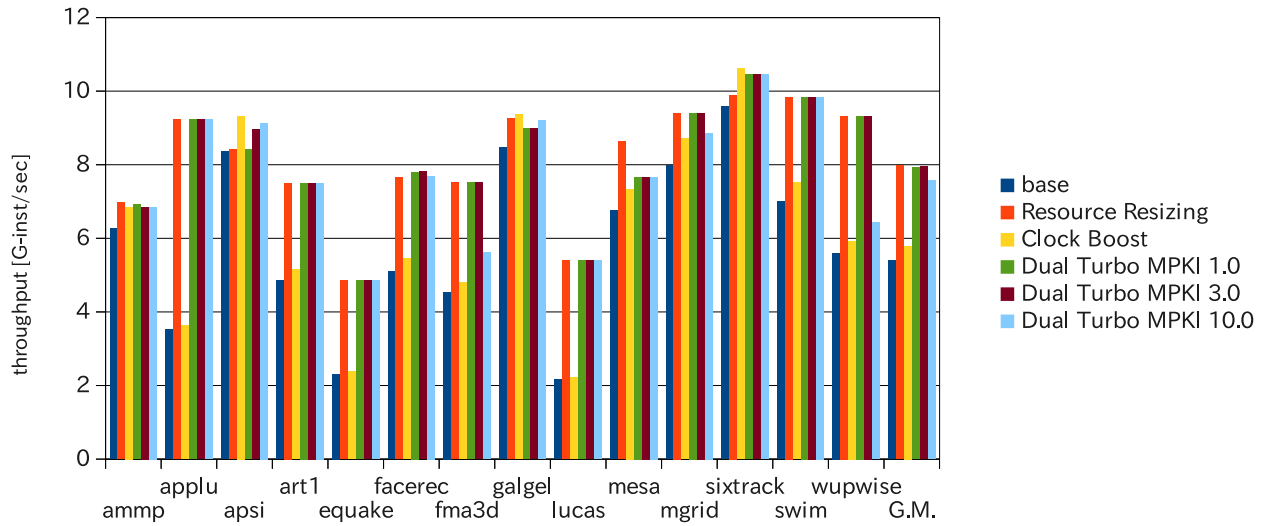
図 3 に，評価した命令処理のスループットを示す．各ベンチマークプログラムには，6 本の棒グラフがあり，以下のモデルの性能 (スループット) を示している．

- **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

デュアルターボモデルでは，メモリインテンシブかどうかを判断する基準として，MPKI = 1.0, 3.0, 10.0 の 3 つ



(a) SPECint2000



(b) SPECfp2000

図 3 命令処理のスループット

の場合を測定した．

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，*bzip2*，*vortex* では，クロックブーストより資源リサイジングが高い性能を示している．*vortex* では，デュアルターボは資源リサイジングにやや劣るものの，*bzip2* では，MPKI しきい値 1.0 のデュアルターボが同等の性能を示している．平均では，単体モードとしてはより高性能なクロックブーストと同等の性能をデュアルターボは達成しており，うまく適応できていると言える．

逆に，SEPCfp2000 では，ほどんどのプログラムで，クロックブーストより資源リサイジングの方が高い性能を示している．これは，SPECfp2000 の多くのプログラムはメモリ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資源リサイジングとほぼ同等の性能を示している．例外として，*apsi*，*sixtrack* では，クロックブーストの方が高い性能を示しているが，いずれも，しきい値の MPKI を適切に選べば，デュアルターボは同等の性能を達成することができる．平均では，単体モードとしてはより高性能な資源リサイジングと同等の性能をデュアルターボは達成おり，SPECint2000 の場合と同様，うまく適応できていると言える．

表 4　モード切り替え回数

スイート	プログラム	遷移回数/遷移機会
SPECint2000	bzip2	2 / 8
	crafty	1 / 6
	eon	1 / 5
	gap	2 / 13
	gcc	3 / 9
	gzip	1 / 7
	mcf	0 / 74
	parser	3 / 8
	perlbmk	1 / 9
	twolf	1 / 8
	vortex	1 / 6
	vpr	2 / 16
SPECfp2000	ammp	1 / 6
	applu	0 / 4
	apsi	1 / 5
	art	0 / 5
	equake	0 / 8
	facerec	2 / 5
	fma3d	0 / 5
	galgel	3 / 5
	lucas	0 / 7
	mesa	2 / 6
	mgrid	0 / 4
	sixtrack	1 / 4
	swim	0 / 4
	wupwise	0 / 4

デュアルターボは，全てのベンチマークの幾何平均で，ベース，クロックブースト単体，資源リサイジング単体に対して，それぞれ，19%，1.5%，28%の性能向上を達成できた．

5.2　モード遷移の回数

表 4 に，デュアルターボブーストにおいて，最も高い性能が得られたしきい値 MPKI=3.0 の場合のモード遷移回数を示す．表からわかることは，まず第 1 に，モード遷移の機会が多くないことである．これは，実行期間に対して，インターバルを 8M サイクルと長く設定したためである．モード遷移時間が 10μs なので，インターバルを短くして，遷移機会をあまり頻繁にすることは好ましくない一方で，より細かく制御することでより最適化できる可能性がある．

第 2 に，遷移回数が少ないことがわかる．これは，評価期間中にフェーズがあまり遷移しないことを表している．今回評価期間を SimPoint で選んだので，これは当然のことではあるが，デュアルターボブーストの遷移アルゴリズムを検証し，より良いものにするには，フェーズが変化する期間を評価することが必要である．

6.　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，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どうかを判断し，他のアイドルコアの電力予算を用いて資源のリサイジングとクロックブーストという 2 つの手法を使い分けるというものである．

測定の結果，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を適切に判別し，2 つの手法を切り替えることにより性能向上が得られることがわかった．

今後の課題としては，メモリインテンシブなフェーズと計算インテンシブなフェーズがよく混合した評価区間を選択し，より高い性能が得られるモード選択アルゴリズムを考案することである．

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）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/52214/Intel-Core-i7-2600K-Processor-(8M-Cache-3_40-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).