低電力 Java プロセッサのための投機的クロック制御

木村篤彦[†1] 中島康彦[†2] 宮田佳昭[†3]
中川伸二[†3] 五島正裕[†4] 森眞一郎[†4]
北村俊明[†5] 富田眞治[†4]

単純なパイプライン構造によりモデル化した Java プロセッサにおいて，演算ユニット単位のクロック投入制御を行った場合の電力削減効果と性能とのトレードオフについて述べる．クロック分配系統のより上流においてクロックを制御するほど，電力削減効果は高くなる一方，デコードサイクルに続く演算サイクルが動作可能となるまでのペナルティは増加するため性能は低下する．SPECJVM98 を用いて，演算ユニットごとにクロックを投機的に投入するための予測表の構成について検討した結果，エントリ数が等しい予測表の中では，メソッド識別子の下位 2 ビットおよび PC の下位 6 ビットを用いた 256 エントリの表，命令に関わるデータ型の識別子 3 ビットを用いた 8 エントリの表が，それぞれ高い効果を発揮することを示す．また，8 ビットの情報を用いた場合，予測による性能向上が消費電力増加を上回るものの，3 ビットの情報では逆転することを明らかにする．

A Speculative Clock Control for Low Power Java Processors

Atsuhiko Kimura,[†1] Yasuhiko Nakashima,[†2] Yoshiaki Miyata,[†3]
Shinji Nakagawa,[†3] Masahiro Goshima,[†4] Shinichiro Mori,[†4]
Toshiaki Kitamura[†5] and Shinji Tomita[†4]

This paper describes a power saving technique for a Java processor modeled with a simple pipeline. We assume several execution units and related clock-trees are individually driven from higher-level gated clock distributors. The program counter or the instruction decoder predicts the associated units and speculatively starts and stops these distributors. The performance degradation derived from the clock delay and the unnecessary power consumption caused by miss predictions are tradeoffs. We evaluate these speculative methods on SPECJVM98 benchmark programs and found a prediction table that holds 256-entries indexed by the lower 2-bits of the method-ID and the lower 6-bits of the program-counter, or a smaller table that holds only 8-entries indexed by 3-bits each corresponds the data types of the stack-top show remarkable effectiveness respectively. Finally we show the 8-bit information gains higher performance than power consumption, though the 3-bit information dissipates the power.

1. はじめに

大量の電力を投入して実行速度を向上させてきたプロセッサ開発競争は，低電力化という新たな方向へ進路を変えつつある．理由の 1 つは，情報機器の小型化および可搬化にともない，バッテリー駆動時間が重視されてきたため．もう 1 つは，サーバ用途プロセッサにおいても，開発コストを下げるためには液冷や液浸などの高価な冷却技術ではなく空冷を採用する必要があるためである．また，超並列計算機の要素プロセッサとして使用する場合，運用時の電力コストが大きな問題となる．現実に，本来保守のために装備されている，商用並列スーパコンピュータの部分的切り離し機構は，夜間の節電のためにも用いられている．このように，低電力化は避けて通れない課題である．

本稿では，命令レベルの情報を用いて，性能を低下させることなく，演算ユニット単位の消費電力を抑え

†1 株式会社ビーコンインフォメーションテクノロジー インターネット事業部
Internet Department, Beacon Information Technology Inc.
†2 京都大学大学院経済学研究科
Graduate School of Economics, Kyoto University
†3 オムロン株式会社技術本部 IT 研究所
Information Technology Research Center, OMRON Corporation
†4 京都大学大学院情報学研究科
Graduate School of Informatics, Kyoto University
†5 広島市立大学情報科学部
Faculty of Information Sciences, Hiroshima City University

ることを狙う．低電力化を考慮しない場合，プロセッサの各演算ユニットには，内部レジスタから読み出されたデータが常時供給され，演算結果を使用するか否かにかかわらず，演算ユニットの内部信号を変化させる．また，演算に複数サイクルを必要とする場合，同様に，演算ユニット内部のラッチが無用のデータを次段に伝播させ，次段の内部信号を変化させる．一方，本稿では，各演算ユニットの入口にラッチを設け，レジスタからの読み出しデータが，演算に関与しない演算ユニットの内部信号に影響を与えないよう工夫することにより，前者に関する電力を削減する．また，演算ユニット内部のラッチに対するクロック供給を停止させることにより，後者に関する電力を削減する．さらに，ラッチを開閉させるために必要なクロック分配器の動作を停止させ，クロック分配系統自身の電力削減を図る．

より多くの電力を削減するためには，クロック分配系統のより上流においてクロックを停止する必要がある．しかしながら，上流のクロック分配器から各演算ユニットのラッチに至る経路には遅延時間が存在する．命令のデコード結果から必要な演算ユニットを特定し，クロック分配器の動作を開始させる場合，動作周波数が高くなるほど，演算ユニットの動作開始が間に合わなくなる．このような場合に，いかに演算ユニットを効率良く動作させるかについての報告はまだなされていない．

以下では，単純なパイプライン構造によりモデル化した Java$^{(TM)}$☆1プロセッサに，現実の Java プロセッサ JeRTy$^{(TM)}$☆2における演算ユニットの電力消費モデルをあてはめ，さらに，演算に必要な演算ユニットを予測する極力小さなハードウェア機構を装備した場合について，クロック分配器の動作開始が演算に間に合わないことによる性能低下，および，クロック分配器の動作停止が間に合わないことによる電力増加をいかに抑えるかについて詳述する．

2. 関 連 研 究

プロセッサの低電力化に関しては，きわめて多くの研究成果が報告されている．プロセッサの全体や一部をスリープモードに遷移させる方法[1]，周波数や電圧を可変とする方法[2)~4)]，状態遷移ループを検出して遷移を止める方法[5),6)]，非同期回路の適用[7)]，パストランジスタなど低電力論理の採用[8),9)]，SOI など素子レベルの工夫[10)]など，様々なレベルの方法が提案されている．低電力化をめざした主な商用マイクロプロセッサとしては，周波数と電圧を動的に変化させる Speedstep$^{(TM)}$☆3技術を採用した Intel Corp. のモバイル Pentium$^{(TM)}$III☆4，および，PowerNow!$^{(TM)}$☆5技術を採用した AMD，Inc. の AMD-K6$^{(TM)}$-2+☆6，同様の LongRun$^{(TM)}$☆7機能に加えて VLIW の採用によりゲート数および電力を抑えた Transmeta Corp. の Crusoe$^{(TM)}$☆8があげられる[11)]．

最近ではアーキテクチャレベルでの低電力化に関する研究が活発である．電力消費モデルを付加したサイクルシミュレータにより，SPEC95 などのベンチマークプログラムの消費電力をモデル化した研究[12),13)]，コンパイラが消費電力を考慮した最適化を行うことにより，レジスタファイルなどの消費電力を引き下げる方法[14),15)]，Direct Rambus DRAM（RDRAM）の電力モードと OS の実ページ割付けとの連携による省電力化手法[16)]，パイプライン・ゲーティングと呼ぶ方法により，性能を低下させずに投機的実行による電力消費を抑制する手法[17)]などが報告されている．Java 関連では，ポケットコンピュータにおいて Java 仮想マシンを動作させた場合の消費電力に関する報告がある[18)]．

さて，N 型と P 型トランジスタを対称に配置する CMOS 回路は，スイッチング時にのみ電荷が移動するため，本来消費電力が小さい性質がある．しかし，最近では，プリチャージが必要なダイナミック回路を多用したり，周波数を極限まで高めることにより，特にクロック分配に関わる消費電力がきわめて多くを占めるようになってきている．プロセッサを構成する機能ブロックのうち，未使用ブロックへの電源供給そのものを遮断することにより，電力を削減する方法が考えられる．しかし，最近の CMOS 回路は全体が巨大なコンデンサであるため，電源供給を再開してからそのブロックの電圧が安定するまでには，かなりのサイクル数を要する．また，機能ブロックへの突入電流が周辺の信号線にインダクティブノイズを引き起こす恐れがあることから，演算サイクル程度の時間間隔で機能ブロックの電力をこまめに節約する方法としては不向きである．このような状況では，クロックを停止することのできるゲート付きクロック（Gated Clock）を導入することにより，機能ブロック単位の電力の大幅な削減が期待できる[19)~21)]．

☆1 Java は SUN Microsystems，Inc の登録商標．
☆2 JeRTy はオムロン株式会社の登録商標．
☆3,4 Speedstep，Pentium は Intel Corp. の登録商標．
☆5,6 PowerNow!，AMD-K6 は AMD，Inc. の登録商標．
☆7,8 LongRun，Crusoe は Transmeta Corp. の登録商標．

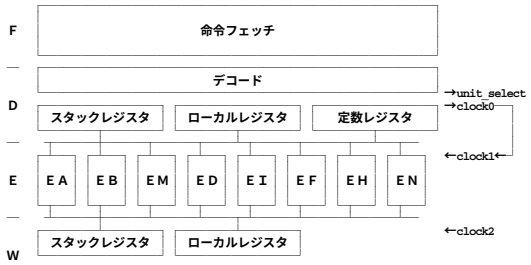


図 1 演算ユニットモデル
Fig. 1 Model of functional units.

3. 演算ユニットと所要サイクル数の仮定

本稿において用いる，Fetch，Decode，Execute，Write の 4 段パイプラインからなる Java プロセッサの簡略モデルを**図 1** に示す．

スタック上に配置される変数およびローカル変数のために，それぞれスタックレジスタとローカルレジスタを備え，命令中に含まれる定数を保持する定数レジスタとともに，D ステージにおける演算ユニットへの入力部を構成している．E ステージに対応する演算ユニットは 8 つの部分に分かれており，入力部の信号変化がユニット内部に伝搬しないよう，後述する clock1 によって開閉するラッチが設けられている．clock1 は，E ステージ各ユニットの上流クロック clock0 から生成され，さらに clock0 は，停止しないクロックおよび命令のデコード結果から得られる unit_select 信号の組合せにより生成されるとする．

なお，実行対象が Java バイトコードであること以外の特殊な状況が生じないよう，命令語の種類に依存する演算ユニットの構成については，現実の Java プロセッサ JeRTy [26] に基づいてモデル化を行う一方，きわめて高い周波数で動作するプロセッサコアおよび相対的に低速な主記憶による典型的な構成を想定し，レジスタ，TLB，キャッシュ，主記憶という記憶階層に基づく一般的なモデル化を行った．

具体的には，評価に用いる SPECJVM98 [25] に合わせて過剰仕様とならないよう，文献 22) に示されているように，スタックレジスタおよびローカルレジスタは各 8 本，また，一般的な RISC プロセッサの TLB やデータキャッシュタグに相当する，ヒープ参照を高速化するためのオブジェクト変換表（4 ウェイ，4096 エントリ，ミスペナルティ=20 サイクル），および，ヒープキャッシュタグ機構（ダイレクトマップ，1024 エントリ，ラインサイズ=64 バイト，ミスペナルティ=20 サイクル）を仮定した．

さて，**表 1** に，各演算ユニットと各命令語との対応，および，所要サイクル数の仮定を示す．EA は基本的に 1 サイクルで済む単純なレジスタ間演算（符号ビットを反転する浮動小数点演算も含む）を担当する．EB は条件判定および分岐先アドレス計算に 2 サイクルを要する分岐命令（多方向分岐は 8 サイクルと仮定）を担当する．EM は 8 ビット単位に演算する整数乗算に合計 4 サイクル，2 ビット単位に求める整数除算に合計 16 サイクルを要すると仮定する．ED はスタックトップ以外に書き込みを行う複雑なスタック操作を担当する．EI はメソッド呼び出しおよびリターンを担当し，前述した各 8 本のスタックレジスタおよびローカルレジスタ，また，内部レジスタの更新に，合計 18 サイクルを要すると仮定する．EF は浮動小数点演算，および，invoke に関連するフレーム生成などを担当し，加減乗算および型変換に 4 サイクル，2 ビット単位に求める単精度除算に 16 サイクル，2 ビット単位に求める倍精度除算に 32 サイクルを要すると仮定する．

EH はヒープ参照命令を担当し，命令オペランドやスタック上のパラメータから，オブジェクト変換表を用いてデータ位置を特定し，ヒープキャッシュを参照するのに 10 サイクルを要すると仮定する．オブジェ

表 1 所要サイクル数の仮定
Table 1 Model of execution cycles.

演算ユニット名	EA	EB	EM	ED	EI	EF	EH	EN
ゲート数	8k	8k	18k	1k	8k	21k	11k	8k
ラッチ数	51	51	196	7	119	333	87	93
消費電力比率 (%)	10	10	22	2	10	24	12	10
nop,popX,dupX	1	0	0	0	0	0	0	0
Xconst,Xipush,Xstore	1	0	0	0	0	0	0	0
IALU,SFT,I2I,Fneg	1	0	0	0	0	0	0	0
Xload	2	0	0	0	0	0	0	0
if,Icmp,goto,jsr,ret	0	2	0	0	0	0	0	0
Xswitch	0	8	0	0	0	0	0	0
Imul	0	0	4	0	0	0	0	0
Idiv,Irem	0	0	16	0	0	0	0	0
swap,dup_X	0	0	0	10	0	0	0	0
dup2_X	0	0	0	20	0	0	0	0
invokeX,Xreturn	0	0	0	0	18	18	0	0
Xaload,Xastore	0	0	0	0	0	0	10	0
ldcX_X,astore	0	0	0	0	0	0	10	0
Xstatic,field	0	0	0	0	0	0	10	0
arraylength	0	0	0	0	0	0	10	0
new,Xnewarray	0	0	0	0	0	0	0	20
Fadd,Fsub,Fmul,Fcmp	0	0	0	0	0	4	0	0
I2F,F2I,F2F	0	0	0	0	0	4	0	0
Fdiv,Frem	0	0	0	0	0	16	0	0
Ddiv,Drem	0	0	0	0	0	32	0	0
invokeinterface	0	0	0	0	0	18	0	0
athrow	0	0	0	0	0	18	0	0
checkcast,instanceof	0	0	0	0	0	10	0	0
multianewarray	0	0	0	0	0	10	0	0
monitorX	0	0	0	0	0	9	0	0

クト変換表ミスやヒープキャッシュミスが発生した場合，前述したペナルティが加算される．ただし，この間，演算ユニットにおける消費電力は 0 であるとする．最後に，EN は new などオブジェクト生成を担当し，一律 20 サイクルを要すると仮定する．なお，X，I，ALU，SFT，F，D は，バイトコードのニモニックについて，それぞれ任意の英数字，整数，整数加減/論理，シフト，単精度，倍精度浮動小数点数を代表して表記したものである．

さらに，表 1 に，JeRTy における各演算ユニットのゲート数（第 2 行）およびラッチ数（第 3 行）を示す．ゲート数とラッチ数は，ほぼ比例関係にある．演算ユニット全体に対して占める割合をそのまま 1 サイクルあたりの各演算ユニットの消費電力ととらえた場合の消費電力比率を第 4 行に示す．本稿では簡単のために，プロセッサ部分のうち，特に E ステージに関わる演算ユニットのみを消費電力の評価対象とする．

4. クロック制御方法に関する仮定

パイプラインの高速化のために，従来一般的であったマスタスレーブ型フリップフロップに代えて，最近ではトランスペアレントラッチ[23]が採用されてきている．ただし，ラッチが開いている時間よりも短い時間で信号が伝播する最短パスが存在する場合，前段ステージの信号変化がラッチをつき抜けて，次々段ステージに到達してしまう．このため，ラッチを開く時間，すなわち，クロックパルスの幅を短く，かつ，正確に決定したうえで，このような最短パスが存在しないよう設計する必要がある．各演算ユニットにおけるクロックパルス幅を正確に制御するためには，文献 24) に示されているように，上流クロックの立ち下がりエッジを基準にして，改めて，一定期間だけ low となるようなローカルクロックを生成する方法が一般的である．

図 2 に，unit_select 信号から各演算ユニットの末端ラッチに至るクロック分配モデルを示す．(c) は各演算ユニットにおける末端ラッチである．clock1 が low である期間は入力 IN を出力 OUT に伝播させ，clock1 が high である期間はインバータ B および弱いインバータ A により OUT を保持する．

末端ラッチ（c）の近くに置かれる（b）は，clock0 と 5 段インバータ通過後の $\overline{clock0'}$ との論理和により，clock1 が low である期間を保証するとともに，ゲートサイズを徐々に大きくして clock1 の駆動力を上げている．途中，波形なまりが蓄積しないよう，1 つのインバータが直接駆動するインバータを最大 2 個としている．また，1 つの（b）あたり 48 個程度の末端ラッチを駆動できるよう，最終段は 16 倍インバータとしている．このため（b）の入力容量はインバータ 17 個分となる．

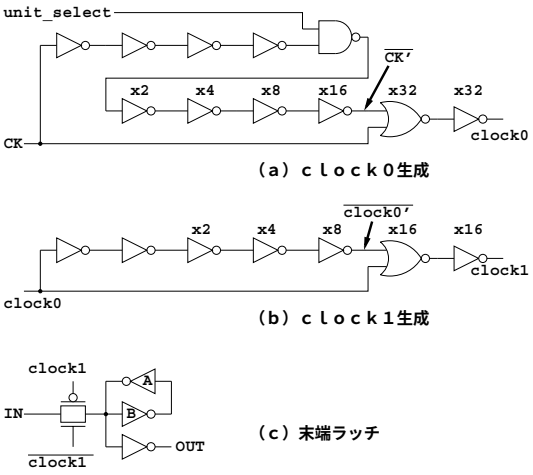


図 2 クロック分配モデル
Fig. 2 Model of clock distributors.

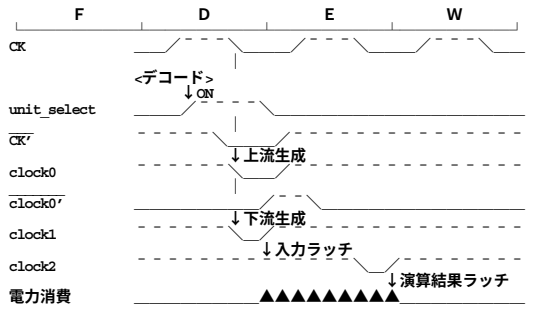


図 3 起動ペナルティ=0 の場合
Fig. 3 Case of penalty=0.

（a）は演算ユニットごとに 1 つ置かれる．clock1 の生成方法から明らかなように，clock0 が low である期間は clock1 が low である期間よりも長い必要がある．このために，CK から $\overline{CK'}$ に至る段数は，(b) におけるインバータ段数よりも多い 9 段としている．EF ユニットにおける 333 個のラッチを 7 組の（b）により駆動し，7 組の（b）を 1 つの（a）が駆動できるよう，unit_select 信号から clock0 に至る経路においてゲートサイズを徐々に大きくし，32 倍インバータまで駆動力を上げている．

以上のクロック分配モデルと前章に述べた演算ユニットの関係を**図 3** に示す．clock1 の立ち上がりエッジすなわち D ステージと E ステージの境界において演算ユニットの入力が確定する．clock1 が正しく生成されるために clock0 の立ち下がりが正確でなければならないことから，図 2（a）の NOR ゲートにおい

て，CK の立ち下がりに先立ち $\overline{CK'}$ が low でなければならない．同様に，unit_select 信号はさらに手前の NAND ゲートにおいて high を保つ必要がある．(a) および (b) のゲート段数を合計すると，unit_select 信号は D ステージ終了の 14 段手前において確定していなければならないことが分かる．

さて，命令デコードとレジスタ読み出しからなる D ステージでは，Java バイトコードのデコードの複雑さに比べて，スタックレジスタとローカルレジスタが各 8 本と小さいことから，D ステージの多くは命令デコードに費やされると考えられる．レジスタ読み出しに要する遅延時間よりも unit_select 信号が clock1 に伝わる 14 段分の遅延時間が大きいと考えられるため，デコード結果に基づき unit_select 信号を確定させる場合には，D ステージと E ステージの間に新たなステージを設けるか，または，周波数を下げる必要がある．ただし，前者の場合，合わせて最大 30%弱に達する分岐，invoke，return 命令[22]の実行が 1 サイクル遅れる．また，バッテリ駆動から外部電源駆動に切り替えて，クロックを常時供給し，最高速度で動作させることを考えた場合，両者とも無意味なオーバヘッドとなり，本来の性能が発揮できない．

仮に周波数を下げた場合，図 3 に示すように，D ステージにおける unit_select 信号の確定が次の E ステージの入口における clock1 の確定に間に合う．この場合を起動ペナルティ=0 と呼ぶことにする．演算ユニットが複数サイクルを要する場合には，clock1 が複数回 low となる．clock2 は，特定のレジスタに対する書き込み信号であり，clock0 および clock1 とは独立して動作する．命令デコードの完了を待ってから，E ステージにおいて必要な各演算ユニットを起動できることから，性能を低下させることなく，E ステージにおける消費電力を最小とすることが可能である．

一方，周波数を下げない場合，**図 4** (c) に示すように，D ステージにおいて unit_select 信号が確定する前に，F ステージにおいて unit_select 信号を投機的に high とする必要がある．この場合を起動ペナルティ=1 と呼ぶことにする．(a) は予測どおり未使用であった場合であり，対応する演算ユニットの消費電力は 0 となる．(b) は unit_select 信号を low としたものの，予測に反して演算ユニットが必要であった場合である．D ステージの終わりに unit_select 信号が high となることから，1 サイクルのバブルが発生する．(c) は予測どおり使用した場合であり，バブルは発生しない．(d) は予測に反して演算ユニットが不要であった場合であり，バブルは発生しないものの，起動ペナルティ

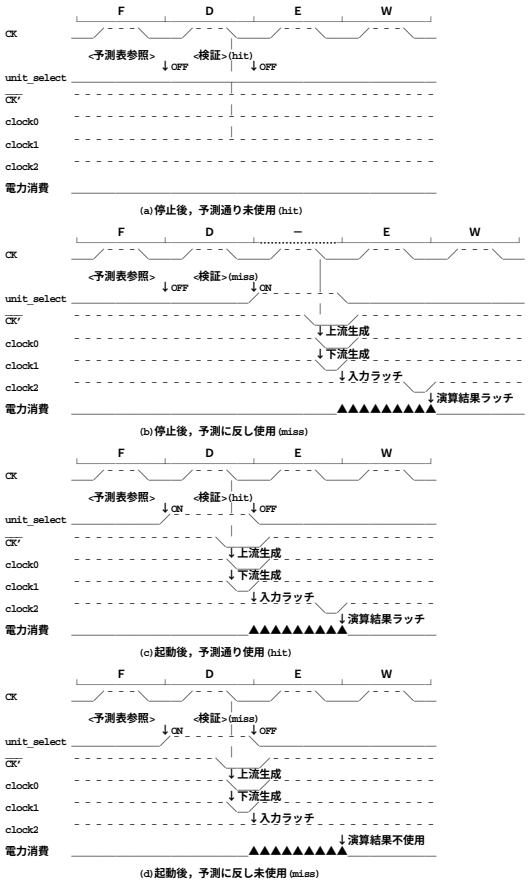


図 4 起動ペナルティ=1 の場合
Fig. 4 Case of penalty=1.

に相当する 1 サイクル分の電力は無駄に消費される．演算ユニットが複数サイクルを要する場合，2 サイクル目以降については正確なデコード結果に基づいてクロックを停止することができるため，無駄な電力消費は最初の 1 サイクル分にとどまる．(b) および (d) の場合，D ステージにおける検証時に，後述する予測表の更新を行うとする．

性能が低下する (b) および電力が増加する (d) の場合が発生しないよう完全な予測を行うことにより，電力削減効果と性能は，起動ペナルティ=0 の場合に等しくなる．しかし，完全な予測のための大規模なハードウェア機構を設けることにより，かえって電力が増加することは本末転倒である．また，(b) の発生を抑えるためにより多くの演算ユニットを起動しておくことは (d) の増加となる．逆に，(d) の発生を抑えるためになるべく演算ユニットを起動しない方針とすると，(b) が増加する．

以下では，次に実行する演算を予測し，必要最小限

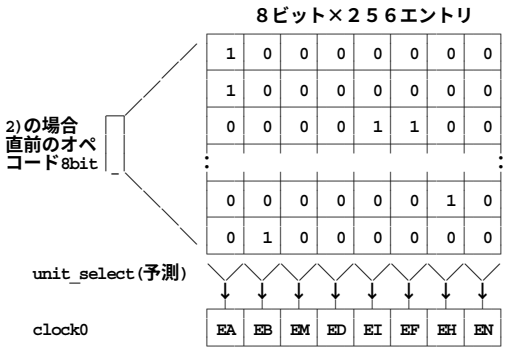


図 5 演算ユニットの使用予測機構
Fig. 5 Prediction model of unit usage.

の演算ユニットのみを起動することとし，いかに小さなハードウェアおよび予測機構により,(b) および (d) を削減することができるかについて，議論を進める．

5. 演算ユニット使用予測方法の提案

起動ペナルティ=1である場合，すなわち，F ステージ完了時に演算ユニットの使用を予測するための情報としては，1）F ステージ開始時におけるプログラムカウンタ（PC）の下位数ビット；2）直前にデコードした命令のオペコード；3）命令に関連するデータ型（int，long，float，double，reference）により直前命令のオペコードをグループ化したもの；4）使用する演算ユニットによりグループ化したもの；の 4 つが考えられる．1）は比較的小さなメソッドに対して有効であるものの，各メソッドの先頭は 0 番地であることから，あまりに小さくループ構造もないようなメソッドが多数実行される場合には，メソッドを識別する情報を加える必要があると考えられる．2）は連続する命令のオペコードに強い相関がある場合に有効である．Java 仮想マシンはスタックマシンであることから，一連のスタック操作を繰り返すバイトコードは，一般の RISC 命令よりもオペコードの相関が強いと考えられる．3）は命令間において授受されるデータ型の連続性を利用して，2）よりもハードウェアを削減する試みである．4）は使用する演算ユニットに着目して，同様にハードウェアを削減する試みである．予測機構の概略を**図 5** に示す．予測表の各エントリは，表 1 の各行と同様に，8 個の演算ユニットそれぞれを起動（1）または停止（0）するための 8 ビットの値を保持している．F ステージにおいて，前述の情報をインデックスとして予測表を参照する．そして，D ステージにおける命令デコード結果に従い，演算ユニットの実際の使用状況を予測表に反映する．

さらに，前章におけるクロック分配に対しては極端な仮定であるものの，傾向を見るために，起動ペナルティ=2 である場合についても，同様の方法により予測を試みる．ただし，1）は 1 サイクル前の PC となるため，条件分岐による撹乱が生じる．2）は 2 命令前のオペコードとなるため，相関が弱くなるはずである．3）および 4）についても，同様に相関が弱くなると推測できる．

6. 起動ペナルティ=0 の場合の理想的電力

演算ユニットの使用率および消費電力のシミュレーションには，表 1 に示した各演算ユニットの所要サイクル数と消費電力比率，および，Java 仮想マシンである Kaffe1.0.6 [27]を用いて，SPECJVM98 の各ベンチマークプログラムを実行サイズ s1，s10，s100 により走行して得た命令トレースを用いた．**表 2** に，起動ペナルティ=0 の場合の各演算ユニット使用率（上段），使用率に表 1 の消費電力比率を乗じて得られる電力比（下段）を示す.「合計」の上段は，所要サイクル数のうち，少なくとも 1 つの演算ユニットが動作した割合を表している．100%との差分，すなわち，演算ユニットが 1 つも動作していないサイクルは，前述したオブジェクト変換表ミスまたはヒープキャッシュミスが発生している期間に対応する．

「合計」の下段は，電力比の合計，すなわち，クロックを常時投入した場合の演算ユニット全体の消費電力に対する，本方式における消費電力の比率を表しており，起動ペナルティ=1，2 の場合において予測がすべて的中した場合の消費電力の下限値に相当する．未使用の演算ユニットおよびクロック分配器における消費電力を抑えることにより，約 80%から 90%の電力を削減できることが分かる．

●は EI および EF ユニットにおいて使用率が 40%を超えること，また，○は 15%未満であることを示す．電力比の合計が 20%を超えるものは，いずれも●を，また，電力比の合計が 10%程度のものは，いずれも○を含んでおり，invoke 命令が多数出現する場合に消費電力が 20%程度に倍増することを示している．これは，表 1 に示した EI および EF ユニットの単位時間あたりの消費電力が 10+24=34%を占め，かつ，他の演算ユニットに比べて 1 命令あたりの所要サイクル数が多いためと考えられる．

7. 起動ペナルティ > 0 の場合の予測の効果

我々の目標は，起動ペナルティ > 0 の場合について，前述した電力下限値に近く，かつ，起動ペナルティに

表 2 演算ユニット使用率（上段%）および電力比（下段%）
Table 2 Activity ratio (upper%) and power consumption (lower%).

	EA	EB	EM	ED	EI	EF	EH	EN	「合計」
compress									
s1	16.3	2.2	0.0	2.6	○ 11.4	○ 11.4	46.7	0.0	**79.2%**
	1.6	0.2	0.0	0.1	1.1	2.7	5.6	0.0	**11.4%**
s10	17.0	2.5	0.0	2.5	○ 10.2	○ 10.2	46.4	0.0	**78.6%**
	1.7	0.2	0.0	0.1	1.0	2.5	5.6	0.0	**11.1%**
s100	15.7	2.2	0.0	2.5	○ 10.9	○ 10.9	44.9	0.0	**76.1%**
	1.6	0.2	0.0	0.1	1.1	2.6	5.4	0.0	**10.9%**
jess									
s1	12.6	3.5	0.3	0.7	35.3	37.2	35.3	1.5	**91.1%**
	1.3	0.4	0.1	0.0	3.5	8.9	4.2	0.1	**18.5%**
s10	14.4	4.9	0.1	0.1	29.5	30.9	40.0	0.3	**90.7%**
	1.4	0.5	0.0	0.0	2.9	7.4	4.8	0.0	**17.2%**
s100	12.4	3.6	0.4	0.2	36.0	38.4	33.6	1.2	**89.7%**
	1.2	0.4	0.1	0.0	3.6	9.2	4.0	0.1	**18.6%**
db									
s1	15.0	4.3	0.4	1.4	35.4	36.2	36.4	1.2	**94.9%**
	1.5	0.4	0.1	0.0	3.5	8.7	4.4	0.1	**18.8%**
s10	16.5	3.3	0.0	0.2	17.2	19.3	38.3	0.6	**78.2%**
	1.6	0.3	0.0	0.0	1.7	4.6	4.6	0.1	**13.0%**
s100	14.2	2.7	0.0	0.3	16.1	19.4	38.2	0.2	**75.1%**
	1.4	0.3	0.0	0.0	1.6	4.7	4.6	0.0	**12.6%**
javac									
s1	15.9	3.6	0.7	2.1	32.0	33.9	37.5	1.0	**94.7%**
	1.6	0.4	0.2	0.0	3.2	8.1	4.5	0.1	**18.1%**
s10	14.0	4.9	0.8	1.4	32.0	33.8	35.7	0.9	**91.5%**
	1.4	0.5	0.2	0.0	3.2	8.1	4.3	0.1	**17.8%**
s100	12.8	5.1	0.6	1.4	31.8	34.0	35.3	0.8	**89.9%**
	1.3	0.5	0.1	0.0	3.2	8.2	4.2	0.1	**17.6%**
mpegaudio									
s1	16.6	1.4	0.1	0.3	○ 6.4	○ 13.1	44.2	0.0	**75.7%**
	1.7	0.1	0.0	0.0	0.6	3.1	5.3	0.0	**10.9%**
s10	16.5	1.2	0.0	0.3	○ 5.0	○ 12.0	44.7	0.0	**74.9%**
	1.7	0.1	0.0	0.0	0.5	2.9	5.4	0.0	**10.5%**
s100	16.7	1.3	0.1	0.3	○ 6.1	○ 12.8	44.2	0.0	**75.5%**
	1.7	0.1	0.0	0.0	0.6	3.1	5.3	0.0	**10.8%**
mtrt									
s1	11.7	1.9	0.0	1.3	● 44.3	● 45.7	29.8	1.2	**91.7%**
	1.2	0.2	0.0	0.0	4.4	11.0	3.6	0.1	**20.5%**
s10	9.9	1.4	0.0	0.7	● 46.4	● 48.8	26.4	0.9	**88.1%**
	1.0	0.1	0.0	0.0	4.6	11.7	3.2	0.1	**20.7%**
s100	8.0	0.8	0.0	0.1	● 49.7	● 53.0	21.6	0.6	**84.2%**
	0.8	0.1	0.0	0.0	5.0	12.7	2.6	0.1	**21.2%**
jack									
s1	10.8	2.8	0.4	1.7	35.0	38.2	35.9	2.9	**92.7%**
	1.1	0.3	0.1	0.0	3.5	9.2	4.3	0.3	**18.8%**
s10	10.8	2.8	0.4	1.7	35.1	38.2	35.9	2.9	**92.7%**
	1.1	0.3	0.1	0.0	3.5	9.2	4.3	0.3	**18.8%**
s100	10.8	2.8	0.4	1.7	35.2	38.4	36.0	3.0	**92.9%**
	1.1	0.3	0.1	0.0	3.5	9.2	4.3	0.3	**18.8%**

よる性能低下が小さくなるような，極力小さい演算ユニット使用予測機構を提案することである．本章では，5 章において述べた予測方法を次のように具体化して，消費電力および所要サイクル数の測定を行った．

NP：予測なし 予測を行わず，必要になるまでクロックを止める方法．消費電力の下限（最良）値および所要サイクル数の上限（最悪）値が得られる．

PC08：PC8 ビット 「起動ペナルティ–1」だけ手前の PC の下位 8 ビットを用いる．

PC26：メソッド識別子 2 ビット + PC6 ビット 「起動ペナルティ–1」だけ手前の，メソッド識別子（メソッドに固有の値）下位 2 ビットおよび PC の下位 6 ビットからなる合計 8 ビットを用いる．

PC44：メソッド識別子 4 ビット + PC4 ビット 「起動ペナルティ–1」だけ手前の，メソッド識別子下位 4 ビットおよび PC の下位 4 ビットからなる合計 8 ビットを用いる．

OP8：オペコード 8 ビット 「起動ペナルティ」だけ手前の命令のオペコード 8 ビットを用いる．ただし WIDE 命令は引き続くオペコードを使用する．

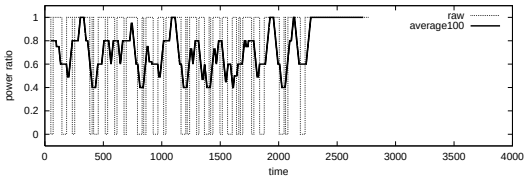
PC3：PC3 ビット 「起動ペナルティ–1」だけ手前の PC の下位 3 ビットを用いる．

TY3：データ型 3 ビット スタック上に出力するデータ型（前述の 5 種類）により分類し，3 ビットにより識別する．出力がない場合は入力データ型を用いる．

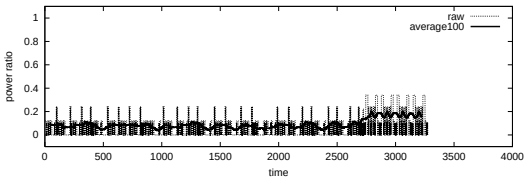
UN3：演算ユニット 3 ビット 表 1 に示したように，使用する演算ユニットに応じてオペコードを 8 つのグループに分類し，3 ビットにより識別する．

まず，演算ユニット全体における電力消費の様子を**図 6** および**図 7** に示す．それぞれ，表 2 において最も電力比の小さい mpeg および電力比の大きい mtrt を実行サイズ s10 により走行させ，途中の同じ 500 命令分の消費電力をサイクルごとに抽出したものである．なお太線は，前後合わせて 100 サイクル分の移動平均値である．

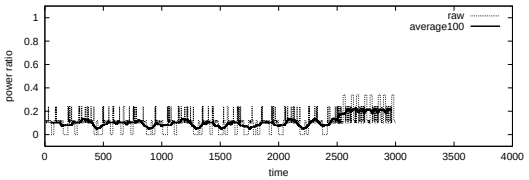
浮動小数点演算が多い mpeg では，演算ユニットの使用率を示す表 2 の「合計」上段の値が 100%に大きく届かないことから分かるように，オブジェクト変換表ミスまたはヒープキャッシュミスが多発する．このため，低電力化対策を施さない（a）において消費電力が激しく変化することが分かる．ただし，横軸は 2800 サイクルまでとなっている．一方,（b）NP の場合，消費電力を抑えることができるものの，横軸は 3300 サイクルまで増加しており，消費電力と引きかえに性能が低下していることが分かる．これに対し,（c）PC08 の場合，消費電力が若干増加するものの，横軸が 3000 サイクルに短縮されている．ところで，PC08 は，必要になるまですべての演算ユニットを止める NP よりも消費電力の変動が緩やかであることが分かる．これは，冒頭において述べたように，電源ノイズ対策の観



（a）クロックを常時投入



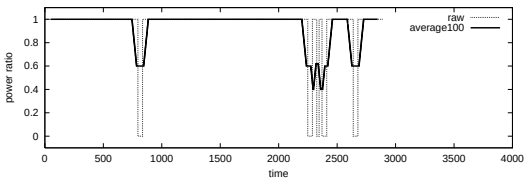
（b）起動ペナルティ=1，予測なし（NP）の場合



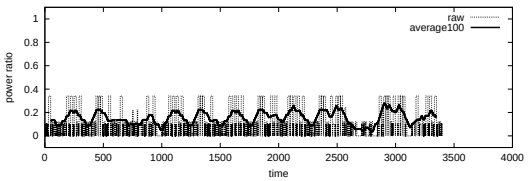
（c）起動ペナルティ=1，PC 下位 8 ビットによる予測（PC08）

図 6 mpeg の電力消費

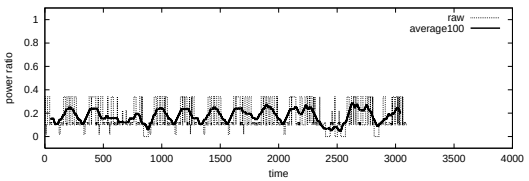
Fig. 6 Power consumption of mpeg.



（a）クロックを常時投入



（b）起動ペナルティ=1，予測なし（NP）の場合



（c）起動ペナルティ=1，PC 下位 8 ビットによる予測（PC08）

図 7 mtrt の電力消費

Fig. 7 Power consumption of mtrt.

点から重要な特長であるといえる．

一方，メソッド呼び出しが多い mtrt では，オブジェクト変換表ミスやヒープキャッシュミスが少なく，(a) における変化は少ないものの，消費電力はむしろ mpeg よりも多い．また，mpeg と同様，(c) PC08 の場合には，(b) NP よりも消費電力が若干増加するものの，性能低下や消費電力の変動が抑えられていることが分かる．

次に，各予測方法について，**図 8** に起動ペナルティ=1 の場合，**図 9** に起動ペナルティ=2 の場合の測定結果を示す．なお，プログラムサイズ s1，s10，s100 の

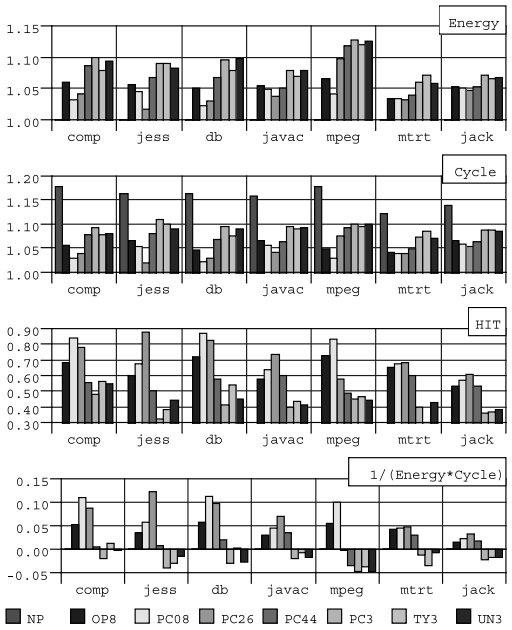


図 8 起動ペナルティ=1 の場合

Fig. 8 Case of penalty=1.

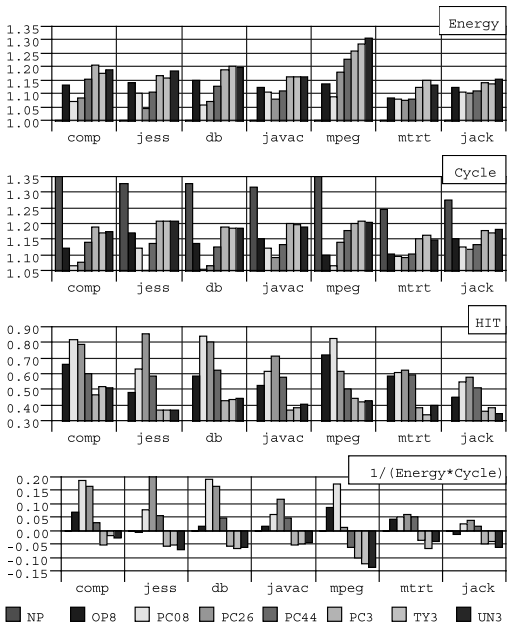


図 9 起動ペナルティ=2 の場合

Fig. 9 Case of penalty=2.

すべてについて測定した結果，s10が全体を最もよく代表していると判断した．見やすさのため，s10の測定結果のみをグラフ表示している．グラフ「Energy」は，各プログラムを完走させるために必要であった総電力である．起動ペナルティ=0の場合の総電力を1としたときの，各予測方法における総電力を表している．グラフ「Cycle」は，同様に，各プログラムを完走させるために必要であった総サイクル数である．起動ペナルティ=0の場合の総サイクル数を1としたときの，各予測方法における総サイクル数を表している．グラフ「HIT」は，図5に示した予測表のヒット率である．

さて，低消費電力の評価尺度には，命令実行速度をどの程度重要視するかによって，$Mips/Watt$，$Mips^2/Watt$，$Mips^3/Watt$が考えられる[19]．このうち$Mips/Watt$は，

$$\frac{\text{総実行命令数/実行時間}}{\text{総電力/実行時間}}. \quad (1)$$

すなわち，

$$\frac{\text{総実行命令数}}{\text{総電力}}. \quad (2)$$

と表現することができ，各プログラムごとに総実行命令数は一定であることから，

$$1/\text{総電力}. \quad (3)$$

による比較と考えることができる．総電力（Energy）=単位時間あたり電力（Power）∗実行時間（Delay）であることから，$Mips/Watt$はPower-Delay積の逆数といえる．

同様に$Mips^2/Watt$は，

$$\frac{(\text{総実行命令数/実行時間})^2}{\text{総電力/実行時間}}. \quad (4)$$

すなわち，

$$\frac{\text{総実行命令数}^2}{\text{総電力} * \text{実行時間}}. \quad (5)$$

であり，同様に総実行命令数は一定であるため，

$$1/(\text{総電力} * \text{実行時間}). \quad (6)$$

つまり，Energy-Delay積の逆数を比較しているといえる．

ところで，前者の$Mips/Watt$は，総電力のみを評価尺度としており，命令実行速度がまったく反映されない．本稿では，命令実行速度も考慮に入れるために，後者の$Mips^2/Watt$を評価尺度として用いることにする．なお，同一アーキテクチャにおけるペナルティサイクルの比較であることから，実行時間を総サイクル数（Cycle）に置き換える．以上の考えに基づき，図8および図9の最後に，各方法における1/（Energy∗Cycle）の値と，NPにおける同様の値との比を求め，1を減じたものを表示した．すなわち，NPを基準とし，＋は$Mips^2/Watt$が優れていること，－は劣っていることを意味する．

さて，クロックの投機的投入を行わないNPでは，電力比は1，サイクル数は，起動ペナルティ=1の場合10%から20%，起動ペナルティ=2の場合25%から35%の増加となっており，それぞれ，各予測方法における電力比の下限およびサイクル数の上限を示している．NPでは予測表のヒット率を0と見なしており，グラフ上には現れない．

8ビットの情報を用いる方法の中では，起動ペナルティ=1，2のいずれの場合にも，comp，db，mpegにおいてはPC08，jess，javac，mtrt，jackにおいてはPC26が最もヒット率が高く，ほぼ60%から90%となっている．起動ペナルティ=1の場合，消費電力およびサイクル数ともに5%程度の増加，また，起動ペナルティ=2の場合，ともに10%程度の増加にとどまっている．NPに対するサイクル数の比では，起動ペナルティ=1の場合約10%，起動ペナルティ=2の場合約20%の減少となっており，予測による性能向上が電力増加を上回っている．これは，グラフ「1/（Energy∗Cycle）」に示すように，PC08やPC26における$Mips^2/Watt$の値が，起動ペナルティ=1の場合は多いもので約10%，起動ペナルティ=2の場合は多いもので約20%改善されていることからも確認できる．

一方，3ビットの情報では，全般的にヒット率が50%未満と低くなる．起動ペナルティ=1の場合，TY3が最も消費電力およびサイクル数が少ないものの，ともに10%程度増加している．また，起動ペナルティ=2の場合，PC3とTY3とが同程度であり，消費電力では最大30%，また，サイクル数では20%程度増加していることが分かる．NPに対するサイクル数の比では，起動ペナルティ=1の場合約5%，起動ペナルティ=2の場合約10%の減少にとどまっており，予測による性能向上よりも電力増加のほうが大きい．グラフ「1/（Energy∗Cycle）」でも，TY3における$Mips^2/Watt$の値の大部分が負の値となっていることが分かる．

8. 考　　察

測定の結果，起動ペナルティ分だけ先の命令が使用する演算ユニットを予測するためには，8ビットの情報では，メソッド識別子の下位2ビットおよびPCの下位6ビット，また3ビットの情報では，データ型3ビットが，それぞれ最も有効であることが明らかになった．ただし，予測をまったく行わない方法に対す

る $Mips^2/Watt$ の改善という点では，3 ビットの情報は不十分であることが分かった．

また，以上に示した測定結果とは別に，予測表のエントリ数を 2^{16} とし，PC の下位 16 ビットを用いて同様に測定したところ，PC08 よりも若干ヒット率が向上したものの，ほぼ同様の効果しか得られないことが分かっている．多くのビット数を用いても効果が上がらないのは，各メソッドの先頭 PC がつねに 0 であること，また，SPECJVM98 の各ベンチマークプログラムでは，メソッドあたりの命令数が比較的小さく，PC の下位 8 ビットを除く上位ビットが 1 になる可能性がきわめて小さいためと考えている．実際，PC08 ではヒット率が 60%に届かない jack について，予測表のエントリ数を 2^{16} とし，PC08 にメソッド識別子の下位 8 ビットを加えた合計 16 ビットをインデックスとして用いたところ，ヒット率が 70%を超えることを確認した．各プログラムにおいて動作するメソッドの種類は，たかだか 300 程度であることから，エントリ数さえ増加させれば，すべてのプログラムについてヒット率を格段に上げられると推測できる．ただし，予測表はすべての命令が参照することから，予測表自身の消費電力が問題となる．最適な予測表のサイズを決定するには，より詳細な電力消費モデルの構築が必要であり，今後の課題である．

ところで，表 2 では最も小さかった mpeg の電力比が，図 8 や図 9 では他のプログラムに比べて際立って大きい．表 2 から，mpeg における EI の使用率が他に比べてきわめて低いこと，また，EI と EF の使用率の差分が 6%程度と，ほかに比べてきわめて大きいことが読みとれる．表 1 から分かるように，EI と EF の使用率差分とは，主に浮動小数点演算が占めるサイクル数の比率である．浮動小数点演算が多いために，予測表のヒット率の若干の低下が，浮動小数点演算ユニットにおける消費電力を大幅に増加していると考えられる．

9. おわりに

本稿では，単純なパイプライン構造によりモデル化した Java プロセッサに，JeRTy における演算ユニットの電力消費モデルをあてはめ，演算ユニットごとにクロックを投入および停止して消費電力を抑える方法について評価を行った．unit_select 信号を決定してから演算ユニットが使用可能となるまでに 1 または 2 サイクルを要する場合でも，クロック分配器を投機的に制御することにより，性能低下を抑えつつ電力を削減できることを示した．特に，予測に用いる情報としては，メソッド識別子の下位 2 ビットおよび PC の下位 6 ビット，または，データ型を表す 3 ビットが，同じビット数を用いる方法の中ではそれぞれ最も高い効果が得られること，また，8 ビットの情報を用いる場合，予測による性能向上が消費電力増加を上回るものの，3 ビットでは逆転することを明らかにした．

謝辞 本研究の一部は文部省科学研究費補助金（基盤研究（B）（2）課題番号 12480072，12558027 ならびに 13480083）による．

参考文献

1) Benini, L., et al.: System-level Dynamic Power Management, *Proc. 1999 IEEE Alessandro Volta Memorial Workshop on Low-Power* (*VOLTA'99*) (1999).
2) Usami, K. and Horowitz, M.: Clustered voltage scaling technique for low-power design, *Proc. Int'l Symp. on Low Power Electronics and Design* (*ISLPED'95*), pp.3–8 (1995).
3) Athas, W.: Low-Power VLSI Techniques for Applications in Embedded Computing, *Proc. VOLTA'99* (1999).
4) Burd, T.D. and Brodersen, R.W.: Design Issues for Dynamic Voltage Scaling, *Proc. ISLPED'00*, pp.9–14 (2000).
5) Koegst, M., et al.: Low Power Design of FSMs by State Assignment and Disabling Self-Loops, *Proc. 23rd Euromicro Conf. on New Frontiers of Information Technology* (1997).
6) Raghavan, N., et al.: Automatic Insertion of Gated Clocks at Register Transfer Level, *Proc. 12th Int'l Conf. on VLSI Design* (1998).
7) Schuster, S., et al.: Asynchronous Interlocked Pipelined CMOS Circuits Operating at 3.3-4.5 GHz, *Proc. IEEE Int'l SolidState Circuits Conf.* (*ISSCC'00*), pp.292–293 (2000).
8) Strollo, A.G.M., et al.: New Clock-Gating Techniques for Low-Power Flip-flops, *Proc. ISLPED'00*, pp.114–119 (2000).
9) Vinereanu, T. and Lidholm, S.: An Improved Pass Transistor Synthesis Method for Low Power, High Speed CMOS Circuits, *Proc. ISLPED'00*, pp.120–124 (2000).
10) Joshi, R.V., et al.: Low Power 900 MHz Register File (8 Ports, 32 Words x 64 Bits) in 1.8 V, 0.25 μ SOI Technology, *Proc. 13th Int'l Conf. on VLSI Design* (2000).
11) Halfhill, T.: Transmeta breaks x86 low-power barrier, *Microprocessor Report* (Feb. 2000).
12) Brooks, D., et al.: Wattch: A Framework for Architectural-Level Power Analysis and Optimizations, *Proc. 27th Int'l Symp. on Computer*

Architecture (*ISCA'00*), pp.83–94 (2000).
13) 井上弘士，村上和彰：実行履歴に基づいた低電力命令キャッシュ向けタグ比較回数削減手法，情報処理学会研究報告 (2000).
14) Mehta, H., et al.: Techniques for low energy software, *Proc. ISLPED'97*, pp.72–75 (1997).
15) Vijaykrishnan, N., et al.: Energy-Driven Integrated Hardware-Software Optimizations Using SimplePower, *Proc. ISCA'00*, pp.95–106 (2000).
16) Lebeck, A.R., et al.: Power Aware Page Allocation, *Proc. Architectural Support for Programming Languages and Operating Systems* (*ASPLOS IX*), pp.105–116 (2000).
17) Manne, S., et al.: Pipeline Gating: Speculation Control for Energy Reduction, *Proc. ISCA'98*, (1998).
18) Farkas, K.I., et al.: Quantifying the energy consumption of a pocket computer and a Java virtual machine, *Proc. Joint Int'l Conf. on Measurement and Modeling of Computer Systems* (*SIGMETRICS'00*), pp.252–263 (2000).
19) Brooks, D., et al.: Power-Aware Microarchitecture: Design and Modeling Challenges for Next-Generation Microprocessors, *IEEE MICRO*, Nov/Dec, pp.26–44 (2000).
20) Garrett, D., et al.: Challenges in Clockgating for a Low Power ASIC Methodology, *Proc. ISLPED'99*, pp.176–181 (1999).
21) Pant, M.D., et al.: An Architectural Solution for the Inductive Noise Problem due to Clock-Gating, *Proc. ISLPED'99*, pp.255–257 (1999).
22) 重田大助，小川洋平，山田克樹，中島康彦，富田眞治：命令畳み込み，データ投機および再利用技術を用いたJava仮想マシンの高速化，情報処理学会論文誌：ハイパフォーマンスコンピューティングシステム，Vol.41, No.SIG5(HPS1), pp.28–38 (2000).
23) Khoo, K.Y. and Willson, A.N.: Single-Transistor Transparent-Latch Clocking, *Proc. 16th Conf. on Advanced Research in VLSI* (*ARVLSI'95*), pp.331–341 (1995).
24) Sigal, J.W.L. and Chan, B.C.Y.: High Performance CMOS Circuit Techniques for the G-4 S/390 Microprocessor, *Proc. 1997 Int'l Conf. on Computer Design* (1995).
25) SPECJVM98 VERSION1.03, the Standard Performance Evaluation Corporation (1998).
26) OMRON Corporation: JeRTy. http://www.jerty.com/
27) Kaffe.org: Welcome to Kaffe. http://www.kaffe.org/

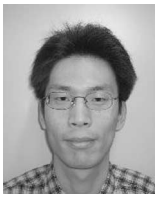
(平成 13 年 3 月 8 日受付)
(平成 14 年 3 月 14 日採録)

木村 篤彦

1978 年生．2001 年京都大学工学部情報学科卒業．同年株式会社ビーコンインフォメーションテクノロジー入社．ネイティブ XML データベース Tamino の開発に従事．

中島 康彦（正会員）

1963 年生．1986 年京都大学工学部情報工学科卒業．1988 年同大学院修士課程修了．同年富士通（株）入社．スーパコンピュータ VPP シリーズの VLIW 型スカラプロセッサ，命令エミュレーション，高速 CMOS 回路設計等に関する研究開発に従事．工学博士．1999 年京都大学総合情報メディアセンター助手．同年同大学院経済学研究科助教授，現在に至る．計算機アーキテクチャに興味を持つ．IEEECS，ACM 各会員．



宮田 佳昭

1965 年生．1989 年京都大学工学部電気工学科卒業．1991 年同大学院修士課程修了．同年オムロン（株）入社．UNIX ワークステーション周辺チップセット，組み込み機器プラットフォーム，Java プロセッサ等の開発に従事．現在に至る．

中川 伸二

1967 年生．1990 年神戸大学工学部電気工学科卒業．同年オムロン（株）入社．UNIX ワークステーション周辺チップセット，FA コントローラ命令実行プロセッサ，Java プロセッサ等の開発に従事．現在に至る．

五島 正裕（正会員）

1968 年生．1992 年京都大学工学部情報工学科卒業．1994 年同大学院工学研究科情報工学専攻修士課程修了．同年より日本学術振興会特別研究員．1996 年京都大学大学院工学研究科情報工学専攻博士後期課程退学，同年より同大学工学部助手．1998 年同大学院情報学研究科助手．高性能計算機システムの研究に従事．2001 年情報処理学会山下記念研究賞受賞．

森 眞一郎（正会員）

1963 年生．1987 年熊本大学工学部電子工学科卒業．1989 年九州大学大学院総合理工学研究科情報システム学専攻修士課程修了．1992 年同大学院総合理工学研究科情報システム学専攻博士課程単位取得退学．同年京都大学工学部助手．1995 年同助教授．1998 年同大学院情報学研究科助教授．工学博士．並列/分散処理，計算機アーキテクチャの研究に従事．IEEE，ACM 各会員．

北村 俊明（正会員）

1955 年生．1978 年京都大学工学部情報工学科卒業．1983 年同大学院博士課程研究指導認定退学．同年富士通（株）入社．汎用コンピュータ，スーパコンピュータ VPP シリーズの VLIW 型 CPU，M アーキテクチャ・命令エミュレーション，米国 HAL 社において SPARC プロセッサ等の研究開発に従事．工学博士．2000 年京都大学総合情報メディアセンター助教授．2002 年広島市立大学情報科学部教授，現在に至る．計算機アーキテクチャに興味を持つ．電子情報通信学会，IEEE，ACM 各会員．

富田 眞治（正会員）

1945 年生．1968 年京都大学工学部電子工学科卒業．1973 年同大学院博士課程修了．工学博士．同年京都大学工学部情報工学教室助手．1978 年同助教授．1986 年九州大学大学院総合理工学研究科教授，1991 年京都大学工学部教授，1998 年同大学院情報学研究科教授，現在に至る．計算機アーキテクチャ，並列処理システム等に興味を持つ．著書「並列計算機構成論」「計算機システム工学」「並列処理マシン」「コンピュータアーキテクチャ I」等．電子情報通信学会，IEEE，ACM 各会員．平成 7，8 年度，10，11 年度本会理事．