2007－OS－105 (6)
2007／4／5

QEMUを用いた命令，アドレストレーサの実現

松尾 和弥[†] 佐藤 未来子[††] 並木 美太郎[†††]

OSをはじめとするシステム構築において，実際の計算機の動作データ，資源利用データは有用な情報となる．そこで本研究では各種アーキテクチャ対応の高速な命令，メモリアクセストレーサを構築した．構築したトレーサによってx86，ARMアーキテクチャでの実行命令列，アクセスされた物理アドレス，物理アドレスに対応する仮想アドレス，アクセス時・命令実行時のCPU特権レベル，実行中プロセスのPIDといった情報がトレースできる．構築はフリー，オープンソースのCPUエミュレータQEMUに300行程度のコードを追加することで実現した．また，有用性を確認するために，構築したトレーサの性能評価を行い，さらに実際に構築したトレーサを省電力手法の考察に応用した例を示す．

Development of instruction and memory access tracer using QEMU

Kazuya Matsuo,[†] Mikiko Sato[††] and Mitaro Namiki [†††]

In development of computer systems such as OS, actual data about behaviour and resource usage of a computer is important. So we developed high speed tracer of executed instruction and main RAM access for various architecture. By our tracer, we can trace data about executed instructions, accessed physical and virtual memory address, CPU mode and PID on x86 and ARM. To develop our tracer, we remodeled QEMU, free and open source CPU emulator. In addition, to show the usefulness of our tracer, we evaluated the performance of our tracer and applied our tracer to the research of low power computer system.

1. はじめに

1.1 研究の背景

OSをはじめとするシステム構築において，実際の計算機の動作データ，資源利用データ（以下，実行時データ）は有用な情報となる．実行時データを分析，考察することでシステム構築の設計指針を得ることができるためである．例は第2章2.1節で示す．

実行時データを採取するためには既存の手法として，回路レベル，HDLレベルでのトレーサの利用が考えられる．これらのトレーサはハードウェアの詳細をトレースするためには有用である．しかし，詳細なハードウェア情報をトレースできるかわりに実行速度が低速である．OS構築を考える場合，ハードウェア情報の詳細は必ずしも必要ではない．また，OS構築では，ハードウェア情報だけでなく実行中プロセスのPIDなどOS，ユーザープロセス依存の情報も必要となる．よって既存のトレーサは必ずしもOS構築のためのトレース環境として十分なものであるとはいえない．

1.2 本研究の概要

既存のトレーサの問題点をふまえ，本研究では，実行速度が高速，かつOSやユーザープロセス依存の情報をトレースできる命令・メモリアクセストレーサの構築を行う．トレーサの構築はフリー，オープンソースのCPUエミュレータQEMUを改造することで実現する．構築したトレーサによってx86，ARMアーキテクチャでの実行命令列，アクセスされた物理アドレス，物理アドレスに対応する仮想アドレス，アクセス時・命令実行時のCPU特権レベル，実行中プロセスのPIDといった情報がトレースできる．また有用性を示すために，構築したトレーサの性能評価も行う．さらにトレーサから得られるデータの有用性を示すために，本トレーサを省電力研究に応用した例を示す．

† 東京農工大学工学部情報コミュニケーション工学科
Department of Computer and Information Science, Tokyo University of Agriculture and Technology
†† 東京農工大学大学院工学府
Graduate School of Technology, Tokyo University of Agriculture and Technology
††† 東京農工大学大学院共生科学技術研究院
Institute of Symbiotic Science and Technology,Tokyo University of Agriculture and Technology

2. 問題分析

2.1 トレーサの有用性

OSをはじめとするシステム構築において，実際の計算機の動作データ，資源利用データ（以下，実行時データ）は有用な情報となる．これら実行時データがシステム構築の有用な指標となるためである．例として省電力のためのOS資源管理を考える．近年の省電力への要求の高まりとともに，ハードウェアによる電力制御技術も細粒度化し[1)2)]，計算機内部のユニット単位での電源管理が可能になりつつある．例えば，CPUであれば利用していない演算ユニットの電源を切るといったことが考えられる．主記憶であれば全体を複数個のブロックに等分し，データを保持していないブロックには電力を供給しないようにするといったことが考えられる．

これらの省電力技術を有効に利用するためにはOSによる資源管理が重要になる．主記憶の電源管理をブロック単位で行うのであれば，OSは使用ブロック数を少なく抑えるメモリ割当てを要求される．このときメモリアクセスに関する実行時データがOS設計の指針として有用になる．実行時データを分析，考察することで，使用ブロック数をどの程度抑えることが可能か，どのようなアルゴリズムが有効かといった情報を得ることができる．また実行時データから得られる数値は，新たな手法の効果を試算する場合にも利用できる．本論文の第6章で述べる例では，新たな省電力手法の適応前後の消費電力を算出，比較するために，メモリアクセスに関する実行時データから得られるメモリアクセス回数を利用している．よって，実行時データを得るためのトレーサは有用である．

2.2 既存のトレーサの問題点

トレーサには既存の手法として回路レベル，HDLレベルでのトレーサがある．これらのトレーサはハードウェアの詳細をトレースする場合には有用である．しかし，OS構築のためのデータとしては，ハードウェア的な情報だけでなく，OSやユーザープロセスに依存したソフトウェア的な情報が必要となる．例えばメモリアクセスに関する情報をトレースする場合であれば，ハードウェア的な情報としてアクセスされた物理アドレス，アクセスの読込み・書込みの種別，アクセス時刻といったものが考えられる．しかしこれ以外に，物理アドレスに対応する仮想アドレス，アクセスを起こしたプロセスのプロセスID（Linux系OSの場合），プロセスの特権レベルといったソフトウェアに依存する情報も有用な実行時データとなる．

また，回路レベル，HDLレベルでのトレーサは詳細なハードウェアデータをトレースできる代わりに実行速度が低速である．OS構築のために必要なデータをトレースするのであれば，必要以上に詳細なデータはトレースせずに実行速度を向上させた方が有用である．これらの理由から，既存のトレーサは必ずしもOS構築のためのトレース環境として有用であるとはいえない．

3. 本研究の目標と方針

これら既存のトレーサの問題点を考慮し，本研究では次の条件を満たす命令・メモリアクセストレーサの構築を行う．

- 実行速度が高速である．
- ハードウェア的な情報だけでなく，OSやユーザープロセスに依存した情報のトレースも可能である．
- x86，ARMなど多様なアーキテクチャのトレースに対応している．

多様なアーキテクチャへの対応を目標としたのは，実行時データがアーキテクチャを問わず有用な情報であるためである．

これらの要求を満たすためトレーサの構築は，フリー，オープンソースのCPUエミュレータであるQEMUを改造することで実現する．QEMUを選んだのは次の理由による．

- 中間コードを介した動的なエミュレーションによって比較的高速な実行速度を実現している．
- x86だけでなくARM，MIPSなど様々なアーキテクチャのエミュレート機能を備えている．

QEMUのソース，バイナリ，ドキュメントなどは文献3)のWebサイトから入手できる．

次章以降の本論文の構成について述べる．第4章で，本トレーサの設計と実装について述べる．第5章では，構築したトレーサの性能評価を行う．そして第6章で本トレーサの有用性を示すために，トレース結果を省電力手法の考察に応用した例について述べる．最後に第7章で今後の課題について述べ，本論文をまとめる．

4. トレーサの設計と実装

4.1 命令トレース機能

命令トレース機能に関しては，実行された全ての機械語を時系列を保った形でトレースできるようにする．QEMUには既存の機能として，ターゲット（エミュレート対象）の機械語をログファイルに出力する機能がある．この機能は，中間コードへの変換が行われる

タイミングでターゲットの機械語をファイルに出力するものである．ただし，中間コードへの変換は毎回行われるとは限らない（QEMU の持つ高速化のための工夫による．詳細は QEMU のソース参照[3]）．そのため既存のトレース機能では，全ての機械語をトレースすることはできない．本トレーサでは既存の命令トレース機能に改造を加えることで，実行された全ての機械語を時系列を保った形でトレースできるようにする．また OS 依存の情報として，機械語実行時の CPU の特権レベル，機械語を実行したプロセスの PID（Linux 系 OS の場合）もトレース結果に含めるようにする．

4.2 メモリトレース機能

メモリアクセスのトレース機能に関しては，図 1 のように，どのアドレスがアクセスされたかという情報を時系列を保った形でトレースできるようにする（ただし図 1 は便宜上，命令フェッチによるアクセスを省いている）． さらにトレース結果には読込み・書込み

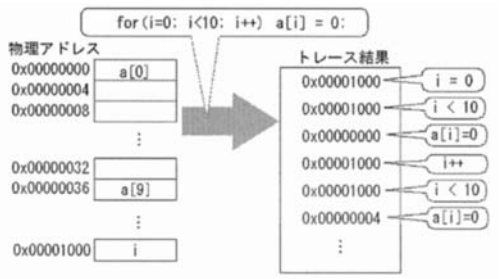

図 1 メモリトレース機能の仕様
Fig. 1 specification of main RAM tracer

の種別，アクセス時の実行命令数（アクセス時刻の指標として）も情報として含める．また OS 依存の情報としてアクセス時の CPU の特権レベル，アクセスを行ったプロセスの PID，アクセスされた物理アドレスに対応する仮想アドレス情報もトレース対象とする．

4.3 実　　装

実装は QEMU のソースコード約 2300 行の範囲に合計約 300 行のコードを追加することで実現した．今回の実装で x86，ARM アーキテクチャのトレースへの対応を実現した．PID のトレースに関しては，エミュレート対象の Linux カーネルにも若干の改造が必要になる．具体的にはグローバル変数をひとつ追加し，プロセス切替のタイミングで追加した変数に切替後の PID を格納するコードを追加する必要がある．そして QEMU 側で PID が格納されたアドレスを参照することで実行中プロセスの PID のトレースを実現する．上記の改造を施していないカーネルでも問題なくエミュレートできるが，この場合 PID としてトレースされる数値は無意味となる．

構築したトレース環境から得られる命令トレース結果と対応するメモリトレース結果の例を図 2 に示す（ただしメモリトレース結果はバイナリとして出力されたデータをテキスト化したものである）． 命令フェッ

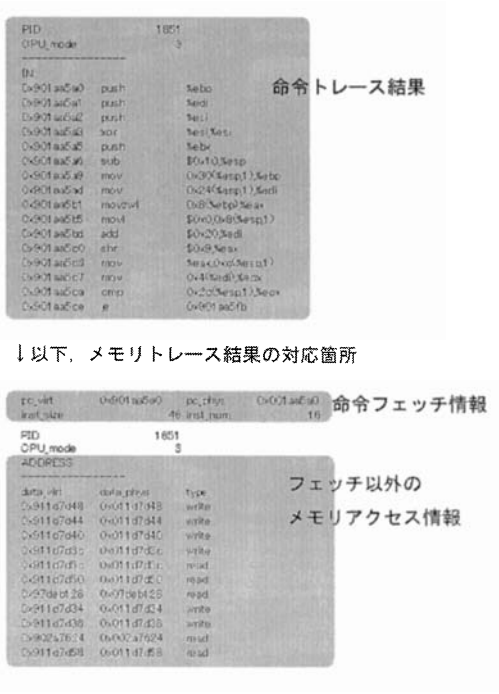

図 2 命令とメモリのトレース結果
Fig. 2 result of instructions and memory access trace

チによるメモリトレースとフェッチ以外でのメモリトレースが分かれているのは QEMU の実装に合わせたためである（複数個の機械語をまとめて変換し，それからまとめて実行する．詳細はソース参照[3]）．最後に本トレーサでトレースできるデータと想定する用途を表 1 にまとめる．

5. 構築したトレーサの性能

構築したトレーサの性能を評価するため，実際に構築したトレーサを用いて実行時データのトレースを行い，性能を測定した．トレース対象には文献 3) の Web サイトで配布されている x86，ARM 用それぞれのテスト用 Linux イメージを用いた．x86，ARM それぞれで電源投入から Linux のログイン画面表示まで（図 3，4）をトレースし，性能を測定した．その結果

表 1　本トレーサで採取できるデータとその用途
Table 1　The data available by our tracer and its usage

データ	用途
実行された機械語	CPU の状態分析
アクセスされた物理アドレス	物理メモリの使用状況分析
アクセスの読込み・書込み種別	物理メモリの使用状況分析
アクセス時の実行命令数	アクセス時刻の指標
アクセスされた仮想アドレス	ソフトウェアの考察
アクセス時，命令実行時の実行中 PID	ソフトウェアの考察
アクセス時，命令実行時の CPU 特権レベル	ソフトウェアの考察

表 2　アクセス回数とトレース結果サイズの関係
Table 2　Relation of access frequency and file size

	命令フェッチ回数（＝実行命令数）	命令フェッチ以外のアクセス回数	メモリトレースのサイズ	命令トレースのサイズ
X86	402653184	49351107	1.7GB	4.1GB
ARM	2625634304	284244737	9.5GB	計測失敗

表 3　構築したトレーサの実行速度
Table 3　Execution speed of our tracer

	未改造の QEMU	メモリアクセスのみをトレースした場合	命令のみをトレースした場合	両方をトレースした場合
X86	5 秒	50 秒	321 秒	365 秒
ARM	15 秒	349 秒	計測失敗	計測失敗

を表 2，3 に示す．ただし，ARM の命令トレースに関しては，途中でフリーズしてしまい，データを得られなかった．

図 3　x86 テスト用 Linux のログイン画面
Fig. 3　Login screen of x86 Linux for test

図 4　ARM テスト用 Linux のログイン画面
Fig. 4　Login screen of ARM Linux for test

まず，メモリアクセスのトレースに関しては，x86 で 10 倍，ARM で約 23 倍の速度低下があった．ただし現在，暫定的なバグ回避のために実行速度が低下している（バグ回避のために QEMU の実行速度向上のためのコードをコメントアウトしているため）．デバッグが完了した場合，速度低下は x86 で 4 倍以内，ARM で 8 倍以内に抑えられる見込みである（バグが含まれていることは度外視し，実行速度だけを計測した数値）．x86 では，実機と比較した場合でも☆70 倍以内の速度低下に抑えられる見込みである．また，メモリトレースのファイルサイズもデバッグ完了時には現在の 3 分の 1 程度に抑えられる見込みである．デバッグ完了後の性能はトレーサとして十分な数値であると考えている．

命令トレース機能に関しては，速度面，ファイルサイズ面ともに改善の必要がある結果となった．ただし現在，命令トレース機能の実装には既存の QEMU の機能を流用しており，機械語のバイナリをニーモニックに変換し，テキスト形式で出力する仕様となっている．そのためトレース結果を直接バイナリとして出力する仕様に変更することで，性能面，サイズ面ともに

☆ 起動から CUI 版 FedoraCore5 のログイン画面表示まで計測

改善は可能である．

6. 省電力手法の考察への応用

本章では構築したトレーサから得られるデータの有用性を確認するために，実際に本トレーサから得られたメモリトレース結果を省電力手法の考案，評価に応用した例を示す．

6.1 省電力の必要性

近年，省電力に対する要求が高まってきている．組込み分野ではバッテリーの有効利用の観点から，省電力は従来より重要な問題であった．近年では加えて，ハイエンドなマシン環境でも省電力への要求が高まってきている．これはマシンの高性能化に伴う消費電力・発熱量の増大が大きな問題となっているためである．冷却装置の限界がクラスタ実装の密度を制限してしまうなど，発熱が性能の足枷となりつつある．文献 4)5) のようにハイエンド環境の低電力化についての研究も多く行われている．

このように省電力への要求が組込み・ハイエンドの別を問わず高まってきている．そのため省電力化を達成するためのハードウェア技術もかなり発展してきている．中には現状で既に製品化されている技術もある．CPU の周波数・電圧制御（以下 VF 制御）技術である Intel 社の SpeedStep，AMD 社の PowerNow! などがその例である[6)7)]．周波数・電圧制御技術についてはその利用法についての研究も盛んに行われている[8)9)10)11)]．

VF 制御以外にも CPU のパイプライン[12)13)] やキャッシュメモリ，主記憶や外部ストレージも省電力研究の対象となる．また，さらに細粒度の電力制御を可能にする技術も研究が進められている[1)2)]．将来的には低電圧動作・クロック数低減だけでなく，さらにハードウェアの内部に踏み込んだ省電力手法が可能になると考えられる．例えば「使わない機能ブロックには電力供給そのものを停止する」といった手法が考えられる．CPU であれば使用していない演算ユニットの電源を切る，メモリであれば使用していないアドレスに電力を供給しないといった手法である．これらの技術を用いた省電力化のためには，OS による資源管理が重要となる．

本章では細粒度の電力制御機能の例として，「N_B 個のブロック分割されており，ブロック単位で OS 側から電源を ON・OFF できる主記憶装置」を仮定する．そしてこの電力制御機能を活かすための OS 機能を 6.2 以降の節で考案，評価していく．

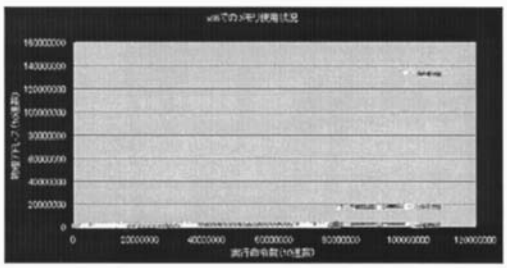

図 5 x86 でのメモリ利用状況
Fig. 5 Access pattern on x86 linux for qemu test

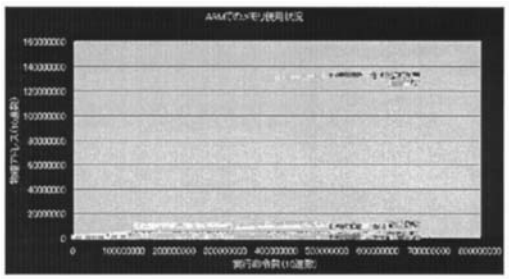

図 6 ARM でのメモリ利用状況
Fig. 6 Access pattern on ARM linux for qemu test

6.2 データ採取

データ採取のために 128MB の物理メモリを搭載した x86 と ARM アーキテクチャ上をエミュレートし，第 5 章のように電源投入から Linux のログイン画面表示までのアクセスをトレースした．トレース結果から時間経過とメモリ利用状況の関係を視覚化したものを図 5，図 6 に示す．いずれの図も X 軸は経過時間に準ずるものとして実行命令数，Y 軸は物理メモリのアドレスとなっている．X 軸 Y 軸ともに 10 進数表記である．プロット箇所は X 座標の時刻に Y 座標の物理アドレスへのアクセス（読込・書込は区別せず）があったことを意味している．

6.3 結果の分析と手法の考案

図 5，図 6 いずれの場合も使用されていないアドレスがかなり存在している．メモリはアクセスが無い場合でもリフレッシュなどのために待機電力を消費する．したがって，これら未使用部分の電源を OFF にすれば待機電力による電力消費を削減できると考えられる．そこで「各ブロックの電源は最初のアクセス時に ON とし，それ以降は ON のままにしておく」というメモリ電源管理手法を考案する．次節以降で実際にこの手法によって，どの程度の改善が見込めるかを評価する．

6.4 SDRAM の電力量モデル

手法の評価を行うには定量的な指標が必要である．

そこでまずメモリの電力量のモデル式を構築する．式の構築は現在主流の SDRAM（ただし DDR ではない）を基に進めていく．Elpida Memory 社の技術資料[14] によると SDRAM の平均消費電流値は (動作電流値 + スタンバイ電流値 + バースト電流値 + リフレッシュ電流値) ÷ トータル時間 で求められる．トータル時間というのはメモリに電源が入っていた時間である．この平均消費電流とメモリの動作電圧 V_m との積を取ることでメモリの平均消費電力が求まる．

$$\text{平均消費電力} = V_m \times \text{平均消費電流} \quad (1)$$

この平均消費電力を時間積分することでメモリが消費した電力量を算出できる．トータル時間を T とおくと電力量は次のようになる．

$$\begin{aligned}\text{電力量} &= \int_0^T \text{平均消費電力}\, dt \\ &= T \times V_m \times \text{平均消費電流値} \quad (2)\end{aligned}$$

(2) 式から分かるように平均消費電流値，つまり動作電流値・スタンバイ電流値・バースト電流値・リフレッシュ電流値の 4 つの要素がメモリの電力量を大きく決定付ける要因となっている．いくつかの仮定を置きながら実際の SDRAM（$\mu PD45128163G5-A75-9JF$）の規格値を代入すると (2) 式は次のようになる．

$$\text{電力量} = 3.3 \times (5945N_{rw} + 301.7F_{all}) \quad (3)$$

単位は $ns \cdot mA \cdot V$ となる．それぞれの要素をどのように計算したかについては以下に示す．ただし詳細については必要に応じて文献 14) を参照していただきたい．

- 動作電流値

  動作電流値の計算式は次の形に帰着できる．

  $$\begin{aligned}&\text{動作電流値} = \\ &\text{定数} \times ACT \text{ コマンドの入力回数} \quad (4)\end{aligned}$$

  ここでは全ての ACT コマンド[14] がメモリへの読み込み・書き込みのために行われたと仮定する．メモリに対して行われた読み込みと書き込みの合計回数を N_{rw} とおき規格値を代入すると動作電流値の計算式は以下のようになる．

  $$\text{動作電流値} = 3770 \times N_{rw} \quad (5)$$

- スタンバイ電流値

  スタンバイ電流値の計算式は次の形に帰着できる．

  $$\begin{aligned}\text{スタンバイ電流値} &= \text{定数} \times \sum tRAS \\ &+ \text{定数} \times \sum tRP \quad (6)\end{aligned}$$

  文献 14) では $\sum tRAS$ は ACT コマンド入力から PRE コマンド[14] 入力までの総時間とされている．ここでは簡単化のため $\sum tRAS$ を「メモリへの読み込み・書き込みに要した時間のトータル」と仮定する．文献 14) では読込み・書込みともにメモリサイクルで 6 クロックを要すると仮定して試算を行っていた．それに従うと $\sum tRAS$ はメモリのクロック周期 tCK を用いて次式で表せる．

  $$\sum tRAS = 6 \times tCK \times N_{rw} \quad (7)$$

  また $\sum tRP$ は次式のように定義されている．

  $$\begin{aligned}\sum tRP = T &- \sum tRAS \\ &- \text{定数} \times RN \quad (8)\end{aligned}$$

  T は前述のとおりメモリが動作していたトータル時間である．RN は時間 T の間に行われたリフレッシュ動作の回数である．規格値によるとリフレッシュは 8192 サイクル間隔で実行される．そのため T をメモリクロック数に変換した値を F_{all} とおくと $RN = \frac{F_{all}}{8192}$ となり $\sum tRP$ は次式で表される．

  $$\begin{aligned}\sum tRP = F_{all} \times tCK &- \sum tRAS \\ &- \text{定数} \times \frac{F_{all}}{8192} \quad (9)\end{aligned}$$

  (7) 式, (9) 式および規格値を代入するとスタンバイ電流値は次のようになる．

  $$\begin{aligned}\text{スタンバイ電流値} &= 900 \times N_{rw} \\ &+ 299.8 \times F_{all} \quad (10)\end{aligned}$$

- バースト動作電流値

  バースト動作電流値は次式で表される．

  $$\begin{aligned}\text{バースト動作電流値} &= \text{定数} \times \\ &(BN - AN) \times tCK \quad (11)\end{aligned}$$

  ここで AN は ACT コマンドの入力回数である．さきほど動作電流値を求める際に仮定したように $AN = N_{rw}$ とする．BN については詳しくは説明しないが，バースト長を全て 4 と仮定することで $BN = 4 \times N_{rw}$ とおける．これらの式と規格値を代入することでバースト動作電流値は次式のようになる．

  $$\text{バースト動作電流値} = 1275 \times N_{rw} \quad (12)$$

- リフレッシュ電流値

  リフレッシュ電流値は次式で表される．

  $$\text{リフレッシュ電流値} = \text{定数} \times RN \quad (13)$$

  RN は前述の通り，SDRAM の動作中に行われたリフレッシュの回数である．スタンバイ電流値のときに求めたように $RN = fracF_{all}8192$ となる．これまでと同様，規格値と RN を代入することでリフレッシュ電流値は以下のようになる．

  $$\text{リフレッシュ電流値} = 1.9 \times F_{all} \quad (14)$$

6.5 分割されたメモリへの拡張

次に（3）式を「N_B 等分されたブロック 1 つあたりの電力量のモデル式」に拡張する．各ブロックの電源を独立して ON・OFF した場合の消費電力を見積もるにはブロック 1 つあたりのモデル式が必須である．（3）式の右辺第 1 項 $5945N_{rw}$ はメモリへのアクセス回数に比例する．右辺第 2 項 $301.7F_{all}$ はメモリ動作時間（のメモリクロック換算値）に比例する．つまり前者がアクセスによる電力消費を，後者が待機電力による電力消費を表している．

ブロック 1 つあたりのモデル式を求めるために次の 2 つの条件を仮定する．

- 1 回のメモリアクセスに要する電力量は一定である（ブロック・アドレスを問わない）
- メモリの待機電力はメモリ容量に比例する

1 回のメモリアクセスに要する電力量は一定なのでブロック分割後もアクセス回数にかかる係数 5945 は変わらない．一方，メモリ動作時間のメモリクロック換算値にかかる係数 299.8 は分割後のブロックサイズに応じて変更する必要がある．メモリ全体を N_B 個のブロックに等分したと仮定すればクロック数にかかる係数は $\frac{301.7}{N_B}$ となる．またブロックごとに独立して電源を ON・OFF できることから各ブロックは互いに並列と仮定し，動作電圧は分割前と同じく 3.3V とする．以後，便宜上，分割したブロックは 0 から $N_B - 1$ までの通し番号で識別することにする．メモリブロック i へのアクセス回数を n_{irw}，動作時間のメモリクロック換算値を f_i とおくとメモリブロック i の電力量は次式で表すことができる．

$$\text{ブロック } i \text{ の電力量} = 3.3 \times (5945n_{irw} + \frac{301.7f_i}{N_B})$$

このときメモリ全体の電力量は次のようになる．

$$\begin{aligned} \text{電力量} &= 3.3 \times \sum_{i=0}^{N-1}(5945n_{irw} + \frac{301.7f_i}{N_B}) \\ &= 3.3 \times (5945N_{rw} + \sum_{i=0}^{N-1}\frac{301.7f_i}{N_B}) \end{aligned} \quad (15)$$

なお，(15) 式は電源の ON・OFF に要するオーバーヘッドを考慮していない．今回の手法では各ブロックともたかだか 1 回の電源 ON しか行われないためである．

6.6 電力量の試算

構築したモデル式とトレース結果から電力量を試算する．今回は簡単化のために次の条件を仮定する．

- $N_B = 128$
- データキャッシュ無し
- データアクセス無しの命令は 1CPU クロック，アクセス有りの命令は 60CPU クロックで実行
- CPU のクロック周期：メモリのクロック周期 = 1 : 10
- 命令キャッシュのキャッシュミスは無視（つまり命令フェッチによるアクセスも無視）

これらの仮定のもと式（3），(15）から省電力機能有りの場合と無しの場合で図 5，6 のケースにおけるメモリ電力量がどの程度変化するかを試算した．その結果，考案した手法によって x86，ARM でのメモリの消費電力をそれぞれ 23.1%，22.5%抑えられるという結果が得られた．

7. ま と め

本論文では，実行速度が高速かつ，多様なデータ，アーキテクチャに対応した命令・メモリアクセストレーサの構築を行った．そして構築したトレーサの有用性を確認するために，本トレーサの性能評価を行った．さらに本トレーサによって得られるデータの有用性を示すために，本トレーサを省電力研究の考察に応用した例を示した．

今後の課題としてまず，実行速度の向上に関するものが挙げられる．まず，現在暫定的に回避しているバグの除去を完了する必要がある．バグの除去が完了すればメモリトレース機能の性能向上が見込める．また，命令トレース機能の高速化，ファイルサイズ縮小も課題である．これについては第 5 章で述べたように，出力のバイナリ化による改善を考えている．また，トレース結果の I/O 出力の非同期化による実行速度向上も現在検討中である．

トレース結果の利用法についてもいくつか課題が挙げられる．第 6 章の例であれば次の課題が挙げられる．

- データキャッシュの考慮
- 命令キャッシュのキャッシュミスの考慮
- より正確な消費電力のモデル式の考案
- 省電力手法によるオーバーヘッドの考慮

また，電力量の試算に関しても，モデル式の正当性を検証していく必要がある．そのために本トレーサから得られた試算結果と実機の結果を比較することを考えている．実機での結果の計測には，非接触型の電力計測装置を利用することを考えている．他に，MIPS アーキテクチャへの対応も課題として考えている．

参 考 文 献

1) : 新世代マイクロプロセッサアーキテクチャ（前編）, 情報処理, Vol. 46, No. 10, pp. 1099–1143 (2005).
2) : 新世代マイクロプロセッサアーキテクチャ（後編）, 情報処理, Vol. 46, No. 11, pp. 1211–1265 (2005).
3) QEMU のソース・バイナリ・ドキュメントなど, http://fabrice.bellard.free.fr/qemu/.
4) 堀田義彦, 佐藤三久, 朴泰祐, 高橋大介, 中島佳宏, 高橋睦史, 中村宏: プロセッサの消費電力測定と低消費電力プロセッサによるクラスタの検討, 情報処理学会論文誌:コンピューティングシステム, Vol. 45, No. SIG11(ACS7), pp. 207–218 (2004).
5) 中島浩, 中村宏, 佐藤三久, 朴泰祐, 松岡聡, 高橋大介, 堀田義彦: 高性能計算のための低電力・高密度クラスタ MegaProto, 情報処理学会論文誌:コンピューティングシステム, Vol. 46, No. SIG12(ACS11), pp. 46–61 (2005).
6) テクノロジー解説 拡張版 Intel Speed-Step テクノロジー, http://www.intel.com/jp/builtin/technology/speedstep/.
7) AMD PowerNow!テクノロジ 概要, http://www.amd.com/.
8) 近藤正章, 中村宏: 主記憶アクセスの負荷情報を利用した動的周波数変更による低消費電力化, 情報処理学会論文誌:コンピューティングシステム, Vol. 45, No. SIG6(ACS6), pp. 1–11 (2004).
9) 渋谷雄, 辻野嘉宏, 倉本到, 新保稔康, 中本幸一: 未知処理量タスクに対する省電力化を目指した周波数制御リアルタイムスケジューリングアルゴリズムとその評価, 情報処理学会論文誌, Vol. 46, No. 6, pp. 1426–1435 (2005).
10) 佐々木広, 浅井雅司, 池田佳路, 近藤正章, 中村宏: 統計情報に基づく動的電源電圧制御手法, 情報処理学会論文誌:コンピューティングシステム, Vol. 47, No. SIG18(ACS16), pp. 80–91 (2006).
11) A.Goel, C.M.Krishna and I.Koren(eds.): *Energy Aware Kernel for Hard Real-Time Systems* (2005). Proceedings of the 2005 international conference on Compilers, architectures and synthesis for embedded systems CASES '05.
12) 嶋田創, 安藤秀樹, 島田俊夫: パイプラインステージ統合によるプロセッサの消費エネルギーの削減, 情報処理学会論文誌:コンピューティングシステム（ACS4）, pp18-30（Jan. 2004）.
13) 市川裕二, 佐々木敬泰, 弘中哲夫, 谷川一哉, 北村俊明, 近藤利夫: 可変パイプラインを用いた低消費エネルギープロセッサの設計と評価, 情報処理学会論文誌:コンピューティングシステム, Vol. 47, No. SIG7(ACS14), pp. 231–242 (2006).
14) SDRAM の使い方, http://www.elpida.com/pdfs/J0123N60.pdf.