2005－OS－100（8）
2005／8／3

プロセス単位電力制御機構の予備評価

宮川大輔[†]　石川　裕[†]

DVS(Dynamic Voltage Scaling) は CPU の動作電圧および動作周波数を変えることで計算機の消費電力を下げる技術である。この DVS を用いてプロセス単位で CPU の動作周波数を変えることが出来れば、応答性能を下げることなく消費電力量を下げることが可能となる。しかし、その実用可能性を示すためには CPU 毎の DVS の性質とプロセス毎に動作周波数を変えるために必要なオーバーヘッドについての評価が不可欠となる。本論文では二つの実験を行ない、プロセス単位電力制御の実用可能性について予備的に評価する。DVS を用いたときのプログラムの電力量 (電力の時間積分値) を測定し、Pentium 4 と Pentium M の電力的性質を示す。次に、周波数の異なる二つのプロセスを同時に実行した時と別々に実行した時の電力量の違いを測定する。結果として、Pentium M プロセッサを用いればプロセス単位電力制御機構をほぼオーバーヘッドなしに実現可能であることを示す。

Preliminary Evaluation of the Per-Process Power Consumption Control Mechanism

Daisuke Miyakawa[†] and Yutaka Ishikawa[†]

The DVS(Dynamic Voltage Scaling) is the technique that decrease power consumption of a computer by changing operating voltage and frequency. If we controlled the operating frequency on process base by using DVS, we would be able to decrease the power consumption without slowing response. However, in order to show its availability, we have to evaluate the DVS's properties of CPU and overheads necessary to change the operating frequencies on process base. In this research, we perform two experiments and preliminarily evaluate the availability of the Per-Process Power Consumption Control Mechanism. First, we look through the electricity characteristics of Penium 4 and Pentium M by measuring the total power consumption (temporal integration of power consumption per unit of time) when using the DVS. Second, we measure the difference of total power consumption of two processes with different frequencies, between simultaneous execution and sequential execution. As a result, we conclude that the Per-Process Power Consumption Controlling System is realized with almost no overhead if we use a Pentium M processor.

1. 背　景

CPU の消費電力を減らすために、主要な CPU は DVS(Dynamic Voltage Scaling) 機構を提供している。例として Intel 社の Speed Step[4] や Enhanced Speed Step[3]、AMD 社の Power Now[2] などが挙げられる。

DVS は CPU の動作電圧を下げることによって CPU の消費電力を減らす技術である。DVS においては、電圧を下げることによって回路の遅延時間も長くなるため、動作を安定させるために動作周波数も下げなくてはならなくなる。DVS は動作周波数を下げることで消費電力を下げる機構、という見方をすることも出来る。OS の提供している DVS インターフェースは動作周波数値を下げることによって消費電力を下げるように出来ている。例えば Linux の cpufreq モジュールでは、ユーザが動作周波数を仮想ファイルに書き込んだときに CPU の動作電圧が変わる。cpufreq モジュールの提供するインターフェースにおいてユーザに見えるパラメータは、動作電圧ではなく動作周波数である。

本論文では DVS を用いて、動作周波数および電力制御を CPU 単位ではなくプロセス単位で行なう機構、すなわちプロセス単位電力制御の有用性について考察する。現状の OS では TSS(Time Sharing System) 方式の採用により、複数のプロセスを疑似的に同時に実行することが出来るが、プロセスの違いによらず、実行時の動作周波数は常に一定になる。100 個のプロセスが同時に動いているとしても、それらはすべて、CPU 毎に設定される指定された周波数で動作する。

しかし一般に、各プロセスに対して要求される消費

† 東京大学
The University of Tokyo

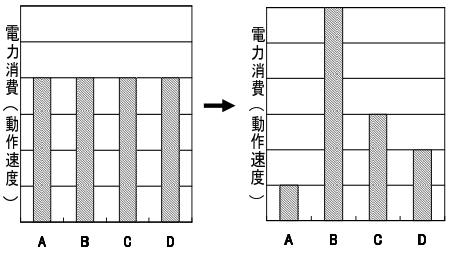

図 1　プロセス毎の動作速度 (電力消費) の傾向

電力および動作速度に対する要求は同じではない。省電力モードによって電力を抑えている状況下においても、ユーザは消費電力を抑えつつ迅速に反応する端末を期待するであろうし、ハードウェアのエラーを処理するプロセスだけは例外的に早くしたいであろう。

しかし、既存の OS では、ユーザに対する応答性を要求されるプロセスと速度を要求されないバッチプロセスを同時に動かした時に、電力的に効率良く、かつユーザから見て高速に処理を実行させることは出来ない。例として、ユーザが作業するために起動させた対話型プロセスと同時に、優先度の低いバッチプロセスが並行して動作している状況を考える。この状況で、ユーザに対する応答性を下げずにシステム全体の電力消費を減らそうとすると、現状のシステムでは次のような問題に陥る。

消費電力を減らそうとするのであれば、CPU の動作電圧を下げれば良い。CPU パワーを使用するバッチプロセスが低速に動作するから、消費電力は小さくなると期待される。しかし、それにともなってユーザが実行するすべてのプロセスの動作速度も落ちる。つまり端末からの応答自体が遅れがちになる。これはユーザにとって望ましくない。

逆に、ユーザからみた動作速度を十分なものに保とうとすると、今度は CPU の動作周波数を上げざるを得ない。すると、消費電力を減らすために速度を下げておきたいバッチプロセスまで高速に動作し、余分に電力を消費することになる。

プロセス単位電力制御がない場合、既存の OS では**図 1** の左図に示す通り、すべてのプロセスの CPU の動作周波数および消費電力が同じままである。よって、消費電力を下げたい場合にはユーザとの応答型プロセスの速度も下げる必要があるし、応答性を上げたい場合にはすべてのプロセスにおいて消費電力は最大にするしかない。

もし図 1 の右図のように各プロセス毎に CPU の動作周波数を設定できれば、上記の問題は解決する。ユーザとの対話型プロセスの動作周波数を上げ、それ以外のプロセスの動作周波数を下げるようにすれば良い。応答型プロセスは通常ほとんどアクティブにならないから、例えそれらのプロセスの動作周波数を上げても全体的な電力消費にはほとんど影響しない。CPU はその稼働時間のほとんどを動作周波数 (消費電力) が小さい状態で過ごし、消費電力は可能な限り最低な状態を保つ。またユーザに応答するプロセスは常に最高の周波数で動作するのだから、ユーザから見た応答性能が下がることはない。プロセス単位電力制御機構を実現することで、現状では両立させることの出来ない、応答速度を下げることなく消費電力を低減する、という一見矛盾する要求を満たすことが出来るようになる。

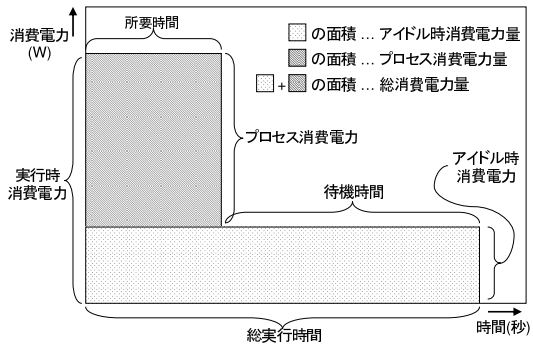

図 2　プログラム実行時の消費電力の変化

本論文の目的は、上記に示したプロセス単位電力制御機構を構築する上で必要な二つの予備評価を行なうことにある。一つは DVS 機構自体の電力量 (消費電力の時間積分値) から見た有用性の評価であり、もう一つはプロセス単位で電力制御を行なった場合のオーバーヘッドに関する評価である。以降の章ではこの二つの評価を順に行なうことで、プロセス単位電力制御機構の実用可能性について考察する。

2. 用語の定義

第 1 章においては、説明を簡潔にするため消費電力という言葉を消費電力量という言葉と同じ意味に用いた。消費電力とは単位時間当たりに消費される電力であり、W(ワット) や kW(キロワット) といった単位で呼ばれる。消費電力量とは消費電力の時間積分値であり、W 時 (W × 時間)、W 秒 (W × 秒) といった単位で呼ばれる。

図 2 は、プロセス実行中における消費電力の推移の様子と、そのグラフの各部分の名前を便宜的に定義したものである。縦軸は消費電力の大きさを示しており、横軸は実験中の時間の遷移を表している。矩形の

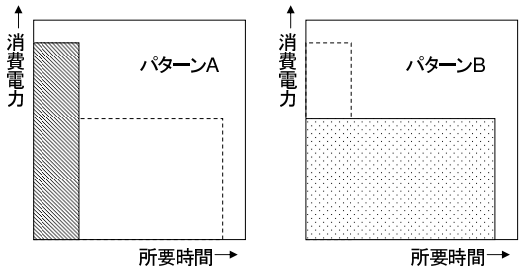

図 3　消費電力と消費電力の関係 A(左) と B(右)

面積が消費電力量になる。

図 2 の斜線部分はそのプログラムを動かしている時に増えた消費電力量を、点で描画された部分はプログラムを実行するしないに関わらず消費されるアイドル時の消費電力量を表している。

単純化するため、本論文ではプログラムの実行によって消費された電力量は濃い灰色部分の面積に相当することとし、この部分をプロセス消費電力量と呼ぶ。濃い灰色部分と薄い灰色部分の面積の総和は、指定された総実行時間の間に、プログラムを任意の動作電圧 (動作周波数) で実行することを許した場合の、その時間までに計算機が消費した電力量である。ここでは、総実行時間を最低の動作周波数における所要時間とし、これを総消費電力量と呼ぶことにした。実際の計算機においては、総実行時間はその計算機の電源を切るまでの時間である。

以降では、主にプロセス消費電力量について見る。また、プロセス消費電力量の算出に関係する所要時間、アイドル時消費電力、プロセス消費電力についても併せて考察する。

3. DVS 機構の電力量から見た有用性の検証

DVS を用いて低電圧動作させることで、プロセスの実行は**図 3** のパターン A からパターン B のように変化する。すなわち、動作周波数 (動作電圧, 消費電力) を下げることによって実行速度が遅くなり、プロセス実行にかかる所要時間が長くなる。確かに消費電力は下がっているが、プロセスの実行が終了するまでに要する所要時間は長くなる。もし、消費電力の減少に対して所要時間があまりに長くなるようなら、DVS を用いて動作周波数を下げた時の方が電力量 (図における矩形の面積に当たる) が大きくなる可能性がある。

本論文ではまずこの点について検証するため、代表的な CPU の内 Pentium 4 と Pentium M をそれぞれ搭載した PC を用いて消費電力量を計測し、DVS を用いることによって電力量がどのように増減しているかについて調べる。

表 1　被測定環境

測定対象	NEC 社製 MY30V/C-F	IBM 社製 Thinkpad X31
CPU	Pentium 4 3.0GHz	Pentium M 1.6 GHz
メモリ	512MB	1GB
OS	Debian GNU/Linux 3.1	Debian GNU/Linux 3.1
Kernel	2.6.12	2.6.12
コンパイラ	gcc 3.3.5	gcc 3.3.5

表 2　測定環境

測定用端末	Proside 社製 Mecolo A120
CPU	Pentium M 1.8GHz
メモリ	1GB
OS	Debian GNU/Linux 3.1
Linux Kernel	2.6.12
コンパイラ	gcc 3.3.5
補足	電力計測のため AD16-16(LPCI)L を搭載

```
pid = fork();
if( pid == 0 ){ // Child Process
  send_packet("begin");
  execvp( file, argv );
}else{ // Parent Process
  waitpid( pid, NULL, 0 );
   send_packet("end");
}
```

図 4　測定プログラム実行用ラッパ

4. 単一プロセスによる電力量測定実験

まず、一定の仕事量を有するプログラムを CPU で利用可能な各周波数で実行した時の電力量を測定し、DVS によって電力量がどのように変化するのかを調べる。

一定量の仕事を与えるプログラムとして、今回は NPB[1](NAS Parallel Benchmarks) の整数ソートプログラム is.A(問題サイズ A) を使用する。なお、予備実験として is.A 以外のプログラム等でも実験を行なっているが、傾向が同じため結果は省略する。

各動作周波数におけるアイドル時消費電力の判定に Linux の sleep コマンドを用いる。sleep コマンドは、与えられた整数秒だけそのプロセスをアイドル状態にするプログラムである。測定対象 PC はシングルユーザモードで動作させており、他にアクティブなプロセスは存在しないため、sleep コマンドが実行されている間、CPU はアイドル状態となる。今回は sleep コマンドに引数 20 を与えることで 20 秒間 CPU アイドル状態である時の消費電力を測った。

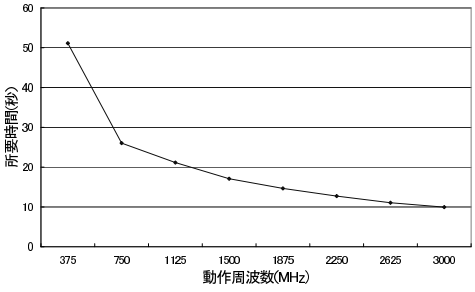

図 5　Pentium 4 搭載の PC で is.A を実行した時の所要時間

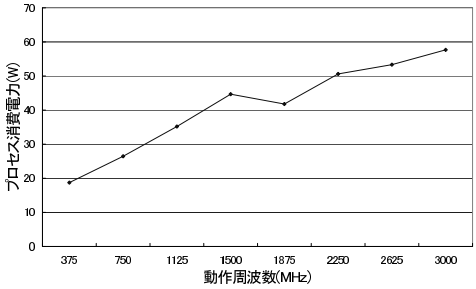

図 6　Pentium 4 搭載の PC で is.A を実行した時のプロセス消費電力

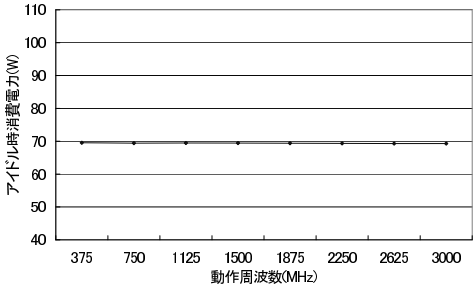

図 7　Pentium 4 搭載の PC で is.A を実行した時のアイドル時消費電力

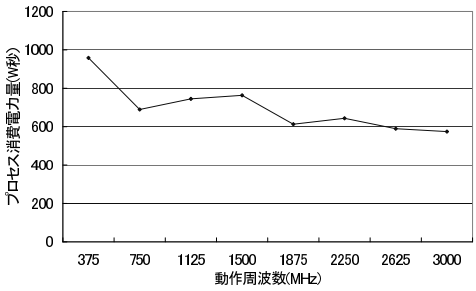

図 8　Pentium 4 搭載の PC で is.A を実行した時のプロセス消費電力量

プログラムは**表 1** で示される二種類の PC で実行し、実行時の所要時間と消費電力を、**表 2** の測定用 PC を用いて測定した。

対象のプログラムは**図 4** のラッパプログラムを通して実行される．`send_packet()` は今回の実験用に作られたライブラリの関数で、引数で与えられた文字を UDP パケットにして、おなじローカルエリアネットワーク内にある測定用 PC に送る．図 4 の場合、対象プログラムを実行する直前と終了して親が `waitpid()` から返って来た直後にパケットを送っていることになる．本実験では、この二つのパケット送信をもって測定用 PC にプログラム実行開始とプログラム実行終了を通知したものとする。

測定用 PC は、実験用に用意された電力監視装置より測定対象 PC に供給される電流値を取得し、その値を元に特定の時刻における測定対象 PC の消費電力を得る。また、測定用 PC は測定対象の PC からプログラム実行開始のパケットとプログラム実行終了のパケットを順に受けとることで、プログラムの所要時間を測る。電力はその間に得られた各時刻の電力を元にした実効電力値となる。

本実験では、プログラムの実行開始時と実行終了時に測定 PC へ UDP パケットを送信することにより、プログラム実行中の CPU の各周波数における所要時間と実効消費電力、消費電力量 (実効消費電力に所要時間を掛けたもの) を測定した。次項の結果では、10 回行った測定結果の平均を取ったものを掲載している。

5. 結　　果

5.1 Pentium 4

実験用プログラムを Pentium 4 搭載の PC で実行したときの動作周波数に対する所要時間、プロセス消費電力、アイドル時消費電力、プロセス消費電力量を**図 5**、**図 6**、**図 7**、**図 8** に示す。図 8 から分かる通り、Pentium 4 搭載の PC においては高い動作周波数でプロセスを実行した方が電力量は減少する。これは DVS に期待される結果と全く逆である。

Pentium 4 の 3 種類の実験結果においてさらに特徴的なのは、375MHz から 1.5GHz にかけて消費電力が上がり、1.875GHz で一度その消費電力が下がっていることである。CPU 内部、もしくはマザーボード側で 1.5GHz と 1.875GHz の間に特別な処理を行なったために電力消費の傾向を大きく乱したものと予想できるが、具体的な原因の特定には至らなかった。なお、この傾向も他の予備実験に対して同様の傾向を示していることから、is.A 実行時に特有な現象ではない。

図 7 が示すように、Pentium 4 を搭載した PC では、CPU の動作周波数をどの値に設定してもアイドル時消費電力は変わらない。

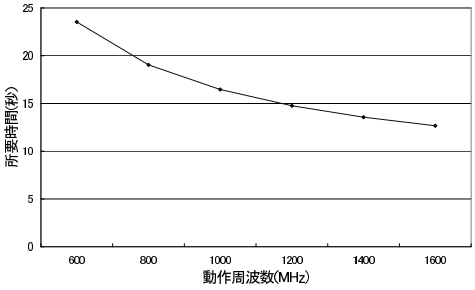

図 9　Pentium M 搭載の PC で is.A を実行した時の所要時間

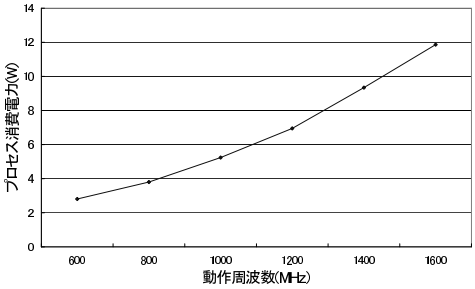

図 10　Pentium M 搭載の PC で is.A を実行した時のプロセス消費電力

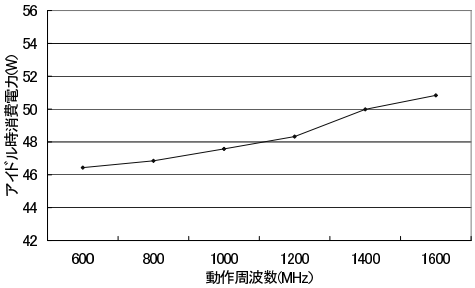

図 11　Pentium M 搭載の PC で is.A を実行した時のアイドル時消費電力

5.2　Pentium M

次に、is.A を Pentium M 搭載の PC で実行したときの動作周波数に対する所要時間、プロセス消費電力、アイドル時消費電力、プロセス消費電力量を図 9、図 10、図 11、図 12 に示す。Pentium 4 搭載の PC においては CPU の動作周波数に対してプロセス消費電力量が減少傾向にあるのに対して、Pentium M 搭載の PC においては増加傾向にあり、CPU が低電圧動作している時の電力消費効率が良いと判定できる。図 12 から分かるように、消費電力量で見た場合、Pentium M 搭載の PC では、最低動作周波数で動作させた時と最高動作周波数で動作させた時で 2 倍を越える性能差が表れている。

図??が示す通り、Pentium M 搭載の PC では、動作周波数が小さいほどアイドル時消費電力が小さくなっている点も Pentium 4 とは異なる。すなわち、動作周波数 (動作電圧) が小さいほどアイドル時消費電力が少なくなっている。

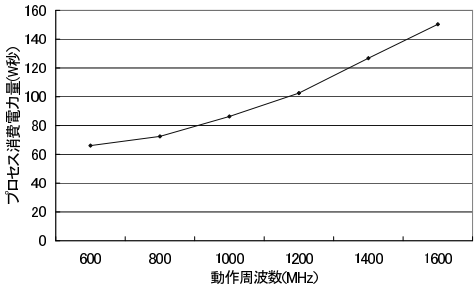

図 12　Pentium M 搭載の PC で is.A を実行した時のプロセス消費電力量

6.　単一プロセス実行による電力量の考察

今回使用した Pentium 4 搭載の PC においては、DVS を用いて電圧および動作周波数を下げるメリットはほとんどない。CPU の動作周波数が下がることで消費電力は現象しても、そのプロセスの終了が遅れて消費電力量が逆に大きくなるからである。アイドル時の消費電力が全く同じであることを考えても、Pentium 4 の実験機で動作周波数を下げるのはプロセスの動作を遅くし、より消費電力量を増やすことにつながるだけだと考えられる。室内の電源から取れる電力の上限が厳しく制限されている環境や、CPU 自体が高熱を発して危険な状態になっているときなど、電力消費を落とさなくてはならない状況であれば Pentium 4 においても DVS を用いることに意味はある。

また、CPU の動作周波数を下げれば消費電力が必ず下がっていくことも断言出来ないことが分かる。本実験の Pentium 4 の 1.5GHz における結果が示す通り、動作周波数を下げた結果、消費電力量ばかりでなく消費電力すら増加するケースが存在する。これは PC 全体の電力的な特性は DVS の理論的な結果とは必ずしも一致せず、電力管理においては常に各 PC の電力的特性を調査しなくてはならないことを示唆している。もし消費電力のみを段階的に下げようとした場合でも、単に動作周波数を順次下げていけば良いと言うわけではない。

一方 Pentium M を搭載した PC は、消費電力、消費電力量ともに低周波数の方が順番に良い値を示すという、消費電力を意識した環境では望ましい性質を有している。また、低周波数状態ではアイドル状態の電力消費も下がっていることから、アイドル時に周波数

を下げておくことで電力を節約することも可能となる。これは Pentium 4 搭載の PC では期待できないことである。ここから、Pentium M を搭載している場合には可能な限り動作周波数を下げて動作させるようシステム側の設定を行なうのが良い、という結論が得られる。なお、Linux の DVS ラッパである cpufreq モジュールは、最新のバージョンにおいて、アイドル状態に応じて動作周波数を下げるというポリシー (cpufreq 内では governor と呼ばれる) の実装として ondemand 機構を提供している。

二つの実験機の一方がデスクトップ PC であり、もう一方がラップトップ PC である点を無視してはならない。デスクトップ PC において、消費電力を意識したマザーボードやメモリ、その他の省電力デバイスを接続していることは通常期待できないが、ラップトップ PC は消費電力的に厳しい環境で使うことを中心に設計されているため、CPU 以外の側面でも省電力を意識した構成になっていると考えられる。

7. プロセス単位電力制御実験

いままでの議論から、Penitum M においては可能な限り CPU の動作周波数を下げておくのが理想的であることが分かる。処理にかかる時間が重要でなければ、動作周波数を小さくしておくことで仕事量に対する消費電力量を下げることが出来る。プロセス単位電力制御機構でも、速度が重要でないプロセスだけ低速に動作させれば良いことになる。

しかし、プロセス単位電力制御機構が有用であることを示すには、この機構を使用した際のオーバーヘッドについても考察しなくてはならない。

Linux Kernel 2.6 の x86 アーキテクチャの標準実装においては、タイマー割り込みの関係上、プロセス切替えは最短で 1 ミリ秒ごとに起こり得る。一方で、プロセス単位で電力を制御する際にはプロセス切替えの度に CPU の動作周波数を変更しなくてはならない。そのため、例え動作周波数変更に伴うオーバーヘッドが数十マイクロ秒単位であっても、システム全体では致命的な速度低下につながる可能性もある。また、1 プロセスで DVS を用いたときと CPU の電力的性質が変わる可能性もあり得る。

以降では Pentium M の DVS 機構を用いて、同じプログラムを周波数の異なる二つのプロセスで実行させ、プロセス単位電力制御機構を実装した時、そのオーバーヘッドがどの程度になるのかを検証する。

```
len=sprintf(buf, "600000");
d=open("/sys/.../cpufreq/scaling_setspeed",
       O_WRONLY);
write(d, buf, len);
...
```

図 13　CPU の動作周波数変更例

8. 設　　計

複数のプロセス毎に動作周波数を変更できるようにするため、Kernel の cpufreq モジュールを改造する．

本論文では，モジュールに含まれる動作周波数を変更する関数群を利用し，それをスケジューリング時に呼び出すよう Linux Kernel を修正する．

修正された Kernel では，スケジューリングを行なう関数 `schedule()` の最後に，速度変更をするための関数を呼ぶ．その関数は `sched_struct` に追加されたポリシー変数を参照し，実際に CPU の動作周波数 (動作電圧) を変える `cpufreq_driver_target()` 関数を呼ぶ．`cpufreq_driver_target()` 自体は cpufreq モジュールの提供する関数で、これによってドライバ毎の詳細が隠蔽される。

実装を簡略化するため、動作周波数を変更するインターフェースは cpufreq が提供するものをそのまま用いる。動作周波数は図 **13** に示されるように変更を行なうことが出来る。cpufreq の本来の動作においては、仮想ファイルに書き込むことによってシステム全体の CPU 動作周波数が設定されるが、修正されたカーネルにおいては、書き込みを行なったプロセスの CPU 動作周波数のみが再設定される。すなわち、図 13 の処理を行なって以降、このプロセスにタイムスライスが渡ったときには CPU が指定された動作周波数で動作し、その他のプロセスのときには標準の動作周波数で動作するようになる。

なお、今回の実装においてはアイドル時には必ず最低動作周波数に下がるようになっているため、アイドル時消費電力は最低動作周波数時の値になる。

9. 二つのプロセスを別周波数で実行する実験

単一プロセスの時の評価と同様、本実験でも測定対象 PC から測定用 PC へ UDP パケットを送ることで速度を測定する。今回は二つのプロセスをほぼ同時に起動し、各プロセスで動作周波数を変更後、start パケットを送信して測定用プログラムを実行する。end パケットは二つのプロセスの終了を待つ親プロセスが送信する。なお、親プロセスは即座に `wait()` を呼び出すため、親プロセスによる消費電力の影響は事実上

```
pid1 = fork();
if( pid1 == 0 ){ // Child Process 1
  freq_set( "800000" )
  send_packet("begin: 800000");
  execvp( file, argv );
}

pid2 = fork()
if( pid2 == 0 ){ // Child Process 2
  freq_set( "1600000" )
  send_packet("begin: 1600000");
  execvp( file, argv );
}

wait();
send_packet("end");
wait();
send_packet("end");
```

図 14　2 プロセス同時実行用ラッパプログラム

表 3　2 プロセス実行時の所要時間

動作周波数 (MHz,MHz)	同時実行時 (秒)	個別実行時 (秒)	差 (%)
(800,800)	38.394	38.093	0.300
(1200,800)	33.990	33.807	0.182
(1200,1200)	29.838	29.521	0.316
(1600, 800)	31.847	31.718	0.129
(1200,1600)	27.700	27.432	0.268
(1600,1600)	25.675	25.343	0.332

存在しない。

使用するプログラムは一つ目の実験と同じく NPB の is.A を用いる。使用する PC は表 1 の Thinkpad X31 を用いる。結果を簡略化するため、本実験で使用する周波数は 800MHz、1.2GHz、1.6GHz に限定する。二つのプロセスはそれぞれこの 3 種類のうち 1 つの動作周波数で動作する。そして二つのプロセスを同時に動かした場合と別々に動かした場合の所要時間とプロセス消費電力量の差を計測する。次項の結果は 10 回行なった測定の平均を取ったものである。

10. プロセス単位電力制御実験結果

結果を**表 3**、**表 4** に示す。この結果から分かる通り、プロセス切替の度に動作周波数を変えても周波数毎の所要時間、すなわち性能はほとんど変わらないことが分かる。消費電力量について、一部 5%〜10%の振れ幅があるが、これらはそもそも動作周波数を変更しないケース ((800MHz,800MHz) の (1600MHz,1600MHz)) で起きており、DVS をプロセス単位で適用したことによるものではないと考えられる。その他のケースでは、動作電圧変更を行なう場合も含めて電力量の増加は 1%前後に抑えられている。単一プロセスにおける実験で、動作周波数を下げることで 50%以上のプロセス消費電力量の削減が行なえることを示せているから、この 1%は無視することが出来る。

表 4　2 プロセス実行時のプロセス消費電力量の合計

動作周波数 (MHz,MHz)	同時実行時 (W 秒)	個別実行時 (W 秒)	差 (%)
(800,800)	177.899	160.141	11.089
(1200,800)	211.427	210.332	0.520
(1200,1200)	262.105	260.524	0.607
(1600, 800)	290.790	285.468	1.864
(1200,1600)	343.652	335.660	2.381
(1600,1600)	433.607	410.796	5.553

この実験から分かる通り、プロセス単位で動作周波数変更を行なうことによる速度低下は事実上存在しない。対話型プロセスの CPU 動作周波数を上げ、それ以外のバッチプロセスの CPU 動作周波数を下げることによって、冒頭に述べた対話型プロセスとバッチプロセスの共存に関する問題をオーバーヘッドなしに解決できることが示されたことになる。

11. 関 連 研 究

我々は論文 6) において、プロセス単位で動作周波数を制御する機構について論じ、消費電力の観点からこの機構が有用であることを示した。一方で、電力量に関する検討は行なってはおらず、また実験機の CPU が Pentium 4 であることにも問題がある。本論文はこの問題を踏まえた上で、DVS が有効に機能する Pentium M 上で同様の実験を行ない、その有効性を述べたものである。

論文 5) では本論文で取り上げた Pentium 4， Pentium M の他に Crusoe，XScale も含めた電力特性について論じられている。この研究はクラスタを構築する上で重要な熱設計や電力当たりの性能、そして低消費電力クラスタの構築に主眼をおいている。一方、本論文では既存の PC やクラスタ環境を OS のレベルで管理する上で必要となる CPU の特性の調査に主眼を置いているため、最大、最小時だけでなく、各周波数における特性やスリープ時の挙動などにも注目している。

12. ま と め

本論文では、DVS を用いてプロセス単位で電力を制御する機構の有用性を説明し、その実用可能性を検証するための二つの予備評価を行なった。まず、Pentium

4 と Pentium M の DVS 機構を用いて低電圧時の各 CPU の電力的な性質を検証した。その結果、Pentium M においては低電圧動作をさせることで電力量を減らせることを示した。また、2 つのプロセスを別々の周波数を用いて動作させた場合の電力量を測定し、プロセス単位で消費電力を制御してもそのオーバーヘッドは微々たるものであり、それ以上にシステム全体の電力量を下げることが可能であることを示した。以上の二つの予備評価から、CPU として Pentium M を搭載した計算機を用いれば、ユーザから見た応答性を落とさずに消費電力を下げるプロセス単位電力管理機構がオーバーヘッドなしに構築可能であることが立証できたと言える。

一方で、本論文では Pentium 4 と Pentium M をそれぞれ搭載した PC1 台ずつでのみ評価を行なっており、CPU の性質から導かれる結果なのか、実験機に特有の性質から導かれる結果なのかについての検討は不十分であると考えられる。今後はさらにいくつかの PC で同様の実験を行ない、プロセス単位電力制御機構の有用性についてのさらなる検討を進める。また、プロセス単位電力制御機構を用いたシステムを実際に構築することで、実利用した場合の電力性能や応答性についての検討も行なう予定である。

謝辞 本論文の一部は，科学技術振興機構 (JST) の戦略的創造研究推進事業 (CREST) の支援を受けた．

参 考 文 献

1) NAS Parallel Benchmarks. `http://www.nas.nasa.gov/Software/NPB/`.
2) AMD Corporation. Powernow! technology.
3) Intel Corporation. Enhanced intel speedstep technology for the intel pentium m processor.
4) Intel Corporation. Speedstep technology.
5) 堀田義彦, 佐藤三久, 朴泰裕, 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 2005, pp. 207–218, 2004.
6) 宮川大輔, 石川裕. プロセス単位電力制御機構の設計と実装. pp. 167–168, 2005.