2004－OS－97　(7)
2004／8／1

# 性能モニタリングカウンタによる一括システムコール機構の評価

飯尾 賢太郎 †　日下部 茂 ‡　谷口 秀夫 ††　雨宮 真人 ‡

† 九州大学大学院システム情報科学府
‡ 九州大学大学院システム情報科学研究院
†† 岡山大学工学部

本稿では、プロセッサの性能モニタリングカウンタを用いた一括システムコール機構の評価について述べる。一括システムコール機構はシステムコールをある閾値までまとめ、それから一括して処理を行うことで、モード遷移回数を減少させ、スループットの向上を測るものである。また、モード遷移回数が減少することにより、各走行モードにおける処理は集約されるため、参照の局所性が増大し、キャッシュミスや TLB ミスなどが減少することが期待できる。そこで、今回プロセッサの性能モニタリングカウンタを用いて測定を行い、その効果の評価を行う。

# A Detailed Evaluation of the Wrapped System Call with the Performance Monitoring Counter

Kentaro Iio†, Shigeru Kusakabe‡, Hideo Taniguchi††, Makoto Amamiya‡

†, ‡Graduate School of Information Scienceand Electrical Engineering,
Kyushu University
††Faculty of Engineering, Okayama University

This paper discusses the detailed evaluation of the Wrapped System Call(WSC) with the performance monitoring counter. We have proposed the WSC as a method for reducing the overhead of mode change frequency caused by system calls so that we can achieve high throughput. Moreover, we expect that the WSC reduces the number of mode change and it help to make the locality of references and higher ratio of cache hits and TLB hits. We measure the performance of the WSC with the performance monitoring coutner and evaluate it.

## 1　はじめに

通常、応用プログラムはユーザプログラムとカーネルプログラムとが連携をして処理を行う。これは、ファイル操作や通信処理などは通常ユーザプログラムでの処理が許されず、カーネルプログラムが処理を行う必要があるためである。ユーザプログラムがカーネルプログラムに対して処理を依頼する場合、ユーザプログラムはシステムコールを実行する。しかし、システムコールを実行する際にはパイプラインのフラッシュや、走行モードの遷移といった処理を行う必要があるため、オーバヘッドが生じる。そのため、性能向上のためにより高速なプロセッサを導入する場合においても、応用プログラムが頻繁にシステムコールを実行する場合は、そのオーバヘッドのため性能が向上しにくい。また、現在プロセッサの動作速度と、メインメモリの動作速度の乖離が

進んでおり、現在の汎用プロセッサの多くは性能向上のためにキャッシュメモリや TLB といったメモリ階層を用いている。メインメモリに比べ、キャッシュメモリは高速だが容量が小さい。しかし、システムコールを実行する際には、ユーザプログラムからカーネルプログラムへと処理が移ってしまうため、アクセスするメモリ領域が大きく変わってしまう。そのため、参照の局所性が低下してしまい、メモリ階層の利用効率も低下してしまう。

そこで、我々は上記に挙げたシステムコールに伴う問題を解決するため、一括システムコール機構 [1] を提案している。一括システムコール機構とはシステムコールの内容を複数個分まとめておき、一定数のシステムコールがまとまった段階でカーネルプログラムを呼び出す機構である。システムコールの内容をまとめる段階ではカーネルプログラムの呼び出しが行われないため、カーネルプログラムの呼び出し回数を削減することが可能となる。また、システムコールは一定数に達するまで実行されないため、ユーザプログラムの処理とカーネルプログラムの処理がそれぞれ集約される。そのため、参照の局所性が向上し、メモリ階層を有効に利用できることが期待できる。

一括システムコール機構を利用する場合、システムコールを実行しても内容を格納するのみであるため、ユーザプログラムの処理が続けて行われる。しかし、システムコールの結果を受けて処理を行う箇所は実行できないため、スケジューリングを行う必要がある。現在、我々は並列分散 OS Communication-Execution Fusion Operating System (CEFOS)[2] におけるマルチスレッド実行環境を用いて、一括システムコール機構を実現している。

本稿では、性能モニタリングカウンタを用いることによる一括システムコール機構の評価について述べる。現在までに、システムコールを多用するプログラムにおいては、一括システムコール機構を用いることにより、処理時間が短縮されることが分かっている [1]。今回は、一括システムコール機構の有効性をより詳細に分析するために、特にメモリ階層に着目して測定を行う。しかし、キャッシュミスや TLB ミスなどのメモリ階層で発生するイベントはソフトウェアからは分からないため、ソフトウェアのみではメモリ階層への効果を測定することは困難である。そこで、今回は性能モニタリングカウンタを用いる。性能モニタリングカウンタはキャッシュミスや TLB ミス、メモリアクセス回数などの、通常ソフトウェアでは認識できないプロセッサ内部で発生するイベントを測定するためのカウンタであり、現在の多くの汎用プロセッサがこの機能を提供している。今回の測定では、最も正確な測定が行えることが期待できる NetBurst アーキテクチャのプロセッサを用いた。NetBurst アーキテクチャでは Precise Event Based Sampling と呼ばれるキャッシュミスの発生をイベントとして扱い、そのイベントを契機として測定を行う機能や、At Retirement 計測と呼ばれる、投機的実行の結果コミットされた命令によって発生したイベントとコミットされずに破棄された命令によって発生したイベントを区別して計測する機能がある [3]。

本論文の構成を以下に示す。まず、第 2 節で一括システムコール機構の概要について述べる。次に、第 3 節で一括システムコール機構の評価について述べる。そして、第 4 節で今後の課題について述べ、最後に、第 5 節でまとめを述べる。

## 2　概要

本節では、まず一括システムコール機構の実行環境である CEFOS についての概要を述べ、次に一括システムコール機構の概要を述べる。

### 2.1　CEFOS

CEFOS は従来のスレッドよりもさらに粒度の小さな細粒度スレッドをプロセッサ割り当て単位とし、細粒度マルチスレッド実行環境を提供する。CEFOS では、通信や入力などにより外部から非同期に割り込みが発生した場合においても、割り込みを行うスレッドを別に生成することで効率よく処理を行う。また、処理結果待ちなどにより遅延が生じる箇所はあらかじめスプリット・フェーズとして複数のスレッドに分割し、結果待ちを行っている間は他の実行可能なスレッドに対してプロセッサ割り当てを行うことで遅延の隠蔽を図る。

CEFOS におけるマルチスレッド実行方式は、同期をプリミティブとして実現される。スレッドは実行終了時に、継続スレッドに対して同期をとる。必要な同期を全て受けたスレッドは実行可能となり、プロセッサ割り当てを待つ。ただし、スレッドが細粒度であるため、スレッド切り替えとスレッドスケジューリングが頻発すると予想される。そのため、スレッドスケジューラはユーザプログラム内におかれる。

### 2.2 一括システムコール

一括システムコール機構は複数のシステムコールをまとめて1つのシステムコールとして扱い、カーネルプログラムで処理を行う。図1に一括システムコール機構を用いた場合の処理の流れを示し、以下で説明する。

1. ユーザプログラムとカーネルプログラムは共有メモリ域を持つ。スレッドは共有メモリにシステムコールの内容を格納し、終了する。

2. スレッドスケジューラは共有メモリ域に格納されているシステムコールの数が一定数に達した場合、カーネルプログラムを呼び出す。そうでない場合、次のスレッドを実行する。

3. カーネルプログラムは共有メモリ域に格納されているシステムコールの内容を順次読み出して、内容に従い実行する。

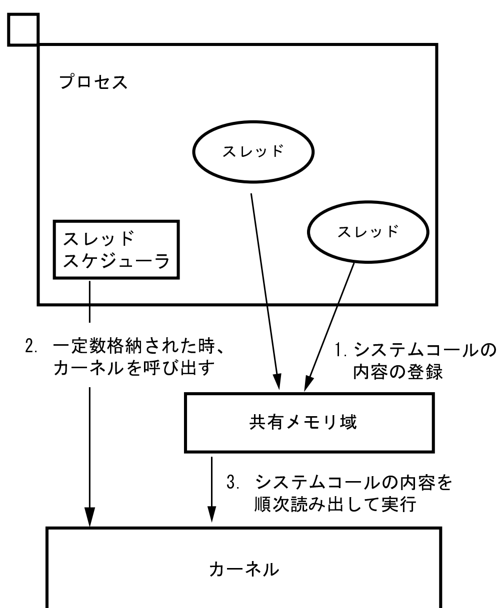


図 1: 処理の流れ

現在、一括システムコール機構の評価はLinux上実装したCEFOSのマルチスレッド実行環境上で行っている。そのため、図中のプロセスはLinuxのプロセスを示す。

システムコールの内容を共有メモリ域に格納および読み出しに必要な処理時間を$S$、システムコールの受付処理および復帰処理に必要な処理時間を$A$、$S$、$A$以外の処理時間を$O$、一括するシステムコールの個数を$N$とする。

この場合、従来のシステムコールを用いてN個のシステムコールを実行する際の処理時間$T_1$は、$T_1 = O_1 + NA$と表せる。また、一括システムコール機構を用いてN個のシステムコールを実行する際の処理時間$T_2$は、$T_2 = O_2 + NS + A$と表せる。よって、処理時間の差は$T_1 - T_2 = (O_1 - O_2) - N(S - A) - A$となる。このうち、$(O_1 - O_2)$は一括システムコール機構を用いることにより、キャッシュミス率などが変化したことによるものであり、値は一定ではないため、これを$\Delta O$とおく。この時、一括システムコール機構を用いることにより減少する処理時間$\Delta T$は、

$$\Delta T = T_2 - T_1 = N(S - A) + A - \Delta O \qquad (1)$$

となる。そのため、まとめるシステムコールの個数$N$が大きいほど、より高い効果を得ることが可能となる。しかし、$N$が大きすぎる場合システムコールの内容を共有メモリ域に格納してから、実際に処理されるまでの遅延が大きくなってしまうため、応答性が必要な場合は$N$の大きさに注意する必要がある。

また、$S$、$A$の値はほぼ一定である。これは処理内容がほぼ一定であるためである。一方、$\Delta O$の値は大きく変化する可能性がある。これはキャッシュミスの発生率がそれ以前にどのような処理を行うかに大きく影響されるためである。

## 3 一括システムコール機構の評価

本節では、テストプログラムを用いることにより、一括システムコール機構の効果の測定を行う。測定には性能モニタリングカウンタを用い、メモリ階層に与える効果について調べ、測定結果をもとに詳細な評価を行う。

### 3.1 測定内容

応用プログラムはユーザプログラム内の処理とカーネルプログラム内の処理を交互に行うが、その処理内容はメモリからデータを読み出し、そのデータを用いて処理を行い、データをメモリに書き込むことの繰り返しである。メモリアクセスを行う際、データがキャッシュ上に存在する場合はキャッシュアクセスのみで済むため、メインメモリに対するアクセスは行われない。この際、キャッシュはメインメモリに比べて高速であるためレイテンシは小さ

い。しかし、キャッシュはメインメモリに比べて小容量であるため、メモリアクセスを行う範囲が広くなるとキャッシュミス率も大きくなり、スループットは低下する。これは TLB に対しても同様である。

一括システムコール機構を用いる場合、ユーザプログラム内の処理とカーネルプログラム内の処理がそれぞれ集約されるため参照の局所性が向上し、メモリ階層の利用効率が向上することが期待できる。しかし、ユーザプログラム内の処理とカーネルプログラム内の処理で、それぞれどの程度の範囲に対してメモリアクセスを行うかは、応用プログラムごとに大きく異なる。そこで、さまざまな範囲に対してメモリアクセスを行うテストプログラムを作成し、このテストプログラムを用いてどのような範囲でメモリアクセスを行う場合に一括システムコール機構が効果的であるかを測定する。今回は処理に要したクロック数、L2 キャッシュミス率、DataTLB ミス率に対して測定を行った。

テストプログラムはユーザプログラム内とカーネルプログラム内のそれぞれで、一定範囲に対してメモリアクセスを行うことを交互に繰り返す。また、メモリアクセスを行う箇所を評価用関数として実装し、ユーザプログラム内で実行されるユーザ関数、およびシステムコールを通じてカーネルプログラム内で実行されるシステムコール関数としてこの評価用関数を用いた。評価用関数の処理の流れを図 2 に示す。

評価用関数は引数としてメモリアクセス範囲 (width) とメモリアクセス間隔 (stride) を取り、データの読み出し先の配列の先頭から、width で指定した範囲の値を、stride で指定した間隔で読み出し、その総和を取る。データの読み出し先の配列はユーザ関数で用いるものはユーザ空間に、システムコール関数で用いるものはカーネル空間におかれ、ページ境界でアライメントされている。

また、テストプログラムの処理の流れを図 3、図 4 に示す。図 3 は一括システムコール機構を用いない従来のシステムコールを用いた場合の処理の流れを、図 4 は一括システムコール機構を用いた場合の処理の流れを示す。

テストプログラムでは評価用関数をユーザ関数、システムコールとして交互に呼び出す。今回の測定では、評価用関数に与える引数は width を 0KB から 256KB までを 4KB 間隔で、また stride を 128B とした。また、評価用関数を呼び出す処理を繰り返す回数 $F$ は 512 とした。1 回の処理でユーザ関数として 1 回、システムコール関数として 1 回呼び出されるため、評価用関数自体は 1024 回実行される。なお、一括するシステムコールの内容の個数 $N$ は 32 とした

第 2.2 節の式 (1) における S は図 4 中の「システムコールを呼び出す評価用関数の内容を共有メモリ域に格納」および図 4 中の「システムコールの内容を共有メモリ域から読み出す」に相当し、A は図 3、4 中の「システムコール前処理」および「システムコール後処理」に相当する。また、破線「カーネルプログラム」で挟まれた箇所はカーネルプログラム内で実行され、それ以外の箇所はユーザプログラム内で実行される。

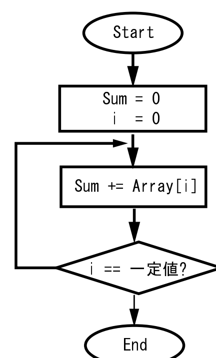


図 2: 評価用関数

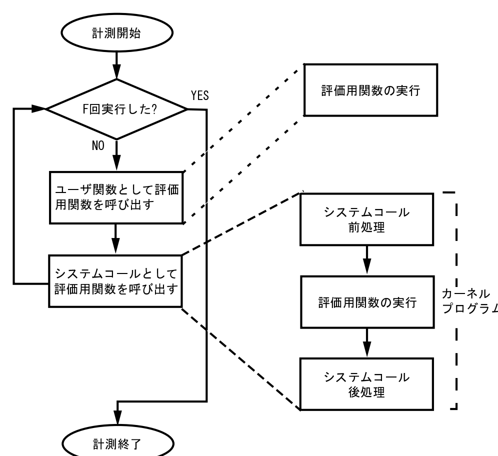


図 3: テストプログラム (従来)

### 3.2 測定環境

今回の測定には、以下のプロセッサを用いた。

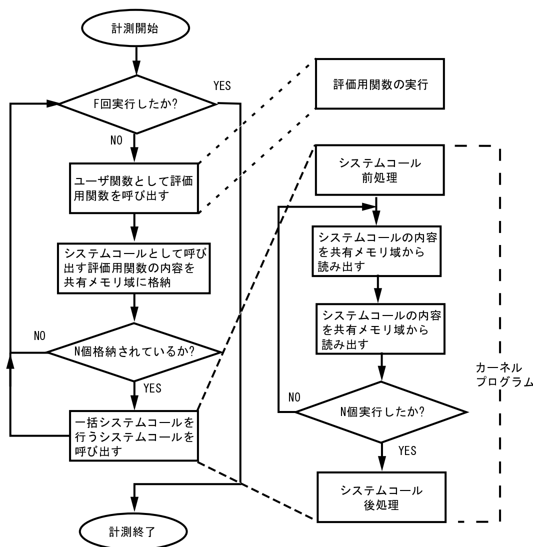


図 4: テストプログラム (一括)

- Pentium4 1.8GHz (L2 キャッシュ 256KB)
- Pentium4 1.8AGHz (L2 キャッシュ 512KB)

一括システムコール機構を用いることにより、ユーザプログラム内の処理とカーネルプログラム内の処理がそれぞれ集約され、参照の局所性が向上することが期待できると述べた。そこで、キャッシュサイズの違いにより一括システムコール機構の効果にどのような違いが生じるかを調べるため、L2 キャッシュサイズの異なるプロセッサを測定に用いた。

また、OS には Linux 2.4.20 を用い、一括システムコール機構を実装した。また、今回性能モニタリングカウンタを利用するためのフロントエンドとして、hardmeter を用いた [4]。

### 3.3 測定結果

#### 3.3.1 処理に要するクロック数

処理に要するクロック数の比を図 5、図 6 に示す。凡例中の値は従来のシステムコールを用いた場合を 1 とした場合の一括システムコール機構を用いた場合の結果であり、値が小さいほど一括システムコール機構の効果は高い。

図 5、図 6 のいずれにおいても、ユーザプログラム内でのアクセス範囲とカーネルプログラム内でのアクセス範囲の和がおよそ 0KB から 64KB までの範囲 (範囲 X) と、およそ 288KB から 448KB までの範囲 (範囲 Y) で高い効果が得られていることが分かる。

範囲 X については、アクセス範囲の和が小さいほど効果が大きくなり、0KB の場合におよそ 40%にまで減少する。アクセス範囲の和が小さい場合、キャッシュや TLB の容量に余裕があるためキャッシュミスや TLB ミスなどがほとんど発生せず、処理に要するクロック数の値自体が小さくなる。そのため、式 (1) の $N(S-A)$ が処理に要するクロック数に対して占める割合が大きくなるため、一括システムコール機構の効果が高い。

また範囲 Y については、アクセス範囲の和も大きくなっているにも拘らず、処理に要するクロック数は最小でおよそ 60%にまで減少している。アクセス範囲の和が大きい場合、キャッシュミスや TLB ミスが増大するため、処理に要するクロック数自体も大きくなる。そのため、式 (1) の $N(S-A)$ が処理に要するクロック数に対して占める割合は小さくなる。よって、$\Delta O$ が大きくなったため、一括システムコール機構の効果が高まったと予想される。範囲 Y についての詳細は第 3.3.2 節で述べる。

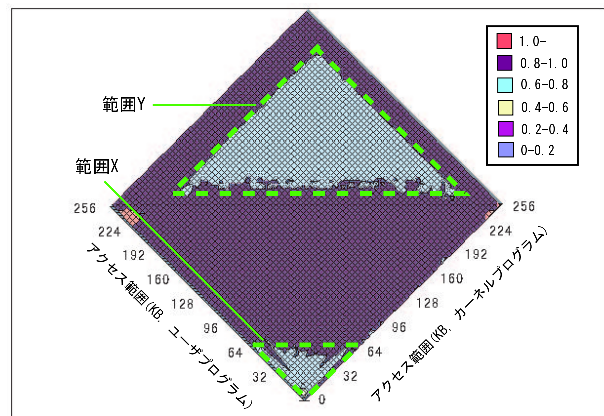


図 5: 処理に要したクロック数の比 (Pentium4 1.8GHz)

#### 3.3.2 DataTLB ミス率

DataTLB ミス率の比を図 7、図 8 に示す。凡例中の値は従来のシステムコールを用いた場合を 1 とした時の一括システムコール機構を用いた場合の結果であり、値が小さいほど一括システムコール機構の効果は高い。

いずれのプロセッサにおいても、3.3.1 における範囲 Y を中心として DataTLB ミス率の比がおよそ 50%以下にまで減少していることが分かる。この理由として、ユーザプログラム内の処理とカー

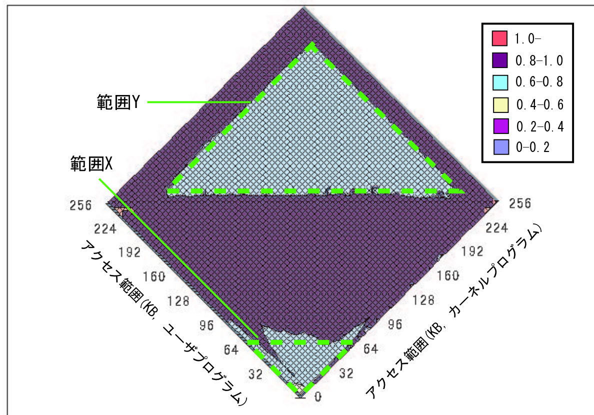


図 6: 処理に要したクロック数の比 (Pentium4 1.8AGHz)

ネルプログラム内の処理におけるアクセス範囲と、DataTLB のエントリ数との関係が挙げられる。ユーザプログラム内におけるアクセス範囲を $U$、カーネルプログラム内におけるアクセス範囲を $K$ とした場合、評価用関数をそれぞれのプログラム内で 1 回ずつ実行した際のアクセス範囲は $U + K$ となる。今回測定に用いたプロセッサの DataTLB は 64 エントリ・フルセットアソシアティブであり、またページサイズは 4KB であるため、4KB × 64 エントリ = 256KB を $U + K$ が超えた場合、ユーザプログラムとカーネルプログラムの間で競合が起こり、DataTLB ミスが発生する。しかし、一括システムコール機構を用いた場合、システムコールがまとめられている間はユーザプログラム内の処理が連続して行われ、システムコールが一括して処理されている間はカーネルプログラム内の処理が連続して行われる。そのため、連続して処理を行っている間は競合が起こる条件が $U >$ 256KB、もしくは $K >$ 256KB となる。従って、$U \leq$ 256KB、かつ $K \leq$ 256KB、かつ $U + K >$ 256KB である場合、一括システムコール機構を用いることにより競合が解消され、DataTLB ミスの発生は大きく減少する。これにより、式 (1) のうち $\Delta O$ の値が大きくなり、処理に要するクロック数が大きく減少している。

逆に、$U+K \leq$ 256KB である場合には、DataTLB ミス率は増大している。これは DataTLB に収まる範囲内でアクセスを行っているため、ユーザプログラムとカーネルプログラムとの間に競合は起こらないためである。このような場合、システムコールを格納するためのメモリ領域を必要とする分、一括システムコール機構を用いた場合の方が DataTLB ミス率は高くなる。

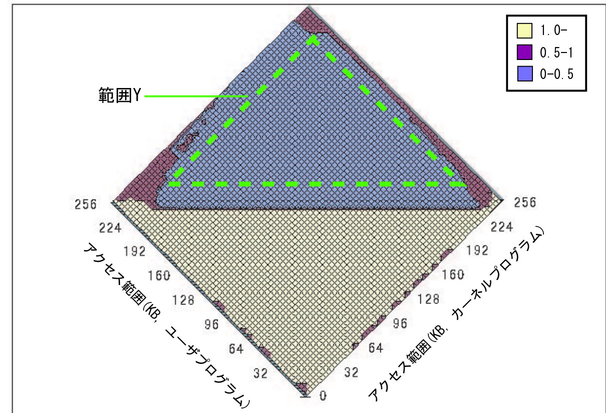


図 7: DataTLB ミス率の比 (Pentium4 1.8GHz)

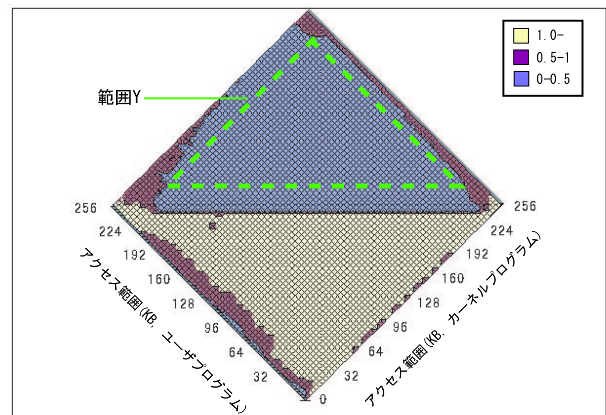


図 8: DataTLB ミス率の比 (Pentium4 1.8AGHz)

### 3.3.3 L2 キャッシュミス率

L2 キャッシュミス率の比を図 9、図 10 に示す。凡例中の値は従来のシステムコールを用いた場合を 1 とした時の一括システムコール機構を用いた場合の結果であり、値が小さいほど一括システムコール機構の効果は高い。

図 9 より Pentium4 1.8GHz の場合、DataTLB ミス率の比と同様に、範囲 Y を中心として L2 キャッシュミス率の比がおよそ 50%以下にまで減少していることが分かる。この理由として、Pentium4 1.8GHz の L2 キャッシュサイズは 256KB であるため、DataTLB と同様のことが考えられる。

Pentium4 1.8GHz における L2 キャッシュミス率の比において特徴的なのは、$U + K \leq 256$KB である場合においても、L2 キャッシュミス率の比が概ね減少していることである。この理由として、今回測定に用いたプロセッサの L2 キャッシュは 8 ウェイ・セットアソシアティブであるため、メモリアクセス範囲が狭い場合においても、キャッシュラインの衝突によりキャッシュミスが発生することが考えられる。そのため、従来のシステムコールを用いた場合はキャッシュラインの衝突によってキャッシュミスが頻発したと予想される。

図 10 に示す結果は図 9 に示す結果とは大きく異なる。これは、Pentium4 1.8AGHz の L2 キャッシュサイズが 512KB と大きいため、$U + K \leq 512$KB である場合は L2 キャッシュミス率は低いためである。ただし、$192\text{KB} \leq K \leq 224\text{KB}$ においては L2 キャッシュミス率がおよそ 50%以下にまで減少していることが分かる。これは、L2 キャッシュの容量自体に余裕があるため、Pentium4 1.8GHz の場合と同様、従来のシステムコールを用いた場合に、キャッシュラインの衝突によるキャッシュミスが頻発したと考えられる。

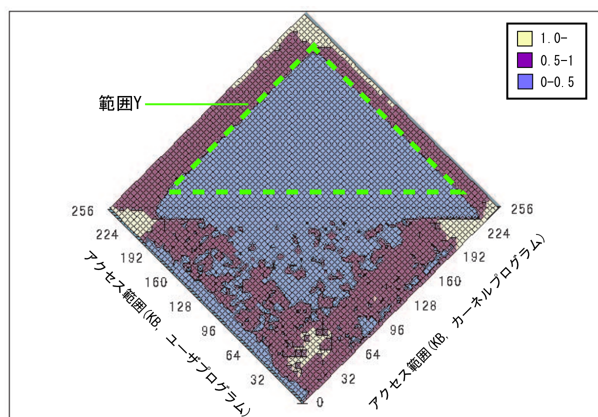


図 9: L2 キャッシュミス率の比 (Pentium4 1.8GHz)

## 4 今後の課題

今後の課題のひとつとして、今回測定に用いた NetBurst アーキテクチャ上において、SYSENTER 命令を利用した場合の測定を行うことが挙げられる。

SYSENTER 命令は特権レベル 0 のシステム・プロシージャへの高速コールを行うための命令で、XP 以降の Windows や 2.5.52 以降の Linux などの近年の OS カーネルでも利用されている。SYSENTER

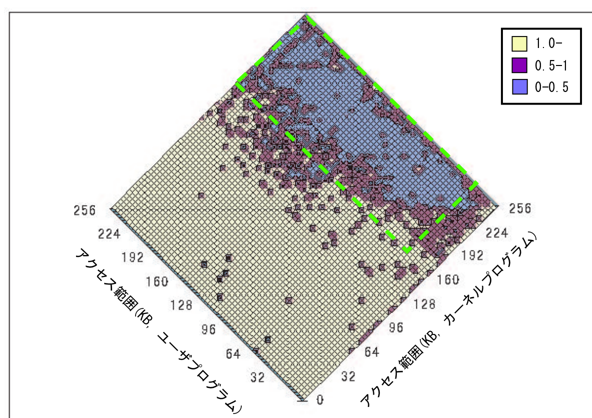


図 10: L2 キャッシュミス率の比 (Pentium4 1.8AGHz)

命令はディスクリプタテーブルの参照を省略することが可能であるため、INT 命令を用いて走行モード遷移を行う場合に比べオーバヘッドが小さい。そのため、SYSENTER 命令を用いた場合には式 (1) のうち、$A$ の値が小さくなることが予想される。ここでは、SYSENTER 命令を用いることにより走行モード遷移のオーバヘッドがどの程度小さくなるかを確認するため、従来のシステムコールと一括システムコール機構の両方で、INT 命令と SYSENTER 命令のそれぞれの命令を用いた場合で getpid システムコールを 512 回実行し、処理に要するクロック数の測定を行う。getpid システムコールはプロセス ID を取得するのみの単純な命令であるため、走行モード遷移に伴うオーバヘッド削減の効果を確認するのに適している。

図 11 に結果を示す。結果は Pentium4 1.8AGHz での測定結果のみであるが、Pentium4 1.8GHz においても同様の結果が得られた。凡例中、従来 (SYSENTER) は SYSENTER 命令と従来のシステムコールを用いた場合の結果を、一括 (SYSENTER) は SYSENTER 命令と一括システムコール機構を用いた場合の結果を、従来 (INT) は INT 命令と従来のシステムコールを用いた場合の結果を、一括 (INT) は INT 命令と一括システムコール機構を用いた場合の結果をそれぞれ表す。また、グラフの横軸は一括するシステムコールの個数 $N$、縦軸は処理に要するクロック数を示す。

結果より、INT 命令を用いた場合は一括システムコール機構は $N$ を 2 以上とすることにより処理に

要するクロック数は減少するが、SYSENTER 命令を用いた場合は $N$ を 32 以上まで大きくする必要があることが分かる。また、従来のシステムコールにおいて INT 命令を用いた場合に比べ、SYSENTER 命令を用いた場合、処理に要するクロック数がおよそ 30%にまで減少していることが分かる。一方、一括システムコールにおいては結果がほとんど変化していない。これは SYSENTER 命令を用いることにより式 (1) のうち $A$ が小さくなり、$N(S-A)$ の値が小さくなっているためと考えられる。$S$、$A$ の値はほぼ一定であるため、$N(S-A)$ を大きくするためには $N$ を大きくする必要がある。しかし、$N$ を大きくした場合、システムコールの内容を共有メモリ域に格納してから実際に処理されるまでの遅延が大きくなるなどの問題が残る。

このように、SYSENTER 命令を用いることにより、システムコールに伴うオーバヘッドは大きく削減される。そのため、一括システムコール機構の効果を活かすために、残る $\Delta O$ を大きくする必要がある。そこで、今後は格納されたシステムコールを一括実行する際に実行順をメモリ階層を効率よく利用できるように並べ替えを行うなど、一括システムコール機構を用いてメモリ階層を有効利用するための手法を検討し、メモリ階層の有効利用の面から一括システムコール機構の効果が得られるようにする必要がある。

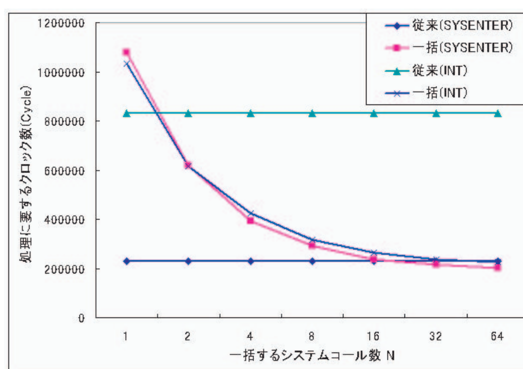


図 11: SYSENTER 命令と INT 命令の比較

その他の課題として、今回の測定で一括システムコール機構の効果が得られるようなメモリアクセス範囲が判明したため、どのような実アプリケーション上でこの効果が得られるかを調べる必要がある。また、今回の測定では NetBurst アーキテクチャのプロセッサで測定を行ったが、その他のアーキテクチャのプロセッサ上における測定を行う必要がある。

## 5 おわりに

本稿では、一括システムコール機構がメモリ階層に与える効果について、性能モニタリングカウンタを用いて詳細な測定を行った。

テストプログラムを用いてさまざまな範囲でメモリアクセスを行わせることにより、従来のシステムコールを用いた場合と一括システムコールを用いた場合のいずれの場合にもキャッシュミス率や DataTLB ミス率が低い場合には高い効果が得られることを示した。また、従来のシステムコールを用いた場合に、ユーザプログラムとカーネルプログラム間で競合を起こしている場合においても、一括システムコール機構を用いることによりその競合を解消し、性能が大きく向上する場合があることを示した。

今後も、プロセッサの動作周波数とメインメモリの動作周波数の乖離は続くと予想されるため、メモリ階層を有効に利用することは性能向上を図る上で特に重要であると考えられる。そのため、ユーザプログラム内の処理とカーネルプログラム内の処理を集約させ、メモリ階層の利用効率を向上させる一括システムコール機構は今後も有効であると予想される。

## 参考文献


[1] 谷口 秀夫, 日下部 茂, 中山 大士, 乃村 能成, 雨宮 真人. “OS の処理を多く含む並列処理の効率化を指向した一括システムコール機能.” 情報処理学会論文誌, Vol.44 No.SIG1(HPS6) pp.81-92, 2003.

[2] 雨宮 真人 他, 通信・放送機構 (TAO) 研究成果報告書, 「情報通信網の基盤技術に関する研究」, 平成 15 年 3 月

日下部 茂, 富安 洋史, 村上 和彰, 谷口 秀夫, 雨宮 真人, “並列分散オペレーティングシステム CEFOS(Communication-Execution Fusion OS),” 信学技報, CPSY99-50, Vol.99, No.251, pp.25-32 (1999).

[3] IA-32 インテル®アーキテクチャ・ソフトウェア・デベロッパーズ・マニュアル

[4] 吉岡 弘隆, “メモリプロファイリングツール (hardmeter) の設計と実装” Linux Conference 2003, October 2003