Bulletin of Networking, Computing, Systems, and Software – www.bncss.org, ISSN 2186–5140
Volume 3, Number 1, pages 83–85, January 2014

Proposal of Less Trap Architecture with Better Cache Utilization

Motoharu Nakajima

College of Information Science
University of Tsukuba
Tsukuba, Ibaraki, JAPAN
s1011426@coins.tsukuba.ac.jp

Shuichi Oikawa

Division of Information Engineering
Faculty of Engineering, Information and Systems
University of Tsukuba
Tsukuba, Ibaraki, JAPAN

***Abstract*—Hardware and software interrupts negatively affect system performance primarily because of pipeline flushing and pollution of TLB and caches. We propose a new architecture that focuses on the performace degradation caused by hardware and software interrupts. To reduce this performance degradation utilizing TLB, data and instruction caches effectively, our proposed architecture includes three proposals: clock interruption reduction with scheduler processor, a syscall thread of FlexSC [2] on scheduler processor, and non-busy waiting polling by non-blocking system calls on the syscall thread of FlexSC.**

I. はじめに

デバイスからの割り込みが性能に与える影響は大きい [1]。デバイスからのハードウェア割り込みは命令パイプラインのフラッシュを引き起こし、ユーザプロセスの実行からカーネル内の実行へとコンテキストスイッチを行い、TLB、データキャッシュ、命令キャッシュ等のキャッシュ機構を汚染していく。システムコールの発行によるソフトウェア割り込みでもこれは同様である。そこで、この論文ではハードウェア割り込み、ソフトウェア割り込みによるパフォーマンスの低下を最小限に抑える為に、発生するハードウェア割り込み、ソフトウェア割り込みの回数を低減し、なおかつ TLB やキャッシュメモリの汚染を抑えることに主眼を置いたアーキテクチャを提案する。

ハードウェア割り込み、ソフトウェア割り込みの回数を抑える為に、また、TLB やキャッシュの汚染を防ぐ為に、提案するアーキテクチャではこれから述べる三点を実装し、より良いパフォーマンスを達成する。

その一つ目はスケジューラプロセッサによるクロック割り込み数削減である。クロック割り込みを受け取るプロセッサを一つに固定し、そのプロセッサが他のプロセッサのスケジューリングを行い、プロセッサ間割り込みでプロセス切り替えのタイミングを通知するというものである。クロック割り込みを受け取るプロセッサは一つだけであり、このプロセッサがクロック割り込み毎の仕事やその他のカーネルの仕事を行うことによって、クロック割り込みを受け取るプロセッサはカーネル空間のキャッシュを有効に利用することができ、その他のプロセッサはユーザ空間のキャッシュを有効に利用することができる。

二つ目は FlexSC[2] のシステムコール処理による命令パイプラインのストール、TLB、キャッシュメモリの汚染の回避である。[2] では、システムコールをバッチすることで非同期にシステムコールを処理することを可能にし、現在実行中のプログラムの別スレッドの実行を継続することでキャッシュミスを削減して全体の実行効率を高める FlexSC という手法が提案されている。カーネルスレッドがユーザプロセスに要求されたシステムコールの処理を非同期的に行うことによってシステムコールの発行による命令パイプラインのフラッシュが無くなり、ユーザプロセスはシステムコールの発行のタイミングで自身のプロセス中のスレッド切り替えを行うことによってユーザ空間のキャッシュを効率的に利用することができる。また、このシステムコールの処理をするカーネルスレッドをクロック割り込みを受け取るプロセッサで行うことによってこのカーネルスレッドにおいてもカーネル空間のキャッシュを有効に利用することができる。

三つ目は FlexSC の syscall スレッドにおけるノンブロッキング処理による非ビジーウェイトポーリングである。FlexSC で提案されているモデルにおいてシステムコールを非同期的に実行するカーネルスレッドを syscall スレッドと言う。この論文では、システムコールの処理のみを扱うスレッドという特殊な性質を持つ syscall スレッドの I/O 処理における複数の方式について議論し、システムコールの処理中のデバイスコマンドの発行からデバイスにおける処理が完了するまでの時間に単一のシステムコールに対してビジーウェイトをすることもなく、ブロック、コンテキストスイッチをすることもない新たな非ビジーウェイトポーリングという手法を提案する。

この論文では以上の三点についてこれから続く節で詳しく論じていく。

II. スケジューラプロセッサによるクロック割り込み数の削減

パフォーマンスに影響を与えるデバイスからのハードウェア割り込みにはクロック割り込みも例外ではなく含まれる。クロック割り込みの度に命令パイプラインがフラッシュされ、コンテキストスイッチが発生し、ユーザ空間における処理とカーネル空間における処理が交互に発生することによって TLB、データキャッシュ、命令キャッシュ等のプロセッサのキャッシュ機構が汚染されていく。命令パイプラインのフラッシュは命令パイプラインのストールを引き起こし、キャッシュミスは多くのクロックサイクルを要するが故にプログラムの実行速度に影響を与えてしまう。そこで、我々はクロック割り込みの回数を最小限に抑え、プロセッサのキャッシュ機構を効率的に利用することのできるモデルを提案する。

各プロセッサがクロック割り込みを受け取り、クロック割り込み毎にクロック割り込み毎の処理を行い、スケジューリングも行う従来の方式においてはクロック割り込みの度

に命令パイプラインがストールし、ユーザ空間とカーネル空間の間の処理の遷移によってプロセッサのキャッシュ機構が汚染され、キャッシュミス頻度が上がり、パフォーマンスが低下してしまう。この問題を解決するための方針として、カーネル空間における処理とユーザ空間における処理をプロセッサによって可能な限り分けるという方法が考えられる。このような方式では、カーネル空間における処理に特化したプロセッサにおいてはプロセッサのキャッシュ機構をカーネル空間の命令やデータのみで満たすことが可能であり、ユーザプロセスの処理に特化したプロセッサにおいてはプロセッサのキャッシュ構造を満たす命令やデータはユーザ空間のものが大部分を占め、カーネル空間への処理への遷移とカーネル空間での処理時間を最小限にすることによってプロセッサのキャッシュ機構の汚染を最小限に抑えることができ、またキャッシュミスも抑えられ、各プロセッサのキャッシュ機構をそれぞれ効率的に利用することが可能である。

また、各プロセッサにおけるクロック割り込みにおける、各プロセスに割り当てられた CPU 時間を使い切る前に発生するクロック割り込みによるユーザ空間の処理からカーネル空間の処理へのコンテキストスイッチによるプロセッサのキャッシュ構造の汚染は、単一のプロセッサのみがクロック割り込みを受け取り、他のプロセッサのスケジューリングを行い、他のプロセスのプロセス切り替えを発生させる為のプロセッサ間割り込みを行うことにより抑制することができる。そして、このクロック割り込みを受け取る単一のプロセッサがクロック割り込み毎の処理やその他のカーネル空間における処理をすることによってカーネル空間の処理に特化したプロセッサとなることで、プロセッサをカーネル空間における処理に特化したプロセッサとユーザ空間における処理に特化したプロセッサに分けることができ、ユーザ空間における処理とカーネル空間における処理の遷移を最小限に抑えることができ、プロセッサのキャッシュ構造を効率的に利用することができる。加えて、後述する FlexSC の syscall スレッドをクロック割り込みを受け取るプロセッサで実行することで更にカーネルにおける処理をこのプロセッサにまとめることができる。

ユーザプロセスがコンテキストスイッチするのはクロック割り込みのタイミングだけではない。プロセスに割り当てられた CPU 時間を使い果たす前にプロセスが終了する場合やブロックする場合には、クロック割り込み無しでコンテキストスイッチが発生し、カーネル空間においてスケジューラの実行に移り、プロセス切り替えを行う必要がある。この場合は我々が提案するプロセッサをカーネル空間の処理に特化したプロセッサとユーザ空間の処理に特化したプロセッサに分ける方式においてもユーザ空間の処理に特化したプロセッサがカーネル空間の処理へコンテキストスイッチしてスケジューラの処理をする必要がある。

III. FlexSC のシステムコール処理によるキャッシュ最適化

システムコールの発行によるソフトウェア割り込みもデバイスによるハードウェア割り込みと同様に命令パイプラインのフラッシュ、ユーザ空間の処理とカーネル空間の処理の遷移によるプロセッサのキャッシュ機構の汚染を引き起こし、パフォーマンスの低下を招く [2]。そこで、[2] では、システムコールをバッチすることで非同期にシステムコールを処理することを可能にし、現在実行中のプログラムの別スレッドの実行を継続することでキャッシュミスを削減して全体の実行効率を高める FlexSC という手法が提案されている。

その提案されている手法では、syscall スレッドというカーネルスレッドがシステムコールの処理を行う。システムコールを発行したいユーザスレッドは syscall ページという共有ページにシステムコール番号や引数等を書き込み、syscall スレッドが syscall ページを調べることによってシステムコールの要求を受け取り、システムコールの処理を行い、システムコールの結果を syscall ページに格納する。ユーザスレッドは syscall ページにシステムコールを書き込んだ後も実行を続けることができ、少なくともそのプロセス内のスレッドが全てシステムコールの結果を待つ必要がある状態になるまではそのユーザ空間で実行を続けることができる。

これにより、従来のようにシステムコール発行することでカーネル中の処理に移行すること無しに現在実行しているプロセスの実行を可能な限り長く実行することができ、アクセスするデータや命令がその間そのプロセスのユーザ空間内に限定されることによってプロセッサのキャッシュ機構を効率的に使用できることになる。また、syscall スレッドにおいてソフトウェア割り込みを使わずにシステムコールの処理を行うことが可能であり、この場合パイプラインのフラッシュも発生することは無く、コンテキストスイッチも不要である。それに加えて、前述のクロック割り込みを受け取り、カーネル空間内の処理に特化したプロセッサで syscall スレッドを実行することにより、このプロセッサに対してアクセスするデータや命令をカーネル空間内のものに限定し、その他のユーザ空間の処理に特化したプロセッサでこの syscall スレッドによるカーネル内の処理を行わないことによりカーネル空間内の処理に特化したプロセッサとユーザ空間内の処理に特化したプロセッサの双方でより良いキャッシュ効率が達成される。

IV. 非ビジーウェイトポーリング

この節では FlexSC の syscall スレッドにおける I/O 処理方式の選択について議論し、syscall スレッド新たな I/O 処理方式である非ビジーウェイトポーリングという方式を提案する。デバイスに対する I/O 処理方式の決定において重要なのはデバイスに対するデバイスコマンドを発行してからデバイスの I/O 処理が完了するまでの時間である。この時間をビジーウェイトで待つことによりポーリングする方式では、デバイスコマンドを発行してからデバイスの I/O 処理が完了するまでの応答時間がそのままポーリングのコストとなる。それに対して、I/O システムコール中でデバイスコマンドを発行してからそのプロセスをブロックし、コンテキストスイッチを発生させプロセスを切り替えて他のユーザプロセスの実行に移り、デバイスの I/O 処理の完了を通知するデバイスからのハードウェア割り込みを受け、再びコンテキストスイッチすることでブロックしていたプロセスの実行を再開する方式では、異なるユーザプロセスへとプロセス切り替えを行うことによるコンテキストスイッチ、スケジューラの実行、ページテーブル、ASID の切り替え等がオーバーヘッドとなり、その後のユーザプロセスの実行によるキャッシュ機構の汚染もパフォーマンスに関わってくる。

ここで、次世代不揮発性メモリベースの SSD のような高速なデバイスに対しては前者のビジーウェイトによるポーリングを行う方式の方が後者のデバイスの I/O 処理完了の通知であるハードウェア割り込みを利用する方式よりもより良いパフォーマンスが出ることが分かっている [3]。しかし、そうでないハードディスクや NAND 型フラッシュメモリ等に対しては依然として従来の割り込み方式の方が良い性能が出る。よって、通常ならばこの関係に従ってデバイスに応じて適切な I/O 処理を行えば良い。

ここで、[2] で提案されている FlexSC 本来の I/O 処理はどのように実装されているかについて確かめてみる。FlexSC の元々の実装では、FlexSC を利用する全てのプロセスはそのプロセスの持つ syscall ページのエントリ数と同じだけの syscall スレッドを持つ。それは何の為かというと、syscall スレッドがユーザプロセスのシステムコール要求を処理している間にブロックした場合に別の syscall スレッドに処理を切り替えてシステムコールの処理を継続することを可能にする為である。そう、FlexSC の元々の実装では I/O 処理は割り込み方式なのだ。そして、システムコールの処理中のブロックにおいて発生するコンテキストスイッチは常にある syscall スレッドから別の syscall スレッドへのスレッド切り替えである為にページテーブルの切り替えや ASID の切り替えが不要であり、その実行はカーネル空間内のみに限られる為にプロセッサのキャッシュ機構の汚染も最小限に抑えられる。このように、FlexSC における syscall スレッドの実行においては従来の処理と異なり、このようにデバイスの I/O 処理方式において割り込み処理方式のオーバーヘッドも少なく、キャッシュ構造の汚染も小さいのでビジーウェイトによるポーリングを行う方式に対する割り込み処理を行う方式の優位性が上がっている。

しかし、前述した通りに割り込みを用いる処理方式ではプロセスの持つ syscall ページのエントリ数と同じだけの syscall スレッドを持つ必要がある。普通一つのプロセスに対して複数の syscall ページを持ち [2] 、それぞれのプロセスがそのプロセスの持つ syscall ページのエントリ数と同じだけの syscall スレッドを持つので、結果として syscall スレッドによるカーネル圧迫や syscall スレッドの生成コストを考慮する必要がある。

そこで、我々は FlexSC における新たな処理方式として非ビジーウェイトポーリングという方式を提案する。syscall スレッドで扱うシステムコールの処理をノンブロッキングなもののみとして実装する。そして、ブロックするようなシステムコールは syscall スレッドの内部においてデバイスの I/O 完了待ちの前と後の二つの処理に分ける。すると、syscall スレッドが syscall ページを巡回する時にビジーループもブロックもコンテキストスイッチも不要なポーリングが可能となる。それに加えて、syscall スレッドで扱うシステムコールの処理においてはブロックしないので他の syscall スレッドに処理を切り替える必要も無く、syscall スレッド一つで全てのプロセスに対してシステムコールの処理を行うことができ、それによって syscall スレッドによるカーネルのメモリ圧迫や syscall スレッドの生成コストを大幅に下げることができる。

V. まとめと今後の課題

ここまで、ハードウェア割り込みとソフトウェア割り込みの回数を抑え、キャッシュ効率を改善するアーキテクチャについて論じてきた。クロック割り込みを単一のプロセッサで受け取るとともにカーネル空間における処理とユーザ空間における処理をプロセッサ毎に分け、クロック割り込みの影響を抑えつつより良いキャッシュ効率を達成する手法、そのクロック割り込みを受け取るカーネル空間における処理に特化したプロセッサで FlexSC の syscall スレッドを実行することによりシステムコールによるソフトウェア割り込みの影響を減らし、ユーザプロセスにおける処理ととカーネルにおける処理のキャッシュ効率を上げる手法、FlexSC の syscall スレッドにおける I/O 処理としての新しいモデルを提案してきた。これらを取り入れたアーキテクチャはより良いキャッシュ効率、より少ないキャッシュミスを達成し、パフォーマンスの向上に繋がると考えられる。その為に、本論文で提案したアーキテクチャの実装とその性能評価の為の取り組みが現在進行中である。

REFERENCES

[1] Dan Tsafrir, Yoav Etsion, Dror G. Feitelson, and Scott Kirkpatrick. System noise, OS clock ticks, and fine-grained parallel applications. In Proceedings of the 19th annual international conference on Supercomputing (ICS '05), pp. 305-312, 2005.

[2] Livio Soares and Michael Stumm. FlexSC: flexible system call scheduling with exception-less system calls. In Proceedings of the 9th USENIX conference on Operating systems design and implementation (OSDI'10), 2010.

[3] Jisoo Yang, Dave B. Minturn, and Frank Hady. When poll is better than interrupt. In Proceedings of the 10th USENIX conference on File and Storage Technologies (FAST'12), 2012.