アドレストレースによるOSの性能評価 *

4F－8

友田 正憲　武村 功司　本田 亮　関戸 一紀†

(株)東芝 情報・通信システム技術研究所‡

1 はじめに

近年、リモートファイルサービスやOLTPなどのビジネス分野においてもマルチプロセッサシステム上のUNIX[1]が広く用いられるようになって来た。このようなアプリケーションに対する性能の予測が、アーキテクチャやOS、MWの最適化を行なったり、効率的なシステム構築を行なったりするうえで不可欠である。そこで、我々はソースコードの解析と関数単位のトレース結果を元にモデル化を行ない、シミュレーションによって性能評価する手法を開発してきた[1]。本発表では、NFSなど、1つの実行時間が短く、色々な種類のオペレーション(NFSではwriteやreadなど)が、混合して処理されるアプリケーションに対して、キャッシュやバスの状態をシミュレーションするために、オペレーション間のキャッシュラインの関係に着目して、ミス率を関数でモデル化する比較的容易な手法を提案する。

2 シミュレーションとバスアクセス

オンチップキャッシュとメインメモリのアクセス時間の差が広がるに従って、キャッシュミスによる待ち時間がCPU性能に大きく影響するようになった。また、CPUを増やした場合のスケーラビリティを求めるにはバストラフィックを正しく求める必要があり、性能評価においてキャッシュミスの影響を評価することが重要である。しかし、ミス率はキャッシュの状態に依存するため、その値は時間とともに変動してしまう。そこでアドレストレースを使ってキャッシュ操作をそのままシミュレートすることが考えられるが、それには膨大なCPUコストがかかる。そこで我々は、プログラムのキャッシュミス率を近似的に関数でモデル化し、そのミス率から実際に生ずるバストラフィックをシミュレーションする手法を選択した。

3 メモリアクセスのモデル化

これまでにもミス率とキャッシュ構成や実行環境との関係について色々な解析的モデルが提案されている。しかし、それらのワークロードは、CPU時間が数秒以上かかる、相互に独立なプログラムをユーザレベルで実行した場合を前提としており、次の特徴をもつNFSやOLTPのオペレーションと合致しない。

- 各オペレーションのCPU時間は0．1秒以内でメモリのアクセス量がキャッシュ容量に比べて少ない。
- オペレーション間のアクセスデータには共通な部分が多い。
- 入出力により無効化されるデータがある。
- オペレーションの数、組合せを限定できない。

3.1 Thiébautモデル

そこで、そのモデルをNFSやOLTPのオペレーションにも適用できるように拡張した。拡張のベースには変更の容易なThiébautの解析的モデル[2]を採用した。このモデルではliveラインとuniqueラインと呼ばれる概念を導入している。liveラインとは1つのプログラム実行の中でキャッシュに取り込まれた後に1度以上アクセスされるメモリのラインで、uniqueラインはキャッシュに取り込まれたことのあるメモリのラインである。liveラインがキャッシュに多く残っているとヒット率が高くなる。Thiébautはメモリアクセスがfractal random walkと同じ性質を持つと仮定し、これらのラインの数の変化を関数でモデル化し、1つのプログラムが無限サイズのキャッシュに取り込むuniqueラインの数$U(r)$は、漸近的に次式に従って増加するとしている。

$$U(r) = Kr^{1/\theta} \quad (1)$$

rはあるプログラムが起動時から行なったメモリ参照回数、Kと$\theta(1 < \theta < 2)$は一定値であり、プログラムのトレースデータより値が求まる。ミス率は(1)の傾きより求まる。

3.2 拡張の方針

我々はunique、liveラインの概念を引き継ぎながらこのモデルを以下のように拡張し、複数の連続オペレーションに対するモデルを作成した。

1. オペレーションがアクセスするキャッシュラインはオペレーション毎に固有のものとオペレーション間で共通なものからなるとする。
2. オペレーションはカーネル内のメモリを使用し、その容量は有限である。よって、無限回のオペレーションでもオペレーション間で共通のライン数には上限がある。
3. I/O空間やDMAで読み込まれたデータはキャッシュミスすると考える。つまり、オペレーション毎に固有のラインに含める。
4. 連続して処理されるオペレーションに対して、処理開始からのオペレーションの総数iの関数として、live、uniqueラインの総数を求める。
5. オペレーション1からiの間で共通なラインの数、つまりliveライン数を$L(i)$とすると、オペレーション1からiまでのuniqueライン数は$U(i)$＝固有なライン数の総和$+L(i)$となる。

以下、我々のモデルにおけるミス率についてオペレーションが1種類の場合と複数種類の場合に分けて説明する。

3.3 1種類のオペレーション

同種のオペレーションはほぼ等しい数のキャッシュラインをアクセスすると仮定する。ライン数が大きく違うものは別種のオペレーションとみなす。また、あるオペレーションがキャッシュに取り込むliveライン数は、それ以前のオペレーションが使用し、かつキャッシュに残っているliveラインの数に依存して一意に決まると仮定する。

3.3.1 無限キャッシュにおけるミス率

簡単のためにまず無限のサイズを持つキャッシュにおいて、固有のラインをuだけ持ち、全体でT回アクセスを行

---

*Performance Evaluation of OS with Address Traces

†Masanori TOMODA, Kouji TAKEMURA, Makoto HONDA, Kazunori SEKIDO

‡Information & Communication Systems Lab. TOSHIBA Corp.

[1]UNIXはUNIX System Laboratories, Inc.が開発し、ライセンスしています。

なう同一オペレーションが、連続的に複数個実行された時のキャッシュミス率を考える。i回めのオペレーションでキャッシュに取り込まれるliveライン数はliveライン関数の定義から$L(i)-L(i-1)$となるので、このオペレーションによって発生するキャッシュミス率$M(i)$は次のようになる。

$$M(i) = (u + L(i) - L(i-1))/T$$

3.3.2 有限キャッシュのliveライン数

次にキャッシュが有限の場合を考える。キャッシュが一杯になるまでは、ミス率は無限キャッシュと同じ式である。キャッシュ制御方式はfully associative方式であり、置き換えはLRU方式で行なうとする。いま、説明を簡単にするために$k-1$回目のオペレーションでキャッシュがちょうど一杯になるとすると、kは次の式で表される。

$$U(k-1) = CacheSize \qquad (2)$$
$$U(i) = u * i + L(i)$$

$k-1$番目が終了した時点で、キャッシュには1から$k-1$まで、$k-1$個のオペレーションが使用したライン存在する。つまり$L(k-1)$個のliveラインが存在するので、上記の仮定からk番目のオペレーションでは$L(k)-L(k-1)$個のliveラインを取り込む。するとキャッシュの容量の制限から同数のliveラインが置き換えられて、キャッシュ内には$L(k-1)$個のliveラインが残る。従って$k+1$番目のオペレーションでも、仮定から$L(k)-L(k-1)$のliveラインを取り込む。このライン数はオペレーションk以降のオペレーションで変わらない。よって有限キャッシュのミス率$M_f(i)$は次のようになる。

$$M_f(i) = \begin{cases} (u + L(i) - L(i-1))/T & i < k \\ (u + L(k) - L(k-1))/T & i \geq k \end{cases}$$

3.4 複数種類のオペレーション

次に、複数種類が混じった処理のメモリアクセスへの拡張を行なう。2つのオペレーションをA、Bとする。双方が共通に参照するメモリ領域がある場合にはその影響を考える必要がある。

3.4.1 有限キャッシュのミス率

2つのオペレーションA、Bが任意の組合せで複数回実行された時のミス率を考える。共通部分以外は、オペレーションごとに前章と同様に考える。そこで共通部分の影響を図1を用いて求める。まずA、Bを無限回実行した時のliveライン全体をL_A、L_Bとし、共通部分をL_{AB}とする。L_{AB}に含まれるライン数は拡張2より一定値であり、その数をLABとする。次に、L_{AB}を面積がライン数に比例する矩形

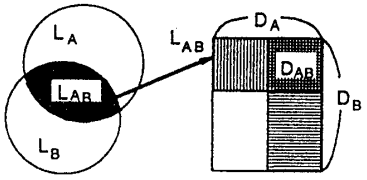

図1

領域で表す。A、Bがそれぞれi_a、i_b回実行されたとき、それぞれがアクセスする共通なラインの領域をD_A、D_Bとし、そのライン数を$LC^A(i_a)$、$LC^B(i_b)$とする。A、Bが独立に共通部分にアクセスすると仮定すると、Aが取り込むliveラインには、Bによって2重に取り込まれるliveラインが含まれる。このラインの領域をD_{AB}とする。図からAのliveラインに対するこのラインの数の割合は、$LC^B(i_b)/LAB$と求まる。そこでこのラインを除いて共通部分のライン数を求める。詳細な導出法は省略するが、i番目のオペレーションがAである場合のキャッシュミス率$M_m(i)$は、

$$M_m(i) = \begin{cases} \frac{1}{T_A}((u_a + L^A(n_a+1) - L^A(n_a)) \\ \quad -(LC^A(n_a+1) - LC^A(n_a)) * \frac{LC^B(n_b)}{LAB}) & i < k \\ \frac{1}{T_A}((u_a + L^A(N_A+1) - L^A(N_A)) \\ \quad -(LC^A(N_A+1) - LC^A(N_A)) * \frac{LC^B(N_B)}{LAB}) & i \geq k \end{cases}$$

となる。T_AはオペレーションAのメモリアクセス回数、u_aは固有部分のライン数である。n_a、n_bはそれぞれオペレーションA、Bの数で、$i = n_a + n_b$ $(i < k)$を満たす。N_A、N_BもオペレーションA、Bの数で、キャッシュにそのデータが残っているオペレーションの数をKとすると、$K = N_A + N_B$を満たす。iについて2つの場合があるが、どちらにおいても第1項は前章と同様に求まる項であり、第2項は3.4.1の共通部分の影響を表す項である。さらにi番目のオペレーションがBである場合も、同様な手法でミス率が求まる。

3.5 $U(i)$と$L(i)$の関数

NFSコールのwriteオペレーションに対する$U(i)$と$L(i)$の関数を、実際のメモリアドレストレースを用いて調べた。トレースはAE[3]を我々が拡張したシステムで採取した。図2、3は入出力により無効化されるデータを考慮する前と後のグラフである。2つの$U(i)$関数が大きく違うことから、NFSでは入出力によるデータを考慮する必要があると言える。また、$L(i)$関数が一定であるから、各オペレーションはほぼ同じliveラインをアクセスすることもわかる。図3の$LC(i)$はreadオペレーションと共通部分のliveライン数である。このグラフからwriteにおけるreadとの共通部分は約1/2であることもわかる。

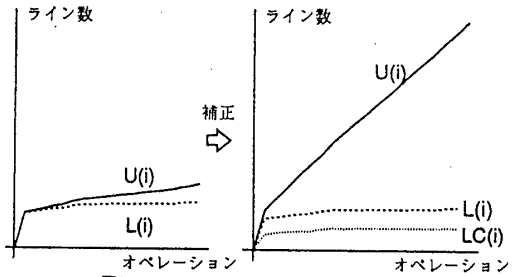

図2

図3

4 おわりに

本発表では、NFSなどの軽いオペレーションが混合して処理されるアプリケーションに対するメモリアクセスを関数でモデル化し、ミス率を求める方法について述べた。オペレーション間のキャッシュラインの関係に着目することにより、シミュレーション時にはキャッシュ内で有効なオペレーション数を計算するだけで$L(i)$関数とオペレーション固有のラインuから簡単にミス率が求まる。今後は、$L(i)$関数の定式化を考えるとともに、ダーティラインについても同様のモデル化を行ない、その書き戻しやプロセッサ間のデータ移動もシミュレーションに組み込む予定である。

参考文献

[1] 本田 亮, 他, UNIXシステムのネットワーク性能評価, 情報処理学会第45回全国大会

[2] D. Thiébaut, "On the Fractal Dimension of Computer Programs and its Application to the Prediction of the Cache Miss Ratio," IEEE Trans. Comput., Vol. 38, No. 7, pp. 1012-1026, Jul. 1989

[3] James R. Larus, "Abstract Execution: A Technique for Efficiently Tracing Programs,"revised version of Tech. Tep. #912, Comput. Sci. Lab., Univ. of Winsconsin, Feb. 1990