社団法人 情報処理学会　研究報告
IPSJ SIG Technical Report

2007－ARC－172　(40)
2007－HPC－109　(40)
2007／3／2

先読みスレッドを用いたDiskアクセスの高速化

深山辰徳†，杉田秀†，蛭田智則†，山名早人††,‡
†早稲田大学大学院理工学研究科
††早稲田大学理工学術院
‡国立情報学研究所

磁気記憶装置からのデータ取得にかかる時間と，主記憶からのデータの取得にかかる時間では，4～10倍の差がある．こうした格差を補うための研究として，磁気記憶装置から効率的にデータを取り出す研究が行われている．本論文ではプログラムがアクセスする可能性の高いデータに対して，先読みスレッドで事前にデータの読み込みを行うことによりプログラムの性能向上を目指す．具体的には，マルチコア環境における先読みスレッドを利用したディスクキャッシュの効果的利用を提案する．メインスレッドとは別のスレッドで先読みを行うことにより，事前にディスクキャッシュ上に目的のデータを乗せることができる．本手法によって，gzipが最大で39.2%性能向上することが確認できた．

Disk Access Speed up Using Prefetching Thread

Tatsunori Fukayama†, Shu Sugita†, Tomonori Hiruta†, Hayato Yamana††,‡
† Computer Science Div. Graduate School of Science and Engineering, Waseda University
†† Science and Engineering, Waseda University
‡ National Institute of Informatics

It takes four to ten times more time to get data from hard disk drive than from DRAM. In this paper, we present a speed up mechanisms using a prefetching thread on a multicore system to overcome this relative deterioration of hard disk drive performance. A Prefetching thread loads data from hard disk before main thread requires the particular data. When main thread requires the data, the data will be found on disk cache so it will take no time to get the data. We have confirmed that the prefetching thread reduces the execution time of gzip. The performance of gzip increased up to 39.2%.

1. はじめに

主記憶装置とHDDとでは，アクセス速度に大きなギャップがある．具体的には，表1の実験環境Bにおいて，主記憶装置上にある1GBのデータにアクセスする際にかかる時間は約6秒であるのに対して，HDD上にある1GBのデータにアクセスする際にかかる時間は約23秒であり，4～10倍の差がある1)2).

アプリケーションの実行に際して，HDDへのアクセスはしばしばボトルネックになるとして，様々な工夫や研究が行われてきた．その一つは，Disk Readahead3)であり，シーケンシャルアクセスされているデータに対して，要求されたデータよりも多くのデータを読み込み，次回のアクセスに備えてディスクキャッシュに載せるというものである．現在すでにLinuxには，この機能が実装されている．シーケンシャルアクセス以外の手法としては，Informed Prefetching1)2)がある．これは，アプリケーションレベルで事前にヒントを与え，次にどのデータが要求されるかを予測し，その結果を元にPrefetchを行う手法である．シーケンシャルでは無いものに対しても効果があるとされている．

しかし，従来手法のDisk Readaheadは，IO待ち時間を軽減することができるものの，依然として繰り返しディスクキャッシュミスが発生している．さらにInformed Prefetchingは，次に要求されるデータの予測が失敗した場合，Prefetchの失敗を意味し，ディスクアクセスの高速化が実現できない．そこで，本稿では，近年一般的に普及してきたHyperThreding対応ＣＰＵ，及びマルチコアCPUの構造に着目し，別スレッドでディスクの先読みを行い，ディスクキャッシュミスを軽減する手法を提案する．具体的には，メインスレッドでアプリケーションを実行していると同時に，別スレッドで次のＨＤＤアクセスを実行する．その結果，提案手法を適用することによって，メインスレッドは，HDDに記録されているデータにアクセスしているにも関らず，あたかもメモリ上のデータにアクセスしているような性能を得ることができる．

本稿では，以下のような構成をとる．第2節では，関連する研究について紹介する．第3節では，提案手法を説明する．第4節では，提案手法を実際に実装し，その評価を行った結果を示す．最後に第5節で，結論を述べる．

2. 関連研究

HDDへアクセスする際に発生する遅延時間を

できる限り減らそうと長年研究が行われてきた．大きく分けるとハードウェア面で性能を向上させる手法2)4)と，ソフトウェア面で性能を向上させる手法1)3)に分けることができる．ハードウェア面で性能を向上させるものには，Disk Striping2)4)により読み取り速度を向上させるものがある．ソフトウェア面で性能を向上させるものは，Disk Cache を効果的に使うことにより，遅延時間を減らすという研究である．

Disk Cache に着目した研究はさらに分類することができ，Prefetch, Readahead, Cache Control のどれかに属するものとして考えることができる．Prefetch はプログラムにヒントを与え，次に必要になりそうなデータを事前に予測して取得するという手法である．主にランダムアクセスの多いプログラムに対して効果が高いが予測が外れると効果が無い．提案手法は，次の Disk アクセスを特定できるため，確実に効果がある．Readahead はシーケンシャルにデータにアクセスする際に効果のある手法で，要求のあったデータの次のいくつか先のブロックまでまとめて読み込むという手法である．Readahead は，すでに Linux に組み込まれている．Cache Control は，主記憶上にあるディスクキャッシュを扱うためのアルゴリズムに関する研究であり，他の Prefetch や Readahead の手法と合わせて利用することができる．

3. 提案手法

I/O によるプログラム実行の遅延を軽減する為に，先読みスレッドを用いて事前にファイルの読み込みを行う手法を提案する．事前に先読みスレッドがデータの読み込みを行うことで，メインスレッドがデータを取得する際には，ディスクキャッシュ上から目的のデータを取得することが可能になり，I/O による遅延を軽減することができる．

3.1 対象プログラム

本手法が対象とするのは，データに対してシーケンシャルにアクセスを行い，かつデータを読み込みつつ逐次演算処理を行うようなプログラムである．その時演算にかかる時間と，I/O にかかる時間の関係は，式(1)の状態にあることが望ましい．逆に式(2)の状態にある時，本手法は有効で無いばかりか性能の低下を招くこともある．式(1)の状態にある時，本来 I/O にかかる時間を演算処理と I/O 処理を並列に行うことにより，演算時間の中に埋め込むことが可能である．

対象プログラムとして I/O 時間と演算時間の関係が式(1)の状態にあるプログラムとしてしまうことで対象となるプログラムは限定されてしまう．しかし，式(1)の状態にあるプログラムには圧縮系のプログラムやマルチメディア系のプログラムが含まれるため有意義であると考えられる．

$$I/O\text{時間} \leq \text{演算時間} \quad (1)$$

$$I/O\text{時間} >> \text{演算時間} \quad (2)$$

3.2 対象アーキテクチャ

本手法は，Hyper-Threading 対応の CPU，複数のコアを持った CPU，メモリが共有されているマルチプロセッサー上で性能向上を得ることができる．3.5 で述べるように，本手法では先読みスレッドの制御に多数の演算処理を伴う為メインスレッドと先読みスレッドが同時に実行できる環境でより高い性能向上を得ることができる．

3.3 提案手法のモデル

提案手法のモデルは，次に示すとおりである．図 1 にあるようにメインスレッド上で，読み込むファイルが確定した時点で先読みスレッドを起動する．先読みスレッドは，指定されたファイルを先行して読み込み続ける．先読みスレッドによって読み込まれたデータはディスクキャッシュに乗ることになる．メインスレッドがデータを必要とする時には，ディスクキャッシュ上にデータが存在する．その結果，HDD から読み込む場合に比べて短時間でデータの取得が可能になる．

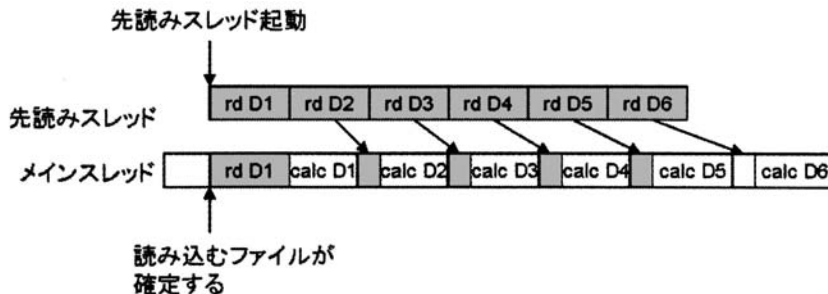

図 1 提案手法のモデル

3.4 ディスクキャッシュの汚染

コンピュータ上のメモリリソースは限られているため，闇雲に先読みスレッドが先読みをしてしまうと，ディスクキャッシュの汚染が発生し逆に何倍も遅くなることがある．この汚染を防ぐために，提案手法ではメインスレッドと先読みスレッドの距離を定義し，距離の値によって先読みを制御する方策をとる[1]．ここで言う距離とは，ある時点でのメインスレッドのファイルディスクリプタが指す位置と先読みスレッドのファイルディスクリプタが指す位置の差である．この距離が空きメモリ領域以上に離れてしまうと，先読みスレッドが先行して読み込んだデータにより，メインスレッドがこれから必要とするデータがディスクキャッシュから追い出されてしまう．結果としてメインスレッドと先読みスレッドの双方が，ディスクキャッシュ上にデータを発見することができなくなるため，二重に HDD へのアクセスが発生してしまう．こうなると，それぞれのスレッドが新たに I/O 命令を発行する度にディスクヘッドが移動することになる．ディスクヘッドが移動する度にシークタイムと回転待ち時間の遅延が発生してしまう．この状態は，メインスレッドと先読みスレッドとの距離が離れてしまう場合だけでなく，他の実行中のプロセスがメモリリソースを使う場合にも発生しうる．

3.5 先読みスレッドの制御

3.4 で述べたように，先読みスレッドは闇雲に先行すればいいわけでは無い．メインスレッドと先読みスレッドの距離が一定以上開かないように制御する必要がある．この制限距離が，小さすぎると思ったような効果を得ることができない．

メインスレッドと先読みスレッドに，それぞれ現在何バイトのデータを読み込んだかを変数として持たせる．データ読み込み後に，読み込んだバイト数を変数に加算する．先読みスレッドは，事前に指定される一定量のデータを読み込んだ段階で変数に格納された値を元に，メインスレッドとの距離を計算する．距離が予め指定された閾値を超えてしまっている場合には，閾値を下回るまで usleep(1)を実行する．

ここでは，メインスレッドと先読みスレッドの距離を計算するまでに読み込むデータ量を ss_distance と定義し，先読みスレッドがメインスレッドよりも先行することを許される限界の距離を limit と定義する．先読みスレッドの制御方法は図 2 のようになる．prefetch_pos，main_pos はそれぞれ先読みスレッドとメインスレッドの現在位置を表す．IFD は入力ファイルディスクリプタ，BUF は入力バッファ，BUFSIZE は入力バッファのサイズをそれぞれ表している．

```
while (keep_reading)  {
  ss_distance =  16MB
  while (ss_distance  >  0 )
    size = read (
      IFD,
      BUF,
      BUFSIZE)  ;

    prefetch_pos  + =  size;

    ss_distance  - =  size;
  }
  while  (
    prefetch_pos - main_pos > limit
    ) {
    usleep ( 1 )  ;
  }
}
```

図 2 先読みスレッドの制御方法

ss_distance が小さすぎると頻繁に先読みスレッドとメインスレッドの距離が計算されてしまい，Hyper-Threading 環境下では性能の低下につながってしまう．ss_distance が制限距離に比べて大きすぎると，ディスクキャッシュの汚染に繋がる可能性がある．

制限距離が小さすぎると，先読みスレッドは頻繁に止まってしまい性能向上を得ることができない．制限距離が空きメモリ領域よりも大きいと，ディスクキャッシュ汚染に繋がってしまう可能性がある．

4. 実験

提案手法の有効性を検証するため，本提案手法を適用させる前の gzip と適用後の gzip の 2 種類の実行時間の比較を行った．ここで，本手法を適用する前の gzip を標準の gzip，本手法を適用後の gzip を高速 gzip と呼ぶ．今回本手法を適用した gzip のバージョンは 1.3.5 である．

4.1 実験環境

本稿で行った実験の環境は表 1 に示す通りである．

4.2 実験内容

まず，標準の gzip を用いて 8 つのファイルを圧縮し，圧縮処理にかかった時間を計測した．次に高速 gzip を用いて 8 つのファイルを圧縮し，圧縮処理にかかった時間を計測した．実行時間の計測方法は，圧縮処理を行っている箇所の前後で，gettimeofday()関数を用いて測定した．プログラム実験の対象のファイルの容量は表 2 に示す通りである．

[1] 具体的な制御方法は 3.5 で後述する．

表 1 実験環境

	実験環境 A	実験環境 B
CPU	Pentium Extreme Edition 3.2GHz	Pentium4 3.4GHz
Memory	1GB	2GB
OS	Fedora Core 5 Kernel	Windows XP Professional
HDD	SAMSUNG HD160JJ	Hitachi Deskstar 7k250
容量	160GB	160GB
プラッターサイズ	80GB	80GB
キャッシュ	8MB	8MB
回転数	7200RPM	7200RPM
内部転送速度	845Mbits/sec	757Mbits/sec
平均シークタイム	8.9ms	8.5ms
最小シークタイム	0.8ms	1.1ms
最大シークタイム	18ms	15.1ms

4.3 実験結果

図 3，4 は実験環境 A における標準 gzip と高速 gzip の実行時間の比較である．図 5，6 は実験環境 B における標準 gzip と高速 gzip の実行時間の比較である．それぞれ gzip の圧縮レベルは 3 を指定している．図 3 のグラフは，先読みの距離の制限値を左から順に 4KB，8KB，16KB，・・・，1MB まで変化させた時の結果になっている．図 4 のグラフは，先読みの距離の制限値を左から順に 2MB，4MB，8MB，・・・512MB まで変化させた時の結果になっている．図 5，6 も同様に先読みの距離の制限値を変化させたものの結果となっている．図 3～6 の折れ線グラフで表示されているものが，標準 gzip の実行時間を表している．いずれのグラフも縦軸が時間(単位は秒)を表し，横軸はファイル番号に対応している．

表 2 実験に用いたファイルのサイズ一覧

	実験環境 A	実験環境 B
File No. 1	0.61GB	0.61GB
File No. 2	1.00GB	1.00GB
File No. 3	1.61GB	1.61GB
File No. 4	2.22GB	2.22GB
File No. 5	2.83GB	2.84GB
File No. 6	3.44GB	3.45GB
File No. 7	4.05GB	4.07GB
File No. 8	4.66GB	4.68GB

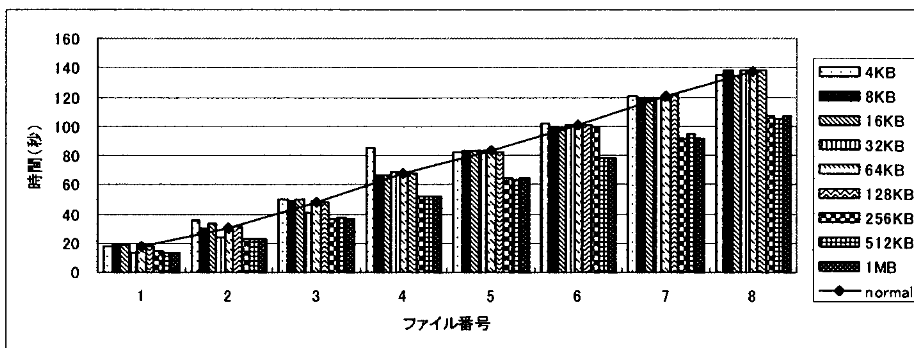

図 3 gzip 実行時間の比較 A-1

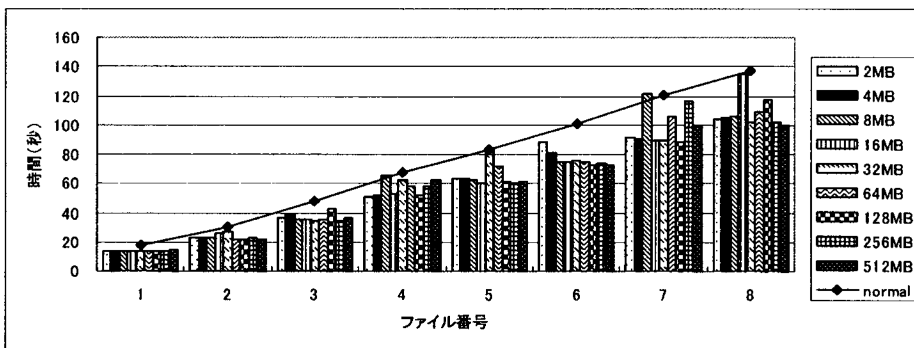

図 4 gzip 実行時間の比較 A-2

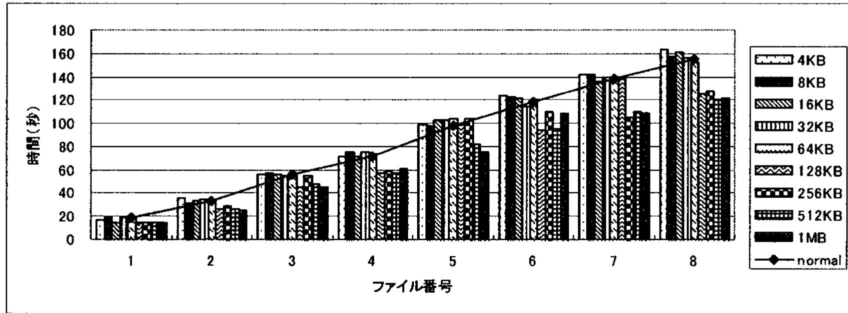

図 5 gzip 実行時間の比較 B-1

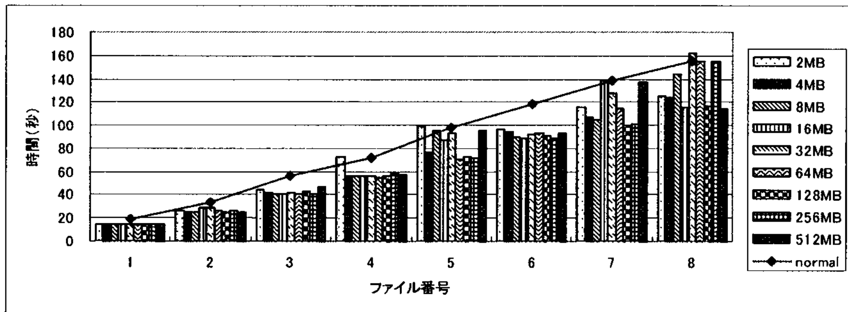

図 6 gzip 実行時間の比較 B-2

表 3 高速 gzip の性能向上率

File No.	標準 gzip A	A	A の向上率	標準 gzip B	B	B の向上率
1	17.31	13.03	32.9%	19.16	14.08	36.1%
2	29.65	21.54	37.6%	32.90	24.80	32.6%
3	47.66	34.55	37.9%	56.25	40.42	39.2%
4	67.11	50.56	32.7%	71.16	55.40	28.5%
5	82.80	60.30	37.3%	97.57	70.20	39.0%
6	101.04	72.79	38.8%	118.11	88.15	34.0%
7	120.04	88.65	35.4%	138.29	99.87	38.5%
8	136.99	100.02	37.0%	154.81	114.52	35.2%

実験環境Aでは，先読み距離の制限値が大きくなるに従い，全体的に標準gzipよりも短い時間で処理を終えていることが分かる．グラフより256KB以降に設定すると安定して性能向上を得られることが分かる．4KB～128KBのものに関しては，先読みする距離が短いため十分な性能向上が得られていないことが分かる．実験環境Aはメモリが1GBあるので，図4より先読みの距離の制限値を 512MB に増やしてもディスクキャッシュの汚染が発生していないことが分かる．

実験環境Bでも同様に，標準gzipよりも短い時間で処理を終えているのが分かる．図5より先読みの距離の制限値が 4KB～64KB のものに関しては，実験環境Aと同様に先読みする距離が短いため十分な性能向上が得られていないことが分かる．実験環境Bはメモリが2GBあるので，図6より先読みの距離の制限値を 512MB に増やしてもディスクキャッシュの汚染が発生していないことが分かる．

表3に標準gzipと高速gzipの実行時間を示す．File No.は表2のファイル1～8に対応している．標準gzipは標準gzipで4回圧縮処理を行った際にかかった時間の最小値である．AとBはそれぞれ実験環境A，Bにおいて高速gzipで圧縮処理を行った際にかかった時間の最小値である．向上率はそれぞれ，標準gzipの時間を高速gzipの時間で割り1を引いた値になっている．正の向上率は，gzipの実行速度がそれだけ上昇したことを示している．逆に負の向上率は，gzip の実行速度がそれだけ低下していることを示している．

5. おわりに

従来から disk read ahead という考えはある．今回の実験環境Aには disk read ahead が実装されている．しかし，この disk read ahead はアプリケーションが現在読み込んでいる位置よりも，常に一定容量読み込むというものでは無い．近年CPUのクロック数が頭打ちになり，コアが増える傾向にある．既存のプログラムでコアが増えることにより即座に性能が向上するものは少ない．今回は，余剰な計算資源を有効活用することにより，既存のプログラムを簡単に高速化する手法として先読みスレッドを提案した．本手法を gzip に実装することにより圧縮レベル 3 において，最大で 39.2%の性能向上を得ることができ，実験環境Aでは平均して 36.2%の性能向上，実験環境Bでは平均して 35.3%の性能向上を得られることを確認した．

謝辞

本研究の一部は、21世紀COEプログラム「プロダクティブICTアカデミア」および，科研費基盤B「ヘルパースレッドを用いたマルチスレッディングプロセッサのための高速化技術研究」によるものである．

参考文献

1) Randy H. Katz, G. A. Gibson, D. A. Patterson, "Disk System Architectures for High Performance Computing," Proceedings of the IEEE, Volume 77 (12), December 1989, pp. 1842-1858.

2) David A. Patterson, Garth A. Gibson, Randy H. Katz, "A Case for Redundant Arrays of Inexpensive Disks (RAID)," Proceedings of the 1988 ACM Conference on Management of Data (SIGMOD), Chicago IL, June 1988, pp.109-116.

3) Garth A. Gibson, R. Hugo Patterson, M. Satyanarayanan, "Disk Reads with DRAM Latency," Third Workshop on Workstation Operating Systems, Key Biscayne, FL, April, 1992, pp. 126-131.

4) Garth A. Gibson, Redundant Disk Arrays: Reliable, Parallel Secondary Storage, Ph.D. Dissertation, University of California, Technical Report UCB/CSD 90/613, March 1991. To be published by MIT Press.