非同期I/Oを用いたFlash SSDによるメインメモリ拡張のための性能調査

丹英之†§，緑川博子†§
†成蹊大学 §JST, CREST

背景と目的

- 巨大な問題サイズのジョブを動かしたい.
- NAND Flashをはじめとした不揮発性メモリの台頭.
  - 省電力・大容量・遅いメモリとして利用する.
- PC接続のフラッシュメモリはフラッシュストレージ.

これまで検討してきた手法

- swap: malloc(3)で確保，仮想メモリのページアウト先としてスワップ用ブロックデバイスにする.
- mmap: mmap(3)でマッピングするファイルを配置するストレージ(ext4fs)にする.

swap, mmapはバイトアクセスできるので実装が楽.
ページアウト状態だとデータ参照に時間を要する.

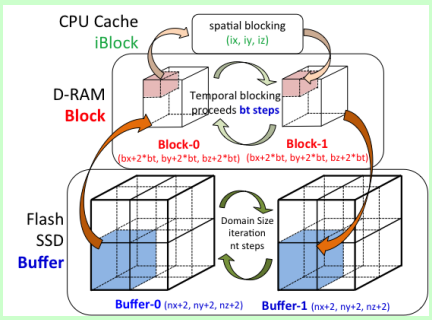
- aio: Linuxネイティブ(libaio)でメインメモリ-ブロックデバイス間のデータ交換を行う.

I/Oの多重発行でデバイスの性能を引き出せる.
バッファアライメントやI/Oサイズの制約で実装が面倒.

計測環境

CPU	Xeon E5-2687W 3.1GHz 2 socket (8x2 cores)
Memory	DDR3-1600 ECC 8GiB x 8 (64GiB)
Flash Storage	Fusion-io ioDrive2 MLC(PCIe2.0x4) 1.2TB
OS	CentOS6.4(x86_64)
Compiler	gcc version 4.4.7 20120313 / -O3

大規模ステンシル計算への応用例



時間発展でのブロッキング

D-RAMとFlash SSDのデータ交換に局所性を作り出す.

ベンチマーク

1. ULONG型(sizeは8B)の配列をそれぞれの手法にてFlash SSD上に確保し，要素番号を値として代入していく(write).
2. 配列の要素を参照し，要素番号と一致することを確認する(read).
3. 扱ったデータサイズと各操作に要した時間から，それぞれの手法におけるメモリバンド幅を得る.

計測パターン

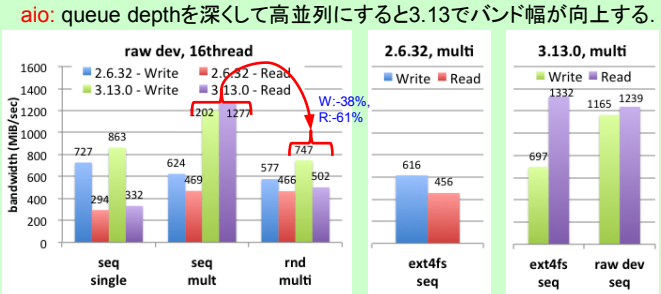
- データサイズ：128GiB
- mmap：1要素ずつ代入・参照(byte) / 要素数512単位(4KiB)でmemcpy(3)(block)した場合
- aio：4KiB単位でi/oを投入(single)，queue depthはスレッド数. / 4KiBのi/oを64K回まとめて投入(multi)，queue depthはスレッド数×64K.
- アクセスパターン：シーケンシャル(seq) / ランダム(rnd)
- open: ext4fs上のファイル(ext4fs) / ブロックデバイス(rawdev)
- スレッド数：1, 8, 16
- kernelのバージョン: 2.6.32 / 3.13.0
- madvise(MADV_SEQUENTIAL)の指定(MADV_SEQ)

結果と考察

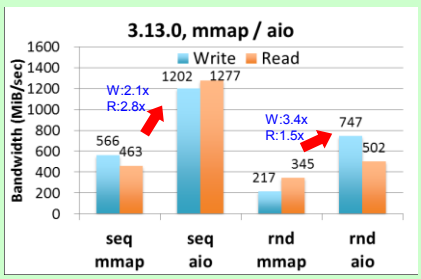
mmap: 2.6.32ではseq→rndで極端にバンド幅が低下する.

※ di-mmap: data-intensive memory-map runtime
大きなデータサイズを扱うために工夫したmmap[1].

aio: queue depthを深くして高並列にすると3.13でバンド幅が向上する.



mmapとaioの比較



- seqでは2.4倍，rndでは2.2倍，mmapよりaioのバンド幅が高い.
- アプリケーションがマルチスレッドでシーケンシャルアクセスであることを考慮すると，処理時間の短縮が期待できる.

まとめと今後の展開

- Flash SSDを拡張メインメモリとして用いることを想定しマイクロベンチマークを行った結果，aio方式のバンド幅はmmap方式より，2.2倍高く，処理時間短縮に対しaio方式の有効性を確認できた.
- 既存アプリケーションにおいて，メモリ確保処理部分をFlash SSD上のファイルmmapに置き換えることは容易だが，aio方式では実装が複雑になる傾向がある.
- 今後は，aio方式での実装を簡略化できるフレームワークを検討していく.

[1] Brian Van Essen, et.al "DI-MMAP—a scalable memory-map runtime for out-of-core data-intensive applications", Cluster Computing , Vol.18, Issu1, pp.15-28, March 2015