Vol.2013-HPC-140 No.44
2013/8/2

メモリサイズを超えるデータ処理を目的とした バス接続型 SSD の性能評価

緑川 博子[†1]　　丹 英之[†2]

最近，利用可能になってきたバス接続型 SSD の IO 基本性能を，従来の SATA 接続 SSD と HD と比較した．さらに，既存 OS で提供される汎用インターフェースや仮想メモリ機構を利用して，物理メモリサイズを超えるデータ処理を行うプログラムが，SSD をメモリ拡張の用途に用いた場合の予備性能調査を行った．

A Preliminary Performance Evaluation of a Bus connected SSD for Larger amount of Data Processing than Local Memory Size

HIROKO MIDORIKAWA[†1]　　HIDEYUKI TAN[†2]

The basic IO performances of a bus connected SSD are investigated in comparison with traditional SATA SSD and HD. A preliminary evaluation to use a SSD as a memory extension is done by using the existing OS general interfaces and a virtual memory mechanism. Several benchmarks that process larger amount of data beyond local memory size are used here.

1. はじめに

長く主記憶として使われてきた DRAM の高速化, 大容量化は著しいものがあった．また，CPU 高速化に伴うメモリとの性能差が問題となって久しい．最近では細密化が進み, DRAM の電荷保持の限界に近くなっているとも言われ，小容量の電荷を常時チャージするための電力は高まる傾向にある．今後，多数ノードからなるシステムにおいて，各ノードに大量の DRAM を搭載することは消費電力の観点からも難しい．一方，MRAM，ReRAM, FeRAM などの様々な不揮発性メモリが研究・実用化されつつあり，今後主メモリを構成する大きな担い手となると考えられる．消費電力低減，大容量記憶というだけでなく「消えないデータ」という新しい概念が，従来の「揮発性主記憶と不揮発性二次記憶」という伝統的枠組みに依存したメモリとファイルという概念そのものに変化を与える可能性さえも秘めている． 既に，ファイルシステムとしては，ハードディスクに代わってフラッシュメモリを備えたパソコンやサーバが広く普及しており，大容量のハードディスクと高速なフラッシュメモリを組み合わせたハイブリッド型のデバイスも利用されている．また，研究段階ではあるが，不揮発性メモリとフラッシュを組み合わせて，フラッシュの弱点である耐久性（書き込み劣化による寿命）を向上させるデバイスなども提案されている．また，メモリを組み込んだ CPU チップの開発や，GPU と CPU のメモリを統一的に扱える仕組みなども作られつつある．

このようなコンピュータの根幹である記憶デバイスハードウエアにまつわる変革が，今後のソフトウエアに影響を及ぼすことは必須である．この状況を反映し，OS カーネルのファイルやメモリ周りの変更，特にフラッシュメモリを意識したファイルシステム[1]や，伝統的にハードディスクを前提としていたスワップシステムの改善なども提案され[2]，一部は Linux のカーネルに取り込まれつつある．このハードウエアの変化は，OS はむろんのこと，プログラミングモデルにも影響する可能性がある．今後，容量・速度パラメタの異なる多種多様な記憶デバイスによるメモリ階層の深化が進むと考えられ，これに対応するシステムソフトウエアの開発が強く求められている．

現在，さまざまな不揮発性メモリの研究・開発される一方，バスに接続されるフラッシュメモリデバイスが普及し始めている[3]-[6]．2012 年，FusionIO 社はバス型接続 SSD 製品向けにメモリセマンティック API を提供するとして話題となった．（2013 年 7 月 4 日時点で未リリース.）

本報告では，このようなバス接続型 SSD について，実際の基本 IO 性能を計測し，現在利用可能な API で，物理メモリを超えるデータ処理を行うプログラムから利用した場合の性能についても調査した．2 節では，従来の SATA 接続 SSD やハードディスク（HDD）と PCIe バス接続型 SSD（FusuinIO, ioDrive2）の基本 IO 性能を計測し比較した．また 3 種の IO 方式による性能の違いについて明らかにした．3 節では，汎用 UNIX API（ファイルとして利用）を用い，物理メモリを超えるデータを扱う応用ベンチマークプログラムを実行させて，実際の応用における性能を調査した．

2. バス接続型 SSD の基本 IO 性能評価

2.1 FIO ベンチマークによる基本 IO 性能

PCIe 接続型フラッシュメモリが実際にどの程度，従来型

†1 成蹊大学，JST CREST
Seikei University, JST CREST
†2（株）アルファシステムズ，JST CREST
Alpha Systems Inc. , JST CREST

の SATA 接続 SSD, HDD と差があるのかを調査した．各デバイスをファイルシステム ext4fs でフォーマットし， IO 性能ベンチマーク FIO (Flexible IO test)[7](バージョン 2.0.15)を用い計測した．比較に用いたデバイスと実験環境を表 1，表 2 に示す．FIO ベンチマークでは，発行する IO 操作（API）やその IO サイズを指定することができる．ここでは，ファイルサイズが 8GiB のファイルを対象に，IO 操作 read(2) / write(2)+lseek(2)(以下，RW-IO 操作)にて，IO サイズ 1MiB で逐次アクセスで読み書きした場合と，4KiB から 4MiB の IO サイズでランダムに読み書きした場合の IO バンド幅を計測した．

表 1　IO 性能評価に用いたデバイス
Table 1　Devices used in IO performance evaluation.

略称	製品名（製造）	容量	接続
FIO	ioDrive2 (FusionIO)	1.2TB	PCIe 2.0 x 4
SSD	A1 SSD (innodisk)	240GB	SATA3
HDD	WD1003FBYX-0(WDC)	1TB	SATA

表 2　実験環境
Table 2 The Experimental Environment.

CPU	Xeon E5-2687W 3.10GHz x 2 (16cores)
Memory	32GB-boot (Total Mem: 128GB)
OS	CentOS6 (2.6.32-279.14.1.el6.x86_64)
Compiler	gcc 4.4.6 -O3
SwapDevice	FIO / SSD
map/r/w File	FIO / SSD / HDD
file system	ext4

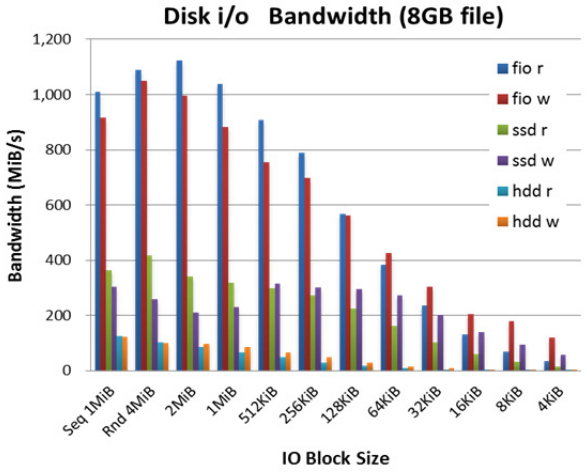

図 1　IO バンド幅（8GiB ファイル，主記憶 32GiB）
Figure 1　IO Bandwidth (8GiB file, 32GiB memory)

図 1 は，1MiB サイズで逐次アクセスで読み書きした場合（左端）と，4KiB から 4MiB のサイズでランダムに読み書きした場合の IO バンド幅を示している．ioDrive2 では，2MiB リードが 1122MiB/s，4MiB ライトが 1048MiB/s でそれぞれ最高値である．4MiB～128KiB の範囲では，ライトよりリードの性能が高いが，それより小さい IO サイズでは，ライト性能が高い．一方，SATA 接続型 SSD では 4MiB のリードが 418MiB/s, ライトが 260MiB/s で最高値となる．HDD では，同じく 4MiB のリードが 101MiB/s，ライトが 101MiB/s で最高値となる．

各 IO サイズにおける HDD のリード・ライト性能を基準としたときの，ioDrive2 と SSD の相対バンド幅を図 2 に示す．ioDrive2 は，ランダムリードでは，4MiB のとき HDD の 10.7 倍，4KiB のとき 57.2 倍となり，ランダムライトでは，4MiB のとき 10.4 倍，4KiB のとき 107.1 倍となる．小さいデータの読み書きでは HDD に比べ大きな性能向上が得られる．

逐次アクセスでは，1MiB リードが 8.1 倍，1MiB ライトが 7.5 倍の性能であるが，ランダムアクセスでは，リードが 15.6 倍，ライトが 10.4 倍と，逐次アクセスに比べ，ランダムアクセスが HDD に比べ，より有利であることを実証している．

通常の SATA 接続 SSD との比較では，ランダムリードでは 4MiB のとき 4.1 倍，4KiB のとき 25.5 倍，ランダムライトでは 4MiB のとき 2.6 倍，4KiB のとき 51.3 倍の性能となる．1MiB の逐次リードでは 2.9 倍，逐次ライトでは 2.5 倍であるが，ランダムリードでは 4.8 倍，ランダムライトでは 2.6 倍となり，リード性能に違いが出ている．

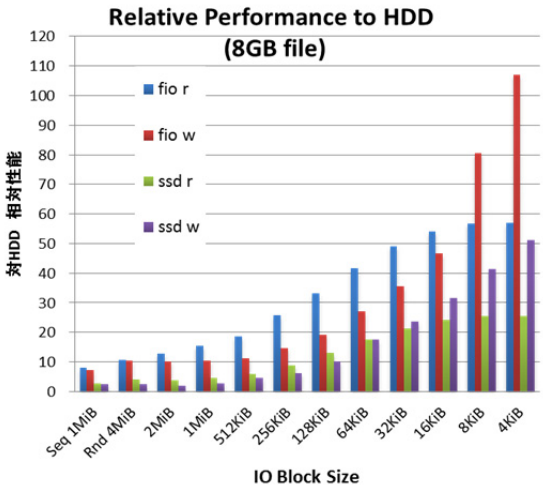

図 2　ハードディスクに対する相対バンド幅性能
Figure 2　Relative Bandwidth to Hard disk.

2.2 バス接続型 SSD に対するアクセス方法による性能差

2.1 では，RW-IO 操作を発行した際のデバイスによる IO 性能について述べた．FIO ベンチマークでは，これ以外の IO 操作として，ファイルの mmap(2)+memcpy(3)+msync(2)(以下，M_SYNC-IO 操作)を指定することができる．これはすなわち，操作対象とするファイルをメモリ上のアドレス空間にマップし，IO サイズ単位で当該アドレスのメモリを読み書きし，その後，ファイルへの同期命令を発行する手

法である．ここでは比較のため，FIO ベンチマークにある M_SYNC-IO 操作に加え，msync(2)を MS_ASYNC で呼び出すことで非同期なファイル同期(データ整合性は保持するが同期タイミングは OS 任せ)による M_ASYNC-IO 操作を追加し，3 種の IO 操作による性能を比較した．この 3 種の IO 操作にて，IO サイズ 1MiB で逐次アクセスで読み書きした場合と，4KiB から 4MiB の IO サイズでランダムに読み書きした場合の IO バンド幅を図 3 に示す．

通常の RW-IO 操作に比べ，ファイル mmap による IO 操作で得られたバンド幅が大きく，M_SYNC-IO 操作では，512KiB ランダムリードが 3326MiB/s，4MiB ランダムライトが 514MiB/s であった．これは，mmap(2)によってカーネル空間とユーザ空間の間でデータのコピーが発生しないことにより IO バンド幅が向上したからである．

M_ASYNC-IO 操作で，512KiB ランダムリードは M_SYNC-IO 操作とほぼ同程度の 3340MB/s であるが，512KiB ランダムライトは 1436MiB/s と約 2.8 倍バンド幅が向上している．更新データを非同期にファイルへ反映する M_ASYNC-IO 操作が，同期で反映する M_SYNC-IO 操作より効率的であることが示された．

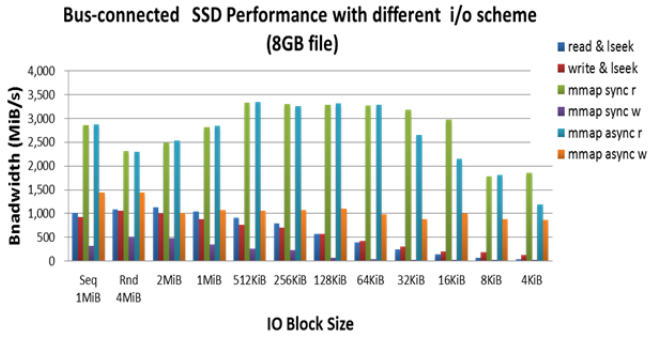

図 3　IO 方法の違いによる性能差
Figure 3　IO Performances with different APIs.

2.3 物理メモリサイズを超えるファイルの IO 性能

2.2 では，物理メモリ 32GiB に対し，ファイルサイズ 8GiB のファイルを対象に読み書きを行う際の IO 性能について調査した．2.3 では，物理メモリを超えるファイルサイズファイルを対象にした場合の IO 性能について述べる．ファイルサイズ 40GiB のファイルを対象に，RW-IO 操作，M_SYNC-IO 操作，M_ASYNC-IO 操作にて，IO サイズ 1MiB で逐次アクセスで読み書きした場合と，4KiB から 4MiB の IO サイズでランダムに読み書きした場合の IO バンド幅を図 4 に示す．

RW-IO 操作の IO バンド幅は，2.1 の場合とほぼ同様，対象とするファイルサイズに依らない結果を得たが，mmap(2)を用いた M_SYNC-IO 操作，M_ASYNC-IO 操作の IO バンド幅は，非常に悪化した．特にランダムリードライトは，サイズに依らず，50MiB/s～60MiB/s 程度の性能に低下した．

2.2 のようにファイルサイズが物理メモリより小さい場合には，新たに更新されたデータをファイルに同期する際，ページキャッシュを利用できるため，IO バンド幅が向上したと考えられる．しかし，物理メモリを超えるファイルサイズを mmap する場合には，ファイルサイズ全体をマッピングできないために，メモリ枯渇状況となり，ファイル IO のためのページキャッシュも用意できない状況となる．

一方，mmap 方式であっても，逐次アクセス（サイズ 1MiB）の場合には，M_SYNC-IO 操作でリード 510MiB/s，ライト 216MiB，M_ASYNC-IO 操作でリード 494MiB/s，ライト 342MiB/s となり，ランダムアクセスほどの性能低下はない．これは，ファイルにマッピングしているページの先読みによる効果が出ているものと考えられる．

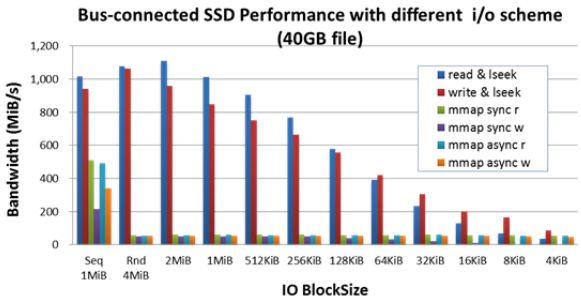

図 4　メモリサイズ超えるファイルの IO 性能
Figure 4　IO Performances for a bigger file than memory size.

3. 物理メモリを超えるデータ処理の性能

ここでは，物理メモリを超えるデータを処理する場合に，バス接続型 SSD を使った場合の性能について調査した．計測に用いたのは，メモリ性能ベンチマーク STREAM[8]，Himeno ベンチマーク[9]，ステンシル処理である．

3.1 メモリ性能ベンチマーク STREAM

STREAM は，メモリバンド幅測定用のベンチマークで，3 つの一次元配列 a[N],b[N],c[N]に対し，COPY(c[i]=a[i]), SCALE(b[i]=k*c[i]), ADD(c[i]=a[i]+b[i]), TRIAD(a[i]= b[i]+k*c[i]) を 1 セットとして繰り返し実行し，それぞれの操作でのメモリバンド幅を出力する．ほとんど計算がなくメモリの読み書きが主体のプログラムであるため，データアクセス速度が大きく実行時間に影響する．

今回は，繰り返し数 10 回とし，配列要素サイズ N を 100M 要素～6.4G 要素（2.2GB～110GB）と変化させ，各サイズでの実行時間（10 回分のプロセスタイム）を，物理メモリサイズ 128GiB と 32GiB で，計測した．物理メモリを超えるデータの割り付けには，3 つ方法をとっている．

(1) malloc 使用＋スワップデバイス：SATA 接続 SSD
(2) malloc 使用＋スワップデバイス：PCIe 接続 SSD
(3) mmap 使用＋PCIe 接続 SSD のファイルを mmap

物理メモリよりもプログラムの使用する仮想メモリサイズが大きい場合，(1)(2)では，メモリを超えるデータ領域にアクセスが起きると，OS の仮想メモリ機構によりスワップデーモンが起動され，スワップデバイスへのページスワッピングが発生する．(1)と(2)は，スワップデバイスとして，SATA 接続 SSD と PCIe 接続 SSD を使った場合の差がどの程度あるのか調査するために行った．(3)は，通常の応用プログラムではあまり行わないが，バス接続型 SSD にあるファイルを指定して，配列サイズに対応するサイズのファイルをメモリ領域にマップするという形で，データアドレス空間を確保してプログラムから利用する．プログラム中で，truncate(2)により対応サイズの初期ファイルを作成して用いる．プログラム終了後には配列サイズ分のデータがファイルに書きこまれた形で残ることになる．この場合，ファイルアクセスのためにメモリを利用しているだけなので，メモリが足りない場合には，随時，ファイルへの同期処理が行われる．したがって，スワップデーモンは通常，起動されず，少ないメモリ（ページキャッシュ）でファイルを読み書きしている時と同じような状況となる．

物理メモリ 128GiB での各サイズの実行時間を基準とし，物理メモリ 32GiB で各サイズの相対実行時間を示したのが図 5 である．ただし，図 5 では，time コマンドで計測したプロセス全体の時間を用いており，3 配列の初期化，初回の配列操作（COPY,SCALE, ADD,TRIAD）も含んでいる．

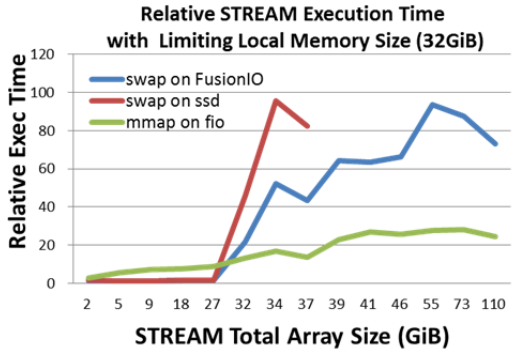

図 5　STREAM　Benchmark
Figure 5　STREAM　Benchmark

これによると，STREAM 配列データが物理メモリ 32GiB に入るサイズの場合には，(1)(2)の malloc によるプログラムが速いが，物理メモリが枯渇するやいなや，スワップデーモンが立ち上がり急激に実行時間は増大する．スワップデバイスとして SATA 接続 SSD を用いると，34GB データを処理する場合，物理メモリ 128GiB を用いたときの約 96 倍の実行時間となる．バス接続 SSD 使用時には 52 倍程度となる．一方，(3)mmap 方式は，データサイズが物理メモリに入る大きさであっても，バックグラウンドでファイルとの同期処理を行うため，物理メモリを 100%利用した場合に比べ，2~8 倍程度実行時間がかかっている．しかし，物理メモリサイズ以上のデータを扱う場合にも，実行時間の増加は比較的穏やかで，110GB データ利用時で，通常実行時の 24.3 倍程度にとどまる．

図 6 は，図 5 の結果をプログラム仮想メモリサイズに対する物理メモリサイズの割合を横軸としてプロットしたものである．比率で示すとスワップデバイス利用がいかに急激な性能低下を引き起こすかが顕著に表れている．

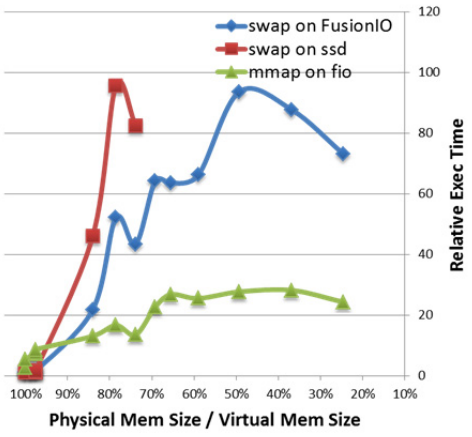

図 6　STREAM　Benchmark
Figure 6　STREAM　Benchmark

3.2 ステンシル処理

ここでは，64K x 64K (double 配列 2 個，64GiB)の 2 次元データにマスク係数を用いて近傍荷重平均化処理を行うステンシル処理について，必要量の半分の物理メモリサイズ(32GiB)で処理した場合を調査した．計算/メモリページアクセス比を変えるため，15x15 と 3x3 の 2 つのサイズのマスク係数を用いている．

計測は，(1)全データが物理メモリに入る場合と，物理メモリ 32GiB で，(2)バス接続 SSD をスワップデバイスとして用いた場合，(3) バス接続 SSD のファイルを mmap した場合について行った．

(1)　mallo/valloc 利用：物理メモリ 128GiB
(2)　file mmap 使用：物理メモリ 32GiB
(3)　malloc/valloc 使用：物理メモリ 32GiB

ここでは，最初の 2 配列の初期化を除き，繰り返し数 2 回分の処理時間を計測した．16 コアを用いて OpenMP で並列処理した場合の 2 種のマスク処理の実行時間を図 7 に示す．いずれのマスクサイズの場合にも，(3)スワップデバイス利用に比べ，(2)ファイル mmap のほうが高速であるが，マスクサイズの小さいほうが，各方式の差が大きく表れている．マスクサイズ 3x3 は，15x15 に比べ計算/メモリページアクセス比が小さく，メモリアクセス性能がより大きく影響するためである．オンメモリで処理する(1)の場合のプロセス全体の平均 CPU 利用率は，15x15 マスクでは 1166%なのに対し，3x3 マスクでは 255%と，16 コアのうち 2.5 コ

ア分しか利用できておらず，ほとんどがデータアクセス待ちになっている．

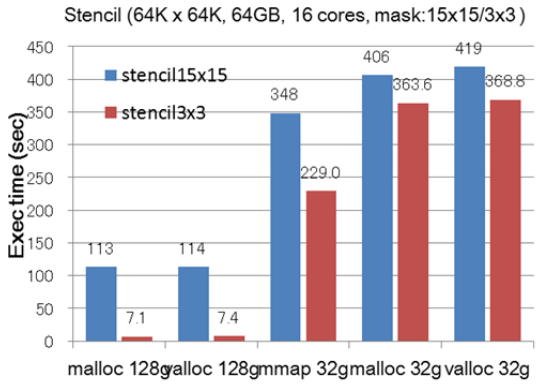

図 7　ステンシル実行時間（マスク 15x15, 3x3, 16 コア）
Figure 7　Stencil Exec Time (Mask 15x 15, 3x3, 16 cores).

各マスク処理のオンメモリでの valloc 利用時の実行時間を基準とした相対実行時間を図 8 に示す．ステンシル処理は，メモリアクセスローカリティの高い処理ではあるが，データ量に対し計算量が少ない処理（3x3 マスク処理）では，オンメモリ時の実行時間の 30 倍から 50 倍に増大し，物理メモリ率 50%による影響は大きい．しかし，データアクセスに比べ十分な計算処理がある応用（15x15 マスク処理）では，3 倍程度の実行時間増大に留まり，その影響は相対的に小さくできることがわかる．

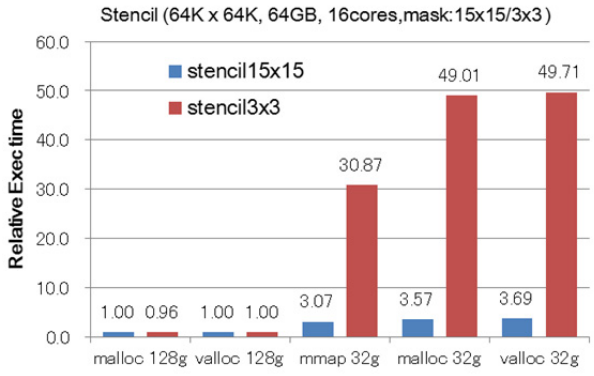

図 8　ステンシル相対実行時間（16 コア）
Figure 8　Stencil Relative Exec Time (16 cores).

2 種のマスク処理を 1 コアによる逐次処理で行った場合の，実行時間と相対実行時間を図 9，図 10 に示す．1 コアの場合には，オンメモリ実行時(1)と方式(1)(2)の差が，16 コア並列処理時に比べ小さい．15x15 マスク処理の場合には，(1)オンメモリ実行と(2)mmap 方式実行の時間差がほとんどなく，半分の物理メモリであっても実行時間に変化が現れない．3x3 マスク処理の場合には，(2)mmap 方式で 1.6 倍に実行時間は増加するが，(3)スワップ方式を用いた場合であっても，最大 5.8 倍程度の実行時間増加にとどまっている．

1 コアのステンシル処理の場合，アクセスされるページはほぼ順番に並んでいて逐次アクセスになる上，近傍要素処理なので，一度に必要とされるページ数もほとんど 1 ページもしくは 2 ページ，まれにある境界部分の処理でも最大 4 ページまでに限られる．

一方，16 コアによる処理では，2 次元配列を列方向に分割したほぼ独立なデータ領域を処理するので，各コアがそれぞれ別のページを必要とし，一度に必要なページ数は 16 倍近くに増大する．したがって計算量の少ない 3x3 マスクの 16 コア処理では，図 8 に示すように，データアクセス性能が大きく影響すると考えられる．

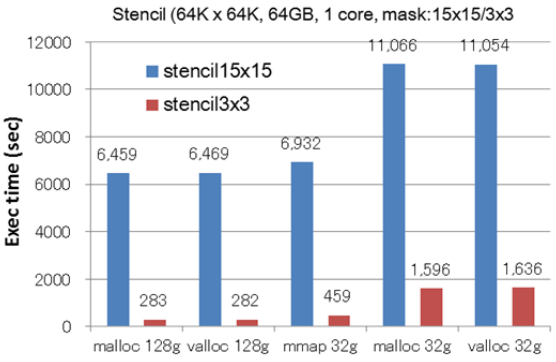

図 9　ステンシル実行時間（1 コア）
Figure 9　Stencil Exec Time (1 core).

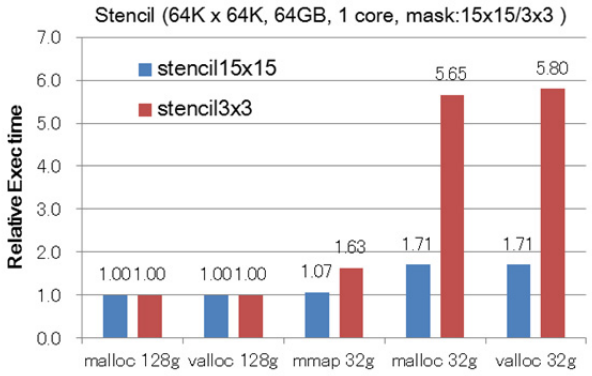

図 10　ステンシル相対実行時間（1 コア）
Figure 10　Stencil Relative Exec Time (1 core).

3.3 Himeno　ベンチマーク

Himeno ベンチマーク(C-OpenMP 版, XXL サイズ作成: 1025x1025x2049・float，112GB)を，3.2 と同様の 3 つの方法で実行した場合の性能を調査した．16 コアと 1 コア（OpenMP オフ）で実行した結果を図 11 図 12 に示す．Himeno ベンチマークは，初回試し実行 3 回と本番計測 1 回の計 4 回の実行を行い，図 11 図 12 の右図は，本番実行の MFLOPS，左図は本番 1 回分の実行時間を示している．32GiB 物理メモリは，プログラムが使うメモリサイズの約 29%に相当する．128GiB の物理メモリで実行した場合と比

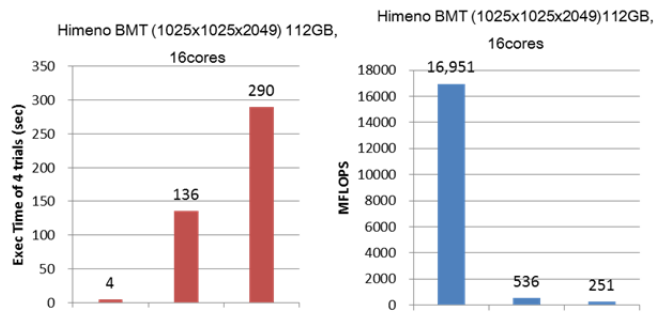

図 11　HimenoBMT 実行時間と MFLOPS（16 コア）
Figure 11　Himeno BMT Exec Time and MFLOPS (16 cores).

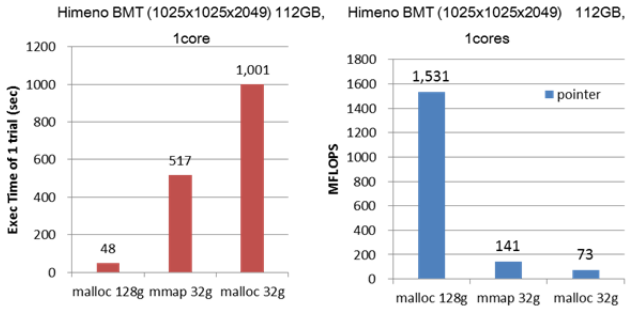

図 12　HimenoBMT　実行時間と MFLOPS（1 コア）
Figure 12　Himeno BMT　Exec.Time and MFLOPS (1 core).

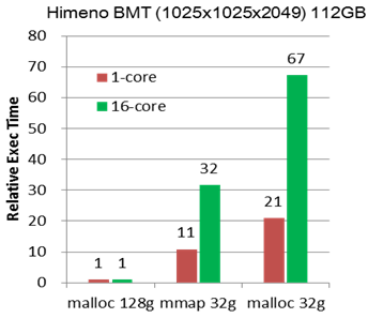

図 13　HimenoBMT　相対実行時間
Figure 13 Himeno BMT Relative Exec.Time

べた相対実行時間を図 13 に示す.

3.3 のステンシル処理の場合に比べ，(2)(3)方式のいずれも性能が大きく低下する．前述のようにステンシル処理では, 各コアがアクセスするページはほとんど 1～2 枚程度に限られる．しかし Himeno ベンチマークでは，ループ中で用いるデータ配列が 7 種類に及び，しかも 4 次元配列など次元が大きいため，大規模サイズ問題では，隣接インデックスであっても，次元方向によっては異なるページのアクセスが必要となる．したがって，1 ループあたり 34 回の演算に対し，アクセスするページ数はステンシル処理の 10 倍以上となり，データアクセス性能が，全体性能により大きく影響する．

ステンシル処理と同様に，　1 コア実行に比べ，16 コア実行はオンメモリ実行に対する性能低下が大きい．複数コアから，より多くのページ要求が発生するため，データアクセス性能が実行時間に影響を及ぼす.

4.　おわりに

本報告では，PCIe バス接続型 SSD に関して，2 つの予備的な調査を行った．基本 IO 性能を調査した結果，実験に用いた ioDrive2 の性能は，HDD に比べて，4MiB～4KiB データのランダムリードで 10.7～57.2 倍，ランダムライトで 10.4～107.1 倍になることがわかった．SATA 接続 SSD(A1 SSD)との比較では，ランダムリードで 2.6～3.3 倍，ランダムライトで 1.5～4.7 倍の性能に匹敵する．

また，現時点で利用可能な汎用 UNIX API を用いて，物理メモリサイズを超えるデータの処理に PCIe バス接続型 SSD を利用する場合には，通常よく用いる malloc＋スワップデバイスとして使うよりも, mmap＋マップファイルとして使うほうが効果的であることが確かめられた．従来から指摘されているように，OS のスワップデーモンの起動は急激な性能低下を引き起こす．このため，同じメモリサイズであれば，ファイル IO のためのページキャッシュとして，穏やかにメモリを利用するほうが効果的といえる．

物理メモリ容量不足分を補う用途としてバス接続型 SSD を用いる場合，その性能は，処理のページアクセス（アクセスページ数やコア間の共有度合，ページアクセスパターンなど）と計算負荷の比率に影響される．アクセスページ数に比べ計算負荷の比が高い場合には，相対的に性能低下を小さくでき, 実用上許容できる範囲にすることもできる．複数コアからの利用であれば，複数台の RAID 構成で用いることも，性能･コストの上で有利になると考えられる．

参考文献

1)　F2FS
http://www.phoronix.com/scan.php?page=news_item&px=MTI1OTU
2) LINUX CON 2013 JAPAN
http://events.linuxfoundation.jp/events/linuxcon-japan/program/presentations
3) ioDrive2（FusionIO 社）
https://www.fusionio.com/products/iodrive2/
4) Violin memory
http://www.violin-memory.com/i/
5) Intel SSD910
http://www.intel.com/content/www/us/en/solid-state-drives/solid-state-drives-910-series.html
6) Flush Summit
http://www.flashmemorysummit.com/
7) Flexible IO Test, FIO　ベンチマーク
http://freecode.com/projects/fio
8) STREAM ベンチマーク
http://www.streambench.org/
9) Himeno　ベンチマーク
http://accc.riken.jp/2444.htm