SC13 参加報告

實本英之　田浦健次朗　大島聡史　中島研吾　片桐孝洋　松本正晴
東京大学情報基盤センター

東京大学情報基盤センターの教職員が，2013年11月17日から22日まで，アメリカのコロラド州デンバーにて開催されたSC13 (The International Conference for High Performance Computing, Networking, Storage and Analysis) に参加した．本会議は，高性能計算(HPC)分野では著名な国際会議であると共に，様々な情報技術関連企業の技術展示会でもある．本稿はその参加報告である．

1　はじめに

SC13の会場となったデンバーはコロラド州最大の都市であり州都である．SC12のソルトレイクシティにつづいて雪の舞う街での開催となった．デンバーは標高が約1マイル（約1600メートル）に位置するマイル・ハイ・シティーとしても知られており，この標高は東京近郊では例えば尾瀬ヶ原や奥日光に相当する．この標高のためか，荷物を持って歩き回ると妙に疲れるという声も聞こえていた．会場であるColorado Convention Centerは発展した市街地に位置し，周囲にはモール街や多数のホテルを備えており，交通の便も良いと感じた．

今回の会議では，昨年度に引き続き，ビッグデータに対する処理や省電力(Green Computing)など，エクサスケールスーパーコンピューティングに向けた，アルゴリズム面，ハードウェア面双方の動向を主な展示内容としていたように思う．

また本年度は会議設立25周年ということもあり，会議主催側を含めいくつかのブースでは歴史を振り返るような展示も行われていた．また，同トピックを主眼としたパネルディスカッションも行われた．

2　SC-XYについて

本会議は以前はSupercomputing-XY(XY:開催年)という名称で，1988年フロリダ州オーランドで第1回が開催されてから，毎年11月にアメリカ各地を転々としながら開催されている．SC-XYという名前に変わったのは1997年で，Supercomputing-88から数えて，今回で25回目の開催である．

会議は，毎朝行われる基調講演や，研究発表，今後のトレンドを占うBoF(Birds of a Feather: 特定のトピックを定めた小規模集会)やパネル討論，主要技術の理解を助けるチュートリアルなどで構成されている．

また，企業や各種研究機関による，最新の製品，技術の展示発表も注目すべき内容である．

3　研究機関／企業展示

今年度の来場者数は 10,550人，58カ国からの参加があった．展示については参加団体数は企業197件，研究機関142件と，昨年より増加している．展示内容としては先にも述べたとおり、エクサスケールスーパーコンピューティングに付随した，高密度ストレージや，ファイルシステムの展示を始め，空冷以外のノード冷却手法などが主体となっていた．昨年に引き続き，メ


図 1　会場および展示の様子（FUJITSU ブース)


図 2　Oakleaf-Kashiwa Aliance ブース全景と集合写真

ニーコアアーキテクチャの展示も散見され，近年開発されてきた新技術を製品に落とし込み，エクサスケールのスーパーコンピュータへの着実な道筋が感じられた．

4　東京大学情報基盤センターによる展示

東京大学情報基盤センターは，SC12に引き続き，東京大学物性研究所と合同で Oakleaf/Kashiwa Alliance としてブース展示を行った．展示内容は，当センターの保有する計算機システムに関する情報、各種プロジェクトや教員の研究内容に関するポスターの展示を主体とし，さらに広報資料の配付やブース内でのショートプレゼンテーションを実施した．また昨年に引き続き「レイテンシコアの高度化・高効率化による将来のHPCIシステムに関する調査研究」による次世代スーパーコンピュータに向けた研究開発についてもポスター展示を行った．さらに今年は筑波大学と共同で設置した「最先端共同HPC基盤施設」(JCAHPC)についてもポスターを用意し，Oakleaf/Kashiwa Allianceブースと筑波大ブースの双方で掲示を行った．

5　基調講演

本年度の初日の基調講演では，文化人類学者である Genevieve Bell女史を迎え，”The Secret Life of Data” という講演を行った．女史はこの中で，昨今取り沙汰されているビッグデータの処理は，1000年以上前から我々人類に取り上げられ有用とされてきており，人間の営みそのも

図 3　来場者へ研究内容を説明する説明員

図 4　プレゼンテーション

のだという．その例の一つとして，10世紀，イギリス王室の開祖ウィリアム1世の御代中に行われた国民調査について挙げた．これは徴税のために誰が，どのような資産（土地や家畜）を保有するか調べたもので，集められたデータは Domesday Book と呼ばれる本に記されており20世紀まで利用された．女史はこの台帳の運用が現在のビックデータと同じ：1）データ集積（国民調査），2）データ運用のフレームワーク（Doomsday Book），3）データの評価（査定）の3要素を持っていると述べた．現在のビッグデータも同様に事実の収集，解析と可視化，評価アルゴリズムとして3要素を持っているが，その複雑さは1000年前の比ではなくなっておりコンピュータによる機械的な処理が欠かせないが，このデータの評価から何を得るかは人間的な作業が必要になると述べた．

このほかにも，Warren Washington 氏の ”Climate Earth System Modeling for the IPCC

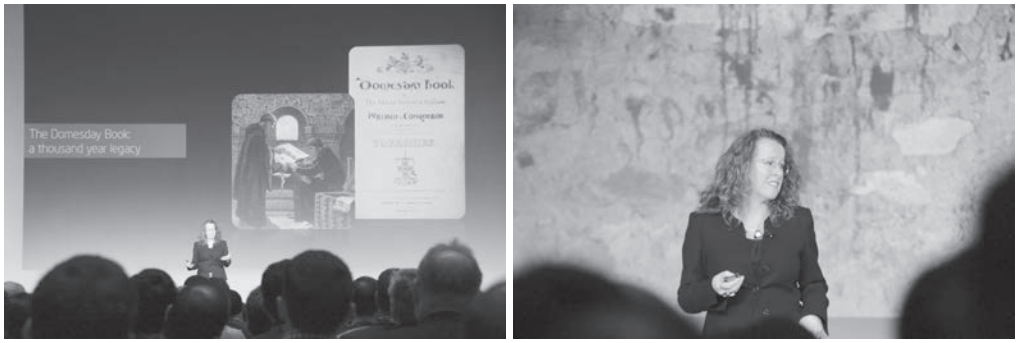

図 5　基調講演・招待講演の模様

Sixth Assessment Report (AR6): Higher Resolution and Complexity”，Saul Perlmutter氏の ”Data, Computation, and the Fate of the Universe”，Alok Choudhary氏の ”BIG DATA + BIG COMPUTE = An Extreme Scale Marriage for SMARTER SCIENCE”，Vern Paxson氏の ”The Interplay Between Internet Security and Scale” の4本の全体講演が行われ全体的にビッグデータ（データ処理）に関するものが多く取り上げられていると感じた.

6　Technical Paper

SC13のテクニカル論文は, 以下の9つの分野に分けて査読が行われた. 括弧内は所属したプログラム委員の数である.

1. アルゴリズム(33)
2. アプリケーション(32)
3. アーキテクチャとネットワーク(25)
4. グリッドとクラウド(22)
5. 性能, 分析, ツール(28)
6. プログラミングシステム(31)
7. 先端的な実践(14)
8. ストレージ, 可視化, 分析(28)
9. システムソフトウェア(24)

ひとつの分野に20-30人くらいのプログラム委員が所属しており,それぞれが単独のカンファレンスのような規模である.このため Face-to-face で行われるプログラム委員会が,あたかもカンファレンスのような感じである. 投稿は分野を指定して行われるが,論文のタイトルやアブストラクトを見て, より適切な分野へ移動することもあり,採否はあくまでプログラム委員全体で合意される.

総計で457件の投稿があり, その中から90件が採択された.さらにその中から13件が, 論文賞(best paper awards), 学生論文賞(best student paper awards)の候補となった. 3日間にわたり, 30のセッションが開催され,同時期には3つのセッションが並行して行われる.議論された分野を概観するために, セッション名を列挙する.

1. Fault-tolerant computing
2. GPU programming
3. Load balancing
4. MPI performance and debugging
5. Memory hierarchy
6. Memory resilience
7. Optimizing numerical code
8. Parallel performance tools
9. Parallel programming models and compilation
10. Performance analysis of applications at large scale
11. Performance management of HPC systems
12. System-wide application performance assessments
13. Tools for scalable analysis
14. Data management in the cloud
15. Graph partitioning and data clustering
16. Inter-node communication
17. Cloud resource management and scheduling
18. Energy management
19. Extreme-scale applications
20. Fault tolerance and migration in the cloud
21. IO tuning
22. Physical frontiers
23. Optimizing data movement
24. In-situ data analytics and reduction
25. Preconditioners and unstructured meshes
26. Engineering scalable applications
27. Improving large-scale computation and data resources
28. Matrix computations
29. Sorting and graph algorithms
30. Application performance characterization

興味深い論文は多数あるが, そのうちのいくつかを以下で紹介する.

6.1 論文紹介

6.1.1 木構造走査を GPU で実行するためのプログラム変換

- 原題: “General Transformations for GPU Execution of Tree Traversals”
- 著者: Michael Goldfarb, Youngjoon Jo, and Milind Kulkarni (Purdue University)

本論文では, 同一の木構造の一部を多数回走査するアルゴリズムにおいて, 「多数回」の走査を効率良くGPUで実行するためのプログラム変換について述べている. 手法はあくまでGPU

(CUDA)向けのものだが, 多数回の走査を最大限に最適化しようと思えばCPUでも似た様な問題が現れる. 本論文はそれを解決する一般的な手法として興味深い.

提案手法の対象となるプログラムは, 擬似コードで書けば, 以下のような形をしたものである.

```
1 TreeNode root = ... // 木構造の根ノード
2 forall (Point p : points) {
3   recurse(p, root); // 木を走査
4 }
```

`recurse(`p`, `n`)`が, 点pに対して木をnから操作する関数で, 素直には以下のような再帰呼び出しを用いて書かれるものである.

```
1 void recurse(Point p, TreeNode n) {
2   if (truncate(p, n, ...)) return;
3   foreach (TreeNode child : n.children()) {
4     recursve(p, child, ...);
5   }
6 }
```

`recurse(`p`, `n`)`が木構造のどの部分を走査するかが, pに応じて異なりうる.

本論文はこのようなパターンのコードを対象として, それをGPUで効率良く並列実行する(外側の`forall (Point p : points) ...` ループを, 並列実行する)ためのプログラム変換を提案している.

このようなパターンのコードは, Barnes-HutのN体問題アルゴリズムにおける力の計算, 相関関数(点の集合に対し, ある距離の範囲にどれだけの点の対が存在するかを記述する関数)の計算などで現れる.

上記のforall文を実行するだけならば, CUDAで一回の走査をカーネル関数としてそれを必要な数だけ実行すれば良い. しかしそれではスレッド実行のdivergenceが激しくなる. nVidia GPUにおいてはそれは, warpというスレッドの最小のグループ(32スレッド)内で並列性が損なわれるという結果をもたらす. 具体的には, ひとつのwarp内のあるスレッドにおいては走査される部分木が, 別のスレッドにおいては走査されなかった場合, 後者は, 前者がその部分木を走査し終えるまで待つという挙動になる.

同様の問題は, CPUにおいてもSIMD並列性を活かそうと思うと発生する. 具体的には, 異なる点に対する走査を並列に実行するために, SIMD命令を用いると発生する.

提案手法の要点は, 上記のような再帰呼び出しを用いたコードを, 明示的に今後走査すべきノードをスタックで管理するようなループに自動的に変換することである(autorope). そのようにすることで, あるスレッドがある部分木を走査している間に, 同じwarp内の別のスレッドが別の部分木を走査する, という動作が実現できる. 一方でこの方式には, warp内のスレッドが木の異なる部分を同時にアクセスすることになり, メモリコアレシングを阻害するという逆の作用がある. そのため本論文では, warp内のスレッドdivergenceを犠牲にして同じ部分を走査する(ロックステップ実行)方式と, メモリコアレシングを犠牲にして各スレッドに別の部分を走査させる方式を比較している. 実験結果は総じて, 前者のほうが良いようである.

CPUでスレッドを用いた並列化との比較も行なっている. GPUはnVidia Tesla C2070 (32 core × 14 ストリーミングマルチプロセッサ), CPUはAMD Opteron 6176 (12 core × 4 プロセッサ)の比較である. 優劣はアルゴリズムやデータセットにもよるが, 32 CPUコアを用いた場合との比較において, 大体5分5分程度である.

発表後, ところでCPU版はSIMD並列化されているのかと質問したところ, していないということであった. CPUの性能を引き出していない実装との比較という事になるが, 一方でそれを行うためにはCPUにおいては, SIMDレーン内のdivergenceを管理する必要が生ずる. そのためのコード生成はGPU (CUDA)を対象にした時よりも面倒である. GPUにおいてはwarp内のdivergenceで性能は劣化するものの, コード自身はスレッド並列化のためのコードと同一でよい. CPUの場合, 少なくとも現状のコンパイラでは, スレッド並列と, SIMDはそれぞれ明示的に行う必要がある. その意味ではGPUないしCUDAの, SIMT (Single Instruction Multiple Threads)と呼ばれるプログラングモデルのメリットが生かされていると言える. もっとも本研究ではコードは自動変換によって生成されているのだから, CPU用にSIMD化されたコードを出すにあたって, 本質的な困難はないと思われる.

6.1.2 階層キャッシュを持つメニーコアプロセッサにおける明示的なキャッシュ管理

- 原題: Location-Aware Cache Management for Many-Core Processors with Deep Cache Hierarchy
- 著者: Jongsoo Park, Richard M. Yoo, Daya S. Khudia, Christopher J. Hughes, and Daehyun Kim (Intel Corporation, University of Michigan)

本論文は, キャッシュ間のデータの移動を細かく, かつ可搬性高く制御できるプロセッサの拡張について述べている. ここで, 可搬性とは, プロセッサの世代や機種が変わることで, キャッシュの段数や容量が変わっても, 同一のコードが動き, かつ個々のプロセッサに合わせた性能を発揮する(性能可搬性)という意味である.

背景は言うまでもないが, データの移動コストを少なくすることは, ノード間, ノード内, コア内など, あらゆるレベルで重要な課題である. そしてそれは, 性能, 消費電力の両方にとって重要である. ノード内においてはデータの移動とはすなわち, メインメモリ, キャッシュ階層, プロセッサ間のデータの移動のことであり, つまりはどのデータをどのキャッシュ(階層およびプロセッサ) に配置するかで定まる. 現状, キャッシュ間のデータ移動はソフトウェアの介在する余地が少なく, プロセッサに実装されている単一のアルゴリズム(セットアソシアティブ, 近似的なLRU, ハードウェアプリフェッチ) で管理される. ソフトウェアの介在する余地としては, プリフェッチ命令, キャッシュをバイパスするstreaming load/store などがある. 特に後者は, キャッシュに収まらないとわかりきっているような大きなデータが, 繰り返し利用される小さなデータをキャッシュから追い出すことを防止する. 一方で, 今日の深いキャッシュ階層を持つマルチコアプロセッサにおいては, 通常のload/storeまたはstreaming load/storeのような2者択一では十分な制御が行えない可能性がある. 例えば繰り返しアクセスされるあるデータが, L3キャッシュには収まるがL2には収まらないというような場合, L2に割り当てる意味が乏しいがL3に割り当てる意味は大いにある. そのため, L3のみ割り当てるような制御が行いたくなる. 一方でそのような細かい制御を行う命令を, 単純に追加していくだけでは, プロセッサのキャッシュ階層や容量が変わった場合に可搬性を確保することが困難になる.

その他のソフトウェアによる制御の余地として, コア間通信のための書き込みなどもある. コア間通信のための書き込みでは, 将来そのデータを用いるプロセッサのキャッシュへ割り当てるなどの制御が行いたくなる.

本論文ではそのための命令セットの拡張を2つ提案し, 評価している.

- unallocating_load reg,[mem],reuse_distance
- pushing_store [mem],reuse_distance,core,time

unallocating_loadは通常のload命令と同じだが, reuse_distanceというヒントを与えることができる(reuse_distanceはオペランドとしてはレジスタを指定し, そのレジスタ内に数値を入れておく). プロセッサはこの値よりも大きい容量を持つキャッシュに対しては, キャッシュラインを割り当てる. 一般に, reuse distanceとは, 同じデータが次に利用されるまでにアクセスされる, 異なるデータの量をあらわす. たとえば1MBのデータを繰り返しスキャンする場合, 同じデータへのアクセスの間に, 約1MB分の異なるデータがアクセスされることになるから, reuse distanceは大体1MBということになる. この場合, 1MBより小さいキャッシュに対しては, データをキャッシュ中に保持しても無意味である. reuse distance自身は, アルゴリズム自身の特徴と言ってよく, 可搬性高く与えることができる.

pushing_storeの方も同様にreuse_distanceを指定できるほか, どのcoreのキャッシュに対してラインを割り当てるか(自分, 特定の1コア, またはそのアドレスに対するラインを持っている全てのコア) を指定できる.

より高水準のインタフェースとして, C言語のpragmaを提案している. それは, Intel C Compilerがすでに持っている, `#pragma nontemporal(A)`というプラグマを拡張し, reuse distanceを指定できるようにしたものである. 例えば,

```
1  int *A = malloc(sizeof(int)*BIG);
2  ...
3  #pragma nontemporal(A(reuse_dist=BIG))
4  for (i = 0; i < BIG; i++)
5    for (j = 0; j < LITTLE; j++)
6      sum += A[i]*B[j];
```

のようにして, Aに対するアクセスがunallocating loadを利用するよう, 指定する.

評価にはマイクロベンチマークの他, NAS Parallel BenchmarkのCG, PARSECのstream clusterなどを含む9のプログラムが用いられた. 性能向上の効果はほぼ0から, 最大で30%程度, 平均すると, 7% 程度であった. また, メモリの消費電力を平均で24% 程度削減したとある.

本論文は, 深くなるばかりのキャッシュ階層に対して, 細かい制御と可搬性を達成するための提案として, 理にかなっていると感じる. 一方, (本論文でも詳しく述べられている通り)同じ問題意識は古くからあり, そのために, updateプロトコルやキャッシュの拡張が提案されてきた. しかしそのような拡張が商用のプロセッサに採用されているという話は, 聞いたことがない. それらと比べて本論文の提案が特別, プロセッサ設計者に実装する意欲を沸かせるものなのかは, プロセッサ設計の専門家ではない筆者にはわからないが, updateプロトコルのように, キャッシュ一貫性プロトコルを複雑にするような拡張がなかなか実装されることがないのはもっともに感じる. 一方のreuse distanceを利用して, 割り当てるキャッシュのレベルを決めるというインタフェースは, 実装もそれほど複雑化せず, 美しく理にかなっていると感じる.

一方, reuse distanceを用いたインタフェースが本当に(実際に高性能を発揮するか以前に), 性能可搬性を持ったインタフェースなのかという点は, 若干の疑問も感じる. たとえば,

```
1  for (t = 0; t < T; t++) {
2    for (i = 0; i < n; i++) {
3      A[i]++;
4      B[i]++;
```

```
5   }
6 }
```

において, A[i], B[i] へのアクセスのreuse distance は, Aのサイズ + Bのサイズである. では, Aのサイズ + Bのサイズとキャッシュサイズを比較して両者の割り当て方針を決めれば良いのかというと, そうではない. 仮にAをキャッシュに割り当てないと決めれば, Bに対するよい方針は, 「Bのサイズ < キャッシュサイズ」であるか否かでキャッシュを割り当てるか否かを決める, というものになるだろう.

言い換えれば, reuse distanceが可搬性のある(実行環境によらない)尺度である, というのはそのとおりだが,「reuse distance < キャッシュサイズ $\Longleftrightarrow$ キャッシュに割り当てる」という方針には可搬性がない. この方針の妥当性に, 他のデータがどのようにキャッシュに割り当てられるかに依存する. 単純に全てのアクセスに対してキャッシュに割り当てるならば, reuse distanceは, 正しく次にアクセスされるまでに, 追い出されるキャッシュのデータ量を近似するが, 一部のデータがそもそもキャッシュに割り当てられないならば, それらの分までreuse distanceに含めるのは正しくない. このようにして, 指定すべきreuse distance自身が, キャッシュにどのデータが実際に割り当てられるのかに依存してしまうわけである.

そこで結局, プログラマに個々のデータをキャッシュ「する・しない」を直接指定させる—現状のstreaming load/storeで足りないならば, さらにレベル指定のパラメータを指定すべく拡張する—という方式と比べて特段の優位性があるのかという疑問も感じる. それ以前に, 現実として, キャッシュ容量や段数に依存した性能可搬性のかけらもないプログラムがすでに今の世の中でも書かれているという現状もある.

6.1.3 100 万ファイルシステム IOPS を廉価・普及品のハードウェアで実現する

- 原題: Toward Millions of File System IOPS on Low-Cost, Commodity Hardware
- 著者: Da Zheng, Randal Burns Alexander, and S. Szalay.

FLASHベースのストレージは, I/Oボトルネックを解消する, 特にランダムなIOの性能(秒あたりのIOの「回数」; IOPS) を向上させる技術として, 不可欠なものとみなされるようになっている. 本研究は, 廉価なSSD + software RAIDで100万IOPSを達成するファイルシステムについて述べている.

背景として, 普及品のSSD単体のIOPSは100K程度である. SSDはFLASHデバイスを用いているが, ホストとの通信は, HDDと同様の, SATAやSASなどのストレージプロトコルを用いる. これに対しFusion-io社のioDriveのように, PCIe経由でFLASHデバイスにアクセスする特別なプロトコルで, 高いIOPS (1M IOPS程度)を達成するデバイスもある. また, いくつかのベンダが提供しているFLASHのarrayでも, 数百K IOPSから, 1M IOPSを広告している. しかしこれらの製品は普及品のSSDに比べ, 容量あたりの値段は高価である. また, それら数値はあくまでデバイスの限界性能を示しているに過ぎず, ファイルシステム経由のIOPSは必ずしも同等の値にはならない.

本論文は, 普及品のSSDを多数, ソフトウェアのみで結合し, **ファイルシステム経由のアクセスで**1M IOPSを達成する実装について述べている. ただし, 公平のために述べると, 現時点では完全なファイルシステムではなく, ディレクトリをサポートしないなど, ファイルシステムと

してはかなり不完全なものである. それを反映し, SSD file system abstraction (SSDFA) という名称が使われている.

本論文のポイントは, ファイルシステム内部の詳細なデータ構造設計ではなく, ファイルシステム, ページキャッシュ関連でカーネル内に存在する, スケーラビリティの阻害要因を指摘し, それを取り除くことである. それにより, 普及品のSSDをソフトウェアでアレイ化するだけで, 高並列なワークロードに対して高いIOPSが達成できる可能性を示した点に意義がある.

Linuxのファイルシステムがスケールしない原因として,

- ファイルシステム内のロック衝突
- ページキャッシュ管理に用いられている粗粒度なロック
- NUMAマシンでのリモートメモリへの頻繁なアクセス
- デバイスからの割り込みが特定のコアに集中すること

などをあげている. また, デフォルトのIOスケジューラである, フェアキューイング(Completely Fair Queueing; CFQ)が, SSDではそのオーバーヘッドの大きさから逆効果であることも指摘している.

本論文で取られている実装はこれらの問題の解決に的を絞ったもので, ファイルシステム内の詳細なデータ構造に立ち入った部分は少ない. 実際SSDFAは, 既存のファイルシステムの上に, ユーザレベルライブラリとして構築される. まず, 個々のSSDデバイス上に, 通常のファイルシステムを個別に構築する. 実験ではXFSを用いているが, ext3/ext4などでも良いと述べられている. それらをさらに上位のレイヤで束ねる(ストライピングする)ことで, SSDFAが構築される.

まず, 一つのSSDにつき一つ, 専用のIOスレッドを用い, SSDが直接アタッチされているプロセッサ上のコアへbindすることで, ロックの衝突やそれに伴うリモートメモリアクセスを減らしている. また, ページキャッシュを単一のロックで保護するのではなく, set associativeキャッシュにし, setごとに並列に変更可能とした. 割り込みが集中する問題は, PCI 3.0に導入されたMSI-Xを用いて, 割り込みをかけるコアを指定することで解決する. IOスケジューラの問題は, Linuxで標準サポートされているnoopスケジューラを用いることで回避する.

もちろんユーザがアクセスを要求したファイルが同一プロセッサ上のIOスレッドで管理されているSSD 上にあるとは限らないため, その場合は異なるIOスレッドにリクエストが転送される. 通常のファイルシステムとの本質的な違いは, グローバルに共有されたデータとロックの変わりに, コアごとのIOスレッドと, IOスレッド間のメッセージで実装されている点にある.

実験で構成されたマシンは,

- Intel E5-4620 (8コア)を4プロセッサ
- LSI SAS 9217-8i ホストアダプタ (8つのSAS/SATA ポートを持つ) を3つ
- それらを通じて接続された16個のOCX Vertex 4 SSD

を搭載している.

実験では, 512バイト単位のランダムな読み出しで1.22M IOPSを達成している(ioDriveは1.19M IOPS). 64kB単位の読み出しと書き込みのスループットはそれぞれ6.8GB/sec, 5.6GB/secでいずれもioDriveを上回っている.

また, 既に利用可能な他の方式との比較では以下が示されている. 以下では, IOの単位は4KBである.

- SSDFAは700K - 900K IOPSを達成している
- Linux標準のsoftware RAIDでも, 1ソケット構成(全てのIOスレッドが一プロセッサ上で実行される) であればSSFDAとほぼ同等の性能が出る(700K IOPS程度)が, 複数ソケット構成(NUMA)では性能が半分程度になる
- EXT4ファイルシステムは読み出し・書きこみ共に200K IOPS程度
- XFSは1ソケット構成で, 読み出しに対してはSSFDAとほぼ同等であるが, 書き込みは100K IOPS程度

まとめると, NUMA構成のマシンでは, SSDFAに比肩する性能を持つ構成はなく, SSDFAは次善の方式と比べても読み出しで2倍, 書き込みで5倍以上のIOPSを達成している.

高いIOPSを達成するデバイスは現在もベンダ間の競争が行われており, 各ベンダが数字を広告している. 本論文は, そのようなベンダの数字が飛び交う中で,

- ファイルシステム経由でのアクセス性能を評価していること
- 普及品のSSDでハイエンドのストレージ機器と同等のIOPSを達成していること

など, 意義深いものである. 現時点では完全なファイルシステムの実装ではない点など, 他の方式と完全に公平に比較できるわけではないところもあるが, 本論文で示されている実験結果自体が十分に興味深い.

7 Grand Challenge

7.1 TOP500(HPCC)/Green500

Top500 List (http://www.top500.org/)は，世界のスーパーコンピュータの性能を，LINPACKという，係数行列が密行列の連立一次方程式を解くベンチマークの処理速度によって競うものである．1993年の開始以来，6月にヨーロッパで行われる会議であるISCと，本会議SCにて年2回の更新を続けている．また，TOP500の結果から，電力当たりのLINPACK性能を比較したランキングとしてGreen500(http://www.green500.org/)がある．今回のTop500Listは，TOP10については非常に変化に乏しく，6位にスイスの計算センターに導入されたCrayのGPUクラスタが入った以外は半年前のリストと変わらない状況であった．そのため，Top500のBoFは盛り上がりに欠けていたように感じた．

この結果発表に加え，今回のTop500 BoFでは今後のTOP500についての発表が行われた．現在Top500で用いられているLinpackベンチマークは計算機システム全体を数時間から1日程度使用して測定する必要があり，消費電力・消費エネルギーが大きいことが問題視されている．そのため別のベンチマークによる置き換えが検討されており，今回はその有力な候補とされているHPCGベンチマークについての情報が提供された．HPCGはCG法による反復計算を用いるベンチマークであるが，前処理が実行時間に与える影響が大きいと予想されることなどからスパコンの計算性能を比較するという意味にも増してプログラミングコンテストの様相を呈するのではないか，などの意見も上がっている．いずれにしても次回のTop500Listをいきな

りHPCGで置き換えようというようなものではなく，いましばらくは既存のLinpackを用いたTop500とあわせて用いられることになると見られている．

Green500では，東京工業大学のTSUBAME-KFCとTSUBAME-2.5がそれぞれ1位と6位，筑波大学のHA-PACSが3位にランクインした．特にTSUBAME-KFCは2位以下を大きく引き離して1位を獲得したため，大きな注目を集めた．TSUBAME-KFCは，従来の計算機が空気や水を用いて冷却を行うのに対して，油を用いて冷却を行うという実験システムである．Top500Listの順位では311位という小規模なシステムではあるが，圧倒的な電力効率を見せつける結果となった．

7.2 Graph500

Graph500は，上位のシステムにおいてはスコアはともかくとして，ランキング自体にはあまり大きな動きが見られなかった．一年前に13位，半年前に18位と推移していたOakleaf-FXは，再び13位に戻っている．Graph500の電力効率版であるGreen Graph 500では，Green500で活躍したTSUBAME-KFCがBig Data版で1位を獲得した．これは中央大学の研究グループにより開発されたコードを用いた成果であるが，より小さな問題サイズで競うSmall Data版でも同じ中央大学の研究グループがリストの上位を独占する結果となった．なおGreen Graph 500 ListはBig Dataで7システム，Small Dataで21システムが登録されている．

7.3 各種表彰

その他にも，SC-XYでは計算科学における業績について様々な表彰が行われる．高性能計算による大きな社会貢献に対して与えられる 1) Ken Kennedy 賞，HPCシステムへの革新性に与えられる2) Seymour Cray 賞，大規模問題に対する革新的なアプリケーション開発に与えられる3) Sidney Frenbach 賞，科学計算アプリケーションに対する高度な実性能および成果に与えられる4) Gordon Bell 賞などがオープニングでも表彰される代表的なものである．これらに加え，本年度は 計算科学に大きな貢献を行った女性研究者に贈られる5) Athena Lecturer 賞が設けられた．受賞者は 1)Jack Dongarra氏(テネシー大学)，2) Marc Snir氏（アルゴンヌ国立研究所)，3) Christopher R. Johnson氏(ユタ大学) 4) Diego Rossinelli氏ら（チューリッヒ工科大学，IBM)，5) Katherine Yelic女史（カリフォルニア大学バークレー）となった．

8 おわりに

本年のSC13は昨年示された今後の計算機のビジョンを実用性のある製品にまで洗練したという感覚を受けた．中でもビッグデータ処理のようなデータ主体の大規模演算を強く意識させる構成だったと思う．来年のSC14は2014年11月16日から21日にかけてルイジアナ州ニューオリンズで開催される予定である．

図 6　開会式と表彰の模様