Vol.2019-HPC-172 No.21
2019/12/19

# SX-Aurora TSUBASAの入出力性能の評価

中井 彩乃[1,a)]　横川 三津夫[2]　小松 一彦[3]　渡辺 裕太[4]　磯部 洋子[4]　政岡 靖久[4]　小林 広明[5]

**概要**：一般に, 時間発展のシミュレーションにおいては, ポスト処理のために計算結果を適切なタイミングで出力する (チェックポイント) ことが多い. すなわち計算結果のファイルへの入出力処理 (I/O 処理) が実行されるが, データが膨大な I/O 処理には当然時間がかかる. したがって, プログラムの高速化のためには, ファイルの I/O 時間の短縮を図る必要がある.
本稿では, ファイルの I/O 処理の効率化を目的として, ベクトル型スーパーコンピュータ SX-Aurora TSUBASA A300-4 と A300-8 における通常の I/O，及び高速 I/O, VH-VE 間の offload 機能を利用した I/O を用いた場合について実行時間を計測し, システム構成と I/O 性能の関係について考察した. この結果, 高速 I/O を用いた場合, A300-4 では 4VE, 8 コア全てから 1GB サイズのファイルを出力した際に約 30GB/s の性能が得られ, A300-8 では 8VE, 8 コア全てから 1GB サイズのファイルを出力した際に PCIe の帯域のほぼ上限値である 10GB/s ×2 の性能を得た. よって, A300-4 と A300-8 のどちらにおいても, 高速 I/O 機能の性能の効率性が確認できた.

**キーワード**：HPC, SX-Aurora TSUBASA, ファイル入出力, 性能評価, 高速入出力性能

## 1. はじめに

大規模シミュレーションは, 多くの分野において非常に重要な技術である. 例えば, 天気予報や地球温暖化予測に対する気象・気候シミュレーション，地震や津波による災害に対する減災シミュレーション，自動車や飛行機など構造物に対する設計シミュレーション，創薬シミュレーション, 太陽や惑星などの宇宙シミュレーション, 自然の摂理を探求する基本的なシミュレーションなど, スーパーコンピュータによる大規模シミュレーションは我々の日常生活にも多大な恩恵をもたらしている. 一般に，高い精度の結果を得るためには，さらに規模の大きなシミュレーションが求められる. 今後, さらなる規模のシミュレーションのためには, 高性能なスーパーコンピュータは勿論，シミュレーションコードの高速化も必要となる.

例えば, スペクトル法による乱流の直接数値シミュレーション (Direct Numerical Simulation;DNS) は方程式に忠実で非常に精密なデータを得ることができるため, 乱流の基礎的研究に用いられているが, 高いレイノルズ数の乱流については計算量が膨大になり, コンピュータの計算速度やメモリ容量などの性能に計算可能な格子数が制限される. コンピュータの性能には限りがあるため, 計算量が膨大になるプログラムにおいてコードの改良及び高速化は必要不可欠である. 2016 年に石原らはスーパーコンピュータ「京」を用いて格子数 $12288^3$ の非圧縮性一様等方性乱流 DNS を行った [1]. この DNS の 1 スナップショットごとのデータ量は, $12288^3$ (格子数) ×3 (3 変数) ×8bytes (倍精度) で約 40TB になった. このような膨大なファイル入出力 (I/O) 処理には当然時間がかかるため, プログラムの高速化のためには, 計算自体の高速化のほか, ファイルの I/O 時間についても考慮する必要がある.

本稿では特徴的な I/O 処理方式のライブラリを持つ, ベクトル型スーパーコンピュータ SX-Aurora TSUBASA システムを利用して, I/O 処理ごとの性能評価を行った. シミュレーションに用いられる言語として主要な Fortran と, C 言語の 2 つの言語における入出力関数に対して, 通常の I/O を用いた場合, ベクトルエンジン (Vector Engine;VE) とベクトルホスト (Vector Host;VH) 間の効率的なデータ転送により高速化した I/O(Accelerated I/O) 機能を用いた場合, 加えて VH offload 機能を用いた場合ついて, ファイルの I/O 時間を計測し比較した.

## 2. SX-Aurora TSUBASA について

本節では, SX-Aurora TSUBASA の概要と, 今回利用し

[1] 神戸大学工学部情報知能工学科
[2] 神戸大学大学院システム情報学研究科
[3] 東北大学サイバーサイエンスセンター
[4] 日本電気（株）
[5] 東北大学大学院情報科学研究科
[a)] 1695093t@stu.kobe-u.ac.jp

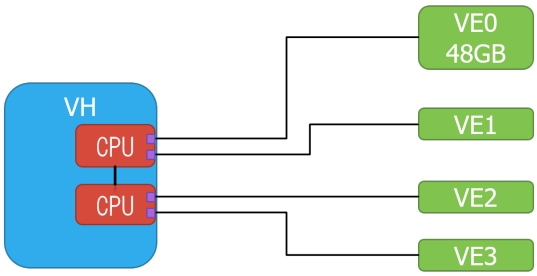


図 1　A300-4 の構成

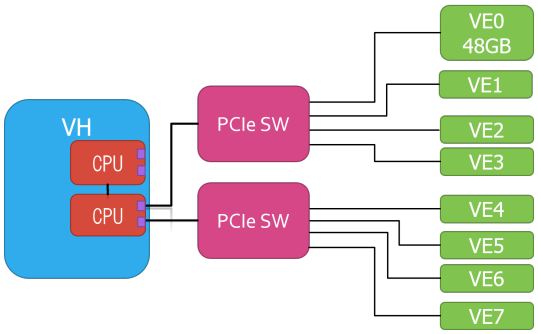


図 2　A300-8 の構成

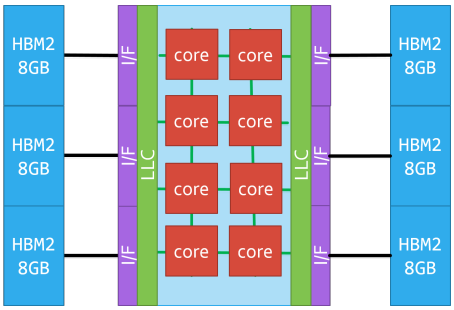


図 3　VE (Type 10B) の構成

たモデルのシステム構成について説明する.

### 2.1　概要

SX-Aurora TSUBASA は NEC のベクトル型スーパーコンピュータ SX シリーズの最新モデルである [2]. 従来の SX シリーズとは異なり, 主にアプリケーション演算処理を行う VE 部と, プロセスのスケジューリングや VE 上で実行されているアプリケーションから呼び出されたシステムコールの処理などを行う VH 部によって構成されている. 各 VE は, ベクトルプロセッサ, 及び高速メモリから構成された Gen 3×16 接続の PCI Express(PCIe) カードとして実装され, オペレーティングシステム (OS) 機能を実行する標準の x86/Linux ノードである VH と PCIe 経由で接続される. VE に OS カーネルはなく, VH 上で動作する VEOS によって VE は制御される. 本研究では, 東北大学に設置されているオンサイトモデルの A300-4 と A300-8 を用いた.

### 2.2　A300-4, 及び A300-8 のシステム構成

図 1, 及び図 2 に A300-4 と A300-8 のシステム構成を示す. A300-4 はプロセッサ (Intel Xeon Gold 6126) を 2 つ持つ VH が 1 個, VE が 4 個で構成されており, 4 個の VE がプロセッサに 2 個ずつ PCIe で接続されている. A300-8 はプロセッサを 2 つ持つ VH が 1 個, VE が 8 個で構成されており, 2 つある PCIe スイッチ (PCIe SW) に VE が 4 個ずつ接続されている. 2 つの PCIe SW はひとつのプロセッサに接続されている. また, A300-8 は 8 個の VE を持つため計算性能は A300-4 よりも高いが, VH 側のプロセッサに繋がっている PCIe 接続は 2 箇所であるため, PCIe 接続が 4 箇所の A300-4 よりも VH-VE 間のデータ転送性能は低いと考えられる.

図 3 に VE (Type 10B) の構成を示す. ベクトルコアが 8 つ, 16MB の last-level cache(LLC) は 8 コアで共有されている. また, メモリは 8GB の High Bandwidth Memory(HBM2)6 個で構成されており, VE 全体のメモリ容量は 48GB になる. SX-Aurora TSUBASA のメモリ帯域幅は, 現在世界最速の 1.22 (TB/s) である.

## 3.　I/O 方式

本節では, SX-Aurora TSUBASA における I/O 方式の概要を説明する. 今回, 測定した I/O 方式は, Fortran の READ/WRITE 文による入出力, 及び C の fread/fwrite 関数による入出力に対し, 通常の I/O 処理と高速化した I/O(Accelerated I/O) 処理を用いた場合を用いた場合, VH offload 機能を用いた C の fread/fwrite 関数の場合の 3 の方式について検討した.

### 3.1　通常 I/O

Fortran ではファイルの Open/Close を OPEN/CLOSE 文, I/O 処理を READ/WRITE 文, C ではファイルの Open/Close を fopen/fclose 関数, I/O 処理を fread/fwrite 関数を用いて行った. 通常の I/O 処理では, glibc の関数が呼び出され, 指定されたサイズのバッファが VE のメモリ上に確保される. VEOS が VE 側のプロセスに対応する VH 上のプロセスとのデータのやり取りを行うことで, データの入出力を行う.

### 3.2　Accelerated I/O(高速 I/O)

SX-Aurora TSUBASA のライブラリには, VE の Direct Memory Accesses(DMA) エンジンによって VH のバッファに直接データ転送することにより, I/O 処理のスループットやレイテンシを改善する Accelerated I/O(高速 I/O) という機能がある.

通常の I/O 処理では, VE で実行しているアプリケーショ

ンでシステムコールが発生すると, VE は停止し, そのアプリケーションに対応する VH 上の擬似プロセスがシステムコールを処理する. VEOS によって一次的にバッファが VH 上に確保され, DMA エンジンが VE 上のデータを VH のバッファに転送し, I/O 処理を行う. I/O 処理に対するシステムコールでは, この処理のためのレイテンシが発生する. I/O 処理の終了後バッファは解放される. このバッファの pin down/release(確保/解放) が I/O 処理の度に実行されるため, 複数回実行されると処理時間が大きくなる. 一方, 高速 I/O では, 最初の I/O 処理の際に, VH 上に DMA エンジンがアクセスできる I/O 専用のバッファをセットし, それ以降の I/O 処理でもこのバッファを利用する. このため, I/O 処理ごとのバッファの確保/解放が不要になり, I/O 処理が複数回実行される場合には, 高速化が期待できる.

ユーザが, この高速 I/O のライブラリを利用するためには, 環境変数 `VE_LD_PRELOAD` に `libveaccio.so.1` を設定すれば良い. この環境変数の設定により, 通常 I/O と同じソースコードを用いて高速 I/O の使用が可能である.

### 3.3 VH offload

SX-Aurora TSUBASA のライブラリには, システムコールとして VE プログラムから VH 上に置かれたユーザの共有ライブラリの関数を呼び出すための VH offload 機能がある. この機能を用いて C の fread/fwrite 関数を VH にオフロードして I/O 処理を行う場合についても測定した.

## 4. 測定用プログラム

I/O 性能の計測に使用したコードの概要を説明する. コードは MPI により並列化されており, 読み書きされるファイルはバイナリファイルである. 全ての MPI プロセスで読み書きするファイルサイズは同じ大きさとした. また, 全ての場合においてメインプログラムは Fortran で実装した. これは, 現在でも多くのシミュレーションコードが Fortran で書かれており, それらのコードをそのまま使用することを考慮したものである. 今回は, 純粋な I/O 性能のみを測定するため, 書き出す配列変数 (以下 var とする) は倍精度実数型とし, その値は全て 1.0 とした. ファイルサイズ, バッファサイズ, プロセス数は実行時に指定し, 実行時間は入出力の前後にバリア同期関数によって同期を取った後, MPI_Wtime() を用いて測定した.

### 4.1 main (Fortran, C)

図 4 に, Fortran で書かれたメインプログラムの概要を示す. file_write(myrank, var) によって var の中身をファイルに書き出した後, その書き出したファイルの中身を file_read(myrank, var) によって読み取った. 入出力されるファイルは, ランク番号によって独立したファイルとして認識される.

```
program
  use mpi
  real(kind=8) :: var(N)
 (... initialization ...)
  call file_write(myrank, var)
  call file_read(myrank, var)
 (... finalization ...)
end program
```

**図 4** メインプログラム

```
subroutine file_read(myrank, var)
  open(ur, file="foo.myrank",format="unformatted")
  call MPI_Barrier( ··· )
  t1 = MPI_Wtime()
  read(ur) var
  call MPI_Barrier( ··· )
  t2 = MPI_Wtime()
end subroutine
```

**図 5** Fortran による file_read 概要

```
subroutine file_write(myrank, var)
  open(uw, file="foo.myrank",format="unformatted")
  call MPI_Barrier( ··· )
  t1 = MPI_Wtime()
  write(uw) var
  flush(uw)
  call MPI_Barrier( ··· )
  t2 = MPI_Wtime()
end subroutine
```

**図 6** Fortran による file_write 概要

### 4.2 Fortran による実装

図 5 及び図 6 に Fortran の READ/WRITE 文による入出力時間を測定するためのサブプログラム file_read, file_write の概要を示す. 図 5 では, open(...) によって開いたファイルを read(...) によって var に値を読み込む. 図 6 では, open(...) によって開いたファイルに write(...) で var の値を書き込む. バッファのデータを全て書き出す様に flush(...) 文を挿入した.

### 4.3 C による実装

図 7 及び図 8 に C の fread/fwrite を測定するためのプログラム file_read, file_write の概要を示す. 図 7 では, fopen(...) によって開いたファイルを fread(...) によって var に値を読み込む. 図 8 では, fopen(...) によって開いたファイルに fwrite(...) で var の値を書き込む. また, Fortran 同様に, fflush(...) によってバッファのデータを書き出した.

```
void file_read_(int *myrank, double *var){
  FILE *fd;
  fd=fopen("foo.myrank", "rb");
  MPI_Barrier( ··· );
  t1 = MPI_Wtime();
  fread(var, sizeof(double), NN, fd);
  MPI_Barrier( ··· );
  t2 = MPI_Wtime();
}
```

**図 7** C による file_read 概要

```
void file_write_(int *myrank, double *var){
  FILE *fd;
  fd=fopen("foo.myrank", "wb");
  MPI_Barrier( ··· );
  t1 = MPI_Wtime();
  fwrite(var, sizeof(double), NN, fd);
  fflush(fd);
  MPI_Barrier( ··· );
  t2 = MPI_Wtime();
}
```

**図 8** C による file_write 概要

### 4.4 VH offload 機能を用いた実装

図 9 に VH offload 機能を用いた C の fread/fwrite 関数による入出力性能を測定したメインプログラムの概要を示す. vhcall_init によって VH offload 機能を使用するための初期設定をした後, 通常 I/O 同様 file_write_ve(myrank, var) によって VH 上で実行される fwrite 関数によって var の中身をファイルに書き出した後, VH 上で fread 関数を実行する file_read_ve(myrank, var) によってファイルの中身を読み取った.

図 10 に VH offload 機能を使用するためのライブラリと関数のハンドラ, 図 11 に vhcall_init, vhcall_close の概要を示した. vh が VH offload 機能で VH 上で実行される関数が入っているライブラリのハンドラ, sym_FO と sym_FC がそれぞれファイルの Open と Close, sym_R と sym_W がそれぞれファイルの Read と Write 用の関数に対応するハンドラである. ハンドラの初期設定を vhcall_init にて行い, 終了処理を vhcall_close にて行う.

図 12 及び図 13 に VH offload 機能を用いた C の fread/fwrite 関数の入出力時間を測定するためのプログラム file_read_ve, file_write_ve の概要を示した. どちらも vhcall_invoke(...) にて, vhcall_init で設定した関数のハンドラを呼び出すことで, ファイルの Open/Close とそれぞれの I/O 処理を行う.

## 5. I/O 時間の計測方法

計測は, 東北大学の SX-Aurora TSUBASA A300-4 と A300-8 を用いて行った. VH 及び VE の仕様をそれぞれ表 1 及び表 2 に示す.

```
program
  use mpi
  real(kind=8) :: var(N)
 (... initialization ...)
  call vhcall_init
  call file_write_ve(myrank, var)
  call file_read_ve(myrank, var)
  call vhcall_close
 (... finalization ...)
end program
```

**図 9** VH offload 機能を用いるメインプログラムの概要

```
vhacll_handle vh;

int64_t sym_FO
int64_t sym_FC
int64_t sym_R
int64_t sym_W
```

**図 10** VH offload 用
ライブラリと関数のハンドラ

```
int vhcall_init_(int *nsize){
  vh = vhcall_install( getenv("LIBVHCALL") );
  sym_FO = vhcall_find(vh, "file_open_VH");
  sym_FC = vhcall_find(vh, "file_close_VH");
  sym_R = vhcall_find(vh, "file_read_VH");
  sym_W = vhcall_find(vh, "file_write_VH");
}

int vhcall_close_(){
  vh = vhcall_uninstall(vh);
  return;
}
```

**図 11** 図 9 の vhcall_init, vhcall_close 概要

**表 1** VH の仕様

| VH | |
|---|---|
| CPU | Intel Xeon Gold 6126 |
| CPUs | 2 |
| Cores / CPU | 12 |
| Frequency | 2.60 GHz/3.70 GHz(Turbo) |
| Peak FP /core | 83.2 GFLOPS |
| Peak DP FLOPS | 998.4 GFLOPS |
| Memory Capacity | 96 GB |
| Memory Bandwidth | 128 GB/s |
| Memory subsystem | DDR4-2666 DIMM 16GB x6 |

使用したコンパイラは nfort 2.4.1 と ncc 2.4.1 である. コンパイルコマンドは mpinfort と mpincc, 最適化オプションに-O3 を使用した. プログラム実行コマンドは mpirun, 実行時のオプションとして`-ve <使用ノード> -np <プロセス数>` により MPI による並列数を変化させた.

入出力に用いられるバッファサイズは 8KB と 512KB の 2 通りとし, ファイルサイズは 1MB, 8MB, 32MB, 128MB,

```
vhcall_invoke( sym_FO, fptr, fsize, NULL, 0 );
MPI_Barrier();
t1 = MPI_Wtime();
vhcall_invoke( sym_R, NULL, 0, vptr, vsize );
var = vptr;
MPI_Barrier();
t2 = MPI_Wtime();
vhcall_invoke( sym_FC, fptr, fsize, NULL, 0 );
```

**図 12** 図 9 の file_read 概要

```
vhcall_invoke( sym_FO, fptr, fsize, NULL, 1 );
MPI_Barrier();
t1 = MPI_Wtime();
tptr = vptr;
memcpy( tptr, n, sizeof(int) );   tptr = tptr + sizeof(int);
memcpy( tptr, var, sizeof(double)*(*usize) );
vhcall_invoke( sym_W, vptr, vsize, NULL, 0 );
MPI_Barrier();
t2 = MPI_Wtime();
vhcall_invoke( sym_FC, fptr, fsize, NULL, 0 );
```

**図 13** 図 9 の file_write 概要

**表 2** VE の仕様

| VE | Type10B |
|---|---|
| Cores | 8 |
| Frequency | 1.4 GHz |
| Peak FP /core | 268.8 GFLOPS |
| Peak DP FLOPS / socket | 2.15 TFLOPS |
| Memory Capacity | 48 GB |
| Memory Bandwidth | 1.2 TB/s |
| Memory subsystem | HBM2 x6 |
| LLC Capacity | 16 MB shared |
| LLC Bandwidth | 2.66 TB/s |

512MB, 1GB の 6 通りに変化させて測定した．また，読み書きするファイルは全ての場合において SSD 上に置かれたものである．MPI プロセス数は，A300-4 で 1, 2, 4, 8, 16, 24, 32 プロセス，A300-8 で 1, 2, 4, 8, 16, 32, 64 プロセスとし，使用ノードは 8MPI プロセス以下で VE0, 16MPI プロセスで VE0 と VE1, 24MPI プロセスで VE0〜VE2, 32MPI プロセスで VE0〜VE3, 64MPI プロセスで VE0〜VE7 を使用した．全ての場合において，全ての MPI プロセスで Read と Write 両方において入出力時間を 10 回計測し，MPI プロセス数 ×10 回の平均を取った．

## 6. Fortran, C, VH offload の I/O 性能

本節では，まず Fortran と C による通常 I/O と，VH offload 機能を用いて VH 上で I/O 処理した場合の性能比較を行う．各場合についてのファイルサイズとバッファ，並列数による性能を比較し，システム構成の観点からその性能を考察した．さらに，プログラム言語の違いによる性能と，VH offload の機能の性能を比較した．

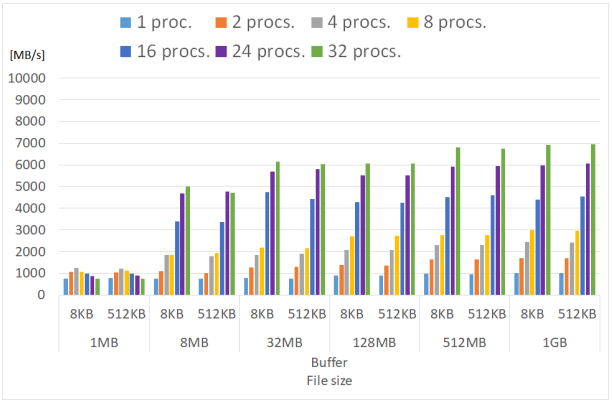


**図 14** Fortran の READ 性能 (A300-4)

### 6.1 A300-4 における性能

#### 6.1.1 Read 性能

図 14 に Fortran の READ 文，図 15 に C の fread 関数，図 16 に VH offload 機能を用いて VH 上で実行した C の fread 関数の性能をそれぞれ示す．各ケースについて左から，ファイルサイズ 1MB, 8MB, 32MB, 128MB, 512MB, 1GB の順で図示されている．全ての場合において，ファイルサイズが 32MB 以上の際に性能が良くなっている．また，ファイルサイズが大きい場合，MPI プロセス数の増加とともに性能が良くなっており，使用するコアが増えると性能が良くなる．しかしながら，コア数と性能は比例していない．8MPI プロセス以上では 8 の倍数プロセスごとに，すなわち使用する VE 数が増えるとともに性能の向上が大きくなっているのは，使用する VH 側の PCIe 接続が増えていることが要因だと考えられる．一方，ファイルサイズが小さい場合はスケーラビリティは良くない．これは，システムコールのレイテンシがボトルネックとなっているためと思われる．

バッファのサイズの影響は少ない．特に Fortran の READ についてはバッファサイズの影響が C の fread よりも小さい傾向がある．

図 17 はバッファサイズが 512KB，ファイルサイズが 1GB でのそれぞれの方式を比較したグラフである．32MPI プロセスでは，VH offload の機能を使った場合に 9.1GB/s の最大性能が得られた．少々 Fortran の性能が劣るが，これは Fortran の READ/WRITE 文が C 言語のライブラリで実装されているためである．C の fread 関数については，VH offload を使用した方が少し性能が良い．

#### 6.1.2 Write 性能

図 18 に Fortran の WRITE 文，図 19 に C の fwrite 関数，図 20 に VH offload 機能を用いて VH 上で実行した C の fwrite 関数の性能をそれぞれ示す．各ケースについて左からファイルサイズ 1MB, 8MB, 32MB, 128MB, 512MB, 1GB の順で図示されている．Read 性能と同じく，ファイルサイズが 32MB 以上の際の性能が良い．加えてファイルサイズ

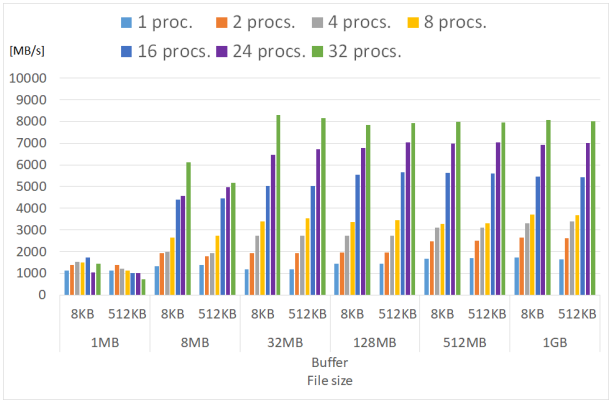


図 15　C 言語の fread 性能 (A300-4)

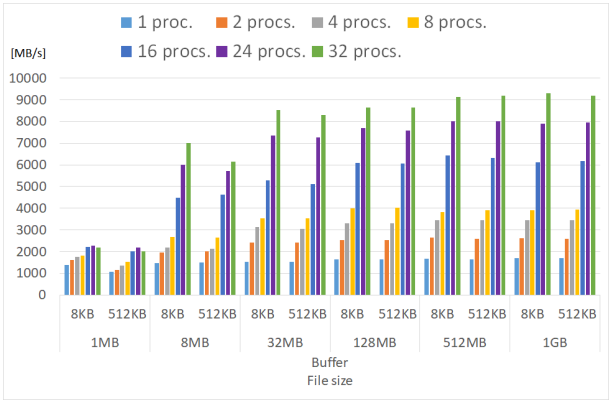


図 16　VH offload を用いた C 言語の fread 性能 (A300-4)

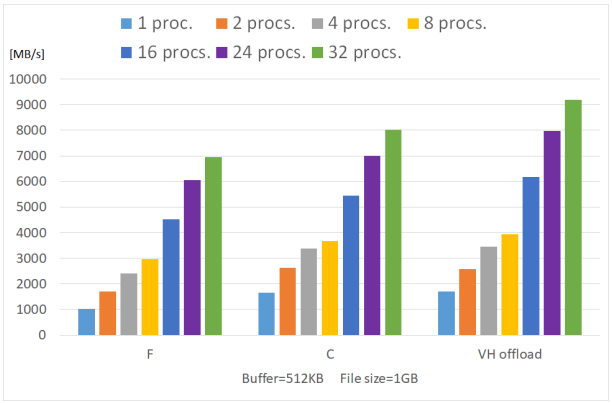


図 17　Fortran, C, VH offload の Read 性能比較 (A300-4)

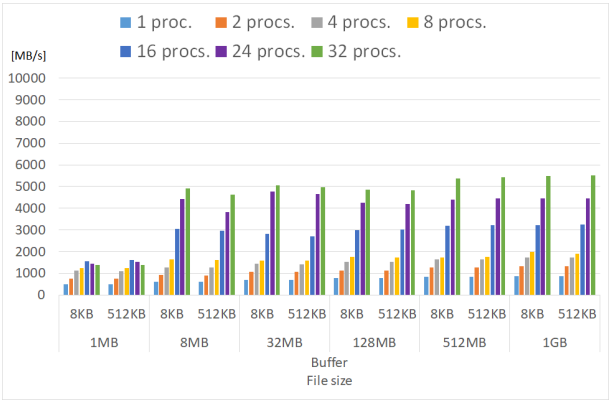


図 18　Fortran の WRITE 性能 (A300-4)

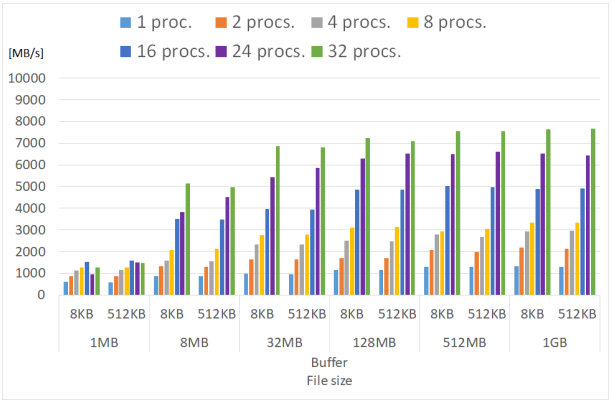


図 19　C 言語の fwrite 性能 (A300-4)

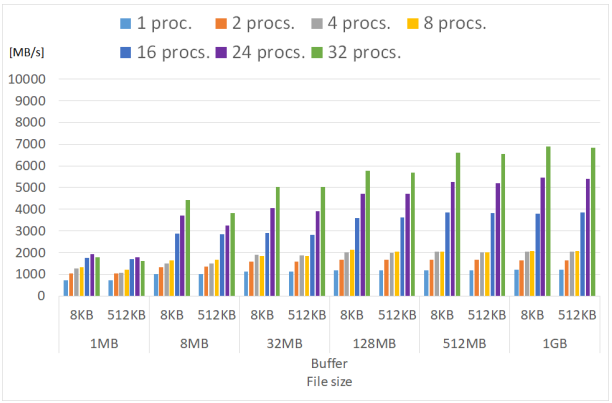


図 20　VH offload を用いた C 言語の fwrite 性能 (A300-4)

が十分に大きい場合は MPI プロセス数によって性能が良くなっているが, スケーラブルではない. また, Read 同様に使用する PCIe 接続が増えると性能が大きく向上することが分かる. バッファのサイズについても同様で, Fortran の場合 C よりも影響が小さいが, 全体的に影響は小さい結果となった. しかしながら, Write 性能は, Fortran の WRITE 文と VH offload 機能で, 4MPI プロセスと 8MPI プロセスでの性能がほぼ同じになっている. 8MPI プロセスまでは同じ 1 つの VE を使用しているため, VE 上のコアと共有メモリとの通信がボトルネックとなっている可能性が高いと考えられるが, その要因についてはさらに検討が必要である.

図 21 はバッファサイズが 512KB, ファイルサイズが 1GB でのそれぞれの方式を比較したグラフである. C による実装では 32MPI プロセスで, 約 8GB/s 性能が得られた. Write については, VH offload 機能よりも通常 I/O の方がわずかながら性能が良い.

### 6.2　A300-8 における性能

#### 6.2.1　Read 性能

図 22 に Fortran の READ 文, 図 23 に C の fread 関数, 図 24 に VH offload 機能を用いて VH 上で実行した C の

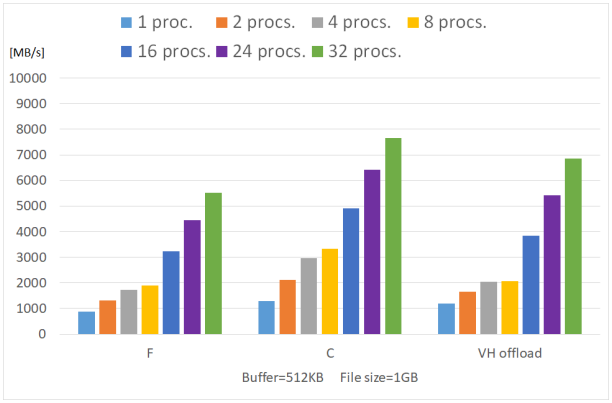


図 21 Fortran, C, VH offload の Write 性能比較 (A300-4)


図 22 Fortran の READ 性能 (A300-8)

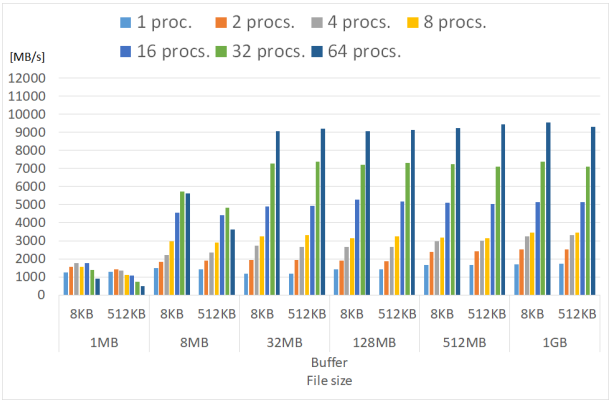


図 23 C 言語の fread 性能 (A300-8)

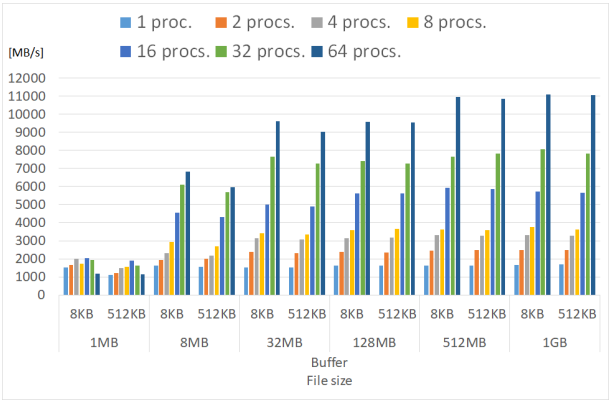


図 24 VH offload を用いた C 言語の fread 性能 (A300-8)

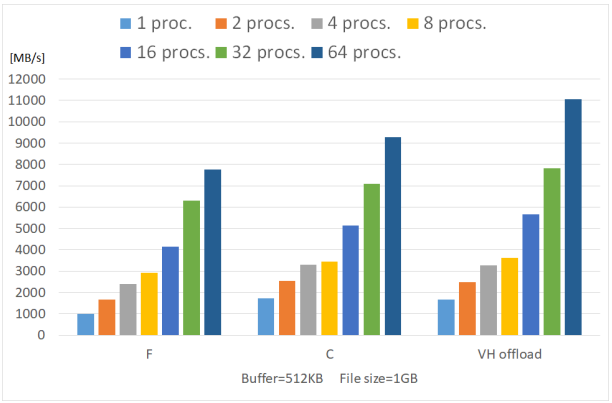


図 25 Fortran, C, VH offload の Read 性能比較 (A300-8)

fread 関数の性能をそれぞれ示す. 各ケースについて左からファイルサイズ 1MB, 8MB, 32MB, 128MB, 512MB, 1GB の順で図示されている. A300-4 の場合同様に, ファイルサイズが大きいほど性能が良い. ファイルサイズが十分に大きい場合, MPI プロセス数によって性能が良くなっているが, こちらも A300-4 同様に, コア数の増加に比例していない. ここで注目したいのは, A300-8 については, 32MPI プロセスと 64MPI プロセスでの性能向上が大きい. これは, A300-8 では 32MPI プロセスと 64MPI プロセスで使用する PCIe SW の数が増えるためと考えられる. また, バッファのサイズについては A300-4 同様に影響は少ない.

図 25 はバッファサイズが 512KB, ファイルサイズが 1GB でのそれぞれの方式を比較したグラフである. A300-8 では, VH offload 機能による実装で, 64MPI プロセスにて約 11GB/s の性能が得られた. A300-8 についても Fortran が少し劣り, VH offload 機能を用いた場合の性能が良いという結果になった. また, 同 MPI プロセス数では A300-4 の方がわずかに性能が良い. これは, 同 MPI プロセス数において使用する PCIe 接続が A300-4 の方が多いためだと考えられるが, VH-VE 間の通信性能などのその他の原因も考えられるため, 今後実験を重ねて検討していく.

### 6.2.2 Write 性能

図 26 に Fortran の WRITE 文, 図 27 に C の fwrite 関数, 図 28 に VH offload 機能を用いて VH 上で実行した C の fwrite 関数の性能をそれぞれ示す. 各ケースについて左からファイルサイズ 1MB, 8MB, 32MB, 128MB, 512MB, 1GB の順で図示されている. A300-4 同様, Read 性能の時と同じくファイルサイズが大きいほど性能が良く, ファイルサイズが十分に大きい場合はコア数によって性能が向上しているものの, 比例していない. Read 同様, 32MPI プロセスと 64MPI プロセスで性能向上が大きいのは, 使用する PCIe SW の数に起因していると考えられる. また, バッファのサイズについても同様, 影響は小さい. A300-4 と同様に, Fortran の WRITE 文と VH offload 機能で, 4MPI プ

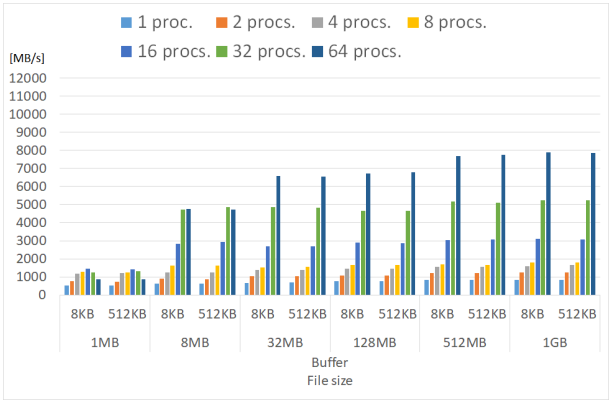


図 26　Fortran の WRITE 性能 (A300-8)

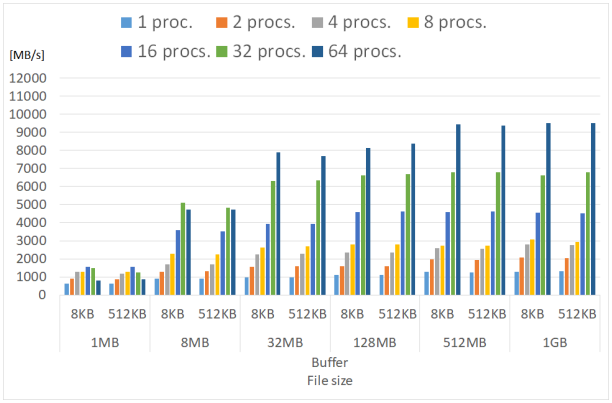


図 27　C 言語の fwrite 性能 (A300-8)

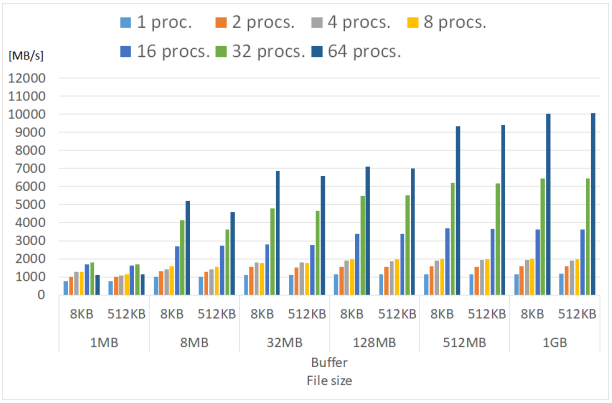


図 28　VH offload を用いた C 言語の fwrite 性能 (A300-8)

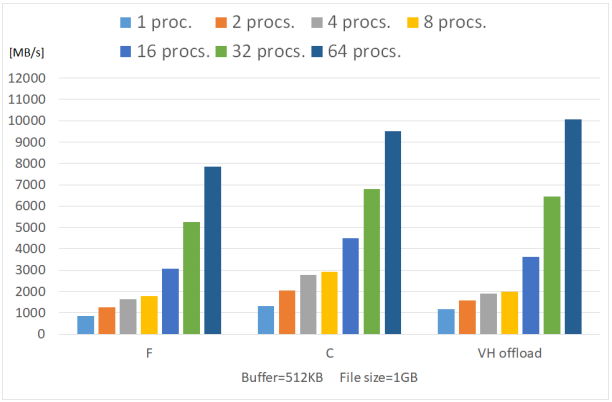


図 29　Fortran, C, VH offload の Write 性能比較 (A300-8)

ロセスと 8MPI プロセスでの性能がほぼ同じになっている. A300-8 においても, 8MPI プロセスまでは同じ 1 つの VE を使用しているため, VE 上のコアと共有メモリの通信がボトルネックとなっていると考えられるが, その要因についてはさらに検討が必要である.

図 29 はバッファサイズが 512KB, ファイルサイズが 1GB でのそれぞれの方式を比較したグラフである. Write 性能については VH offload 機能を用いた実装で, 64MPI プロセスにて約 10GB/s の最大性能が得られた. Write 性能においては, VH offload 機能を使用した際と通常 I/O の Write 性能の差はほぼなく, Fortran が少々劣る. また Read 同様, A300-4 の方が同 MPI プロセス数の場合はわずかに性能が良い. こちらも同様に, 同 MPI プロセス数において使用する PCIe 接続が A300-4 の方が多いためだと考えられるが, A300-8 の VH-VE 間の通信などのその他の要因がボトルネックになっている可能性も考えられる.

## 7.　高速 I/O との比較

本節では, C と Fortran について, 通常 I/O と高速 I/O の性能比較を行う. また, 並列数による性能から, データ転送のボトルネックとなる部分の考察も行う.

### 7.1　A300-4 における性能

#### 7.1.1　Read 性能

図 30 に A300-4 におけるバッファサイズが 512KB, ファイルサイズが 1GB での I/O 方式による Read 性能を示す. C 言語, Fortran ともに高速 I/O において, I/O 専用バッファを一度確保することでバッファの確保/解放のコストを減らすことができ, これによって大幅な性能向上が見られる. また, 高速 I/O についても, 使用する PCIe 接続が増えるほど性能向上が見られる. 8MPI プロセスでは, ひとつの PCIe 接続で期待できる最大性能である約 10GB/s が得られており, 最大性能を引き出すには 1VE 辺りの MPI プロセス数を増やす必要がある. 一方, 16MPI プロセスから 32MPI プロセスにおいてはスケーラブルに性能向上していない. 32MPI プロセスの高速 I/O で, Fortran と C の場合の両方で, 性能は約 32GB/s になっている. この PCIe 接続のデータ転送性能は 1 つにつき約 10GB/s であり, 4 つの各 VE が PCIe によるプロセッサの PCIe ポートと独立に繋がっているため, 全 VE 使用時の 32MPI プロセスでは 40GB/s 程度の性能が出ても良いはずである. 使用する VE が 1 つ (8MPI プロセス) の場合では, 期待される最大性能を得られており, VE 数が増えるにつれてスケーラビリティが落ちていることを考えると, 1 プロセッサごとの負荷が増えていることが原因として考えられる. 特に 3VE

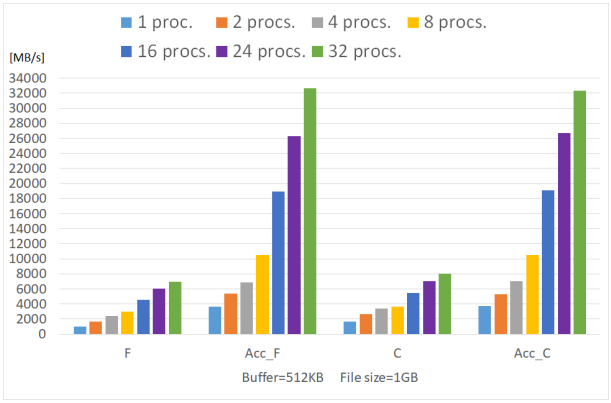


図 30　Read 性能の比較 (A300-4)

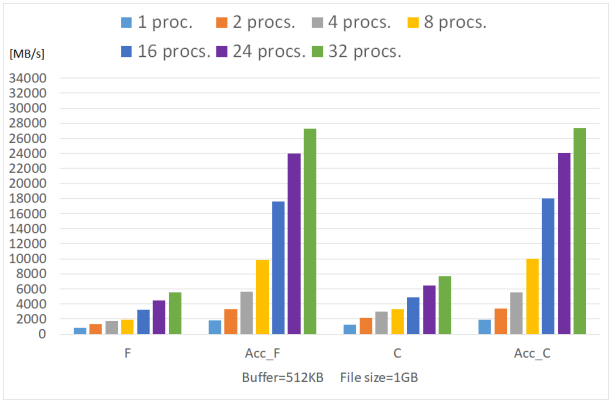


図 31　Write 性能の比較 (A300-4)

以上の場合では, VH 上の 2 つのプロセッサ間の通信がボトルネックとなっている可能性が高い. しかしながら, SSD の性能による劣化も考えられるため, 今後の実験で VH に付随する I/O disk が遅い場合, すなわち HDD や NAS などに変えた場合についても測定をして, 検討していきたい.

#### 7.1.2　Write 性能

図 31 に A300-4 におけるバッファサイズが 512KB, ファイルサイズが 1GB での I/O 方式による Write 性能を示す. Read の場合同様に, Fortran, C ともに高速 I/O において非常に大きな性能向上が見られる. また, 高速 I/O についても, 使用する PCIe 接続が増えるほど性能向上が見られるが, 16MPI プロセスと 24MPI プロセス間と比較した際に, 24MPI プロセスと 32MPI プロセスの間の性能向上が少ない. また, Read 性能同様に 16MPI プロセスから 32MPI プロセスにおいてはスケーラブルに性能向上しておらず, 高速 I/O の 32MPI プロセスで約 27GB/s しか得られなかった. Read 同様, 8MPI プロセスの場合では, 期待される最大性能である約 10GB/s を得られており, VE 数が増えるにつれてスケーラビリティが落ちていることを考えると, 1 プロセッサごとの負荷が増えていることが原因として考えられるが, I/O disk を変更して測定するなど, さらなる検討が必要である.

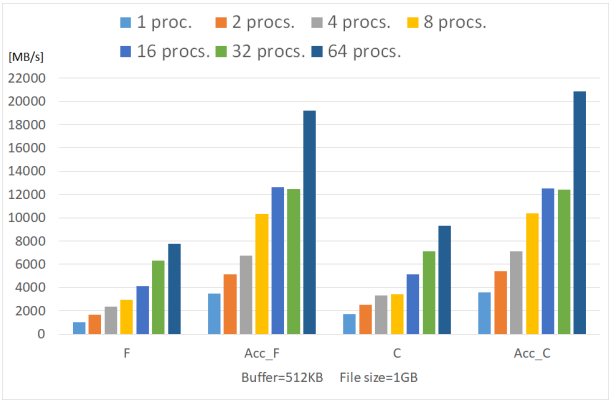


図 32　Read 性能の比較 (A300-8)

### 7.2　A300-8 における性能

#### 7.2.1　Read 性能

図 32 に A300-8 におけるバッファサイズが 512KB, ファイルサイズが 1GB での I/O 方式による Read 性能を示している. A300-8 についても高速 I/O による大きい性能向上が見られる. 64MPI プロセスで性能が大きく向上しているのは, VE4〜7 が接続されている PCIe SW のパスが使用され, 合計約 20GB/s の帯域を確保できたためと考えられる. A300-8 の Read 性能については, バッファサイズが 512KB, ファイルサイズが 1GB, 64MPI プロセスの場合, C の高速 I/O について約 21GB/s が得られた. また, 高速 I/O については, 16MPI プロセスと 32MPI プロセスでの性能が 12GB/s 程度で同じになっている. これらは PCIe SW が, VH のプロセッサの 1 つの PCIe ポートに接続されているため, 実効性能の約 10GB/s で抑えられているためである. A300-8 は使用できる PCIe 接続が最大 2 つのため, 使用する PCIe 接続が 1 つである 32MPI プロセスで 10GB/s 程度, 使用する PCIe 接続が 2 つである 64MPI プロセスで 20GB/s 程度の性能が最大になると考えられる. よって十分に PCIe 接続の性能が出ているといえる. 今後は, MPI プロセスの VE へのマッピングによって使用される PCIe SW が変わるので, 同じ数の MPI プロセスの VE へのマッピングを変えた測定も行っていきたい.

#### 7.2.2　Write 性能

図 33 に A300-8 におけるバッファサイズが 512KB, ファイルサイズが 1GB での I/O 方式による Write 性能を示す. Read 同様に, 高速 I/O による大きい性能向上が見られる. さらに, これまで見てきた場合と同様に, 使用する PCIe 接続が増える 64 プロセスでの性能向上が大きい. Write 性能については, バッファサイズが 512KB, ファイルサイズが 1GB, 64MPI プロセスの場合, C の高速 I/O について約 19GB/s が得られ, 16MPI プロセスと 32MPI プロセスでの性能が約 12GB/s で同じになっている. 使用する PCIe 接続が 1 つである 32MPI プロセスで 10GB/s 程度, 使用する PCIe 接続が 2 つである 64MPI プロセスで 20GB/s 程

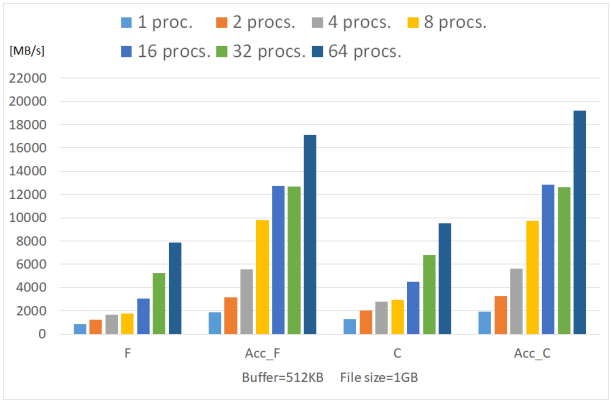


図 33　Write 性能の比較 (A300-8)

度の性能が最大になると考えられる．よって Read の際同様, Write についても PCIe 接続の性能が十分に出ているといえる．Read 同様, 使用する VE のマッピングを変えた測定も行っていきたい．

### 7.3　A300-4 と A300-8 との比較

A300-4 では, PCIe 接続が 4 つであるため, バッファサイズが 512KB, ファイルサイズが 1GB での 32MPI プロセスの高速 I/O の場合にて, Read 性能で 32GB/s 程度, Write 性能で 27GB/s 程度の性能を達成できた．一方 A300-8 では PCIe 接続は 2 つのため, バッファサイズが 512KB, ファイルサイズが 1GB での 64MPI プロセスの高速 I/O の場合, Read 性能で約 21GB/s, Write 性能で約 19GB/s となり, どちらも期待できる最大性能の, 約 20GB/s に上限が縛られていると考えられる．予測の通り, A300-4 の方が A300-8 よりもデータ転送の最大性能は良くなったものの, A300-4 については, 期待した性能の上限値までは得られておらず, その要因についてはさらに実験をする予定である．

## 8.　まとめ

本研究では SX-Aurora TSUBASA の I/O 処理について, 計測による性能評価を行った．ファイルサイズが十分に大きい場合に性能が良くなる傾向があるといえる．また, バッファのサイズによる影響は少ない．さらに, A300-4, A300-8 のどちらの場合も, 高速 I/O を使うのが有効であり, 高速 I/O は通常 I/O と比較して非常に良い性能が見られる効率的な I/O 方式であるといえる．シミュレーションで主に利用される Fortran を使用したとしても, 高速 I/O の機能によって大幅に性能が向上することが分かった．以上から, 既存の Fortran アプリケーションは高速 I/O を使うようにすれば, 入出力時間が短縮される．

加えて A300-4 は, PCIe 接続が多い分, 最大性能は A300-8 よりも良い．しかしながら, 今回の測定では期待される性能が十分に出ていないため, VH 上のプロセッサ間の通信性能や I/O disk の性能についても測定を重ね, 原因を追究したい．一方 A300-8 では, VE-VE 間の通信や VE-InfiniBand 間の通信を重視した構成になっているため, PCIe 接続の多い A300-4 ほど VH-VE 間のデータ転送性能は必要とならないという背景がある．こちらにおいては, 今回の測定において期待できる最大性能が得られ, PCIe 接続は十分に活用出来ていると分かった．

今回の評価の目的は, 大規模シミュレーションのコードに組み込む時にどのような I/O 処理の方法をとるのが良いかを見ることにあった．実際のアプリケーションにおいては, 並列処理を行うとチェックポイント時のひとつのファイルサイズは並列数によって小さくなるため, これらのシステム構成で十分であると考えられる．よって今後の課題としては, 具体的なアプリケーションを用いた評価もしていきたい．さらに, SX-Aurora TSUBASA には計算と非同期的に I/O 処理を行うことができる非同期 I/O(Asyncronous I/O) 機能のライブラリも有している．この非同期 I/O についての性能評価も進めていきたい．また, 今回の結果を踏まえ, データ転送性能でより良い A300-4 と最大並列数が倍の 64MPI プロセスまで実行できる A300-8 について, 実際のアプリケーションでは計算と I/O 処理の比率によってどちらの構成が有効であるかの評価もしていきたい．

**謝辞**　本研究の一部は，文部科学省「次世代領域研究開発」（高性能汎用計算機高度利用事業費補助金）量子アニーリングアシスト型次世代スーパーコンピューティング基盤の開発の一環として実施したものです．

### 参考文献

[1] Ishihara, T., Morishita, K., Yokokawa, M., Uno, A., Kaneda, Y.: *Energy spectrum in high-resolution direct numerical simulations of turbulence*, Phys. Rev. Fluids, 1, pp. 082403(2016).
[2] NEC：SX-Aurora TSUBASA シリーズ (online)，入手先 ⟨https://jpn.nec.com/hpc/sxauroratsubasa/index.html⟩ (2019.07.02).
[3] NEC：libsysve 2.1.1(online)，入手先 ⟨https://veos-sxarr-nec.github.io/libsysve/index.html⟩ (2019.07.31).
[4] Komatsu, K., Momose, S., Isobe, I., Watanabe, O., Musa, A., Yokokawa, M., Aoyama, T., Sato, M., Kobayashi, H.: *Performance evaluation of a vector supercomputer SX-aurora TSUBASA*, SC '18 Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis Article No. 54.
[5] Intel：Intel®Xeon®Gold 6126 Processor(online)，入手先 ⟨https://ark.intel.com/content/www/jp/ja/ark/products/120483/intel-xeon-gold-6126-processor-19-25m-cache-2-60-ghz.html⟩ (2019.07.02).
[6] NEC：SX-Aurora TSUBASA プログラム実行クイックガイド (online)，入手先 ⟨https://www.hpc.nec/documents/guide/pdfs/ProgramExecutionQuickGuide_J.pdf⟩ (2019.10.08).
[7] HPCwire Japan：スパコンにおける大規模シミュレーションのポスト処理環境の重要性 (online)，入手先 ⟨https://www.hpcwire.jp/archives/25761⟩ (2019.10.01).