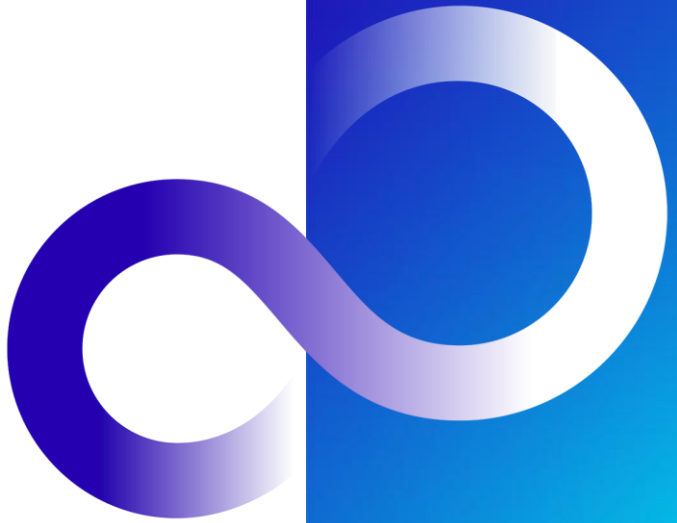
HPCアプリケーションのGPGPU化事例

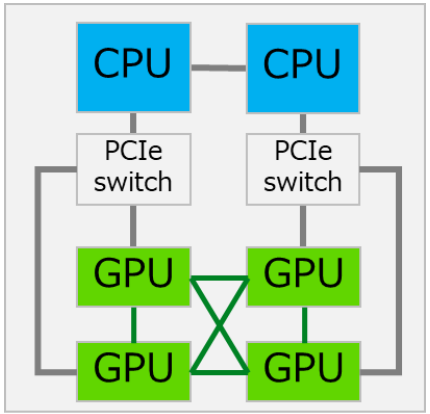
2025年8月18日
富士通株式会社
ミッションクリティカルシステム事業本部
テクニカルコンピューティング事業部

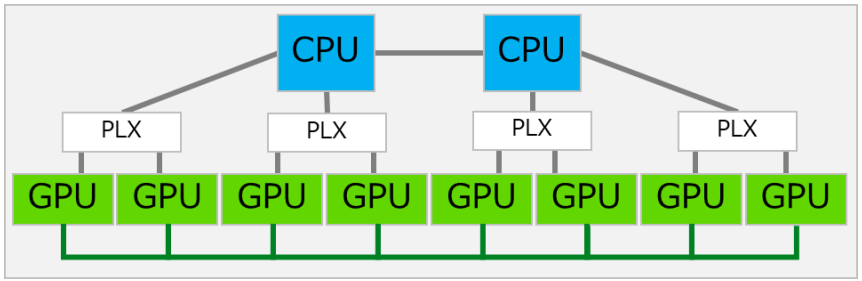
多湖 和馬

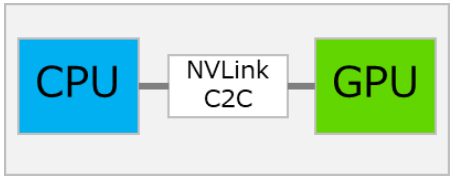


GPGPU化作業とは

- GPU搭載スーパーコンピュータのハードウェア構成
  - ノード当たり複数GPUを搭載する構成や1ノード1GPUのみ搭載する構成など様々なハードウェア構成が存在
- アプリケーションはそのままではGPU上で実行できないため、GPU上で実行できるようにする作業が必要
  - プログラミング言語(CUDA/CUDA Fortran)、 コンパイラ指示行(OpenACC)、ライブラリ(cuBLAS、cuSPARSE等)を利用することにより、GPU上で実行することが可能



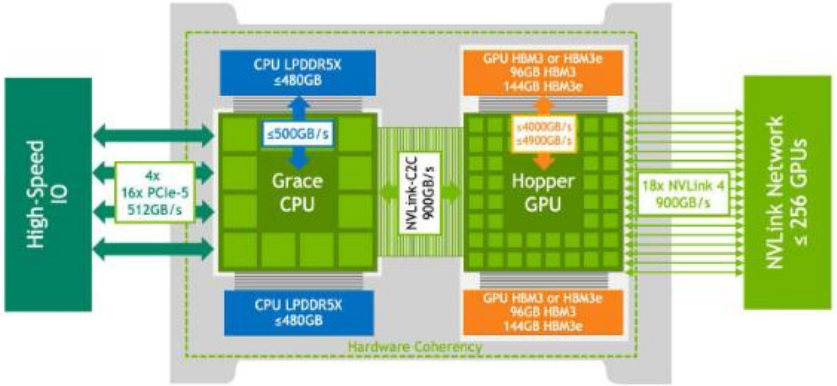




Miyabiスーパーコンピュータ

- GPUを搭載した演算加速ノード「Miyabi-G」と汎用CPUノード「Miyabi-C」で構成される
- Miyabi-Gは、NVIDIA GH200 Grace Hopper Superchipを搭載した国内初の汎用大規模システム
- 東京大学 情報基盤センターと筑波大学 計算科学研究センターが共同で運営、2025年1月より本稼働
- 本システムの稼働にあたり、従来のCPU中心のHPCユーザがGPUをスムーズに活用できるように、既存のHPCアプリケーションのGPGPU化が重要な課題となっていた

項目		Miyabi-G	Miyabi-C
総理論演算性能 (PFLOPS)		79.9	1.29
総ノード数		1,120	190
CPU	プロセッサ名	NCIDIA Grace CPU	Intel Xeon Max 9480
	理論演算性能 (TFLOPS)	3.456	6.8096
GPU	プロセッサ名	NVIDIA Hopper H100 GPU	
	理論演算性能 (TFLOPS)	66.9	



NVIDIA GH200 Grace Hopper Superchip

https://resources.nvidia.com/en-us-grace-cpu/grace-hopper-superchip

GPGPU化事例：ユーザアプリケーション

FUJITSU

- 富士通はアプリケーションのGPUポーティングサービスを展開
- Miyabi-G稼働に合わせて、ユーザアプリケーションのGPGPU化を実施

分類	アプリケーション名
非線形FEMソルバー	FrontISTR
電磁流体力学乱流ソルバー	MUTSU-T3/iHallMHD3D
気候シミュレーションソルバー	NICAM
次世代気象気候科学基盤ライブラリ	SCALE

GPGPU化事例：オープンソースソフトウェア

FUJITSU

- ユーザアプリケーションに加え、広く利用されているオープンソースソフトウェア(OSS)のGPGPU化を実施
  - Miyabi-G、Miyabi-Cで利用できるようにOSSを整備

項目	Miyabi-G	Miyabi-C
OS	Rocky Linux 9 (ログインノードはRed Hat Enterprise Linux 9)	
コンパイラ	GNU コンパイラ	
	NVIDIA HPC SDK (Fortran、C、C++、OpenMP、OpenACC) NVIDIA CUDA Toolkit (CUDA C、CUDA C++)	Intel コンパイラ(Fortran、C、C++)
メッセージ通信ライブラリ	Open MPI、NVIDIA HPC-X	Intel MPI
ライブラリ	cuBLAS、cuSPARSE、cuFFT、MAGMA、cuDNN、NCCL	-
	BLAS、CBLAS、LAPACK、ScaLAPACK、SuperLU、SuperLU MT、SuperLU DIST、METIS、MT-METIS、ParMETIS、Scotch、PT-Scotch、PETSc、Trillinos、FFTW、GNU Scientific Library、NetCDF、Parallel netCDF、HDF5、Parallel HDF5、OpenCV、Xabclib、ppOpen-HPC、MassiveThreads、Standard Template Library (STL)、Boost C++	
アプリケーション	OpenFOAM、ABINIT-MP、PHASE、FrontFlow/blue、FrontISTR、REVOCAP-Coupler、REVOCAP-Refiner、OpenMX、MODYLAS、GROMACS、BLAST、R packages、bioconductor、BioPerl、BioRuby、BWA、GATK、SAMtools、Quantum ESPRESSO、Xcrypt、ROOT、Geant4、LAMMPS、CP2K、NWChem、DeepVariant、Paraview、Vislt、POV-Ray、TensorFlow、PyTorch、JAX、Keras、Horovod、MXNet、Miniforge、Kokkos	
フリーソフトウェア	autoconf、automake、bash、bzip2、cvs、emacs、findutils、gawk、gdb、make、grep、gnuplot、gzip、less、m4、python、perl、ruby、screen、sed、subversion、tar、tcsh、tcl、vim、zsh、git、Julia、CMake、Ninja、Java JDK、Grid Community Toolkit、Gfarm、FUSE など	

流体解析で広く用いられているOpenFOAMのGPGPU化を報告

※FrontISTR、SCALEはユーザアプリケーションのGPGPU化事例で紹介するが、ソースコードを公開しておりOSSである

東京大学情報基盤センタースーパーコンピュータシステムの紹介：https://www.cc.u-tokyo.ac.jp/supercomputer/miyabi/system.php

FUJITSU

ユーザアプリケーションの GPGPU化実装事例

ユーザアプリケーションのGPU化方針

FUJITSU

- Fortran、C/C++コードのGPU化の選択肢
  - NVIDIA GPU向けにソースコードをGPGPU化する場合にはプログラミング言語であるCUDA/CUDA Fortranか
    コンパイラ指示行であるOpenACCが選択肢
- CUDA/CUDA Fortran
  - メリット：利用するGPUメモリ、スレッドの構成、スレッドの同期などを細かく制御できるため実装の自由度が高い
  - デメリット：GPUカーネル用の新規コードの開発が必要
- OpenACC
  - メリット：既存コードにコンパイラ指示行（ディレクティブ）を追加するだけでGPU並列実行が可能
    - 可読性が高く、 GPU用コード、CPU用コードと分ける必要がなく保守しやすい
    - CUDA/CUDA Fortraに比べて実装が容易
  - デメリット：CUDA/CUDA Fortranのような柔軟な実装・最適化は難しい
    - シェアードメモリのような高速メモリを自由に使うことが難しい

実装の容易さ、保守性の観点から
OpenACCを用いてGPGPU化を実施

FrontISTR

FUJITSU

- アプリケーション概要
  - 有限要素法構造解析プログラムとして、Windows や Linux の PC クラスタから富岳などの超並列スパコンまで、様々な環境で並列計算を実行可能
  - 静的・動的応力解析、固有値・周波数応答解析、熱伝導解析など様々な解析機能を提供
  - ソースコードは公開されておりオープンソースソフトウェアである
- アプリケーション構成
  - プログラム全体のFortranコードの規模は128Kstep程度
  - MPIとOpenMPでプロセス並列、スレッド並列実行可能
  - 本作業で3つの作業区分、合計43ルーチンに対してOpenACC実装を実施
  - FrontISTRのFortranコードにおける総サブルーチン数は1933個程度

作業区分	OpenACC実装ルーチン数
反復法ソルバーの拡充	5
SpMVルーチンの自由度拡張	24
前処理の拡充	14

https://www.frontistr.com/

FrontISTR 実装事例

- OpenMPディレクティブとOpenACCディレクティブが競合する処理は_OPENACCマクロでコンパイル時に切り替えるように実装
  - コンパイルオプション-accを利用するとOpenACCディレクティブが有効になる。このとき、_OPENACCマクロも同時に定義されため、ifdefで処理を切り替える方式とした。
    これにより、OpenMPとOpenACCを同時に有効にした場合にも対応可能な実装とした。

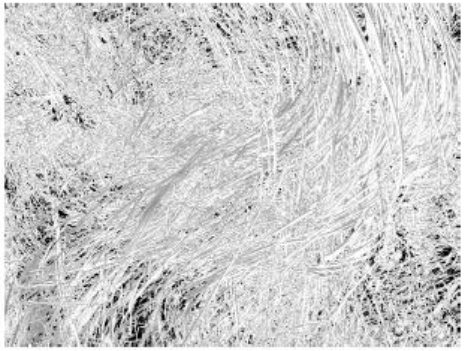
```
    !$omp parallel
default(none),private(ii,ALUtmp,k,i,j,PW),shared(N,ALU,SIGMA_DIAG)
    !$omp do
    do ii= 1, N
      ALUtmp(1,1)= ALU(16*ii-15) * SIGMA_DIAG
      ALUtmp(1,2)= ALU(16*ii-14)
<省略>
      do k= 1, 4
        ALUtmp(k,k)= 1.d0/ALUtmp(k,k)
        do i= k+1, 4
          ALUtmp(i,k)= ALUtmp(i,k) * ALUtmp(k,k)
          do j= k+1, 4
            PW(j)= ALUtmp(i,j) - ALUtmp(i,k)*ALUtmp(k,j)
          enddo
          do j= k+1, 4
            ALUtmp(i,j)= PW(j)
          enddo
        enddo
      enddo
<省略>
    enddo
    !$omp end do
    !$omp end parallel
```

```
#ifdef _OPENACC                                                          } OpenACC
    !$acc kernels                                                        } ディレクティブ
    !$acc loop independent private(ALUtmp,PW)                            }
#else
    !$omp parallel default(none),private(ii,ALUtmp,k,i,j,PW),shared(N,ALU,SIGMA_DIAG)  } OpenMP
    !$omp do                                                                          } ディレクティブ
#endif
    do ii= 1, N
      ALUtmp(1,1)= ALU(16*ii-15) * SIGMA_DIAG
      ALUtmp(1,2)= ALU(16*ii-14)
<省略>
      do k= 1, 4
        ALUtmp(k,k)= 1.d0/ALUtmp(k,k)
        do i= k+1, 4
          ALUtmp(i,k)= ALUtmp(i,k) * ALUtmp(k,k)
          do j= k+1, 4
            PW(j)= ALUtmp(i,j) - ALUtmp(i,k)*ALUtmp(k,j)
          enddo
          do j= k+1, 4
            ALUtmp(i,j)= PW(j)
          enddo
        enddo
      enddo
<省略>
    enddo
#ifdef _OPENACC                  } OpenACC
    !$acc end kernels            } ディレクティブ
#else
    !$omp end do                 } OpenMP
    !$omp end parallel           } ディレクティブ
#endif
```

MUTSU-T3/iHallMHD3D

- アプリケーション概要
  - 核融合科学研究所で開発が進められている電磁流体力学乱流の高精度・高並列LESシミュレーションコード
- アプリケーション構成
  - プログラム全体のFortranコードの規模は62Kstep程度
  - MPIとOpenMPでプロセス並列、スレッド並列化済み
  - 本作業で7つの機能、合計39ルーチンに対してOpenACC実装を実施
  - MUTSUのFortranコードにおける総サブルーチン数は585個程度

機能	OpenACC実装ルーチン数
相関関数(Correlation_function)	8
相関関数(ESS_structure_function)	8
相関関数(KarmanHowarth_correlation)	1
相関関数(Galtier_correlation)	3
相関関数(Politano_correlation)	3
確立密度関数(PDFs)	9
エネルギー伝達関数(Transfer)	6



「学際大規模情報基盤共同利用・共同研究拠点 2021 年度共同研究 最終報告書　jh210004-NAH 電磁流体力学乱流の高精度・高並列LES シミュレーションコード開発研究」より引用

MUTSU-T3/iHallMHD3D 実装事例

- OpenACCループのreductionに大きな配列を指定したとき実行時エラーが発生
  - 並列化されたループで配列strYに値を足し合わせる処理
  - reductionに指定した変数はGPUカーネル実行時にプライベートコピーが作成されるため、配列サイズが大きいとメモリ容量を超過する恐れがある
  - 配列をスカラーに変更してデータサイズを縮小、合わせて並列化階層を変更

```
!$acc kernels
!$acc loop reduction(+:strY)
do k=pstaY(3),pendY(3)
    do j=pstaY(2),pendY(2)
        do i=pstaY(1),pendY(1)
            do iy = 0, g_ny/2-1
                ic = mod(j+iy,g_ny)
                strY(iy) = strY(iy) + b_wk1(i,j,k)
* b_wk2(i,ic,k)
            end do
        end do
    end do
end do
!$acc end kernels
```

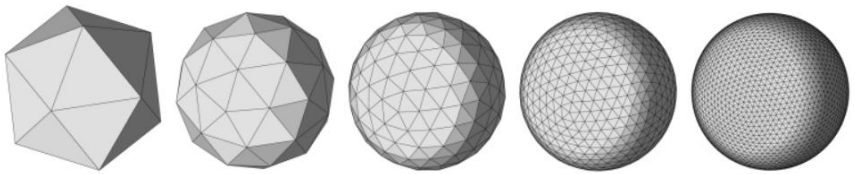
```
do iy = 0, g_ny/2-1
    strYtmp = 0.0d0
    !$acc kernels
    !$acc loop reduction(+:strYtmp) independent
    do k=pstaY(3),pendY(3)
        do j=pstaY(2),pendY(2)
            do i=pstaY(1),pendY(1)
                ic = mod(j+iy,g_ny)
                strYtmp = strYtmp + b_wk1(i,j,k) *
b_wk2(i,ic,k)
            end do
        end do
    end do
    !$acc end kernels
    strY(iy) = strYtmp
end do
```

NICAM

FUJITSU

- アプリケーション概要
  - Nonhydrostatic ICosahedral Atmospheric Model（非静力学正20面体格子大気モデル）の略称であり、全球雲解像モデルとして使用する気象解析コード
  - 地球全体で雲の発生や挙動を忠実に表現することにより、高精度のシミュレーションを実現
- アプリケーション構成
  - プログラム全体のFortranコードの規模は603Kstep程度
  - MPIとOpenMPでプロセス並列、スレッド並列実行可能
  - 本作業で6つの機能、合計54ルーチンに対してOpenACC実装を実施
  - NICAMのFortranコードにおける総サブルーチン数は4916個程度

機能	OpenACC実装ルーチン数
プロセス間データ通信	1
変数の数値範囲チェック	18
変数の緩和係数の設定	1
凝結物計算	7
積雲過程計算	14
サブグリッド乱流計算	13



https://nicam.jp/dokuwiki/doku.php?id=Top

NICAM 実装事例

- OpenACC区間からさらにサブルーチンを呼び出す箇所では、呼び出し先にacc routineディレクティブを追加
  - acc routine内でmodule変数を利用している箇所は、acc declareディレクティブを用いてデバイスメモリにデータを確保する実装とした

```
!$acc kernels &
!$acc pcopy(DELP,GDQS,FDQS,GAM,GDS,GDH,GDHS) &
!$acc pcopyin(GDRHO,GDZM,GDP,GDT,GDP,GDZ,GDQ)
do K = 1, KMAX
  do IJ = ISTS, IENS
    DELP ( IJ,K ) = GDRHO( IJ,K )*(GDZM( IJ,K+1 )-
    GDZM( IJ,K ) )*GRAV
    GDQS ( IJ,K ) = FQSAT( GDT( IJ,K ), GDP( IJ,K )
    FDQS ( IJ,K ) = FDQSAT( GDT( IJ,K ), GDQS( IJ,K
    GAM ( IJ,K ) = EL/CP*FDQS( IJ,K )
    GDS ( IJ,K ) = CP*GDT( IJ,K ) + GRAV*GDZ( IJ,K )
    GDH ( IJ,K ) = GDS( IJ,K ) + EL*GDQ( IJ,K,1 )
    GDHS ( IJ,K ) = GDS( IJ,K ) + EL*GDQS( IJ,K )
  enddo
enddo
!$acc end kernels
```

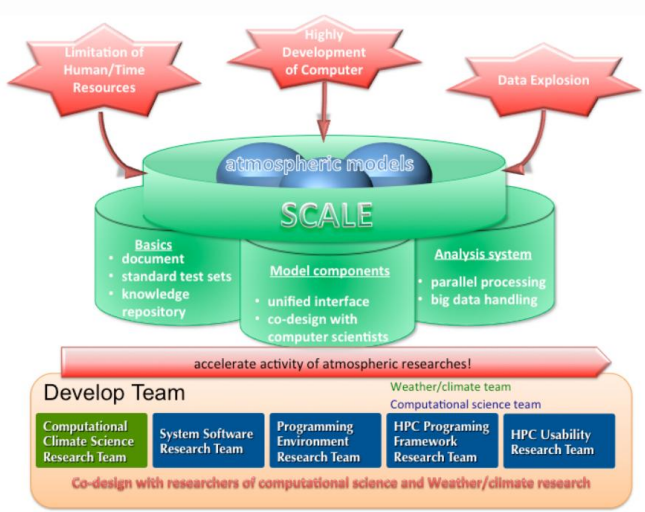
```
real(8) function FQSAT(T,P)
!$acc routine seq
use mod_cnst, only: &
RVAP => CNST_Rvap, &
LH0 => CNST_LHV0, &
PSAT0 => CNST_PSAT0, &
EPSV => CNST_EPSvap, &
EMELT => CNST_EMELT, &
TMELT => CNST_TMELT, &
TQICE => CNST_TMELT

real(8) :: T
real(8) :: P

FQSAT = EPSV * PSAT0 / P &
* exp( ( LH0 + EMELT * (0.5D0-sign(0.5D0,T-TQICE)) ) / RVAP * ( 1.D0/TMELT - 1.D0/T ) )
return
end function FQSAT
```

```
module mod_cnst
・・・
real(RP), public, parameter :: CNST_TMELT = 273.15_RP
!$acc declare copyin(CNST_TMELT)
・・・
```

SCALE

- アプリケーション概要
  - 地球・惑星における気象・気候モデルのための共通基盤ライブラリ
  - 次世代のスーパーコンピュータから汎用計算機にいたるまで広く用いられることを念頭に開発されており、気候・気象科学の専門家と計算機科学の専門家が共同して開発
  - ソースコードは公開されておりオープンソースソフトウェアである
- アプリケーション構成
  - プログラム全体のFortranコードの規模は344Kstep程度
  - MPIとOpenMPでプロセス並列、スレッド並列実行可能
  - 本作業では3つの機能、合計43ルーチンに対してOpenACC実装を実施
  - SCALEのFortranコードにおける総サブルーチン数は3108個程度

機能	OpenACC実装ルーチン数
時間積分法HEVE	1
雲微物理SN14	19
雲微物理SUZUKI10	23



https://scale.riken.jp/ja/

SCALE 実装事例 1

- ルーチンのOpenACC化
  - ルーチン内のループを対象にOpenACCのacc kernelsディレクティブを追加
  - サブルーチンの初めから終わりまでカバーするようにacc dataディレクティブを用いてデータ領域を定義し、ルーチン内で利用するデータはGPUメモリに常駐するように実装

```
subroutine ATMOS_DYN_Tstep_short_fvm_heve( …… )
!$acc data copy(buoy,cor,dpres,mflx_hi,&
!$acc pgf,qflx_hi,rhot_rk) &
!$acc copyin(cdz,corioli,ddiv,dens,dens_t,dens0,&
[中略]
!$acc rhot,rhot_t,rhot0,rt2p,tflx_anti,&
!$acc wdamp_coef) &
!$acc copyout(dens_rk,dens_uvw,dens0_uvw,pott,&
!$acc momx_rk,momy_rk,momz_rk,tflx_hi,velx,vely,velz)
[中略]
do JJS = JS, JE, JBLOCK
  JJE = JJS+JBLOCK-1
  do IIS = IS, IE, IBLOCK
    IIE = IIS+IBLOCK-1
    !$omp parallel private(i,j,k)
    !$omp do OMP_SCHEDULE_ collapse(2)
    !$acc kernels
    do j = JJS-1, JJE+1
      do i = max(IIS-1,1), min(IIE+1,IA)
        do k = KS, KE
          DPRES(k,i,j) = DPRES0(k,i,j) + RT2P(k,i,j) * ( RHOT(k,i,j) - REF_rhot(k,i,j) )
        enddo
[中略]
      enddo
    enddo
    !$acc end kernels
    !$omp end do nowait
    !$omp do OMP_SCHEDULE_ collapse(2)
    !$acc kernels
    do j = JJS-JHALO, JJE+JHALO
      do i = IIS-IHALO, IIE+IHALO
        do k = KS, KE
          POTT(k,i,j) = RHOT(k,i,j) / DENS(k,i,j)
        enddo
      enddo
    enddo
  !$acc end kernels
  !$omp end do nowait
  enddo
enddo
[中略]
!$acc end data
end subroutine ATMOS_DYN_Tstep_short_fvm_heve
```

サブルーチンの初めから終わりまで
dataディレクティブを指定して
GPUメモリにデータを常駐させる

各ループを対象にkernels
ディレクティブを追加

SCALE 実装事例 2

FUJITSU

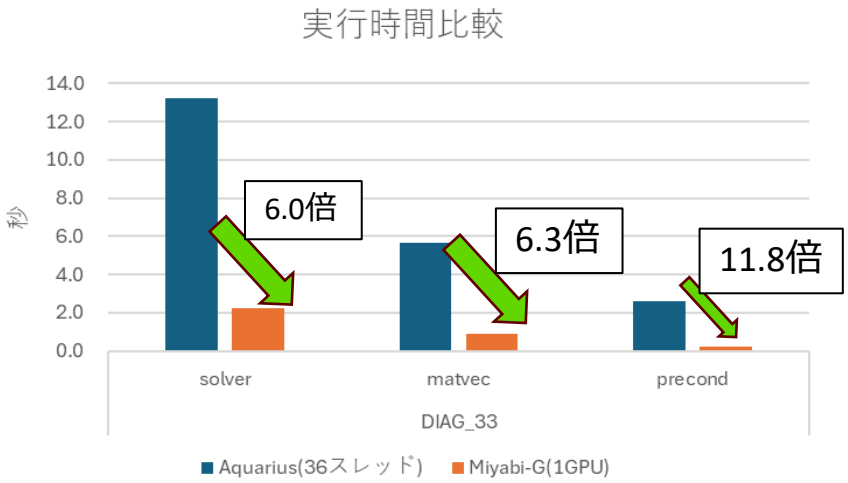
- Miyabi-GのNVIDIA HPC SDKでSCALEをコンパイルした際、モジュールサイズが2GBを超過し、コンパイルエラーが発生した
  - NVIDIA HPC SDKコンパイラにモジュールサイズ2GBの制限があり、その制限を超過
- GPUアーキテクチャごとに設定されるCompute Capabilityの値を適切に指定することでモジュールサイズの削減が可能
  - Compute Capability：GPUアーキテクチャごとにハードウェア機能とサポートされる命令を定義する
  - コンパイル時にシステムで検出されたGPUと一致するCompute Capabilityとなるが、検出されない場合にはサポートされているすべてのCompute Capabilityに対応する実行モジュールを作成する
  - コンパイルオプションの指定で実行モジュールのCompute Capabilityを指定可能
    - 例)H100： -gpu=cc90、A100：-gpu=cc80

ユーザアプリケーションの CPUとGPUの実行時間比較

FrontISTRのCPUとGPUの実行時間比較

- CPU(AquariusのIntel Xeon Platinum 8360Y) 実行と GPU(Miyabi-GのNVIDIA H100)実行の比較
  - Aquarius実行：1ノード1プロセス36スレッドで実行
  - Miyabi-G実行：1ノード1プロセス1GPUで実行
  - solver区間はmatvec区間、precond区間を含む

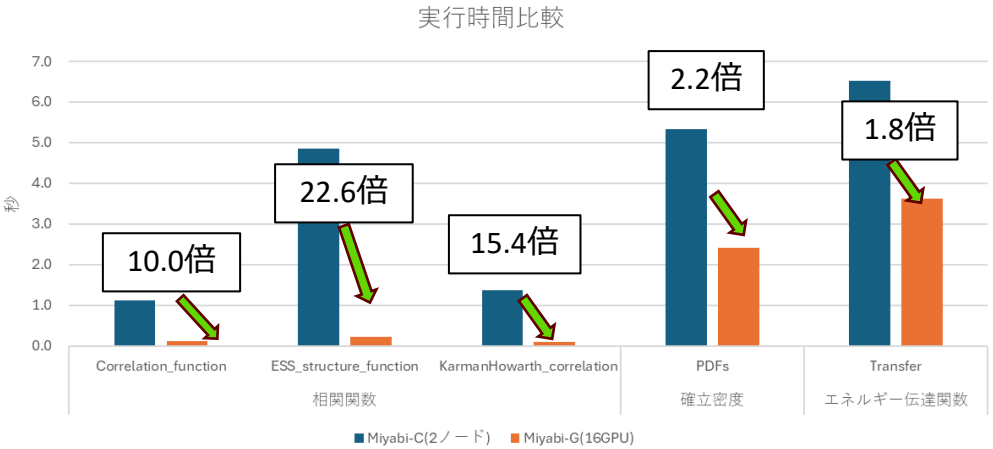
システム	プロセッサ名	使用リソース	使用リソースの倍精度演算性能
Aquarius(CPU)	Intel Xeon Platinum 8360Y	36コア	2.8TFLOPS
Miyabi-G(GPU)	NVIDIA Hopper H100	1GPU	66.9TFLOPS



MUTSU-T3/iHallMHD3DのCPUとGPUの実行時間比較

- CPU (Miyabi-CのIntel Xeon MAX 9480)実行とGPU(Miyabi-GのNVIDIA H100)実行の比較
  - Miyabi-C実行：2ノード16プロセス14スレッドで実行
  - Miyabi-G実行：16ノード16プロセス16GPUで実行
  - 確立密度(PDFs)とエネルギー伝達関数(Transfer)区間はスレッド間でデータ更新競合を避けるためにatomic処理を含んでいる。その影響で相関関数区間と高速化の差が出ていると考えられる。

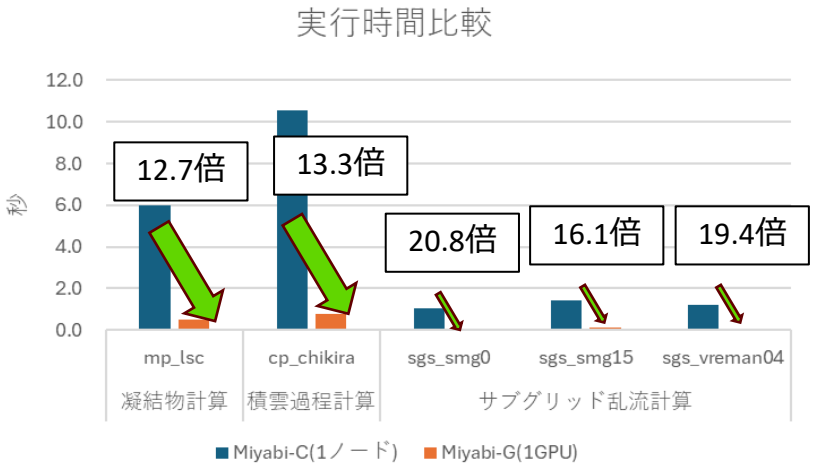
システム	プロセッサ名	使用リソース	使用リソースの倍精度演算性能
Miyabi-C(CPU)	Intel Xeon MAX 9480	2ノード224コア	13.6TFLOPS
Miyabi-G(GPU)	NVIDIA Hopper H100	16GPU	1070.4TFLOPS



NICAMのCPUとGPUの実行時間比較

- CPU(Miyabi-CのIntel Xeon MAX 9480)実行とGPU(Miyabi-GのNVIDIA H100)実行の比較
  - Miyabi-C実行：1ノード1プロセス56スレッドで実行
  - Miyabi-G実行：1ノード1プロセス1GPUで実行

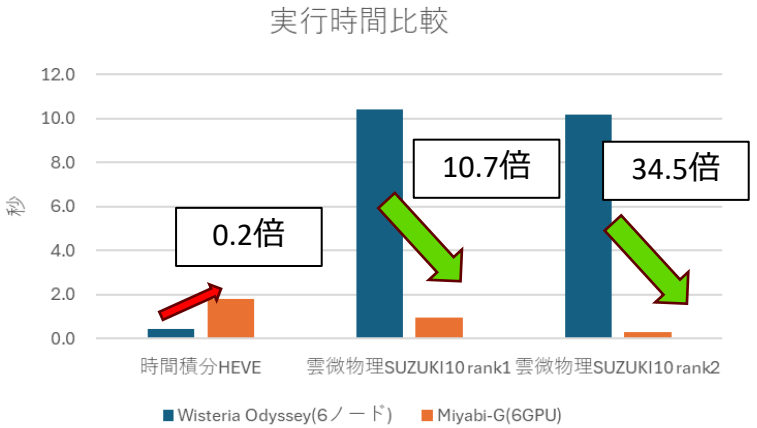
システム	プロセッサ名	使用リソース	使用リソースの倍精度演算性能
Miyabi-C(CPU)	Intel Xeon MAX 9480	56コア	3.4TFLOPS
Miyabi-G(GPU)	NVIDIA Hopper H100	1GPU	66.9TFLOPS



SCALEのCPUとGPUの実行時間比較

- CPU(OdysseyのFujitsu A64FX)実行とGPU(Miyabi-GのNVIDIA H100)実行の比較
  - Odyssey実行　:6ノード6プロセス48スレッド実行
  - Miyabi-G実行　：6ノード6プロセス6GPU実行
  - 時間積分HEVEはGPU化前の計算時間が短いため、OpenACC化による演算処理の実行時間短縮より、CPU-GPU間のデータ転送、カーネルの起動オーバーヘッドのコストが優位になっていると考えられる。

システム	プロセッサ名	使用リソース	使用リソースの倍精度演算性能
Odyssey(CPU)	Fujitsu A64FX	6ノード288コア	20.2TFLOPS
Miyabi-G(GPU)	NVIDIA Hopper H100	6GPU	401.4TFLOPS



FUJITSU

OSSのGPGPU化事例

OpenFOAMのGPGPU化

FUJITSU

- OpenFOAM概要
  - オープンソースの流体解析ソフトウェア
  - 科学反応、乱流、熱伝導を伴う流体から、固体力学や電磁気学まで対応できる幅広い機能を備える
- OSSのGPGPU化
  - GPU対応しているライブラリの活用
    - cuBLAS、cuFFT、cuSOLVERなどのNVIDIA製ライブラリ
    - サードパーティが公開しているGPUに対応したライブラリ

OpenFOAM®

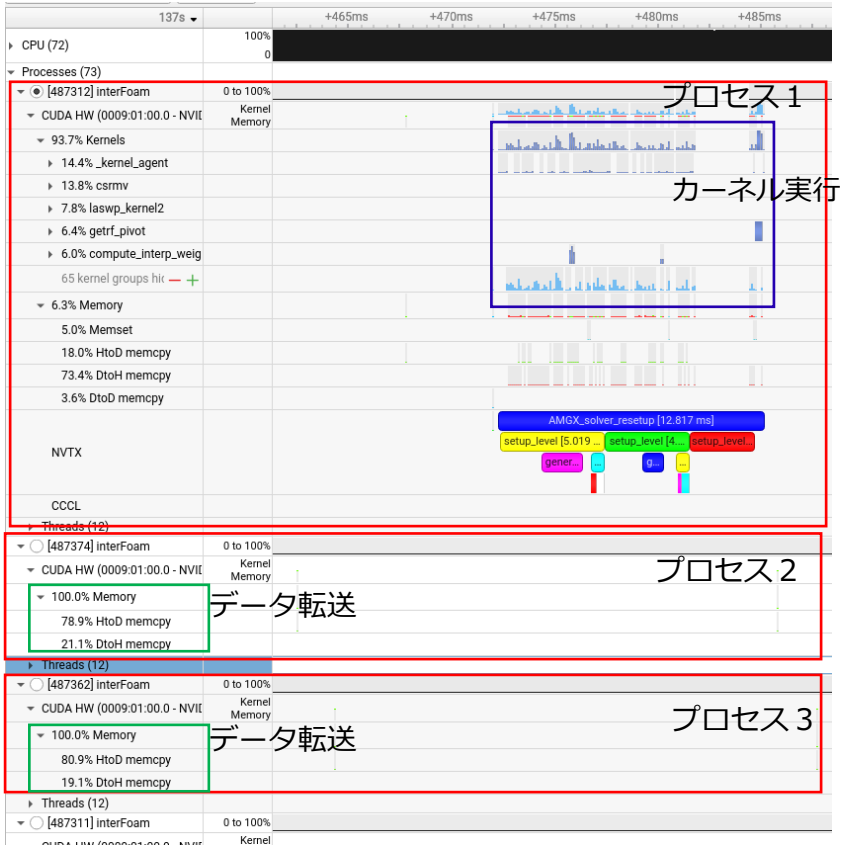
https://www.openfoam.com/

- OpenFOAMのGPGUP化では、GPU対応しているライブラリを利用
  - AmgX：GPUを活用した代数マルチグリッドソルバーライブラリであり、計算負荷が大きい線形ソルバー部分を高速化可能
  - AmgX含むOpenFOAMのGPU実行で必要なライブラリのビルド手順をNVIDIAと協力して検証し、導入後も連携して安定動作のサポートを実施

OpenFOAMのプロセス並列実行時のGPU利用状況

FUJITSU

- Miyabi-G 1ノードはCPU 72コア、1GPU搭載
  - 実行にあたりCPUコアをフルに活用するため、72プロセスを起動させた場合のGPU利用の状況を確認した
- プロファイラによるGPU利用状況の確認
  - OpenFOAMを72プロセス並列で実行し、NVIDIA Nsight Systemsを用いてプロファイラを取得した
  - GPUカーネルを実行するプロセスは1つに集約されており、その他のプロセスではCPU-GPU間のデータ転送の動きが確認できる
  - MPIプロセス数がGPUデバイス数より多い場合には、AmgXWrapperライブラリ※1が、1GPUに対して1つのMPIプロセスのみGPUを使用するように調整



※1:https://github.com/barbagroup/AmgXWrapper

OpenFOAMのCPUとGPUの計算時間比較

FUJITSU

- 「2025年1月29日（水）第240回お試しアカウント付き並列プログラミング講習会「OpenFOAM中級・3次元ダムブレイク解析」」※1で紹介されているダムブレイク解析※2をMiyabi-G上で実行した。
- CPU実行はMiyabi-GのNVIDIA Grace CPU、GPUオフロード実行はMiyabi-GのNVIDIA H100を利用した。

領域分割数(プロセス並列数) ：72
ノード数 ： 1
セル数 ：50,625
時間条件：startTime 0, endTime 2,deltaT 0.001
■GPUオフロード実行時間
ExecutionTime = 169.18 s
■CPU実行時間
ExecutionTime = 15.75 s

5万セルではGPUオフロード実行はCPU実行に対して約10倍ほど遅い

セル数を増加

領域分割数(プロセス並列数) ：72
ノード数 ：1
セル数 ：8,000,000
時間条件：startTime 0, endTime 0.1, deltaT 0.001
■GPUオフロード実行時間
ExecutionTime = 110.12 s
■CPU実行時間
ExecutionTime =117.41 s

800万セルではGPUオフロード実行の方がCPU実行よりわずかに速い

セル数を増やすことでGPUオフロード実行による高速化が望めると考えられる

※1:https://gitlab.com/OpenCAE/Supercomputer-OpenFOAM-Training/-/wikis/lectures/The-University-of-Tokyo-Information-Technology-Center/20250129
※2:https://gitlab.com/OpenCAE/Supercomputer-OpenFOAM-Training/-/tree/master/Miyabi/damBreakMARIN?ref_type=heads

FUJITSU