Wisteria/BDEC-01 利用事例（4）
データ受け渡しライブラリ h3-Open-SYS/WaitIO (2/2)

住元真司（東京大学情報基盤センター）
坂口吉生（富士通株式会社）
松葉浩也* (日立製作所)
*2021 年度客員教授時の成果
中島研吾（東京大学情報基盤センター）

1. はじめに

今回は、前回（2022 年 3 月号）と次回（2022 年 7 月号）との 3 回からなる 2 回目で, Wisteria/BDEC-01［1,2,3］のようなヘテロジニアスな環境で「計算・データ・学習」融合を実現するためのツールであ h3-Open-SYS/WaitIO［4］について、特に Wisteria/BDEC-01 システムでの実機利用を想定した利用事例について紹介する。

2. WaitIO Hostname Server

前回の実行では WaitIO のプログラムをバッチ実行するには PBID が 0 のホスト名とポート番号をバッチ実行開始後にログファイルから獲得して PBID 1 以降のバッチジョブスクリプトを変更した上で実行する必要があった。バッチ処理は実行時にホストが決まるからである。この状況を改善するため、実行時に決まるホスト名を実行時に中継する WaitIO Hostname Server を開発している。現在、システム連携に向けての改善継続中のため、今後仕様などが変更される可能性があるので注意されたい。今回は現状の動作仕様について説明する。

WaitIO Hostname Server は計算ノードから直接ネットワーク通信が可能なホストで実行可能なサーバであれば実行可能である。Wisteria/BDEC-01 システムではログインノードで実行可能である。実行バイナリとしては、intel CPU(waitio-serv, waitio-serv-intel : ログインノードと Aquarius)用と Arm CPU(waitio-serv-a64fx : Odyssey)用があり実行するノードで使い分ける必要がある。

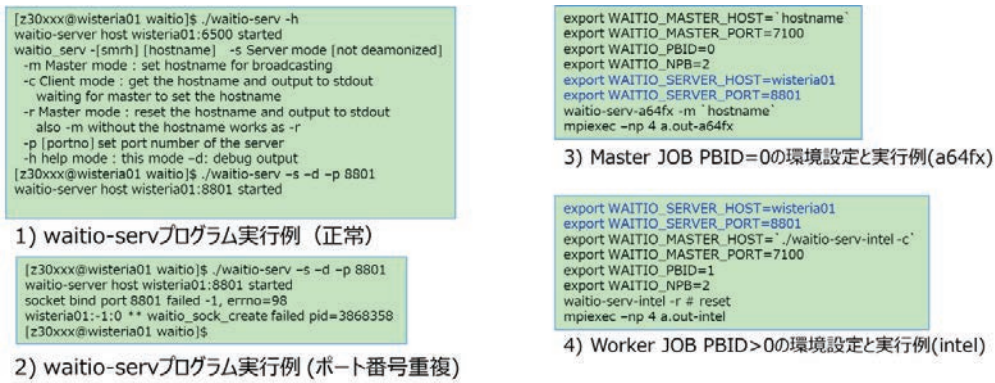

図 1. WaitIO Hostname Server の環境設定例

図 1 に WaitIO Hostname Server の環境設定例を示す。前回に示した実行例に WAITIO_SERVER_HOST と WAITIO_SERVEWR_PORT の環境変数を設定して ログインノードな

どで waitio_serv コマンドをサーバモードで起動（-s オプション、図 1、1)）し、各バッチスクリプトで実行（図 1、3）, 4)）することでバッチジョブ実行時に waitio_serv サーバを介してマスターホスト情報の中継が可能である。なお、waitio-serv コマンドをサーバモードで立ち上げる時、既にポート番号が使われていると Socket の取得に失敗することがある。（図 1 、2)）このような場合は重複した番号と異なるポート番号を再設定して実行する。

3. WaitIO-Socket のアプリケーション適用事例

WaitIO-Socket の利用事例としては、h3-Open-UTIL/MP［5］と観測データ同化による長周期地震動リアルタイム予測［11］で活用が進んでいる。

h3-Open-UTIL/MP は WaitIO-Socket と各 PB で実行される MPI ライブラリを用いて Hybrid な MPI 実行環境を実現し、複数の MPI アプリケーションを融合するカップラーとしてアプリケーション間のデータコンバージョンを含む実行環境を提供している。シミュレーションとデータ同化による学習の融合を実現する。詳しくは次回（2022 年 7 月号）の記事を参照されたい。

観測データ同化による長周期地震動リアルタイム予測［11］において WaitIO は日本各所に設置されている地震観測点からのデータをリアルタイムで転送し Wisteria/BDEC-01 システムでの予測に活用するために開発を進めている。

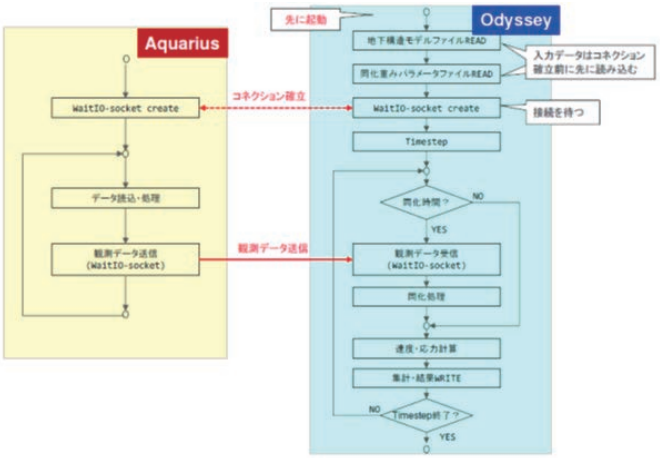

図 2. 観測データ同化による長周期地震動リアルタイム予測における WaitIO 活用例

図 2 に観測データ同化による長周期地震動リアルタイム予測における WaitIO 活用例を示す。観測済みデータを用いて地震動を解析する場合、通常蓄積されたデータをファイルに格納してこのファイルをアプリケーションに読み込ませて解析処理を実行する。

しかし、長周期地震動リアルタイム予測においては逐一記録される地震データをリアルタイムに解析する必要がある。WaitIO-Socket はこのファイル読み出しを通信に置き換える。

```
program dmy_filter
<省略: 型宣言等>
call mpi_init (ierr)
call mpi_comm_size (MPI_COMM_WORLD, nprocs, ierr)
call mpi_comm_rank (MPI_COMM_WORLD, myrank,  ierr)
call WAITIO_CREATE_UNIVERSE (WAITIO_COMM_UNIVERSE, ierr)

if (myrank==0) then
  open(100,file='./obsfile_list.txt', form='formatted', status='old', iostat=ierr)
  do i=1,300
   <省略: obsデータ読み込み処理>
    print *,"Send obs data ..... "
    call WAITIO_MPI_ISEND (NTMAX1_o, 1,          WAITIO_MPI_INTEGER,  2,1, WAITIO_COMM_UNIVERSE,req(1,1), ierr)
    call WAITIO_MPI_ISEND (DT_o,     1,          WAITIO_MPI_FLOAT,    2,2, WAITIO_COMM_UNIVERSE,req(1,2), ierr)
    call WAITIO_MPI_ISEND (NST_o,    1,          WAITIO_MPI_INTEGER,  2,3, WAITIO_COMM_UNIVERSE,req(1,3), ierr)
    call WAITIO_MPI_ISEND (AT_o,     1,          WAITIO_MPI_FLOAT,    2,4, WAITIO_COMM_UNIVERSE,req(1,4), ierr)
    call WAITIO_MPI_ISEND (T0_o,     1,          WAITIO_MPI_FLOAT,    2,5, WAITIO_COMM_UNIVERSE,req(1,5), ierr)
    call WAITIO_MPI_ISEND (ISO_X_o,  NSMAX,      WAITIO_MPI_INTEGER,  2,6, WAITIO_COMM_UNIVERSE,req(1,6), ierr)
    call WAITIO_MPI_ISEND (ISO_Y_o,  NSMAX,      WAITIO_MPI_INTEGER,  2,7, WAITIO_COMM_UNIVERSE,req(1,7), ierr)
    call WAITIO_MPI_ISEND (ISO_Z_o,  NSMAX,      WAITIO_MPI_INTEGER,  2,8, WAITIO_COMM_UNIVERSE,req(1,8), ierr)
    call WAITIO_MPI_ISEND (ISTX_o,   NST,        WAITIO_MPI_INTEGER,  2,9, WAITIO_COMM_UNIVERSE,req(1,9), ierr)
    call WAITIO_MPI_ISEND (ISTY_o,   NST,        WAITIO_MPI_INTEGER,  2,10,WAITIO_COMM_UNIVERSE,req(1,10),ierr)
    call WAITIO_MPI_ISEND (ISTZ_o,   NST,        WAITIO_MPI_INTEGER,  2,11,WAITIO_COMM_UNIVERSE,req(1,11),ierr)
    call WAITIO_MPI_ISEND (STC_o,    6*NST,      WAITIO_MPI_CHAR,     2,12,WAITIO_COMM_UNIVERSE,req(1,12),ierr)
    call WAITIO_MPI_ISEND (VxAll_obs,NST*NOBS_LEN,WAITIO_MPI_FLOAT,   2,13,WAITIO_COMM_UNIVERSE,req(1,13),ierr)
    call WAITIO_MPI_ISEND (VyAll_obs,NST*NOBS_LEN,WAITIO_MPI_FLOAT,   2,14,WAITIO_COMM_UNIVERSE,req(1,14),ierr)
    call WAITIO_MPI_ISEND (VzAll_obs,NST*NOBS_LEN,WAITIO_MPI_FLOAT,   2,15,WAITIO_COMM_UNIVERSE,req(1,15),ierr)
    call WAITIO_MPI_WAITALL (15,req, status, ierr)
    call sleep(1)
  enddo
  close (100)
endif
call WAITIO_FINALIZE (ierr)
call mpi_finalize (ierr)
end
```

図 3. 観測データ同化による長周期地震動リアルタイム予測における WaitIO 活用例：データ供給側

図 3, 図 4 に WaitIO-Socket を用いた長周期地震動リアルタイム予測で用いるデータ転送用のプログラムを示す。通常、システム間の通信は TCP/IP プロトコル上の Socket を使うことになるが、Socket の作成、相手サーバとの通信確立など Socket 通信自体のプログラム知識が必要であるが、WaitIO-Socket を用いることで Socket の知識がなくても MPI プログラムを書くことができれば、サーバ間の通信プログラムを簡易に書くことができる。

図 3 は地震データを格納するサーバにおけるプログラムでサーバ上のファイルを読んで WaitIO MPI Conversion ライブラリを用いてリアルタイム予測を実行するシステムに転送する。

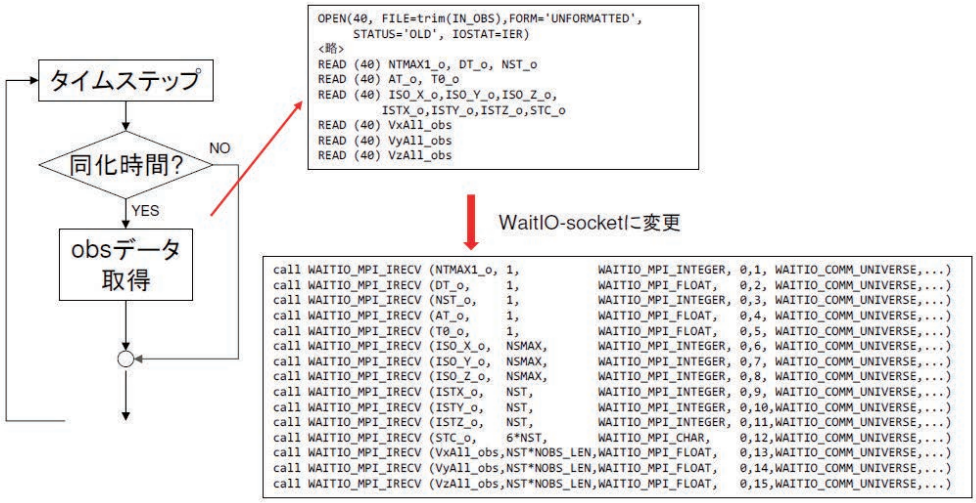

図 4. 観測データ同化による長周期地震動リアルタイム予測における WaitIO 活用例：データ活用側

図 4 は元々ファイルから地震データを読んで処理するプログラムであったものを WaitIO MPI Conversion ライブラリを用いて別のサーバからネットワークを経由してデータを獲得している。

このように Socket 通信のプログラム技術がなくても WaitIO Socket を用いて手軽にネットワークを介してデータを転送できることがわかる。

4. Toy プログラムによる実アプリケーションコード実行

本章では実際のアプリケーションプログラムを簡略化した Toy プログラムにより実際の WaitIO-Socket によるアプリケーション実行を試してみる。アプリケーション実行については、6 月より開始予定の Wisteria/BDEC-01 システムでの WaitIO 試行サービス環境下での実行手順を説明する。

ここでは、点源による発熱量計算結果を出力するプログラム(Code-1)のファイル出力を、有限要素法による三次元非定常熱伝導解析するプログラム(Code-2)、並びに Code-2 と Code-1 からのファイル出力を元に有限要素法による三次元非定常熱伝導解析を行い Z=Zmax 面の境界条件へ展開するプログラム(Code-3)の 3 つのプログラムを WaitIO-Socket を用いて書き換えてみる。ソースコード一式は Wisteria/BDEC-01 システム上の /work/share/waitio/src/examples/SYStest-waitio-socket.tgz に置かれているのでそれぞれのアカウントの work ディレクトリに展開して参照されたい。

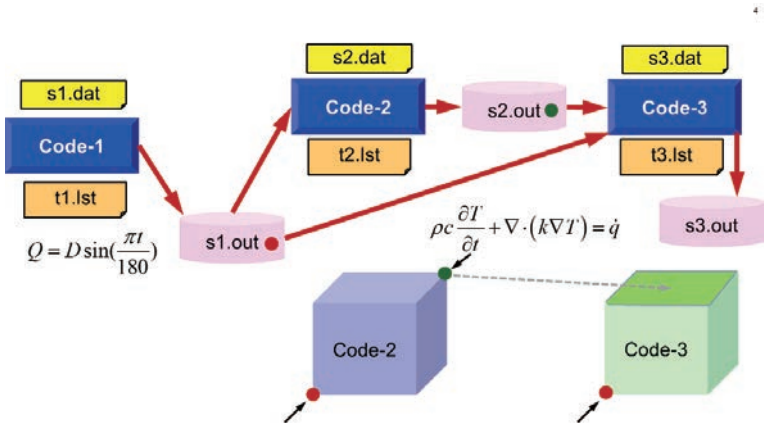

図 4. 有限要素法による三次元非定常熱伝導解析 Toy プログラムの構成

図 4 に本 Toy プログラムの全体像を、表 1 に各プログラムの概要と入出力ファイルについて示す。Code-1 の出力するファイル s1.out は Code-2 と Code-3 から読み込まれる。また、Code-2 の出力する s2.out は Code-3 から読み込まれる。これらのファイルによる入出力を WaitIO-Socket のデータ転送に書き換える。

表 1. 有限要素法による三次元非定常熱伝導解析 Toy プログラムの概要

	Code-1 (Dir:SYStest-waitio-socket/src1)	Code-2 (Dir:SYStest-waitio-socket/src2)	Code-3 (Dir:SYStest-waitio-socket/src3)
概要	発熱量(点源)計算	FEMによる非定常三次元熱伝導方程式, MPI並列化	FEMによる非定常三次元熱伝導方程式, MPI並列化
入力 (Dir:SYStest-waitio-socket/run)	s1.dat	s2.dat s1.out	s3.dat s1.out, s2.out
出力 (Dir:SYStest-waitio-socket/run)	s1.out, t1.lst	s2.out, t2.lst	s3.out, t3.lst

4.1 ソースコードの修正

ファイル出力から WaitIO でのデータ連携への変更は、データ共有しているファイルへの入出力部分のソースコードを WaitIO による送受信のコードによる送受信のソースコードに変更することで実現可能である。

WaitIO におけるソースコードの修正手順を以下に示す。

1. 各アプリケーションプロセス間の接続方式と利用 API の決定

   WaitIO MPI Conversion ライブラリでは 2 種類の簡易初期化関数を準備しています。

   1) 各アプリケーションの MPI プロセスのすべてを用いる waitio_create_universe()関数
   2) 各アプリケーションの MPI プロセスの先頭プロセス(MPI ランク番号 0)だけを使う waitio_create_universe_pbhead()関数

   1), 2)以外の場合は waitio_create_group()関数を用いて個別に作成することが可能です。

   また、実際の通信のデータ形式や通信パターンに応じて WaitIO 関数(waitio_isend, waitio_irecv)、もしくは WaitIO MPI Conversion ライブラリ(waitio_mpi_xxx()関数)を選択可能です。ここでは、アプリケーション間の接続方式として、各アプリケーションの先頭プロセスのみを選択する waitio_create_universe_pbhead()関数を選択し、利用関数として表記法が MPI と類似している WaitIO MPI Conversion ライブラリで記述することにします。
2. 各アプリケーションの PBID へのマッピング

   1 で述べた接続方式と PBID により、WaitIO で用いるプロセスランク番号が異なるため、アプリケーションの結合順序を決めるために各アプリケーションの PBID を決定します。ここでは、Code-1 を PBID=0, Code-2 を PBID=1, Code-3 を PBID=2 と定義します。
3. アプリケーション間のデータ連携コードのプログラミング

   さて、実際のデータ連携コードのプログラミングであるが、今回は、ファイル入出力によるデータ交換を WaitIO で書き換えることになるため、実際のファイル入出力がどのように行われているのかを整理する必要がある。

以下の節では実際のコード修正を元に詳しく述べる。

4.2 アプリケーション間のデータ連携コードのプログラミングの実際

本節では Toy プログラムのデータ連携コードを WaitIO MPI Conversion ライブラリの関数を用い

て書き換える。Toy プログラムである SYStest-waitio-socket ディレクトリ下には各プログラムを格納する src1(Code-1), src2(Code-2), src3(Code-3)の各ディレクトリと実行を行う run ディレクトリから構成される。

```
001   implicit REAL*8(A-H,O-Z)
002   real(kind=8) :: Interval
003   include 'mpif.h'

004   call MPI_Init(ierr)
005   open (11,file='s1.dat',status='unknown')
006   read (11,*) ITERmax, Period, Interval, Val
007   write (*,'(a,i8)')     '##ITERmax ', ITERmax
008   write (*,'(a,1pe16.6)') '##Period  ', Period
009   write (*,'(a,1pe16.6)') '##Interval', Interval
010   write (*,'(a,1pe16.6//)') '##Val     ', Val
011   close (11)

012   open (12,file='s1.out',status='unknown')

013   pi  = 4.d0*datan(1.d0)
014   coef= pi/180.d0
015   time= 0.d0
016   S0time= mpi_wtime ()
017   do
018    S1time= mpi_wtime ()

019    do
020     do iter= 1, ITERmax
021       a= 1.d0
022     enddo
023     E0time= mpi_wtime (ierr)
024     if ((E0time-S1time).gt.Interval) exit
025    enddo

026    ttt= E0time - S0time
027    Source= Val * dsin(ttt*coef)
028 !  rewind (12)
029    write (12,'(2(1pe16.6))') ttt, Source
030   enddo

031   call MPI_Finalize()
032   stop
033   end
```

図 5. Toy プログラムの Original プログラム：Code-1 全体ソースコード(FORTRAN)

File:SYStest-waitio-socket/src1/test1.f

図 5 に、Toy プログラムの内 Code-1(FORTRAN)の Original の全体ソースコードを示す。図 5 中、001-016 行目までは初期化コードであり、004 行で MPI の初期化後、005 行でパラメータファイルを Open し、006 行でパラメータを読み込んでいる。そして、012 行で出力用のファイル s1.out を Open した後、017 行からのループ本体のための変数初期化を行っている。

017-030 行が Code-1 の本体ループである。017-030 行のループ本体では計算結果を 029 行の write 文で変数 ttt と Source をループ毎に書き出している。

今回、図 5 の Code-1 コードを WaitIO 対応のコードにするために、まず、最初に各ファイルのプログラム間の依存関係を把握する必要がある。図 4、表 1 より、Code-1 にて出力される s1.out は、Code-2, Code-3 で読みだされている。このため、Code-1 の s1.out 出力は Code-2, Code-3 に送信する必要がある。表 1 より、Code-2 においては Code-1 からのデータを読み込み、Code-2 へ送信、Code-3 においては、Code-1 と Code-2 からのデータを読み込む必要がある。

```
001    implicit REAL*8(A-H,O-Z)
002    real(kind=8) :: Interval
003    integer :: dest
004    integer(kind=4 ), dimension(:,:), save,allocatable :: sta1
005    integer(kind=4 ), dimension(:,:), save,allocatable :: req1

006    include 'mpif.h'
007    include 'waitio_mpif.h'

008    call MPI_Init(ierr)
009    open (11,file='s1.dat',status='unknown')
010    read (11,*) ITERmax, Period, Interval, Val
011    write (*,'(a,i8/)')     '##ITERmax ', ITERmax
012    write (*,'(a,1pe16.6)') '##Period  ', Period
013    write (*,'(a,1pe16.6)') '##Interval', Interval
014    write (*,'(a,1pe16.6//)') '##Val     ', Val
015    close (11)

016    allocate (sta1(WAITIO_STATUS_SIZE,2*2))
017    allocate (req1(WAITIO_REQUEST_SIZE,2*2))
018    call WAITIO_CREATE_UNIVERSE_PBHEAD (WAITIO_SOLVER_COMM, ierr)

019    open (12,file='s1.out',status='unknown')
```

```
037      write (*,'(2(1pe16.6))') ttt, Source
038      do dest=1, 2
039        call WAITIO_MPI_Isend (ttt, 1,
040    &        WAITIO_MPI_DOUBLE_PRECISION,
041    &        dest, 0, WAITIO_SOLVER_COMM, req1(1,2*(dest-1)+1), ierr)
042        if(ierr.ne.0) goto 100
043        call WAITIO_MPI_Isend (Source, 1,
044    &        WAITIO_MPI_DOUBLE_PRECISION,
045    &        dest, 0, WAITIO_SOLVER_COMM, req1(1,2*(dest-1)+2), ierr)
046        if(ierr.ne.0) goto 100
047      enddo
048      call WAITIO_MPI_Waitall (4, req1, sta1, ierr)
049      if(ierr.ne.0) goto 100
050    enddo
051 100  continue
```

図 6. Toy プログラムの WaitIO プログラム：Code-1 修正部

File:SYStest-waitio-socket/src1/test1_waitio.f

図 6 に、Toy プログラムの WaitIO プログラム Code-1 の修正済み FORTRAN コードを示す。図 6 中、青字のラインが追加したコードである。図 6 にて 038-048 行のループでファイル出力される変数 ttt と Source を WaitIO MPI Conversion ライブラリを用いて Code-2(WatiIO ランク番号= 1)と Code-3(WatiIO ランク番号=2)に送信している。この 2 つの変数の送信には 2 つの WAITIO_MPI_Isend()関数を用い、2 か所に送信するために WaitIO ランク番号 1,2 にループを用いている。それぞれの WAITIO_MPI_Isend()関数に必要な Request 構造体は 005,017 行で 2 次元配列として割り当てている。038-048 行のループでそれぞれの WaitIO ランク番号 1,2 に対して WAITIO_MPI_Isend()関数を 2 回発行した後、048 行の WATIO_MPI_Waitall()関数で相手が受信したことを待ち合わせる。

なお、018 行の WAITIO_CREATE_UNIVERSE_PBHEAD() 関数の引数である WAITIO_SOLVER_COMM 変数は 8 バイトのポインターが格納されるため 8 バイトの変数定義が必要である。本コードでは implicit 定義により 8 バイトの REAL 変数として定義されている。

```
007    integer :: dest
008    integer(kind=kint ), dimension(:,:), save,allocatable :: sta1
009    integer(kind=kint ), dimension(:,:), save,allocatable :: req1

010    allocate (sta1(WAITIO_STATUS_SIZE,1*6))
011    allocate (req1(WAITIO_REQUEST_SIZE,1*6))

!=== SNIP SNIP ===

053!c       read (12,*,end=100) ttt, Source
054         dest=0
055         call WAITIO_MPI_Irecv (ttt, 1,
056    &        WAITIO_MPI_DOUBLE_PRECISION,
057    &        dest, 0, WAITIO_SOLVER_COMM, req1(1,1), ierr)
058         call WAITIO_MPI_Irecv (Source, 1,
059    &        WAITIO_MPI_DOUBLE_PRECISION,
060    &        dest, 0, WAITIO_SOLVER_COMM, req1(1,2), ierr)

061         call WAITIO_MPI_Waitall (2, req1, sta1, ierr)
```

```
100      write (22,'(6(1pe16.6))') TIME, ttt, Source, X(ii1),
101    &                           X(ii2), VAL
102      dest= 2
103      call WAITIO_MPI_Isend (TIME, 1,
104    &      WAITIO_MPI_DOUBLE_PRECISION,
105    &      dest, 0, WAITIO_SOLVER_COMM, req1(1,1), ierr)
106      call WAITIO_MPI_Isend (ttt, 1,
107    &      WAITIO_MPI_DOUBLE_PRECISION,
108    &      dest, 0, WAITIO_SOLVER_COMM, req1(1,2), ierr)
109      call WAITIO_MPI_Isend (Source, 1,
110    &      WAITIO_MPI_DOUBLE_PRECISION,
111    &      dest, 0, WAITIO_SOLVER_COMM, req1(1,3), ierr)
112      call WAITIO_MPI_Isend (X(ii1), 1,
113    &      WAITIO_MPI_DOUBLE_PRECISION,
114    &      dest, 0, WAITIO_SOLVER_COMM, req1(1,4), ierr)
115      call WAITIO_MPI_Isend (X(ii2), 1,
116    &      WAITIO_MPI_DOUBLE_PRECISION,
117    &      dest, 0, WAITIO_SOLVER_COMM, req1(1,5), ierr)
118      call WAITIO_MPI_Isend (VAL, 1,
119    &      WAITIO_MPI_DOUBLE_PRECISION,
120    &      dest, 0, WAITIO_SOLVER_COMM, req1(1,6), ierr)

121      call WAITIO_MPI_Waitall (6, req1, sta1, ierr)
```

図 7. Toy プログラムの WaitIO プログラム：Code-2 修正部

File:SYStest-waitio-socket/src2/test1_waitio.f

図 7 に、Toy プログラムの WaitIO プログラムの内 Code-2 の修正済み FORTRAN コードを示す。この修正の他初期化コードも Code-1 と同様に追加されている。Code-1 と同様 WaitIO MPI Conversion ライブラリに置き換えている。

図 7 中、054-061 行は Code-1(WatiIO ランク番号=0)からのデータ受信である。コメントアウトされた 053 行を WaitIO MPI Conversion ライブラリに書き換えたものである。また、102-121 行は同じく 100 行のファイル出力を WaitIO MPI Conversion ライブラリに書き換え、Code-3(WatiIO ランク番号=2)への通信に変換したものである。

```
007     integer :: dest
008     integer(kind=kint ), dimension(:,:), save,allocatable :: sta1
009     integer(kind=kint ), dimension(:,:), save,allocatable :: req1

010     allocate (sta1(WAITIO_STATUS_SIZE,1*6))
011     allocate (req1(WAITIO_REQUEST_SIZE,1*6))

!=== SNIP SNIP ===

054IC       read (12,*,end=200) ttx, Source
055         dest=0
056         call WAITIO_MPI_Irecv (ttx, 1,
057     &       WAITIO_MPI_DOUBLE_PRECISION,
058     &       dest, 0, WAITIO_SOLVER_COMM, req1(1,1), ierr)
059         call WAITIO_MPI_Irecv (Source, 1,
060     &       WAITIO_MPI_DOUBLE_PRECISION,
061     &       dest, 0, WAITIO_SOLVER_COMM, req1(1,2), ierr)

062         call WAITIO_MPI_Waitall (2, req1, sta1, ierr)
063         if (ttx.ge.TIME) goto 200
```

```
071IC       read (13,*,end=100) ttt, tt1, s0, s1, s2, TBOU
072         dest= 1
073         call WAITIO_MPI_Irecv (ttt, 1,
074     &       WAITIO_MPI_DOUBLE_PRECISION,
075     &       dest, 0, WAITIO_SOLVER_COMM, req1(1,1), ierr)
076         call WAITIO_MPI_Irecv (tt1, 1,
077     &       WAITIO_MPI_DOUBLE_PRECISION,
078     &       dest, 0, WAITIO_SOLVER_COMM, req1(1,2), ierr)
079         call WAITIO_MPI_Irecv (s0, 1,
080     &       WAITIO_MPI_DOUBLE_PRECISION,
081     &       dest, 0, WAITIO_SOLVER_COMM, req1(1,3), ierr)
082         call WAITIO_MPI_Irecv (s1, 1,
083     &       WAITIO_MPI_DOUBLE_PRECISION,
084     &       dest, 0, WAITIO_SOLVER_COMM, req1(1,4), ierr)
085         call WAITIO_MPI_Irecv (s2, 1,
086     &       WAITIO_MPI_DOUBLE_PRECISION,
087     &       dest, 0, WAITIO_SOLVER_COMM, req1(1,5), ierr)
088         call WAITIO_MPI_Irecv (TBOU, 1,
089     &       WAITIO_MPI_DOUBLE_PRECISION,
090     &       dest, 0, WAITIO_SOLVER_COMM, req1(1,6), ierr)

091         call WAITIO_MPI_Waitall (6, req1, sta1, ierr)
092         if (ttt.ge.TIME) goto 100
```

図 8. Toy プログラムの WaitIO プログラム：Code-3 修正部

File:SYStest-waitio-socket/src3/test1_waitio.f

図 8 に、Toy プログラムの WaitIO プログラムの内 Code-3 の修正済み FORTRAN コードを示す。この修正の他初期化コードも Code-1, Code-2 と同様に追加されている。Code-1, Code-2 と同様 WaitIO MPI Conversion ライブラリに置き換えている。

図 8 中、Code-2 と同様に 055-062 行は Code-1(WatiIO ランク番号=0)からのデータ受信である。コメントアウトされた 054 行を WaitIO MPI Conversion ライブラリに書き換えたものである。また、072-091 行は 071 行のファイル入力を WaitIO MPI Conversion ライブラリに書き換え、Code-2(WatiIO ランク番号=2)からのデータ受信に変換したものである。

以上のように、ファイル入出力を WaitIO MPI Conversion ライブラリに置き換えるのは機械的に行うことが可能である。

4.3 Wisteria/BDEC-01 における WaitIO 実行環境の概要

本節では 6 月よりサービス開始を予定している WaitIO の試行環境について説明する。

- WaitIO 関連のファイルは Wistria/BDEC-01 の/work/share/waitio 下に配置される。
  Toy プログラム等、実行用のテストサンプルは以下に配置される /work/share/waitio/src/examples/
- module 実行環境として waitio が提供され PATH、LD_LIBRARY_PATH 等の実行に必要な設定が提供される。例えばバッチスクリプト内で module load waitio とするとジョブ実行に必要な環境設定が実行される。

- この他、Wisteria/BDEC-01 の Odyssey と Aquarius 間でのバッチ処理協調実行のための仕組みも準備中である。

これ以降の Toy プログラムのコンパイルと実行の説明では以上の環境下における試行について説明する。

4.4 Toy プログラムのコンパイルと実行

本節では 6 月に Wistria/BDEC-01 での WaitIO サービス開始時の環境を想定したプログラムのコンパイルと実行の概要を説明する。前節で書いた通り WaitIO に関連するファイル群は Wistria/BDEC-01 の/work/share/waitio ディレクトリに置かれている。Toy プログラムのソース並びに実行環境は/work/share/waitio/src/examples/SYStest-waitio-socket.tgz に一式アーカイブされているので各ユーザの/work 下のディレクトリに展開して利用する。図 9 に一連の処理を示す。

Toy プログラムのアーカイブを各ユーザの work ディレクトリに展開するとコード一式は SYStest-waitio-socket ディレクトリ下に展開されるので、このディレクトリ下で make コマンドを実行すると SYStest-waitio-socket 配下の run ディレクトリに Odyssey 用バイナリ(-a64fx)と Aquarius 用バイナリ(-intel)の 3 種類 x 2 CPU タイプ＝計 6 種の WaitIO 用の実行バイナリとオリジナルの実行バイナリ 3 種が生成される。(図 9)

```
0) Extract the Toy program to your /work directory and compile
[z30xxx@wisteria01 sample]$ realpath .
/work/jh21yyyya/z30xxx/sample
[z30xxx@wisteria01 sample]$ tar -zxf /work/share/waitio/src/examples/SYStest-waitio-socket.tgz
[z30xxx@wisteria01 sample]$ cd SYStest-waitio-socket/
[z30xxx@wisteria01 SYStest-waitio-socket]$ make
(cd src1 ; make )
....
make[1]: Leaving directory '/work/01/jh210022a/z30455/sample/SYStest-waitio-socket/src3'
[z30xxx@wisteria01 SYStest-waitio-socket]$
[z30xxx@wisteria01 SYStest-waitio-socket]$ ls -t run/
sol3-a64fx  sol2       sol1    s3-intel.sh  s1-a64fx.sh  s2.dat  OrgResults
sol3-intel  sol2-intel  s3.out  s2-intel.sh  s3-a64fx.sh  s3.sh
sol3        sol1-a64fx  s2.out  s2-a64fx.sh  s1.dat       s2.sh
sol2-a64fx  sol1-intel  s1.out  s1-intel.sh  s3.dat       s1.sh
[z30xxx@wisteria01 SYStest-waitio-socket]$ cd run/
[z30xxx@wisteria01 run]$
```

図 9. WaitIO 版 Toy プログラムのファイル展開とコンパイル

図 10. 11. に Wisteria/BDEC-01 システム(Odyssey+Aquarius)で実行可能なバッチスクリプトを示す。これらのバッチスクリプトは wisteria01 ノード上で 2 章説明の waitio-serv コマンドによるサーバの利用が前提である。これにより、ジョブ実行時のバッチスクリプトの修正は不要である。また、waitio module の利用により環境変数 PATH と LD_LIBRARY_PATH の設定が不要になっている。

図 10. Toy プログラムの WaitIO バッチ実行用スクリプト: a64fx

図 11. Toy プログラムの WaitIO バッチ実行用スクリプト: intel

図 10、図 11 のバッチスクリプトは Code-1+Code-2+Code-3 の組み合わせであれば a64fx 用、intel 用どの組み合わせでも動作可能である。実行時に注意することは Code-1, Code-2 のジョブを Submit した後、Code-2 が実行開始した後に Code-3 を Submit することである。これは、Code-3 のジョブスクリプトの mpiexec 行の前に実行される waitio-serv –r コマンドが登録されたサーバ設定をリセットするためである。このため、リセット後に Code-2 のジョブが実行開始されてもサーバ名を獲得できず実行待ちになる。

図 12, 図 13 に実際の WaitIO 版 Toy プログラムの実行例を示す。図 12 が実際のジョブ実行の様子を示しており、図 13 が一連のジョブ実行に必要な waitio-serv プログラムの実行出力である。

図 12 では、Code-1(Intel)+Code-2(a64fx)+Code-3(a64fx)の組み合わせでの実行結果であるが、Code-1, Code-2, Code-3 のそれぞれ任意の a64fx 並びに intel 用のスクリプトを選択可能である。

図 13 の waitio-serv プログラムの出力であるが、Code-1 による Master Host の登録に始まり、Code-3 による Master Host 登録のリセットで一連のジョブ実行が終了する。

```
1) Submit Code-1 and Code-2 Jobs and wait until both jobs running
[z30xxx@wisteria01 run]$ pjsub s1-intel.sh
[INFO] PJM 0000 pjsub Job 335684 submitted.
[z30xxx@wisteria01 run]$ pjsub s2-a64fx.sh
[INFO] PJM 0000 pjsub Job 335685 submitted.
[z30xxx@wisteria01 run]$ pjstat
Wisteria/BDEC-01 scheduled stop time: 2022/04/22(Fri) 09:00:00 (Remain:  1days 18:55:30)

JOB_ID    JOB_NAME  STATUS  PROJECT   RSCGROUP      START_DATE      ELAPSE      TOKEN      NODE  GPU
335684    SYStest-wa RUNNING jhxxxxxxa  debug-a       04/20 14:04:22<  00:00:09      0.1        1   8
335685    SYStest-wa RUNNING jhxxxxxxo  debug-o       04/20 14:04:23<  00:00:08      0.0        1   -

2) Then Submit Code-3 Job and wait until Code-2 and Code-3 finish
[z30xxx@wisteria01 run]$ pjsub s3-a64fx.sh
[INFO] PJM 0000 pjsub Job 335686 submitted.
[z30xxx@wisteria01 run]$ pjstat
Wisteria/BDEC-01 scheduled stop time: 2022/04/22(Fri) 09:00:00 (Remain:  1days 18:55:15)

JOB_ID    JOB_NAME  STATUS  PROJECT   RSCGROUP      START_DATE      ELAPSE      TOKEN      NODE  GPU
335684    SYStest-wa RUNNING jhxxxxxxa  debug-a       04/20 14:04:22<  00:00:24      0.2        1   8
335685    SYStest-wa RUNNING jhxxxxxxo  debug-o       04/20 14:04:23<  00:00:23      0.0        1   -
335686    SYStest-wa RUNNING jhxxxxxxo  debug-o       04/20 14:04:44<  00:00:02      0.0        1   -
[z30xxx@wisteria01 run]$

3) After Code-2 and Code-3 has finished, then kill Code-1 Job
[z30xxx@wisteria01 run]$ pjstat
Wisteria/BDEC-01 scheduled stop time: 2022/04/22(Fri) 09:00:00 (Remain:  1days 18:51:31)

JOB_ID    JOB_NAME  STATUS  PROJECT   RSCGROUP      START_DATE      ELAPSE      TOKEN      NODE  GPU
335684    SYStest-wa RUNNING jhxxxxxxa  debug-a       04/20 14:04:22<  00:04:08      1.7        1   8
[z30xxx@wisteria01 run]$ pjdel 335684
[INFO] PJM 0100 pjdel Accepted job 335684.
[z30xxx@wisteria01 run]$
```

図 12. WaitIO 版 Toy プログラムの実行例: Code-1(Intel)+Code-2(a64fx)+Code-3(a64fx)

```
[z30xxx@wisteria01 waitio]$ ./waitio-serv -s -p 8801 -d
waitio-server host wisteria01:8801 started
wisteria01:-1::waitio_serv_loop:epoll called 4, 0x4 (sfd=4)!
 wisteria01:-1/-1(1722609):waitio-server: waitio_serv_loop accepted from 10.1.6.1 port=7858 fd=5
==== wisteria01:(1722609):waitio_serv_loop:master host has set to wa01-ib0 ====
wisteria01:-1::waitio_serv_loop:epoll called 4, 0x4 (sfd=4)!
 wisteria01:-1/-1(1722609):waitio-server: waitio_serv_loop accepted from 10.11.101.1 port=3210 fd=5
wisteria01:-1::waitio_serv_loop:epoll called 4, 0x4 (sfd=4)!
 wisteria01:-1/-1(1722609):waitio-server: waitio_serv_loop accepted from 10.11.101.9 port=30390 fd=5
wisteria01:-1::waitio_serv_loop:epoll called 4, 0x4 (sfd=4)!
 wisteria01:-1/-1(1722609):waitio-server: waitio_serv_loop accepted from 10.11.101.9 port=30902 fd=5
==== wisteria01:(1722609):waitio_serv_loop:master host  reset! ====
```

図 13. WaitIO 版 Toy プログラムの実行例 waitio-serv 出力: Code-1(Intel)+Code-2(a64fx)+Code-3(a64fx)

```
#   時間          s2.out時間     ●発熱量        ●結果          ●の隣結果       ●結果          ●結果
   1.000000E-01  1.000000E-01  1.745347E-03  5.602511E-02  3.225460E-03  0.000000E+00  0.000000E+00
   2.000000E-01  2.000000E-01  1.663485E-02  5.372070E-01  3.184216E-02  0.000000E+00  0.000000E+00
   3.000000E-01  3.000000E-01  1.838060E-02  6.213567E-01  4.488534E-02  1.749074E-64  1.749074E-64
   4.000000E-01  4.000000E-01  2.012594E-02  6.853573E-01  5.309708E-02  1.931968E-35  1.931968E-35
   5.000000E-01  5.000000E-01  2.187122E-02  7.472359E-01  5.987997E-02  7.930161E-32  7.930161E-32

<Snip><Snip><Snip>

   2.920000E+01  2.920000E+01  4.993690E-01  1.739279E+01  1.646121E+00  9.098841E-05  9.098841E-05
   2.930000E+01  2.930000E+01  5.008806E-01  1.744550E+01  1.651154E+00  9.284215E-05  9.284215E-05
   2.940000E+01  2.940000E+01  5.023908E-01  1.749815E+01  1.656181E+00  9.472301E-05  9.472301E-05
   2.950000E+01  2.950000E+01  5.038994E-01  1.755075E+01  1.661204E+00  9.663123E-05  9.663123E-05
   2.960000E+01  2.960000E+01  5.054064E-01  1.760330E+01  1.666221E+00  9.856701E-05  9.856701E-05
   2.970000E+01  2.970000E+01  5.069119E-01  1.765580E+01  1.671233E+00  1.005306E-04  1.005306E-04
   2.980000E+01  2.980000E+01  5.084159E-01  1.770824E+01  1.676241E+00  1.025222E-04  1.025222E-04
   2.990000E+01  2.990000E+01  5.099183E-01  1.776062E+01  1.681243E+00  1.045420E-04  1.045420E-04
   3.000000E+01  3.000000E+01  5.114191E-01  1.781295E+01  1.686240E+00  1.065902E-04  1.065902E-04
```

図 14. WaitIO 版 Toy プログラムの実行例: s3.out ファイル出力: Code-1(Intel)+Code-2(a64fx)+Code-3(a64fx)

図 14 に最終出力である s3.out ファイルの出力結果の抜粋を示す。本プログラムでは最後の 2 つの結果が一致することで計算が正しく実行されているかを確認することができる。

5. まとめと今後の予定

本稿では Wisteria/BDEC-01 利用事例として h3-Open-BDEC と h3-Open-BDEC の一つのモジュールの一つである h3-Open-SYS/WaitIO の概要について、適用事例を含めたプログラミングと実行例を交えて紹介した。本号の別記事では h3-Open-UTIL/MP の概要、次号では h3-Open-UTIL/MP を活用した GP-GPU 連携を含めた多機能カップラーとして「計算・データ・学習」融合の事例も紹介するので合わせて参考にされたい。

WaitIO-Socket は 6 月より Wisteria でのサービスを開始する予定で、徐々にアプリケーションユーザへの適用事例を増やしていくべくユーザを募集中である。広くユーザに使っていただくため、今年度チュートリアル・講習会の開催を計画している。ご興味のある方は参加いただければ幸いである。

今後は実運用に耐えるべく正常系の処理に加え異常系の処理を拡充すると共にセンター内だけでなくセンター間でのデータ通信にも活用可能なライブラリを拡張していく予定である。

参考文献

[1] Wisteria/BDEC-01（「計算・データ・学習」融合スーパーコンピュータシステム）:https://www.cc.utokyo.ac.jp/supercomputer/wisteria

[2] 中島研吾, 塙敏博, 下川辺隆史, 伊田明弘, 芝隼人, 三木洋平, 星野哲也, 有間 英志, 河合直聡, 坂本龍一, 岩下武史, 八代尚, 長尾大道, 松葉浩也, 荻田武史, 片桐孝洋, 古村孝志, 鶴岡弘, 市村強, 藤田航平, 近藤正章,「計算・データ・学習」融合スーパーコンピュータシステム「Wisteria/BDEC- 01」の概要, 情報処理学会研究報告（2020-HPC-179-01）, 2021

[3] 塙 敏博, 中島 研吾, 下川辺隆史, 芝隼人, 三木洋平, 星野哲也, 河合直聡, 似鳥啓吾, 今村俊幸, 工藤周平, 中尾 昌広,「計算・データ・学習」融合スーパーコンピュータシステム「Wisteria/BDEC- 01」の性能評価, 情報処理学会研究報告（2021-HPC-180-22）, 2021

[4] 住元真司, 荒川隆, 坂口吉生, 松葉浩也, 八代尚, 塙敏博, 中島研吾, WaitIO-Socket：異種システム上の複数 MPI プログラムを結合する通信ライブラリの試作, 情報処理学会研究報告（2021-HPC-181-07）, 2021

[5] Arakawa, T., Yashiro, H., Nakajima, K., Development of a coupler h3-Open-UTIL/MP, ACM Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region (HPC Asia 2022), 2022

[6] ppOpen-HPC: https://github.com/Post-Peta-Crest/ppOpenHPC

[7] h3-Open-BDEC：https://h3-open-bdec.cc.u-tokyo.ac.jp/

[8] Iwashita, T, Nakajima, K., Shimokawabe, T., Nagao, H., Ogita, T., Katagiri, T., Yashiro, H., Matsuba, H., h3-Open-BDEC: Innovative Software Platform for Scientific Computing in the Exascale Era by Integrations of (Simulation + Data + Learning), Project Poster, ISC-HPC 2020

[9] Nakajima, K., Matsuba, K., Hanawa, T., Furumura, T., Tsuruoka, H., Nagao, H., Integration of 3D

Earthquake Simulation & Real-Time Data Assimilation on h3-Open-BDEC, MS290: Progress & Challenges in Extreme Scale Computing & Data SIAM Conference on Computational Science & Engineering (CSE21) (Online, March 4, 2021)

[10] SIAM News (March 10, 2021), Supercomputer Simulations of Earthquakes in Real Time (by Jillian Kunze), https://sinews.siam.org/Details-Page/supercomputer-simulations-of-earthquakes-in-real-time

[11] 中島研吾，古村孝志，鶴岡弘，坂口吉生，松葉浩也，住元真司，八代尚，荒川隆，塙敏博，観測データ同化による長周期地震動リアルタイム予測へ向けた試み，情報処理学会研究報告（2021-HPC-182-08），2021

[12] Nakajima, K., Parallel Iterative Solvers of GeoFEM with Selective Blocking Preconditioning for Nonlinear Contact Problems on the Earth Simulator. ACM/IEEE Proceedings of SC'03, https://doi.org/10.1145/1048935.1050164, 2003