\Orchestrating a brighter world

NEC Deutschland GmbH
Fritz-Vomfelde-Straße 14-16
D-40547 Düsseldorf/Germany

3rd Aurora Deep Dive Workshop

Using NEC-MPI

NEC & RWTH Aachen University

\Orchestrating a brighter world

NEC brings together and integrates technology and expertise to create
the ICT-enabled society of tomorrow.
We collaborate closely with partners and customers around the world,
orchestrating each project to ensure all its parts are fine-tuned to local needs.

Every day, our innovative solutions for society contribute to
greater safety, security, efficiency and equality,
and enable people to live brighter lives.

Aurora Deep Dive Workshop

1. Preliminaries of MPI
   - MPI-Introduction
   - Compilation
2. Executing MPI-Programs
   - MPI on VE
   - MPI on VH
   - MPI on VE/VH Hybrid
3. Profiling & Debugging
   - Ftrace
   - Optional: Profiling Layer
   - Optional: Message Verification

Preliminaries of MPI

MPI-Introduction

Introduction

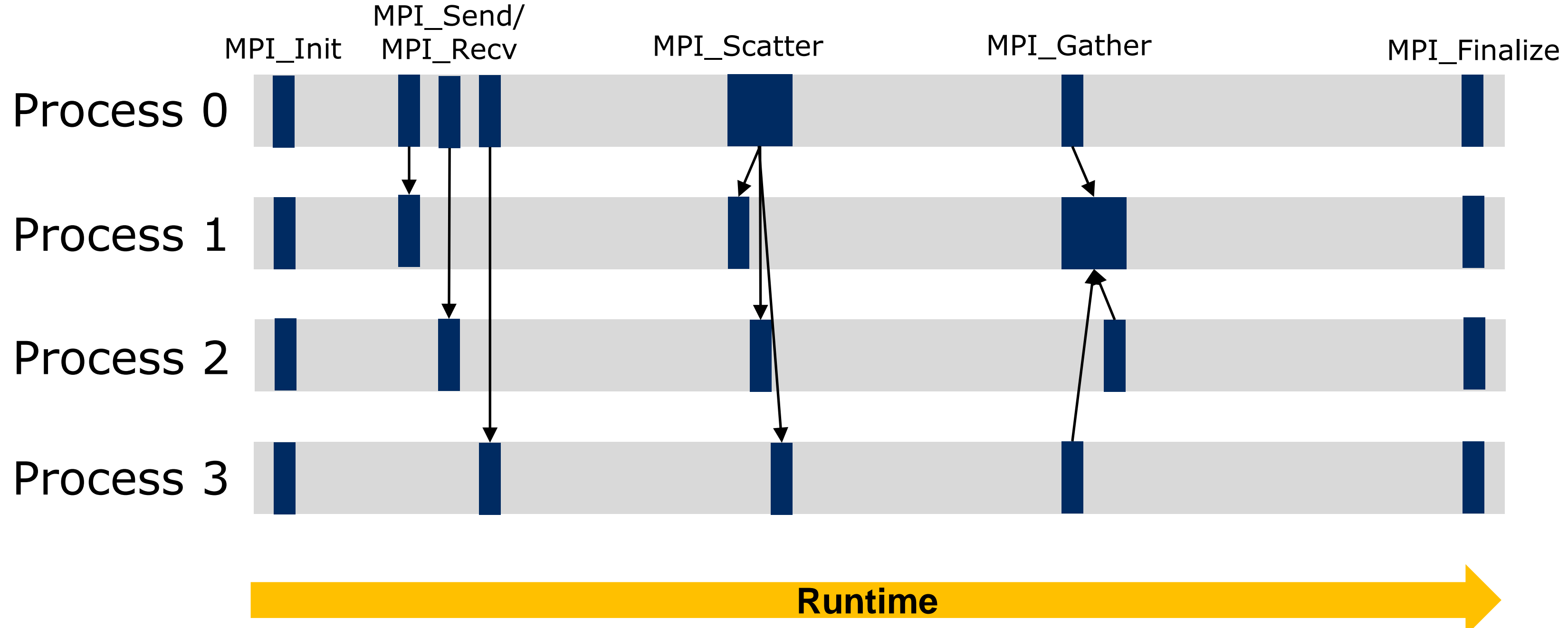
- MPI (Message Passing Interface) is a communication protocol for programming parallel computers.

- Digital (PDF/HTML) versions of the standards are freely available at https://www.mpi-forum.org/
- Famous implementations are: OpenMPI, NEC-MPI, IntelMPI, MPICH, HP-MPI, MS-MPI, …

© NEC Corporation 2019

Introduction

- MPI defines how two processes can exchange data by sending and receiving explicit messages.



- MPI defines Point-to-Point, Collective, Reductions, One-Sided, Blocking, Non-Blocking, and Persistent communications.

© NEC Corporation 2019
\Orchestrating a brighter world NEC

Compilation

MPI Compiler Wrapper

mpincc	C Aurora Compiler
mpinc++	C++ Aurora Compiler
mpinfort	Fortran Aurora Compiler

Syntax:

```
$ <compiler> <flags> <source>
```

Examples:

```
$ mpinfort -o test.x test.F90 #compilation of Fortran
$ mpinfort -c test.F90              #object creation
$ mpinfort -o test.x test1.o test2.o #linking

$ mpincc -o test.x test.c           #compilation of C
$ mpincc -c test.c                  #object creation
$ mpincc -o text.x test1.o test2.o #linking
```


© NEC Corporation 2019
\Orchestrating a brighter world NEC

Executing MPI Programs

\Orchestrating a brighter world NEC

MPI on VE

MPI on VE test codes

- The following code will be used to illustrate how MPI programs run.

```c
#include <mpi.h>
#include <stdio.h>
int main(int argc, char **argv) {
   MPI_Init(&argc, &argv);
   int my_rank, namel;
   char name[MPI_MAX_PROCESSOR_NAME];
   MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
   MPI_Get_processor_name(name, &namel);
   printf("Process %2d is running on %s\n",
          my_rank, name);
   MPI_Finalize();
   return 0;
}
```

```fortran
PROGRAM testmpi
   USE mpi
   IMPLICIT NONE
   INTEGER :: my_rank, namel
   CHARACTER(LEN=MPI_MAX_PROCESSOR_NAME) :: name
   INTEGER :: my_thread_id, num_threads, ierr
   CALL MPI_Init(ierr)
   CALL MPI_Comm_rank(MPI_COMM_WORLD, my_rank, ierr)
   CALL MPI_Get_processor_name(name, namel, ierr)
   WRITE(*,'(A,I2,2A)') "Process ", my_rank, &
      " is running on ", TRIM(name)
   CALL MPI_Finalize(ierr)
END PROGRAM testmpi
```

```
$ mpincc -o mpi_VE mpi.c
```

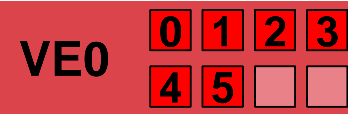
```
$ mpinfort -o mpi_VE mpi.F90
```


© NEC Corporation 2019

MPI on 1 VE

- -c, -n, -np determines the number of processes to create.

```
$ mpirun -np 6 ./mpi_VE
Process  0 is running on node0, ve id 0
Process  1 is running on node0, ve id 0
Process  2 is running on node0, ve id 0
Process  3 is running on node0, ve id 0
Process  4 is running on node0, ve id 0
Process  5 is running on node0, ve id 0
```



© NEC Corporation 2019

MPI on 1 VE

- -c, -n, -np determines the number of processes to create.

```
$ mpirun -np 6 -v ./mpi_VE
mpid: Creating 6 processes of './mpi_VE' on VE 0 of
local host node0
Process  0 is running on node0, ve id 0
Process  1 is running on node0, ve id 0
Process  2 is running on node0, ve id 0
Process  3 is running on node0, ve id 0
Process  4 is running on node0, ve id 0
Process  5 is running on node0, ve id 0
```

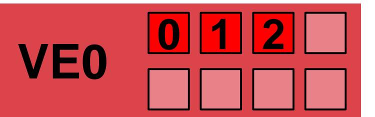
Note that the process placement can be checked for arbitrary programs with the -v flag.

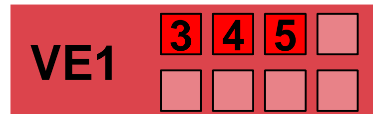
© NEC Corporation 2019

MPI on multiple VEs

- -ve <first_ve>[-<last_ve>] specifies the VEs to use. Processes are distributed equally among all VEs.

```
$ mpirun -ve 0-1 -np 6 ./mpi_VE
Process  0 is running on node0, ve id 0
Process  1 is running on node0, ve id 0
Process  2 is running on node0, ve id 0
Process  3 is running on node0, ve id 1
Process  4 is running on node0, ve id 1
Process  5 is running on node0, ve id 1
```





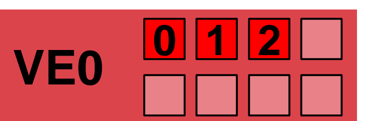
Note that if you reserved the VEs via NQSV with the --venode option mpirun needs the -venode flag. -ve X needs to be replaced by -node X!

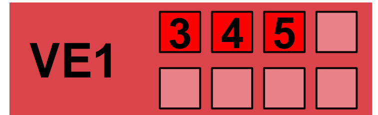
© NEC Corporation 2019

MPI on multiple VEs

- -vennp, -ve_nnp, -nnp_ve determines the number of processes to create per VE.

```
$ mpirun -ve 0-1 -vennp 3 ./mpi_VE
Process  0 is running on node0, ve id 0
Process  1 is running on node0, ve id 0
Process  2 is running on node0, ve id 0
Process  3 is running on node0, ve id 1
Process  4 is running on node0, ve id 1
Process  5 is running on node0, ve id 1
```



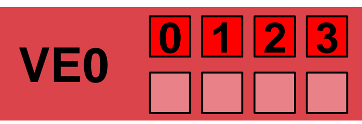


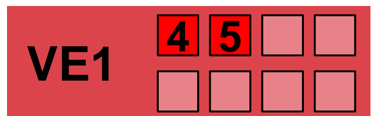
 © NEC Corporation 2019

MPI on multiple VEs

- With ":" multiple sets of options can be specified

```
$ mpirun -ve 0 -vennp 4 ./mpi_VE : \
         -ve 1 -vennp 2 ./mpi_VE
Process  0 is running on node0, ve id 0
Process  1 is running on node0, ve id 0
Process  2 is running on node0, ve id 0
Process  3 is running on node0, ve id 0
Process  4 is running on node0, ve id 1
Process  5 is running on node0, ve id 1
```





© NEC Corporation 2019

MPIsep

MPI Output separation with MPIsep

- If a program produces output on stdout and/or stderr with several processes, deciphering which line belongs to which process can be cumbersome.
- The output order from several processes is undefined in MPI.

```
$ mpirun -np 6                                ./mpi_VE
Process  3 is running on node0, ve id 0
Process  4 is running on node0, ve id 0
Process  1 is running on node0, ve id 0
Process  5 is running on node0, ve id 0
Process  2 is running on node0, ve id 0
Process  0 is running on node0, ve id 0
```


© NEC Corporation 2019

MPI Output separation with MPIsep

- If a program produces output on stdout and/or stderr with several processes, deciphering which line belongs to which process can be cumbersome.
- The output order from several processes is undefined in MPI.
- NEC-MPI provides a script for output separation.

```
$ mpirun -np 6 ${NMPI_ROOT}/bin/mpisep.sh ./mpi_VE
Process  3 is running on node0, ve id 0
Process  4 is running on node0, ve id 0
Process  1 is running on node0, ve id 0
Process  5 is running on node0, ve id 0
Process  2 is running on node0, ve id 0
Process  0 is running on node0, ve id 0
```


© NEC Corporation 2019

MPI Output separation with MPIsep

- If a program produces output on stdout and/or stderr with several processes, deciphering which line belongs to which process can be cumbersome.
- The output order from several processes is undefined in MPI.
- NEC-MPI provides a script for output separation.
- It requires the variable NMPI_SEPSELECT to be exported.
  - 1: Separates stdout in stdout.${mpirank}

```
export NMPI_SEPSELECT=1
$ mpirun -np 6 ${NMPI_ROOT}/bin/mpisep.sh ./mpi_VE
$ ls
stdout.0:0  stdout.0:2  stdout.0:4
stdout.0:1  stdout.0:3  stdout.0:5
$ cat stdout.0\:0
Process  0 is running on node0, ve id 0
```


© NEC Corporation 2019

MPI Output separation with MPIsep

- If a program produces output on stdout and/or stderr with several processes, deciphering which line belongs to which process can be cumbersome.
- The output order from several processes is undefined in MPI.
- NEC-MPI provides a script for output separation.
- It requires the variable NMPI_SEPSELECT to be exported.
  - 1: only stdout in stdout.${mpirank}
  - 2: only stderr in stderr.${mpirank}

```
export NMPI_SEPSELECT=2
$ mpirun -np 6 ${NMPI_ROOT}/bin/mpisep.sh ./mpi_VE
Process  3 is running on node0, ve id 0
Process  4 is running on node0, ve id 0
Process  5 is running on node0, ve id 0
Process  1 is running on node0, ve id 0
Process  2 is running on node0, ve id 0
Process  0 is running on node0, ve id 0
$ ls
stderr.0:0  stderr.0:2  stderr.0:4
stderr.0:1  stderr.0:3  stderr.0:5
```


© NEC Corporation 2019

MPI Output separation with MPIsep

- If a program produces output on stdout and/or stderr with several processes, deciphering which line belongs to which process can be cumbersome.
- The output order from several processes is undefined in MPI.
- NEC-MPI provides a script for output separation.
- It requires the variable NMPI_SEPSELECT to be exported.
  - 1: only stdout in stdout.${mpirank}
  - 2: only stderr in stderr.${mpirank}
  - 3: stdout in stdout.${mpirank}, stderr in stderr.${mpirank}

```
export NMPI_SEPSELECT=3
$ mpirun -np 6 ${NMPI_ROOT}/bin/mpisep.sh ./mpi_VE
$ ls
stderr.0:0  stderr.0:3  stdout.0:0  stdout.0:3
stderr.0:1  stderr.0:4  stdout.0:1  stdout.0:4
stderr.0:2  stderr.0:5  stdout.0:2  stdout.0:5
```


© NEC Corporation 2019

MPI Output separation with MPIsep

- If a program produces output on stdout and/or stderr with several processes, deciphering which line belongs to which process can be cumbersome.
- The output order from several processes is undefined in MPI.
- NEC-MPI provides a script for output separation.
- It requires the variable NMPI_SEPSELECT to be exported.
  - 1: only stdout in stdout.${mpirank}
  - 2: only stderr in stderr.${mpirank}
  - 3: stdout in stdout.${mpirank}, stderr in stderr.${mpirank}
  - 4: stdout and stderr in std.${mpirank}

```
export NMPI_SEPSELECT=4
$ mpirun -np 6 ${NMPI_ROOT}/bin/mpisep.sh ./mpi_VE
$ ls
std.0:0  std.0:2  std.0:4
std.0:1  std.0:3  std.0:5
```


© NEC Corporation 2019

MPI on VH

MPI on VH test codes

- The following code will be used to illustrate how MPI programs run. They are identical to the VE version!

```c
#include <mpi.h>
#include <stdio.h>
int main(int argc, char **argv) {
   MPI_Init(&argc, &argv);
   int my_rank, namel;
   char name[MPI_MAX_PROCESSOR_NAME];
   MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
   MPI_Get_processor_name(name, &namel);
   printf("Process %2d is running on %s\n",
          my_rank, name);
   MPI_Finalize();
   return 0;
}
```

```fortran
PROGRAM testmpi
   USE mpi
   IMPLICIT NONE
   INTEGER :: my_rank, namel
   CHARACTER(LEN=MPI_MAX_PROCESSOR_NAME) :: name
   INTEGER :: my_thread_id, num_threads, ierr
   CALL MPI_Init(ierr)
   CALL MPI_Comm_rank(MPI_COMM_WORLD, my_rank, ierr)
   CALL MPI_Get_processor_name(name, namel, ierr)
   WRITE(*,'(A,I2,2A)') "Process ", my_rank, &
      " is running on ", TRIM(name)
   CALL MPI_Finalize(ierr)
END PROGRAM testmpi
```

```
$ export NMPI_CC_VH=gcc
$ mpincc -vh -o mpi_VH mpi.c
```

```
$ export NMPI_F90_VH=gfortran
$ mpinfort -vh -o mpi_VH mpi.F90
```

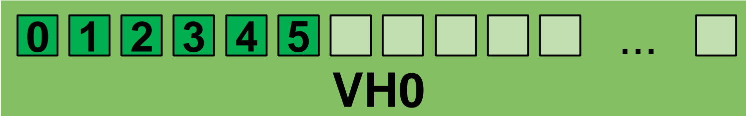
- The -vh flag compiles the program for the VH. It uses the compilers specified by the variables NMPI_F90_VH, and NMPI_CC_VH.

© NEC Corporation 2019

MPI on VH

- The -vh flag executes the program on the VH. It needs to be compiled for the target architecture.

```
$ mpirun -vh -np 6 ./mpi_VH
Process  0 is running on node0, VH
Process  1 is running on node0, VH
Process  2 is running on node0, VH
Process  3 is running on node0, VH
Process  4 is running on node0, VH
Process  5 is running on node0, VH
```



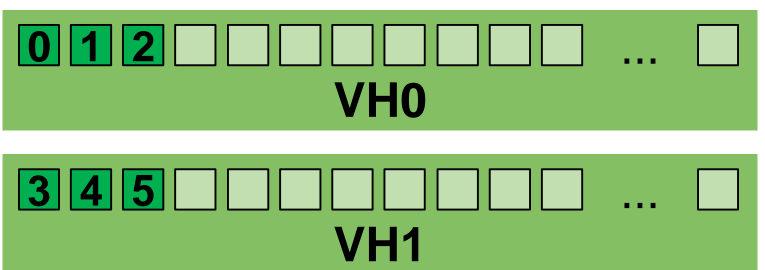
Note that VH-processes can be pinned using the -pin_mode, -pin_cpu, -cpu_list. Check mpirun –h for pinning examples

© NEC Corporation 2019

MPI on VH

- -nnp, -npernode, -perhost, -ppn, -N determines the number of processes per node.

```
$ mpirun -vh -nnp 3 ./mpi_VH
Process  0 is running on node0, VH
Process  1 is running on node0, VH
Process  2 is running on node0, VH
Process  3 is running on node1, VH
Process  4 is running on node1, VH
Process  5 is running on node1, VH
```



 © NEC Corporation 2019

\Orchestrating a brighter world

MPI on VE/VH Hybrid

MPI on VE/VH test codes

- The following code will be used to illustrate how MPI programs run.

```
#include <mpi.h>
#include <stdio.h>
int main(int argc, char **argv) {
   MPI_Init(&argc, &argv);
   int my_rank, namel;
   char name[MPI_MAX_PROCESSOR_NAME];
   MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
   MPI_Get_processor_name(name, &namel);
   printf("Process %2d is running on %s\n",
          my_rank, name);
   MPI_Finalize();
   return 0;
}
```

```
$ mpincc     -o mpi_VE mpi.c
$ mpincc -vh -o mpi_VH mpi.c
```

```
PROGRAM testmpi
   USE mpi
   IMPLICIT NONE
   INTEGER :: my_rank, namel
   CHARACTER(LEN=MPI_MAX_PROCESSOR_NAME) :: name
   INTEGER :: my_thread_id, num_threads, ierr
   CALL MPI_Init(ierr)
   CALL MPI_Comm_rank(MPI_COMM_WORLD, my_rank, ierr)
   CALL MPI_Get_processor_name(name, namel, ierr)
   WRITE(*,'(A,I2,2A)') "Process ", my_rank, &
      " is running on ", TRIM(name)
   CALL MPI_Finalize(ierr)
END PROGRAM testmpi
```

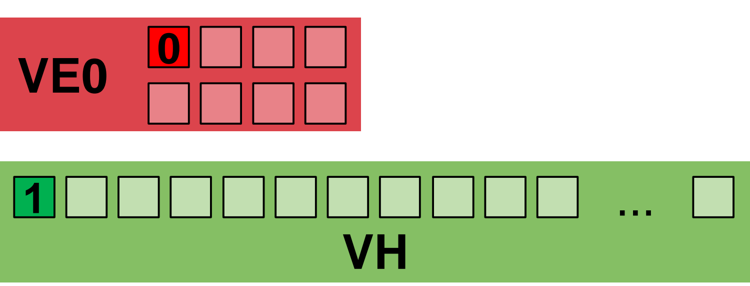
```
$ mpinfort     -o mpi_VE mpi.F90
$ mpinfort -vh -o mpi_VH mpi.F90
```


© NEC Corporation 2019

MPI on VE/VH

- VE/VH Hybrid execution is easily achieved by chaining VE and VH commands with ":" with the corresponding executables.

```
mpirun     -np 1 ./mpi_VE : \
       -vh -np 1 ./mpi_VH
Process  0 is running on node0, ve id 0
Process  1 is running on node0, VH
```

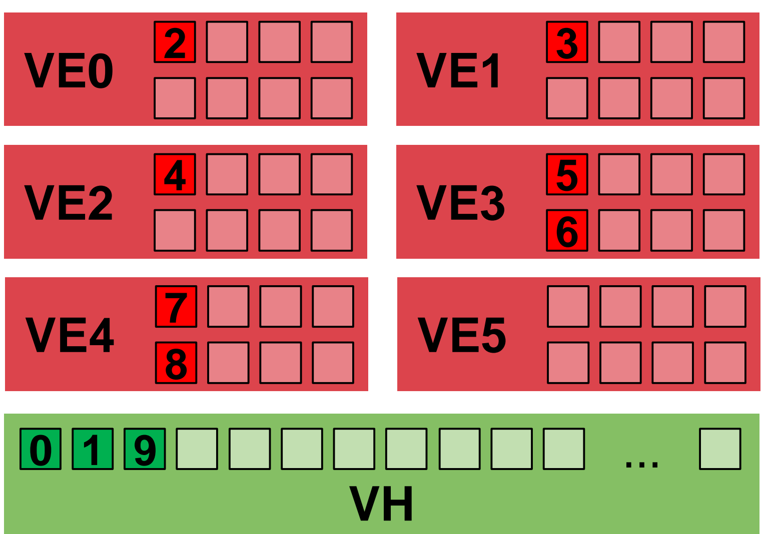


© NEC Corporation 2019
\Orchestrating a brighter world NEC

MPI on VE/VH

- VE/VH Hybrid execution is easily achieved by chaining VE and VH commands with ":" with the corresponding executables.

```
mpirun -vh         -np 2    ./mpi_VH : \
           -ve 0-2 -vennp 1 ./mpi_VE : \
           -ve 3-4 -vennp 2 ./mpi_VE : \
       -vh         -np 1    ./mpi_VH
Process  0 is running on node0, VH
Process  1 is running on node0, VH
Process  2 is running on node0, ve id 0
Process  3 is running on node0, ve id 1
Process  4 is running on node0, ve id 2
Process  5 is running on node0, ve id 3
Process  6 is running on node0, ve id 3
Process  7 is running on node0, ve id 4
Process  8 is running on node0, ve id 4
Process  9 is running on node0, VH
```



© NEC Corporation 2019

IO-Process on VHs

- Every IO Operation on a VE requires a system call that halts the VE.

```
PROGRAM write
   USE mpi
   IMPLICIT NONE
   INTEGER, PARAMETER :: n = 4096
   INTEGER, PARAMETER :: m = 16384
   INTEGER, DIMENSION(n,m) :: arr
   INTEGER :: myrank, ierr, i, j
   CALL MPI_Init(ierr)
   CALL MPI_Comm_rank(MPI_COMM_WORLD, &
                      myrank, ierr)
   IF (myrank == 0) THEN
      OPEN(42, FILE="myfile",
           STATUS="unknown")
      DO i = 1, m
         WRITE(42,*) arr(:,i)
      END DO
      CLOSE(42)
   END IF
   CALL MPI_Finalize(ierr)
END PROGRAM write
```

```
PROGRAM read
   USE mpi
   IMPLICIT NONE
   INTEGER, PARAMETER :: n = 4096
   INTEGER, PARAMETER :: m = 16384
   INTEGER, DIMENSION(n,m) :: arr
   INTEGER :: myrank, ierr, i
   CALL MPI_Init(ierr)
   CALL MPI_Comm_rank(MPI_COMM_WORLD, &
                      myrank, ierr)
   IF (myrank == 0) THEN
      OPEN(42, FILE="myfile")
      DO i = 1, m
         READ(42,*) arr(:,i)
      END DO
      CLOSE(42)
   END IF
   CALL MPI_Bcast(arr, n*m, &
                  MPI_INTEGER, 0, &
                  MPI_COMM_WORLD,ierr)
   CALL MPI_Finalize(ierr)
END PROGRAM read
```

```
$ mpinfort     -o write_VE write.F90
$ mpinfort -vh -o write_VH write.F90
```

```
$ mpinfort     -o read_VE read.F90
$ mpinfort -vh -o read_VH read.F90
```


© NEC Corporation 2019

IO-Processes on VHs

```
$ time mpirun -np 8 ./write_VE.x
real        0m8.822s
user        0m0.296s
sys         0m0.672s
```

```
$ time mpirun -np 8 ./read_VE.x
real        0m46.340s
user        0m0.412s
sys         0m0.791s
```

```
$ time mpirun -vh -np 1 ./write_VH.x :\
                  -np 8 ./write_VE.x
real        0m7.084s
user        0m4.158s
sys         0m1.366s
```

```
$ time mpirun -vh -np 1 ./read_VH.x :\
                  -np 8 ./read_VE.x
real        0m17.727s
user        0m16.004s
sys         0m1.121s
```

- Using an VH-IO process can significantly decrease runtime.
- Moving existing IO-processes to VH frees VE-cores for computations.
- The additional VH-VE MPI-communication time is negligible.
- Note that the IO-performance also strongly depends on the utilized Filesystem.

© NEC Corporation 2019

Profiling & Debugging

\Orchestrating a brighter world

Parallel Ftrace

FTRACE Profile – Test Program

- A stream and dgemm routine, communicating results to an IO-process. Ftrace regions for better analysis (needed later).

```
#include <stdlib.h>
#include <stdio.h>
#include <cblas.h>
#include <mpi.h>
#include <ftrace.h>
void stream(int n, int iter) {
   double *a = (double*) malloc(n*sizeof(double));
   double *b = (double*) malloc(n*sizeof(double));
   double *c = (double*) malloc(n*sizeof(double));
   for (int i=0; i<n; i++) {
      a[i] = 1.1*i+1;
      b[i] = 0.9*i+1;
      c[i] = 0.0;
   }
   for (int i=0; i<iter; i++) {
      ftrace_region_begin("stream-work");
      for (int j=0; j<n; j++) {
         c[j] = a[j]+2.1*b[j];
      }
      ftrace_region_end("stream-work");
      ftrace_region_begin("stream-comm");
      MPI_Send(c, n, MPI_DOUBLE, 0, 0,
               MPI_COMM_WORLD);
      ftrace_region_end("stream-comm");
   }
   free(a); free(b); free(c);
}
```

```
void matrix(int n, int iter) {
   int nsq = n*n;
   double *a = (double*) malloc(nsq*sizeof(double));
   double *b = (double*) malloc(nsq*sizeof(double));
   double *c = (double*) malloc(nsq*sizeof(double));
   // fill the matrices
   for (int i=0; i<nsq; i++) {
      a[i] = 0.42*(i%(n+1));
      b[i] = 0.137*(i%(n+1));
      c[i] = 0.0;
   }
   //run the iterations
   for (int i=0; i<iter; i++) {
      cblas_dgemm(CblasRowMajor, CblasNoTrans,
         CblasNoTrans, n, n, n, 1.0, a, n, b,
         n, 0.0, c, n);
      MPI_Send(c, nsq, MPI_DOUBLE, 0, 0,
               MPI_COMM_WORLD);
      }
   free(a); free(b); free(c);
}
```


© NEC Corporation 2019

FTRACE Profile – Test Program

- IO-process routines to receive and print results.

```
void IO_stream(int comm_size, int ssize) {
   double *recvbuf = (double*)
                     malloc(ssize*sizeof(double));
   int nstream_members = comm_size/2;
   for (int imem=1; imem<=nstream_members; imem++) {
      int stream_member = 2*imem-1;
      MPI_Status status;
      MPI_Recv(recvbuf, ssize, MPI_DOUBLE,
               stream_member, 0,
               MPI_COMM_WORLD, &status);
      printf("stream %lf\n", recvbuf[0]);
   }
   free(recvbuf);
}
```

```
void IO_matrix(int comm_size, int msize) {
   double *recvbuf = (double*)
                     malloc(msize*sizeof(double));
   int nmatrix_members = (comm_size-1)/2;
   for (int imem=1; imem<=nmatrix_members; imem++) {
      int matrix_member = 2*imem;
      MPI_Status status;
      MPI_Recv(recvbuf, msize, MPI_DOUBLE,
               matrix_member, 0,
               MPI_COMM_WORLD, &status);
      printf("matrix: %lf\n", recvbuf[0]);
   }
   free(recvbuf);
}
```


© NEC Corporation 2019

FTRACE Profile – Test Program

- Main routine to define the IO-process, stream- and matrix-processes.

```
int main(int argc, char **argv) {
   MPI_Init(&argc, &argv);
   int iter = 10;
   int ssize = 33554432;
   int msize = 4096;
   int my_rank, comm_size;
   MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
   MPI_Comm_size(MPI_COMM_WORLD, &comm_size);
   if (my_rank ==0) {
      for (int i=0; i<iter; i++) {
         IO_stream(comm_size, ssize);
         IO_matrix(comm_size, msize*msize);
      }
   } else if (my_rank%2==0) {
      matrix(msize, iter);
   } else {
      stream(ssize, iter);
   }
   MPI_Finalize();
   return 0;
}
```

- The -ftrace flag enables tracing of individual routines and MPI-performances.

```
mpincc -ftrace -I/opt/nec/ve/nlc/2.3.0/include \
   -o operation_performance.x \
   test.x \
   -L/opt/nec/ve/nlc/2.3.0/lib/ \
   -lcblas -lblas_sequential
```


© NEC Corporation 2019

FTRACE Profile – Collective Routine Performance

```
$ mpirun -np 5 ./test.x
$ ftrace -f ftrace.out.0.*
…
FREQUENCY  EXCLUSIVE       AVER.TIME     MOPS   MFLOPS  V.OP  AVER.    VECTOR L1CACHE CPU  PORT VLD LLC PROC.NAME
           TIME[sec](  % )    [msec]                    RATIO V.LEN      TIME    MISS      CONF HIT E.%

        2    11.206( 39.8)  5603.101 253858.5 245266.9  99.68 254.2    11.206   0.000     0.425    99.05 matrix
        2    10.101( 35.9)  5050.319    527.6    159.5  93.12  24.0    10.087   0.008     0.000    23.00 stream
       10     5.533( 19.7)   553.295    239.8      0.0  85.55  11.9     5.524   0.006     0.000    40.89 IO_matrix
        5     1.244(  4.4)   248.735    513.2      0.0   2.79   5.0     0.213   0.025     0.000   100.00 main
       10     0.060(  0.2)     6.019  22499.9      0.0  99.19 247.1     0.059   0.001     0.000     0.07 IO_stream
------------------------------------------------------------------------------------------------------------------
       29    28.144(100.0)   970.471 101388.3  97717.3  99.64 243.6    27.089   0.041     0.425    96.45 total

       20    10.022( 35.6)   501.106    112.1      0.0  71.35   5.0    10.008   0.008     0.000    99.99 stream-comm
       20     0.071(  0.3)     3.563  47645.0  18837.2  98.84 256.0     0.071   0.000     0.000     0.00 stream-work
------------------------------------------------------------------------------------------------------------------
…
```

- All values are either averaged or summed up over all processes.
- For an explanation on the individual columns please refer to the NEC compiler training.

© NEC Corporation 2019
\Orchestrating a brighter world NEC

FTRACE Profile – Collective MPI Performance

```
…
  ELAPSED      COMM.TIME  COMM.TIME    IDLE TIME  IDLE TIME  AVER.LEN      COUNT  TOTAL LEN PROC.NAME
  TIME[sec]        [sec]  / ELAPSED        [sec]  / ELAPSED    [byte]                [byte]

      5.608        0.018                   0.016              128.0M         20       2.5G matrix
      5.541        5.535                   5.506              128.0M         20       2.5G IO_matrix
      5.056        5.015                   5.009              256.0M         20       5.0G stream
      0.891        0.007                   0.312                0.0           0       0.0  main
      0.067        0.061                   0.012              256.0M         20       5.0G IO_stream
---------------------------------------------------------------------------------------------------

      5.015        5.015                   5.009              256.0M         20       5.0G stream-comm
      0.036        0.000                   0.000                0.0           0       0.0  stream-work
---------------------------------------------------------------------------------------------------
…
```

- Elapsed time: Maximum time any process spend in the routine.

© NEC Corporation 2019
\Orchestrating a brighter world NEC

FTRACE Profile – Collective MPI Performance

```
…
ELAPSED      COMM.TIME  COMM.TIME   IDLE TIME  IDLE TIME  AVER.LEN      COUNT  TOTAL LEN PROC.NAME
TIME[sec]        [sec]  / ELAPSED       [sec]  / ELAPSED    [byte]                [byte]

    5.608        0.018                  0.016              128.0M         20       2.5G matrix
    5.541        5.535                  5.506              128.0M         20       2.5G IO_matrix
    5.056        5.015                  5.009              256.0M         20       5.0G stream
    0.891        0.007                  0.312                 0.0          0       0.0  main
    0.067        0.061                  0.012              256.0M         20       5.0G IO_stream
----------------------------------------------------------------------------------------------------

    5.015        5.015                  5.009              256.0M         20       5.0G stream-comm
    0.036        0.000                  0.000                 0.0          0       0.0  stream-work
----------------------------------------------------------------------------------------------------
…
```

- Elapsed time: Maximum time any process spend in the routine.
- Comm. Time: Maximum time any process spends in communication.

© NEC Corporation 2019

FTRACE Profile – Collective MPI Performance

```
…
 ELAPSED      COMM.TIME  COMM.TIME   IDLE TIME  IDLE TIME   AVER.LEN       COUNT  TOTAL LEN PROC.NAME
 TIME[sec]        [sec]  / ELAPSED       [sec]  / ELAPSED     [byte]                 [byte]

     5.608        0.018                  0.016                128.0M          20       2.5G matrix
     5.541        5.535                  5.506                128.0M          20       2.5G IO_matrix
     5.056        5.015                  5.009                256.0M          20       5.0G stream
     0.891        0.007                  0.312                  0.0            0       0.0  main
     0.067        0.061                  0.012                256.0M          20       5.0G IO_stream
---------------------------------------------------------------------------------------------------


     5.015        5.015                  5.009                256.0M          20       5.0G stream-comm
     0.036        0.000                  0.000                  0.0            0       0.0  stream-work
---------------------------------------------------------------------------------------------------
…
```

- Elapsed time: Maximum time any process spend in the routine.
- Comm. Time: Maximum time any process spends in communication.
- Idle Time: Time spend waiting for other processes to send/receive.

© NEC Corporation 2019

FTRACE Profile – Collective MPI Performance

```
…
 ELAPSED      COMM.TIME  COMM.TIME   IDLE TIME  IDLE TIME  AVER.LEN      COUNT  TOTAL LEN PROC.NAME
 TIME[sec]        [sec]  / ELAPSED       [sec]  / ELAPSED    [byte]                [byte]

     5.608        0.018                  0.016               128.0M         20       2.5G matrix
     5.541        5.535                  5.506               128.0M         20       2.5G IO_matrix
     5.056        5.015                  5.009               256.0M         20       5.0G stream
     0.891        0.007                  0.312                  0.0          0       0.0  main
     0.067        0.061                  0.012               256.0M         20       5.0G IO_stream
-------------------------------------------------------------------------------------------------------


     5.015        5.015                  5.009               256.0M         20       5.0G stream-comm
     0.036        0.000                  0.000                  0.0          0       0.0  stream-work
-------------------------------------------------------------------------------------------------------
…
```

- Elapsed time: Maximum time any process spend in the routine.
- Comm. Time: Maximum time any process spends in communication.
- Idle Time: Time spend waiting for other processes to send/receive.
- Len & Count: Amount of send data.

© NEC Corporation 2019

FTRACE Profile – Individual Routine Performance

```
…
FREQUENCY  EXCLUSIVE       AVER.TIME      MOPS    MFLOPS  V.OP  AVER.     VECTOR L1CACHE CPU PORT VLD LLC PROC.NAME
           TIME[sec](  % )   [msec]                       RATIO V.LEN       TIME    MISS     CONF HIT E.%

        2    11.206( 39.8)  5603.101 253858.5 245266.9  99.68 254.2     11.206   0.000     0.425    99.05 matrix
        1     5.602         5601.803 253917.3 245323.7  99.68 254.2      5.602   0.000     0.212    99.04  0.2
        1     5.604         5604.398 253799.8 245210.1  99.68 254.2      5.604   0.000     0.212    99.06  0.4
        2    10.101( 35.9)  5050.319    527.6    159.5  93.12  24.0     10.087   0.008     0.000    23.00 stream
        1     5.049         5048.957    519.0    159.5  93.49  25.5      5.042   0.004     0.000    21.57  0.1
        1     5.052         5051.682    536.3    159.4  92.76  22.7      5.044   0.004     0.000    24.38  0.3
       10     5.533( 19.7)   553.295    239.8      0.0  85.55  11.9      5.524   0.006     0.000    40.89 IO_matrix
       10     5.533          553.295    239.8      0.0  85.55  11.9      5.524   0.006     0.000    40.89  0.0
        5     1.244(  4.4)   248.735    513.2      0.0   2.79   5.0      0.213   0.025     0.000   100.00 main
        1     0.027           26.756   2564.8      0.0   0.01   8.5      0.000   0.001     0.000    99.84  0.0
        1     0.581          581.207    356.5      0.0   4.03   5.0      0.108   0.010     0.000   100.00  0.1
        1     0.030           29.687   2338.5      0.0   0.08   5.3      0.001   0.001     0.000    99.98  0.2
        1     0.579          579.175    387.4      0.0   4.19   5.0      0.104   0.011     0.000   100.00  0.3
        1     0.027           26.851   2557.3      0.0   0.02   7.2      0.000   0.001     0.000   100.00  0.4
       10     0.060(  0.2)     6.019  22499.9      0.0  99.19 247.1      0.059   0.001     0.000     0.07 IO_stream
       10     0.060            6.019  22499.9      0.0  99.19 247.1      0.059   0.001     0.000     0.07  0.0
----------------------------------------------------------------------------------------------------------------------
       29    28.144(100.0)   970.471 101388.3  97717.3  99.64 243.6     27.089   0.041     0.425    96.45 total

       20    10.022( 35.6)   501.106    112.1      0.0  71.35   5.0     10.008   0.008     0.000    99.99 stream-comm
       10     5.010          501.022    103.3      0.0  71.35   5.0      5.004   0.004     0.000    99.99  0.1
       10     5.012          501.190    121.0      0.0  71.35   5.0      5.005   0.004     0.000    99.99  0.3
       20     0.071(  0.3)     3.563  47645.0  18837.2  98.84 256.0      0.071   0.000     0.000     0.00 stream-work
       10     0.035            3.511  48351.1  19116.4  98.84 256.0      0.035   0.000     0.000     0.00  0.1
       10     0.036            3.615  46959.2  18566.1  98.84 256.0      0.036   0.000     0.000     0.00  0.3
----------------------------------------------------------------------------------------------------------------------
…
```

- Routine performance for individual processes.

© NEC Corporation 2019

FTRACE Profile – Individual MPI Performance

```
…
   ELAPSED      COMM.TIME   COMM.TIME    IDLE TIME   IDLE TIME   AVER.LEN       COUNT  TOTAL LEN PROC.NAME
   TIME[sec]        [sec]   / ELAPSED        [sec]   / ELAPSED     [byte]                 [byte]

       5.608        0.018                    0.016                 128.0M          20       2.5G matrix
       5.606        0.015       0.003        0.013       0.002     128.0M          10       1.2G  0.2
       5.608        0.018       0.003        0.016       0.003     128.0M          10       1.2G  0.4
       5.541        5.535                    5.506                 128.0M          20       2.5G IO_matrix
       5.541        5.535       0.999        5.506       0.994     128.0M          20       2.5G  0.0
       5.056        5.015                    5.009                 256.0M          20       5.0G stream
       5.054        5.013       0.992        5.008       0.991     256.0M          10       2.5G  0.1
       5.056        5.015       0.992        5.009       0.991     256.0M          10       2.5G  0.3
       0.891        0.007                    0.312                    0.0           0       0.0  main
       0.295        0.007       0.025        0.000       0.000        0.0           0       0.0   0.0
       0.849        0.007       0.008        0.312       0.367        0.0           0       0.0   0.1
       0.297        0.007       0.024        0.002       0.006        0.0           0       0.0   0.2
       0.891        0.007       0.008        0.306       0.344        0.0           0       0.0   0.3
       0.294        0.007       0.024        0.000       0.001        0.0           0       0.0   0.4
       0.067        0.061                    0.012                 256.0M          20       5.0G IO_stream
       0.067        0.061       0.911        0.012       0.176     256.0M          20       5.0G  0.0
------------------------------------------------------------------------------------------------------------

       5.015        5.015                    5.009                 256.0M          20       5.0G stream-comm
       5.013        5.013       1.000        5.008       0.999     256.0M          10       2.5G  0.1
       5.015        5.015       1.000        5.009       0.999     256.0M          10       2.5G  0.3
       0.036        0.000                    0.000                    0.0           0       0.0  stream-work
       0.035        0.000       0.000        0.000       0.000        0.0           0       0.0   0.1
       0.036        0.000       0.000        0.000       0.000        0.0           0       0.0   0.3
------------------------------------------------------------------------------------------------------------
…
```

- MPI performance for individual processes.

© NEC Corporation 2019
\Orchestrating a brighter world

\Orchestrating a brighter world

Parallel Ftrace – Example Analysis

FTRACE Profile – Example Analysis

```
…
FREQUENCY  EXCLUSIVE       AVER.TIME     MOPS   MFLOPS  V.OP  AVER.    VECTOR L1CACHE CPU PORT VLD LLC PROC.NAME
           TIME[sec](  % )    [msec]                    RATIO V.LEN      TIME    MISS     CONF HIT E.%

        2    11.206( 39.8)  5603.101 253858.5 245266.9  99.68 254.2    11.206   0.000    0.425   99.05 matrix
        2    10.101( 35.9)  5050.319    527.6    159.5  93.12  24.0    10.087   0.008    0.000   23.00 stream
       10     5.533( 19.7)   553.295    239.8      0.0  85.55  11.9     5.524   0.006    0.000   40.89 IO_matrix
        5     1.244(  4.4)   248.735    513.2      0.0   2.79   5.0     0.213   0.025    0.000  100.00 main
       10     0.060(  0.2)     6.019  22499.9      0.0  99.19 247.1     0.059   0.001    0.000    0.07 IO_stream
------------------------------------------------------------------------------------------------------------------
       29    28.144(100.0)   970.471 101388.3  97717.3  99.64 243.6    27.089   0.041    0.425   96.45 total

       20    10.022( 35.6)   501.106    112.1      0.0  71.35   5.0    10.008   0.008    0.000   99.99 stream-comm
       20     0.071(  0.3)     3.563  47645.0  18837.2  98.84 256.0     0.071   0.000    0.000    0.00 stream-work
------------------------------------------------------------------------------------------------------------------
…
```

- The matrix routine is the most time consuming routine in our program.
- It shows close to optimal values for vector length, vector operation ratio, and ratio of exclusive to vector time.

© NEC Corporation 2019

FTRACE Profile – Example Analysis

```
…
FREQUENCY  EXCLUSIVE        AVER.TIME      MOPS    MFLOPS  V.OP  AVER.    VECTOR L1CACHE CPU  PORT VLD LLC PROC.NAME
           TIME[sec](  % )    [msec]                       RATIO V.LEN      TIME    MISS      CONF HIT E.%

        2     11.206( 39.8)  5603.101 253858.5 245266.9  99.68 254.2    11.206   0.000     0.425    99.05 matrix
        2     10.101( 35.9)  5050.319    527.6    159.5  93.12  24.0    10.087   0.008     0.000    23.00 stream
       10      5.533( 19.7)   553.295    239.8      0.0  85.55  11.9     5.524   0.006     0.000    40.89 IO_matrix
        5      1.244(  4.4)   248.735    513.2      0.0   2.79   5.0     0.213   0.025     0.000   100.00 main
       10      0.060(  0.2)     6.019  22499.9      0.0  99.19 247.1     0.059   0.001     0.000     0.07 IO_stream
---------------------------------------------------------------------------------------------------------------------
       29     28.144(100.0)   970.471 101388.3  97717.3  99.64 243.6    27.089   0.041     0.425    96.45 total

       20     10.022( 35.6)   501.106    112.1      0.0  71.35   5.0    10.008   0.008     0.000    99.99 stream-comm
       20      0.071(  0.3)     3.563  47645.0  18837.2  98.84 256.0     0.071   0.000     0.000     0.00 stream-work
---------------------------------------------------------------------------------------------------------------------
…
```

- The stream routine is the second most time consuming routine in our program.
- It shows poor performance in vector length, vector operation ratio, MFLOPS.

© NEC Corporation 2019
\Orchestrating a brighter world NEC

FTRACE Profile – Example Analysis

```
…
FREQUENCY  EXCLUSIVE         AVER.TIME      MOPS    MFLOPS  V.OP  AVER.     VECTOR L1CACHE CPU PORT VLD LLC PROC.NAME
           TIME[sec](  % )    [msec]                        RATIO V.LEN       TIME    MISS      CONF HIT E.%

        2    11.206( 39.8)  5603.101 253858.5 245266.9  99.68 254.2    11.206   0.000     0.425    99.05 matrix
        2    10.101( 35.9)  5050.319    527.6    159.5  93.12  24.0    10.087   0.008     0.000    23.00 stream
       10     5.533( 19.7)   553.295    239.8      0.0  85.55  11.9     5.524   0.006     0.000    40.89 IO_matrix
        5     1.244(  4.4)   248.735    513.2      0.0   2.79   5.0     0.213   0.025     0.000   100.00 main
       10     0.060(  0.2)     6.019  22499.9      0.0  99.19 247.1     0.059   0.001     0.000     0.07 IO_stream
------------------------------------------------------------------------------------------------------------------
       29    28.144(100.0)   970.471 101388.3  97717.3  99.64 243.6    27.089   0.041     0.425    96.45 total

       20    10.022( 35.6)   501.106    112.1      0.0  71.35   5.0    10.008   0.008     0.000    99.99 stream-comm
       20     0.071(  0.3)     3.563  47645.0  18837.2  98.84 256.0     0.071   0.000     0.000     0.00 stream-work
------------------------------------------------------------------------------------------------------------------
…
```

- The stream routine is the second most time consuming routine in our program.
- It shows poor performance in vector length, vector operation ratio, MFLOPS.
- Custom ftrace regions reveal that the actual work in the stream routine performs well.

© NEC Corporation 2019

FTRACE Profile – Example Analysis

…

FREQUENCY	EXCLUSIVE TIME[sec](%)	AVER.TIME [msec]	MOPS	MFLOPS	V.OP RATIO	AVER. V.LEN	VECTOR TIME	L1CACHE MISS	CPU PORT CONF	VLD LLC HIT E.%	PROC.NAME
2	11.206(39.8)	5603.101	253858.5	245266.9	99.68	254.2	11.206	0.000	0.425	99.05	matrix
2	10.101(35.9)	5050.319	527.6	159.5	93.12	24.0	10.087	0.008	0.000	23.00	stream
10	5.533(19.7)	553.295	239.8	0.0	85.55	11.9	5.524	0.006	0.000	40.89	IO_matrix
5	1.244(4.4)	248.735	513.2	0.0	2.79	5.0	0.213	0.025	0.000	100.00	main
10	0.060(0.2)	6.019	22499.9	0.0	99.19	247.1	0.059	0.001	0.000	0.07	IO_stream
29	28.144(100.0)	970.471	101388.3	97717.3	99.64	243.6	27.089	0.041	0.425	96.45	total
20	10.022(35.6)	501.106	112.1	0.0	71.35	5.0	10.008	0.008	0.000	99.99	stream-comm
20	0.071(0.3)	3.563	47645.0	18837.2	98.84	256.0	0.071	0.000	0.000	0.00	stream-work

…

- The stream routine is the second most time consuming routine in our program.
- It shows poor performance in vector length, vector operation ratio, MFLOPS.
- Custom ftrace regions reveal that the actual work in the stream routine performs well.
- Almost all time is spend in the communication with the IO-process, but the IO-process spends little time in communicating with the stream processes.

© NEC Corporation 2019

\Orchestrating a brighter world NEC

FTRACE Profile – Example Analysis

```
…
   ELAPSED      COMM.TIME  COMM.TIME   IDLE TIME  IDLE TIME  AVER.LEN      COUNT  TOTAL LEN PROC.NAME
   TIME[sec]        [sec]  / ELAPSED       [sec]  / ELAPSED    [byte]                [byte]

       5.608        0.018                  0.016               128.0M         20       2.5G matrix
       5.541        5.535                  5.506               128.0M         20       2.5G IO_matrix
       5.056        5.015                  5.009               256.0M         20       5.0G stream
       0.891        0.007                  0.312                 0.0           0       0.0  main
       0.067        0.061                  0.012               256.0M         20       5.0G IO_stream
----------------------------------------------------------------------------------------------------

       5.015        5.015                  5.009               256.0M         20       5.0G stream-comm
       0.036        0.000                  0.000                 0.0           0       0.0  stream-work
----------------------------------------------------------------------------------------------------
…
```

- The stream routine spends most of it’s communication time idling.

© NEC Corporation 2019
\Orchestrating a brighter world NEC

FTRACE Profile – Example Analysis

```
…
   ELAPSED      COMM.TIME  COMM.TIME    IDLE TIME  IDLE TIME  AVER.LEN      COUNT  TOTAL LEN PROC.NAME
   TIME[sec]        [sec]  / ELAPSED        [sec]  / ELAPSED    [byte]                [byte]

       5.608        0.018                   0.016               128.0M         20       2.5G matrix
       5.541        5.535                   5.506               128.0M         20       2.5G IO_matrix
       5.056        5.015                   5.009               256.0M         20       5.0G stream
       0.891        0.007                   0.312                 0.0           0       0.0  main
       0.067        0.061                   0.012               256.0M         20       5.0G IO_stream
----------------------------------------------------------------------------------------------------

       5.015        5.015                   5.009               256.0M         20       5.0G stream-comm
       0.036        0.000                   0.000                 0.0           0       0.0  stream-work
----------------------------------------------------------------------------------------------------
…
```

- The stream routine spends most of it’s communication time idling.
- Also the IO-process has high idle-times, but in the matrix-IO routine.

© NEC Corporation 2019
\Orchestrating a brighter world NEC

FTRACE Profile – Example Analysis

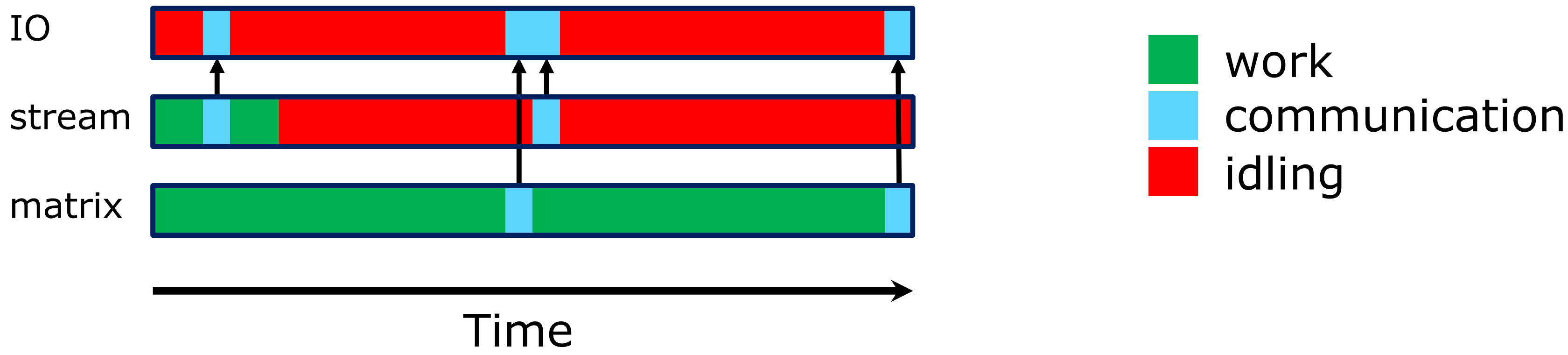
```
…
  ELAPSED       COMM.TIME  COMM.TIME   IDLE TIME  IDLE TIME  AVER.LEN      COUNT  TOTAL LEN PROC.NAME
  TIME[sec]         [sec]  / ELAPSED       [sec]  / ELAPSED    [byte]                [byte]

      5.608         0.018                  0.016              128.0M         20        2.5G matrix
      5.541         5.535                  5.506              128.0M         20        2.5G IO_matrix
      5.056         5.015                  5.009              256.0M         20        5.0G stream
      0.891         0.007                  0.312                 0.0          0        0.0  main
      0.067         0.061                  0.012              256.0M         20        5.0G IO_stream
----------------------------------------------------------------------------------------------------

      5.015         5.015                  5.009              256.0M         20        5.0G stream-comm
      0.036         0.000                  0.000                 0.0          0        0.0  stream-work
----------------------------------------------------------------------------------------------------
…
```

- The stream routine spends most of it’s communication time idling.
- Also the IO-process has high idle-times, but in the matrix-IO routine.
- Conclusion:
  - The stream work is much faster than the matrix work.
  - Stream processes have to wait for the IO-process to handle the matrix processes, which are the bottleneck.

© NEC Corporation 2019

FTRACE Profile – Example Analysis

- Conclusion:
  - The stream work is much faster than the matrix work.
  - Stream processes have to wait for the IO-process to handle the matrix processes, which are the bottleneck.
  - This imbalance artificially blows up the stream time, leading to poor performance values.



© NEC Corporation 2019

Profiling Layer

MPI-Profiling Layer

- Every MPI-routine exists in two versions but with identical functionality:
  - With MPI_ prefix
  - With PMPI_ prefix
- Thus, MPI-routines can be overwritten by users to insert custom code into MPI-calls.
- Within overwritten MPI-routines PMPI-routines provide the MPI-functionality.
- In order to use the profiling layer the special flag -mpiprof needs to be passed to the compiler.

© NEC Corporation 2019

MPI-Profiling Layer – Example Code

- We suspect that uneven workload leads to imbalances thus wasted computing power in some processes.

```c
#include <mpi.h>

int main(int argc, char **argv) {
   MPI_Init(&argc, &argv);
   int rank;
   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
   sleep(rank); // simulates uneven workload
   MPI_Barrier(MPI_COMM_WORLD);
   MPI_Finalize();
}
```

```
$ mpincc            -o prof.x prof.c
```

```fortran
PROGRAM Profiling_layer
   USE mpi
   INTEGER error, rank
   CALL MPI_Init(error)
   CALL MPI_Comm_rank(MPI_COMM_WORLD, rank, error)
   CALL SLEEP(rank) !simulates uneven workload
   CALL MPI_Barrier(MPI_COMM_WORLD, error)
   CALL MPI_Finalize(error)
END PROGRAM Profiling_layer
```

```
$ mpinfort            -o prof.x prof.F90
```


© NEC Corporation 2019

MPI-Profiling Layer – Example Code

- We suspect that uneven workload leads to imbalances thus wasted computing power in some processes.
- Overwriting the MPI_Barrier routine gives access to imbalances.

```c
#include <stdio.h>
#include <time.h>
#include <unistd.h>
#include <mpi.h>

int MPI_Barrier(MPI_Comm comm) {
   time_t barr_time;
   barr_time = time(NULL);
   int rank;
   PMPI_Comm_rank(comm, &rank);
   printf(" Rank %d Reached Barrier at %s",
          rank, ctime(&barr_time));
   PMPI_Barrier(comm);
}

int main(int argc, char **argv) {
   MPI_Init(&argc, &argv);
   int rank;
   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
   sleep(rank);
   MPI_Barrier(MPI_COMM_WORLD);
   MPI_Finalize();
}
```

```fortran
SUBROUTINE MPI_Barrier(comm, error)
   USE mpi, ONLY : PMPI_Comm_rank, &
                   PMPI_Barrier
   INTEGER, INTENT(IN) :: comm
   INTEGER, INTENT(OUT) :: error
   INTEGER :: rank
   CHARACTER(LEN=30) date
   CALL FDATE(date)
   CALL PMPI_Comm_rank(comm, rank, error)
   WRITE(*,*) "Rank", rank, &
      "Reached Barrier at ", date
   CALL PMPI_Barrier(comm, error)
END SUBROUTINE MPI_Barrier

PROGRAM Profiling_layer
   USE mpi
   INTEGER error, rank
   CALL MPI_Init(error)
   CALL MPI_Comm_rank(MPI_COMM_WORLD, rank, error)
   CALL SLEEP(rank) !simulates uneven Workload
   CALL MPI_Barrier(MPI_COMM_WORLD, error)
   CALL MPI_Finalize(error)
END PROGRAM Profiling_layer
```

```
$ mpincc          -o prof.x prof.c
```

```
$ mpinfort          -o prof.x prof.F90
```


© NEC Corporation 2019

MPI-Profiling Layer – Example Code

- We suspect that uneven workload leads to imbalances thus wasted computing power in some processes.
- Overwriting the MPI_Barrier routine gives access to imbalances.
- The special flag –mpiprof is needed for the profiling layer.

```c
#include <stdio.h>
#include <time.h>
#include <unistd.h>
#include <mpi.h>

int MPI_Barrier(MPI_Comm comm) {
   time_t barr_time;
   barr_time = time(NULL);
   int rank;
   PMPI_Comm_rank(comm, &rank);
   printf(" Rank %d Reached Barrier at %s",
          rank, ctime(&barr_time));
   PMPI_Barrier(comm);
}

int main(int argc, char **argv) {
   MPI_Init(&argc, &argv);
   int rank;
   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
   sleep(rank);
   MPI_Barrier(MPI_COMM_WORLD);
   MPI_Finalize();
}
```

```
$ mpincc -mpiprof -o prof.x prof.c
```

```fortran
SUBROUTINE MPI_Barrier(comm, error)
   USE mpi, ONLY : PMPI_Comm_rank, &
                   PMPI_Barrier
   INTEGER, INTENT(IN) :: comm
   INTEGER, INTENT(OUT) :: error
   INTEGER :: rank
   CHARACTER(LEN=30) date
   CALL FDATE(date)
   CALL PMPI_Comm_rank(comm, rank, error)
   WRITE(*,*) "Rank", rank, &
      "Reached Barrier at ", date
   CALL PMPI_Barrier(comm, error)
END SUBROUTINE MPI_Barrier

PROGRAM Profiling_layer
   USE mpi
   INTEGER error, rank
   CALL MPI_Init(error)
   CALL MPI_Comm_rank(MPI_COMM_WORLD, rank, error)
   CALL SLEEP(rank) !simulates uneven Workload
   CALL MPI_Barrier(MPI_COMM_WORLD, error)
   CALL MPI_Finalize(error)
END PROGRAM Profiling_layer
```

```
$ mpinfort -mpiprof -o prof.x prof.F90
```


© NEC Corporation 2019

MPI-Profiling Layer – Example Code

```
$ mpirun -np 4 prof.x
 Rank 0 Reached Barrier at Thu Oct 21 06:17:28 2021
 Rank 1 Reached Barrier at Thu Oct 21 06:17:29 2021
 Rank 2 Reached Barrier at Thu Oct 21 06:17:30 2021
 Rank 3 Reached Barrier at Thu Oct 21 06:17:31 2021
```

- Using the profiling layer reveals the imbalance of workload on the processes.
- NEC has already done the work for you by overwriting all MPI-communication routines in the vftrace profiler to enable logging of MPI-specific information.

© NEC Corporation 2019

\Orchestrating a brighter world NEC

Message Verification

MPI Message Verification

```
#include <stdlib.h>
#include <mpi.h>
int main(int argc, char **argv) {
   MPI_Init(&argc, &argv);
   int comm_size, my_rank;
   MPI_Comm_size(MPI_COMM_WORLD, &comm_size);
   MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
#define NINTS 32
   int sbuffer[NINTS];
   if (my_rank > 0) {
      MPI_Gather(sbuffer, NINTS, MPI_INT,
                 NULL, NINTS, MPI_INT,
                 0, MPI_COMM_WORLD);
   }
   MPI_Finalize();
}
```

- NEC-MPI provides the -mpiverify compiler flag to notify you during runtime if collective calls are inconsistent.
- Note the missing else branch in the testcode, leading to no Gather call on the root process.

```
$ mpincc            -o msg_ver.x msg_ver.c
$ mpirun -np 4 ./msg_ver.x
```

- Process 0 waits for all processes in Finalize().
- All other processes wait for process 0 to accept the Gather.
- The program hangs and needs to be terminated.

© NEC Corporation 2019

MPI Message Verification

```
#include <stdlib.h>
#include <mpi.h>
int main(int argc, char **argv) {
   MPI_Init(&argc, &argv);
   int comm_size, my_rank;
   MPI_Comm_size(MPI_COMM_WORLD, &comm_size);
   MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
#define NINTS 32
   int sbuffer[NINTS];
   if (my_rank > 0) {
      MPI_Gather(sbuffer, NINTS, MPI_INT,
                 NULL, NINTS, MPI_INT,
                 0, MPI_COMM_WORLD);
   }
   MPI_Finalize();
}
```

- NEC-MPI provides the -mpiverify compiler flag to notify you during runtime if collective calls are inconsistent.
- Note the missing else branch in the testcode,
  leading to no Gather call on the root process.
- mpiverify informs you about the inconsistency and aborts the program.

```
$ mpincc -mpiverify -o msg_ver.x msg_ver.c
$ mpirun -np 4 ./msg_ver.x
VERIFY MPI_Gather(3): call inconsistent with call to MPI_Finalize by 0
VERIFY MPI_Gather(1): call inconsistent with call to MPI_Finalize by 0
VERIFY MPI_Gather(2): call inconsistent with call to MPI_Finalize by 0
VERIFY MPI_Finalize(0): call inconsistent with call to MPI_Finalize by 0
[0,2] - MPI_Gather : Inconsistent call of collective operation
[0,3] - MPI_Gather : Inconsistent call of collective operation
[0,0] - MPI_Finalize : Inconsistent call of collective operation
[0,1] - MPI_Gather : Inconsistent call of collective operation
[0,3]  Aborting program !
[0,2]  Aborting program !
…
```


© NEC Corporation 2019

\Orchestrating a brighter world