MPI Runtime Error Detection with MUST Hands-On

At the 27th VI-HPS Tuning Workshop

Joachim Protze
IT Center RWTH Aachen University
April 2018

Content

- Motivation
- MPI usage errors
- Examples: Common MPI usage errors
  - Including MUST's error descriptions
- Correctness tools
- MUST usage
- **Hands-on**

```
$ cp -r ~lu23voz/tutorial/must must-examples
```

Hands On – Build for MUST

- Go into the NPB directory
- Edit config/make.def or copy intel.def from example-dir
- Disable any other tool (i.e. use mpifort, unset PREP)
- Use intel tool chain
- Build:

```
COMPFLAGS = -openmp -g -xMIC-AVX512 # intel
…
MPIF77 = mpifort
```

```
% module purge
% module load lrz/default
% make clean
% make bt-mz NPROCS=8 CLASS=B
  ========================================
  =     NAS PARALLEL BENCHMARKS 3.3      =
  =     MPI+OpenMP Multi-Zone Versions   =
  =     F77                              =
  ========================================

cd BT-MZ; make CLASS=B NPROCS=8
make[1]: Entering directory
…
mpif77 -O3 -g -openmp -extend-source  -o ../bin/bt-mz_B.8
bt_scorep_user.o  …
```

```
$ module use ~lu23voz/.modules
$ module load must
$ cp -r ~lu23voz/tutorial/must must-examples
```

Hands On - Prepare Job

- Create and edit the jobscript

```
$ cp -r ~lu23voz/tutorial/must must-examples
$ cd bin
$ cp ../must-examples/must-intel.sbatch .
```

- Jobscript:

```
…
module use ~lu23voz/.modules
module load must/intel
…
export OMP_NUM_THREADS=6
CLASS=B
NPROCS=8
…
mustrun --must:mpiexec mpiexec -n $NPROCS -t $OMP_NUM_THREADS  $EXE
```

MUST needs one extra process!
We use 8 processes * 6 threads + 1 tool process

Hands On – Executing with MUST

```
$ module use ~lu23voz/.modules
$ module load must
$ cp -r ~lu23voz/tutorial/must must-examples
```

- Submit the jobscript:

```
sbatch must-intel.sbatch
```

- Job output should read like:

```
…
[MUST] Using prebuilt infrastructure at /home/hpc/a2c06/lu23voz/sw/TOOLS/must/modules//mode1-layer2
…
[MUST] Executing application:

NAS Parallel Benchmarks (NPB3.3-MZ-MPI) - BT-MZ MPI+OpenMP Benchmark
…
Total number of threads:     48  (  6.0 threads/process)
Calculated speedup =     47.97

Time step    1
…
Verification Successful
…
[MUST] Execution finished, inspect "(…)/MUST_Output.html"!
```

```
$ module use ~lu23voz/.modules
$ module load must
$ cp -r ~lu23voz/tutorial/must must-examples
```

BT – MUST Results

- Open the MUST output: <Browser> MUST_Output.html

Rank(s)	Type	Message		
0-3	**Warning**	You requested 3 threads by OMP_[illegible]_THR requested thread level MPI_THREAD_FUNNELED from the mpi library but thr library provides no thread support.This is ok as long as your application doesn't make use of OpenMP	**MPI_Init_thread** (1st occurrence) called from: #0 MAIN__@bt.f:90 #1 main@bt.f:319	
0-3	**Error**	There are 1 communicators that are not freed when MPI_Finalize was issued, a quality application should free all MPI resources before calling MPI_Finalize. Listing information for these communicators: -Communicator 1: Communicator created [illegible] reference 1 size=4	Representative location: **MPI_Comm_split** (1st occurrence) called from: #0 MAIN__@bt.f:90 #1 main@bt.f:319	References of a representative process: reference 1 rank 2: **MPI_Comm_split** (1st occurrence) called from: #0 MAIN__@bt.f:90 #1 main@bt.f:319

BT-MZ should evaluate the “provided” thread level and don’t use threads.

Resource leak:
A communicator created with MPI_Comm_split is not freed

Stacktraces in MUST

- We use an external lib for stacktraces
- This lib has no support for Intel compiler
  - But: in most cases it's compatible to icc compiled C applications
  - Nevertheless, the must-intel module is built without stacktrace support
- Ifort compiled FORTRAN applications can lead to segfault:
  - Use MUST w/o stacktraces for fortran applications
  - Use GNU compiler to build your application and use MUST w/ stacktraces
- Supposed your application has no faults you won't need stacktraces ☺

From
Representative location: **MPI_Init_thread** (1st occurrence) called from: #0 MAIN__@bt.f:90 #1 main@bt.f:319
Representative location: **MPI_Comm_split** (1st occurrence) called from: #0 MAIN__@bt.f:90 #1 main@bt.f:319

Rank(s)	Type	Message	From	References
	Information	MUST detected no MPI usage errors nor any suspicious behavior during this application run.		

Conclusions

- Many types of MPI usage errors
  - Some errors may only manifest sometimes
  - Consequences of some errors may be “invisible”
  - Some errors can only manifest on some systems/MPIs
- Use MPI correctness tools
- Runtime error detection with MUST
  - Provides various correctness checks
  - Verifies type matching
  - Detects deadlocks
  - Verifies collectives

Thank You