pawsey

Stress-free MPI Stress Tests

Dr. Pascal Jahan Elahi, Craig Meyer
Pawsey Supercomputing Research Centre
CUG 23

Australian Government
NCRIS National Research Infrastructure for Australia
CSIRO
Curtin University
Murdoch University
Government of Western Australia
ECU
The University of Western Australia

Outline

- Why develop a new suite of MPI tests?
  - Problems with MPI on Setonix, HPE Cray EX system
- How the tests are designed
  - Some examples
- How can other centres and yourself make use of them?
- Lessons Learnt

Pawsey Supercomputing Research Centre

- Headquarters located in Perth, Western Australia
- Pawsey has a 20-year long history.
- Offers critical support to radioastronomy research around the Square Kilometre Array (SKA).
- The centre underwent a 70m capital refresh financed by the Australian government.
- Currently employs over 60 staff.
- Houses the Setonix, a HPE Cray EX system, which is the largest supercomputer in Australia.

Pawsey

Setonix Phase-1, Phase-2 and MPI

- Setonix was deployed in two phases.
- Phase 1:
  - 512 CPU compute nodes (2 AMD Milan CPUs)
  - Mellanox NICs and libfabric 1.11.x
  - Cray MPICH 8.1.14
  - Passed acceptances tests of HPL, OSU Micro Benchmarks (OMB), amongst others (e.g., single-node LAMMPS)
- Phase 2:
  - 192 GPU compute nodes (4 MI250X GPUs), 1600 CPU compute nodes (2 AMD Milan CPUs)
  - Cassini NICs and libfabric 1.15.x
  - Cray MPICH 8.1.19

Setonix Phase-2

MPI Issues Encountered

- Despite passing acceptance tests, a number of researchers running more complex workflows encountered issues during Setonix Phase-1
- These ranged from unexplained segfaults to hangs to unexpected memory usage to poor scaling.

The main issues can be categorised as

Delay-Hang	Software with point-to-point (pt2pt) communication would hang if there was a long time-difference between associated send and receives.
Memory Leaks	Multi-node jobs were crashing with a variety of reported errors: bus errors; generic SLURM kill errors; out-of- memory errors; xpmem or Open Fabrics Interface errors.
Poor Scaling	Multi-node scaling of software with significant pt2pt communication did not scale well past two nodes. Even a much older Cray XC system outperformed Setonix by factors of $\gtrsim$5 when using $\gtrsim$ 96 cores across multiple nodes.
Reduced Node Memory	Available memory on idle nodes slowly decreased.
Large-comm Instability	Crashes occurred when running jobs with pt2pt communication with large number of processes ($\gtrsim$ 700). Additionally, hangs were observed when using asynchronous pt2pt communication with high message counts.

Understanding MPI Issues

- Number of issues encountered by users. Not obvious what the underlying issues were and whether they were all related.
- Not all workflows impacted but some key stakeholders could not run.
- We did NOT have a simple set of diagnostic-oriented MPI tests.
  - MPI performance was measured using OSU Micro Benchmarks (OMB).
  - However, these are not designed for debugging.
  - OMB could pass when production codes would fail.

- Motivated by understanding these issues and realising there was a gap, we developed a suite of MPI stress tests focused on
  - MPI communication patterns seen in production code
  - Providing diagnostic information


CLUEDO
Klassinen mysteeripeli
Tapaa UUSI epäilty: TOHTORI PINKKILÄ
IKÄ 8+
2–6 PELAAJAA
Hasbro Gaming


BARINA
W.A.
SEGFAULT
WA
MEMLEAK

Stress-free MPI Stress tests

- Current bare-bones version available via https://github.com/PawseySC/Refame-MPI-Stress-Tests
- Current setup for
  - SLURM batch scripts
  - Cray Programming Environments
  - Lmod modules
  - Reframe >=3.10
- To deploy local will require updates.
  - add systems, accounts, and some modifications to reframe test related to system

Stress-free MPI Stress Tests

- Our tests are simple C++ codes that only use c++14
- Covers a variety of MPI communication patterns seen in production codes.
- Provide diagnostic information using simple utilities like `dmesg`.

Design

Before

Record Node State (`dmesg`, `free`, etc)

During

Run MPI Test

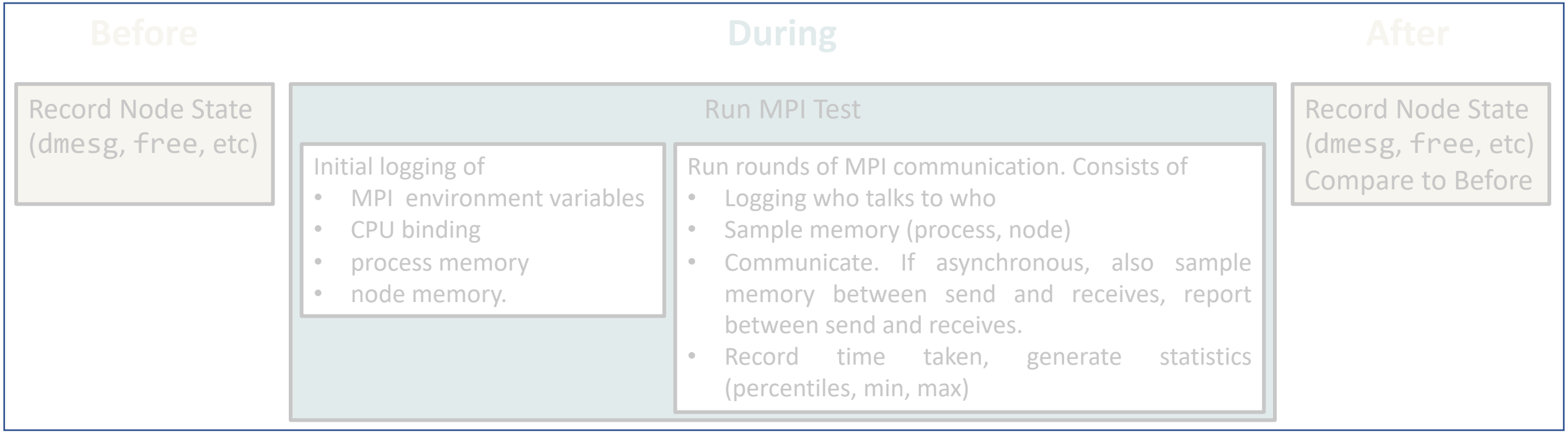
Initial logging of
- MPI environment variables
- CPU binding
- process memory
- node memory.

Run rounds of MPI communication. Consists of
- Logging who talks to who
- Sample memory (process, node)
- Communicate. If asynchronous, also sample memory between send and receives, report between send and receives.
- Record time taken, generate statistics (percentiles, min, max)

After

Record Node State (`dmesg`, `free`, etc)
Compare to Before

Design



Available Communication Patterns

Collective:

- Synchronous, Asynchronous.
- Variable message size.
- Variable MPI Communicator size (starts small with 2 processes per communicator, up to entire comm world size

Point-to-Point:

- Synchronous, Asynchronous send, Asynchronous send & receive.
- Variable message size.
- Variable delay between send and receives
- Variable amount of the MPI comm world each rank communicates with

Example C++ Source code: Testing a Delay

```
// Test a long delay between a send/receive. This can happen in real-time processing software
void MPITestLongDelay(int delay, int rootproc, int otherproc, int usesend)
{
  MPI_Status status; MPI_Request request;
  auto comm = MPI_COMM_WORLD; int NProcs; MPI_Comm_size(comm, &NProcs);
  std::vector<double> data(1000); void * p1 = nullptr;
  // sleep all processes but the immediate process that places a send
  if (ThisProc != otherproc) sleep(delay);
  // process that receives all messages
  if (ThisProc == rootproc) {
    for (auto iproc = 0; iproc < NProcs; iproc++) {
      if (iproc == rootproc) continue;
      LocalLogger(); // log memory, communication, etc
      // receive message
      mpi_err = MPI_Recv(&size, 1, MPI_UNSIGNED_LONG, iproc, 0, comm, &status);
      LocalLogger(); // log info
    }
  }
  // all other processes, including one without a delay sending
  else {
    LocalLogger(); // log info
    // send message based on the usesend flag indicates how messages are sent, either blocking or non-blocking.
    if (usesend == USESEND) mpi_err = MPI_Send(&size, 1, MPI_UNSIGNED_LONG, rootproc, 0, comm);
    else if (usesend == USEISEND) {
      mpi_err = MPI_Isend(&size, 1, MPI_UNSIGNED_LONG, rootproc, 0, comm, &request);
      mpi_err = MPI_Wait(&request, &status);
    }
    LocalLogger(); // log info
  }
}
```

Example Node State Script

```
#!/bin/bash

# Get hosts to see what state they are in (this returns the host name of every process, not unique list of hosts)
hostnames=($(srun hostname))
# Create logging directory which stores the dmesg output log files
logdir=logs/${SLURM_JOB_ID}
if [[ ! -d $logdir ]]
then
    mkdir -p $logdir
fi
# Check node health via `dmesg` and node memory via `free`
hostnames=($(for hn in "${hostnames[@]}"; do echo "${hn}"; done | sort -u)) # Get unique nids
timestamp=$(date -Iseconds)
logfile=${logdir}/node-state.${h}.${timestamp}.job-${SLURM_JOB_ID}.txt
for h in ${hostnames[@]}
do
    srun -w ${h} --nodes=1 --ntasks=1 --ntasks-per-node=1 --mem=1GB dmesg -T > ${logfile}
    srun -w ${h} --nodes=1 --ntasks=1 --ntasks-per-node=1 --mem=1GB free -h >> ${logfile}
    numerr=$(grep -aic "error" ${logdir}/node-state.${h}.${timestamp}.job-${SLURM_JOB_ID}.txt)
    echo "There are $numerr errors in the dmesg output for node ${h}"
done
```

Example Reframe

```python
import reframe as rfm
import reframe.utility.sanity as sn
# Base MPI communications test class
class MPI_Comms_Base(rfm.RegressionTest):
    def __init__(self, name, **kwargs):
        # set metadata, valid systems and PE.
        # …
        # Compilation
        self.build_system = 'SingleSource'
        self.build_system.cppflags = [‘…',]
        self.prebuild_cmds = ['./copy.sh']
        # Output MPI environment variables
        self.prerun_cmds = [
            'export MPICH_ENV_DISPLAY=1’, 'export MPICH_MEMORY_REPORT=1',
            'export MPICH_OFI_VERBOSE=1’, 'export FI_CXI_DEFAULT_VNI=$(od -
vAn -N4 -tu < /dev/urandom)',]
        self.num_cpus_per_task = 1
        self.keep_files = ['logs/*']
        self.tags = {'MPI'}
    # Compile profile_util
    @run_before('compile')
    def compile_prof_util(self):
        self.prebuild_cmds += ['cd profile_util', './build_cpu.sh',
'PROFILE_UTIL_DIR=$(pwd)', 'cd ../',]
    # defined functions to set account for billing, srun commands
    # …
    # Check node health pre- and post-job
    @run_before('run')
    def check_node_health(self):
        self.prerun_cmds += ['./node_check.sh',]
        self.postrun_cmds = ['./node_check.sh',]
    # Test passes if the end of the job is reached
    @sanity_function
    def assert_complete(self):
        return sn.assert_found(r'Job completed at.+', self.stdout)
```

```python
# Point-to-point communication test
@rfm.simple_test
class Pt2Pt(MPI_Comms_Base):
    def __init__(self, **kwargs):
        super().__init__('Pt2Pt', **kwargs)
        # Metadata
        # …
        # Compilation – source and executable
        self.sourcepath, self.executable = 'pt2pt.cpp', 'pt2pt.out’
        # Set the defaults of executable options
        # …
        # Set sbatch script directives
        self.num_tasks_per_node = 24
        self.num_tasks = self.num_nodes * self.num_tasks_per_node
        # Reference value when run with base conditions (one node, one task,
etc.)
        self.ref_val = 2e6
        # Dictionary holding performance reference values
        scaling_factor =
        self.reference = {‘system': {'Average': (self.ref_val * scaling_factor,
None, 0.2)}, }

    # Parameters
    num_nodes = parameter([1, 2, 4, 8])#, 16, 32])
    # Performance function for the recorded time statistics
    @performance_function('us')
    def extract_timing(self, kind='Average'):
        if kind not in ('Average', 'Standard Deviation', 'Maximum', 'Minimum',
'IQR'):
            raise ValueError(f'Illegal value in argument kind ({kind!r})')
        # Extract timing for redistribution phase with pt2pt communication
        return sn.extractsingle(rf'@redistributeData.+{kind}\s=\s(\S+),.+',
                                self.stdout, 1, float)
```

Example Reframe Output

```
reframe -C settings.py -c ./reframe-mpi-tests/mpi/ -t MPI -r

          ==============================================================================
          SUMMARY OF FAILURES
          ------------------------------------------------------------------------------
          FAILURE INFO for MemoryLeak_2_24
            * Expanded name: MemoryLeak %num_nodes=2 %ntasks_per_node=24
            * Description: Test memory sampling/reporting during MPI comms
            * System partition: setonix:work
            * Environment: PrgEnv-gnu
            * Stage directory: …
            * Node list: nid002422,nid002440
            * Job type: batch job (id=1508167)
            * Dependencies (conceptual): []
            * Dependencies (actual): []
            * Maintainers: ['Craig', 'Pascal Jahan Elahi']
            * Failing phase: sanity
            * Rerun with '-n MemoryLeak_2_24 -p PrgEnv-gnu --system setonix:work -r'
            * Reason: sanity error
```

Revisit the Issues: What did we find and how did we find it?

Delay-Hang	Software with point-to-point (pt2pt) communication would hang if there was a long time-difference between associated send and receives.
Poor Scaling	Multi-node scaling of software with significant pt2pt communication did not scale well past two nodes. Even a much older Cray XC system outperformed Setonix by factors of $\gtrsim 5$ when using $\gtrsim$ 96 cores across multiple nodes.
Memory Leaks + Reduced Node Memory	Multi-node jobs were crashing with a variety of reported errors: bus errors; generic SLURM kill errors; out-of- memory errors; xpmem or Open Fabrics Interface errors. Available memory on idle nodes slowly decreased.

What did we find and how did we find it?

Delay-Hang

Crime Scene:

- showed up in multi-node pt2pt communication when communication was across nodes.
- Node state did not impact hang, only time and comm size.
- Receiving task would never receive the message, regardless of how it was sent.

Clues found by looking at verbose communication messages.

Lines of Evidence: pointed to libfabric since communication had to be across nodes.

Workaround: Can be resolved by using `MPICH_OFI_STARTUP_CONNECT=1`
Solution: Update to newer libfabric (and newer Cassini NICs) for bug fixes.

What did we find and how did we find it?

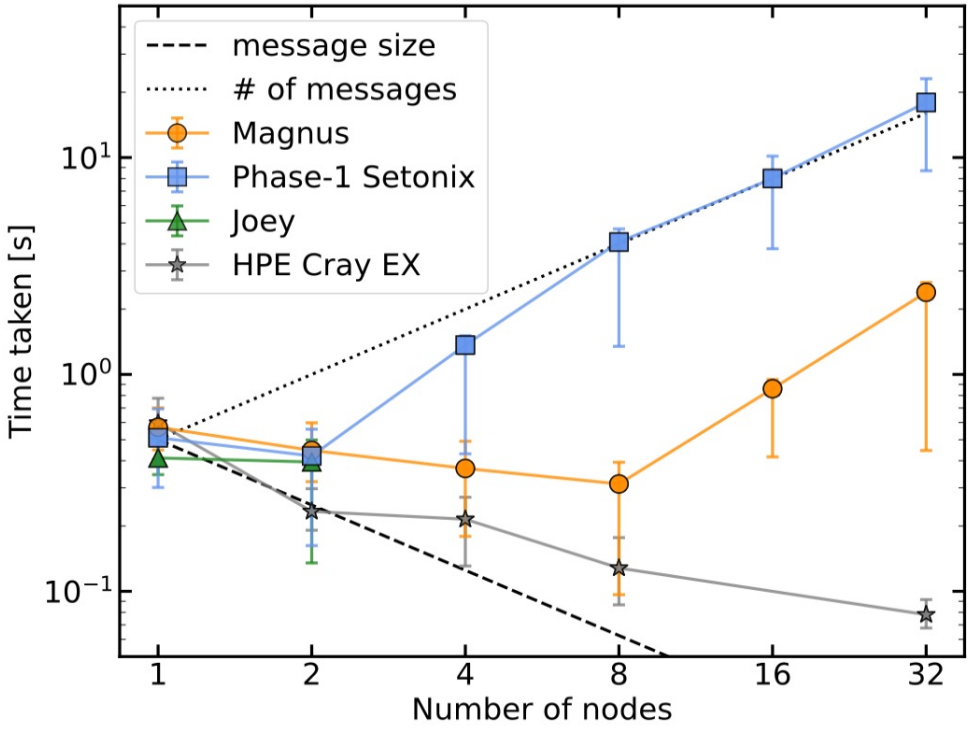
Poor Scaling

Crime Scene:

- showed up in multi-node pt2pt communication when communication was across nodes. Collectives not impacted. Mostly limited to pt2pt where each process communicated to almost all other processes.
- Significant jumps in time taken dependent on comm size, number of nodes and how much communication each rank undertook.

Clues found by looking at statistics and different specific communication patterns.

Lines of Evidence pointed to libfabric since communication had to be across nodes and libfabric variable related to on-demand communication could drastically improve performance.



Weak-scaling test of time taken to send fixed amount of data across MPI ranks in a pt2pt fashion (MPI domain decomposition and redistribution). Results are for 24 ranks per node to compare results with older Cray XC system

Workaround: Partially resolved by using `MPICH_OFI_STARTUP_CONNECT=1`

Solution: Update to newer libfabric and newer Cassini NICs solved issue.

What did we find and how did we find it?

Memory Leaks (and Reduced Node Memory)

Crime Scene:

- showed up in multi-node communication when communication included inter-node communication.
- Process memory reasonable but node memory showed clear reduction in available memory during communication.
- Amount of memory consumed dependent on comm size.
- Errors occurred when the amount of available node memory not enough given memory consumed by process or beyond total physical memory available on node.
- Jobs that completed would slowly reduce the amount of memory available on the node afterwards.
- Crashes not only left messages in kernel ring buffer that could involve the memory but could leave errors pertaining to HSN. In this case, node unusable as all subsequent MPI jobs would crash upon initialization.

Evidence found by looking at kernel ring buffer messages and node memory state and memory used by process.

Lines of evidence pointed to libfabric since communication had to involve inter-node communication and memory consumed not visible in user space.

Solution: Update to newer libfabric and newer Cassini NICs solved issue.

Even other uses

Segfaults when running internode MPI

Crime scene:

- Showed up after upgrade from Mellanox to Cassini NICs and newer libfabric.
- Segfault occurred for any MPI communication when communication was across nodes.
- Left nodes unable to MPI_Init afterwards, meaning no multi-node job would run.
- Crashes not only left messages in kernel ring buffer that could involve the memory but could leave errors pertaining to HSN. In this case, node unusable as all subsequent MPI jobs would crash upon initialization.

Evidence found by looking at kernel ring buffer messages.

Lines of evidence pointed to incorrect network configuration after upgrade.

Solution: Fix configuration.

Lessons Learned

- We believe there is currently a gap in stress-testing MPI implementations on HPC systems
- Tests such as OMB good for benchmarking system, but can allow issues with the MPI implementation to fall through the cracks undetected
- MPI tests that stress the system and use communication patterns replicating production codes should be a part of HPC cluster test suites
- Detailed logging before, during, and after tests invaluable in detection and triage of issues

Any Questions?

pawsey