JuRoPA - Tuning for the platform

Usage of ParaStation MPI

2015-05-18

Peter Niessen
Outline

▪ Why ParaStation
▪ Why tuning
▪ Compiling your program
▪ Running your program
▪ Tuning parameters
▪ Resources
ParaStation History

- 1995: University project
- 2004: Open source
- 2004: Cooperation with FZJ/JSC/IAS
  - various precursor clusters
  - JuRoPA2
  - Judge
  - DEEP
  - JuRoPA3
  - JURECA
Why ParaStation

- Scalability
  - “Desirable property of system, network or process to handle growing amounts of work in a graceful manner [...]”
  - Runtime proportional to number of operations
  - Runtime inverse proportional to amount of resources
Performance (hpl with ipsmpi)
Robustness

- Ability of a computer system to cope with errors during execution [...] despite abnormalities in input [...] 
- Crash of a parallel job:
  - *Must not crash master control*
  - *Must release resources*
- ParaStation solution: daemon concept
  - *No single point of failure*
  - *Management of entire process lifecycle*
  - *Resource monitoring*
- Job clean up after
  - *Process dies (intentionally or unintentionally)*
  - *Node dies*
End user benefits

- Centralized I/O forwarding
- Signal forwarding
- Hybrid Applications OpenMP/MPI supported
- Selecting most performing communication automatically
- No hostfile necessary
- Easy access to resource accounting information
Schematic Overview

- **J2 psmom**
- **psmgmt**
  - process startup and control
- **JTest psslurm**
- **pscom**
  - communication abstraction layer
- **psmpi**
  - MPI library
Compiling the application (jtest)

- Only MPI3 version (5.1.2) available
- module avail 2>&1 | grep psmpi
- [i|g]psmpi/2015.04[mt]
  - single thread task/multi-thread tasks (mt)
  - mt only available for intel (might have changed by now)
- ChangeLog available at $(dirname $(which mpicc))/../ChangeLog

- <compiler>
  - GCC-5.1.0
  - Intel-2015.3.187
- Clean slate:
  - module purge
  - module load ...
Current versions

- **JuRoPATest**
  - *concept of tool chains*
  - *psmpi-5.1.2*
  - *includes all the Intel tools*

- **JuRoPA2**
  - *parastation/mpi2-intel12-5.0.29-1*
  - *intel/13.1.3 (includes MKL)*
  - *parastation/mpi-intel13-5.1.0*
Wrapper vs. manual compilation

- **Wrappers**
  - `mpicc (C)`
  - `mpicxx (C++)`
  - `mpif90 (Fortran 90)`
  - `mpif77 (Fortran 77)`
- `mpi<LANG> -show`
  - *shows what would happen*
  - *useful for legacy Makefiles (e.g. xhpl)*
  - *allows to tweak compiler (MPICH_CC, CXX, F90, F77)*
- **When using the mt version, add**
  - `-fopenmp (gcc)`
  - `-openmp (intel)`
Did the wrapper link correctly?

- Libraries are linked at runtime according to `LD_LIBRARY_PATH`.
- `ldd` shows the libraries attached to your binary.
- Look for ParaStation/Intel libraries.
- E.g., `ldd hello_mpi_default`:
  - `linux-vdso.so.1 => (0x00007fffd1b0000)`
  - `libmpich.so.1.2 => /usr/local/parastation/mpi2-intel-mt-5.0.26-1/lib/libmpich.so.1.2 (0x00007f5c788c2000)`

Wrong module!!!
Jtest: start via srun

- `srun -N <nodes> -n <tasks>` spawns the tasks inside
  - *Interactively*
  - *From batch job*
- Full environment is exported via srun
  - `PSP_*` *(communication)*
  - `PSI_*` *(process management)*
  - `LD_LIBRARY_PATH`
  - `__PSI*`
- Pressing `^C` (in interactive) will be passed to all tasks
- No manual clean-up needed
/* C Example */
#include <stdio.h>
#include <mpi.h>

int main (int argc, char **argv) {
    int numprocs, rank, namelen;
    char processor_name[MPI_MAX_PROCESSOR_NAME];

    MPI_Init (&argc, &argv);
    MPI_Comm_rank (MPI_COMM_WORLD, &rank);
    MPI_Comm_size (MPI_COMM_WORLD, &numprocs);
    MPI_Get_processor_name (processor_name, &namelen);
    printf ("Hello world from process %d of %d on %s\n",
            rank, numprocs, processor_name);
    MPI_Finalize ();
    return 0;
}
Compiling/running on JuRoPAtest

- mpicc -O3 -o hello_mpi_default hello_mpi.c
- # no msub to get interactive job
- # we're still on the frontend
- # -l nodes=2:ppn=1 translates to -N 2 -n 2
- srun -N 2 -n 2 ./hello_mpi_default
  - Hello world from process 1 of 2 on j3c062
  - Hello world from process 0 of 2 on j3c061
- # same with redirection: join stderr/out
- srun -N 2 -n 2 ./hello_mpi_default 2>&1 | sort
  - Hello world from process 0 of 2 on j3c061
  - Hello world from process 1 of 2 on j3c062
Manipulating process placement

- `psiadmin -c 'show cpumap <node>'`
  - `63 0 1 2 3 ... 54 55`

- Manipulate via
  - `--cpu_bind=map_cpu:C_1,\ldots,C_n`
  - `--cpu_bind=mask_cpu:<mask>,\ldots,<mask>`

- `srun --cpu_bind=map_cpu=0,4,8,12 -n 4 ./testcore`
  - Process 0 of 4, mask 00000001 on j3c110
  - Process 1 of 4, mask 00000010 on j3c110
  - Process 2 of 4, mask 00000100 on j3c110
  - Process 3 of 4, mask 00001000 on j3c110

- Large memory
- few tasks
- Try your application
#include <stdio.h>
#include "mpi.h"
#include <omp.h>

int main(int argc, char *argv[]) {
    int numprocs, rank, namelen;
    char processor_name[MPI_MAX_PROCESSOR_NAME];
    int iam = 0, np = 1;

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Get_processor_name(processor_name, &namelen);

    #pragma omp parallel default(shared) private(iam, np)
    {
        np = omp_get_num_threads();
        iam = omp_get_thread_num();
        printf("Hello from thread %d out of %d from process %d out of %d on %s\n",
               iam, np, rank, numprocs, processor_name);
    }

    MPI_Finalize();
}
Multithreaded application on jtest

- module load ipsmpi/2015.04-mt
- mpicc -O3 -fopenmp -o hello_hybrid hello_hybrid.c
- srun -N 2 -n 4 -cpus-per-task=4
  
  bash -c 'OMP_NUM_THREADS=
  {SLURM_CPUS_PER_TASK} ./hello_hybrid'

Hello from thread 0 out of 4 from process 0 out of 4 on j3c117
Hello from thread 1 out of 4 from process 0 out of 4 on j3c117
Hello from thread 2 out of 4 from process 0 out of 4 on j3c117
Hello from thread 3 out of 4 from process 0 out of 4 on j3c117
Hello from thread 0 out of 4 from process 1 out of 4 on j3c117
Hello from thread 1 out of 4 from process 1 out of 4 on j3c117
Hello from thread 2 out of 4 from process 1 out of 4 on j3c117
Hello from thread 3 out of 4 from process 1 out of 4 on j3c117
Hello from thread 0 out of 4 from process 2 out of 4 on j3c118
Hello from thread 1 out of 4 from process 2 out of 4 on j3c118
Hello from thread 2 out of 4 from process 2 out of 4 on j3c118
Hello from thread 3 out of 4 from process 2 out of 4 on j3c118
Hello from thread 0 out of 4 from process 3 out of 4 on j3c118
Hello from thread 1 out of 4 from process 3 out of 4 on j3c118
Hello from thread 2 out of 4 from process 3 out of 4 on j3c118
Hello from thread 3 out of 4 from process 3 out of 4 on j3c118
Further defining execution

- Don't do process pinning at all
  - \_PSI\_NO\_PINPROC=1 srun ...
- Allow the Intel OpenMP library thread placing
  - KMP\_AFFINITY=[verbose,]...
    compact: place threads as close as possible
    scatter: as evenly as possible
- Export to every task
  - KMP\_AFFINITY=... srun ...
- GCC: GOMP\_CPU\_AFFINITY (manual)
OMP_NUM_THREADS=4, -n 2

- **scatter**

  OMP: Info #147: KMP_AFFINITY: Internal thread 0 bound to OS proc set {0,8}
  OMP: Info #147: KMP_AFFINITY: Internal thread 0 bound to OS proc set {0,8}
  OMP: Info #147: KMP_AFFINITY: Internal thread 1 bound to OS proc set {4,12}
  OMP: Info #147: KMP_AFFINITY: Internal thread 1 bound to OS proc set {4,12}
  OMP: Info #147: KMP_AFFINITY: Internal thread 2 bound to OS proc set {1,9}
  OMP: Info #147: KMP_AFFINITY: Internal thread 2 bound to OS proc set {1,9}
  OMP: Info #147: KMP_AFFINITY: Internal thread 3 bound to OS proc set {5,13}
  OMP: Info #147: KMP_AFFINITY: Internal thread 3 bound to OS proc set {5,13}

- **compact**

  OMP: Info #147: KMP_AFFINITY: Internal thread 0 bound to OS proc set {0,8}
  OMP: Info #147: KMP_AFFINITY: Internal thread 0 bound to OS proc set {0,8}
  OMP: Info #147: KMP_AFFINITY: Internal thread 1 bound to OS proc set {0,8}
  OMP: Info #147: KMP_AFFINITY: Internal thread 1 bound to OS proc set {0,8}
  OMP: Info #147: KMP_AFFINITY: Internal thread 2 bound to OS proc set {1,9}
  OMP: Info #147: KMP_AFFINITY: Internal thread 2 bound to OS proc set {1,9}
  OMP: Info #147: KMP_AFFINITY: Internal thread 3 bound to OS proc set {1,9}
  OMP: Info #147: KMP_AFFINITY: Internal thread 3 bound to OS proc set {1,9}
ParaStation MPI

Debugging

**PSP_DEBUG=2**

```
srun -n 2 ./hello_jtest
<PSP:j3c075:48453:set PSP_DEBUG = 2>
...
<PSP:jj01c60:10503:Register arch shm with priority 1.9>
<PSP:jj01c60:10503:Register arch openib with priority 1.2>
...
<PSP:r0000000:CONNECT (192.168.12.94,<PID>,....,r0000000) to (192.168.12.94,<PID>,....,r0000000) via loop>
...
<PSP:r0000001:CONNECT (192.168.12.95,<PID>,....,r0000001) to (192.168.12.94,<PID>,....,r0000000) via openib>
Hello world from process 0 of 2 on j3c075
Hello world from process 1 of 2 on j3c076
<PSP:r0000001:Reqs:5 GenReqs: (cnt:0 used:0)>
<PSP:r0000001:Byee.>
<PSP:r0000000:Reqs:5 GenReqs: (cnt:0 used:0)>
<PSP:r0000000:Byee.>
```
Tuning

▪ Is it worth it?
  ▪ Use a suitable tool to find out

▪ What can you tune?
  ▪ Message queue length
  ▪ Memory allocation
PMI Barrier Timeout

- If you see ...
  - PSIlogger: Timeout: Not all clients joined the first pmi barrier:
    
    joined=327  left=9  round=1

- Typically in large applications (-np >> 1)
- ... you should increase the timeout ...
  - export PMI_BARRIER_TMOUT=<secs>, -1 for infinite wait
  - default is 60 seconds
- ... or increase the number of rounds
  - export PMI_BARRIER_ROUND=<N>
  - default is 1
Large Job considerations

- Every MPI task talks to all others: \((N-1) \times 0.55\) MB buffer space
- Example:
  - `srun -n 3584` *(64 56 core machines)*
  - \(\rightarrow 3583 \times 0.55\) MB \(\sim 2\) GB / task
  - x 56 tasks / machine \(\rightarrow 110\) GB buffer space
  - 128 GB main memory \(\rightarrow \sim 20\) GB left for application
- J2:
  - `mpiexec -np 4096` *(512 8 core machines)*
  - \(\sim 2\) GB / task
  - x 8 tasks / machine \(\rightarrow 16\) GB buffer space
  - 24 GB main memory \(\rightarrow \sim 8\) GB left for application
On Demand/Buffer Size

- To create buffers on demand only:
  - `export PSP_ONDEMAND=0 1`
  - `J2: mpiexec --ondemand`

- To reduce the buffer queue length:
  - `export PSP_OPENIB_SENDQ_SIZE=3`
  - `export PSP_OPENIB_RECVQ_SIZE=3`
  - *Do not go below 3, deadlocks might occur*

- Tradeoff: Performance penalty
On-Demand/Queue size guidelines

- On-Demand works best with nearest neighbour communications
  - scatter/gather
  - all-reduce
- All-to-all communication:
  - queue size modification only viable option
NUMA considerations

- Non Uniform Memory Access
NUMA policies

- Memory is bound to processes
- Only local memory is accessible
  - `niessen@j3l03:~/src/memgrow> srun -n 1 ./blockmem_mpi 112
    srun: error: j3c090: task 0: Exited with exit code 1`
  - `niessen@j3l03:~/src/memgrow> __PSI_NO_MEMBIND=1 srun -n 1 ./blockmem_mpi 112
    Binding suppressed for rank 0
    blocked 12 G`
- `membind on -> ~6 GB/s`
- `membind off, crossing CPUs -> ~5 GB/s`
- works on J2 the same way
Bandwidth drop across socket

- Allocates Memory
- Writes into it
Resources

- www.parastation.com
- www.fz-juelich.de/ias/jsc/EN/Expertise/Supercomputers/JUROPA/JUROPA_node.html
- /opt/parastation/doc/pdf/userguide.pdf
- support@par-tec.com
- Download ParaStation MPI at github:
  - https://github.com/ParaStation/psmgmt
  - https://github.com/ParaStation/pscom
  - https://github.com/ParaStation/psmpi2
  - git clone https://github.com/ParaStation/psmpi2.git
Compiling the application (JuRoPA2)

- Select the right version
  - module avail 2>&1 | grep parastation
- **MPI2 Implementation**: mpi2-<compiler>-[mt]-<version>
  - *single thread task/multi-thread tasks (mt)*
- **MPI3 Implementation**: mpi-intel13[-mt]-5.1.0
- only for intel/13.1.3
- **ChangeLog available at** $MPIHOME/ChangeLog
- **<compiler>**
  - gcc *(different versions)*
  - intel *(different versions)*
- The default (module list, a little outdated due to legacy applications)
  - parastation/mpi2-intel-5.0.26-1
  - mkl/10.2.5.035
  - intel/11.1.072
- **Clean slate:**
  - module purge
  - module load ...
JuRoPA2: mpiexec start mechanism

- `mpiexec -np <N>` spawns the tasks inside (torque/moab) job
- certain variables are exported to all tasks
  - `PSP_*` (*communication*)
  - `PSI_*` (*process management*)
  - `LD_LIBRARY_PATH`
  - `__PSI*`
- other variables can be passed via
  - `--exports=VAR1,VAR2,...,VARn`
- Pressing `^C` (in interactive) will be passed to all tasks
- No manual clean-up needed
Compiling and running on J2

- `mpicc -O3 -o hello_mpi_default hello_mpi.c`
- `msub -I -l nodes=2:ppn=1`
- `# we're on a node right now`
- `mpiexec -np 2 ./hello_mpi_default`
  - *Hello world from process 0 of 2 on jj01c56*
  - *Hello world from process 1 of 2 on jj01c55*
mpiexec options/environment

- **Options (man mpiexec)**
  - `-np`: number of tasks started
  - `-x`: export complete environment: *don't do this*
    places memory load on system
    
    PSIlogger: Timeout: Not all clients joined the first pmi barrier
  - `-e`: export selected environment variables (comma sep.)
    
    e.g. `-e VAR1,VAR2,VAR3,...,VARn`
  - *Don't specify machine files, hostfiles -> Batch System*

- **Environment (man ps_environment)**
  - `PSI_*`: Program Startup (psmgmt)
  - `PSP_*`: Communication Layer (pscom)
  - `PSP_DEBUG`: Debugging level
    
    1: shows connections between tasks
    2: + settings of variables, priorities of connections
  - *All PSP_*/PSI_*/LD_LIBRARY_PATH variables are exported*
Compiling and running multi-thread

- module load parastation...
- mpicc -O3 -openmp -o hello_hybrid hello_hybrid.c
- msub -I -l nodes=2:ppn=8 -v tpt=4
- OMP_NUMTHREADS=$tpt mpiexec
  --export=OMP_NUM_THREADS -np 4 ./hello_hybrid
- might replace 4 by
  $((PBS_NUM_NODES * PBS_NUM_PPN / $tpt))

Hello from thread 0 out of 4 from process 2 out of 4 on jj01c61
Hello from thread 3 out of 4 from process 2 out of 4 on jj01c61
Hello from thread 1 out of 4 from process 2 out of 4 on jj01c61
Hello from thread 0 out of 4 from process 3 out of 4 on jj01c61
Hello from thread 2 out of 4 from process 3 out of 4 on jj01c61
Hello from thread 3 out of 4 from process 3 out of 4 on jj01c61
Hello from thread 1 out of 4 from process 3 out of 4 on jj01c61
Hello from thread 0 out of 4 from process 0 out of 4 on jj01c62
Hello from thread 3 out of 4 from process 0 out of 4 on jj01c62
Hello from thread 1 out of 4 from process 0 out of 4 on jj01c62
Hello from thread 2 out of 4 from process 0 out of 4 on jj01c62
Hello from thread 3 out of 4 from process 1 out of 4 on jj01c62
Hello from thread 0 out of 4 from process 1 out of 4 on jj01c62
Hello from thread 2 out of 4 from process 1 out of 4 on jj01c62
Hello from thread 1 out of 4 from process 1 out of 4 on jj01c62
Hello from thread 2 out of 4 from process 2 out of 4 on jj01c61
Manipulating process placement

- psiadmin -c 'show cpumap <node>'
  - 3 0 1 2 3 4 5 6 ... 52 53 54 55
- Manipulate via __PSI_CPUMAP="C_1-C_i,C_j-C_k,C_1-C_n"
- __PSI_CPUMAP="0,4,8,12,..." mpiexec -np 4 ./testcore
  - Hello world from process 0 of 4, mask 00000001 on j3c068
  - Hello world from process 1 of 4, mask 00000010 on j3c068
  - Hello world from process 2 of 4, mask 00000100 on j3c068
  - Hello world from process 3 of 4, mask 00001000 on j3c068

- Large memory
- few tasks
- Try your application
Pinning

- ParaStation process pinning:
  - Avoid task switching
  - Make better use of CPU cache
  - normally (SMT),
    - threads 0-3, 8-11 -> CPU0
    - threads 4-7, 12-15 -> CPU1
  - Tell ParaStation not to pin itself
    - `__PSI_NO_PINPROC=1 __PSI_NO_MEMBIND=1 srun` ...
  - Allow the Intel OpenMP library thread placing
    - `export KMP_AFFINITY=[verbose,]...`
      - compact: place threads as close as possible
      - scatter: as evenly as possible
  - Export to every task
    - `mpiexec --exports=OMP_NUM_THREADS,KMP_AFFINITY`
  - GCC: `GOMP_CPU_AFFINITY` (manual)
Printing accounting information

- **PSMOM_LOG_ACCOUNT**: switch accounting on
- **PSMOM_DIS_ACCOUNT**: switch accounting off
- **Places accounting information into stderr of job**:
  ```
  -- job accounting information --
  exit_code  0
  mem        827768kb
  vmem       924396kb
  cput       00:07:09
  walltime   00:04:59
  ```
- **Needs to be passed to the job**
  - `msub ... -v PSMOM_LOG_ACCOUNT`
  - `#MOAB -v PSMOM_LOG_ACCOUNT`
- **requires job durations above ~60 seconds**
Conclusions

▪ You now should be able to
  ▪ compile
  ▪ run your application
  ▪ diagnose and fix specific errors
  ▪ know where to turn to in case of problems
Thank you!