JURECA – Tuning for the platform

Usage of ParaStation MPI

2016-05-23

Frank Robel
Outline

▪ ParaStation MPI
▪ Compiling your program
▪ Running your program
▪ Tuning parameters
▪ Resources
ParaStation MPI

▪ Based on MPICH
  ▪ supports all MPICH tools (tracing, debugging, …)
▪ Proven to scale up to 3,000 nodes and 85,000 procs per job
  ▪ JuRoPA running ParaStation MPI: 0.274 PFLOPS (2009)
  ▪ JURECA running ParaStation MPI: 1.42 PFLOPS (2015)
▪ Supports a wide range of interconnects, even in parallel
  ▪ e.g. InfiniBand EDR on JURECA cluster in Jülich
ParaStation MPI

- MPI libraries for several compilers
  - especially for GCC and Intel
- Recently added features include:
  - Improved scalability
  - Improved InfiniBand bandwidth performance
  - Improved (dynamic) process management
  - MPI-3 compliance (e.g. non-blocking collectives)
ParaStation History

- 1995: University project (→ University of Karlsruhe)
- 2004: Open source (→ ParaStation Consortium)
- 2004: Cooperation with JSC
  - various precursor clusters
  - JUDGE
  - DEEP Cluster Booster
  - JuRoPA2 (J2)
  - JuRoPA3 (J3)
  - JUAMS
  - JURECA
Recent Versions

▪ JURECA
  ▪ ParaStation MPI → psmpi-5.1.5-1 (MPI-3)
  ▪ Intel Compilers → v 16.0.2
  ▪ Gnu gcc → v 5.3.0
Compiling on JURECA

- Only MPI-3 version (5.1.5-1) available
- `module load GCC|Intel`
  - `module avail 2>&1 | grep ParaStation`
  - `ParaStationMPI/5.1.5-1[−mt]`
  - `single thread task/multi-thread tasks (mt)`
  - `mt currently only available for Intel`
- ChangeLog available with
  - `less $(dirname $(which mpicc))/../ChangeLog`
- Gnu and Intel compilers available
  - `gcc-5.3.0 (GCC)`
  - `intel-2016.0.2 (Intel)`
- see also the previous talk about Toolchains / Lmod usage...
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
  - allows to tweak compiler
- When using the “mt” version (and using OpenMP), add
  - `–fopenmp (gcc)`
  - `–qopenmp (intel)`
Wrapper vs. Manual Compilation

- Intel C-Compiler + ParaStation MPI
  - `module load Intel`
  - `module load ParaStationMPI/5.1.5-1`
  - `mpicc -show`

```c
```
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 libraries

  e.g. `ldd hello_mpi_default`:
  - `linux-vdso.so.1 => (0x00007fff1bf75000)`
  - `libmpi.so.12 => /usr/local/software/jureca/Stages/2016a/software/psmpi/5.1.5-1-iccifort-2016.2.181-GCC-4.9.3-2.25-mt/lib/libmpi.so.12 (0x00002b34eb287000)`
  - ...

Wrong Module !?!
(e.g. if application is not multi-threaded)
JURECA: start via srun

**WARNING:** DO NOT USE `mpiexec`  
- Unexpected results may happen!  
- Use `srun` instead!

- `srun -N <nodes> -n <tasks>` spawns the tasks, either:  
  - **interactively via `salloc` (to be called previously)**  
  - **from batch job (inside an `sbatch` script)**

- Full environment is exported via `srun`  
- 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;
}
Running on JURECA

- module load Intel
- module load ParaStation/5.1.5-1
- mpicc -O3 -o hello_mpi hello_mpi.c

Interactive:
- salloc -N 2 # get an allocation
- srun -n 2 ./hello_mpi

  - Hello world from process 0 of 2 on jrc0491
  - Hello world from process 1 of 2 on jrc0492

Batch:
- sbatch ./hello_mpi.sh
Process Placement

- ParaStation process pinning:
  - Avoid task switching
  - Make better use of CPU cache
- JURECA is pinning by default:
  - So `--cpu_bind=rank` may be omitted
- Manipulate pinning:
  - e.g. for “large memory / few task” applications
  - Manipulate via `--cpu_bin=mask_cpu:<mask1>,<mask2>,...`
    - CPU masks are always interpreted as hexadecimal values
- For example on JURECA:
  
  ```bash
  srun --cpu_bind=[verbose,]mask_cpu:0x1,0x1000
       -n 2 ./testcore
  ```
```c
#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 %02d out of %d from process %d out of %d on %s\n",
               iam, np, rank, numprocs, processor_name);
    }

    MPI_Finalize();
}
```

Example:
2 Nodes, 2x2 Procs, 2x2x12 Threads

Node x
- P0
- P1

Node y
- P2
- P3
On JURECA

- module load Intel
- module load ParaStationMPI/5.1.5-1-mt
- mpicc -O3 -qopenmp -o hello_hybrid hello_hybrid.c
- salloc -N 2 --cpus-per-task=12
- export OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK}
- srun -n 4 ./hello_hybrid

Hello from thread 00 out of 12 from process 0 out of 4 on jrc0491
Hello from thread 01 out of 12 from process 0 out of 4 on jrc0491
Hello from thread 02 out of 12 from process 0 out of 4 on jrc0491
Hello from thread 03 out of 12 from process 0 out of 4 on jrc0491
.
.
.
Hello from thread 09 out of 12 from process 3 out of 4 on jrc0492
Hello from thread 10 out of 12 from process 3 out of 4 on jrc0492
Hello from thread 11 out of 12 from process 3 out of 4 on jrc0492
Pinning: Which core for a thread?

- **JURECA:**
  - 2 Sockets, 12 Cores per Socket
  - 2 HW-Threads per Core
  - → 48 Threads possible

- Normally (SMT):
  - Threads 0-11, 24-35 → CPU0
  - Threads 12-23, 36-47 → CPU1

```
<table>
<thead>
<tr>
<th>Node</th>
<th>Socket 0</th>
<th></th>
<th>Socket 1</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Core 0</td>
<td>Core 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>HWT 0</td>
<td>HWT 1</td>
<td>...</td>
<td>HWT 12</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Core 10</td>
<td>Core 13</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>HWT 10</td>
<td>HWT 13</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Core 11</td>
<td>Core 22</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>HWT 22</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Core 2</td>
<td>Core 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>HWT 24</td>
<td>HWT 25</td>
<td>...</td>
<td>HWT 23</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Core 10</td>
<td>Core 13</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>HWT 34</td>
<td>HWT 37</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Core 11</td>
<td>Core 22</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>HWT 46</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>HWT 47</td>
</tr>
</tbody>
</table>
```
Pinning: Which core for a thread?

- No thread pinning by default on JURECA

- Allow the Intel OpenMP library thread placing
  - `export KMP_AFFINITY=[verbose,]...`
    - **compact**: place threads as close as possible
    - **scatter**: as evenly as possible

- Full environment is exported via `srun` on JURECA

- For GCC: set `GOMP_CPU_AFFINITY` (see manual)
PMI Barrier Timeout

- If you see ...
  - *PSIlogger: Timeout: Not all clients joined the first pmi barrier:*
    
    \[
    \text{joined}=36765 \quad \text{left}=99 \quad \text{round}=10
    \]

- Typically in large applications (-n > 20,000)
- ... you should **not** increase the timeout ...
  - *The timeout is calculated from the requested job size*
- ... but it is possible to increase the number of rounds
  - *export PMI_BARRIER_ROUND=<N>*
  - default is 10

- **Max job size:** 256 x 48 = 12,288
  - you should not see any PMI Barrier Timeout
Large Job Considerations

- Every MPI process talks to all others:
  - \((N-1) \times 0.55 \text{ MB communication buffer space per process!}\)

- Example 1 on JURECA:
  - \(\text{max job size } 256 \times 48 = 12,288 \text{ processes}\)
  - \(12,288 \times 0.55 \text{ MB} \rightarrow \sim 6758 \text{ MB / process}\)
  - \(\times 48 \text{ process / node} \rightarrow \sim 317 \text{ GB communication buffer space}\)
  - \(\text{But there are only } 128 \text{ GB main memory per node}\)

- Example 2 on JURECA:
  - \(\text{Max job size with only one process per core}\)
  - \((128 \times 24 - 1) \times 0.55 \text{ MB} \times 24 \rightarrow \sim 40 \text{ GB / node}\)
  - \(\text{Main memory – OS – GPFS – } 40 \text{ GB} \rightarrow \sim 60 \text{ GB}\)
  - \(60 \text{ GB / 24 procs} \rightarrow 2.5 \text{ GB left for each process}\)
On Demand / Buffer Size

Two possible solutions:

1. Create buffers on demand only:
   - `export PSP_ONDEMAND=[0|1]`

2. Reduce the buffer queue length:
   - *(Default queue length is 16)*
   - `export PSP_OPENIB_SENDQ_SIZE=3`
   - `export PSP_OPENIB_RECVQ_SIZE=3`
   - *Do not go below 3, deadlocks might occur!*
   - *Trade-off: Performance penalty*
     - *small queue lengths may lead to low bandwidths*
On-Demand / Queue Size Guidelines

- On-Demand works best with nearest neighbor communications
  - (Halo) Exchange
  - Scatter/Gather
  - All-reduce
  - ...

- But for All-to-all communication:
  - queue size modification only viable option...
NUMA Considerations

- Non Uniform Memory Access (NUMA)
NUMA Policies

- If memory is bound to processes, only local memory is accessible → and can get exhausted:
  - `srun -n 1 --mem_bind=rank/local ./blockmem_mpi`
    
srun: error: jrc0075: task 0: Killed
    srun: Force Terminated job step 1505858.15

- If memory is not bound to processes, all memory is accessible:
  - `srun -n 1 --mem_bind=none ./blocknen_mpi`

- On JURECA is `--mem_bind=none` used by default so it can be omitted

- But: membind off → data is crossing CPUs (NUMA) → ~15–20% performance drop!

- Use the First-Touch Policy
Resources

- www.parastation.com
- www.fz-juelich.de/ias/jsc/jureca
- /opt/parastation/doc/pdf
- by mail: support@par-tec.com
- by mail: sc@fz-juelich.de
- 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
Summary

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