JUWELS & JURECA
Tuning for the platform

Usage of ParaStation MPI

2018-11-22
Outline

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

- Based on MPICH (3.2)
  - supports all MPICH tools (tracing, debugging, …)
- Proven to scale up to 3,000 nodes and 85,000 procs per job
  - JURECA running ParaStation MPI: 1.42 PFLOPS (2015)
  - JURECA & Booster ParaStation MPI: 3.78 PFLOPS (2017)
  - JUWELS running ParaStation MPI: 6.18 PFLOPS (2018)
- Supports a wide range of interconnects, even in parallel
  - InfiniBand EDR on JURECA Cluster and JUWELS
  - OmniPath on JURECA Booster
  - Extoll on DEEP projects research systems
- Tight integration with Cluster Management (healthcheck)
ParaStation MPI

- MPI libraries for several compilers
  - especially for GCC and Intel
- Recently added features include:
  - Support for modular jobs
  - Improved Omni-Path performance
  - Improved scalability
  - Improved InfiniBand bandwidth performance
  - Improved (dynamic) process management
ParaStation History

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

- **JURECA**
  - ParaStation MPI → psmpi-5.2.1-1 *(MPI-3.1)*
  - Intel Compilers → v 19.0.0.117
  - Gnu gcc → v 8.2.0

- **JUWELS**
  - ParaStation MPI → psmpi-5.2.1-1 *(MPI-3.1)*
  - Intel Compilers → v 19.0.0.117
  - Gnu gcc → v 8.2.0
Compiling on JURECA

- Currently MPI-3.1 version (5.2.1-1) available
- single thread tasks
  - `module load Intel ParaStationMPI`
  - `module load GCC ParaStationMPI`
- multi-thread tasks (mt)
  - `module load Intel ParaStationMPI/5.2.1-1-mt`
  - no multi-thread GCC version available
- ChangeLog available with
  - `less $(dirname $(which mpicc))/../ChangeLog`
- Gnu and Intel compilers available
  - `gcc-8.2.0 (GCC)`
  - `intel-2019.0.117 (Intel)`
- see also the previous talk JURECA Cluster and Booster
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 ParaStationMPI`
  - `mpicc -show`

```bash
```
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

```bash
ldd hello_mpi:
...
libmpi.so.12 => /usr/local/software/jureca/Stages/2018b/software/psmpi/5.2.1-1-iccifort-2019.0.117-GCC-7.3.0/lib/
libmpi.so.12 (0x00002aba171df000)
...

vs.

libmpi.so.12 => /usr/local/software/jureca/Stages/2018b/software/psmpi/5.2.1-1-iccifort-2019.0.117-GCC-7.3.0-mt/lib/
libmpi.so.12 (0x00002b3523fd4000)
```
JURECA: start via srun

- Use `srun` to start MPI processes
- `srun -N <nodes> -n <tasks>` spawns task
  - directly
  - interactively via `salloc`
  - from batch script via `sbatch`
- Exports full environment
- Stop interactive run with (consecutive) `^C`
  - passed to all tasks
- No manual clean-up needed
- You can log into nodes which have an allocation/running job step
  - `squeue -u <user>`
  - `sgoto <jobid> <nodenumber>`
  - e.g. `sgoto 2691804 0`
- Do not use `mpiexec`
/* 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
", rank, numprocs, processor_name);
    MPI_Finalize ();
    return 0;
}
Running on JURECA (Intel chain)

- module load Intel
- module load ParaStationMPI
- 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

Increase verbosity:
  - PSP_DEBUG=[1,2,3,...] srun -n 2 ./hello_mpi
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:
  - srun --cpu_bind=[verbose,]mask_cpu:0x1,0x1000
    -n 2 ./testcore
  - rank 0 running on core 0
  - rank 1 running on core 12
#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();
}
On JURECA

- module load Intel ParaStationMPI/5.2.1-1-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>Socket 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core 0</td>
<td>Core 1</td>
<td>...</td>
</tr>
<tr>
<td>HWT 0</td>
<td>HWT 1</td>
<td>...</td>
</tr>
<tr>
<td>HWT 24</td>
<td>HWT 25</td>
<td>...</td>
</tr>
<tr>
<td>Core 12</td>
<td>Core 13</td>
<td>...</td>
</tr>
<tr>
<td>HWT 12</td>
<td>HWT 13</td>
<td>...</td>
</tr>
<tr>
<td>HWT 36</td>
<td>HWT 37</td>
<td>...</td>
</tr>
</tbody>
</table>

"Package"
Pinning: Which core for a thread?

- **JUWELS:**
  - 2 Sockets, 24 Cores per Socket
  - 2 HW-Threads per Core
  - → 96 Threads possible

- Normally (SMT):
  - Threads 0-23, 48-71 → CPU0
  - Threads 23-47, 72-95 → CPU1

<table>
<thead>
<tr>
<th>Node</th>
<th>Socket 0</th>
<th></th>
<th>Socket 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Core 0</td>
<td>Core 1</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>HWT 0</td>
<td>HWT 1</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>HWT 48</td>
<td>HWT 49</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>HWT 24</td>
<td>HWT 25</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>HWT 72</td>
<td>HWT 73</td>
<td>...</td>
</tr>
</tbody>
</table>

"Package"
Pinning: Which core for a thread?

- No thread pinning by default on JURECA and JUWELS
- Allow the Intel OpenMP library thread placing
  - `export KMP_AFFINITY=[verbose,modifier,]...`
  - `compact`: place threads as close as possible
  - `scatter`: as evenly as possible
  - `KMP_AFFINITY=granularity=fine,verbose,scatter srun ...
    OMP: Info #171: KMP_AFFINITY: OS proc 0 maps to package 0 core 0
    OMP: Info #242: KMP_AFFINITY: pid 4940 thread 1 bound to OS proc set {1}
- Full environment is exported via `srun` on JURECA and JUWELS
- For GCC: set `GOMP_CPU_AFFINITY` (see manual)
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:
  - \textit{job size} \(256 \times 48 = 12,288\) processes
  - \(12,288 \times 0.55 \text{ MB} \rightarrow \sim 6,758\) MB / process
  - \(\times 48\) process / node \(\rightarrow \sim 317\) GB communication buffer space
  - \textit{But there is only 128 GB of main memory per node}

- Example 2 on JUWELS:
  - \textit{job size} \(256 \times 96 = 24,576\) processes
  - \(24,576 \times 0.55 \text{ MB} \rightarrow \sim 13,517\) MB / process
  - \(\times 96\) process / node \(\rightarrow \sim 1,267\) GB mpi buffer space
  - \textit{But there is only 96 GB of main memory per node}
On Demand / Buffer Size

Two possible solutions:

1. Create buffers on demand only:
   - `export PSP_ONDEMAND=1`
   - Activated on JUWELS by default!

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
     (sending many small messages)
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...

- Example
  - rank 0: for ( ; ; ) MPI_Send()
  - rank 1: for ( ; ; ) MPI_Recv()
  - PSP_OPENIB_SENDQ/RECVQ_SIZE=4: 1.8 seconds
  - PSP_OPENIB_SENDQ/RECVQ_SIZE=16: 0.6 seconds
  - PSP_OPENIB_SENDQ/RECVQ_SIZE=64: 0.5 seconds
NUMA Considerations

- Non Uniform Memory Access (NUMA)
NUMA Policies

- If memory is bound to processes, only local memory is accessible → and can get exhausted (at about 55 GB):
  - `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 ./blockmem_mpi`

- On JURECA and JUWELS is `--mem_bind=none` used by default so it can be omitted
- But: membind off → data is crossing CPUs (NUMA) → ~15–20% performance drop!
- First-Touch Policy: Memory is allocated locally
Resources

- www.parastation.com
- www.fz-juelich.de/iasjscjureca
- /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!