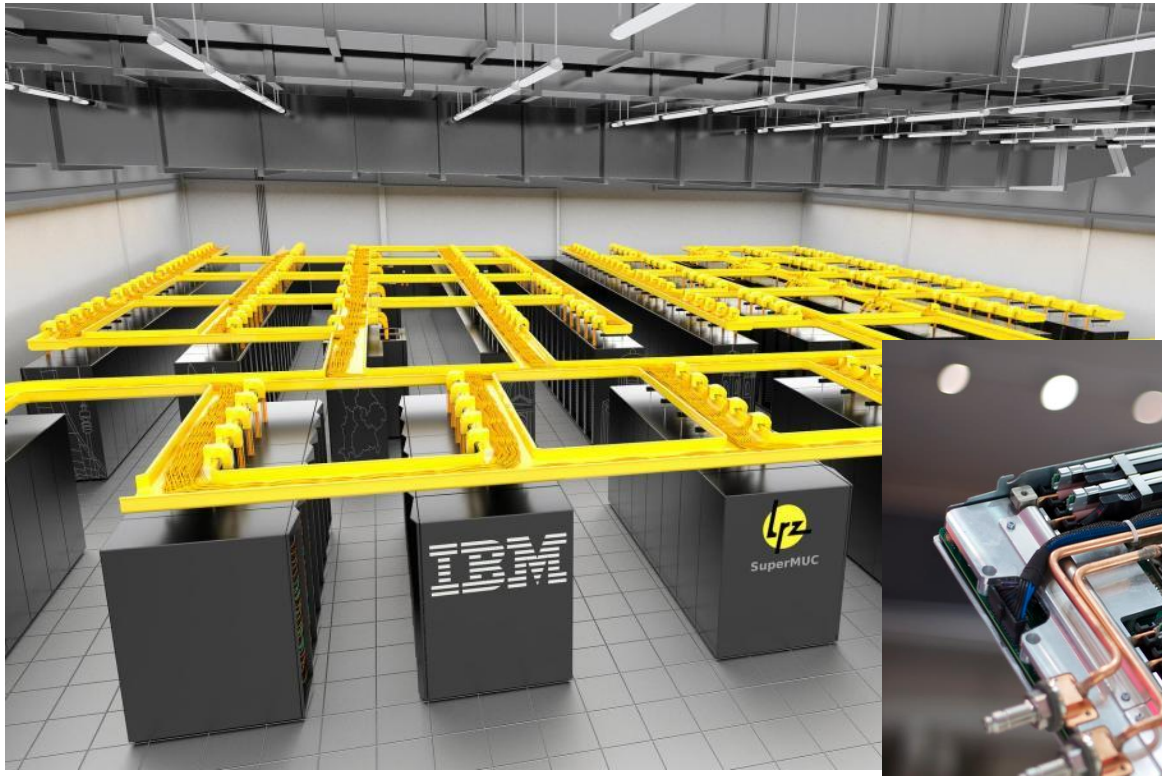
Programming of Supercomputers

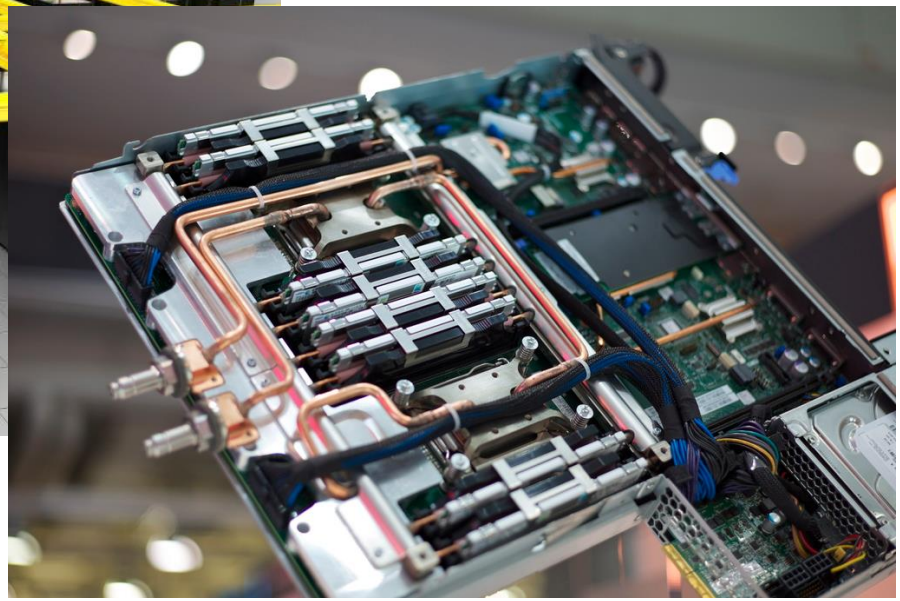
Assignment 1:
Single Node Performance

Prof. Michael Gerndt

Isaias A. Compres Urena

SuperMUC





Introduction

Supercomputing:

- Performance as one of the main goals
- Large amount of parallelism in hardware
  - Multiple sockets
    - Multiple cores
    - SIMD
  - Accelerators
- Large number of nodes aggregated
  - Distributed memory
- Multiple socket nodes and high core counts with SIMD
- ***Specialized networks***
  - Low latency
  - High bandwidth
  - Complex topologies

Course Format

Assignments:

- 4 assignments
- 1 Final summary video

Grading (groups of 2 or 3 students):

- Each assignment and final video will be given a 0-100 grade
  - Linear conversion to German grading
    - 40 : 4,0
    - >93 : 1,0
- These grades will be weighted as follows:
  - Assignment 1: 15%
  - Assignment 2: 15%
  - Assignment 3: 25%
  - Assignment 4: 30%
  - Video: : 15%

Login to SuperMUC, Documentation

- First change the standard password
  - https://idportal.lrz.de/r/entry.pl
- Login via
  - lxhalle due to restriction on connecting machines
  - ssh <userid>@supermuc.lrz.de
  - No outgoing connections allowed
- Documentation
  - http://www.lrz.de/services/compute/supermuc/
  - http://www.lrz.de/services/compute/supermuc/loadleveler/
  - Intel compiler: http://software.intel.com/sites/products/documentation/hpc/composerxe/en-us/2011Update/cpp/lin/index.htm

Building the Benchmark

- Load the required modules:

```
module unload mpi.ibm
module load mpi.intel
```

- Update your Makefile (refer to the provided instructions)
  - Baseline
  - OpenMP
  - MPI
  - MPI + OpenMP
- Build and verify that the binaries were created
- Run the benchmark in the login node
- Identify your **performance metric** from the output!
  - More is better or less is better?
  - Check the benchmark's documentation online

Batch Scripts

- Advantages
  - ***Reproducible performance***
  - Run larger and longer running jobs

- Several job classes available
  - ***Test (recommended for this assignment‘s tasks)***
    - Phase 1:
      - Max 1 island, 32 nodes, 30 minutes, 1 job in queue
    - Phase 2:
      - Max 1 island, 20 nodes, 30 minutes, 1 job in queue
  - Micro
    - Phase 1:
      - Max 1 island, 32 nodes, 48 hours, 8 jobs in queue
    - Phase 2:
      - Max 1 island, 20 nodes, 48 hours, 8 jobs in queue

Submitting a Batch Job

- llsubmit ll.sh
  - Submission to batch system
- llq –u $USER
  - Check status of own jobs
- llcancel <jobid>
  - Kill job if no longer needed
  - Obtain the <jobid> from the llq output

```
#!/bin/bash
#@ wall_clock_limit = 00:20:00
#@ job_name = pos-lulesh-openmp
#@ job_type = MPICH
#@ class = micro
#@ output =
pos_lulesh_openmp_$(jobid).out
#@ error =
pos_lulesh_openmp_$(jobid).out
#@ node = 1
#@ total_tasks = 16
#@ node_usage = not_shared
#@ energy_policy_tag = lulesh
#@ minimize_time_to_solution = yes
#@ island_count = 1
#@ queue

. /etc/profile
. /etc/profile.d/modules.sh

export OMP_NUM_THREADS=16
./lulesh2.
```

Use CPU hours responsibly

- Specify job execution as tight as possible
  - In this assignment, 10 minutes is sufficient
- Only request the number of nodes required.
  - 1 node is sufficient for all tasks in assignment 1.
- Small tests can be done in the login node
  - Create a batch only after you are ready to collect results
  - Running in a batch eliminates interference from other users.
- All types of runs can be tested in the login node
  - Baseline
  - MPI,
  - OpenMP and
  - Hybrid

Assignment 1: Single Node Performance

Single-thread Performance

- GPROF
  - Flat profile
  - Call graph
- Compiler Flags
  - GCC
  - ICC
- Optimization Pragmas
  - GCC pragmas
  - ICC pragmas

Multi-thread Performance

- OpenMP
  - Single process
  - Scaling with threads
    - Shared address space
    - Direct load and stores
    - Coherency, locks, etc.
- MPI
  - Multiple processes
  - Only certain process counts valid
  - Scaling with processes
    - Separate address spaces
    - Messages
- MPI + OpenMP