Scientific Computing RTP

WARWICK

Part 2 : Batch processing and HPC

`warwick.ac.uk/scrtp`

Prof. David Quigley
Director, Centre and RTP for Scientific Computing

Shared hardware

- Funded centrally by the university, not individual groups/departments
- Subject to resource restrictions
  - –Maximum runtime (currently 48 hours)
  - –Maximum number of CPUs, RAM etc per user
- Restrictions are in place to prevent monopolization by single user
- Good codes support checkpoint & restart
  - e.g. dump parameters at final iteration and restart from these
  - e.g. store state of simulation and read on restart

Cluster of Workstations (CoW)

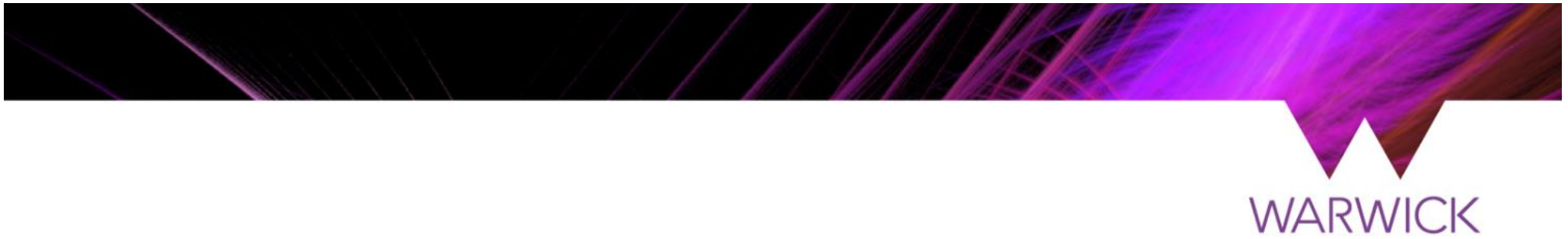
warwick.ac.uk/scrtp/desktop/cow/

The Cluster of Workstations (CoW)

- System for batch processing of calculations
- Mostly suitable for serial, low I/O computations
  - –Generally available `taskfarm` queue (shared)
  - –Some research groups have their own servers/nodes (not shared)

```
#!/bin/bash
#SBATCH --ntasks=1
#SBATCH --mem=512
#SBATCH --time=48:00:00

module load GCC/7.3.0-2.30 OpenMPI/3.1.1 Python/3.6.6

python my_cool_thing.py
```

Submit job script to the CoW, it will run on the next available node in the queue

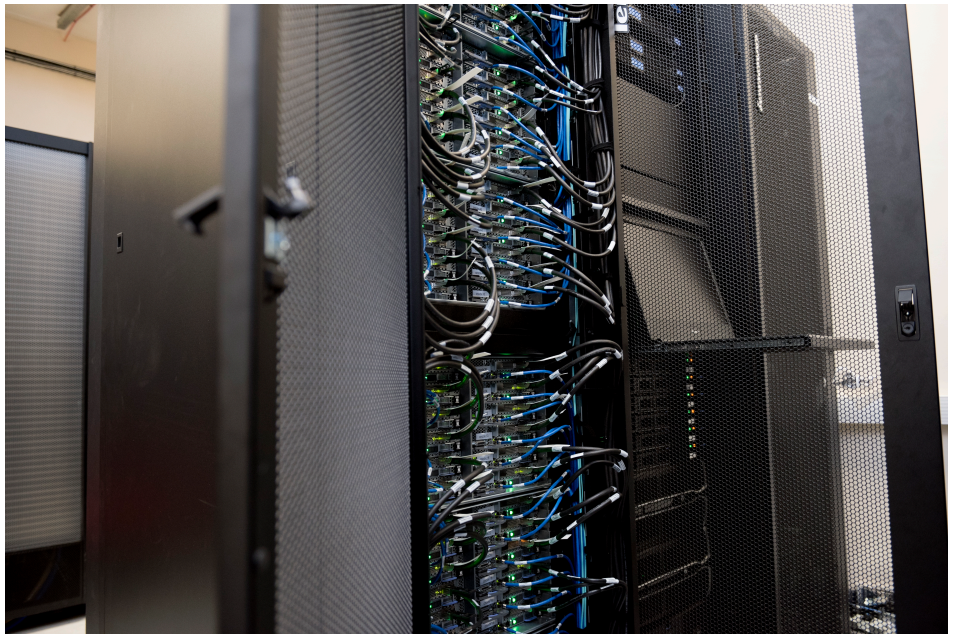


High Performance Computing

warwick.ac.uk/scrtp/hpc/

Orac (2017)

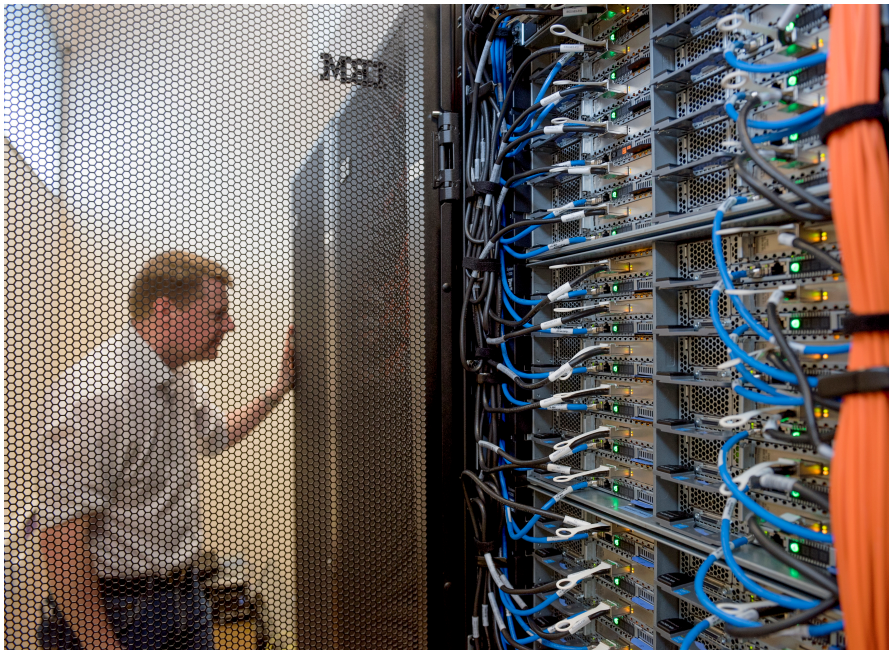
- **Compute nodes:** Lenovo NeXtScale nx360 M5 servers with 2 x Intel Xeon E5-2680 v4 (Broadwell) 2.4 GHz 14-core processors; 28 cores per node; 84 nodes; 2352 cores; 128 GB 2400 MHz DDR4 memory per node
- **Interconnect:** Intel Omni-Path X16, 100 Gbit/s with 1 µs latency



- **Phi node testbed:** Intel Xeon Phi 7250F (Knights Landing) 1.4 GHz 68-core host processors; 4 nodes; 272 cores; 96 GB 2400 MHz DDR4 memory per node
- **OpenPOWER node testbed:** 2 x IBM POWER8 3.259 GHz 8-core processors; 16 cores per node; 1 node; 256 GB DDR4 memory; 4 x NVIDIA P100 GPGPUs (SXM2 NVLink-enabled)

Tinis (2015)

- **Compute nodes:** Lenovo NeXtScale nx360 M5 servers with 2 x Intel Xeon E5-2630 v3 2.4 GHz (Haswell) 8-core processors; 16 cores per node; 203 nodes; 3488 cores; 64 GB DDR4 memory per node / 4 GB per core
- **Interconnect:** QLogic TrueScale InfiniBand, 40GB/s with 1 µs latency



- **GPU nodes:** 8 x NVIDIA Tesla K80 GPU cards; 2 GPU cards per node; 4 GPUs per node; 4 nodes; 64 GB DDR4 memory per node
- **Fat nodes:** 128 x Intel Xeon E7-4809 v3 2.0 GHz Haswell cores; 32 cores per node; 4 nodes; 1 TB DDR3 memory per node; 1 x NVIDIA GRID K2 GPU per node

Parallel code (Python)

- Tinis and Orac are primary for *parallel computing*

```
from mpi4py import MPI
import numpy as np

# Number of points to integrate over
n = 1400

comm = MPI.COMM_WORLD              # Initialise an MPI communicator
my_rank = comm.Get_rank()          # Get the rank of the current process
p = comm.Get_size()                # Get the size of the current communicator

i1 = int(n/p) * my_rank            # Current rank starts at point number i1
i2 = int(n/p) * (my_rank+1).       #     "     "    "     "   "     "    i2

tot = 0.0                          # Every rank computes 4/(1+x^2) at points
for i in range(i1,i2):             # in the segment of [0,1] it has been assigned
    x = (i+0.5)/n
    tot += 4.0/(1.0+x**2)
```

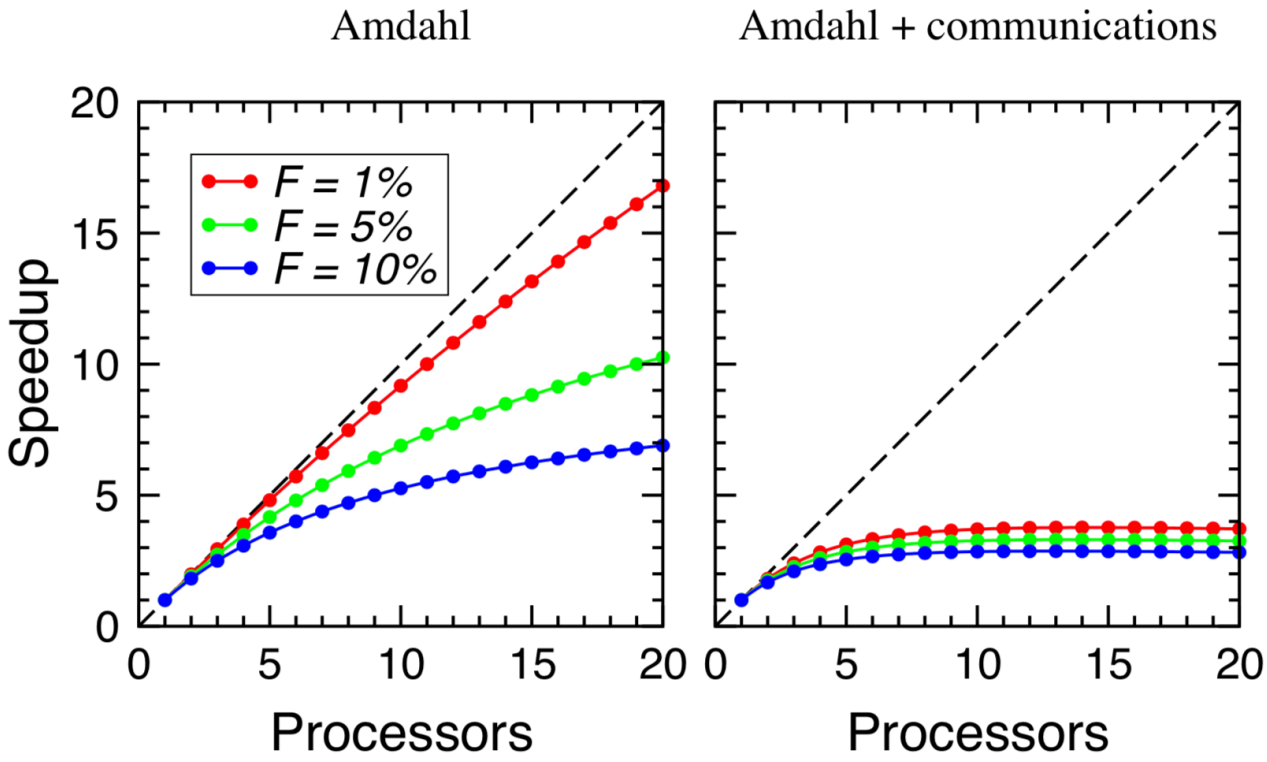
Parallel code (Python)

- Parallel code requires *communications*
- Part of the calculation cannot be done in parallel

```
if ( my_rank != 0 ):                    # Everyone but rank 0 sends to rank 0

    comm.send(tot, dest=0, tag=999)

else:                                   # Rank 0 has to receive p-1 messages
    for partner in range(1,p):
        # Add what rank 0 received into the total
        tot += comm.recv(source=partner, tag=999)

    tot = tot/n                         # This is only done by rank 0 - serial
    print("Final result = %s "%tot)    # and prints result
```

Amdahl's Law

- Consider a code with a *serial fraction* of *F*
- Limits parallel speedup



PX425 – High Performance Computing

- Covers code optimisation and parallel programming in C
- 15 lectures + 5 two-hour workshops starting this week
- OpenMP – multithreaded parallelism for multicore machines
- MPI – message passing parallelism for clusters
- Contact Dr Nick Hine (N.D.M.Hine@Warwick.ac.uk) if wanting to audit or take for credit (7.5 CATS)

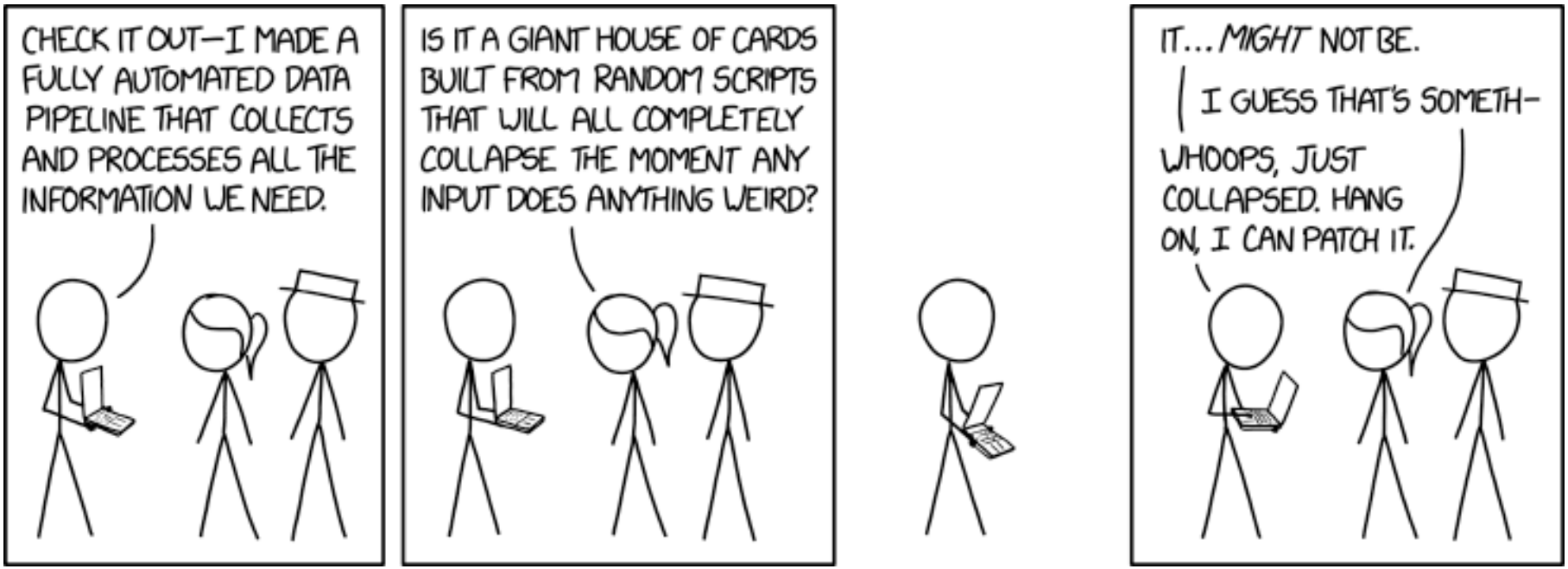
Serial jobs on the HPC clusters

- Individual serial runs should be run on a dedicated machine or submitted to the CoW
- Workflows involving hundreds or even thousands of serial runs *can and should* be submitted to the HPC clusters as a single job

```
#!/bin/bash
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=28
#SBATCH --time=08:00:00
#SBATCH --mem-per-cpu=4571

module load parallel intel/2017.4.196-GCC-6.4.0-2.28 impi/2017.3.196 Python/3.6.6

MY_PARALLEL_OPTS="-N 1 --delay .2 -j $SLURM_NTASKS --joblog parallel-${SLURM_JOBID}.log"
MY_SRUN_OPTS="-N 1 -n 1 --exclusive"
MY_EXEC="python my_code.py {1}"

parallel $MY_PARALLEL_OPTS srun $MY_SRUN_OPTS $MY_EXEC ::: {0..99}
```

Research Software Engineering

- warwick.ac.uk/rse/
- More detailed talk from Chris Brady/Heather Ratclifffe



Take home message

warwick.ac.uk/scrtp