Transitioning from your laptop to CRC

PittCRC
Center for Research Computing

Leonardo Bernasconi

Center for Research Computing and
Department of Chemistry
University of Pittsburgh

07 March 2024

Aims of this workshop

1. What resources CRC provides to Pitt researchers
2. How to request resources on and access the CRC systems
3. How to run jobs efficiently
4. How to get help
5. How to port, profile, and benchmark software
6. How to move data to/from the CRC systems

Why should I use the CRC systems instead of my own PC/laptop or large supercomputing facilities?

How can I optimize my software/computational workflows to make the most efficient use of what CRC offers?

PittCRC
Center for Research Computing

Table of contents

1. The CRC ecosystem
2. How to request compute time: initial allocations and “proposals”
3. Options for accessing CRC resources
4. How to start: useful commands and other things to know
5. Slurm: how to submit jobs
6. Example: from your laptop to smp
7. Serial and parallel computation
8. How to set up your workflows

Our website

https://crc.pitt.edu

MISSION

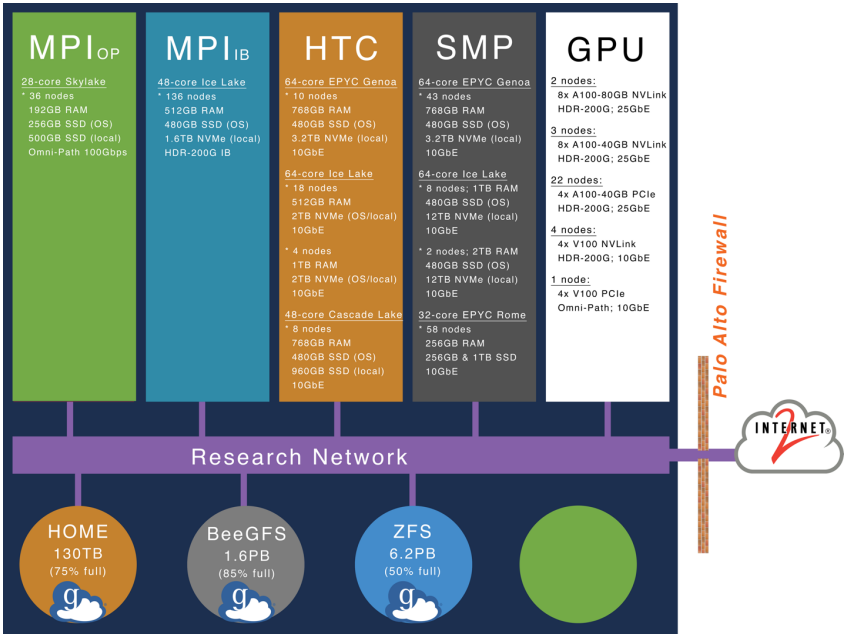
The Center for Research Computing supports leading-edge research with free access to advanced computing hardware and software for fields across the entire research community, along with training and consultation by CRC research faculty. Under the umbrella of Pitt Research, CRC helps shape innovative ideas into reality using methods including simulation, data analysis, image and text analysis, and genomic sequencing analysis. Whether incorporating machine learning, building humanities data resources, or improving computation, CRC helps expand the possible.

PittCRC
Center for Research Computing

1.
The CRC ecosystem

PittCRC
Center for Research Computing

Computing hardware: the four CRC clusters



- Hardware
- Storage
- Software
- People

PittCRC
Center for Research Computing

Computing hardware: the four CRC clusters

https://crc.pitt.edu/resources/computing-hardware

1. MPI
   "Massively parallel" calculations on two or more nodes, e.g., molecular dynamics on large systems, computational fluid dynamics

2. SMP
   Shared-memory calculations, e.g., quantum chemistry

3. HTC
   High-throughput calculations, e.g., gene sequencing

4. GPU
   GPU acceleration, e.g., machine learning

Partitions

PittCRC
Center for Research Computing

Computing hardware: the four CRC clusters

Which cluster should we select for our work?

It depends on the software we plan to use.

If the software has MPI capabilities, the mpi (or smp) clusters are the best choice. Similarly, if the software has been optimized for GPU, the choice is easy.

In general, for software that has neither MPI capabilities nor GPU acceleration, smp or htc are good choices. The software will work exactly as on a PC/laptop, but it will benefit from higher memory availability, multi-core capabilities, and higher execution speed.

https://crc.pitt.edu/resources/computing-hardware

Storage

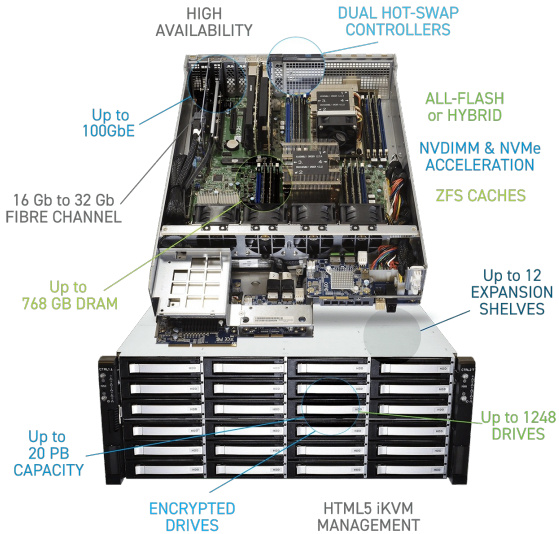
https://crc.pitt.edu/resources/data-storage

Each **user** has 75GB available under ihome, e.g., `/ihome/leb140g/ritu`

In addition, each **group** has 5TB of persistent data storage, available to all users in the group, e.g., `/ix/leb140g`

Additional storage can be requested, at a cost of $65/TB/year from our webform.

In total, currently CRC makes ca. 6PT of persistent storage available to Pitt users.



PittCRC
Center for Research Computing

Software

https://crc-pages.pitt.edu/user-manual/applications/software-list/

CRC maintains over 1200 software packages covering wide varieties of disciplines. The complete list can be displayed using the LDAP command:

```
module spider
```

CRC personnel builds, optimizes, and maintains these packages on the clusters. Some of them are licensed software, which requires special permissions to run, other is open-source or non-licensed software, which can be used freely.

People

https://crc.pitt.edu/about-us/people

CRC provides scientific, as well as technical support, to its users. CRC with specific expertise can be contacted directly or via the ticketing system.

Kim Wong (Physical Chemistry) kimwong@pitt.edu
Leonardo Bernasconi (Quantum Chemistry) leb140@pitt.edu
Nickolas Comeau (Research Computing) nlc60@pitt.edu
Yassin Khalifa (Data Science, GPU Programming) yak73@pitt.edu
Fangping Mu (Bioinformatics, Health Sciences) fangping@pitt.edu
Daniel J. Perrefort (Physics) djperrefort@pitt.edu
Cheng Xiao (Engineering, GPU Programming) chx33@pitt.edu

2.
Requesting resources on the CRC clusters

PittCRC
Center for Research Computing

Service Units (SUs)

https://crc-pages.pitt.edu/user-manual/slurm/service-units/

One service units is *approximately* equal to the resources consumed by one core running for one hour. The exact SU charge of a job depends on:

- which cluster/partition a job is submitted to
- the number of cores requested
- the RAM (memory) the job required
- the number of cards, if the job runs on the GPU cluster

To figure out the SU charge for a given *cluster*, look for the "TRESBillingWeights" in the output of the command

```
scontrol -M cluster show partition
```

where *cluster* = smp, htc, mpi, gpu.

How to request SUs: Initial allocation

Each user receives an initial allocation of 25,000 SUs, which can be used on any of the CRC clusters. This amount is typically sufficient to carry out test runs and estimate the total SUs that a project may require.

CRC provides free user accounts to all Faculty and PIs at Pitt. Faculty/PIs can request their One-time Startup Allocation (25,000 SUs, 75GB /ihome storage, and 5TB ix storage) by submitting a webform.

The Faculty/PIs can add users (students, postdocs, and staff) by submitting a help ticket. Each of these users will receive their initial allocation (25,000 SUs and 75GB /ihome storage) and get access to the PI's ix storage.

A user can share resources with more than one group. The groups to which a *user* belongs can be displayed using:

```
id user
```

Requesting SUs for research (or for teaching)

https://crc.pitt.edu/Pitt-CRC-Allocation-Proposal-Guidelines

CRC relies on “proposals” (or resource requests) to distribute SUs to Pis and track their usage. Proposals can be submitted using a template and a webform, which should contain the following information:

- brief description of scientific background and aims of the project
- estimates of SUs requested on each cluster for one year
- funding sources for the project (if any)
- scientific publications or other research products derived from the project.

A maximum of 3,200,000 SUs can be requested. SU allocation can be renewed (on request) after one year. If more resources are required than initially projected during the one-year time, supplemental allocations can be requested.

PittCRC
Center for Research Computing

3.
How to access the CRC clusters

PittCRC
Center for Research Computing

Accessing the CRC clusters

https://crc.pitt.edu/getting-started/accessing-cluster

The **login nodes** of the h2p or htc clusters can be accessed in a variety of ways. The most common and flexible method makes use of the `ssh` command available in Mac and Linux (or with Xming or Putty in Windows):

```
ssh username@h2p.crc.pitt.edu
```

where username is your Pitt ID.

Login nodes should be used exclusively for editing files and submitting jobs to the **compute nodes** (on the smp, htc, mpi, or gpu clusters) *via* the Slurm scheduler.

Accessing the CRC clusters from an off-campus or wireless connection requires prior connection to the Pitt VPN .

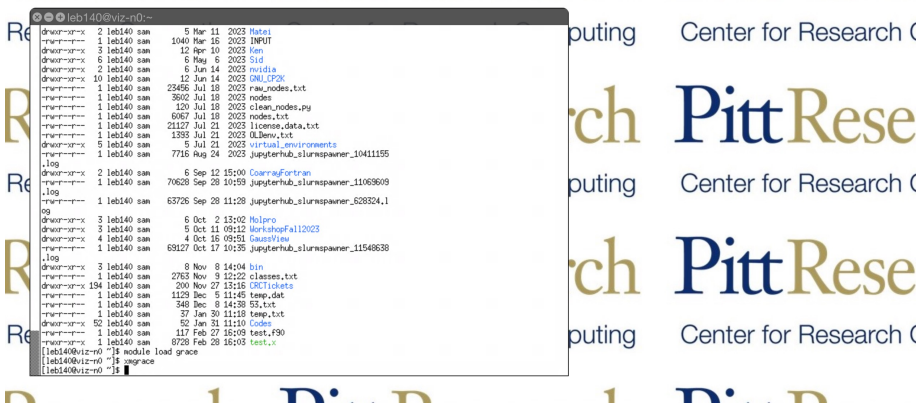
PittCRC
Center for Research Computing

Other ways to access the CRC clusters

https://crc.pitt.edu/getting-started/accessing-cluster

OpenOnDemand allows one to open a terminal and automatically connect to htc, as well as to use Jupyter, RStudio Server, and Matlab.

VIZ can be used to connect open a terminal and automatically connect to a login node. VIZ is particularly useful when we want to use software requiring a GUI.



4.
How to start: useful commands and other things to know

PittCRC
Center for Research Computing

Finding your group(s) and the storage available to you

```
pwd
id user
crc-quota (one of the CRC wrappers)
```

Finding software packages installed with LMOD

```
module spider
module spider package
module load package
module list
module unload package
```

Note some software packages have *dependencies*, i.e., modules that need to be loaded in advance. Sometimes, looking at these dependencies gives clues on the best way to run the software.

PittCRC
Center for Research Computing

Transferring data to/from the clusters

Secure File Transfer Protocol (SFTP)

```
sftp user@h2p.crc.pitt.edu (or user@htc.crc.pitt.edu)
```

Secure Copy Protocol (SCP)

- Copy origin from your local computer to the login nodes:

```
scp origin user@h2p.crc.pitt.edu:/target
```

- Copy origin from the login nodes to your local computer:

```
scp user@h2p.crc.pitt.edu:/origin .
```

PittCRC
Center for Research Computing

Large data transfers

rsync

- Copy a directory from your local computer to the cluster:

```
rsync –azP /Users/leo/MEGA/PyDF/library/
leb140@h2p.crc.pitt.edu:/xhome/crc/leb140/Codes/PyDFx/lib
rary
```

- Copy a directory from the cluster to your local computer:

```
rsync –azP
leb140@h2p.crc.pitt.edu:/xhome/crc/leb140/Codes/PyDFx/lib
rary/ /Users/leo/MEGA/PyDF/library
```

Globus

PittCRC
Center for Research Computing

5.
The Slurm Workload Manager

PittCRC
Center for Research Computing

Overview

Slurm is a job scheduler for computer clusters and large supercomputers. It provides the most important method to access the compute nodes on all CRC clusters, allocate resources, run jobs, and collect their results.

Typically, calculations are submitted to Slurm using batch jobs, which contain specific directives instructing Slurm on the details of the calculations to be performed.

Slurm automatically check availability of resources, dispatch the job to available node(s) on a cluster, and control the execution of the calculation.

Users can fine-tune Slurm's behaviour by using suitable directives in the batch job.

Interactive sessions

It is sometimes a good idea to run tests on or benchmark new software before submitting production calculations through Slurm. This can be done by creating a session on one or more nodes with the `srun` command, on which we can then run software without having to go through a Slurm submission.

CRC also provides a specific wrapper (`crc-interactive`) that simplifies setting up interactive sessions on the cluster.

Important: Running jobs using Slurm or an interactive session are the only two ways to perform calculations on the CRC clusters. Running jobs directly on the login nodes is strictly forbidden!

Useful Slurm commands

The `sinfo` command (and its CRC wrapper `crc-sinfo`) provide an overview of the node availability on all CRC clusters and in the partitions within the clusters. It is useful to figure where to submit a jobs, for example for software that can run on more than one cluster.

Batch scripts (see below) are submitted using the `sbatch` command. The state of a job can be verified using the command `squeue` (or the wrapper `crc-squeue`). See this Table for the meaning of the job state.

We can get more information on pending or allocated jobs using the `scontrol` command:
`scontrol -M cluster show job jobid`

Jobs (pending or allocated) can be cancelled using `scancel` (or `crc-scancel`).

PittCRC
Center for Research Computing

Batch scripts

A minimal example:

```
#!/bin/bash
#SBATCH --job-name=<job_name>
#SBATCH --nodes=<number of nodes>
#SBATCH --ntasks-per-node=<tasks per node>
#SBATCH --cluster=<cluster name>
#SBATCH --partition=<partition>
#SBATCH --time=<days-HH:MM:SS>

program.x < input > output
```

Save this to a file (e.g., job.slurm) and submit using:

```
sbatch job.slurm
```

Batch scripts

Loading modules:

```
#!/bin/bash
#SBATCH --job-name=<job_name>
#SBATCH --nodes=<number of nodes>
#SBATCH --ntasks-per-node=<tasks per node>
#SBATCH --cluster=<cluster name>
#SBATCH --partition=<partition>
#SBATCH --time=<days-HH:MM:SS>

module purge
module load module1 module2

program.x < input > output
```

PittCRC
Center for Research Computing

Batch scripts

Runs with shared memory parallelism:

```
#!/bin/bash
#SBATCH --job-name=<job_name>
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=<tasks per node>
#SBATCH --cluster=<cluster name>
#SBATCH --partition=<partition>
#SBATCH --time=<days-HH:MM:SS>

module purge
module load module1 module2

srun < program.x (with parameters, if any)
```

Batch scripts

Runs with shared message passing interface (MPI, multiple nodes):

```
#!/bin/bash
#SBATCH --job-name=<job_name>
#SBATCH --nodes=4
#SBATCH --ntasks-per-node=<nodes * tasks per node>
#SBATCH --cluster=<cluster name>
#SBATCH --partition=<partition>
#SBATCH --time=<days-HH:MM:SS>

module purge
module load module1 module2

mpirun -np $SLURM_NTASKS program.x
```

Batch scripts

GPU runs:

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --time=0-00:01:00
#SBATCH --ntasks-per-node=<NUMBER OF GPU NODES>
#SBATCH --gres=gpu:<NUMBER OF GPUs PER NODE>
#SBATCH --cluster=gpu
#SBATCH --partition=a100

<USER_SPECIFC COMMAND FOR GPU CODE TO BE EXECUTED>
```

Batch scripts

Monitoring performance with `crc-job-stats`:

```
#!/bin/bash
#SBATCH --job-name=<job_name>
#SBATCH --nodes=4
#SBATCH --ntasks-per-node=<nodes * tasks per node>
#SBATCH --cluster=<cluster name>
#SBATCH --partition=<partition>
#SBATCH --time=<days-HH:MM:SS>

module purge
module load module1 module2

mpirun -np $SLURM_NTASKS program.x

crc-job-stats
```

PittCRC
Center for Research Computing

Batch scripts

Using the scratch space:

```
#!/bin/env bash
#SBATCH --job-name=cp2k
#SBATCH --output=output.out
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=28
#SBATCH --time=0-48:00:00
#SBATCH --cluster=mpi
#SBATCH --error=run.err

module purge
module load intel/2019.4 intel-mpi/2019.4 cp2k/7.1

files=(ammonia.inp NH3.xyz)
for i in ${files[@]}; do
    sbcast $SLURM_SUBMIT_DIR/$i $SLURM_SCRATCH/$i
done

run_on_exit(){
    cp -R $SLURM_SCRATCH/* $SLURM_SUBMIT_DIR
    pkill --uid=$SLURM_JOB_USER cp2k.popt
}
trap run_on_exit EXIT

cd $SLURM_SCRATCH
mpirun -np $SLURM_NTASKS cp2k.popt -i ammonia.inp
wait

crc-job-stats
```

PittCRC
Center for Research Computing

Batch scripts

Using the scratch space:

```
#!/bin/env bash
#SBATCH --job-name=cp2k
#SBATCH --output=output.out
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=28
#SBATCH --time=0-48:00:00
#SBATCH --cluster=mpi
#SBATCH --error=run.err

module purge
module load intel/2019.4 intel-mpi/2019.4 cp2k/7.1

files=(ammonia.inp NH3.xyz)
for i in ${files[@]}; do
    sbcast $SLURM_SUBMIT_DIR/$i $SLURM_SCRATCH/$i
done

run_on_exit(){
    cp -R $SLURM_SCRATCH/* $SLURM_SUBMIT_DIR
    pkill --uid=$SLURM_JOB_USER cp2k.popt
}
trap run_on_exit EXIT

cd $SLURM_SCRATCH
mpirun -np $SLURM_NTASKS cp2k.popt -i ammonia.inp
wait

crc-job-stats
```

While a job is running, it is possible to ssh to the node(s) on which it is running and monitor files in the scratch space. These are located in `/scratch/jobID`.

PittCRC
Center for Research Computing

Batch scripts

Submitting multiple jobs (job arrays):

```
#!/bin/bash
#SBATCH -N 1
#SBATCH --time=0-01:00:00
#SBATCH -J testJA
#SBATCH --output=test-%A\_%a.out
#SBATCH --array=1-5 # job array index
#SBATCH --cpus-per-task=1
#SBATCH --cluster=smp

echo ${SLURM_ARRAY_TASK_ID}
program.x (parameters)
```

Job arrays offer, in some cases, a simple way to parallelize a code.

Batch script example: ANSYS

```
#!/bin/bash
#SBATCH --job-name=job
#SBATCH --output=fluent.o%j
#SBATCH --error=fluent.e%j
#SBATCH --job-name="ansys"
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=64
#SBATCH --cluster=smp
#SBATCH --time=1:12:00

# Load Modules
module purge
module load ansys

echo  $SLURM_NTASKS

fluent 3ddp  -i fluent_test.jou -gu  -t$SLURM_NTASKS -driver null
```

PittCRC
Center for Research Computing

Batch script example: AMBER MD on gpu

```
#!/bin/bash
#SBATCH --job-name=gpus-2
#SBATCH --output=gpus-2.out
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=2
#SBATCH --cluster=gpu
#SBATCH --partition=v100
#SBATCH --gres=gpu:2
#SBATCH --time=24:00:00

# Load Modules
module purge
module load intel/2017.3.196
module load amber/18

# Amber input files and output name
INP=md.in
TOP=mocvnhlysm.top
CRD=mocvnhlysm.crd
OUT=mocvnhlysm

# Executable
SANDER=pmemd.cuda.MPI

# Launch PMEMD.CUDA
echo AMBERHOME    $AMBERHOME
echo SLURM_NTASKS $SLURM_NTASKS
nvidia-smi

mpirun -n $SLURM_NTASKS \
          $SANDER  -O     -i   $INP   -p   $TOP   -c   $CRD   -r   $OUT.rst \
                          -o   $OUT.out   -e   $OUT.ene   -v   $OUT.vel   -inf $OUT.nfo   -x   $OUT.mdcrd
```

PittCRC
Center for Research Computing

Batch script example: CP2K on mpi using Singularity

```
#!/bin/env bash
#SBATCH --job-name=dcp2k
#SBATCH --output=output.out
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=48
#SBATCH --time=0-24:00:00
#SBATCH --cluster=mpi
#SBATCH --error=run.err
#SBATCH --partition=mpi


module purge
module load singularity/3.9.6


# mpich
module load gcc/5.4.0
module load mpich/3.1

export FI_PROVIDER=tcp
export I_MPI_OFI_PROVIDER_DUMP=enable


export OMP_NUM_THREADS=1
mpirun -np 96 singularity run ../cp2k.sif cp2k -i H2O-32.inp


wait
crc-job-stats.py
```

PittCRC
Center for Research Computing

Batch scripts

We provide a small collection of sample Slurm scripts under

`/ihome/crc/how_to_run/`

Often, it is possible to modify easily existing scripts to make them work for new software.

For more difficult cases you can always submit a help ticket (or contact us by email), and we will help setting your script for the CRC clusters.

6.
Example: from your laptop to smp

A simple Python program

A code for computing prime numbers within a given range:

`/xhome/crc/leb140/WorkshopMarch2024/prime_numbers.py`

It needs an input file like

`/xhome/crc/leb140/WorkshopMarch2024/input.inp`

Which contains the first and last integer number in the desired range.

A simple Python program

1) Copy the Python file and the input file to your local computer using `scp` or `sftp`

2) Run it using:

```
python prime_numbers.py input.inp
or
python3 prime_numbers.py input.inp
```

3) Note the execution time, which is printed out

4) Now run the same task on smp, by submitting it to Slurm using the sample script available in this directory:

`/xhome/crc/leb140/WorkshopMarch2024/job1.slurm`

Try running by requesting 1, 2, or more CPUs.
What do you notice?

PittCRC
Center for Research Computing

“Parallelization” using job arrays

We can use a Slurm job array to submit two or more instances of `prime_numbers.py`. Each instance uses a different input file, defining a different range, for instance: 2-10000, 10001-20000, etc.

All instances will work simultaneously, and each instance will produce its own output, with its own part of the prime numbers between 2 and 50000.

An example of how this can be done is available here:

`/xhome/crc/leb140/WorkshopMarch2024/JobArray/job1.slurm`

“Parallelization” using job arrays

/xhome/crc/leb140/WorkshopMarch2024/JobArray/job1.slurm

```
!/bin/bash
#SBATCH -N 1
#SBATCH --time=0-01:00:00
#SBATCH -J testJA
#SBATCH --output=test-%A\_%a.out
#SBATCH --array=1-2 # job array index
#SBATCH --cpus-per-task=1
#SBATCH --cluster=smp


echo "Reading input.inp_"${SLURM_ARRAY_TASK_ID}

python prime_numbers.py "input.inp_"${SLURM_ARRAY_TASK_ID}
```

PittCRC
Center for Research Computing

7.
Serial and parallel computation

Thanks to Dr. Fangping Mu

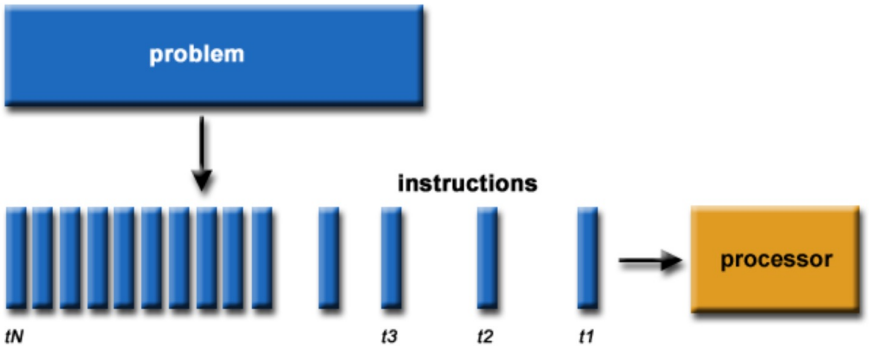
PittCRC
Center for Research Computing

Serial calculations

Traditionally, software has been written for serial computers, often with only one CPU (or processor).

A problem is coded as a series of instructions, compiled to machine language, and dispatched to the CPU.

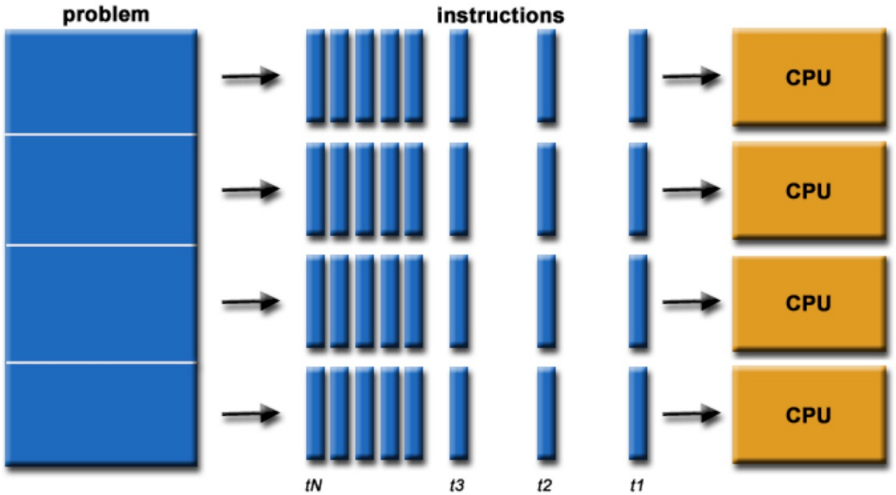
Only one instruction can be executed at a given time.



Multicore calculations

In its simplest sense, parallel computing is the simultaneous use of more than CPU to solve a problem.

The problem is broken into discrete parts that can be solved concurrently. Instructions from each part execute simultaneously on differenc CPUs.



PittCRC
Center for Research Computing

Parallelization approaches

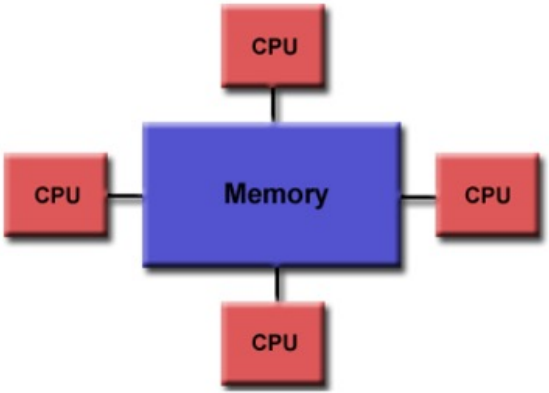
In addition to job arrays, we have:

1. Shared memory parallelization
2. Distributed memory parallelization
3. Hybrid shared-memory/distributed parallelization

2. and 3. distribute the workload over different (and, potentially, many) nodes, each of which contains several CPUs.

Unlike job arrays, 1.-3. require changes in the source code, compared to the serial code. 1. is usually considered the easiest way to parallelize a code. 2.-3. can lead to much higher efficiency (massive parallelization) but they require extensive, and often complex, code rewriting.

Shared memory parallelization (smp and htc)



Calculations are carried out on single node, but the workload is broken into parts that are executed by different CPUs. Each of these processes (or threads) has access to the same memory space of all the other processes.

What is the difference between shared memory parallelization and job arrays?

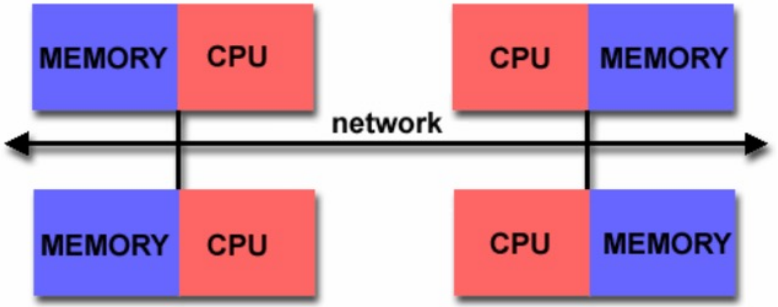
Shared memory parallelization (smp and htc)

Many codes are parallelized using this approach. If you have a serial code that you would like to make parallel, shared memory is probably the first approach to consider.

For instance, it is usually quite straightforward to use packages like `multiprocessing` to parallelize code in Python. For software written in traditional HPC programming languages, like C or Fortran, specific libraries are available to enable shared memory parallelization, like Pthreads and OpenMP.

Running shared-memory applications is relatively easy: we just need to instruct Slurm on how many threads we want to run, how many CPUs are used for each thread, and, potentially, how much memory in total the program requires.

Distributed memory parallelization (mpi)



The workload is distributed among different tasks, each of which executes its own instructions independently from all the others. Each task has its own private portion of data in memory, which is not visible to other tasks.

Tasks can run on multiples nodes (using multiple CPUs on each node), which must be connected using a fast a fast network, or fabric, e.g., InfiniBand (IB) or Omni-Path Architecture (OPA).

Rapid communication among tasks is made possible by the Message Passing Interface (MPI) libraries.

Distributed memory parallelization (mpi)

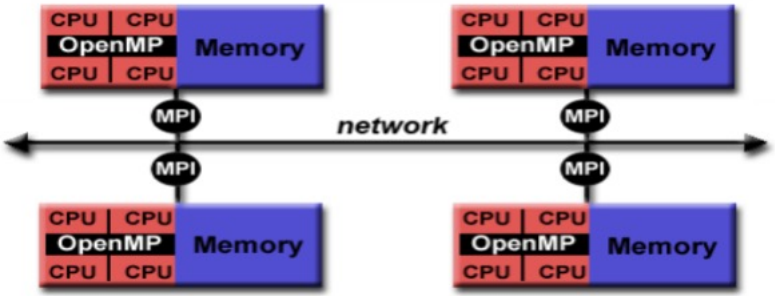
MPI codes typically target massively-parallel HPC calculations. There is a substantial effort required to make distributed memory software work efficiently, both at the programming level and during the compilation of the code.

Running MPI applications requires specifying the number of nodes on which the program is running and how many MPI tasks are going to run on each node. Typically (but nor necessarily) one CPU is used for each task.

Increasing the number of MPI tasks does not always increase performance. Larger numbers of tasks will carry out their individual operations in less time, but the communications between tasks may decrease efficiency.

It is always wise to test the execution speed versus the number of MPI tasks (see example later).

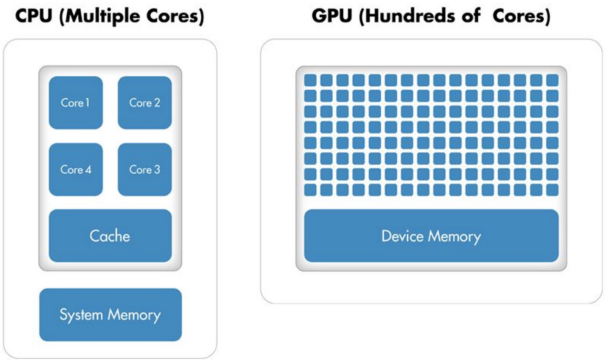
Hybrid distributed-shared memory parallelization (mpi)



This is a very powerful approach, which exploits the advantages of multi-core shared-memory and multiple node communication. Very few codes implement this approach successfully (an example is the quantum-chemistry code CP2K).

Typically, it is quite complex to figure out how many threads should be used for a given number of nodes to obtain efficiency superior to pure MPI. Although powerful, this approach requires substantial investment of time in preparing the calculations.

GPU “parallelism” (gpu)



Graphical processing units (GPUs) have many more cores that CPUs. Initially develop to accelerate graphical applications, they has found extensive use also in other fields, e.g., machine learning, classical molecular dynamics, and cryptocurrency mining.

They are ideal for embarrassingly parallel problems, in which little or no effort is required to break the problem into separate tasks.

GPU “parallelism” (gpu)

Modifying existing code to exploit GPU acceleration requires a skill-set that can be difficult to acquire (e.g., CUDA programming). Furthermore, code must be fine tuned and optimized for the specific type of GPU you are using.

However, if GPU versions of specific packages are installed, or available, it is often a good idea to assess their efficiency compared to their CPU counterparts.

GPU experts at CRC

Yassin Khalifa (Data Science, GPU Programming) yak73@pitt.edu
Cheng Xiao (Engineering, GPU Programming) chx33@pitt.edu

PittCRC
Center for Research Computing

8.
Setting up your workflows

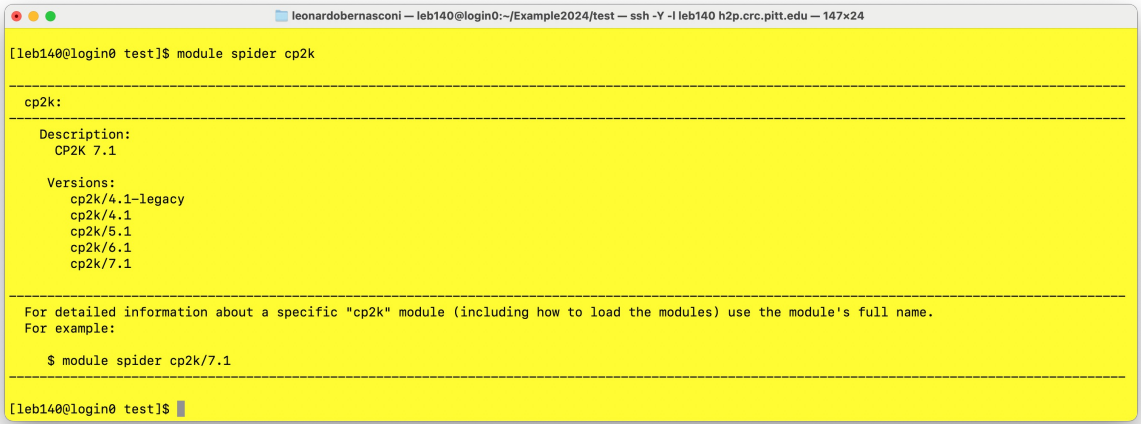
Thanks to Dr. Fangping Mu, Dr. Cheng Xiao

PittCRC
Center for Research Computing

Building your workflow on the CRC clusters

1. Is the software that I need available at CRC?

The `module spider` *<software>* commands provides this information. It also shows which versions of the software are installed.



If it is not installed, you can request installation by submitting a help ticket (or try installing it yourself).

Pitt CRC
Center for Research Computing

Building your workflow on the CRC clusters

2. On which cluster(s) will the software work best?

Serial and shared-memory parallel software will work well on smp or htc. MPI software is ideal for the mpi cluster, but it can also run on smp. GPU software will work well on the gpu cluster.

How to find out?
`module spider <software>` gives this information in most cases (e.g., cp2k, tensorflow, pytorch, amber, etc.)

Software that does not depend on MPI or CUDA libraries likely works well on smp and htc.

Building your workflow on the CRC clusters

3. How can I know if smp/htc software runs in parallel?

Looking at the dependencies of the software can give an idea of whether the software is shared-memory parallelized.

```
[fangping@login0b ~]$ cd /ihome/crc/install/star/STAR-2.7.9a/bin/Linux_x86_64
[fangping@login0b Linux_x86_64]$ ldd STAR
        linux-vdso.so.1 =>  (0x00007fffa6b61000)
        libz.so.1 => /lib64/libz.so.1 (0x00007f95eca57000)
        libstdc++.so.6 => /lib64/libstdc++.so.6 (0x00007f95ec750000)
        libm.so.6 => /lib64/libm.so.6 (0x00007f95ec44e000)
        libgomp.so.1 => /lib64/libgomp.so.1 (0x00007f95ec228000)
        libgcc_s.so.1 => /lib64/libgcc_s.so.1 (0x00007f95ec012000)
        libpthread.so.0 => /lib64/libpthread.so.0 (0x00007f95ebdf6000)
        libc.so.6 => /lib64/libc.so.6 (0x00007f95eba28000)
        /lib64/ld-linux-x86-64.so.2 (0x00007f95ecc6d000)
```

libgomp is the GNU OpenMP library

Building your workflow on the CRC clusters

4. How do I set up my Slurm script(s)?

The best way is to open an interactive session (see crc-interactive) on the cluster you want to use, and test the software, with no need to submit it to Slurm. Once the correct command(s) and parameters to launch the software have been identified, you can use the sample scripts given in /ihome/crc/how_to_run to create your template.

Of course, you can always contact us if you need help at this stage.

Building your workflow on the CRC clusters

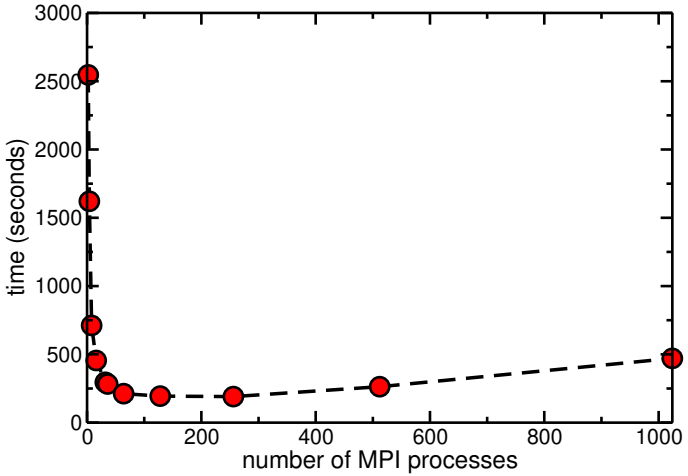
5. How do I figure out the optimal resources to use?

Scaling tests

```
#!/bin/env bash
#SBATCH --job-name=v22n
#SBATCH --output=output.out
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=48
#SBATCH --time=0-24:00:00
#SBATCH --cluster=mpi
#SBATCH --error=run.err
#SBATCH --partition=mpi

module load …
<execute command>
crc-job-stats
```

CP2K 6.1 – AIMD, 64 water molecules



Information about code performance is also important for requesting additional SUs with proposals.

PittCRC
Center for Research Computing

9.
Summary

PittCRC
Center for Research Computing

Should I transition to CRC from my laptop?

Most likely, yes.

CRC provides state-of-the-art hardware, a rich and growing library of pre-installed and optimized software, domain-specific expertise, and dedicated support to Pitt users.

The initial free SU allocation is generally more than sufficient to test software, create workflows, and estimate how much more computer time we may need to complete a project.

Getting help on any aspect of creating a workflow, from installing and testing software to more specific issues about its usage is easy: either submit a help ticket or contact the CRC consultants at any time. We will be very happy to help.

crc.pitt.edu

PittCRC
Center for Research Computing