Introduction to CSC Computing Environment

Running your jobs

Compiling and resource management

Outline

Part 1: Compiling programs

Part 2: Using Makefiles

Part 3: Resource mangement

Part 1

Compiling programs

Compiling programs

- Multiple compiler environments are available
  - Portland Group Inc. (PGI),
  - Pathscale,
  - GNU,
  - INTEL.
- To change the environment, for example, from PGI to GNU:

```
module switch PrgEnv-pgi PrgEnv-gnu
```

Manual pages of compilers

- Environment dependent manual information is available.

```
man [pgcc | pgCC | pgf95]
man [craycc | crayCC | crayftn]
man [gcc | g++ | gfortran]
man [icc | icpc | ifort]
```

Compiling and linking MPI programs

- Message Passin Interface (MPI)
  - Communication protocol used to program parallel computers.
- Use always wrappers.
  - Same wrapper for all compiler suites.
  - Include most of the relevant libraries.

Language	Compiler (Louhi)	Compiler (Vuori)	File suffixes
Fortran 77	f77	mpif77	.f, .F
Fortran 90/95	ftn	mpif90	.f, .F, .f90, .F90, .f95, .F95
C	cc	mpicc	.c, .i
C++	CC	mpiCC	.C, .cc, .ii

Simple compiling example

```
pgcc -o hello hello.c
```

```
#include <stdio.h>
int main (argc, argv)
     int argc;
     char *argv[];

{
  int rank, size, i,j,k,maxval,onepercent;
  double r;
  maxval = 2000000000;
  onepercent = maxval/100;
  rank = 0;
  size = 1;

  for (i=0,j=0,k=0,r=0.0;i<maxval;++i,++j){
    r =+ 0.1234567*((double) i*(rank+1))/((double)
   size);
    if (j==onepercent) {
      j=0;
      k++;
      if (rank == 0) printf( "%d percent
   processed\n", k);
    }
  }
  printf( "Hello world: %8.4e\n", rank+1, rank,
   size, r);
  return 0;
}
```

Compiler switches

`-c` Compiles only. Produces unlinked object filename.o.

`-o` Filename assigns to the filename of the executable.

`-g` Produces symbolic debug information

`-O[`*`n`*`]` Specifies whether to optimize or not and at which level *n*.

Code optimization

- Compiler switch `-O[n]` controls the optimization of the compilation.
  - *n* = {1,2,3,4}
  - Levels 3 and 4 should be used with caution, because they may change the results.
- Example

```
pgcc -O2 -o hello hello.c
```

Using libraries

- Do not re-invent the wheel, use libraries!

```
mpif90 -L~/lib -llibname
```

  - uses library *libname* and adds *~/lib* to library search path

```
mpicc -lm
```

  - math library *math.h*

- Compiler switches for libraries.

`-I`*dirname*	Searches directory *dirname* for include files or module files
`-L`*dirname*	Searches directory *dirname* for for library files specified by `-l`
`-l`*libname*	Searches the specified library file with the name *liblibname.a*

Part 2

Using Makefiles

Makefiles

- Makefiles will help to organize the compilation of the code.
- Some benefits, when using Makefiles:
  - Simplifies the test/modify/debug cycle,
  - Helps to organize the project files,
  - Build commands of the project can be found from one file.
- Command make executes the compilation commands as they have been written in the Makefile.

Makefiles: A project example

- The project consists of three files.
  - main.cpp
  - hello.cpp
  - factorial.cpp
- The name of the executable is hello.
- Should be compiled on Vuori.

```
All:   main.cpp hello.cpp factorial.cpp
       PgCC -fastsse -tp barcelona-64 -Mipa=fast
       main.cpp hello.cpp factorial.cpp -o hello
```

Makefiles: A project example

```
CC           = pgCC
CFLAGS       = -c -fastsse barcelona-64 -Mipa=fast
LDFLAGS      =
SOURCES      = main.cpp hello.cpp factorial.cpp
OBJECTS      = $(SOURCES:.cpp=.o)
EXECUTABLE   = hello

All:            $(SOURCES) $(EXECUTABLE)

$(EXECUTABLE):  $(OBJECTS)
                $(CC) $(LDFLAGS) $(OBJECTS) -o $@

.cpp.o:
                $(CC) $(CFLAGS) $< -o $@

clean:
                rm -rf *o hello
```

Part 3

Resource management

Batch jobs

- Why batch job system?
  - Usually the demand of resources is higher than the supply.
  - Optimizing the load of machines.
  - Optimal experience to the user.
- A batch job system is always a compromize between the points above.

Batch jobs

- How do the resource managers work?
  - Different queues (for long, short, large and interactive jobs).
  - Optimizes the way, how free resource slots are filled with requests.
  - Tools for communication
    - Monitoring runs,
    - Output files,
    - Job status manipulation.

CSC server Vuori: SLURM

- Simple Linux Utility for Resource Management (SLURM).
  - Open source solution for larger Linux clusters of all sizes
- Two types of jobs:
  - Interactive
  - Batch
- Allocate a serial job:

```
salloc -p interactive -n 1 -t 02:00:00
salloc: Granted job allocation job_id
```

Allocates one processor for two hours

Resources are allocated

- Run the job

```
srun ./my_serial_executable
```

CSC server Vuori: Parallel interactive jobs

- Parallel interactive job:
  - Simply replace the -*n* 1 with -*n* N, where N is the number of cores, e.g.

    ```
    salloc -p interactive -n 6 -t 02:00:00 --mem-per-cpu=1000
    ```
  - Run the job the same way as before
- Alternatively – all-in-one (serial job):

  ```
  salloc -p interactive -n 1 -t 02:00:00 srun ./my_serial_executable
  ```

CSC server Vuori: Serial batch jobs

- All directives for SLURM start with #SBATCH

`-J`	job name
`-e`	stderr
`-o`	stdio
`%j`	adds the job id

- A serial or parallel job is submitted by using sbatch.

```
sbatch my_job_script
```

```
#!/bin/sh
#SBATCH -J my_jobname
#SBATCH -e my_output_err_%j
#SBATCH -o my_output_%j
#SBATCH --mem-per-cpu=1000
#SBATCH -t 01:01:00
#SBATCH -n 1
 ./my_serial_program
```

CSC server Vuori: Parallel batch jobs

- Each node has two six core CPU's.
- Use multiple of six for parallel runs.
- `--ntasks-per-node` influences the number of used nodes (communication).

```
#!/bin/sh
#SBATCH -J my_jobname
#SBATCH -e my_output_err_%j
#SBATCH -o my_output_%j
#SBATCH --mem-per-cpu=1000
#SBATCH -t 11:01:00
#SBATCH -n 24
#SBATCH -ntasks-per-node=12
#SBATCH -p parallel
srun ./my_mpi_program
```

CSC server Vuori: Batch job handling

- Status of the jobs

```
sinfo -all
```

- Submitting jobs

```
sbatch my_job_script
```

- Monitoring jobs (displays job_id)

```
squeue [-u userid]
```

- Deleting jobs

```
scancel job_id
```

CSC server Louhi: PBS

- Jobs are submitted to PBS.
  - Only one type of job: parallel batch jobs.
  - Interactive only for debugger session.
- CNL (Compute Node Linux) on compute nodes → cross- platform-compilation.
- More info about PBS:

```
man pbs
```

CSC server Louhi: Parallel batch jobs

`-N`	job name
`-j oe`	combined output
`-l`	resource option
`-n`	option for `aprun`, corresponds to `mppwidth`
`-m`	e-mail notification
`-r n`	cannot be re-run

```
#!/bin/sh
#PBS -N my_jobname
#PBS -j oe
#PBS -l walltime=1:00:00
#PBS -l mppwidth=256
#PBS -m e
#PBS -M user1@univ2.fi
#PBS -r n
cd $PBS_O_WORKDIR
aprun -n 256 my_prog
```

CSC server Louhi: Batch job handling

- Queue status

```
qstat -Q
```

- Submitting jobs (`-h` displays job_id):

```
qsub [-h] my_job_script.sh
```

- Monitoring jobs

```
qstat [-u user] [-a job_id]
```

- Deleting jobs:

```
qdel job_id
```

Machine guides

- Hippu
  `http://www.csc.fi/english/pages/hippu_guide/index.html`
- Louhi
  `http://www.csc.fi/english/pages/louhi_guide/index.html`
- Vuori
  `http://www.csc.fi/english/pages/vuori_guide/index.html`

Other interesting material

- Data services guide

  `http://www.csc.fi/english/pages/data-services`

- Linux basics

  `http://www.csc.fi/english/pages/data-services/linux_basics`

- Software and databases

  `http://www.csc.fi/english/research/software/index_html`