

PROGRAMMING MODEL EXAMPLES

DEMONSTRATION EXAMPLES OF VARIOUS PROGRAMMING MODELS

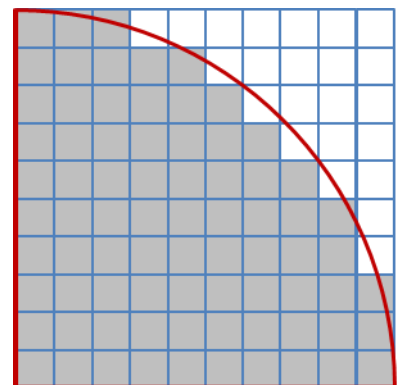
OVERVIEW

Building an application to use multiple processors (cores, cpus, nodes) can be done in various ways and in particular the High Performance Computing (HPC) community has a long experience in using multithreaded and multi-process programming models to develop fast scalable applications on a range of hardware platforms. Typical APIs in use are OpenMP (source threading markup) and MPI (message-passing). The examples provided and described here can be used to compare these models when used to tackle a very simple problem and as a platform to learn how to build and launch parallel applications on the HPE Cray EX supercomputer.

The demonstrator problem was chosen for its simplicity and lack of subtle issues across the range of programming models.

THE SAMPLE PROBLEM AND ALGORITHM

The example we have chosen is to calculate π by piecewise approximation of the area of a circle. You could do this by drawing a quarter circle on graph paper and counting the number of squares inside and outside the circle. This approximates the area within the circle and hence π can be determined. A more complicated example might be to find the area/volume of a fractal structure; this would be much more computationally intensive and analytically complex. The algorithm covers the quarter circle with an N by N grid and uses a test in the centre to determine if the grid square is inside the circle (the error should average out somewhat). Each program loops over values of i and j indices to get the coordinates of the centre of each small square within the much larger unit square. The approximate value for π is then 4 times the number of squares within the quarter circle divided by the total number of squares. (We can of course do this is a much better way but not if we want to generalise to something like a fractal shape.)



CODE EXAMPLES PROVIDED

The algorithm previously mentioned was coded using various programming models. You will need to unpack `ProgrammingModelExamples_SLURM.tar` or access a location/file provided to you. Once unpacked, you will find a directory with a README, sample job scripts and subdirectories C and Fortran which contain source for multiple versions of the pi program. The following versions are available:

Programming Model	Source File
Basic Fortran	pi_serial.f90
Fortran using parallel language features	pi_coarray.f90
Basic Fortran with OpenMP threading	pi_openmp.f90
Fortran using MPI	pi_mpi.f90
Fortran using MPI and OpenMP (hybrid)	pi_mpi_openmp.f90
Fortran using SHMEM	pi_shmem.f90
Fortran using OpenACC	pi_openacc.f90
Fortran using OpenMP offload	pi_openmp_device.f90
Basic C	pi_serial.c
C with pthreads	Pi_pthreads.c
C with OpenMP	pi_openmp.c
C with MPI	pi_mpi.c
C with MPI and OpenMP	pi_mpi_openmp.c
C with SHMEM	pi_shmem.c
C with OpenSHMEM	pi_openshmem.c
UPC	pi_upc.c
C with OpenMP offload	pi_openmp_device.c
C++ with HIP	pi_hip.cpp
C++ using coarray implementation (CCE, no longer supported)	pi_coarray.cpp

Note that the pthreads version uses `OMP_NUM_THREADS` to pick up the number of threads to use so that it can be run from a job set up to run an OpenMP application.

Note that *no attempt has been made to optimize these examples* as the goal was to provide the simplest possible code to illustrate the programming models. In some cases idioms are used that illustrate a feature of the programming model even if you would not normally code that way (for example the SHMEM version uses atomic update instead of reduction, the HIP version uses atomics instead of implementing on-gpu reduction)

In each language directory (Fortran and C) is a Makefile that will build most of the examples with the Cray CCE compiler. The SHMEM and accelerator examples need specific modules to be loaded before they will compile. In addition, a more complex version of the source is supplied (Fortran_timing and C_timing) with timer code added and the variable `nmax` defined externally, this is more useful for scaling studies as the internal timing is the elapsed time of the computation only. If you are using a system with accelerators then the `acc` makefile target can be used to build examples to run on the accelerators.

Job scripts are provided in the jobscripts directory which you can use as templates to create your own scripts. You may need to change the scripts to be appropriate your environment. These scripts are...

JOB SCRIPT	SCENARIO
pi.slurm	Serial run (use also for OpenACC)
pi_threads.slurm	Multithreaded run (OMP_NUM_THREADS)
pi_tasks.slurm	Multi process run (MPI,coarrays,SHMEM)
pi_hybrid.slurm	Hybrid (for example MPI/OpenMP)
pi_hybrid.slurm	For accelerator examples

In the following sections there is more information concerning the programming environment which you can use if you need it. There is also some very basic information about the programming models covered.

Have a look over those sections or alternatively just dive in or work as suggested by your instructor or guided by your own interests.

You might wish to consider the following activities:

- Decide on your area of interest
- Learn how to compile and run some of the examples.
You can choose serial, threaded, many processes or hybrid.
- Compare the source code to get a feel for each programming model although note that these examples only use the most basic features available.
- Look at scalability as you increase the number of processes or threads.
Does the code run 2 times faster as you double the resources available?
Does it matter how large (or small) a problem size you pick?

BASIC NOTES ON THE PROGRAMMING MODELS

Various programming models are covered by the examples. If you do not have much background in High Performance Computing these could seem daunting. The models are all attempting to provide the HPC programmer with a platform to decompose some problem efficiently onto complex distributed hardware that can range from a multi-core processor to thousands of complex nodes in a supercomputer.

MULTITHREADED MODELS

The model familiar to a C Systems programmer would be pthreads but this does not have much traction with the HPC community due to the requirement to explicitly manage thread creation and requirement for C. More prevalent is the approach to add compiler directives that advise the compiler how to target computation to a set of threads. OpenMP is the example we choose and only one OpenMP directive/pragma is shown which parallelizes the outer I loop.

COOPERATING PROCESS MODELS

These form a family where normally the same binary is run multiple times encapsulated as a Linux process. The model allows the programmer to organize computation by providing a way to determine how many processes are executing and by enumerating each process instance. There will also be a mechanism to communicate data between processes (message-passing) either point-to-point operations or collective operations (for example a sum over all processes). Traditional message-passing APIs are two-sided (a *send* on one process is matched by a *receive* on the other), it is also possible to have single-sided communication where only one process is involved in a *put* or *get* operations without explicit involvement of the other process.

Examples here are MPI and SHMEM

If you look at the examples, they follow the same pattern. The API is used to determine the number of processes/ranks/PEs (npes,size) and which one is executing (mype,rank). The API is also used to coordinate the processes (barriers and synchronization) and to add up the contribution to the final count from each process.

PGAS MODELS

These models are more complex and are typically languages that support access to local and remote data directly (this is why we put SHMEM in the previous section because it is an explicit API).

Examples are Fortran Coarrays, UPC and Chapel.

Fortran coarrays is the simplest, UPC allows data distribution across “threads” and provides special syntax to operate collectively on distributed (or shared) variables and Chapel is the most featured with advanced support a range of parallel programming paradigms.

MANIPULATING YOUR PROGRAMMING ENVIRONMENT

The Cray Programming Environment is designed to simplify the process of building parallel applications for the HPE Cray EX. It supports multiple different underlying tool chains (compilers) from different vendors (CCE, GNU, AOCC, AMD(gpu) and Intel).

Only one on the available tool chains, or Programming Environments (PrgEnvs), can be active at any one time; to see the list of currently loaded modules type:

```
module list
```

A list will be displayed on screen, one of which will be of the form PrgEnv-*, e.g.

```
20) PrgEnv-cray/5.2.40
```

To see which other modules are available to load run the command

```
module avail
```

On most systems this may be a very long list, to be more selective run:

```
module avail PrgEnv
```

which will list all available modules that start with PrgEnv.

To swap from the current PrgEnv to another PrgEnv (e.g. to select a different tool chain), run:

```
module swap PrgEnv-cray PrgEnv-gnu
```

This will unload the Cray PrgEnv and replace it with the Gnu PrgEnv.

NOTE: for a short time there was a version of the Programming Environment that used Environment Modules 4, if that is the case for the system you have access to then the relevant command to view programming environments is 'module savelist' and the command to load a new programming environment is 'module restore PrgEnv-xxx')

COMPILING AN APPLICATION

To build applications to run on the HPE Cray EX users should always use the CCE compiler driver wrappers, `ftn` (Fortran), `cc` (C) and `CC` (C++) to compile their applications. These wrappers interface with the Cray Programming Environment and make sure the correct libraries and flags are supplied to the underlying library.

When compiling an application make sure that the appropriate modules are loaded before starting to compile, then modify the build system to call the appropriate wrapper scripts in place of the direct compilers.

If you wish to try other compilers you may need to add compiler flags, for example to enable OpenMP.

SUBMITTING JOBS TO THE BATCH SYSTEM

Nearly all system will run some form of batch scheduling system to fairly manage the nodes of the system between all the users. Therefore, users must submit all their work in the form of batch job scripts for the scheduler to run in the future.

Example scripts for the SLURM batch scheduler are provided in the example directory,

```
./jobscripts/
```

They are called

```
pi.slurm          - Runs a program in serial on a compute node
pi_tasks.slurm    - Runs a non-threaded program on multiple PEs
pi_threads.slurm  - Runs a threaded program on one PE with threads
pi_hybrid.slurm   - Run a hybrid program on a mixture of PEs and threads
```

You will need to edit these scripts to choose what to run and provide any site specific options such as partition, account, qos and reservation information. You may be provided with a script which you can source to set these defaults for any jobs or application runs.

These scripts include special SLURM comments which are used to provide information to the batch system about the job resource requirements, e.g.

```
#SBATCH -J pi
```

specifies the name of the job (displayed in queues etc).

```
#SBATCH --exclusive
```

Makes sure that a whole node is allocated (likely to be the default)

```
#SBATCH --nodes=
#SBATCH --ntasks=
```

specifies the number of nodes and/or tasks required.

```
#SBATCH --time=0:05:00
```

specifies the length of time the job will take (no longer than five minutes).

Some systems, like ARCHER2 or LUMI, will require additional accounting (or partition/qos/reservation) information:

```
#SBATCH -A y11
```

Jobs are then submitted for execution using the sbatch command:

```
sbatch pi.slurm
```

which will return a job id that uniquely identifies the job. Any of the #SBATCH settings contained within a file can be overridden at submission time, e.g.

```
sbatch -A y11 -nodes=8 pi_hybrid.job
```

changes the number of nodes requested and the project account the job will run under.

To view the status of all the jobs in the system use the `squeue` command. To find information about a specific job using the jobid:

```
squeue -j <jobid.sdb>
```

To find information about jobs belonging to a specific user run:

```
squeue -u <username>
```

By default, when a run has completed, the output from the run is left in the run directory and is named based on the pattern defined in the batch scripts.

LAUNCHING APPLICATIONS ON THE HPE CRAY EX

Some time after a job is submitted the scheduler will assign the resources the job requested and start the job script running. At this point the supplied script will begin to execute on a node, and it is important to understand that at this stage most of the programs in the script are not yet running in parallel on the Cray XC30 compute nodes.

So for example, a simple statement like this:

```
cd ..
```

(which changes the current working directory to the same directory that the original `qsub` command was executed) is actually run in serial on one of the nodes.

Only programs/commands that are preceded by an `srun` statement will run in parallel on the compute nodes, e.g.

```
srun -n 16 -c 16 ./pi_mpi
```

Each batch job may contain multiple programs with `srun` statements, as long as:

- An individual parallel program does not request more resources than have been allocated by the batch scheduler
- The program does not exceed the wall clock time limit specified in the submission script.

See the lecture notes and the manual page, `man srun` for a full explanation of the command line arguments to `srun`.

Launching Hybrid MPI/OpenMP applications

The `srun` command cannot launch application threads directly. This is done by the application itself, commonly via the OpenMP runtime. However, SLURM can reserve space on the CPUs for threads to execute via the `-c` parameter.

For a typical OpenMP application the script should include the following settings:

```
export OMP_NUM_THREADS=4
sprun -n ${NPROC} --cpus-per-task ${OMP_NUM_THREADS} ./pi_mpi_openmp
```