OpenACC Workshop Notes

OpenACC (Open Accelerator) is a directive based programming environment which parallelises blocks of code for execution on the GPU. User code is prefixed with special directives to indicate to the compiler to parallelise. The code uses OpenACC environment variables to control run time behaviour and is linked against parallel libraries by the compiler. The only compiler on the Grace cluster that provides this functionality is the PGI compiler version 12.4 and greater.

Below are compilation (and link) commands for the PGI compiler that supports OpenACC:

PGI C compiler: `pgcc -acc -Minfo=accel program.c -o program`

PGI Fortran compiler:
```
pgfortran -acc -Mpreprocess -Minfo=accel \
program.f90 -o program
```

When compiling with the `-Minfo=accel` flag, it prints extra useful information:

```
33, Loop is parallelizable
    Accelerator kernel generated
    33, #pragma acc loop gang(8), vector(256)
    /* blockIdx.x threadIdx.x */
   CC 1.0 : 5 registers; 44 shared, 4 constant, 0 local memory bytes
   CC 2.0 : 8 registers; 4 shared, 56 constant, 0 local memory bytes
```

The above states that the loop at line 33 was parallelised and has produced two computer capability (CC) versions of the code. The version of interest is 2.0 which states that:

1. the number of registers per thread which is 8 in the above example;
2. amount of bytes of shared memory being used per block;
3. size of extra registers required which occurs when there is spilling.

Log on to the Grace login node and download the OpenACC programs:

```
wget http://grace-head00.uea.ac.uk/grace-docs/openacc_examples.tar
tar -xvf openacc_examples.tar
```

Load the module PGI 12.5 module file:

```
module load pgi/12.5
```

Please use the job submission script `gpu_job.sh` for all GPU executions.

Practical 1 - Counting the Number of GPU Devices

Use either program `get_num_devices.c` or `get_num_devices.f90` and follow these steps:

1. Compile the code;
2. Change the job submission script to execute the program:

```
./get_num_devices
```

3. Submit your job to the LSF queue. This should simply list the number of GPU cards available on a single node on the `gpu` LSF queue.

Practical 2 - Vector Addition

Use either the `vec_add.c` or `vec_add.f90` programs and follow these steps:

1. Add OpenACC directives to `copyin` vectors `c` and `b`, and `copyout` vector `a`;
2. Set the environment variable `PGI_ACC_TIME` using the command:

```
export PGI_ACC_TIME=1
```

3. Parallelise the second loop using the OpenACC `loop` construct and execute. How long did the initialisation take? How long did the data transfer take? How long did the main calculation take?
4. Parallelise the second loop using the combined `kernels loop` construct with the following `gang` (block), `worker` (thread) and `vector` parameters:

gang	worker (not Fortran)	vector
4	32	-
6	-	32
8	64	64
16	-	96
32	96	-

Time the execution time for each combination;

5. Which loop construct is the fastest and with which clauses and corresponding values?

Practical 3 - Matrix Multiplication

User either `matrix_mult.c` or `matrix_mult.f90` and follow these steps:

1. Set the environment variable `PGI_ACC_TIME` using the command:

```
export PGI_ACC_TIME=1
```

2. Copy the matrices `a`, `b` and `c`. Use the `copyout` clause for matrix `a` and `copyin` for matrices `b` and `c`;
3. Parallelise the outer loops using the `kernels loop` construct;
4. Parallelise the inner loop using the `loop` construct. Remember that a `kernel` construct cannot contain another `kernel` or `parallel` construct;
5. For the outer loop, use the following `gang` (block), `worker` (thread) and `vector` parameters:

gang	worker	vector
8	32	-
16	-	32

32	64	-

6. Which combination runs quicker?

Feedback

When you have completed the workshop, please remember to complete the course questionnaire at:

http://rscs.uea.ac.uk/events/feedback

All feedback is greatly appreciated!