Workshop 9: Dense matrix-matrix multiplication

1 Introduction

In this workshop we will investigate the performance of dense matrix-matrix multiplication on Isca. We will compare the performance of the BLAS DGEMM[1] routine from the Intel Math Kernel Library with a simple implementation using `for` loops.

2 Getting set up

Isca is physically located in the university data centre so it is accessed remotely by connecting to a log in node using SSH. On the module ELE page there are instructions describing how to log in to a remote Linux computer under the "Workshop 2" section.

- Start by logging in to Isca as described in the Workshop 6 instructions.
- The example programs for this workshop are provided as a tar file. There are two versions of the tar file (one for ECM3446 and one for ECMM461) Take a copy of the tar file with the correct module code by running either

  ```
  cp /gpfs/ts0/projects/Research_Project-IscaTraining/HPC/workshop9_ecm3446.tar .
  ```

  or

  ```
  cp /gpfs/ts0/projects/Research_Project-IscaTraining/HPC/workshop9_ecmm461.tar .
  ```

  where the dot at the end of the command causes the file to be copied into the current working directory. When you first log in to Isca the current working directory is your home directory, which is mounted on the log in nodes and all the compute nodes. There should be sufficient space in your home directory for all the workshops on this module.
- Unpack the tar file by running

  ```
  tar xvf workshop9_ecm3446.tar
  ```

  or

  ```
  tar xvf workshop9_ecmm461.tar
  ```

  This will create a directory called `workshop9` which contains the example program for this workshop.

[1]Double precision multiplication of general dense matrices (`https://software.intel.com/content/www/us/en/develop/documentation/mkl-tutorial-c/top/multiplying-matrices-using-dgemm.html`).

3 Test programs: `for` loops and `DGEMM`

In this section we will run two small test programs which perform matrix-matrix multiplication using either nested `for` loops or `DGEMM`. The first program we will look at is the version which uses `for` loops.

- Start by loading the Intel Cluster Toolkit module by running the command

  ```
  module load intel/2021b
  ```

- The `for` loop matrix-matrix multiplication program is called `matmul.c`. Compile the program using the supplied Makefile by running:

  ```
  cd workshop9
  make matmul
  ```

When the program runs it multiplies two matrices ($\mathbf{A}$ and $\mathbf{B}$) and adds the result to a matrix $\mathbf{C}$ ($\mathbf{C} := \mathbf{C} + \mathbf{AB}$). The matrices are all 3×3 and $\mathbf{C}$ is initially the zero matrix. At the end of the program the matrices $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$ are printed out.

- This is a very small test program, so it does not need to be run in the queue. Run the program directly from the command line

  ```
  ./matmul
  ```

  and check that the results are as expected.

The second version of the program multiplies the matrices using the BLAS routine `DGEMM`. `DGEMM` calculates $\mathbf{C} \leftarrow \alpha\mathbf{AB} + \beta\mathbf{C}$. In the example program $\alpha = 1.0$ and $\beta = 0.0$ and the matrices are the same 3×3 matrices used in the previous example.

- Build the `DGEMM` test program by running

  ```
  make dgemm
  ```

- This is also a small program which can be run directly from the command line. Run the `DGEMM` test program

  ```
  ./dgemm
  ```

  and check that the results match the previous results.

4 Comparing performance

In this section we will compare the performance of the two implementations of matrix-matrix multiplication. For these tests the problem size needs to be increased and the program run in the queue using the supplied job scripts.

The program `matmul_big.c` is a modified version of `matmul.c` which has been parallelised using OpenMP. The size of the matrices to use is read in as a command line argument and this value is supplied in the job script. The program records the time taken to multiply the matrices and uses this to calculate the flops (floating point operations per second) achieved.

- Compile the OpenMP version of the `matmul` program using make:

  ```
  make matmul_big
  ```

- Then run the program using the job script `run_matmul_big.sh`

```
sbatch run_matmul_big.sh
```

This will run the program using 16 OpenMP threads.

Question 1: How long does the `matmul` program take to multiply two 8000×8000 matrices?

The `DGEMM` function in the Math Kernel Library is multi-threaded so we can run the `DGEMM` version of the program using 16 threads and compare the results with `matmul_big.c`. The program `dgemm_big.c` reads in the size of a matrix as a command line argument and calculates matrix-matrix multiplication using `DGEMM`. It times the call to `DGEMM` and uses this to calculate the flops achieved based on the size of the matrices and the time taken by the `DGEMM` call.

- Build and run the corresponding `DGEMM` version (`dgemm_big.c`) of the program.

```
make dgemm_big
sbatch run_dgemm_big.sh
```

Question 2: How long does the `DGEMM` program take to multiply two 8000×8000 matrices and how much faster it is than the `matmul` program?

5 Scaling of `DGEMM` program

In this section we will see how well the `DGEMM` program scales. The job script `run_dgemm_multi.sh` runs the `dgemm_big` program with 2,4,8,12 and 16 threads.

- Submit the job to the queue to run the scaling test

```
sbatch run_dgemm_multi.sh
```

Question: 3 What fraction of peak performance does the DGEMM program achieve? The processor can deliver 16 operations per cycle and the base clock frequency is 2.60 GHz.

6 Changing the matrix size

In this last section we will investigate the effect of the matrix size on the performance. The job script `run_dgemm_var.sh` runs the `DGEMM` program with different size matrices using 8 threads.

- Submit the to the queue to run the test

```
sbatch run_dgemm_var.sh
```

Question: 4 How does the measured performance (flops) depend on the size of the matrix and is this what you would expect?