332 Advanced Computer Architecture

Exercise 4: Influence of problem size in Jacobi2d

Background

Carl Gustav Jacobi (1804-1851, see http://www-groups.dcs.st-and.ac.uk/~history/Mathematicians/Jacobi.html) gives his name to a particularly simple algorithm which is widely used in solving partial differential equations. For an example, see http://www.psc.edu/training/Drexel/Code_Development/Code_Development.html.

The idea is very simple indeed; here is a C implementation:

```
    typedef double gridtype[MAXiDIM][MAXjDIM];
    gridtype grid1;
    gridtype grid2;
    int it,i,j;
    double onequarter = 1.0/4.0;

    InitGrid(grid1, iDim, jDim);
    InitGrid(grid2, iDim, jDim);

    for (it=0; it<Niters; it+=1) {
      for (i=1; i<iDim-1; ++i)
        for (j=1; j<jDim-1; ++j)
          grid2[i][j] = onequarter *
            (grid1[i-1][j] + grid1[i+1][j] + grid1[i][j-1] + grid1[i][j+1]);
      it++;
      if (it >= Niters) break;

      for (i=1; i<iDim-1; ++i)
        for (j=1; j<jDim-1; ++j)
          grid1[i][j] = onequarter *
            (grid2[i-1][j] + grid2[i+1][j] + grid2[i][j-1] + grid2[i][j+1]);
    }
#ifdef PRINT_GRID
    /* the grid to print depends on whether Niters was odd or even */
    PrintGrid(Niters%2==1 ? grid2 : grid1, iDim, jDim);
#endif
```

As you can see, the loop simply smooths the initial grid by replacing each element `grid[i][j]` with the average of its four neighbours, north, east, west and south. This is often called a “four-point stencil”. Note that:

- Each iteration i produces a new grid$_i$ from an old grid$_{i-1}$.
- Instead of storing all the old grids, we generate `grid2` from `grid1`, then reuse `grid1` to store the next iterate, alternating.
- The grid region which is assigned to is smaller then the whole grid, by one cell in each direction. The boundary cells are read but never written.

Compiling and running the Jacobi benchmark

You can find the source code in

```
/homes/phjk/ToyPrograms/C/Jacobi2d-2003/Jacobi2d.c
```

Copy this file to your own directory. You should compile the application using the the following command line:

```
~phjk/gcc-3.0-x86/bin/gcc -O3 -DMAXiDIM=$x -DMAXjDIM=$x -o Jacobi2d Jacobi2d.c
```

where `$x` is a shell variable set to the array size to be used. Good values lie in the range 500-5000. Now run the application as follows:

```
./Jacobi2d $x $x 100
```

Here the value 100 is the number of iterations of the Jacobi sweep to be repeated. Choose this value to keep the elapsed time under control while ensuring the times are large enough to measure reliably - at least 1 second.

For `$x`=1000 the program produces output like this:

```
MAXiDIM 1000 MAXjDIM 1000 sweepi 1000 sweepj 1000 Niters 100 time 2970000 mflops 134.000000
After 100 iterations, sumcheck: 2.4389163277941414e+04
```

This shows that the loop ran at 134 MFLOPs (floating-point operations per second) on this machine (assuming 4 floating-point operations per loop iteration).

What to do

Your job is to characterise the behaviour of the Jacobi program, and study how the performance varies with array size.

1. Choose a linux machine on the DoC network[1] and measure the performance of the Jacobi2d loop in MFLOPs for problem size 1024.

   Note that you need to recompile the application for each problem size so that the arrays are declared with the right dimensions.

   Try to make sure no other processes are active as they will interfere with your results. How many times do you need to repeat your experiments to get a reliable result?

2. Plot a graph that shows how the performance, in MFLOPs, for Jacobi2d varies with problem size, and explain the result. Start by looking at problems in the range 1020-1030.

[1] See `https://www.doc.ic.ac.uk/csg/computers/`.

Tools and tips

Running the experiments

To do this you need to write a shell script. You might find the following Bash script useful:

```
#!/bin/bash -f

echo > results

for ((x=1010; x <= 1030 ; x++))
do
  echo Size $x:
  ~phjk/gcc-3.0-x86/bin/gcc -O3 -DMAXiDIM=$x -DMAXjDIM=$x -o Jacobi2d Jacobi2d.c
  ./Jacobi2d $x $x 100 >> results
done
```

The output is delivered to the file “results”. You can find this script in

```
~phjk/ToyPrograms/C/Jacobi2d-2003/scripts/Jacobi2d-script.sh
```

Producing a table

You need to convert the “results” file to a neat table of problem size against performance. Try:

```
grep MAXiDIM results | awk '{print $2, $14}' > table
```

This selects the results lines (not the sumcheck lines), then selects columns 2 and 14. The output is sent to the file “table”.

Plotting a graph

Try using the gnuplot program. Type “gnuplot”. Then, at its prompt type:

```
plot 'table' with linespoints
```

Paul Kelly, Imperial College London, 2003