Intro to HPC
Assignment 1

Wesley Liao
2019-09-13

1. In one sentence per bullet, define or expand each of the following terms or acronyms:

- Supercomputer

A large, high performance computing system in the top levels of computing capability of their time.

- High Performance Computing (HPC)

The application of computing technology on a large scale in order to accomplish computational tasks that would otherwise take impossibly long to be useful.

- metric

A method of evaluating performance of high performance computers.

- flops, gigaflops, teraflops, petaflops, exaflops

FLoating-point Operations Per Second, may be used with .

- benchmark

A defined procedure that can be run on various HPCs to compare their performance.

- Linpack, HPL

The LINPACK benchmark is used in determine the top 500 list.

- Top 500 list

The Top 500 list is a biannual measure of the top 500 supercomputers in existence.

- Parallel processing

Contrasted by sequential execution, parallel processing performs many simultaneous computations.

- MPI

The most widely used standard interface for writing HPC programs.

- Sustained performance

Sustained performance is the actual amount of performance (lower than the peak performance) that can be achieved by a supercomputer.

- OpenMP

OpenMP is a more recent open-source implementation of MPI.

- Moore’s Law

Moore’s law predicted that the number of transistors that could be placed in the same area on a semiconductor would double about every 2 years.

- Wall clock time, time to solution

The amount of time from the start to completion of a program, not considering the total time taken to run on each individual core or node.

- Scaling, scalability

The scalability of a problem refers to how increasing the computing resources improves the performance.

- Weak scaling

Weak scaled problems increase in difficulty as more computing resources are added to the problem.

- Strong scaling

Strong scaled problems can reduce the time to solution by adding more computing resources.

- Performance degradation

Many factors influence performance degradation, including bottlenecks or downtime from unreliability.

- Von Neumann architecture

The primary architecture of computers today.

- Shared memory

In a shared memory architecture, all processing nodes can access and work in the same memory.

- Distributed memory

In a distributed memory architecture, memory is divided among processing nodes that can only access their own memory.

- Non-uniform memory access cache coherent model

Memory is organized in multiple tiers of speed and size, balancing the advantages of distributed and shared memory.

- Commodity cluster

Rather than implement a completely custom processing and networking solution, commodity clusters achieve higher performance per cost by being built out of mass produced off the shelf computing parts.

- GPU

A Graphical Processing Unit is designed for mass parallel computation of simple operations used in computer graphics.

- Multicore socket

Multiple processor cores in a single processor socket.

- Many-core

A computing system utilizing a large number of processor cores.

2. What is the primary requirement that diferentiates HPC from other computers? What other requirements are also important?

HPC is used when the computing problems are so large that there is no other way to compute them (with today's technology) other than organizing a very large amount of computing resources in a single implementation. Practical limitations include cost, communication speeds, power requirements, cooling, physical space requirements.

3. Describe the computer recognized as the first true supercomputer?

The CDC 6600 implemented many of the same concepts in modern supercomputers, like parallel processing, and transistor-based logic.

4. Suppose you have a computer that has a 5-stage pipeline and you have a workload that has an input set of operand values of size 200. Assuming that each stage takes one unit of time and that passing results from one stage to the next is instantaneous, what is the average speedup in your computer for this workload as compared to a computer with 1-stage pipeline?

Processing 200 operands in 5 steps with no pipelining: 1000 units of time.
Processing 200 operands in a 5 stage pipeline: 44 units of time (22.7 times faster)

Loading:
1/u
2/u
3/u
4/u
Full load:
180/5 =36*5/u
Unloading
4/u
3/u
2/u
1/u

5. What was the fastest computer the year that you were born?
According to the top 500 list, the top computer in 1995 was the Numerical Wind Tunnel, from the National Aerospace Laboratory of Japan.

6. Write a matrix-vector multiply code in C. Provide a Makefle in order to compile your code on BigRed 2. The code has to compile upon invocation of the ‘make’ command on BigRed 2. Use dynamic memory allocation (malloc) for both the matrix and vector. Download the following matrix (size 443 e 443): https://sparse.tamu.edu/HB/bcspwr05 and compute the matrix-vector product of this matrix with the vector with elements defined by

$v_i = \sin((2 \pi i)/442)$

where $i \in [0,442]$ is the vector element index.
Report the l 1 norm of the resulting matrix-vector product. Compute the number of foating point operations per second achieved in the matrix-vector computation.

Program Result:

```
$ time ./assignment1 bcspwr05.mtx
…
l1 norm of result: -28.333006
total number of flops: 2805

real 0m0.035s
user 0m0.013s
sys  0m0.005s
```

2805flops/0.018s = 155,833 floating point operations per second

Assignment1.c

```
// Copyright 2019 Wesley Liao
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <string.h>

#include "mmio.h"


int main(int argc, char *argv[]) {

  //
  // Open the matrix file
  //
  char *matrix_filename = argv[1];

  FILE *matrix_file;
  matrix_file = fopen(matrix_filename,"r");

  if (matrix_file == 0) {

    printf(strcat(strcat("could not open file \"", matrix_filename), "\"\n"));

  }
  else {

    //
    // Parse matrix data from file
    //

    MM_typecode matrix_typecode;
    int number_of_rows;
    int number_of_columns;
    int number_of_nonzeros;

    mm_read_banner(matrix_file, &matrix_typecode);
    printf(strcat(mm_typecode_to_str(matrix_typecode), "\n"));

    if (mm_is_coordinate(matrix_typecode)
        && mm_is_pattern(matrix_typecode)
        && mm_is_symmetric(matrix_typecode)) {
```

```
      mm_read_mtx_crd_size(matrix_file, &number_of_rows, &number_of_columns, 
&number_of_nonzeros);

      printf("%i %i %i\n", number_of_rows, number_of_columns, number_of_nonzeros);

      int *matrix_data = calloc(number_of_rows*number_of_columns, 4);

      for (int i = 0; i<number_of_nonzeros; i++){
        // Read in the next coordinate pair from the matrix file
        int row;
        int col;
        fscanf(matrix_file, "%i %i", &row, &col);
        printf("%i %i\n", row, col);

        // Decrement the indexes because MM format indexes from 1 instead of 0
        row -= 1;
        col -= 1;

        // Set the corresponding matrix location to 1
        *(matrix_data + ((row*number_of_rows) + col)) = 1;
        // Because the matrix is symmetric, also set the opposite location
        *(matrix_data + ((col*number_of_rows) + row)) = 1;
      }

      // Print our imported matrix
      for (int i = 0; i < (number_of_rows*number_of_columns); i++) {
        // Because our data is 1d, print one character at a time
        // and when we hit a multiple of the width print a newline
        if (i % number_of_rows == 0)
            printf("\n");
        printf("%i", matrix_data[i]);
      }
      printf("\n");

      //
      // Create our vector
      //

      double *operation_vector = calloc(number_of_columns, 8);

      for (int i = 0; i < number_of_columns; i++) {
        *(operation_vector+i) = sin((2*M_PI*i) / (number_of_columns-1));
      }

      //
      // Compute matrix-vector product
      //

      double *result_vector = calloc(number_of_rows, 8);

      for (int row = 0; row < number_of_rows; row++) {
        for (int col = 0; col < number_of_columns; col++) {

            int matrix_value = *(matrix_data + ((row*number_of_rows) + col));

            if ( matrix_value == 1 ) {
              *(result_vector + col) += *(operation_vector + col);
            }
```

```c
        }
      }

      for (int i = 0; i < number_of_columns; i++) {
        printf("%f, %f\n", *(operation_vector+i), *(result_vector+i));
      }

      //
      // Calculate l1 norm of result vector
      //

      double l1norm = 0.0;
      for (int i = 0; i < number_of_rows; i++) {
          l1norm += *(result_vector + i);
      }
      printf("l1 norm of result: %f\n", l1norm);

      //
      // Calculate total number of floating-point operations
      //

      int flops =
        (3*number_of_columns)// 3 for each element in our operation vector
        + number_of_nonzeros // one fp addition per non-zero value in the matrix
        + number_of_columns; // final l1 norm sum

      printf("total number of flops: %i\n", flops);

      // Clean up allocated memory
      free(matrix_data);
      free(operation_vector);
      free(result_vector);
    }
    else {
        printf("bad matrix type. abort.\n");
    }

    fclose(matrix_file);
  }
}
```