HPC: Assignment #1

Due on August 7, 2018 at 11.59pm

Prof. Thomas Sterling

Akash B. Sheth

Problem 1

In one sentence per bullet, define or expand each of the following terms or acronym.

Solution

- **Supercomputer**: A computing system with high end capability to perform intensive scientific computations.
- **High performance Computing (HPC)**: It is a practice of combining computing power (parallel processing) in a way that delivers much higher performance than normal desktop computers in order to solve scientific problems.
- **metric**: it is a way of quantitative assessment
- **flops, gigaflops, teraflops, petaflops, exaflops**
    1. **flops**: floating point operations per second. A measure of computer performance.
    2. **gigaflops**: one billion floating point operations per second
    3. **teraflops**: one trillion floating point operations per second
    4. **petaflops**: one quadrillion floating point operations per second
    5. **exaflops**: one quintillion floating point operations per second
- **benchmark**: A standard point of reference used to compare or assess things against.
- **Linpack, HPL**
    1. **Linpack**: A software library for performing numerical linear algebra on digital computers.
    2. **HPL**: It is an implementation of Linpack used to test highly parallel computers.
- **Top 500 List**: A list of 500 top supercomputers in the world.
- **Parallel Processing**: Breaking down of a process into component parts and each of these component parts are run on different processors to be reassembled later to get the result of the processing.
- **MPI**: Message Passing Interface would allow programmers to write parallel applications that were portable to all major parallel architectures.
- **Sustained performance** Consistent level of performance over long periods of time.
- **OpenMP** A programming platform that allows one to parallelize code over shared memory system.
- **Moore's Law**: An observation made by Gordon Moore that the number of transistors in a chip will double every two years
- **Wall Clock Time, Time to solution**
    1. **Wall Clock Time**: Time from the start of the process till the end of the process.
    2. **Time to solution**: Performance time
- **Scaling, Scalability**
    1. **Scaling**: Reducing or increasing the ability of the system to handle work efficiently
    2. **Scalability**: Ability of system to handle increasing amount of work or ability to produce increased output when resources are added to handle added load.

Problem 1 continued on next page...

- **Weak Scaling**: Trying to reduce time to solution by keeping problem size same while increasing the system size.
- **Strong Scaling**: Trying to keep time to solution constant by increasing problem size along with the growing system size.
- **Performance degradation**: Software issues that could cause the same hardware to perform less effectively.
- **Von Neumann Architecture**: A digital system design comprising of Control Unit, Arithmetic Logic Unit/Processingi Unit, Input, Output and Memory.
- **Shared Memory**: Memory that is shared among different processes.
- **Distributed Memory**: In this, all of the processors have their own private memory.
- **Non-uniform memory access cache coherent model** Maintaining cache coherency across two or more processors by means of software, special purpose hardware or both.
- **Commodity Cluster**: A parallel computing system constructed from commodity subsystems which are employed as fully operational standalone mainstream systems
- **GPU**: Graphics Processing Unit, a computer chip that performs rapid mathematical calculations in a parallel fashion for the purpose of rendering images.
- **Multi-core Socket**: A socket is designed on the PCB to hold CPU which can have many cores inside it and making it a multi-core socket.
- **Many-core**: Many-core is a multi-core device with more than 8-10 cores

Problem 2

What is the primary requirement that differentiates HPC from other computers? What other requirements are also important? The primary requirements that differentiates HPC from other computers are

- More number of CPUs
- More number of GPUs
- More main memory - RAM
- Hardware to do specific operations quickly that would otherwise take more time

Even though about things are primary requirements, it not just the hardware that is important. A good harmony between hardware and software is equally important. Code that could take advantage of the hardware by running multiple operations efficiently is quite important.

Problem 3

Describe the computer recognized as the first true supercomputer?

Solution

CDC 6600 is the computer recognized as the first true supercomputer manufactured by Control Data Corporation. It achieved performance of up-to three megaflops. There were many operating systems that CDC 6600 machine was run on but the two that persisted were SCOPE and KRONOS. SCOPE OS provided better file system while KRONOS was used for its time sharing and BATCHIO remote batch features. In order to combine benefits of both the operating systems, Network Operating System (NOS) was produced as the sole operating systems for all CDC machines.

In any typical machine, CPU is the driving force for the entire system and they were big in size. Back in those days, CPUs were considerably slower than the main memory and due to this, main memory was idle for a significant time. In order to get a better performance, CDC 6600 CPUs handled only arithmetic and logic and due to this, it became possible to reduce CPU size and run at higher clock speed. The CPU used in CDC 6600 was a 60-bit processor running at 10MHz and it had main memory of up to 982kilobytes.

Problem 4

Suppose you have a computer that has a 4-stage pipeline and you have a workload that has an input set of operand values of size 100. 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?

Solution

Time per stage is 1 unit time.

$$\begin{aligned} SpeedUp &= \frac{Time\ required\ to\ complete\ 100\ tasks\ with\ 4\ stage\ pipelining}{Time\ required\ to\ complete\ 100\ tasks\ without\ pipelining} \\ &= \frac{4 * \frac{1}{unittime} * 100}{\frac{1}{unittime} * 100} \\ &= 4 \end{aligned}$$

$\therefore$ Speedup = 4 times with 4 stage pipelining

Problem 5

What was the fastest computer the year that you were born?

Solution

I was born in 1992 and the fastest computer was **Intel Delta**.

Problem 6

```
// MATRIX-VECTOR MULTIPLICATION CODE
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <sys/time.h>
// Populate matrix from the file
void load_matrix(double **matrix_ptr, int *row, int *column) {
    FILE *fp;
    char *line = NULL;
    size_t len = 0;
    ssize_t read;

    fp = fopen("fs_183_1.mtx", "r");
    if (fp == NULL)
        exit(EXIT_FAILURE);

    int line_count = 0;
    while ((read = getline(&line, &len, fp)) != -1) {
        int total_lines;
        line_count += 1;
        if(line_count == 1) {
            continue;
        } else if (line_count == 2) {
            *row = atoi(strtok(line, " "));
            *column = atoi(strtok(NULL, " "));
            total_lines = atoi(strtok(NULL, " "));
            *matrix_ptr = (double *) malloc(sizeof(double) * (*row) * (*column));
            memset(*matrix_ptr, 0, (*row) * (*column) * sizeof(double));
        } else {
            int temp_r, temp_c;
            double temp_value;
            char *eptr;
            temp_r = atoi(strtok(line, " "));
            temp_c = atoi(strtok(NULL, " "));
            temp_value = atof(strtok(NULL, " "));

            *(*matrix_ptr + (temp_r-1) * (*column) + (temp_c-1)) = temp_value;
        }
    }

    fclose(fp);
    if (line)
        free(line);
}
```

```
// Populate the vector with formula: sin(2*pi*i/182)
void get_vector(double **vector_ptr, int column) {
    int i;
    *vector_ptr= (double *) malloc(sizeof(double) * column);
    memset(*vector_ptr, 0, column * sizeof(double));

    for (i = 0; i < column; i++) {
        // *(*vector + i) = sin((2 * M_PI * i)/182) ;
        *(*vector_ptr + i) = sin((2*M_PI*i)/182);
    }
}

void mat_vec_product(double **matrix_ptr, double **vector_ptr, double **prod_result_ptr,
                    double *l2_norm, int row, int column, int times) {

    struct timeval start, end;

    int i, j, k;
    double sum = 0.0, prod_sum = 0.0;

    *prod_result_ptr = (double *) malloc(sizeof(double) * row);

    int x;
    gettimeofday(&start, NULL);
    for (x = 0; x < times; x++) {

        memset(*prod_result_ptr, 0, row * sizeof(double));
        for (i = 0; i < row; i++) {
            for(j = 0; j < column; j++) {
                *(*prod_result_ptr + i) += *(*matrix_ptr + i * column + j) * *(*vector_ptr + j);
            }
        }
    }
    gettimeofday(&end, NULL);


    // fprintf(stdout, "Start | sec: %d, usec %d \n", start.tv_sec, start.tv_usec);
    // fprintf(stdout, "End | sec: %d, usec %d \n", end.tv_sec, end.tv_usec);

    double time_taken = ((end.tv_sec + end.tv_usec*1.0e-6) -
                            (start.tv_sec + start.tv_usec*1.0e-6))/(1.0*times);
    for (i = 0; i < row; i++) {
        sum += pow(*(*prod_result_ptr + i), 2.0);

    }
    *l2_norm = sqrt(sum);
    fprintf(stdout, "l2_norm : %lf \n", *l2_norm);
    fprintf(stdout, "time_taken in seconds: %lf \n", time_taken);
    fprintf(stdout, "FLOPS: %lf \n", (2*row*column)/(time_taken*1000000.0));
```

```
}

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

    fprintf(stdout, "\n");
    // fprintf(stdout, "Argc --> %d\n", argc);
    int i, j, times;
    if (argc > 1) {
        times = atoi(argv[1]);
    } else {
        times = 10000;
        fprintf(stdout, "Running  %d by default and taking average.\n", times);
        fprintf(stdout, "It can also be run as: ./mat_vev_prod <times>.\n");
        fprintf(stdout, "\n");
    }
    double *matrix_ptr, *vector_ptr, *prod_result_ptr;
    double l2_norm;
    int row, column;

    // Load the matrix
    load_matrix(&matrix_ptr, &row, &column);

    // Generate the vector
    get_vector(&vector_ptr, column);

    // Peform calculations
    mat_vec_product(&matrix_ptr, &vector_ptr, &prod_result_ptr, &l2_norm, row, column, times);
    fprintf(stdout, "\n");

    // fprintf(stdout, "Rows: %d | ", row);
    // fprintf(stdout, "Columns: %d \n", column);
    // for (i = 0; i < row; i++) {
    //     for (j = 0; j < column; j++) {
    //         fprintf(stdout, "i: %d, j: %d ==> %.13e \n", i, j, *(matrix + i*(column) + j));
    //     }
    //     fprintf(stdout, "\n");
    // }

    // Release memory
    free(matrix_ptr);
    free(vector_ptr);
    free(prod_result_ptr);

    return 0;
}
```

```
// MAKEFILE
CC = gcc
CCFLAGS =
TIMEDEF =
MHZ =
OBJ = mat_vec_prod.o
mat_vec_prod: $(OBJ)
$(CC) $(CCFLAGS) -o $@ $(OBJ) -lm
mat_vec_prod.o : mat_vec_prod.c
$(CC) $(CCFLAGS) -c mat_vec_prod.c
clean :
rm -f $(OBJ)
rm -f mat_vec_prod
rm -f *.xf
rm -f *.ap2
rm -f *.apa
rm -f  *+pat
kill :
rm -f mat_vec_prod $(OBJ)
```

l2_norm : 1127832605.489966
time_taken in seconds: 0.000107
megaFLOPS: 625.505503