Question 1  [20 marks]

*Shared Memory Programming with OpenMP*

(a) Consider three distinct \( N \times N \) matrices, A, B and C. The following code performs a matrix multiplication using a single thread of execution.

```c
for ( I = 0 ; I < N ; I++)
    for ( J = 0 ; J < N ; J++) C[I][J] = 0;

for ( I = 0 ; I < N ; I++)
    for ( J = 0 ; J < N ; J++)
        for ( K = 0 ; K < N ; K++) C[I][J] += A[I][K]*B[K][J] ;
```

Two attempts to develop an OpenMP parallel implementation are made:

**Parallel Version #1**

```c
#pragma omp parallel
{
    #pragma omp for nowait
    for ( I = 0 ; I < N ; I++)
        for ( J = 0 ; J < N ; J++) C[I][J] = 0;

    #pragma omp for
    for ( I = 0 ; I < N ; I++)
        for ( J = 0 ; J < N ; J++)
            for ( K = 0 ; K < N ; K++) C[I][J] += A[I][K]*B[K][J] ;
}
```

**Parallel Version #2**

```c
#pragma omp parallel
{
    #pragma omp for
    for ( I = 0 ; I < N ; I++)
        for ( J = 0 ; J < N ; J++) C[I][J] = 0;

    #pragma omp for
    for ( K = 0 ; K < N ; K++)
        for ( J = 0 ; J < N ; J++)
}
```

You test both implementations using multiple threads on a variety of different computer systems. You find that under some circumstances both versions give **incorrect** results.

(i) Explain why each parallel implementation can sometimes give **correct** results, but can also sometimes give **incorrect** results. Describe the hardware/runtime circumstances where each scenario is likely to hold.
(ii) In each case what modifications to the OpenMP directives would you make to ensure that both parallel implementations were always correct. (You should make no modifications to the basic loop structure, but are free to move, rewrite or add new OpenMP directives.)

(iii) Discuss the performance of your two correct OpenMP versions. You should consider issues like the number of threads used, the problem size etc. Define, as required details of the parallel computer system you are running on.

[16 marks]

(b) Suppose that you had six C functions

```c
void functionA(void)
void functionB(void)
void functionC(void)
void functionD(void)
void functionE(void)
void functionF(void)
```

that together solve some problem (since there are no parameters or return values, you should assume that the functions modify globally visible variables). These functions depend on each other according to the following dependency graph:

```
A ----> B ----> C  
|        |        |   
|        v        v   
|        D ----> E  
|           |        |   
|           v        v   
|           F        
```

where, for example, functionA must complete execution before functionF can start. Sketch a C program that uses OpenMP to execute the above six functions in a way that is maximally parallel, but that adheres to the above dependency graph. (Hint, consider use of OpenMP sections.)

[4 marks]
Question 2 [20 marks]

Hardware/Software Interactions

The following C code performs a matrix vector product:

```c
void matvec(int M, int N, double x[], double a[], double b[]){
    int i, j;
    for ( i = 0; i < N; i++){
        for ( j = 0; j < M; j++){
            x[i] += a[i*M+j]*b[j];
        }
    }
}
```

The code is run on a 2GHz in-order processor that has the following characteristics:

- A group of up to 4 instructions can be issued per cycle, with (i) up to 2 integer instructions, (ii) up to 1 load or store instruction, (iii) up to 2 different floating point instructions and (iv) up to 1 branch instruction.
- The latency from cache for a double precision load is 2 cycles, with store operations to cache also completing in 2 cycles. The latency of a floating point instruction is 3 cycles, and all integer instructions take at most 1 cycle.

(a) What is the minimum cycle time required to complete the floating point operations associated with one iteration of the inner \( j \) loop if all data is in cache. Assume no loop unrolling and ignore all integer operations associated with the loop index. Give the cycle time from issuing the first floating point load until completion of the final floating point store. Based on this time what MFLOP rate will this loop achieve? Show exactly how you derive your answers.

[6 marks]

(b) You are told that there is no aliasing between the function arguments. What does this mean, and how does knowing this relate to loop unrolling?

[4 marks]

(c) By unrolling the inner loop a much higher rate of performance can be achieved. In terms of MFLOPS, what is the maximum possible performance that can be achieved for the inner \( j \) loop by unrolling it. What level of loop unrolling is required to achieve this level of performance? (Show how you derive this number).

[5 marks]

(d) Write C-code showing how you would perform 2-way unrolling of the outer \( i \) loop. Is manual unrolling of this loop index likely to be beneficial? Justify your answer.

[5 marks]
Question 3  [20 marks]

**GPU Systems**

(a) A vector rotation around the Z-axis can be expressed as

\[
\begin{pmatrix}
V_x \\
V_y \\
V_z
\end{pmatrix} =
\begin{pmatrix}
\cos \theta & -\sin \theta & 0 \\
\sin \theta & \cos \theta & 0 \\
0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
V_x \\
V_y \\
V_z
\end{pmatrix}
\]

The following kernel performs the above rotation on a list of N vectors stored as \( V = \{ [V^1_x, V^1_y, V^1_z], [V^2_x, V^2_y, V^2_z], \ldots, [V^N_x, V^N_y, V^N_z] \} \) using CUDA:

```c
__global__ void rotate(double ctheta, double stheta, double *V, int N) {
    int idx = blockIdx.x*blockDim.x + threadIdx.x;
    double tmp[3];
    if (idx < N) {
        tmp[0] = ctheta * V[3*idx] - stheta*V[3*idx+1];
        tmp[1] = stheta * V[3*idx] + ctheta*V[3*idx+1];
        tmp[2] = V[3*idx+2];
        V[3*idx] = tmp[0];
        V[3*idx+1] = tmp[1];
        V[3*idx+2] = tmp[2];
    }
}
```

void main() {
    ........
    block_size = 400;
    n_blocks = N / block_size + (N % block_size == 0 ? 0 : 1);
    rotate <<<n_blocks, block_size>>> (cos(theta), sin(theta), V, N);
    ........
}

There are at least three different performance issues with this kernel that are associated with i) the array \( \text{tmp} \), ii) the way the array \( V \) is accessed, and iii) the block size used. What are these three different performance issues, and how can they be addressed. (Assume that a GTX295 GPU system is being used, i.e. an identical system to that used in this course.)

[6 marks]

(b) Inside a modern GPU, the memory hierarchy includes local memory, shared memory, global memory, and constant memory. For each type of memory state its i) visibility, ii) access time, and iii) common/recommended usage.

[8 marks]

(c) On the GTX295, a multiprocessor can support up to 1024 resident threads at one time; i) Define what is meant by occupancy? ii) What is the advantage of maintaining high occupancy? and iii) What are the factors which determine the level of occupancy?

[6 marks]
Question 4 [30 marks]

Short Questions

(a) Define the term truncation error. [2 marks]

(b) Each of the following statements are either true or false. Which are they? Give a few sentences to explain your answers.

(i) If two real numbers are exactly representable as floating-point numbers, then the result of any real arithmetic operation on them will also be exactly representable as a floating-point number providing that the result is not larger/smaller than the overflow/underflow limit.

(ii) Between the underflow limit and overflow limit there exists groups of floating point numbers that are distributed uniformly within the group.

(iii) In a floating point number system the machine precision is the smallest positive number that perturbs the number 0 when added to it. [6 marks]

(c) Give three examples of events you might seek to measure using hardware performance counters, stating why each event is useful to measure. [3 marks]

(d) Consider a machine that uses 32bit addressing and has byte-addressable memory.

(i) How many entries are there in the page table if there are 4KB pages?

(ii) What is the full name of the special purpose cache that is used to speed up the mapping of virtual to physical addresses?

(iii) How many entries does this special purpose cache typically hold?

(iv) Sketch some simple code for which the observed performance may be adversely affected because the size of this special purpose cache is too small. Provide some explanation of your code and when it exhibits poor performance. [4 marks]

(e) Give a few lines of code to indicate what is meant by

(i) A control dependency

(ii) A data dependency

(iii) An output dependency [3 marks]
(f) The large Intel Nehalem based cluster you saw at the National Computational Infrastructure National Facility exhibits parallelism at multiple levels. Name five different types of parallelism found in this system.  

[5 marks]

(g) Most compilers translate code into an intermediate representation (IR) that is then subjected to optimisation and eventual code generation. Usually the IR incorporates the core features of the RISC architecture.

(i) Give two core RISC features that the IR might try to model.

(ii) Illustrate your answer with small bits of C code and their corresponding IR.  

[4 marks]

(h) Post RISC processors are associated with out-of-order execution.

(i) Define what is meant by out-of-order execution.

(ii) Name one advantage and one disadvantage associated with out-of-order execution.

(iii) Name one processor that supports out-of-order execution and one processor that does not support out-of-order execution.  

[3 marks]
Question 5  [10 marks]

In this course you developed an elementary molecular dynamic (MD) application using C and Python. In this application the force calculation was the most computationally intensive component. Code for the force evaluation is given below:

```c
inline void pairwise_force(double **coord, double **force, int i, int j)
{
    double r = dist(coord[i], coord[j]);
    double Fc = 12.0 * (1.0/pow(r, 14.0) - 1.0/pow(r, 8.0));
    double Fx = Fc * (coord[i][0] - coord[j][0]);
    double Fy = Fc * (coord[i][1] - coord[j][1]);
    double Fz = Fc * (coord[i][2] - coord[j][2]);

    force[i][0] += Fx;
    force[i][1] += Fy;
    force[i][2] += Fz;

    force[j][0] -= Fx;
    force[j][1] -= Fy;
    force[j][2] -= Fz;
}

void updateForce(double **coord, double **force, int n_atoms) {
    int i = 0, j = 0;

    for (i = 0; i < n_atoms; i++) {
        for (j = 0; j < i; j++) {
            pairwise_force(coord, force, i, j);
        }
    }
}
```

(a) Describe how you would implement the above routine using CUDA on a GTX295 GPU system (i.e. a GPU identical to that used in this course). Provide pseudo code as required. Give consideration to the data and memory types used, and to the allocation of threads to blocks.  

[5 marks]

(b) Discuss the parallel performance of your code on the GTX295 GPU.  

[3 marks]

(c) Under what circumstances do you expect it to be beneficial to use a GPU accelerator to improve the performance of an MD application running on a standard processor.  

[2 marks]
Question 6  [10 marks]

(a) In the context of this course define “benchmark”, and briefly describe one purpose, property and common mistake (i.e. three things) associated with benchmarking.  

[5 marks]

(b) In the labs and assignments you worked with an UltraSPARC T2 system called Wallaman and a GTX295 GPU system. Both are highly multi-threaded architectures.

(i) Compare and contrast the types of computations that are well suited to the T2 system and the GPU system.

(ii) Name one advantage of the T2 over the GPU and one advantage of the GPU over the T2.

(iii) If you had to develop a multi-threaded molecular dynamics code similar to that which you wrote in assignment 1 of this course, which of these two systems would you use and why?  

[5 marks]