Programming for performance on multi-core and many-core 22nd March 2012

Quiz: **25 marks** **[1 hour]**

1. Why **Loop unrolling** is often useful in improving the performance of numerical code? [1m]

a) It helps in having less instruction cache misses.
b) It enables other optimizations, for example scalar replacement or reassociation.
c) It enables better instruction scheduling.

Pick the true statements [**b, c**].

2. Consider the following code: [2m]

```
double * func (const double N, const double *x, int size) {
for (i=1, i < size; i++)
x[i] = (x[i]/size) * N;
}
```

The previous loop can be optimized as follows:

```
double * func (const double N, const double *x, int size) {
double tmp = N/size;
for (i=1, i < size; i++)
x[i] = x[i] * tmp;
}
```

Does compiler perform this optimization or not? Explain?

Ans: Floating point multiplication is not associative.

3. A GPU SM can load and store to and from (circle all that apply): [**a, b**] [1m]
a) the GPU's device memory (on-board, off-chip)
b) the SM's local shared memory (on-chip)
c) the shared memories of other SM's
d) the L1 caches of other SMs

4. GPU thread divergence happens if (circle one): [**d**] [1m]
a) different SMs execute different code
b) different thread blocks in one SM execute different code
c) different warps in one thread block execute different code
d) none of the above
e) all of the above

5. The code below was optimized in different ways resulting in versions v1 and v2. Both versions were tested on arrays of different sizes in the same computer. When v1 was used to compute on arrays of size 2000, the execution time was 64,000 cycles. When v2 was used when arrays of size 100, the execution time was 4000 cycles. Which version do you think is better optimized? Justify your answer. **[2m]**

```
for (int i = 0; i < size; i++){
C[i] = sqrt(A[i])+sqrt(B[i]);
}
```

Ans: The data given is insufficient to make any valid conclusions on the performance of v1 and v2. To make any conclusive decisions both v1 and v2 should be run on same set of sizes of different range.

6. Does a commercial compiler (icc) with higher optimization flags vectorize this loop? If not why? If so, how? **[3m]**

```
loc=maxint;
for (i=0; i<n; i++)
  if (x[i]<0){
      loc=i;
      break;
   }
```

Ans: (pseudocode)

```
loc=maxint;
res=n%m;
for (i=0;i<n-res;i+=m){
        if (x[i:i+m-1]<0) != \(0,0,0..0\) {
                for (j=i;j<i+m;i++){
                        if x[j]<0{
                                loc=maxint;
                                break;
                        }
                }
                break;
        }
}
If loc==maxint{
        for (i=n-res; i<n; i++)
                if x[i]<0{
                   loc=i;
                   break;
              }
```

7. Optimize the cache performance of the following code. Do not parallelize it. Do not consider prefetching nor TLB behavior. You can assume that none of the operations overflow. Assume that N and M are extremely large powers of 2. Assume that A[], B[], and C[] are integer arrays, and that an int is 4B. **[3m]**

```
for (i=0;i<N;i++)
 for (j=0;j<M;j++)
 A[i] += B[j][j] * C[i];
```

Ans:

```
int tmp = 0;
for (j=0;j<M;j++)
 tmp += B[j][j];
for (i=0;i<N;i++)
 A[i] += tmp * C[i];
```

8. a) The following data structure Data2 is used to store one million samples of (x,y,z) 3D positioning data. How many bytes of memory will Data2 occupy?

```
struct mystruct {
 int x; // 4
 int y; // 4
 int z; // 4
} Data2[1000000];
```

Ans: = 12B * 1000000 = 12000000B

b) How many bytes of memory will the following alternative data structure Data3 occupy?

```
struct mystruct {
 int x[1000000]; // int is 4
 int y[1000000]; // int is 4
 int z[1000000]; // int is 4
} Data3;
```

Ans: = 4B * 1000000 * 3 = 12000000B

c) Data2 and Data3 above can both be used to store the same set of data, but would have different layouts in memory. If you were asked to write code to compute the distance from the origin of (x,y,z) for 1000 randomly-selected samples, which of Data2 or Data3 would be the preferred data structure and why is it better than the alternative?
Ans: Data2, since you are accessing x, y, and z for each randomly-selected element. There would be cache-block locality within each element. Data3 would require accessing 3x as many cache blocks, since x, y, z would always reside on independent cache blocks.

d) If you were asked to write code to compute the mean of y across all million values, which of

Data2 or Data3 would be the preferred data structure and why?
Ans: Data3, since you are only accessing y for each element. There would be cache-block locality from one element to the next, which would also enable hardware/compiler prefetching.

9. Generally, in which case would function inlining be least likely to harm performance: A) a large function called from many locations; B) a large function called from a few locations; C) a small function called from many locations; D) a small function called from a few locations. [**D**] **[1m]**

10. Why are array based codes easier for compilers to parallelize than pointer-based codes? **[1m]**

Ans: Compilers have trouble disambiguating pointers, while array index functions can often be fully understood.

11. Tiling: Given this machine:
1st level cache **[3m]**
128B cache blocks
8-way set associative
16 sets,
Page size: 8KB,
TLB: fully associative, 128 entries

If you were to tile the following code

// assume D[] is already initialized to zeros

```
void mmmc(int A[], int B[], int C[], int D[], int n){
for (i=0;i<n;i++){
for (j=0;j<n;j++){
for (k=0;k<n;k++)
D[i*n+j] += A[i*n+k] * B[k*n+j] * C[i*n+j];
      }
    }
  }
```

Ignoring storage for instructions and small variables, and assuming that there are no unlucky conflict misses (only capacity misses), what is the largest tile size T (in number of elements, not bytes) that you can use to tile to minimize misses for the first level data cache ?

Ans: capacity = 128*8*16 B = (2^7)*(2^3)*(2^4) B = (2^4)*(2^10) B = 16*1KB = 16KB

4*(4B*tile_size_in_elements**2) <= 16KB
4B*tile_size_in_elements**2 <= 4KB
tile_size_in_elements**2 <= 1024
tile_size_in_elements <= 32 elements

Tile size is 32 elements

12. Compiler Optimization [5m]

Name 8 optimizations that you would hope your compiler would do to this code. For each, give the formal name of the optimization and very briefly describe which variable(s) or code parts it will modify/improve and how. The first answer of 8 is given as an example.

```
1: x=5;
2: y=2;
3: debug = 0;
4: z=x+y;
5:
6: for (i=0;i<100;i++){
7: m += i*x+y;
8: n = z*y;
9: A[i] += A[m];
10:
11: if (debug){
12: printf("m: %d\n",m);
13: }
14: q += i*x;
15:}
16:
17:printf("result: %d %d %d %d %d %d\n",m,n,q,x,y,z);
```

ANSWERS:

constant propagation: line4 changed to z=5+2
1 assign m,n,q to registers to reduce load/store latency
1 constand folding: line 4 changed to z=7
1 constant propagation: line 7 changed to m+= i*5+2
1 constant propagation: line 8 changed to n = 7*2
1 constant folding: line 8 changed to n = 14
3 loop invariant code motion: line 8 moved outside of loop
1 dead code elimination: lines 11-13 deleted (cannot execute since debug=0)
2 common subexpression elimination: both line 7 and line 14 compute i*x (i*5 after ConstPropagation)
1 line 9 is dead (if you assume that A[] is unused later in the code)
1 const propagation line14 changes to i*5
1 strength reduction: line8 becomes n = z<<1 (if you didn't propagate a constant for z)
1 dead code elimination: line3 is dead if you eliminated lines 11-13
1 constant propagation: line11 if (0)
1 reorder instructions to hide the latency of loading A[m], or prefetch A[m], or q, or A[i]
NOTE: maybe not all of these optimization make sense together.

13. The following C program is run (with no optimizations) on a processor with a direct-mapped cache that has eight-word (32-byte) blocks and holds 256 words of data:

```
int i,j,c,stride,array[512];
...
for(i=0;  i<10000;  i++)
   for (j=0;  j<512;  j+=stride)
      c += i%(array[j]);
```

If we consider only the cache activity generated by references to the array and we assume that integers are words, what possible **miss rates** (there may be multiple) can we expect

1. if stride = 256?
2. if stride = 255 ?

Explain your answers clearly in a few sentences.

Ans:

a. 100%

Explanation:

Every access is a conflict with the previous.

b. 100%, 67%

Explanation:

100% if element 0, 255, and 510 all map to the same block. This can happen if 0 maps to the middle of a block.

67% if element 0 maps to the beginning of a cache block. Then 255 and 510 would map to the block "before" it.