Fall 2013 CSE Qualifying Exam
Core Subjects

September 28, 2013

Architecture

1. Consider the following five microarchitectures:

   (a) a single processor core that supports 8-issue superscalar instruction dispatch, out-of-order instruction execution, branch speculation, and robust branch prediction,

   (b) a single in-order processor core that is single issue but supports 6-way simultaneous multithreading, having four program counters and four register files but all threads share a single pool of functional units,

   (c) a four-core multicore processor, where each processor core is single issue, in order, and can execute only one thread,

   (d) a single processor with a vector unit that can perform 8-lane vector operations, containing a vector register file having eight vector registers of 64 elements each, and a scatter-gather load/store unit, and

   (e) a dual core processor where each core is single issue but can execute VLIW instructions containing 4 FP and 2 integer instructions, and each core contains a 4-wide SIMD unit.

   Describe which types of applications will achieve the best performance on each of these microarchitectures and why.

2. Consider a cacheless processor connected to a DRAM having the following address mapping:

```
bits 5:3       column address

bits 8:6       bank address

bits 18:9      row address
```

Assume you're implementing a kernel that performs a 3x3 averaging filter for an 8-bit monochrome image:

```
unsigned char image[640*480];
...
    for (i=0;i<480;i++)
        for (j=0;j<640;j++) {
            sum=0;
            for (k=0;k<3;k++)
                for (l=0;l<3;l++) {
                    idx=(i+k-1)*640 + (j+l-1);
                    sum += image[idx];
                }
            image2[i*640+j] = sum/9;
        }
```

Assume that consecutive accesses to different DRAM rows requires 10 cycles of latency, consecutive accesses to different DRAM columns (with same row) requires 5 cycles of latency, and consecutive accesses to different DRAM banks (with same row and column) requires 2 cycles of latency.

Would it be possible to change the code above such that its behavior is the same (i.e., performs the same 2D filter) but has a different memory access pattern that achieves better performance given the characteristics of the memory system? Why or why not? If so, give a brief explanation how this can be done (you don't need to actually transform the code).

3. You are given an application that runs in 1 minute and executes 1G instructions on a single processor.

   (a) If the application is 90% parallelizable (both time and instructions) and you have 100 processors, how fast will the application run?

   (b) If the CPI of the sequential portion of the program equals the CPI of the parallelizable part, what does Gustaphason's Law say about the number of instructions that could be executed with 100 processors in that 1 minute?

   (c) In Tomasulo's algorithm explain in detail what happens when an instruction is issued (from the Queue to Reservation Station), including conditions that allow the issuing.

   (d) Explain in detail the workings of the CDB, who writes, what they write, where results go. Include how you can have a structural hazard with the CDB and how it would be handled.

Compilers

1. Design an EBNF grammar that represents expressions for a language with the operators in the following table. The start symbol of your grammar should be *Expression*. Introduce nonterminal symbols as needed for the grammar to be unambiguous. Write the production rules so that parse trees represent associativity and precedence correctly. The following table lists the symbols in decreasing order of precedence. For example, `~` (which in this language, as in ML, indicates unary minus) and `!` have the highest precedence.

Operators	**Type**	**Associativity**
`~ !`	unary prefix	none
`* / %`	binary infix	left
`< <= > >=`	binary infix	none
`== !=`	binary infix	none
`&&`	binary infix	left
`||`	binary infix	left

   Assume that (round) parentheses, literals, and identifiers can be used to build up expressions using the operators in the table. The parentheses are used to override the associativity and precedence described in the table. You do not need to write production rules that define literals and identifiers. You may use your favorite flavor of EBNF.

2. Consider the following program in a C-like language with *dynamic* scope:

```
int n;
int fact()
{
  int loc;
  if (n > 1){
    loc = n--;
    return loc * fact();
  }
  else
    return 1;
}

main{}
{
  get(n);
  if (n >= 0)
    print(fact(n));
  else
```

```
    print("input error");
}
```

Assume that the program is run on a two-memory machine with separate code and data memory. Assume further that 3 is read.

(a) Draw data memory as a stack of activation records at its deepest point. Show the control link (sometimes called dynamic link), the return pointer, the returned value, and local data for each activation record. Since you are not asked to translate the code into assembly language, you cannot give a precise value for the return point, but explain what is it for.

(b) What would change in the activation records if the language had static scope?

3. The following fragment of 3-address code was produced by a nonoptimizing compiler (gcc -S sumss.c):

```
 1  sumss
 2  .LFB0     ss = 0
 3            i = 1
 4            goto .L2
 5  .L6       j = 2
 6            goto .L3
 7  .L5       t1 = i + 5
 8            if j > t1 goto .L4
 9            t2 = 4*i
10            t3 = a[t2]
11            t4 = j*4
12            t5 = t4 + t3
13            t6 = *t5    // dereference
14            t7 = 4*i
15            t8 = a[t7]
16            t9 = j*4
17            t10 = t9+t8
18            t11 = *t10
19            t12 = t6*t11
20            ss = ss + t12
21  .L4       j = j +1
22  .L3       if j < m goto .L5
23            i = i + 1
24  .L2       if i < n goto .L6
25            ret ss
```

Assume that there are no entry points into the code from outside other than at `sumss`.

(a) (20% credit) Decompose the code into basic blocks $B_1, B_2, \ldots,$ giving a range of line numbers for each.

(b) (20% credit) Draw the control flow graph, and describe any unreachable code.

(c) (40% credit) Fill in a 25-row table listing which variables are live at which control points. Treat the array `a` as a single variable. Assume that `ss` is the only live variable immediately before line 25. Your table should look like this:

Before line	**Live variables**
1	...
2	...
3	...
...	...

(d) (20% credit) Describe any simplifying tranformations that can be performed on the code (i.e., transformations that preserve the semantics but reduce (i) the complexity of an instruction, (ii) the number of instructions, (iii) the number of branches, or (iv) the number of variables).

Algorithms

1. Find tight asymptotic bounds on any positive function $T(n)$ satisfying the recurrence

$$T(n) = T(\sqrt{n}) + T(\sqrt[3]{n}) + T(\sqrt[6]{n}) + \lg n \; .$$

Show how you arrived at your answer. You may assume any implicit floors or ceilings are of no consequence. [Note: $\sqrt[k]{n} = n^{1/k}$ for any $n, k > 0$.]

2. You are given a directed acyclic graph $G = (V, E)$ as input, represented in the adjacency list representation with vertex array $V[1 \ldots n]$, where $n = |V|$. Describe an algorithm that returns the length of the longest (unweighted) path in G from $V[1]$ to $V[n]$ (or some arbitrary negative number if no such path exists). Here, the *length* of a path is the number *edges* along the path. Your algorithm should run in time $O(|V| + |E|)$.

NOTA BENE: You can assume that the vertices of G are in topologically sorted order, that is, any directed edge $(V[i], V[j]) \in E$ implies $i < j$.

[Hint: Use dynamic programming to build a table of longest distances to $V[n]$ from every vertex in G.]

3. Consider the following decision problem:

APPROXIMATE SUBSET SUM
Input: A list $\langle a_1, \ldots, a_n \rangle$ of positive integers and positive integers t and E (all given in binary).
Question: Is there a subset $J \subseteq \{1, \ldots, n\}$ such that

$$\left| t - \sum_{j \in J} a_j \right| \leq E \; ?$$

Show that APPROXIMATE SUBSET SUM is NP-hard by giving a polynomial reduction from SUBSET SUM.

Theory

Not given in Fall 2013.