Fall 2014 CSE Qualifying Exam
Core Subjects

October 4, 2014

Architecture

1. Consider the following loop, written in C:

```
for (i=0; i>n; i++) {
     x[i]=0;
     for (j=0;j<4;j++)
          x[i] = coeff[j] + x[i];
}
```

   Sketch an implementation of this loop on the following types of architectures:

   - SIMD
   - VLIW
   - Vector

   For each type, estimate the expected performance. You may use metrics such as floating point operations per second, instructions per second, instructions per floating point operation, or speedup relative to a reference serial implementation.

2. Consider the following three hypothetical processors, for which we characterize their performance with a benchmark:

   (a) A simple MIPS two-issue static pipe running at a clock rate of 4 GHz and achieving a pipeline CPI of 0.8. This processor has a cache system that yields 0.005 misses per instruction on average.

   (b) A deeply pipelined version of the two-issue MIPS processor with slightly smaller caches and a 5 GHz clock rate. The pipeline CPI of the processor is 1.0, and the smaller caches yield 0.0055 misses per instruction on average.

(c) A speculative superscalar with a 64-entry window but achieves an average issue rate of 4.5. This processor has the smallest caches, which lead to 0.01 misses per instruction, but it hides 25% of the miss penalty on every miss by dynamic scheduling. This processor has a 2.5 GHz clock.

Assume that the main memory time (which sets the miss penalty) is 50 ns. Determine the relative performance of the three processors. State any assumptions.

3. In this problem we will compare the performance of a vector processor with a system that contains a scalar processor and a GPU-based coprocessor. In the hybrid system, the host processor has superior scalar performance to the GPU, so in this case all scalar code is executed on the host processor while all vector code is executed on the GPU. We will refer to the first system as the vector computer and the second system as the hybrid computer.

   Assume your target application contains a kernel with an arithmetic intensity of 0.5 FLOPs per DRAM byte accessed. However, the application also has a scalar component which must be performed before and after the kernel in order to prepare the input vectors and output vectors, respectively.

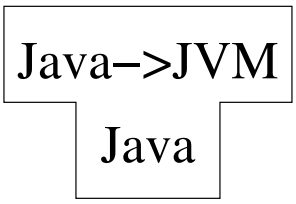
   For a sample dataset, the scalar portion of the code requires 400 ms of execution time on both the vector processor and the host processor in the hybrid system. The kernel reads input vectors consisting of 200 MB and has output data consisting of 100 MB.

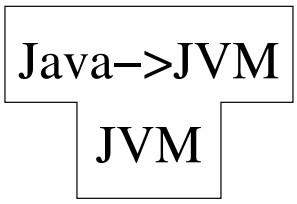
   The vector processor has a peak memory bandwidth of 30 GB/s and the GPU has a peak memory bandwidth of 150 GB/s. The hybrid system has an additional overhead that requires all input vectors to be transferred between the host memory and GPU local memory before and after the kernel is invoked. The hybrid system has a DMA bandwidth of 10 GB/s and an average latency of 10 ms.

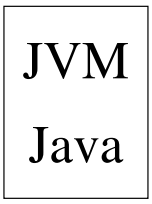
   Assume that both the vector processor and GPU are both performance bound by memory bandwidth. Compute the execution time for both computers for this application.

Compilers

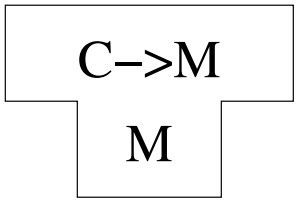
1. Assume that you have a "portable compiler kit" that contains the following components, described using tombstone diagrams:



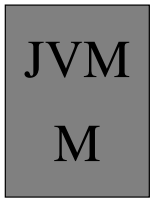




   (a) Suppose that you want to run this kit on some machine M for which there is a C compiler, described by the following tombstone diagram:



   Explain how the C compiler is used to obtain the following interpreter:



   (b) Describe the resulting interpretive compiler for machine M, using tombstone diagrams to explain how you would compile program P written in Java on machine M.

   (c) Provide a way to bootstrap the interpretive compiler to produce a native compiler whose source is Java and whose target is M. Use tombstone diagrams to explain your scheme. Clearly explain what needs to be implemented in your scheme, and what can be reused.

   (d) There are two common solutions to the previous question. One results in a two-phase compiler, while the other results in a one-phase compiler. If you obtained the two-phase compiler, also describe how the one-phase compiler can be obtained. If you obtained the one-phase compiler, also describe how the two-phase compiler can be obtained.

2. 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	Associativity
`~ !`	none
`* / %`	left
`< <= > >=`	none
`== !=`	none
`&&`	left
`||`	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.

3. 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?

Algorithms

1. Find tight asymptotic bounds on any positive-valued function $T(n)$ satisfying the following recurrence for all $n \geq 4$:

$$T(n) = 4T(\lfloor n/4 \rfloor) + \frac{n}{\lg n} .$$

That is, find an expression $f(n)$, as simple as possible, such that $T(n) = \Theta(f(n))$. Show your work. You may safely ignore the floor.

2. The country of Fredonia has a unit of currency called the Fred and coins of $k > 0$ different denominations $d_1 < d_2 < \cdots < d_k$, which are all positive integral numbers of Freds with $d_1 = 1$. Describe an algorithm that, given a positive integer n and a sorted list $\langle 1, d_2, \ldots, d_k \rangle$ of distinct positive integers starting with 1, returns a shortest possible list of coins from the list that add up to n Freds. For example, if $k = 3$ and $\langle d_1, d_2, d_3 \rangle = \langle 1, 8, 13 \rangle$, then here are two of the possible ways of obtaining 20 Freds:

$$\begin{aligned} 20 &= 1 + 1 + 1 + 1 + 1 + 1 + 1 + 13 \\ &= 1 + 1 + 1 + 1 + 8 + 8 . \end{aligned}$$

The second sum has the fewest possible terms, and so your algorithm should return $\langle 1, 1, 1, 1, 8, 8 \rangle$ (the order within the list is not important). Ties for the shortest list may be broken arbitrarily. Your algorithm should run in time $O(nk)$, and you may assume that all arithmetic operations, comparisons, array accesses, and basic list operations each take $O(1)$ time. [Hint: Ask yourself if this problem displays optimal substructure.]

3. Describe Dijkstra's algorithm for single-source shortest paths in a directed graph. Give enough detail so that a reasonably good programmer with a sophisticated knowledge of data structures can implement the algorithm efficiently given your description. What conditions on the digraph are necessary to guarantee correctness?

Theory

1. Show that every regular language is recognized by an ε-NFA whose transition graph has maximum out-degree at most two. That is, for any alphabet Σ and any regular language $L \subseteq \Sigma^*$, there is a nondeterministic finite automaton $N = \langle Q, \Sigma, \delta, q_0, F \rangle$ with ε-moves, where $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to \mathcal{P}(Q)$, such that $L = L(N)$ and for any $q \in Q$,

$$\bigcup_{a \in \Sigma \cup \{\varepsilon\}} \delta(q, a)$$

contains at most two states. (Recall that $\mathcal{P}(Q)$ is the power set of Q, i.e., the set of all subsets of Q.) [Hint: Show how to transform an arbitrary ε-NFA into an equivalent one satisfying the constraints above.]

2. We assume the standard model of Turing machine with a single one-way infinite tape (infinite going to the right).

   Say that a Turing machine *moves right* during a computation whenever the head moves from a cell to the cell immediately to its right.

   (a) Let

   $$L_1 = \{\langle M, k \rangle \mid M \text{ is a TM that moves right at least } k \text{ many times on input } \varepsilon\} \ .$$

   Show that L_1 is decidable.

   (b) Let

   $$L_2 = \{\langle M \rangle \mid M \text{ is a TM that moves right infinitely many times on input } \varepsilon\} .$$

   Show that L_2 is undecidable.

3. Show that if $L \in \text{NP}$, then $L^* \in \text{NP}$. [Recall that L^* is the Kleene $*$-closure of L.]