Fall 2011 CSE Qualifying Exam
Core Subjects

September 24, 2011

Architecture

1. The following loop is the so-called DAXPY loop (double-precision aX plus Y) and the central operation in Gaussian elimination. The following code implements the DAXPY operation, $Y = aX + Y$, for a vector length 100. Initially, `R1` is set to the base address of array `X` and `R2` is set to the base address of `Y`.

```
        DADDIU  R4,R1,#800      ; R1 = upper bound for X
foo:    L.D     F2,0(R1)        ; (F2) = X(i)
        MUL.D   F4,F2,F0        ; (F4) = a*X(i)
        L.D     F6,0(R2)        ; (F6) = Y(i)
        ADD.D   F6,F4,F6        ; (F6) = a*X(i) + Y(i)
        S.D     F6,0(R2)        ; Y(i) = a*X(i) + Y(i)
        DADDIU  R1,R1,#8        ; increment X index
        DADDIU  R2,R2,#8        ; increment Y index
        DSLTU   R3,R1,R4        ; test:  continue loop?
        BNEZ    R3,foo          ; loop if needed
```

Assume the functional unit latencies as shown in the Table below. Assume a 1-cycle delayed branch that resolves in the ID stage. Assume results are fully bypassed.

Instruction producing result	**Instruction using result**	**Latency in clock cycles**
FP multiply	FP ALU op	6
FP add	FP ALU op	4
FP multiply	FP store	5
FP add	FP store	4
Integer operations and all loads	Any	2

(a) Assume a single-issue pipeline. Show how the loop would look both unscheduled by the compiler and after compiler scheduling for both floating-point operation and branch delays, including any stalls or idle clock cycles. What is the execution time (in cycles) per element of the result vector, `Y`, unscheduled and scheduled? How much faster must the clock be for processor hardware alone to match the performance improvement achieved by the scheduling compiler (neglect the possible increase in the number of clock cycles necessary for memory system access effects of higher processor clock speed on memory system performance)?

(b) Assume a VLIW processor with instructions that contain five operations, consisting of two memory references, two floating-point operations, and one integer or branch operation. We will compare two degrees of loop unrolling. First, unroll the loop 6 times to extract ILP, and schedule it without any stalls (i.e., completely empty issue cycles), collapsing the loop overhead instructions, and then repeat the process but unroll the loop 10 times. Ignore the branch delay slot. Show the

two schedules. What is the execution time per element of the result vector for each schedule? What percent of the operation slots are used in each schedule? How much does the size of the code differ between the two schedules? What is the total register demand for the two schedules?

2. You are trying to decide on a new processor for a specific application. After profiling this application on your old processor, you find it has the following characteristics:

   - The application can be modified to run in multi-threaded mode, in which only 1% of the total execution time must be serialized. In this case, you need one processor for each thread. However, since the application is memory intensive, multithreading it for a single-chip multiprocessor would create a memory bottleneck that would slow down each individual thread by $4n\%$, where n = number of concurrent threads (e.g. using four threads would cause each individual thread to run 16% slower).
   - When running in multi-threaded mode, 90% of the execution time can be accelerated using a SIMD processing unit that speeds up the vector code by a factor of n, where n = number of SIMD lanes. However, using a SIMD unit places additional demands on the memory interface, resulting in a 30% loss of SIMD throughput for every four SIMD lanes added.

   Assuming all other performance factors are equal, what is the speedup achieved by each of the following proposed processors relative to the old processor?

   (a) 1. Processor 1 has eight cores and each core has a four-wide SIMD unit.

   (b) 1. Processor 2 has four cores and each core has an eight-wide SIMD unit.

   (c) 1. Processor 3 has two cores and each core has a four-wide SIMD unit. However, since these processors are inexpensive, six of them can be purchased for the same price as either of the above processors. These six processors can then be arranged as a cluster computer and parallelized using a message-passing model as opposed to a shared memory model. Due to I/O contention in the cluster interconnect, each thread now experiences a slowdown of $6n\%$, where n = total number of threads.

3. You're a computer architect at Qualcomm designing the next generation Snapdragon processor for smart phones. The Android team at Google just notified you that the new version of the Android OS, "Ice Cream Sandwich," will feature a new user interface that is more memory intensive than previous versions.

   At this time, your processor contains four cores and has a shared write-back last-level cache that consumes 2.3 μJ of energy for a clean miss and 3.7 μJ for a dirty miss. At this point in time you have the option of doubling the number of cache lines for the new processor. You estimate that this will reduce the miss rate by 60% but will consume an additional 1.2 μW of static power and add 1 μJ of energy to each type of

miss. Assuming half of all misses are dirty, what miss rate must be reached under the smaller cache to make the larger cache more energy efficient?

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	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.

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 it is for.

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

3. This question is about the (two-way) conditional ("if then else") statement.

(a) Show that the following BNF grammar (similar to, but simpler than, the ones used in Algol and C) for the conditional statement is ambiguous. (Hint: "dangling else".)

```
<ifStatement> ::= if (<expression>) <statement> |
                  if (<expression>) <statement> else <statement>
<statement> ::= <assignment> | <ifStatement> | <block>
<block> ::= {<statements>}
<statements> ::=   <statements><statement> | <statement>
```

(b) In Java, the ambiguity is resolved by introducing a new nonterminal symbol, `<statementNoShortIf>`. Give a modified grammar for the one-way and two-way conditional that eliminates the dangling else ambiguity and argue that it is not ambiguous. Do not give explicit production rules that define `<statementNoShortIf>`, but simply explain its meaning as a category of statements.

(c) Other languages eliminate the ambiguity of conditional statement by introducing a delimiter that indicates the conclusion of an "if then" or "if then else" statement. Let the delimiter be `fi`. Rerwrite the rules for the conditional statement to eliminate the ambiguity.

(d) A variation of the technique just described is used in Ada, in the following EBNF grammar, where the braces and square brackets and metalevel symbols indicating "zero or more" and "optional" respectively:

```
<ifStatement> ::= if <expression> then <block>
                  {elseif <expression> then <block>} [else <block>] end if;
```

Show that this technique also eliminates the "dangling else" ambiguity.

Algorithms

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

$$T(n) = nT(\sqrt{n}) + n^2 \ .$$

You may assume that any implicit floors or ceilings never have any effect (that is, $n = 2^{2^k}$ for some natural number k). You answer should be of the form $\Theta(f(n))$, where $f(n)$ is expressed as simply as possible. [Hint: consider a change of variables.]

Show your work.

2. Describe a sequential algorithm MULTIMERGE that takes a collection $L_1, \ldots, L_k$ of nonempty (simply) linked lists of numbers, where for each i the list L_i is sorted in ascending order, and merges them into a single sorted linked list. You may destroy the original lists in the process. Your algorithm should run in time $O(n \lg k)$, where n is the total number of items in all the lists combined. Your algorithm should also use $O(k)$ extra space besides the input. (Assume that each number uses $O(1)$ space.)

Your description must include a high-level description of the basic strategy employed. You are not required to prove either the correctness of your algorithm or its time and space bounds.

3. Describe an explicit polynomial reduction from SUBSET-SUM to PARTITION, where both decision problems are defined below:

SUBSET-SUM
Instance: A list $\langle a_1, \ldots, a_n \rangle$ of positive integers and a positive integer t (all given in binary).
Question: Is there a subset $S \subseteq \{1, \ldots, n\}$ such that $\sum_{i \in S} a_i = t$?

PARTITION
Instance: A list $\langle a_1, \ldots, a_n \rangle$ of positive integers (all given in binary).
Question: Is there a subset $S \subseteq \{1, \ldots, n\}$ such that $\sum_{i \in S} a_i = (1/2) \sum_{i=1}^{n} a_i$?

You are not required to prove that your reduction is correct to get full credit, but if your answer is incorrect, some explanation may earn you partial credit.

Theory

1. Fix an integer $k \geq 2$, and let $\Sigma = \{0, 1, \ldots, k-1\}$. The *lexicographic ordering* (also called the *dictionary ordering*) on Σ^* is defined as follows: For all $x, y \in \Sigma^*$, we say that $x \leq_{\text{lex}} y$ iff either x is a prefix of y or else there exist a string $w \in \Sigma^*$ and symbols $a, b \in \Sigma$ such that

   - wa is a prefix of x,
   - wb is a prefix of y, and
   - $a < b$.

   You may take it for granted that $\leq_{\text{lex}}$ is a total ordering on Σ^*.

   (a) (50%) Show that, for any string $w \in \Sigma^*$, the language

   $$\leq_{\text{lex}}(w) := \{x \in \Sigma^* : x \leq_{\text{lex}} w\}$$

   is regular.

   (b) (50%) For any language $A \subseteq \Sigma^*$, define the language

   $$\leq_{\text{lex}}(A) := \{w \in \Sigma^* : (\exists x \in A)\ w \leq_{\text{lex}} x\}.$$

   Show that if A is regular, then $\leq_{\text{lex}}(A)$ is regular.

2. For this question, we fix a finite alphabet Σ that does not contain the symbol #. Recall that $\Sigma^*\#\Sigma^* = \{x\#y : x, y \in \Sigma^*\}$.

   Let $A \subseteq \Sigma^*\#\Sigma^*$ be a Turing-recognizable language such that for every $x \in \Sigma^*$ there exists $y \in \Sigma^*$ such that $x\#y \in A$.

   (a) (75%) Show that there exists a language B such that

   i. $B \subseteq A$,
   ii. B is Turing-recognizable, and
   iii. for every $x \in \Sigma^*$, there exists *exactly one* $y \in \Sigma^*$ such that $x\#y \in B$.

   (b) (25%) Show that *any* language B satisfying (2(a)i)–(2(a)iii) above is actually decidable.

3. Recall that a language A is *polynomially (mapping) reducible* to a language B (written $A \leq_m^p B$) iff there exists a polynomial-time-computable function $f : \Sigma^* \to \Sigma^*$ such that, for all $x \in \Sigma^*$,

   $$x \in A \iff f(x) \in B\ .$$

   Also recall that $\overline{A}$ denotes the complement of A (that is, $\overline{A} = \Sigma^* - A$).

   Show that if A is in NP and $A \leq_m^p \overline{A}$, then $\overline{A}$ is also in NP.