Computer Science Ph. D. Qualifier Examination
Spring 2008
Programming Languages and Compilers Area

Note:
1. Pick and answer any 6 out of 9 questions below.
2. Answers must be crisp, to the point and contain details of sufficient magnitude. Clear algorithms and pseudo-codes must accompany answers wherever necessary to bring out the generality of the answer.
3. State suitable assumptions as necessary highlighting them.
4. Questions that are open ended need a discussion and answers should factor that in. Answers must bring out critical issues clearly in such cases.
5. Draw figures, flow-graphs, as necessary to bring out the details of the example used.

1. **[Pointer Analysis]**
   Pointer alias analysis is one of the most important analyses in compilers, and many algorithms have been proposed to solve the problem.

   a) Why is this analysis so important?
   b) What are the primary distinctions that differentiate the various proposed algorithms for pointer alias analysis?
   c) If you were designing a compiler for embedded media applications (which are typically written using many heap allocated structs and global allocated arrays), what characteristics would be most important for your pointer alias analysis? Why?

2. **[Profiling]**
   What program information is typically collected in modern profiling compilers? Describe two compiler optimizations that are either enabled or greatly enhanced by the use of profiling information. Include examples.

3. **[Pipeline parallelism]**
   Exploiting pipeline parallelism (e.g., via decoupled software pipelining) is an effective technique to automatically parallelize loops that have dependencies between successive iterations. Describe some of the characteristics the compiler tries to optimize for when creating a thread decomposition using pipeline parallelism. For example, in traditional DOALL style automatic parallelism, where the compiler creates threads from separate loop iterations, communication between threads is often minimized.
4. [Efficient path profiling]

Draw the corresponding control flow graph for the following source code. Instrument the code to collect path profiling data. Use “Ball-Larus” algorithm to find the minimum number of instrumented instructions. Write assembly (any ISA you like) if you need.

```c
if (a == 0) {
    if (b == 0) {
        c++;
    }
} else {
    if (a == 1 || b == 1) {
        c = 0;
    } else {
        c = 1;
    }
}
```

5. [Static Dependence Analysis]

Static dependence analysis has limitations due to the scope of the underlying analysis (local, intra-procedural vs. inter-procedural). Show these limitations through examples using GCD and Banerjee Tests. Show what type of inter-procedural analysis would be most useful to get around these limitations.

6. [Parallel Languages] These days, CUDA is becoming one of the most popular parallel program languages. What are the major differences between CUDA and OpenMP/MPI in terms of program languages? Why do you think CUDA is becoming popular?
7. [SSA and Optimizations]

Compilers reason on the name space of the programs and perform optimizations. Inherent name spaces of the program pose limitations to the analysis. Through examples, show how inherent-name space based analyses run into limitations and may miss out on certain optimization opportunities. Show how SSA-based analysis may get around these limitations explaining why. Propose a variant on SSA that does even better value-name-space separation than SSA and show its utility through a (better) optimization opportunity exposed.

8. [Register-pair allocation]

On a peculiar embedded architecture, registers can be only allocated in terms of a register pair. R1-R2 defines a register pair (each register is 16-bit in terms of width) and assume that a certain number of register pairs exist in the architecture. Two types of load/stores instructions exist: LDW/STW R1-R2, @M can load or store a 32 bit word located at M into R1-R2 with high 16 bit part loaded in R1 and low 16 bit part loaded in R2. On the other hand, LDHW/STHW R1-R2 @M behaves as follows: only high 16-bit half word located at M is loaded into or stored from R1; R2 remains unmodified in a load or is not stored in a store. In addition, SHL R1-R2 and SHR R1-R2 instructions are supported that perform a 16 bit left shift (contents of R2 are shifted to R1 and original contents of R1 are lost, new contents of R2 become NULL) or a right shift (works in a similar way except contents of R1 are shifted to R2 and new contents of R1 become NULL and original contents of R2 are lost). The goal of this instruction set is to allow compact generation of load/stores but in order to do so, data layout optimization will have to be performed. Most of the operands on this architecture are 16 bit wide. Devise a framework to maximize generation of LDW/STW instructions: expected answer should contain data layout decision algorithm as well as a register allocation algorithm.

9. [Branch Hint/Codegen]

Branch hint feature on Cell processor allows the compiler to hint a potential outcome of the branch. How to generate a good branch hint is an open problem but is critical to the performance of the Cell. Study and classify different processor architectures that use different types of branch hints/outcome related fields that are to be filled in by the compiler (we will not worry about the prediction of the branch targets in this discussion – typically it is done by BTB in hardware). Discuss potential compiler analyses that could be used to populate the branch hints/outcomes for each of the above cases. Give more details for Cell’s code generation problem in terms of how to most effectively generate branch hints/outcome-related fields.