Exam in EDAF15 Algorithm Implementation

June 3, 2010, 14-18

Inga hjälpmedel!

Examinator: Jonas Skeppstedt

30 out of 60p are needed to pass the exam.

1. (10p) Pipelining

   (a) (3p) Does pipelining reduce the number of clock cycles to execute an instruction, or what is the purpose of pipelining?

   Answer *No, the number of clock cycles remains the same. The purpose is to start a new instruction every clock cycle so a new result can be produced every clock cycle. This does not always work due to various delays, e.g. in the following sequence, since the data will not arrive to the add instruction in time:*

   ```
   load    R1, R2, R3    // loads data into R1
   add     R4,R1,R5      // uses R1
   ```

   (b) (4p) Why is it important to design a pipeline so that the different pipeline stages need approximately the same time to perform their work?

   Answer

   The clock cycle time must be sufficiently large so the slowest pipeline stage can complete its work. Thus, there is no point in making one stage faster than the others since the slowest will determine the clock frequency.

   (c) (3p) What is branch-prediction and which performance problem is it aimed at reducing?

   Answer

   Branch-prediction is a hardware mechanism which guesses whether a branch will be taken or not, and starts fetching and executing instructions in the predicted path. Such instructions are called speculative and are cancelled if it turns out the guess was wrong. Such instructions may not modify state (e.g. memory) before it's certain they should be executed.

   The performance problem it can reduce is waiting for the fetching of instructions that are located where a branch jumps to.

2. (10p) Cache Memories

   (a) (4p) Which two types of locality of references are exploited with caches to reduce the execution times of programs? Give examples of C code fragments in which each type of locality can be exploited.

   Answer

   Temporal and spatial locality. See book for examples.

   (b) (4p) In C programs with nested for-loops and matrices it is often important to try to order the loops to improve performance. Why and what is the goal?

   Answer

   It is important to have the innermost loop's index variable select a matrix column and not a matrix row. The reason for this is that matrices in C (and most other languages but not FORTRAN) are put in memory one row at a time, i.e. all elements in one row are close to each other while elements in different rows are not. Accessing the same row thus can result in spatial locality.

   (c) (2p) What does cache associativity mean?

   Answer

   A cache is divied into a number of sets and an address maps to one particular set. The number of cache lines, N, in one set is its associativity. With N larger than one, it's less likely that two different variables will overwrite each other since they can be present in the same set at the same time.

3. (20p) C

   (a) (3p) Memory allocated with `alloca` or for variable length arrays should not be returned to the calling function. Why?

   Answer

   The lifetime ends when the function returns since they are allocated from the stack.

   (b) (3p) Why is `alloca` so much faster than `malloc`?

   Answer

   It simply adjusts the stack pointer while `malloc` *must search for a suitable memory block in its data structure.*

(c) (1p) Write a recursive function in C to compute factorials.

Answer

```
int f(int n)
{
        if (n <= 1)
                return 1;
        else
                return n * f(n - 1);
}
```

(d) (5p) Which data in your program from the previous question do you expect the compiler will save in memory for later use? Where is it saved? Draw a simple picture!

Answer

It needs to save the parameter n and the return address. See book in Chapter 1 or slides.

(e) (8p) Write a function to append a list to the end of another list, where a list node is represented by:

```
typedef struct list_t list_t;
struct list_t {
        list_t* succ;
        list_t* pred;
        void*   data;
};
```

The lists are circular and an empty list is represented by `NULL`. The function `append` should have return type `void`. Specify suitable types of the two parameters to the function.

Answer

```
void append(list_t** list1, list_t* list2)
{
        list_t*        p;

        if (*list1 == NULL)
                *list1 = list2;
        else if (list2 != NULL) {
                (*list1)->pred->succ    = list2;
                list2->pred->succ       = *list1;
                p                       = (*list1)->pred;
                (*list1)->pred          = list2->pred;
                list2->pred             = p;
        }
}
```

4. (20p) Tuning

   (a) (6p) Important words in performance optimization are

   make the common case fast

   What is meant by that, and which tools would you use for this on Linux and MacOS X, and which information can they provide?

   Answer

   See book and slides. Shark is similar to Oprofile except that it has a different user interface.

   (b) (14p) Assume you discover that it is the dynamic memory allocation which takes most of the time due to you allocate numerous (1) short strings and (2) objects of type `struct s { int a, b; }`.

   Both the strings and the objects of type `struct s` may be used in the rest of the program's execution. Implement alternative memory allocation to speed up your program.

   Answer

   Brief solution without implementation follows:

   For instance, an arena for the strings (see slides for how to implement an arena), and an array for the structs will be faster than using malloc. The arena is better than malloc since short strings don't have to be aligned and should avoid the overhead (both size and time) of malloc. An array of such structs can be allocated and a variable can be used to keep track of which element to allocate next. The array can be allocated with malloc or calloc. If one array is not sufficient, one can allocate another array etc. These arrays should be put in a list so they can be deallocated before the program terminates. The purpose of deallocating them explicitly before program termination is to avoid risking to introduce a memory leak in case the program will be modified in the future. However, calling free just before calling exit is strictly not necessary but it is strongly recommended practise.