COMP206 **Class Project** Assigned: November 20, 2002
Prof. Montek Singh Due: December 4, 2002

Instructions:

1. You may work on this project either individually, or in teams of two. If you work in a group, only one written report should be submitted, with names of both members of the team.

2. This project consists of two parts. Part I deals with branch prediction. Part II deals with caching.

3. Each part has several questions, some of which are marked *"Extra Credit."*

   - If you are working alone on this project, these extra credit questions are completely optional for you, but you will receive extra credit if you decide to attempt them.[1]
   - However, *if you are working in a team, you must attempt the extra credit questions; i.e.,* all questions are compulsory.

[1]Extra credit earned on the project will be kept separate from your grade on this project, but will help you partly compensate for a not-so-good performance on homework assigments or midterm exam.

1. **(PART I)** [65 points] This part deals with branch prediction. Consider the following C program:

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>

hrtime_t _time_start, _time_end;
#define tick()  (_time_start = gethrvtime())
#define tock()  (_time_end   = gethrvtime())
#define elapsed_nanoseconds()  (_time_end - _time_start)

#define SIZE 10000
#define THRESH 200

int array[SIZE];

int main()
{
  int i, j, k, sum;

  for (k = 0; k < 1000; k++) {
    for (i = 0; i < SIZE; i++) array[i] = k*drand48(); /* Init loop */

    tick();
    for (sum = i = 0; i < SIZE; i++) { /* Loop-back branch */
      j = array[i];
      if (j >= THRESH) sum &= (THRESH-1); /* Conditional branch */
      sum += j;
    }
    tock();

    fprintf(stderr, "", sum);
    printf("%5d \t%lld\n", k, elapsed_nanoseconds());
  }
  return 0;
}
```

Explanation: The loop marked `/* Init loop */` initializes the array with uniformly distributed random numbers in the range $[0, k)$.

(a) Briefly explain what the program does, and how. (*Hint:* The program's output consists of two columns, the first column gives k, and the second column gives the runtime of the innermost loop for that value of k.)

(b) Quantify the predictability of the branch marked `/* Conditional branch */` as a function of k and THRESH.

(c) Note that there are two branches in the code—the loop-back branch and the conditional branch—and that they alternate in time. Characterize the behavior of an (m, n) correlating branch predictor as a function of m. In particular, does choosing a non-zero value of m improve prediction accuracy?

(d) Compile the code on a Sun SPARC machine (*e.g.,* capefear.cs.unc.edu, uncle-leo.cs.unc.edu, etc.). Use *gcc* as your compiler, and generate four different executable by selecting the following optimization switches in *gcc*:

i. gcc -O0 proj.c
ii. gcc -O1 proj.c
iii. gcc -O2 proj.c
iv. gcc -O3 proj.c

Which optimizations are selected by each of these command-line options?

(e) Once you have generated the four different executables, run them using the *ptime* utility:

```
$ /usr/proc/bin/ptime a.out > out
```

Plot the program output, *e.g.,* by importing the data into a spread-sheet. Try to use a "scatter plot," *i.e.,* do not connect the points by lines; simply plot the points so that the plots look clean.

(f) Analyze the four plots generated above, and give brief explanations. Identify *all trends* in the plots and try to account for them.

(g) **(Extra Credit)** Modify the code to add at least one extra conditional branch. Be sure that the modified code will indeed have the extra branch, in addition to the branches already there, even after aggressive code optimization by gcc. That is, it should be non-trivial for the compiler to "optimize away" your new branch. Try to be creative in how you select the new branch. In particular, the characteristics of the new branch should be fairly different from those of the branches already

present in the code, and you must be able to indentify these characteristics from the runtimes generated by the program.

(h) **(Extra Credit)** Generate assembly code for the original program code using appropriate switches for *gcc*. Briefly explain the differences in the runtime plots obtained above using knowledge of the assembly code generated. You may refer to the SPARC 9 architecture manual at http://www.sparc.com/resource.htm for information on the instruction set.

2. **(PART II)** [35 points] This part deals with reducing cache misses using *blocking*.

(a) Refer to pages 433–434 of Hennessy/Patterson 3rd ed. The textbook gives an example of how the matrix multiplication operation can be made to run faster by using the technique of blocking, which exploits temporal locality. Design an experiment with the same, or similar, program fragments and quantify the benefit of this optimization.

(b) Generate plots of runtimes for different blocking factors.

(c) Analyze the plots generated and briefly discuss what you observe.

(d) **(Extra Credit)** What can you conclude about the size of the cache from your experiments?