Indian Institute of Information Technology, Vadodara

Parallel Programming(CS403)

Prof. Reshmi Mitra

Name - Hemant Kumar
ID - 201352026
Collaborator - Dilip Puri(201351014)

Lab 06

Submission Date - November 25, 2016 **Deadline** - Nov 26, 11:59 PM

Introduction

In computing, an optimizing compiler is a compiler that tries to minimize or maximize some attributes of an executable computer program. The most common requirement is to minimize the time taken to execute a program; a less common one is to minimize the amount of memory occupied.

Optimization Table

option	optimization level	execution time	code size	memory usage	compile time
-O0	optimization for compilation time (default)	+	+	-	-
-O1 or -O	optimization for code size and execution time	-	-	+	+
-O2	optimization more for code size and execution time	–		+	++
-O3	optimization more for code size and execution time	—		+	+++
-Os	optimization for code size		–		++
-Ofast	O3 with fast none accurate math calculations	—		+	+++

note:- +increase, ++increase more, +++increase even more, -reduce, –reduce more, —reduce even more

Methods

These options control various sorts of optimizations:
-O
-O1
Optimize. Optimizing compilation takes somewhat more time, and a lot more memory for a large function.
Without -O, the compiler's goal is to reduce the cost of compilation and to make debugging produce

the expected results. Statements are independent: if you stop the program with a breakpoint between statements, you can then assign a new value to any variable or change the program counter to any other statement in the function and get exactly the results you would expect from the source code.

With -O, the compiler tries to reduce code size and execution time, without performing any optimizations that take a great deal of compilation time.

-O2
Optimize even more. GCC performs nearly all supported optimizations that do not involve a space-speed tradeoff. The compiler does not perform loop unrolling or function inlining when you specify -O2. As compared to -O, this option increases both compilation time and the performance of the generated code.
-O2 turns on all optional optimizations except for loop unrolling, function inlining, and register renaming. It also turns on the -fforce-mem option on all machines and frame pointer elimination on machines where doing so does not interfere with debugging.

Please note the warning under -fgcse about invoking -O2 on programs that use computed gotos.

-O3
Optimize yet more. -O3 turns on all optimizations specified by -O2 and also turns on the -finline-functions and -frename-registers options.
-O0
Do not optimize.
-Os
Optimize for size. -Os enables all -O2 optimizations that do not typically increase code size. It also performs further optimizations designed to reduce code size.
If you use multiple -O options, with or without level numbers, the last such option is the one that is effective.

Examples

```
Loop unrolling
Example:
   // old loop
   for(int i=0; i<3; i++) {
        colormap[n+i] = i;
   }
   // unrolled version
   int i = 0;
   colormap[n+i] = i;
   i++;
   colormap[n+i] = i;
   i++;
   colormap[n+i] = i;

   *and many more example which we've already given at the time of demo.
```