CSE 504 HW1
Prateek Roy SBU ID: 111481907

1. The result when CG is run on different optimization level:

Optimization Level	File Size	Branch Instr	Arithmetic Instr	Memory Instr
-o0	87137	210	152	1096
-o1	59368	142	148	251
-o2	96607	203	313	468
-o4	103920	218	349	507
-ofast	125075	235	481	591

I have submitted the .ll files for all the optimization level in the folder under cg<level>.ll
I was able to relate the output IR to the C code of CG. The IR contained mostly the instructions.

Difference in output of different optimization level :

a) As we increase the optimization level, the size of the file increases as well as number of instructions also increase because of many reasons : Deeper inner loop unrolling, Better loop scheduling, Reordering of floating-point computations.
b) At optimization level -o1 the instructions decrease as Elimination of redundant instructions, Elimination of instructions whose results are unused or that cannot be reached by a specified control flow, also known as *dead code elimination*.

2. I understood how to write the pass. Please check the code (gitdiff.txt).

__Generate the .ll file:__
Generate the .ll file on which you want to run your pass on by **clang -emit-llvm -S hello.c**

__Run a LLVM pass:__
Run your llvm pass(libSkeletonPass.*) on the generated .ll file
opt -load ~/llvm-pass-skeleton/build/skeleton/libSkeletonPass.* -mypass ~/NPB3.0-omp-C/CG/bk/hello.ll

3. My LLVM pass(mypass : gitdiff.txt) to count the number of branch, arithmetic, memory instructions in the CG code (function wise):

Function Name	Branch Inst	Arithmetic Inst	Memory Inst
main()	56	44	213
makea()	34	14	195
conj_grad()	45	51	267
sprnvc()	15	5	78
vecset()	9	2	49
sparse()	50	35	288
icnvrt()	1	1	6

Branch Instructions : I have considered BranchInst, TerminatorInst class type to identify the type of the operand in the instruction.

Memory Instructions: I have considered AllocaInst, LoadInst, StoreInst, AtomicCmpXchgInst AtomicRMWInst, FenceInst, GetElementPtrInst as type of operator in Instruction to identify it as memory instruction.

Arithmetic Instructions: I have considered BinaryOperator as type of operator in instruction to identify it as arithmetic instruction.

Sample Code File(hello.c): This is additional test cases.
i) I used a function **loopfun**() which had a for loop and a increment statement. So, it had branch, arithmetic and memory operations.

ii) **branchfun**() had assignment(memory) operation and if else statement which is branch instruction.

Function	Branch Inst	Arithmetic Inst	Memory Inst
loopfun()	5	2	11
branchfun()	4	0	3
main()	1	0	2