Bonus

September 16, 2020

1 Bonus question

Raj Patil
CS18BTECH11039

[7]:
```
from time import perf_counter as pc
import numpy as np
import matplotlib.pyplot as plt
from collections import namedtuple
```

2 nomenclature:

clg: clang related
gcc: gcc related
prp: preprocessing times
prs: parsing times
cg_ot: optimization times with code generation: using O2
cg: code generation times with no optimization

NOT USING old files and beginning compilation again at all times

[65]:
```
num_runs = range(50)
```

3 Clang toolchain

[64]:
```
#inbuilt analysis tools
!time clang++ -ftime-report test4.cpp
```

```
===-------------------------------------------------------------------------===
                         Miscellaneous Ungrouped Timers
===-------------------------------------------------------------------------===

   ---User Time---   --System Time--   --User+System--   ---Wall Time---  ---
Name ---
```

```
   0.0580 ( 68.2%)   0.0000 (  0.0%)   0.0580 ( 67.6%)   0.0580 ( 66.6%)  Code
Generation Time
   0.0271 ( 31.8%)   0.0007 (100.0%)   0.0278 ( 32.4%)   0.0290 ( 33.4%)  LLVM
IR Generation Time
   0.0851 (100.0%)   0.0007 (100.0%)   0.0858 (100.0%)   0.0871 (100.0%)  Total

===--------------------------------------------------------------------------===
                      Instruction Selection and Scheduling
===--------------------------------------------------------------------------===
  Total Execution Time: 0.0042 seconds (0.0042 wall clock)

   ---User Time---   --User+System--   ---Wall Time---  --- Name ---
   0.0009 ( 21.9%)   0.0009 ( 21.9%)   0.0009 ( 21.8%)  Instruction Selection
   0.0009 ( 21.2%)   0.0009 ( 21.2%)   0.0009 ( 21.0%)  Instruction Scheduling
   0.0007 ( 16.5%)   0.0007 ( 16.5%)   0.0007 ( 16.9%)  DAG Combining 1
   0.0005 ( 12.8%)   0.0005 ( 12.8%)   0.0005 ( 12.9%)  Instruction Creation
   0.0005 ( 11.2%)   0.0005 ( 11.2%)   0.0005 ( 11.1%)  DAG Combining 2
   0.0003 (  6.4%)   0.0003 (  6.4%)   0.0003 (  6.3%)  DAG Legalization
   0.0003 (  6.0%)   0.0003 (  6.0%)   0.0003 (  6.1%)  Type Legalization
   0.0001 (  1.9%)   0.0001 (  1.9%)   0.0001 (  2.0%)  Instruction Scheduling
Cleanup
   0.0001 (  2.0%)   0.0001 (  2.0%)   0.0001 (  2.0%)  Vector Legalization
   0.0042 (100.0%)   0.0042 (100.0%)   0.0042 (100.0%)  Total

===--------------------------------------------------------------------------===
                                 DWARF Emission
===--------------------------------------------------------------------------===
  Total Execution Time: 0.0010 seconds (0.0011 wall clock)

   ---User Time---   --User+System--   ---Wall Time---  --- Name ---
   0.0006 ( 59.1%)   0.0006 ( 59.1%)   0.0006 ( 57.9%)  DWARF Exception Writer
   0.0004 ( 40.7%)   0.0004 ( 40.7%)   0.0005 ( 41.9%)  Debug Info Emission
   0.0000 (  0.2%)   0.0000 (  0.2%)   0.0000 (  0.2%)  DWARF Debug Writer
   0.0010 (100.0%)   0.0010 (100.0%)   0.0011 (100.0%)  Total

===--------------------------------------------------------------------------===
                      … Pass execution timing report …
===--------------------------------------------------------------------------===
  Total Execution Time: 0.0381 seconds (0.0381 wall clock)

   ---User Time---   --User+System--   ---Wall Time---  --- Name ---
   0.0118 ( 31.0%)   0.0118 ( 31.0%)   0.0118 ( 31.0%)  X86 DAG->DAG Instruction
Selection
   0.0051 ( 13.3%)   0.0051 ( 13.3%)   0.0051 ( 13.4%)  X86 Assembly Printer
   0.0026 (  6.8%)   0.0026 (  6.8%)   0.0026 (  6.8%)  Prologue/Epilogue
Insertion & Frame Finalization
   0.0016 (  4.3%)   0.0016 (  4.3%)   0.0016 (  4.3%)  Expand Atomic
instructions
```

```
   0.0017 (  4.4%)   0.0017 (  4.4%)   0.0016 (  4.3%)  Fast Register Allocator
   0.0009 (  2.4%)   0.0009 (  2.4%)   0.0010 (  2.6%)  Two-Address instruction
pass
   0.0008 (  2.0%)   0.0008 (  2.0%)   0.0008 (  2.0%)  Insert stack protectors
   0.0007 (  1.8%)   0.0007 (  1.8%)   0.0007 (  1.8%)  Dominator Tree
Construction
   0.0006 (  1.6%)   0.0006 (  1.6%)   0.0006 (  1.6%)  MachineDominator Tree
Construction
   0.0006 (  1.4%)   0.0006 (  1.4%)   0.0006 (  1.5%)  Free MachineFunction
   0.0005 (  1.3%)   0.0005 (  1.3%)   0.0005 (  1.3%)  Exception handling
preparation
   0.0005 (  1.4%)   0.0005 (  1.4%)   0.0005 (  1.3%)  Machine Natural Loop
Construction
   0.0004 (  1.2%)   0.0004 (  1.2%)   0.0004 (  1.1%)  Dominator Tree
Construction
   0.0004 (  1.1%)   0.0004 (  1.1%)   0.0004 (  1.1%)  Post-RA pseudo
instruction expansion pass
   0.0004 (  1.0%)   0.0004 (  1.0%)   0.0004 (  1.0%)  MachineDominator Tree
Construction
   0.0004 (  0.9%)   0.0004 (  0.9%)   0.0004 (  0.9%)  Expand reduction
intrinsics
   0.0003 (  0.9%)   0.0003 (  0.9%)   0.0004 (  0.9%)  X86 pseudo instruction
expansion pass
   0.0004 (  0.9%)   0.0004 (  0.9%)   0.0004 (  0.9%)  Eliminate PHI nodes for
register allocation
   0.0003 (  0.9%)   0.0003 (  0.9%)   0.0003 (  0.9%)  StackMap Liveness
Analysis
   0.0004 (  0.9%)   0.0004 (  0.9%)   0.0003 (  0.9%)  Inliner for
always_inline functions
   0.0003 (  0.9%)   0.0003 (  0.9%)   0.0003 (  0.9%)  Insert fentry calls
   0.0004 (  1.0%)   0.0004 (  1.0%)   0.0003 (  0.9%)  Machine Natural Loop
Construction
   0.0003 (  0.8%)   0.0003 (  0.8%)   0.0003 (  0.8%)  Expand indirectbr
instructions
   0.0003 (  0.8%)   0.0003 (  0.8%)   0.0003 (  0.8%)  Implement the
'patchable-function' attribute
   0.0003 (  0.8%)   0.0003 (  0.8%)   0.0003 (  0.8%)  Basic Alias Analysis
(stateless AA impl)
   0.0003 (  0.8%)   0.0003 (  0.8%)   0.0003 (  0.8%)  Insert XRay ops
   0.0003 (  0.8%)   0.0003 (  0.8%)   0.0003 (  0.8%)  Expand ISel Pseudo-
instructions
   0.0003 (  0.7%)   0.0003 (  0.7%)   0.0003 (  0.8%)  Bundle Machine CFG Edges
   0.0003 (  0.7%)   0.0003 (  0.7%)   0.0003 (  0.7%)  Instrument function
entry/exit with calls to e.g. mcount() (post inlining)
   0.0003 (  0.7%)   0.0003 (  0.7%)   0.0003 (  0.7%)  Live DEBUG_VALUE
analysis
   0.0003 (  0.7%)   0.0003 (  0.7%)   0.0003 (  0.7%)  Machine Optimization
Remark Emitter
```

```
   0.0003 (  0.7%)   0.0003 (  0.7%)   0.0003 (  0.7%)  Remove unreachable
blocks from the CFG
   0.0003 (  0.7%)   0.0003 (  0.7%)   0.0003 (  0.7%)  X86 PIC Global Base Reg
Initialization
   0.0002 (  0.7%)   0.0002 (  0.7%)   0.0003 (  0.7%)  Local Stack Slot
Allocation
   0.0003 (  0.7%)   0.0003 (  0.7%)   0.0003 (  0.7%)  Contiguously Lay Out
Funclets
   0.0003 (  0.7%)   0.0003 (  0.7%)   0.0003 (  0.7%)  Instrument function
entry/exit with calls to e.g. mcount() (pre inlining)
   0.0003 (  0.7%)   0.0003 (  0.7%)   0.0003 (  0.7%)  X86 Retpoline Thunks
   0.0003 (  0.7%)   0.0003 (  0.7%)   0.0003 (  0.7%)  X86 FP Stackifier
   0.0003 (  0.7%)   0.0003 (  0.7%)   0.0003 (  0.7%)  X86 WinAlloca Expander
   0.0002 (  0.7%)   0.0002 (  0.7%)   0.0003 (  0.7%)  Machine Optimization
Remark Emitter
   0.0002 (  0.6%)   0.0002 (  0.6%)   0.0003 (  0.7%)  Lazy Machine Block
Frequency Analysis
   0.0002 (  0.6%)   0.0002 (  0.6%)   0.0002 (  0.7%)  Lazy Machine Block
Frequency Analysis
   0.0003 (  0.7%)   0.0003 (  0.7%)   0.0002 (  0.6%)  Analyze Machine Code For
Garbage Collection
   0.0002 (  0.6%)   0.0002 (  0.6%)   0.0002 (  0.6%)  Safe Stack
instrumentation pass
   0.0002 (  0.6%)   0.0002 (  0.6%)   0.0002 (  0.6%)  X86 vzeroupper inserter
   0.0002 (  0.6%)   0.0002 (  0.6%)   0.0002 (  0.6%)  Scalarize Masked Memory
Intrinsics
   0.0002 (  0.6%)   0.0002 (  0.6%)   0.0002 (  0.6%)  Lower Garbage Collection
Instructions
   0.0002 (  0.6%)   0.0002 (  0.6%)   0.0002 (  0.6%)  Shadow Stack GC Lowering
   0.0002 (  0.5%)   0.0002 (  0.5%)   0.0002 (  0.5%)  CallGraph Construction
   0.0001 (  0.1%)   0.0001 (  0.1%)   0.0001 (  0.1%)  Assumption Cache Tracker
   0.0000 (  0.0%)   0.0000 (  0.0%)   0.0000 (  0.0%)  Pre-ISel Intrinsic
Lowering
   0.0000 (  0.0%)   0.0000 (  0.0%)   0.0000 (  0.0%)  Rewrite Symbols
   0.0000 (  0.0%)   0.0000 (  0.0%)   0.0000 (  0.0%)  Force set function
attributes
   0.0000 (  0.0%)   0.0000 (  0.0%)   0.0000 (  0.0%)  A No-Op Barrier Pass
   0.0000 (  0.0%)   0.0000 (  0.0%)   0.0000 (  0.0%)  Assumption Cache Tracker
   0.0000 (  0.0%)   0.0000 (  0.0%)   0.0000 (  0.0%)  Target Transform
Information
   0.0000 (  0.0%)   0.0000 (  0.0%)   0.0000 (  0.0%)  Target Pass
Configuration
   0.0000 (  0.0%)   0.0000 (  0.0%)   0.0000 (  0.0%)  Machine Module
Information
   0.0000 (  0.0%)   0.0000 (  0.0%)   0.0000 (  0.0%)  Machine Branch
Probability Analysis
   0.0000 (  0.0%)   0.0000 (  0.0%)   0.0000 (  0.0%)  Target Library
Information
```

```
   0.0000 (  0.0%)   0.0000 (  0.0%)   0.0000 (  0.0%)  Profile summary info
   0.0000 (  0.0%)   0.0000 (  0.0%)   0.0000 (  0.0%)  Target Library
Information
   0.0000 (  0.0%)   0.0000 (  0.0%)   0.0000 (  0.0%)  Create Garbage Collector
Module Metadata
   0.0381 (100.0%)   0.0381 (100.0%)   0.0381 (100.0%)  Total

===-------------------------------------------------------------------------===
                          Clang front-end time report
===-------------------------------------------------------------------------===
  Total Execution Time: 0.8360 seconds (0.8366 wall clock)

   ---User Time---   --System Time--   --User+System--   ---Wall Time---  ---
Name ---
   0.7756 (100.0%)   0.0604 (100.0%)   0.8360 (100.0%)   0.8366 (100.0%)  Clang
front-end timer
   0.7756 (100.0%)   0.0604 (100.0%)   0.8360 (100.0%)   0.8366 (100.0%)  Total

clang++ -ftime-report test4.cpp  0.92s user 0.11s system 83% cpu 1.229 total
```

[66]:

```python
clang = namedtuple('Clang','prp prs cg cg_ot')
```

[67]:

```python
# custom runs
clang_runs=[]
for i in num_runs:
    t0 = pc()
    !clang++  -E test4.cpp &>/dev/null
    t1=pc()-t0
    !clang++ -fsyntax-only test4.cpp &>/dev/null
    t2=pc()-(t0+t1)
    !clang++ -O0 -S test4.cpp &> /dev/null
    t3=pc()-(t0+t1+t2)
    !clang++ -O2 -S test4.cpp &>/dev/null
    t4=pc()-(t0+t1+t2+t3)
    clang_runs.append(clang(t1,t2,t3,t4))
```

[68]:

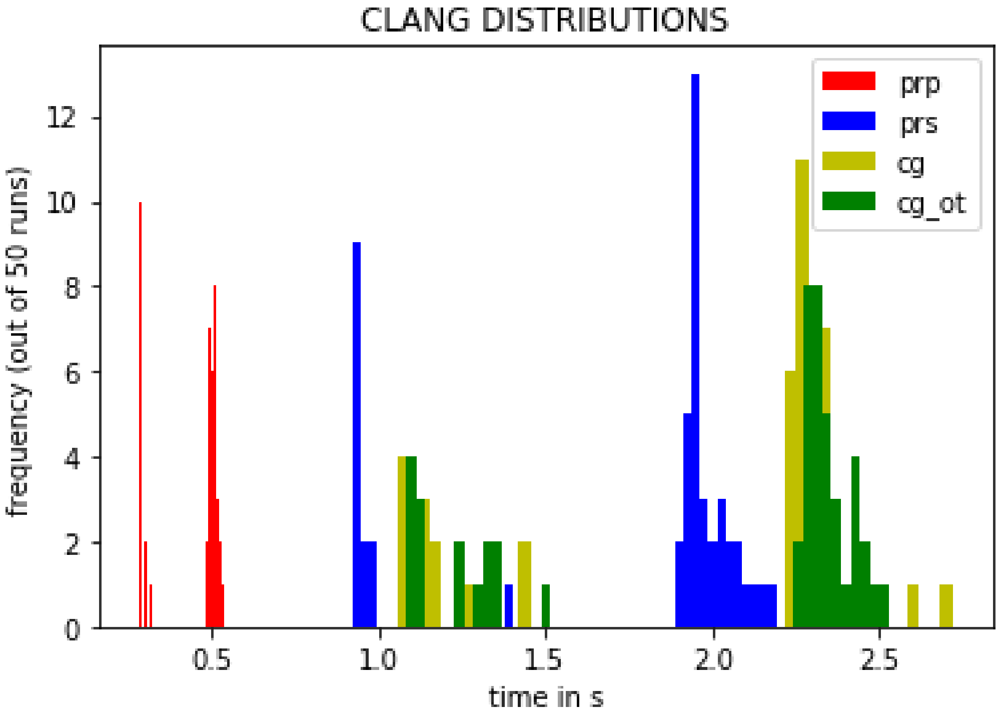
```python
np_clg = np.asarray(clang_runs)
np_clg.shape
```

[68]:

```
(50, 4)
```

[69]:

```python
plt.hist(np_clg[:,0],bins=50,color='r',label="prp")
plt.hist(np_clg[:,1],bins=50,color='b',label="prs")
plt.hist(np_clg[:,2],bins=50,color='y',label="cg")
plt.hist(np_clg[:,3],bins=50,color='g',label="cg_ot")
plt.title("CLANG DISTRIBUTIONS")
plt.legend()
plt.xlabel("time in s")
plt.ylabel("frequency (out of " + str(len(num_runs)) + " runs)")
plt.show()
```



4 GCC toolchain

[60]:
```
# gcc inbuilt analysis
!time g++ -ftime-report test4.cpp
```

```
Execution times (seconds)
 phase setup             :   0.00 ( 0%) usr   0.00 ( 0%) sys   0.00 ( 0%) wall
1495 kB ( 1%) ggc
 phase parsing           :   1.26 (85%) usr   0.54 (89%) sys   1.80 (86%) wall
135869 kB (82%) ggc
 phase lang. deferred    :   0.15 (10%) usr   0.02 ( 3%) sys   0.18 ( 9%) wall
18801 kB (11%) ggc
 phase opt and generate  :   0.07 ( 5%) usr   0.05 ( 8%) sys   0.11 ( 5%) wall
8608 kB ( 5%) ggc
 |name lookup            :   0.36 (24%) usr   0.08 (13%) sys   0.48 (23%) wall
10851 kB ( 7%) ggc
 |overload resolution    :   0.31 (21%) usr   0.01 ( 2%) sys   0.28 (13%) wall
```

```
29805 kB (18%) ggc
 dump files              :   0.01 ( 1%) usr   0.00 ( 0%) sys   0.00 ( 0%) wall
0 kB ( 0%) ggc
 callgraph construction  :   0.01 ( 1%) usr   0.01 ( 2%) sys   0.02 ( 1%) wall
801 kB ( 0%) ggc
 callgraph optimization  :   0.00 ( 0%) usr   0.00 ( 0%) sys   0.01 ( 0%) wall
4 kB ( 0%) ggc
 trivially dead code     :   0.00 ( 0%) usr   0.00 ( 0%) sys   0.01 ( 0%) wall
0 kB ( 0%) ggc
 df scan insns           :   0.01 ( 1%) usr   0.00 ( 0%) sys   0.01 ( 0%) wall
8 kB ( 0%) ggc
 df live regs            :   0.00 ( 0%) usr   0.00 ( 0%) sys   0.01 ( 0%) wall
0 kB ( 0%) ggc
 preprocessing           :   0.24 (16%) usr   0.13 (21%) sys   0.30 (14%) wall
5075 kB ( 3%) ggc
 parser (global)         :   0.23 (16%) usr   0.15 (25%) sys   0.49 (23%) wall
34358 kB (21%) ggc
 parser struct body      :   0.14 ( 9%) usr   0.05 ( 8%) sys   0.21 (10%) wall
22475 kB (14%) ggc
 parser function body    :   0.12 ( 8%) usr   0.03 ( 5%) sys   0.15 ( 7%) wall
8946 kB ( 5%) ggc
 parser inl. func. body  :   0.06 ( 4%) usr   0.04 ( 7%) sys   0.08 ( 4%) wall
3892 kB ( 2%) ggc
 parser inl. meth. body  :   0.09 ( 6%) usr   0.06 (10%) sys   0.21 (10%) wall
14458 kB ( 9%) ggc
 template instantiation  :   0.53 (36%) usr   0.10 (16%) sys   0.54 (26%) wall
65346 kB (40%) ggc
 tree SSA other          :   0.00 ( 0%) usr   0.01 ( 2%) sys   0.00 ( 0%) wall
24 kB ( 0%) ggc
 out of ssa              :   0.00 ( 0%) usr   0.00 ( 0%) sys   0.01 ( 0%) wall
21 kB ( 0%) ggc
 expand                  :   0.01 ( 1%) usr   0.00 ( 0%) sys   0.00 ( 0%) wall
712 kB ( 0%) ggc
 integrated RA           :   0.01 ( 1%) usr   0.00 ( 0%) sys   0.01 ( 0%) wall
4231 kB ( 3%) ggc
 LRA non-specific        :   0.01 ( 1%) usr   0.01 ( 2%) sys   0.00 ( 0%) wall
31 kB ( 0%) ggc
 reload                  :   0.00 ( 0%) usr   0.00 ( 0%) sys   0.01 ( 0%) wall
0 kB ( 0%) ggc
 rest of compilation     :   0.01 ( 1%) usr   0.02 ( 3%) sys   0.02 ( 1%) wall
419 kB ( 0%) ggc
 TOTAL                 :   1.48             0.61             2.10
164785 kB
g++ -ftime-report test4.cpp  1.56s user 0.63s system 95% cpu 2.298 total
```

This, unlike clang, provides the complete time analysis and

[70]:

```
gcc = namedtuple('GCC','prp prs cg cg_ot')
```

[71]:

```
# custom runs
gcc_runs=[]
for i in num_runs:
    t0 = pc()
    !g++  -E test4.cpp &>/dev/null
    t1=pc()-t0
    !g++ -fsyntax-only test4.cpp &>/dev/null
    t2=pc()-(t0+t1)
    !g++ -O0 -S test4.cpp &> /dev/null
    t3=pc()-(t0+t1+t2)
    !g++ -O2 -S test4.cpp &>/dev/null
    t4=pc()-(t0+t1+t2+t3)
    gcc_runs.append(gcc(t1,t2,t3,t4))
```

[72]:

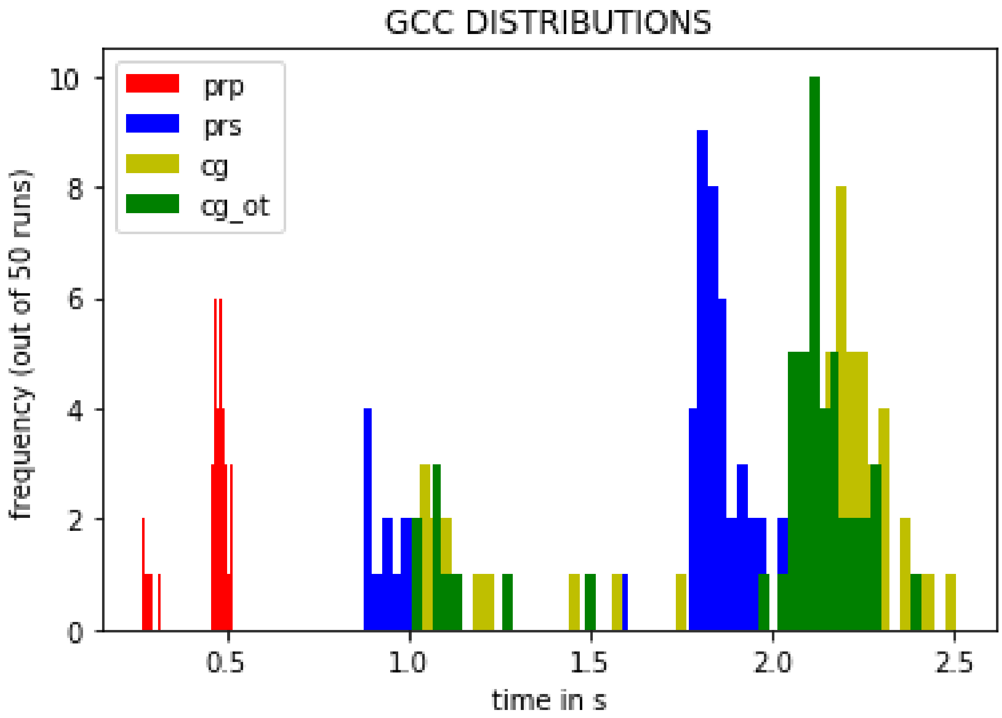
```
np_gcc = np.asarray(gcc_runs)
np_gcc.shape
```

[72]:

```
(50, 4)
```

[73]:

```
plt.hist(np_gcc[:,0],bins=50,color='r',label="prp")
plt.hist(np_gcc[:,1],bins=50,color='b',label="prs")
plt.hist(np_gcc[:,2],bins=50,color='y',label="cg")
plt.hist(np_gcc[:,3],bins=50,color='g',label="cg_ot")
plt.title("GCC DISTRIBUTIONS")
plt.legend()
plt.xlabel("time in s")
plt.ylabel("frequency (out of " + str(len(num_runs)) + " runs)")
plt.show()
```



5 collecting means

[82]:

```
print("\n","GCC")
print("gcc_prp",np_gcc[:,0].mean())
print("gcc_prs",np_gcc[:,1].mean())
print("gcc_cg",  np_gcc[:,2].mean())
print("gcc_cg_ot", np_gcc[:,3].mean())

print("\n","CLANG")
print("clg_prp" ,np_clg[:,0].mean())
print("clg_prs" , np_clg[:,1].mean())
print("clg_cg" , np_clg[:,2].mean())
print("clg_cg_ot" , np_clg[:,3].mean())
```

```
 GCC
gcc_prp 0.4530730939997011
gcc_prs 1.6781543300001067
gcc_cg 1.995258301999711
gcc_cg_ot 1.9667276100002347

 CLANG
```

```
clg_prp 0.4522343220002949
clg_prs 1.706136847999587
clg_cg 2.00593066399917
clg_cg_ot 2.011734538000019
```

That is very close

6 Notes

- cg_ot is not very different from cg and is in fact lesser in the case of gcc which is not expected.
- inbuilt tools paint a different picture which is expected : the python interpreter adds its own delays into this
    - ftime-report shows clang to be a bit faster
    - but the pythonic analysis doesn't show a significant difference
- However, given the effect weared of due to python's delays, We have reason to believe that clang is faster.
- Also note that the histograms observed are bimodal due to the presence of a cache and hence we should only compare corresponding peaks.