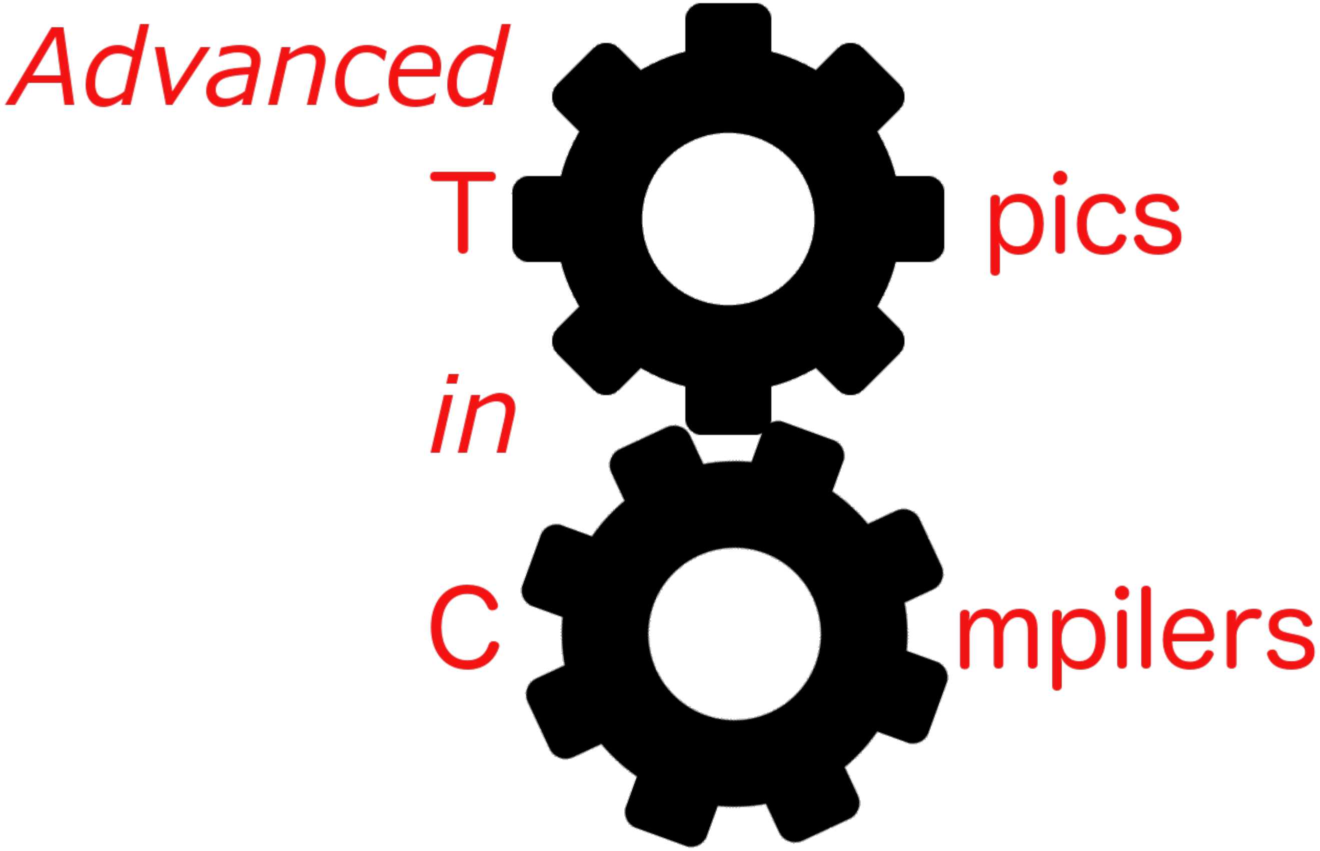


Advanced

T pics

in

C mpilers

NOELLEGym

Simone Campanoni
simone.campanoni@northwestern.edu

NORTHWESTERN UNIVERSITY
QUAECUMQUE SUNT VERA
ὁ λόγος πλήρης χάριτος καὶ ἀληθείας
1851

Outline

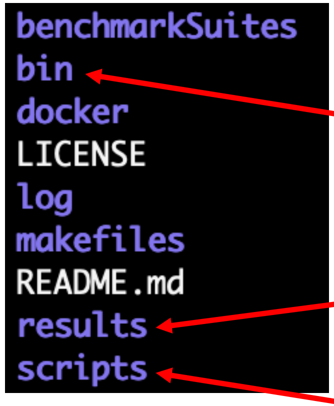
- Introduction
- Compile and optimize benchmarks
- Run benchmarks
- Inspect and modify the sources of a benchmark

NOELLEGym: introduction

- Infrastructure to test NOELLE-based optimizations on benchmarks typically used in research venues
  link

- Not particularly well designed
  - Started as a quick “put-together” infrastructure to quickly collect results
  - We are *slowly* improving its design
    - Feel free to make changes and do pull-requests
      (we’ll all appreciate it!)

NOELLEGym: structure

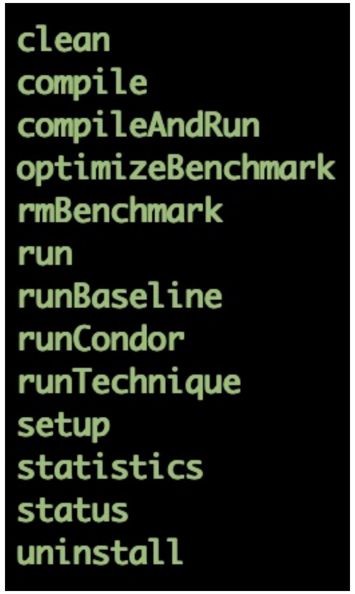


```
benchmarkSuites
bin
docker
LICENSE
log
makefiles
README.md
results
scripts
```

What an user can run

Results (e.g., speedups, IR) obtained

Scripts that will be invoked by



```
clean
compile
compileAndRun
optimizeBenchmark
rmBenchmark
run
runBaseline
runCondor
runTechnique
setup
statistics
status
uninstall
```

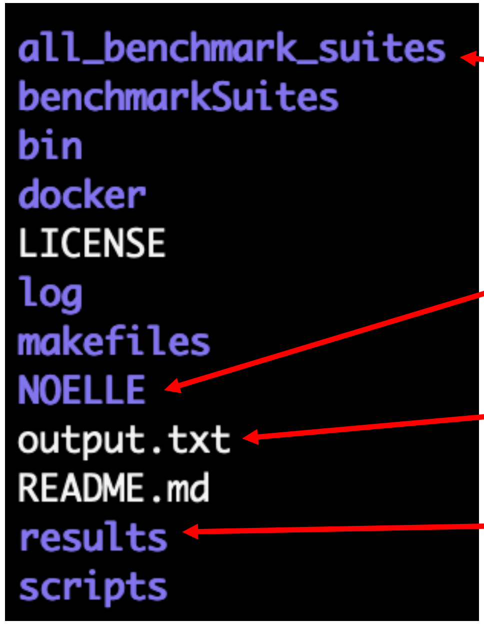
Setup on hanlon and alike

- export PATH=/home/software/go1.17.13/bin/:$PATH
- export PATH=/home/software/llvm-9.0.0/bin/:$PATH
- ./bin/setup

Setup on Zythos

- source /project/go/go_1.13.7/enable
- source /project/extra/llvm/9.0.0/enable
- source /project/gllvm/enable
- ./bin/setup

NOELLEGym: structure after setup



```
all_benchmark_suites
benchmarkSuites
bin
docker
LICENSE
log
makefiles
NOELLE
output.txt
README.md
results
scripts
```

The benchmarks to compile, optimize, and run

The NOELLE installation downloaded, compiled, and installed

Output generated by the commands executed by the user

The results obtained after the compilation, optimization, and execution of the benchamrks

Outline

- Introduction

- Compile and optimize benchmarks

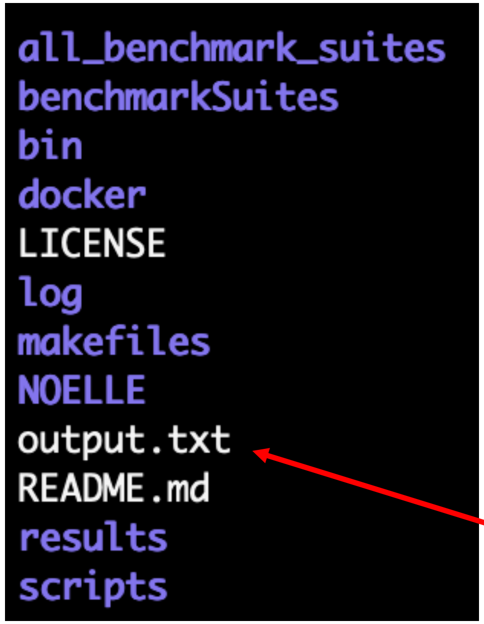
- Run benchmarks

- Inspect and modify the sources of a benchmark

Parallelize a benchmark with NOELLE

./bin/clean

./bin/optimizeBenchmark MiBench/search DOALL



```
all_benchmark_suites
benchmarkSuites
bin
docker
LICENSE
log
makefiles
NOELLE
output.txt
README.md
results
scripts
```

The first time this command executes, it performs the following:

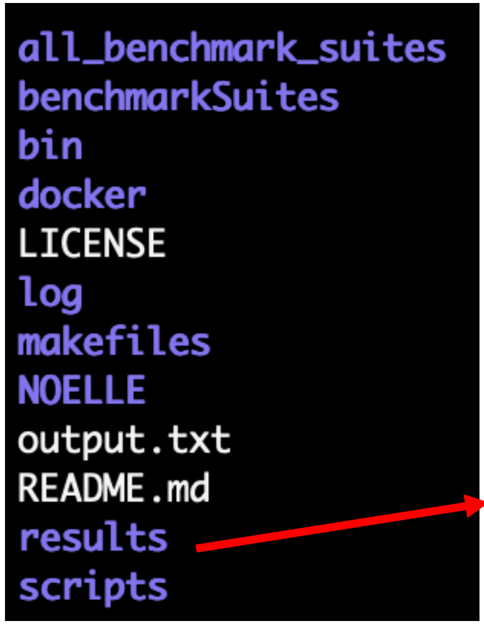
1. It generates the single IR file for an entire benchmark, for all benchmarks, in all benchmark suites
2. It runs the optimization/parallelization for only the benchmark specified as input

Sub-sequent invocations of the same command will only perform 2.

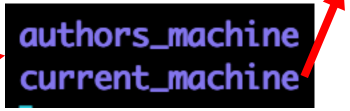
Output generated by the commands executed by the user

Parallelize a benchmark with NOELLE: Checking the output

```
./bin/optimizeBenchmark MiBench/search DOALL
```

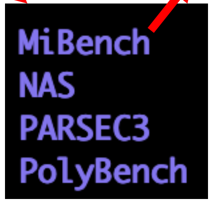


```
all_benchmark_suites
benchmarkSuites
bin
docker
LICENSE
log
makefiles
NOELLE
output.txt
README.md
results
scripts
```



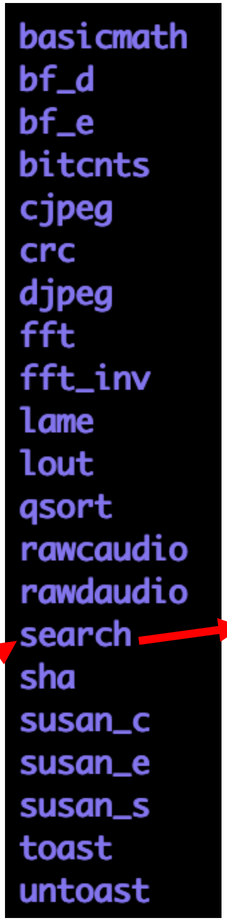
```
authors_machine
current_machine
```

```
IR
```

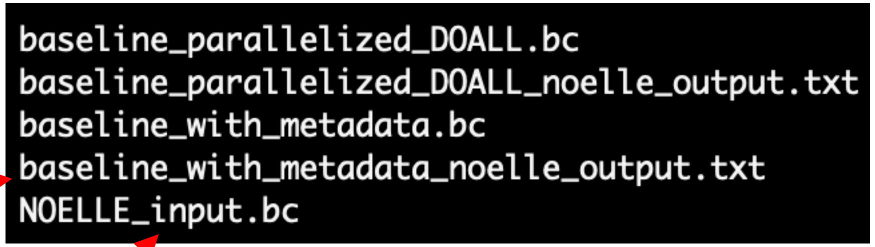


```
MiBench
NAS
PARSEC3
PolyBench
```

```
benchmarks
```



```
basicmath
bf_d
bf_e
bitcnts
cjpeg
crc
djpeg
fft
fft_inv
lame
lout
qsort
rawcaudio
rawdaudio
search
sha
susan_c
susan_e
susan_s
toast
untoast
```

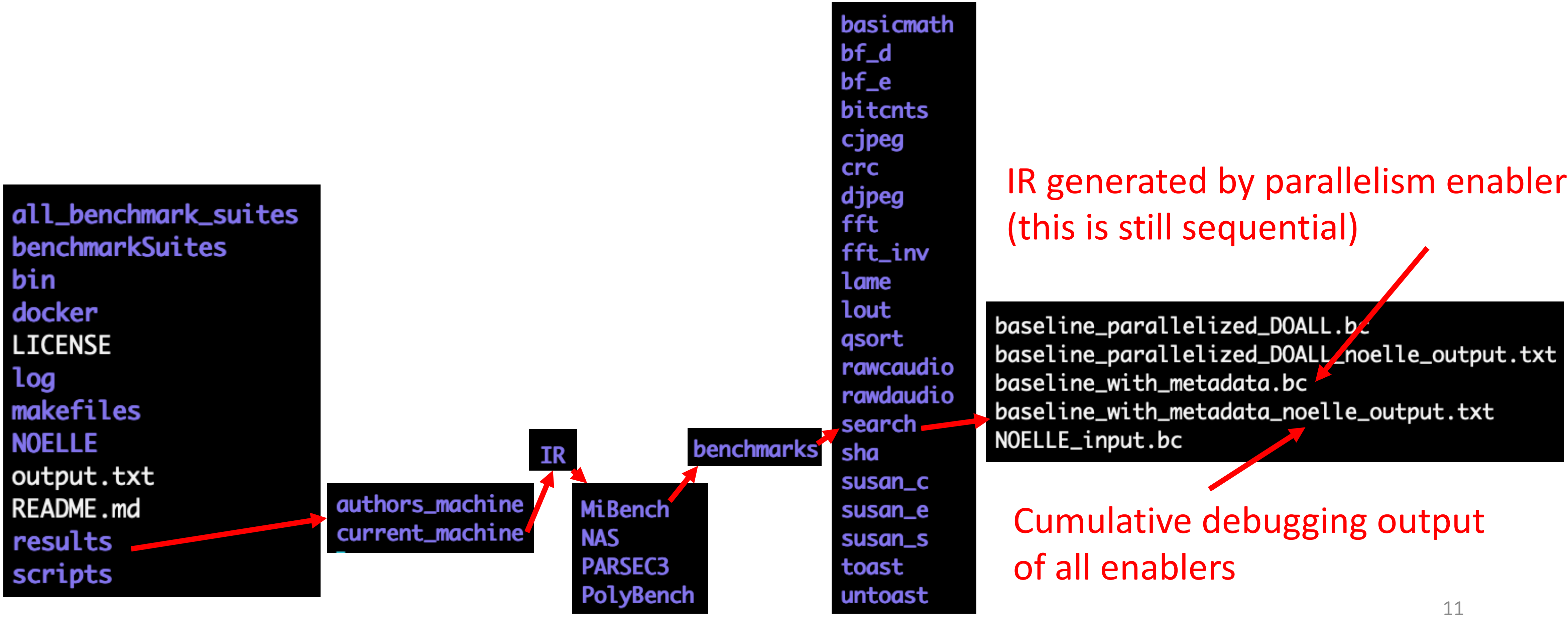


```
baseline_parallelized_DOALL.bc
baseline_parallelized_DOALL_noelle_output.txt
baseline_with_metadata.bc
baseline_with_metadata_noelle_output.txt
NOELLE_input.bc
```

Single IR for the entire program generated by the compiler front-end

Parallelize a benchmark with NOELLE: Checking the output

./bin/optimizeBenchmark MiBench/search DOALL



```
all_benchmark_suites
benchmarkSuites
bin
docker
LICENSE
log
makefiles
NOELLE
output.txt
README.md
results
scripts
```

```
authors_machine
current_machine
```

```
IR
```

```
MiBench
NAS
PARSEC3
PolyBench
```

```
benchmarks
```

```
basicmath
bf_d
bf_e
bitcnts
cjpeg
crc
djpeg
fft
fft_inv
lame
lout
qsort
rawcaudio
rawdaudio
search
sha
susan_c
susan_e
susan_s
toast
untoast
```

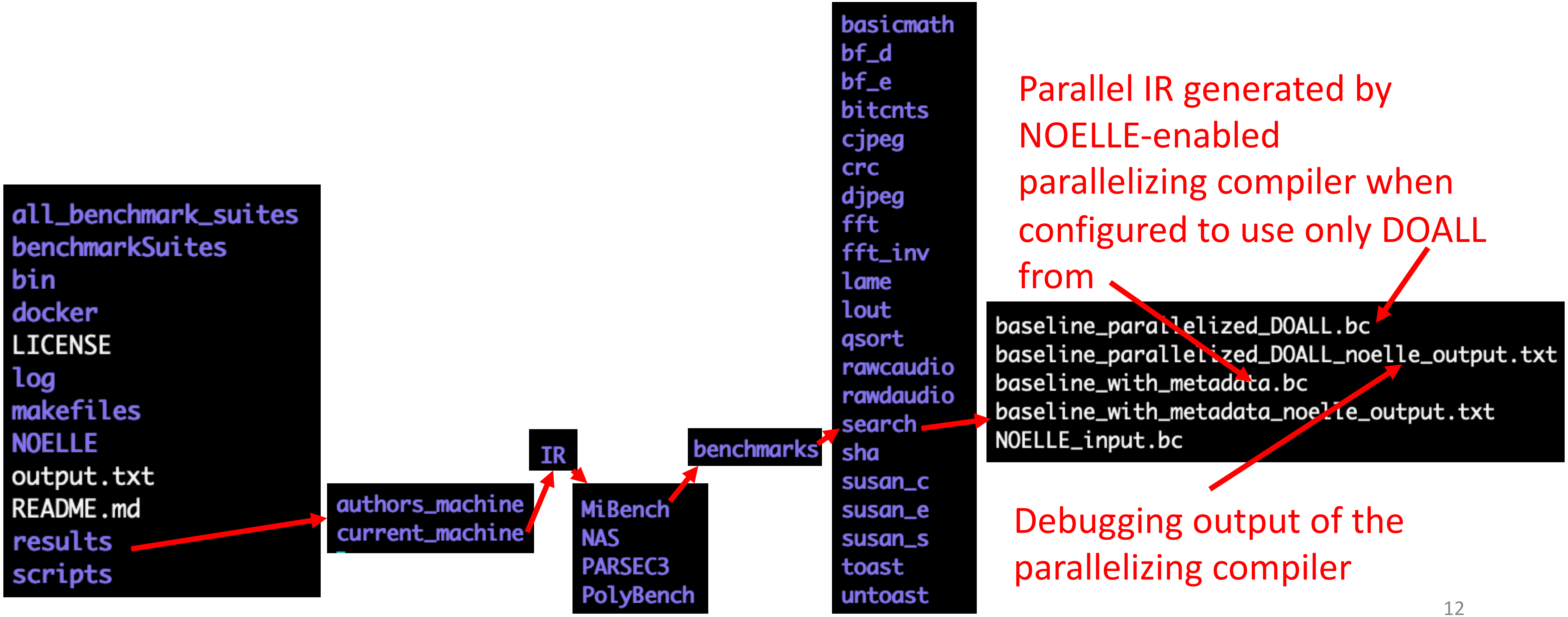
```
baseline_parallelized_DOALL.bc
baseline_parallelized_DOALL_noelle_output.txt
baseline_with_metadata.bc
baseline_with_metadata_noelle_output.txt
NOELLE_input.bc
```

IR generated by parallelism enablers (this is still sequential)

Cumulative debugging output of all enablers

Parallelize a benchmark with NOELLE: Checking the output

./bin/optimizeBenchmark MiBench/search DOALL



```
all_benchmark_suites
benchmarkSuites
bin
docker
LICENSE
log
makefiles
NOELLE
output.txt
README.md
results
scripts
```

```
authors_machine
current_machine
```

```
IR
```

```
MiBench
NAS
PARSEC3
PolyBench
```

```
benchmarks
```

```
basicmath
bf_d
bf_e
bitcnts
cjpeg
crc
djpeg
fft
fft_inv
lame
lout
qsort
rawcaudio
rawdaudio
search
sha
susan_c
susan_e
susan_s
toast
untoast
```

```
baseline_parallelized_DOALL.bc
baseline_parallelized_DOALL_noelle_output.txt
baseline_with_metadata.bc
baseline_with_metadata_noelle_output.txt
NOELLE_input.bc
```

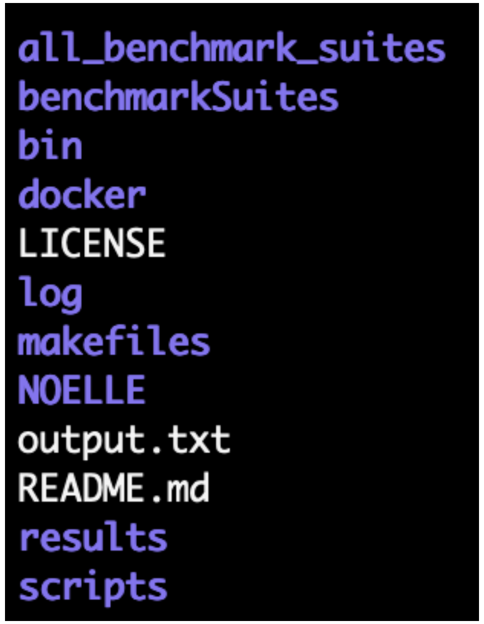
Parallel IR generated by NOELLE-enabled parallelizing compiler when configured to use only DOALL from

Debugging output of the parallelizing compiler

Parallelize all benchmarks with NOELLE

./bin/clean

./bin/compile



```
all_benchmark_suites
benchmarkSuites
bin
docker
LICENSE
log
makefiles
NOELLE
output.txt
README.md
results
scripts
```

The first time this command executes, it performs the following:

1. It generates the single IR file for an entire benchmark, for all benchmarks, in all benchmark suites
2. It runs the optimization/parallelization for all benchmarks, in all benchmark suites

Sub-sequent invocations of the same command will only perform 2.

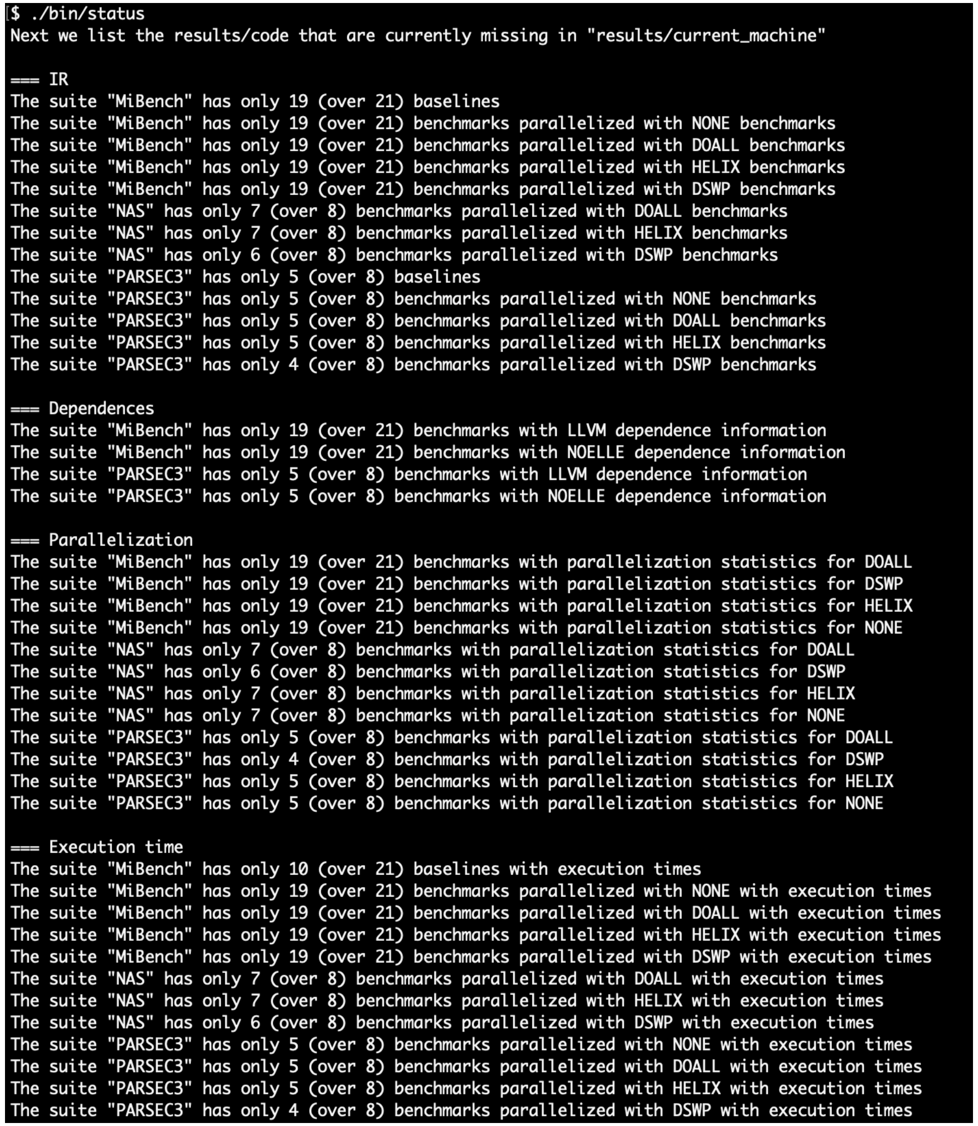
Check the status

./bin/status

It checks the status of results/current_machine of:

1. IR generated
2. Statistics about dependences in IR, parallelization performed
3. Execution times of the different IRs

It prints what is missing



```
$ ./bin/status
Next we list the results/code that are currently missing in "results/current_machine"

=== IR
The suite "MiBench" has only 19 (over 21) baselines
The suite "MiBench" has only 19 (over 21) benchmarks parallelized with NONE benchmarks
The suite "MiBench" has only 19 (over 21) benchmarks parallelized with DOALL benchmarks
The suite "MiBench" has only 19 (over 21) benchmarks parallelized with HELIX benchmarks
The suite "MiBench" has only 19 (over 21) benchmarks parallelized with DSWP benchmarks
The suite "NAS" has only 7 (over 8) benchmarks parallelized with DOALL benchmarks
The suite "NAS" has only 7 (over 8) benchmarks parallelized with HELIX benchmarks
The suite "NAS" has only 6 (over 8) benchmarks parallelized with DSWP benchmarks
The suite "PARSEC3" has only 5 (over 8) baselines
The suite "PARSEC3" has only 5 (over 8) benchmarks parallelized with NONE benchmarks
The suite "PARSEC3" has only 5 (over 8) benchmarks parallelized with DOALL benchmarks
The suite "PARSEC3" has only 5 (over 8) benchmarks parallelized with HELIX benchmarks
The suite "PARSEC3" has only 4 (over 8) benchmarks parallelized with DSWP benchmarks

=== Dependences
The suite "MiBench" has only 19 (over 21) benchmarks with LLVM dependence information
The suite "MiBench" has only 19 (over 21) benchmarks with NOELLE dependence information
The suite "PARSEC3" has only 5 (over 8) benchmarks with LLVM dependence information
The suite "PARSEC3" has only 5 (over 8) benchmarks with NOELLE dependence information

=== Parallelization
The suite "MiBench" has only 19 (over 21) benchmarks with parallelization statistics for DOALL
The suite "MiBench" has only 19 (over 21) benchmarks with parallelization statistics for DSWP
The suite "MiBench" has only 19 (over 21) benchmarks with parallelization statistics for HELIX
The suite "MiBench" has only 19 (over 21) benchmarks with parallelization statistics for NONE
The suite "NAS" has only 7 (over 8) benchmarks with parallelization statistics for DOALL
The suite "NAS" has only 6 (over 8) benchmarks with parallelization statistics for DSWP
The suite "NAS" has only 7 (over 8) benchmarks with parallelization statistics for HELIX
The suite "NAS" has only 7 (over 8) benchmarks with parallelization statistics for NONE
The suite "PARSEC3" has only 5 (over 8) benchmarks with parallelization statistics for DOALL
The suite "PARSEC3" has only 4 (over 8) benchmarks with parallelization statistics for DSWP
The suite "PARSEC3" has only 5 (over 8) benchmarks with parallelization statistics for HELIX
The suite "PARSEC3" has only 5 (over 8) benchmarks with parallelization statistics for NONE

=== Execution time
The suite "MiBench" has only 10 (over 21) baselines with execution times
The suite "MiBench" has only 19 (over 21) benchmarks parallelized with NONE with execution times
The suite "MiBench" has only 19 (over 21) benchmarks parallelized with DOALL with execution times
The suite "MiBench" has only 19 (over 21) benchmarks parallelized with HELIX with execution times
The suite "MiBench" has only 19 (over 21) benchmarks parallelized with DSWP with execution times
The suite "NAS" has only 7 (over 8) benchmarks parallelized with DOALL with execution times
The suite "NAS" has only 7 (over 8) benchmarks parallelized with HELIX with execution times
The suite "NAS" has only 6 (over 8) benchmarks parallelized with DSWP with execution times
The suite "PARSEC3" has only 5 (over 8) benchmarks parallelized with NONE with execution times
The suite "PARSEC3" has only 5 (over 8) benchmarks parallelized with DOALL with execution times
The suite "PARSEC3" has only 5 (over 8) benchmarks parallelized with HELIX with execution times
The suite "PARSEC3" has only 4 (over 8) benchmarks parallelized with DSWP with execution times
```

Outline

- Introduction

- Compile and optimize benchmarks

- Run benchmarks

- Inspect and modify the sources of a benchmark

Run benchmarks

./bin/clean

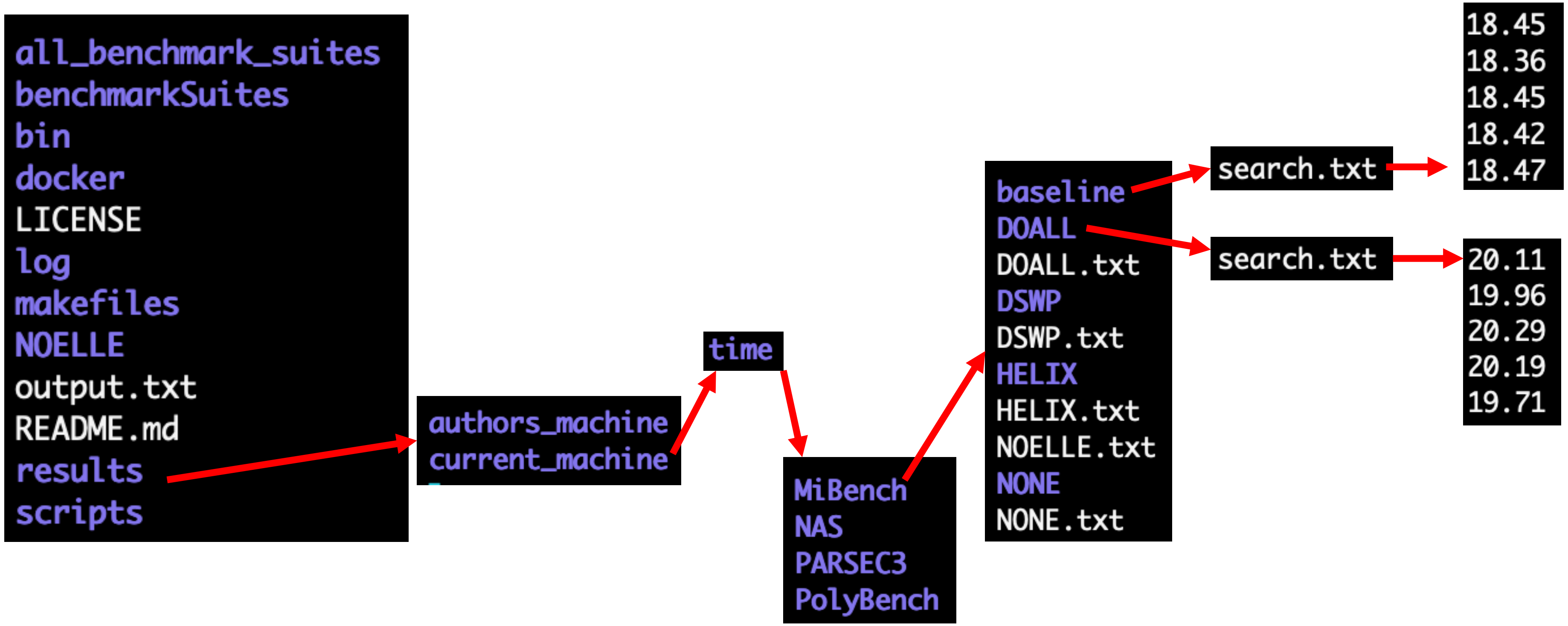
./bin/run

It performs the following for every benchmark that has an IR:

1. If the baseline time of benchmark X is not available in results/current_machine/time, then X is optimized using clang –O3 w/o using NOELLE, and
   the so-generated binary runs Y times

2. If the IR of an optimization (DOALL) is available and its execution time isn't available in results/current_machine/time, then it generates the binary from the optimized IR (e.g., baseline_parallelized_DOALL.bc), and it runs that binary Y times

Checking the times



```
all_benchmark_suites
benchmarkSuites
bin
docker
LICENSE
log
makefiles
NOELLE
output.txt
README.md
results
scripts
```

```
authors_machine
current_machine
```

```
time
```

```
MiBench
NAS
PARSEC3
PolyBench
```

```
baseline
DOALL
DOALL.txt
DSWP
DSWP.txt
HELIX
HELIX.txt
NOELLE.txt
NONE
NONE.txt
```

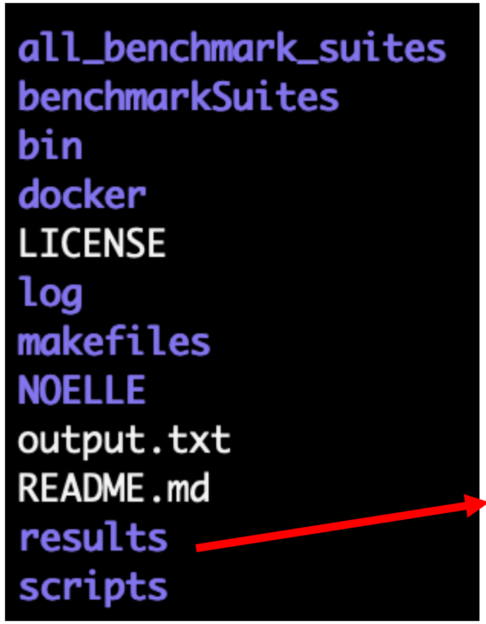
```
search.txt
```

```
18.45
18.36
18.45
18.42
18.47
```

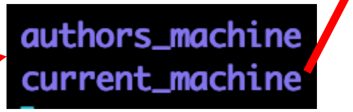
```
search.txt
```

```
20.11
19.96
20.29
20.19
19.71
```

Checking the speedups

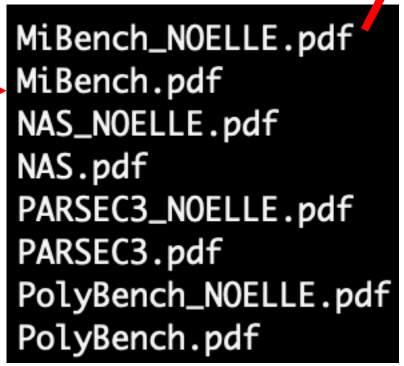


```
all_benchmark_suites
benchmarkSuites
bin
docker
LICENSE
log
makefiles
NOELLE
output.txt
README.md
results
scripts
```

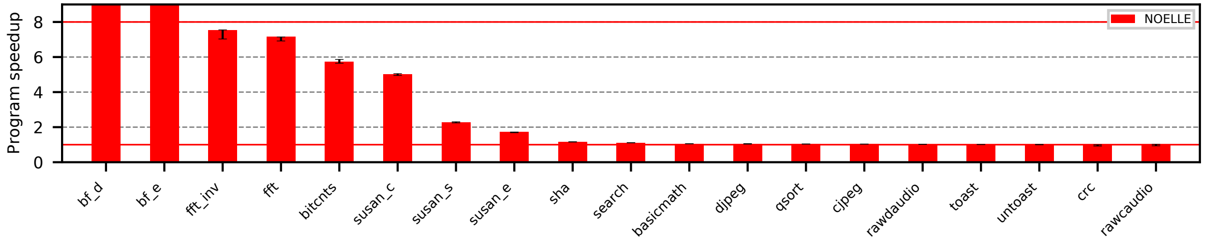


```
authors_machine
current_machine
```

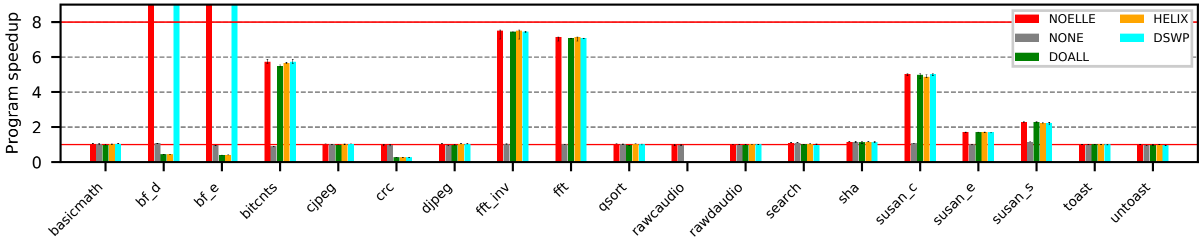
```
plots
```

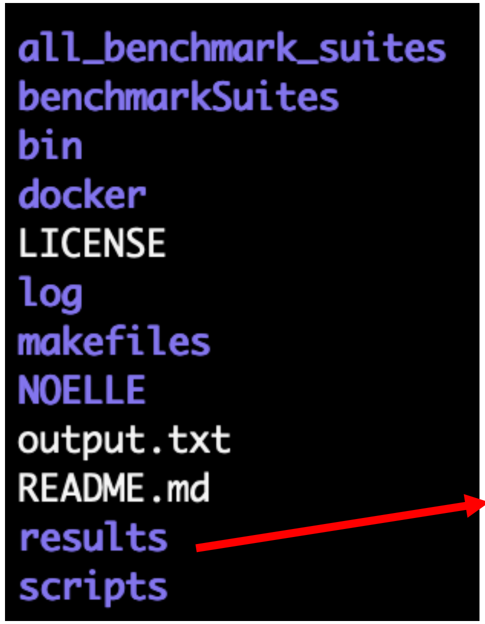


```
MiBench_NOELLE.pdf
MiBench.pdf
NAS_NOELLE.pdf
NAS.pdf
PARSEC3_NOELLE.pdf
PARSEC3.pdf
PolyBench_NOELLE.pdf
PolyBench.pdf
```

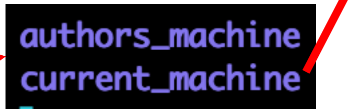


Checking the speedups



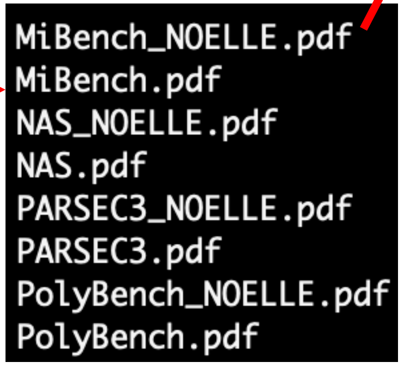


```
all_benchmark_suites
benchmarkSuites
bin
docker
LICENSE
log
makefiles
NOELLE
output.txt
README.md
results
scripts
```



```
authors_machine
current_machine
```

```
plots
```

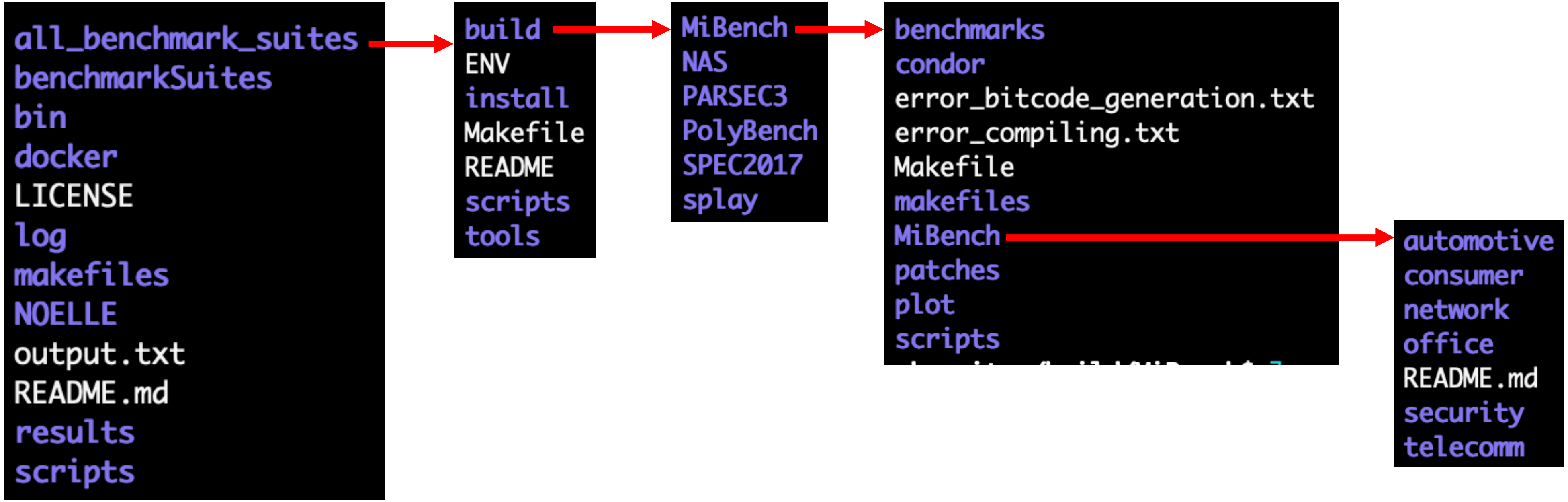


```
MiBench_NOELLE.pdf
MiBench.pdf
NAS_NOELLE.pdf
NAS.pdf
PARSEC3_NOELLE.pdf
PARSEC3.pdf
PolyBench_NOELLE.pdf
PolyBench.pdf
```

Outline

- Introduction
- Compile and optimize benchmarks
- Run benchmarks
- Inspect and modify the sources of a benchmark

Checking the sources of a benchmark



```
all_benchmark_suites
benchmarkSuites
bin
docker
LICENSE
log
makefiles
NOELLE
output.txt
README.md
results
scripts
```

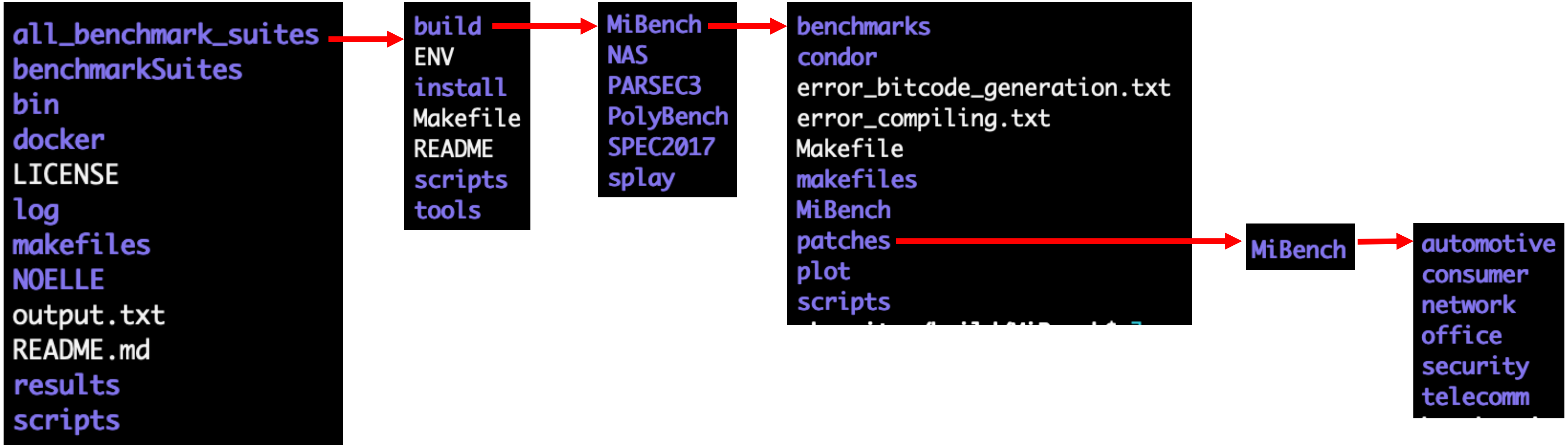
```
build
ENV
install
Makefile
README
scripts
tools
```

```
MiBench
NAS
PARSEC3
PolyBench
SPEC2017
splay
```

```
benchmarks
condor
error_bitcode_generation.txt
error_compiling.txt
Makefile
makefiles
MiBench
patches
plot
scripts
```

```
automotive
consumer
network
office
README.md
security
telecomm
```

Changing the sources of a benchmark



```
all_benchmark_suites
benchmarkSuites
bin
docker
LICENSE
log
makefiles
NOELLE
output.txt
README.md
results
scripts
```

```
build
ENV
install
Makefile
README
scripts
tools
```

```
MiBench
NAS
PARSEC3
PolyBench
SPEC2017
splay
```

```
benchmarks
condor
error_bitcode_generation.txt
error_compiling.txt
Makefile
makefiles
MiBench
patches
plot
scripts
```

```
MiBench
```

```
automotive
consumer
network
office
security
telecomm
```

After it, you need to delete results/current_machine and re-run your optimization

Always have faith in your ability

Success will come your way eventually

Best of luck!