Profiling your code

- gdb sampling
- Manual timing
- Tools: gperftools, valgrind,...

GDB sampling: the forbidden technique

- Compile with -g

GDB sampling: the forbidden technique

- Compile with -g
- Run gdb

GDB sampling: the forbidden technique

- Compile with -g
- Run gdb
- Run the program: "r"

GDB sampling: the forbidden technique

- Compile with -g
- Run gdb
- Run the program: "r"
- Press Ctrl+C

GDB sampling: the forbidden technique

- Compile with -g
- Run gdb
- Run the program: "r"
- Press Ctrl+C
- Backtrack: "bt"

GDB sampling: the forbidden technique

- Compile with -g
- Run gdb
- Run the program: "r"
- Press Ctrl+C
- Backtrack: "bt"
- Repeat

Measuring time

```cpp
#include <iostream>
#include <chrono>
using namespace std::chrono;
int main() {
  auto start = high_resolution_clock::now();
  // Code to measure
  auto end = high_resolution_clock::now();
  auto elapsed = end - start;
  std::cout << "Elapsed time: ";
  std::cout << elapsed.count() << " s\n";
  auto elapsed_ms =
   ↪  duration_cast<milliseconds>(elapsed);
}
```

Benchmarking

```
template <typename Foo>
auto bench(int ntest, Foo foo) {
  using namespace std::chrono;
  for (int i = 0; i < 10; i++)
    foo();
  auto start = high_resolution_clock::now();
  for (int i = 0; i < ntest; i++)
    foo();
  auto end = high_resolution_clock::now();
  auto ms = duration_cast<milliseconds>(end -
  ↪  start).count();
  return ms / (ntest * 1.0);
}
```

General advice

- Cache optimizations

General advice

- Cache optimizations
- SIMD Vectorization

General advice

- Cache optimizations
- SIMD Vectorization
- Branch prediction

Compiling with optimization

```
clang++ -O3 -march=native -o myprog myprog.cpp
```

- O3: Maximum optimization
- march=native: Optimize for current CPU

Inlining

```
inline void my_function1() {
  // This function can be inlined
}

void __attribute__((noinline)) my_function2() {
  // This function should not be inlined
}
```

Volatile

```
volatile int x = 0;
x = 1;
```

- Never optimize away this variable

Google performance tools

- Install:
  ```
  conda install gperftools
  ```
- Compile:
  ```
  g++ -o myprog myprog.cpp -lprofiler -g
  ```
- Run:
  ```
  CPUPROFILE=myprog.prof ./myprog
  ```
- Analyze:
  ```
  pprof myprog myprog.prof
  ```