More computing sins are committed in the name of efficiency (without necessarily achieving it) than for any other single reason -

William A. Wulf
We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil.

Donald E. Knuth (quoting Tony Hoare)
Rules of Optimization:
Rule 1: Don't do it.
Rule 2 (for experts only): Don't do it yet.

M.A. Jackson
Levels of Optimization

- High-level Program Structure
- Data Structures / Algorithms
- Efficient Program Code
- Assembly
- Hardware
80/20 Rule

"80% of the time is spent in 20% of the code"

So?

How?
Measuring Code

time command
Manual Instrumentation

int calls_to_foo = 0;

void foo(string arg)
{
    ++calls_to_foo;
    // ...do the actual work...
}
# include <sys/time.h>
struct timeval start, end;

gettimeofday(&start, NULL);
// do the work.
gettimeofday(&end, NULL);

fprintf(stderr, "%f ms\n",
 (end.tv_sec - start.tv_sec) * 1000.0 +
 (end.tv_usec - start.tv_usec) / 1000.0 );
bool iszero(int n) {
    return n == 0;
}

double total_us = 0.0;

bool iszero(int n) {
    struct timeval start, end;
    gettimeofday(&start, NULL);

    bool answer = (n == 0);
    gettimeofday(&end, NULL);
    total_us +=
          (end.tv_sec - start.tv_sec) * 1.0e6
     + (end.tv_usec - start.tv_usec);

    return answer;
}
Profilers

Tools exist to instrument the code for you.

Counting calls (possibly in a smart fashion)

PC sampling
  Flat profile
  Hierarchical profile

One widely-available tool is gprof
Assigning Time?
Using gprof

Compile with the -gp flag.

Run the program (which will generate gmon.out)

Get "readable" though verbose output via

```
gprof <executable-name> gmon.out
```
Other issues

What to do about cycles?

Input dependence
Profile
Call graph