COMP 2400 UNIX Tools

Christian Grothoff
christian@grothoff.org

http://grothoff.org/christian/

Profiling: Basic Approach

1. Start with high-level testcases
2. Increase problem size
3. Use tools to determine crucial portion of code
4. Find or write testcases specific to crucial algorithm
5. Use tools to locate and resolve bottlenecks
6. Validate performance improvement using high-level applications

Profiling in C

- No JIT!
- No Garbage Collector!

$\Rightarrow$ More deterministic execution!

$\Rightarrow$ Profiling is much easier!

$\Rightarrow$ Expectations are also higher...

Possible Expectations

- Include operating system calls and performance
- Give precise accounting of resource consumption: CPU, memory, IO, code size, etc.
- Include low-level performance effects: scheduling, locking, cache misses, branch prediction

Important C Profiling Tools

- `time` – as used with Java
- `gprof` – similar to TPTP and `java -Xprof`
- `oprofile` – operating-system profiling
- `valgrind` – cache simulation, memory usage

Using gprof

- Enable compiler optimizations (at least level `-O2`)
- Make sure problem size is appropriate (at least 5s runtime)
- Compile with option `-pg` and `-g`
- Run application as usual (will generate `gmon.out`)
- Run `gprof binary-name` (will use `gmon.out`)
- Study output, improve performance, validate without `-pg`

Issues with `gprof`

- Does **not** work for multi-threaded applications

- Only gives per-function output

+ Lightweight instrumentation – programs run at near-native speed

+ Lightweight instrumentation – results are quite accurate

oprofile

- Can profile interrupt handlers
- Captures behavior of the entire system (including OS!)
- Can obtain hardware performance metrics
- Requires `root` permissions
- Linux only

$\Rightarrow$ Use if `time` shows significant time was spent in `sys`

valgrind, callgrind, kcachegrind

- Interprets the application code
- Can **simulate** low-level behavior (different cache size)
- Can obtain tons of performance metrics

$\Rightarrow$ Simulation may not always be precise, verify results!

$\Rightarrow$ Simulation is costly, program will run **very** slow

valgrind – massif

- Used to profile memory utilization

o Faster than `memcheck` or `callgrind`

o Requires specification of `malloc` wrappers

\+ Precise analysis of allocation sites

\- Ignores memory fragmentation

What is "fast enough"

Rule of thumb:

You should stop trying to make your code run faster if the programmer time spent to make it faster is close to the savings in execution time (thougout the lifetime of the code) that you hope to obtain.

Questions

?