An Introduction to Data-Oriented Programming

Yaser Zhian

Dead Mage Studio

Target Audience

Programmers who face:

- Hard problems
- Performance/resource constraints
- Many solutions

Agenda

- Define a simple sample problem
- Discuss solutions
- Go over couple of solutions (hopefully today)
- Problem areas and reasoning about behavior
- Another (better) solution
- Theoretical groundwork for “data-oriented” programming

Problem Statement

- Simulate the behavior of, and display many ants crawling on a predefined map, from “city” to “city”.
  - Various behaviors
  - Each ant is a pixel
  - This is a benchmark; we’re going to evaluate multiple solutions
    - Not particularly focused on performance
- What more do you need to know?

Let’s Discuss

- What is our map?
- What’s an ant? What can it do?
  - “Ant” stories?!
- How do we render?
- How to organize the solution?
  - Make it work,
  - Make it right,
  - Make it fast.
- What else?

Coding

Part 1

Where Is My Time?!

- How long should it take?
  - Let's count…
    - How many operations do we have? (ALU/FPU*, branches, memory, etc.)
    - How long do each of these take?
- Cost of abstractions
- ALU/SSE operations
  - Throughput vs. Latency
- Branch (mis)predictions
- Memory

Memory

- What do we mean by “speed”?!
- What does the spec say?
- How can we measure?
- Let’s measure…

Coding

Part 2

RAM isn’t RAM (or, why is RAM so slow?!)

- The “stored program” machine
- RAM is S…L…O…W…
- RAM isn’t random-access
- Other problems
  - Multiple cores
  - NUMA
  - Mispredicted branches in a slow-RAM world
- What to do?

Cache

- General operation (levels, algorithms)
- Cache line
- Associativity
- Multi-CPU/Multi-Core
- “Why” does a cache work?!!!

Costs (on a Haswell)

- (Mostly throughput)
- ALU:
  - Mostly in 0.5-2 cycle range
  - 64-bit (signed) division is ~20-80 cy
- Branches: mostly in 1-2 cy range
- FPU/SSE/AVX/etc.:
  - Mostly in 0.5-2 cy range
  - (Various incarnations of) division and sqrt are in 7-28 cy range
  - Transcendentals (sin, cos, tan, etc.) are in ~50-150 cy range
- Cost of hitting main memory: ~100-200 cycles

Optimization

- If *"p"* is the portion of time that we spend in processing, and we improve that by a factor of *"s",* then the "speed up" we gain is:
  - $speedup = \frac{1}{(1-p)+\frac{p}{s}}$
  - This is called "Amdahl's Law"
- One result of the above law is this:
  - It's always better to optimize the larger (largest) parts!
  - It's all about memory access
- So let's write some code…

Coding

Part 3

Data-Oriented Design Principles

Or “Gospel according to Mike (Acton)”

Step 0:

"The purpose of all programs, and all parts of those programs, is to transform data from one form to another."

Step 1:

“If you don’t understand the data you don’t understand the problem.”

“Conversely, understand the problem by understanding the data.”

Step 2:

“Different problems require different solutions.”

Step 3:

“If you have different data, you have a different problem.”

Step 4:

“If you don’t understand the cost of solving the problem, you don’t understand the problem.”

Step 5:

“If you don’t understand the hardware, you can’t reason about the cost of solving the problem.”

(also, understand your code; i.e. language, compiler, OS, etc.)

Step 6:

“Everything a data problem. Including usability, maintenance, debug-ability, etc. Everything.”

Step 7:

“Solving problems that you probably don’t have, creates problems that you definitely do.”

Step 8:

“Latency and throughput are only the same in sequential systems.”

Step 9:

"Software does not run in a magic fairy aether powered by the fevered dreams of CS PhDs."

Step 10:

“Reason must prevail.”

- This page is intentionally left blank!

Three Big Lies (#1)

Software is a platform

Three Big Lies (#2)

Code should be designed around a model of the world

Three Big Lies (#3)

Code is more important than data

More Lies

- If you put in enough abstractions, the solution becomes self-evident.
- You can abstract the real world away.
- Ignorance is bliss.
- Most misused quote:
  - “Premature optimization is the root of all evil.”

Some Final Thoughts

- Cult of Ignorance
  - Reusability
    - Patterns!
    - Algorithms vs. patterns
  - Genericity
  - Performance
- Encapsulation for maintainability vs. data hiding
- Abstraction vs. transformation
- Ignoring the real world doesn’t make it disappear!
- Top-down vs. bottom-up vs. side-to-side

Beware of delusion!

- Everything can be used to justify anything.
- There is no “One True Way”.
- Look at the data, and write the simplest solution that satisfies all constraints.
  - Input/output data
  - Meta data: how many, how often, when
  - Constraints: time, ability, cognitive load, hardware

Thank You!

- References
  - CppCon 2014: Mike Acton, “Data-Oriented Design and C++” https://www.youtube.com/watch?v=rX0ItVEVjHc
  - Richard Fabian, “Data-Oriented Design” http://www.dataorienteddesign.com/dodbook/
  - Agner Fog, Software Optimization Manuals (Microarch and timings) https://www.agner.org/optimize/
- Any questions?