COHERENT:LABS

Data-oriented design principles

Stoyan Nikolov

@stoyannk
stoyannk.wordpress.com
github.com/stoyannk

- Eutelia - Software developer on large ERP
- Masthead Studios - Senior graphics developer
- Coherent Labs - Co-Founder & Software architect
  - Coherent UI (Desktop)
  - Coherent GT (Desktop + Consoles)
  - Project Colibri (Mobile)
  - Renoir Graphics Library




What I do?

Why design matters?

- Many developers pay too much attention to “code”
- High-quality software requires sound design - like engineering
- Code is just a building block
- Good design is hard, code is easy!
- **HPC begins from the design!**

Quick OOP overview

- Introduces “real-world” abstractions
- Couples “data” with the operations (code) on them
- Treats objects as black boxes
- Promises easier code reuse and maintenance

But..

- Was born in an era when machines were different than the ones we now have
- Tries to hide the data instead of embracing them
- Reuse and maintainability are hurt through excessive coupling

Cache misses.. ouch!

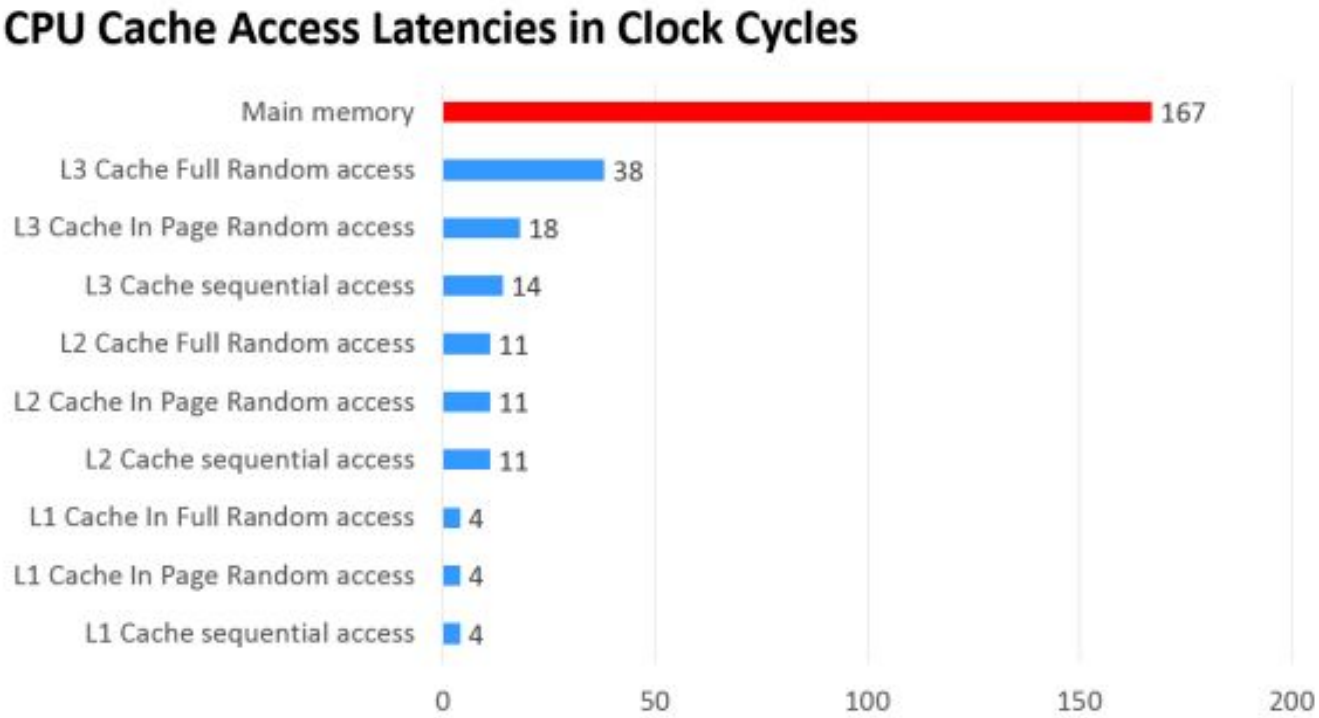

Image by Chris Adkin - http://chrisadkin.org/

A word on performance

- Big O notation gives a general idea of the perf. characteristics of an algo/data structure
- In real-world apps the constant factors are **crucial**
- Example - http://bit.ly/1MOYSdV
- **Takeaway -> Know your data!**

Data-oriented design

- Relatively recent trend primarily in game development (but the idea is old)
- Think about the **data** & flow first
- Think about the hardware your code runs on
- Build software around data & **transforms** on it, instead of an artificial abstraction

Goals

- **Higher performance**
- Easier to parallelise
- Easier to maintain & debug

Sounds good, but..

- Although simple in essence data-oriented design can be difficult to achieve
- Probably we need more time to shake-off years of OOP indoctrination
- Many “text-book” examples in presentations are too obvious

Classic examples

- Breaking classes into pieces for better cache utilization
- AoS -> SoA
- Arrays of Components in a game engine
- “Where there’s one - there are many”

OOP Textbook example

- Draw a collection of moving shapes (circles & rects)
- Polymorphic hierarchy
- A lot of state is kept in the objects
- https://gist.github.com/stoyannk/49b94fc794b643e175fb

Usual OOP horror

- “Shape” interface
  - Embraces individual memory allocations
  - Virtual everything
- Enforces inheritance (tight coupling)
  - “Transformed” shape
- A lot of state hidden inside objects
  - Complex state machines

Usual OOP horror (cont.)

- Operations bring-in unrelated data due to the memory layout of objects
  - Position calculation cares nothing about rendering data but it'll still be pulled in the cache
- Difficult to parallelize
  - I don't know what the "black-box" object is doing when we call "Update". It could touch literally anything in the object state.

Data-oriented approach

- https://gist.github.com/stoyannk/49b94fc794b643e175fb
- Think about the data & transforms
  - We have a bunch of geometries to draw - circles and rectangles
  - Circles have radii
  - Rects have sizes
  - All of them have a position
  - Some move, others are stationary

Data-oriented approach

- Each frame we move some shapes (change the positions based on time & a target)

```
struct Positions
{
    unsigned Count;
    float* Xs; // Simplify SIMD-ification
    float* Ys;
};
```

Existence-based state

```
struct ShapePositions
{
    unsigned ImmobileCount; // [0..Immobile Count are not moving)
    Positions Pos; // Contains both immobile and moving (the state is implied)
};

void MoveShapes(float timeSinceStart,
    ShapePositions positions,
    Positions target)
{
    // Move only the needed ones
    // Trivial to parallelize; SIMD-ify whatever
}
```

Circles & Rectangles

- The data of the shapes are different

```
struct Circles
{
    unsigned Count;
    float* Radii;
    ShapePositions Pos;
};
Circles AllCircles;
Positions CircleTargets;
```

```
struct Rectangles
{
    unsigned Count;
    float* SizeX;
    float* SizeY;
    ShapePositions Pos;
};
Rectangles AllRects;
Positions RectTargets;
```

Draw data

- Organize as tables again
- Variant 1: Keep a sorted list of ID spans of the circles/rects to draw
- Variant 2: In a MT environment you could gather all circles/rects to draw. Collect rendering-related data and pass them down the pipeline.

Data organization

- Tables
- Existence-based evaluation (no branching)
- Stateless functions
- Organize based on access patterns
- Gather-transform-scatter where necessary

Key take-away

Think what is happening on the machine when it executes your code.

Algorithmic complexity is *rarely* the problem - constant factors often hit performance!

Thank you!

@stoyannk
stoyannk.wordpress.com
github.com/stoyannk

Come talk to me if you are interested in what we do!

References

- Data-Oriented design, Richard Fabian, http://www.dataorienteddesign.com/dodmain/
- Pitfalls of Object Oriented Programming, Tony Albrecht, http://harmful.cat-v.org/software/OO_programming/_pdf/Pitfalls_of_Object_Oriented_Programming_GCAP_09.pdf
- Data-Oriented Design in C++, Mike Acton, https://www.youtube.com/watch?v=rX0ItVEVjHc
- Typical C++ Bullshit, Mike Acton, http://macton.smugmug.com/gallery/8936708_T6zQX#!i=593426709&k=BrHWXdJ
- Introduction to Data-Oriented Design, DICE, http://www.dice.se/wp-content/uploads/2014/12/Introduction_to_Data-Oriented_Design.pdf
- Culling the Battlefield: Data Oriented Design in Practice, Daniel Collin, http://www.slideshare.net/DICEStudio/culling-the-battlefield-data-oriented-design-in-practice
- Adventures in data-oriented design, Stefan Reinalter, https://molecularmusings.wordpress.com/2011/11/03/adventures-in-data-oriented-design-part-1-mesh-data-3/
- Building a Data-Oriented Entity System, Niklas Frykholm, http://bitsquid.blogspot.com/2014/08/building-data-oriented-entity-system.html
- Intro to data-oriented design, Stoyan Nikolov, https://stoyannk.wordpress.com/2015/07/09/intro-to-data-oriented-design/