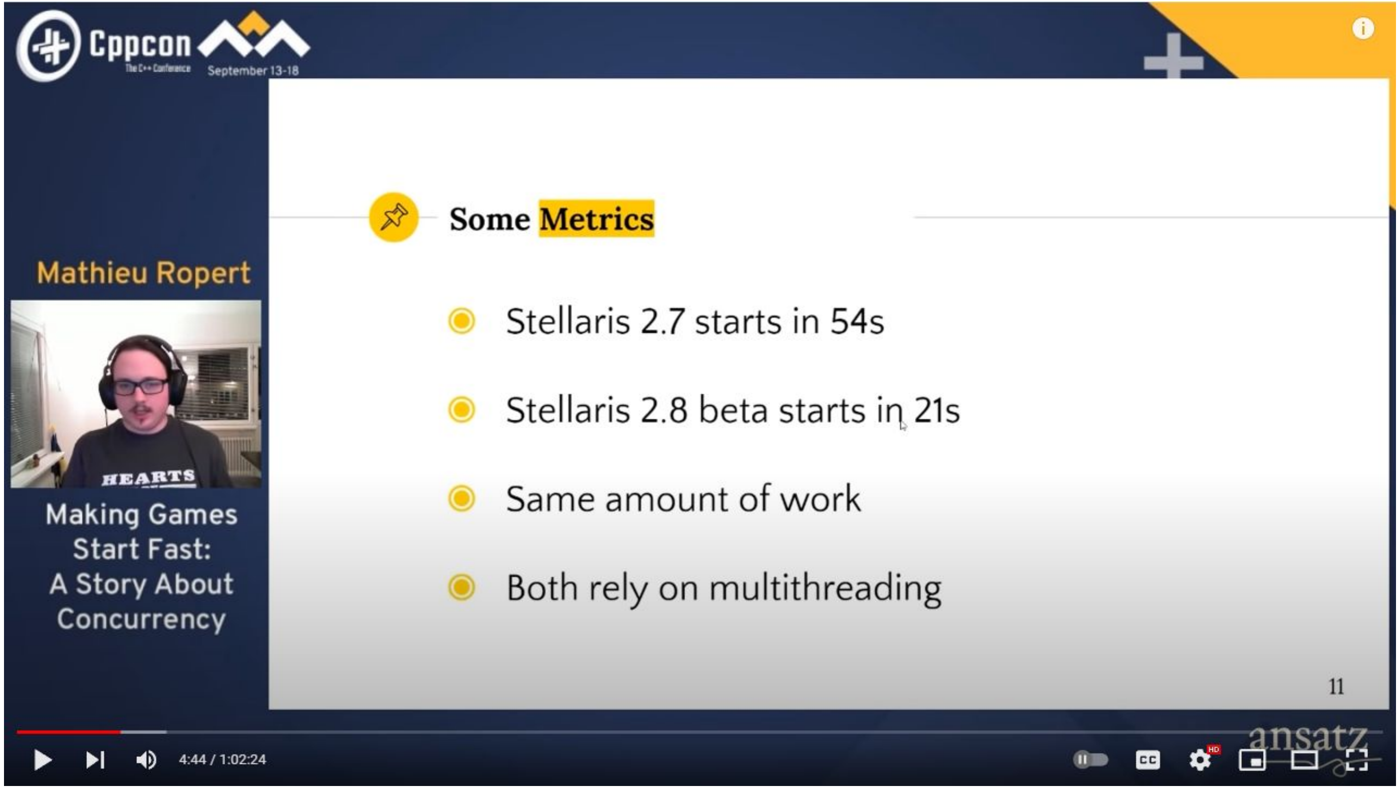
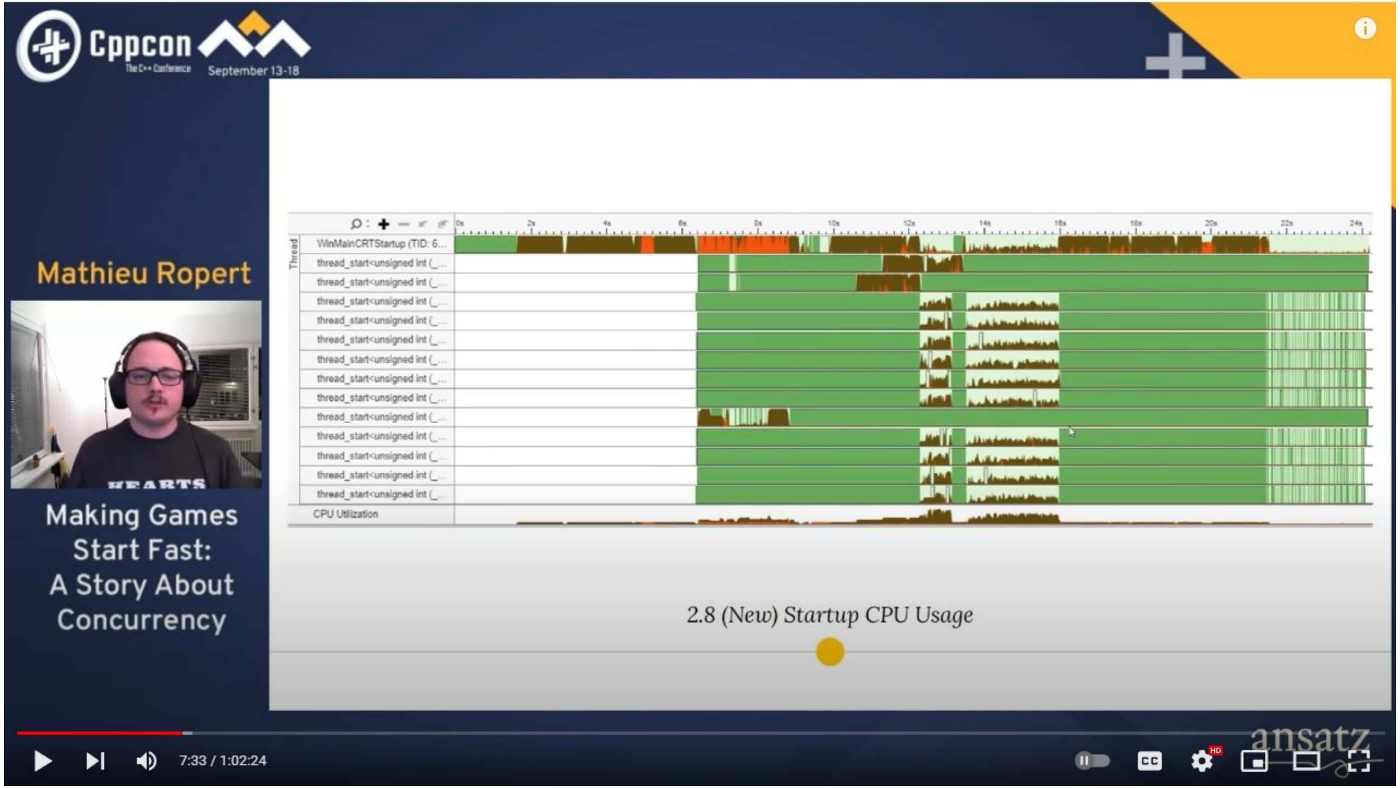
The Basics of Profiling



Previously on CppCon...



Previously on CppCon...

“Here’s how I made things faster”

“Here’s how I found what was slow”

Hello!

I am Mathieu Ropert

I'm a Tech Lead at Paradox Development Studio where I make Hearts of Iron IV.

You can reach me at:

- mro@puchiko.net
- @MatRopert
- https://mropert.github.io

We’re Hiring

paradox
INTERACTIVE®

paradox
DEVELOPMENT
STUDIO

https://career.paradoxplaza.com/

About this talk

- Profiling
- Tools for profiling
- Building an intuition

1

Profiling 101

Just enough theory to be dangerous

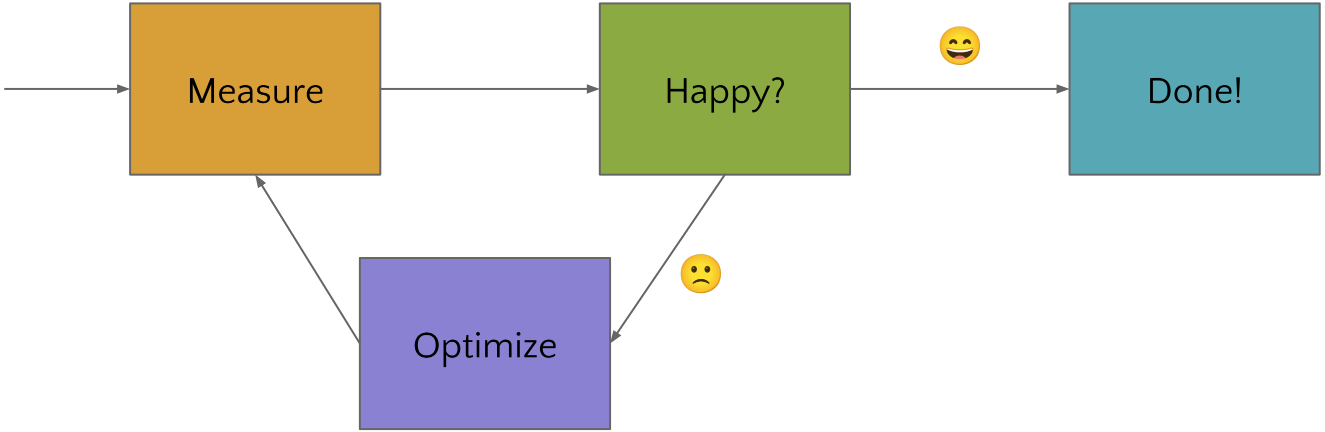
“The real problem is that programmers have spent far too much time worrying about efficiency in the wrong places and at the wrong times”

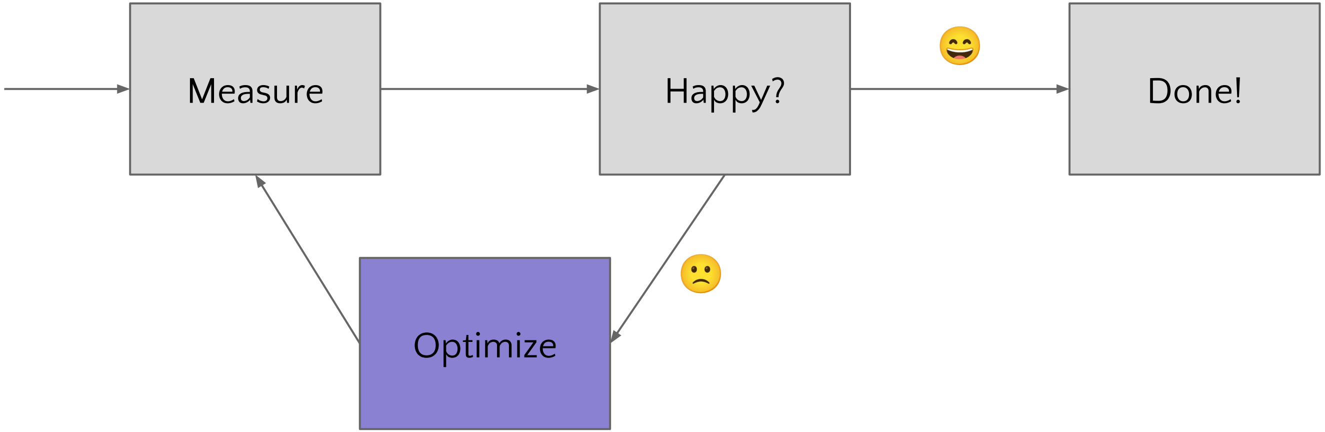
Why profiling?

- Figuring why a program is slow is hard
- Reading the code can easily mislead
- Modern CPUs are quite complex
- Measure, measure, measure!

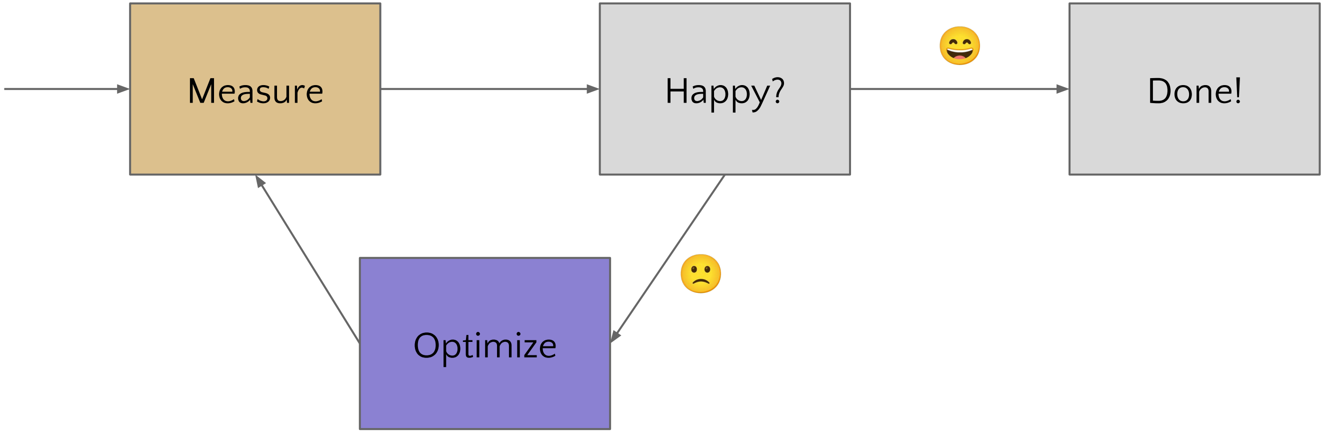
Profilers

Tools to help programmers measure and reason about performance


Measure
Happy?
Done!
Optimize



Profiling & Optimization


Measure
Happy?
Done!
Optimize

Profiling vs Optimization

- Profilers are one of the tools that can be used during an optimization iteration cycle
- Better used to investigate where to optimize
- Can be used to measure if an optimization was effective, within limits

Profiler usage

- Identify hotspots & bottlenecks
- Visualize execution timeline
- Collect & compute metrics

Sampling Profiling

- Attach to program, periodically interrupt and record the stack trace
- Sampling frequency is customizable
- Results are statistical averages
- Example tool: vTune

Sampling Profiling

- Only needs to be able to read stack trace
- Minimal debug info is enough
- Works out of the box on any executable
- Inlined functions are usually invisible

Instrumentation Profiling

- Add code hooks to explicitly record metrics
- Can provide both averages *and* exact breakdown by execution frame
- Not affected by inlining or statistical anomalies
- Example tool: Optick

Instrumentation Profiling

- Requires programmers to add collection macros in tactical places in the code
- Supports adding extra business metadata
- Can fallback on sampling
- Build implications

Profiler **Families**

Sampling

- Periodically interrupt program and record stack
- Works out of the box
- Susceptible to inlining

Instrumentation

- Add code to collect metrics
- Records usually match business logic better
- Need to recompile and link a 3rd party

2

Profiling in Practice

Let’s put the theory to use!

Setting up goals

- Set up a reproducible scenario
- Measure its performance
- Define an objective

Using the right tool

- Instrumentation (+ some sampling) is the recommended way to go

- Sampling alone is cheaper to start with

- Consider adding instrumentation as an investment

Demo Time!

Finding the needle

- First time look at a profile can be overwhelming
- Look at what sticks out
- Domain knowledge is key

Know thy program

- A profiler can tell what takes the most time
- It can explain why
- It can't tell if it should

Hunting disprecencies

- Performance regressions become easy to spot once the normal profile outline is known
- What takes time vs what *should* take time

Profile Time!

Best Work Is No Work

- Most efficient code does nothing
- Profiling can highlight useless computations
- No need to dive deep into metrics!

Profiling First Time

- Assess the big picture
- Understanding the domain is key to figure out where to start digging
- Get quick wins out of the way before delving deeper

3 Profiling Analysis

We have to go deeper

Profiling Metrics

- CPU Time
- Wait Time
- System Time

Profiling **Metrics**

- CPU Time
- Wait Time
- System Time

Demo Time!

High CPU Time

- Inefficient algorithms or data structures
- Spin locks
- Single threaded code
- Branch misprediction, cache misses

High Wait Time

- Disk I/O
- Network calls
- Locks
- Synchronization

Filtering Metrics

- Sampling views usually aggregate call stacks across threads
- Consider filtering on main bottleneck thread
- 2D control flow view from instrumented profilers helps a lot

A* Refresher

- Open set: nodes/square reachable but not explored
- Closed set: nodes/squares fully explored
- Pick best candidate in open set, add neighbours to open set, repeat until destination is reached

Inefficient Algorithms

Source	CPU Time: Total	CPU Time: Self
`while (fScore.size() > 0) {`		
`double minFScore = std::numeric_limits<double>::max();`		
`int current = -1;`		
`// Find the next cheapest node to visit`		
`for(const auto & openNode : fScore) {`	9.0%	7.159s
`int fs = openNode.second;`	33.8%	27.028s
`if (minFScore > fs) {`	47.8%	38.262s
`minFScore = fs;`	0.0%	0.014s
`current = openNode.first;`	0.0%	0.016s
`}`		
`}`		

Inefficient Algorithms

- Time spent in loops, recursive calls and `<algorithm>`
- Check the Big O
- Can computations be cached and reused?

Inefficient Data Structures

Source Line ▲	Source	CPU Time: Total	CPU Time: Self
235	`// Loop on possible adjacent cells`		
236	`// Map::findNeighbors() will remove uneligible cells from the list (out of bounds and impassable cells)`		
237	`for (const Coordinates& nextCell : _map.findNeighbors(currentCell))`	0.2%	0.063s
238	`{`		
239	`auto findCurrentCost = costFromStart.find(currentCell);`	15.5%	5.509s
240	`assert(findCurrentCost != costFromStart.end());`		
241	`int newCost = findCurrentCost->second + 1; // it costs 1 to go from one cell to the next`	0.0%	0.014s
242			
243	`// Only examine the next cell it if it's the first time,`		
244	`// or if a shorter path from Start cell has been found.`		
245	`auto findIt = costFromStart.find(nextCell);`	18.9%	6.702s
246	`if (findIt == costFromStart.end() || newCost < findIt->second)`		
247	`{`		
248	`const int heuristics = _map.distance(nextCell, _target); // distance without obstacle`	0.1%	0.031s
249	`int priority = newCost + heuristics;`		
250	`q.put(nextCell, priority);`	0.4%	0.155s
251	`costFromStart[nextCell] = newCost;`	4.9%	1.753s
252	`shortestPathMap[nextCell] = currentCell;`	8.3%	2.957s
253	`}`		
254	`}`	0.3%	0.117s

Inefficient Data Structures

- Time spent in `find`, `insert` or `operator[]`
- Easier to spot in bottom-up without inlining
- Know your data structures strengths and weaknesses

Spin Locks

- High spin time in profiler or equivalent tagged functions in instrumented profiles
- Look at the bigger picture and threading model
- Check out talks about concurrency 😉

Single Threaded Code

- Low core usage in timeline
- Consider parallel algorithms…
- … or a task scheduler

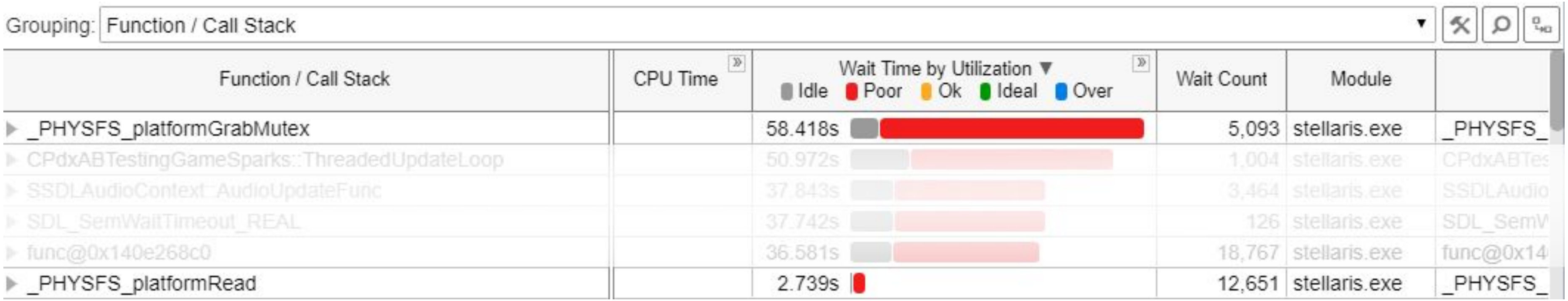
Micro-architecture Usage

- High CPI rate
- More and more important on modern CPUs
- Micro-optimization on large applications is tricky
- Keep for last

Blocking I/O

- High wait/system time in filesystem or network API
- Can it be put in an async task instead?
- See my 2020 CppCon Talk: *Making Games Start Fast – A Story About Concurrency*

Wait on Mutex or Semaphore



Grouping: Function / Call Stack

Function / Call Stack	CPU Time	Wait Time by Utilization (Idle, Poor, Ok, Ideal, Over)	Wait Count	Module	
_PHYSFS_platformGrabMutex		58.418s	5,093	stellaris.exe	_PHYSFS_
CPdxABTestingGameSparks::ThreadedUpdateLoop		50.972s	1,004	stellaris.exe	CPdxABTes
SSDLAudioContext::AudioUpdateFunc		37.843s	3,464	stellaris.exe	SSDLAudio
SDL_SemWaitTimeout_REAL		37.742s	126	stellaris.exe	SDL_SemW
func@0x140e268c0		36.581s	18,767	stellaris.exe	func@0x14
_PHYSFS_platformRead		2.739s	12,651	stellaris.exe	_PHYSFS_

Wait on Mutex or Semaphore

- High wait time on synchronization functions
- Remember: “it shouldn’t be called *mutex*, it should be called *bottleneck*”
- Consider changing concurrency model

Profiling Analysis

- Profiler will show what sticks out
- Some filtering needs to be done by the developer to focus on the right part
- Deal with inefficient algorithms, data structures and locks first

4

Wrapping Up

Have you been listening closely?

In conclusion

- Profilers help pinpointing performance bottlenecks
- Domain knowledge can speed up the analysis by a lot
- Add instrumentation support to your program

Furthermore

“

Furthermore, I think your build should be destroyed

Thanks!

Any questions ?

You can reach me at

- mro@puchiko.net
- @MatRopert
- @mropert
- https://mropert.github.io

We're Hiring

https://career.paradoxplaza.com/