Introduction to Job System, ECS, Burst compiler

Himanshu Maurya 13/07/2018

Performance by default?

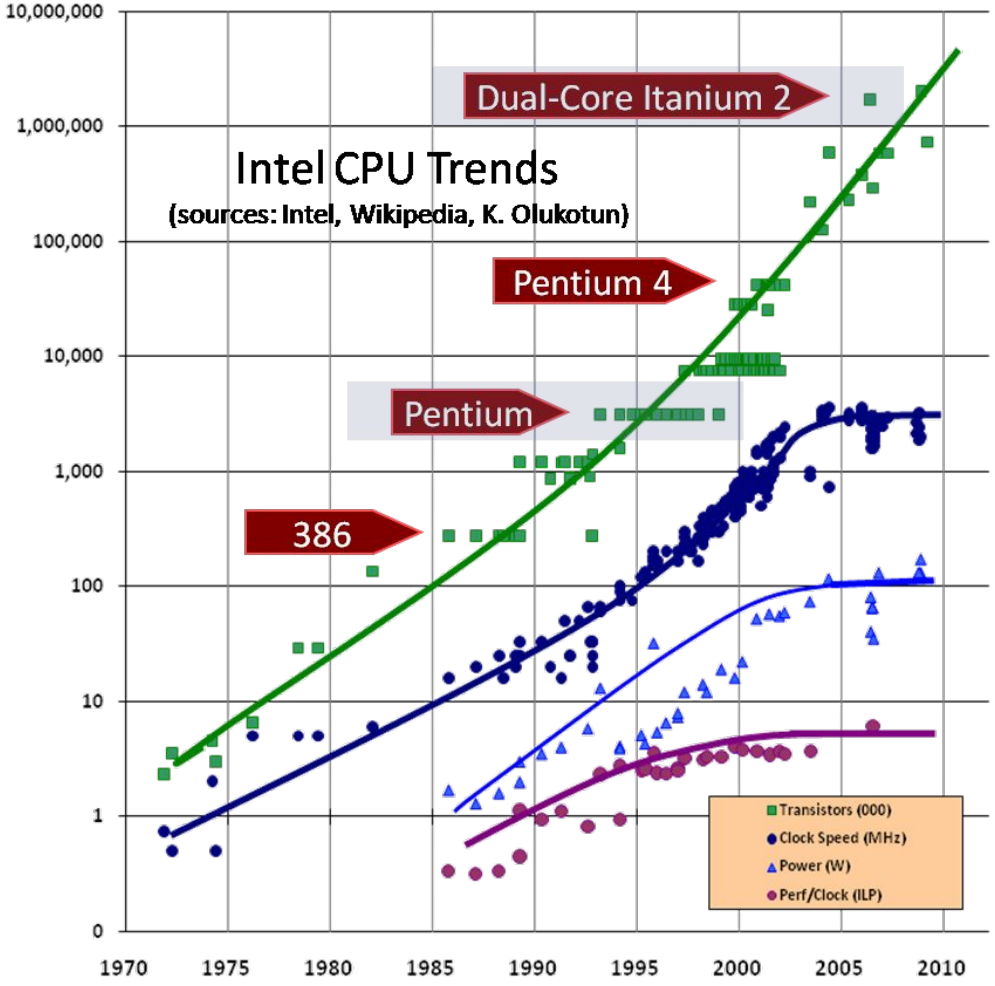
https://unity3d.com/unity/features/job-system-ECS

Why, What, and How?

We were getting the performance by default

~Till around 2010

By increase in CPU frequency or speed of execution or lower clock ticks or from superscaler architecture



http://www.gotw.ca/publications/concurrency-ddj.htm

Power wall

Processors power dissipation **P ~ Cdyn*V(2)*f**

Linear scale in frequency causes power dissipation to cubed. So processor can get melt but in practice it won't because CPU power controller chip will turn off the chip

Summary: why multicore revolution?

- Today single thread performance of a cpu is increasing slowly
- To run significantly faster, programs must utilize multiple processing elements

Challenges

- Writing parallel/concurrent program is hard
- Problem partitioning, communication, synchronization

Themes

Memory wall

Latency numbers that every programmer should know. (Now definitely should know after everyone is jumping on data orientation related efficiency)

https://gist.github.com/jboner/2841832

Access latency animation

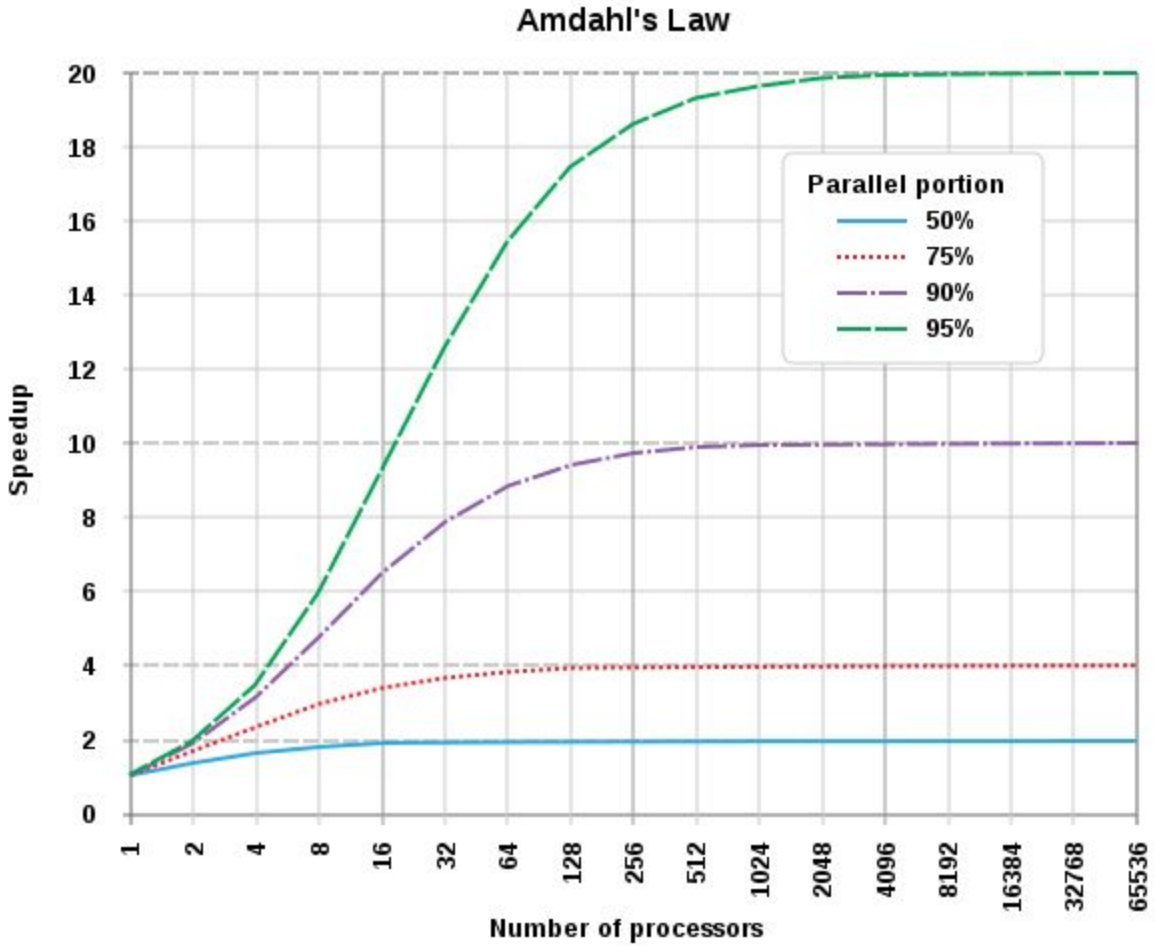
http://www.overbyte.com.au/misc/Lesson3/CacheFun.html

Fast != Efficient

- If you are able to speed up your program using multiple cpu/core does not imply your program is efficient
  - E.g. Is 2x speed up over 10 processor a good result?
  - In the era of mobile computing it is important to use resources efficiently

Amdahl’s law

- Defines the critical path in computing
- Pessimistic in nature
- No matter what you do you will always be bound by critical path


Amdahl's Law
Speedup
Number of processors
Parallel portion
50%
75%
90%
95%
0
2
4
6
8
10
12
14
16
18
20
1
2
4
8
16
32
64
128
256
512
1024
2048
4096
8192
16384
32768
65536

Speed up

1 = S + P

S = serial part of program

P = parallel part of program

N = num of processors

SU = 1 / (S + P/N)

Gustafson’s law

- Optimistic law
- Sees the speedup factor as a multiplier which scales up with scaling load
- Which means that the parallel portion of the programs scales up with increasing workload and increasing resources
- E.g. with more cores in processor you can process more data (that falls in to non-serial composition of program)

Gustafson’s law

SU = 1-p + s*p

This law agrees with past trends. Workloads scaling was proportional to resources scaling

But still Amdahl’s law can haunt us anytime soon

Some other observations

- Transistors in cpu still tend to increase per year despite being limited by the power wall
- Scientist started using this extra transistors to make processors smart aka braniac processors that do magic like out of order execution
- Building vector registers (SIMD registers and associated instructions SSE, AVX)

Lets ask the question again..

Performance by default?


MAYBE
OR MAYBE NOT
memegen.fr

What is the 

with current system in Unity?


NO PROBLEM
YOUR WELCOME
makeameme.org

What is the problem with current system in Unity?

No problem

It just works fine

Favours composition of objects to give it a behaviour *

Objects define data and behaviour

*Inheritance often create more problems. Always try to avoid it https://www.thoughtworks.com/insights/blog/composition-vs-inheritance-how-choose

To get the benefit

Address memory wall

Use Gustafson’s law and avoid amdahl’s law

Democratizing writing scripts that gives you parallelism minus all the problem

Mapping the domain

Lets maps the themes that we know to new Unity systems

ECS

ECS: Entity component system

Focus heavily on data organization in linear fashion. Tries to address the memory wall issue.

ECS...

Turn out game loops can be highly parallel per system

```
while( user doesn't exit )
  check for user input
  run AI
  move enemies
  resolve collisions
  draw graphics
  play sounds
end while
```

ECS...

Not very new. People have identified this problem long ago in bits and pieces.

Many game studios proprietary engine have such framework.

Most of the innovation is lead by game studios and engine developers, specially consoles

Job System

Tries to take advantage of Gustafson’s law. Design your parallel processing components in a way that it gets automatic scaling when resources in a machine increases

Job System...

Also try to reduce Amdahl’s law by properly load balancing the system

Job System...

New way of thinking about concurrency. You think about the jobs/tasks and dependencies and not about threads/synchronization

Burst compiler

Burst compiler is Auto vectorization

Data parallelism by using vector registers in the cpu. Vector registers can process multiple float/int data in single cpu instruction.

In flynn’s taxonomy called as SIMD (Single Instruction Multiple Data)

https://en.wikipedia.org/wiki/Automatic_vectorization

https://en.wikipedia.org/wiki/Streaming_SIMD_Extensions

https://gcc.gnu.org/projects/tree-ssa/vectorization.html

End of part 1