High Performance Computing for Engineers

David Thomas

dt10@ic.ac.uk / https://github.com/m8pple

Room 903

https://github.com/HPCE/hpce-2016

High Performance **Computing for Engineers**

- Research
  - Testing communication protocols
  - Evaluating signal-processing filters
  - Simulating analogue and digital designs

High Performance **Computing for Engineers**

- Research
  - Testing communication protocols
  - Evaluating signal-processing filters
  - Simulating analogue and digital designs
- Tools
  - CAD tools: synthesis, place-and-route, verification
  - Libraries/toolboxes: filter design, compressive sensing

High Performance **Computing for Engineers**

- Research
  - Testing communication protocols
  - Evaluating signal-processing filters
  - Simulating analogue and digital designs
- Tools
  - CAD tools: synthesis, place-and-route, verification
  - Libraries/toolboxes: filter design, compressive sensing
- Products
  - Oil exploration and discovery
  - Mobile-phone apps
  - Financial computing

High Performance *Computing for Engineers*

- Types of performance metrics

High Performance *Computing for Engineers*

- Types of performance metrics
  - Throughput
  - Latency
  - Power
  - Design-time
  - Capital and running costs

High Performance *Computing for Engineers*

- Types of performance metrics
  - Throughput
  - Latency
  - Power
  - Design-time
  - Capital and running costs
- Required versus desired performance
  - *Subject to a throughput of X, minimise average power*
  - *Subject to a budget of Y, maximise energy efficiency*
  - *Subject to Z development days, maximise throughput*

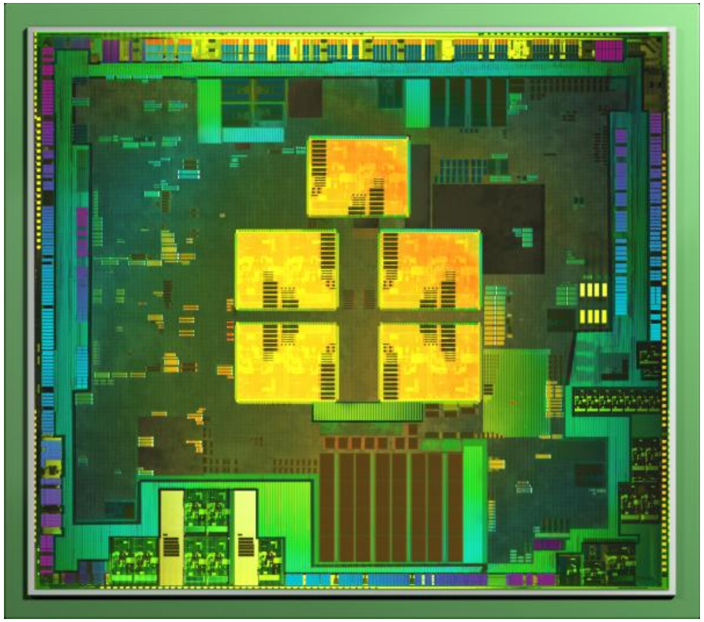
What is available to you

- Types of compute device
  - Multi-core CPUs
  - GPUs (Graphics Processing Units)
  - MPPAs (Massively Parallel Processor Arrays)
  - FPGAs (Field Programmable Gate Arrays)

What is available to you

- Types of compute device
  - Multi-core CPUs
  - GPUs (Graphics Processing Units)
  - MPPAs (Massively Parallel Processor Arrays)
  - FPGAs (Field Programmable Gate Arrays)
- Types of compute system
  - Embedded Systems
  - Mobile Phones
  - Tablets
  - Laptops
  - Grid computing
  - Cloud computing

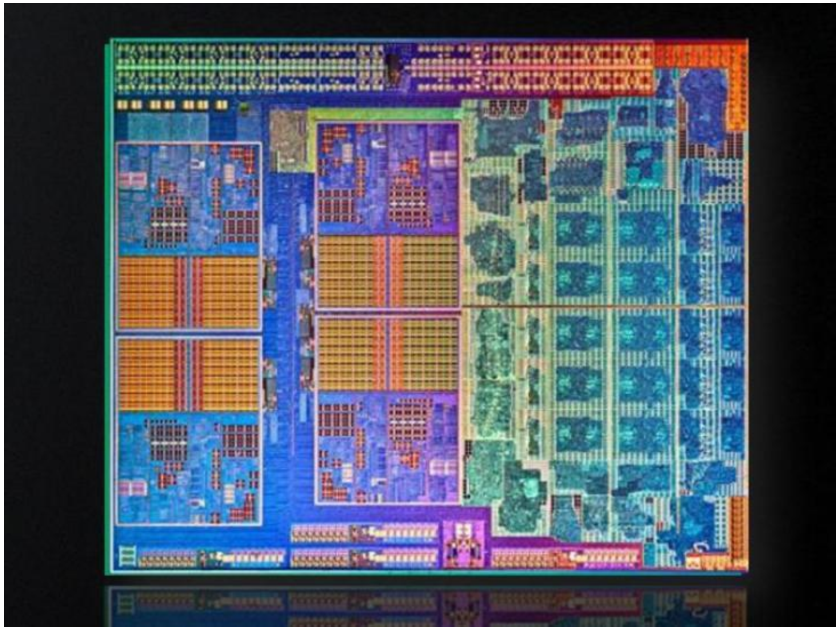
HTC Droid DNA



Snapdragon S4 Pro

- CPU : Quad-core Krait (ARM derivative)
- GPU : Adreno 320 GPU (OpenCL compatible)

Lenovo Thinkpad Edge E525



AMD Fusion A8-3500M

- CPU : Quad-Core 2.4GHz Phenom-II
- GPU : HD 6620G 400MHz (320 cores)

Img:http://laptops-specs.blogspot.com/2011/09/lenovo-thinkpad-edge-e525-specs.html, http://www.techradar.com/images/zoom/amd-llano-965315/index1

Imperial HPC Cluster

- cx1 – cluster of networked machines
  - 1395 nodes (boxes) -> 13558 CPU cores
- cx2 - SGI Altix ICE 8200 EX
  - 456 nodes -> 5272 CPU cores
  - Optimised for MPI (message processing) tasks
- ax4 – one machine: 15TB of RAM + 1280 cores

- Grid-management system
  - Run program on 1000 PCs with one command
  - Available to researchers and undergrads (if they ask nicely)

https://www.imperial.ac.uk/admin-services/ict/self-service/research-support/hpc/
https://wiki.imperial.ac.uk/pages/viewpage.action?title=High+Performance+Computing&spaceKey=HPC

Cloud Computing

- There are now ***big*** commodity cloud providers
  - Amazon Web Services (EC2) : 10x bigger than anyone else
  - Microsoft Azure : 2x bigger than all the rest
  - Google Cloud Platform : public facing cloud is fairly small (?)

Cloud Computing

- There are now ***big*** commodity cloud providers
  - Amazon Web Services (EC2) : 10x bigger than anyone else
  - Microsoft Azure : 2x bigger than all the rest
  - Google Cloud Platform : public facing cloud is fairly small (?)
- Multiple instance types
  - t2.micro: 1 CPU, 1 GB
  - g2.2xlarge: 8 CPUs, 15 GB + GPU
  - c3.8xlarge: 32 CPUs, 108 GB

Cloud Computing

- There are now ***big*** commodity cloud providers
  - Amazon Web Services (EC2) : 10x bigger than anyone else
  - Microsoft Azure : 2x bigger than all the rest
  - Google Cloud Platform : public facing cloud is fairly small (?)
- Multiple instance types
  - t2.micro: 1 CPU, 1 GB
  - g2.2xlarge: 8 CPUs, 15 GB + GPU
  - c3.8xlarge: 32 CPUs, 108 GB
- Multiple pricing options : on-demand vs. spot-price
  - On-demand : fixed price for as long as you want

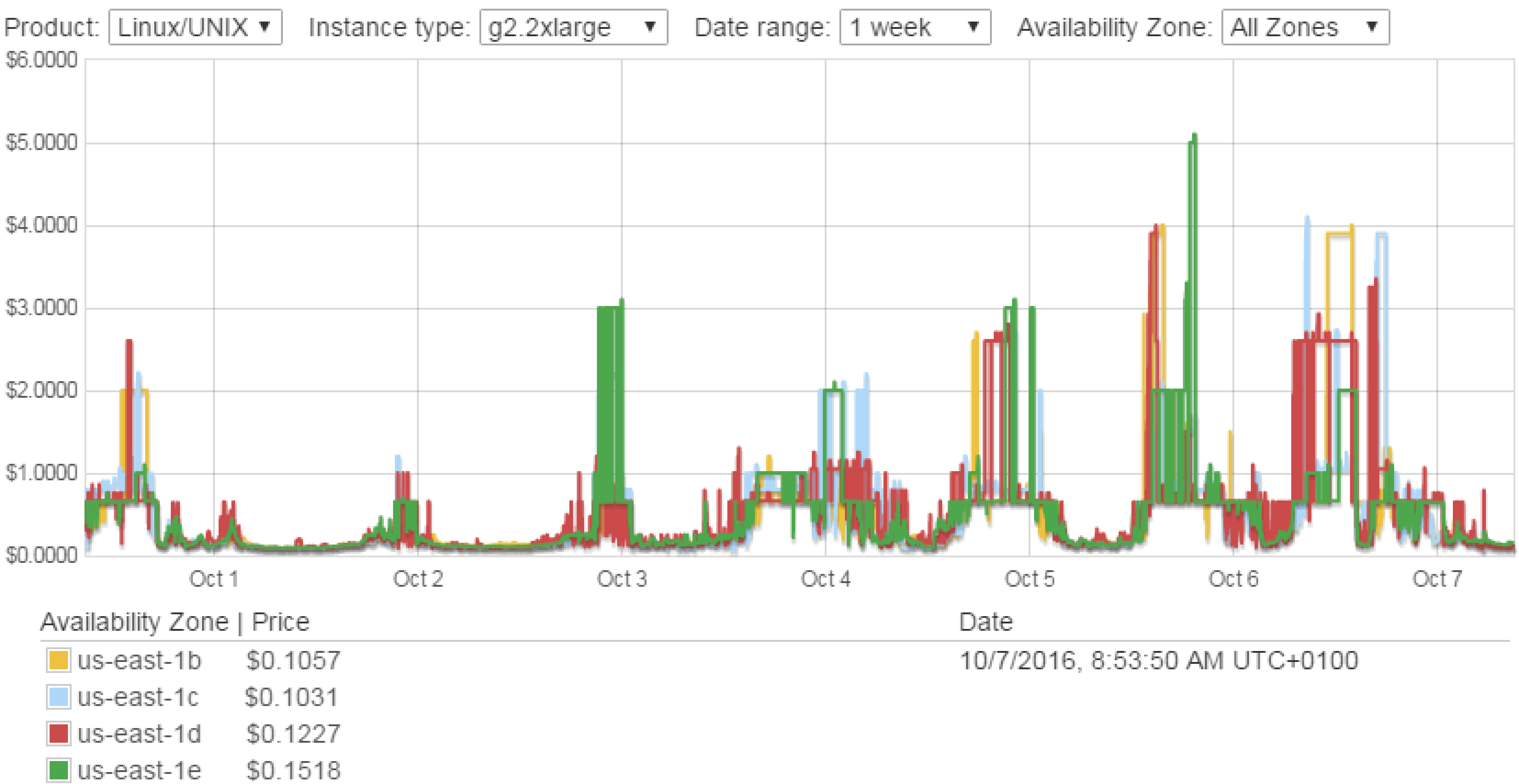
Cloud Computing

- There are now ***big*** commodity cloud providers
  - Amazon Web Services (EC2) : 10x bigger than anyone else
  - Microsoft Azure : 2x bigger than all the rest
  - Google Cloud Platform : public facing cloud is fairly small (?)
- Multiple instance types
  - t2.micro: 1 CPU, 1 GB $0.013 / hour
  - g2.2xlarge: 8 CPUs, 15 GB + GPU $0.650 / hour
  - c3.8xlarge: 32 CPUs, 108 GB $1.680 / hour
- Multiple pricing options : on-demand vs. spot-price
  - On-demand : fixed price for as long as you want

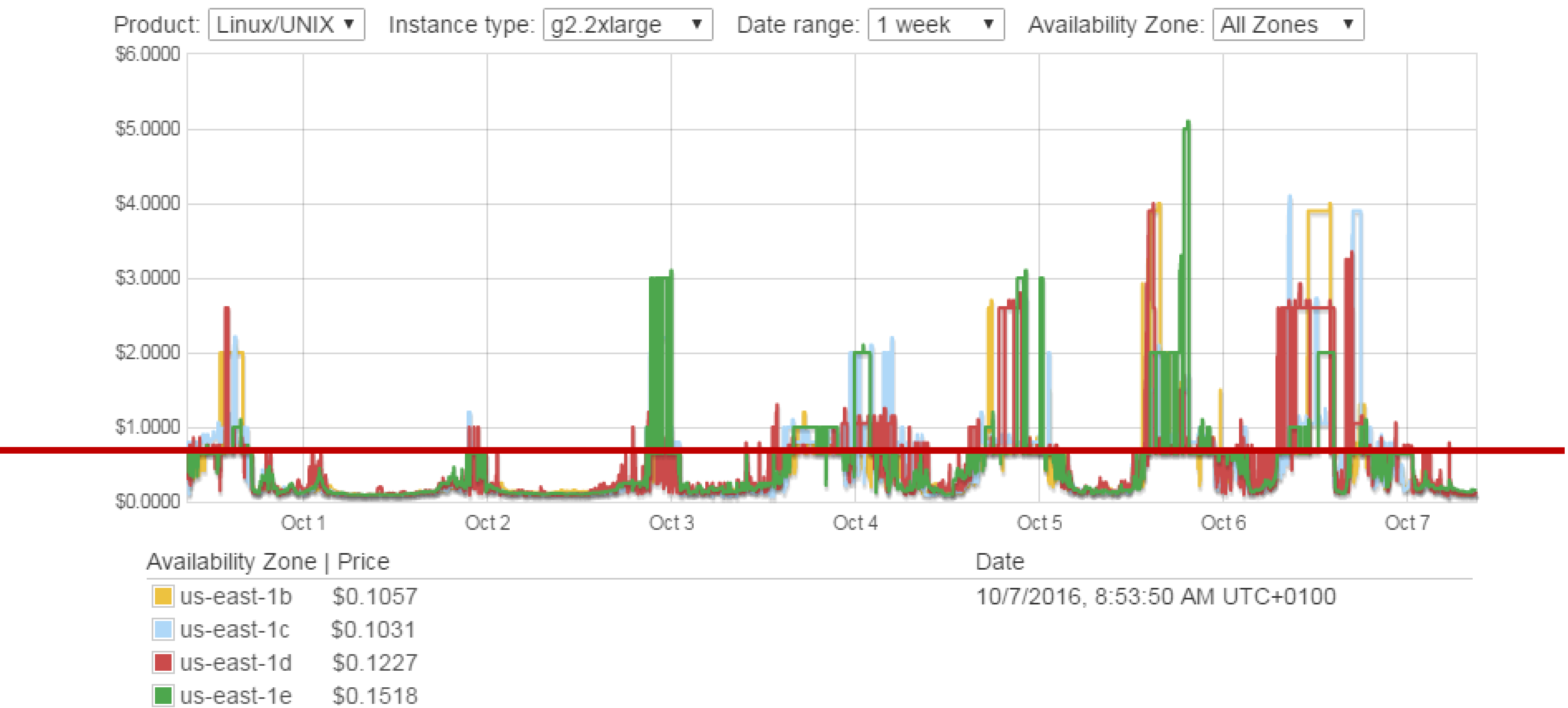
Cloud Computing

- There are now ***big*** commodity cloud providers
  - Amazon Web Services (EC2) : 10x bigger than anyone else
  - Microsoft Azure : 2x bigger than all the rest
  - Google Cloud Platform : public facing cloud is fairly small (?)
- Multiple instance types
  - t2.micro: 1 CPU, 1 GB — $0.013 / hour
  - g2.2xlarge: 8 CPUs, 15 GB + GPU — $0.650 / hour — $0.103 / hour
  - c3.8xlarge: 32 CPUs, 108 GB — $1.680 / hour — $0.311 / hour
- Multiple pricing options : on-demand vs. spot-price
  - On-demand : fixed price for as long as you want
  - Spot-price : price fluctuates according to demand

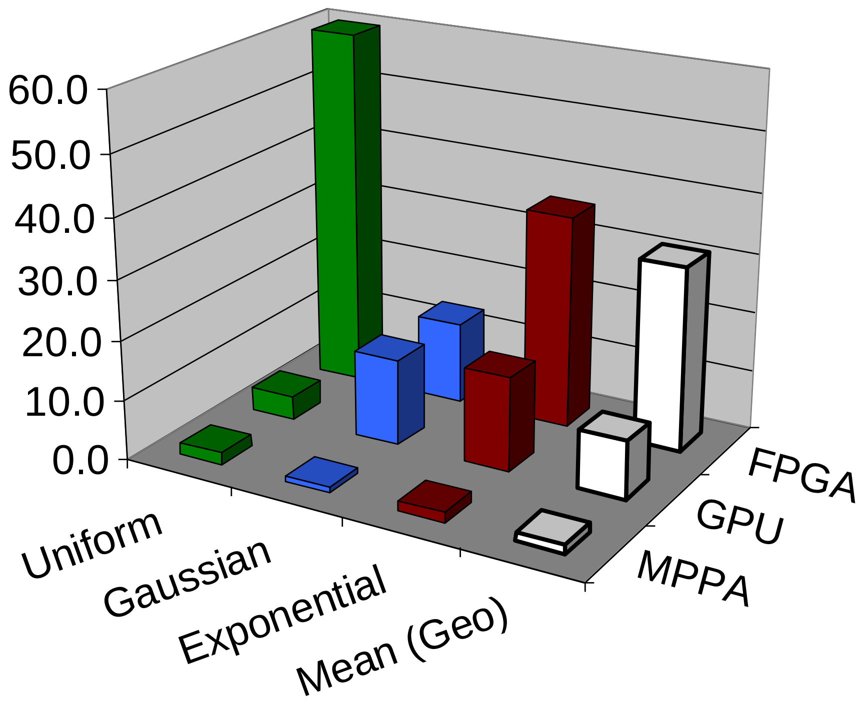
Pricing can be volatile (and make no sense)



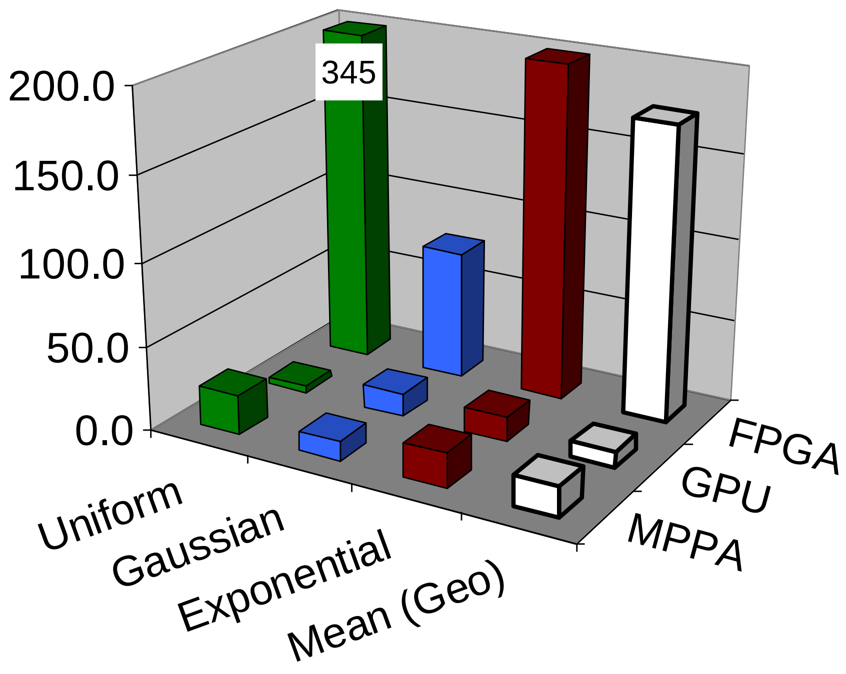
Pricing can be volatile (and make no sense)



Performance and Efficiency Relative to CPU



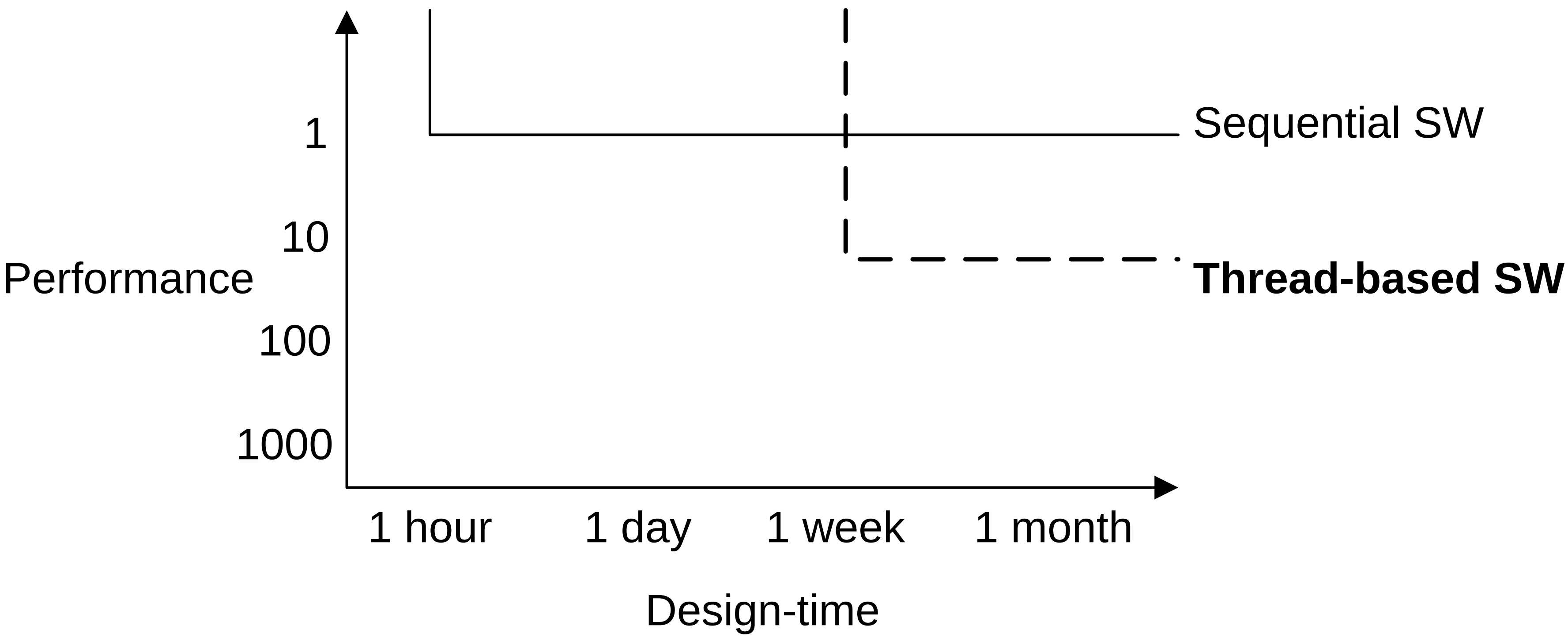
Performance



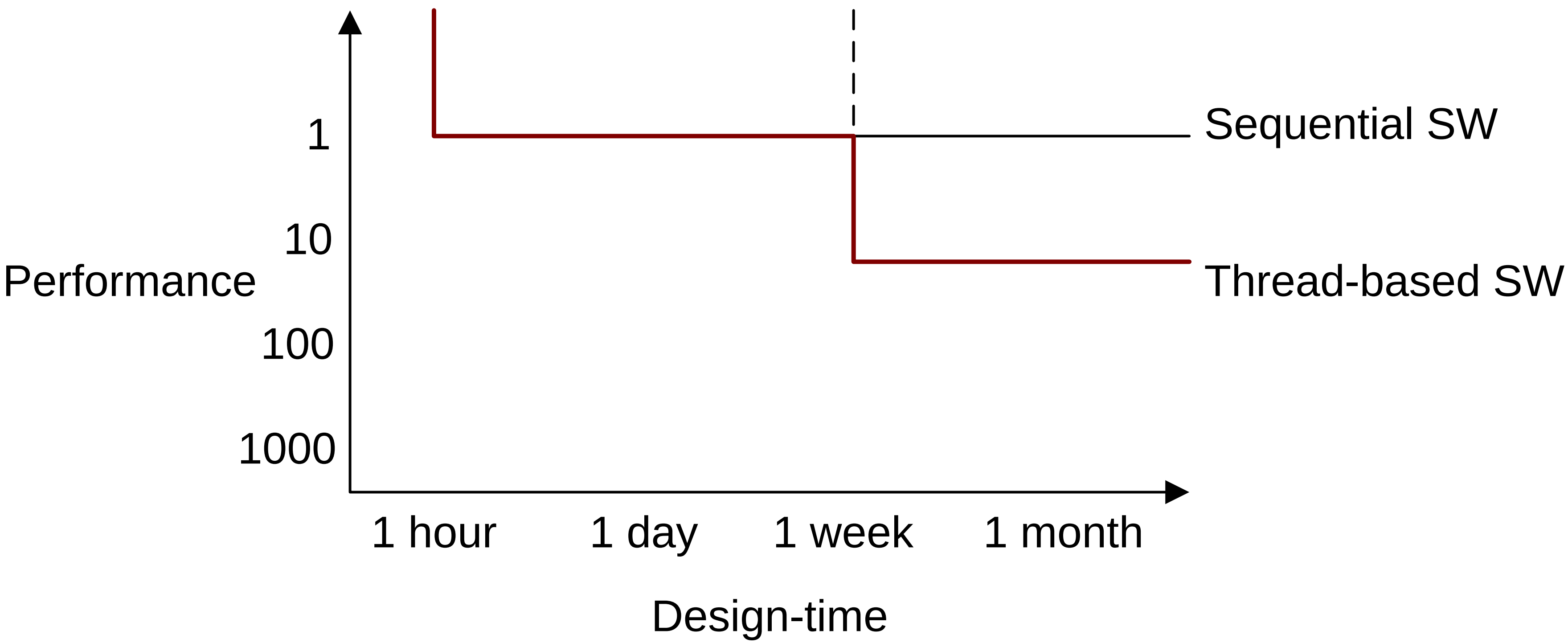
Power Efficiency

Design tradeoffs

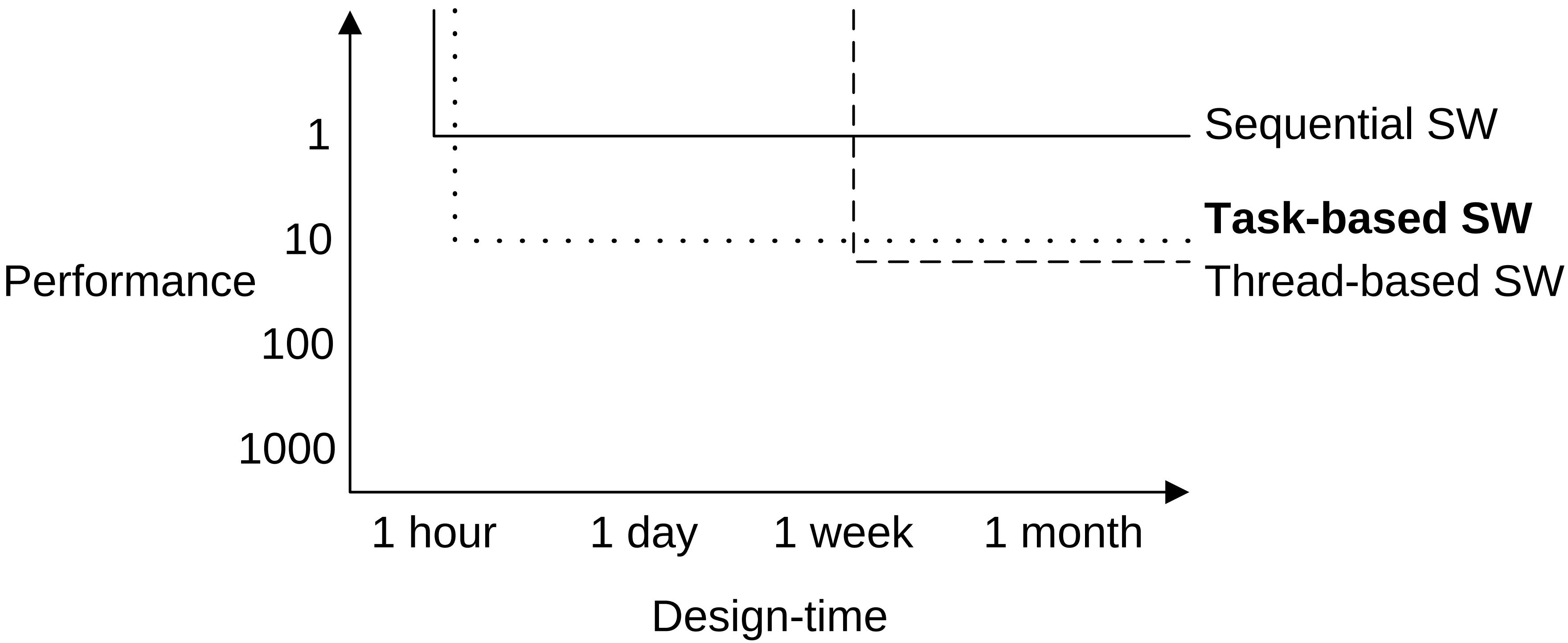
Design tradeoffs



Design tradeoffs



Design tradeoffs



Design tradeoffs

- **Task-based** parallelism vs threads
  - Easy to program (less time coding)
  - Easy to get right (less time testing)
- Many implementations and APIs
  - Intel Threaded Building Blocks (TBB)
  - Microsoft .NET Task Parallel Library
  - OpenCL

ntial SW

ased SW

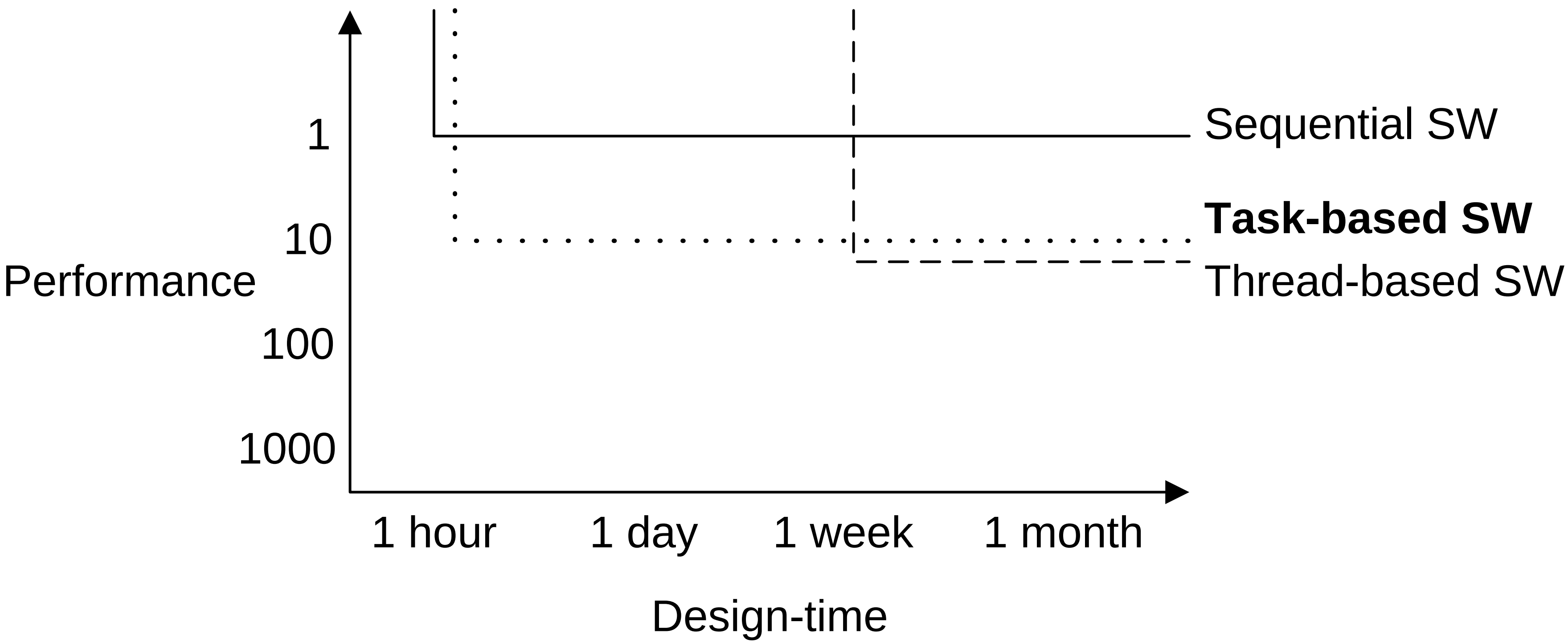
Performance

-based SW

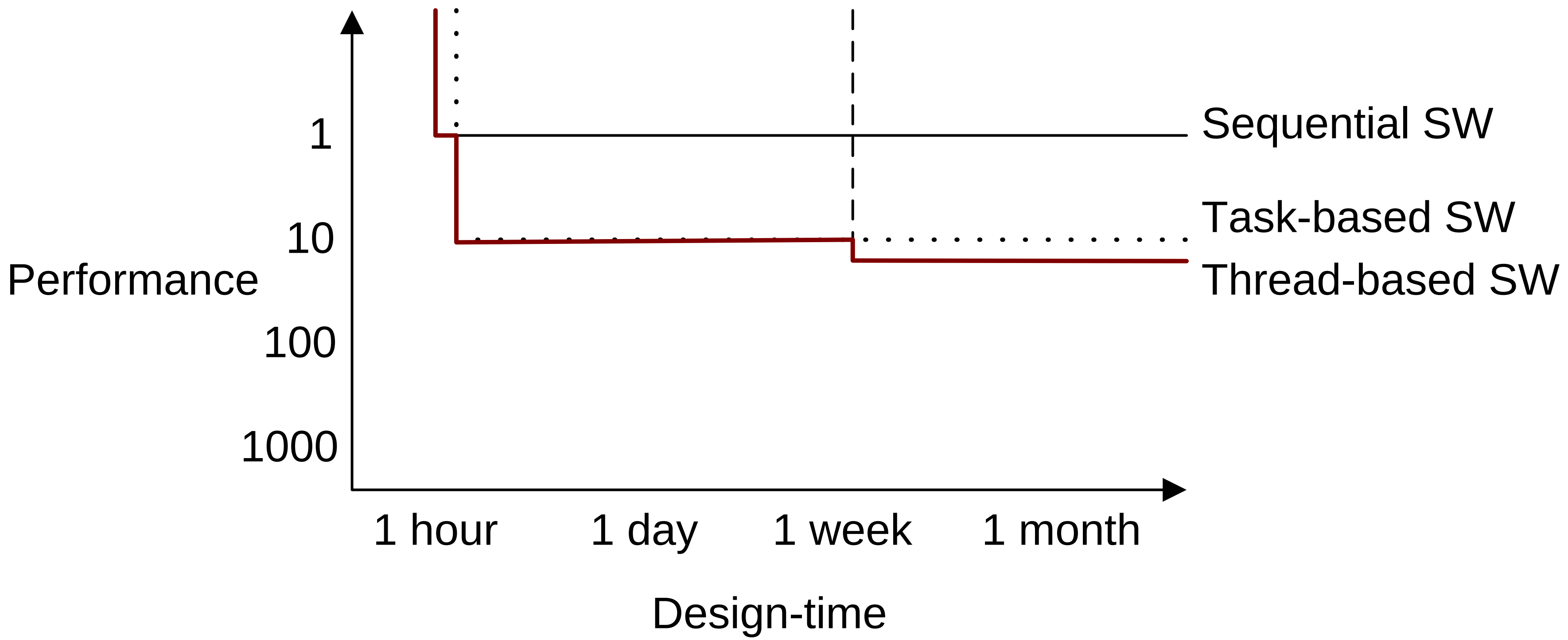
1

Design-time

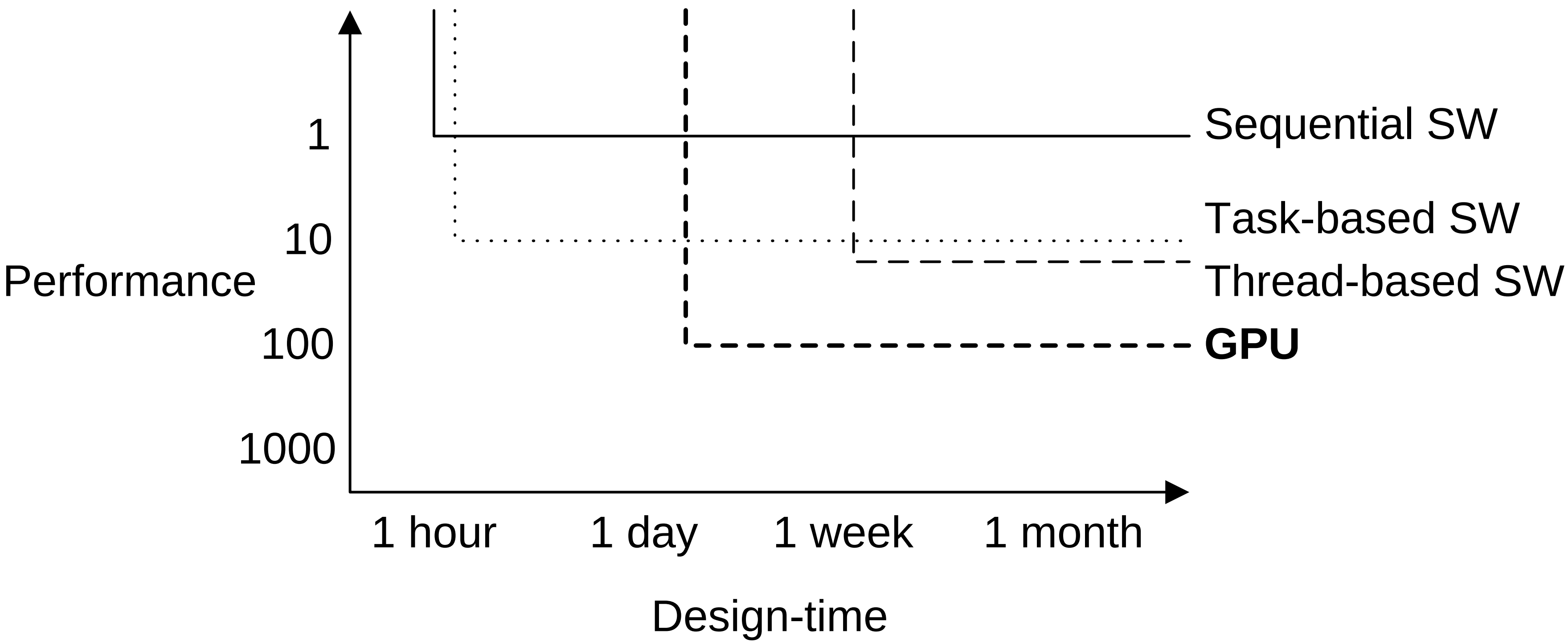
Design tradeoffs



Design tradeoffs

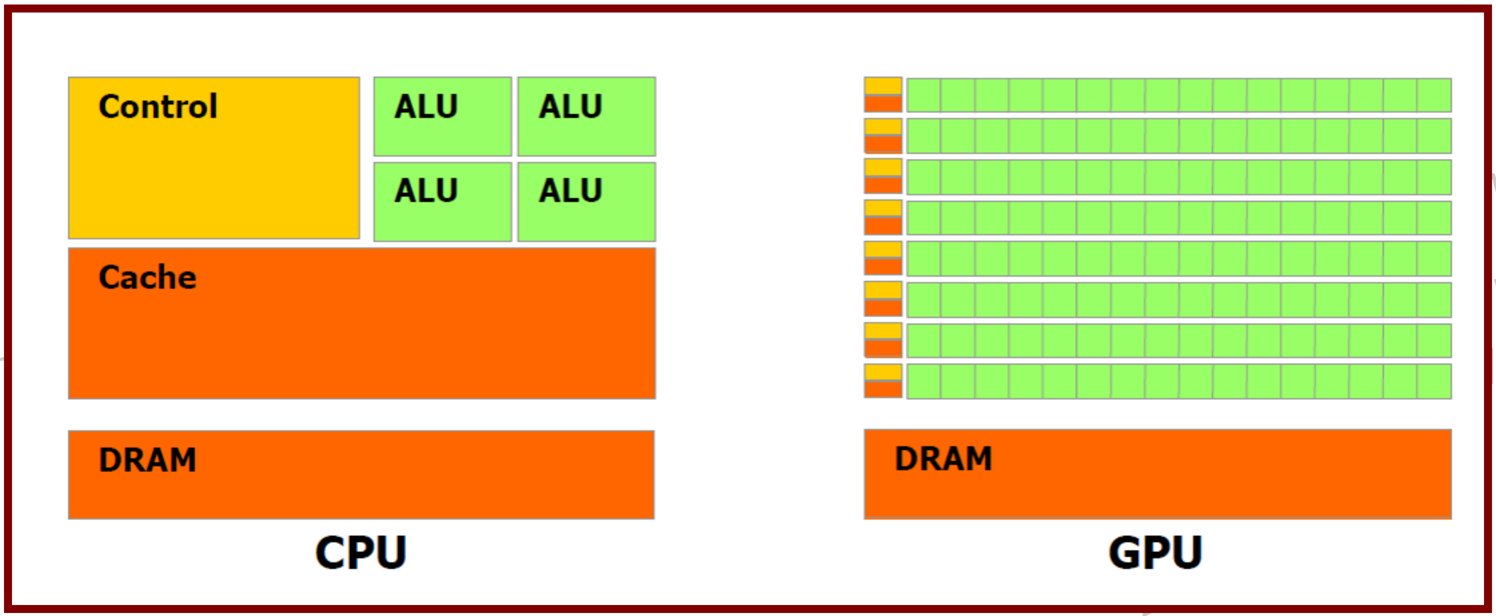


Design tradeoffs



Design tradeoffs

Perform



V

W

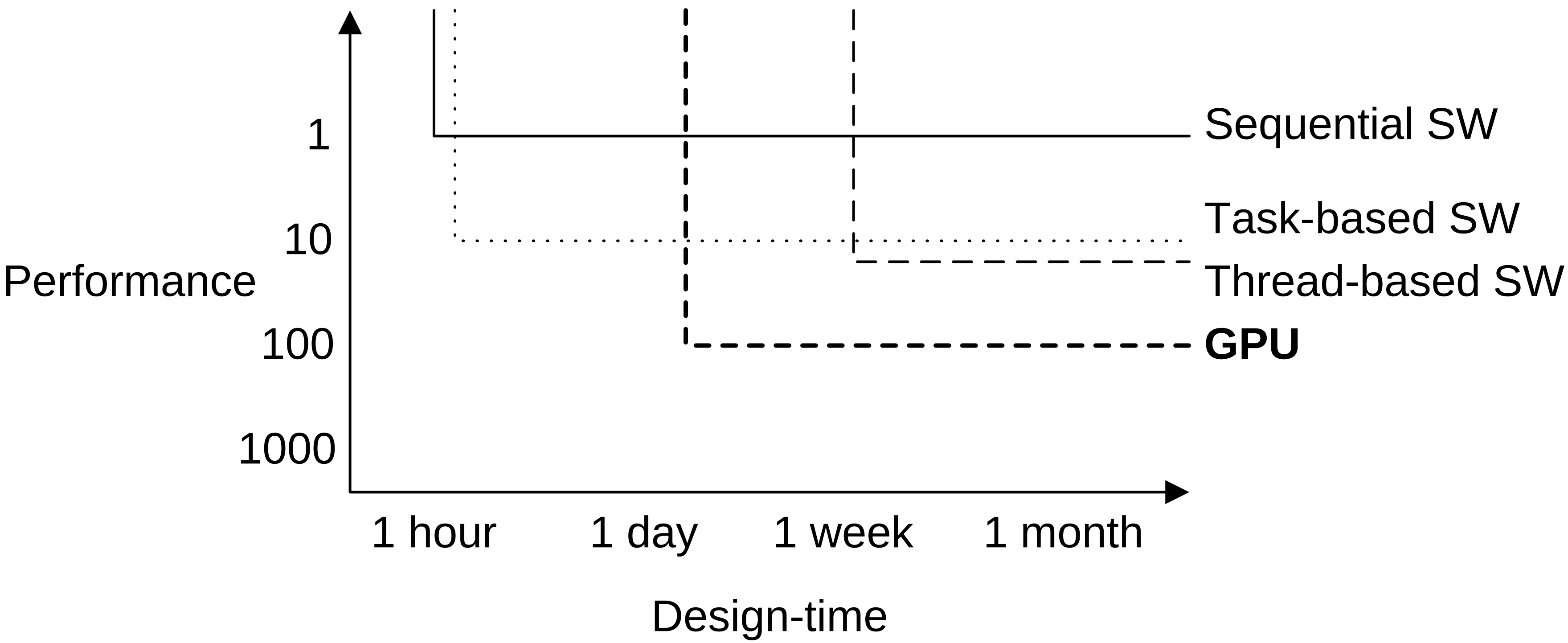
SW

1 hour 1 day 1 week 1 month

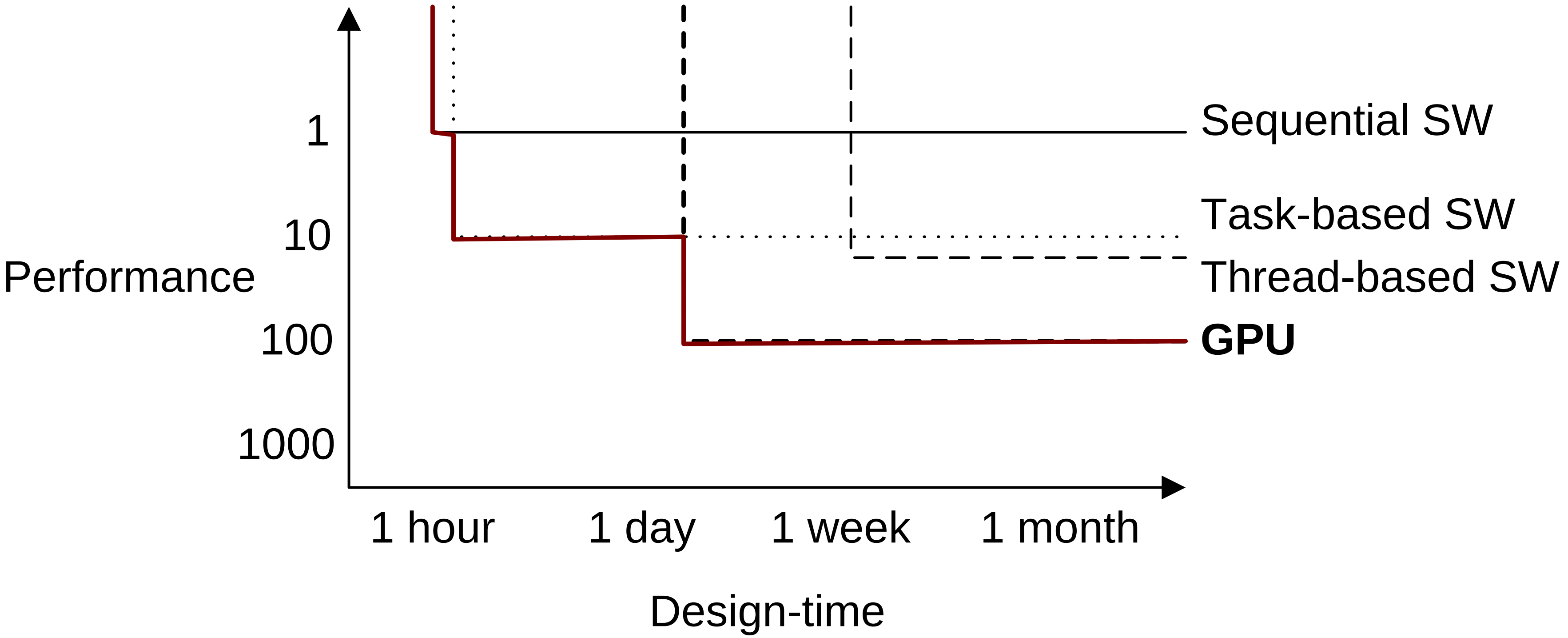
Design-time

Src: NVIDIA CUDA Compute Unified Device Architecture, Programmers Guide

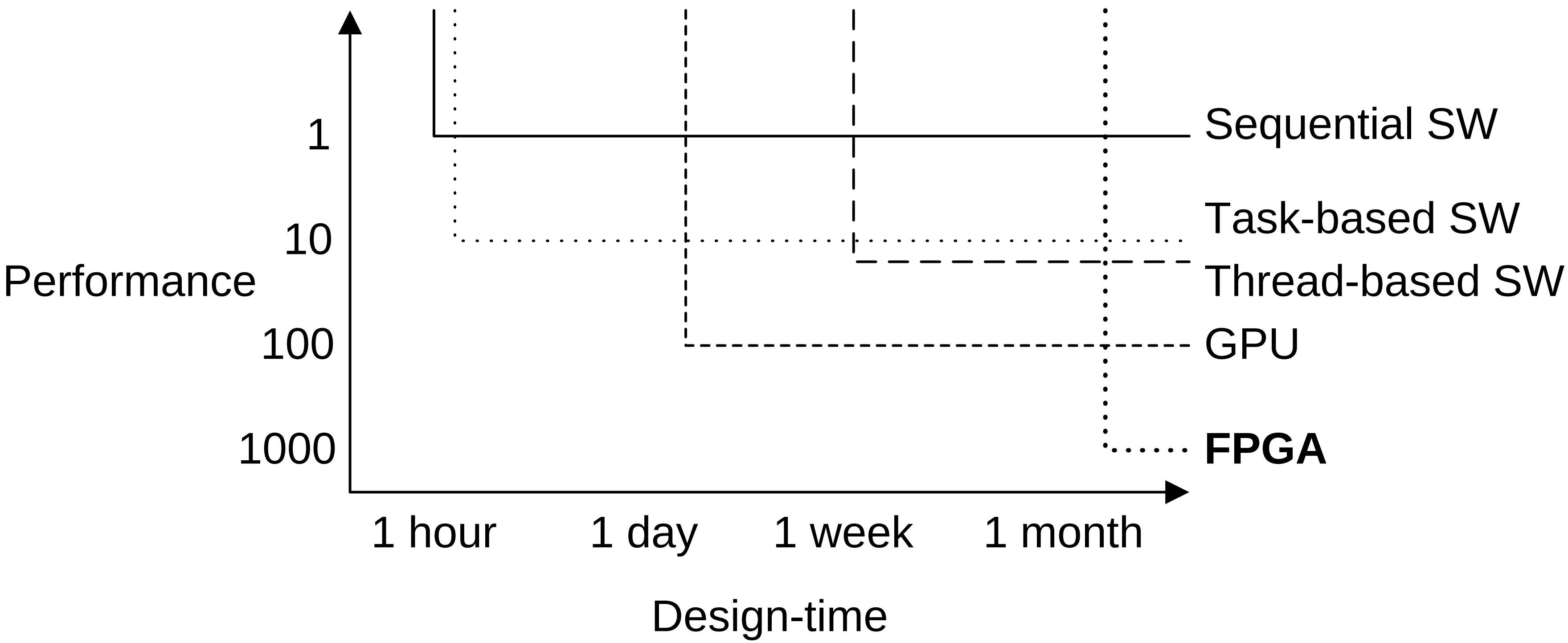
Design tradeoffs



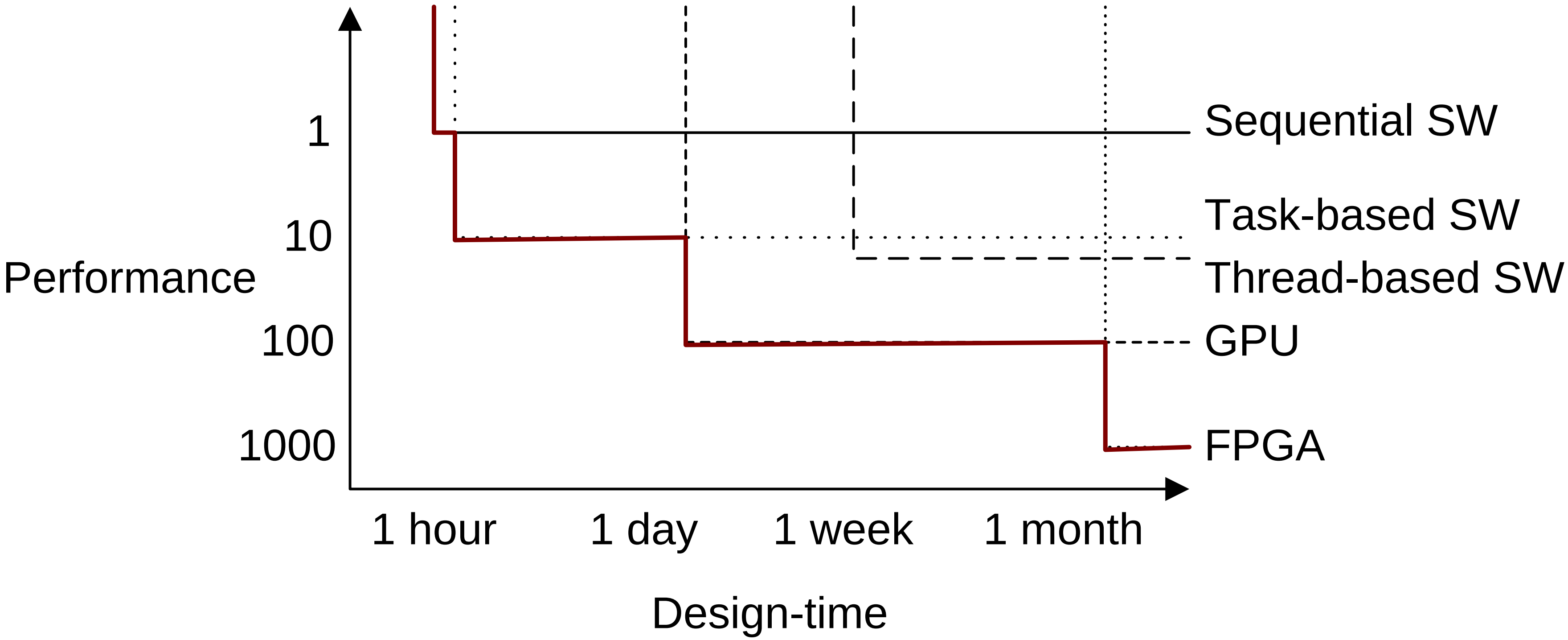
Design tradeoffs



Design tradeoffs

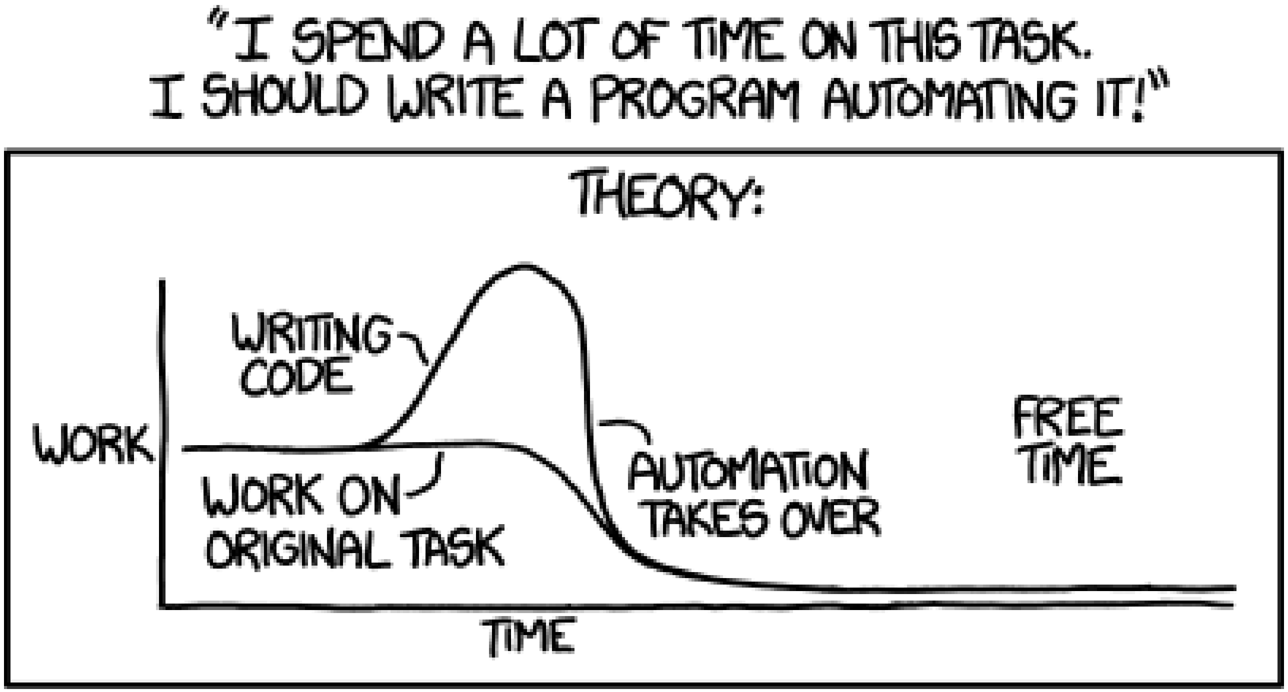


Design tradeoffs



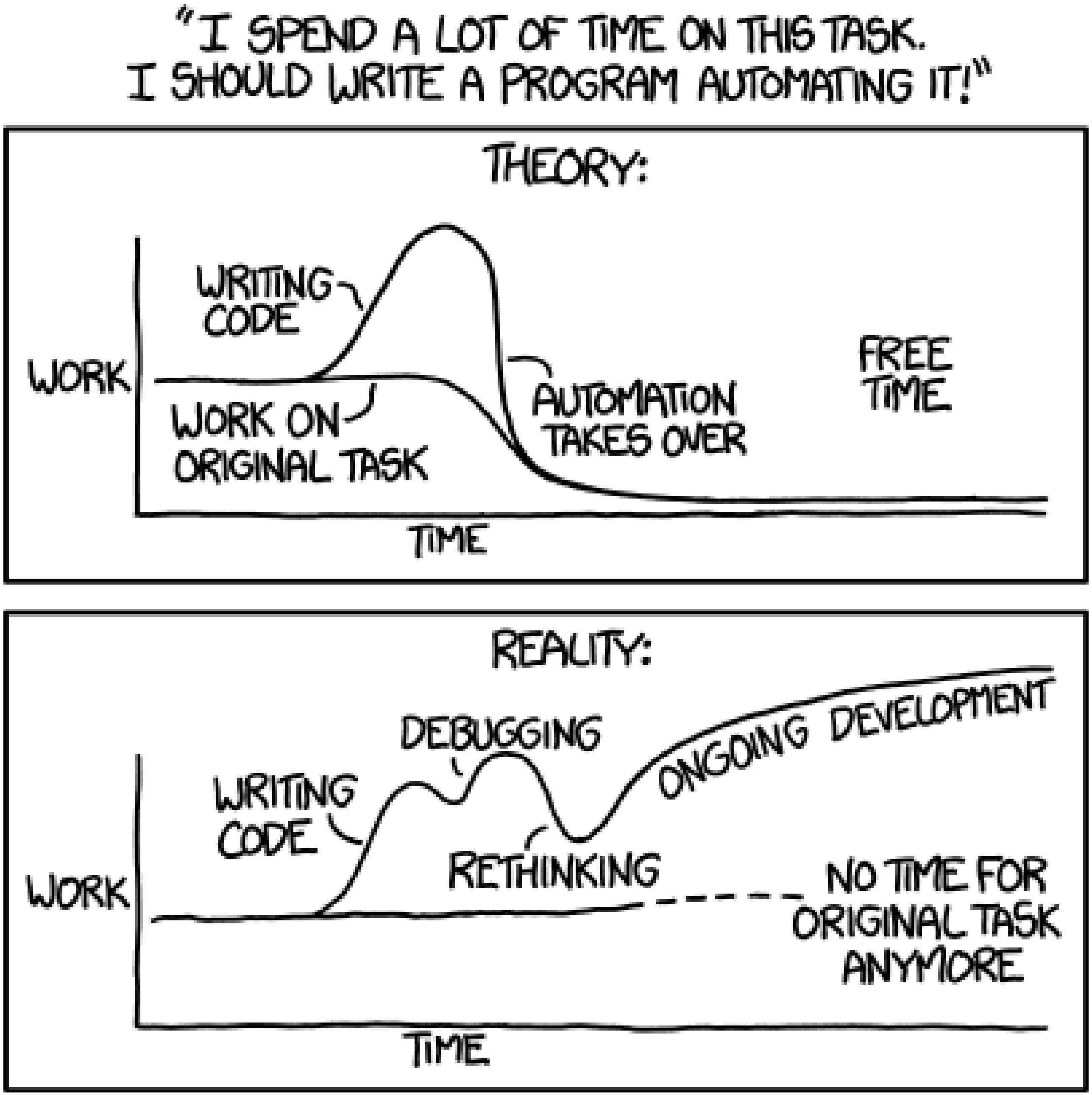
What you will learn

- **Systems**: what high-performance systems are available
- **Methods**: how these systems can be programmed
- **Practise**: concrete experience with multi-core and GPUs
- **Analysis**: knowing what to use and when
- **Tools**: making better use of your time



Developer productivity
is also part of performance

http://xkcd.com/1319/



http://xkcd.com/1319/

Re: XKCD - My Professional Context

- Undergraduate degree and PhD from Computing
  - If pushed, I self-identify as a “programmer”
- Research focuses on hardware acceleration
  - Both academic and industrial applications
- My motivation for this course
  - Supervising final year project students
  - Working with PhD students
  - Talking to industry people

Why are you here?

Course Assessment

- 40% : Four short exercises to build skills
  - Get familiar with environments and how to do common tasks
  - Structured and quite linear – should not be taxing
  - Force people to do work earlier in term

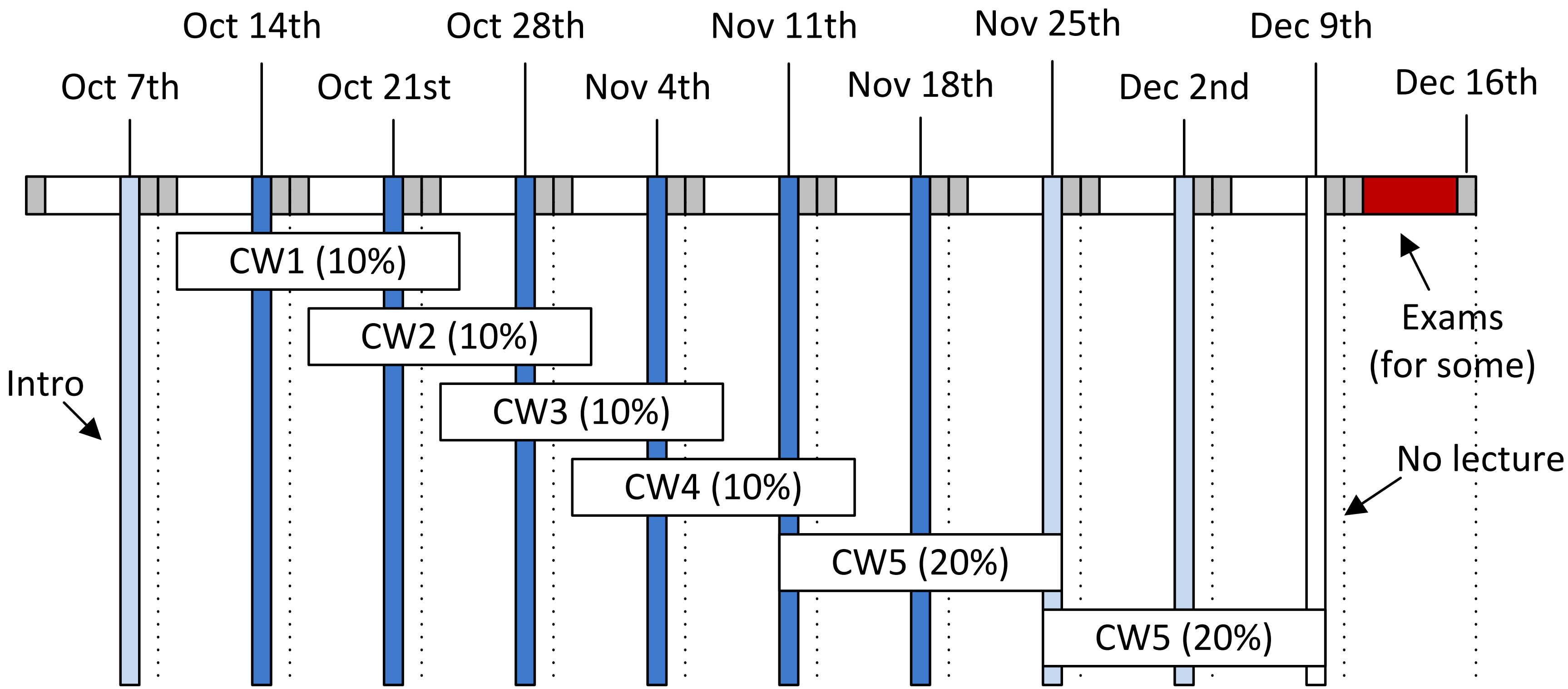
Course Assessment

- 40% : Four short exercises to build skills
  - Get familiar with environments and how to do common tasks
  - Structured and quite linear – should not be taxing
  - Force people to do work earlier in term
- 40% : Two larger tasks to apply skills to real problems
  - Allow demonstration of knowledge and skills
  - Unstructured; open-ended; competitive; hard

Course Assessment

- 40% : Four short exercises to build skills
  - Get familiar with environments and how to do common tasks
  - Structured and quite linear – should not be taxing
  - Force people to do work earlier in term
- 40% : Two larger tasks to apply skills to real problems
  - Allow demonstration of knowledge and skills
  - Unstructured; open-ended; competitive; hard
- 20% : Oral assessment; individual
  - 30 minutes each, will happen at the start of spring term
  - Test ability to communicate about your code and solutions
  - (Check that you did the work)

Timetable



- Moved to 2-hour blocks (Bleh. Makes the timetable easier)
- Try to place lectures *within* first four courseworks
- Avoid exams at the end of term

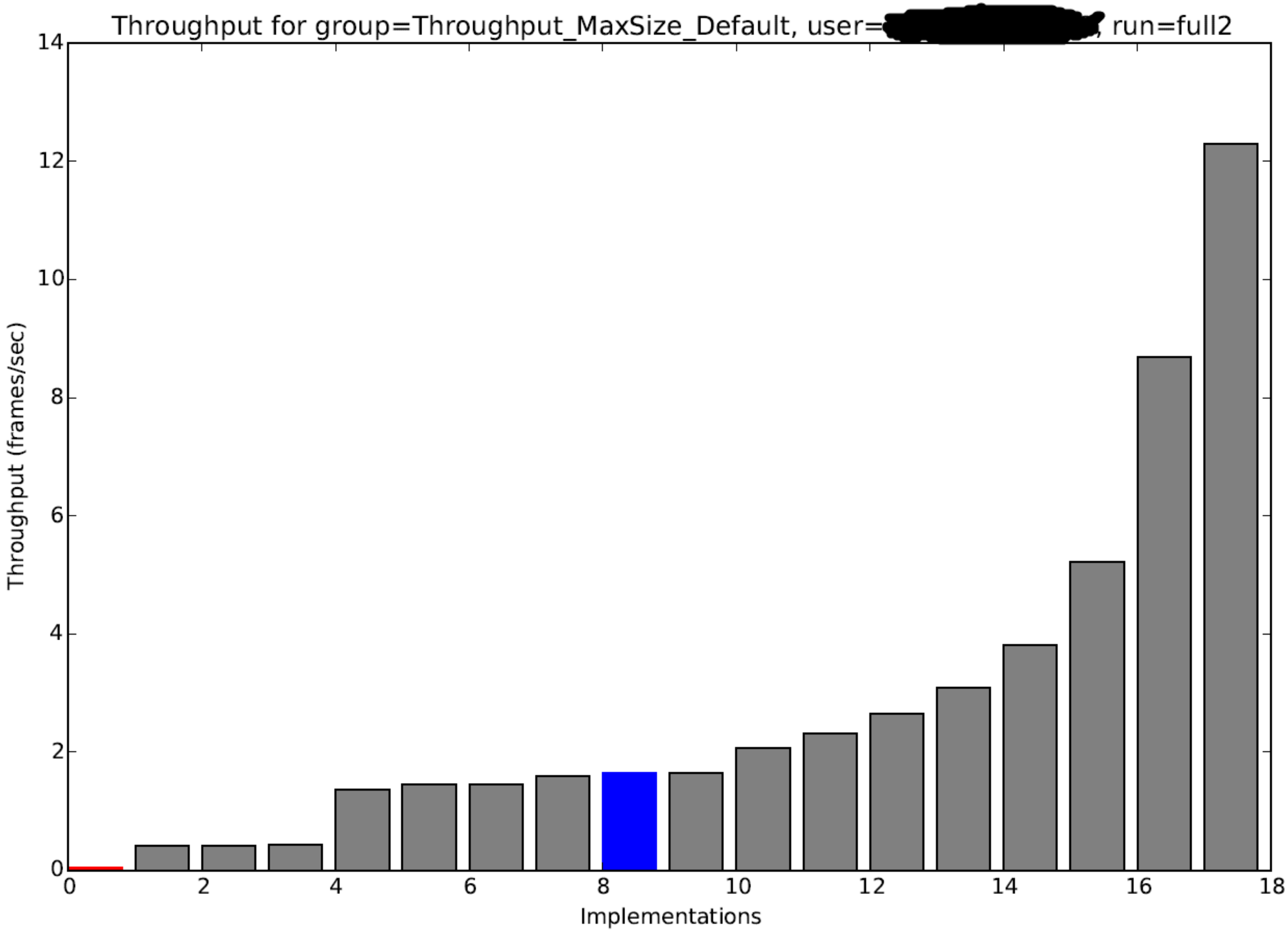
Feedback

- Feedback != grades
  - Feedback is formative : what worked, what is going well, ...
- 100% coursework isn't intended to give instant marks
  - But... it is supposed to have fast feedback
  - Should be fast enough to be useful during learning process

Feedback

- Feedback != grades
  - Feedback is formative : what worked, what is going well, ...
- 100% coursework isn't intended to give instant marks
  - But... it is supposed to have fast feedback
  - Should be fast enough to be useful during learning process

- Looking at previous years:
  - Success : discussions with me + students via github issues and PRs
  - Success : feedback during CW5 and CW6 via git
  - Success : orals are a good point for reflection (students say so!)
  - Failure : timing of CW1-CW4. Too variable, takes too long


Throughput for group=Throughput_MaxSize_Default, user=[redacted], run=full2
Throughput (frames/sec)
Implementations
0
2
4
6
8
10
12
14
16
18

Approach to CW1-CW4 feedback

- CW1-CW4 are ***supposed*** to be easy
  - Everyone should be able to get 100%
  - (as a consequence, CW5+CW6+oral are marked on wide range)

Approach to CW1-CW4 feedback

- CW1-CW4 are ***supposed*** to be easy
  - Everyone should be able to get 100%
  - (as a consequence, CW5+CW6+oral are marked on wide range)
- Problem: assessment is mechanical but breaks
  - Student's code tends to fail in weird ways

Approach to CW1-CW4 feedback

- CW1-CW4 are ***supposed*** to be easy
  - Everyone should be able to get 100%
  - (as a consequence, CW5+CW6+oral are marked on wide range)
- Problem: assessment is mechanical but breaks
  - Student’s code tends to fail in weird ways
- Solution: enable self-assessment
  - Assessment scripts distributed in 2nd week of CW1-CW4
  - Students can run it locally and see how it fails
  - Can iterate on it till it works, get immediate result
  - If committed to git, it will get run remotely as well

Approach to CW1-CW4 feedback

- CW1-CW4 are ***supposed*** to be easy
  - Everyone should be able to get 100%
  - (as a consequence, CW5+CW6+oral are marked on wide range)
- Problem: assessment is mechanical but breaks
  - Student's code tends to fail in weird ways
- Solution: enable self-assessment
  - Assessment scripts distributed in 2nd week of CW1-CW4
  - Students can run it locally and see how it fails
  - Can iterate on it till it works, get immediate result
  - If committed to git, it will get run remotely as well
- Formative feedback is on demand
  - Ask a question about submission on github
  - Ask a question in class

Skills needed

- Basic programming
  - If you can’t program in _any_ language then worry

Skills needed

- Basic programming
  - If you can’t program in _any_ language then worry
- Intel TBB uses C++ rather than C
  - Some weird C++ stuff, but not scary: *explained in lectures*
  - Setup and basics covered in coursework
- GPU programming uses OpenCL (C-like)
  - Let’s you use whatever graphics card you happen to have
  - Working examples, explained in lectures
  - Language and compiler setup covered in coursework

- Not expected to become a guru, just make it faster

Key Focus: Engineering

- How does this apply to you?

- Examples from Elec. Eng. problems
  - Mathematical analysis
  - Simulation of digital circuits
  - VLSI circuit layout
  - Communication channel evaluation

- Tools and languages used in EE
  - C / C++
  - MATLAB

Course admin

- Slides on the course homepage
  - https://github.com/HPCE/hpce-2016
- Blackboard site is not used very much
  - (Why? Because I can automate git. No clicks)
- Other tools/sites we will be using
  - github : various forms of code distribution and submission
  - AWS (Amazon Web Services) for multi-core and GPUs later on
- Bring a device to lectures (laptop, tablet, charged phone)

The almighty git

- Git (and github) is used extensively in this course
  - As a method of distributing information + coursework
  - As a means of communication and clarification (issues)
  - As a way to provide online feedback (pushes during CW)
  - To allow pair-working between students
  - As a way to submit code (for later courseworks)

The almighty git

- Git (and github) is used extensively in this course
  - As a method of distributing information + coursework
  - As a means of communication and clarification (issues)
  - As a way to provide online feedback (pushes during CW)
  - To allow pair-working between students
  - As a way to submit code (for later courseworks)

- You don’t need to know git already
  - It isn’t that complicated anyway

The almighty git

- Git (and github) is used extensively in this course
  - As a method of distributing information + coursework
  - As a means of communication and clarification (issues)
  - As a way to provide online feedback (pushes during CW)
  - To allow pair-working between students
  - As a way to submit code (for later courseworks)

- You don’t need to know git already
  - It isn’t that complicated anyway

- You ***do*** need a github account

Platforms

- I don't care what platform/OS you use, as long as:
  - You have access to a bash-like command line
  - You have a fairly modern C++ compiler
  - There is more than one CPU
- Reasonable choices are:
  - Windows (*wsl* or *mingw* or vm+linux)
  - OS-X (using *brew* or *ports*)
  - Linux

Platforms

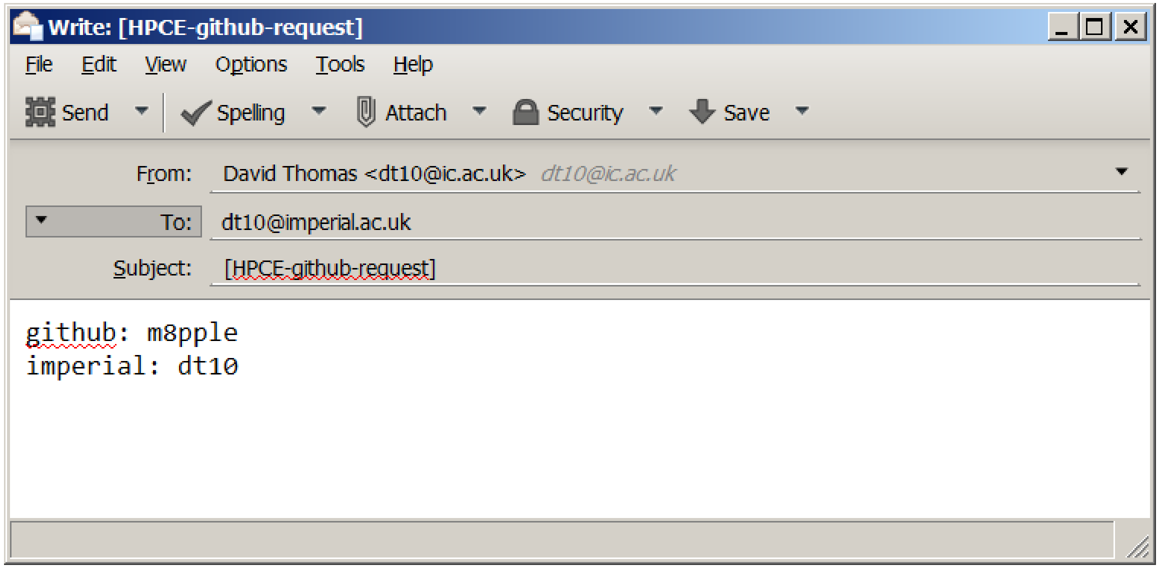
- I don't care what platform/OS you use, as long as:
  - You have access to a bash-like command line
  - You have a fairly modern C++ compiler
  - There is more than one CPU
- Reasonable choices are:
  - Windows (*wsl* or *mingw* or vm+linux)
  - OS-X (using *brew* or *ports*)
  - Linux
- You are responsible for your platform
  - There is setup info in the coursework
  - I can help you, and you can help each other
  - Note: you can do dev on one platform, eval on another

Platforms

- I don't care what platform/OS you use, as long as:
  - You have access to a bash-like command line
  - You have a fairly modern C++ compiler
  - There is more than one CPU
- Reasonable choices are:
  - Windows (*wsl* or *mingw* or vm+linux)
  - OS-X (using *brew* or *ports*)
  - Linux
- You are responsible for your platform
  - There is setup info in the coursework
  - I can help you, and you can help each other
  - Note: you can do dev on one platform, eval on another
- Eventually you will use AWS GPU instances
  - No GUI. Not even on this continent.

Action

- If you want to take this course then:
  1. Get a github account
  2. Send me an email:
     - Subject: “[HPCE-github-request]”
     - Body: github id + your Imperial ***login*** (the short one)



How do you do well in this course?

Simple example : Totient function

- Eulers totient function: totient(n)
  - Number of integers in range 1..n which are relatively prime to n
  - Integers i and j are relatively prime if gcd(i,j)=1

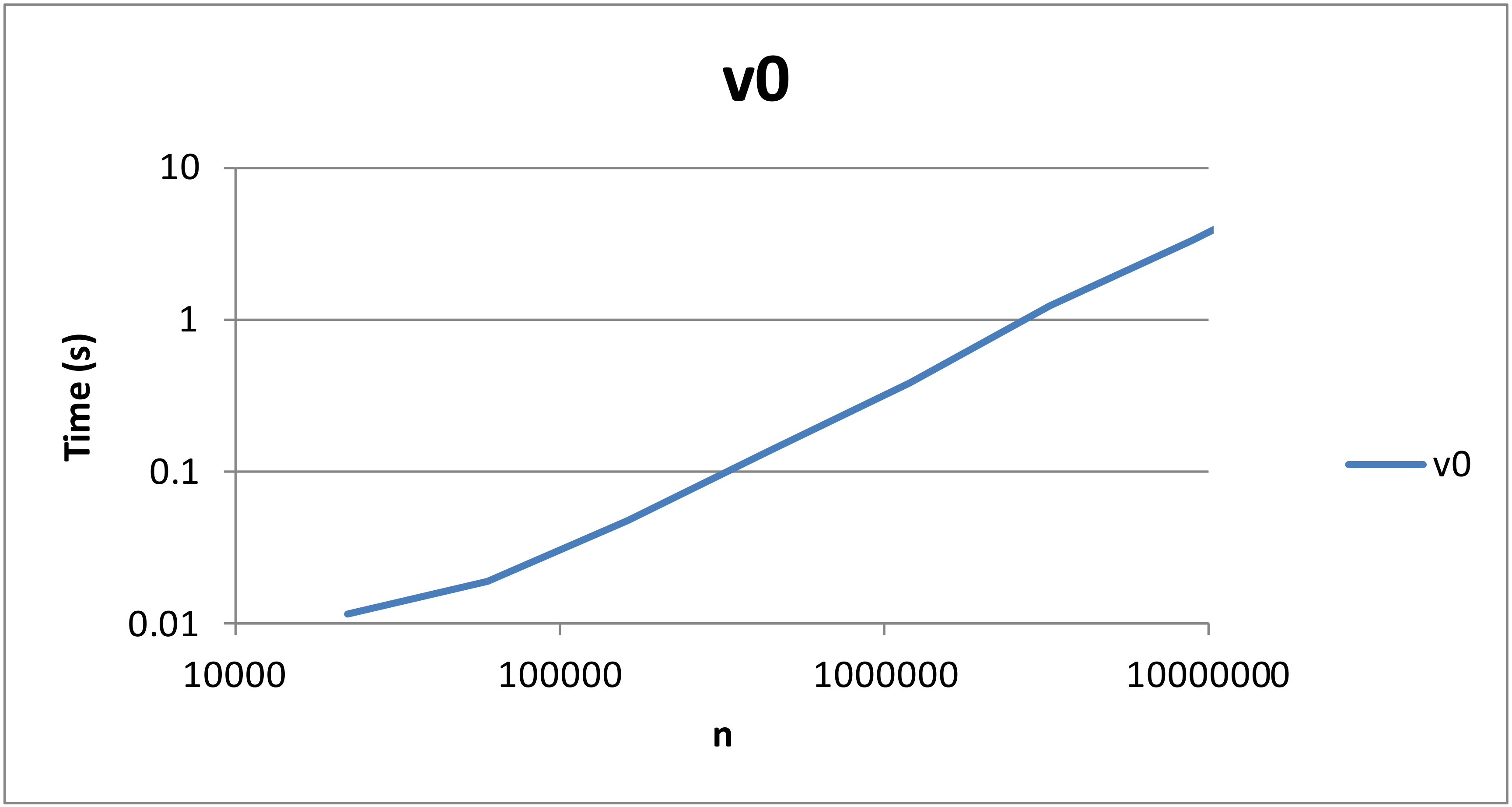
Version 0 : Simple loop

- Eulers totient function: totient(n)
  - Number of integers in range 1..n which are relatively prime to n
  - Integers i and j are relatively prime if gcd(i,j)=1

```
unsigned totient_v0(unsigned begin, unsigned end)
{
    unsigned count=0;

    for(unsigned i=begin; i<end; i++){
        count = count + gcd(i);
    }

    return count;
}
```

/vagrant/lec0

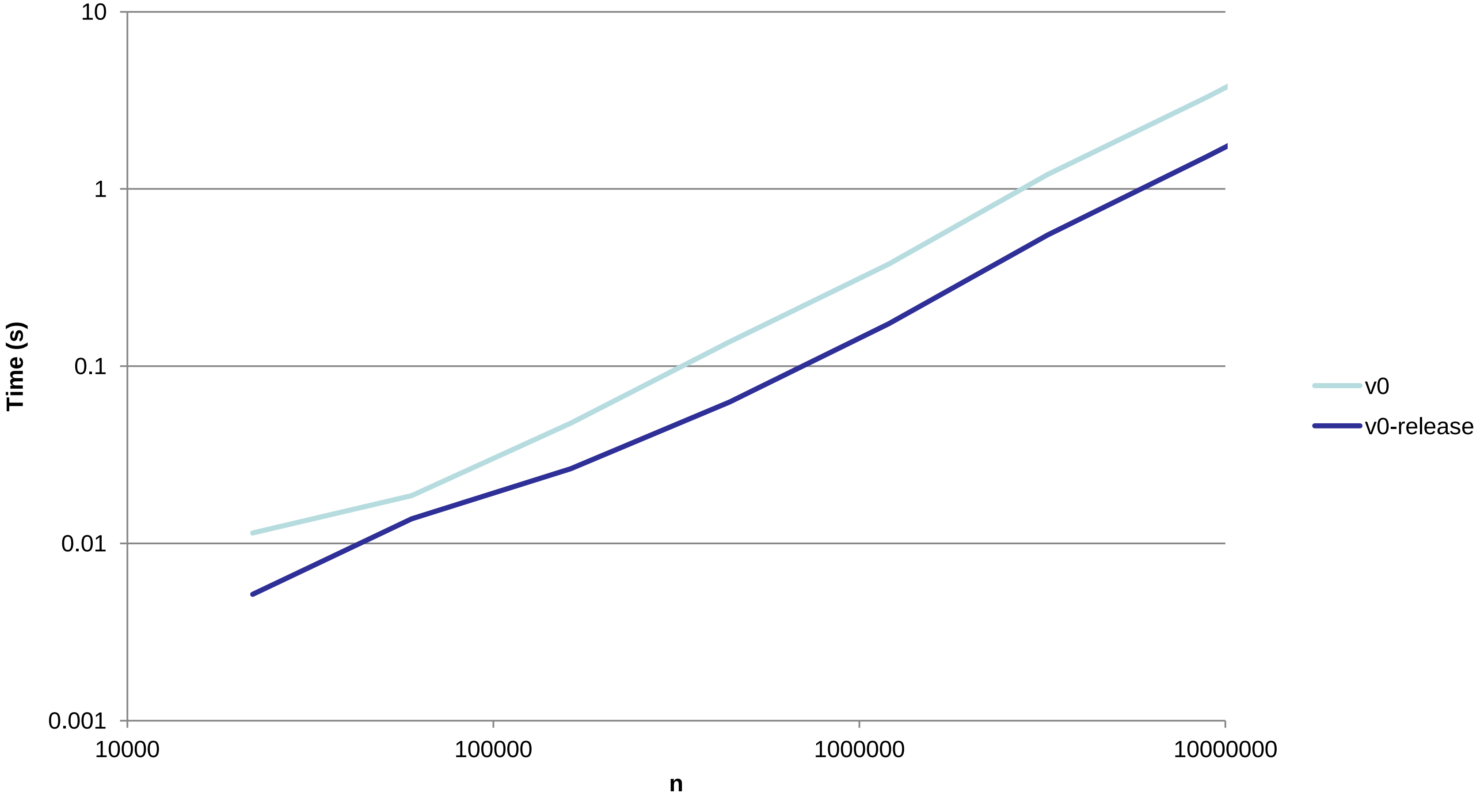
```
vagrant@debiancontrib-jessie /vagrant/lec0
$

vagrant@debiancontrib-jessie /vagrant/lec0
$ g++ -std=c++11 -o totient_v0 totient_v0.cpp

vagrant@debiancontrib-jessie /vagrant/lec0
$ ES=" 10 11 12 13 14 15 16 17 18";

vagrant@debiancontrib-jessie /vagrant/lec0
$ for e in $ES; do ./totient_v0 $e; done
e^10.000,      22026,       7340,  0.011545
e^11.000,      59874,      18752,  0.029131
e^12.000,     162754,      77076,  0.047173
e^13.000,     442413,     294936,  0.141477
e^14.000,    1202604,     369408,  0.384869
e^15.000,    3269017,    3264016,  1.241035
e^16.000,    8886110,    3367296,  3.366319
e^17.000,   24154952,   11599680,  9.706297
^C

vagrant@debiancontrib-jessie /vagrant/lec0
$ |
```


v0
10
1
0.1
0.01
Time (s)
10000
100000
1000000
10000000
n
v0

Turn on optimisation!

/vagrant/lec0

```
e^13.000,     442413,     294936,  0.141477
e^14.000,    1202604,     369408,  0.384869
e^15.000,    3269017,    3264016,  1.241035
e^16.000,    8886110,    3367296,  3.366319
e^17.000,   24154952,   11599680,  9.706297
^C

vagrant@debiancontrib-jessie /vagrant/lec0
$ g++ -O3 -std=c++11 -o totient_v0 totient_v0.cpp

vagrant@debiancontrib-jessie /vagrant/lec0
$ for e in $ES; do ./totient_v0 $e; done
e^10.000,      22026,       7340,  0.005334
e^11.000,      59874,      18752,  0.015950
e^12.000,     162754,      77076,  0.031679
e^13.000,     442413,     294936,  0.066495
e^14.000,    1202604,     369408,  0.180856
e^15.000,    3269017,    3264016,  0.574568
e^16.000,    8886110,    3367296,  1.570109
e^17.000,   24154952,   11599680,  4.524068
^C

vagrant@debiancontrib-jessie /vagrant/lec0
$
```


10
1
0.1
0.01
0.001
Time (s)
10000
100000
1000000
10000000
n
v0
v0-release

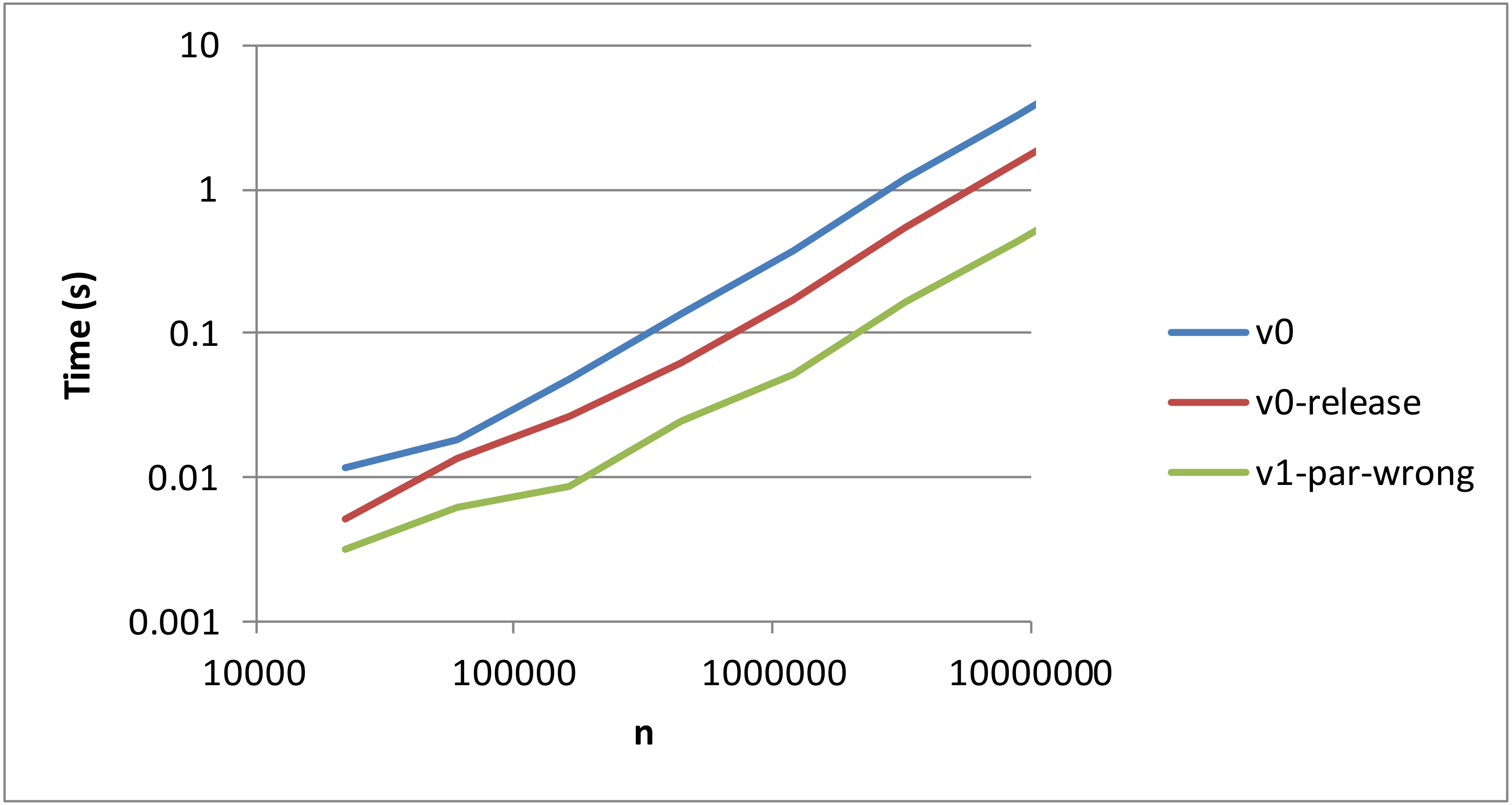
Convert the for loop to parallel loop

```
#include "tbb/parallel_for.h"

uint64_t totient_v1(uint64_t n)
{
    uint64_t count=0;

    //for(uint64_t i=1; i<=n; i++){
    tbb::parallel_for(uint64_t(1), (n+1), [&](uint64_t i){
        if(gcd(i,n)==1){
            count = count + 1;
        }
    });

    return count;
}
```

Compile with TBB

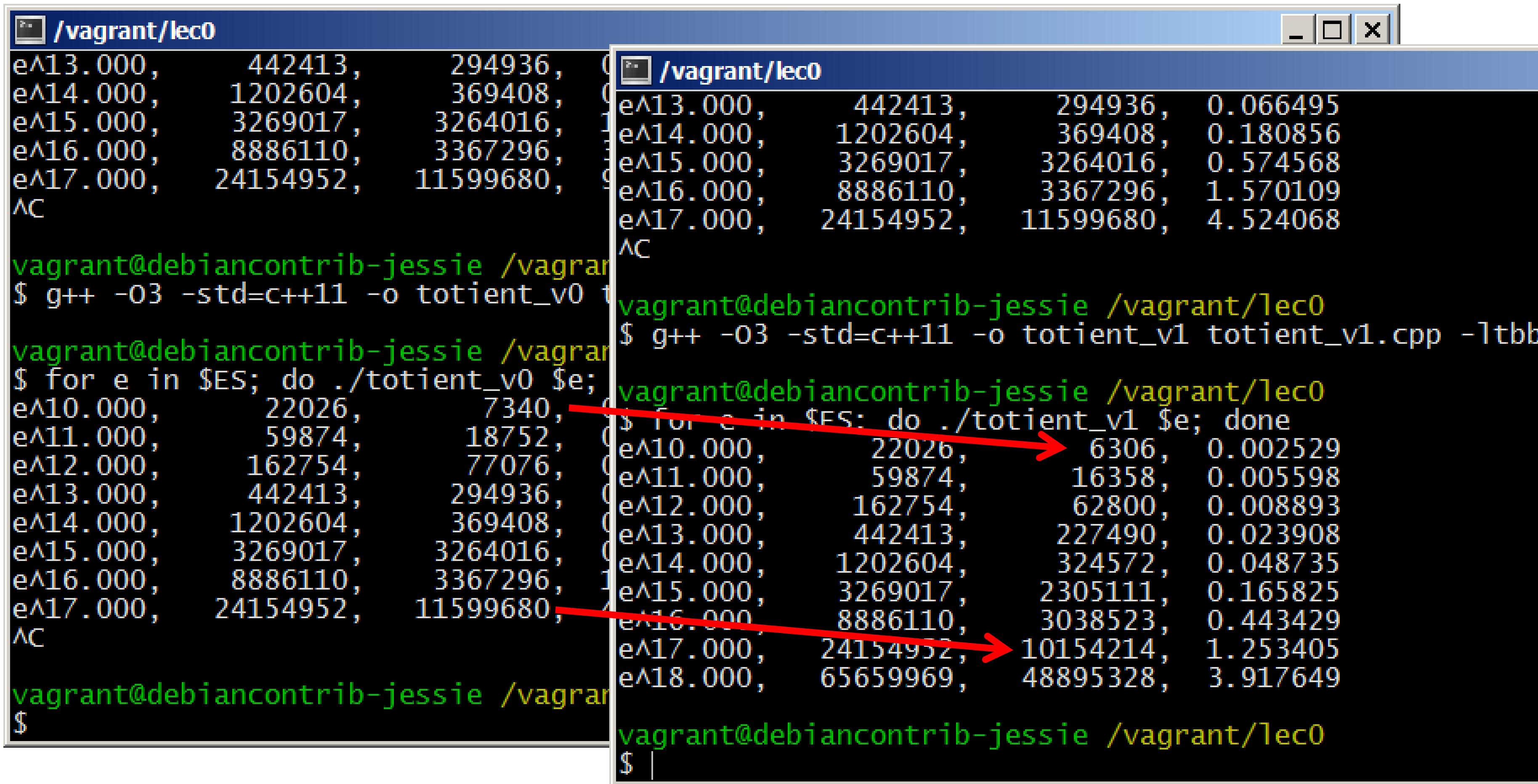
```
/vagrant/lec0
e^13.000,     442413,      294936,   0.066495
e^14.000,    1202604,      369408,   0.180856
e^15.000,    3269017,     3264016,   0.574568
e^16.000,    8886110,     3367296,   1.570109
e^17.000,   24154952,    11599680,   4.524068
^C

vagrant@debiancontrib-jessie /vagrant/lec0
$ g++ -O3 -std=c++11 -o totient_v1 totient_v1.cpp -ltbb

vagrant@debiancontrib-jessie /vagrant/lec0
$ for e in $ES; do ./totient_v1 $e; done
e^10.000,      22026,        6306,   0.002529
e^11.000,      59874,       16358,   0.005598
e^12.000,     162754,       62800,   0.008893
e^13.000,     442413,      227490,   0.023908
e^14.000,    1202604,      324572,   0.048735
e^15.000,    3269017,     2305111,   0.165825
e^16.000,    8886110,     3038523,   0.443429
e^17.000,   24154952,    10154214,   1.253405
e^18.000,   65659969,    48895328,   3.917649

vagrant@debiancontrib-jessie /vagrant/lec0
$ |
```

Faster, but...



10
1
0.1
0.01
0.001
Time (s)
10000
100000
1000000
10000000
n
v0
v0-release
v1-par-wrong

Faster but wrong ☹



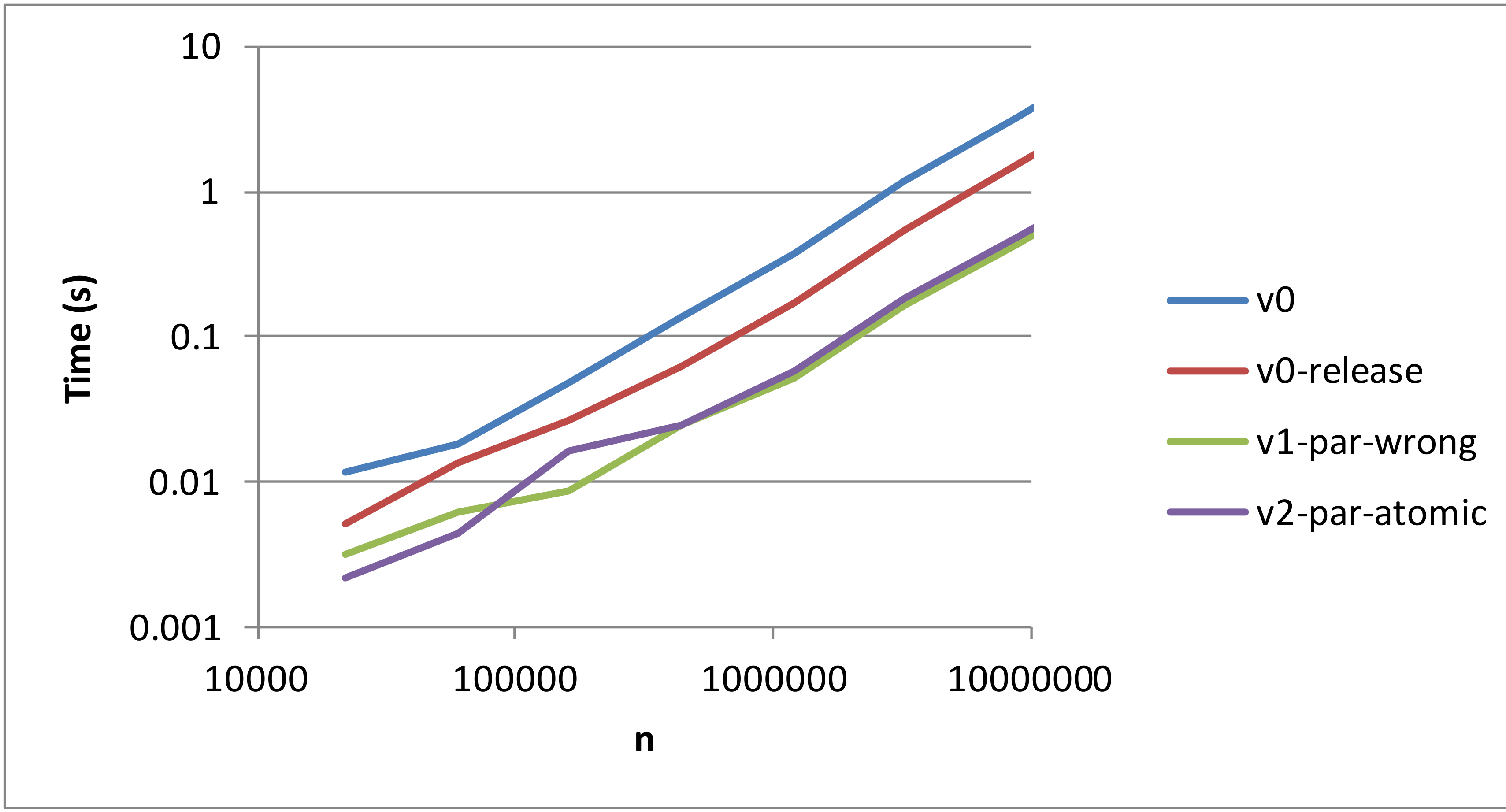
Unsafe use of shared variable

```
#include "tbb/parallel_for.h"

uint64_t totient_v1(uint64_t n)
{
    uint64_t count=0;

    //for(uint64_t i=1; i<=n; i++){
    tbb::parallel_for(uint64_t(1), (n+1), [&](uint64_t i){
        if(gcd(i,n)==1){
            count = count + 1;
        }
    });

    return count;
}
```

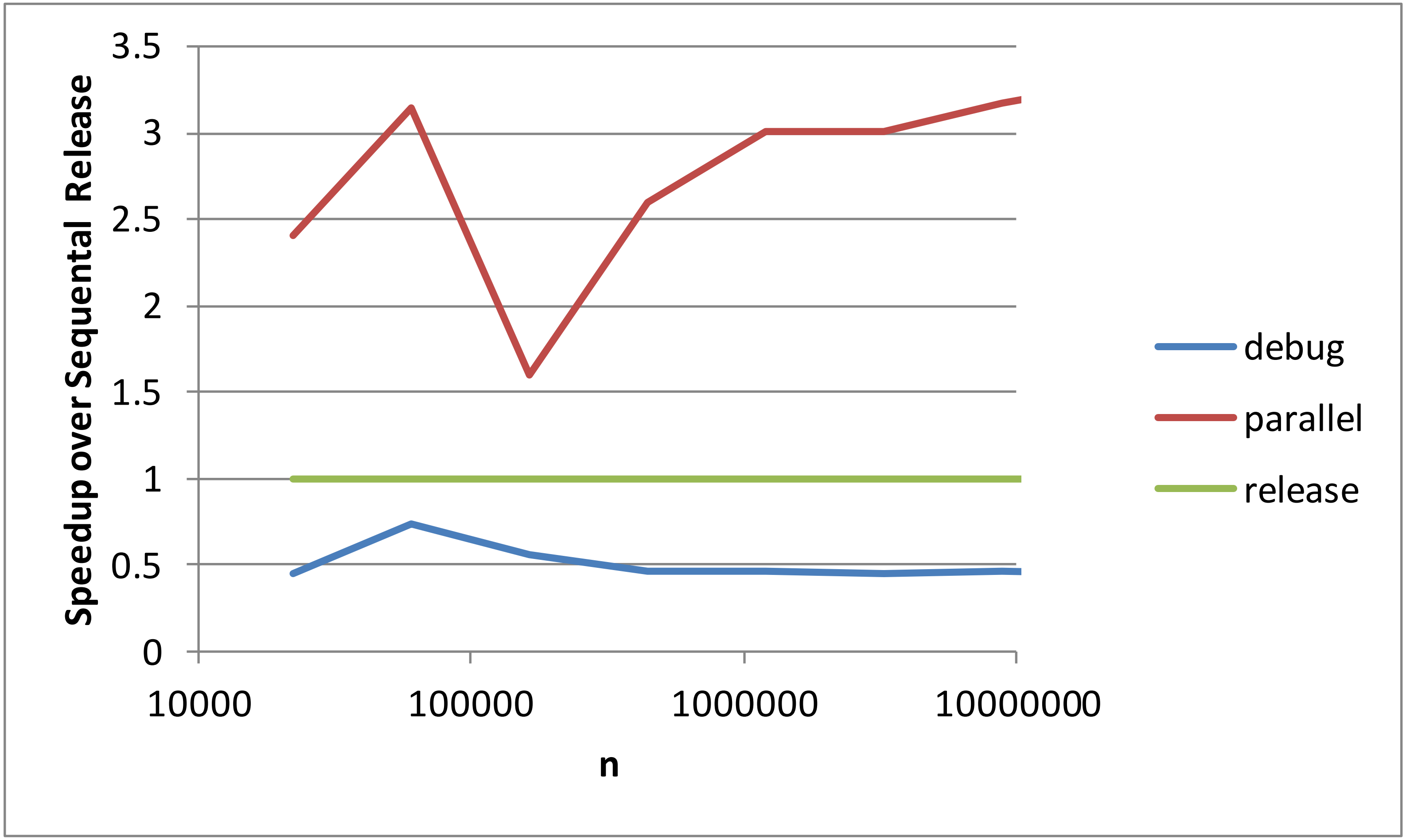
Make it atomic

```
uint64_t totient_v2(uint64_t n)
{
    std::atomic<uint64_t> count;

    //for(uint64_t i=1; i<=n; i++){
    tbb::parallel_for(uint64_t(1), (n+1), [&](uint64_t i){
        if(gcd(i,n)==1){
            count += 1;
        }
    });

    return count;
}
```

Fast and correct



Speedup (4 CPU machine)



Initial Lessons

- Speeding up loops *can* be easy
- Need to watch out for shared variables
- The speedup in P cores is less than P

Reminder Action

- If you want to take this course then:
  1. Get a github account
  2. Send me an email:
     - Subject: “[HPCE-github-request]”
     - Body: github id + your Imperial ***login*** (the short one)