/INFOMOV/

Optimization & Vectorization

J. Bikker - Sep-Nov 2016 - Lecture 1: “Introduction”

Welcome!

Today’s Agenda:

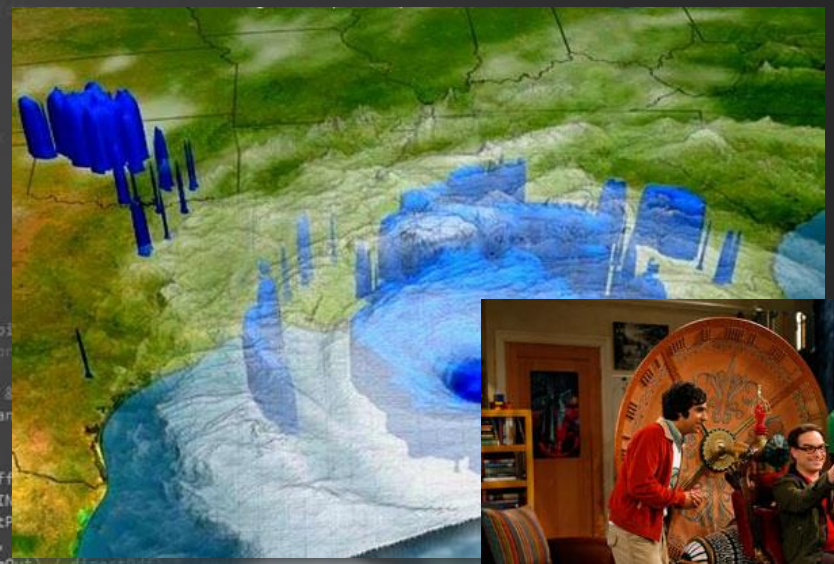
- Introduction
- Course Formalities
- High Level Overview
- Profiling

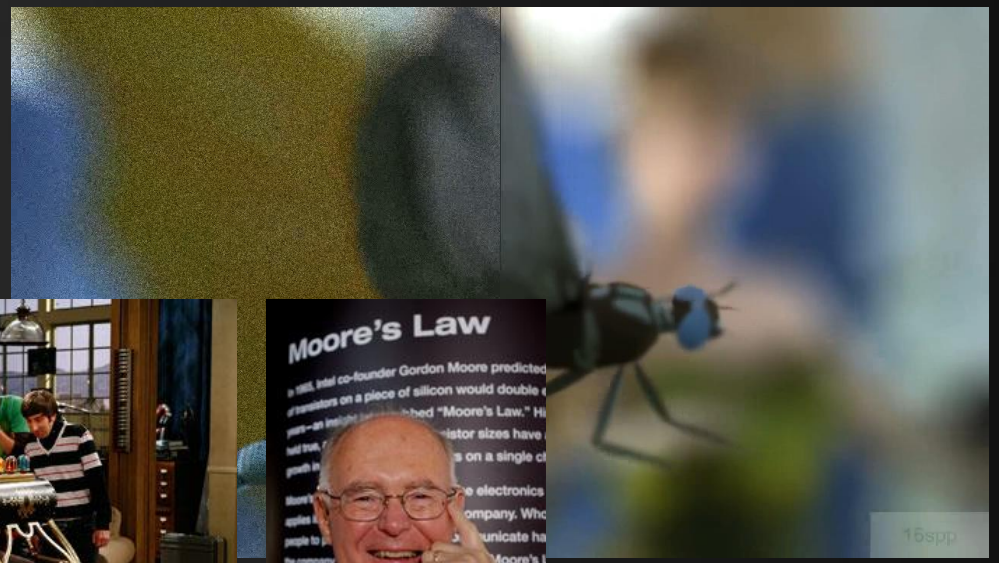
Introduction

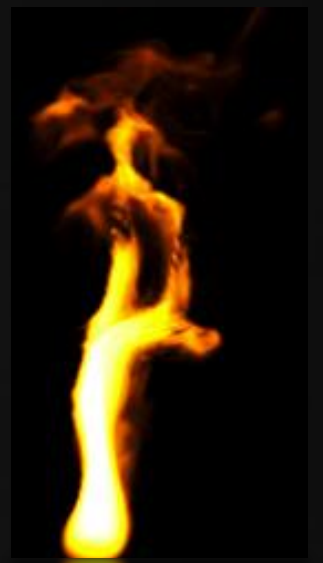
Why?

Some problems require the supercomputer of the future.

- Anything that depends on Moore’s Law and time to become feasible.







Introduction

Why?

Games want to raise the bar.

- More, better, faster. Also: be scalable.

Introduction

Why?

Some software needs to run on pretty weak hardware.

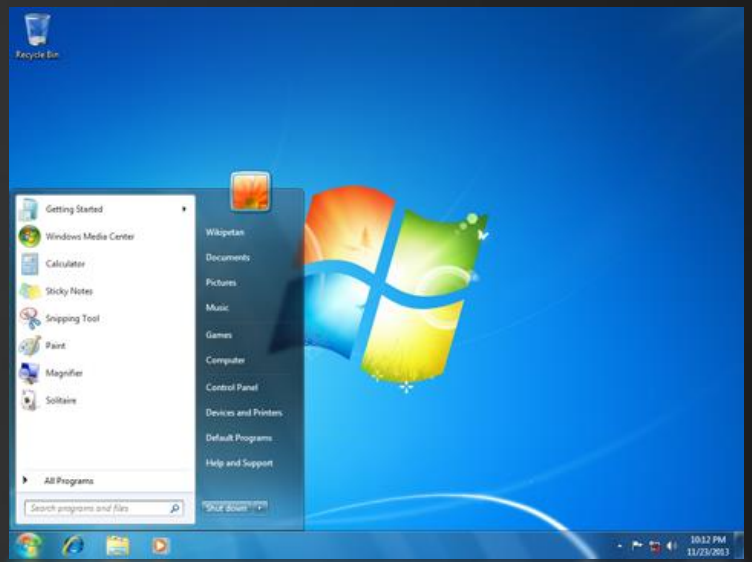
- Limited CPU, limited RAM (limited controls).

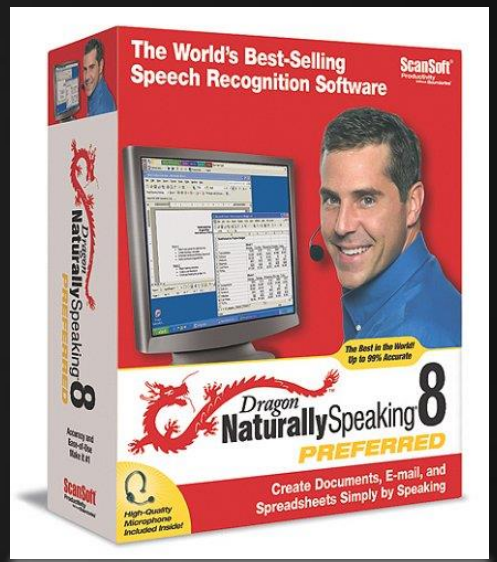
Introduction

Why?

Some software should not use 90% of your CPU.

- Leave room for other applications, be invisible.





Introduction

Why?

Sometimes the cheapest / lowest power CPU is the best.

- What is the lowest end CPU this will still run on? Can we go lower?

Introduction

Why?

Some things are done so frequently, they must be efficient.

- Memory manager
- Garbage collector
- JIT compiler
- Compilers in general
- Image processing in Photoshop
- OS startup / resume
- …
- …

Introduction

What is optimization?

Part of it is:

- INFOB3CC - Concurrency
- INFONW - Computerarchitectuur en netwerken
- INFOB3TC - Talen en compilers

And of course: any course that deals with improving existing algorithms.

Specific purpose of INFOMOV:

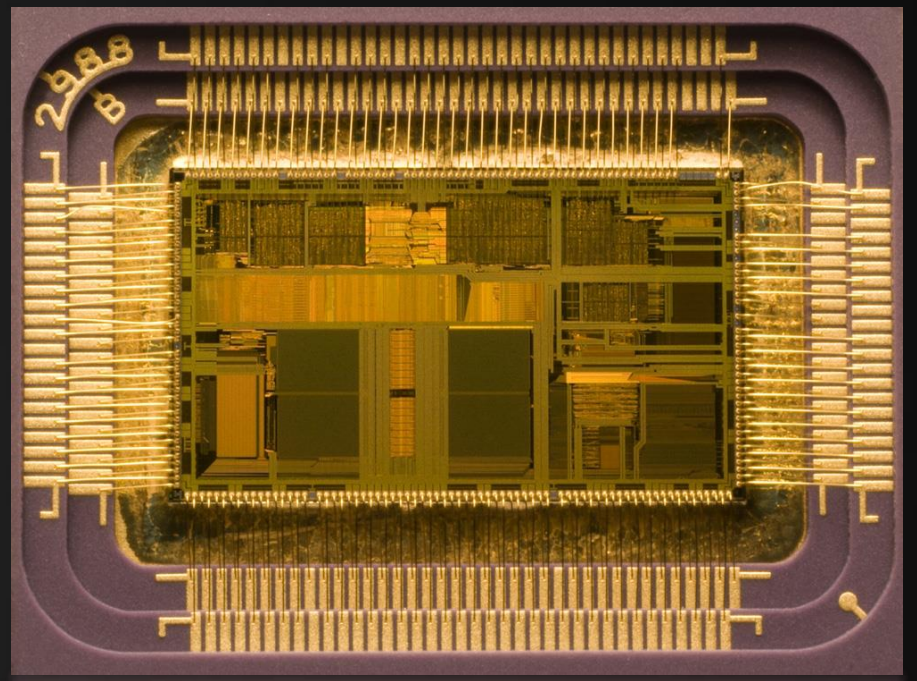
- To gain understanding of performance aspects of the hardware we use;
- To gain an intuition for what affects performance;
- To learn to apply a structured process to improve performance.

Introduction

What is optimization?

Working with the hardware: *think like a CPU*

- Instruction pipelines
- Latencies
- Dependencies
- Bandwidth
- Cycles
- Floating point versus integer
- SIMD



Introduction

What is optimization?

Work smarter, not harder: algorithm scalability

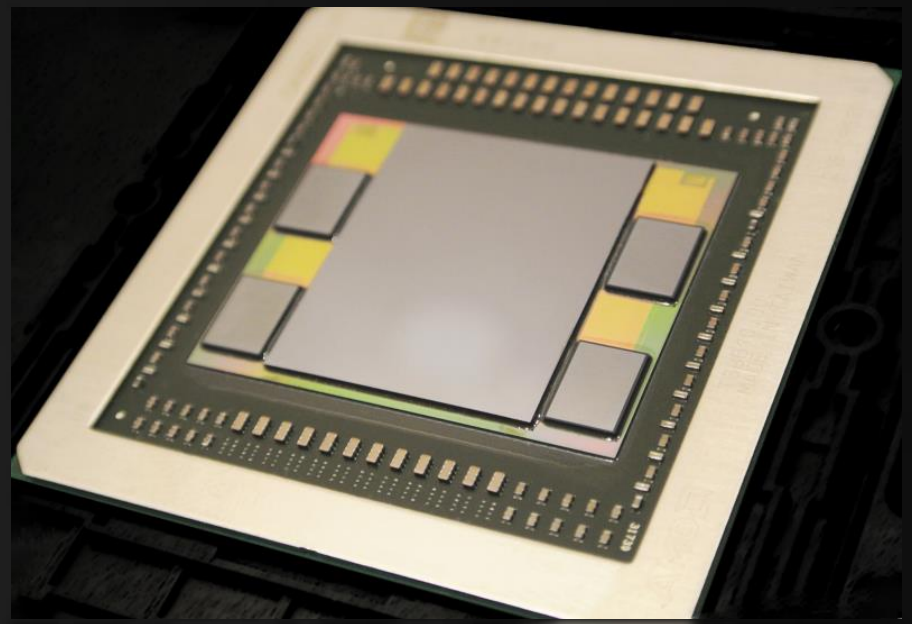
- Big O
- Research: not reinventing the wheel
- Data characteristics & algorithm choice
- STL: Trust No One
- As accurate as necessary (but not more)
- Balancing accuracy, speed and memory

Introduction

What is optimization?

Memory hierarchy: caches

- Cache architecture
- Cache lines
- Hits, misses and collisions
- Eviction policies
- Prefetching
- Cache-oblivious
- Data-centric programming

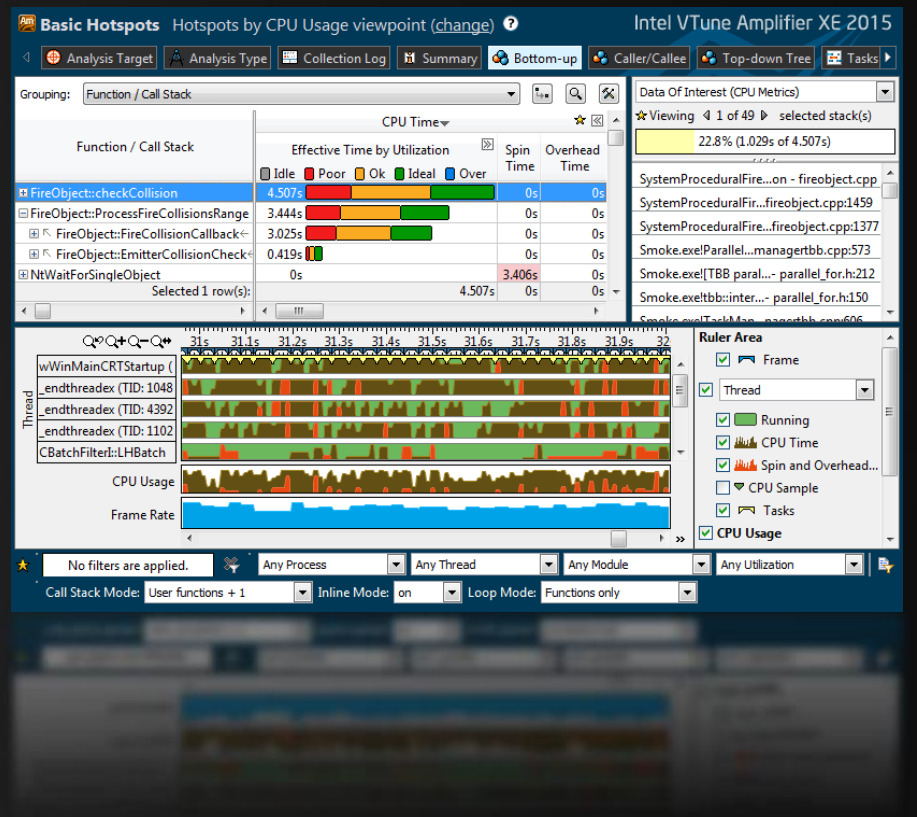


Introduction

What is optimization?

Don’t assume, measure

- Profilers
- Interpreting profiling data
- Instrumentation
- Bottlenecks
- Steering optimization effort



Introduction

What is optimization? – Project Management

Keeping code maintainable

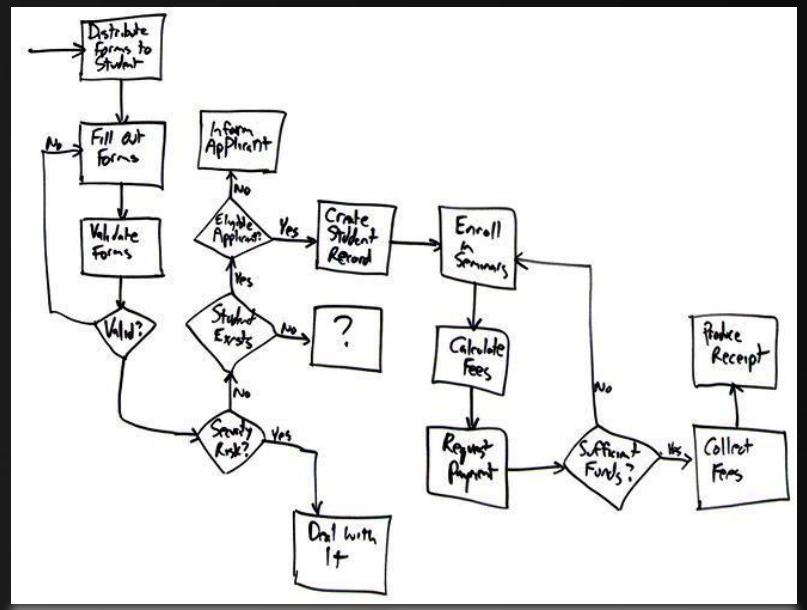
- Pareto principle / 80-20 rule: roughly 80% of the effects are caused by 20% of the causes.
- 1% of the code takes 99% of the time.

The curse of premature optimization

- Optimization, rule 1: “Don’t do it”.
- Rule 2 (for experts only!), “Don’t do it yet”.

Optimization as a deliberate process

- Get predictable gains using a consistent approach.

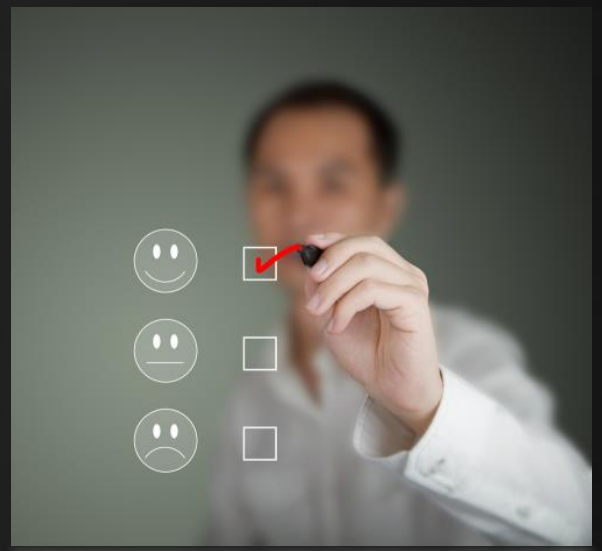


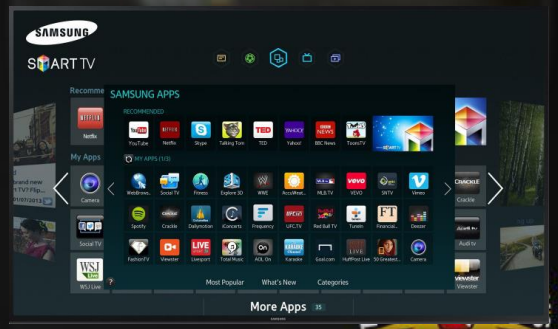
Introduction

What is optimization? – Project Management

“Perceived Performance”

1. Wait for user input
2. Respond to user input *as quickly as possible*
3. Execute requested operation.





Introduction

At the end of this course:

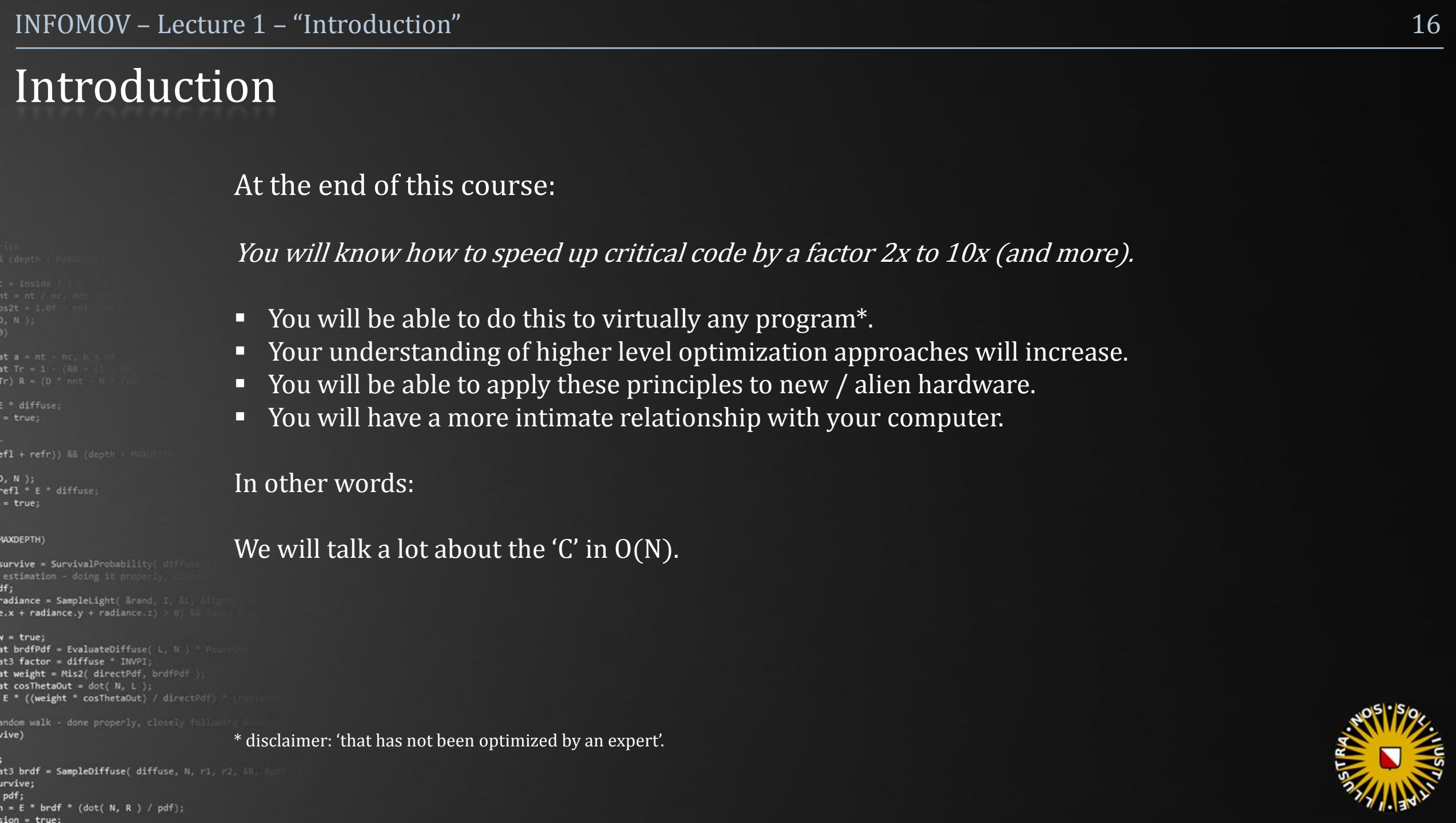
You will know how to speed up critical code by a factor 2x to 10x (and more).

- You will be able to do this to virtually any program*.
- Your understanding of higher level optimization approaches will increase.
- You will be able to apply these principles to new / alien hardware.
- You will have a more intimate relationship with your computer.

In other words:

We will talk a lot about the ‘C’ in O(N).

* disclaimer: ‘that has not been optimized by an expert’.



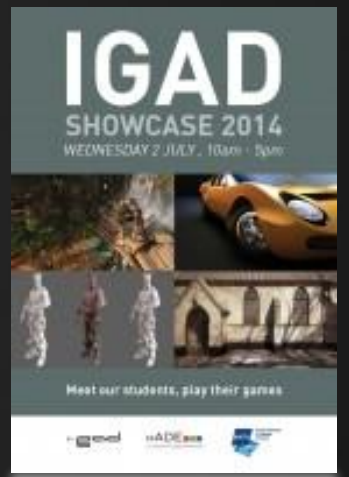
Today’s Agenda:

- Introduction
- Course Formalities
- High Level Overview
- Profiling

Formalities

Lecturer

Dr. Jacco Bikker
j.bikker@uu.nl
Room 4.24 BBG



Formalities

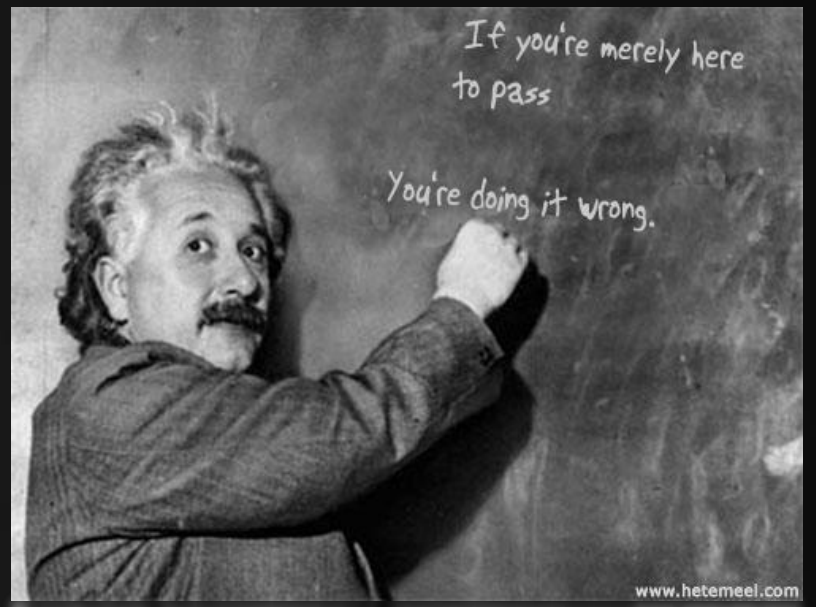
Course Layout

8 weeks + exam week:

- 2 lectures per week (for exceptions: see website)

Assessment:

- 2 assignments (25% each, in groups of 2 or 3 students);
- 1 final assignment (50%, in groups of 1-2 students);
- 1 final theory exam.



Formalities

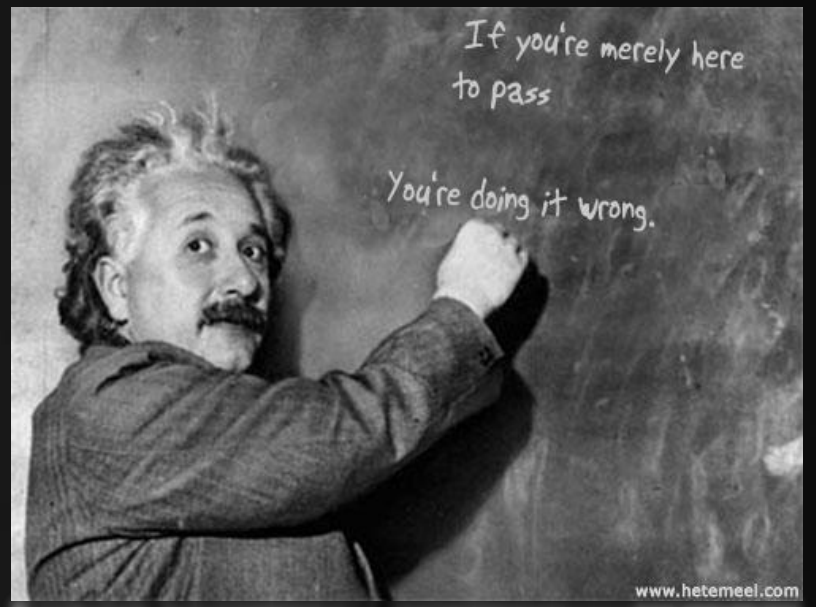
Prerequisites

C++
English

Hardware / software

You’ll need access to a computer with a CPU that supports SSE2 and OpenCL.
Obtaining VTune (Intel CPU) or CodeXL (AMD CPU) is beneficial (VTune is free to try for 30 days).
Visual Studio 2013 or 2015 will be needed.

Other tools will be free.



Formalities

Literature

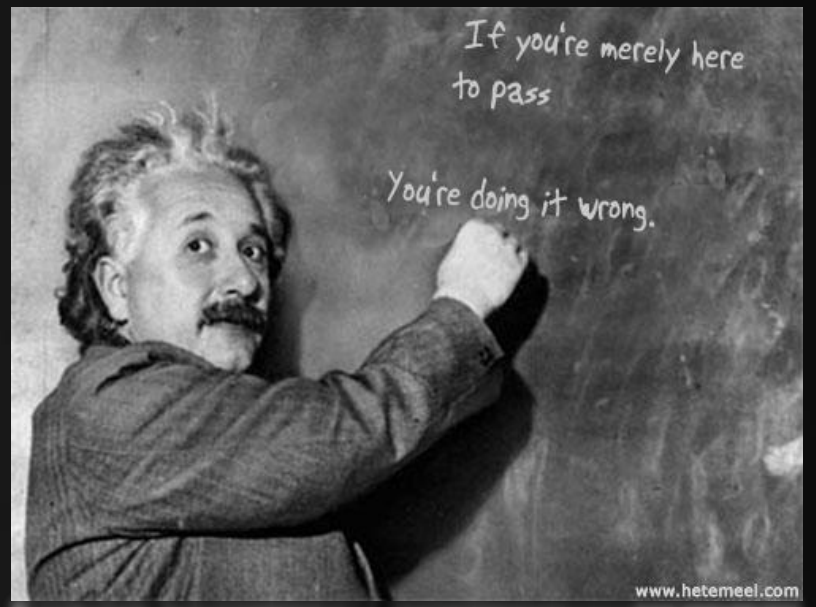
No book!
But that doesn’t mean you won’t be reading.

Main documents:

Agner Fog, 2014, “Optimizing C++”
(also see his website: http://agner.org)

Ulrich Drepper, 2007, “What Every Programmer Should Know About Memory”

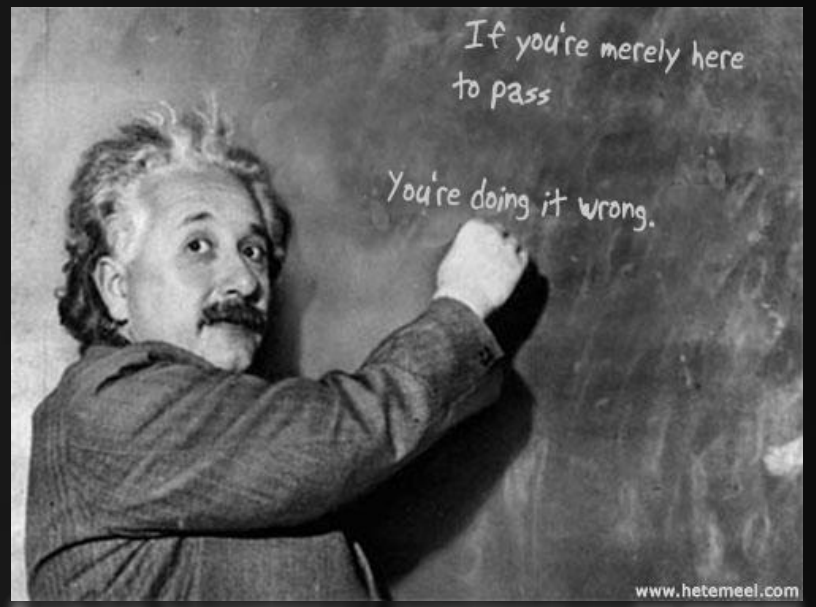
You are encouraged to do research into specific topics of interest yourself, and to report on this in class.



Formalities

Audience

Any computer science student
(with a slight bias towards games)



Make sure <u>you</u> get as much as possible out of this course. This automatically includes a free pass.

Today’s Agenda:

- Introduction
- Course Formalities
- High Level Overview
- Profiling

Overview

Consistent Approach

(0.) Determine optimization requirements
1. Profile: determine hotspots
2. Analyze hotspots: determine scalability
3. Apply high level optimizations to hotspots
4. Profile again.
5. Parallelize
6. Use GPGPU
7. Profile again.
8. Apply low level optimizations to hotspots
9. Repeat steps 7 and 8 until time runs out
10. Report.

Overview

Consistent Approach

(0.) Determine optimization requirements

- Target hardware (or range of hardware)
- Target performance
- Time available for optimization
- Constraints related to maintainability / portability
- …

1. Profile: determine hotspots
2. Analyze hotspots: determine scalability
3. Apply high level optimizations to hotspots
4. Profile again.
5. Parallelize
6. Use GPGPU
7. Profile again.
8. Apply low level optimizations to hotspots
9. Repeat steps 7 and 8 until time runs out
10. Report.

From here on, we will assume that:

- the code is ‘done’ (feature complete);
- a speed improvement is desired;
- we have a finite amount of time for this.

Overview

Consistent Approach

(0.) Determine optimization requirements
1. Profile: determine hotspots
2. Analyze hotspots: determine scalability
3. Apply high level optimizations to hotspots
4. Profile again.
5. Parallelize
6. Use GPGPU
7. Profile again.
8. Apply low level optimizations to hotspots
9. Repeat steps 7 and 8 until time runs out
10. Report.

Overview

Consistent Approach

(0.) Determine optimization requirements
1. Profile: determine hotspots
2. Analyze hotspots: determine scalability
3. Apply high level optimizations to hotspots
4. Profile again.
5. Parallelize
6. Use GPGPU
7. Profile again.
8. Apply low level optimizations to hotspots
   - caching, data-centric programming,
   - removing superfluous functionality and precision,
   - aligning data to cache lines, vectorization,
   - checking compiler output, fixed point arithmetic,
   - …
9. Repeat steps 7 and 8 until time runs out
10. Report.

Overview

Consistent Approach

(0.) Determine optimization requirements
1. Profile: determine hotspots
2. Analyze hotspots: determine scalability
3. Apply high level optimizations to hotspots
4. Profile again.
5. Parallelize
6. Use GPGPU
7. Profile again.
8. Apply low level optimizations to hotspots
9. Repeat steps 7 and 8 until time runs out
10. Report.

Profiling

High Level

Basic Low Level

Cache & Memory

Data-centric

Compilers

Fixed-point Arithmetic

CPU architecture

SIMD

GPGPU

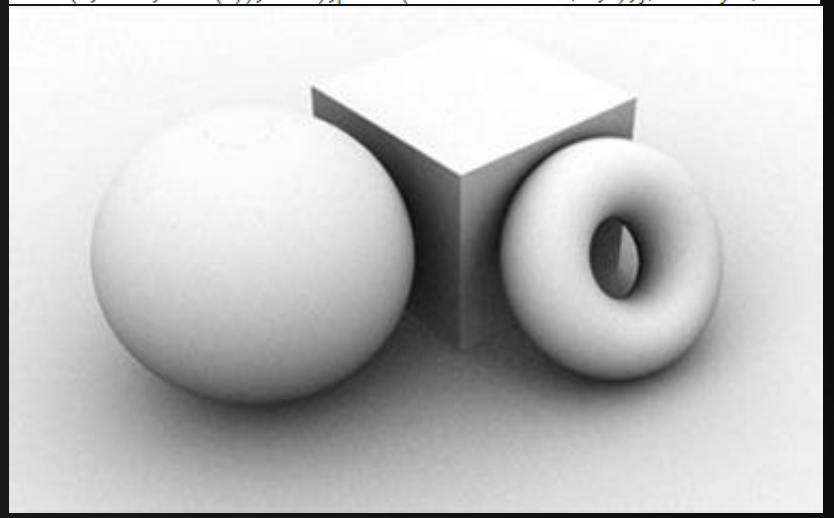
Overview

Assembler

In this course, we will <u>not</u> write assembler:

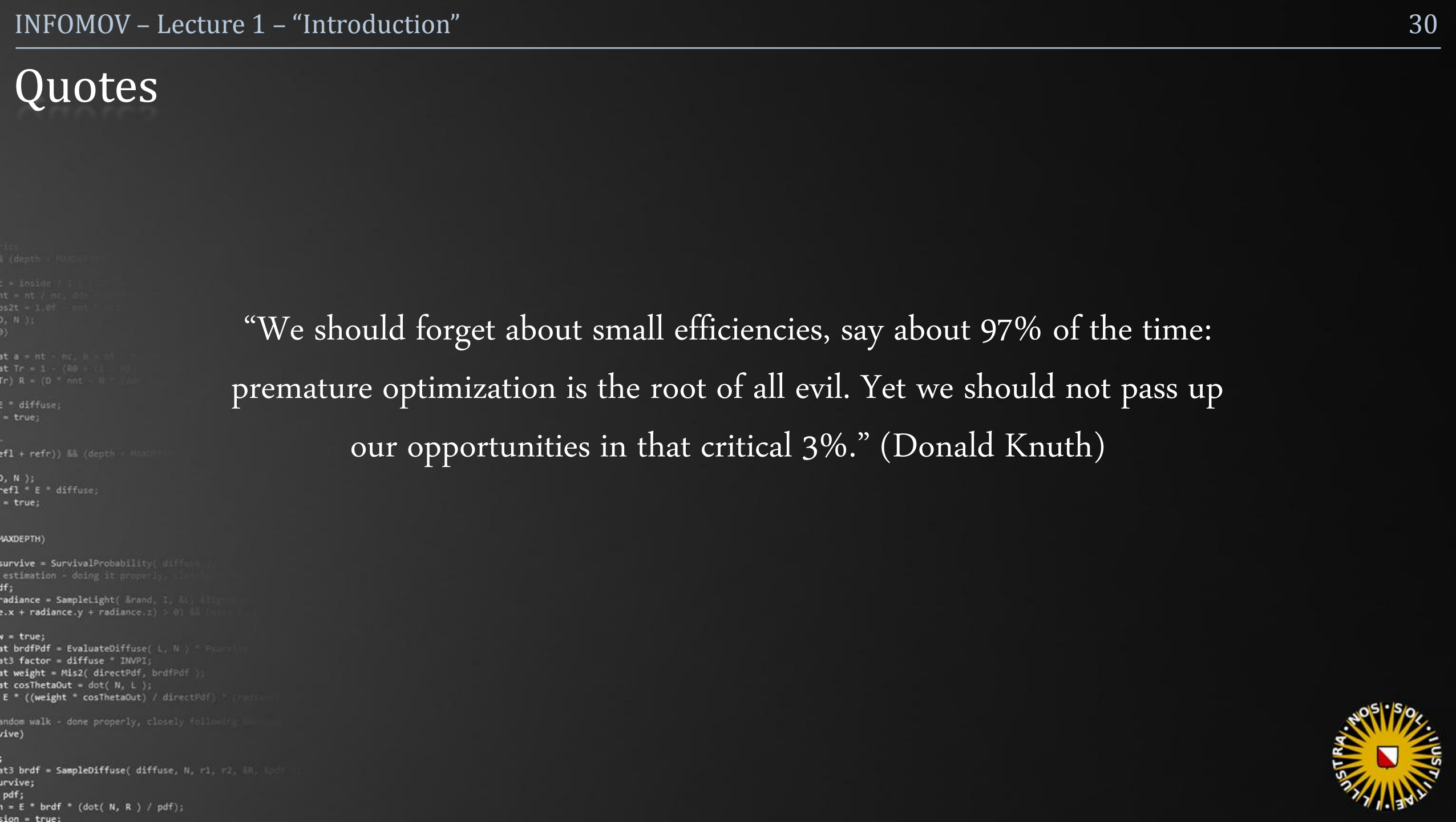
- It takes a pro to outperform the compiler
- You will be fighting the compiler
- You will have to redo the optimization for every target processor
- Maintainability will be zero.

```
typedef struct{double x,y,z}vec;vec U,black,amb={.02,.02,.02};struct sphere{
vec cen,color;double rad,kd,ks,kt,kl,ir}*s,*best,sph[]={0.,6.,.5,1.,1.,1.,.9,
.05,.2,.85,0.,1.7,-1.,8.,-.5,1.,.5,.2,1.,.7,.3,0.,.05,1.2,1.,8.,-.5,.1,.8,.8,
1.,.3,.7,0.,0.,1.2,3.,-6.,15.,1.,.8,1.,7.,0.,0.,0.,.6,1.5,-3.,-3.,12.,.8,1.,
1.,5.,0.,0.,0.,.5,1.5,};yx;double u,b,tmin,sqrt(),tan();double vdot(A,B)vec A
,B;{return A.x*B.x+A.y*B.y+A.z*B.z;}vec vcomb(a,A,B)double a;vec A,B;{B.x+=a*
A.x;B.y+=a*A.y;B.z+=a*A.z;return B;}vec vunit(A)vec A;{return vcomb(1./sqrt(
vdot(A,A)),A,black);}struct sphere*intersect(P,D)vec P,D;{best=0;tmin=1e30;s=
sph+5;while(s-->sph)b=vdot(D,U=vcomb(-1.,P,s->cen)),u=b*b-vdot(U,U)+s->rad*s
->rad,u=u>0?sqrt(u):1e31,u=b-u>1e-7?b-u:b+u,tmin=u>=1e-7&&u<tmin?best=s,u:
tmin;return best;}vec trace(level,P,D)vec P,D;{double d,eta,e;vec N,color;
struct sphere*s,*l;if(!level--)return black;if(s=intersect(P,D));else return
amb;color=amb;eta=s->ir;d= -vdot(D,N=vunit(vcomb(-1.,P=vcomb(tmin,D,P),s->cen
)));if(d<0)N=vcomb(-1.,N,black),eta=1/eta,d= -d;l=sph+5;while(l-->sph)if((e=l
->kl*vdot(N,U=vunit(vcomb(-1.,P,l->cen))))>0&&intersect(P,U)==l)color=vcomb(e
,l->color,color);U=s->color;color.x*=U.x;color.y*=U.y;color.z*=U.z;e=1-eta*
eta*(1-d*d);return vcomb(s->kt,e>0?trace(level,P,vcomb(eta,D,vcomb(eta*d-sqrt
(e),N,black))):black,vcomb(s->ks,trace(level,P,vcomb(2*d,N,D)),vcomb(s->kd,
color,vcomb(s->kl,U,black))));}main(){printf("%d %d\n",32,32);while(yx<32*32)
U.x=yx%32-32/2,U.z=32/2-yx++/32,U.y=32/2/tan(25/114.5915590261),U=vcomb(255.,
trace(3,black,vunit(U)),black),printf("%.0f %.0f %.0f\n",U);}/*minray!*/
```



Quotes

“We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil. Yet we should not pass up our opportunities in that critical 3%.” (Donald Knuth)



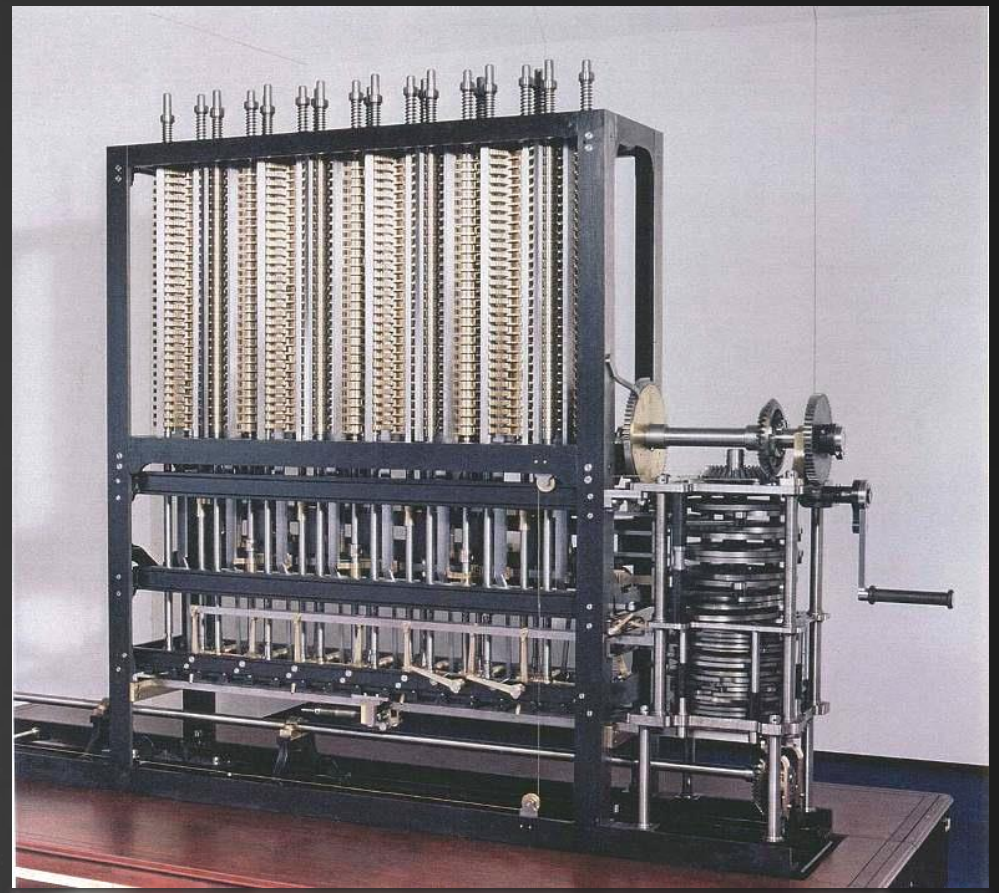
Quotes

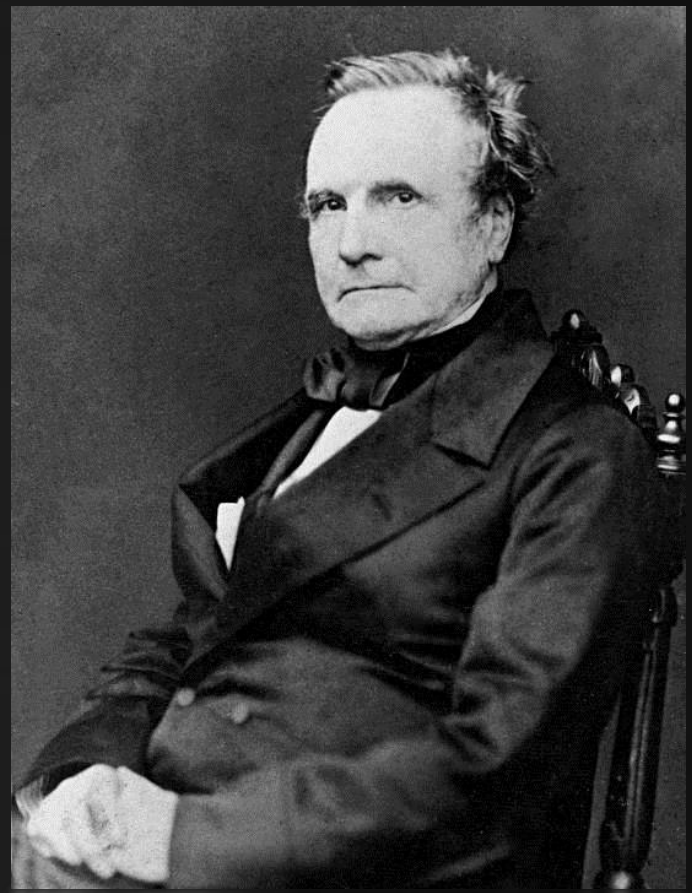
“A significant improvement in performance can often be achieved by solving only the actual problem and removing extraneous functionality.” (Wikipedia)

Quotes

“More computing sins are committed in the name of efficiency (without necessarily achieving it) than for any other single reason – including blind stupidity.” (W.A. Wulff)

Quotes





Quotes

“Dear Charles,

In almost every computation a great variety of arrangements for the succession of the processes is possible, and various considerations must influence the selection amongst them (...).

One essential object is to choose that arrangement which shall tend to reduce to a minimum the time necessary for completing the calculation.

Therefore, one should attend INFOMOV and learn from it.

Love, Ada.”

Today’s Agenda:

- Introduction
- Course Formalities
- High Level Overview
- Profiling

time ran out

/INFOMOV/

END of “Introduction”

next lecture: “Low Level”