

▼ Department of Information Technology

Uppsala University / Information Technology /

Listen

High Performance Computing and Programming, VT12, 1TD351

Welcome to the course "High Performance Computing and Programming"!

Teachers

[Marcus Holm](#) (Room 2404)

[Elias Rudberg](#) (Room 2440)

Guest lecturers

[Sverker Holmgren](#)

[Carl Nettelblad](#)

[Stefan Engblom](#)

Post boxes are on MIC level 2, house 2

Literature

- *Writing Scientific Software - A Guide to Good Style* by Sueley Oliveira and David Stewart, Cambridge University Press, 2006, ISBN 0-521-67595-2

For further reading see [More literature](#) and [Links](#) in the menu on the left. Please let us know if any link is not current or if you have any good reading suggestions yourself.

Examination

Compulsory assignments, laborations, and exam. Assignments and laborations can be done in group and are therefore graded on a pass/fail basis.

Course Details

We recommend that you log in to Studentportalen ([\[1\]](#)) and navigate to the course there, to find lecture notes and hand in your assignments.

Lab files for lab 1: [lab1.tgz](#)

Lab files for lab 2: [lab2.tgz](#)

Lab files for lab 3: [lab3.tgz](#)

Lab files for lab 4: [lab4.tgz](#) [lab4.pdf](#) 

Assignment files for assignment 1: [Inlupp_1_student.tgz](#)

Assignment files for assignment 2: [Inlupp_2_student.zip](#) 

Lecture slides for lecture 1: [Lect1_intro_2012.pdf](#) 

Lecture slides for lecture 2: [Lect2_C_2012.pdf](#) 

Lecture slides for lecture 3: [Lect3_Debugging_2012.pdf](#) 

Lecture slides for lecture 4: [Lect4_Perf_2012.pdf](#) 

Lecture slides for lecture 5: [Lect5_Optimization_2012.pdf](#) 

Lecture slides for lecture 6: [Lect6_Optimization_2012.pdf](#) 

Lecture slides for lecture 7: [Lect7_Optimization_2012.pdf](#) 

Lecture slides for lecture 8: [Lect8_Optimization_2012.pdf](#) 

Lecture slides for lecture 9: [Lect9_Multicore_2012.pdf](#) 

Lecture slides for lecture 10: [Lect10_FloatingPoint_2012.pdf](#) 

Lecture slides for lecture 11: [Lect11_HPCLibs_2012.pdf](#) 

Lecture slides for lecture 12: [Lect12_ReviewandFuture_2012.pdf](#) 

Some information will be replicated on this page, see the menu on the left or below.

- Course Presentation
- Lectures and Schedule
- More literature
- Links

Updated 2012-05-23 13:44:23 by [Marcus Holm](#).

CONTACT US

[Contact persons/functions](#)

[Staff](#)

VISIT US

[Adresses](#)

[Map](#)

SHORTCUTS

[Evacuation Plan](#)

[Emergency Contacts](#)

[Report an Error](#)

[Edit this page](#)

Uppsala universitet använder kakor (cookies) för att webbplatsen ska fungera bra för dig. [Läs mer om kakor.](#) [OK](#)



© 2020 Uppsala University | Phone: +46 18 471 00 00 | P.O. Box 337, SE-751 05 Uppsala, Sweden

Registration number: 202100-2932 | VAT number: SE202100293201 | [Registrar](#) | [About this website](#) | Editor: [Anneli Folkesson](#).

[GO TO TOP](#)





Svensk startsida

Log in

Listen



Uppsala University campus guide

SWEDEN'S FIRST UNIVERSITY

Study at Uppsala University

Join us at Uppsala University

NEWS



Professor of AI: "This technology is here
to stay" 29 Sep 2020

Atom-Billiards with X-Rays: a new
Approach to look inside of Molecules
29 Sep 2020

Soluble ligands as drug targets

21 Sep 2020

New high-speed test shows
how antibiotics combine to kill bacteria

[More news →](#)

EVENTS



30 SEP

13:15

Amerikanska rasrelationer och Sverige

5 OCT

15:15

Do it now! Stop procrastinating

5 OCT

16:15

Death at the End of the Rainbow:
Rethinking Queer Kinship, Rituals of
Remembrance and the Finnish Culture of
Death

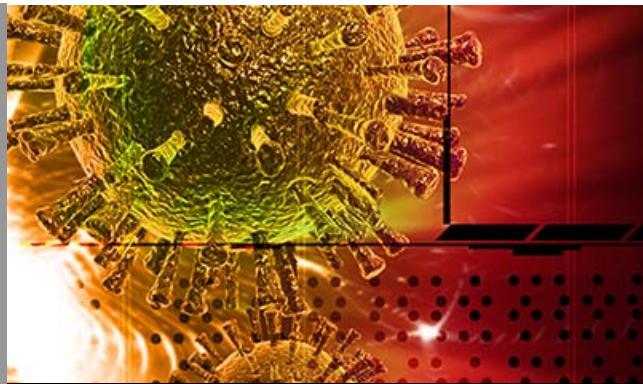
6 OCT

14:30

Brief seminars and group discussions: 4
topics

[More events →](#)

CORONAVIRUS



Latest updates on Covid-19.

Covid-19: Recommendations for employees →

Covid-19: Recommendations for students →

VICE-CHANCELLOR'S BLOG



28 sept: Challenges and opportunities for the University magnified at Campus Gotland



CONTACT THE UNIVERSITY

Telephone: [+46 18 471 00 00](tel:+46184710000)

[Contact the University](#)

[Find researchers & staff](#)

FOLLOW UPPSALA UNIVERSITY ON



VISIT THE UNIVERSITY

[Departments & units](#)

[Campuses](#)

[Museums & gardens](#)

[Map of Uppsala University](#)

QUICK LINKS

[Web shop](#)

[News & media services](#)

[Library](#)

[Jobs & vacancies](#)

[University management](#)

[Support Uppsala University](#)

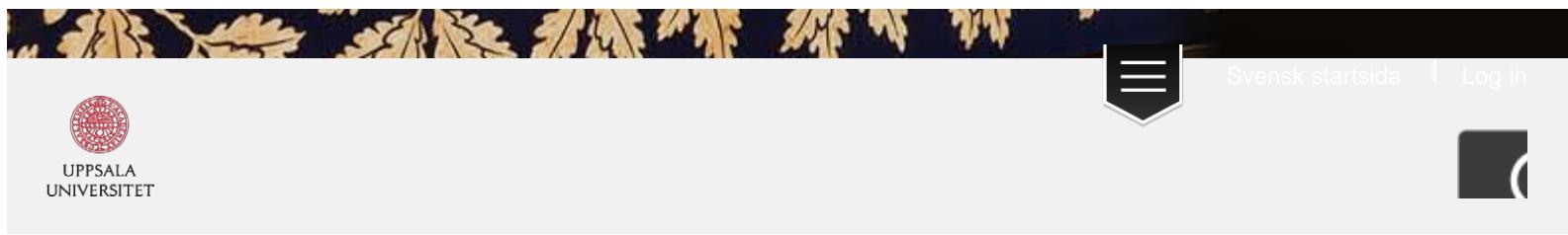
[International Faculty & Staff Services](#)

[Medarbetarportalen – Employee portal](#)

[Student portal](#)



© Uppsala University | Tel.: +46 18 471 00 00 | P.O. Box 256, SE-751 05 Uppsala, SWEDEN
Registration number: 202100-2932 | VAT number: SE202100293201 | PIC: 999985029 | [Registrar](#) |
[About this website](#) | [Privacy policy](#) | Editor: [David Naylor](#)



▼ Department of Information Technology

Uppsala University / Information Technology

Denna sida på svenska

Listen

Programmes and courses

Follow us on Facebook

Contact & Staff

Open Positions

DEPARTMENT OF INFORMATION TECHNOLOGY

Teaching and research focusing on advanced design and use of computer systems in science and technology.

NEWS



Communicator

2020-09-24

Researcher

2020-09-24

Researcher position in Scientific
Computing, with a focus on
computational biomechanics

2020-09-24

Postdoctoral fellow in Machine Learning
with a focus on deep learning and

More news →

CALENDAR



02 OCT Disputation | PhD defence:
kl 13:00 Rethinking Dynamic
Instruction Scheduling and
Retirement for Efficient
Microarchitectures

09 OCT Seminar ONLINE:

INCREASE YOUR SKILLS
ONLINE!



Are you an engineer or natural scientist and temporarily laid-off? Or do you have the time and interest anyway?

Find out what we can offer! →

OUR EDUCATION

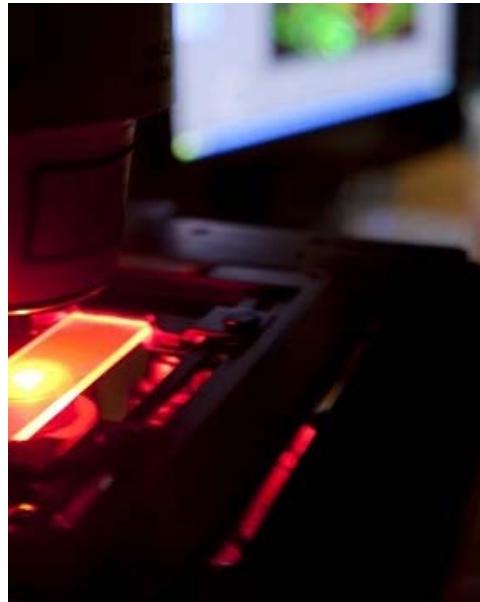


Do you want to understand how information technology works, be a part of the IT-development and create the technology of tomorrow? At the Department of Information

Technology you study in an international environment with teachers who are leading researchers within their fields. We cover a wide range of issues, from construction of computer systems, via programming of computers, storage and handling of data, to information retrieval and methods for applying computers in a variety of contexts. [Find out more about our courses and international master's programmes.](#)

OUR RESEARCH

Our research is grouped into five themes: Computer Systems, Computing Science, Scientific Computing, Systems & Control and Visual Information & Interaction. Each of these themes is in itself a major subject area, where we both conduct basic research and have projects with links to applications in for example Engineering, Biology, Medicine, Economy and Psychology. The broad scientific foundation is reflected in a large number of collaborations across academic disciplines as well as with industry and the public sector. [Find out more about our research.](#)



COLLABORATION

We collaborate in different ways
with the society around us.



ABOUT US

Read about the department and
our activities.



Updated 2020-05-06 16:54:26 by [Lina von Sydow](#).

CONTACT US

[Contact persons/functions](#)
[Staff](#)

VISIT US

[Adresses](#)

[Map](#)

SHORTCUTS

[Evacuation Plan](#)

[Emergency Contacts](#)

[Report an Error](#)

[Edit this page](#)

Uppsala universitet använder kakor (cookies) för att webbplatsen ska fungera bra för dig. [Läs mer om kakor.](#) [OK](#)

© 2020 Uppsala University | Phone: +46 18 471 00 00 | P.O. Box 337, SE-751 05 Uppsala, Sweden

Registration number: 202100-2932 | VAT number: SE202100293201 | [Registrar](#) | [About this website](#) | Editor: [Lina von Sydow](#).

[GO TO TOP](#)



▼ Department of Information Technology

Uppsala University / Information Technology / Search for staff /

 Denna sida på svenska Listen

Marcus Lundberg



Coordinator at [Department of Information Technology, Uppsala Multidisciplinary Centre for Advanced Computational Science](#)

vCard 

Email:

marcus.lundberg@uppmx.uu.se

Telephone:

[+4618-471 6201](tel:+46184716201)

Visiting address:

Room POL 1442 ITC, Lägerhyddsvägen 2, hus 1,2 & 4
752 37 Uppsala

Postal address:

Box 337
751 05 Uppsala

Coordinator at [Department of Cell and Molecular Biology, NBIS - National Bioinformatics Infrastructure Sweden](#)

vCard 

Visiting address:

Husargatan 3
75237 Uppsala

Postal address:

Box 596
75124 Uppsala

Short presentation 

I am coordinator for advanced user support at UPPMAX. Among other things, I am responsible for organising user training events and other activities that engage the application experts at UPPMAX. My focus as an application expert is high-performance parallel programming and algorithm development.

Also available at 

Search for organisation or staff 



CONTACT US

[Contact persons/functions](#)

[Staff](#)

VISIT US

[Adresses](#)

[Map](#)

SHORTCUTS

[Evacuation Plan](#)

[Emergency Contacts](#)

[Report an Error](#)

[Edit this page](#)

Uppsala universitet använder kakor (cookies) för att webbplatsen ska fungera bra för dig. [Läs mer om kakor](#). [OK](#)



© 2020 Uppsala University | Phone: +46 18 471 00 00 | P.O. Box 337, SE-751 05 Uppsala, Sweden

Registration number: 202100-2932 | VAT number: SE202100293201 | [Registrar](#) | [About this website](#) | Editor: [Marcus Lundberg](#).

[GO TO TOP](#)





Contact - persons & functions

Search for staff

List by employment

List by building

List by last name

List by first name

Portrait gallery

Support and service

Search the directory

Write the first and/or last name of the person you are looking for.

Name:

No person "Elias Rudberg" was found in the directory.

Other search options

 Search the university's [directory](#)[Show printer-friendly page](#) | Copyright © 2020 Uppsala University, Department of Information Technology.Last updated 10 Nov 2018. Responsible: [Björn Victor](#). Web: [Contact](#).



UPPSALA
UNIVERSITET



[Svensk startsida](#) | [Log in](#)



▼ Department of Information Technology

[Uppsala University](#) / [Information Technology](#) / [Search for staff](#) /

Listen

Sverker Holmgren



Professor at [Department of Information Technology](#), [**Division of Scientific Computing**](#)

[vCard](#)

Email:

[Sverker.Holmgren\[AT-sign\]it.uu.se](mailto:Sverker.Holmgren@it.uu.se)

Telephone:

[+4618-471 2992](tel:+4618-4712992)

Mobile phone:

[+46 70 4250797](tel:+46704250797)

Visiting address:

Room POL ITC 2411 ITC, Lägerhyddsvägen 2, hus 2
752 37 UPPSALA

Postal address:

Box 337
751 05 UPPSALA

[Download CV](#) 

Also available at

Biography

Search for organisation or staff 

Updated 2016-11-18 13:04:10 by [Sverker Holmgren](mailto:Sverker.Holmgren).

CONTACT US

Contact persons/functions

Staff

VISIT US

Adresses

Map

SHORTCUTS

Evacuation Plan

Emergency Contacts

Report an Error

Edit this page

Uppsala universitet använder kakor (cookies) för att webbplatsen ska fungera bra för dig. Läs mer om kakor. OK

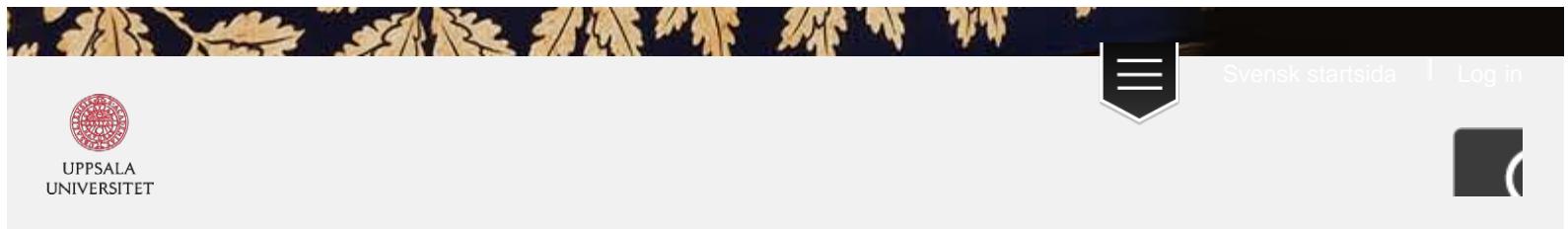


© 2020 Uppsala University | Phone: +46 18 471 00 00 | P.O. Box 337, SE-751 05 Uppsala, Sweden

Registration number: 202100-2932 | VAT number: SE202100293201 | Registrar | About this website | Editor: Sverker Holmgren.

[GO TO TOP](#)





Uppsala University / Information Technology / Search for staff /

Denna sida på svenska

Listen

Carl Nettelblad



Senior Lecturer/Associate Professor at [Department of Information Technology](#).

[**Division of Scientific Computing**](#)

vCard

Email:

[carl.nettelblad\[AT-sign\]it.uu.se](mailto:carl.nettelblad@it.uu.se)

Mobile phone:

[+46 70 3591242](tel:+46703591242)

Visiting address:

Room POL 2422 ITC, Lägerhyddsvägen 2, hus 2
752 37 UPPSALA

Postal address:

Box 337
751 05 UPPSALA

Researcher at [Department of Cell and Molecular Biology, **Molecular Biophysics**](#)

vCard 

Mobile phone:

[+46 70 3591242](tel:+46703591242)

Visiting address:

Husargatan 3
752 37 Uppsala

Postal address:

Box 596
751 24 UPPSALA

Technical Coordinator at [Department of Information Technology, **Uppsala Multidisciplinary Centre for Advanced Computational Science**](#)

vCard 

Email:

[carl.nettelblad\[AT-sign\]uppmx.uu.se](mailto:carl.nettelblad@uppmx.uu.se)

Mobile phone:

[+46 70 3591242](tel:+46703591242)

Visiting address:

ITC, Lägerhyddsvägen 2, hus 1,2 & 4
752 37 Uppsala

Postal address:
Box 337
751 05 Uppsala

Short presentation



My research is in the field of scientific computing, but with a firm focus on life science application data analysis, utilizing modern computing architectures (including GPU computations and massive parallelism in varying forms). My basic question is "how can we trade experiment result quality for more sophisticated computational methods", giving better results with worse original data.

Keywords: qt xfel hpc genomics hidden markov models

Also available at



Biography



Search for organisation or staff



CONTACT US

[Contact persons/functions](#)

[Staff](#)

VISIT US

[Adresses](#)

[Map](#)

SHORTCUTS

[Evacuation Plan](#)

[Emergency Contacts](#)

[Report an Error](#)

[Edit this page](#)

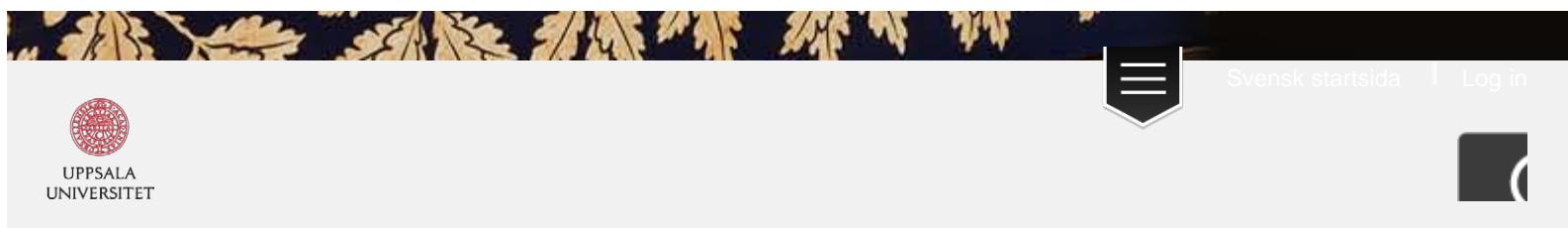
Uppsala universitet använder kakor (cookies) för att webbplatsen ska fungera bra för dig. [Läs mer om kakor.](#) [OK](#)



© 2020 Uppsala University | Phone: +46 18 471 00 00 | P.O. Box 337, SE-751 05 Uppsala, Sweden
Registration number: 202100-2932 | VAT number: SE202100293201 | [Registrar](#) | [About this website](#) | Editor: Carl
Nettelblad.

[GO TO TOP](#)





▼ Department of Information Technology

Uppsala University / Information Technology / Search for staff /

Listen

Stefan Engblom

Senior Lecturer/Associate Professor at [Department of Information Technology](#),
[**Division of Scientific Computing**](#)

vCard

Email:

Stefan.Engblom@it.uu.se

Telephone:

[+4618-471 2754](tel:+46184712754)

Mobile phone:

[+46 70 6206220](tel:+46706206220)

Visiting address:

Room POL 2414a ITC, Lägerhyddsvägen 2, hus 2
752 37 UPPSALA

Postal address:

Box 337

751 05 UPPSALA

Academic merits: Excellent Teacher

Also available at



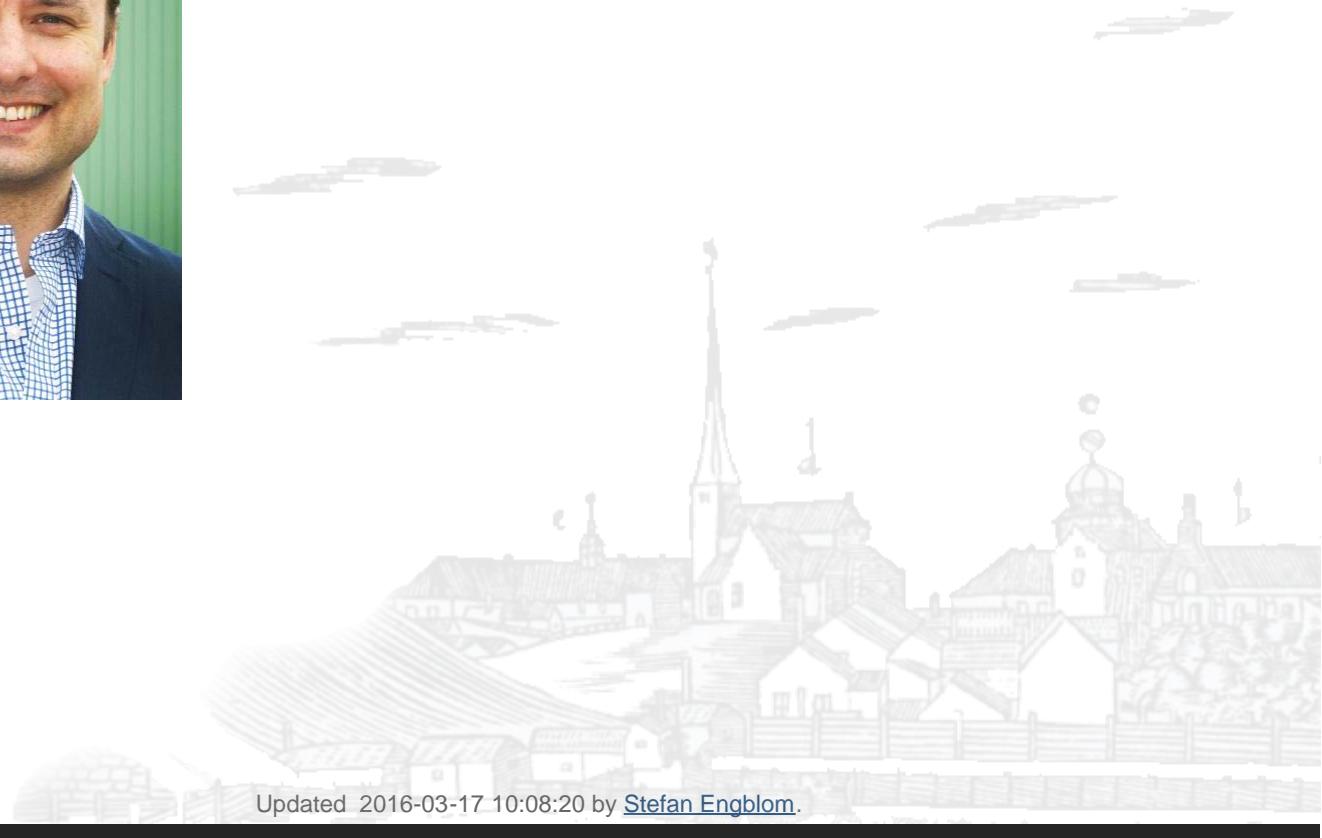
Biography



Research



Search for organisation or staff



Updated 2016-03-17 10:08:20 by [Stefan Engblom](#).

CONTACT US

[Contact persons/functions](#)

[Staff](#)

VISIT US

[Adresses](#)

[Map](#)

SHORTCUTS

[Evacuation Plan](#)

[Emergency Contacts](#)

[Report an Error](#)

[Edit this page](#)

Uppsala universitet använder kakor (cookies) för att webbplatsen ska fungera bra för dig. [Läs mer om kakor.](#) [OK](#)



[GO TO TOP](#)



▼ Department of Information Technology

Uppsala University / Information Technology /



More literature

Below some relevant books are listed (in order of importance). Note that some of this material can be found through the 'links' section.

1. Computer Systems: a programmers perspective by Randal E. Bryant and David R. O'Hallaron (Pearson, 2002)
2. Practical C Programming by Steve Oualline, 3rd Edition, (O'Reilly 1997)
3. High Performance Computing by Kevin Down and Charles Severance, 2nd Edition (O'Reilly, 1998)
4. Performance Optimization of Numerically Intensive Codes by Stefan Goedecker and Adolfy Hoisie (SIAM, 2001)
5. Measuring Computer Performance: a practitioner's guide by David Lilja (Cambridge University Press, 2000)
6. Techniques for Optimizing Applications by Rajat P. Garg and Ilya Sharpov (Sun Microsystems Press, 2002)
7. The C Programming Language by Brian Kernighan and Dennis Ritchie, 2nd Edition (Prentice-Hall, 1988)
8. Computer Architecture: a quantitative approach by John Hennessy and David Patterson, 3rd Edition (Morgan Kaufmann, 2003)
9. Java Performance Tuning by Jack Shirazi, 2nd Edition (O'Reilly, 2003)



Updated 2012-03-13 08:53:09 by [Marcus Holm](#).

CONTACT US

[Contact persons/functions](#)

[Staff](#)

VISIT US

[Adresses](#)

[Map](#)

SHORTCUTS

[Evacuation Plan](#)

[Emergency Contacts](#)

[Report an Error](#)

[Edit this page](#)

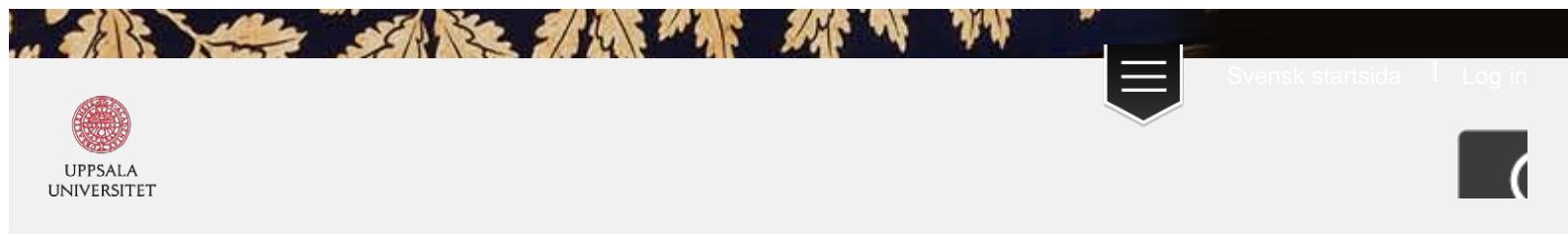
Uppsala universitet använder kakor (cookies) för att webbplatsen ska fungera bra för dig. [Läs mer om kakor.](#) [OK](#)



© 2020 Uppsala University | Phone: +46 18 471 00 00 | P.O. Box 337, SE-751 05 Uppsala, Sweden
Registration number: 202100-2932 | VAT number: SE202100293201 | [Registrar](#) | [About this website](#) | Editor: [Anneli Folkesson](#).

[GO TO TOP](#)





Uppsala University / Information Technology /

Listen

Useful links on the Internet

C PROGRAMMING

- Good page for reference, literature and links on C programming:
<http://www.lysator.liu.se/c/>
- Walkthrough of the C integer types: <http://home.att.net/~jackklein/c/inttypes.html>
- KTH course material for those who know Java:
<http://www.nada.kth.se/datorer/haftena/java2c/>
- Nice and compact overview of C programming:
<http://computer.howstuffworks.com/c.htm>
- Material on pointers over at Standford: <http://cslibrary.stanford.edu/>
- What's new in C99? http://home.tiscalinet.ch/t_wolf/tw/c/c9x_changes.html
- Sun ONE Studio compilers,
<http://developers.sun.com/prodtech/cc/reference/docs/index.html>
- UU material on C/C++ object-oriented programming,
<http://user.it.uu.se/~ngssc/OOP04/module3/>

PROGRAMMING TOOLS

- Common gcc flags, <http://www.freeos.com/articles/3185>
- Material on linking and loading, <http://www.iecc.com/linker/>
- Make tutorial, <http://www.edc.ncl.ac.uk/pages/comp/tutorial/make/>
- SUN's material on debugging with DBX, <http://docs.sun.com/db/doc/817-0923?q=dbx>
- Debugging tutorials,
<http://www.eeng.brad.ac.uk/help/.packlangtool/.langs/.dbx/.resource.html>
- Document on reversed engineering. Includes debugging, low-level stuff and linking, <http://www.acm.uiuc.edu/sigmil/RevEng/>
- Very nice articles about many thing from Sun,
<http://developer.sun.com/prodtech/cc/reference/techart/index.html>
- More on calling C from Fortran and vice versa, http://docs.sun.com/source/817-0929/11_cfort.html
- Documentation for the Sun integrated developer environment (IDE),
<http://wwws.sun.com/software/sundev/previous/studio4u1/documentation/index.html>
- Development Tools on Linux, <http://www.hotfeet.ch/~gemi/LDT/>
- GNU tools for software development, <http://www.gnu.org/directory-devel/>

PERFORMANCE MEASUREMENT

- A site for the methodology of performance analysis, <http://labq.com/>
- SPEC, <http://www.spec.org/>
- Article about misleading SPEC numbers (only works from UU),
<http://doi.acm.org/10.1145/859618.859625>

- PAPI, library for hardware counters on several platforms,
<http://icl.cs.utk.edu/papi/index.html>
- The Sun Performance Analyzer,
http://developers.sun.com/prodtech/cc/analyzer_index.html
- Using gprof, the GNU profiler, <http://www.cs.utah.edu/dept/old/texinfo/as/gprof.html>
- Performance Analyzer Tutorial. A must do. Replace install-dir with
/opt/SUNWspro <http://docs.sun.com/source/817-0922/Demo.html>

OPTIMIZATION

- Site for the *Computer Systems: a programmers perspective* book. Must see.
<http://csapp.cs.cmu.edu/>
- Nice text about RISC CPUs, <http://en.wikipedia.org/wiki/RISC>
- Compiling for the UltraSPARC IIICu processor,
<http://developers.sun.com/tools/cc/articles/US3Cu/US3Cu.content.html>
- Basic Linear Algebra Subroutines, BLAS <http://www.netlib.org/blas/>
- Overview of the routines in Sun Performance Library,
http://docs.sun.com/source/817-0935/plug_lib_list.html
- Nice Sun article about floating point precision,
http://developers.sun.com/tools/cc/articles/fp_errors.html
- Intel compilers and documentation,
<http://www.intel.com/software/products/compilers/>
- Fast inverse square-root library at UPPMAX,
<http://www.uppmax.uu.se/Members/daniels/fastisqrt/a-library-for-the-computation-of-fast-inverse-square-roots>

Updated 2012-03-13 08:53:17 by [Marcus Holm](#).

CONTACT US

[Contact persons/functions](#)

[Staff](#)

VISIT US

[Adresses](#)

[Map](#)

SHORTCUTS

[Evacuation Plan](#)

[Emergency Contacts](#)

[Report an Error](#)

[Edit this page](#)

Uppsala universitet använder kakor (cookies) för att webbplatsen ska fungera bra för dig. [Läs mer om kakor.](#) OK



© 2020 Uppsala University | Phone: +46 18 471 00 00 | P.O. Box 337, SE-751 05 Uppsala, Sweden
Registration number: 202100-2932 | VAT number: SE202100293201 | [Registrar](#) | [About this website](#) | Editor: [Anneli Folkesson](#).

[GO TO TOP](#)





International site

Logga in

Lyssna



Vägen till din utbildning – så gör du för att anmäla dig

SVERIGES FÖRSTA UNIVERSITET

Bli en del av något stort.

Hitta din utbildning

Jobba hos oss

NYHETER



AI-professorn: "Tekniken har kommit för att stanna" 29 sep 2020

Atombiljard avslöjar kemiska reaktioner på 28 sep 2020

elektronnivå

Möt Uppsala universitet online på
Innovationsveckan 2020 26 sep 2020

Anmälan till högskoleprovet öppnar 25
september 24 sep 2020

Fler nyheter →

KALENDARIUM



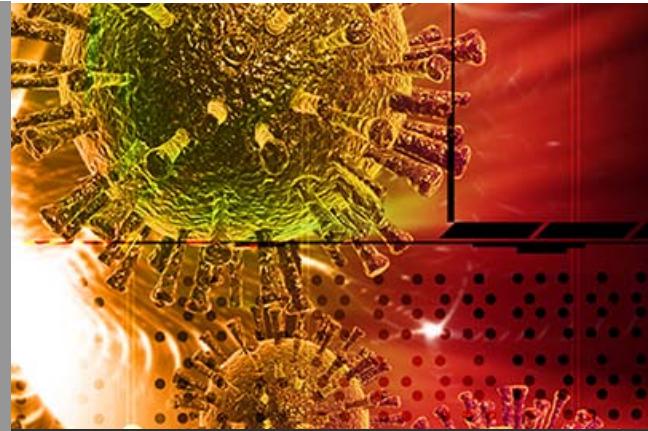
30 SEP
kl. 13.15
Amerikanska rasrelationer och Sverige

30 SEP
kl. 15.30
Våga tala

1 OKT
kl. 15.15
Gör det nu!

Fler evenemang →

CORONAVIRUSET



Läs de senaste uppdateringarna om covid-19.

Coronainformation till

medarbetare →

Coronainformation till studenter →

REKTORSBLOGGEN



28 sept: Universitetets utmaningar
och möjligheter förstärks på
Campus Gotland



KONTAKT

Telefon: [018-471 00 00](tel:018-4710000)

[Kontakta universitetet](#)

[Hitta forskare & personal](#)

FÖLJ UPPSALA UNIVERSITET PÅ



HITTA HIT

[Institutioner & enheter](#)

[Våra campus](#)

[Museer & trädgårdar](#)

[Karta över Uppsala universitet](#)

GENVÄGAR

[Universitetets webbutik](#)

[Nyheter & press](#)

[Lediga jobb](#)

[Bibliotek](#)

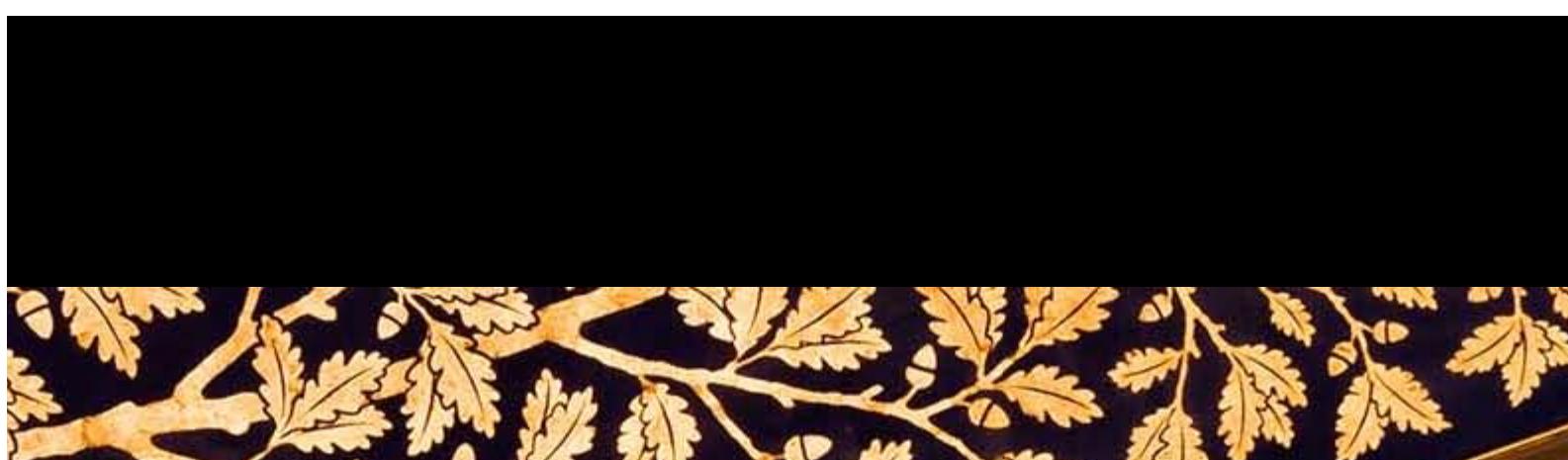
[Mål & regler](#)

[Stöd Uppsala universitet](#)

[Pågående upphandlingar](#)

[Medarbetarportalen](#)

[Studentportalen](#)



© Uppsala universitet | Telefon: 018-471 00 00 | Box 256, 751 05 Uppsala

Organisationsnummer: 202100-2932 | Momsregistreringsnummer: SE202100293201 | PIC: 999985029 | [Registrator](#) |

[Om webbplatsen](#) | [Dataskyddspolicy](#) | Sidansvarig: webbredaktionen@uadm.uu.se

 LOG IN

 Read aloud | Svenska



UPPSALA
UNIVERSITET

Student Portal

START HELP SEARCH MENU 

HOW DO I REGISTER?

Coronavirus

[Current recommendations on coronavirus \(covid-19\)](#)

Information about when, where and how the student is to register is found on the course page in Student Portal.

As a new student, you must:

1. [Activate your student account](#) *

2. [Log in to the Student Portal](#)

Here you see information on what courses you are admitted to and how to register.

3. Register for your course

If you do not register, you may lose your place!

4. [Get Campus card](#) *

Tip! Information concerning how to register can be found by searching for the name or application code for the course via [Search](#) in the Student Portal.

[Welcome to Uppsala University](#)

NEWS



[Students infected with coronavirus – get tested and stay home if you have symptoms](#) *

Nyheter från Uppsala universitet 2020-09-15

[Problems with the systems for access control](#) *

Nyheter från Uppsala universitet 2020-09-09

[Nominate for the Martin Henriksson Holmdahl Prize](#) *

Nyheter från Uppsala universitet 2020-09-03

[Nominate candidates for Uppsala Student of the Year](#) *

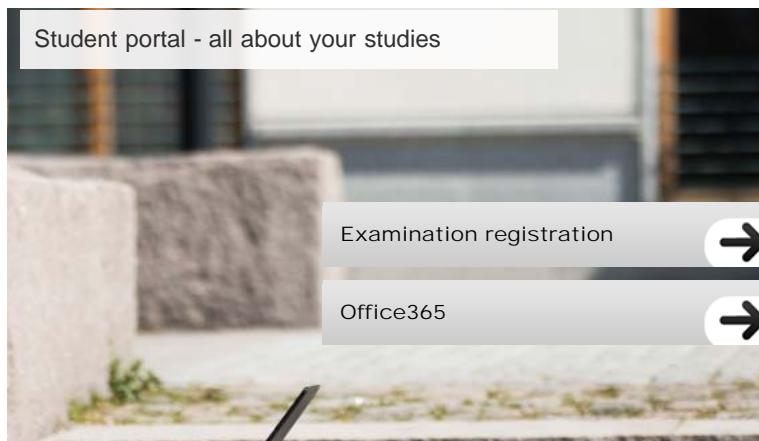
Nyheter från Uppsala universitet 2020-09-03

[Scholarship opportunities](#) *

Nyheter från Uppsala universitet 2020-08-26

Search for courses and programmes

Include courses before this semester.



SUPPORT AND SERVICES



If you are a student at Uppsala University there are different kinds of support and service offered.



STUDENT UNIONS AND NATIONS



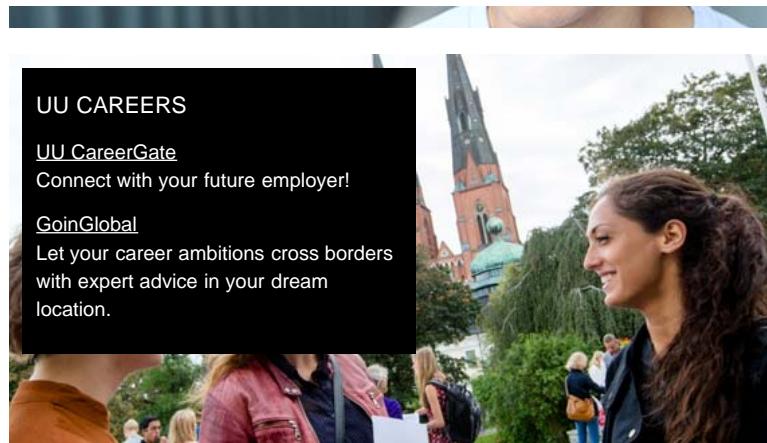
Information about membership in Student Unions and Nations at Uppsala University.



THE STUDENT HEALTH SERVICE

The Student Health Service have psychologists and counsellors, all specialised in study-related health care. If you need support or counselling we are here for you - welcome!





UU CAREERS

UU CareerGate

Connect with your future employer!

GoinGlobal

Let your career ambitions cross borders with expert advice in your dream location.

STUDY ABROAD

For students who wish to study part of their degree abroad.



AVAILABLE SCHOLARSHIPS

Find and apply for available scholarships



LIBRARY

The University Library has millions of printed and electronic books that are available for students at Uppsala University.



WORKING IN TANDEM WITH YOUR STUDIES

The opportunity for practical work experience



UPPSALA UNIVERSITY'S ALUMNI NETWORK

A global network for students like yourself who are just about to enter the job market. Our alumni network is an important resource for your career, for maintaining contacts from your university years and for developing new contacts.



High Performance Computing and Programming

Lab 4 — SIMD and Vectorization

Marcus Holm <marcus.holm@it.uu.se>

1 Introduction

The purpose of this lab assignment is to give some experience in using SIMD instructions on x86 and getting compiler auto-vectorization to work. We will use matrix-vector and matrix-matrix multiplication to illustrate how SIMD can be used for numerical algorithms.

You will be using GCC in this lab. GCC supports two sets of intrinsics, or built-ins, for SIMD. One is native to GCC and the other one is defined by Intel for their C++ compiler. We will use the intrinsics defined by Intel since these are much better documented.

Both Intel¹ and AMD² provide excellent optimization manuals that discuss the use of SIMD instructions and software optimizations. These are good sources for information if you are serious about optimizing your software, but they are not mandatory reading for this assignment. You will, however, find them, and the instruction set references, useful as reference literature when using SSE. Another useful reference is the Intel C++ compiler manual³, which documents the SSE intrinsics supported by ICC and GCC.

We will, for various practical reasons, use the Linux lab machines for this lab assignment. Section 3 introduces the tools and commands you need to know to get started.

2 Introduction to SSE

The SSE extension to the x86 consists of a set of 128-bit vector registers and a large number of instructions to operate on them. The number of available registers depends on the mode of the processor, only 8 registers are available in 32-bit mode, while 16 registers are available in 64-bit mode. The lab systems you'll be using are 64-bit machines.

The data type of the packed elements in the 128-bit vector is decided by the specific instruction. For example, there are separate addition instructions for adding vectors of single and double precision floating point numbers. Some operations that are normally independent of the operand types (integer or floating point), e.g. bit-wise operations, have separate instructions for different types for performance reasons.

¹<http://www.intel.com/products/processor/manuals/>

²<http://developer.amd.com/documentation/guides/>

³http://software.intel.com/sites/products/documentation/hpc/compilerpro/en-us/cpp/lin/compiler_c/index.htm

Header file	Extension name	Abbrev.
xmmmintrin.h	Streaming SIMD Extensions	SSE
emmmintrin.h	Streaming SIMD Extensions 2	SSE2
pmmmintrin.h	Streaming SIMD Extensions 3	SSE3
tmmmintrin.h	Supplemental Streaming SIMD Extensions 3	SSSE3
smmmintrin.h	Streaming SIMD Extensions 4 (Vector math)	SSE4.1
nmmmintrin.h	Streaming SIMD Extensions 4 (String processing)	SSE4.2
gmmmintrin.h	Advanced Vector Extensions Instructions	AVX

Table 1: Header files used for different SSE versions

When reading the manuals, it's important to keep in mind that the size of a *word* in the x86-world is 16 bits, which was the word size of the original microprocessor which the entire x86-line descends from. Whenever the manual talks about a *word*, it's really 16 bits. A 64-bit integer, i.e. the register size of a modern x86, is known as a quadword. Consequently, a 32-bit integer is known as a doubleword.

2.1 Using SSE in C-code

Using SSE in a modern C-compiler is fairly straightforward. In general, no assembler coding is needed. Most modern compilers expose a set of vector types and intrinsics to manipulate them. We will assume that the compiler supports the same SSE intrinsics as the Intel C-compiler. The intrinsics are enabled by including the correct header file. The name of the header file depends on the SSE version you are targeting, see Table 1. You may also need to pass an option to the compiler to allow it to generate SSE code, e.g. `-msse3`. A portable application would normally try to detect which SSE extensions are present by running the CPUID instruction and use a fallback algorithm if the expected SSE extensions are not present. For the purpose of this assignment, we simply ignore those portability issues and assume that at least SSE3 is present, which is the norm for processors released since 2005.

The SSE intrinsics add a set of new data types to the language, these are summarized in Table 2. In general, the data types provided to support SSE provide little protection against programmer errors. Vectors of integers of different size all use the same vector type (`_m128i`), there are however separate types for vectors of single and double precision floating point numbers.

The vector types do not support the native C operators, instead they require explicit use of special intrinsics. All SSE intrinsics have a name on the form `_mm_<op>_<type>`, where `<op>` is the operation to perform and `<type>` specifies the data type. The most common types are listed in Table 2.

The following sections will present some useful instructions and examples to get you started with SSE. This is not intended to be an exhaustive list of available instructions or intrinsics. In particular, most of the instructions that rearrange data within vectors (shuffling), various data-packing instructions and generally esoteric instructions have been left out. Interested readers should refer to the optimization manuals from the CPU manufacturers for a more thorough introduction.

2.2 Loads and stores

There are three classes of load and store instructions for SSE. They differ in how they behave with respect to the memory system. Two of the classes require their memory

Intel Name	Elements/Reg.	Element type	Vector type	Type
Bytes	16	int8_t	__m128i	epi8
Words	8	int16_t	__m128i	epi16
Doublewords	4	int32_t	__m128i	epi32
Quadwords	2	int64_t	__m128i	epi64
Single Precision Floats	4	float	__m128	ps
Double Precision Floats	2	double	__m128d	pd

Table 2: Packed data types supported by the SSE instructions. The fixed-length C-types requires the inclusion of `stdint.h`.

Listing 1: Load store example using *unaligned* accesses

```
#include <pmmINTRIN.h>

static void
my_memcpy(char *dst, const char *src, size_t len)
{
    /* Assume that length is an even multiple of the
     * vector size */
    assert((len & 0xF) == 0);
    for (int i = 0; i < len; i += 16) {
        __m128i v = _mm_loadu_si128((__m128i *) (src + i));
        _mm_storeu_si128((__m128i *) (dst + i), v);
    }
}
```

operands to be naturally aligned, i.e. the operand has to be aligned to its own size. For example, a 64-bit integer is naturally aligned if it is aligned to 64-bits. The following memory accesses classes are available:

Unaligned A “normal” memory access. Does not require any special alignment, but may perform better if data is naturally aligned.

Aligned Memory access type that requires data to be aligned. Might perform slightly better than unaligned memory accesses. Raises an exception if the memory operand is not naturally aligned.

Streaming Memory accesses that are optimized for data that is streaming, also known as non-temporal, and is not likely to be reused soon. Requires operands to be naturally aligned. Streaming stores are generally much faster than normal stores since they can avoid reading data before the writing. However, they require data to be written sequentially and, preferably, in entire cache line units. We will not be using this type in the lab.

See Table 3 for a list of load and store intrinsics and their corresponding assembler instructions. A usage example is provided in Listing 1. Constants should usually not be loaded using these instructions, see section 2.4 for details about how to load constants and how to extract individual elements from a vector.

	Intrinsic	Assembler	Vector Type
Unaligned	_mm_loadu_si128	MOVDQU	__m128i
	_mm_storeu_si128	MOVDQU	__m128i
	_mm_loadu_ps	MOVUPS	__m128
	_mm_storeu_ps	MOVUPS	__m128
	_mm_loadu_pd	MOVUPD	__m128d
	_mm_storeu_pd	MOVUPD	__m128d
	_mm_load1_ps	Multiple	__m128
Aligned	_mm_load1_pd	Multiple	__m128d
	_mm_load_si128	MOVDQA	__m128i
	_mm_store_si128	MOVDQA	__m128i
	_mm_load_ps	MOVAPS	__m128
	_mm_store_ps	MOVAPS	__m128
Streaming	_mm_load_pd	MOVAPD	__m128d
	_mm_store_pd	MOVAPD	__m128d
	_mm_stream_si128	MOVNTDQ	__m128i
	_mm_stream_ps	MOVNTPS	__m128
	_mm_stream_pd	MOVNTPD	__m128d
	_mm_stream_load_si128	MOVNTDQA	__m128i

Table 3: Load and store operations. The `load1` operation is used to load one value into all elements in a vector.

2.3 Arithmetic operations

All of the common arithmetic operations are available in SSE, see Table 4. Addition, subtraction and multiplication is available for all vector types, while division is only available for floating point vectors.

A special *horizontal add* operation is available to add pairs of values, see Figure 1 and Listing 2, in its input vectors. This operation can be used to implement efficient reductions. Using this instruction to create a vector of sums of four vectors with four floating point numbers can be done using only three instructions.

There is an instruction to calculate the scalar product between two vectors. This instruction takes three operands, the two vectors and an 8-bit flag field. The four highest bits in the flag field are used to determine which elements in the vectors to include in the calculation. The lower four bits are used as a mask to determine which elements in the destination are updated with the result, the other elements are set to 0. For example, to include all elements in the input vectors and store the result to the third element in the destination vector, set flags to F4₁₆.

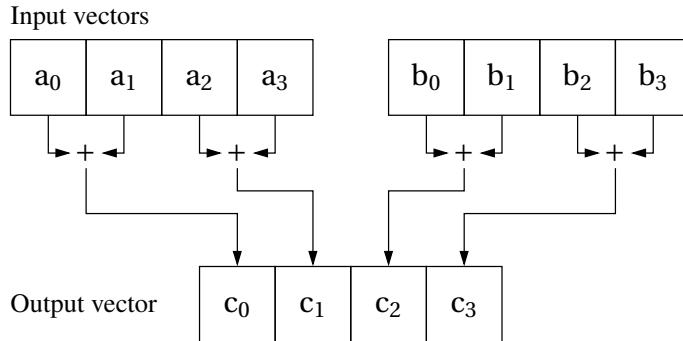
A transpose macro is available to transpose 4×4 matrices represented by four vectors of packed floats. The transpose macro expands into several assembler instructions that perform the in-place matrix transpose.

Individual elements in a vector can be compared to another vector using compare intrinsics. These operations compare two vectors; if the comparison is true for an element, that element is set to all binary 1 and 0 otherwise. Only two compare instructions, equality and greater than, working on integers are provided by the hardware. The less than operation is synthesized by swapping the operands and using the greater than comparison. See Listing 3 for an example of how to use the SSE compare instructions.

Intrinsic	Operation
<code>_mm_add_<type>(a, b)</code>	$c_i = a_i + b_i$
<code>_mm_sub_<type>(a, b)</code>	$c_i = a_i - b_i$
<code>_mm_mul_(ps pd)(a, b)</code>	$c_i = a_i b_i$
<code>_mm_div_(ps pd)(a, b)</code>	$c_i = a_i / b_i$
<code>_mm_hadd_(ps pd)(a, b)</code>	Performs a horizontal add, see Figure 1
<code>_mm_dp_(ps pd)(a, b, FLAGS)</code>	$\mathbf{c} = \mathbf{a} \cdot \mathbf{b}$ (dot product)
<code>_MM_TRANSPOSE4_PS(a, ..., d)</code>	Transpose the matrix $(a^t \dots d^t)$ in place
<code>_mm_cmpeq_<type>(a, b)</code>	Set c_i to -1 if $a_i = b_i$, 0 otherwise
<code>_mm_cmpgt_<type>(a, b)</code>	Set c_i to -1 if $a_i > b_i$, 0 otherwise
<code>_mm_cmplt_<type>(a, b)</code>	Set c_i to -1 if $a_i < b_i$, 0 otherwise

Table 4: Arithmetic operations available in SSE. The transpose operation is a macro that expands to several SSE instructions to efficiently transpose a matrix.

Figure 1: Calculating $c = \text{ _mm_hadd_ps }(a, b)$



Listing 2: Sum the elements of four vectors and store each vectors sum as one element in a destination vector

```
#include <pmmintrin.h>

static __m128
vec_sum(const __m128 v0, const __m128 v1,
        const __m128 v2, const __m128 v3)
{
    return _mm_hadd_ps(
        _mm_hadd_ps(v0, v1),
        _mm_hadd_ps(v2, v3));
}
```

Intrinsic	Operation
<code>_mm_set_<type>(p₀, ..., p_n)</code>	$c_i = p_i$
<code>_mm_setzero_(ps pd si128)()</code>	$c_i = 0$
<code>_mm_set1_<type>(a)</code>	$c_i = a$
<code>_mm_cvts_ss_f32(a)</code>	Extract the first float from a
<code>_mm_cvts_sd_f64(a)</code>	Extract the first double from a

Table 5: Miscellaneous operations. Most of the operations expand into multiple assembler instructions.

Listing 3: Transform an array of 16-bit integers using a threshold function. Values larger than the threshold (4242) are set to FFFF₁₆ and values smaller than the threshold are set to 0

```
#include <stdint.h>
#include <pmmmintrin.h>

static void
threshold(uint16_t *dst, const uint16_t *src, size_t len)
{
    const __m128i t = _mm_set1_epi16(4242);
    for (int i = 0; i < len; i += 8) {
        const __m128i v = _mm_loadu_si128((__m128i *) (src + i));
        _mm_storeu_si128((__m128i *) (dst + i),
                          _mm_cmpgt_epi16(v, t));
    }
}
```

2.4 Loading constants and extracting elements

There are several intrinsics for loading constants into SSE registers, see Table 5. The most general can be used to specify the value of each element in a vector. In general, try to use the most specific intrinsic for your needs. For example, to load 0 into all elements in a vector, `_mm_set_epi64`, `_mm_set1_epi64` or `_mm_setzero_si128` could be used. The two first will generate a number of instructions to load 0 into the two 64-bit integer positions in the vector. The `_mm_setzero_si128` intrinsic uses a shortcut and emits a `PXOR` instruction to generate a register with all bits set to 0.

There are a couple of intrinsics to extract the first element from a vector. They can be useful to extract results from reductions and similar operations.

2.5 Data alignment

Aligned memory accesses are usually required to get the best possible performance. There are several ways to allocate aligned memory. One would be to use the POSIX API, but `posix_memalign` has an awkward syntax and is unavailable on many platforms. A more convenient way is to use the intrinsics in Table 6. Remember that data allocated using `_mm_malloc` must be freed using `_mm_free`.

It is also possible to request a specific alignment of static data allocations. The preferred way to do this is using GCC attributes, which is also supported by the Intel compiler. See Listing 4 for an example.

Intrinsic	Operation
<code>_mm_malloc(s, a)</code>	Allocate <code>s</code> B of memory with a B alignment
<code>_mm_free(*p)</code>	Free data previously allocated by <code>_mm_malloc(s, a)</code>

Table 6: Memory allocation

Listing 4: Aligning static data using attributes

```
float foo [SIZE] __attribute__((aligned (16)));
```

3 Getting started

In this lab we will be using the Linux lab machines. To log in on them, first log into the university Solaris (Unix) computers, then connect to a Linux machine with *linuxlogin*. In some cases (e.g. if you are logged in to a university machine remotely), this may not work, in which case you may use *rlogin* or *xrlogin* to a Linux server of your choice. If you're using your own computer, be aware that vectorization support and implementation varies completely from computer to computer, so the lab instructions may not work at all.

Download the source package from Studentportalen or the course homepage and extract the files.

4 The code framework

Makefile Automates the compilation. You can use **make clean** to remove automatically generated files from the working directory.

matmul.c Skeleton code for multiplying matrices. Also contains reference code for testing and timing.

matvec.c Skeleton code for multiplying a matrix and a vector. Also contains reference code for testing and timing.

matvec_AUTO.c As matvec.c, but for the auto-vectorization task.

util.(cl)h Utility functions for printing vectors and measuring time. See the header file for more information.

5 Multiplying a matrix and a vector

Multiplying a matrix and a vector can be accomplished by the code in Listing 5, this should be familiar if you have taken a linear algebra course. The first step in vectorizing this code is to unroll it four times. Since we are working on 32-bit floating point elements, this allows us to process 4 elements in parallel using the 128-bit SIMD registers. The unrolled code is shown in Listing 6.

Listing 5: Simple matrix-vector multiplication

```
static void
matvec_simple(size_t n, float vec_c[n],
              const float mat_a[n][n], const float vec_b[n])
{
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            vec_c[i] += mat_a[i][j] * vec_b[j];
}
```

Listing 6: Matrix-vector multiplication, unrolled four times

```
static void
matvec_unrolled(size_t n, float vec_c[n],
                const float mat_a[n][n], const float vec_b[n])
{
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j += 4)
            vec_c[i] += mat_a[i][j + 0] * vec_b[j + 0]
                        + mat_a[i][j + 1] * vec_b[j + 1]
                        + mat_a[i][j + 2] * vec_b[j + 2]
                        + mat_a[i][j + 3] * vec_b[j + 3];
}
```

5.1 Tasks

1. Implement your version of the matrix-vector multiplication in the `matvec_sse()` function. Run your code and make sure that it produces the correct result. Is it faster than the traditional serial version?
2. Can you think of any optimizations that may make this code faster? *You don't need to implement them.*

6 Matrix-matrix multiplication

The simplest way to multiply two matrices is to use the algorithm in Listing 7. Again, the first step in converting this algorithm to SSE is to unroll some of the loops. The simplest vectorization of this code is to unroll the inner loop 4 times, remember that we can fit four single precision floating point numbers in a vector, and use vector instructions to compute the results of the inner loop.

6.1 Tasks

1. Implement a vectorized version of Listing 7 in the `matmul_sse()` function that belongs to the SSE mode. (Search for the `TASK:` comment). Run your solution to check that it is correct and measure its speedup compared to the serial version. What is the speedup?

Listing 7: Matrix-matrix multiplication

```
static void
matmat(size_t n, float mat_c[n][n],
       const float mat_a[n][n], const float mat_b[n][n])
{
    for (int i = 0; i < n; i++) {
        for (int k = 0; k < n; k++) {
            for (int j = 0; j < n; j++) {
                mat_c[i][j] += mat_a[i][k] * mat_b[k][j];
            }
        }
    }
}
```

7 Auto-vectorization using gcc

Modern compilers can try to automatically apply vector instructions where possible. For gcc, the flag to enable auto-vectorization is `-ftree-vectorize`. However, this process is often hindered by the way code is written. The first step in ensuring that auto-vectorization is doing what is possible is to ask the compiler to tell us about what it is trying to do. This is done with by giving gcc the flag `-ftree-vectorizer-verbose=2`. You can set this flag up to 7, with more information being displayed for each level, see `man gcc` for details.

Once you see which loops that gcc can or cannot auto-vectorize, you can begin to make changes in the program to improve auto-vectorization.

7.1 Tasks

1. Edit `Makefile` to enable auto-vectorization and output vectorization results for the matrix-vector multiplication program.
2. Edit `matvec_AUTO.c` so gcc can auto-vectorize the `matvec_auto` function.
Note: It may be difficult to get any speedup. Why?
3. Edit `matmul.c` to auto-vectorize the `matmul_ref` function. What is the speed compared to the hand-vectorized version?



UPPSALA
UNIVERSITET

High-performance computing and programming I

Introduction to this course

Practical matters

Goals

Motivation



First and foremost

Make **sure** that you have a working UNIX account
(log in on the systems in P2510)

Make **sure** that you have access to this course in
Studentportalen (studentportalen.uu.se)

Make sure that you are on the registration list
circulating now. If you are not, you must register
online.



UPPSALA
UNIVERSITET

Personal introductions

Marcus (marcus.holm@it.uu.se, P2404)

Lectures, labs, assignments



Elias (Elias.Rudberg@it.uu.se, P2440)

Lectures, labs, assignments



Sverker (sverker.holmgren@it.uu.se, P2312)

Professor, really smart guy





Course goals

- To pass, the student should be able to
 - transform algorithms in the computational area to efficient programming code for modern computer architectures;
 - write, organize and handle large programs for numerical computations;
 - use tools for performance optimization and debugging;
 - analyze code with respect to performance and suggest and implement performance improvements.



Course outline

Lectures:

<http://www.it.uu.se/edu/course/homepage/hpb/vt12/lectures>

Week 12: C programming and debugging in our lab environment

Week 13: Performance analysis

Week 16: Optimization techniques

Week 17-20: Various topics in HPC



Course outline

Labs

Lab 1: getting started with C, makefiles, etc

Lab 2: debugging with ddd/gdb

Lab 3: profiling

Lab 4: vectorization

No report needed, oral examination in the lab room

Individual reports required for anyone unable to attend

Lab 1 optional but highly recommended if you are unfamiliar with C or our UNIX systems.



Course outline

Assignments

Emphasis is placed on the analysis and insights in your reports.

Assignment 1: Performance analysis

Describe the performance behavior of a few different implementations of a given algorithm. Code will be provided.

Assignment 2: Program optimization

Multigrid

- Efficient solving of the Poisson equation

N-body

- Simulating a gravitational system efficiently



Exam

UPPSALA
UNIVERSITET

Mandatory exam May 28

Please remember to sign up well ahead of time so we can provide sufficient seating and exam papers.



UPPSALA
UNIVERSITET

Practical aspects, assignments

Assignments preferably done in pairs

Assignments should be runnable on lab machines

Handed in through Studentportalen



Practical aspects, labs

Get your UNIX account in order

Labs will take place in room 1515D.

You can work in the assignment groups you choose

Lab reports (for missing students) and late assignment reports will be corrected at a pace that suits your teachers.

Please try logging in to studentportalen.uu.se



Practical aspects, literature

A lecture schedule is published on the course site,
linked through Studentportalen

Reading suggestions provided in that schedule

Lectures and feedback on assignments are critical
aspects of course content

Lecture slides will be published in Studentportalen
(but they're *lecture* slides)



Practical aspects

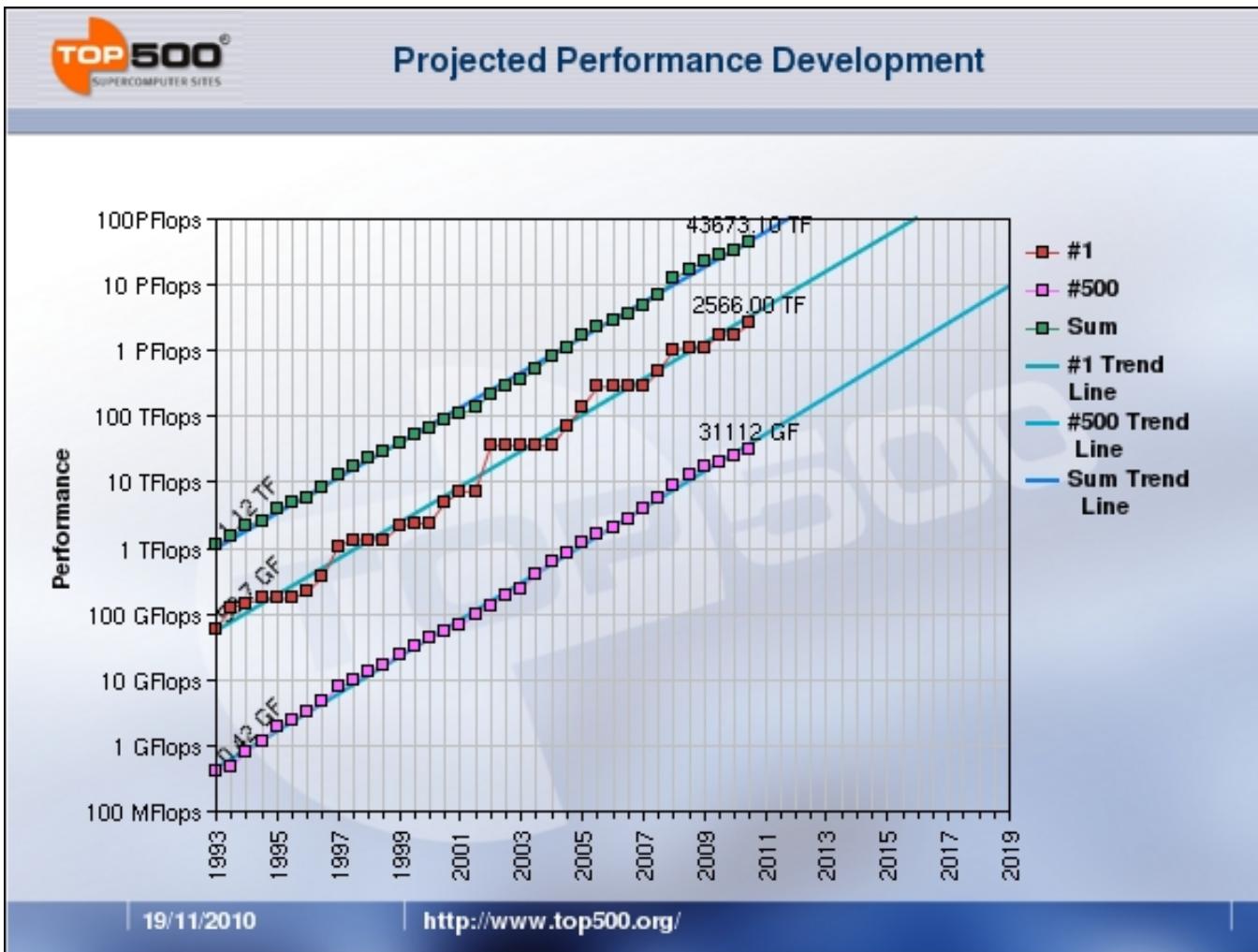
Information about corrections, room changes, etc will be emailed out through Studentportalen and written to the Studentportalen homepage.

Make sure you receive emails from Studentportalen!



Moore's law

UPPSALA
UNIVERSITET



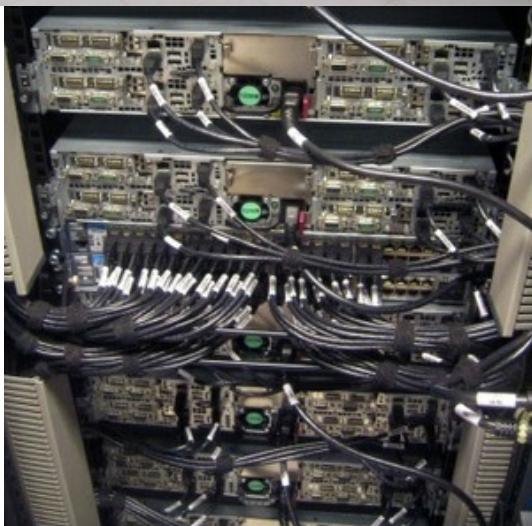
Source: top500.org – http://www.top500.org/lists/2010/11/performance_development



High-performance computing in general



Tianhe-1A. 14,336 Xeon CPUs +
7,168 Nvidia Tesla GPGPUs, 229
TB main memory,
2.5 petaFLOPS peak
performance (Nov 2010).



Kalkyl, 348x2 Xeon CPUs, 9.5
TB main memory, 20.5 TFlops.



What is happening?

Computations used to be expensive

Cheaper than ever

Memory used to be expensive

Cheaper than ever

...but not as cheap as computing

Single-core processors no longer exist

Bandwidth and latency are limiting factors



UPPSALA
UNIVERSITET

TOP 500[®]

NOVEMBER 2011

PRESENTED BY
UNIVERSITY OF
MANNHEIM

ICL
INNOVATIVE
COMPUTING LABORATORY
UNIVERSITY OF TENNESSEE

BERKELEY LAB
Lawrence Berkeley
National Laboratory

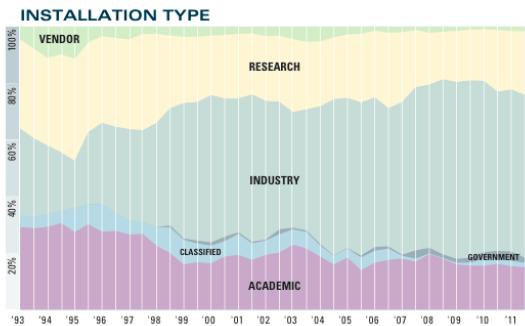
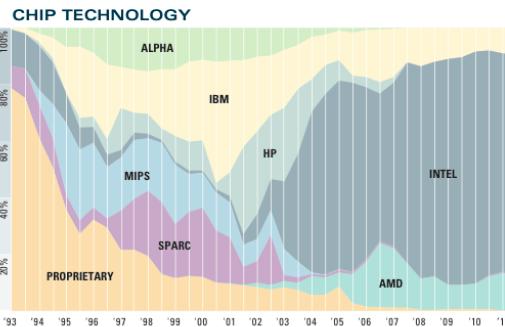
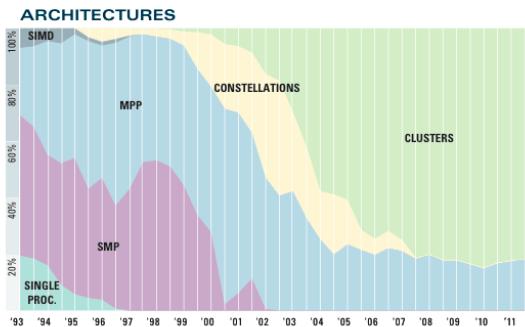
FIND OUT MORE AT
www.top500.org

NAME/MANUFACTURER/COMPUTER	SITE	COUNTRY	CORES	$\frac{R_{max}}{P_{flop/s}}$
1 K computer SPARC64 VIIIfx 2.0GHz, Tofu interconnect	RIKEN	Japan	705,024	10.5
2 Tianhe-1A 6-core Intel X5670 2.93 GHz + Nvidia M2050 GPU w/custom interconnect	NUDT/NSCC/Tianjin	China	186,368	2.57
3 Jaguar Cray XT-5 6-core AMD 2.6 GHz w/custom interconnect	DOE/OS/ORNL	USA	224,162	1.76
4 Nebulae Dawning TC3600 Blade Intel X5650 2.67 GHz, NVidia Tesla C2050 GPU w/lband	NSCS	China	120,640	1.27
5 Tsubame 2.0 HP Proliant SL390s G7 nodes (Xeon X5670 2.93GHz), NVIDIA Tesla M2050 GPU w/lband	TiTech	Japan	73,278	1.19

PERFORMANCE DEVELOPMENT



PROJECTED



HPLINPACK

A Portable Implementation of the High Performance Linpack Benchmark for Distributed Memory Computers

Algorithm: recursive panel factorizations, multiple lookahead depths, bandwidth reducing swapping

Easy to install, only needs MPI + BLAS or VSIP!

Highly scalable and efficient from the smallest cluster to the largest supercomputers in the world

FIND OUT MORE AT <http://icl.eecs.utk.edu/hpl/>



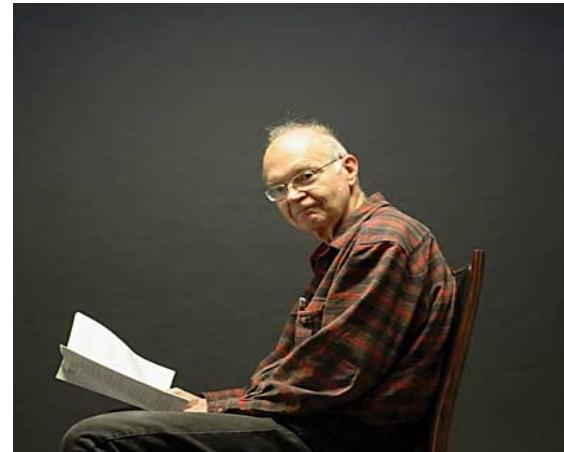
What do we learn in this context?

Writing code for a single core/thread:

1. Access memory
2. Do something
3. Write back to memory

"We should forget about small inefficiencies, about 97% of the time. **Premature optimization is the root of all evil**".

~Donald Knuth,
living legend in
computer science





Fast, faster, wrong

It is much easier to optimize correct code than it is to correct optimized code

Priorities (in order):

Correctness

Flexibility

Performance



Old software is good software

Well-documented and tested software is a very valuable asset

Always **consider** which **existing libraries** that you can build on

Corollary: What resources can *you* provide to the rest of the world?

Slow code can also be incorrect and inflexible

What happens when you get more data, what if you want higher precision?

Make your code age with dignity!



Productivity

Programmer time is not free

Not when writing new code

Not when maintaining old code

Saving 10% runtime by spending 200% more developer time is rarely useful

Consider the total time needed to solve the problem you need to solve

Develop for 10 months, run for 2 months... or

Develop for 2 months, run for 8 months



Complex software

Hardware complexity is increasing

Software complexity increases to make use of it

Scientific computations are often multidisciplinary,
multiscale, multiphysics, multiplatform, ...

More tasks are performed in real-time or subject to
other constraints (e.g. on embedded systems)



Algorithm choice and complexity

Performance is dependent on *what* you do

Only rarely on exactly *how* you do it

Examples:

Searching for a text by scanning every file, or using a small index

Doing 3D by ray-tracing versus Z-buffer rendering

Processing a small preview in real-time versus recomputing the full dataset for each change

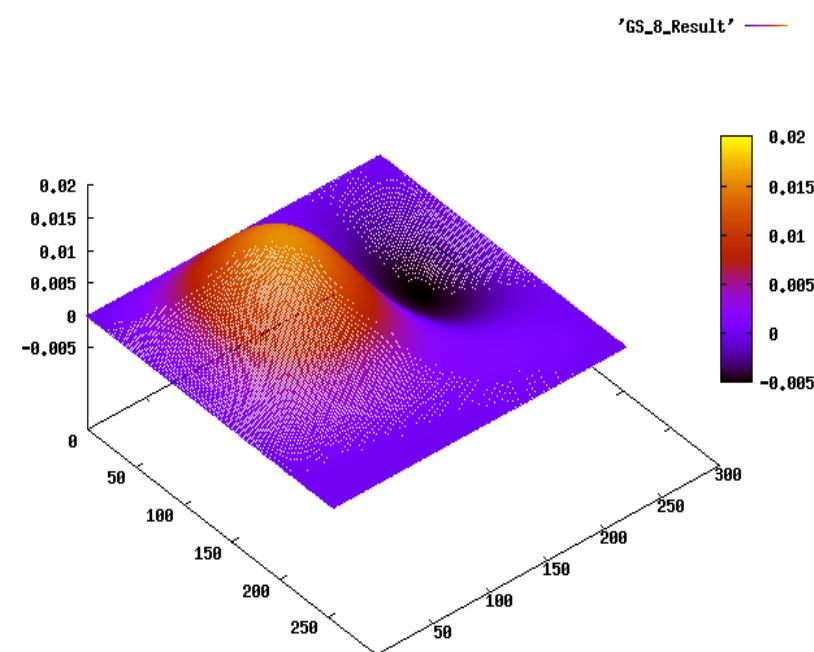
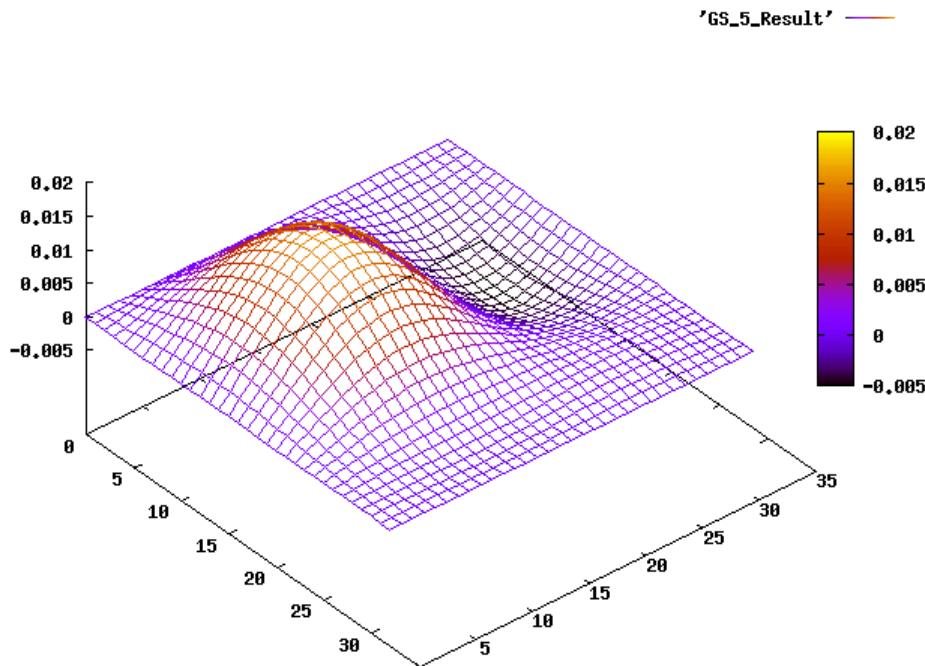
Algorithm choice affects performance vastly

Implementation choices affects performance only a little



Example from Assignment-II

- Solution using
 - Gauss-Seidel Method
 - Multi-Grid Method





Final notes

Remember:

Make sure that you Upunet-S account works

- studentportalen.uu.se

Make sure that your UNIX account works

- Log in on the systems in situ or connect by ssh

Lab 1 this afternoon strongly recommended



UPPSALA
UNIVERSITET

High-performance computing and programming I

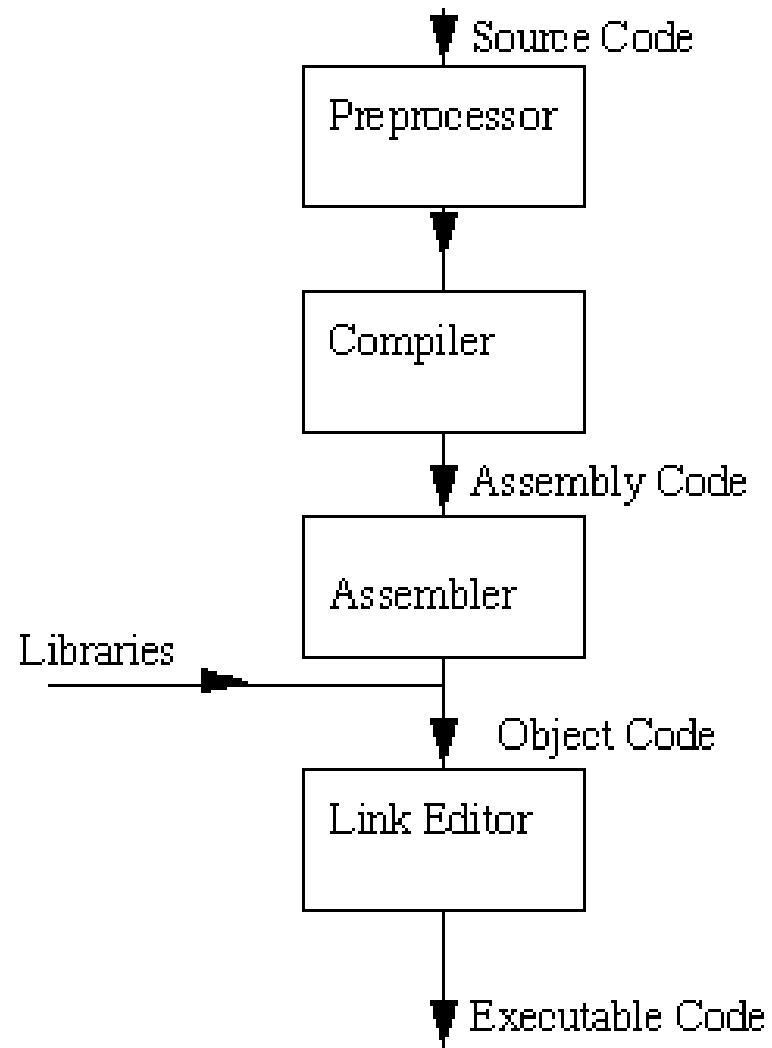
Introduction to C programming in ***NIX**



Writing C programs in Unix/Linux

Writing code

- Building code
 - compile with gcc (or cc)
 - link with gcc (or ld)
 - make
- Environment variables





Hello World

UPPSALA
UNIVERSITET

```
#include <stdio.h>

// This function prints Hello, World!
int main( int argc, char** argv )
{
    printf("Hello, World! %d arguments\n", argc - 1);
    return 0;
}
```



Building code

```
> gcc -o hello helloworld.c
```

What happens?

- Preprocessor changes your code
- Compiler generates object file `helloworld.o`
 - Wonder what's in a `.o` file? Use `nm`
- Linker (`ld`) turns `.o` files into an executable



The C Preprocessor

- Runs before compilation, makes changes to the program code
- **#include**
 - Inserts code into a file (typically a .h file with function prototypes for functions you want to use in a .c file)
- **#define**
 - Define a macro or a constant
- **#if, #ifdef, #ifndef, #endif**

```
#define SQR(x) (x*x)
```



Predefined Macros

<code>_LINE_</code>	Line of the current file
<code>_FILE_</code>	Filename
<code>_DATE_</code>	Date
<code>_TIME_</code>	Time
<code>_STDC_</code>	= 1 if STD C is used
<code>_STDC_VERSION_</code>	= 199409L if amendment to STD C is used



UPPSALA
UNIVERSITET

Makefile

```
CC = gcc          # C compiler
LD = gcc          # C linker
CFLAGS = -Wall   # Flags to be passed to the compiler
LDFLAGS = -lm    # Flags for the linker
RM = /bin/rm -f  # Deletion program
OBJS = maxnorm.o main.o
EXEC = maxnorm
all:maxnorm
maxnorm: $(OBJS)
    $(LD) $(LDFLAGS) $(OBJS) -o $(EXEC)

maxnorm.o: maxnorm.c maxnorm.h
    $(CC) $(CFLAGS) -c maxnorm.c

main.o: maxnorm.h main.c
    $(CC) $(CFLAGS) -c main.c
clean:
    $(RM) $(EXEC) $(OBJS)
```



Compiler flags

- Used to tell the compiler and linker what to do.
- -O0 to -O5 changes the level of automatic optimizations.
- -S tells the compiler to output assembly code
- -c generates .o files but no executable
- -Wall turns on all warnings
- For more information, run `man gcc`



Function declaration & definition

```
// declaration: tells compiler that this function will be used.  
// Must come before function is called.  
// Often placed in a .h file.  
  
double random(double min, double max);  
  
  
void main(int argc, char** argv) {  
    double r = random(0.0, 1.0);  
}  
  
  
// definition: defines what the function does when called.  
// Often placed in .c file, is turned into object code by compiler.  
double random(double min, double max)    // definition  
{  
    double r = (double)rand() / RAND_MAX;  
    r = r * (max - min) + min;  
    return r;  
}
```



Using external libraries

You have the library “mylib”. The object code has been compiled into the file “libmylib.a”, and the header file “mylib.h” provides references to the library.

In your code: `#include "mylib.h"`

Pass to gcc: `-Imyincludedir -Lmylibdir -lmylib`

When the linker sees references to a function, it goes looking in all the .o files and all the library files specified to the linker.



Standard C Library

`<assert.h>` Assert something is true that causes an error if it isn't

`<ctype.h>` E.g. `isdigit`, `isupper`, etc

`<errno.h>` Set error codes

`<float.h>` Can tell you things about floating point variables

`<limits.h>` Can tell you things about integer variables

`<locale.h>` Localization functions

`<math.h>` Math functions

`<signal.h>` Signal handling

`<stdarg.h>` Read commandline arguments

`<stddef.h>` `size_t` and `ptrdiff_t`

`<stdio.h>` `printf` and `scanf`

`<stdlib.h>` memory allocation

`<string.h>` Character strings

`<time.h>` `clock`, `gettimeofday`



Some useful functions

atof(), atoi(), atol()	<stdlib.h>	ASCII to float, integer ..
rand(), srand()	<stdlib.h>	Random numbers
abort(), exit()	<stdlib.h>	Aborts or exits the program
assert()	<assert.h>	Error checking
strcpy(), strlen() ..	<string.h>	Manipulating strings
INT_MAX, INT_MIN, ...	<limits.h>	Limits and bounds of integer types
DBL_EPSILON, ..	<float.h>	Limits and bounds of floating point types
sin(), floor(), ..	<math.h>	Elementary functions and rounding
printf, fscanf, ...	<stdio.h>	Input and Output (I/O)



Output with printf

- `printf(format string, [var1, var2, ...])`

Ex:

```
printf("var=%d\n", a);
```

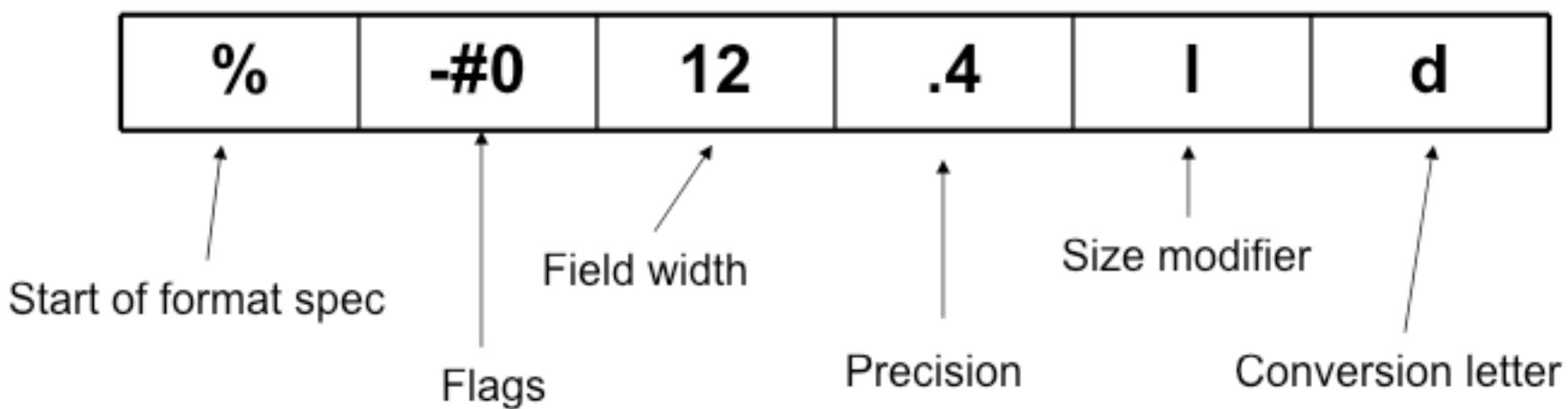
```
printf("in %s: %d\n",
__FILE__, __LINE__)
```

d	Signed integer
u	Unsigned integer
f	Floating point
e	FP in exponential format
c	Character
s	Null terminated string
p	Pointer



Format Specification

- Apart from the conversion letter (d,u,f,c,...), you can control the output in a number of ways:





Special characters

\a	Alert, bell
\b	Backspace
\n	Newline
\r	Carriage return
\t	Tab
\xxx	ASCII number xxx
\', \"	Citation marks
\	Backslash



UPPSALA
UNIVERSITET

Command-line Arguments

```
#include <unistd.h>      // gives you getopt
#include <stdio.h>

int main (int argc, char **argv)
{
    printf("executable name: %s\n", argv[0]);
    printf("number of command-line options: %d\n", argc-1);

    int arg;

    while ((arg = getopt (argc, argv, "abc:")) != -1)
    { /* logic for handling options */ }

    return 0;
}
```



UPPSALA
UNIVERSITET

Variable scope

A variable exists within its set of { }.

```
int myGlobal;  
  
int main(int argc, char** argv) {  
    int myVar;  
    if(argc > 1) {  
        int myVar = argc;  
    }  
    printf("argc greater than 1? %d\n", myVar);  
}
```



Pointers

- A pointer is a variable that points to a place in memory
- It is an integer

```
double foo = 5.5;
double * foo_pointer = &foo; // address of
double bar = *foo_pointer; // ptr dereference
```

- Use pointers for pass-by-reference
 - BUT BEWARE OF SCOPE
 - When a function ends, its local stack variables go away



UPPSALA
UNIVERSITET

Pointers example

```
void increment(int * a) {  
    *a = *a + 1;  
}
```

```
void main() {  
    int a = 0;  
    increment(&a);  
}
```



UPPSALA
UNIVERSITET

Arrays

```
double a[3] = {0.3, 0.4, 0.5};  
  
double *b;  
b = (double*) malloc(3*sizeof(double));  
  
b[1] = a[0];  
  
printf("%f\n", b[1]);  
  
free(b);
```



UPPSALA
UNIVERSITET

Pointer arithmetic

```
double *a;  
a = (double*) malloc(3*sizeof(double));  
  
for(i = 0; i < 3; i++)  
    a[i] = i;  
  
double *b = a;  
  
for(i = 0; i < 3; i++) {  
    printf("%f\n", *b);  
    b++;  
}
```



Arrays vs Pointers

```
int static[3] = {0,2,4};  
int *dynamic = (int*) malloc(3*sizeof(int));  
  
printf("%d\n", sizeof(static)); // 12  
printf("%d\n", sizeof(dynamic)); // 4  
  
/*  
dynamic = static; // legal  
static = dynamic; // illegal  
dynamic++; // legal  
static++; // illegal  
*/
```



2D arrays

- Here's how to allocate on the heap:

```
int rows = 5, columns = 6;

double **array = (double**)malloc(rows*sizeof(double*));
for( int i=0; i<rows; i++)
{
    array[i]=(double*) malloc(columns*sizeof(double));
}
```

- On the stack:

```
double array[3][4];
```



2D arrays, contiguous

- Here's how to allocate on the heap:

```
int rows = 5, columns = 6;  
  
double * data;  
data = (double*)malloc(rows*columns*sizeof(double));  
  
double ** array;  
array = (double **)malloc(rows*sizeof(double*));  
for( int i=0; i<rows; i++)  
{  
    array[i] = &data[i*columns];  
}
```



Structures, typedef

- Struct is a class without methods

```
typedef struct point { double a, b; } point_t;
```

```
struct triangle {
    point_t a, b;
    struct point c;
};
```

```
Struct triangle my_triangle;
my_triangle.a.a = 1.0;
```

- Use typedef to
 - avoid typing 'struct'
 - Hiding the actual type used in your code – save rewriting time!



UPPSALA
UNIVERSITET

Example

```
struct big_one { int a,b,c,d,e,f,g,h,i,j,k,l,m,n,o; };  
  
int f(struct big_one *b, int array[]);  
  
int main(void)  
{  
    struct big_one a_big_one;  
    int array[10];  
    a_big_one.d = 5;  
    (void)f(&a_big_one,array);  
    return 0;  
}  
  
int f(struct big_one *b, int array[])  
{  
    b->d = 6;           // b->d is equivalent to (*b).d  
    array[b->a] = 23;  
    return b->k;  
}
```



Typecasting

- When you need to assign a variable of one type to a variable of another type

```
int a = 5, b = 4;
float x = a / b;           // 1       :(
float y = (float) a / b;   // 1.25   :)

int c = (int) y;           // 1
```



UPPSALA
UNIVERSITET

Hello World II

```
#include <stdio.h>

int main( void )
{
    int i, loops;
    printf("How many times?\n");
    scanf("%d", &loops);

    if( loops > 100 )
        loops = 100;

    for( i=0; i<loops; i++ )
    {
        printf("%d: Hello, World!", i);
    }
    return 0;
}
```



Bitwise operators

```
char a = 1;           // 0000 0001
char b = a << 2;     // 0000 0100
b == 4;               // TRUE

char c = a & b;       // 0000 0000
c = a | b;           // 0000 0101
c == 5;               // TRUE

char d = ~c;          // 1111 1010
c = c | 2;            // 0000 0111
d = d & ~(1<<3);    // 1111 0010

char e = c ^ d;        // 1111 0101
```



Useful C lore

extern myvar – variable is declared in another file

unsigned int – always positive integer.

assert(expr) – if expr is false, then you get the output
file:###:function: Assertion 'expr' failed.

NDEBUG – environment variable that removes all your asserts.

const int var – hint to the compiler that var will not change



Function pointers

You can decide at runtime which function to call.

```
#include <stdio.h>

void my_int_func(int x) {
    printf( "%d\n", x );
}

int main() {
    void (*foo)(int);
    foo = &my_int_func;

    /* call my_int_func (no need to write (*foo)(2) ) */
    foo( 2 );

    /* but if you want to, you may */
    (*foo)( 2 );
    return 0;
}
```



Function pointers – real example

```
#include<stdlib.h>

#define frand() 2.0*(rand()/(double)RAND_MAX-.5)

int Compare(const void *x, const void *y);

int main(void) {
    int i, N=10;
    float *a = (float *)malloc(N*sizeof(float));
    for(i=0;i<N;i++)
        a[i]=frand();

    qsort(a,N,sizeof(float),Compare);

}

int Compare(const void *x, const void *y) {
    if(*(float *)x > *(float *)y)
        return 1;
    else if(*(float *)x < *(float *)y)
        return -1;

    return 0;
}
```



Memory

Most modern systems provide a flat memory space to a process

Every *address* can be expressed as a 32- or 64-bit integer

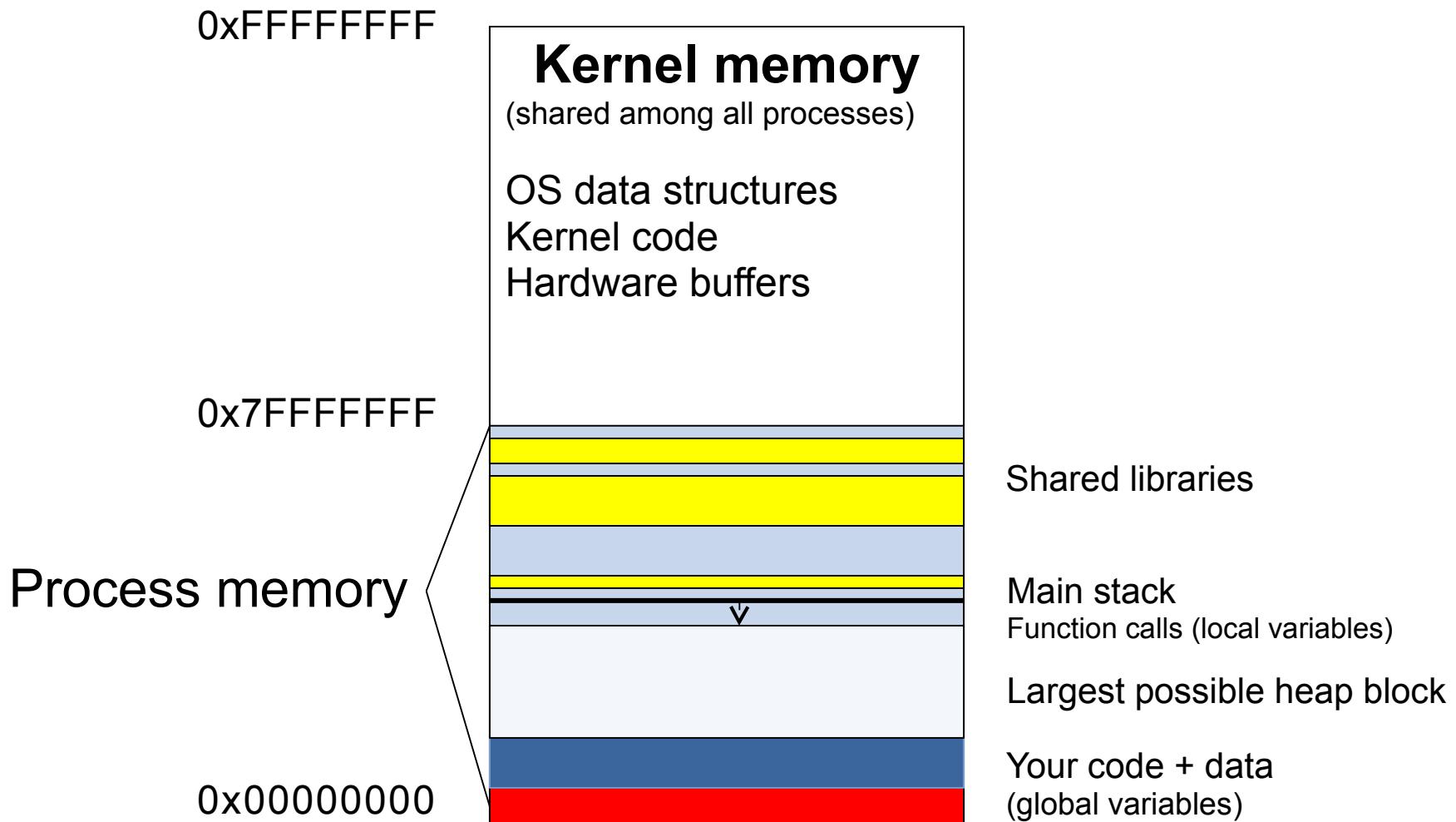
Memory is virtualized:

Process addresses are mapped to physical memory only indirectly

Why? Memory protection, flexibility



A common 32-bit process memory layout





Consequences

4 GB theoretical maximum for a 32-bit process

Expect 1 or 2 GB to be reserved by the OS

Not actual memory consumption by the kernel

Might be better if running in 64-bit OS

External libraries might use significant amounts of
memory *or address space*

Largest allocatable block of ~512 MB



Heap vs Stack

- The Stack

- Call a function, stuff pops on the stack
- Return a function, stuff pops off the stack
- *Local variables vanish at the end of their scope*

- The Heap

- User-managed
- Allocated with malloc() or calloc()
- Stays around until free() is called
- *#1 cause of programmer error*



UPPSALA
UNIVERSITET

Don't do this

```
int * dont_do_this()
{
    int a;
    a = 5;
    return &a;
}
```



Segmentation Faults

- A program tries to access a forbidden segment of memory
- Triggers the signal SIGSEGV
- Easiest to debug with a debugger (gdb or ddd)
- Common causes:
 - Using uninitialized pointers
 - Array out of bounds
 - Using a “freed” pointer
 - Dereferencing NULL pointers



malloc() and free()

- The C std library function malloc() allocates memory on the heap
- You can deallocate with free() at any time
- This lets you dynamically allocate arrays
- Also lets you **leak** memory, cause **seg faults**, generally make life **miserable**
- Next lecture: how to debug these problems



calloc()

- calloc() is like malloc() but different.
- `calloc(int num_elements, int element_size);`
- initializes elements to 0
- calloc'd memory must also be free'd



Caches

UPPSALA
UNIVERSITET

Main memory is slow:

Expect a latency of 100 cycles to transfer a byte at random

Bandwidth also limited

In 100 cycles, a single modern core can do $4 * 100$ multiplications

- $C = A * B$, reading 3200 bytes, writing 1600
- Theoretical maximum memory bandwidth might be 800 bytes for 100 CPU cycles
- Processor wants over 4x more data than it can get from memory

One instruction is often dependent on the next



Caches

UPPSALA
UNIVERSITET

How do we hide latency? How do we improve the bandwidth?

Move data into "cache" memory

Several levels with varying latency

L1 (approx. 100 kB) 1-5 cycles

L2 (256 kB-) 10-20 cycles

L3 (> 2 MB), slower

Some caches can be shared between cores

Complex synchronization if different cores write to the same memory

Caches organized into "cache lines", approx. 128 bytes



Cache optimization

An example of “hardware dependent optimization”

- Blocking:
 - Reorder data accesses to improve data locality
 - But! Modern prefetchers are *great*.
- Can be relevant to parallel applications.



Virtual memory

Not all memory bytes are created equal

Memory is organized in 4KB *pages*

Large/huge pages of 2+ MB used in large servers

A page is stored in physical RAM, or triggers a *page fault*

A page fault means that the kernel is requested to serve the correct page from some other source

"Page file", but also mapped directly to executables on disk

Large files might be "read" by memory-mapped io (mmap)



UPPSALA
UNIVERSITET

Wrap-up: C & memory basics

- Building a C program
 - Preprocess, compile, link
- C programming
 - Pointers
 - Arrays
 - Syntax
- Memory
 - Addressing
 - Heap & stack
 - Virtual memory
 - Cache



Debugging and Testing

- Bugs bugs bugs
- Highly optimized code is ugly code
- Remember your priorities:
 - Correctness
 - Flexibility
 - Performance





Debugging/Testing Code

The “80-20 rule”

“Software developers spend 80% of their time on debugging already written code and 20% on writing new code”



Testing Code

Incremental testing

Make sure each set of lines, routine, set of routines, code etc is working before the complete system is put together

Regression testing

Problem that triggers a bug is included in the test suite

Re-do all tests when changes are made



Testing Code

Unit testing

Test each function and method (unit of code)
independently

Vary the input and verify the output state

Framework libraries (JUnit and ports), coverage-checking tools

Debug builds help!

assert, assert, assert



Testing Code

“Extreme Programming”

Construct tools for testing and set up test data
before you start writing a new code

Automate the testing and add new tests as you
refine and find bugs

When you think everything is working, rerun the
complete set of tests

A version of this: “Nightly builds”



Unit tests

- Can be difficult to implement
- Enforce forethought, planning, *structure*
- Key points to test:
 - Erroneous cases – When unit gets wrong input
 - Pathological cases – When unit gets bad or strange input
- Key things to consider:
 - How can a correct output be characterized?
 - How can you catch an incorrect result?



Assert

UPPSALA
UNIVERSITET

- **Ex.** `assert(size <= LIMIT);`
- **#define NDEBUG to suppress asserts**
 - or compile with `-DNDEBUG`
- **Use to check:**
 - preconditions – Is my function receiving valid parameters?
 - postconditions – Did my function produce reasonable results?



Localizing bugs

UPPSALA
UNIVERSITET

Print statements (Not stone age!)

```
printf("Entering routine xxx-yy\n");  
printf("Before main loop in xxx  
algorithm\n");  
printf("%s: %d\n", __FILE__, __LINE__);
```



Localizing bugs

Print contents of variables and data structures

Write print routines for displaying the contents of your data structures in a readable format.

Print data in condensed form, e.g. the norm of a vector instead of all the vector entries.

Use the more or less tools for examining output, or use a text editor.

Display results graphically.



Localizing bugs

Compare and contrast

A working program in another language, on another system etc is a very valuable resource

BUT: Can the "working program" be trusted?

Subproblems, or simplified problems, can be easily implemented in e.g. Matlab

Use the same test data for both codes!

Regression testing by giving identical input to "known good" stable version



Localizing bugs

Things to try if you can't find a bug:

- Identify code areas that do NOT contain the bug
- Remove chunks of code until bug disappears
- Fix hacks & kluges, restructure code
- Rewrite the buggy code from scratch
- Write code on paper and execute



Debugging

Find causes of errors in your code

Executes your code like an interpreter

- You can step through your program

- Set breakpoints

- Print and set variables

- Catch run-time exceptions

Two standard debuggers

- GNU gdb

- dbx



Compiling for debugging

To be able to refer to names and symbols in your source code it must be stored in the executable file

Standard flag “-g”

A “debug build” can also enable lower optimization (-O0), debug mode in libraries (asserts)

Without “-g” you can only debug using virtual addresses



UPPSALA
UNIVERSITET

Using GDB

```
$ gdb ./pi
(gdb) break main
Breakpoint 1 at 0x8048380: file pi.c, line 10.
(gdb) run
Starting program: /home/sverker/test/trunk/pi
Breakpoint 1, main (argc=1 ,...) at debug_me.c:19
(gdb) print pi
$2 = 4.8542713620543657e-270
(gdb) next
11          printf("Pi is %lf\n",pi);
(gdb) print pi
$3 = 3.1415926535897931
(gdb) list
6          int main(void) {
7
8          double pi;
9
10         pi = 4.0*atan(1.0);
11         printf("Pi is %lf\n",pi);
12
13         return 0;
14     }
(gdb) where
#0  main () at pi.c:11
```



Some GDB Commands

step [<i>count</i>]	Next line of code, step into functions
next [<i>count</i>]	Next line of code, execute function
list	Print surrounding source
where	Print call hierarchy (stack)
print <i>expr</i>	Print the value of a variable
break	Set breakpoints (many options)



More on GDB

Type **help** in the GDB prompt

help breakpoints, help running

Check out the gdb man page

GDB can also attach to already running processes owned by you

There is a graphical interface to GDB called the “Dynamic Data Debugger” DDD

```
$ ddd ./a.out
```

Allows you to visualize data and breakpoints



Core files

The OS can be configured to dump the virtual address space of a process to a **core** file

```
> ulimit -c unlimited
```

Typically this happens at a fatal error

“Segmentation fault (core dumped)”

This file can be used to debug your program post-mortem

Especially useful is debug symbols are present

```
> gdb a.out core
```



Memory bugs

`malloc()` and `free()` – *with friends like these, who needs enemies?*

Can cause problems “far from” where the bug really is

Common to only occur for special data sets

May occur in a seemingly non-deterministic way

Influence from the environment (system load etc)

May be found in codes that were supposed to be “tested and verified”, e.g. when porting to another system



Memory bugs

Unallocated memory

```
double *A;  
...  
for (i=0; i<n; i++)  
    A[i] = 0.0;
```

Can cause the execution to stop when the variable is used or an attempt to deallocate the memory is made

Most compilers can check for unallocated memory



Memory bugs

Overwriting or “overreading” memory

Common error in C and Fortran: Using arrays outside the array bounds

- double * A = (double*) malloc(N*sizeof(double));
-
- for(i=0; i <= N; i++)
- Diff[i]=(A[i+1]-A[i])/h;
-

May result in erroneous results or program crash (page fault in invalid page)

Program crash may occur later, e.g. when calling memory allocation routine

Many compilers can generate code that checks array bounds during execution (add option for this) . NOTE: This reduces performance!

Handled in Java and e.g. Pascal (Runtime error)



Memory bugs

Stack buffer overrun/overflow

Special case of memory overreading/overwriting where you run out of stack space.

Caused by:

- Too many very large stack variables
- Too many (recursive) function calls



Memory bugs

Dangling pointers

A pointer to a piece of memory that once was allocated to a variable, but then was deallocated

Common error when copying and deallocating derived types with dynamically allocated fields

Another common error: returning a **pointer to a local** variable

Hard to debug! Some debugging tools may help you

0xBAADF00D, 0xDEADBEEF



Memory bugs

Memory leaks

Allocated memory is lost without deallocation

The program runs out of memory for large problems, many iterations ...

Hard to detect. Use e.g. `top` to monitor memory usage

Hard to debug. Instrumented system call libraries may give some information; allocation “tagging”.

Valgrind memcheck can help

Handled in e.g. Java, and other languages with built-in garbage collection



Memory bugs and testing

- Tests often miss memory bugs or
- Tests only *sporadically* catch memory bugs
- “Heisenbugs” stop failing when run in gdb
 - pointer initialization usually the culprit
- What to do?
 - Check memory addresses
 - Isolate program components
 - Use Valgrind



Valgrind

UPPSALA
UNIVERSITET

- Use of uninitialized memory
- Reading/writing memory after it has been free'd
- Reading/writing off the end of malloc'd blocks
- Reading/writing inappropriate areas on the stack
- Memory leaks -- where pointers to malloc'd blocks are lost forever
- Mismatched use of malloc/new/new [] vs free/delete/delete []
- Overlapping src and dst pointers in memcpy() and related functions
- Some misuses of the POSIX pthreads API

```
% valgrind --tool=memcheck program_name
...
==18515== malloc/free: in use at exit: 0 bytes in 0 blocks.
==18515== malloc/free: 1 allocs, 1 frees, 10 bytes allocated.
==18515== For a detailed leak analysis, rerun with: --leak-check=yes
```



Valgrind caveats

Valgrind will NOT perform bounds checking on static arrays
(allocated on the stack)

X 2 memory usage

Program will have terrible performance

Valgrind may not find all the errors – especially those that don't happen every time.



Computational bugs

Floating-point problems

Cancellation

Precision (using numbers of very different magnitude)

Accuracy of intrinsic functions and constants (embedded code, GPUs)

Strict equality is always dangerous. Always use:

`fabs(x-y) < EPSILON`, *not* `x == y`

Ordering is crucial!

Algorithms used near their limit of applicability

Near-indefinite matrix

Near-singular matrix

Near-degenerate geometry in computational grid

Stress testing: Include “extreme inputs” in test data set



Computational bugs

UPPSALA
UNIVERSITET

Overflow/underflow errors

Ex. get middle array element in a segment of a very large array:

```
int middle = (left + right)/2; //DANGEROUS
```



UPPSALA
UNIVERSITET

Logical bugs and typos

Just about anything...

Mixing up negation

Assigning when intending equality

Misusing a library, or another part of your code

"That case will *never* arise"

"I use this argument in a clever way to indicate two things"



What to do?

Tedious code is tedious to test

Respect compiler warnings (use `-Wall`)

Reusing code is good, *copying* code is a slippery slope

Use the libraries of others:

Data structures, common routines & operations

e.g. Boost (www.boost.org)



Performance

UPPSALA
UNIVERSITET

- Measuring
- Analyzing
- Reporting



What is performance?

Computers are systems of components that interact with each other.

Complex and hard to model analytically

Performance analysis is an experimental discipline of computer science involving:

- Measurement
- Interpretation
- Communication
- Cost – money, time, energy?



Goals of performance analysis

UPPSALA
UNIVERSITET

Compare alternatives when buying computers

Determining the impact of a feature when
designing a system or upgrading

Set runtime expectations

Performance debugging/optimizations



Basic methodologies

- **Measurements**
 - Not very general
 - Hard to change system parameters
 - Time consuming
 - Intrusion of probes
- **Simulation**
 - Easy to change system parameters
 - Hard to model every detail
 - Needs validation
- **Analytical modeling**
 - Hard



Measuring execution time

UPPSALA
UNIVERSITET

Why not just use a stopwatch?

In a modern (time-sharing) operating system your code may not execute the entire time.

Processes are interrupted by I/O, kernel activity and other processes

The **wall clock time**, is the classic stopwatch time

The **CPU time** is the accumulated time the process actually ran on a CPU.

This can further be divided into **system** and **user** CPU time



Unix time Command

```
$ time make osevent
gcc -O2 -Wall -g -c clock.c
gcc -O2 -Wall -g -c options.c
gcc -O2 -Wall -g -c load.c
gcc -O2 -Wall -g -o osevent osevent.c . .
0.820u 0.300s 0:01.32 84.8%      0+0k 0+0io 4049pf+0w
```

0.82 seconds user time

Corresponds to 82 timer intervals (ticks)

0.30 seconds system time

Corresponds to 30 timer intervals (ticks)

1.32 seconds wall time

84.8% of total was used running these processes

$(.82+0.3)/1.32 = .848$



“Time” on a Computer System



real (wall clock) time



= **user time** (*time executing instructions in the user process*)



= **system time** (*time executing instructions in kernel on behalf of user process*)



= **some other user's time** (*time executing instructions in different user's process*)



+ + = **real (wall clock) time**

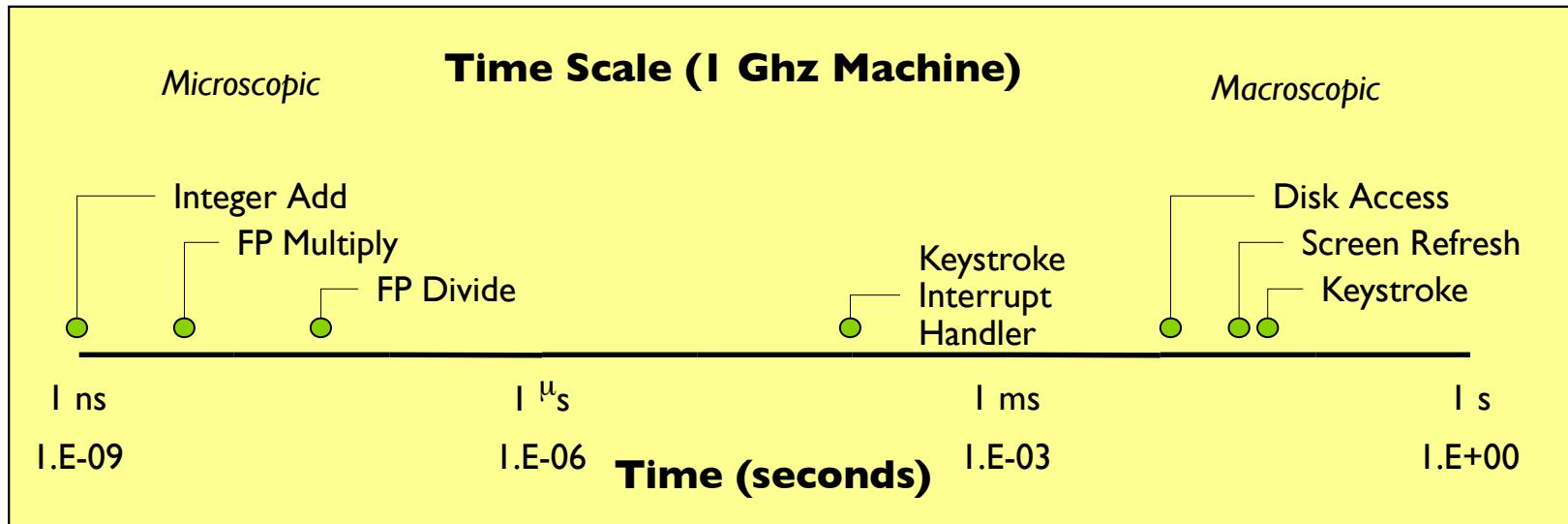
We will use the word “time” to refer to user time.



cumulative user time



Computer Time Scales



Two Fundamental Time Scales

Processor: $\sim 10^{-9}$ sec.

External events: $\sim 10^{-2}$ sec.

• Implication

- Can execute many instructions while waiting for external event to occur
- Can alternate among processes without anyone noticing



Measurement Challenge

How Much Time Does Program X Require?

CPU time

- How many total seconds are used when executing X?
- Measure used for most applications
- Small dependence on other system activities
 - NOT independent, though

Actual (“Wall”) Time

- How many seconds elapse between the start and the completion of X?
- Depends on system load, I/O times, etc.

Important Factors

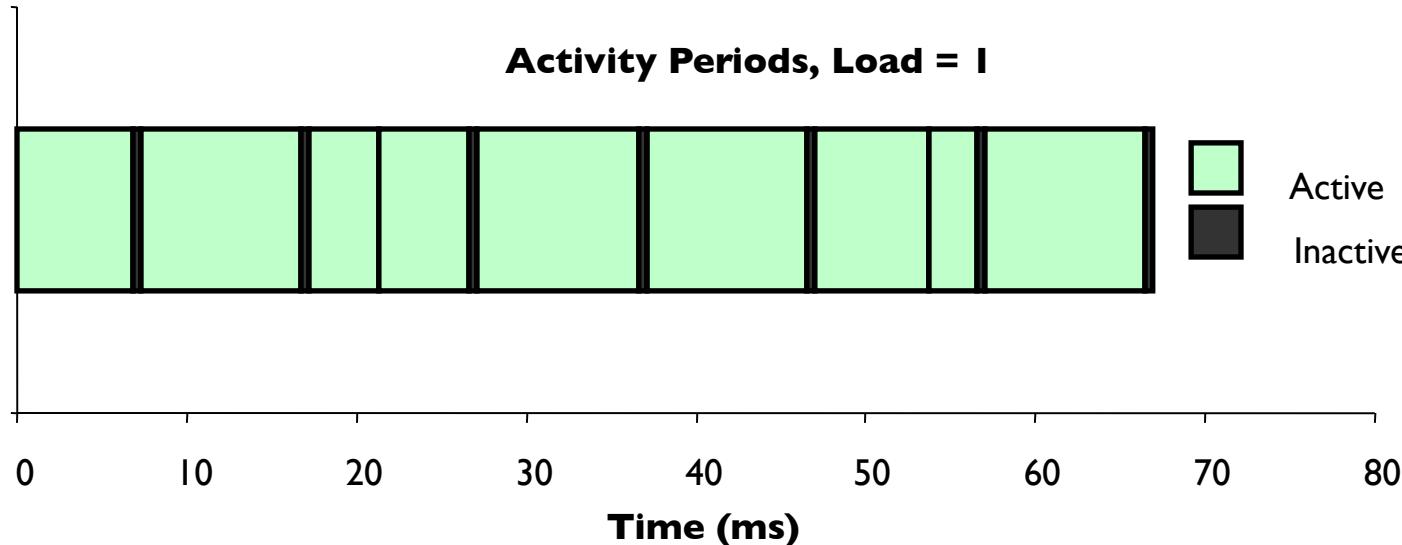
How does time get measured?

Many processes share computing resources

- Transient effects when switching from one process to another
- The effects of alternating among processes become noticeable



Activity Periods: Light Load



Most of the time spent executing one process

Periodic interrupts, e.g. every 10ms

Keep system from executing only one process

- Other interrupts
- Due to I/O activity
- Inactivity periods
- System time spent processing interrupts
- ~250,000 clock cycles



Interval Counting

UPPSALA
UNIVERSITET

OS Measures Runtimes Using Interval Timer

Maintain 2 counts per process

- User time
- System time

Each time you get a timer interrupt, increment counter for executing process

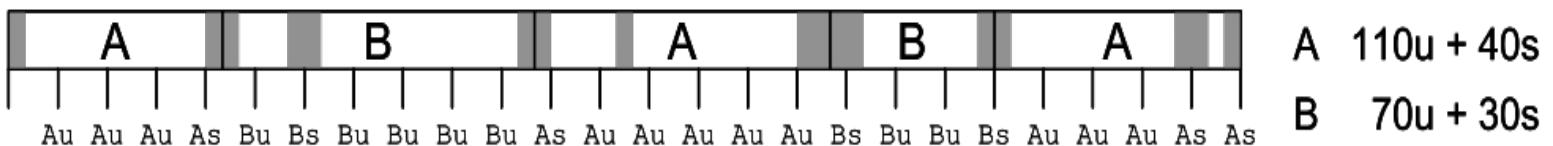
- This is called a clock tick
- User time if running in user mode
- System time if running in kernel mode



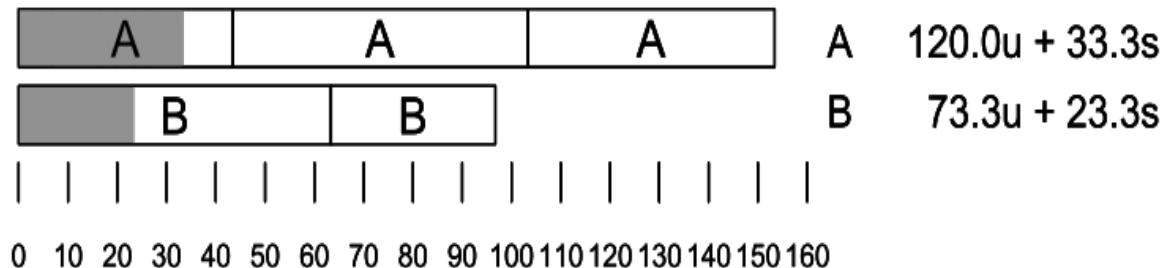
Interval Counting Example

UPPSALA
UNIVERSITET

(a) Interval Timings

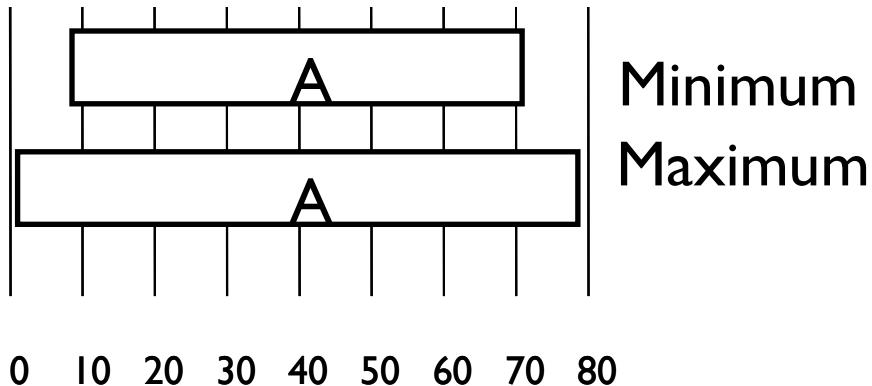


(b) Actual Times





Accuracy of Interval Counting



- **Estimated time = 70ms**
- **Min Actual = $60 + \varepsilon$**
- **Max Actual = $80 - \varepsilon$**

Worst Case Analysis

Timer Interval = T

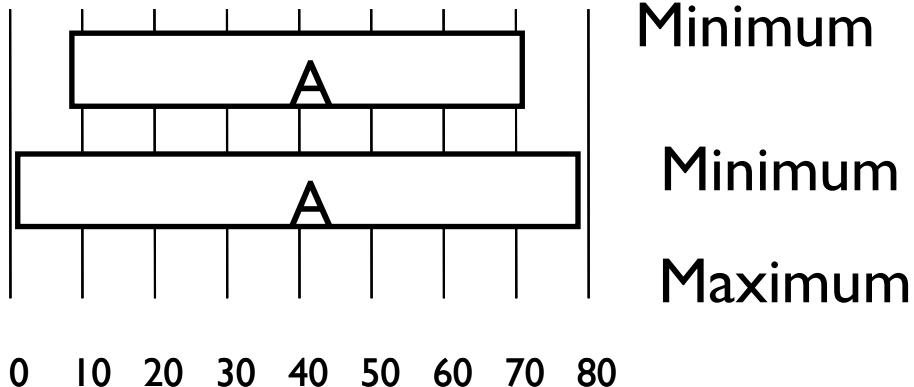
Single process segment measurement can be off by $\pm T$

No bound on error for multiple segments

- Could consistently underestimate, or consistently overestimate



Accuracy of Int. Cntg. (cont.)



Minimum
Minimum
Maximum

- **Computed time = 70ms**
- **Min Actual = $60 + \varepsilon$**
- **Max Actual = $80 - \varepsilon$**

Average Case Analysis

Over/underestimates tend to balance out

As long as total run time is sufficiently large

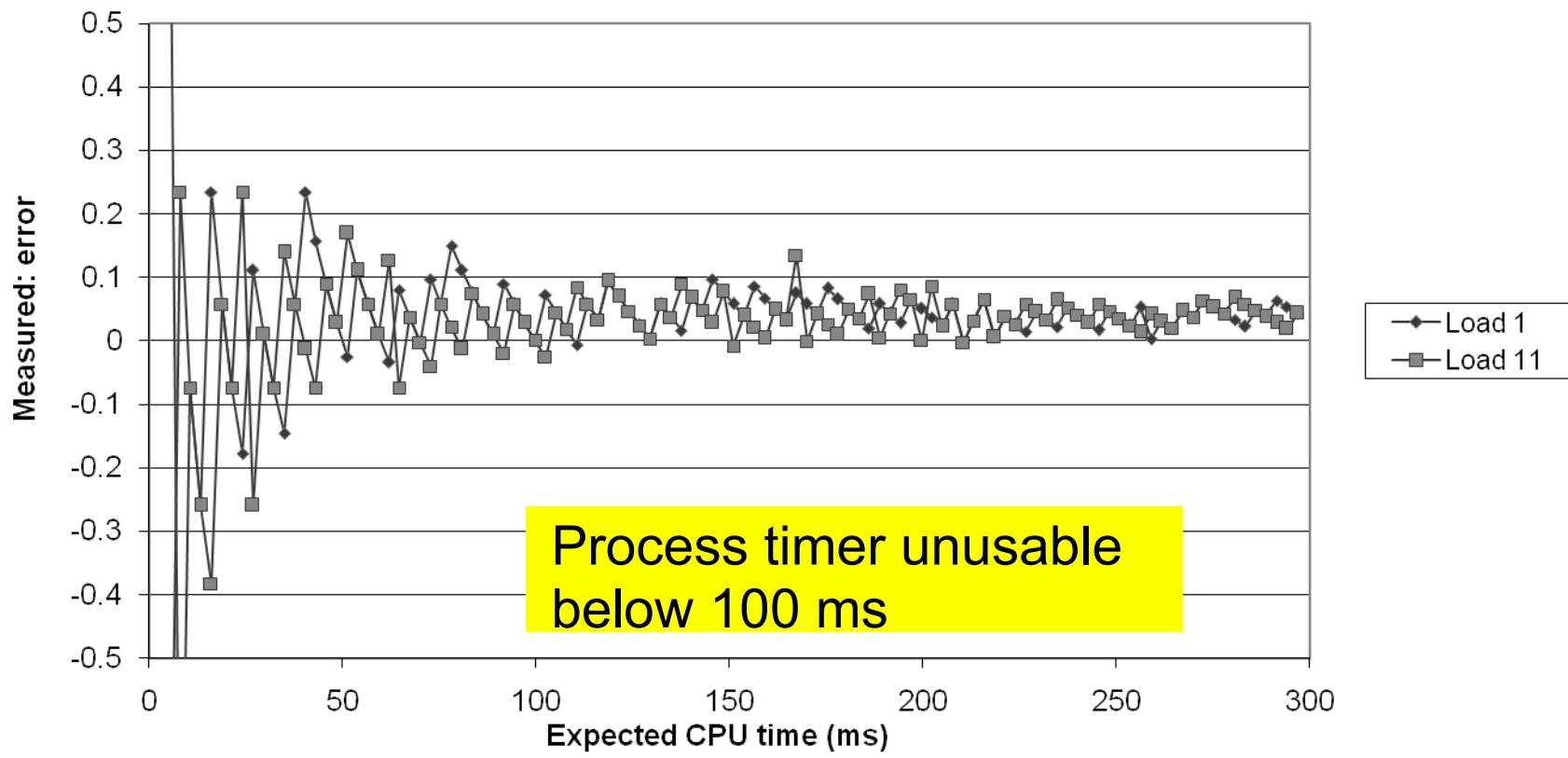
- Min run time ~1 second
- 100 timer intervals

Consistently miss overhead due to timer interrupts



Accuracy of process timer

Intel Pentium III, Linux, process timer





Cycle Counters

UPPSALA
UNIVERSITET

Most modern systems have built-in registers that are incremented every clock cycle

- Very fine grained
- Sometimes maintained as part of process state
 - In Linux, counts elapsed global time
- Problems in SMP, dynamic frequency scaling

Special assembly code instruction to access
On (recent model) Intel machines:

- 64 bit counter.
- RDTSC instruction



Counters and overflow

UPPSALA
UNIVERSITET

Consider a 32-bit unsigned int

Ticks are microseconds (10^{-6} s):

- Maximum time = 2^{32} cycles = 1.2 h

Ticks are milliseconds (10^{-3} s):

- Maximum time = 2^{32} ms = 49 days

A 64-bit unsigned int in microseconds gives a maximum time of > 500,000 yr



Multitasking Effects

UPPSALA
UNIVERSITET

Cycle Counter Measures Elapsed Time

Keeps accumulating during periods of inactivity

- System activity
- Running other processes

Key Observation

Cycle counter never underestimates program run time

Possibly overestimates by large amount



UPPSALA
UNIVERSITET

Tools for analyzing performance

“What is taking so much time?”

“Which section of my program should be made better?”

- Profilers
 - gprof
 - fancy profilers like Shark
- Timing utilities
 - time.h



Accessing timer in code

The process interval timers can also be accessed from within your code

times(2) will return the number of clock ticks since an arbitrary point in the past (userCPU+sysCPU). Use `sysconf(_SC_CLK_TCK)` to determine the number of clock ticks per second.

clock(3) will return userCPU time used so far in clocks. Use `CLOCKS_PER_SEC` to convert to seconds

<sys/times.h>:

```
struct tms {  
    clock_t tms_utime; /* user time */  
    clock_t tms_stime; /* system time */  
} /  
...  
};
```

```
clock_t times(struct tms *buf);
```

<time.h>:

```
#define CLOCKS_PER_SEC something  
clock_t clock(void);
```



Timing, gettimeofday()

Standard UNIX interval timer

Returns Wall-clock time

“Microsecond” resolution

```
#include <sys/time.h>
#include <time.h>

int gettimeofday(struct timeval *tv, struct timezone *tz);
int settimeofday(const struct timeval *tv , const struct timezone *tz);

<sys/time.h>:
struct timeval {
    time_t tv_sec; /* seconds */
    suseconds_t tv_usec; /* microseconds */
};
```



UPPSALA
UNIVERSITET

gettimeofday(), example

```
#include <time.h>
#include <sys/time.h>

static struct timeval start_time, end_time;
.....
double start_count, end_count, elapsed_time;

gettimeofday(&start_time,NULL);
/* Stuff to be measured */
gettimeofday(&end_time,NULL);

start_count = (double)start_time.tv_sec + 1.e-6 * (double) start_time.tv_usec;
end_count = (double)end_time.tv_sec + 1.e-6 * (double) end_time.tv_usec;
elapsed_time = (end_count - start_count);
printf("The total elapsed time is: %f seconds\n", elapsed_time);
```



Reporting computational experiments

Indicators should if possible be independent of the experiment.

Traditional indicators are e.g.

CPU time

Numerical accuracy

Number of iterations

Scope, applicability

Portability

Storage requirement

Operation count (flops)

Relative indicators or ranking can be misleading



Reporting computational experiments

Presentation of algorithms

Complete description of the algorithm

Specification of the domain of applicability of the algorithm

If possible: Computational complexity, in time and space

If relevant: Convergence results

If relevant: Rate of convergence, order of accuracy



Reporting computational experiments

Presentation of implementation

- Programming language used
- Compiler name and options used
- Computer environment, system and OS (*not* “a Linux machine with an AMD processor”)
- Input data
- Settings

Is there a “gold standard”?

Run it in your specific environment!



Reporting computational experiments

Presentation of experiments

- Objective of the experiment
- Description of problem generator/input data set
- (Why) Are the results general?



Reporting computational experiments

Presentation of results, e.g. CPU time:

- Description of how the timings were produced
- What parts of the code are included in the measurement?
- What is the variability of the timings?
- How is the variability handled?



Guidelines for measuring execution time

Use a high resolution timer, such as `gettimeofday()`

Check the amount of time your program spends in the OS using the `time` command

Measure at least 6-7 runs

Pick the shortest time as a representative or an average

Beware of outliers!

If you have room, report all samples



Measuring FLOPS

1. Count the number of operations in your source code
2. Time the operation
3. Calculate the rates

Example,

Matrix($n*k$)-vector product: $n*((k-1)+n)$



Scalar Product, example

```
sum = 0.0;  
  
for( i=0; i<N; i++ ) {  
  
    sum += a[i]*b[i];  
  
}  
  
printf("Scalar product of a and b is %f\n",sum);
```

~N additions
~N multiplications

~2*N FLOPS



Profiler Basics

UPPSALA
UNIVERSITET

Generate an execution profile, by sampling the execution

Stop the program at regular intervals and sample what the program is doing

More insights can be obtained using instrumentation

Adds bookkeeping code

Gives you a view of where in your code time is spent

Can be on functional level (prof or gprof) or basic block level (tcov)



Instrumentation

Manual

You add it yourself (e.g. static variable counting the number of calls)

Timing printouts in your code

Compiler-assisted

Compiler inserts sampling code

Used by “gprof” -pg and “prof” -p

Runtime instrumentation

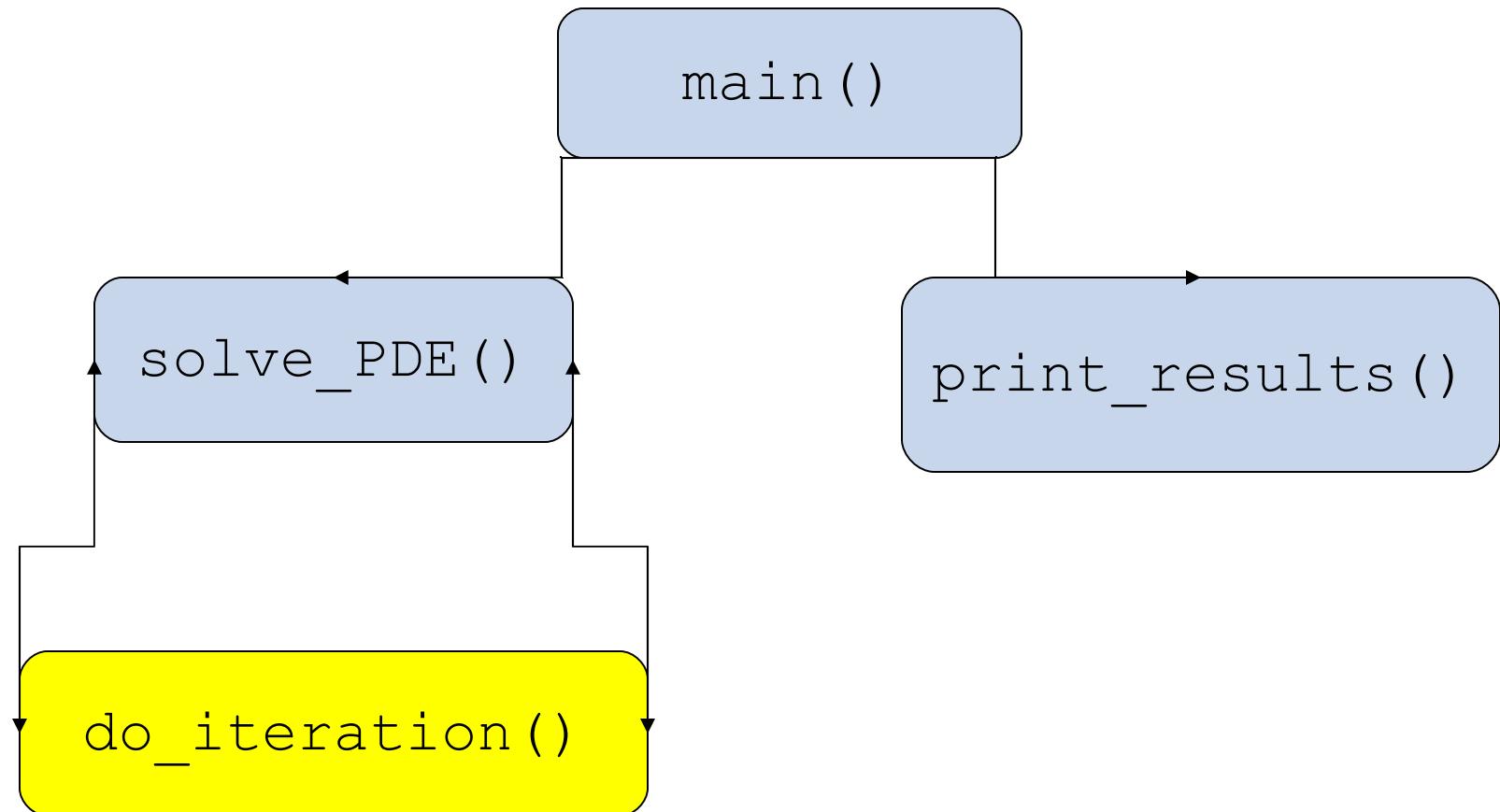
No extra code, but still profiling overhead

Intel Vtune, Sun Analyzer, Valgrind



Call Graph

UPPSALA
UNIVERSITET





Inclusive/Exclusive CPU time

A call-graph is composed of functions

Typically, we have parents and children for each parent.

Inclusive CPU time is the time spent in a parent and all its children

Exclusive (self) CPU time is the time spent only executing code in the parent



prof

UPPSALA
UNIVERSITET

Simplest tool

Compile and link with -p

Run your code

mon.out is produced

Run \$ prof <executable>

Results in a short list of your most expensive functions



prof Example

UPPSALA
UNIVERSITET

%Time	Seconds	Cumsecs	#Calls	msec/call	Name
32.3	11.58	11.58	1	11580.	matrix_Determinant
25.0	8.96	20.54	140825570	0.0001	_mcount
12.7	4.57	25.11			_libc_threads_interface
7.9	2.82	27.93	37384625	0.0001	_free_unlocked
7.1	2.54	30.47	28671511	0.0001	__pow
5.9	2.13	32.60	37384627	0.0001	malloc
4.1	1.46	34.06			.div
2.2	0.78	34.84	37384625	0.0000	free
1.6	0.59	35.43			_libc_pthread_getspecific



gprof

UPPSALA
UNIVERSITET

Slightly more advanced profiler

Compile with and link with -pg

Run your program

Generates gmon.out

```
$ gprof <executable>
```

Also gives you break-downs per function, call-graph



gprof Example

UPPSALA
UNIVERSITET

%	cumulative	self		self	total		
time	seconds	seconds		calls	ms/call	ms/call	name
65.8	72.99	72.99					internal_mcount [1]
10.4	84.50	11.51	1	11510.00	36449.99		recursive_det [5]
6.4	91.56	7.06	108505237	0.00	0.00		.umul [8]
2.5	94.33	2.77	19958400	0.00	0.00		det22 [6]
2.2	96.78	2.45	37384625	0.00	0.00		_free_unlocked [14]
2.0	98.96	2.18	74769266	0.00	0.00		mutex_unlock [12]
1.7	100.88	1.92	28671511	0.00	0.00		__pow [15]
1.7	102.74	1.86	74769266	0.00	0.00		mutex_lock [13]
1.6	104.49	1.75	37384631	0.00	0.00		_malloc_unlocked <cycle 1> [11]
1.3	105.93	1.44					_mcount (674)
1.1	107.11	1.18	149538532	0.00	0.00		_return_zero [17]
1.0	108.18	1.07	17426232	0.00	0.00		cleanfree [16]
0.8	109.03	0.85	37384627	0.00	0.00		malloc [7]
0.7	109.84	0.81	37384625	0.00	0.00		free [9]
0.5	110.42	0.58	17426201	0.00	0.00		_salloc <cycle 1> [18]
0.4	110.88	0.46	17426227	0.00	0.00		realfree [19]
0.0	110.88	0.00	121	0.00	0.00		rand [39]



Gprof, individual functions

called/total parents					
index	%time	self	descendents	called+self	name
index					
					called/total children
					28671511 recursive_det [5]
		11.51	24.94	1/1	matrix_Determinant [4]
[5]	33.3	11.51	24.94	1+28671511	recursive_det [5]
		2.77	5.19	19958400/19958400	det22 [6]
		0.85	6.47	37384623/37384627	malloc [7]
		0.81	5.06	37384623/37384625	free [9]
		1.92	0.00	28671511/28671511	__pow [15]
		1.87	0.00	28671511/108505237	.umul [8]
				28671511	recursive_det [5]



Profiler-Assisted Optimization

Use a profiler to check where your code spends its time

Spend your optimization efforts on “hotspots”

Remember: premature optimization is the root of all evil

Be aware of the sampling resolution of your profiler



Profiler-Guided Optimization

PGO supported in Intel, Microsoft (some ed.) and Portland compilers

Allows:

- Increase code memory locality

- Smart decisions on inlining

- Improved branching

- Good experiments just got a lot harder



Performance bugs

When things don't make sense, consider:

- Change of algorithms in libraries.
 - E.g. FFT for prime number length vectors is performed as matrix-vector multiplication ($O(n^2)$ instead of $O(n \log(n))$)
- Complexity hidden in code layers or library calls
- Memory access patterns
- (Unhelpful) optimizations
- Unnecessary or erroneous “NaN”, “Inf”, denormalized numbers etc



Take-home message

- Use profilers and timing utilities to:
 - Understand what the computer is doing
 - Expose performance bottlenecks
 - Target your optimization effort
 - Target your parallelization effort
- Do not optimize code before it is correct



Performance Analysis

UPPSALA
UNIVERSITET

- Read Roofline.pdf



UPPSALA
UNIVERSITET

High Performance Computing -- Code Optimization II

Code optimization:

- Reducing number of operations
- Reducing memory usage
- Making your program run faster

Elias Rudberg, room P2440

elias.rudberg@it.uu.se



UPPSALA
UNIVERSITET

Optimization techniques

- Choose good **data structures**
- Reduce number of operations
- Use cheap operations – strength reduction
- Avoid too many small function calls – **inlining**
- Improve **data locality**
- Use compiler **optimization flags**
- Help the compiler! Use **const** and **restrict**



Optimizing Compilers

- Most compilers support optimizations
- Standard optimization flag is -O
- Measure performance using different optimization flags. "highest level" of opt. not always best!
- For example, with gcc compiler, try compiler optimization flags -O2, -O3, -funroll-loops and -funroll-all-loops
- See “man gcc” for more



Limitations of Optimizing Compilers

- Behavior that may be obvious to the programmer can be obfuscated by languages and coding styles
- Most analysis is performed only within functions -- whole-program analysis is too expensive in most cases
- When in doubt, the compiler must be conservative



Compiler versions matter!

- Check which version of the compiler you are using:

```
[eliasr@kalkyl3 ~]$ gcc --version
gcc (GCC) 4.4.6 20110731 (Red Hat 4.4.6-3)
Copyright (c) 2010 Free Software Foundation, Inc.
```

- Optimizations may be done differently depending on compiler version
- When reporting performance results, include information about compiler version and used optimization flags



Check what the compiler is doing

- The gcc compiler flag -S can be used to get an assembler code file:

```
gcc -O2 -S testfun.c
```

--> produces assembler code file testfun.s

- Looking at the assembler code can help understanding the effects of compiler optimizations



Generating assembly code

```
int testfun(int x, int y, int z)
{
    int t1 = 2 * x;
    int t2 = 55 * x * y;
    int t3 = 77 * y * z;
    return t1 + t2 + t3;
}
```

gcc -O -S code.c

--> code.s

```
testfun:
    imull $55, %edi, %eax
    imull %esi, %eax
    leal  (%rax,%rdi,2), %edi
    imull $77, %esi, %esi
    imull %esi, %edx
    leal  (%rdi,%rdx), %eax
    ret
```



Why do function calls give overhead?

Status before function call saved on stack, to be restored after function call

Parameters passed via stack



Function calls

- A function typically has input and output ***arguments*** and local variables
- To continue execution at correct location after function call, the ***return address*** is needed
- Storage for these purposes are usually allocated on a ***stack***



Stacks

UPPSALA
UNIVERSITET

- Works like a stack of paper
- Two operations:
- ***Push*** (place something on the top)
- ***Pop*** (remove something from the top)
- Last-in-first-out -- LIFO



Stacks and function calls

- Most machines push the return address onto the stack before doing the call
- After this the program counter is set to the address of the subroutine
- At the end of the subroutine, the return address can be popped from the stack



Doing a function call

- Subroutines typically need to use many registers to do things efficiently
- Before a call, register contents need to be saved to memory – typically pushed onto the stack
- Next, push the return address
- Finally, push the arguments onto the stack
- Now the subroutine can be executed!
- When we get back from the subroutine, we can pop the saved register contents from the stack



Function calls, example

```
double mainfun(params)
{
    ...
    double result = compute(x, y);
    ...
}

double compute(int x, int y) {
    ...
    tmp1 = funA(x, 2);
    tmp2 = funB(y, 3);
    ...
}

double funA(int a, int b) {
    ...
}
```



Avoid too many "small" function calls

Inlining:

- Saves function call overhead
- Same as preprocessor macro but with type checking
- C99 (and gcc) has an `inline` keyword
- (can also be controlled using flags)

gcc: `-finline-functions`

```
inline int get_num_rows(Matrix m) {  
    return m->num_rows;  
}
```



Effects of inlining

Effects of inlining:

- The function code is appearing in multiple places
- Simple inlining is only possible if the function is defined within the same compilation unit -- Put candidates for inlining in header files
- Too much inlining can give too large code segments, can be bad for performance.

```
inline int get_num_rows(Matrix m) {  
    return m->num_rows;  
}
```



Reduce frequency with which a computation is performed

- If it will always produce the same result
- Loop invariant
- Especially moving code out of the innermost loop
- Not only full lines of code

```
for (i = 0; i < n; i++)
  for (j = 0; j < n; j++)
    a[n*i + j] = b[j];
```

```
for (i = 0; i < n; i++) {
  int ni = n*i;
  for (j = 0; j < n; j++)
    a[ni + j] = b[j];
}
```



Strength Reduction

- Replace a costly operation with a simpler one
- Shift, add instead of multiply or divide
- Recognize sequence of products, turn into addition
- Replace divisions by multiplication with the reciprocal (computed once)
- Replace `pow()` function with an algebraic expression
(Compiler may not know what `pow` does, can modify a, i or j)



Strength Reduction, example

```
for(i=0; i<N; i++) {  
    for(j=0; j<M; j++) {  
        a[i][j] = (a[i+1][j]-a[i-1][j])/pow(dx, 2.0);  
    }  
}
```

```
temp = 1.0/(dx*dx);  
for(i=0; i<N; i++) {  
    for(j=0; j<M; j++) {  
        a[i][j] = temp*(a[i+1][j]-a[i-1][j]);  
    }  
}
```



Share Common Subexpressions

- Reuse portions of expressions
- Compilers often not very sophisticated in exploiting arithmetic properties

```
up = val[(i-1)*n + j];
down = val[(i+1)*n + j];
left = val[i*n + j-1];
right = val[i*n + j+1];
sum = up + down + left + right;
```

```
int inj = i*n + j;
up = val[inj - n];
down = val[inj + n];
left = val[inj - 1];
right = val[inj + 1];
sum = up + down + left + right;
```



Function side effects are *bad*

- A function side effect occurs when you modify something other than the return value of the function
 - Global variables
 - Function arguments
- Function side effects are bad for flexibility, and bad for performance.



More on function side effects

```
int func1(x) {  
    return f(x)+f(x)+f(x)+f(x);  
}
```

```
int func2(x) {  
    return 4 * f(x);  
}
```

- Are these two equivalent?
Do you know? Does the compiler know?



Side effects in loops

```
char *str = "Hello World!";
int i;
for (i = 0; i < strlen(str); i++)
{
    if (str[i] == '!') str[i] = '?';
}
```

- What's the complexity?
- Can you improve it?
- Can the compiler improve it?
- Does the compiler know the implementation of `strlen`?



UPPSALA
UNIVERSITET

Summary

- Choose data structures carefully
- Explore compiler optimization flags
- Avoid too many small function calls
- Move computations out of inner loops whenever possible
- Avoid function side effects



High Performance Computing -- Code Optimization III

- Function side effects
- Important keywords: **const** and **restrict**
- Hardware issues: **cache** memory, data **locality**,
pipelines
- How to make your program run faster on
modern hardware

Elias Rudberg, room P2440
elias.rudberg@it.uu.se



UPPSALA
UNIVERSITET

Optimization techniques

- Choose good **data structures**
- Reduce number of operations
- Use cheap operations – strength reduction
- Avoid too many small function calls – **inlining**
- Improve **data locality**
- Use compiler **optimization flags**
- Help the compiler! Use **const** and **restrict**



Side effects in loops

```
char *str = "Hello World!";
int i;
for (i = 0; i < strlen(str); i++)
{
    if (str[i] == '!') str[i] = '?';
}
```

- What's the complexity?
- Can you improve it?
- Can the compiler improve it?
- Does the compiler know the implementation of `strlen`?

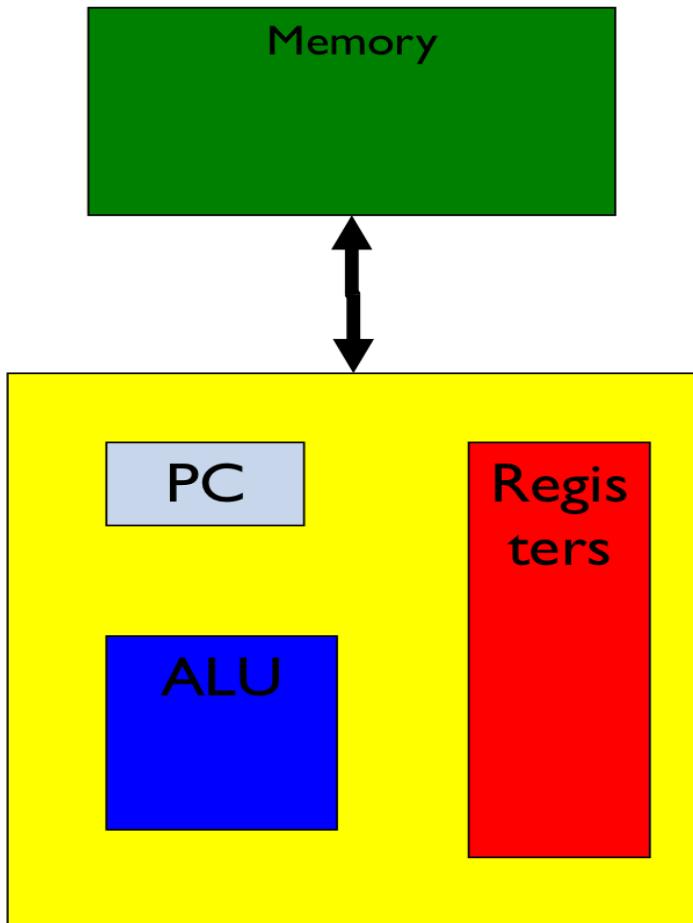


The "const" keyword

- In ANSI C you add the keyword “const” to arguments that are read-only
 - The memory object will never be written to
- Can also be used for variables to make them into constants
- Helps the compiler in analysis



The von Neumann Model



- Fetches instructions from the memory referenced by the program counter (PC) and computes results based on the data the instruction specified
- The Arithmetical-Logic Unit (ALU) does the actual work
- In this simple model access times to memory are uniform



Aliasing

UPPSALA
UNIVERSITET

- Any pointer of unknown origin can reference a value that is accessed through another variable
- Any pointer might be used as an array
 - Of unknown size
- Multiple “aliases” for the same memory location
- Makes compile-time optimization very hard



The "strict aliasing" rule

- Default mode in C99 and recent GCC
- Pointers of different types should not refer to the same memory
- Not a problem until you start being “clever”
- Significant compilation benefits



The "restrict" keyword

- restrict is another element of the C99 standard
- Available in many C/C++ compilers, including recent gcc (sometimes as `__restrict`)
- Within this context, any memory locations accessed by a restricted (pointer) variable will only be accessed through that pointer
- E.g. `strcpy(char * restrict dest, char * restrict from)`



UPPSALA
UNIVERSITET

Hardware considerations





Hardware considerations

Basic operation:

- Modern computers are of LOAD-STORE type
- These machines store operands in “registers”
- A “register”: small scratch memory very close to the arithmetical units
- Data must be loaded from memory into a register, operated upon, and then stored back



Registers

UPPSALA
UNIVERSITET

- Registers are a scarce resource
 - – Store words (32/64 bit)
 - – Special registers for floating-point, SIMD (128/256/512 bits!)
 - • The compiler tries to maximize the usage of the registers
 - Called register allocation
 - If you run out of registers, you must temporarily store results back to memory and then retrieve them again
 - Register spill
 - Degrades performance



Registers in C

- There are two keywords that control register allocation from C
- The **register** keyword suggests (forces?) a variable to a register
 - Used for heavily accessed variables
 - Today, most compilers can figure this out themselves
 - You cannot take the address of something that is stored in a register
- The **volatile** keyword forces the results to be written back to memory
 - Used in low-level and parallel programming



Examples

```
// tell compiler that a should be stored in a register
register int a = 23;
```

```
// force b to be written back to memory
volatile int b = 43;
```



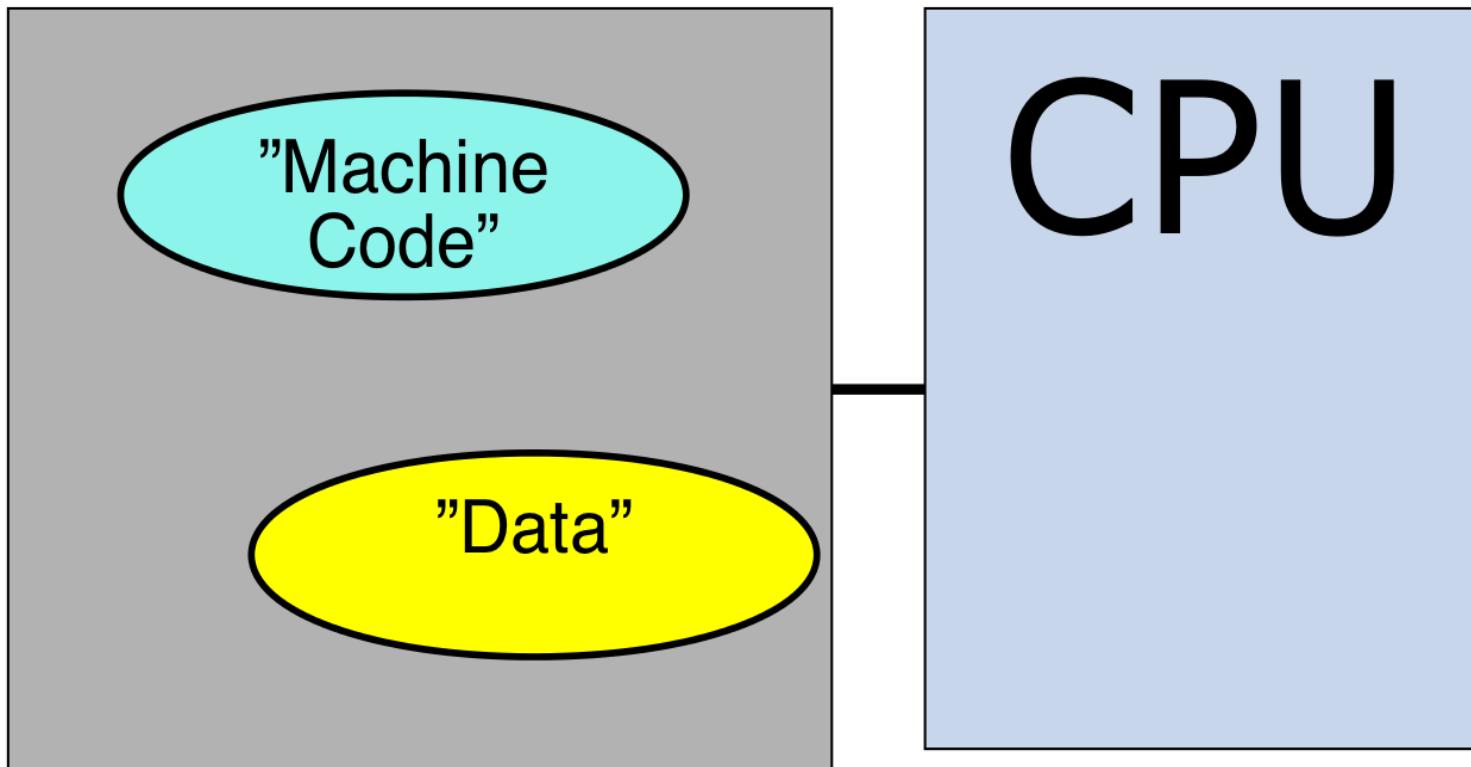
Machine Instructions in modern LOAD-STORE machines

Basic types of machine instructions:

- Data movement between memory and registers: “move”
- Arithmetic and logical operations: add, multiply, bitwise ops, ...
- Conditions and jumps: “goto”, conditional goto
- Procedure calls: “call” and “return”



Execution in a CPU

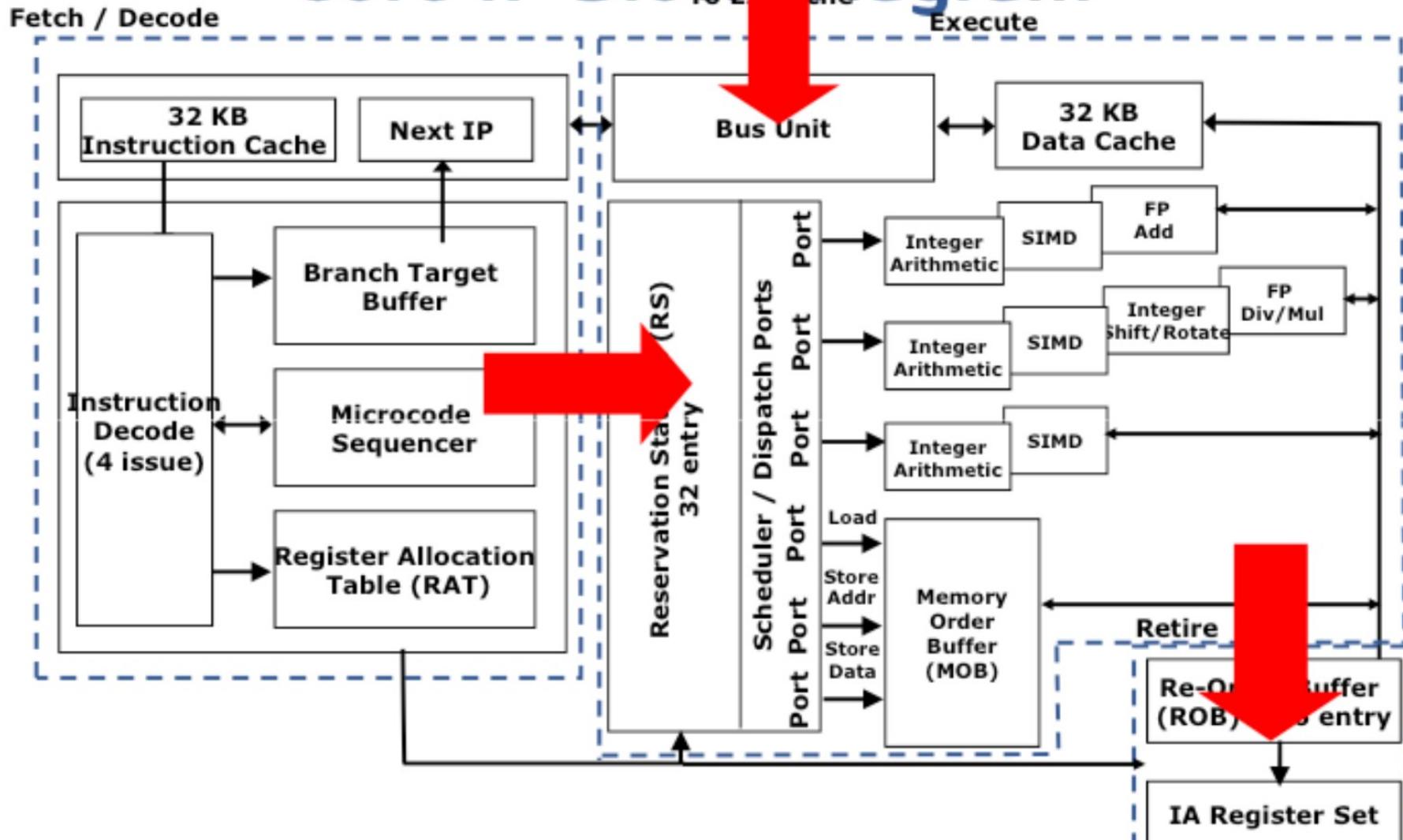




Architecture

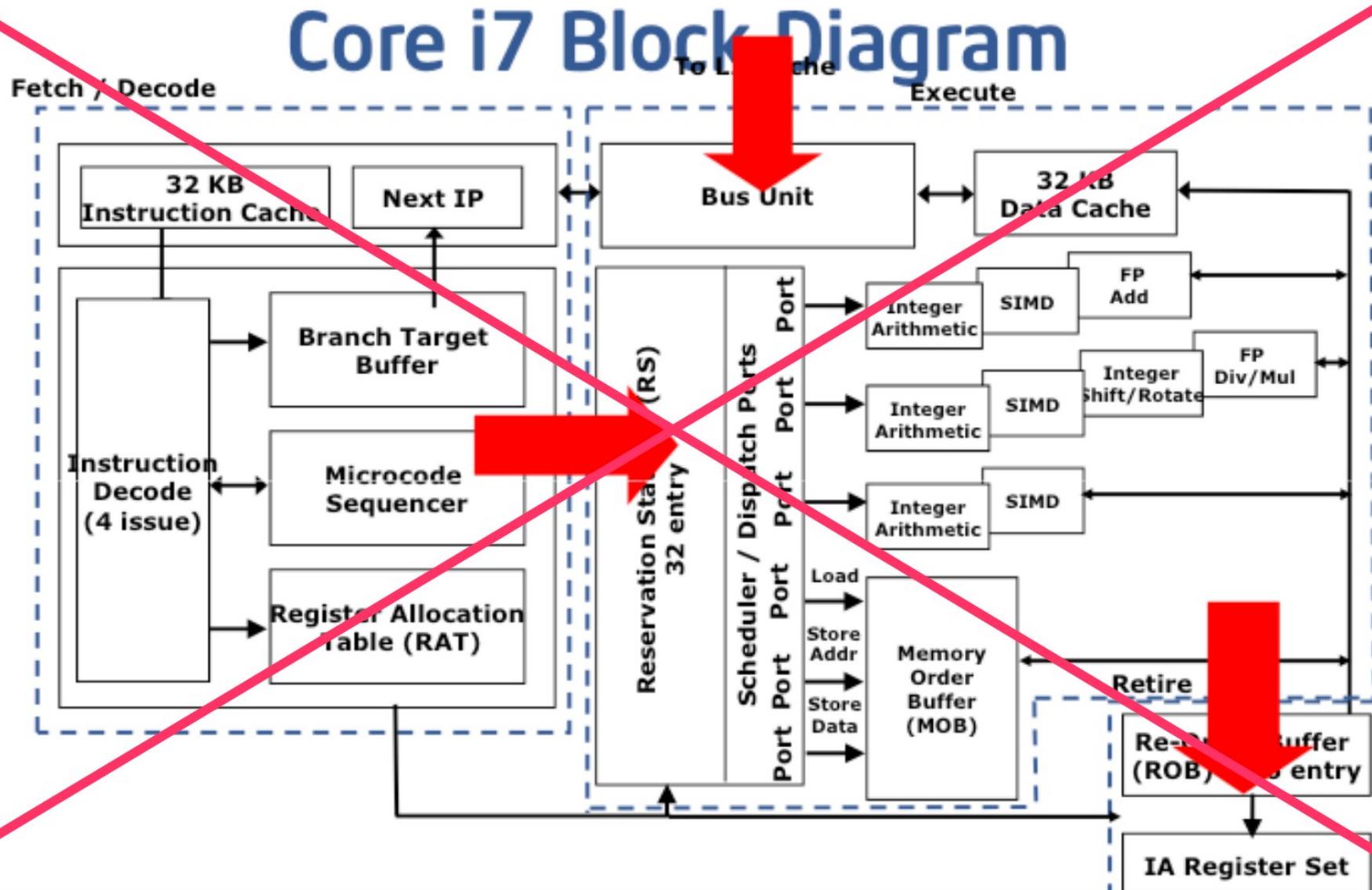
UPPSALA
UNIVERSITET

Core i7 Block Diagram





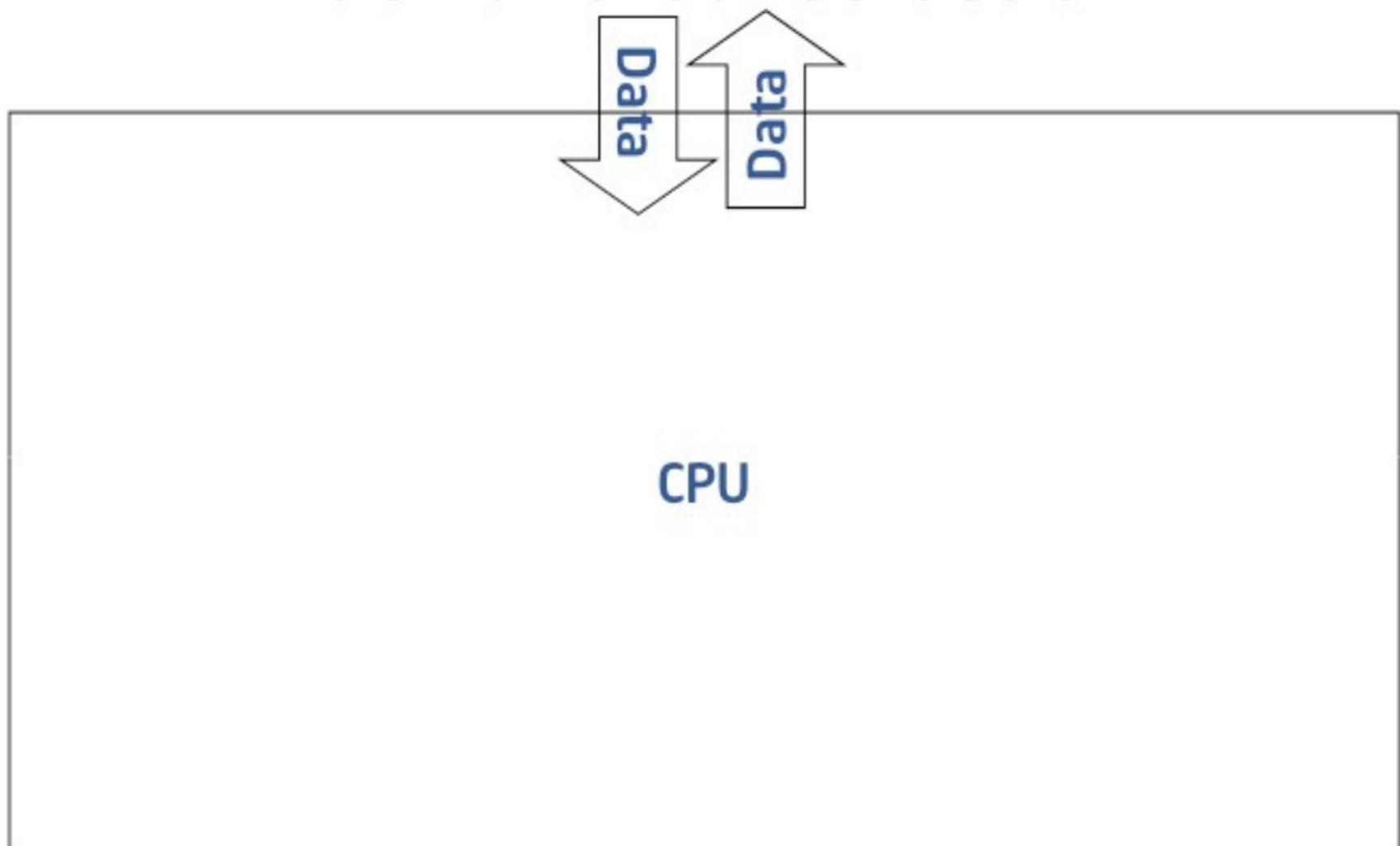
Architecture – too complex





UPPSALA
UNIVERSITET

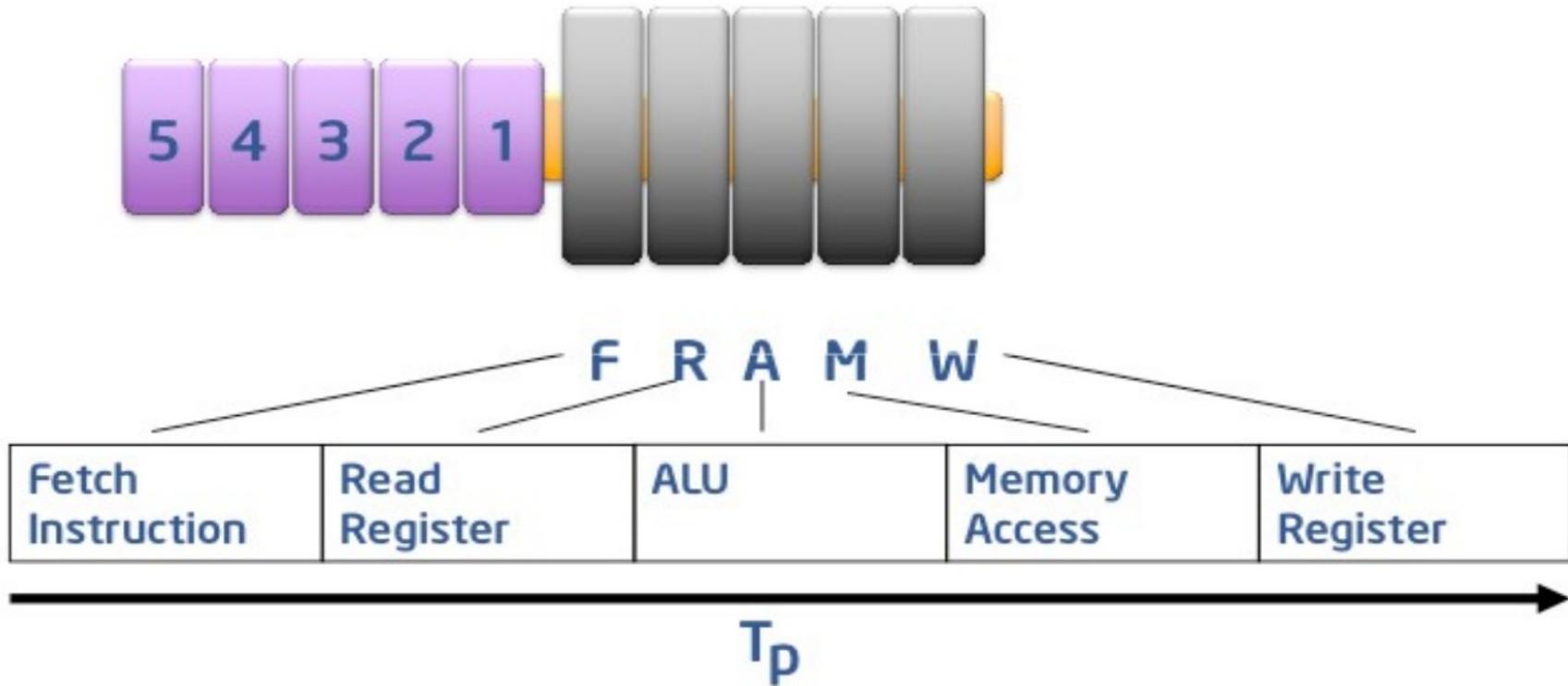
CPU -- a simpler model





Five common steps

UPPSALA
UNIVERSITET



$$\text{Execution time} = N_{\text{instructions}} * T_p$$



UPPSALA
UNIVERSITET

How "long" is a CPU cycle?

1982: 5MHz clock

1 cycle <--> 200ns

2002: 3GHz clock

1 cycle <--> 0.3ns

2012: 3GHz clock

1 cycle <--> 0.3ns



Instruction stages

- Instructions need to be fetched from memory and then decoded
- To know which parts of the CPU logic that is to be used
- Many instructions read/write memory from/to registers
- Every instruction is an encoded binary number
- Arithmetic work can be done once data is stored in the registers
- The different stages are performed using logic from different parts of the chip



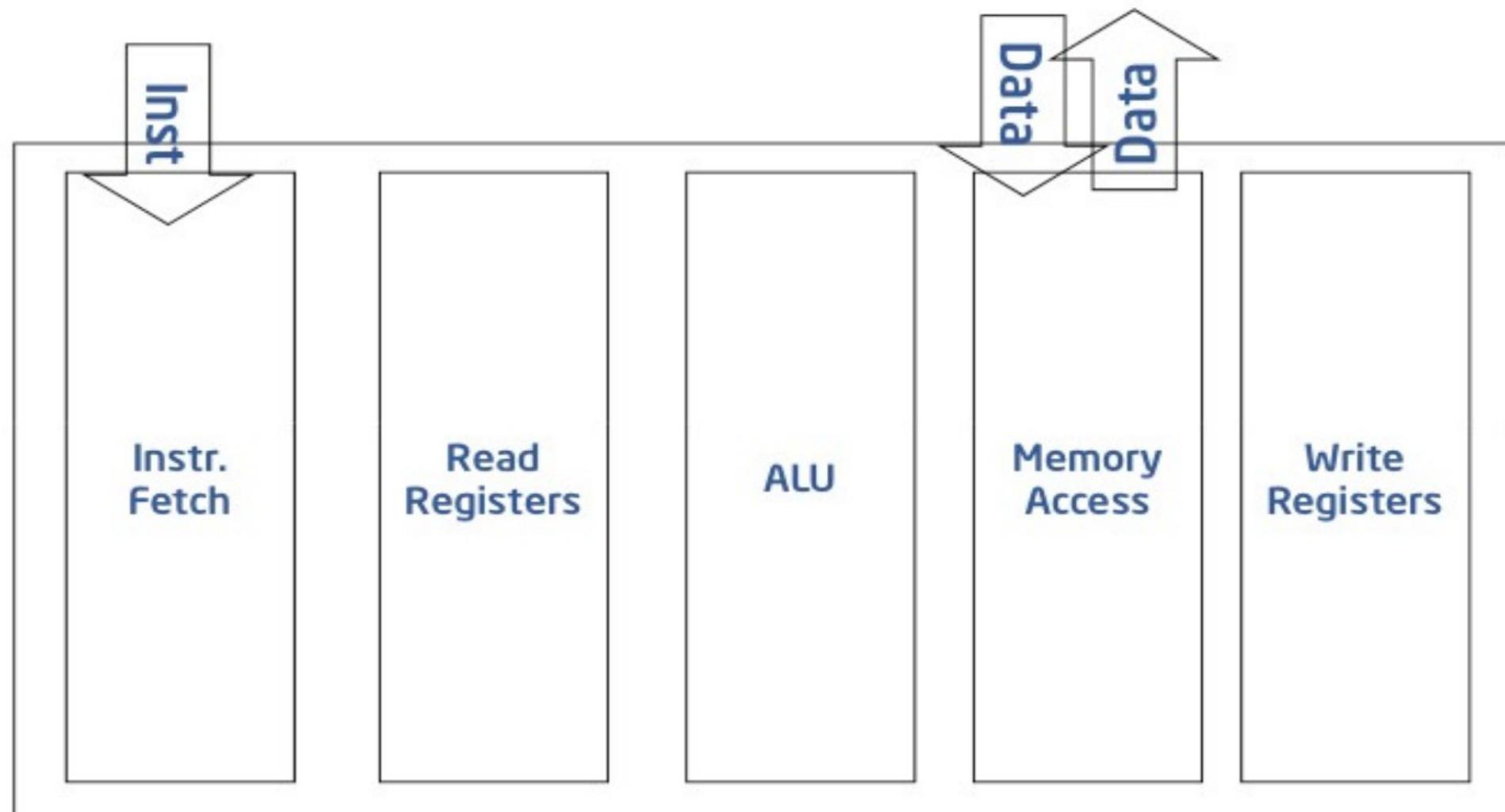
Instruction stages, example

Instructions can often be divided into several independent parts (stages). Example:

- 1. Instruction fetch (IF)
- 2. Instruction decode/register fetch (ID)
- 3. Execution (EX)
- 4. Memory access (MEM)
- 5. Write-back (WB)



Instruction stages, example





Observations

- All steps are performed in succession
- Steps are waiting for work most of the time
- Logic is idling
- The CPI (cycles per instruction) is at least 5
- Solution: use **pipelining**
- Start a new instruction each cycle
- Each clock cycle becomes a pipe stage
- Maximize use of logic



Pipelines

UPPSALA
UNIVERSITET

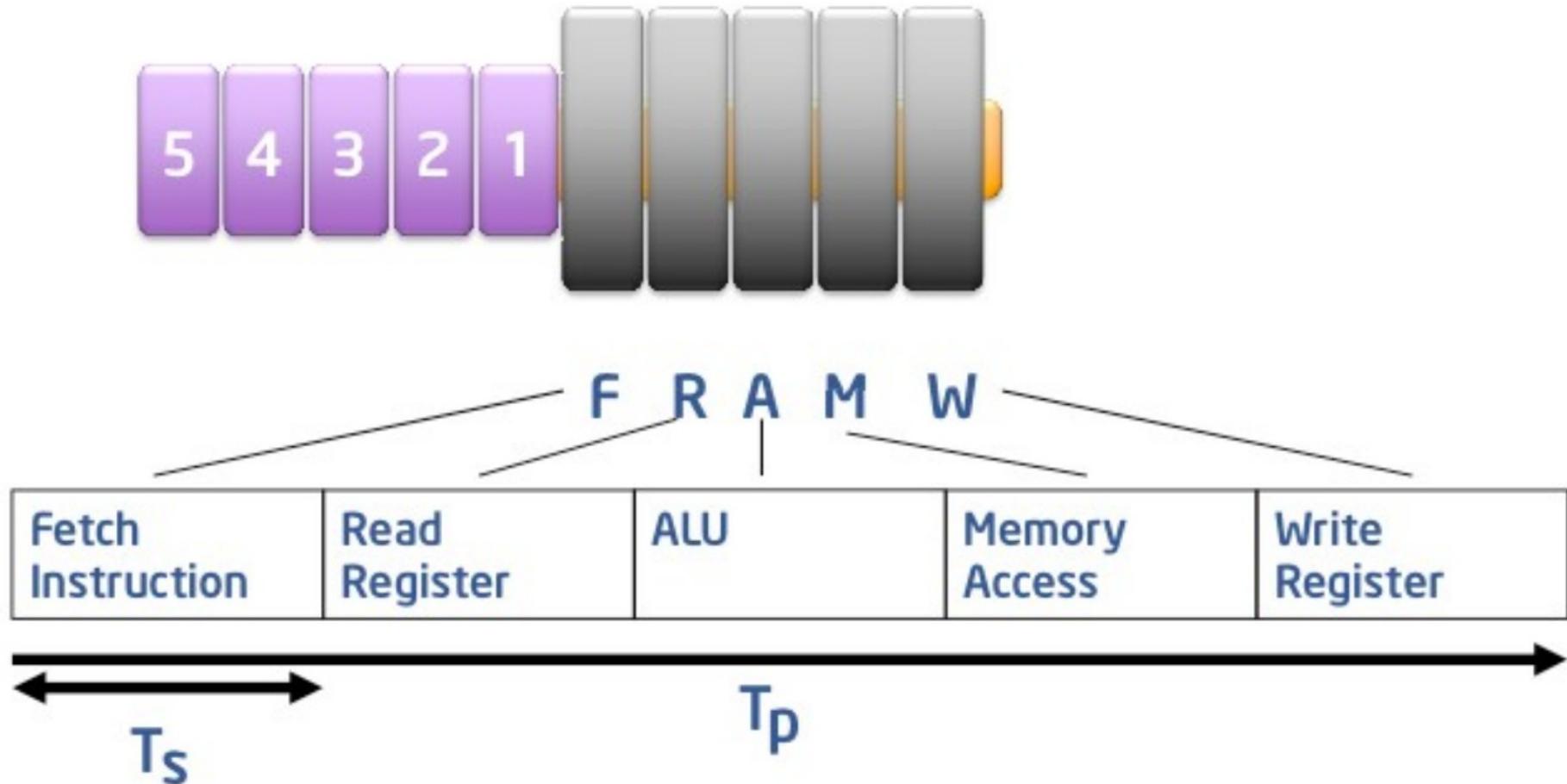
	-----> time ----->											
	1	2	3	4	5	6	7	8	9	10	11	12
Inst1	IF	ID	EX	MEM	WB							
Inst2		IF	ID	EX	MEM	WB						
Inst3			IF	ID	EX	MEM	WB					
Inst4				IF	ID	EX	MEM	WB				
Inst5					IF	ID	EX	MEM	WB			
Inst6						IF	ID	EX	MEM	WB		
Inst7							IF	ID	EX	MEM	WB	

- Pipelining allows several instruction steps to execute simultaneously
- In this example, up to 5 x faster execution
- Requires that instructions are *independent*



Pipelines

UPPSALA
UNIVERSITET



$$\text{Execution time} = N_{\text{instructions}} * T_s$$



Pipeline speedup

- For a 5-stage pipeline we can execute one instruction per cycle if the pipe is full
- In general a pipe of length n will give a speedup of n if we ignore startup cost
- Same technique used in fabrication industry
 - Cars, mobile phones etc.



Problems with pipelines

- Works fine if we can start a new instruction every clock cycle
- Instructions must be independent
- If the pipeline detects a hazard its stalls, pipeline bubble
- Branch hazards
- A branch can make instructions halfway through the pipe unnecessary
- The pipe needs to be flushed



Avoiding branches

- Branching can be expensive
- Don't repeat checking the same condition

```
for (i = 0; i < 42; i++)  
    if (a) dosomething(i);
```

```
if (a)  
    for (i = 0; i < 42; i++)  
        dosomething(i);
```



How to get more independent instructions?

- Loop unrolling – rewrite loop to do more work in each loop iteration
- Loop fusion – turn two (or more) loops into one



Today: ~10-20 stages and 4-6 pipes

- + Shorter cycletime (more MHz)
- + ILP (parallel pipelines)
- - Branch delay even more expensive
- - Even harder to find "enough" independent instructions.

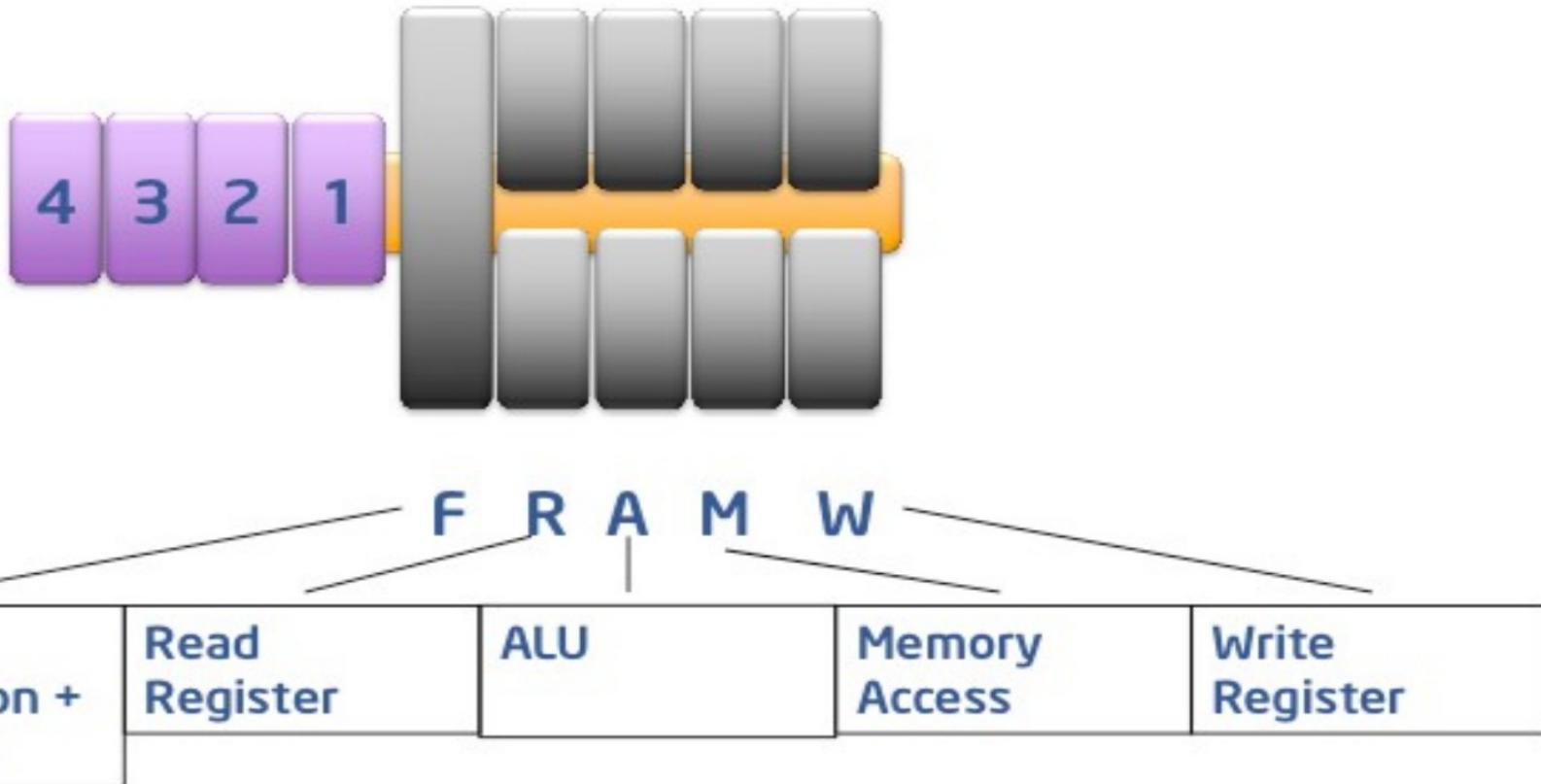


Instruction scheduling

- Compilers know which resources are available and how long instructions take
- Compilers schedule the instructions
- Tries to minimize pipeline bubbles
- Keep all resources busy (increase parallelism)
- Sometimes we will have bubbles due to hazards
- Branch prediction
- Out-of-order execution



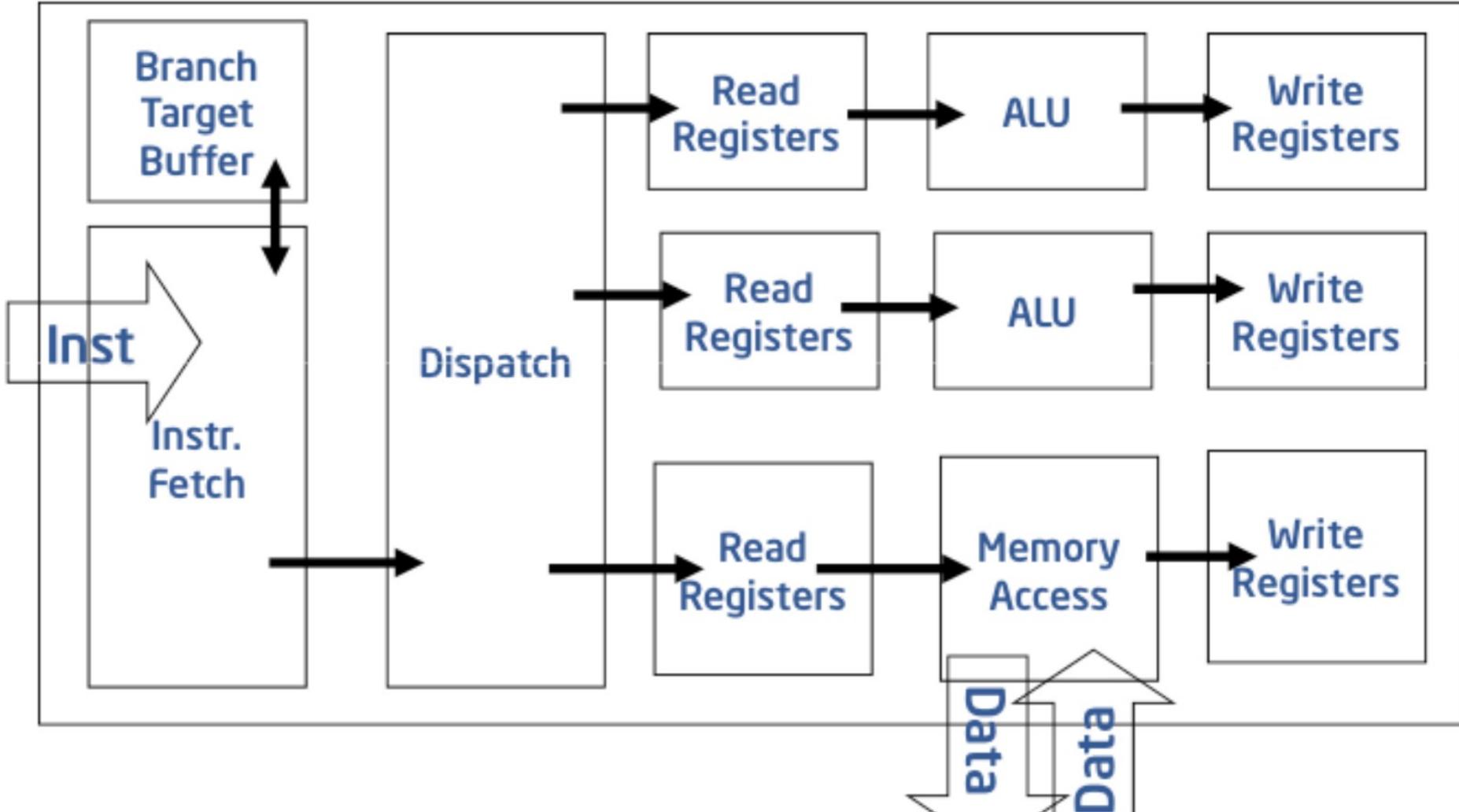
Multiple pipelines



Execution time = $N_{\text{instructions}} * T_s / N_{\text{Pipelines}}$
Problems of the Pipeline remain!



Model of CPU with several pipelines





Modern memory: ~150 CPU cycles access time

- + Shorter cycletime (more MHz)
- - Memory access even more expensive.
- Memory access is **slow**.



Implications for pipelines

- Waiting for data stalls the pipeline
- We will need a lot of registers to hide this latency
- Solution: cache memories
- A cache memory is a smaller SRAM (faster) memory which acts as a temporary storage to hide the main memory latencies



Data locality

- We can predict what instructions and data a program will use based on its history
- Temporal locality, recently accessed items are likely to be accessed in the near future
- Spatial locality, items whose addresses are near one another tend to be referenced close together in time



Caches

UPPSALA
UNIVERSITET

- Several levels of cache with varying latency
- L1 (approx. 10-100 kB) 1-5 cycles
- L2 (256 kB-) 10-20 cycles
- L3 (> 2 MB), slower
- Caches organized into "cache lines", approx. 128 bytes each



Example of costs associated with cache/memory access

To Where	Cycles
Register	<= 1
L1d	~3
L2	~14
Main Memory	~240

(Numbers listed by Intel for a Pentium M)



Cache optimization

- An example of “hardware dependent optimization”
- Blocking:
Reorder data accesses to improve data locality
- Modern prefetchers are great, can help.



Cache optimization – how?

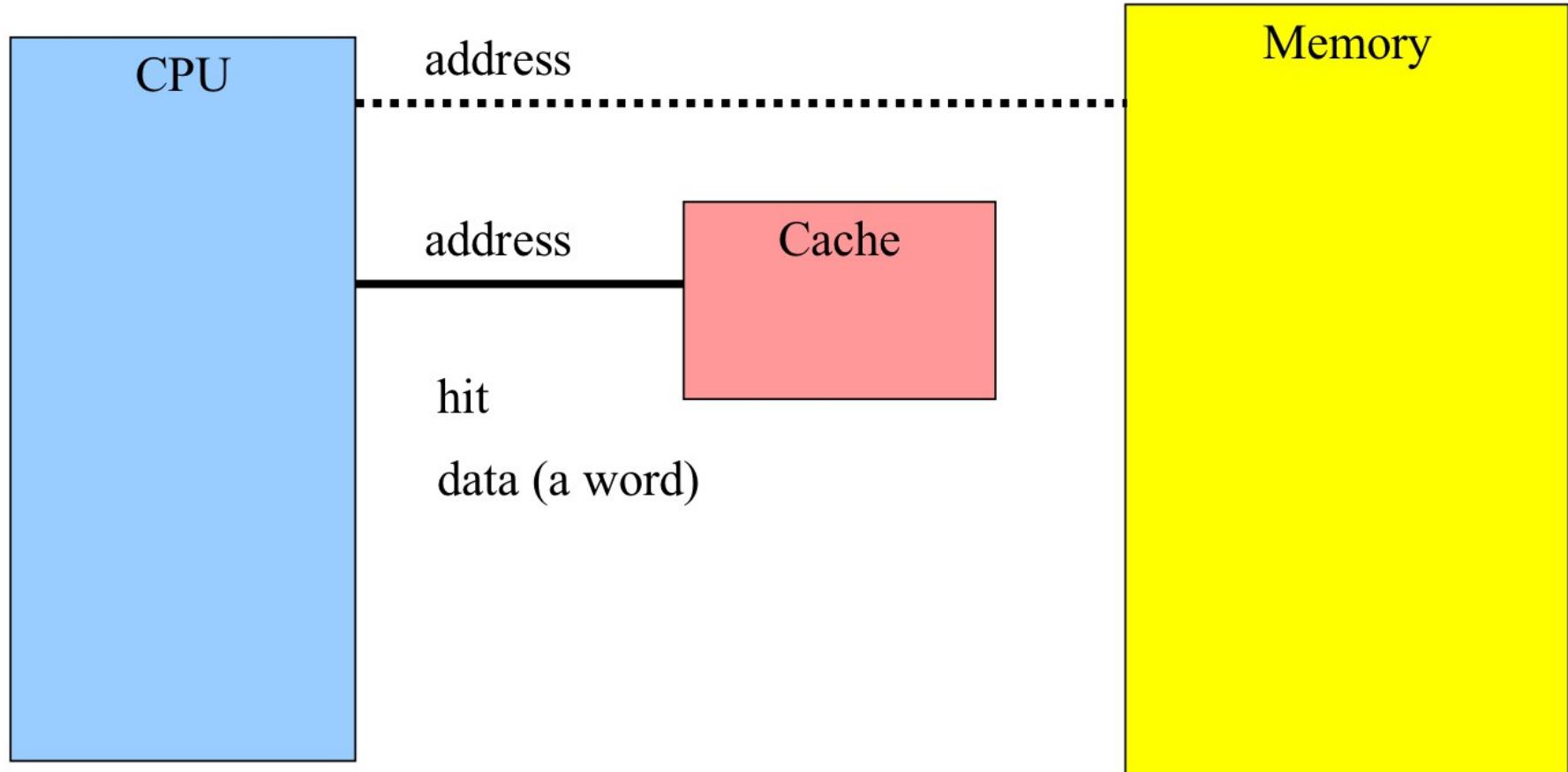
Try to increase the chances that the needed memory locations are already in cache:

- Nearby memory accesses in time should be to nearby locations in memory



Cache

UPPSALA
UNIVERSITET





Quick summary – important hardware-related concepts

- Pipelining --> independent instructions are good
- Cache memory --> data locality is good



High Performance Computing -- Code Optimization IV

- More about machine-dependent optimizations
- **cache** memory, data **locality**, **pipelines**
- How to make your program run faster on modern hardware

Elias Rudberg, room P2440

elias.rudberg@it.uu.se



Quick summary – important hardware-related concepts

- Pipelining --> independent instructions are good
- Cache memory --> data locality is good



Caches

UPPSALA
UNIVERSITET

- Several levels of cache with varying latency
- L1 (approx. 10-100 kB) 1-5 cycles
- L2 (256 kB-) 10-20 cycles
- L3 (> 2 MB), slower
- Caches organized into "cache lines", approx. 128 bytes each



Cache optimization – how?

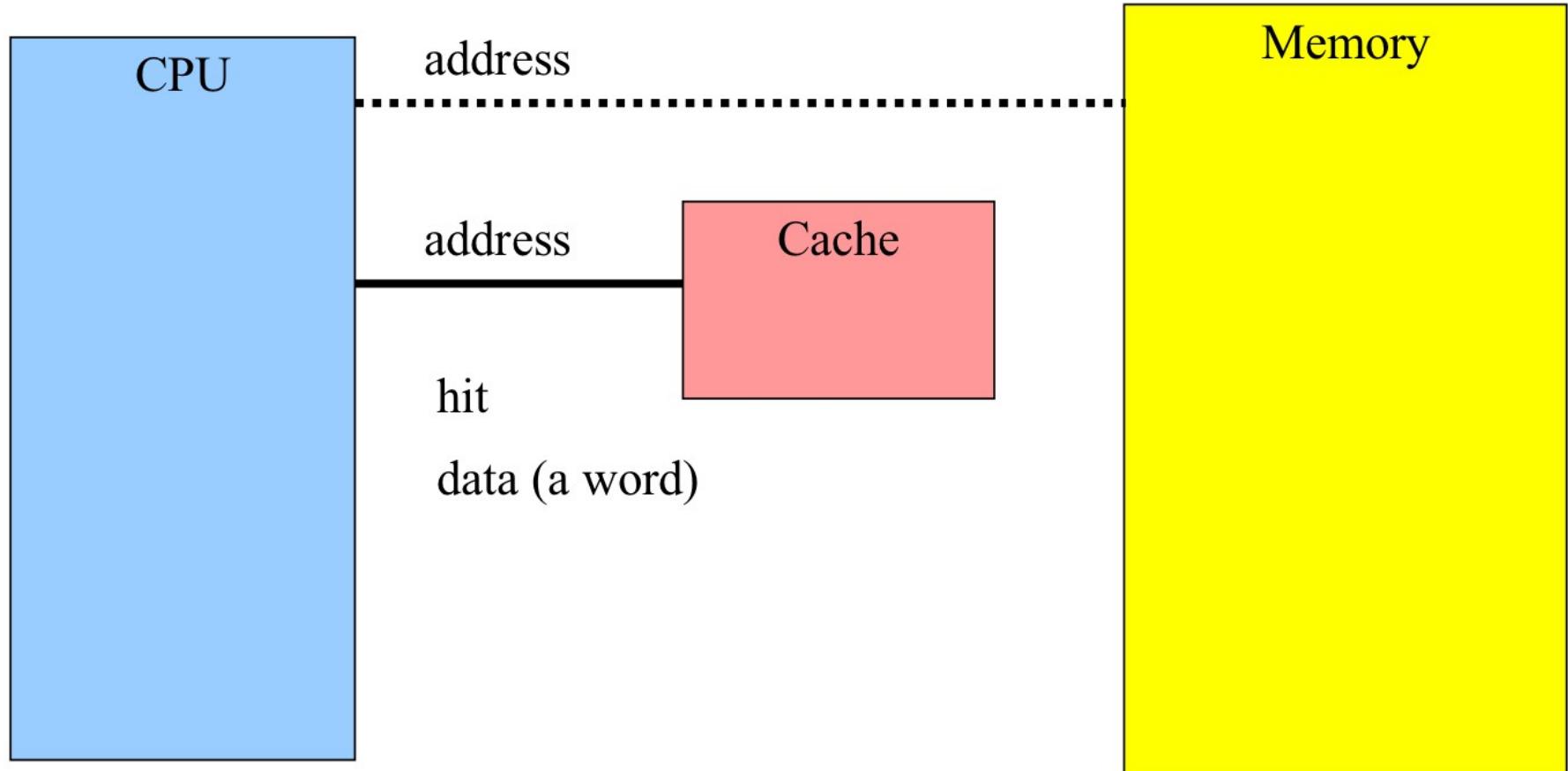
Try to increase the chances that the needed memory locations are already in cache:

- Nearby memory accesses in time should be to nearby locations in memory



Cache

UPPSALA
UNIVERSITET

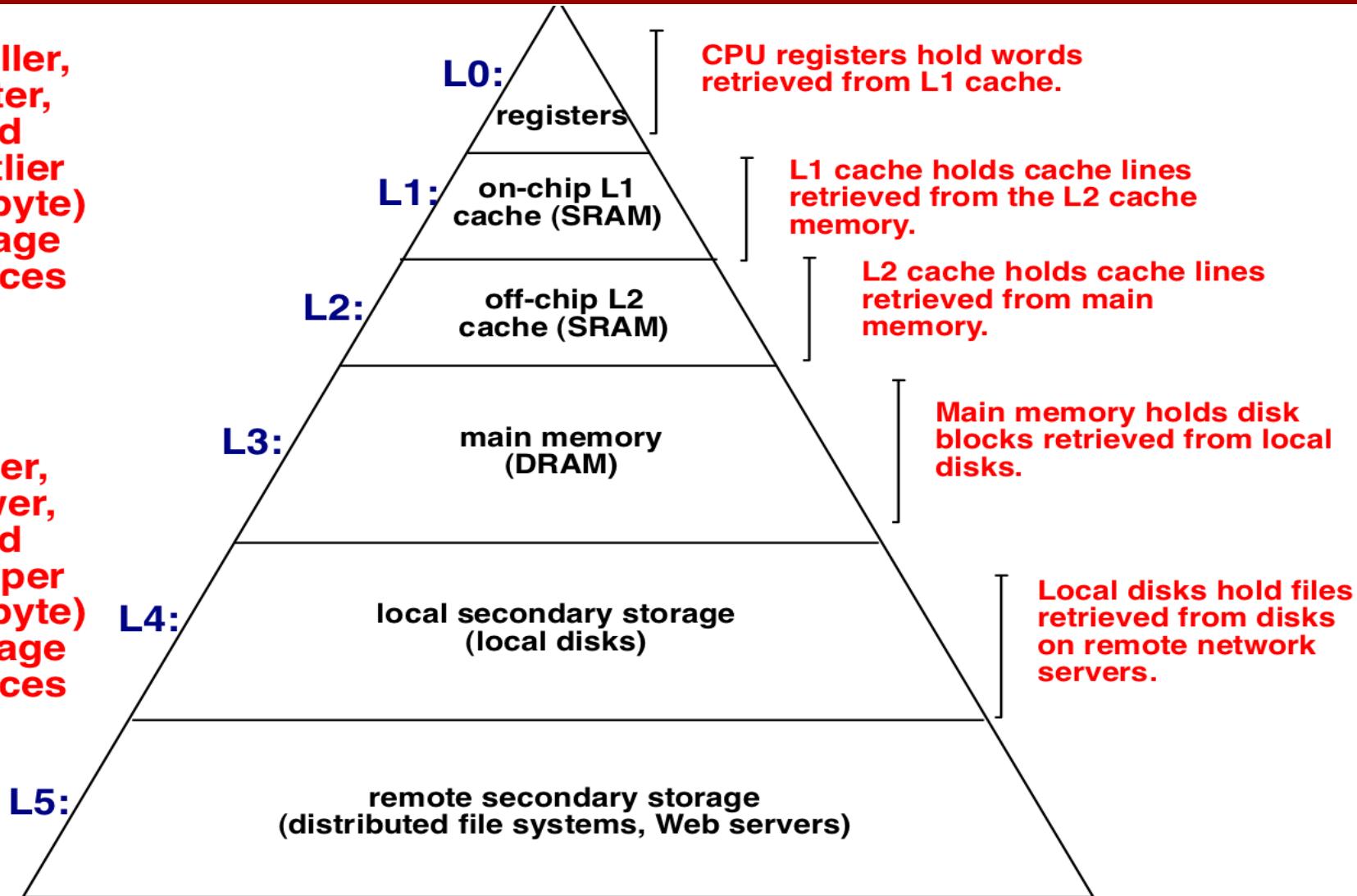




Example of Memory Hierarchy

**Smaller,
faster,
and
costlier
(per byte)
storage
devices**

**Larger,
slower,
and
cheaper
(per byte)
storage
devices**





UPPSALA
UNIVERSITET

General Caching Concepts

- “Cache hit”
- “Cache miss”
- Cache Performance Metrics: "hit rate" or "miss rate"



Caches and performance

- Caches are extremely important for performance
- Level 1 latency is usually 1 or 2 cycles
- Work well for problems with nice locality properties
- Caching can be used in other areas as well, example: web-caching (proxies)
- Modern CPUs have two or three levels of cache
- Most of the chip area is used for caches



How caches work -- cache lines

- Cache divided into **cache lines**
- When memory is read into cache, a whole cache line is always read at the same time
- Good if we have data locality: nearby memory accesses will be fast
- Typical cache line size: 64-128 bytes



How do caches work?

- "**Direct-mapped cache**":
 - Hash function maps to a single place for every unique address
- "**Fully associative cache**":
 - Cache space is divided into n sets
 - Address is distributed modulo n
 - k -way set associative ($k = 2 \dots 24$)
- "**Set associative cache**":
 - Maps to any slot
 - Hard to build in practice



Caches in hierarchies

To synchronize data in hierarchies caches can either be:

- **Write-through**
 - Reflect change immediately
 - L1 is often write-through
- **Write-back**
 - Synchronize all data at a given signal
 - Less traffic, but other drawbacks instead



SSE SIMD instructions

- Single instruction, multiple data (SIMD)
- Streaming SIMD Extensions (SSE)
 - > “SSE SIMD” instructions
- Allows increased performance by doing the same operation for several data entries at once.
- As with pipelining, requires independent instructions.
- Tricky to use in your own code. Intrinsics. The compiler can (hopefully) do this for us
- Heavily used in optimized linear algebra libraries



How to get more independent instructions?

- Loop unrolling – rewrite loop to do more work in each loop iteration
- Loop fusion – turn two (or more) loops into one



Loop unrolling

- Loop unrolling – rewrite loop to do more work in each loop iteration

```
for(i = 0; i < N; i++) {  
    // do something related to i  
}
```

-->

```
for(i = 0; i < N; i+=3) {  
    // do something related to i  
    // do something related to i+1  
    // do something related to i+2  
}  
// Afterwards: take care of remaining part, if any  
// (needed in case N does not divide evenly by 3)
```



Loop fusion

- Loop fusion – turn two (or more) loops into one

```
for(i = 0; i < N; i++) {  
    // do "work A" related to i  
}  
for(i = 0; i < N; i++) {  
    // do "work B" related to i  
}
```

-->

```
for(i = 0; i < N; i++) {  
    // do "work A" related to i  
    // do "work B" related to i  
}
```



Theoretical Peak Performance

- Assumptions:
 1. Functional units produce a result every clock cycle
 2. No cache miss penalties
 3. Infinite memory bandwidth
 4. Enough independent instructions to use all units
- Theoretical Peak Performance

$\text{Peak} = \text{Clock_freq} * \text{No_functional_units}$

Ex, 2 FP Units, 3GHz --> $\text{Peak} = 3 * 10^9 * 2 = 6 \text{ GFlops}$



Modern computers, summary

- Practically all modern computers are LOAD/STORE architectures
- Pipelining heavily used
- They have a two, or three level memory hierarchy with large on-chip caches
- Clock frequencies are measured in GHz
- Most CPU:s are at least 4-way superscalar which, as an example, could mean that it can execute one LOAD/STORE, one integer ALU, and two FP instructions per clock cycle



Problems in modern computers

- Hard to find independent instructions
- Caches only work if programs exhibit good spatial or temporal locality
- Superscalar pipelined microprocessors require very high memory bandwidth
- The CPU-Memory gap is increasing
- Caches get more important
- Compilers get better and better but cannot keep up with the innovations in hardware



UPPSALA
UNIVERSITET

Compilers are improving

- Example: “restrict test” code performs differently depending on gcc version
- Tested at the Kalkyl computer at UPPMAX
- Version 4.4.6 of gcc (from 2011) gives no speedup from using restrict in this test case
- Version 4.5.3 of gcc gives nice speedup



UPPSALA
UNIVERSITET

Recap of optimization stuff

- Remember: care about correctness and flexibility before performance



Priorities: correctness comes first!

Priorities:

- 1: Correctness
- 2: Flexibility
- 3: Performance



UPPSALA
UNIVERSITET

Do profiling first!

- Always measure performance of different parts of the code before doing optimizations

Remember:

Premature optimization is the root of all evil



Always test if attempted optimizations give improvements

- Is there any real benefit? TEST!
 - If results almost unchanged, go for the simpler version



That's all!

UPPSALA
UNIVERSITET

- Questions?



Parallelization & Multicore

Increasing hardware utilization through parallelism

- Types
 - Instruction-Level Parallelism
 - Shared memory
 - Distributed memory
- Algorithmic considerations



Why parallelism?

Physics puts constraints on hardware design.

We have multicore hyper-threaded pipelined vector processors



Intel “Sandy Bridge” (SNB) / Mainstream Quad-Core

32 nm Process / ~225 mm² Die Size / 85W TDP / A0 Stepping / Tape Out : WW23'09

Expected : Q11 @ 3.0 - 3.8(T) GHz



Instruction-level parallelism (ILP)

Multiple instructions that can be run at once

Only a CPU without a pipeline would have no ILP

Compiler and CPU both attempt to maximize ILP

- You can help
- Compare optimization lectures



Different forms of parallelism

Multiple instructions, multiple data (MIMD)

- Threads

- Independent units of execution
- Some shared data

- Processes

- No shared data

Single instruction, multiple data (SIMD) ***

- Vectorization instructions (e.g. SSE, Altivec)

Multiple instructions, single data (MISD)

- Pipelining, branch prediction

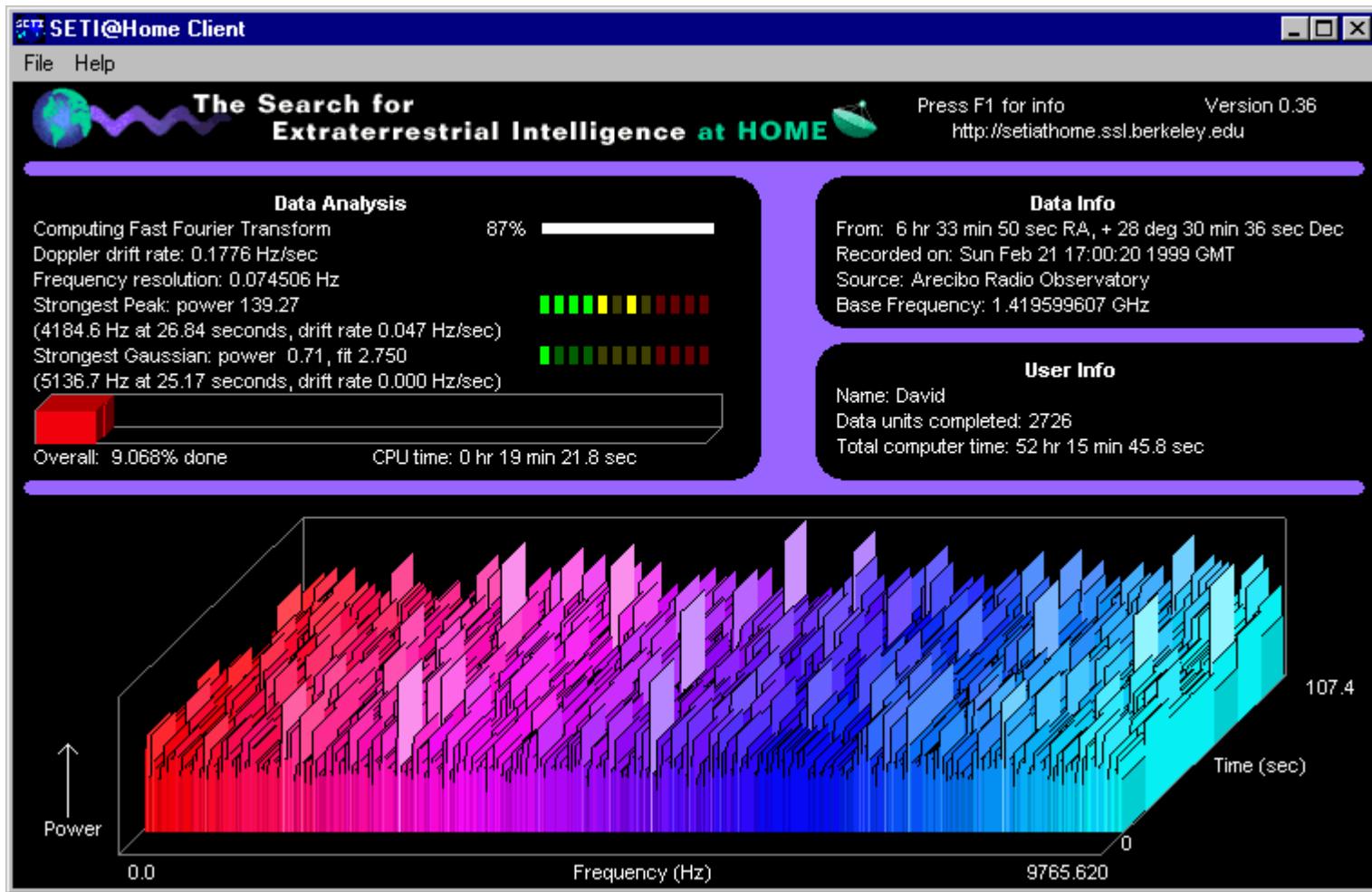
- Fault-tolerant systems



UPPSALA
UNIVERSITET

Parallelism everywhere

SETI@home





Multithreading on a Single Core

- Regular switching between threads
- Also switching when a thread *waits*
 - Can wait for user input
 - Might wait for synchronization on another thread
 - Any synchronous I/O call can cause a wait
- More threads than cores can make sense if
 - If you have a lot of I/O
 - Simultaneous Multithreading (SMT or Hyper-threading)
- Thread priority will control scheduling
- Cost of thread creation/destruction
- **Cost of *context switching***



Classic example

Asynchronous I/O

1. Make an I/O call
2. Do something
3. Check status for I/O

I/O latency doesn't change, but you can useful work instead.

This could be done with threads instead... but each thread comes at a cost



Symmetric Multithreading

- Called HyperThreading by Intel
- Double register files
- Context switching extremely cheap
- Benefits are application-specific:
 - Hide memory latency
 - Decreased cache size/thread
- Helps if memory *latency* is a bottleneck



Vectorization

UPPSALA
UNIVERSITET

- True SIMD
- SSE (Intel & AMD) & AltiVec (IBM & Motorola)
 - 128-bit registers: 16 chars, 8 shorts, 4 floats, 2 doubles.
- AVX: 256-bit registers
- Usage:
 - *Intrinsics* provided by compiler
 - Auto-vectorization by compiler



UPPSALA
UNIVERSITET

Parallelism everywhere

Dreamworks, Industrial Light&Magic





Multithreading on Multiple Cores

- Threads will still sometimes wait...
 - ...possibly resuming on a different core
 - Consequences for cache behavior
 - Cache coherency maintained by hardware – sort of
- Thread *CPU affinity* can be set manually for performance reasons
 - This will make your code more sensitive to scheduling or other processes running
 - Possibly recommended for a dedicated machine



Inter-thread communication

Used to be expensive

Getting cheaper, at least between some threads

Synchronizing between *all* threads is expensive

Modern profilers can help assess the overhead



OpenMP

Allows simple implementation of threading

Basic constructs for stating several paths to be executed at the same time

Execute loop iterations separately

Realized through compiler pragmas

Some code can work with and without OpenMP



OpenMP example

```
void M_mult(float A[n][n], float B[n][n], float C[n][n]) {  
    int i,j,k;  
    float sum;  
  
#pragma omp parallel for, local(i,j,k,sum)  
for (i=0; i<n; i++)  
    for (j=0; j<n; j++) {  
        sum = 0;  
        for (k=0; k<n; k++)  
            sum += A[i][k]*B[k][j];  
        C[i][j]=sum;  
    }  
}
```



OpenMP

UPPSALA
UNIVERSITET

Compile with -fopenmp (in GCC)

Difficulty:

- The iterations in your for-loop must be *independent*
- Properly declaring local, private and shared variables



Memory allocation in Multicore Environments

NUMA and memory affinity is one consideration

NUMA: Non-Uniform Memory Access

Memory allocated on one CPU might consistently be used by another

Libraries and e.g. heap implementations may not be NUMA-aware

Usually: *NUMA memory allocated on “first-touch” basis*



Memory Allocation in Threaded Environments

Heap implementation might lock all threads

Java and some malloc implementations keep a “thread cache” of memory soon to be allocated

Some mallocs might not even be thread-safe

Be sure to link to the right runtime library (should generally be ok)



Other Shared Resources

- Libraries might not be thread-safe at all:
 - Undefined behavior when used concurrently
 - Undefined behavior when used from multiple threads *at all*
 - Libraries might not be re-entrant
- Libraries might scale badly:
 - Might use a single lock for all threads
 - Unwarranted use of resources on a per-thread level



MPI

UPPSALA
UNIVERSITET

Message Passing Interface

Standardized communication between processes

Not shared memory

- Although can be implemented using shared memory

Ideal for lots of memory, no shared caches

Can be distributed over closely connected clusters

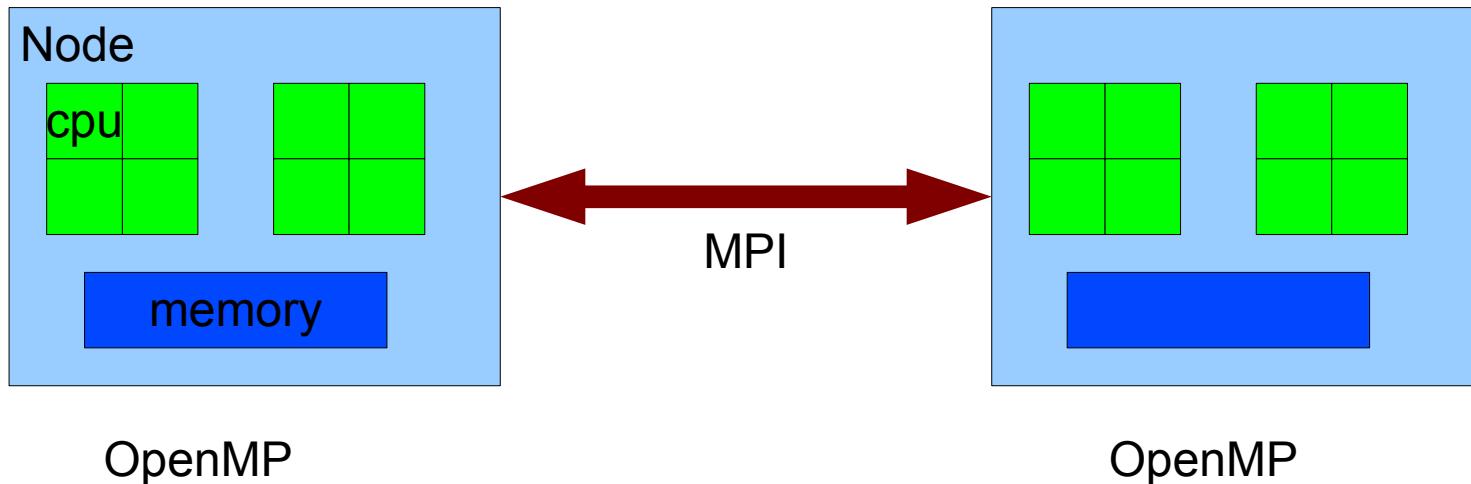


Solution

OpenMP on closely connected cores

MPI between machines

Sockets for truly distributed computing

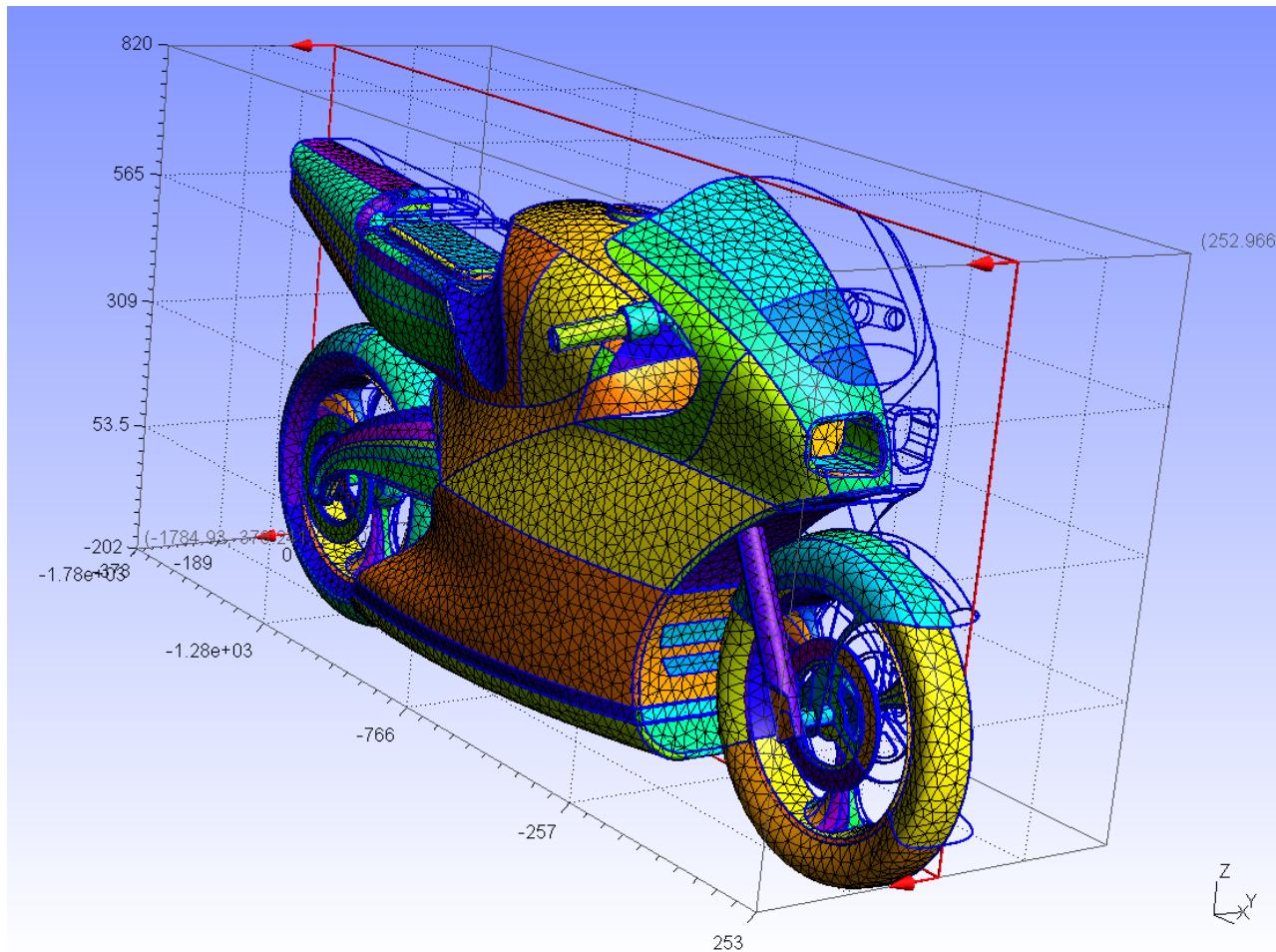




UPPSALA
UNIVERSITET

Parallelism Everywhere

Engineering: Finite Element Method





GPUs – Complicated parallelism

- Multiple Multiprocessors
 - Each multiprocessor: Single Instruction Multiple Thread (SIMT)
- Warps
 - 32 (or so) identical threads, different datasets
 - Some shared memory
 - A single program counter
- Branching is expensive
- So are memory accesses (any thread will stall all threads)
- Solution: Multiple warps in each multiprocessor
 - One warp runs while others access memory



Some words on Cell

Cell Broadband Engine

POWER-based

Main core (PPU)

8 Synergistic Processing Element (SPE)

- Vector machine
- Very simple cores

Not a shared-memory architecture

What would normally be a cache is an explicit local memory

Asynchronous access to main memory

Prefetching goes from nice to mandatory



Algorithms

UPPSALA
UNIVERSITET

An algorithm:

Do A

Do B

Do C

In order to parallelize, one must determine which of A, B, and/or C are *independent*. There may also be parallelism inside A, B, or C.

Once again, *well-structured* code is key. Avoid side-effects!



Performance considerations

- What is the bottleneck?
 - Computation
 - Communication bandwidth
 - Comm. latency (synchronization)
 - Load balancing
- What can we do about it?
 - Increase parallelism
 - Increase arithmetic intensity
 - Change algorithm



Arithmetic Intensity

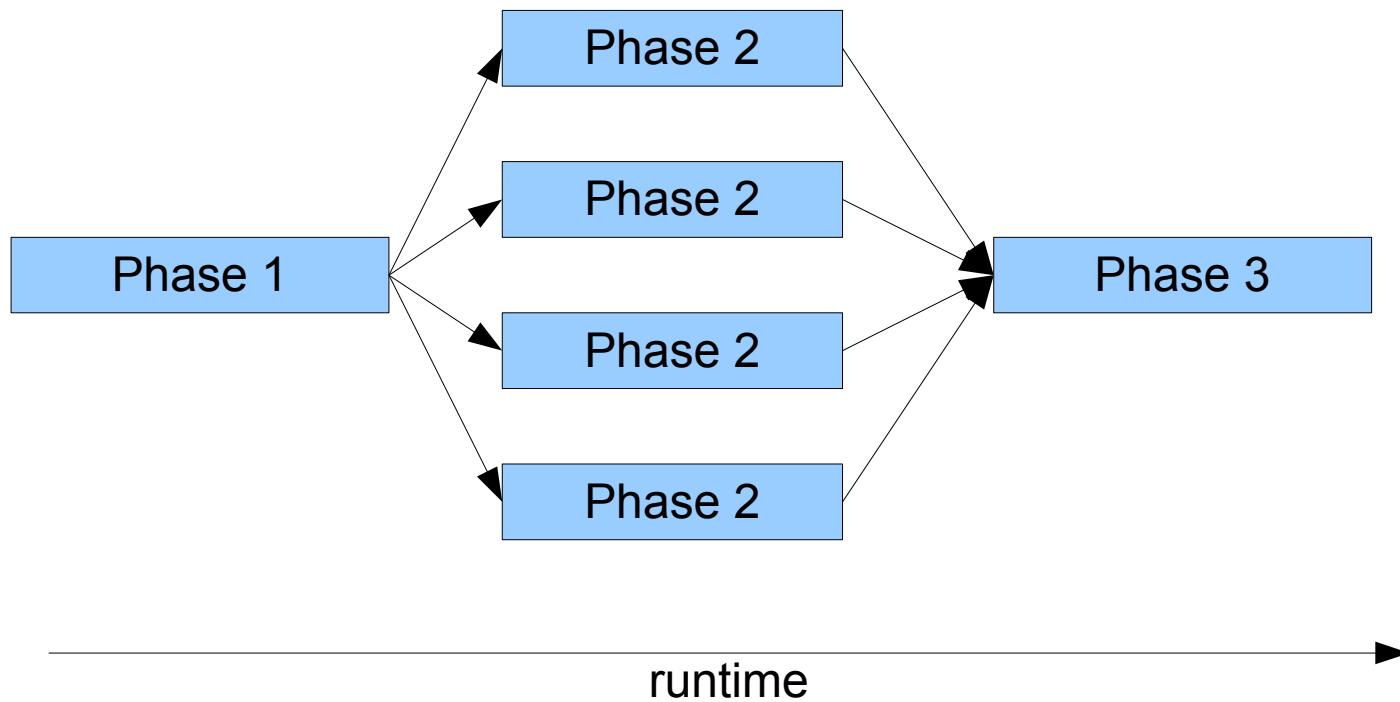
- Ratio of math operations and data transfer operations
- Units are e.g. operations/word
- Can vary according to
 - Algorithm
 - Implementation
 - Problem size (per node)
- A process with low arithmetic intensity is likely to be bandwidth bound — multithreading will not help



Parallel algorithmics

UPPSALA
UNIVERSITET

- Fork-join parallelism





Amdahl's Law

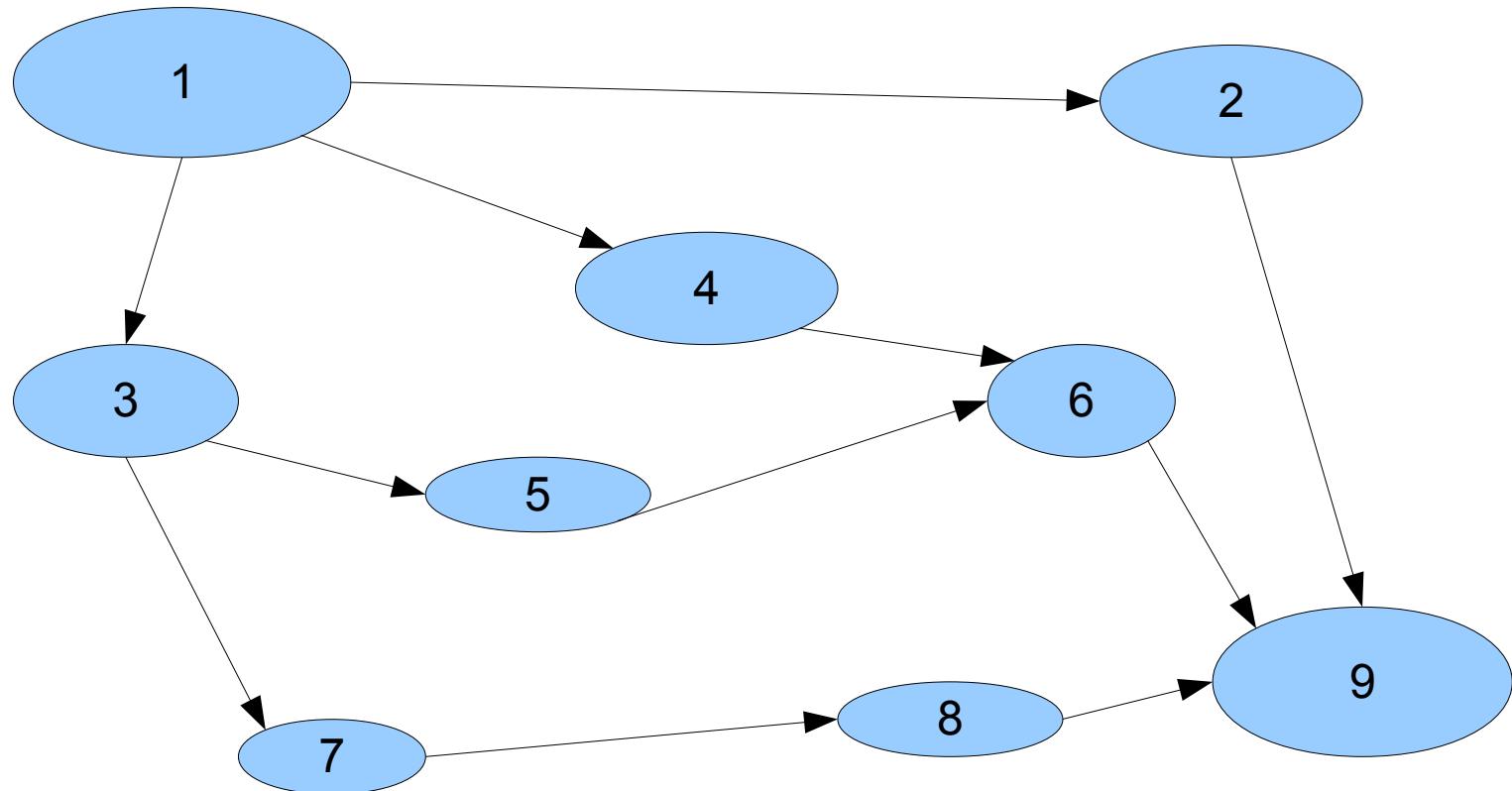
UPPSALA
UNIVERSITET

- For a given algorithm, there's a maximum possible speedup
 - P_{serial} = serial proportion of runtime
 - P_{parallel} = parallelizable proportion
 - $1 = P_{\text{parallel}} - P_{\text{serial}}$
 - $S_{\text{max}} = 1/(P_{\text{serial}})$
- If your program is 80% parallelizable:
- Only 5x faster with ∞ number of processors



Dependency Analysis

Directed Acyclic Graph (DAG)





Work/span Law

- For a given algorithm, there's a maximum possible speedup
 - Work = total number of execution units
 - Span = length of longest execution path
 - Maximum parallelism = work/span
- In the toy example: $9/5 = 1.8$



Parallel Algorithm Design

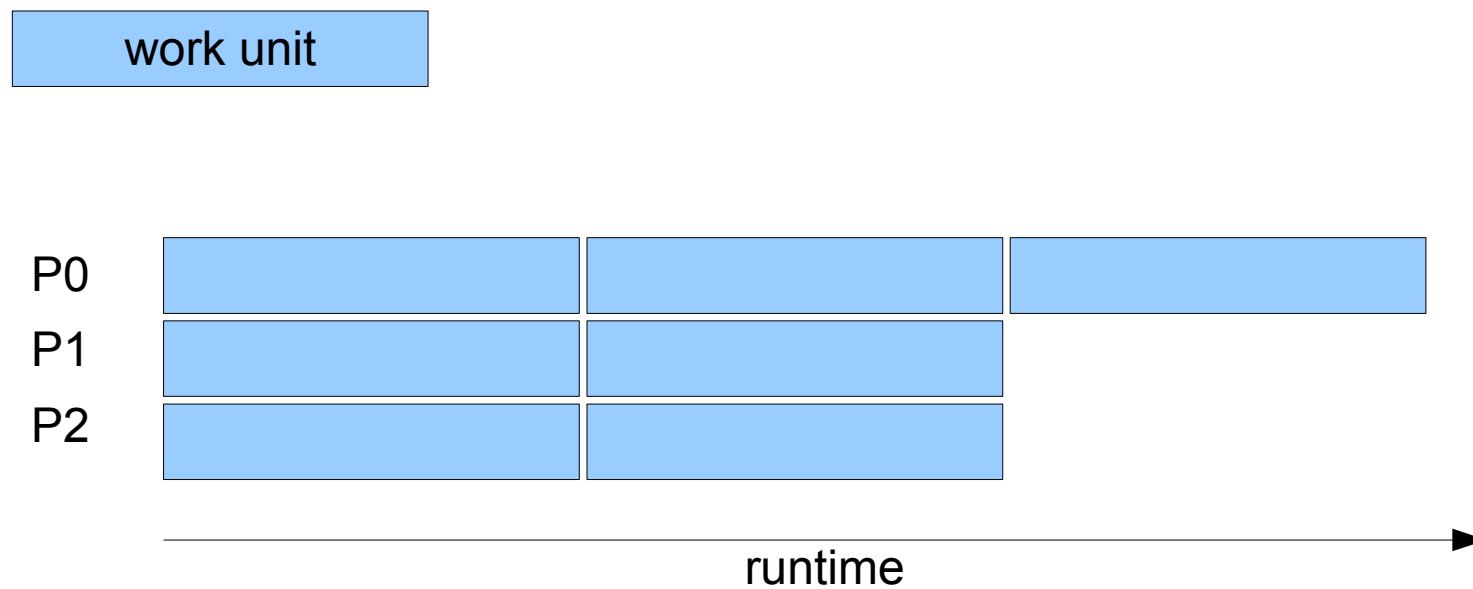
- Avoiding synchronization stall
 - Locks
 - Peer-to-peer communication
 - Overlap with useful work
 - Transactional memory
- Minimizing communication cost
 - Overlap with computation
 - Recompute locally instead of transferring
 - Postpone communication calls and coalesce into one



Load Balancing

UPPSALA
UNIVERSITET

Load balancing is always an issue

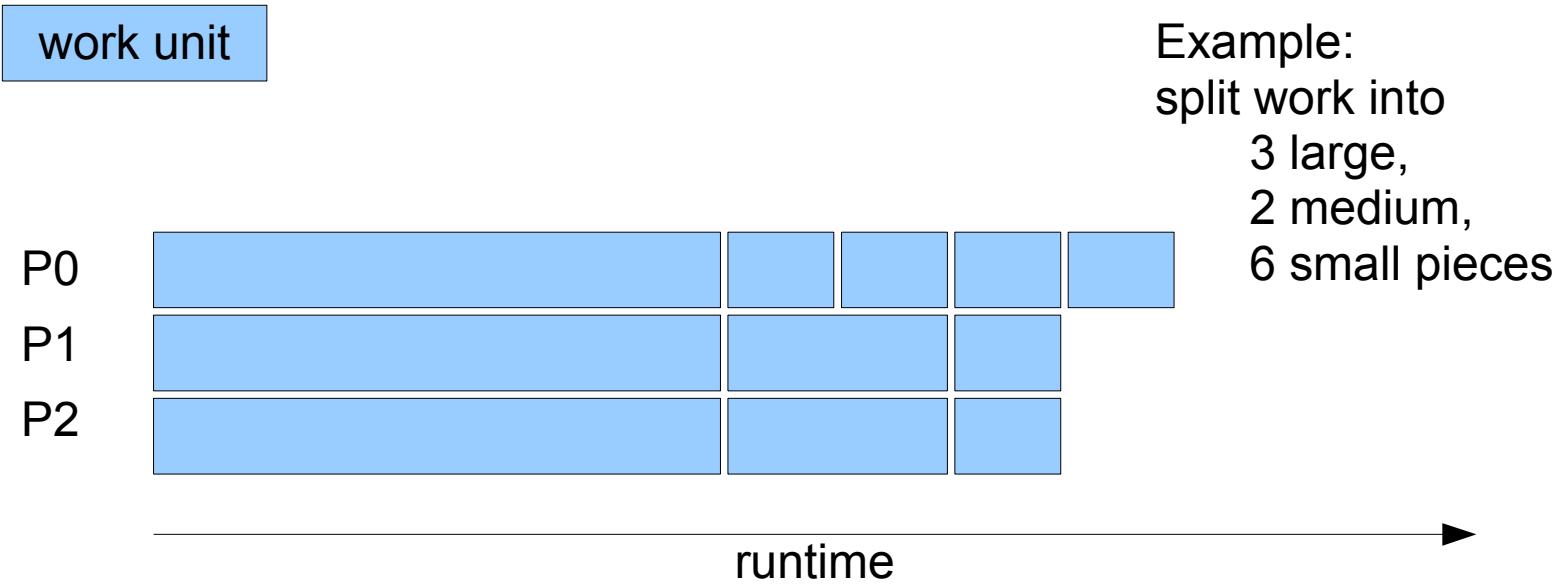




Improving Load Balancing

Option 1: Be very smart

Option 2: Schedule small work units at the end





Parallelizing Algorithms

UPPSALA
UNIVERSITET

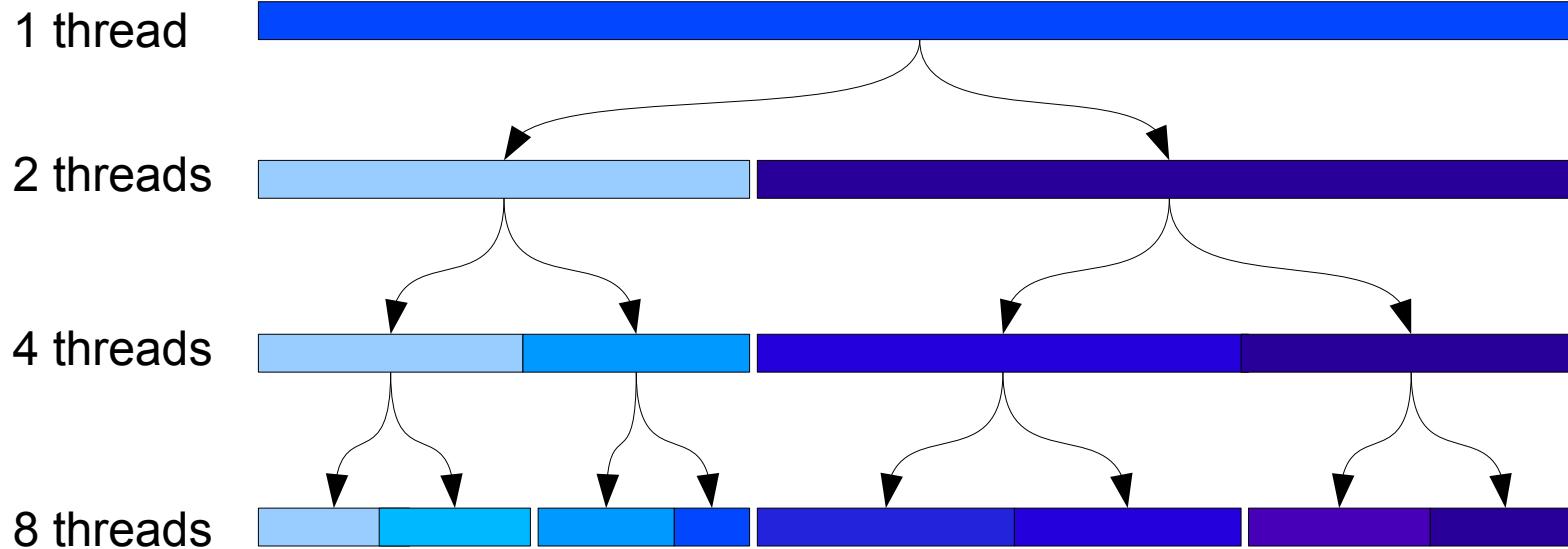
- Can be easy, can be tough

```
function quicksort(array, start, end)
    if(end-start < 2)
        swapifgt(array, start, end)
        return
    index = getPivot(array, start, end)
    partition(array, start, index, end)
    quicksort(start, index)
    quicksort(index, end)
```



UPPSALA
UNIVERSITET

Parallelizing Quicksort



Problem:



<





UPPSALA
UNIVERSITET

Task-based parallelism

- Define task from
 - input/output data
 - operations on data (kernel or function)
- Hand off the task to a runtime
- Runtime has a scheduling policy that gives tasks to worker threads



Benefits

- A runtime can select what is executed, where, and when
- Scheduler and runtime can be very smart and hardware-aware
- Reduce tight ordering in an algorithm
- Avoid stalling on unnecessary synchronization points
- ... But writing a smart implementation by hand can always yield even better performance.



Parallelizing Quicksort

- Reformulating the algorithm

```
function quicksort(array, start, end)
    stack s
    s.push({start, end})
    parallel while(! s.isempty())
        task = s.pop()
        if(task.end-task.start < 2)
            swapifgt(array, task.start, task.end)
            continue
        index = getPivot(array, task.start, task.end)
        partition(array, task.start, index, task.end)
        s.push({start, index})
        s.push({index, end})
```



Taskifying Quicksort

UPPSALA
UNIVERSITET

```
global scheduler s

function quicksort(array, start, end)
    s.push({start, end})

function workerLoop()
    while(s.isempty()) // wait for task
        task = s.pop();
        if(task.end - task.start < 2)
            swapifgt(array, task.start, task.end)
            return
        index = getPivot(array, task.start, task.end)
        partition(array, task.start, index, task.end)
        s.push({start, index})
        s.push({index, end})
```



Transactional memory

Bring database semantics to single memory accesses

The problem of threading is that multiple accesses need to be synchronized

Locking before every read and write is expensive

- Most of the time, no conflict will arise

Instead, rollback and throw an exception in the exceptional case where a conflict actually arises

Keep transactions short, as rollbacks will be expensive instead



Transaction-style thinking

Hardware transactional memory is not coming soon.

Software transactional memory can be useful. Often seen as “lock-free” algorithms

- 1) Do edits
- 2) Check that no one was doing anything behind your back
(Quite tricky, but can be done)

A little tip:

Atomic exchange of pointer to “state” object

Counter increased for any change



Other important aspects of high-performance parallel computing

- Lots of underlying technology that makes threads possible:
 - e.g. Operating System process scheduling
 - Network technology
 - Network topology
- Supercomputing centers
 - Uppmax, SNIC
 - Grids



Network Topologies

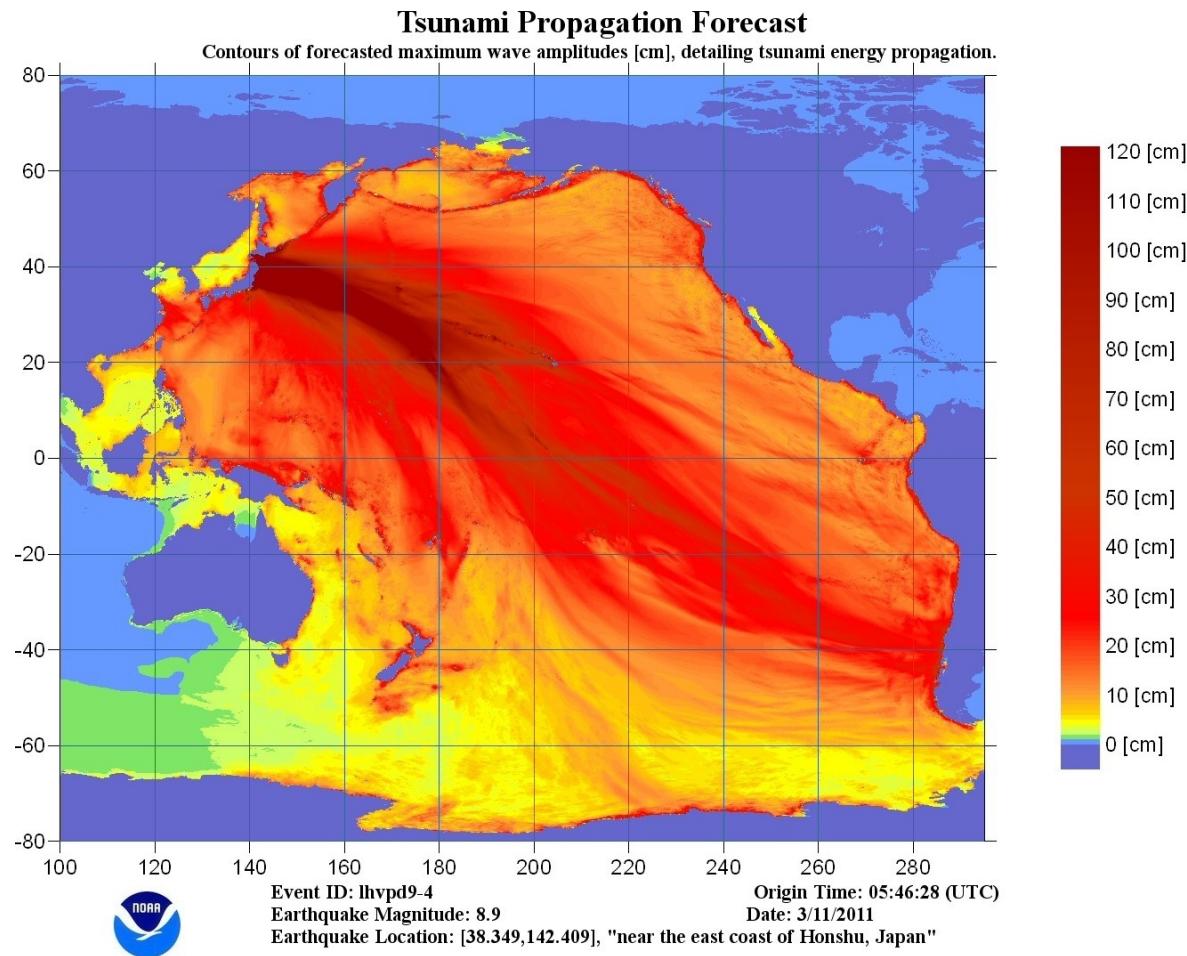
- Suppose you have a *lot* of processors
- How do you hook them up?
 - Ring/mesh/torus, 1-d, 2-d, 3-d?
 - Fully connected?
 - Tree hierarchy?
- How does this impact usability?



Parallelism Everywhere

UPPSALA
UNIVERSITET

- Disaster management





UPPSALA
UNIVERSITET

**REMEMBER TO REGISTER
FOR THE
EXAM**

<http://tenta.angstrom.uu.se/tenta/>



UPPSALA
UNIVERSITET

Floating point representation

How are numbers actually stored?

Some performance consequences and tricks...



Encoding Byte Values

Byte = 8 bits

Binary 00000000_2 to 11111111_2

Decimal: 0_{10} to 255_{10}

Hexadecimal 00_{16} to FF_{16}

- Base 16 number representation
- Use characters ‘0’ to ‘9’ and ‘A’ to ‘F’
- Write $FA1D37B_{16}$ in C as $0xFA1D37B$
 - Or $0xfa1d37b$

	Hex	Decimal	Binary
0	0	0000	
1	1	0001	
2	2	0010	
3	3	0011	
4	4	0100	
5	5	0101	
6	6	0110	
7	7	0111	
8	8	1000	
9	9	1001	
A	10	1010	
B	11	1011	
C	12	1100	
D	13	1101	
E	14	1110	
F	15	1111	



Machine Words

The computer has a “Word Size”

Nominal size of a “natural data unit”

- Including addresses
- Ideally register size == integer size == address size

Most older machines are 32 bits (4 bytes)

- Limits addresses to 4GB

New systems are 64 bits (8 bytes)

- Potentially address 1.8×10^{10} GB

Computers can often support multiple data formats

- Fractions or multiples of word size
- Always integral number of bytes



Data Representations

Sizes of C Objects (in Bytes)

C Data Type	Compaq Alpha	Intel IA32
• int	4	4
• long int	8	4
• char	1	1
• short	2	2
• float	4	4
• double	8	8
• long double	8	10
• void *	8	4

(Or any other pointer)

C standard defines “int” as “the most efficient integer type”

Use `sizeof(type)`



Specifying representation

The Standard C library header **<stdint.h>**

define types with a specific representation

int8_t, int16_t, uint8_t, uint32_t, ...

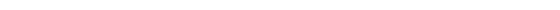
The include file **<limits.h>** defines constants for the maximum and minimum values

INT_MAX, INT_MIN, CHAR_MIN, CHAR_MAX, ...



Byte Ordering

- How should bytes within multi-byte word be ordered in memory?

`int64_t x = 0x26AC =`  or 
little BIG

- Conventions
 - SPARC, PowerPC tend to be “Big Endian”
 - x86 is “Little Endian”
 - Alpha, most PowerPC, MIPS, ARM, IA64 are all “switchable endian”
 - Code with strong x86 heritage tends to use little endian



Byte Ordering Example

BigEndian

Least significant byte has highest address

LittleEndian

Least significant byte has lowest address

Example

Variable `x` has 4-byte representation `0x01234567`

Address given by `&x` is `0x100`

BigEndian

`0x100 0x101 0x102 0x103`



LittleEndian

`0x100 0x101 0x102 0x103`





Converting between endians

The UNIX network header <netinet/in.h> contains functions for converting between “network order” (Big endian, TCP/IP standard) and “host order” (might be big or little endian)

```
unsigned long int htonl(unsigned long int hostlong);  
unsigned short int htons(unsigned short int hostshort);
```

```
unsigned long int ntohl(unsigned long int netlong);  
unsigned short int ntohs(unsigned short int netshort);
```

```
$ man htonl
```

htonl = “host to network long”



UPPSALA
UNIVERSITET

Casting in Endians

```
int a = 4;  
  
int* ptr = &a;  
  
short b = * (short*) ptr;  
  
if (a == b)  
{  
    printf("Little endian!");  
} // else big endian
```

A value that fits within a smaller type is represented correctly in LE.

Only applies to casting pointers. The compiler will always dereference a scalar cast "correctly".



Encoding Integers

signed int	msb		unsigned
127	0	1111111	127
126	0	1111110	126
2	0	0000010	2
1	0	0000001	1
0	0	0000000	0
-1	1	1111111	255
-2	1	1111110	254
-127	1	0000001	129
-128	1	0000000	128



Integers example

```
short int x = 15213;  
short int y = -15213;
```

C short is 2 bytes long and is *signed*

Signed integers are represented in *2's complement*

Most significant bit indicates sign

- 0 for nonnegative
- 1 for negative



UPPSALA
UNIVERSITET

Encoding Example (Cont.)

```
x = 15213:  
00111011 01101101  
y = -15213:  
11000100 10010011
```

- 2's complement is the *bitwise negation plus one*.



Unsigned & Signed Numeric Values

X	B2U(X)	B2T(X)
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	-8
1001	9	-7
1010	10	-6
1011	11	-5
1100	12	-4
1101	13	-3
1110	14	-2
1111	15	-1

- Equivalence
 - Same encodings for nonnegative values
- Uniqueness
 - Every bit pattern represents unique integer value
 - Each representable integer has unique bit encoding
- Addition, subtraction, and multiplication can be done regardless of sign.



Casting Signed to Unsigned

C Allows Conversions from Signed to Unsigned

```
short int          x = 15213;
unsigned short int ux = (unsigned short) x;
short int          y = -15213;
unsigned short int uy = (unsigned short) y;
```

Resulting Value:

No change in bit representation

Nonnegative values unchanged

- $ux == 15213$

Negative values change into (large) positive values

- $uy == 50323$



Signed vs. Unsigned in C

```
#define Constants
```

By default considered to be signed integers

Unsigned if “U” suffix is used:

- 0U, 4294967259U

Explicit casting between signed & unsigned

- int tx, ty;
- unsigned ux, uy;
- tx = (int) ux;
- uy = (unsigned) ty;

Implicit casting also occurs via assignments and procedure calls

- tx = ux;
- uy = ty;



Why Should I Use Unsigned?

Don't use just because you think a variable will never be negative.

- After all, it's "int main(**int** argc, char ** argv)"
- Unsigned ints in array subscripts may perform badly

Do Use When:

- Doing bit masks
- Performing modular arithmetic
- When you need the extra bit of range



Bit-Level Operations in C

Operations &, |, ^, ~ available in C

Apply to any “integral” data type

- long, int, short, char

View arguments as bit vectors, operations applied bit-wise

Examples (Char data type)

<code>~0x41</code>	\rightarrow	<code>0xBE</code>
<code>~01000001</code>	\rightarrow	<code>10111110</code>
<code>~0x00</code>	\rightarrow	<code>0xFF</code>
<code>~00000000</code>	\rightarrow	<code>11111111</code>
<code>0x69 & 0x55</code>	\rightarrow	<code>0x41</code>
<code>01101001 & 01010101</code>	\rightarrow	<code>01000001</code>
<code>0x69 0x55</code>	\rightarrow	<code>0x7D</code>
<code>01101001 01010101</code>	\rightarrow	<code>01111101</code>



Shift Operations

Left Shift:

 $x \ll y$

Shift bit-vector x left y positions

- Throw away extra bits on left
- Fill with 0's on right

Right Shift:

 $x \gg y$

Shift bit-vector x right y positions

- Throw away extra bits on right

Logical shift

- Fill with 0's on left

Arithmetic shift

- Replicate most significant bit on right
- Useful with two's complement integer representation

Argument x	01100010
<< 3	00010000
Log. >> 2	00011000
Arith. >> 2	00011000

Argument x	10100010
<< 3	00010000
Log. >> 2	00101000
Arith. >> 2	11101000



Bit fiddling

Set bit #n:

```
value |= (1 << n);
```

Swap bit #n:

```
value ^= (1 << n);
```

Clear bit #n:

```
value &= -1 ^ (1 << n);
```

-1 has all bits set



Swapping numbers

Should do:

```
int tmp = a;
```

```
a = b;
```

```
b = tmp;
```

- Popularly believed "efficient" (no temp variable):

```
a ^= b ^= a ^= b;
```

$$\begin{array}{rcl} x & & y \\ \hline 1010 \oplus 0011 & = & 1001 \rightarrow x \\ 1001 \oplus 0011 & = & 1010 \rightarrow y \\ 1001 \oplus 1010 & = & 0011 \rightarrow x \\ \hline & & 0011 \quad 1010 \end{array}$$

Diagram illustrating the bit-level operations for the swap. A green arrow points from the first row to the second, and a blue arrow points from the second row to the third. The final result shows the values swapped.

But this ruins pipelining, and the usual temporary variable is usually optimized away anyway



Binary fractions

UPPSALA
UNIVERSITET

Value	Representation
$5+1/4$	101.01
$2+5/8$	10.101
$63/64$	0.111111

Observations

Divide by 2 by shifting right

Multiply by 2 by shifting left

Numbers of form 0.11111... equal just below 1.0

$$1/2 + 1/4 + 1/8 + \dots \approx 1.0$$



Representable Numbers

Limitation of binary representation:

Can only exactly represent numbers of the form $x/2^k$

Other numbers have repeating bit representations

Value	Representation
1/3	0.0101010101[01]...
1/5	0.001100110011[0011]...
1/10	0.0001100110011[0011]...



IEEE floating point

IEEE Standard 754

Established in 1985 as uniform standard for floating point computation

Supported by all major CPUs

Established number formats and rules for how to treat the formats.

Can be hard to implement efficiently

- “Numerical Analysis – Computer Architecture: 1-0”



Floating Point Representation

Numerical Form: $-1s M 2^E$

- Sign bit **s** determines whether number is negative or positive
- Significand **M** normally a fractional value in range [1.0,2.0).
- Exponent **E** weights value by power of two



Most Significant Bit is sign bit

exp field encodes E

frac field encodes M



Floating Point Precision



Sizes:

Half precision: 5 exp bits, 10 frac bits

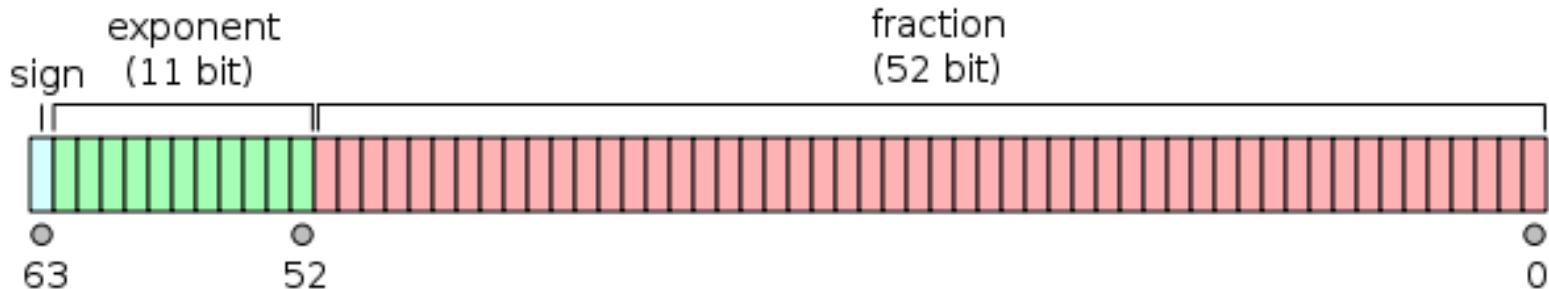
Single precision: 8 exp bits, 23 frac bits

Double precision: 11 exp bits, 52 frac bits

Quadruple precision: 15 exp bits, 112 frac bits



Double Precision



- Significand has 53 digits of accuracy *normally*
 - This extra digit = 1 unless exp = 0
- This gives 15 - 17 significant decimal digits precision
- Single precision gives 6-9 significant digits
- Quadruple precision gives 33-36 significant digits!!



UPPSALA
UNIVERSITET

Subnormalized Values

What happens with very small values?

Exp = 000...0

Frac begins to have leading zeroes:

- Numbers very close to 0.0 ($< 10^{-308}$ for doubles)
- Significand loses precision as value gets smaller
- “Gradual underflow”



Special Values

What happens with very large values?

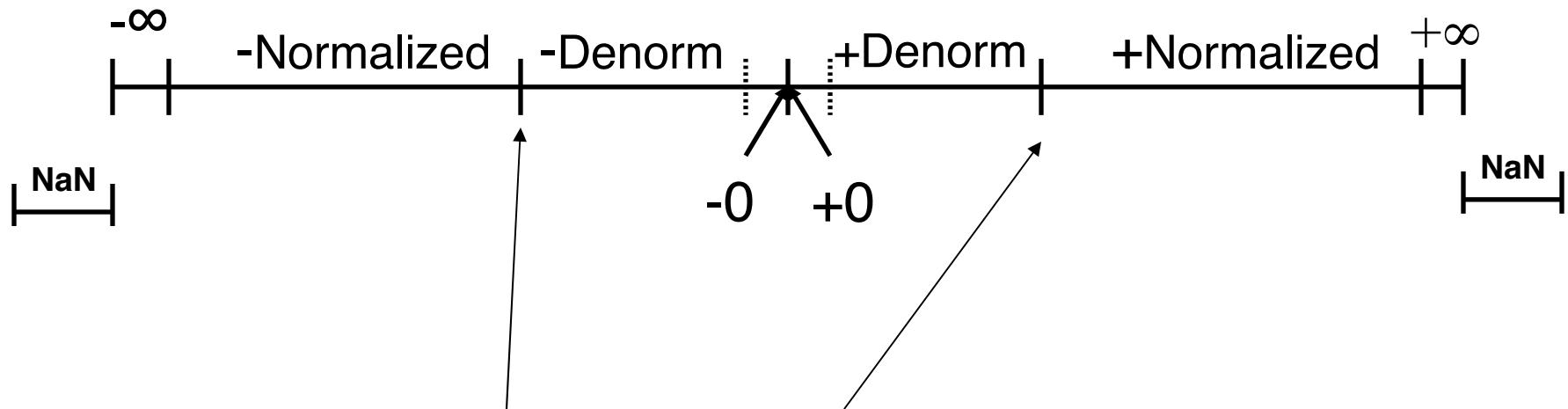
$\exp = 111\dots1$

Cases

- $\exp = 111\dots1, \text{frac} = 000\dots0$
 - Represents value infinity, ∞ , inf
 - Operation that overflows
 - Both positive and negative
 - E.g. $1.0/0.0 = -1.0/-0.0 = +\infty, -1.0/0.0 = -\infty$
- $\exp = 111\dots1, \text{frac} \neq 000\dots0$
 - Represents Not-a-Number, NaN
 - Represents case when no numeric value can be determined
 - E.g. $\sqrt{-1}, 0/0$



Summary of IEEE Floating Point Encodings



Switch between normalized and denormalized



UPPSALA
UNIVERSITET

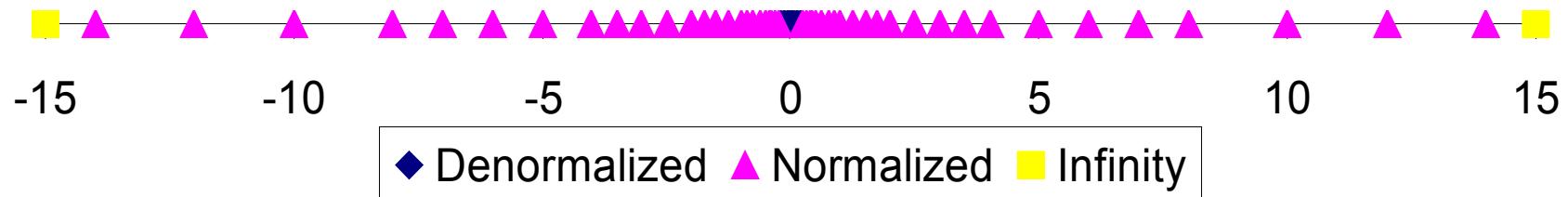
Distribution of Values

6-bit IEEE-like format

e = 3 exponent bits

f = 2 fraction bits

Bias is 3



Notice how the distribution gets denser toward zero.



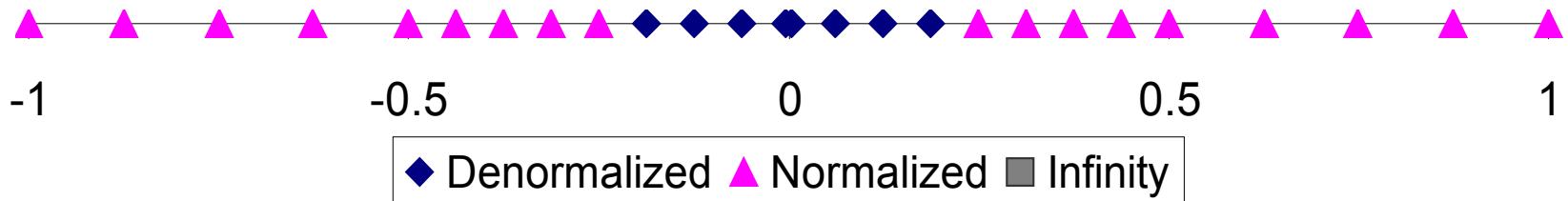
Distribution of Values (close-up view)

6-bit IEEE-like format

$e = 3$ exponent bits

$f = 2$ fraction bits

Bias is 3





Distribution of Values (Double Precision)

- Between $2^{52}=4,503,599,627,370,496$ and $2^{53}=9,007,199,254,740,992$ the representable numbers are exactly the integers.
- From 2^{53} to 2^{54} , everything is multiplied by 2, so every other integer is representable.
- For the range from 2^{51} to 2^{52} , the spacing is $\frac{1}{2}$ integers.
- The spacing as a fraction of the numbers in the range from 2^n to 2^{n+1} is 2^{n-52} .
- The maximum *relative* rounding error when rounding a number to the nearest representable one (the machine epsilon) is $2^{-53} \approx 10^{-16}$.



Patriot missile failure

- Velocity = $(x_2 - x_1)/(t_2 - t_1)$
- Suppose t measured in seconds, and the machine is turned on for a week?



SCUD Missile

vs.



Patriot Missile



Floating Point in C

C Guarantees Two Levels

`float` single precision

`double` double precision

Conversions

Casting between `int`, `float`, and `double` changes numeric values

`double` or `float` to `int`

Truncates fractional part

Not defined when out of range

`int` to `double`

Exact conversion, as long as `int` is \leq 53 bits



<float.h>

`FLT_MANT_DIG, FLT_RADIX,`

Number of digits in the mantissa,
and exponent

`FLT_MIN_EXP, FLT_MAX_EXP`

Maximum and minimum values of
exponent in binary

`FLT_MIN_10_EXP,`
`FLT_MAX_10_EXP`

Maximum and minimum values of
exponent in decimal

`FLT_EPSILON`

Machine epsilon

`FLT_ROUNDS`

Rounding mode

`FLT_EVAL_METHOD` (c99 only)

Intermediate representation used for
arithmetic



FLT_ROUNDS

- 1 Indeterminable
- 0 Towards zero
- 1 To nearest
- 2 Positive infinity
- 3 Negative infinity
- 4- Implementation dependent



FLT_EVAL_METHOD (c99)

- 1 Indeterminable
- 0 Evaluate all operations and constant just to the precision of the type
- 1 Evaluate operations and constants of type float and double to the range and precision of the double type
- 2 Evaluate all operations and constants to the range and precision of the long double type
- 3 Implementation dependent



fenv.h (c99)

UPPSALA
UNIVERSITET

Defines the floating-point environment

Get/set rounding modes

Toggle floating point exceptions for division by zero, inexact, invalid, overflow and underflow



One step back

Floating-point division and square root are *slow*

So are *log, exp, pow, sin, cos,...*

These tend to be implemented as Newton-Raphson or repeated ranges of polynomial approximations

Tip: Use squared distances unless `sqrt` is *absolutely* needed



Reciprocal square root

Super useful!

Normalize to the unit vector: $\hat{r} = \frac{\vec{r}}{|\vec{r}|} = \frac{\vec{r}}{\sqrt{x^2 + y^2}}$

Also $\text{rsqrt}(x) * x = \sqrt{x}$

Compute by Newton-Raphson

What about the starting guess?

- Used to be from lookup table, now we use *fancy trickery*



UPPSALA
UNIVERSITET

“Fancy Trickery” (from Quake III)

```
float Q_rsqrt( float number )
{
    long i;
    float x2, y;
    const float threehalfs = 1.5F;

    x2 = number * 0.5F;
    y = number;
    i = * ( long * ) &y;                                // evil floating point bit level hacking
    i = 0x5f3759df - ( i >> 1 );                      // what the fuck?
    y = * ( float * ) &i;
    y = y * ( threehalfs - ( x2 * y * y ) );          // 1st iteration
//    y = y * ( threehalfs - ( x2 * y * y ) );          // 2nd iteration, this can be removed

    return y;
}
```



UPPSALA
UNIVERSITET

Bits will always be bits (from Hacker's Delight)

```
/* This is a very approximate but very fast version of rsqrt. It is just
two integer instructions (shift right and subtract), plus instructions
to load the constant.
```

```
The constant 0x5f37642f balances the relative error at +-0.034213. */
```

```
float rsqrt2(float x) {
    int i = *(int *)&x;                  // View x as an int.
    i = 0x5f37642f - (i >> 1);        // Initial guess.
    x = *(float *)&i;                 // View i as float.
    return x;
}
```



Selecting FP type for performance

- Doubles:
 - Pro: Doubles incur less roundoff error
 - Pro: Doubles have a larger range
 - Con: Doubles take up $\frac{1}{2}$ of a 128-bit register
 - Con: Doubles require 2x bandwidth
 - Con: Doubles consume 2x cache
- Conclusion: SP is preferred where possible
- Consider Single Precision computation as an optimization method



Single Precision as optimization

- A basic program may use DP as default
- Then:
 - Convert performance-critical parts to SP, or
 - Use SP to get “rough” answer, then refine using DP, kind of like an iterative method



Single Precision as optimization

- Intuitively, you can:
 - Compute a 32 bit *result*,
 - Calculate a *correction* to 32 bit result using selected higher precision and
 - Perform the *update* of the 32 bit results with the correction using high precision.
- Moreover, by limiting your use of SP you can be more aware and responsible about managing round-off error



Directory info

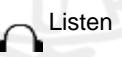
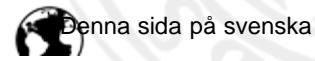
No information available for "Marcus Holm"

Search for information about a person in the directories for IT, Uppsala University, and students at Uppsala University.
Write the name of the person (poss. abbrev.).

Name:

▼ Department of Information Technology

Uppsala University / Information Technology / Contact / Contact - persons & functions



Contact - persons and functions

Head of department

Lina von Sydow, +46 (0)18 471 2785, prefekt@it.uu.se

Robin Strand, Deputy Head of dept., +46 (0)18 471 3469 , prefekt@it.uu.se

Head of Research

Gunilla Kreiss, forskningsprefekt@it.uu.se

Head of Education

Tobias Wrigstad, +46 (0)18 - 471 1072, utbildningsprefekt@it.uu.se

Administrators

Lists sorted by [task](#).

Head of Divisions

- Joachim Parrow, Computing Science Division, +46 (0)18 471 5704
- Philipp Rümmer, Computer Systems, +46 (0)18 471 3156
- Stefan Engblom, Systems and Control, +46 (0) 18-471 2754

- [Emanuel Rubensson](#), Scientific Computing, +46 (0)18 471 7893
- [Robin Strand](#), Visual Information and Interaction, +46 (0)18 - 471 3469

Registrar

[Elisabeth Lindqvist](#) is registrar at the department. Deputy is [Loreto Skoknic](#).

E-mail for letters to registrar registrator@it.uu.se.

Please note that undergraduate students asking for registration to courses should contact [Student service](#)

Archivist

Archivist for the department is [Loreto Skoknic](#).

Student services including counselling

[See their web page](#).

Director of PhD studies

[Pierre Flener](#), +46 (0)18 - 471 1028, Director-PhD-Studies@it.uu.se

Collaboration

Contact for collaboration Ida-Maria.Sintorn.samverkan@it.uu.se [webpage](#)

Gender equity group

[See special page](#).

Contact person for diversity issues (mångfaldsfrågor)

[Anna-Lena Forsberg](#)

Contact person for gender-related violations

[Anna-Lena Forsberg](#) or [Karolina Malm Holmgren](#)

Directors

- [Hans Karlsson](#), SNIC
- [Elisabeth Larsson](#), UPPMAX

Environmental officer

- [vacant](#)

Media contact

- [Lina von Sydow](#), Head of Department, +46 (0)18 471 2785, prefekt@it.uu.se 
- [Gunilla Kreiss](#), Head of Research, forskningsprefekt@it.uu.se 
- [Tobias Wrigstad](#), Head of Education, utbildningsprefekt@it.uu.se 

Updated 2020-09-07 09:30:33 by [Elizabeth Neu Morén](#).

CONTACT US

[Contact persons/functions](#)

[Staff](#)

VISIT US

[Adresses](#)

[Map](#)

SHORTCUTS

[Evacuation Plan](#)

[Emergency Contacts](#)

[Report an Error](#)

[Edit this page](#)

Uppsala universitet använder kakor (cookies) för att webbplatsen ska fungera bra för dig. Läs mer om kakor. [OK](#)

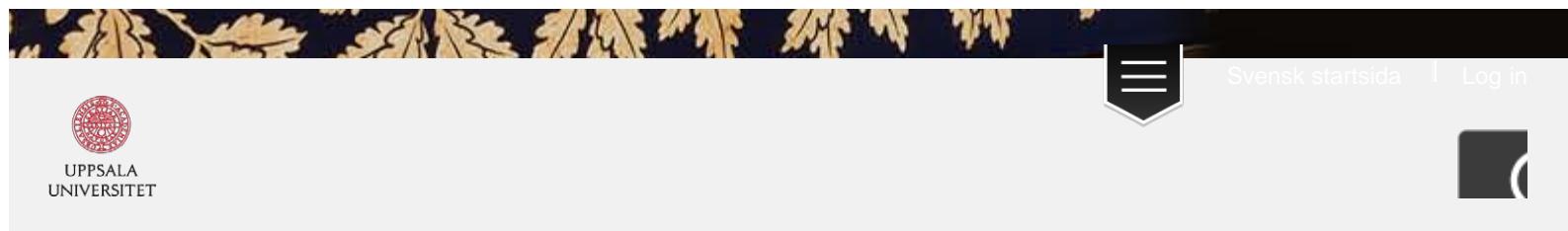


© 2020 Uppsala University | Phone: +46 18 471 00 00 | P.O. Box 337, SE-751 05 Uppsala, Sweden

Registration number: 202100-2932 | VAT number: SE202100293201 | [Registrar](#) | [About this website](#) | Editor: [Kajsa Örjavi](#).

[GO TO TOP](#)





UPPSALA
UNIVERSITET



Svensk startsida | Log in



▼ Department of Information Technology

Uppsala University / Information Technology / Search for staff



Denna sida på svenska



Listen

Directory of people working at the department

Our directory is based on information from the university directory database, enhanced with local information such as personal presentation, picture, and functions at the department. The information can be shown sorted in several ways:

- [by employment type](#)
- [by last name \(brief version\)](#)
- [by first name \(brief version\)](#)
- [by building and floor](#)
- [as a portrait gallery](#) (may take a while to load)

It is also possible to [search the directory](#). When searching, you don't have to spell precisely, and can also abbreviate names: e.g., a search for "b viktor" will find "Björn Victor".

Name:

The information is gathered from the university directory database, and most of it can be edited in [Medarbetarportalen](#).

When changing phone extensions, always contact [Marina Nordholm](#).



Updated 2020-08-21 16:14:45 by [Ulrika Andersson](#).

CONTACT US

[Contact persons/functions](#)

[Staff](#)

VISIT US

[Adresses](#)

[Map](#)

SHORTCUTS

[Evacuation Plan](#)

[Emergency Contacts](#)

[Report an Error](#)

[Edit this page](#)

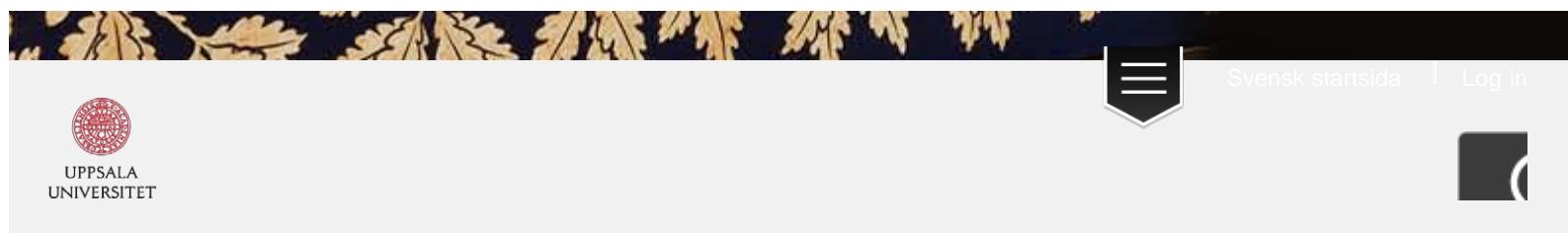
Uppsala universitet använder kakor (cookies) för att webbplatsen ska fungera bra för dig. [Läs mer om kakor.](#) [OK](#)



© 2020 Uppsala University | Phone: +46 18 471 00 00 | P.O. Box 337, SE-751 05 Uppsala, Sweden
Registration number: 202100-2932 | VAT number: SE202100293201 | [Registrar](#) | [About this website](#) | Editor: [Kajsa Örjavi](#)k.

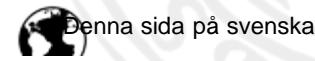
[GO TO TOP](#)





▼ Department of Information Technology

Uppsala University / Information Technology / Contact



Addresses

Visiting address

Department of Information Technology
Polacksbacken (Lägerhyddsvägen 2)
Uppsala

[Travel directions to the campus area by car.](#)

[To get here by bus, train etc or by walking from Uppsala city.](#)

[Map](#) of the Polacksbacken area.

[Campus map](#).

[Information about Uppsala](#).

Delivery address

Lägerhyddsvägen 2
SE-752 37
Uppsala, SWEDEN

With delivery address, please note (if possible):

Uppsala University: [Department/Unit]

Recipient: [Name]

Building: [1, 2, or 4]

Contact Info: [tel./email]

Invoice address

Reference on invoice:106

Uppsala universitet

PG1254

737 84 Fagersta

Postal address

Uppsala University

Department of Information Technology

Box 337

SE-751 05 Uppsala

Sweden

Fax

+46 (0)18 511925 (Please note that some of the subdepartments have their own faxnumbers, but this number is the official one.)

The ITC-area

The department is located at [the Information Technology Centre \(ITC\) at Polacksbacken campus](#).

Updated 2018-01-23 13:58:06 by [Peter Waites](#).

CONTACT US

[Contact persons/functions](#)

[Staff](#)

VISIT US

[Adresses](#)

[Map](#)

SHORTCUTS

[Evacuation Plan](#)

[Emergency Contacts](#)

[Report an Error](#)

[Edit this page](#)

Uppsala universitet använder kakor (cookies) för att webbplatsen ska fungera bra för dig. [Läs mer om kakor.](#) OK



© 2020 Uppsala University | Phone: +46 18 471 00 00 | P.O. Box 337, SE-751 05 Uppsala, Sweden

Registration number: 202100-2932 | VAT number: SE202100293201 | [Registrar](#) | [About this website](#) | Editor: [Kajsa ÖrjaviK.](#)

[GO TO TOP](#)





UPPSALA
UNIVERSITET

**REMEMBER TO REGISTER
FOR THE
EXAM**

<http://tenta.angstrom.uu.se/tenta/>



HPC Libraries

UPPSALA
UNIVERSITET

- HPC languages
- Things to consider about libraries
- Sources of libraries
- Example libraries



HPC Languages

UPPSALA
UNIVERSITET

- C
- C++
- Java
- Python
- Julia
- Matlab
- R



Language Performance

	Julia 3f670dao	Python 2.7.1	Matlab R2011a	Octave 3.4	R 2.14.2	JavaScript V8 3.6.6.11
<code>fib</code>	1.97	31.47	1336.37	2383.80	225.23	1.55
<code>parse_int</code>	1.44	16.50	815.19	6454.50	337.52	2.17
<code>quicksort</code>	1.49	55.84	132.71	3127.50	713.77	4.11
<code>mandel</code>	5.55	31.15	65.44	824.68	156.68	5.67
<code>pi_sum</code>	0.74	18.03	1.08	328.33	164.69	0.75
<code>rand_mat_stat</code>	3.37	39.34	11.64	54.54	22.07	8.12
<code>rand_mat_mul</code>	1.00	1.18	0.70	1.65	8.64	41.79

Figure: benchmark times relative to C++ (smaller is better).



Perform what?

```
function randmatstat(t)
n = 5
v = zeros(t)
w = zeros(t)
for i = 1:t
    a = randn(n,n)
    b = randn(n,n)
    c = randn(n,n)
    d = randn(n,n)
    P = [a b c d]
    Q = [a b; c d]
    v[i] = trace((P.'*P)^4)
    w[i] = trace((Q.*Q)^4)
end
std(v)/mean(v), std(w)/mean(w)
end
```



Julia random matrix statistics
benchmark.

Equivalent C++ benchmark

```
struct double_pair randmatstat(int t) {
    int n = 5;
    struct double_pair r;
    double *v = (double*)calloc(t, sizeof(double));
    double *w = (double*)calloc(t, sizeof(double));
    double *a = (double*)malloc(n*n*sizeof(double));
    double *b = (double*)malloc(n*n*sizeof(double));
    double *c = (double*)malloc(n*n*sizeof(double));
    double *d = (double*)malloc(n*n*sizeof(double));
    double *P = (double*)malloc(4*n*n*sizeof(double));
    double *Q = (double*)malloc(4*n*n*sizeof(double));
    double *PtP1 = (double*)malloc(n*n*sizeof(double));
    double *PtP2 = (double*)malloc(n*n*sizeof(double));
    double *QtQ1 = (double*)malloc(4*n*n*sizeof(double));
    double *QtQ2 = (double*)malloc(4*n*n*sizeof(double));
    for (int i=0; i < t; i++) {
        randmtzq_fill_random(a, n*n);
        randmtzq_fill_random(b, n*n);
        randmtzq_fill_random(c, n*n);
        randmtzq_fill_random(d, n*n);
        memcpy(P+0*n*n, a, n*n*sizeof(double));
        memcpy(P+1*n*n, b, n*n*sizeof(double));
        memcpy(P+2*n*n, c, n*n*sizeof(double));
        memcpy(P+3*n*n, d, n*n*sizeof(double));
        for (int j=0; j < n; j++) {
            for (int k=0; k < n; k++) {
                Q[2*n*j+k] = a[k];
                Q[2*n*j+n+k] = b[k];
                Q[2*n*(n+j)+k] = c[k];
                Q[2*n*(n+j)+n+k] = d[k];
            }
        }
        cblas_dgemm(CblasColMajor, CblasTrans, CblasNoTrans,
                    n, n, 4*n, 1.0, P, 4*n, 0.0, PtP1, n);
        cblas_dgemm(CblasColMajor, CblasNoTrans, CblasNoTrans,
                    n, n, 1.0, PtP1, n, PtP1, n, 0.0, PtP2, n);
        cblas_dgemm(CblasColMajor, CblasNoTrans, CblasNoTrans,
                    n, n, 1.0, PtP2, n, PtP2, n, 0.0, PtP1, n);
        for (int j=0; j < n; j++)
            v[i] += PtP1[(n+1)*j];
        cblas_dgemm(CblasColMajor, CblasTrans, CblasNoTrans,
                    2*n, 2*n, 2*n, 1.0, Q, 2*n, Q, 2*n, 0.0, QtQ1, 2*n);
        cblas_dgemm(CblasColMajor, CblasNoTrans, CblasNoTrans,
                    2*n, 2*n, 2*n, 1.0, QtQ1, 2*n, QtQ1, 2*n, 0.0, QtQ2, 2*n);
        cblas_dgemm(CblasColMajor, CblasNoTrans, CblasNoTrans,
                    2*n, 2*n, 2*n, 1.0, QtQ2, 2*n, QtQ2, 2*n, 0.0, QtQ1, 2*n);
        for (int j=0; j < 2*n; j++)
            w[i] += QtQ1[(2*n+1)*j];
    }
    free(PtP1);
    free(PtP2);
    free(QtQ1);
    free(QtQ2);
    free(P);
    free(Q);
    free(a);
    free(b);
    free(c);
    free(d);
    double v1=0.0, v2=0.0, w1=0.0, w2=0.0;
    for (int i=0; i < t; i++) {
        v1 += v[i];
        v2 += v[i]*v[i];
        w1 += w[i];
        w2 += w[i]*w[i];
    }
    free(v);
    free(w);
    r.s1 = sqrt((t*(t*v2-v1*v1))/((t-1)*v1*v1));
    r.s2 = sqrt((t*(t*w2-w1*w1))/((t-1)*w1*w1));
    return r;
}
```



UPPSALA
UNIVERSITET

Remember the HPC priorities

- 1) Correctness
- 2) Flexibility
- 3) Performance



C++ vs C

UPPSALA
UNIVERSITET

- Templates vs typedefs, code duplication, and macros
- Classes & methods vs structs
- Really, it's the template libraries in C++
- And C maps closely to machine code



Downsides of C/C++

- Error-prone programming
- Huge and ugly code
 - Hard to maintain
 - Hard to move
 - Hard to communicate
- Portability must be done explicitly



Java as HPC language

- Just-In-Time Compiler (JIT) is smart.
- Cross-platform
- Easier to use than C/C++
- Growing availability of HPC libraries
- Up to 90% as fast as C/C++
- Performance portable?



Python for HPC

- Use Python to *coordinate* computation codes
- Lots of ties to C/C++ and Fortran
- SciPy and NumPy libraries
- For interested students:
<http://skillsmatter.com/podcast/home/high-performance-python/js-1630>



Julia

UPPSALA
UNIVERSITET

- Easy like Python, fast like C?
- JIT compiled
- Message-passing parallelism
- Cloud computing mode
- Built-in C & Fortran libs for e.g. linear algebra and FFT's



Julia

UPPSALA
UNIVERSITET

- Go to <http://julia.forio.com/repl.htm>
- Try typing `plot(x->sin(x^2)/x, -2pi, 2pi)`
- Neat!



Matlab

UPPSALA
UNIVERSITET

- Uses Intel's MKL library for matrix operations
- Versatile but expensive packages
- Decent data plotting functions
- Very slow as a programming language
- Write your own computational kernels in an external language



The R Project for Statistical Computing

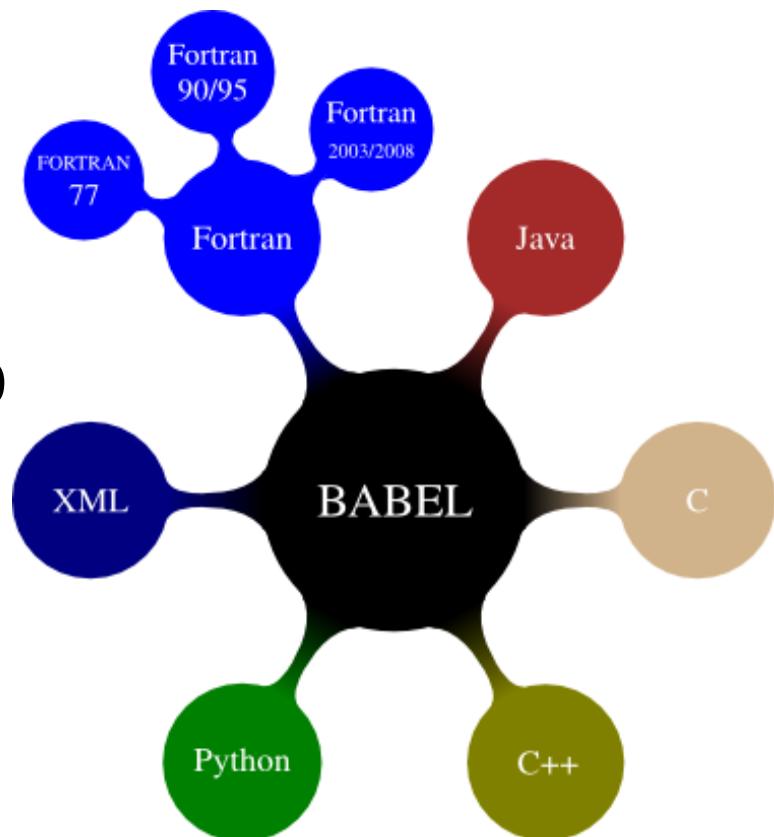
- Free software environment for statistical computing and graphics
- With Rcpp package:
 - Like Matlab, it can call C++
 - Supports “inline” C++ code



Babel

UPPSALA
UNIVERSITET

- High-Performance Language Interoperability
- Uses an Interface Definition Language to express software interfaces
- Babel generates glue code and skeletons

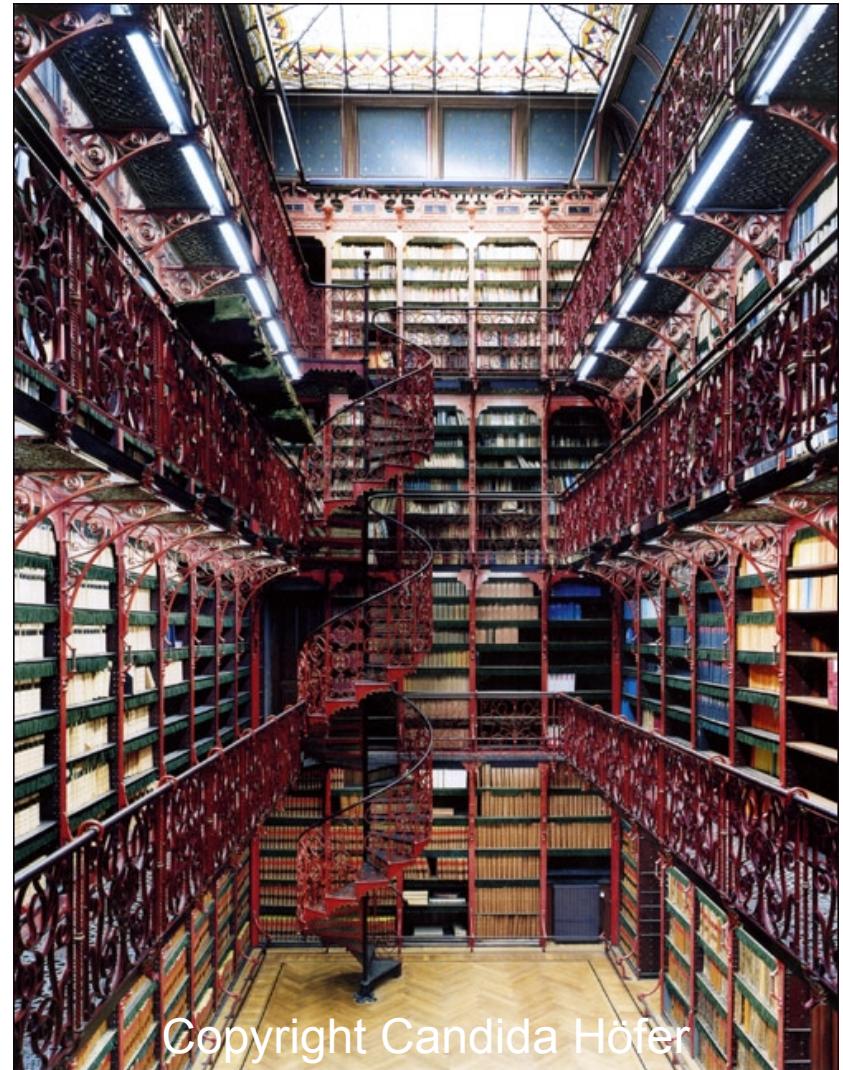




UPPSALA
UNIVERSITET

Things to consider about libraries

- Cost
- Licensing
- Performance
- Thread safety
- Stability
- Feature-completeness
- Update frequency
- Generality
- Interface



Copyright Candida Höfer



UPPSALA
UNIVERSITET

The general case

- Some scientific computing libraries are all-purpose
- Typically, the more specialized, the more up-to-date
- But all-purpose libraries can be a good first effort



UPPSALA
UNIVERSITET

Sources:

NUMERICAL RECIPES™

The Art of
Scientific Computing

Third Edition

- nr.com, a classic source of source-code
- Older editions in C and Fortran are free online
- A great go-to place for basic and understandable algorithms
- Not so advanced, not always fast



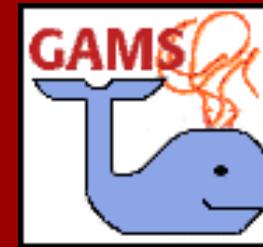
Sources: the Netlib



- Netlib.org
 - A huge repository:
http://www.netlib.org/master/expanded_liblist.html
 - Maintained by AT&T, Bell Labs, U Tennessee, Oak Ridge Nat'l Lab
 - It is OLD: Mostly Fortran code
 - Contains among others:
 - BLAS
 - EISPACK
 - LAPACK



Sources: GAMS



- Guide to Available Mathematical Software
- <http://gams.nist.gov/>
- Indexes Netlib and a set of commercial packages
- Provides a decision-tree for finding useful functions



UPPSALA
UNIVERSITET

Boost



- Truly general-purpose C++ libraries
- Over 80 peer-reviewed libraries
- Includes linear algebra, pseudorandom numbers, graphs, logic, data structures, ...
- Heavy use of templates
- Most libraries are header based and don't need compilation
- Several Boost libraries are headed for incorporation into standard C++



Boost



- What about Boost performance?
 - Boost reports comparisons with R with mixed results
- Should you use Boost?
 - “Use of [Boost] encourages people to *fit* a solution to a problem rather than *finding* a solution to a problem. The solution should always be appropriate for the data at hand and the constraints of the hardware, etc. Boost has an extremely narrow view of the "world" and its appropriate use is limited. Really, I discourage it because it leads programmers down the wrong direction right away, I often say **if you feel like you need [Boost] you probably don't really understand the problem that you're trying to solve.**” - Mike Acton (Engine Director at Insomniac Games)



UPPSALA
UNIVERSITET

NAG C Library



- Commercial product
- Large collection of mathematical and statistical algorithms
- Wide range of applicability: from Excel in Windows to Matlab in Linux
- Think “R but in a really fast library”



- Gnu Scientific Library
- Numerical library for mathematical routines
 - BLAS (through e.g. ATLAS)
 - Monte Carlo Integration
 - Differentiation
 - Differential Equations
 - Simulated Annealing
 - ...
- Distributed under GPL license.



SciPy

UPPSALA
UNIVERSITET

- Open-source
- Depends on NumPy
 - Fast N-dimensional arrays
 - Basic linear algebra
 - Basic Fourier transforms
- Reasonably wide range of functionality



Basic Linear Algebra Subprograms

Linear Algebra Package

- Lots of problems are solved in a linearized form
- Linear algebra software is incredibly well-optimized
- There are *many* BLAS packages
- They all have the same functions and interface
- How to decide which is right?
 - Intel MKL may be the best
 - ATLAS is free and is portable and is tunable
 - PLASMA – multicore
 - MAGMA - GPU



Basic Linear Algebra Subprograms

Linear Algebra Package

- Using a BLAS or LAPACK library:
- Different routines exist for most ordinary operations
- Usually must choose between:
 - Dense or sparse?
 - Single, double, complex?
 - Transpose or not?
- Think about if:
 - You want to take advantage of complicated structure in matrix
 - Preconditioning is necessary
 - The order you do the operations is important



Intel MKL

UPPSALA
UNIVERSITET

- Commercial product
- Probably the fastest in general
- Math Kernel Library
- Optimized for Intel processors
- C & Fortran
- Includes BLAS, LAPACK, also sparse solvers, FFT,..



GotoBLAS

UPPSALA
UNIVERSITET

- Discontinued project at U of Texas
- MKL-like speeds in many benchmarks
- Goto is the developer's last name, not "go to"
- Still in use on top supercomputers
- Hand-optimized assembly routines!?
- <http://www.utexas.edu/features/2006/goto/index.html>



Eigen

UPPSALA
UNIVERSITET

- Open Source
- Template Library for linear algebra
- Supports dense and sparse matrices
- Supports complex numbers
- Various matrix decompositions
- Good online documentation
- Quite fast!



ATLAS

UPPSALA
UNIVERSITET

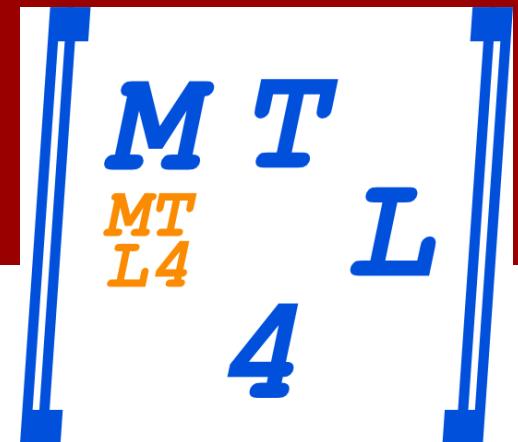
- Automatically Tuned Linear Algebra Software
- In Netlib, free
- Speed of BLAS3 routines comparable to MKL's
- Can be compiled as threaded or non-threaded
 - If you do your own threading, link to the serial interface and ATLAS' threading will be turned off.
- Recommended to use “architectural defaults”
 - Tuning for best performance is difficult and seldom necessary.



Some benchmarks

UPPSALA
UNIVERSITET

<http://eigen.tuxfamily.org/index.php?title=Benchmark>



- Matrix Template Library
- Open Source, “Supercomputing”, or GPU editions
- Supports dense and sparse matrices
- Enables natural expression of matrices in a C++ environment
- Performance not stellar, but not bad at all



Scientific Visualization

UPPSALA
UNIVERSITET

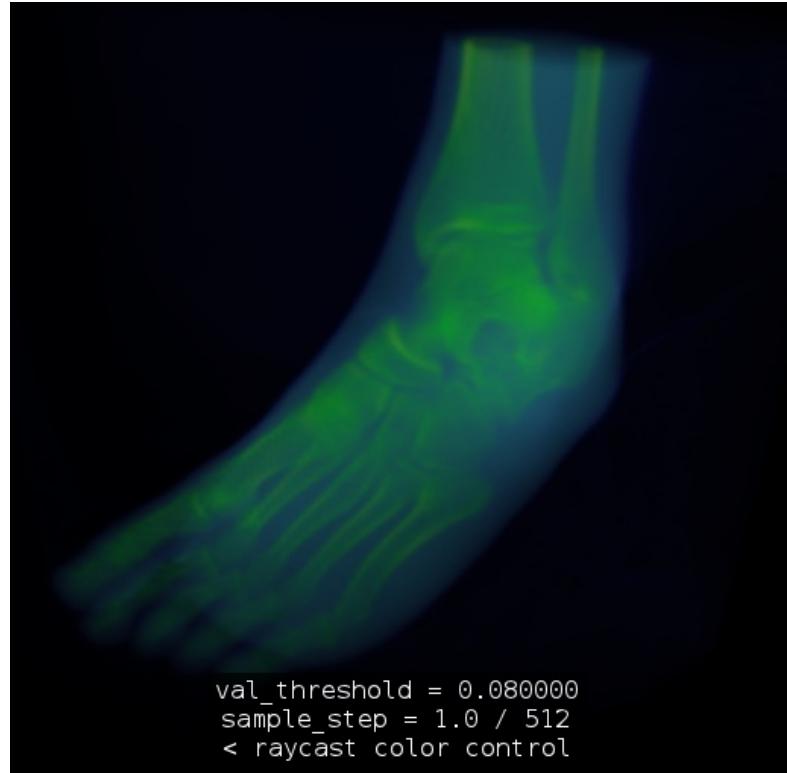
- So you have some data.
- What does it look like?
- A big pile of numbers looks like nothing at all.



UPPSALA
UNIVERSITET

VL, Visualization Library

- Open-source
- Wrapper for OpenGL
- Represent large datasets
(like a foot)
- Classes for things like
molecules

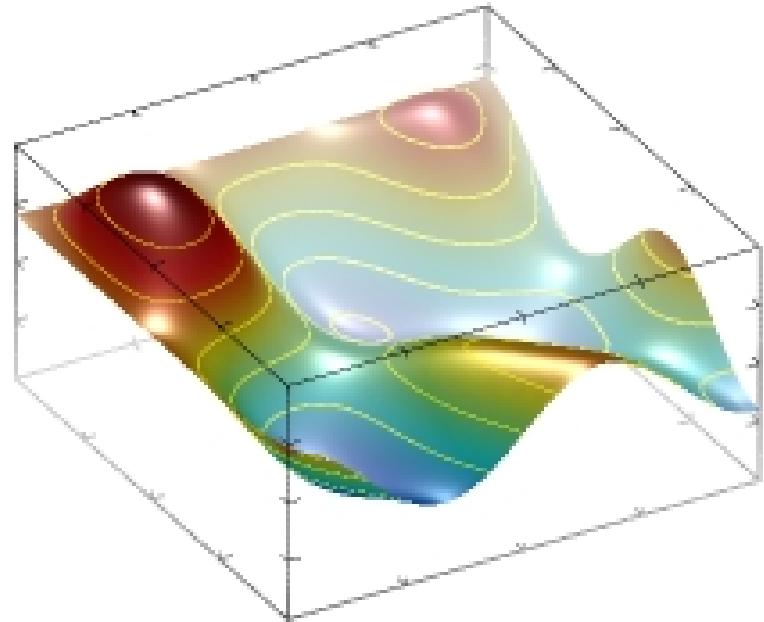




MathGL

UPPSALA
UNIVERSITET

- GPL license
- Interfaces with:
 - OpenGL
 - Qt
 - FLTK
- 55 general types of graphics
- Can be used from many languages and platforms

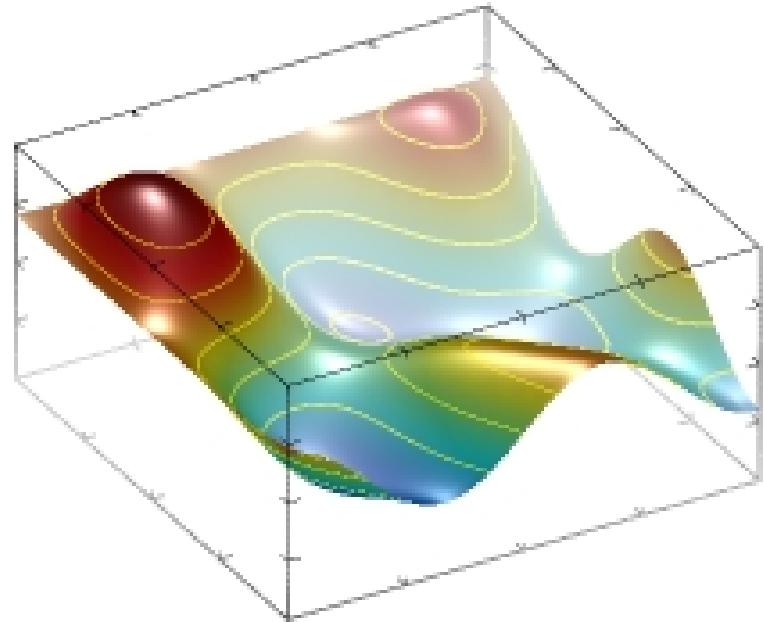




MathGL

UPPSALA
UNIVERSITET

- GPL license
- Interfaces with:
 - OpenGL
 - Qt
 - FLTK
- 55 general types of graphics
- Can be used from many languages and platforms





UPPSALA
UNIVERSITET

HPC Course Highlights

- Performance analysis
- Computer architecture
- Optimization techniques
 - Barnes-Hut assignment
 - Multigrid assignment
- Putting it all in context
- What does the future hold?



UPPSALA
UNIVERSITET

Performance Analysis

- Motivation:
 - Direct your optimization efforts
 - Communicate program characteristics
 - Make decisions about hardware acquisition



Performance Analysis

UPPSALA
UNIVERSITET

- Tools:
 - Unix `time` command
 - user time + system time = cpu time
 - `<time.h>` functions
 - Profilers
 - Insight into computer architecture



Performance Analysis

- There are more tools and techniques!
- For example:
 - Measure Cycles Per Instruction
 - Measure cache usage/misses/...
 - Measure memory bandwidth usage



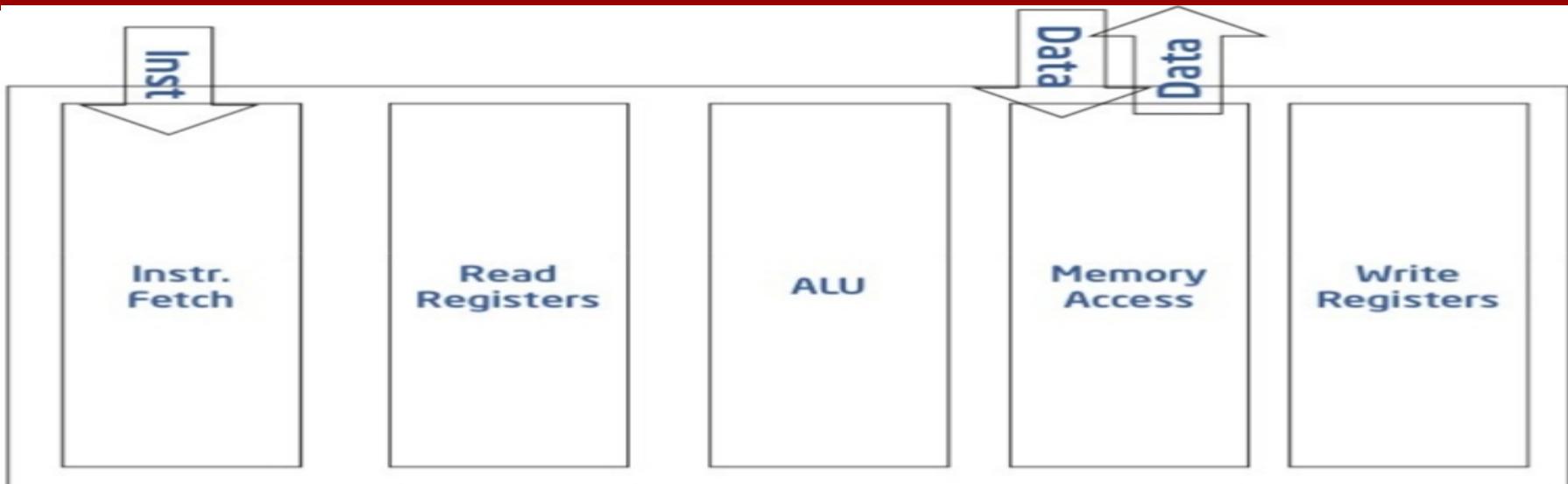
UPPSALA
UNIVERSITET

Computer architecture summary

- Fetch-Decode-Execute-Memory-Write
- Hierarchical memory structure
 - Proper cache usage is critical
- Superscalar Pipelined CPU with SIMD instructions
 - High Instruction-Level Parallelism yields low CPI
 - Vectorize by hook or by crook?



Fetch-execute cycle



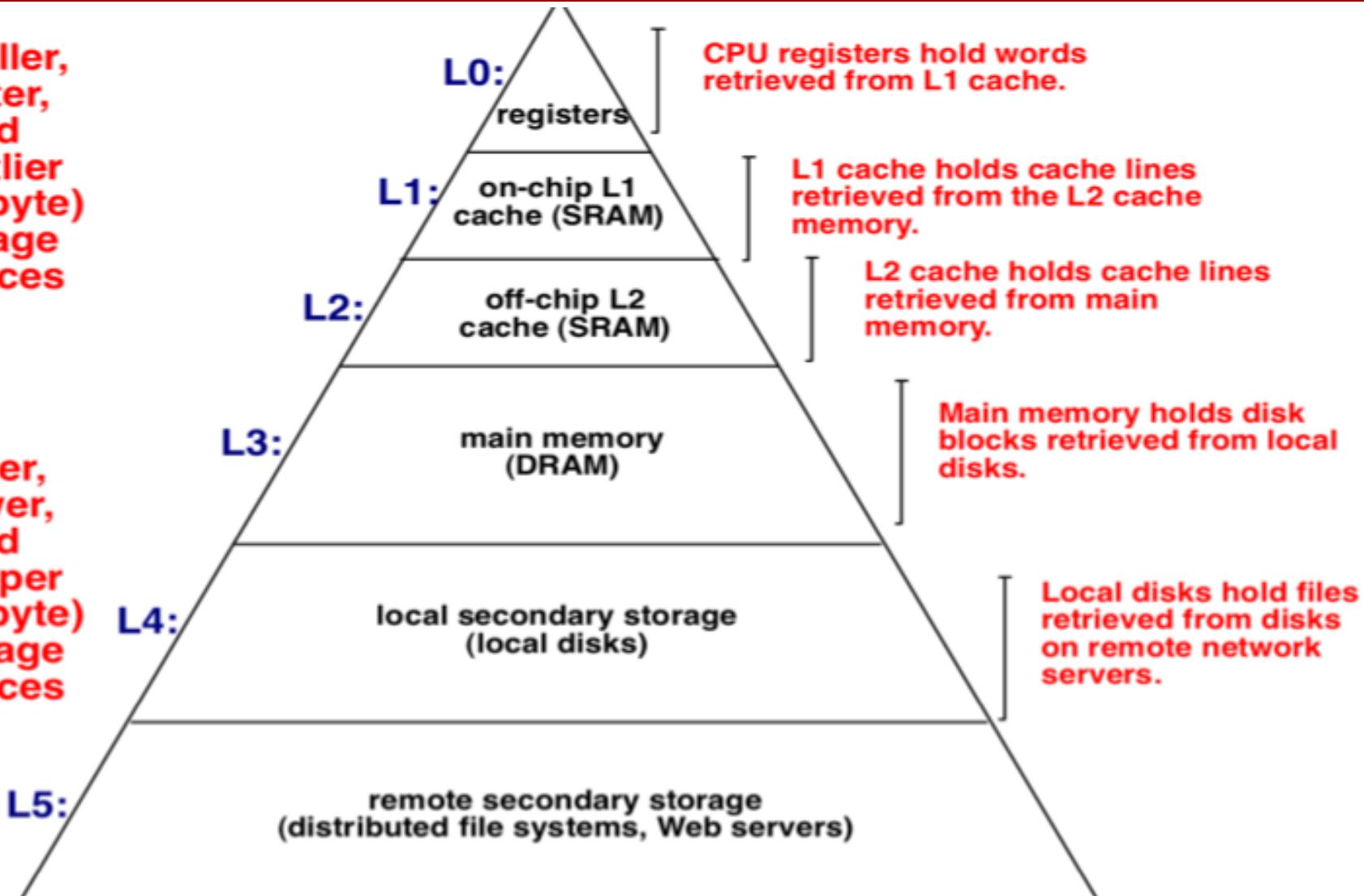
- Superscalar pipelining works best in large code blocks with many independent instructions:
 - Loop unrolling
 - Loop fusion



Memory hierarchy

**Smaller,
faster,
and
costlier
(per byte)
storage
devices**

**Larger,
slower,
and
cheaper
(per byte)
storage
devices**





Vector FPU

UPPSALA
UNIVERSITET

- SIMD vector registers
- Vectorize by hook
 - Intrinsics — wrappers for assembly code
- Vectorize by crook
 - Auto-vectorizing compiler (in -O3)



Multithreading

- Multicore and parallelism
- What is the maximum benefit?
 - Amdahl's Law
 - Work/span Law
- When can multithreading help?
 - Access more ALU's ...
 - ... to consume data faster



UPPSALA
UNIVERSITET

Floating Point Representation

- Floating point precision matters
- Single precision is usually 2x faster
- Double is numerically “safer”
- Use good judgment!



Optimization techniques

- Cache optimization — data locality
- Caches are complicated:
 - cache lines
 - associativity
 - hardware prefetching
- Analyze your *data access pattern*
- Use “block” algorithms



UPPSALA
UNIVERSITET

Optimization techniques

- Avoid work!
- Strength reduction
- Function inlining
- Do as little as possible in inner loop
- Use appropriate algorithm & data structure



UPPSALA
UNIVERSITET

Optimization techniques

- Help the compiler help you!
- Avoid function side-effects
 - Global variables
 - Use `restrict`
 - Use `const`
- Explore compiler optimization flags



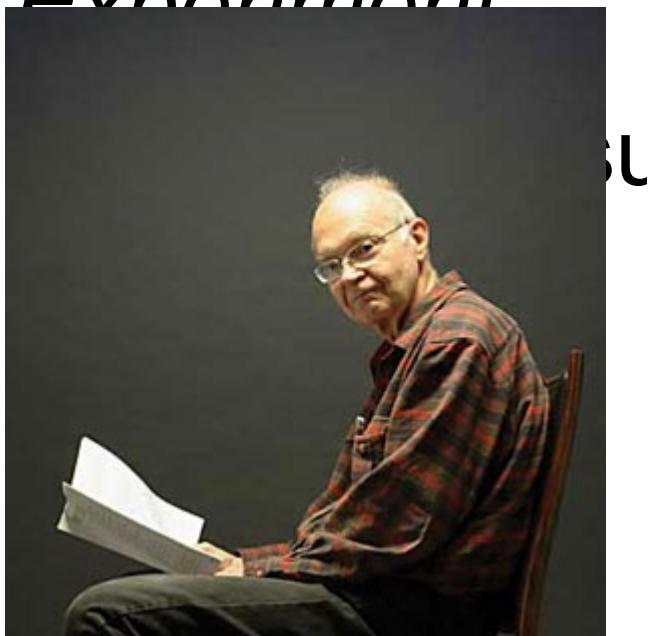
Optimization

UPPSALA
UNIVERSITET

- Bottom line:

- Experiment

- ...



Premature
optimisation
is the root
of all evil

Donald Knuth

suranyami



UPPSALA
UNIVERSITET

Assignment 2: Multigrid

- Huge algorithmic speedup
- Can be difficult to optimize
- Why?
- What did work?



Assignment 2: Barnes-Hut

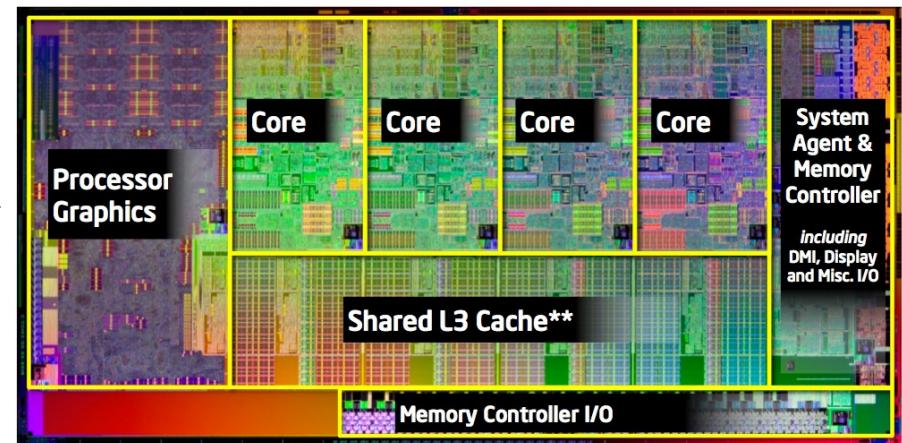
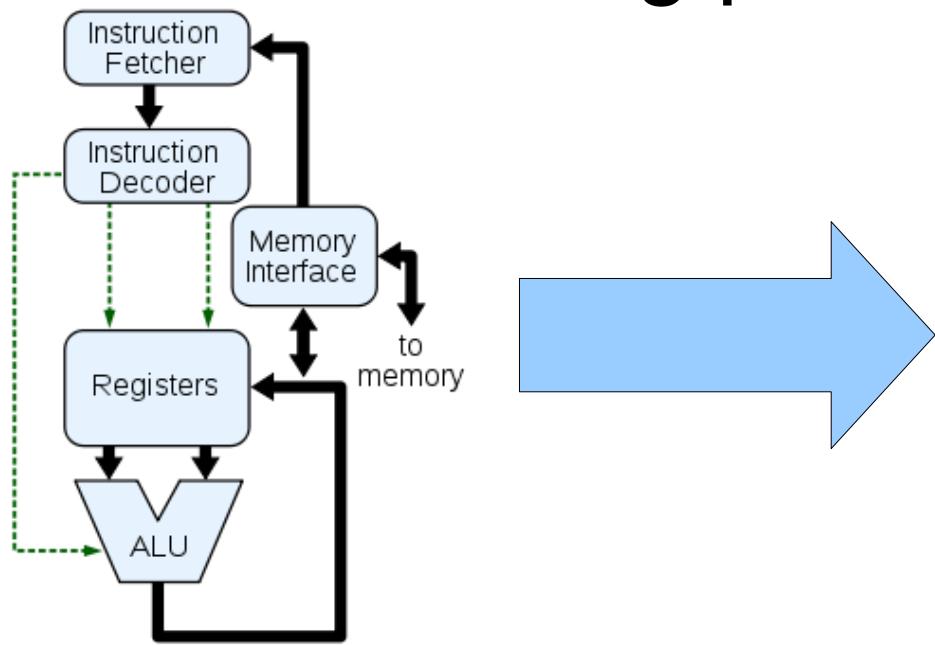
UPPSALA
UNIVERSITET

- Huge algorithmic speedup
- More optimization opportunities
- What did you do/try?



HPC in the 2010s

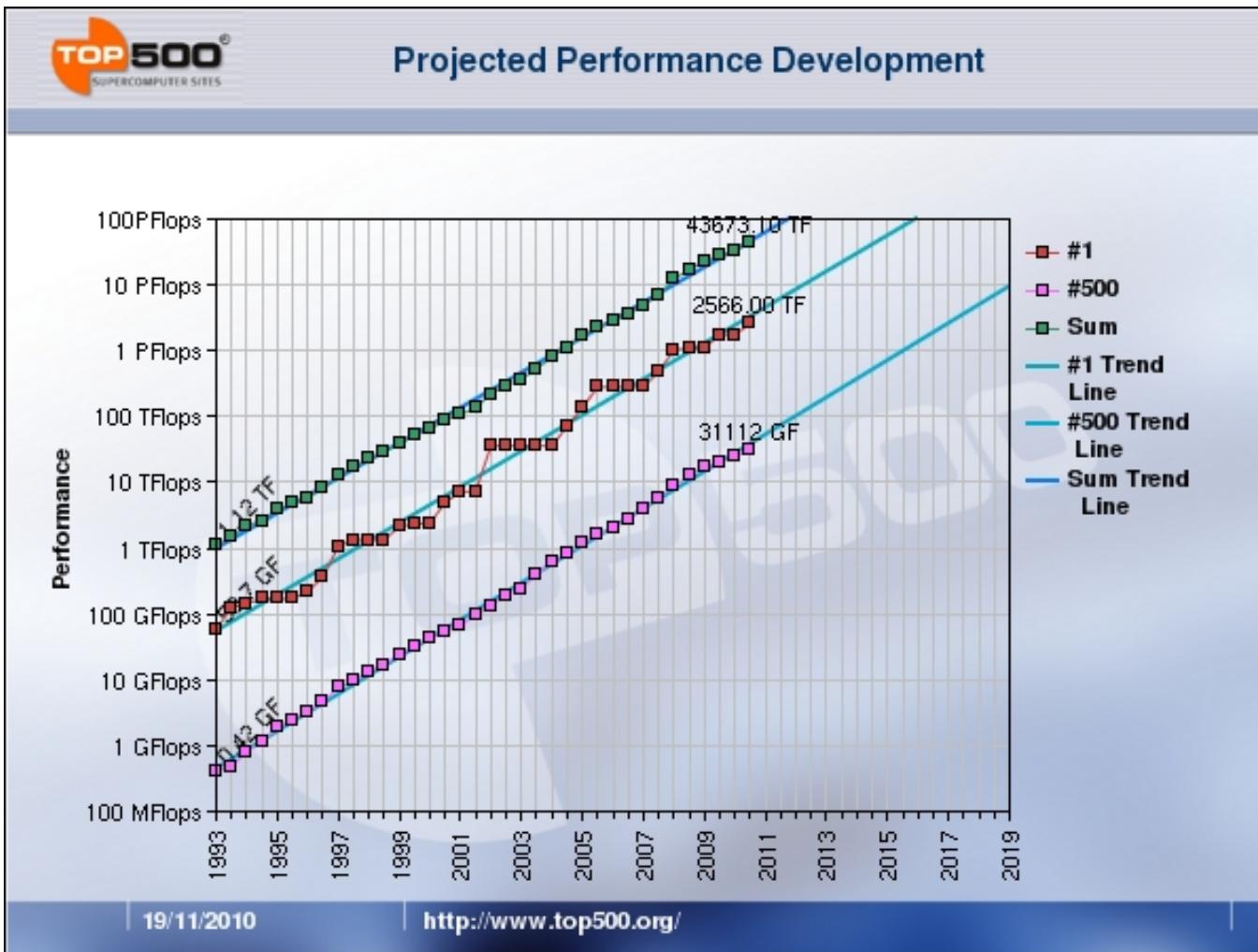
- Where is hardware heading?
 - Increasing complexity
 - Increasing parallelism





Moore's law

UPPSALA
UNIVERSITET

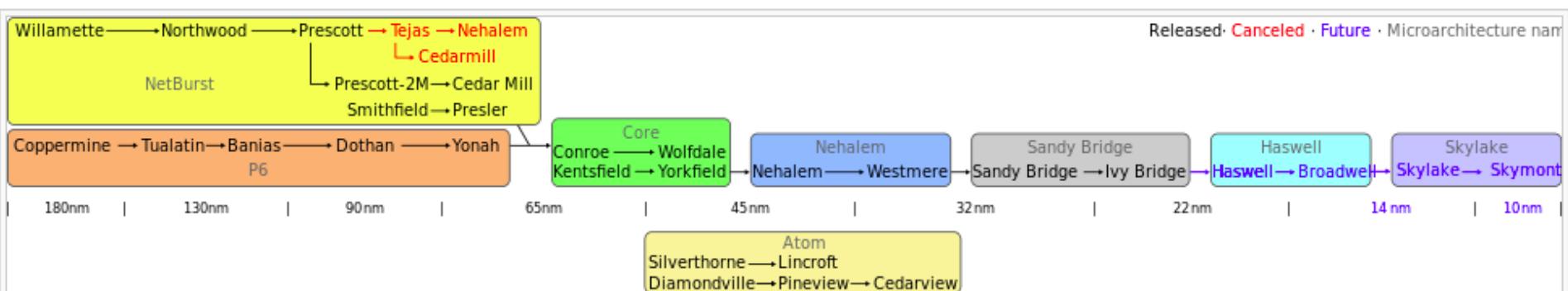


Source: top500.org – http://www.top500.org/lists/2010/11/performance_development



Intel Roadmap

UPPSALA
UNIVERSITET



Intel CPU core roadmaps from [NetBurst](#) and [P6](#) to [Skylake](#)



HPC in the 2010s

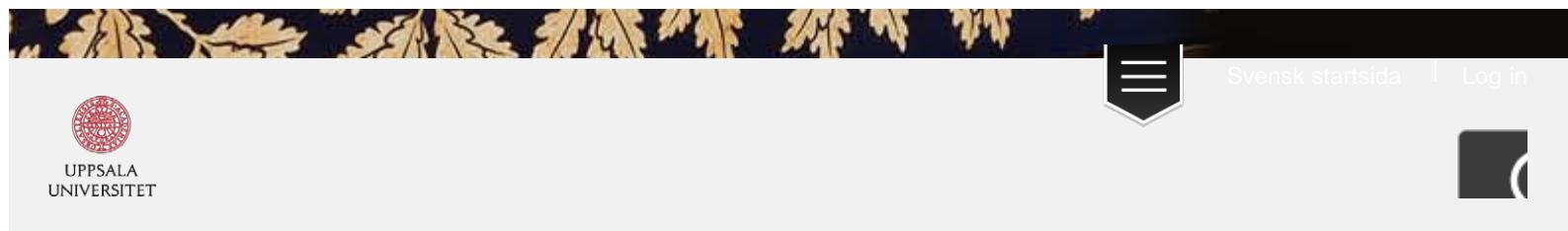
- Where is software heading?
 - Have to deal with all this complexity
 - More emphasis on *programmer efficiency*
 - *More automation*
- Uppsala Programming for Multicore Architectures Research Center (UPMARC)
 - New languages?
 - New programming paradigms?
 - New problems!



UPPSALA
UNIVERSITET

HPC in the 2010s

- Ubiquitous computing poses “new” constraints
- Power-efficient programming
- New applications of old tricks in novel embedded devices



▼ Department of Information Technology

Uppsala University / Information Technology /

Listen

Welcome to the course High Performance Computing and Programming

Computer systems have gone from being the exclusive domain of a few scientists and engineers who used them to speed up manual calculations, to being so common that they go practically unnoticed. Many applications that were thought of as impossible to perform, such as flow simulation around aircrafts, cellular video phones and digital movie characters, have now become things that we use or encounter every day. Many of these inventions were somewhat easy to imagine in theory, but to actually implement them required raw computer power of astronomical proportions at that time. A few years later, things that were impossible could actually be performed.

This is of course due to the rapid development of the computer hardware, but also, to a very high degree, due to developments in algorithms and their implementations on contemporary computers. Computer software are bound to different rules than computer hardware. First of all, the lifespan of many software products is much longer than the lifespan of most hardware. This fact puts software in a situation where a lot of effort is put into maintenance which requires that the software product is well structured, modular, well documented and built on standards. Unfortunately these notions often give rise to program code that cannot exploit all the latest features found in computer hardware. So there is a tradeoff between developing abstract reusable software modules and fast monolithic -type programs.

In this course we will try to understand how to do this tradeoff. How shall we build good, structured software that can compile into fast code for modern microprocessors? To be able to do this, we need first to learn how to identify actual and potential performance bottlenecks. Second, we need tools and knowledge to be able to do something with the code that helps. In brief the course will cover

- Basics of computer systems, from a programmers perspective
- Programming ANSI C
- Compiling and linking using open source and commercial compilers
- Tools such as compilers, debuggers, and make
- Profiling tools and methods for program benchmarking
- Methods for code optimization such as subroutine inlining, loop optimizations, cache blocking

Updated 2012-03-13 08:54:25 by [Marcus Holm](#).

CONTACT US

[Contact persons/functions](#)

[Staff](#)

VISIT US

[Adresses](#)

[Map](#)

SHORTCUTS

[Evacuation Plan](#)
[Emergency Contacts](#)
[Report an Error](#)
[Edit this page](#)

Uppsala universitet använder kakor (cookies) för att webbplatsen ska fungera bra för dig. [Läs mer om kakor.](#) OK



© 2020 Uppsala University | Phone: +46 18 471 00 00 | P.O. Box 337, SE-751 05 Uppsala, Sweden
Registration number: 202100-2932 | VAT number: SE202100293201 | [Registrar](#) | [About this website](#) | Editor: [Anneli Folkesson](#).

[GO TO TOP](#)



▼ Department of Information Technology

Uppsala University / Information Technology /

 Listen

Lectures

Lecture notes, assignments and laborations can be found in Studentportalen.

SCHEDULE

Note that the schedule might change, and this page may not be updated immediately.

Please check the [TimeEdit](#) page regularly for up-to-date information.

MH = Marcus Holm, ER = Elias Rudberg, ST = Salman Toor

Lecture 1

Lab 1 (Optional)

Lecture 2

Lecture 3

Lab 2

Lecture 4

Lab 3

Lecture 5

Lecture 6

Lecture 7

Lecture 8

Lecture 9

Lab 4

Lecture 10

Lecture 11

Lecture 12

Exam

TOPICS

Lecture 1

Lecture 2

Lecture 3

Lecture 4

Lectures 5&6

Lectures 7&8

Lecture 9

Lecture 10

Lecture 11

Lecture 12

Updated 2012-05-08 19:29:32 by [Marcus Holm](#).

CONTACT US

[Contact persons/functions](#)

[Staff](#)

VISIT US

[Adresses](#)

[Map](#)

SHORTCUTS

[Evacuation Plan](#)
[Emergency Contacts](#)
[Report an Error](#)
[Edit this page](#)

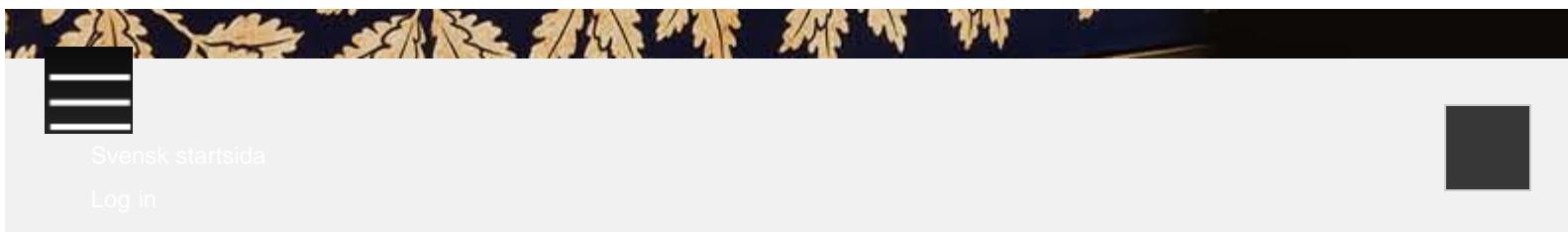
Uppsala universitet använder kakor (cookies) för att webbplatsen ska fungera bra för dig. [Läs mer om kakor.](#) [OK](#)



© 2020 Uppsala University | Phone: +46 18 471 00 00 | P.O. Box 337, SE-751 05 Uppsala, Sweden
Registration number: 202100-2932 | VAT number: SE202100293201 | [Registrar](#) | [About this website](#) | Editor: [Anneli Folkesson](#).

[GO TO TOP](#)





[Svensk startsida](#)

[Log in](#)

[Uppsala University](#) / [The University](#) / [Contact](#) / [Registrar](#)

[Listen](#)

The Office of the Registrar

The Office of the Registrar handles incoming mail, keeps a record of the public documents of the University, and answers questions from the general public. The Office is responsible for

- administration of the case and document management system for the entire University
- information about registered matters
- service to the general public.

Opening hours

Weekdays 09.00–11.30, 13.00–15.00

Weekdays between two public holidays: 10.00–12.00

Contact details

Tel.: +46 (0)18 471 00 00 (University switchboard)

E-mail: registrator@uu.se

Postal address: Uppsala University, Box 256, 751 05 Uppsala

Visiting address: Segerstedthuset (Segerstedt Building), Dag Hammarskjölds väg 7

Fax: +46 (0)18 471 20 00

[Staff att the Registrar's Office](#)



CONTACT THE UNIVERSITY

Telephone: +46 18 471 00 00

Contact the University

Find researchers & staff

FOLLOW UPPSALA UNIVERSITY ON



VISIT THE UNIVERSITY

Departments & units

Campuses

Museums & gardens

Map of Uppsala University

QUICK LINKS

Web shop

[News & media services](#)

[Library](#)

[Jobs & vacancies](#)

[University management](#)

[Support Uppsala University](#)

[International Faculty & Staff Services](#)

[Medarbetarportalen – Employee portal](#)

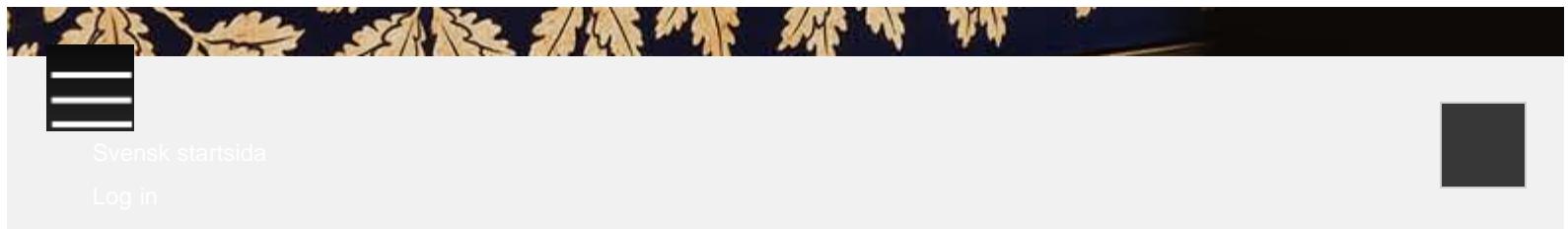
[Student portal](#)



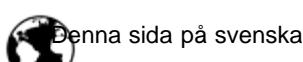
© Uppsala University | Tel.: +46 18 471 00 00 | P.O. Box 256, SE-751 05 Uppsala, SWEDEN

Registration number: 202100-2932 | VAT number: SE202100293201 | PIC: 999985029 | [Registrar](#) |

[About this website](#) | [Privacy policy](#) | Editor: webbredaktionen@uadm.uu.se



Uppsala University / About the website



Listen



About the Uppsala University website

Technical problems

If you have questions regarding the web servers, accessibility etc., send them to webmaster@uu.se.

Accessibility

This website is provided by Uppsala University. We want as many people as possible to be able to use the website. This website is partially compatible with the Act on the Accessibility of Digital Public Services. Contents that are not accessible are described in the [accessibility report](#) for uu.se.

Security matters

Contact security@uu.se to report computer trespassing, inappropriate web pages, unauthorised network use, etc.

How to use the ReadSpeaker Listen function

The [listen function](#) allows the text on the website to be read out loud to you.

Content

Heads of departments (or the equivalent) are responsible for information in their sphere of activity. For addresses to heads of all departments, see the [University Directory](#).

Information about cookies on the Uppsala University website

This website uses cookies. Under the Act Regarding Electronic Communication, which took effect on July 25, 2003, everyone who visits a website with cookies must be provided with information:

- that the website contains cookies,
- what these cookies are used for, and
- how cookies can be avoided.

In certain parts of the Uppsala University website, cookies are used to give the user access to extra functions, such as web mail and placing orders for materials.

If you do not accept the use of cookies, you will not be able to fully utilise the Uppsala University web site. Instead you can contact the Uppsala University Administration for help in ordering materials, etc., at postsorteringen@uadm.uu.se.

Web browsers can be configured to automatically refuse cookies or to inform you that a website contains cookies. Owing to the technology used by certain parts of the website, these parts of the website will not function without cookies.

CONTACT THE UNIVERSITY

Telephone: [+46 18 471 00 00](tel:+46184710000)

[Contact the University](#)

[Find researchers & staff](#)

FOLLOW UPPSALA UNIVERSITY ON



VISIT THE UNIVERSITY

[Departments & units](#)

[Campuses](#)

[Museums & gardens](#)

[Map of Uppsala University](#)

QUICK LINKS

[Web shop](#)

[News & media services](#)

[Library](#)

[Jobs & vacancies](#)

[University management](#)

[Support Uppsala University](#)

[International Faculty & Staff Services](#)

[Medarbetarportalen – Employee portal](#)

[Student portal](#)



© Uppsala University | Tel.: +46 18 471 00 00 | P.O. Box 256, SE-751 05 Uppsala, SWEDEN

Registration number: 202100-2932 | VAT number: SE202100293201 | PIC: 999985029 | [Registrar](#) |

[About this website](#) | [Privacy policy](#) | Editor: [David Naylor](#)

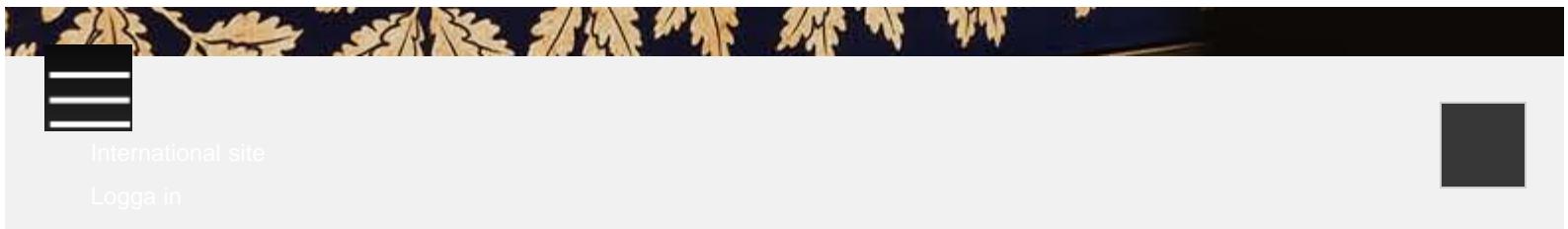


Directory info

No information available for "it:N98-143"

Search for information about a person in the directories for IT, Uppsala University, and students at Uppsala University.
Write the name of the person (poss. abbrev.).

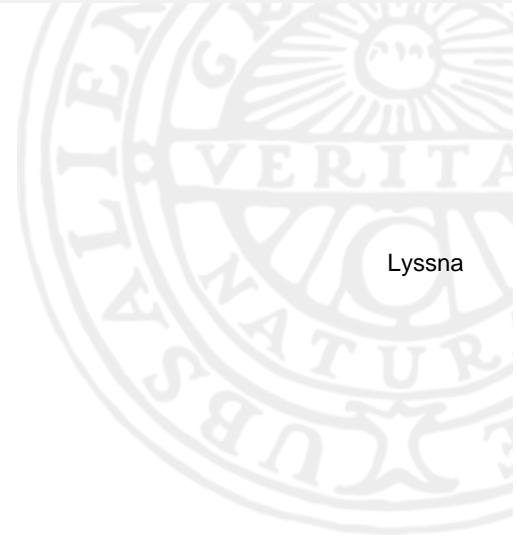
Name:



Uppsala universitet / Om webbplatsen

This page in English

Lyssna



Om Uppsala universitets webbplats

Personuppgifter och dataskydd

dataskyddsombud@uu.se

[Uppsala universitets hantering av personuppgifter](#)

Tillgänglighet

Uppsala universitet strävar efter att uu.se ska kunna uppfattas, förstas och hanteras av alla användare. Webbplatsen är delvis förenlig med lagen om tillgänglighet till digital offentlig service. Vilket innehåll som inte är tillgängligt redovisas i [tillgänglighetsredogörelsen för uu.se](#)

Tekniska problem

Kontakta webmaster@uu.se för frågor om webbservern, tillgänglighet och liknande.

Säkerhetsfrågor

Kontakta security@uu.se för rapportering om dataintrång, olämpliga webbsidor, otillbörligt nätnyttjande och liknande.

Använda lyssnafunktionen

Du kan få allt innehåll uppläst, [läs mer här.](#)

Innehåll

Prefekt eller motsvarande ansvarar för information inom sitt verksamhetsområde. För adressuppgifter gällande prefekter vid samtliga institutioner, se [Universitetskatalogen](#).

KAKOR PÅ UPPSALA UNIVERSITETS WEBBPLATSER

Uppsala universitets webbplatser använder kakor. En kaka eller cookie är en liten textfil som webbplatser lagrar på din dator och som innehåller någon information. Uppsala universitet sparar ingen personlig information i kakor. Om du inte vill ha några kakor kan du ställa in detta i din webbläsare.

Användningsområden

Kakor används för att samla in statistik över hur besökare använder universitetets webbplatser. På vissa delar av universitetets webbplatser används kakor för att ge användaren tillgång till extra funktioner, t ex webbpost, beställning av material, besöksstatistik samt för att bibehålla inloggningsinformation så att användaren skall kunna vara inloggad i till exempel Studentportalen och Medarbetarportalen.

Stänga av kakor i webbläsaren

Om du inte vill att webbplatsen använder kakor kan du stänga av funktionen i din webbläsares inställningar. Du kan även ställa in webbläsaren så att du får en varning varje gång webbplatsen försöker sätta en kaka på din dator. Vissa funktioner på universitetets webbplats kommer inte att fungera om du stänger av kakor helt.

Mer om våra analysverktyg för besöksstatistik

Uppsala universitets webbplats använder analysverktygen Google Analytics och Siteimprove för att få en bild av hur besökare använder webbplatsen. Syftet är att kunna förbättra innehåll, navigation och struktur på webbplatsen. Analysverktygen använder JavaScript och kakor, och den information som genereras av dessa genom din användning av webbplatsen (inklusive din IP-adress) kommer att vidarebefordras till leverantörerna Google och Siteimprove och lagras av dem på servrar i USA respektive EU. Google kan också överföra denna information till tredje parter om det krävs enligt lag eller i de fall en tredje part behandlar informationen för Googles räkning. Google kommer inte att koppla samman IP-adresser med andra data som Google innehåller.

Här hittar du mer information om vilken information våra analysverktyg sparar:

- [Google Analytics](#)
- [Siteimprove](#)

Genom att använda Uppsala universitets webbplats utan att avböja kakor från tredje

part, samtycker du till att Google och Siteimprove behandlar dina uppgifter på det sätt och för de syften som beskrivs ovan.

Om du inte vill att dina besök på Uppsala universitets webbplats ska finnas med i statistiken i Google Analytics så finns [ett tillägg som du kan installera i din webbläsare](#).

[Mer information om kakor](#)

KONTAKT

Telefon: [018-471 00 00](#)

[Kontakta universitetet](#)

[Hitta forskare & personal](#)

FÖLJ UPPSALA UNIVERSITET PÅ



HITTA HIT

[Institutioner & enheter](#)

[Våra campus](#)

[Museer & trädgårdar](#)

[Karta över Uppsala universitet](#)

GENVÄGAR

[Universitetets webbutik](#)

[Nyheter & press](#)

[Lediga jobb](#)

[Bibliotek](#)

[Mål & regler](#)

[Stöd Uppsala universitet](#)

[Pågående upphandlingar](#)

[Medarbetarportalen](#)

[Studentportalen](#)



© Uppsala universitet | Telefon: 018-471 00 00 | Box 256, 751 05 Uppsala

Organisationsnummer: 202100-2932 | Momsregistreringsnummer: SE202100293201 | PIC: 999985029 | Registrator |

[Om webbplatsen](#) | [Dataskyddspolicy](#) | Sidansvarig: Åke Johansson



Evacuation plan

Department of Information Technology

When the alarm signal sounds

Use *the nearest* exit and go to *the nearest* meeting-place.

Note! The elevators mustn't be used during evacuation!

Don't get back into the premises before the responsible co-ordinator gives the ALL-CLEAR!

Evacuation and search of the premises

Stop all activities immediately!

All teachers who are having a class when the alarm signal sounds are responsible to make sure the room gets evacuated.

Please check that the rooms that you pass on your way to the exit are empty. Also remember to check the toilets and rest rooms (vilmrum).

Help persons with a disability to get safe either by helping them down the stairs or moving them into a so called safe fire cell. Go to meet with the Rescue Service when they arrive and inform them about where in the building the disabled person is located. They will then go and help the person to get out of the building in a safe way.

Meeting-places

The lawn in front of building 1, 2 & 4

Co-ordination between the different meeting-places will be done.

Co-ordination place

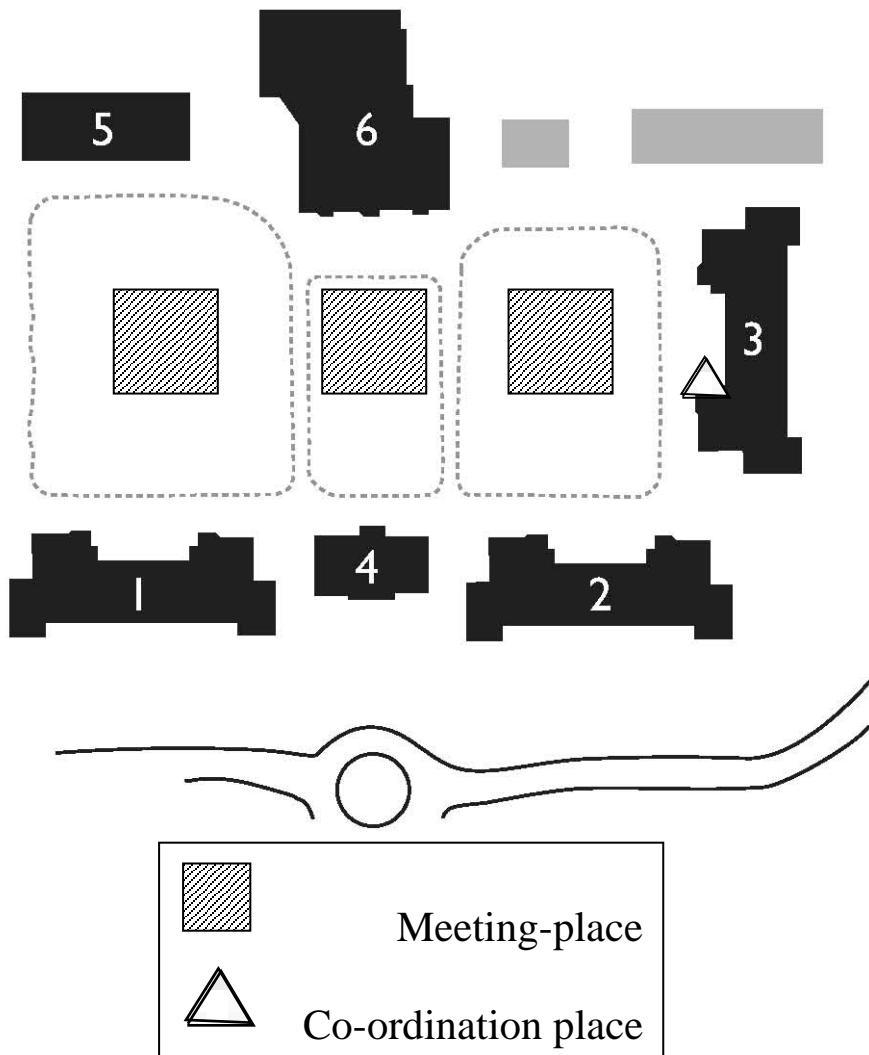
The entrance at Building 3 is where the Rescue Service will go to get information when they arrive. The coordinator shall immediately go there as soon as the alarm sounds.



UPPSALA
UNIVERSITET

Evacuation plan

Department of Information Technology



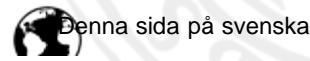
Let the Rescue Service come through!

Make sure that the Rescue Service will be able to get through.
Don't stop on the asphalted areas outside the buildings!

Don't get back into the premises before the responsible co-ordinator gives the ALL-CLEAR!

▼ Department of Information Technology

Uppsala University / Information Technology / Contact /



Emergency Contacts

In the case of immediate hazard to your life, health, or property, call (00) 112.

- The Uppsala University Emergency Number: 018-471 25 00
- Emergency contacts at the hospital: 018-611 00 00
- SOS Alarm Information Number: (00) 113 13 - This is Sweden's national number for information in case of serious accidents and crises.
- In case of a chemical spill, call Relita at 0771 - 103 500
- Corporate healthcare can be reached by phone 018-418 80 10

[Find more information about who you can contact in case of an emergency.](#) ↗

Acute errors at Polacksbacken – Ångström & ITC

Problems with our buildings and fixed installations, sewerage, electricity, fixtures, elevators, heating and cooling, gravel and snow removal, ventilation, water, WC, et cetera, are reported to [Akademiska Hus.](#) ↗

The Akademiska Hus emergency telephone is open around the clock at 018-68 32 04

If something is wrong in any of our buildings, you can report it quickly and easily at the [Akademiska Hus website](#).  Select the building with the error you want to report. Sort the columns by clicking on the respective headings.

Official-on-Call (TiB)

Official-on-Call (TiB) is available outside office hours (weekdays 16:30 to 08:00, Friday 16:30-Monday 08:00 and public holidays) and has the following assignments:

- Answer calls within 5 minutes.
- ' Receive alarms or serious event information.
- Be in place at Uppsala University within one hour if required.
- Make an initial assessment of the extent of the event and possible consequences.
- Initiate information work for crisis management (security, communication, staff and student departments).
- Establish contact with the parties involved in the event (police, rescue service, county council, security companies).

Official-on-Call (TiB) is reachable at 018-471 25 00.

[Other internal and external contacts](#). 

[Contact page for Polacksbacken Campus Management](#). 

CONTACT US

[Contact persons/functions](#)

[Staff](#)

VISIT US

[Adresses](#)

[Map](#)

SHORTCUTS

[Evacuation Plan](#)

[Emergency Contacts](#)

[Report an Error](#)

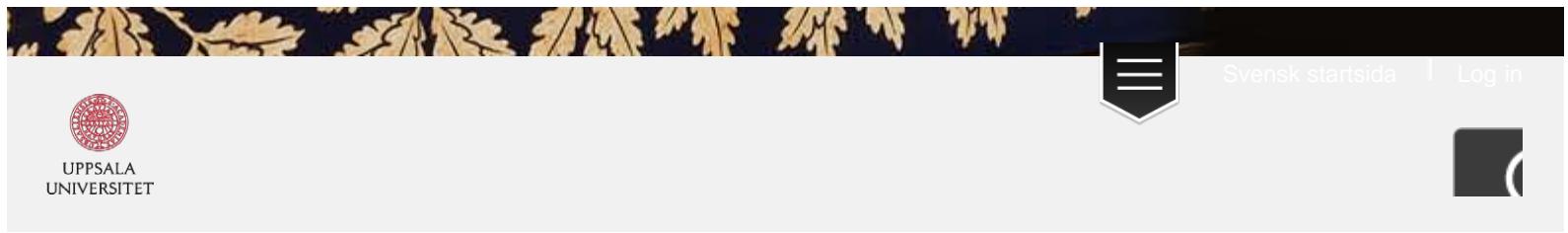
[Edit this page](#)

Uppsala universitet använder kakor (cookies) för att webbplatsen ska fungera bra för dig. [Läs mer om kakor.](#) [OK](#)



[GO TO TOP](#)





▼ Department of Information Technology

Uppsala University / Information Technology / Contact /



Denna sida på svenska



Listen

Report an Error

Find information about [emergency contacts here](#).

[Error reporting at Polacksbacken Campus](#) ↗



Updated 2020-02-04 13:17:14 by [Lina von Sydow](#).

CONTACT US

[Contact persons/functions](#)

[Staff](#)

VISIT US

[Adresses](#)

[Map](#)

SHORTCUTS

[Evacuation Plan](#)

[Emergency Contacts](#)

[Report an Error](#)

[Edit this page](#)

Uppsala universitet använder kakor (cookies) för att webbplatsen ska fungera bra för dig. [Läs mer om kakor.](#) [OK](#)



© 2020 Uppsala University | Phone: +46 18 471 00 00 | P.O. Box 337, SE-751 05 Uppsala, Sweden
Registration number: 202100-2932 | VAT number: SE202100293201 | [Registrar](#) | [About this website](#) | Editor: [Kajsa Örjavi](#)k.

[GO TO TOP](#)





UPPSALA
UNIVERSITET

Joint Web Login

Logging into internal services at Uppsala University

User identity:

Password:

[Lost password?](#)

Log in 

Information about the web service

Security information

When you are finished

For privacy and security reasons, always exit your web browser when you are done accessing services that require authentication. When using a public computer it is especially important to exit the web browser before leaving the computer.

[Uppsala University](#) | [Contact](#) | [Information about Joint Web Login](#)