External Memory and Data Structures

Introduction

Gerth Stølting Brodal
Rolf Fagerberg

Course

Lectures:

- Based on articles.
- Theoretical.
- New stuff: 1995-2001.
- Aim: General principles and methods.

Evaluation:

- Programming projects (a few small and one large).
- Aim: Hands-on.

Homepage:

- `www.daimi.au.dk/~gerth/emF01`

Course

Literature:

- Papers.

Prerequisites:

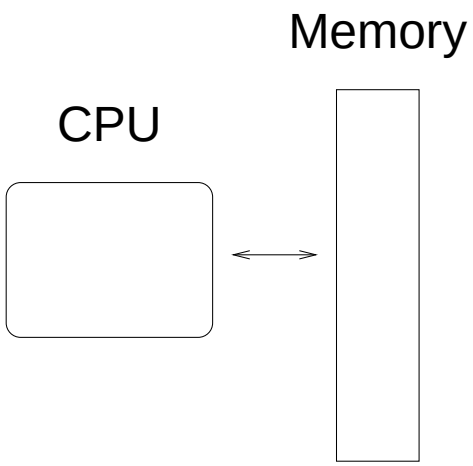
- dADS and dAlg.

Credits:

- 2 points.

Analysis of algorithms

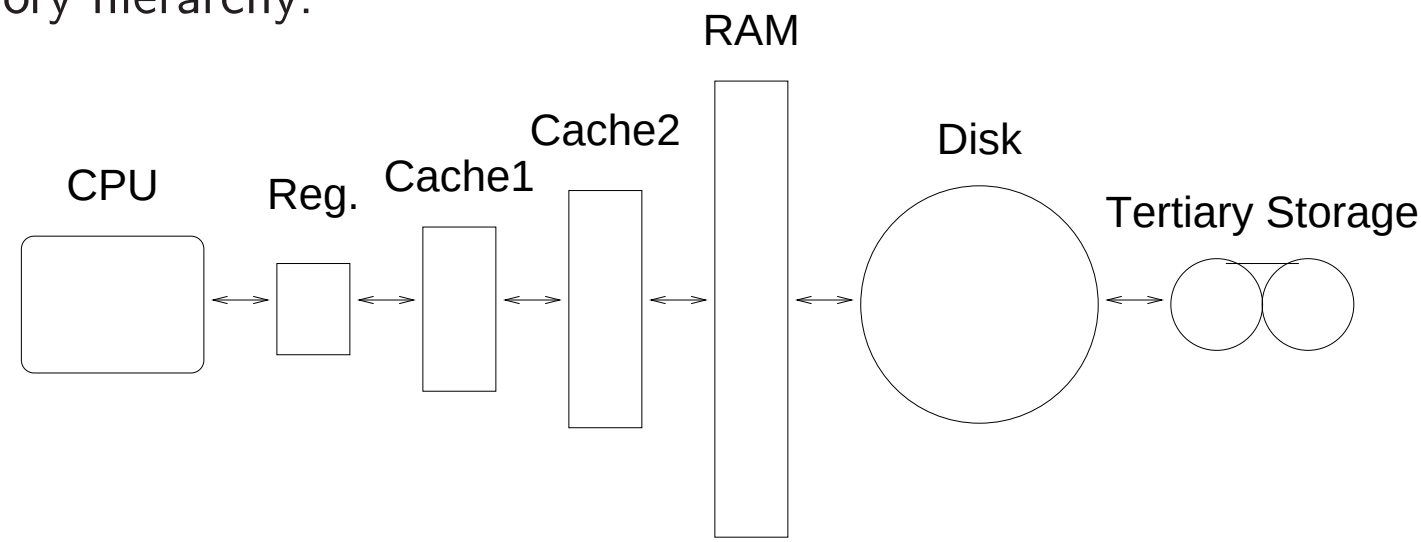
The standard model:



- ADD: 1 unit of time
- MULT: 1 unit of time
- BRANCH: 1 unit of time
- MEMACCESS: 1 unit of time

Reality

Memory hierarchy:



	Access time	*Volume*
Registers	1 cycle	1 Kb
Cache	5 cycles	512 Kb
RAM	50 cycles	128 Mb
Disk	2,000,000 cycles	20 Gb

CPU speed improves faster than RAM access time and much faster than disk access time

Reality

Many real-life problems of **Gigabyte** and **Terabyte** size:

- Databases
  - weather
  - geology/geograpy
  - financial
  - WWW
  - phone companies
  - banks
- Geographic Information Systems (maps).
- Computer graphics, animation.
- VLSI design.

I/O bottleneck

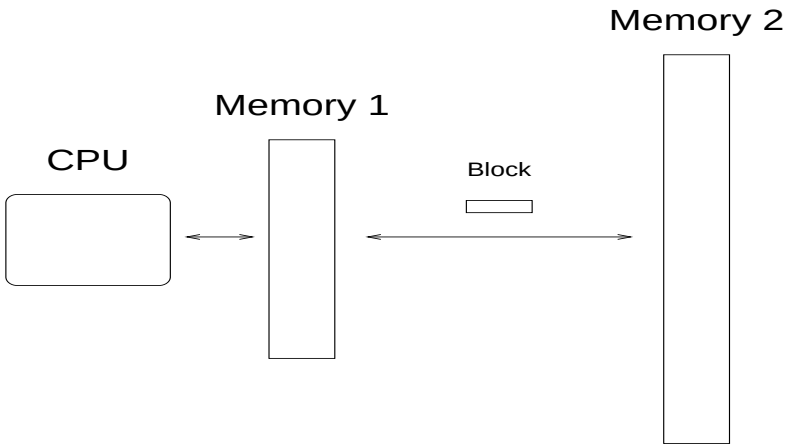
I/O is the bottleneck

$\Downarrow$

I/O should be optimized (not instruction count)

Analysis of algorithms

New **I/O-model:**



Parameters:

N = no. of elements in problem.

M = no. of elements that fits in RAM.

B = no. of elements in a block on disk.

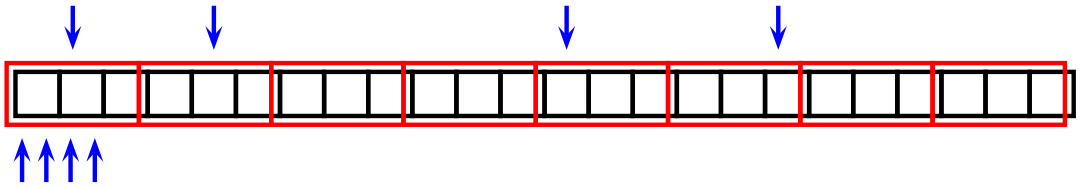
D = no. of disks (copies of Memory 2)

Cost: Number of I/O's (block transfers) between Memory 1 and Memory 2.

Generic Example

Consider two $O(n)$ algorithms:

1. Memory accessed randomly $\Rightarrow$ page fault at each memory access.
2. Memory accessed sequentially $\Rightarrow$ page fault every B memory accesses.



$O(N)$ I/Os *vs.* $O(N/B)$ I/Os

Typically, $B \sim 10^3$.

Specific Examples

QuickSort $\sim$ sequential access

vs.

HeapSort $\sim$ random access

QuickSort:	$O(N \log_2(N/M)/B)$
HeapSort:	$O(N \log_2(N/M))$

(By analysis on black board)

Course

Contents (approximate):

- The I/O model(s).
- Algorithms, data structures, and lower bounds for basic problems:
  - Permuting
  - Sorting
  - Searching
- I/O efficient algorithms and data structures for problems from
  - computational geometry,
  - strings,
  - graphs.

Along the way I: Generic principles for designing I/O-efficient algorithms.
Along the way II: Hands-on experience via projects.