BOF

course also online
https://www.cs.ubbcluj.ro/~rlupsa/edu/pdp/

LECTURE W1

multithreaded programming - google images

concurrent = several tasks that are in progress at the same time, not completely independent
switching tasks is problematic
??? core pins

parallel =

distributed = higher cost of communication,

1970' -> 1Mhz
1999 -> 4Ghz
now it is the same, because increasing the freq would lead to too much heat

time to do addition operation = depends on the distance from the input to the output, through the transistors

processor pipeline - starting the next operation when the current is still on-going

"there's no cloud, there's only someone else's computer"

miracle on the hot?? - dual engine failure
4-engine plane flew through a volcano ash cloud and all 4 engines failed
arian 5 because of a programming mistake

why not:
race condition - needs serialization w.r.t. the shared resource

spectrum & meltdown security issues

getting rid of race conditions - mutexes deadlock - circular wait

non-determinism - a bug that manifest often enough to be problematic, but seldom enough to not be able to fix it

0xCDCDCDCD in visual studio standard library on unallocated memory in order to make bugs easier to trace

lack of global state, lack of universal chronology (distributed system only)

H1: Lecture 2

10s of nanoseconds to access RAM memory
ideal average time on cpu 100-200 picosecond
solution: cache memory

main memory: latency vs throughput

- dynamic ram - DRAM - means content is continuously refreshed
- SRAM - static ram - doesn't need refresh

how it works?
you have several capacitors, one for each cell
reading voltage and interpreting it in 0 and 1
you throw a glass of water on a corridor, and someone in the corner checks the level of water

reading is destructive
larger the memory, slower the reading process
large parallelism, but large latency

vector in the cache

memory chips have small increases in access time, but the throughput doubles

visit the cache from the other CPU, w/ a sync mechanism

few variables accessed from one thread, very good

access an array, depends on array size & access pattern; ex: row vs column traversal of a matrix

communication between cache and memory = cache line, 64 bytes = 512 bits

several threads with different variables which are packed close to one another. if they get in the same ?resource?
example :fale-sharing.cpp

standard: the implemenation can reorder the operation as long as the observable effect is the same

std::atomic<T>::fetch_ad can be used to check if you should have done an operation

tell the compiler to do everything after a given operation

atomic is very restrictive, but its the basc that the processor provides

cond variables, mutexes & everything is built on top of that

LAB W1

FLCD is the missing piece of the puzzle

why logic gates & Computational Logic - venn & karnaugh diagrams? to make cheaper microchips
business idea: fabrica de tranzistori si fabrica de microcipuri
    farmezi materia prima
    procesorul Dacic 3
    o sa ti-l ia si ?apple

big data + deep learning = recommendations

Q: why the empty executable takes so much space?
A: routines to handle interrupts

error detection & error correction from Coding Theory from Algebra used in HTTP

parallel matrix calculation on the GPU

why amazon is the boss? e-commerece site pe steroizi - global

American Made tom cruize

why work in cluj? when you can work remote in US

"backbone in cluj is videochat"

50% drugs; 50% ubb

he's more afraid of you, than you are of him
he's an encyclopedic mind
ask him anything you're curious
ask him interesting question

el e super timid, atunci fa tu pasul …. so he can be comfortable

how do plan now to be well in the future?
like chess, think 7 moves ahead
how could you now make your life better in 5, 10 year?

software architect, project manager
"orice soldat are … de maresal"

Sharpe https://en.wikipedia.org/wiki/Sharpe_(TV_series)

Bradley Cooper & Emily in Paris

feel better after cursing ads

lab1: nu alegeti 3

we'll not measure algorithms by # of operations
we want to optimize implementation

they started increasing the core numbers when they could no longer increase the number of transistors veritasium: The Universe is Hostile to Computers - because there could come stuff from space which changes the bits

trebuie sa pui niste entitati sa faca niste chesti si sa acceseze aceasi resursa printr-un mutex
read the statement with attention
write on teams regarding problems with implementation

no code til next lab, only ideas

how Gabi would do it in a corporation:
write on paper the pseudocode/scheme ideas
follow soundness & completeness

complete = covers all cases

go the senior with the paper and ask him if you're covering all cases
you can trust that algorithm

now you're done being the informatician

now rtfm instead of going to stackoverflow

advice: eighty % on thinking; 20% on implementation; 0% on debugging
if debugging > coding; then you're not gonna go far

0 progress for switching companies
if you don't feel like you're grew in six months, run away

ce thread-uri ai? ce o sa faca? cum le sincronizei?
care o sa fie contextele de live/dead-lock?

next time: just a paper with ideas
ne va la curs probabil niste hint-uri

LECTURE W2

behind the scenes of C++ λ-function compilation

passing custom information to the OS - to the thread

waiting threads consume only memory

switching threads uses a few hundred instructions

blocking threads have a timeout

break

final restriction imposed on closures variables + C# comparison

    the difference is seen when creating a λ and using it much later

    similar to Python's mutable default value

```
def f l = []
    l.append 1
    print l

f []
f []
f [1, 2, 3]
f []
f[ ]
```

try to run this

mechanism for synchronization: atomic variables & mutexes

    atomic = indivisible

    std::atomic blocks other threads from doing operations on the same variable
    limited to certain simple types and simple operations

    time went from 2ms to 7ms

    the access to that variable will be serialized in hardware

purpose: get rid of the shared variable, which cause bottleneck

multi-threaded vector sum: have a thread-local variable for the sum, then at the end do the sum of those variables

why sometimes single thread is faster: most of the time is spent in the bottleneck

    that's why putting more threads doesn't help

very expensive to create threads

cache ping pong is more expensive than making sure of exclusive access to a variable
chip manufacturers consider this

real-time computation = the program runs within an allocated time
2 kinds: soft real time - it's ok from time to time for a computation to take longer than usual
- skipping frames that take longer than the time to switch
hard real time - rocket guiding systems, industrial

LECTURE W3

today: mutexes, using them, issues in using them

problems of early return and exceptions when using mutex lock and unlock
solution: std::unique_lock

in C++ you can do a forced exit by creating a custom exception and catching it in main() or something

in Java:
mtx.lock try finally mtx.unlock
synchronized() = {mtx.lock … mtx.unlock }

class invariant

a mutex protects all the promises between data variables

the code isn't problematic, but the data it accesses

if you modify => don't allow other read/write
if you read => don't allow writes

using a single mutex for everything is correct, but kills all paralelism

when you read "mutex" somewhere you should read it as "bottleneck" - aka, if possible, avoid modifying

FP = relies on pure functions, depend on input args, don't modify anything

multi-thread multiple vectors sum is much slower
if each thread does sum in a chunk then all is summed up, then it is more efficient

idea: share as little data as possible

bank problem: 1 mutex per account

race condition with several objects

bank transfer - the problem of audit() happing in the middle of the transfer
    problem: dead lock for A->B and B->A transfers in the same time

usually a deadlock involves 2 threads, but can involve more
    the probability decreases exponentially w/ the size of the cycle

often enough that is problem, seldom enough that is easy enough to debug

deadlock best scenario: it doesn't do anything

how to solve the cycle, 2 approaches

I. compulsory order for locking the mutexes
    ex: increasing order of the id

special case: if a process can lock at most 1 mutex at a time

<u>try_lock</u>
how much to wait until you try again?

you have to release all the mutexes

live lock = threads are not completely blocked, but they cannot progress; like the diff between checkmate and stalemate in chess

lock-free algorithms - you can do a lot of stuff w/o using mutexes at all
basic idea: freezing one of the thread will not prevent the other thread from achieving their goals
those threads cannot have locks

compare_and_exchange() - if it has expected value then it changes to desired value. if successful, you get true
    can be used for transfer when subtracting/adding to an account

how to do it on 2 accounts? You can't, you have to implement some more abstract data structure

ABA problem

"if you can solve a non-trivial problem with a lock-free algorithm, you can get a PhD for that"

std::shared_mutex - lock (lock is exclusive) & shared_lock

way more complicated than a regular mutex

a new Reader comes when the Writer is in stand-by => starvation

ex: waiting to enter a busy main road

solution: block all the reads until the writer comes, but it's expensive

<u>recursive mutexes</u>

3 types of mutexes in pthread: non-checked, checked, recursive

non-checked fails to do the lock

default - result in undefined behavior

recursive - blocks if you lock from a different threads,

<u>recursive mutexes</u> - usually in Java and C#

is usually thought to be a good idea, but it's not a good idea

if a method is called from the inside, the invariant might no longer hold

solution: f_internal() - must be called under mutex, mention all pre-conditions and call it inside f() & g(). Much cleaner, much easier.

original collections in Java were thread-safe
later implementations are not. Why? experience showed that usually it's useless because usually the access to the collection is already bottleneck by mutex by the programmer code

in Java all functions are by default virtual. bad idea, because it makes the semantic of a function depending on the implementation details

ex: Collection with insert() & append() and derived CountedCollection

insert() { count++; Collection.insert(); }

append() {}

what if Collection.append() calls insert() ? => the counter is increased twice

solution: never call a public function from another public function. refactor into a private helper function

next time: producer-consumer communication

another operation that can be involved in a deadlock: join()

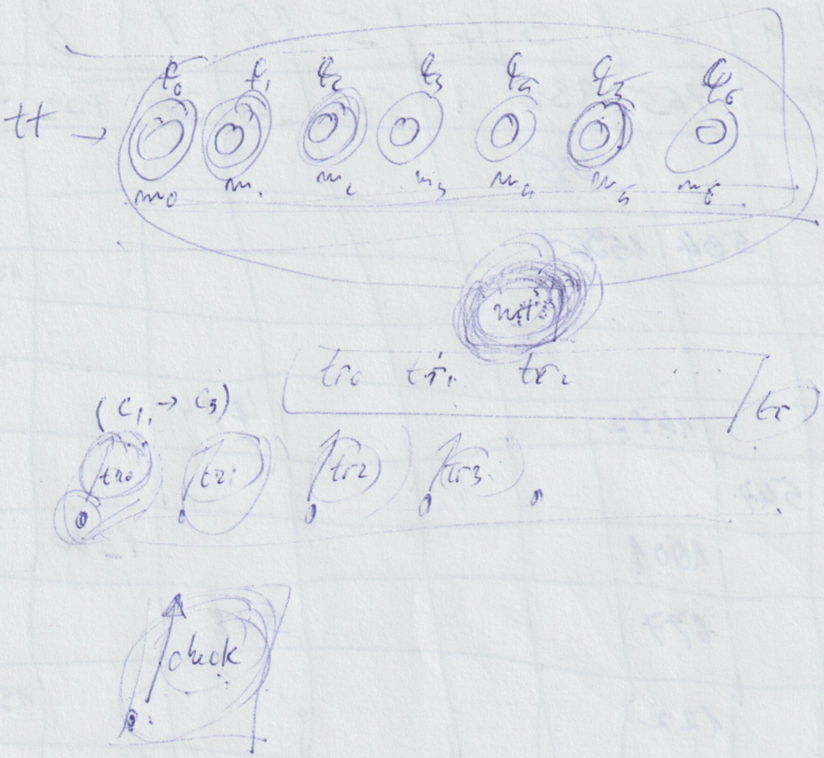
normally you should join() all the threads you spawn

- if the destructor is called before join() => an exception is thrown
- you should have controller over when the thread ends
- join() is blocking and all blocking functions may result in deadlock
- example ???
- very often the ending of a multi-threaded application is hard to analyze

LAB W3

aroganta
cryptosecurity
qubit

LAB W4


tt
tr0 tr1 tr2
tr
(c1 → c3)
tr0 tr1 tr2 tr3
check

LAB W05

What are people looking for in a thesis? What are the expectations?

undergraduate exam consists of 3 parts

1. theoretical contest - 3 big areas: OS, DB, DSA; on paper/oral
2. the scientific paper of the thesis
3.

scientific articles
scientific books
undergraduate thesis
dissertation thesis
doctorate thesis

question to your helper teacher: like how to find the bibliography

Gabi's way:

1. initialState = 0 knowledge
2. you find the problem you want to solve (ex: predict what players wins a chess game). it must be an open problem
3. How did others tackle it?: through research
    a. how do you explore anything that has been written on the topic? use AI. search a big space w/o a lot of time. GA balances exploration & exploitation
    b. at 1st, you just explore (filter the water from the rivers and find nuggets => source of gold). How do you explore scientific work? Google related words to your topic in random order and add the word 'pdf' to the end
    c. balancing with exploitation: harvest PDF files with the name: YEAR_KEYWORDS_VALUE, VALUE = enjoyment of reading 1 (crap) -> 5 (omg). you can even use real numbers
    d. skimming articles: resemblance between scientific paper & Romanian's fairytail (basmul) - the chapters: introduction, intrigue (find a niche that can be improved/hasn't been tackled yet)/proposed approach/abstract, implementation, compare your performance on the problem vs the state of the art, conclusion - did we do something to improve?
4.

comparing two architectures is a valid scientific paper

!!! you also need to put the shitty papers there. Why?

The Pareto principle - 80% of the wealth is owned by 20% of the population extrapolating: 80% of the content on the topic is in the top 20% of it

golden tool: https://www.connectedpapers.com/

sqrt(n) splitting

winter break - free time to take the 20 papers and put a .docx in which you highlight main ideas, advantages, disadvantages. but you read everything

what is asked for the thesis: originality, no matter if it's basic

at the end of the paper highlight the disadvantages: conclusion & future improvements
THIS CHAPTER CONTAINS HINTS

??, adapt, improve

title: using <TECHNIQUE> to solve <PROBLEM>

for bachelor thesis, you need to show that you can create a layered architecture app

2 apps:
1. deliverable intr-un REST API
2. client care consuma API-ul

you need to convince that is motivational and relevant - it increases the quality of life

if you have a discussion, you have science

the app should show that you know you've been through all subjects

licenta poate sa fie personal project: git repo, documentatie, testare, layered architecture

LECTURE W5

PDP LECTURE W05

creating threads is expensive, especially for small / elementary operations

ThreadPool is a solution

system limit on the # of threads

going to sleep (wait) is expensive

shutting down multithread programs

enqueue only if the pool isn't closed

what to do w/ the existing items?

there should NEVER BE any concurrency between a destructor and a member function - taken care by the outside code

dynamic # of threads

more threads than CPU bound can be useful when

waiting for a thread in the queue - caveat

can of worms in real-time apps and critical apps

usually all mem. allocation is done at the beginning (to avoid mem. alloc. failures) about Futures (10:38):

set -> promise
-> 1-shot value

in C++ shared_future, but you can call get() just once

when main() blocks

std::async

in C# Task.Factory uses a ThreadPool

buffer for send() (aka send() might block if the data is too large; take into account ill clients: bugs or attacks)

1 thread/client might not be doable
- a lot of clients
- even with just 2 clients

solution: select() (event-driven)
problems: -
- control is upside down
- a single while-loop

callbacks (event-driven)

Windows: select() accepts only socket descriptors, no keyboard input

warning for callbacks:

PDP LECTURE W05

mechanism for argument binding for callbacks

~11:25 example in C

frameworks running data-agnostic code

problems in C: void* and mem. alloc.
why UB is so big deal?
where debugging, every line of code is a suspect.

~11:37 example in Java (pre Java 8)

example in C++
static type and dynamic type

in C++ you have to use pointers
raw pointer => you have to control the lifetime

smart pointer => move, copy

lifetime considerations for lambda captures

lambda = anonymous function
closure = function obtained from another function by binding args

questions:
1. what can you do/do not inside the callback?
next time: how to solve it
2. on what thread the callback runs?

classical problems w/ callbacks:

possible solution: don't pass args
postpone running

LECTURE W6

PDP LECTURE W06

limitation of Future

callbacks were invented on single core, for async

future continuation operation in C# (futures demo 2 - cascade1.cs)
stores somewhere the function and continues when the future completes
ContinueWith returns only a future

10:30 cascade 2
TaskCompletionSource = Promise
.Task() returns the future part

void tasks are meaningful only w/ side effects

std::async and Task.Factory.StartWith create a new thread

10:29 whenAll() - called when all tasks are

10:36 there's also whenAny() + use case

sw-switch.cs

context switch done locally through await is much cheaper

the original function must also return a future?

- it is lab 4, to implement a client

11:08 why something else than the callbacks?

1. tight link between the call and the callback
2. inverted control
3. (not very well documented) restriction on what I can call in the framework

=> we need something better

1 sol.: separate the ? call

linux bottom half (interrupt vector)

11:10 modifying begin(), end() functions to return a a future

the callback sets the promise's result

simple way of converting a begin(), end() pair

simplifies a bit, but not very much

11:15 advantages

1. ...
2. the callback always happens on a

PDP LECTURE W06

thread from the framework
still limited but I have to put something
99.99% similar to a callback

3rd example - let the compiler do the heavy lifting

11:19 async = telling the compiler is not a regular function
executes the func. from start to 1st await
everything between the current await and
the next is set as a continuation
for the current task

enqueuing a continuation means using
a thread pool (which runs on a
different thread)

11:26 what is requested for L4

connect takes a while to
complete
begin Connect w/ passed callback

BeginSend() w/ a GET request
if data ! complete, call begin Receive
w/ end receive as callback
let main know you finish (promise, future)
or condition

3

download controller + cond-var
main waits in a while loop

"please copy-paste most of the code"

ver 2 : single functions that return a future

ver 3 : a download function that returns a task.

```
async Task Download(string url)
{   await Connect()
}
void Main()
{  call several times the download
   function
   put the results into a futures (tasks) vector
   call whenAll ?
     t = Task.Factory.Continue
       t.Wait();
```

mixing synchronous style w/ async code

in C++ you can write up to the 2nd impl.
2nd if you write the continuation part
3rd can theoretically be implemented w/ C++20 coroutines - as a project
next: intro to parallel computing. decomposing data; programming 4

PDP LECTURE W06

std:: async doesn't use a thread pool

11:46 returning a future that will be computed later

futures [i][j] = std::async(a, b, rowColProd(i, j))

large overhead for creating a thread for each element

~11:50 about L3 strategies

slightly better w/ column

11:52 CPU throttle on laptops

LAB W6

the IT Crowd

LECTURE W7

PDP LECTURE W07

Simple parallel algorithms

for power consumption problems it is better to use a single thread

CPU addition

nicer to use futures

performance analysis in debug mode has overhead. use release

?->10:32 analysis
mem. access is not so parallel.

10:32 vector sum both strategy

?->10:43 CAS latency of RAM
~10:43-47 matrix multip. col vs row
10:47 stencil pattern

11:05 recursive decomposition

adition of floating point nrs is not associative
adding numbers of roughly same magnitude reduces errors (binary tree)
joke and why timestamps are given using integers and not floats
11:12 took lattel

11:15 going to the kernel and the atomicity is costly

we'll revisit this in a tougher context in distributed systems

11: merge sort

11:34 undo & reverse order due dependency

11:40 can we improve?

merge on >1 threads

LECTURE W8

PDP LECTURE W08

from lecture 7

improving the total time w/a reasonable CPU consumption

$B_k = A_0 + \dots + A_k$

explanation of

10:22 lecture 8

$$P(x) = \sum_{i=0}^{n-1} r_i x^i$$

$$Q(x) = \sum_{i=0}^{n-1} q_i x^i$$

$$R(x) = \sum_k \left(\sum_i r_i q_{k-i}\right) x^k$$

10:3 Advanced recursive decomposition

10:40 3 additions over 1 multiplication

$$(P_1(x) + P_2(x))(Q_1(x) + Q_2(x))$$

1 pol. mult of 2n-1 → 3 pol. mult. of n-1

$$T(2n) = 3T(n) + O(n)$$

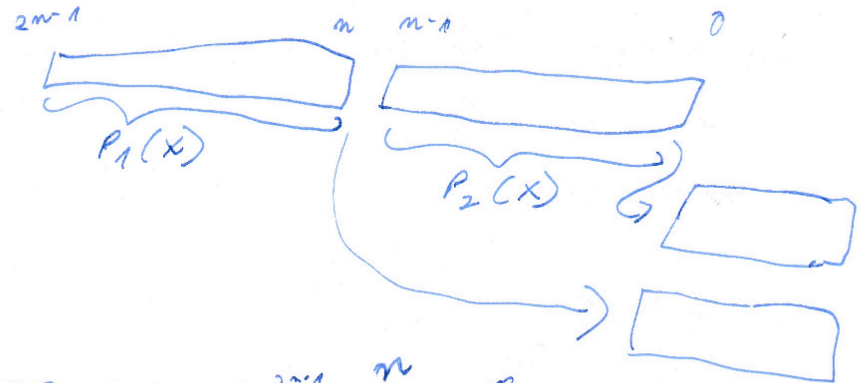
$$T(A \cdot n) = B \cdot T(n) + f(n)$$

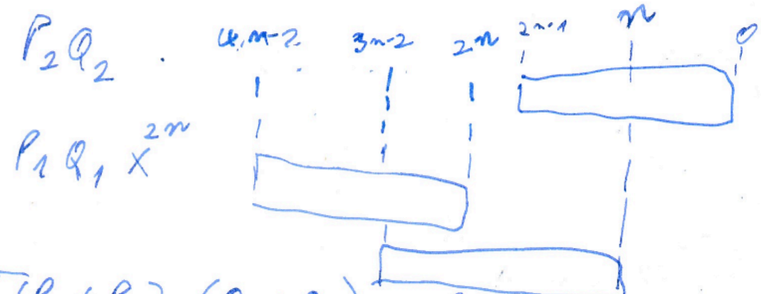
for f that grows slowly: $T(n) = O(n^{\log_A B})$

$T(n) = O(n^{\log_2 3})$ $\quad \log_2 3 \simeq 1.6$

4 multipl => $n^{\log_2 4} = n^2$

$P(x)$





$[(P_1+P_2)\cdot(Q_1+Q_2) - P_1Q_1 - P_2Q_2]\cdot X^n$

11:17 algorithm

Karatsuba has overhead for small elements
=> use the classical alg.

11:23 Karatsuba for big numbers
problem of 1 digit longer

? power of 2

another problem: Fast Fourier Transform
(signal/image processing)
($O(n \log n)$ normally)

PDP LECTURE W08

11:28 parallel explore - backtr. in parallel

11.33 why 'other.wait'?
!!! when passing by ref to another thread always have in mind lifetime of objects
alternative: pure functions

11:45 administrative things:
- next week holiday + reschedule Sa, 17 dec (will be on course page)
- discuss and schedule the exam
  preferably 4 dates w/ pairs of 2 and 1 group alone

LECTURE W10 - MPI

PDP LECTURE №10

Lecture 8-9. MPI Programming

- mpirun demo

everything between Init() and Finalize()
output redirection through an ssh tunnel
MPI_COMM_WORLD - communicator

Synchronous vs. Buffered send
mpirun hangs if init() or finalize() are missing
~10:35 vector sum demo explained
10:44 -n 0 should also work (distinction w/ threads)

(10:47 nsqr in batch)
10:49 mpic++ is a wrapper over g++
10:50 vector sum demo run

11:08 broadcast

11:14 MPI scatter

11:20 MPI gather
11:22 MPI all gather = gather + broadcast
11:27 Waitany
11:28 summary of MPI
11:31 ssh
11:35 windows pageant
RSA broke by Quantum Computers
tutorial
agent forwarding

11:~44 diagram of forwarding

next lecture: recursive decomposition

LECTURE W11

PDP LECTURE W11

about L6
ideally # threads a bit > # cpu cores
10:13 MPI: specify network interface - sol. to example hanging

10:19 Recap

10:21 on L7
- sequential ($O(n^2)$)
  - diagonally (MPI_Gather can be used)
  - horizontally w/ an array for each line

  x sent to all via broadcast
  y sent via broadcast
  problem of the full diagonal for a square
  the overhead of communication depends on comm if.
  an early mechanism of distributing workload
  bottom line: experimental analysis

10:31 recursive decomposition over MPI

10:37 deciding parent
10:46 problems workers that have nothing to do &
propagation of stopping

11:05, 11:17 quicksort example
~11:07 direct child computation & formula
~11:09 - 11:13 parent and Level()

11:18 - interesting idea of matrix multiplication
element = K-size block (square), K:N
needs only A_{ij}, B_{jk} at a time

the idea (around 11:30)
circular shifts

11:32 - 11:40 the code

11:36 problem of waiting for the receiver w/ send

send, then receive for all but one, who
does the other way around
careful: order of send and receive ops

11:40 lecture from tomasson
re lab 4, coroutine, futures, callbacks

interrupts are kind of the callback for async ops

hw and sw interrupts only share the
interrupts vector table

task switching
from batch systems to time-sharing
systems, going to responsive task

LECTURE W11 Recovery from W10

PDP LECTURE W11 (Recovery)

C#, Python, Lua, C, C++ windows, Kotlin & others

coroutine =
- like a thread but the control is explicitly to the program

10:03 Window API Fiber

10:15 - explicit control
- co. run in user space
- application: python generators
- the concept dates back to '60s

10:18 clarification
- symmetrical : any <-> any
- asymmetrical : parent <-> child

- stack-full: Fiber, allows in depth f calls
- stack-less: more eff., cheaper, more limited
  ex: Python
  all calls from the coroutine happens on the main thread

10:23 Activation Record

10:26 async-await

it behaves like a coroutine
aka it can be suspended and continued later

10:35 await uses an internal thread pool

-10:53 OS details
Begin Receive
- Wait
- poll
- callback (interrupts)

-10:54 trick

10:55-57 bottom half

coroutines clarif: 1st class vs constrained
generators - use case of coroutines for a single thread

11:02 -11:12 C++ coroutines

LAB W11

LECTURE W12

PDP LECTURE W12

10:08 information propagation

10:14 consistent chronology problem

consensus over an arbitrary order

10:22 Finding causal relations

10:27 new Clock = $(x_1 \ldots x_n)$, x_i = max (currentClock$_i$, otherClock$_i$)

10:28 relationship of dependence

10:30 ?

10:31 Distributed transactions

10:40

10:45 Distributed shared memory
- lots of implementations

11:06 using distributed transactions idea

11:14-16 PAXOS

11:17 next lectures: OpenGL, fault tolerance, questions

11:19 bank transactions

std::memory-order

11:33 end

LECTURE W13 - OpenCL programming

PDP LECTURE W13

OPENCL PROGRAMMING

10:08 intro - history

from RAM & CPU to dedicated process on GPU

(it knows how to draw simple things;

now more complex ones;

optimized for drawing data

standard interface like OpenGL bc. its proprietary

MS DirectX

Vulkan

10:13 why not use it for computations too?

OpenGL & NVIDIA'S CUDA

in some cases, the impl. can be used by the CPU too

10:14 setup

10[16]-10[19] library integration in VS

build / external / vendor. .. / ... dll

10:19 concepts

10:20-21 C restrictions : mem. model & recursivity

10:21 work item

10:22 binary is hidden

10:24 memory

10:25 code

10:41 queue is actually a bag w/ dependencies

1

11:13 limitations

reachinity (simulated)

vector binary sum example

11:25-37 merge sort example

khronos.org/

documentation is large, but very-well written

last lecture - fault tolerance

+ q & a

exam - describe problem and its cause

- you can choose the language

- just convey the idea / be clear with your intention

(ex: 7 args function from mpi)

mail for 2nd day date

other students - choose the group, but announce, at least 1 day in advance

LECTURE W14 - Fault Tolerance

only scratching the surface, cuz the topic is huge

independence of airplane engines

case when redundancy didn't help cuz the software was the same - same error

manufacturing issue/overload
bottom line: failures are not always independent, software bugs will affect all devices that have the same software

10:14 consensus problem

advice: floating point arithmetics errors (associativity doesn't hold, casting to boolean, line example)

10:20 variations, General's problem

10:22 failure types

crash failure - best cuz it does not send bad data
byzantine failure - there could a traitor in the army, even the general
communication failure

10:26 sync vs async

no upper bound on computing time for async operations.

synchronous. if time limit is passed => process is considered faulty

no change of solving Consensu problem in the async case

sync case - byzantine failure

1 lieutenant failure
general failure

limit of $3t < n$

all distinct => every1 sees distinct value => resort to the default value timeout, you can create a default value for it

output does not uniquely depend on the output

why is the async case such a big deal?

deadline for non-leaf node that has a fault

puzzle: you have n prisoners. they are locked in individual sense (cannot communicate w/ one another).
from time to time, single prisoners are taken, arbitrarily, to the court. eventually, all prisoners will go.
communication device w/ 2 positions in courtyard. originally the switch is 0, the guardian doesn't touch it.
at an time, in a finite amount of time, a prisoner should be able to declare that all prisoners were in the courtyard
prisoners know the number n.

you have n prisoners switch https://jaylorch.net/brainteasers/ThePrisonersAndTheSwitch/

exam subjects - everything besides OpenCL
time: 2h
official cheats sheet A4 (aka 2 pages) that must be turned in at the end
retake exams has same rules and same difficulty

coming w/ another date - write an email

emphasis on parallel and distributed stuff and less on programming

∀ language, csharp, c++, java
he won't be picky w/ the syntax
don't exaggerate w/ the comments, try to be concise w/ the explanation

mutex, cond var, creation/joining of threads
    but it's ok if you something else

---

recursive decomposition

always think about range of processors that are assigned to solve a particular problem

split the work and the range (ex: for karatsuba in 3)
    pass the subproblem & the # of processors it can use

id formula: id_ind? = i + floor(nr_processs/2)

ex: mergesort-simplified-mpi.cpp

get parent id - from status param of MPI_Recv()

from binary tree:
    childId1 = parent * 2;
    childId2 = parent * 2 + 1;
works only if the parent doesn't do anything. but this means the parent idles

solution to the problem:
2 prisoners case
3 prisoners case
- idea
    - ?
    - 3 declares only if he sees in pos 0 after he saw it in pos 1
    - fault - taking the prisoners always in the order 1, 2, 3

rules to satisfy by ∀ algorithm
- not output an incorrect result (partial correctness)
- eventually output a result

hash table based mapping

probability of 0 (ex: always flipping heads)

probability of 0 for infinite loop for a fair random generator for generating edges in a graph

1 prisoner becomes the counter, all others flip the switch to pos 1 the 1st time they see it.
the counter flips it to pos 0, if it's in pos 1

exam prep 07 feb

cheat sheet writing idea: write on paper, scan it, make it smaller, print it, now u got free space

Astea sunt toate variantele posibile pe care le poate da la ppd? (normal sau cu mpi)

1. big_numbers_product
2. scalar_product_simple
3. scalar_product_tree
4. convolution
5. hamiltonian
6. permutations
7. combinations
8. k_combinations
9. k_coloring
10. merge_sort
11. quick_sort
12. producer_consumer

resources

1. lectures
2. bookmarks?
3. https://github.com/alexovidiupopa/pdp/tree/main/exam-workspace/src/ro/alexpopa/mpi
4. exam-subjects 2022
5. study session Jinga 4 feb
    a. problem 1
        i. 00:00 intro
        ii. 03:08 A fals?
        iii. 06:40 B adv
        iv. 09:05 C
        v. 1?:??
    b. 33:22 problem 2
    c. ~53 - about problem 3
    d. problem 1
    e. 1:09:10 problem 2 ProducerConusmerQueue
    f. 1:21:59 sub 4 pr 1
6.

subject 2

1. https://www.cs.ubbcluj.ro/~rlupsa/edu/pdp/progs/futures-demo1-with-impl.cpp
2. https://www.cs.ubbcluj.ro/~rlupsa/edu/pdp/progs/producer-consumer.cpp
3. https://www.cs.ubbcluj.ro/~rlupsa/edu/pdp/exam-example.pdf future with continuation
4.

cheatsheets

1. cheat sheet pdp.pdf by Cristian Gherman
2.

2023 subject 2 problem 1

Consider the following excerpt from a program that is supposed to merge-sort a vector. The function worker() is called in all processes except process 0, the function mergeSort() is called from process 0 (and from the places described in this excerpt), the function mergeSortLocal() sorts the specified vector and the function mergeParts() merges two sorted adjacent vectors, given the pointer to the first element, the total length and the length of the first vector.

```
void mergeSort(int* v, int dataSize, int myId, int nrProc) {
   if (nrProc == 1) {
      mergeSortLocal(v, dataSize);
   } else {
      int halfLen = dataSize / 2;
      int halfProc = (nrProc+1) / 2;
      int child = myId+halfProc;
      MPI_Ssend(&halfLen, 1, MPI_INT, child, 1, MPI_COM_WORLD);
      MPI_Ssend(&halfProc, 1, MPI_INT, child, 2, MPI_COM_WORLD);
      MPI_Ssend(v, halfSize, MPI_INT, child, 3, MPI_COM_WORLD);
      mergeSort(v+halfSize, halfSize, myId, nrProc-halfProc);
      MPI_Recv(v, halfSize, MPI_INT, child, 4, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
      mergeParts(v, dataSize, halfSize);
   }
}

void worker(int myId) {
   MPI_Status status;
   int dataSize, nrProc;
   MPI_Recv(&dataSize, 1, MPI_INT, MPI_ANY_SOURCE, 1, MPI_COMM_WORLD, &status);
   auto parent = status.MPI_SOURCE;
   MPI_Recv(&nrProc, 1, MPI_INT, parent, 3, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
```

```
    std::vector v(dataSize);
    MPI_Recv(v.data(), dataSize, MPI_INIT, parent, 3, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
    mergeSort(v.data(), dataSize, myId, nrProc);
    MPI_Ssend(v.data(), dataSize, MPI_INT, parent, 3, MPI_COMM_WORLD);
}
```

Which of the following issues are present? Describe the changes needed to solve them.

A: the application can deadlock if the length of the vector is smaller than the number of MPI processes.

B: the application can produce a wrong result if the input vector size is not a power of 2.

C: some worker processes are not used if the number of processes is not a power of 2.

D: the application can deadlock if the number of processes is not a power of 2.

EOF