mynatix

Latency Optimised Code Segmentation (LACOS)

Using Read-After-Read Dependencies to Control Task-Granularity

Dr. Andres Gartmann & PD Dr. Mathias Müller

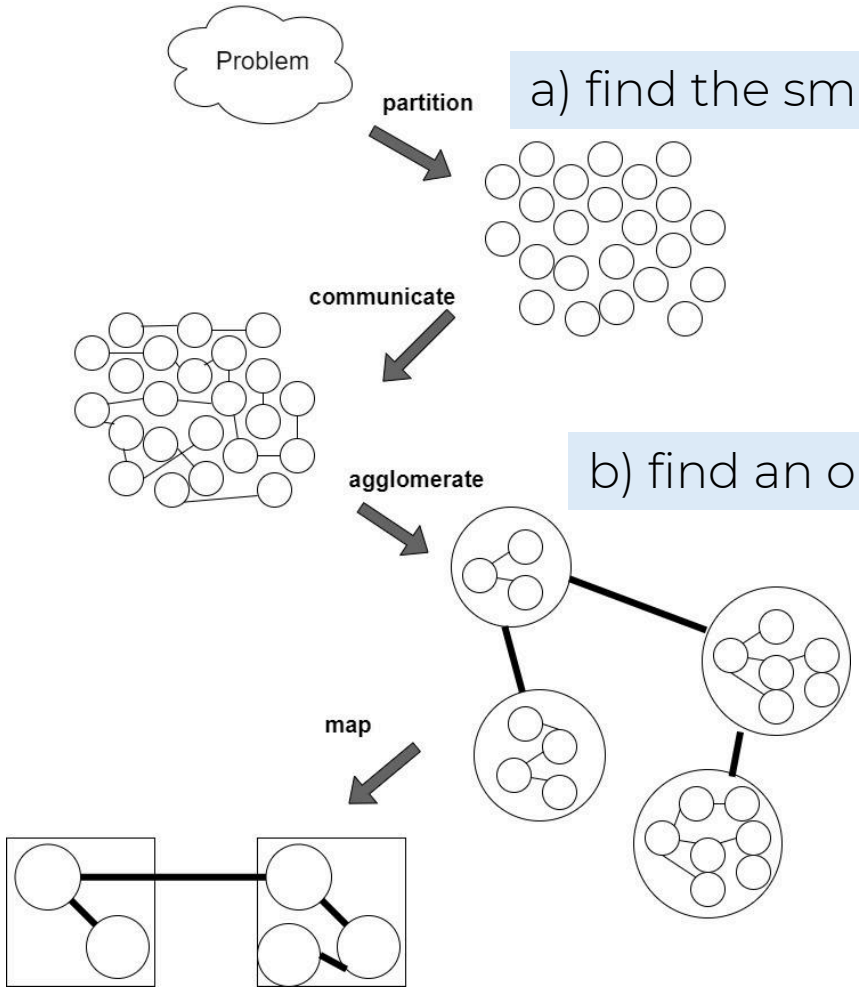
Overview

mynatix

Auto-parallelization with computation blocks

1. Communication vs. computing
2. Basic physical relations
3. Introduction computation blocks: a novel form of segmenting codes in compilers
4. Application in middle- and back-end:
    - in the middle-end: auto-parallelize
    - in the back-end: addressing different frameworks / architecture types

The practical challenge: communication vs. computation



a) find the smallest partition

b) find an optimal agglomeration

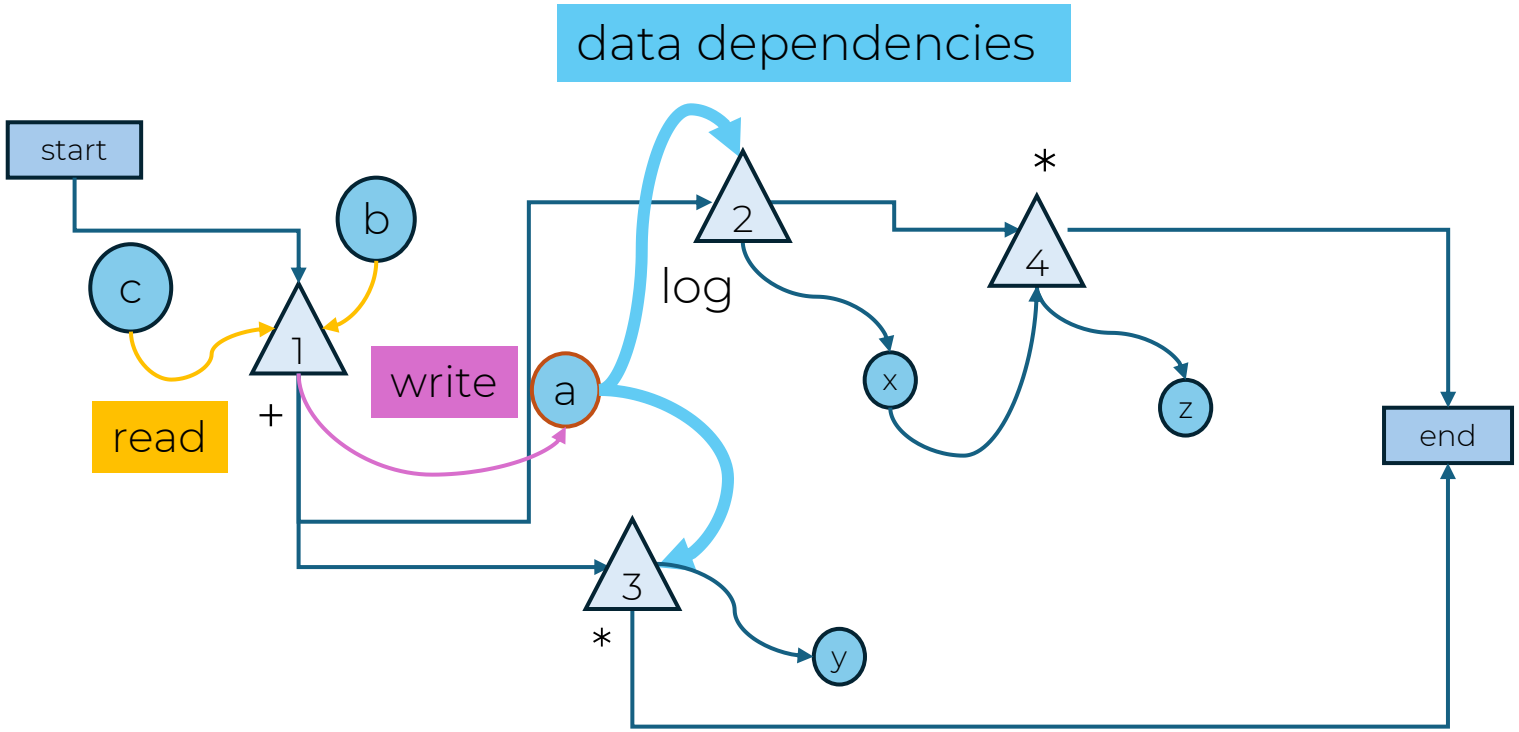
practical challenges:

- find task granularity
  communication overhead vs. computation
- Writing parallel code
  for a target hardware

Figure adapted from Ian Foster. 1995. Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering. Addison-Wesley Longman Publishing Co., Inc., USA

A simple example using a data flow graph

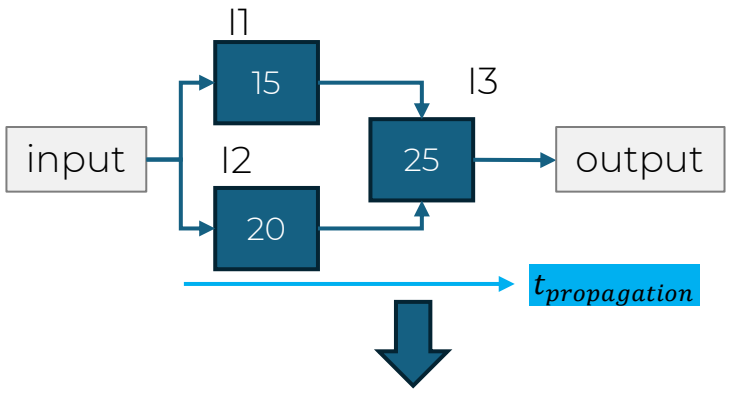
```
a = b + c;
x = log(a);
y = a*a;
z = x*4;
```



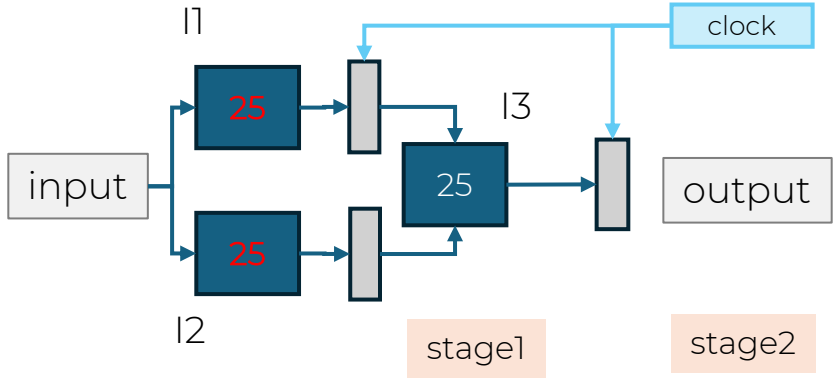
data dependencies = central aspect for
Instruction Level Parallelisem

Combinational to pipelining

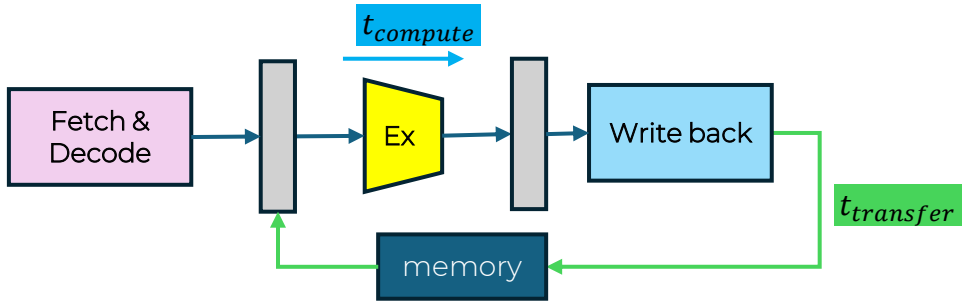
- 3 instructions: I1, I2, I3
- combinational: $t_{propagation}$



- buffer registers make inputs stable [1]
- sequential -> introducing clock, memory



- pipelining increases latency, but also throughput
- pipelined processor execution instructions in **parallel** [2]



Time (in clock cycles)

execution order

	cc1	cc2	cc3	cc4	cc5	cc6
I1	FD	EX	WB			
I2		FD	EX	WB		
I3			FD	EX	WB	

pipeline hazard

	cc1	cc2	cc3	cc4	cc5	cc6
I1	FD	EX	WB			
I2		FD	EX	WB		
no-op						
I3				FD	EX	WB

solutions for pipeline hazards [2]

(1) Hardware

(2) Software: instruction scheduling by compiler

→ NP-complete problem (known since the 80s [3])

[1] https://ocw.mit.edu/courses/6-004-computation-structures-spring-2017/pages/c7/c7s1/#2

[2] Baer, J.-L. (2009) Microprocessor Architecture: From Simple Pipelines to Chip Multiprocessors. Cambridge: Cambridge University Press.

[3] John L. Hennessy and Thomas Gross. 1983. Postpass Code Optimization of Pipeline Constraints. ACM Trans. Program. Lang. Syst. 5, 3 (July **1983**), 422–448

Software solutions for instruction scheduling

Software solutions:
- Since 1985 all processors have Out-Of-Order execution [1]
- ILP in one single Basic Block is very small: 3-4 parallel instructions [2]
- Compilers: keep order of instructions in correct order = prevent: **data hazards** [1]
- Use data, name dependencies

potential data hazard:

```
I1:   R1 = R2 + R4
I2:   R2 = R4 – 25
```

Read-after-Write (RAW)
Be sure to read (I1) bevor write (I2)

- possible **data hazards: RAW, WAW, WAR**
- No hazard: RAR

NO potential data hazard:

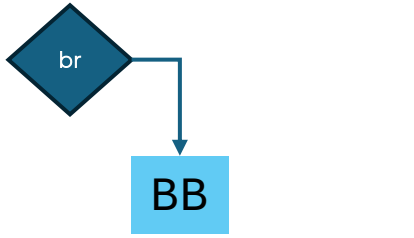
```
I1:   R1 = R2 + R4
I2:   R2 = R4 – 25
```

Read-after-Read (RAR)
Read order can not change result

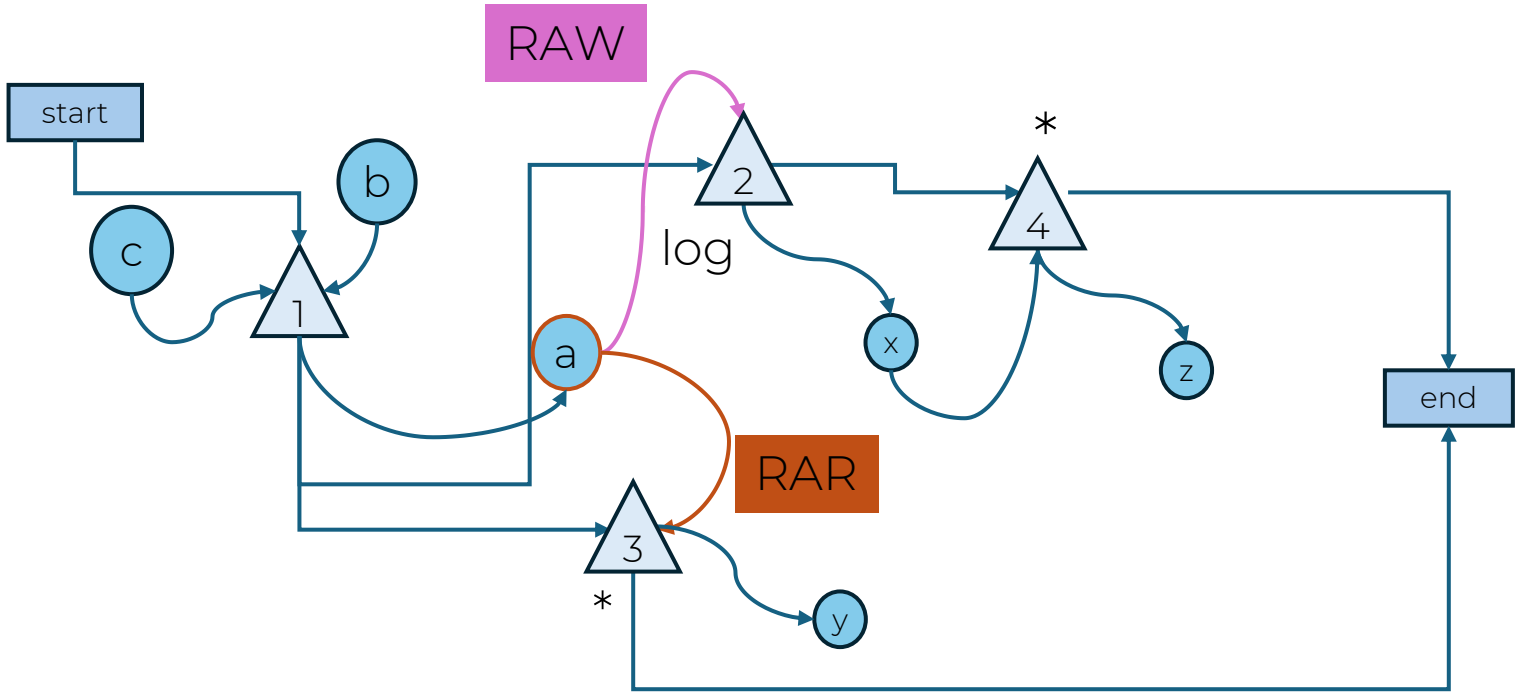
No hazard, but
Read-after-Read's contain information about ILP

[1] David A. Patterson and John L. Hennessy. 1990. Computer architecture: a quantitative approach. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.
[2] David W. Wall. 1991. Limits of instruction-level parallelism. In Proceedings of the fourth international conference on Architectural support for programming languages and operating systems (ASPLOS IV). Association for Computing Machinery, New York, NY, USA, 176–188

Using Read-After-Read to exploit parallelism




```
a = b + c;
x = log(a);
y = a*a;
z = x*4;
```

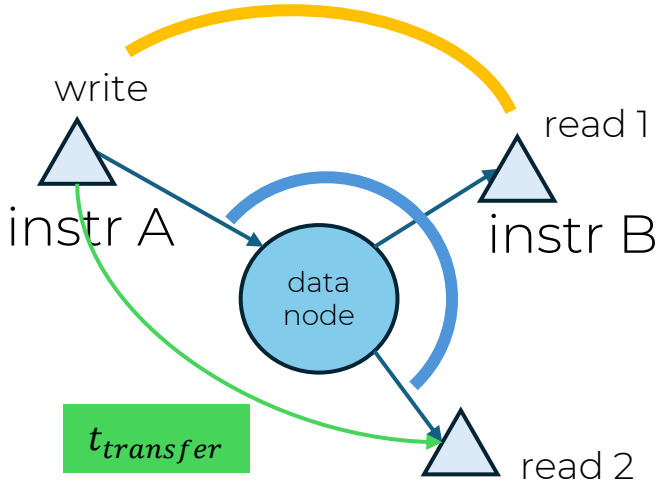


The glue of instructions = true dependencies

code

instr A -> write
instr B -> read
instr C -> read

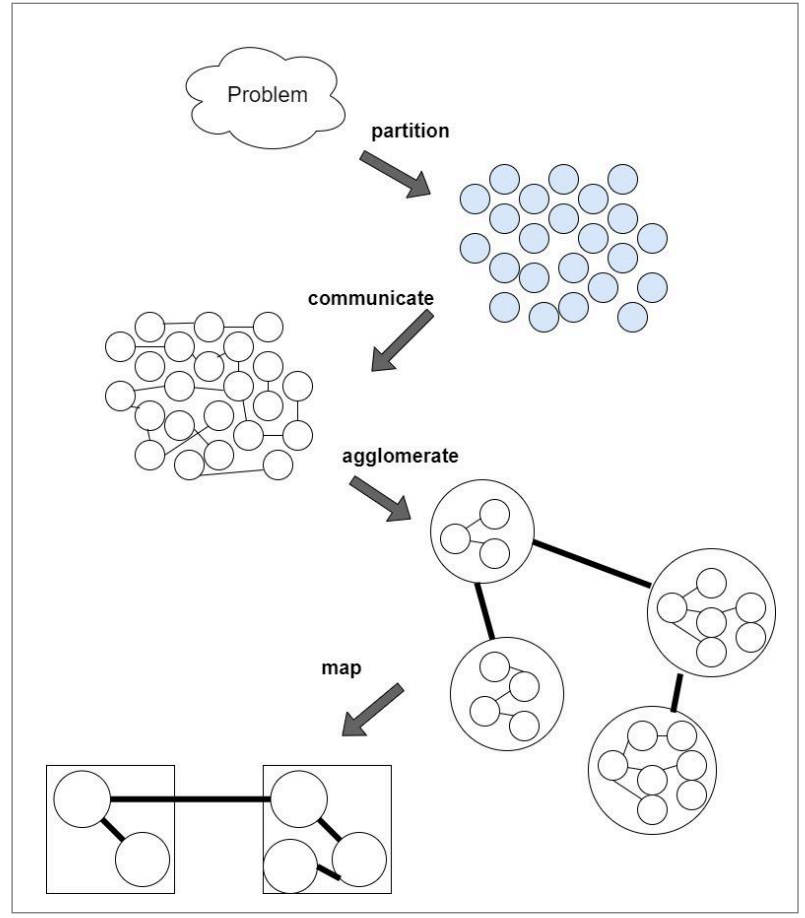
same location
→ physics makes this the best option



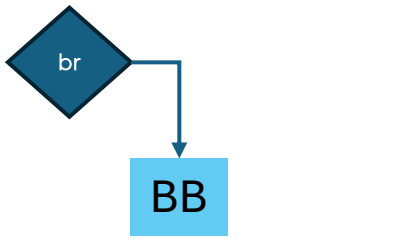
RAW

in parallel
→ data (node) must be transferred
→ introduce $t_{transfer}$

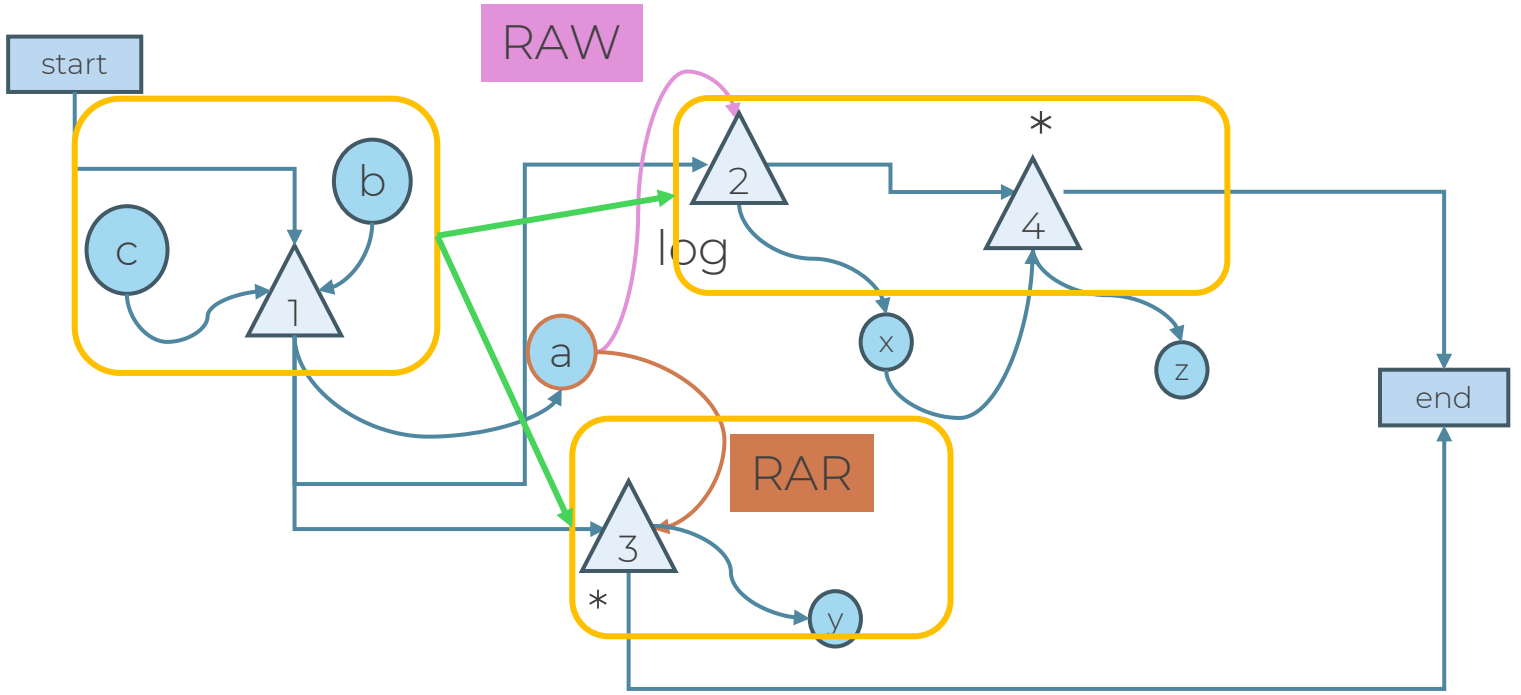
RAR



A novel segmentation of instructions: Computation blocks




```
a = b + c;
x = log(a);
y = a*a;
z = x*4;
```



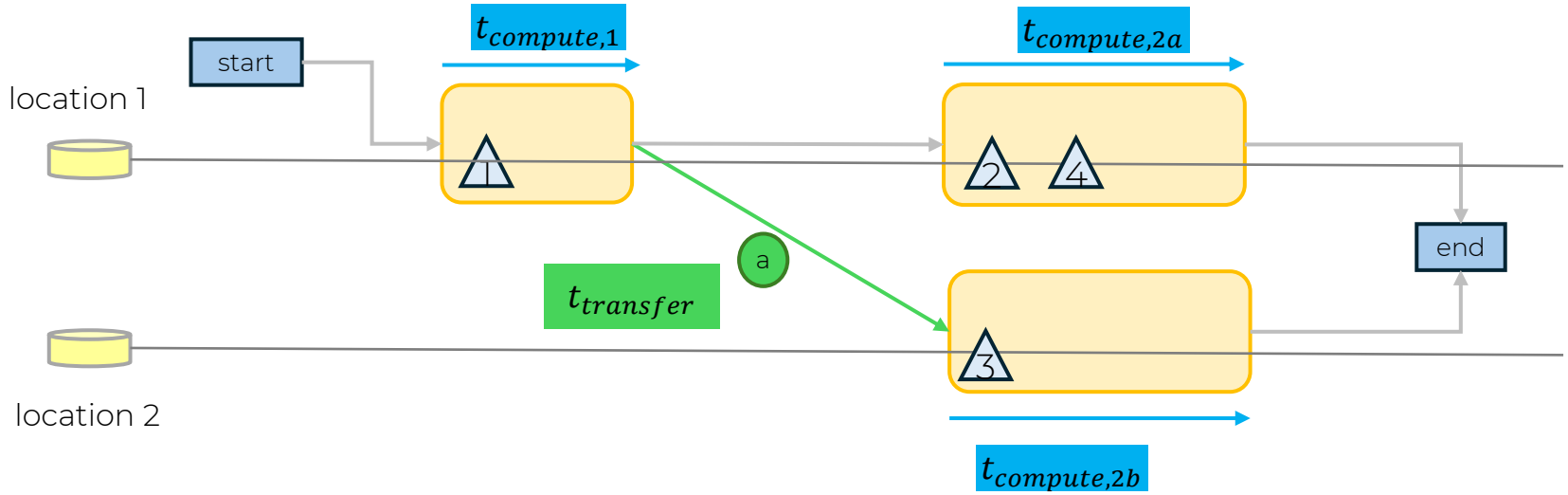
Computation blocks

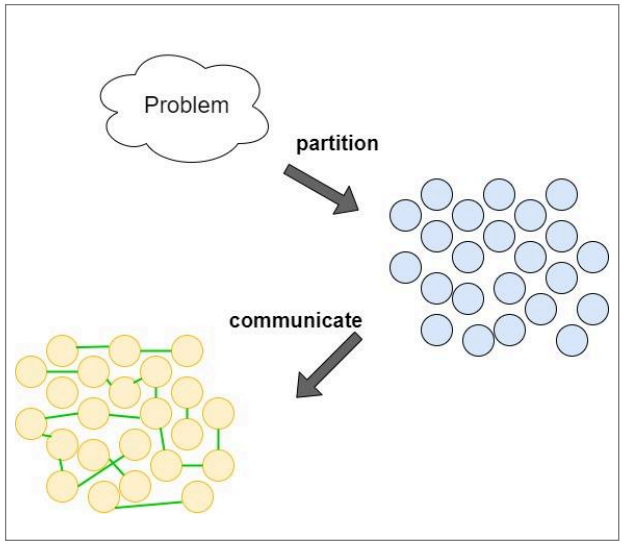
computation block
- direct RAW (no «gaps» = direct write->read) = true dependencies
- combinational
- group of instructions = physical fastest form of binary computing

potential transfers
- needed data transfers to compute a CB in parallel

Computation blocks and ideal scheduling

```
a = b + c;
x = log(a);
y = a*a;
z = x*4;
```



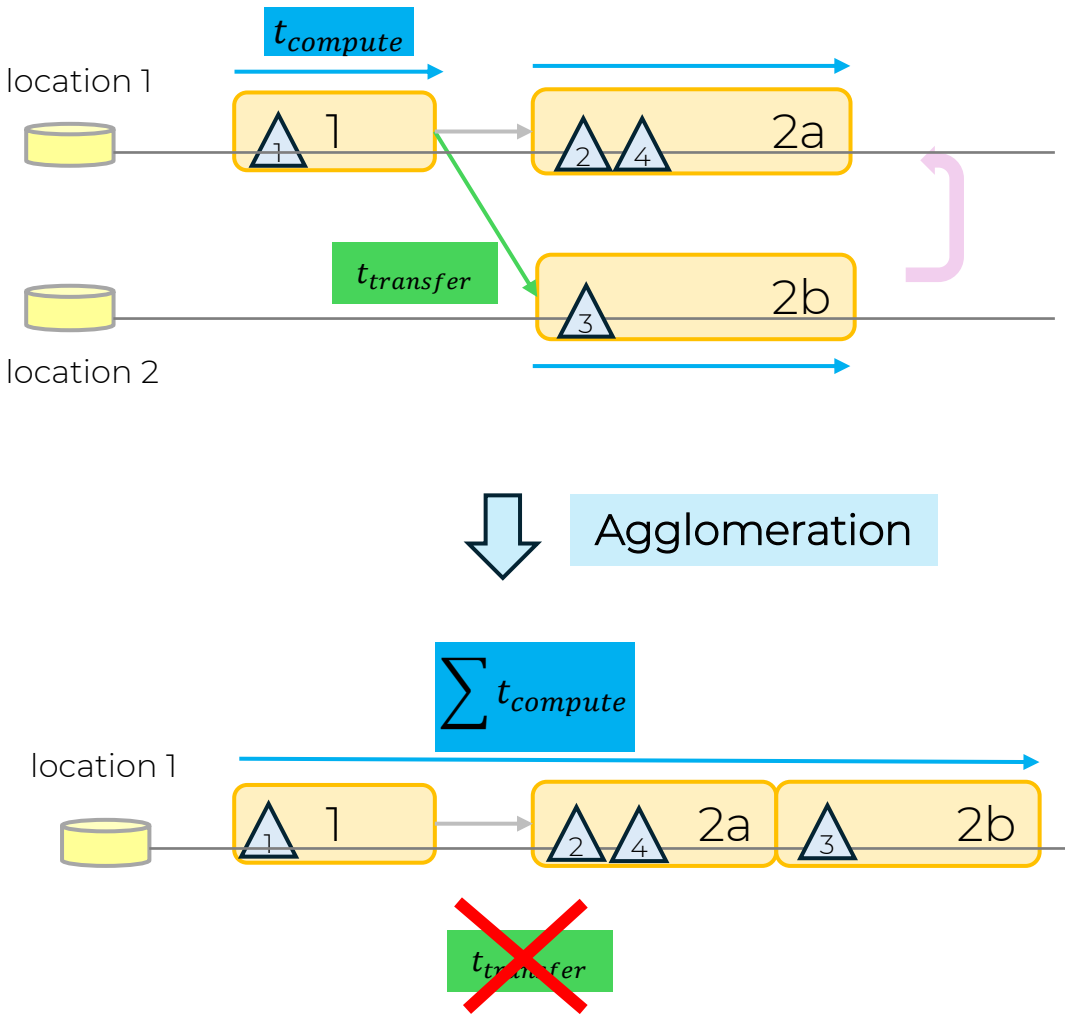


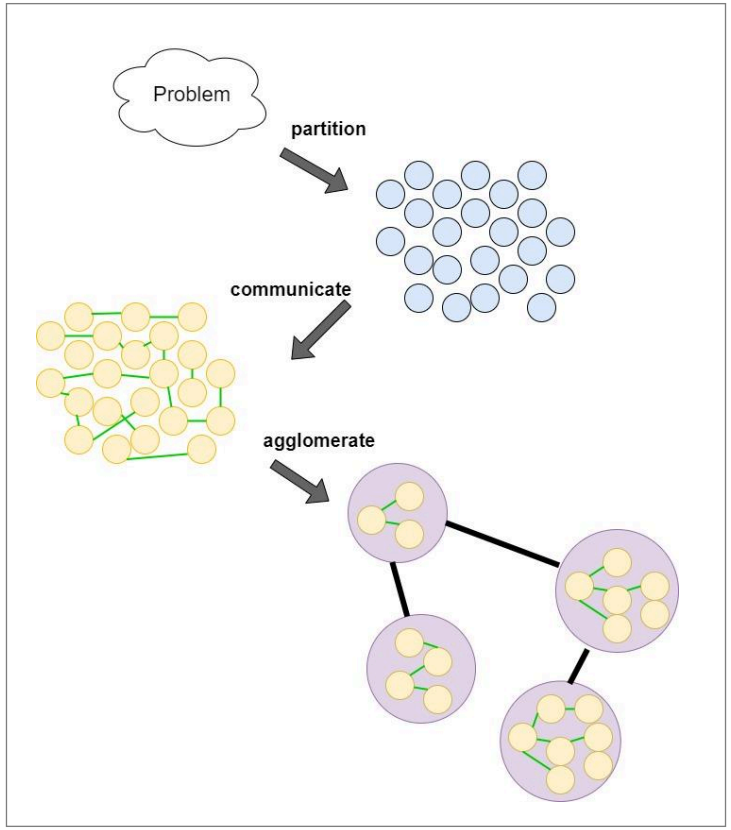
in case:

- no resource restrictions in locations
- locations have same performance properties

→ ideal scheduling

Agglomeration of computation blocks





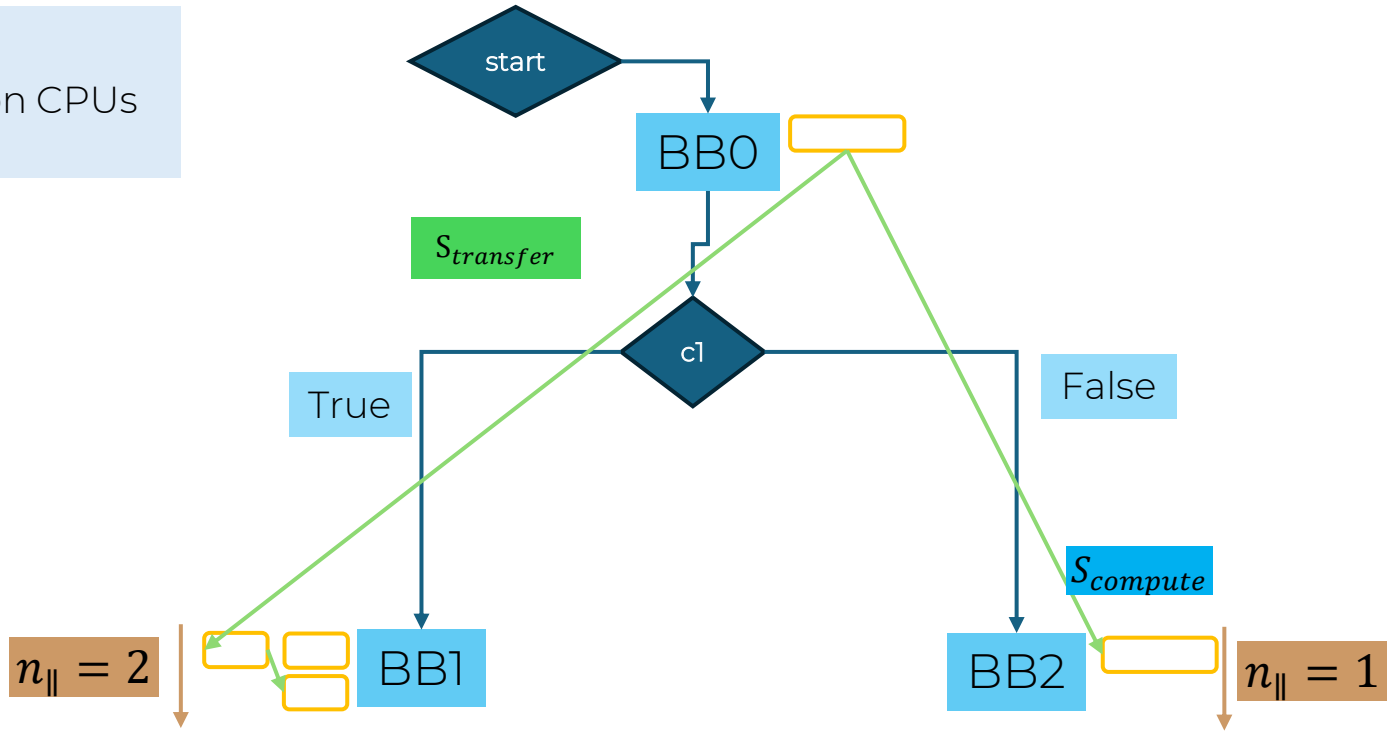
agglomerate by combination of parallel CBs:

- computations sum up
- transfers vanish

→ increasing **task granularity**

Computation blocks and control dependencies

- every control dependency introduces transfers
- **Benefit** of Computation Blocks over Basic Block on CPUs **limited** -> covered by Hardware-support



- $\boldsymbol{n_{\|}}$: number of parallel Computation Blocks as a function of conditions = runtime variables
- Each computation block:
  - size to transfer $\boldsymbol{S_{transfer}}$
  - size to compute $\boldsymbol{S_{compute}}$

Application in the middle-end

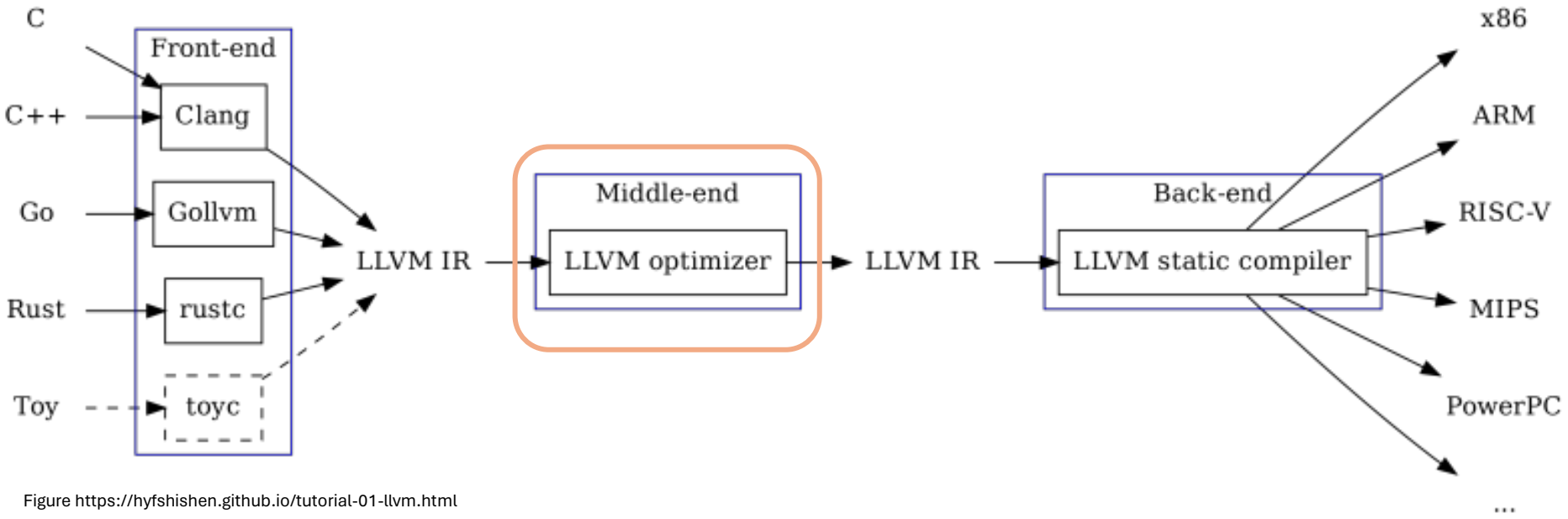


Figure https://hyfshishen.github.io/tutorial-01-llvm.html

Recursive implementation Fibonacci sequence

```
int fib(int n)
{
    if (n <= 1)
        return n;
    return fib(n - 1) + fib(n - 2);
}
```

LLVM IR [10]

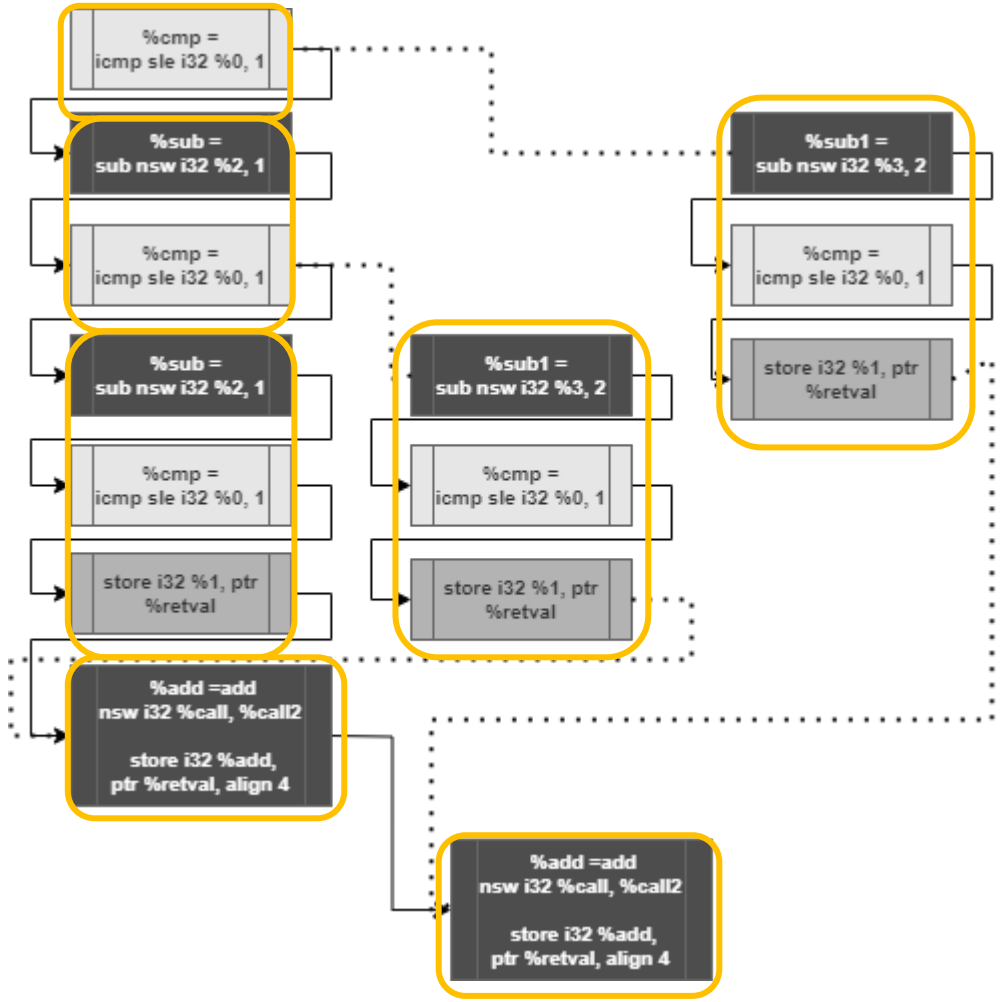
```
entry:
  %retval = alloca i32, align 4
  %n.addr = alloca i32, align 4
  store i32 %n, ptr %n.addr, align 4
  %0 = load i32, ptr %n.addr, align 4
  %cmp = icmp sle i32 %0, 1
  br i1 %cmp, label %if.then, label %if.end

if.then:
  %1 = load i32, ptr %n.addr, align 4
  store i32 %1, ptr %retval, align 4
  br label %return

if.end:
  %2 = load i32, ptr %n.addr, align 4
  %sub = sub nsw i32 %2, 1
  %call = call noundef i32 @fib(int)(i32 noundef %sub)
  %3 = load i32, ptr %n.addr, align 4
  %sub1 = sub nsw i32 %3, 2
  %call2 = call noundef i32 @fib(int)(i32 noundef %sub1)
  %add = add nsw i32 %call, %call2
  store i32 %add, ptr %retval, align 4
  br label %return

return:
  %4 = load i32, ptr %retval, align 4
  ret i32 %4
}
```

godbolt.org -> x86-64 clang 18.1.0

recursive call for fib(3)

potential transfers

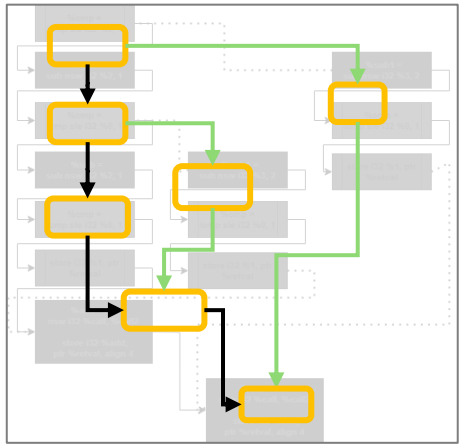
Computation blocks

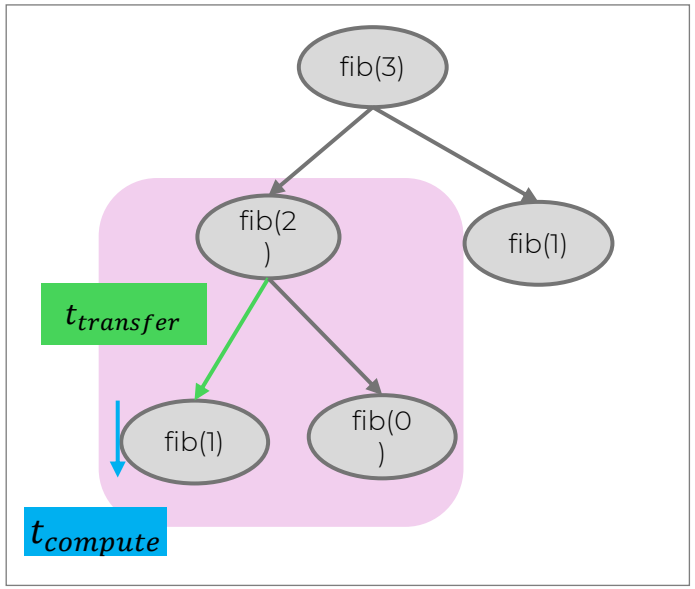


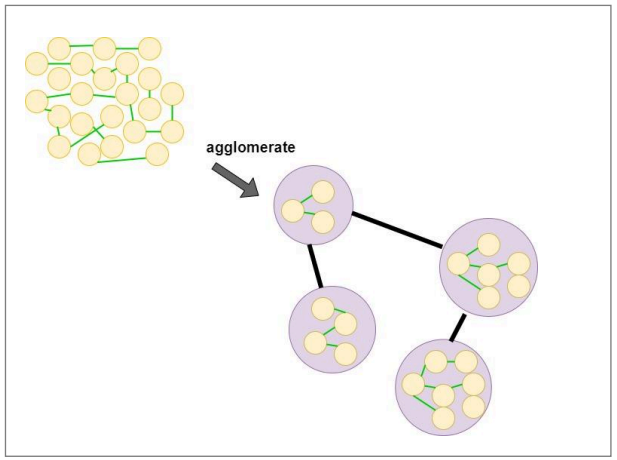
Agglomeration

«familiar» task graph

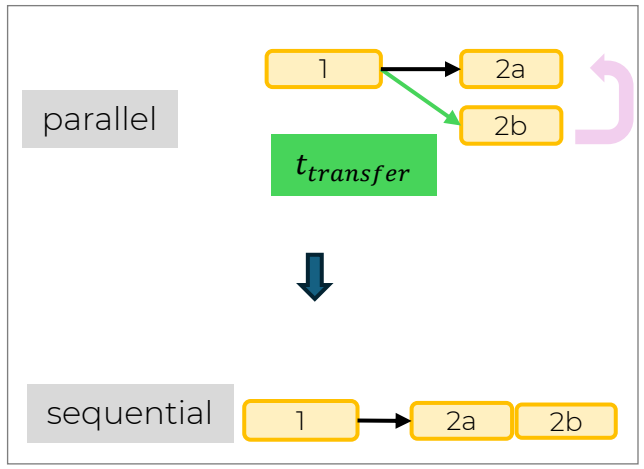
CB-graph



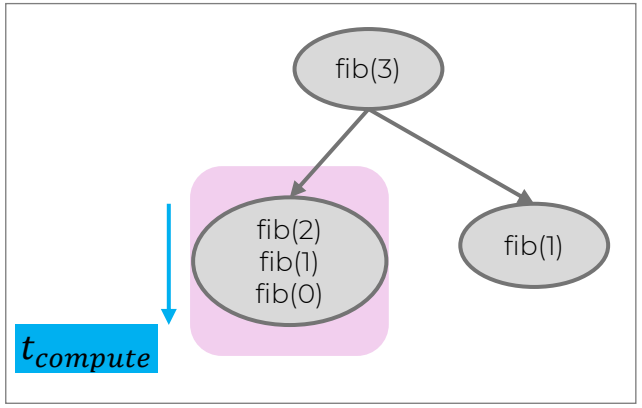




agglomerate

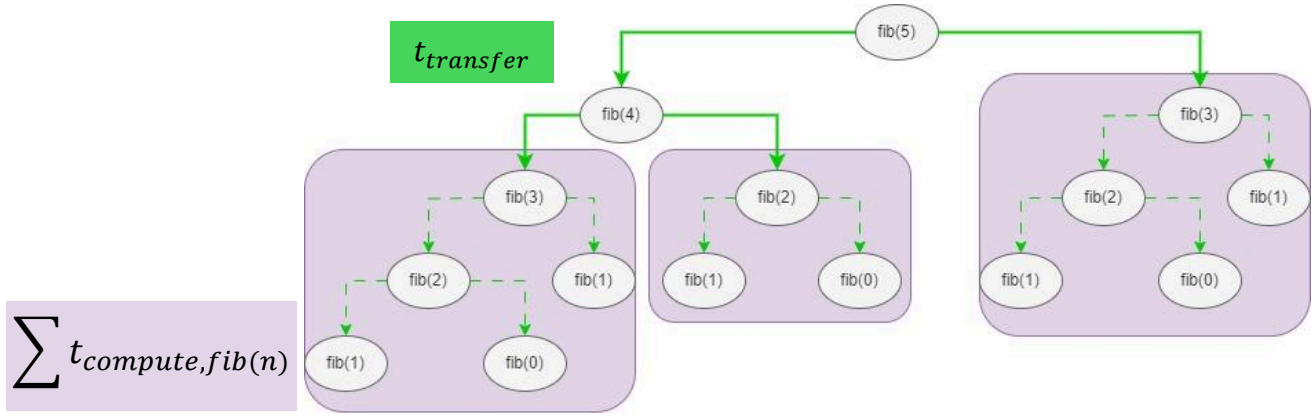


- Aggregate child-nodes in tree
- balance $t_{compute}$ to overhead $t_{transfer}$



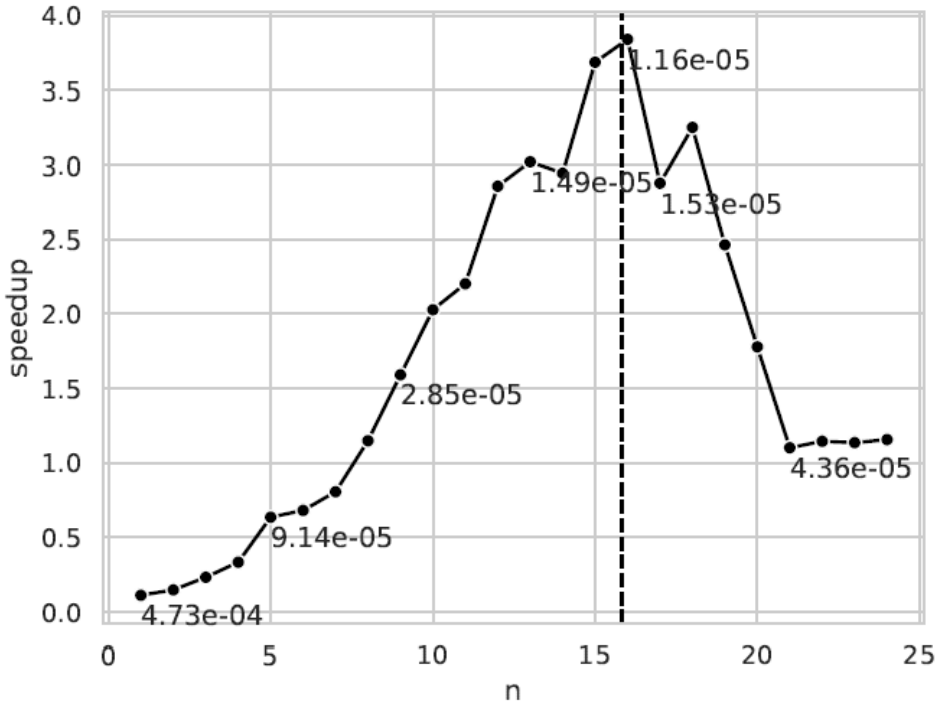
Optimise to hardware

- Benchmarking (speedup) [1]
  - using oneTBB Task scheduler [2]
  - start each function call as a seperate task
  - compare to grouped task by threshold (=n)



- forecast: when context switch overhead beneficial
- Intel Xeon CPU (12 cores, 24 threads)
- $t_{transfer} = t_{contextswitch} = 3860ns$
- $t_{compute} = t_{IPC} = 1.13$
- Cumulate functions calls (t_{IPC}) till time longer then transfer time ($t_{contextswitch}$)

→ run all computations for $n \leq 16$ in **one** task



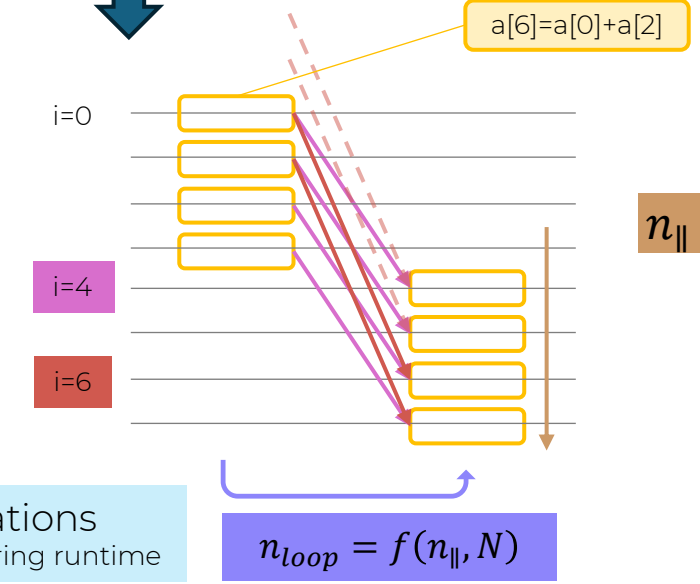
[1] COMP 322: Fundamentals of Parallel Programming Module 1: Parallelism, Vivek Sarkar and Mackale Joyner
[2] https://github.com/oneapi-src/oneTBB

Computation blocks in loops

```
for (i=0, i<N, i++) {
        a[i+6] = a[i]+a[i+2];
}
```

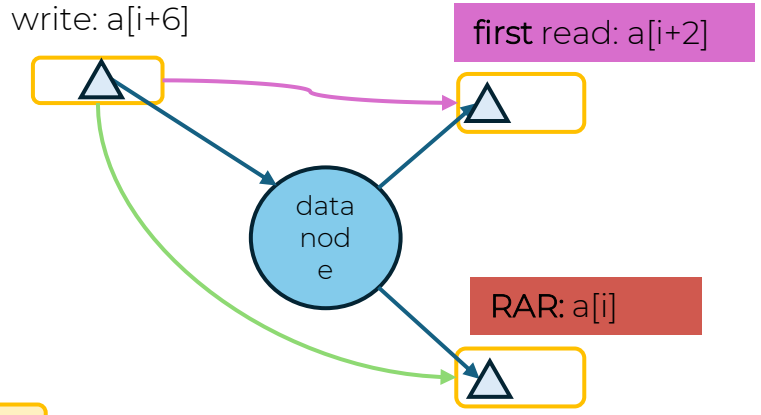
index distances

$$\overrightarrow{\Delta rw} = \begin{bmatrix} (i+6)-(i) \\ (i+6)-(i+2) \end{bmatrix} = \begin{bmatrix} 6 \\ 4 \end{bmatrix}$$



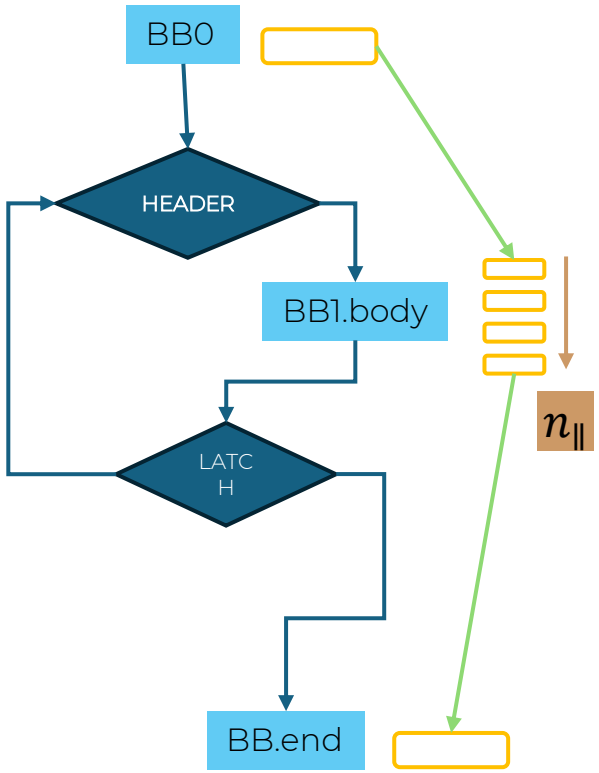
iterations
known during runtime

$$n_{loop} = f(n_{\|}, N)$$



unlimited resources

$$n_{\|} = first\ read = 4 \neq f(N)$$

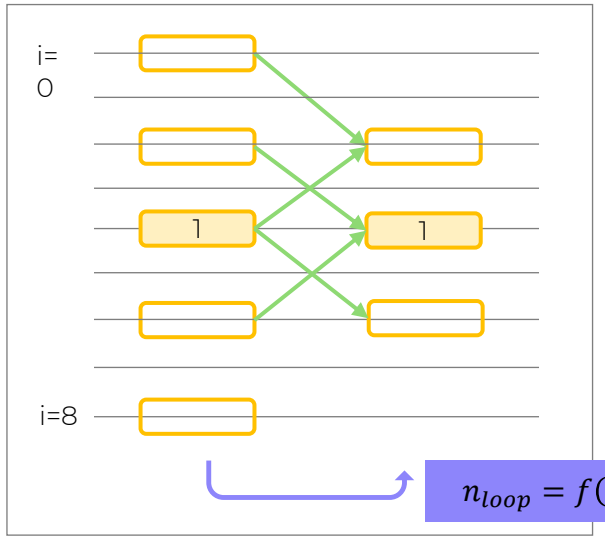


- $n_{\|}$ = function distance index vector $\overrightarrow{\Delta rw}$
- known at compile time
- do not need to unroll loop to define $n_{\|}$

Agglomerate parallel CBs // minimize transfers

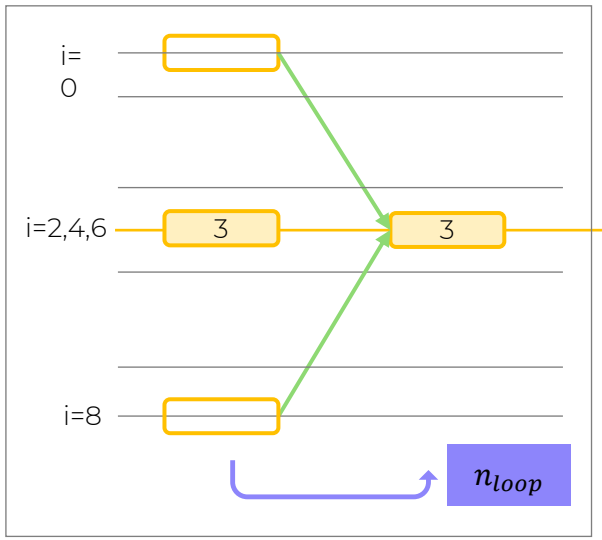
- $n_{\parallel}$: parallel CBs with ideal parallelism (unlimted number of available units)
- Real-world: aggregating to number of available units $n_{locations}$
- Map:
  1. combine parallel CBs $n_{\parallel}$ -> more computing, less transfers
  2. Exploit Read-after-Read information to optimise mapping

a[i+2] = a[i] + a[i+4]

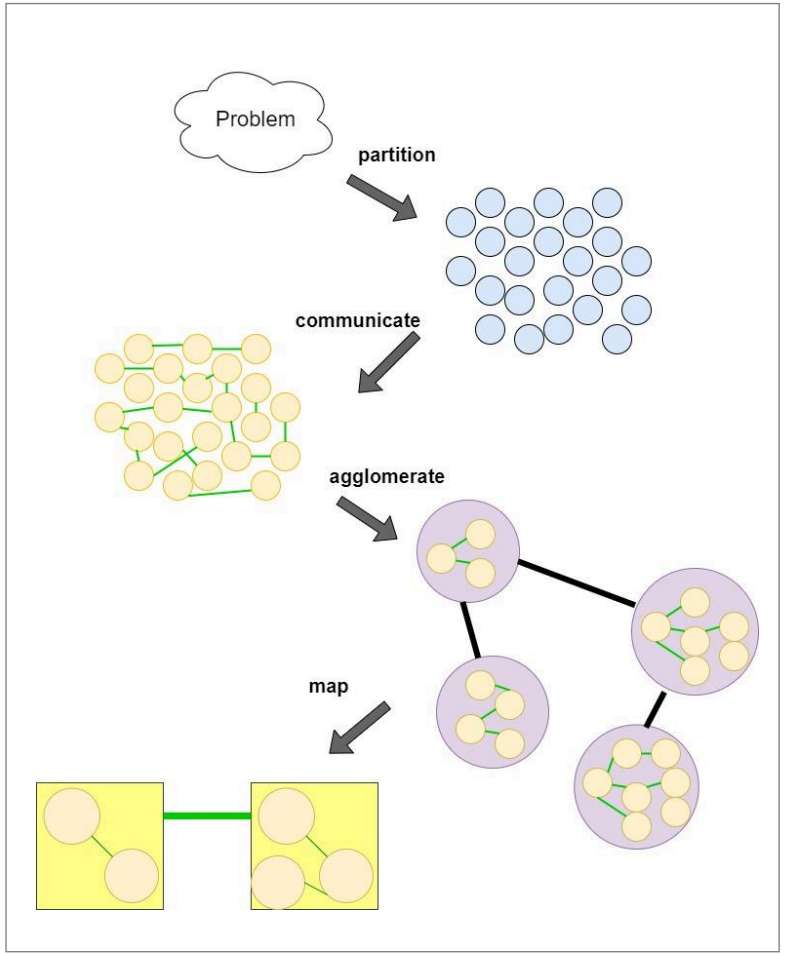


distance index vector

$\overrightarrow{\Delta rw} = \begin{bmatrix} 2 \\ -2 \end{bmatrix}$



location

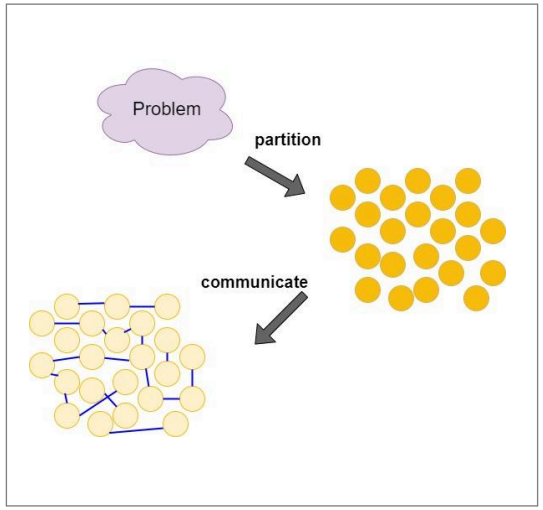


A classical problem: 2D heat equation

1) PDE

$$\frac{\partial u}{\partial t} - \alpha\left(\frac{\partial^2 u}{\partial x} + \frac{\partial^2 u}{\partial y}\right) = 0$$

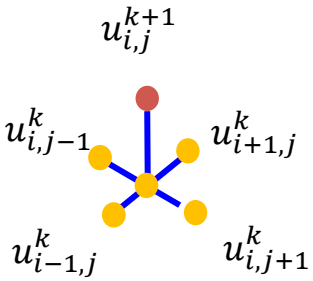
Finite Difference (FD)



2) mesh points

$$u_{i,j}^{k+1} = \gamma\left(u_{i+1,j}^{k} + u_{i-1,j}^{k} + u_{i,j+1}^{k} + u_{i,j-1}^{k} - 4u_{i,j}^{k}\right) + u_{i,j}^{k}$$

3) stencil



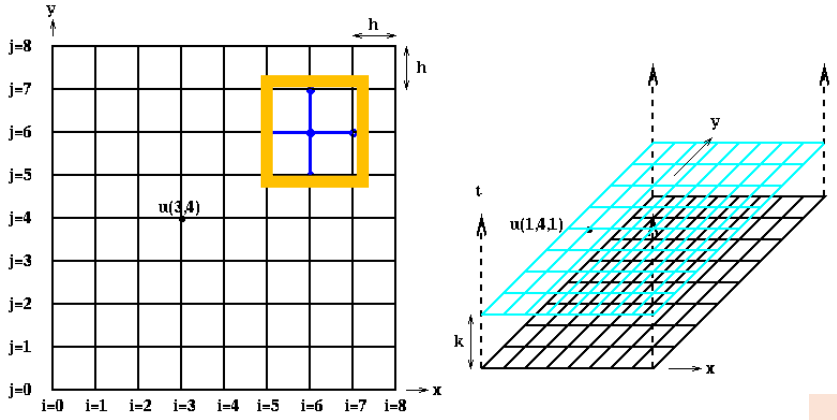
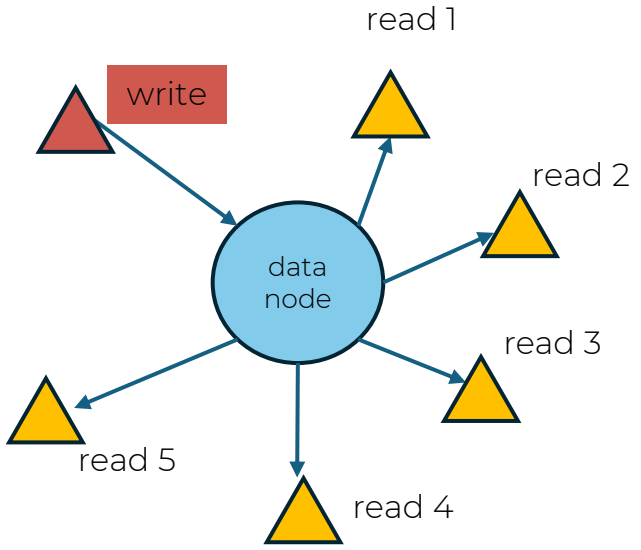


Figure https://people.eecs.berkeley.edu/~demmel/cs267/lecture17/lecture17.html



→ many reads
→ not auto-parallelizable with sota compiler

Get maximal parallel partitions

problem

$$u_{i,j}^{k+1} = \gamma\left(u_{i+1,j}^{k} + u_{i-1,j}^{k} + u_{i,j+1}^{k} + u_{i,j-1}^{k} - 4u_{i,j}^{k}\right) + u_{i,j}^{k}$$

code

```
void compute(double ***u, double gamma, int max_iter_time, int plate_length, int plate_width) {
    for (int k=0; k<max_iter_time-1; k++) {
        for (int i=1; i<plate_length-1; i++) {
            for (int j=1; j<plate_width-1; j++) {
                u[k+1][i][j] = gamma * (u[k][i+1][j] + u[k][i-1][j] + u[k][i][j+1] + u[k][i][j-1] - 4*u[k][i][j]) + u[k][i][j];
}}}}
```

memory
access only with natural numbers (i,j,k)

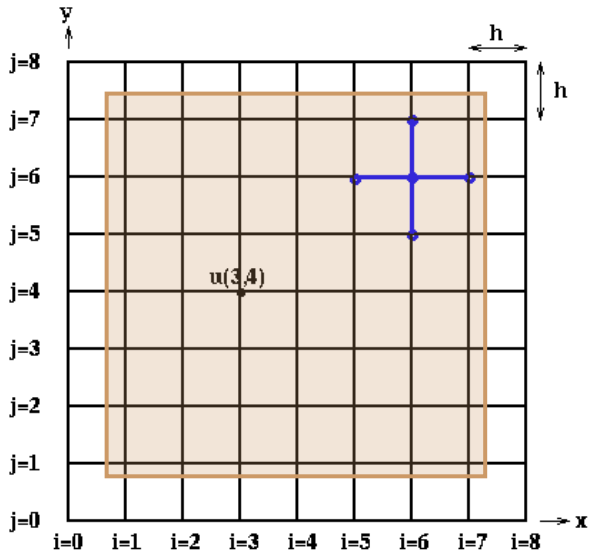
$$u[i][j][k] = (i) + n_x \cdot \left(j + n_y \cdot k\right)$$

distance index vector

$$\overrightarrow{\Delta rw} = \begin{bmatrix} u_{i,j}^{k+1} - u_{i+1,j}^{k} \\ u_{i,j}^{k+1} - u_{i-1,j}^{k} \\ u_{i,j}^{k+1} - u_{i,j+1}^{k} \\ u_{i,j}^{k+1} - u_{i,j-1}^{k} \\ u_{i,j}^{k+1} - u_{i,j}^{k} \end{bmatrix} = \begin{bmatrix} (n_x \cdot n_y) - 1 \\ (n_x \cdot n_y) + 1 \\ (n_x \cdot n_y) - n_x \\ (n_x \cdot n_y) + n_x \\ (n_x \cdot n_y) + 0 \end{bmatrix}$$

RARs

first read-> $n_{\|}$



$$n_{\|} = f(n_x, n_y)$$

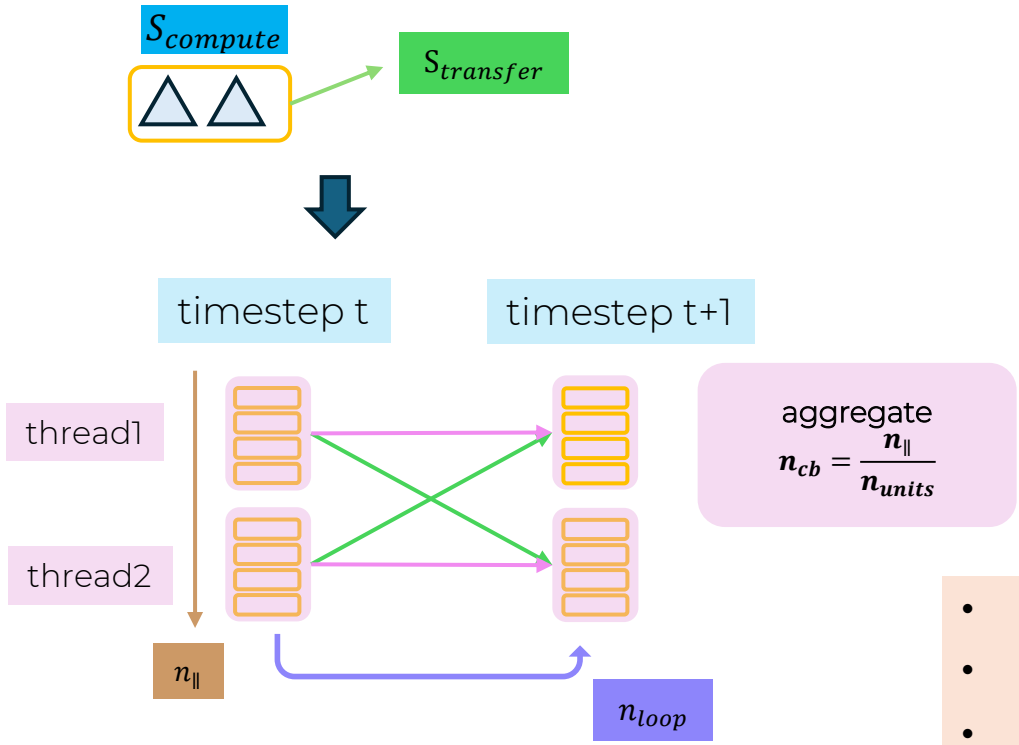
number of parallel CBs

$$n_{\|} = f(n_x, n_y) = (plate\_length - 2) \cdot (plate\_width - 2)$$

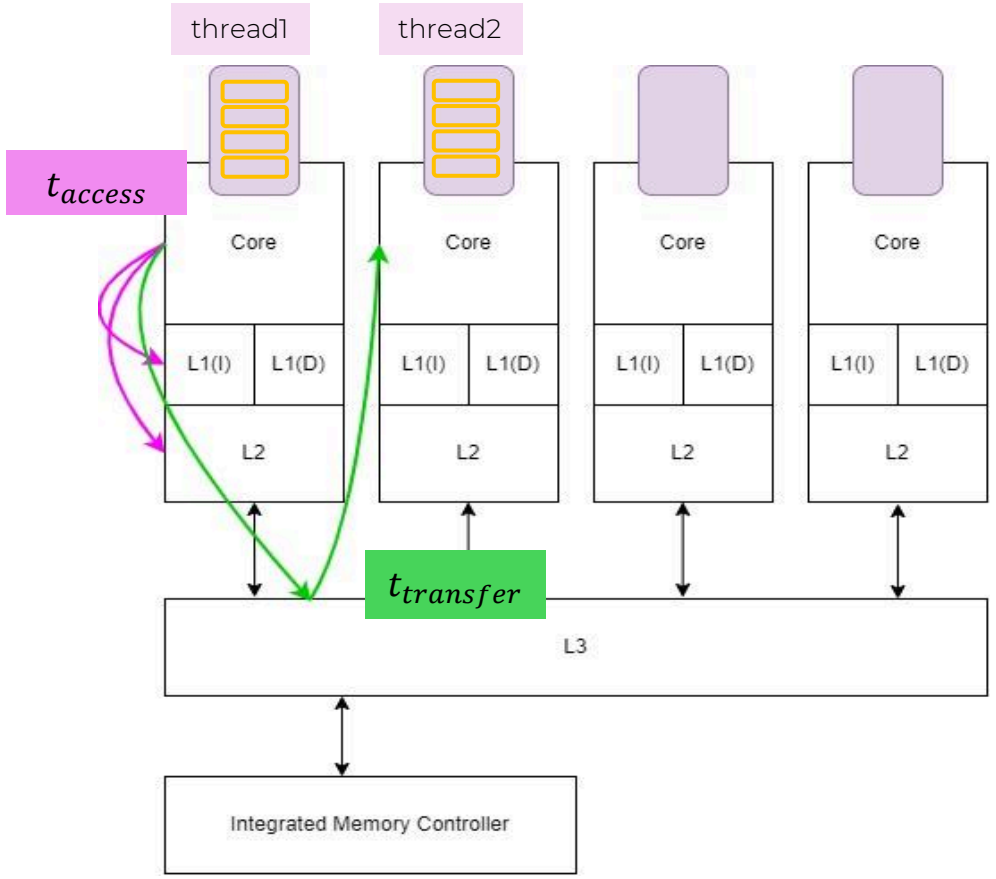
from loop-headers

Optimise mapping to cache-levels

- large mesh -> large number of parallel CBs $n_{\parallel}$
- symmetrical platform: $\frac{n_{\parallel}}{n_{units}}$
- data size of problem to compute in parallel is known: $\sum_{n_{\parallel}} S_{compute}$
- balance the size per thread to optimise cache-level



aggregate
$n_{cb} = \frac{n_{\parallel}}{n_{units}}$

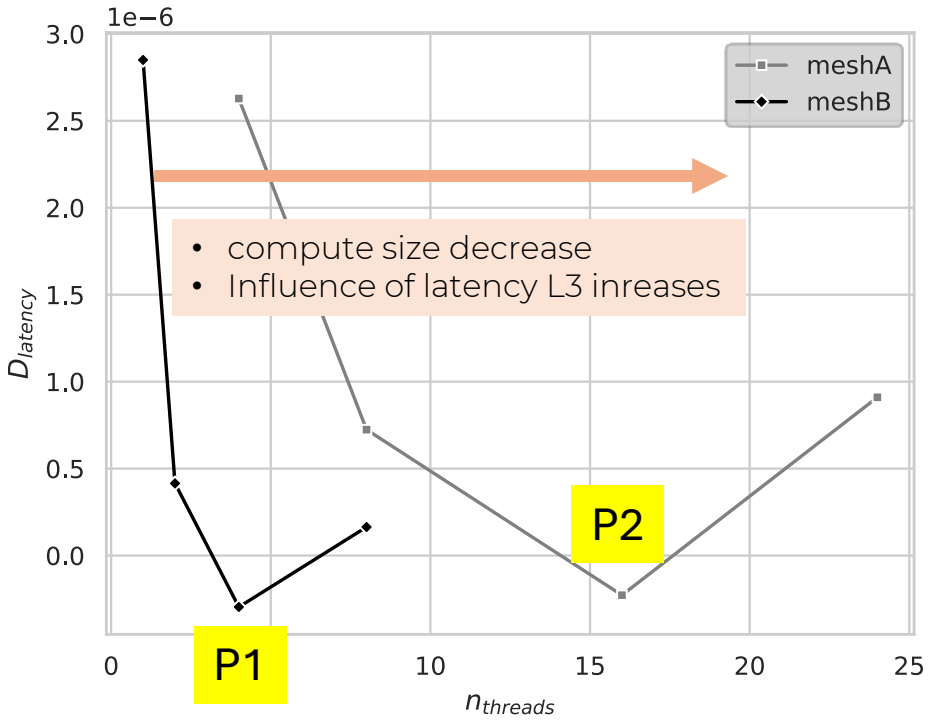


- distribute parallel CBs to available n_{units}
- LACOS = size to compute in parallel $S_{compute}$ and to transfer $S_{transfer}$
- approximate access latencies between iterations using L3-properties
- get optimal number of parallel threads

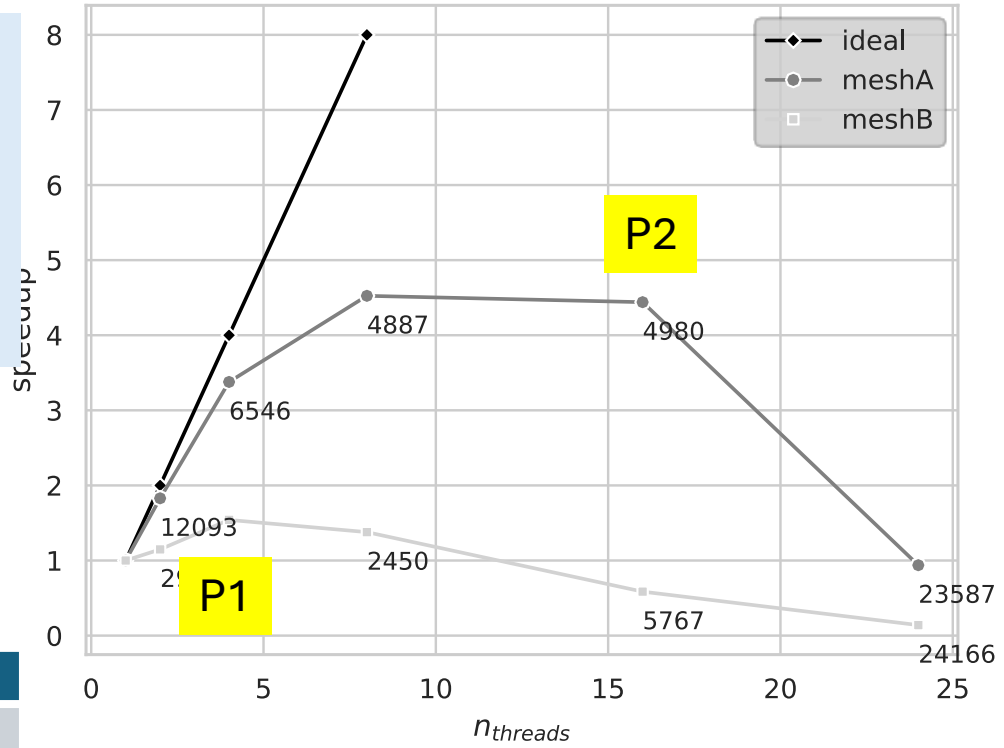
Mapping Computation Blocks to threads

- Approximate data access latencies:
  - compute $t_{compute} = f(n_{cbs,thread}) = f(IPC)$
  - access data $t_{access} = f(S_{computecbs,thread}) + f(S_{transfer})$
  - latency time depending on L1 and L2 cache access = known: $S_{computecbs,thread}$
  - exchange only by L3 (Intel Xeon CPU) -> latency known: $S_{transfer}$

  → $D_{latency}$



	$n_{\parallel}$
Mesh A	17464
Mesh B	3264



- LACOS was able to:
  - **produces parallel code** from loop-section
  - define **optimal number of threads** (two hardware properties)
- **during compile time** and automatically

Application in the back-end

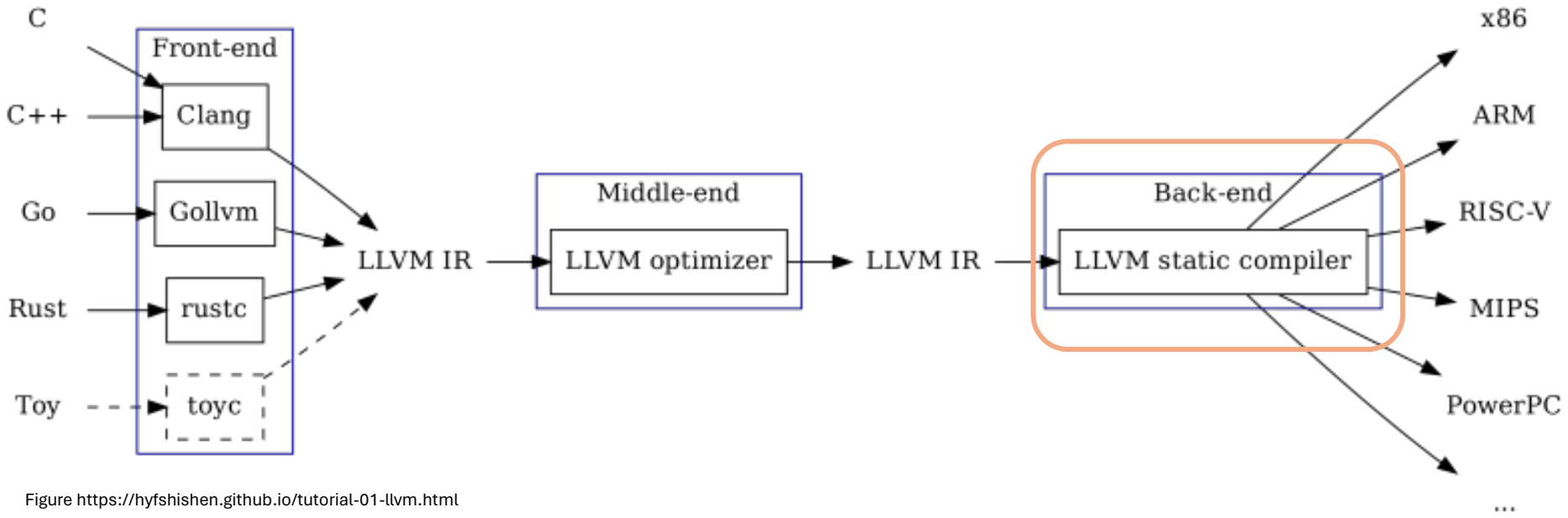
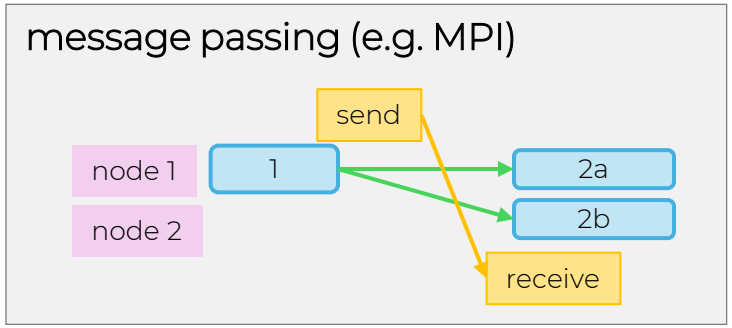
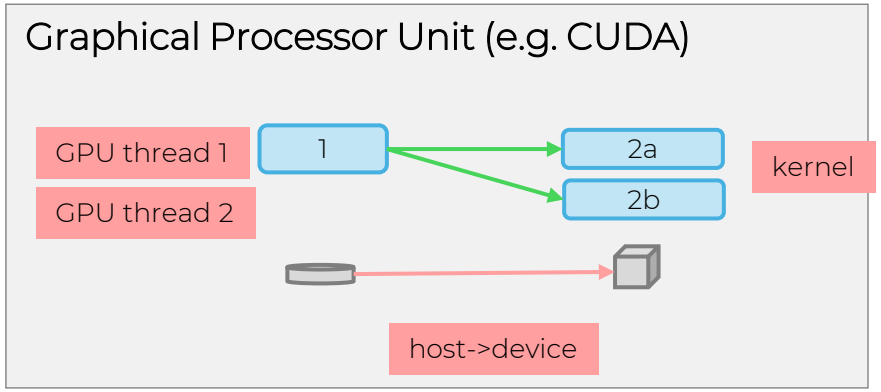


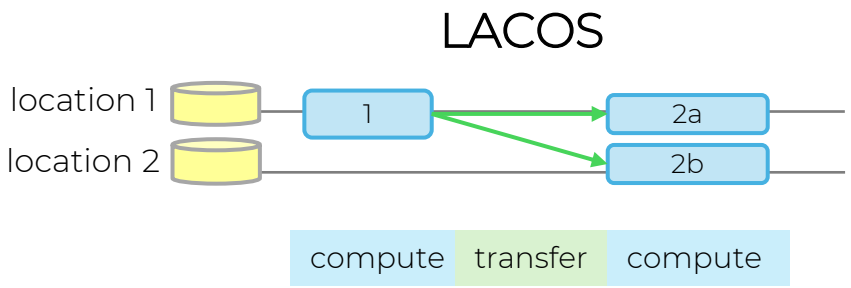
Figure https://hyfshishen.github.io/tutorial-01-llvm.html

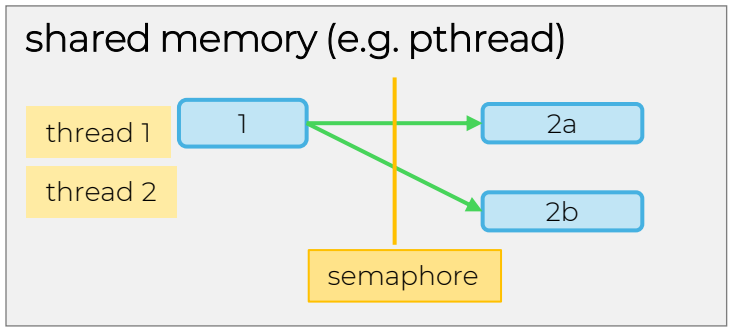
Use Computation Blocks in back-end

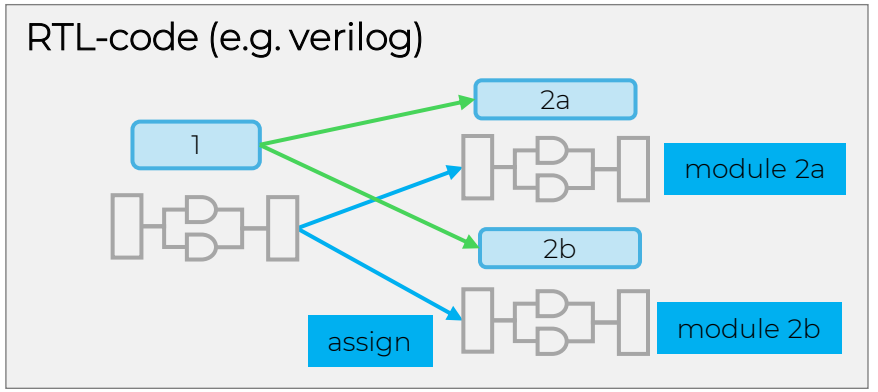
mynatix











Synthesis

mynatix

Synthesis

- RARs seem to be forgotten, but contain valuable information for auto-parallelization
- LACOS:
  - finds maximal number of parallel chains of arbitrary sequential instructions
  - provides generic aggregation to build tasks with optimal granularity
  - Physics-based

First results with selected cases

- analytic solution to map for symmetrical platforms
- auto-parallelization not possible with state-of-the-art methods
- Successfull transpiling to:
  - pthread, OpenMP (multicore CPU)
  - NVIDIA CUDA (GPU)
  - MPI (cluster)
  - freeRTOS (microcontroller)
  - verilog (FPGA)

Outlook

- More benchmarking and comparison with established methods and platforms needed
- We are building a service targeting interpreted languages – only small aspect of potential applications
- We see industry potential for example: i.e. in compilers, task schedulers, for High Level Synthesis
- Looking for cooperation: industrial and institutional -> please get in touch!

Thank you – any questions?

Contact details: Dr. Andres Gartmann

- by email: andres.gartmann@mynatix.com
- linkedin: https://ch.linkedin.com/in/andres-gartmann-737aa212a