Scientific and Technical Computing

Hardware and Code Optimization

Lars Koesterke

UT Austin, 8/31/20 & 9/8/20 & …

Discussion

What are the **primary components** of a computer?

Discussion

What are the **primary components** of a computer?

Can you already detect some limitations?

Discussion

What are the **primary components** of a computer?

- CPU
- Memory
- (Storage)
- (Motherboard, lots of wires)
- (Keyboard, screen)

Can you already detect some limitations?

- Number of pins connecting CPU and Memory to motherboard

TACC

Scope of this class

What **sciences and technologies** are

involved in designing and building a

CPU/Memory/Computer?

Scope of this class

What sciences and technologies are involved in designing and and building a CPU?

- Electrical engineering, physics, chemistry, math … (all the things you study)
- Cutting edge research
- A lot of institutional knowledge by people in companies and research institutions
  - Not everything is an exact science; many tricks are applied (a bit like cooking)
  - Design decisions are made on incomplete facts (humans weigh the pros and cons)
- Certainly, many computers equipped with previous generations of CPUs

This is **not** what we will talk about in this section of the class segment (Hardware and Code Optimization)

Scope of this class

What are we going to <u>explore</u> in the next 4 weeks?

High level overview of the architecture

- High level of abstraction
- Simplified implementation details
- Features that allow for a very high peak performance

<u>How to write code that exploits the hardware features?</u>

TACC

Scope of this class

Matching software to hardware: Why?

Much higher performance

- Orders of magnitude!

There is, of course, the idea to match the hardware to the software/purpose

This is done in other areas
In HPC the idea is not feasible (with one notable exception)

Conceptual understanding of hardware features guides software design

Assumption: You are in this class (and other TACC classes) to learn how to

- learn about high-performance computing (HPC)
- use a supercomputer (or any computer!) in an **efficient** and **effective** way
- write **fast** code
    - This class: exploiting parallelism of the hardware
    - Note: some/many bad code design decision cannot be reversed later
- write **parallel** code with OpenMP and MPI (PCSE in the spring semester)

Getting some scientific calculations done
Better than the competition

Terminology

Today's supercomputers are clusters

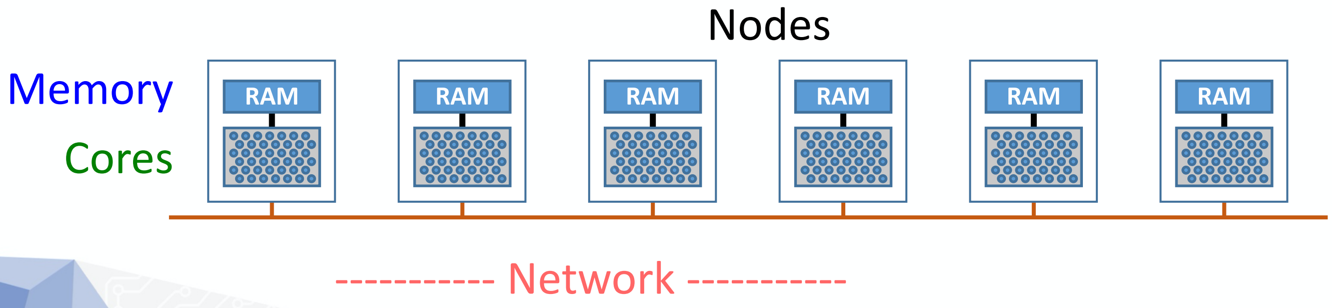
Components of a cluster

- Nodes
  - CPUs and memory
- Network
- (File system)

In this class we strongly focus on a single-node!

A cluster is build of individual computers which are called nodes

The nodes are connected through an interconnect



Discussion

What is a clock tick?

What is a clock cycle?

Discussion

What is a clock tick?

What is a clock cycle?

What does this mean?

Clock frequency = 2.2 GHz

Discussion

What is a clock tick?

What is a clock cycle?

Think of an assembly line

- Smallest unit of time to 'do' something
- One or multiple instructions are executed
- An instruction may take several cycles

Examples of instructions are

- Multiply two numbers
- Load data into a register

'Instruction' can mean many things. Let's leave it a bit vague for now

Some answers from the web

Computers use an internal clock to synchronize all of their calculations. The clock ensures that the various circuits inside a computer work together at the same time.

Same as a cycle, the smallest unit of time recognized by a device. For personal computers, clock ticks generally refer to the main system clock, which runs at 66 MHz. This means that there are 66 million clock ticks (or cycles) per second. Since modern CPUs run much faster (up to 3 GHz), the CPU can execute several instructions in a single clock tick.

"The processor clock coordinates all CPU and memory operations by periodically generating a time reference signal called a *clock cycle or tick*. Clock frequency is specified in gigahertz (GHz), which specifies billions of ticks per second. Clock speed determines how fast instructions execute. Some instructions require one tick, others multiple ticks, and some processors execute multiple instructions during one tick."

Intermission

Let’s talk about how to proceed

<u>I’d like to organize this class having this in mind:</u>

- What and how to learn?
- What will be on the slides?
- How to <u>participate?</u>
- Give me <u>feedback!</u>
- How participation will affect your grade?
- Teamwork
- Let me know (at a later point) what you are interested in

TACC

Experiment: Prepare something at home

Your tasks

- Look up what the ‘Horner scheme’ is
  - Wikipedia entry (English Wikipedia site) is very good
  - To be specific, Horner’s notation: $((((z+\ldots)\times z+\ldots)\times z \ldots$
- Describe the ‘Horner notation’
  - Three to four sentences
  - General context (Note: you don’t have to explain what a polynomial is)
  - What is the ‘trick’?
  - Why would you use it when writing code?
- ‘Present’ in class next week
  - Nothing dramatic, I’ll explain
  - Write the 3 sentences down, if that helps you

TACC

Experiment: Prepare something at home

Your tasks

- Look up what the 'Horner scheme' is
  - Wikipedia entry (English Wikipedia site) is very good
  - To be specific, Horner's notation: ((((z+…)×z+…)×z …
- Describe the 'Horner notation'
  - Three to four sentences
  - General context (Note: you don't have to explain what a polynomial is)
  - What is the 'trick'?
  - Why would you use it when writing code?
- 'Present' in class next week
  - Nothing dramatic, I'll explain
  - Write the 3 sentences down, if that helps you

TACC

Let's make our computer do something

$$a = b + c$$

Let's make our computer do something

$$a = b + c$$

What needs to happen so that the CPU can calculate?

Where is the data coming from?

Where is the data going?

What part of the hardware performs the operation?

Let's make our computer do something

$$a = b + c$$

What needs to happen so that the CPU can calculate?

Where is the data coming from?	Memory
Where is the data going?	Memory
What part performs the operation?	Floating Point Unit (FPU) in the CPU (Central Processing Unit)

Let's make our computer do something

$$a = b + c$$

What needs to happen so that the CPU can calculate?

How long does it take?

Where is the data coming from?	Memory
Where is the data going?	Memory
What part performs the operation?	FPU in the CPU

Let’s make our computer do something

$$a = b + c$$

What needs to happen so that the CPU can calculate?		How long does it take?
Where is the data coming from?	Memory	A very long time
Where is the data going?	Memory	A very long time
What part performs the operation?	FPU in the CPU	A very short time

Let’s make our computer do something

$$a = b + c$$

What needs to happen so that the CPU can calculate?		How long does it take?
Where is the data coming from?	Memory	300 cycles
Where is the data going?	Memory	300 cycles
What part performs the operation?	FPU in the CPU	3 cycles

What a bummer! Actual work takes 0.5% of the total time
We may have built the ‘most ineffective computer’ ever!

TACC

Does this help us?

$$a(i) = b(i) + c(i)$$

Does this help us?

$$a(i) = b(i) + c(i)$$

Hint: where would you likely find such a statement?

Does this help us?

$$a(i) = b(i) + c(i)$$

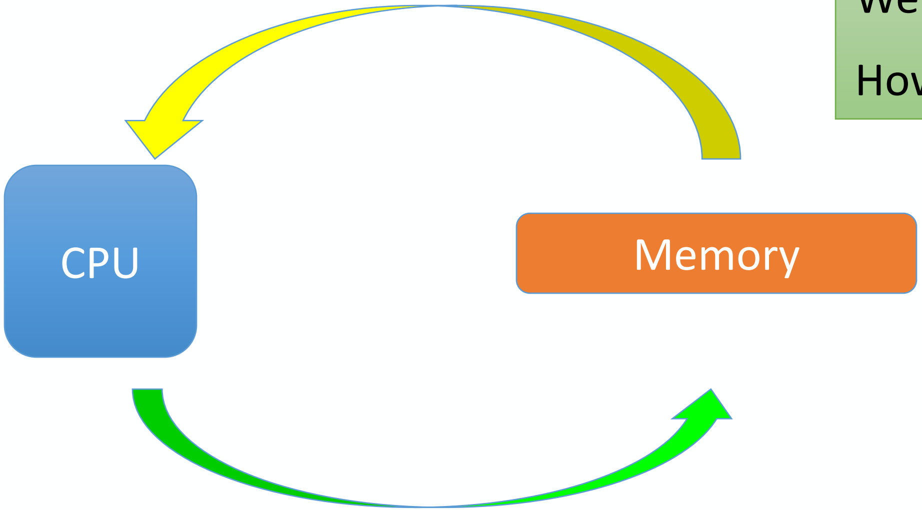
```
loop with index i
  a(i) = b(i) + c(i)
end loop
```

Data

a(i) = b(i) + c(i)

We have a lot of data to process

How can that help to **<u>'getting more done'</u>**?

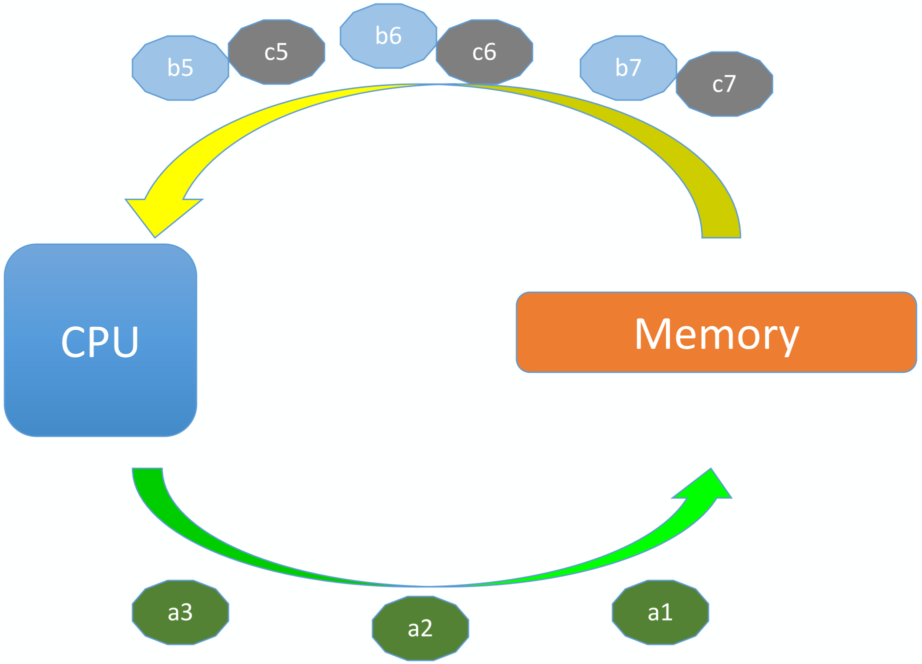


<u>'getting more done'</u>

We are mostly interested in floating point operations (flops) that move the calculation closer to the solution

Data Streams

a(i) = b(i) + c(i)



Some data is 'en route'

b and c: from memory to CPU

a: from CPU to memory

Some data is being processed

a(i) = b(i) + c(i)

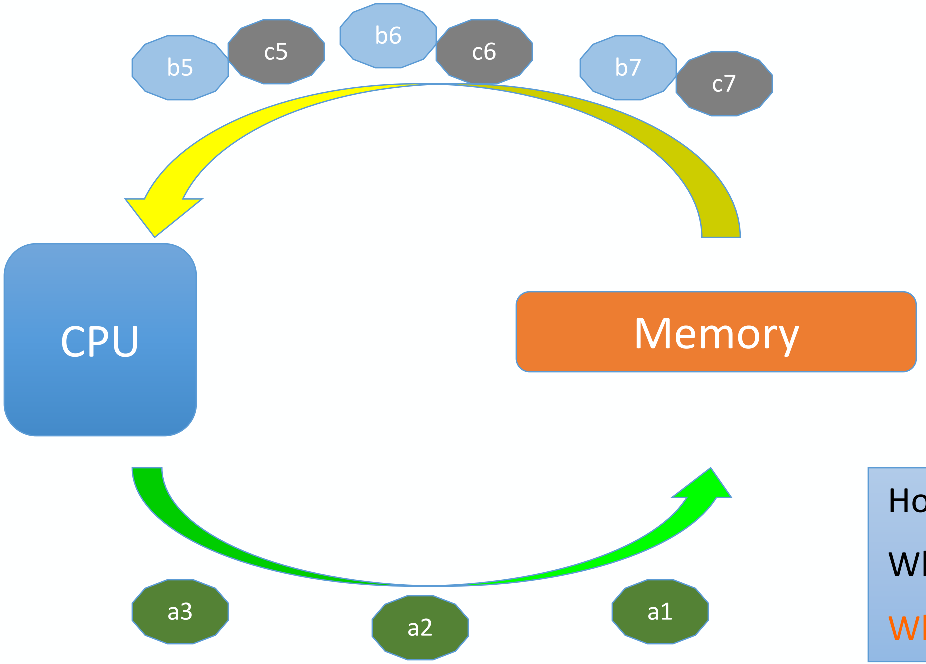
How much data has to be 'en route'?

What are the main factors?

TACC

Data Streams

$a(i) = b(i) + c(i)$



<u>Some data is 'en route'</u>

b and c: from memory to CPU

a: from CPU to memory

<u>Some data is being processed</u>

$a(i) = b(i) + c(i)$

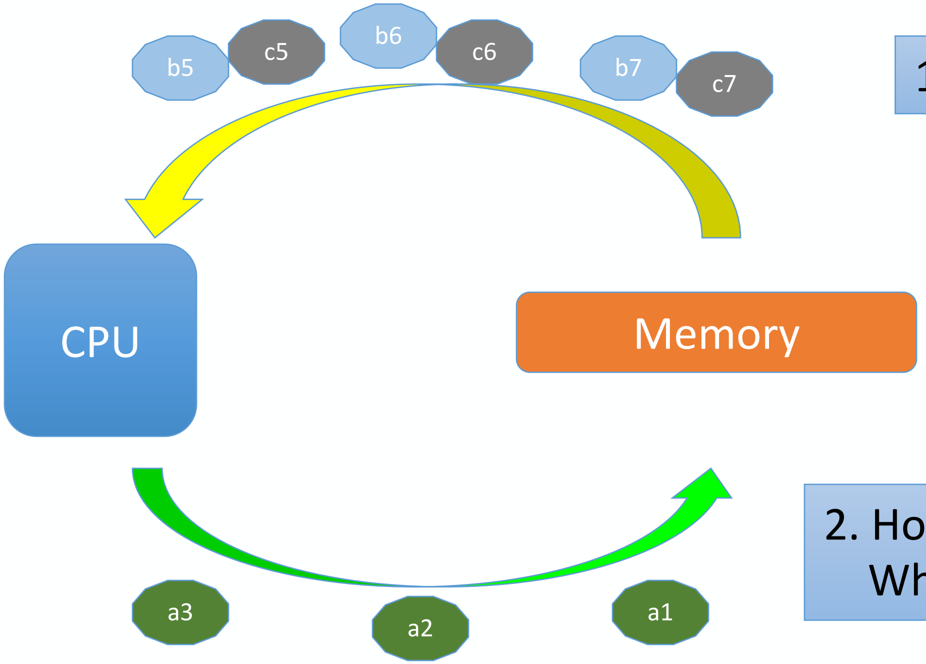
How much data has to be 'en route'?

What is one of the main factors?

What if I had drawn CPU and Memory further apart?

Data Streams

a(i) = b(i) + c(i)



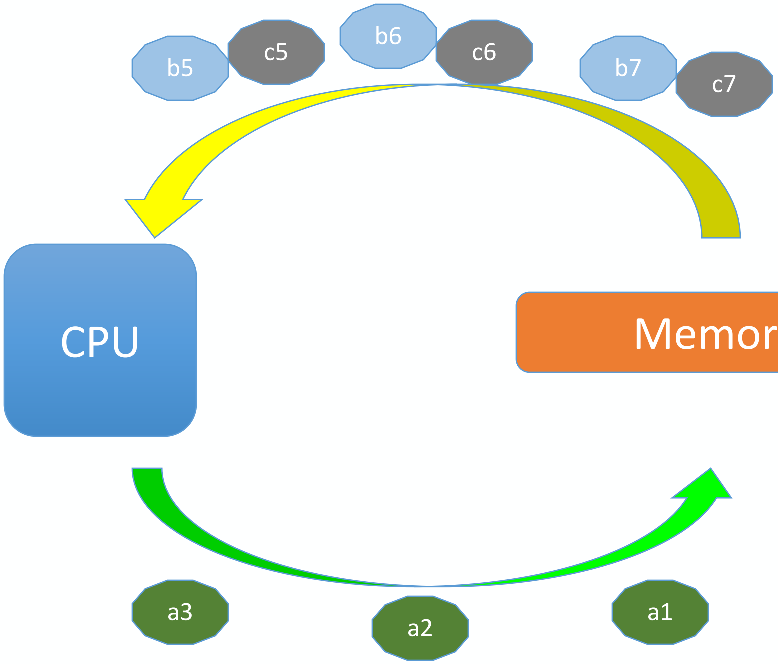
1. How much data has to be ‘en route’?

Operation:	3 cycles
Data transfer:	300 cycles

2. How many results ‘a’ do we get per cycle?
   What is the speedup?

Data Streams

a(i) = b(i) + c(i)



1. How much data has to be 'en route'?

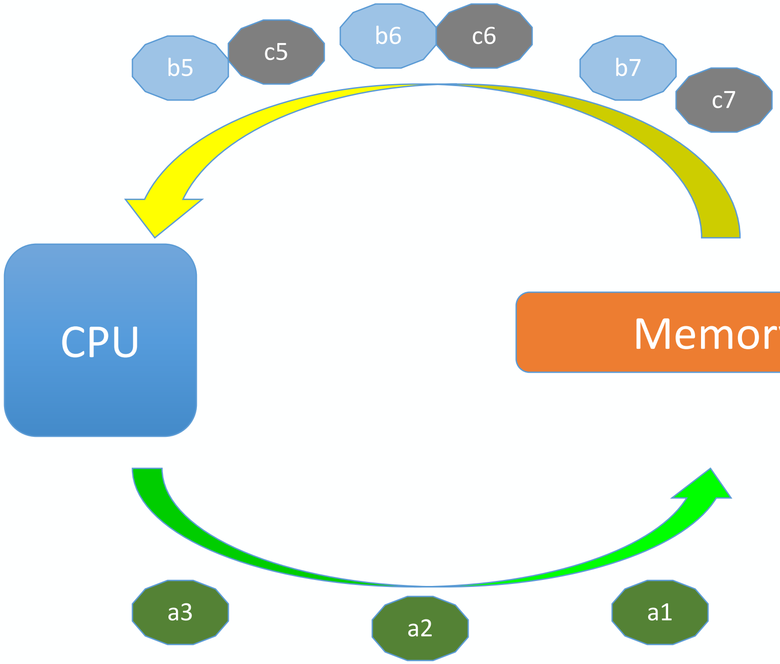
Operation: 3 cycles
Data transfer: 300 cycles

100 elements of a, b, and c

2. How many results (a) do we get per cycle?
What is the speedup?

Data Streams

$a(i) = b(i) + c(i)$



1. How much data has to be ‘en route’?

Operation:	3 cycles
Data transfer:	300 cycles
Ratio:	1/100

100 elements of a, b, and c

2. How many results (a) do we get per cycle?
   What is the speedup?

1 result every 3 cycles
Speedup is 100×

Hooray!

We have just discovered one of the most important hardware features in a CPU: Data streams

- Data Streams
  - Long distance (in terms of cycles) between main memory and CPU
  - Short time to execute ‘add’ operation (few cycles)
  - Streaming data: Data ‘en route’ filling the stream between memory and CPU
- Questions for later
  - How do we or the CPU ‘organize’ the data stream?
  - How does this look in code?

There are 2 fundamental bottlenecks

1. The data supply to the CPU
2. The actual operations (flops)

TACC

Hooray!

We have just discovered one of the most important hardware features in a CPU: Data streams

- Data Streams
  - Long distance (in terms of cycles) between main memory and CPU
  - Short time to execute ‘add’ operation (few cycles)
  - Streaming data: Data ‘en route’ filling the pipeline between memory and CPU
- Questions for later
  - How do we or the CPU ‘organize’ the data stream?
  - How does this look in code?

There are 2 fundamental bottlenecks

1. The data supply to the CPU
2. The actual operations (flops)

There are 3 major technologies that are applied
(we can argue about the exact number)

1. Data streams
2. ‘Improving CPU throughput’
3. ‘Improving data movement’

TACC

Discussion

Can we speed-up the actual numerical operation ‘add’?

If so, **how**?

Assume

1. ‘add’ takes 3 cycles
2. The data streams supply data effortlessly, i.e. without delay and at infinite bandwidth

TACC

Discussion

How can we speed-up the actual operation ‘add’?

Assume

1. ‘add’ takes 3 cycles
2. The data streams supply data effortlessly

Think of Henry Ford’s <u>moving</u> assembly line

The assembly line predates H. Ford (see R. Olds in the automotive industry; but earlier assembly lines in other industries)

Discussion: Pipelining

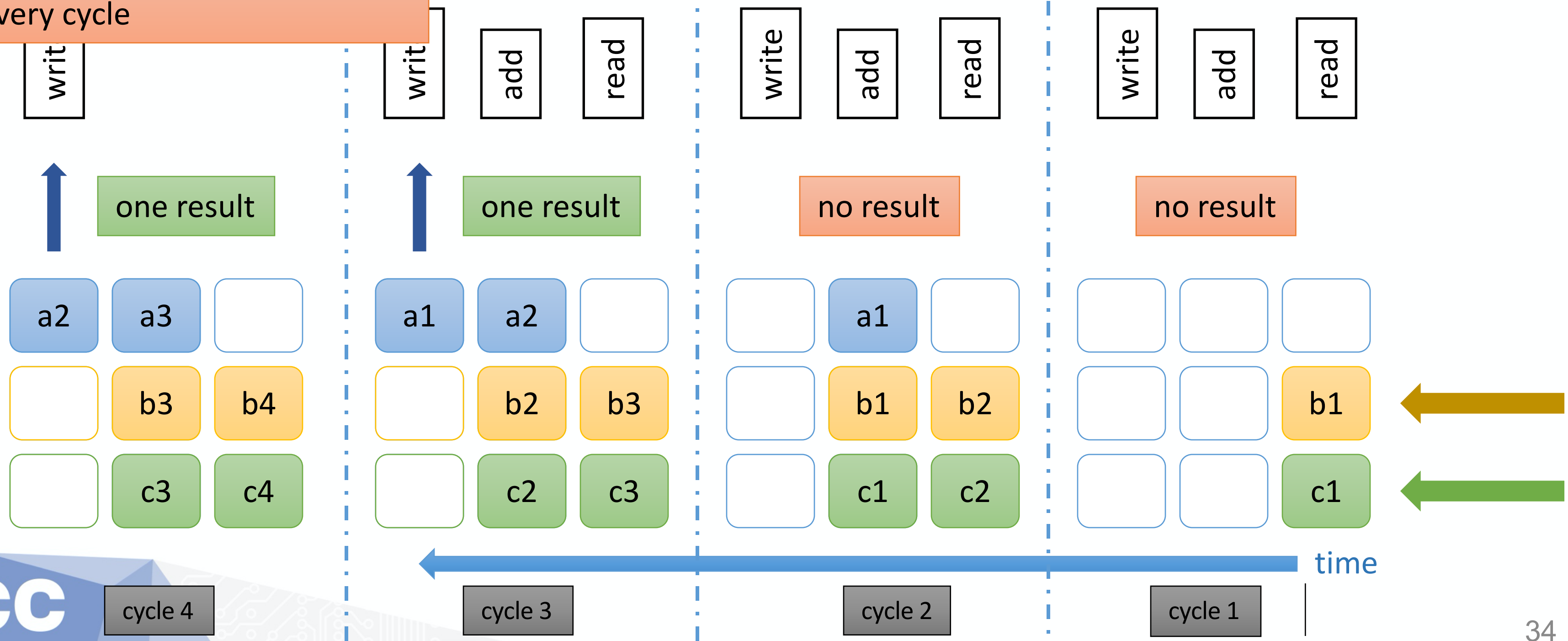
Simple toy model of the implementation of ‘add’

‘add’ takes 3 cycles: For sake of argument, ‘read’, ‘add’, write’

One result per cycle, once the pipeline is filled
3× performance increase

Operation: 1 cycle
Data transfer: 300 cycles
Ratio: 1/300
Ratio is now worse (no good deed goes unpunished)

After 2 cycles (ramp up phase) a result is produced every cycle



Discussion: Pipelining

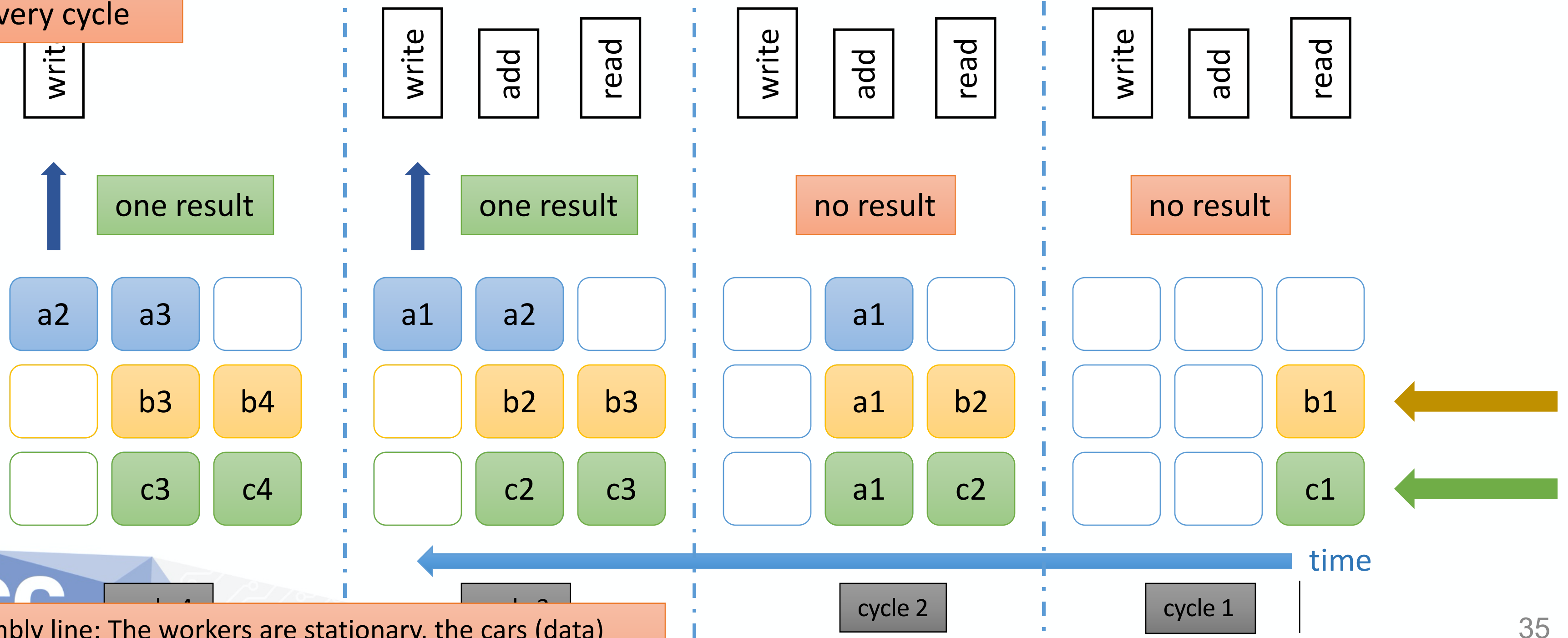
Simple toy model of the implementation of 'add'

'add' takes 3 cycles: 'read', 'add', write'

One result per cycle, once the pipeline is filled
3× performance increase

Operation: 1 cycle
Data transfer: 300 cycles
Ratio: 1/300
Ratio is now worse (no good deed goes unpunished)

After 2 cycles a result is produced every cycle



Moving assembly line: The workers are stationary, the cars (data)

We have just discovered <u>another</u> important hardware features in a CPU: Pipelining

- Pipelining
  - Single operation (add) takes more than a cycle
  - Pipelining (moving assembly line) allows to calculate <u>**one result per cycle**</u> once the pipeline is filled
- Questions for later
  - How do <u>we or the CPU</u> 'organize' pipelining?
  - How does this look in code?

There are 2 fundamental bottlenecks

1. The data supply to the CPU
2. The actual operations

There are 3 major technologies in a CPU
(we can argue about the exact number)

1. Data streams (R=1/100)
2. Pipelining (R=1/300 data supply even more important)
3. 'Improving data movement'

Recap ‘Data Streams’
Bandwidth & Latency

Bandwidth is the amount of data transferred per unit of time
Most convenient units for now based on words and cycles

So far we have talked about data streams and how to improve bandwidth

Recap ‘Data streams’

- 300 cycles to move data between main memory to the CPU
- This is the latency (300 cycles to get the first data)

I’m making this unit up
wpc: word per cycle

Assume **no streams** for the example: a = b + c

- Average bandwidth = 1 ‘word’ per 200 cycles (0.005 wpc)
  - ‘a’, ‘b’, and ‘c’ are 4-byte or 8-byte words
- Let’s pause here: Why exactly is the average bandwidth 1 word per 200 cycles?
- Let’s discuss …

Data latency is the time it takes for the first data to arrive
Most convenient unit is cycles

TACC

Recap ‘Data Streams’

Bandwidth

So far we have talked about data streams and how to improve bandwidth

Recap ‘Data streams’

- 300 cycles to move data between main memory to the CPU

Assume no streams for the example: a = b + c

- Average bandwidth = 1 ‘word’ per 200 cycles (0.005 wpc)
  - ‘a’, ‘b’, and ‘c’ are 4-byte or 8-byte words

wpc: our unit

- Moving ‘b’ and ‘c’ from memory to CPU: 2 words per 300 cycles
- Moving ‘a’ from CPU to memory: 1 word per 300 cycles
- Average: 3 words per 600 cycles = 0.005 wpc

round-trip: 600 cycles

Recap 'Data Streams'
Bandwidth

Same example, but now with streams and with pipelining: a = b + c

- Bandwidth = 3 wpc
- Again let's pause here: Why exactly is the bandwidth 3 words per cycle?

Recap 'Data Streams'
Bandwidth

Bandwidth is the amount of data transferred per unit of time
Most convenient units for now: words and cycles

Same example, but now with streams and with pipelining: a = b + c

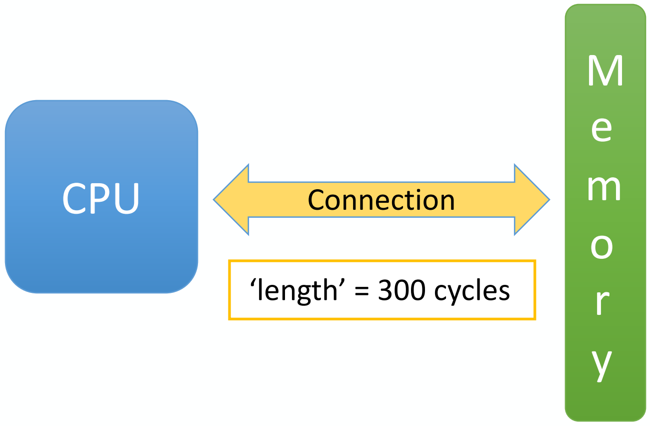
- Bandwidth = 3 wpc
- Every cycle one element of 'b' and 'c' are received, respectively
- Every cycle one element of 'a' is sent back
- In total, 3 words are received and sent every cycle
- Pipelining: one 'add' operation per cycle

TACC

Our first Computer: a(i) = b(i) + c(i)

Let's 'build' a computer and look at the requirements to achieve full performance

- CPU
  - Compute requirements: ...
  - Data movement: ...
- Memory
  - Data movement: ...
- Connection
  - Data movement: ...

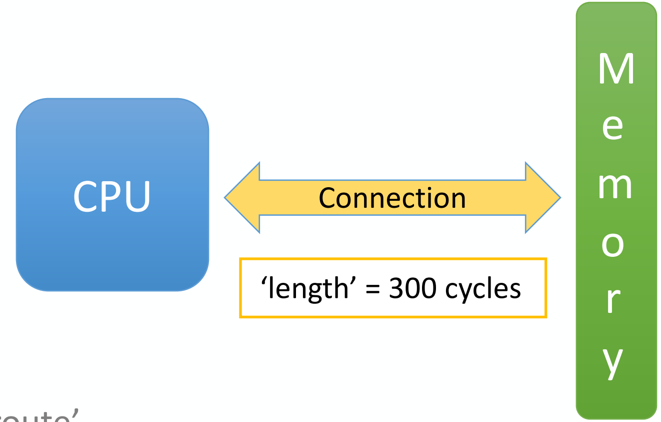


Performance goal: 1 operation (add) per cycle
(we are going to ignore the ramp-up and ramp-down phase for now)

Our first Computer: a(i) = b(i) + c(i)

Let's 'build' a computer and look at the requirements to achieve full performance

- CPU
  - Compute requirements: pipelined, 1 opc
  - Data movement: 3 wpc
- Memory
  - Data movement: 3 wpc
- Connection
  - Data movement: 3 wpc and a total of 900 words 'en route'



Performance goal: 1 operation per cycle
(we are going to ignore the ramp-up and ramp-down phase for now)

But I'm starting to wonder how long these phases may be …

Our <u>first</u> computer and our <u>first</u> source code

```
'Code kernel'
n = 10000
do i=1, n
  a(i) = b(i) + c(i)
enddo
```

Pretty simple code

- Data streams between memory and CPU
- CPU executes the 'add' operation

```
'Full code'

program add
real,dimension(:),allocatable :: a,b,c
n = 10000
allocate(a(n),b(n),c(n))
do i=1, n
  a(i) = b(i) + c(i)
enddo
end program

Hints
- 'real'      = 'float'
- 'allocate' = 'malloc'
- arrays count from 1, unless noted
                              otherwise
- 'do'        = 'for'
```

Technically, the average performance depends on the value of 'n', but we'll make it easy

1. Large 'n': bandwidth = 3 wpc ➔ performance 1 opc (operation per cycle)
2. Small 'n': bandwidth and performance limited (down to fractions of a percent)

TACC

When is 'n' large enough? That depends on the situation

Recap: class period 1

Things we have discussed:

- Primary components of a computer
- Clock tick, clock frequency and its limitations
- Units: word, wpc, opc
- Data streams
- Latency & bandwidth
- Pipelining