

THE EVOLUTION AND ARCHITECTURE OF MODERN COMPUTERS

Professor Ken Birman CS4414/5416 Lecture 2

IDEA MAP FOR TODAY

Goal: Learn just a little about NUMA architectures.

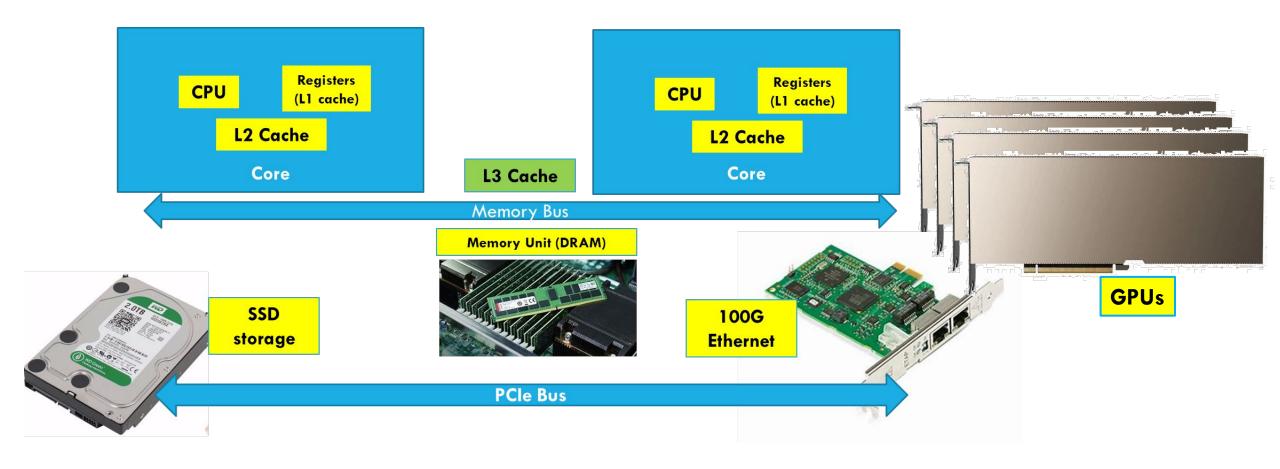
We are not trying to be an architecture course. But we do need to be able to visualize what we are "asking the hardware to do"

Computers are multicore
NUMA machines capable
of many forms of parallelism.
They are extremely complex
and sophisticated.

Individual CPUs don't make this NUMA dimension obvious. The whole idea is that if you don't want to know, you can ignore the presence of parallelism

Compiled languages are translated to machine language.
Understanding this mapping will allow us to make far more effective use of the machine.

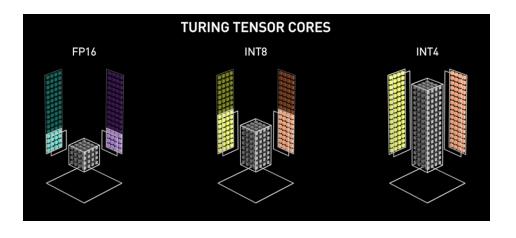
WHAT'S INSIDE? ARCHITECTURE = COMPONENTS OF A COMPUTER + OPERATING SYSTEM



INTEL XENON

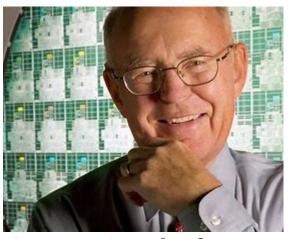
Each core is like a little computer, talking to the others over an on-chip network (the CMS)

NVIDIA TESLA



The GPU has so many cores that a photo of the chip is pointless. Instead they draw graphics like these to help you visualize ways of using hundreds of cores to process a tensor (the "block" in the middle) in parallel!

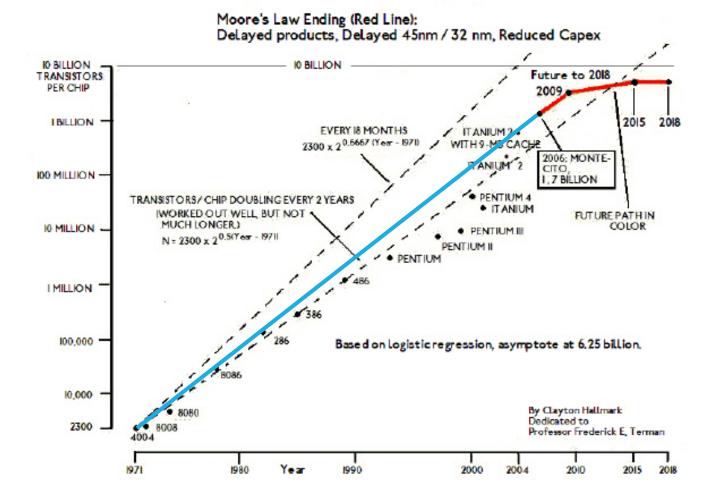
HOW DID WE GET HERE?

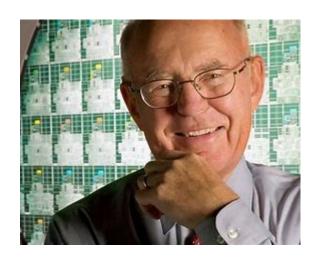


In the early years of computing, we went from machines built from distinct electronic components (earliest generations) to ones built from integrated circuits with everything on one chip.

Quickly, people noticed that each new generation of computer had roughly double the capacity of the previous one and could run roughly twice as fast! Gordon Moore proposed this as a "law".

BUT BY 2006 MOORE'S LAW SEEMED TO BE ENDING





WHAT ENDED MOORE'S LAW?

To run a chip at higher and higher speeds, we use a faster clock rate and keep more of the circuitry busy.



If you overclock your desktop this can happen...

Computing is a form of "work" and work generates heat... as roughly the square of the clock rate.

Chips began to fail. Some would (literally) melt or catch fire!

BUT PARALLELISM SAVED US!

A new generation of computers emerged in which we ran the clocks at a somewhat lower speed (usually around 2 GHz, which corresponds to about 1 billion instructions per second), but had many CPUs in each computer.

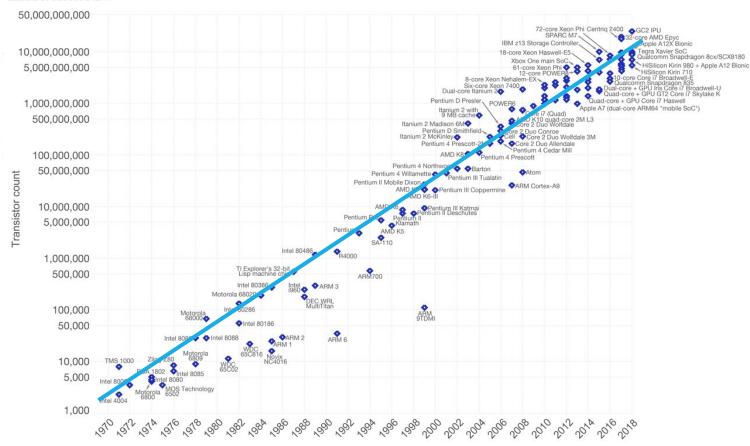
A computer needs to have nearby memory, but applications needed access to "all" the memory. This leads to what we call a "non-uniform memory access behavior": NUMA.

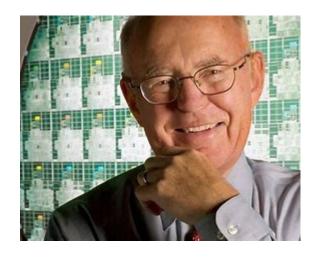
MOORE'S LAW WITH NUMA

Moore's Law – The number of transistors on integrated circuit chips (1971-2018)



Moore's law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years. This advancement is important as other aspects of technological progress – such as processing speed or the price of electronic products – are linked to Moore's law.





AMDAHL'S LAW



Gene on the family farm in Norway

Gene Amdahl was a leading research on parallelism and supercomputing in IBM's HPC division.

He became interested in a basic question. How fast can computations be performed, with infinite parallelism?

A DAY TRIP TO NIAGARA FALLS



You and your friends want to check out Niagara falls.

There are six of you. One option is to rent a plus-sized car (but those big vehicles are slow)



A DAY TRIP TO NIAGARA FALLS

Better plan: You rent three convertible sports cars.

Each holds two people, and these are "insanely fast".







Gene Amdahl's Tractor in Norway

But as you head north, the narrow road has a bottleneck! Until you all pass this slow tractor, the group will have to wait.

HOW AMDAHL THOUGHT ABOUT PARALLELISM

In any computation, we have some parts that are highly parallel, such as scanning our 74,000 different files. Parallelism can speed those up.

But the computation will also have sequential tasks, which could include sequential logic buried in the operating system or the hardware.

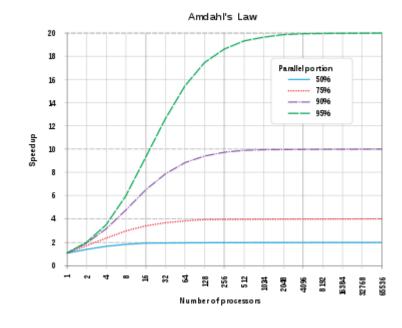
The sequential work will limit the speedup due to parallelism!

HOW AMDAHL EXPRESSED HIS LAW

Suppose that p represents the percentage of the task that can be parallelized and we have N parallel CPUs.

Then
$$\frac{1}{1-p+\frac{p}{N}}$$
 is the maximum speedup

Insight: The parallel tasks could all be done simultaneously, yet we would still have to do the sequential parts step by step.



... MAKING MODERN PERFORMANCE-FOCUSED PROGRAMMING COMPLICATED!

Prior to 2006, a good program

- Used the best algorithm: computational complexity, elegance
- > Implemented it in a language like C++ that offers efficiency
- Ran on one machine

... MAKING MODERN PERFORMANCE-FOCUSED PROGRAMMING COMPLICATED!

Today, a good program

- Used the best algorithm: computational complexity, elegance
- > Implemented it in a language like C++ that offers efficiency
- Uses many threads and perhaps runs on many machines
- Shows smart awareness of hardware properties that can shape performance, forcing us to take implicit measures to ensure that the compiler and optimizer will produce great code

... MAKING MODERN PERFORMANCE-FOCUSED PROGRAMMING COMPLICATED!

Today, a good program

- > Used the best algorithm, computational complexity, elegance
- Vibe coding is often easier but won't win performance prizes!
 ++ that offers efficiency
- Uses man won't win performance prizes:

 n many machines
- Shows smart awareness of hardware properties that can shape performance, forcing us to take implicit measures to ensure that the compiler and optimizer will produce great code

THE HARDWARE SHAPES THE APPLICATION DESIGN PROCESS

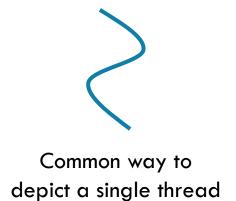


We need to ask how a NUMA architecture impacts our designs.

If not all variables are equally fast to access, how can we "code" to achieve the fastest solution?

And how do we keep all of this hardware "optimally busy"?

HOW A SINGLE THREAD COMPUTES



In CS4414 we think of each computation in terms of a "thread"

A thread owns a pointer into the program instructions. The CPU loads the instruction that the "PC" points to, fetches any operands from memory, does the action, saves the results back to memory.

Then the PC is incremented to point to the next instruction

ASSEMBLY/MACHINE CODE VIEW

Registers Data Condition Codes Instructions Memory Code Data Stack

Programmer-Visible State

- PC: Program counter
 - Address of next instruction
 - Called "RIP" (x86-64)
- Register file
 - Heavily used program data
- Condition codes
 - Store status information about most recent arithmetic or logical operation
 - Used for conditional branching

Memory

- ➤ Byte addressable array
- Code and user data
- Stack to support procedures

Puzzle:

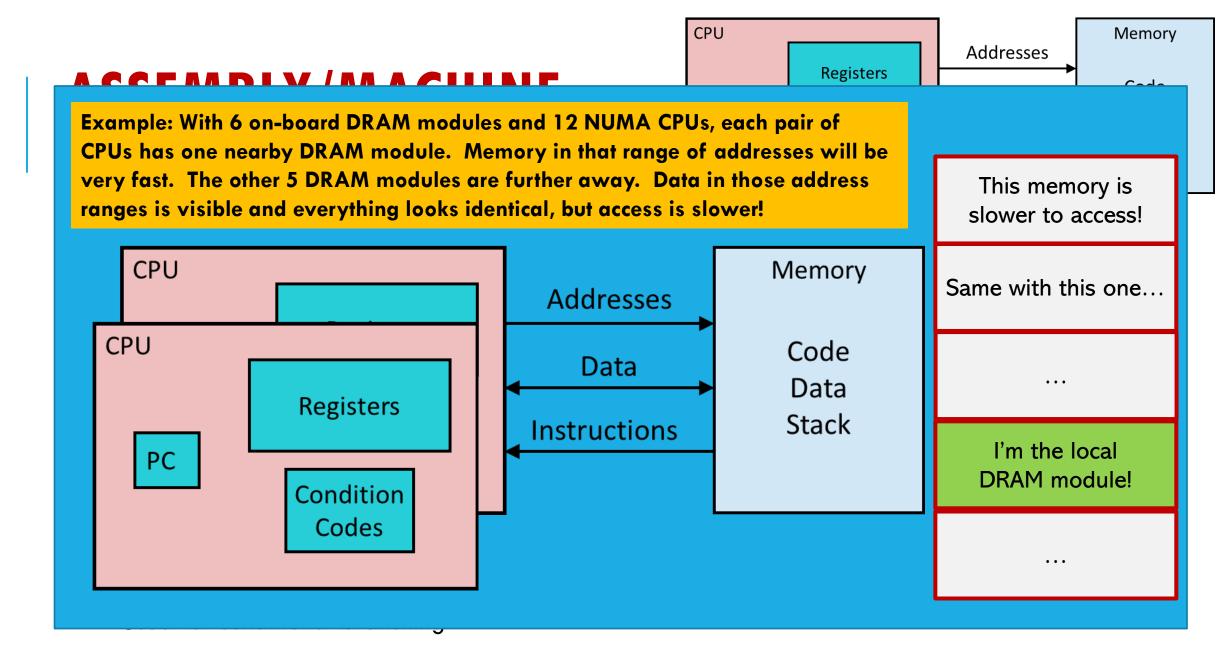
- On a NUMA machine, a CPU is near a fast memory but can access all memory.
- How does this impact software design?

NUMA OFFERS THE ILLUSION THAT NOTHING HAS CHANGED!

You can write code exactly the way you did before.

The only issue is... it might run very slowly in ways caused by the hardware trying to pretend to be an old single core system.

Understanding the causes enables us to write the same code in slightly smarter ways to avoid these overheads.



C++ ADDRESSING MODES AND NUMA ACCESS COSTS

Addressing Mode	Estimated Access Cost (ns)
Register or D-cache	0.3 - 1 ns (1-3 cycles)
Local NUMA memory	30 - 60 ns (100-200 cycles)
Remote NUMA memory	90 - 150 ns (300-500 cycles)
Pointer dereference	Varies with memory location

Data structures often cost more ("pointer chasing")

Data Structure	Estimated Access Cost (ns)
Array of ints	0.3 - 1 ns (in cache), 30-150 ns (memory)
Array of 64-bit floats	0.3 - 1 ns (in cache), 30-150 ns (memory)
std::vector	Similar to array, with bounds checking overhead
std::list	100 - 300 ns (pointer chasing)
2D matrix	Depends on layout and access pattern

HOW DOES THIS CHANGE YOUR CODING?

Knowing these implicit costs, you need to design your code with the cost model in mind.

Decisions about data structures, how to perform computations on larger objects such as matrices, and where objects live in memory start to reshape the design process!

Some people love "vibe coding"... but MLs mess this type of design up. They always pick code styles they were trained to favor, without attention to the properties of today's common NUMA servers.

.... IN EFFECT

NUMA servers pretend to be old fashioned single core servers

This means old coding styles work, even if they perform poorly.

Vibe coding (ML assisted) reinforces the issue because the MLs are trained on coding styles and examples that were the ultimate in elegance on a <u>prior generation</u> of hardware!

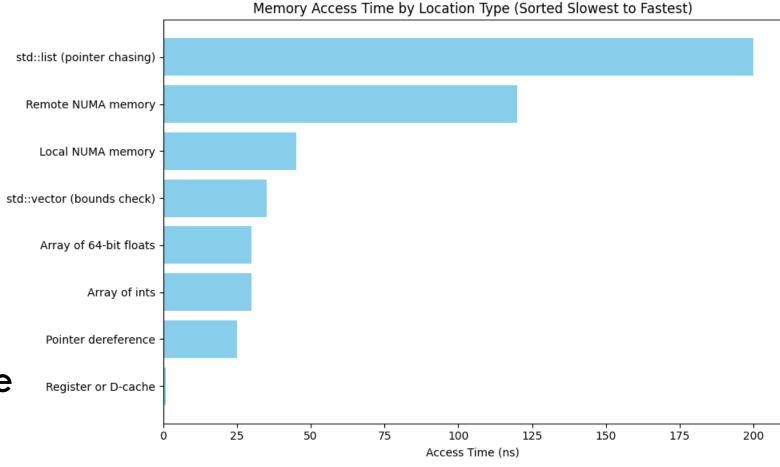
MEMORY LAYOUT AND ACCESS TIME MATTERS TOO. CONSIDER 2D MATRICES

- Row-major layout stores rows contiguously in memory
- Column-major layout stores columns contiguously
- Cache locality is better when accessing data with small "strides"
- Stride access patterns affect cache performance:
 - Row-wise access in row-major layout is fast if the index is fast to compute, like an integer.
 - Column-wise access in row-major layout causes cache misses

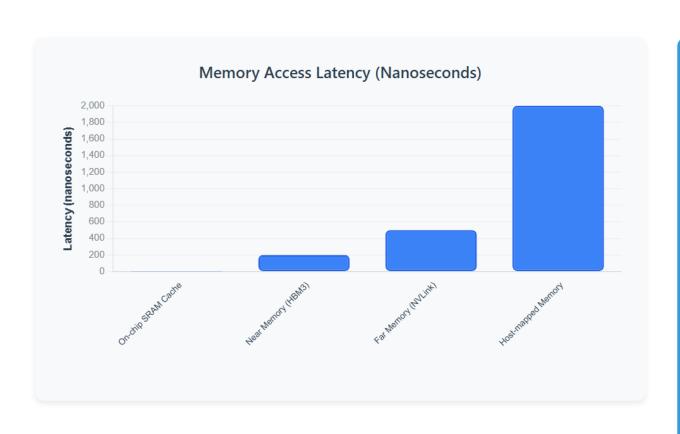
MEMORY ACCESS TIME BY LOCATION TYPE

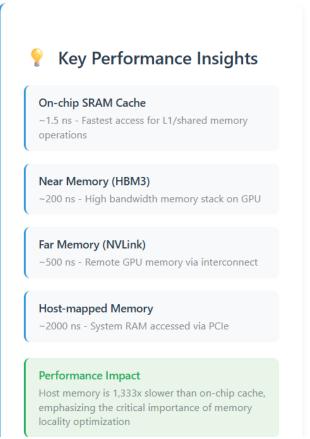
Core message?

Exactly the same standard code might run far faster/slower depending on choices being made



NVIDIA H100 GPU MEMORY ACCESS TIME BY LOCATION TYPE





WOULDN'T CACHING CONCEAL THIS? OR PREFETCHING?

Both techniques help a lot! But the hardware and runtime environments aren't always able to guess at what should be prefetched and what should be cached.

Also, the cache for a server core or for a GPU is far smaller than the nearby memory, which this in turn is smaller than the aggregated memory of a cluster of GPUs living under one host. Big objects might not fit in the local memory, or in the local cache.

So caching might not be able to hide costs for big LLM/LRM models!

TO MAXIMIZE PERFORMANCE WE NEED TO CONTROL THESE FACTORS

Yet they only sometimes involve explicit coding decisions.

- An **explicit** choice occurs when you pick a std::vector and not a simple array. If you pick std::list you are "forcing" a slow option
- You can annotate an inline variable with register to **explicitly** request fast memory (but with -O3 the compiler usually makes good choices)
- An **implicit** choice occurs when an active thread allocates an object (memory will be in a nearby heap), or if an object is in global memory (it will probably not be in nearby memory). A C++ **const** is fastest of all.

LINUX AND THE HARDWARE: TWO SIDES OF THE SYSTEM <u>ARCHITECTURE</u>

We will be learning about the modern computer hardware, not so much from an internals perspective, but as users.

Linux lets you design applications that correspond closely to the hardware. But then we need a programming language that lets us talk directly to the operating system and the hardware.

WHAT ABOUT THE CHOICE OF PROGRAMMING LANGUAGE?

The idea behind programming is very universal:

- We code in a higher level way
- The programming language maps our code to instructions, or perhaps even performs the operations itself, in a runtime

This makes us more productive and hopefully, the code is even better than we could have created by hand.

CHOICE OF PROGRAMMING LANGUAGE...

Most people are familiar with Java and Python.

Java has lots of data types (and lots of fancy syntax!), generics, threads, other elaborate language features. It compiles to a mix of machine code and programming language runtime logic.

Python automates much more: No need to fuss with data types, easy to create arrays and transform all the objects with just one step. And because Python threads hold a global interpreter lock while active, it doesn't have true parallelism.

CHOICE OF PROGRAMMING LANGUAGE...

```
Which is better?

    Java
    Python

What about C++? Rust? Objective C?
O'Caml? Julia?
```

arrays and transform all the objects with just one step.

CONSIDERATIONS PEOPLE OFTEN CITE

Expressivity and Efficiency: Can I code my solution elegantly and easily? Will my solution perform well?

Correctness: If I end up with buggy code, I'll waste time (and my boss won't be happy). A language should facilitate correctness.

Productivity: A language is just a tool. The easier it is to do the job (which is to solve some concrete problem), the better!

DRILL-DOWN CONSIDERATIONS

```
Which performs better?
1) Java2) Python3) ... something else?
                                                       pay
what you use. So performance = φφφ.
```

WHAT CAN MAKE PYTHON AND JAVA EXPENSIVE?

Python: Interpreted

Compiles to a high-level representation that enables an "interpretive" execution model.

In fact, Python is like a "general machine" controlled by your code: Python <u>itself</u> runs on the hardware. Then your code runs on Python!

Gradual typing: Python is very laissez-faire and can't optimize for specific data types.

Java: Runtime overheads

Compiles (twice: to byte code, then via JIT) but rarely exploits full power of hardware. Limited optimizations, parallelism

Dynamic types and polymorphism are costly.

Everything is an object, causing huge need for copying and garbage collection.

It feels as if your programs run inside layers and layers of "black boxes"

DOES C++ AVOID THESE PITFALLS?

C++ objects are a compile-time feature. At runtime, all the type-related work is finished: no runtime dynamics.

The compiler "inline expands" and infers types, which makes coding easier. Then it optimizes heavily. You help it.

Computers execute billions of instructions per second, yet we can write code that will minimize the instructions and shape the choices.

Parallelism is easy, and the compiler automatically leverages modern hardware features to ensure that you will have highly efficient code.

LET'S DRILL DOWN ON SPEED



For some situations, C++ can be <u>thousands of times faster</u> than Python or Java, on a single machine!

- > Typically, these are cases where the application has a lot of parallelism that the program needs to exploit.
- For example, identifying animals in a photo entails a lot of steps that involve pixel-by-pixel analysis of the image
- > But in fact, we can get substantial speedups just scanning large numbers of big files... hence our word-count demo

PARALLELISM

... in fact, it is very hard to exploit parallelism in a single Python program.

This is because the Python model is "single threaded". Even so, <u>PyTorch</u> is highly parallel, because it leverages GPUs.

Java does allow parallelism, via "parallel threads". But high performance coding in Java requires a lot of skill.

LET'S DRILL DOWN ON SPEED



We said that Python is slowest, Java is pretty good, but C++ can beat both. C++ knocks the socks off Java for parallel tasks.

What would be a good way to "see that in action"?

A small example: "word count" in Python, Java and C++

WORD COUNT TASK

Basically, we take our input files and "parse" them into words. All three languages have prebuilt library methods for this. Discard non-words (things like punctuation marks).

Keep a sorted list of words. As we see a word, we look it up and increment a count for that word (adding it if needed).

At the end, print out a nicely formatted table of the words/counts in descending order by count, alphabetic order for ties

THE PARTICIPANTS

Ken, back when he was kind of new to C++ in 2020

Sagar, our head PhD TA in the early days, who was a hard-code C++ coder, spent two summers as a Microsoft employee.

Lucy, undergraduate coding superstar

[Added in 2025]: Andrew Myers, world champion Java programmer (using a version with "virtual threads" that wasn't available to Lucy back in 2020)

THE SCOREBOARD



```
#1-A: Ken's C++ Faster, but more complex...
                                                         #3-A Lucy's Java version (no threads)
    real 4.645s
                                                              real 1m49.373s
    user 14.779s
                                                              user 3m16.950s
                                                              sys 8.742s
         1.983s
    SYS
                                                         #3-B Andrew's Java version (virtual threads)
#1-B (Sagar's code, shorter & better use of C++...)
    real 8.200s
                                                               real 5.5s [but on a slower computer]
    user 49.295s
                                                               (user and sys time not reported)
          2.145s
                                                         What if Andrew had used the same server? His Java is
    SYS
                                                         probably as fast as Ken's C++... a likely tie!
#2 Lucy's Python version
                                                         #4: Pure Linux (buggy sort order)
         1m30.857s
                                                              real 2m38.965s
          1m30.276s
                                                              user 2m43.999s
    user
                                                              sys 27.084s
         0.572s
    SYS
                                                                                         CORNELL CS4414/5416 - FALL 2025
This was only 19 lines of code!
```



FOULS

THE SCOPE

C++ version was 34x faster than Linux, 20x faster than unthreaded Java or Python

```
#1-A: Ken's C++ Faster, but more complex...
                                                          #3 Lucy's Java version (no "true" threads)
    real (4.645s
                                                               real 1m49.373s
    user 14.779s
                                                                    3m16.950s
                                                               sys 8.742s
         1.983s
    SYS
#1-B (Sagar's code, shorter & better use of C++...)
                                                          #3-B Andrew's Java version (virtual threads)
    real 8.200s
                                                               real 5.5s [but on a slower computer]
    user 49.295s
                                                               (user and sys time not reported)
          2.145s
                                                          What if Andrew had used the same server? His Java is
    SYS
                                                          probably as fast as Ken's C++... a likely tie!
#2 Lucy's Python version
                                                          #4: Pure Linux (buggy sort order)
         1m30.857s
                                                               real 2m38.965s
    real
          1m30.276s
                                                              user 2m43.999s
    user
         0.572s
                                                                    27.084s
    SYS
                                                               sys
                                                                                          CORNELL CS4414/5416 - FALL 2025
This was only 19 lines of code!
```



Notice that the user time is 3x larger than the real time.

Puzzle: how can this be true?



```
#1-A: Ken's C++ Faster, but more complex...
                                                          #3 Lucy's Java version (no "true" threads)
    real 4.645s
                                                                     1m49.373s
          14.779s
                                                                     3m16.950s
    user
                                                               sys 8.742s
          1.983s
    SYS
#1-B (Sagar's code, shorter & better use of C++...)
                                                          #3-B Andrew's Java version (virtual threads)
    real
         8.200s
                                                               real 5.5s [but on a slower computer]
          49.295s
                                                               (user and sys time not reported)
    user
          2.145s
                                                          What if Andrew had used the same server? His Java is
    SYS
                                                          probably as fast as Ken's C++... a likely tie!
#2 Lucy's Python version
                                                          #4: Pure Linux (buggy sort order)
          1m30.857s
                                                                     2m38.965s
                                                               real
          1m30.276s
                                                                     2m43,999s
    user
          0.572s
                                                                     27.084s
    SYS
                                                               sys
                                                                                          CORNELL CS4414/5416 - FALL 2025
This was only 19 lines of code!
```

HOW TO DO 14.7779s OF COMPUTING IN 4.645s?



A 3-horsepower system

Concurrency!

... if a process is using more than one thread it can harness more than one CPU at the same time.

With 3 CPUs running continuously at full speed, it can do 3x more work than the elapsed wall-clock time!

QUICK DIVE INTO WORD COUNT IN C++

We'll learn all of this over a few weeks

But today, we already might have a glimpse.

Code and data set is linked to the syllabus page for lecture 2

EXAMPLE: HELLO WORLD IN C++

```
// My first C++ program

#include<iostream>

int main() {
    std::cout << "Hello World" << std::endl;
    return 0;
}</pre>
```

First you'll create a file, hello.cpp

Next, it must be compiled, for example:

g++ -std=c++20 hello.cpp -o hello

... and finally, launched: ./hello

Hello World

We will be doing these steps from within Visual Studio Code

This "IDE" includes support for editing, compiling, debugging and executing your programs.

EXAMPLE: WORD COUNT IN C++

This is the "core" of the counting logic:

EXAMPLE: WORD COUNT IN C++

... and here is the core of the sorting logic:

```
struct SortOrder: public std::binary_function<std::pair<int, std::string>, std::pair<int, std::string>, bool>
  bool operator()(const std::pair<int, std::string>& lhs, const std::pair<int, std::string>& rhs) const
     return lhs.first > rhs.first | | (lhs.first == rhs.first && lhs.second < rhs.second);
using SO = std::map<std::pair<int, std::string>, int, SortOrder>;
SO sorted totals;
for(auto wc: totals)
     std::pair<int,std::string> new_pair(wc.second, wc.first);
     sorted_totals[new_pair] = wc.second;
```

EXAMPLE: WORD COUNT IN C++

Same logic but expressed using the c++23 decltype feature

MY CODE VERSUS SAGAR'S

My code understood that in files, data is just a long "vector" of characters – bytes – with some '\n' characters (end of line).

My word-count kept the data in that form and only created std::string objects at the last moment, to increment the count:

"wptr" is a pointer directly to the bytes in the input buffer

```
inline void found(int& tn, char*& wptr)
{
    sub_count[tn][std::string(wptr)]++;
}
```

A CHUNK OF LINUX SOURCE CODE

Notice: this has text (words) but also lots of other stuff, like spaces and tabs, special chars like () $\{\}$;/ $_{*}$ etc.

End of line is a special ascii char, '\n' (code == 0x12).

```
#ifdef CONFIG DMA PERNUMA CMA
void init dma pernuma cma reserve(void)
        int nid;
        if (!pernuma size bytes)
                return;
        for each online node(nid) {
                int ret:
                char name[CMA MAX NAME];
                struct cma **cma = &dma contiguous pernuma area[nid];
                snprintf(name, sizeof(name), "pernuma%d", nid);
                ret = cma declare contiguous nid(0, pernuma size bytes, 0, 0,
                                                 0, false, name, cma, nid);
                if (ret) {
                        pr warn("%s: reservation failed: err %d, node %d", func ,
                                ret, nid);
                        continue;
                pr_debug("%s: reserved %llu MiB on node %d\n", __func__,
                        (unsigned long long)pernuma size bytes / SZ 1M, nid);
#endif
```

VISUALIZATION OF MY WORD COUNT RUNNING

Read data into memory from disk file

Some file with Linux source code, like .../kernel/dma/contiguous.c

int ret;\nchar name[CMA_MAX_NAME];\nstruct cma **cma = &dma_contiguous_pernuma_area[nid];\nsnprintf(name, sizeof(name), "pernuma%d", nid);\nret = \n cma_declare_contiguous_nid(0, pernuma_size_bytes, 0, 0,\n 0, false, name, cma, nid);\n if (ret) {\n pr_warn ("%s: reservation failed: err %d, node %d", __func__,\n ret, nid);\n continue;\n }\n pr_debug("%s: reserved %llu MiB on node %d\n",\n __func__,\n (unsigned long long)pernuma_size_

Memory buffer

Ken's word-count process, when running

WHAT DO WE MEAN BY "READ DATA INTO MEMORY?"

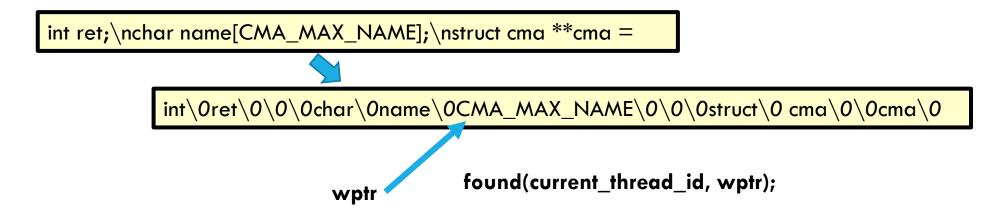
In my program, some space gets allocated – set aside – in the address space as a place for file data to be held.

The program opened a source file and told Linux to copy 4096 bytes (one block) into that buffer area.

The text that you saw in that screenshot was stored there as a series of ascii bytes, a code that uses values 0..128

HOW MY CODE ACTUALLY WORKED

Change all "white space" to \0 (byte containing 0). Now each word is a null-terminated char* vector (a "c-string")



Converted from a c-string to std::string in found:

sub_count[tn][std::string(word)]++;

WHEN I FIRST CODED MY SOLUTION, MY PROGRAM WAS VERY SHORT, BUT RATHER SLOW.

I added parallel threads – which complicated the solution but helped a lot. Then because file opening was slow, I added a thread to "preopen" files before they were needed.

The C++ library for file opening and reading files was a bottleneck, so I switched to calling Linux file open and Linux file read, directly. This gave an additional speedup

WHAT MADE SAGAR'S VERSION SLOWER?

If you look at his code, you'll find that it converts the whole file into std::string objects, line by line

Then it splits lines into substrings using a "splitter" method. Each chunk will be a std::string. But many won't be "words"

If the substring matching the rule for a word, Sagar's code uses a map like Ken's code and increments the count.

HOW CAN WE "ANTICIPATE" THE COSTS OF TOO MANY USES OF STD::STRING?

We know that a file is basically a long vector of bytes.

A text file holds ascii chars with '\n' for newline. A c-string is a region holding chars, ending with '\0'. Ken worked from this.

In contrast, a std::string is an object. At a minimum it has a string length and its own copy of the c-string holding the string data. It must be constructed and freed. That has to be costly.

WHAT MADE SAGAR'S CODE SLOWER?

This means Sagar was creating perhaps 5-10x more std::string objects. At scale, with 50,000 files and millions of lines to scan, he does a lot of object creation, splitting and deletion, copying, garbage collection. Ken's code "skipped" 95% of that work!

... So Ken's code was way faster! Yet Sagar's was closer to being pure C++. Ken's mixed C++ with C

```
int ret;\nchar name[CMA_MAX_NAME];\nstruct cma **cma = &dma_contiguous_pernuma_area[nid];\nsnprintf(name, sizeof(name), "pernuma%d", nid);\nret = \n cma_declare_contiguous_nid(0, pernuma_size_bytes, 0, 0, \n 0, false, name, cma, nid);\n if (ret) {\n pr_warn ("%s: reservation failed: err %d, node %d", __func__,\n ret, nid);\n continue;\n }\n pr_debug("%s: reserved %llu MiB on node %d\n",\n _func__,\n (unsigned long long)pernuma_size_
```





Total compute "load" was actually a lot lower for Ken's C++ program. Hence... less energy consumed

```
#1-A: Ken's C++ Faste

✓a version (no "true" threads)

                                                                      1m49.373s
         4.645s
    real
                                                               real
    user 14.779s
                                                               user 3m16.950s
                                                               sys \ 8.742s
          1.983s
    SYS
#1-B (Sagar's code, shorter & better use of C++...)
                                                          #3-B Andrew's Java version (virtual threads)
                                                               real 5.5s [but on a slower computer]
          8.200s
    real
    user 49.295s
                                                                (user and sys time not reported)
                                                          What if Andrew had used the same server? His Java is
         2.145s
    sys
                                                          probably as fast as Ken's C++... a likely tie!
#2 Lucy's Python version
                                                          #4: Pure Linux (buggy sort order)
                                                               real 2m38.965s
    real 1 m 30.857 s
    user 1m30.276s
                                                               user 2m43.999s
          0.5/2s
                                                                     27.084s
    SYS
                                                               sys
                                                                                          CORNELL CS4414/5416 - FALL 2025
                                                                                                           62
This was only 19 lines of code!
```

BUT MUCH MORE IS REALLY GOING ON!

The Linux file system is involved when we scan folders to find the files we plan to run WC on, and it reads the data for us.

- How good a job is it doing? Is it prefetching? Caching?
- Should we be directly mapping files into memory?
- Could our threads be somehow contending for the Linux file system layer, and slowing things down?

A TENNIS ANALOGY



When you hit a tennis ball, you swing the racket...

- But gravity and the dynamics of spin shape the trajectory
- Your opponent is in motion. Anticipating their position/angle decides if your shot will be easy for them to return or hard
- > The layout of the tennis court itself sets the real constraints

We talked about implicit versus explicit decisions. Does tennis have this too?

A MODERN COMPUTER IS EVEN MORE LIKE A SWISS WATCH!



Tennis is a good analogy for pairs of elements that work concurrently yet influence one another.

But a single NUMA computer (even with just a single LLM or LRM task on it) is doing dozens of things at once, and has *many* of these relationships

Getting the best performance? It is like "designing clockwork"!

EXAMPLES OF IMPLICIT PROGRAMMING

Using a separate thread to open files well before they are needed for scanning.

Interacting with the file system efficiently, by reading 4KB chunks at a time or mapping the file, and accessing it sequentially from a single thread.

Allocating separate std::map objects, each created by a thread that will **exclusively** use it (ensures memory will be local)

TO BE A GREAT PROGRAMMER...

You need to have all these concepts in mind as you work. And you need to understand performance both of the hardware and the O/S

You **explicitly** code lots of things, but the way you write your code **implicitly** avoids lock contention, allocates memory in ways that should maximize locality, accesses files in ways that promote file prefetch and caching, etc.

A great programmer is always conscious of both kinds of choices

There was a big idea about computer architectures, the architecture of the OS and the way that programs running on the computer interact with both. <u>Summarize that big idea in your own words.</u>

Java and Python and C++ are all reasonable languages. Why do many companies favor C++ for "systems programming"?

List all the things happening concurrently when word count is running.

We saw some big ideas, like using threads, but to maximize the speed of those threads we also saw some "small" ideas.

Why did it matter whether or not threads share a single counter data structure? Shouldn't it be faster to share just one and not have to merge lots of them?

Why did the way that text is represented in files on disk matter in the WC application?

How does understanding how the file system component of Linux influence the way a word count program is implemented?

Why might it matter which part of memory a particular variable was placed in by the compiler?

Is this something we can control in Python? Java? C++? (Hint: this question might require a bit of dialog with your Al copilot!)

MORE SELF-TEST

You are given code that you can use, but not modify, like libraries of ML kernels (computations). To create a new ML you call one, then feed the output from to another as its input, etc.

Using a profiler, you discover that the code uses many threads and many GPUs, and has a memory bottleneck. While this program runs, 96% of the compute capacity is idle. The NUMA memory busses of the host and GPUs are 100% saturated.

What options would you have that could possibly help?

MORE SELF-TEST

All of this is happening on one server with lots of memory and many GPUs

You are given code that you can use, but not modify, like libraries of ML kernels (computations). To create a new ML you call one, then feed the output from to another as its input, etc.

Using a profiler, you discover that the code uses many threads and many GPUs, and has a memory bottleneck. While this program runs, 96% of the compute capacity is idle. The NUMA memory busses of the host and GPUs are 100% saturated.

What options would you have that could possibly help?

MOST IMPORTANT ML KERNEL?

... it turns out to be matrix multiply. Best algorithms are surprisingly fast. For example, with square matrices, rather than $O(n^3)$ the best known algorithm runs in $O(n^{2.371339})$...

But today we've seen other considerations: where data actually is located in memory, GPU versus host compute, delay to launch a GPU computation, integer versus float (and size in bits)...

SELF-TEST

Suppose you need to convince a teammate or manager that these things actually matter.

Could you find an off-the-shelf ML library method and call it in C++ in two ways, such that it performs 100x or even 1000x better depending only on where the input object is located?

Extra self-test: Try doing it!

SAME SELF-TEST

At a group meeting, someone is skeptical that professional LLM or LRM solutions really could be running slowly on the company hardware

Separate from writing your own code and "setting it up" to perform badly, how would you convince that teammate that there really could be a large opportunity, worth pursuing?

NOTICE THAT OUR SLIDES DIDN'T TALK ABOUT THESE EXACT CASES!!

In CS4414 and CS5416 we try to show you how to think about problems, but you can't learn the material by memorizing slides

In fact most people find that the only way to really deeply learn it is to try some of these things out.

Be sure to compile with the -O3 flag when playing with C++. This tells the compiler to do its very best. Often -O3 by itself can give a 2x performance difference... and sometimes, far more.

ARE THERE "RIGHT ANSWERS"?

Some people are incredible at speeding things up... Our TAs are those kinds of people. It involves talent.

The ideas they consider are the same you probably came up with! But they also know how easy or hard it would be to actually try those ideas out. Start with easier ones!

When you have a flash of insight, always ask: can I validate this? Can I use my insight to do something that would pay off? How much work will be required?

OPTIONAL EXTRA READINGS

Code for the word count programs is available here.

If you find this topic exciting, you might enjoy reading about

- Research done by Chris da Sa on these topics
- How DeepSeek actually achieved such good speedups for training (or some people suspect, "fine tuning") LLMs
- How LPU accelerators from Grok speed up LLMs
- Or <u>this cool paper</u> Shouxu Lin and Alicia Yang found, about running LLMs on commodity laptops and desktops.