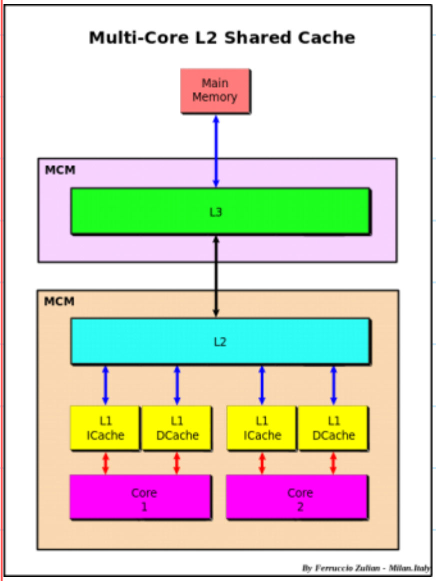
Memory models in a computer

(simplified for mathematicians)

Friday, August 23, 2024 4:51 PM

- What scientists must know about hardware to write fast code, see https://viralinstruction.com/posts/hardware/
- https://book.sciml.ai/lectures/#optional_extra_resources

Memory Hierarchy



Slow (but large)

transferring memory is a major slowdown

Fast (but small)

Transferring memory to another computer or to a GPU is also slow

Cache misses

Transferring data is slow, has both a latency and bandwidth restrictions.

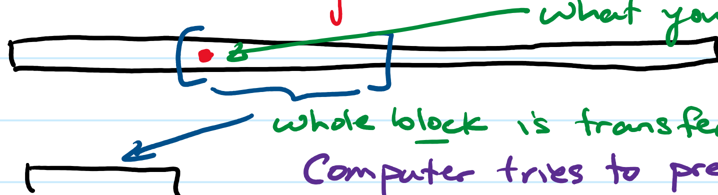
i.e. cost to move "x" bytes is $b \cdot x + \ell$

in time or clock cycles

So... due to latency, we don't like to transfer small amounts of data (since it's wasteful)

Instead, transfer a large block

RAM



what you need in cache

whole block is transferred.

Cache

Computer tries to predict what memory it will use in the future.

Memory (page 2)

Wednesday, September 11, 2024 7:16 AM

whenever you need to move memory into cache (because it wasn't there already) it's a cache miss and really slows things down.

Effect

Ex: Storing a large matrix $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$

What's this look like in memory?

Python does row-major order

a b c d

location n, location $n+\Delta$, $n+2\Delta$, $n+3\Delta$

(Δ = size of float)

Matlab/fortran does column-major order $[a\ c\ b\ d]$

Let's say our cache can fit 2 numbers at once.

We compute $\|A\|_F^2 = \underbrace{\sum_{i=1}^{2} \sum_{j=1}^{2}}_{\text{we can choose the order}} A_{ij}^2$

i = row

j = column

Say we're in Python

$\sum_i \left(\sum_j A_{ij}^2 \right)$

$i=1$	$j=1$	a	cache miss (Request "a", get [a,b])
	$j=2$	b	
$i=2$	$j=1$	c	cache miss (Request "c", get [c,d])
	$j=2$	d	

2 cache misses... good!

vs.

$\sum_j \left(\sum_i A_{ij}^2 \right)$

$j=1$	$i=1$	a	cache miss	get [a,b]
	$i=2$	c	cache miss	get [c,d]
$j=2$	$i=1$	b	cache miss	get [b,c]
	$i=2$	d	cache miss	get [d, ?]

4 cache misses, bad!

Even more important (and complicated) for sparse matrices

Related to idea of blocked algorithms (= works on contiguous chunks of data) and exploiting fast BLAS for vector and matrix operations (multiplies)

BLAS: Intel MKL, very optimized, already parallelized

and to cache oblivious algorithms (Frigo et al. '99) where you don't need to explicitly know cache size in order to get efficient performance

Memory and GPUs (page 3)

Wednesday, September 11, 2024 9:33 AM

In "main memory" (aka RAM), one physical location split into two (virtual) types: stack (includes call stack, and small amount of room for variables)

heap = the rest, used for large data ex. in C, any malloc calloc calls

SIMD = Single Instruction Multiple Data

In most modern CPU processors (mmx, sse, sse2, avx ...)

If you want to apply $\underbrace{f(x)=x^2}_{\text{or anything else simple enough}}$ to multiple data $\{x_i\}_{i=1}^{256}$

a SIMD CPU does it efficiently, much less than 256× cost of doing it once. It's a hardware thing, less overhead

They excel at matrix multiplies... the core of much Science and ML

GPU = Graphical Processing Unit, aka video/graphics card

Like CPU but can only do simpler things, slower clock speed, but huge number (eg. 1000's) of parallel cores.

Like extreme SIMD, though at a higher level

- NVIDIA's CUDA is dominant driver/language

aa100 amc on CURC Alpine are NVIDIA A100's

Not all GPUs (even NVIDIA ones) are CUDA-compatible

gaming GPUs often don't have much memory, or only support low-precision

- OpenCL and AMD's ROCm and Mac MPS (Metal Performance Shader) are alternatives though not as widespread

ami100 on CURC Alpine are AMD MI100's

- Data transfer from CPU to GPU is costly
- GPUs have small RAM compared to GPUs
- GPUs usually work in single (32 bit) floating point precision, or less

(vs. CPU/NumPy standard is double precision, 64 bit)