Lecture 22: Memory

CS4787/5777 — Principles of Large-Scale Machine Learning Systems

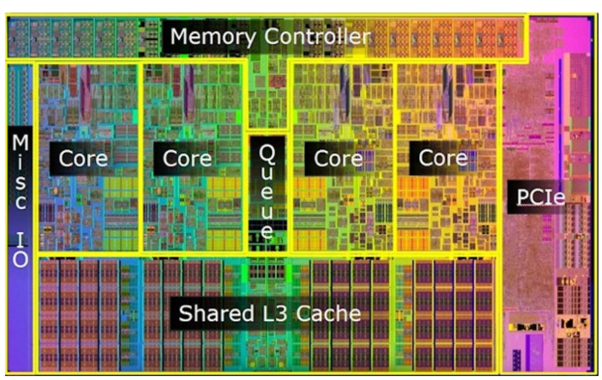
- Last time we talked about parallel computing in machine learning, which allows us to take advantage of the parallel capabilities of our hardware to substantially speed up training and inference.
- This is an instance of the general principle: **Use algorithms that fit your hardware, and use hardware that fits your algorithms**.

- But compute is only half the story of making algorithms that fit the hardware.
- How data is stored and accessed can be just as important as how it is processed.
- This is especially the case for machine learning tasks, which often run on very large datasets that can push the limits of the memory subsystem of the hardware.

Today, we'll be talking about how memory affects the performance of the machine learning pipeline.

How do modern CPUs handle memory?

CPUs have a deep **cache hierarchy**. In fact, many CPUs are mostly cache by area.



The motivation for this was the ever-increasing gap between the speed at which the arithmetic units on the CPU could execute instructions and the time it took to read/write data to system memory.

Without some faster cache to temporarily store data, the performance of the CPU would be bottlenecked by the cost of reading and/or writing to RAM after every instruction.

A simplified view of memory on a CPU

This is what the "shared memory" programming model sees.

But CPUs also have caches

Caches are small and fast memories that are located physically on the CPU chip, and which mirror data stored in RAM so that it can be accessed more quickly by the CPU.

The usual setup of memory on a CPU

- a fast L1 cache (typically about 32KB) on each core
- a somewhat slower, but larger L2 cache (e.g. 256 KB) on each core
- an even slower and even larger L3 cache (e.g. 2 MB/core) shared among cores
- DRAM — off-chip memory
- Persistent storage — a hard disk or flash drive

A model of a multi-socket computer

Multiple CPU chips on the same motherboard communicate with each other through physical connections on the motherboard.

The full view across multiple machines

Multiple CPU chips on the same motherboard communicate with each other through physical connections on the motherboard.

One important thing to notice here:

As we zoom out, much more of this diagram is "memory" boxes than compute boxes.

Hand-wavy consequence: as we scale up, the effect of memory becomes more and more important.

Another important take-away:

Memory has a hierarchical structure

- Memories lower in the hierarchy are faster, but smaller
- Memories higher in the hierarchy are larger, but slower, and are often shared among many compute units

Two ways to measure performance of a part of the memory hierarchy.

- **Latency**: how much time does it take to access data at a new address in memory?
- **Throughput** (a.k.a. bandwidth): how much data total can we access in a given length of time?

We saw these metrics earlier when evaluating the effect of parallelism.

Ideally, we'd like all of our memory accesses to go to the fast L1 cache, since it has high throughput and low latency.

What prevents this from happening in a practical program?

Result: **the hardware needs to decide what is stored in the cache at any given time.**

It wants to avoid, as much as possible, a situation in which the processor needs to access data that's not stored in the cache—this is called a **cache miss**.

Hardware uses **two heuristics**:

- The principle of temporal locality: **if a location in memory is accessed, it is likely that that location will be accessed again in the near future.**
- The principle of spatial locality: **if a location in memory is accessed, it is likely that other nearby locations will be accessed in the near future.**

Memory Locality

Temporal locality and spatial locality are both types of **memory locality**.

- We say that a program has good spatial locality and/or temporal locality and/or memory locality when it conforms to these heuristics.

- When a program has good memory locality, it makes good use of the caches available on the hardware.

- In practice, the throughput of a program is often substantially affected by the cache, and can be improved by increasing locality.

Prefetching

A third important heuristic used by both the hardware and the compiler to improve cache performance is **prefetching**.

Prefetching loads data into the cache **before it is ever accessed**, and is particularly useful when the program or the hardware can predict what memory will be used ahead of time.

Question: What can we do in the ML pipeline to increase locality and/or enable prefetching?

- Access training exmaples in the order they appear in memory
  - Prefetch the training examples (prefetch the ones we're going to be using next)
- Efficient matrix multiplications

DEMO

A matrix multiply of $A \in \mathbb{R}^{m \times n}$ and $B \in \mathbb{R}^{n \times p}$, producing output $C \in \mathbb{R}^{m \times p}$, can be written by running

$$C_{i,k} \mathrel{+}= A_{i,j} \cdot B_{j,k}$$

for each value of $i \in \{1, \ldots, m\}$, $j \in \{1, \ldots, n\}$, and $k \in \{1, \ldots, p\}$. The natural way to do this is with three for loops. But what order should we run these loops? And how does the way we store A, B, and C affect performance?

In [1]:
```
using Libdl # open a dynamic library that links to the C code
tm_lib = Libdl.dlopen("demo/test_memory.lib");
```

In [2]:
```
function test_mmpy(loop_order::String, Amaj::String, Bmaj::String, Cmaj::Strin
    @assert(loop_order in ["ijk","ikj","jki","jik","kij","kji"])
    @assert(Amaj in ["r","c"])
    @assert(Bmaj in ["r","c"])
    @assert(Cmaj in ["r","c"])
    f = Libdl.dlsym(tm_lib, "test_$(loop_order)_A$(Amaj)B$(Bmaj)C$(Cmaj)")
    ccall(f, Float64, (Int32, Int32, Int32, Int32), m, n, p, num_runs)
end
```

Out[2]:
```
test_mmpy (generic function with 1 method)
```

In [11]:
```
d = 512;
num_runs = 10;
measurements = []
for loop_order in ["ijk","ikj","jki","jik","kij","kji"]
    for Am in ["r","c"]
```

```
        for Bm in ["r","c"]
            for Cm in ["r","c"]
                push!(measurements, (loop_order * "_" * Am * Bm * Cm, test_mmpy
            end
        end
    end
end
```

```
time elapsed: 0.295000 seconds
time elapsed: 0.283000 seconds
time elapsed: 0.299000 seconds
time elapsed: 0.277000 seconds
time elapsed: 0.278000 seconds
time elapsed: 0.276000 seconds
time elapsed: 0.283000 seconds
time elapsed: 0.278000 seconds
time elapsed: 0.273000 seconds
time elapsed: 0.276000 seconds

average time: 0.281800 seconds

digest: 1.549011e+27
time elapsed: 0.439000 seconds
time elapsed: 0.424000 seconds
time elapsed: 0.429000 seconds
time elapsed: 0.431000 seconds
time elapsed: 0.430000 seconds
time elapsed: 0.425000 seconds
time elapsed: 0.427000 seconds
time elapsed: 0.428000 seconds
time elapsed: 0.425000 seconds
time elapsed: 0.429000 seconds

average time: 0.428700 seconds

digest: 1.546028e+27
time elapsed: 0.306000 seconds
time elapsed: 0.306000 seconds
time elapsed: 0.304000 seconds
time elapsed: 0.304000 seconds
time elapsed: 0.302000 seconds
time elapsed: 0.306000 seconds
time elapsed: 0.303000 seconds
time elapsed: 0.303000 seconds
time elapsed: 0.305000 seconds
time elapsed: 0.304000 seconds

average time: 0.304300 seconds

digest: 1.547263e+27
time elapsed: 0.515000 seconds
time elapsed: 0.518000 seconds
time elapsed: 0.514000 seconds
time elapsed: 0.536000 seconds
time elapsed: 0.516000 seconds
time elapsed: 0.513000 seconds
time elapsed: 0.546000 seconds
time elapsed: 0.513000 seconds
time elapsed: 0.518000 seconds
time elapsed: 0.514000 seconds

average time: 0.520300 seconds

digest: 1.547960e+27
time elapsed: 0.274000 seconds
time elapsed: 0.277000 seconds
time elapsed: 0.274000 seconds
time elapsed: 0.273000 seconds
```

```
time elapsed: 0.273000 seconds
time elapsed: 0.277000 seconds
time elapsed: 0.276000 seconds
time elapsed: 0.273000 seconds
time elapsed: 0.277000 seconds
time elapsed: 0.272000 seconds

average time: 0.274600 seconds

digest: 1.547434e+27
time elapsed: 0.424000 seconds
time elapsed: 0.431000 seconds
time elapsed: 0.427000 seconds
time elapsed: 0.431000 seconds
time elapsed: 0.429000 seconds
time elapsed: 0.425000 seconds
time elapsed: 0.427000 seconds
time elapsed: 0.427000 seconds
time elapsed: 0.425000 seconds
time elapsed: 0.434000 seconds

average time: 0.428000 seconds

digest: 1.550654e+27
time elapsed: 0.304000 seconds
time elapsed: 0.306000 seconds
time elapsed: 0.303000 seconds
time elapsed: 0.307000 seconds
time elapsed: 0.309000 seconds
time elapsed: 0.303000 seconds
time elapsed: 0.310000 seconds
time elapsed: 0.306000 seconds
time elapsed: 0.304000 seconds
time elapsed: 0.304000 seconds

average time: 0.305600 seconds

digest: 1.545477e+27
time elapsed: 0.521000 seconds
time elapsed: 0.520000 seconds
time elapsed: 0.516000 seconds
time elapsed: 0.522000 seconds
time elapsed: 0.519000 seconds
time elapsed: 0.537000 seconds
time elapsed: 0.520000 seconds
time elapsed: 0.519000 seconds
time elapsed: 0.519000 seconds
time elapsed: 0.516000 seconds

average time: 0.520900 seconds

digest: 1.546651e+27
time elapsed: 0.193000 seconds
time elapsed: 0.205000 seconds
time elapsed: 0.210000 seconds
time elapsed: 0.210000 seconds
time elapsed: 0.191000 seconds
time elapsed: 0.178000 seconds
time elapsed: 0.174000 seconds
time elapsed: 0.174000 seconds
```

```
time elapsed: 0.176000 seconds
time elapsed: 0.174000 seconds

average time: 0.188500 seconds

digest: 1.543978e+27
time elapsed: 0.186000 seconds
time elapsed: 0.180000 seconds
time elapsed: 0.173000 seconds
time elapsed: 0.174000 seconds
time elapsed: 0.187000 seconds
time elapsed: 0.176000 seconds
time elapsed: 0.183000 seconds
time elapsed: 0.187000 seconds
time elapsed: 0.174000 seconds
time elapsed: 0.173000 seconds

average time: 0.179300 seconds

digest: 1.544756e+27
time elapsed: 0.163000 seconds
time elapsed: 0.162000 seconds
time elapsed: 0.162000 seconds
time elapsed: 0.157000 seconds
time elapsed: 0.156000 seconds
time elapsed: 0.159000 seconds
time elapsed: 0.161000 seconds
time elapsed: 0.165000 seconds
time elapsed: 0.166000 seconds
time elapsed: 0.160000 seconds

average time: 0.161100 seconds

digest: 1.545344e+27
time elapsed: 0.117000 seconds
time elapsed: 0.121000 seconds
time elapsed: 0.118000 seconds
time elapsed: 0.117000 seconds
time elapsed: 0.118000 seconds
time elapsed: 0.119000 seconds
time elapsed: 0.119000 seconds
time elapsed: 0.118000 seconds
time elapsed: 0.118000 seconds
time elapsed: 0.117000 seconds

average time: 0.118200 seconds

digest: 1.546831e+27
time elapsed: 0.335000 seconds
time elapsed: 0.324000 seconds
time elapsed: 0.370000 seconds
time elapsed: 0.341000 seconds
time elapsed: 0.384000 seconds
time elapsed: 0.391000 seconds
time elapsed: 0.394000 seconds
time elapsed: 0.374000 seconds
time elapsed: 0.396000 seconds
time elapsed: 0.382000 seconds

average time: 0.369100 seconds
```

```
digest: 1.546806e+27
time elapsed: 0.385000 seconds
time elapsed: 0.385000 seconds
time elapsed: 0.385000 seconds
time elapsed: 0.335000 seconds
time elapsed: 0.339000 seconds
time elapsed: 0.383000 seconds
time elapsed: 0.366000 seconds
time elapsed: 0.350000 seconds
time elapsed: 0.325000 seconds
time elapsed: 0.372000 seconds

average time: 0.362500 seconds

digest: 1.545685e+27
time elapsed: 0.212000 seconds
time elapsed: 0.199000 seconds
time elapsed: 0.201000 seconds
time elapsed: 0.205000 seconds
time elapsed: 0.201000 seconds
time elapsed: 0.178000 seconds
time elapsed: 0.191000 seconds
time elapsed: 0.179000 seconds
time elapsed: 0.179000 seconds
time elapsed: 0.181000 seconds

average time: 0.192600 seconds

digest: 1.544665e+27
time elapsed: 0.199000 seconds
time elapsed: 0.199000 seconds
time elapsed: 0.185000 seconds
time elapsed: 0.179000 seconds
time elapsed: 0.185000 seconds
time elapsed: 0.206000 seconds
time elapsed: 0.209000 seconds
time elapsed: 0.207000 seconds
time elapsed: 0.204000 seconds
time elapsed: 0.194000 seconds

average time: 0.196700 seconds

digest: 1.544349e+27
time elapsed: 0.543000 seconds
time elapsed: 0.537000 seconds
time elapsed: 0.594000 seconds
time elapsed: 0.566000 seconds
time elapsed: 0.568000 seconds
time elapsed: 0.558000 seconds
time elapsed: 0.572000 seconds
time elapsed: 0.558000 seconds
time elapsed: 0.563000 seconds
time elapsed: 0.560000 seconds

average time: 0.561900 seconds

digest: 1.543135e+27
time elapsed: 0.210000 seconds
time elapsed: 0.204000 seconds
```

```
time elapsed: 0.214000 seconds
time elapsed: 0.205000 seconds
time elapsed: 0.216000 seconds
time elapsed: 0.214000 seconds
time elapsed: 0.211000 seconds
time elapsed: 0.208000 seconds
time elapsed: 0.219000 seconds
time elapsed: 0.209000 seconds

average time: 0.211000 seconds

digest: 1.548973e+27
time elapsed: 0.558000 seconds
time elapsed: 0.566000 seconds
time elapsed: 0.567000 seconds
time elapsed: 0.566000 seconds
time elapsed: 0.571000 seconds
time elapsed: 0.572000 seconds
time elapsed: 0.572000 seconds
time elapsed: 0.570000 seconds
time elapsed: 0.564000 seconds
time elapsed: 0.565000 seconds

average time: 0.567100 seconds

digest: 1.549237e+27
time elapsed: 0.227000 seconds
time elapsed: 0.209000 seconds
time elapsed: 0.215000 seconds
time elapsed: 0.222000 seconds
time elapsed: 0.212000 seconds
time elapsed: 0.214000 seconds
time elapsed: 0.213000 seconds
time elapsed: 0.215000 seconds
time elapsed: 0.218000 seconds
time elapsed: 0.219000 seconds

average time: 0.216400 seconds

digest: 1.543348e+27
time elapsed: 0.448000 seconds
time elapsed: 0.447000 seconds
time elapsed: 0.451000 seconds
time elapsed: 0.446000 seconds
time elapsed: 0.452000 seconds
time elapsed: 0.446000 seconds
time elapsed: 0.453000 seconds
time elapsed: 0.455000 seconds
time elapsed: 0.446000 seconds
time elapsed: 0.433000 seconds

average time: 0.447700 seconds

digest: 1.548308e+27
time elapsed: 0.053000 seconds
time elapsed: 0.054000 seconds
time elapsed: 0.050000 seconds
time elapsed: 0.052000 seconds
time elapsed: 0.051000 seconds
time elapsed: 0.051000 seconds
```

```
time elapsed: 0.052000 seconds
time elapsed: 0.050000 seconds
time elapsed: 0.052000 seconds
time elapsed: 0.052000 seconds

average time: 0.051700 seconds

digest: 1.548871e+27
time elapsed: 0.422000 seconds
time elapsed: 0.441000 seconds
time elapsed: 0.420000 seconds
time elapsed: 0.417000 seconds
time elapsed: 0.429000 seconds
time elapsed: 0.426000 seconds
time elapsed: 0.416000 seconds
time elapsed: 0.416000 seconds
time elapsed: 0.421000 seconds
time elapsed: 0.418000 seconds

average time: 0.422600 seconds

digest: 1.543851e+27
time elapsed: 0.053000 seconds
time elapsed: 0.053000 seconds
time elapsed: 0.051000 seconds
time elapsed: 0.053000 seconds
time elapsed: 0.052000 seconds
time elapsed: 0.052000 seconds
time elapsed: 0.052000 seconds
time elapsed: 0.058000 seconds
time elapsed: 0.054000 seconds
time elapsed: 0.051000 seconds

average time: 0.052900 seconds

digest: 1.547475e+27
time elapsed: 0.279000 seconds
time elapsed: 0.275000 seconds
time elapsed: 0.276000 seconds
time elapsed: 0.275000 seconds
time elapsed: 0.281000 seconds
time elapsed: 0.279000 seconds
time elapsed: 0.275000 seconds
time elapsed: 0.279000 seconds
time elapsed: 0.275000 seconds
time elapsed: 0.275000 seconds

average time: 0.276900 seconds

digest: 1.543076e+27
time elapsed: 0.424000 seconds
time elapsed: 0.421000 seconds
time elapsed: 0.422000 seconds
time elapsed: 0.435000 seconds
time elapsed: 0.423000 seconds
time elapsed: 0.419000 seconds
time elapsed: 0.424000 seconds
time elapsed: 0.422000 seconds
time elapsed: 0.426000 seconds
time elapsed: 0.444000 seconds
```

```
average time: 0.426000 seconds

digest: 1.547920e+27
time elapsed: 0.282000 seconds
time elapsed: 0.280000 seconds
time elapsed: 0.280000 seconds
time elapsed: 0.289000 seconds
time elapsed: 0.293000 seconds
time elapsed: 0.278000 seconds
time elapsed: 0.289000 seconds
time elapsed: 0.310000 seconds
time elapsed: 0.297000 seconds
time elapsed: 0.285000 seconds

average time: 0.288300 seconds

digest: 1.550322e+27
time elapsed: 0.519000 seconds
time elapsed: 0.519000 seconds
time elapsed: 0.519000 seconds
time elapsed: 0.519000 seconds
time elapsed: 0.516000 seconds
time elapsed: 0.514000 seconds
time elapsed: 0.516000 seconds
time elapsed: 0.519000 seconds
time elapsed: 0.517000 seconds
time elapsed: 0.525000 seconds

average time: 0.518300 seconds

digest: 1.545686e+27
time elapsed: 0.294000 seconds
time elapsed: 0.291000 seconds
time elapsed: 0.292000 seconds
time elapsed: 0.294000 seconds
time elapsed: 0.293000 seconds
time elapsed: 0.293000 seconds
time elapsed: 0.294000 seconds
time elapsed: 0.293000 seconds
time elapsed: 0.291000 seconds
time elapsed: 0.293000 seconds

average time: 0.292800 seconds

digest: 1.548871e+27
time elapsed: 0.424000 seconds
time elapsed: 0.426000 seconds
time elapsed: 0.424000 seconds
time elapsed: 0.419000 seconds
time elapsed: 0.423000 seconds
time elapsed: 0.428000 seconds
time elapsed: 0.422000 seconds
time elapsed: 0.425000 seconds
time elapsed: 0.425000 seconds
time elapsed: 0.428000 seconds

average time: 0.424400 seconds

digest: 1.544744e+27
```

```
time elapsed: 0.280000 seconds
time elapsed: 0.276000 seconds
time elapsed: 0.275000 seconds
time elapsed: 0.284000 seconds
time elapsed: 0.276000 seconds
time elapsed: 0.278000 seconds
time elapsed: 0.278000 seconds
time elapsed: 0.278000 seconds
time elapsed: 0.280000 seconds
time elapsed: 0.279000 seconds

average time: 0.278400 seconds

digest: 1.549813e+27
time elapsed: 0.514000 seconds
time elapsed: 0.518000 seconds
time elapsed: 0.516000 seconds
time elapsed: 0.516000 seconds
time elapsed: 0.516000 seconds
time elapsed: 0.512000 seconds
time elapsed: 0.519000 seconds
time elapsed: 0.520000 seconds
time elapsed: 0.517000 seconds
time elapsed: 0.521000 seconds

average time: 0.516900 seconds

digest: 1.546276e+27
time elapsed: 0.202000 seconds
time elapsed: 0.208000 seconds
time elapsed: 0.199000 seconds
time elapsed: 0.192000 seconds
time elapsed: 0.192000 seconds
time elapsed: 0.193000 seconds
time elapsed: 0.198000 seconds
time elapsed: 0.183000 seconds
time elapsed: 0.203000 seconds
time elapsed: 0.185000 seconds

average time: 0.195500 seconds

digest: 1.549451e+27
time elapsed: 0.198000 seconds
time elapsed: 0.181000 seconds
time elapsed: 0.196000 seconds
time elapsed: 0.187000 seconds
time elapsed: 0.194000 seconds
time elapsed: 0.194000 seconds
time elapsed: 0.195000 seconds
time elapsed: 0.195000 seconds
time elapsed: 0.188000 seconds
time elapsed: 0.199000 seconds

average time: 0.192700 seconds

digest: 1.550307e+27
time elapsed: 0.162000 seconds
time elapsed: 0.161000 seconds
time elapsed: 0.162000 seconds
time elapsed: 0.162000 seconds
```

```
time elapsed: 0.162000 seconds
time elapsed: 0.159000 seconds
time elapsed: 0.159000 seconds
time elapsed: 0.158000 seconds
time elapsed: 0.156000 seconds
time elapsed: 0.158000 seconds

average time: 0.159900 seconds

digest: 1.547840e+27
time elapsed: 0.117000 seconds
time elapsed: 0.118000 seconds
time elapsed: 0.117000 seconds
time elapsed: 0.116000 seconds
time elapsed: 0.119000 seconds
time elapsed: 0.117000 seconds
time elapsed: 0.116000 seconds
time elapsed: 0.117000 seconds
time elapsed: 0.116000 seconds
time elapsed: 0.117000 seconds

average time: 0.117000 seconds

digest: 1.541549e+27
time elapsed: 0.324000 seconds
time elapsed: 0.337000 seconds
time elapsed: 0.322000 seconds
time elapsed: 0.322000 seconds
time elapsed: 0.386000 seconds
time elapsed: 0.358000 seconds
time elapsed: 0.352000 seconds
time elapsed: 0.385000 seconds
time elapsed: 0.380000 seconds
time elapsed: 0.380000 seconds

average time: 0.354600 seconds

digest: 1.542940e+27
time elapsed: 0.391000 seconds
time elapsed: 0.370000 seconds
time elapsed: 0.378000 seconds
time elapsed: 0.375000 seconds
time elapsed: 0.369000 seconds
time elapsed: 0.363000 seconds
time elapsed: 0.322000 seconds
time elapsed: 0.382000 seconds
time elapsed: 0.393000 seconds
time elapsed: 0.370000 seconds

average time: 0.371300 seconds

digest: 1.547824e+27
time elapsed: 0.200000 seconds
time elapsed: 0.201000 seconds
time elapsed: 0.202000 seconds
time elapsed: 0.175000 seconds
time elapsed: 0.187000 seconds
time elapsed: 0.187000 seconds
time elapsed: 0.201000 seconds
time elapsed: 0.196000 seconds
```

```
time elapsed: 0.177000 seconds
time elapsed: 0.183000 seconds

average time: 0.190900 seconds

digest: 1.549576e+27
time elapsed: 0.175000 seconds
time elapsed: 0.179000 seconds
time elapsed: 0.190000 seconds
time elapsed: 0.186000 seconds
time elapsed: 0.179000 seconds
time elapsed: 0.186000 seconds
time elapsed: 0.190000 seconds
time elapsed: 0.182000 seconds
time elapsed: 0.189000 seconds
time elapsed: 0.186000 seconds

average time: 0.184200 seconds

digest: 1.545518e+27
time elapsed: 0.518000 seconds
time elapsed: 0.517000 seconds
time elapsed: 0.514000 seconds
time elapsed: 0.511000 seconds
time elapsed: 0.518000 seconds
time elapsed: 0.517000 seconds
time elapsed: 0.514000 seconds
time elapsed: 0.515000 seconds
time elapsed: 0.515000 seconds
time elapsed: 0.512000 seconds

average time: 0.515100 seconds

digest: 1.550872e+27
time elapsed: 0.184000 seconds
time elapsed: 0.195000 seconds
time elapsed: 0.195000 seconds
time elapsed: 0.198000 seconds
time elapsed: 0.197000 seconds
time elapsed: 0.192000 seconds
time elapsed: 0.197000 seconds
time elapsed: 0.182000 seconds
time elapsed: 0.182000 seconds
time elapsed: 0.180000 seconds

average time: 0.190200 seconds

digest: 1.548188e+27
time elapsed: 0.512000 seconds
time elapsed: 0.512000 seconds
time elapsed: 0.528000 seconds
time elapsed: 0.514000 seconds
time elapsed: 0.516000 seconds
time elapsed: 0.515000 seconds
time elapsed: 0.514000 seconds
time elapsed: 0.515000 seconds
time elapsed: 0.515000 seconds
time elapsed: 0.513000 seconds

average time: 0.515400 seconds
```

```
digest: 1.550548e+27
time elapsed: 0.195000 seconds
time elapsed: 0.192000 seconds
time elapsed: 0.192000 seconds
time elapsed: 0.185000 seconds
time elapsed: 0.194000 seconds
time elapsed: 0.186000 seconds
time elapsed: 0.193000 seconds
time elapsed: 0.186000 seconds
time elapsed: 0.187000 seconds
time elapsed: 0.195000 seconds

average time: 0.190500 seconds

digest: 1.547162e+27
time elapsed: 0.428000 seconds
time elapsed: 0.424000 seconds
time elapsed: 0.424000 seconds
time elapsed: 0.421000 seconds
time elapsed: 0.424000 seconds
time elapsed: 0.425000 seconds
time elapsed: 0.423000 seconds
time elapsed: 0.425000 seconds
time elapsed: 0.417000 seconds
time elapsed: 0.424000 seconds

average time: 0.423500 seconds

digest: 1.549798e+27
time elapsed: 0.052000 seconds
time elapsed: 0.049000 seconds
time elapsed: 0.050000 seconds
time elapsed: 0.049000 seconds
time elapsed: 0.049000 seconds
time elapsed: 0.050000 seconds
time elapsed: 0.050000 seconds
time elapsed: 0.051000 seconds
time elapsed: 0.050000 seconds
time elapsed: 0.051000 seconds

average time: 0.050100 seconds

digest: 1.545493e+27
time elapsed: 0.428000 seconds
time elapsed: 0.423000 seconds
time elapsed: 0.423000 seconds
time elapsed: 0.420000 seconds
time elapsed: 0.419000 seconds
time elapsed: 0.422000 seconds
time elapsed: 0.418000 seconds
time elapsed: 0.419000 seconds
time elapsed: 0.424000 seconds
time elapsed: 0.421000 seconds

average time: 0.421700 seconds

digest: 1.549679e+27
time elapsed: 0.065000 seconds
time elapsed: 0.064000 seconds
```

```
time elapsed: 0.063000 seconds
time elapsed: 0.064000 seconds
time elapsed: 0.067000 seconds
time elapsed: 0.066000 seconds
time elapsed: 0.064000 seconds
time elapsed: 0.063000 seconds
time elapsed: 0.063000 seconds
time elapsed: 0.063000 seconds

average time: 0.064200 seconds

digest: 1.545748e+27
```

In [12]:

```
m_sorted = [(m,t) for (t,m) in sort([(t,m) for (m,t) in measurements])]
for (m,t) in m_sorted
    println("$m --> $t");
end
```

```
kji_crc --> 0.0501
jki_crc --> 0.0517
jki_ccc --> 0.0529
kji_ccc --> 0.0642
kij_rcc --> 0.11700000000000002
ikj_rcc --> 0.1182
kij_rcr --> 0.1599
ikj_rcr --> 0.1611
ikj_rrc --> 0.17930000000000001
kij_ccc --> 0.18419999999999997
ikj_rrr --> 0.18849999999999995
kji_rrc --> 0.19019999999999998
kji_rcc --> 0.1905
kij_ccr --> 0.19090000000000001
ikj_ccr --> 0.19260000000000005
kij_rrc --> 0.1927
kij_rrr --> 0.1955
ikj_ccc --> 0.1967
jki_rrc --> 0.211
jki_rcc --> 0.2164
ijk_crr --> 0.27460000000000007
jik_rrr --> 0.2769
jik_ccr --> 0.2784000000000001
ijk_rrr --> 0.28180000000000005
jik_rcr --> 0.2883
jik_crr --> 0.2928
ijk_rcr --> 0.3043
ijk_ccr --> 0.3056
kij_crr --> 0.3546
ikj_crc --> 0.3625
ikj_crr --> 0.36910000000000004
kij_crc --> 0.3713
kji_ccr --> 0.4217000000000001
jki_ccr --> 0.4226
kji_crr --> 0.42349999999999993
jik_crc --> 0.4244
jik_rrc --> 0.42600000000000005
ijk_crc --> 0.42799999999999994
ijk_rrc --> 0.42869999999999997
jki_crr --> 0.44770000000000004
kji_rrr --> 0.5151
kji_rcr --> 0.5153999999999999
jik_ccc --> 0.5169
jik_rcc --> 0.5183000000000001
ijk_rcc --> 0.5203
ijk_ccc --> 0.5208999999999999
jki_rrr --> 0.5619
jki_rcr --> 0.5671000000000002
```

In [13]:
```
maximum(t for (m,t) in m_sorted) / minimum(t for (m,t) in m_sorted)
```

Out[13]:
```
11.319361277445113
```

An over $10\times$ difference just from accessing memory in a different order!

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

[a,b,c,d]

[a,c,b,d]

Scan order

Scan order refers to the order in which the training examples are used in a learning algorithm.

As you saw in the programming assignment, using a non-random scan order is an option that can sometimes improve performance by increasing memory locality.

Here are a few scan orders that people use:

- **Random sampling with replacement** (a.k.a. random scan): every time we need a new sample, we pick one at random from the whole training dataset.
- **Random sampling without replacement**: every time we need a new sample, we pick one at random and then discard it (it won't be sampled again). Once we've gone through the whole training set, we replace all the samples and continue.
- **Sequential scan** (a.k.a. systematic scan): sample the data in the order in which it appears in memory. When you get to the end of the training set, restart at the beginning.
- **Shuffle-once**: at the beginning of execution, randomly shuffle the training data. Then sample the data in that shuffled order. When you get to the end of the training set, restart at the beginning.
- **Random reshuffling**: at the beginning of execution, randomly shuffle the training data. Then sample the data in that shuffled order. When you get to the end of the training set, reshuffle the training set, then restart at the beginning.

How does the memory locality of these different scan orders compare?

- Worst memory locality: random scan with and without resampling
- Okay memory locality: random reshuffling
- Very good: shuffle once
- Best: sequential scan

Two of these scan orders are actually statistically equivalent! Which ones?

- Random sampling w/o replacement and random reshuffling

How does the statistical performance of these different scan orders compare?

random reshuffling = w/o replacement > shuffle once > sequential scan > random with replacement

A good first choice when compute is light: shuffle once

Generally it performs quite well statistically (although it might have weaker theoretical guarantees), and it has good memory locality.

Another good choice: without-replacement sampling

This is particularly good when you're doing some sort of data augmentation, since you can construct the without-replacement minibatches on-the-fly.

Memory and sparsity

How does the use of sparsity impact the memory subsystem?

Two major effects:

- Sparsity lowers the total amount of memory in use by the program.
- Sparsity lowers the memory locality.
  - Why? Accesses are not dense and so are less predictable.

What else can we do to lower the total memory usage of the machine learning pipeline?