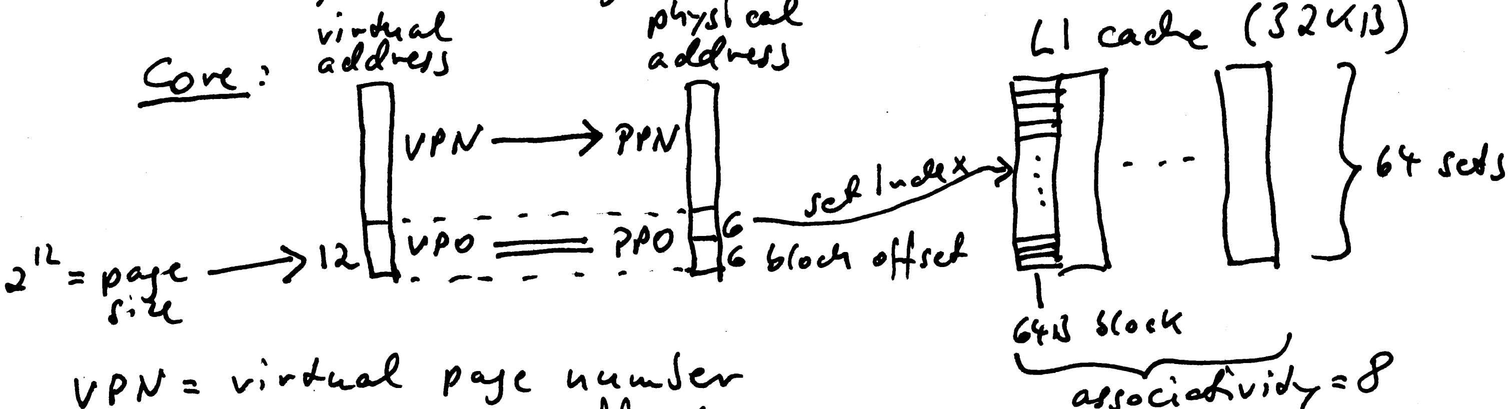
Optimizations related to virtual memory

Background: (Core)
- the processor works with virtual addresses
- all caches work with physical addresses
- both address spaces are organized in pages
  page size: 4KB
- address translation: virtual page number → physical page number



VPN = virtual page number
VPO = " " offset
PPN = physical page number
PPO = " " offset

VPO = PPO = set index ∨ block offset ⇒ cache lookup can start in parallel with address translation

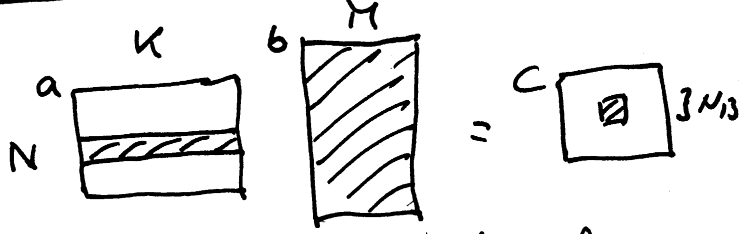
Address translation
- uses a cache called translation lookaside buffer (TLB)
- Core 2: two levels of TLB for data
  - DTLB0: 16 entries
  - DTLB1: 256 "

DTLB0 hit: no penalty
DTLB1 hit: 2 cycles penalty
DTLB1 miss: expensive

Consequence
repeatedly accessing a working set that is spread over ≫ 256 pages leads to TLB misses → slowdown

one possible solution: copy to contiguous memory

How does this affect MMM?



working set at highest level is shaded

we look for parts of the working set that are spread in memory.

- block row of a: contiguous
- all of b: "
- block of c: if $M > 512$ (doubles = 4KB), then spread over $\geq N_B$ pages

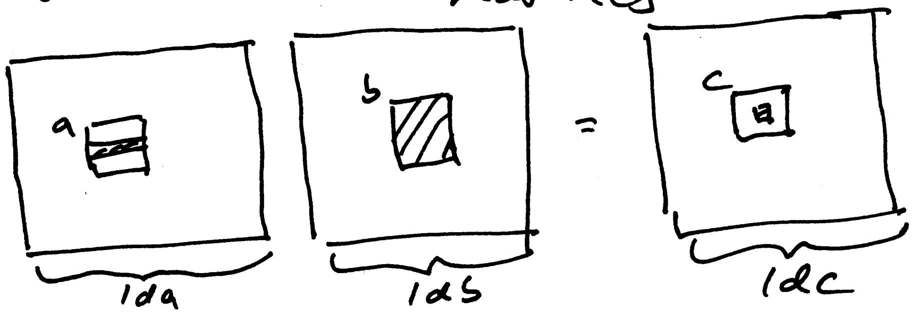
but: N_B typically < 256 so no problem
↑ DTLB1 size

So no problems.

But: the BLAS3 function dgemm has this interface

dgemm(a, b, c, N, K, M, lda, ldb, ldc)
(a, b, c: matrices; N, K, M: sizes; lda, ldb, ldc: leading dimensions)

Leading dimensions: enable use on matrices inside matrices



assume lda, ldb, ldc > 512

- block row of a: spread over $\geq N_B$ pages
- all of b: spread over $\geq K$ pages
- block of c: spread over $\geq N_B$ pages

So copying may pay off. Code:

```
// all of b reused: possibly copy
for i = 0 : N_B : N-1
    // block row of a reused: possibly copy
    for j = 0 : N_B : M-1
        // block of c reused: possibly copy
        for k = 0 : N_B : K-1
```