Optimizations related to virtual memory

Background:
- The processor works with virtual addresses.
- All caches work with physical addresses.
- Both address spaces are organized in pages.
  - Page size: 4 KB
- Address translation: virtual page number 
  \[ \rightarrow \] physical page number

\[ \begin{array}{c|c|c}
  \text{virtual address} & \text{physical address} \\
  \hline
  \text{Core}: & \text{VPN} \rightarrow \text{PPN} & \text{set index} \\
  \hline
  \text{VPO} & \text{PRO} & \text{block offset}
\end{array}\]

\[2^{12} = \text{page size} \rightarrow 12\]

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

VPO = PRO = set index \& block offset \rightarrow 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 entries
- DTLB0 hit: no penalty
- DTLB1 hit: 2 cycles penalty
- DTLB1 miss: expensive

Consequence
- Repeatedly accessing a conflicting set that is spread over \(\gg 256\) pages leads to TLB misses \(\rightarrow\) slow down

One possible solution: copy to contiguous memory
How does this affect \( \mathbb{Z}_p \)?

\[
\begin{array}{ccc}
\alpha & b & c \\
N & = & \text{shaded}
\end{array}
\]

we look for parts of the \( \mathbb{Z}_p \) set that are spread in memory.

- block row of \( a \): contiguous
- all of \( b \)
- block of \( c \): if \( N > 512 \) (double = 4KB), then spread over \( \geq N \) pages

but: \( N \) typically \( < 256 \) so no problem

so no problems.

**But:** the BLAS3 function \( \text{gemm} \) has this interface:

\[
\text{gemm} \left( a, b, c, N_k, N_k, N, \text{ld}_a, \text{ld}_b, \text{ld}_c \right)
\]

leading dimensions: easily use on matrices inside

leading dimensions: assume \( \text{ld}_a, \text{ld}_b, \text{ld}_c > 512 \)

- block row of \( a \): spread over \( \geq N \) pages
- all of \( b \): spread over \( \geq K \) pages
- block of \( c \): spread over \( \geq N \) pages

so copying may pay off. Code:

```c
// all of \( b \) reused: possibly copy
for \( i = 0 \) : \( N_b \) : \( N-1 \)
    // block row of \( a \) reused: possibly copy
    for \( j = 0 \) : \( N_b \) : \( N-1 \)
        // block of \( c \) reused: possibly copy
        for \( k = 0 \) : \( N_b \) : \( K-1 \)
```