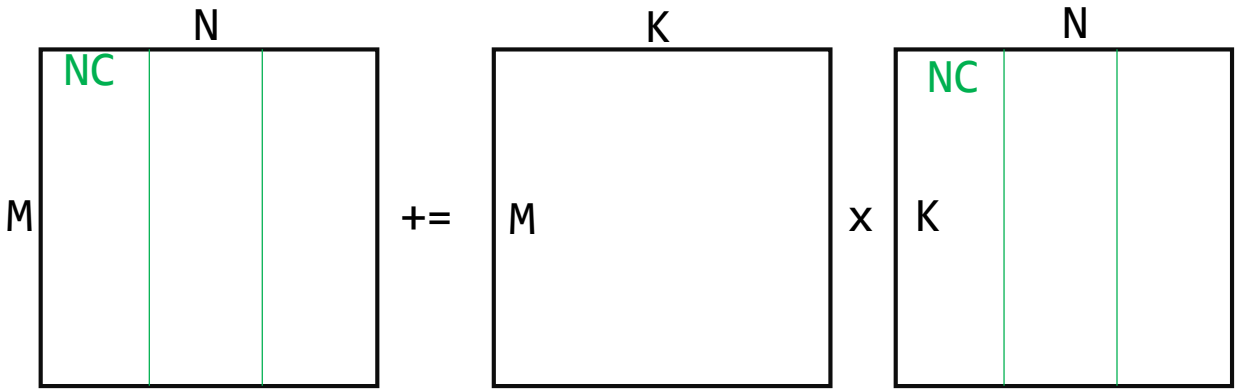
CPU GEMM Blocking & optimization

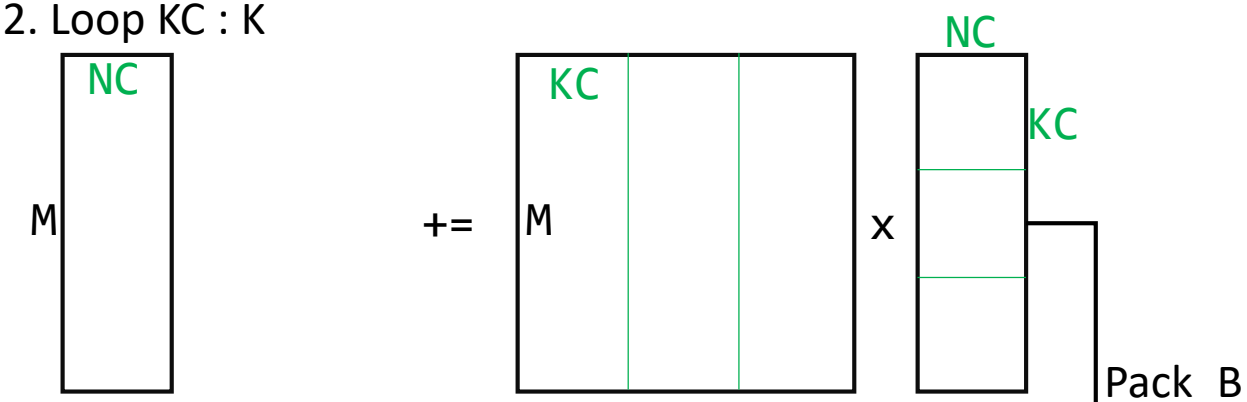
carlushuang@hotmail.com

1). Gemm blocking, described in blislab (col major)

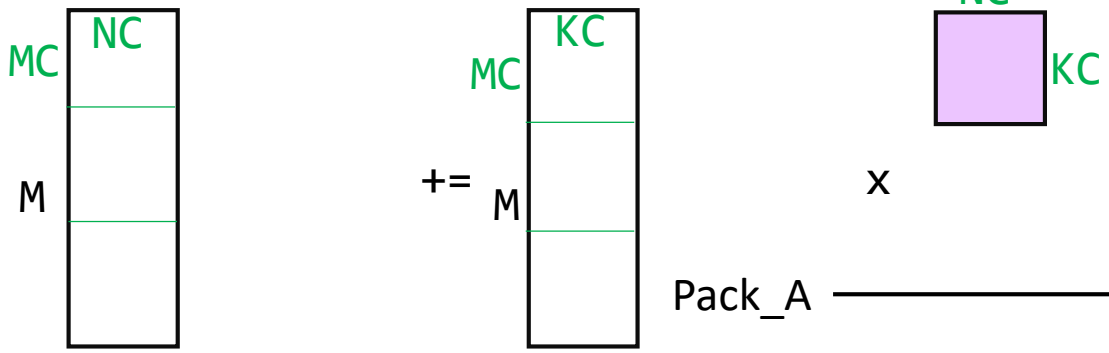
1. Loop NC : N



2. Loop KC : K

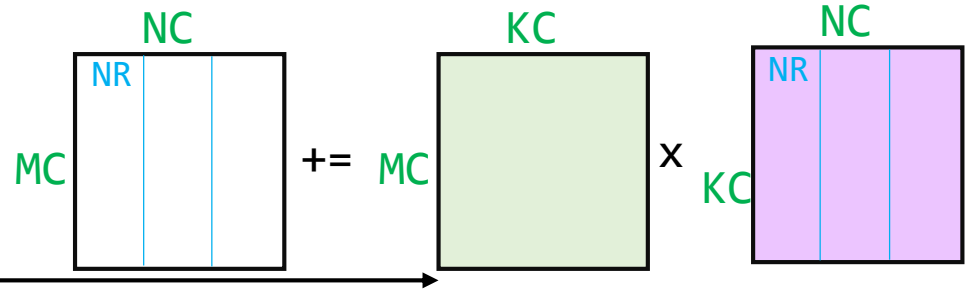


3. Loop MC : M (GEPB, in GOTO)

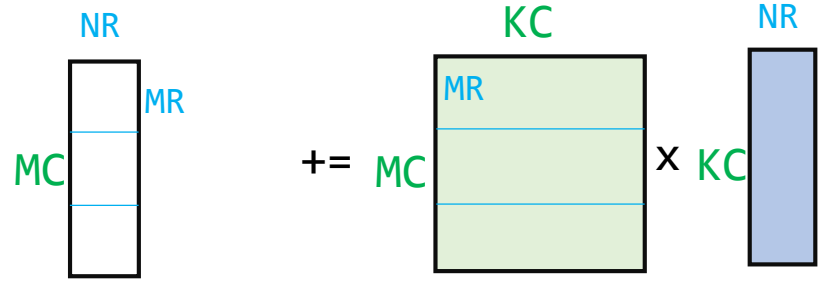


Macro Kernel

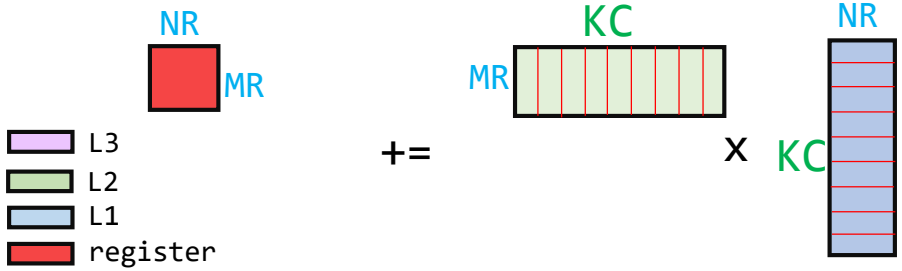
4. Loop NR : NC

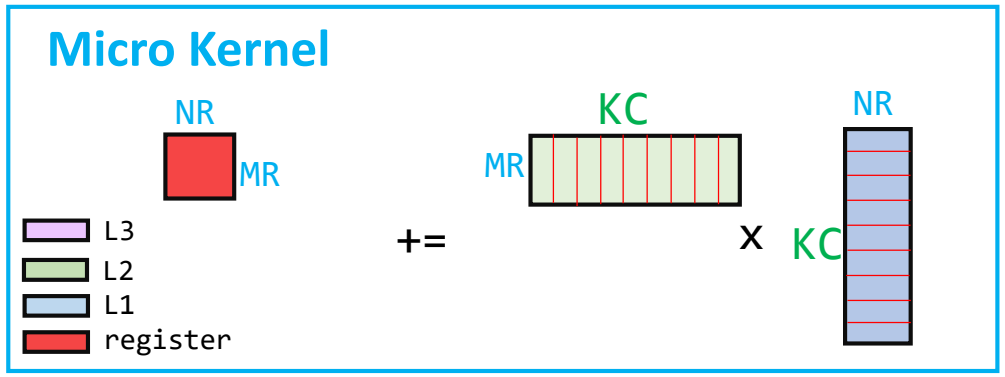


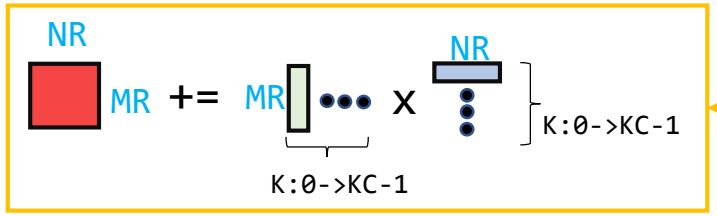
5. Loop MR : MC



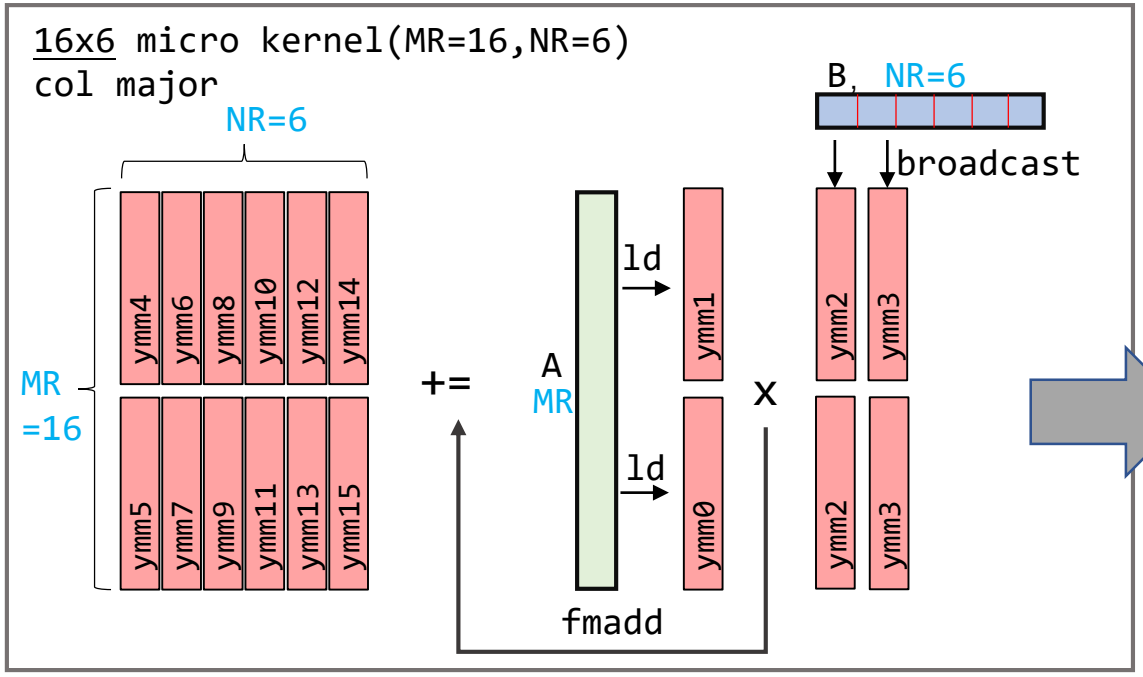
Micro Kernel







X86_64, Use avx256, ymm0-15, total **16 registers**,
256bit each(can load 8 float)

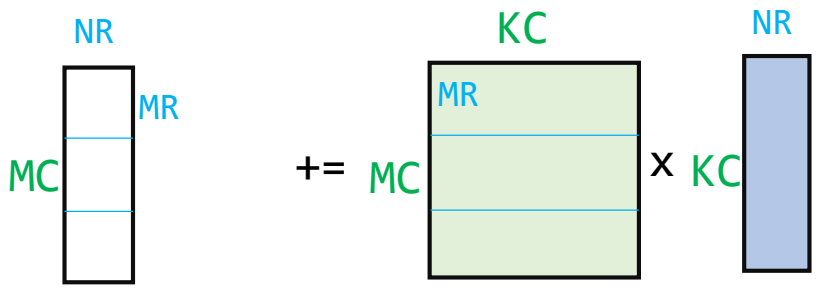


2). Micro kernel design for X86_64, avx256 (col major)

```
Load NR*MR C into register
For k : 0->KC-1
    load MR float of A into register
    load NR float of B into register
    dot product MR, NR,
       accumulate into MR*NC C
End for
Store MR*NC C into memory
```

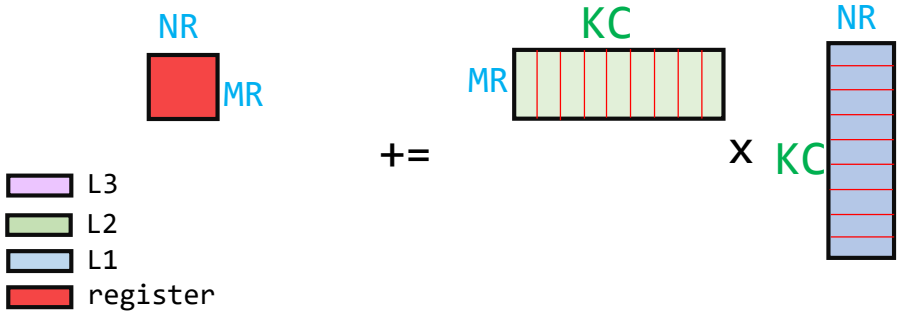
```
"vmovaps 0*32(%%rax), %%ymm0          \n" // A panel 0
"vmovaps 1*32(%%rax), %%ymm1          \n" // A panel 1

"vbroadcastss 0*4(%%rbx), %%ymm2      \n" // B broadcast 0
"vbroadcastss 1*4(%%rbx), %%ymm3      \n" // B broadcast 1
"vfmadd231ps %%ymm0, %%ymm2, %%ymm4   \n"
"vfmadd231ps %%ymm1, %%ymm2, %%ymm5   \n"
"vfmadd231ps %%ymm0, %%ymm3, %%ymm6   \n"
"vfmadd231ps %%ymm1, %%ymm3, %%ymm7   \n"

"vbroadcastss 2*4(%%rbx), %%ymm2      \n" // B broadcast 2
"vbroadcastss 3*4(%%rbx), %%ymm3      \n" // B broadcast 3
"vfmadd231ps %%ymm0, %%ymm2, %%ymm8   \n"
"vfmadd231ps %%ymm1, %%ymm2, %%ymm9   \n"
"vfmadd231ps %%ymm0, %%ymm3, %%ymm10  \n"
"vfmadd231ps %%ymm1, %%ymm3, %%ymm11  \n"

"vbroadcastss 4*4(%%rbx), %%ymm2      \n" // B broadcast 4
"vbroadcastss 5*4(%%rbx), %%ymm3      \n" // B broadcast 5
"vfmadd231ps %%ymm0, %%ymm2, %%ymm12  \n"
"vfmadd231ps %%ymm1, %%ymm2, %%ymm13  \n"
"vfmadd231ps %%ymm0, %%ymm3, %%ymm14  \n"
"vfmadd231ps %%ymm1, %%ymm3, %%ymm15  \n"
```

5. Loop MR : M



3). More detailed consideration for size choice of MC/NC/KC/MR/NR

Micro Kernel




```
load MR*NR of C into register
for k = 0 -> KC:
    load MR*1 of A into register
    load NR*1 of B into register
    perform C += A*B
end
Store MR*NR of C into memory
```

L1 consideration:

Micro kernel is preferred all in L1

```
MR*KC + NR*KC + MR*NR + MR + MR*NR <
                                    L1_size/(dtype_size)
```

A panel | B panel | C ld/st | next loop A panel, single MR*1

next loop, C ld/st

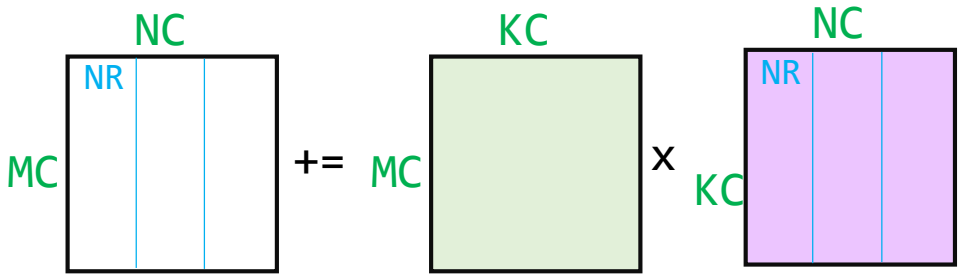
Hence B panel(NR*KC) can make sure not evicted from L1 cache*

** LRU cache update*

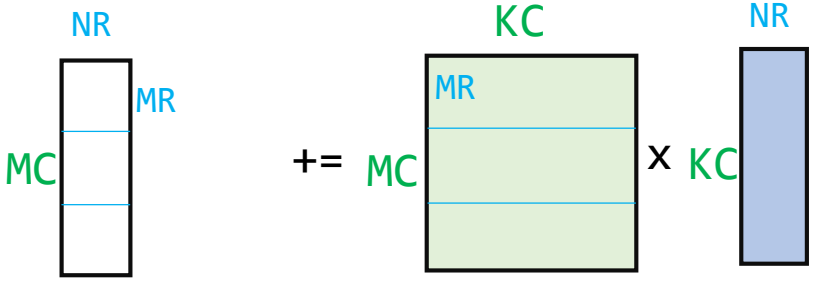
Further: consider CACHE_LINE_SIZE alignment

Macro Kernel

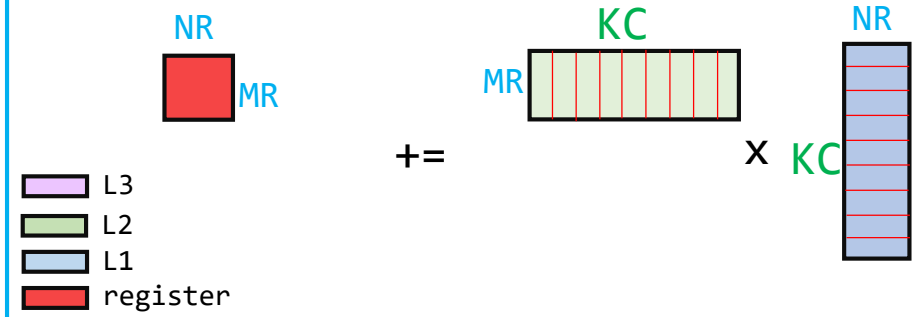
4. Loop NR : NC



5. Loop MR : MC



Micro Kernel



L2 consideration:

Loop5 is preferred all in L2

```
MC*KC + NR*KC + MC*NR + NR*KC + MC*NR < L2_size/(dtype_size)
```

A block B panel C panel next loop B panel

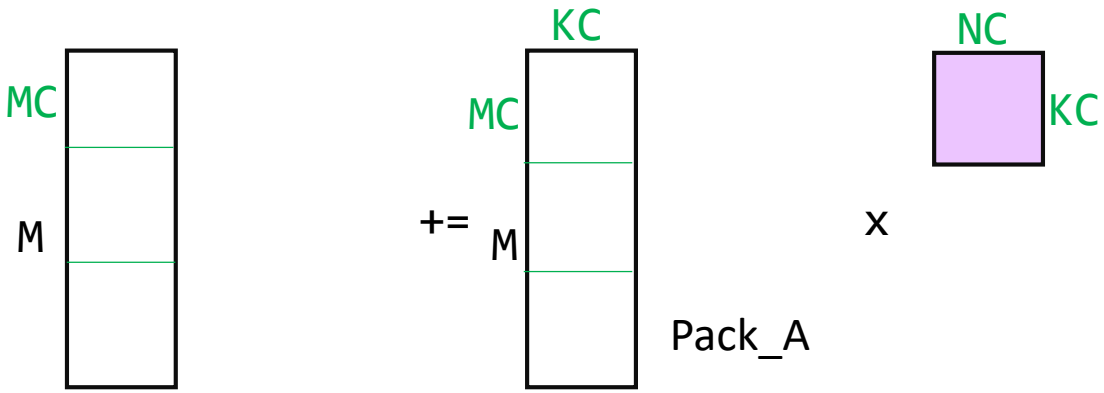
next loop, C panel

Hence A block(MC*KC) can make sure not evicted from L2 cache*

** LRU cache update*

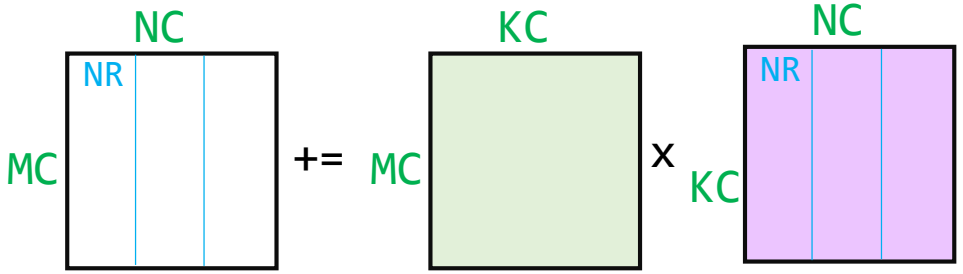
Further: consider CACHE_LINE_SIZE alignment

3. Loop MC : M (GEPB, in GOTO)



Macro Kernel

4. Loop NR : NC



L3 consideration:

Loop4(macro kernel) is preferred all in L3

```
MC*KC + NC*KC + MC*NC + NC*KC + MC*NC <
                                        L3_size/(dtype_size)
```

A block B block C block next loop B block

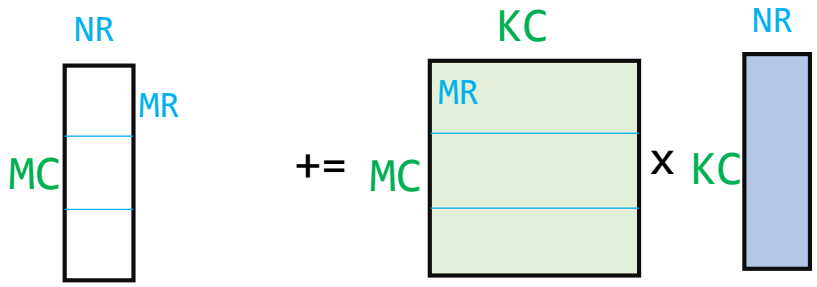
next loop, C block

Hence B block(NC*KC) can make sure not evicted from L3 cache*

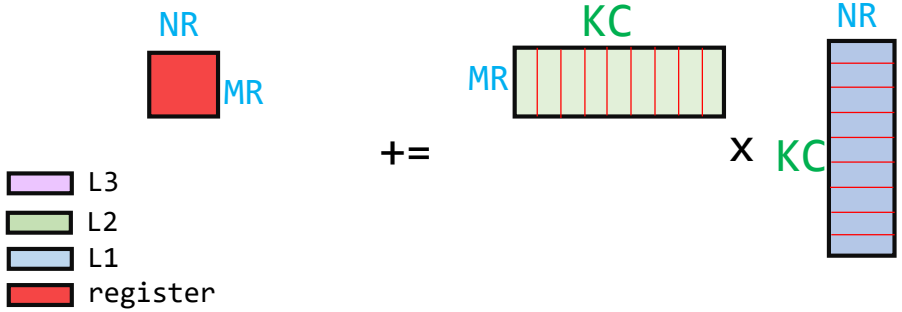
** LRU cache update*

Further: consider CACHE_LINE_SIZE alignment

5. Loop MR : MC



Micro Kernel



TLB consideration: (only L1D TLB)

In every micro kernel, TLB miss preferred not happen:

2*TA + TB + TC < T_entry_total

TA: TLB entry used for A block(MR*KC)
TB: TLB entry used for B block(KC*NR)
TC: TLB entry used for C block(MC*NR)

2*TA, current TA and next loop TA. TC is not multipled by 2 because in current layout(col major), each col use probably the same TLB entry for every loop of MR*NR of C.

TA: CEIL(MR*KC*d_size/PAGE_SIZE)+1
TB: CEIL(NR*KC*d_size/PAGE_SIZE)+1
TC: up to MR (assume ldc is every big)

CEIL()+1, is the worst case a memory can consume of PAGE_SIZE. e.g, for 4K page, a 5K memory can cover at least 2 pages, but can consume 3 pages in the worst case.

*Note: TLB miss can take a big place when M/N/K is big enough.(>4K)

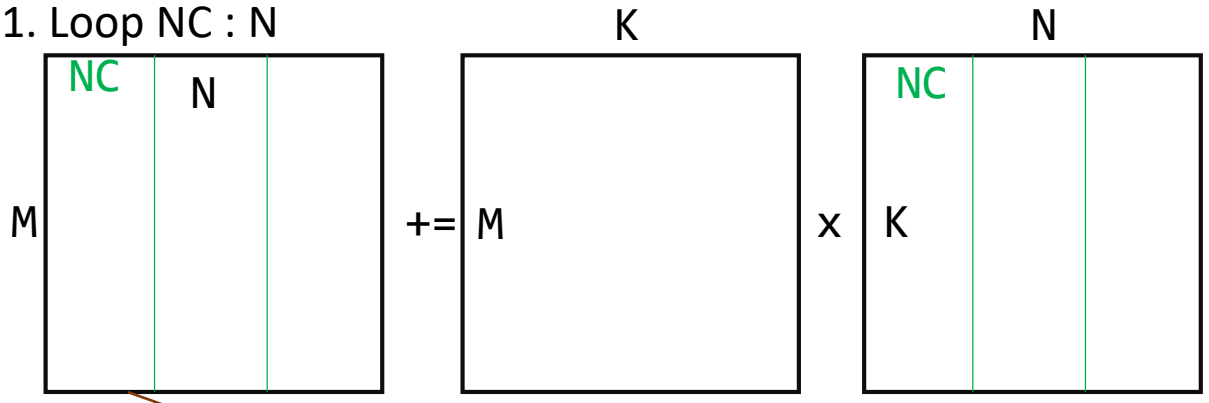
*Note: L1D TLB entry is not big enough to cover loop5 on my x86 cpu. But if entry size is very big, or the MR/MR/KC is not so big due to L1/L2/L3 consideration, maybe can try to cover loop2 when analysis TLB.

col major

4). Optimal blocking for row/col major

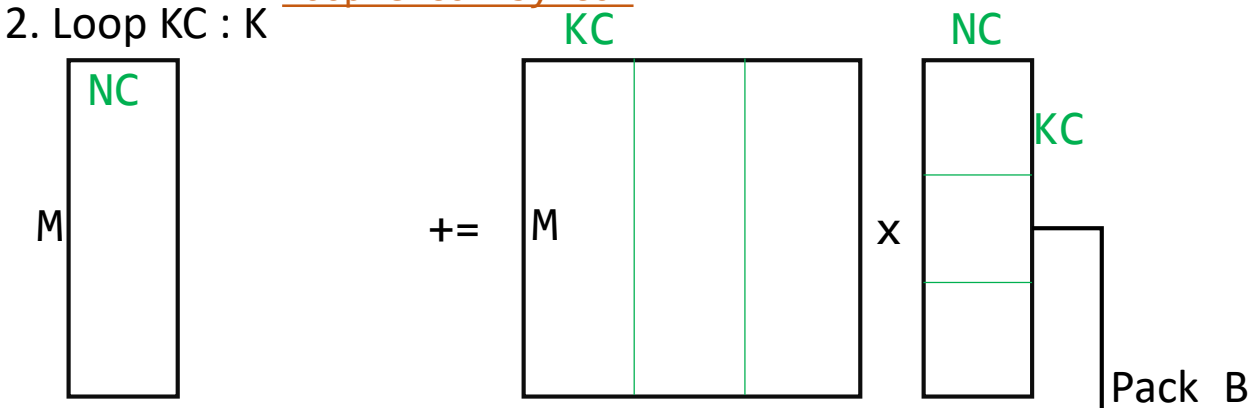
row major

1. Loop NC : N

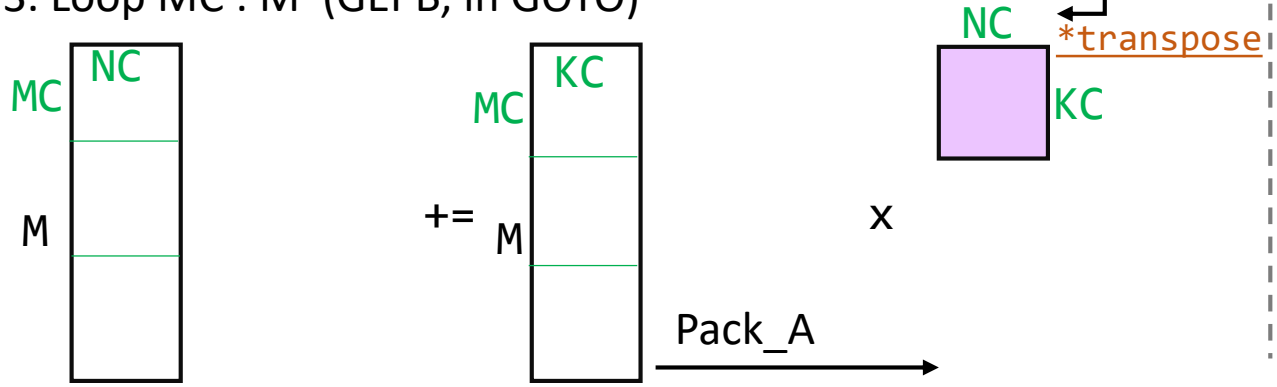


Loop C col by col

2. Loop KC : K

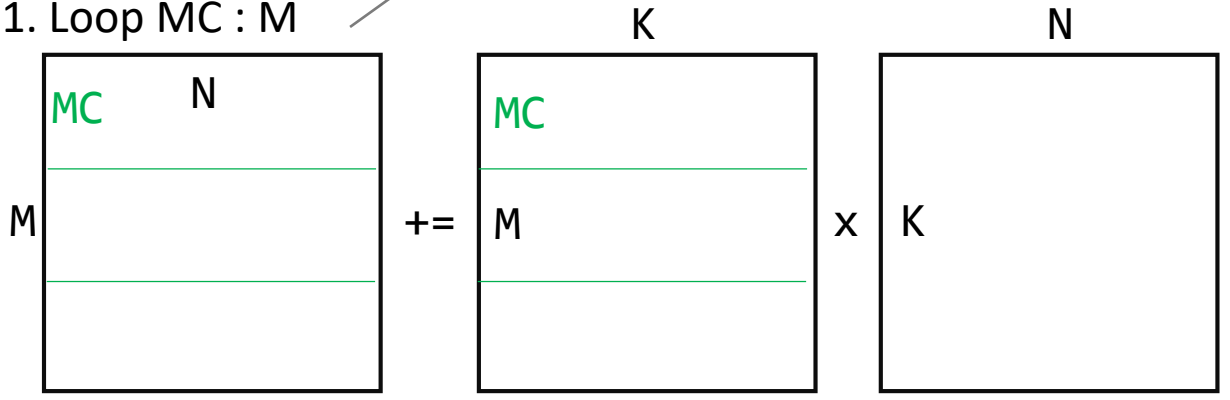


3. Loop MC : M (GEPB, in GOTO)

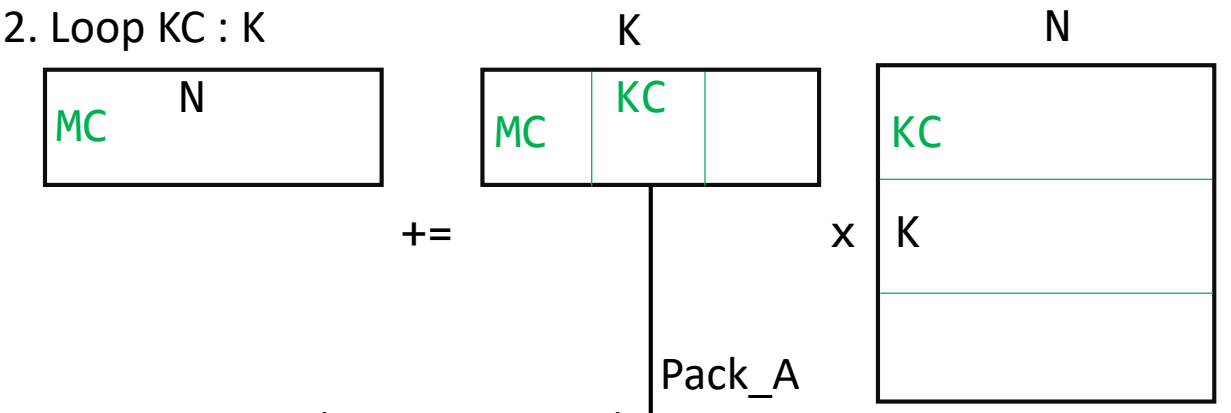


Loop C row by row

1. Loop MC : M

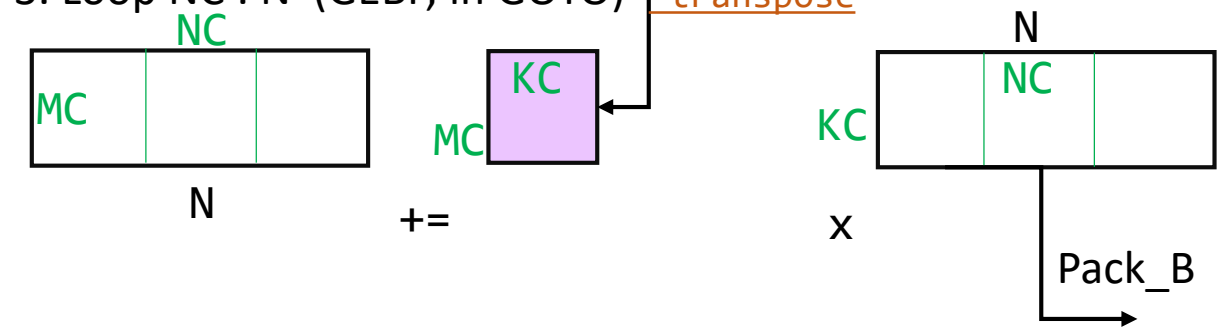


2. Loop KC : K



3. Loop NC : N (GEBP, in GOTO)

*transpose

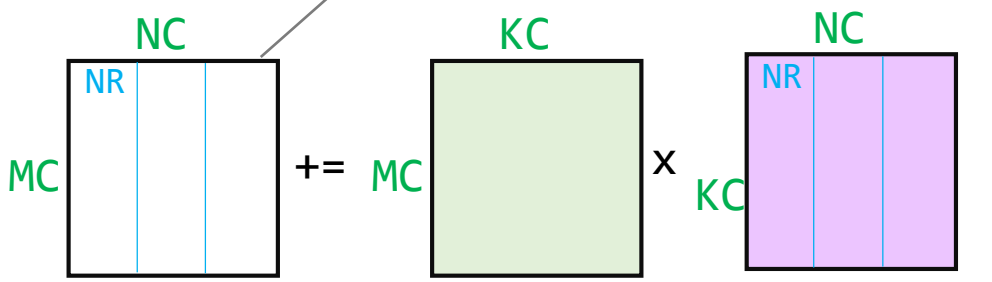


col major

Macro Kernel

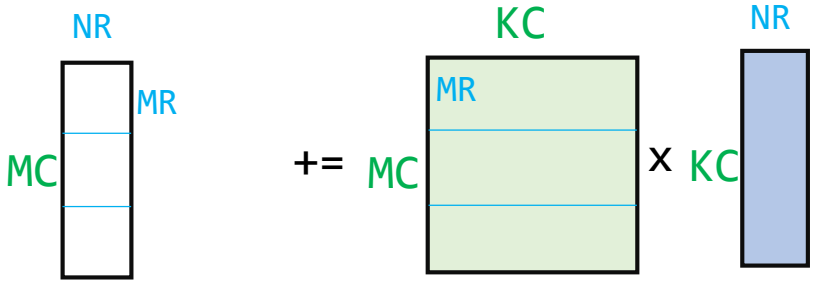
Loop C col by col

4. Loop NR : NC

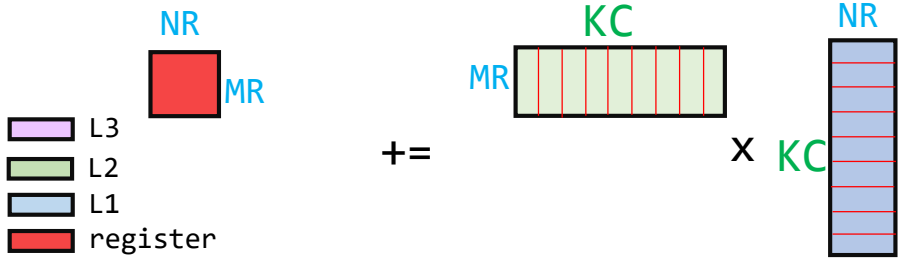


*transposed
-> row major

5. Loop MR : MC



Micro Kernel

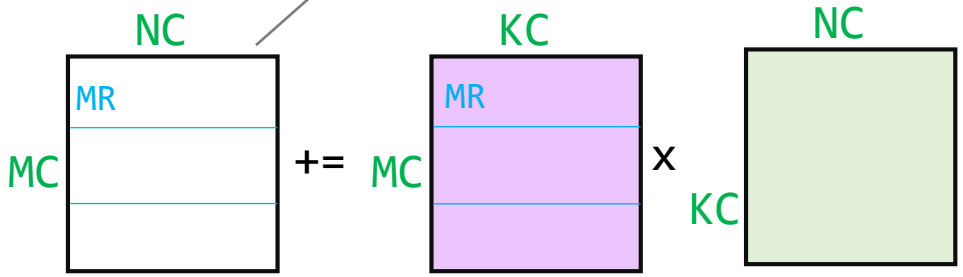


row major

Macro Kernel

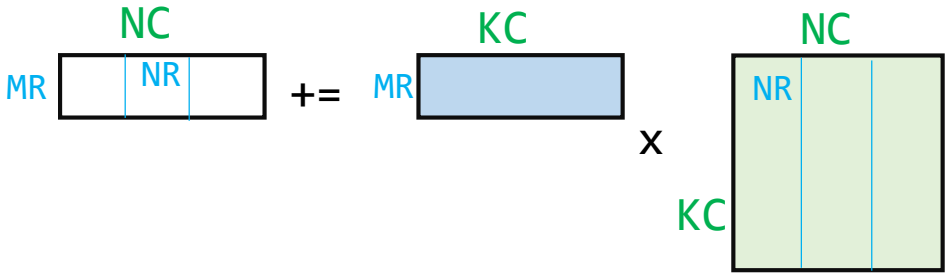
Loop C row by row

4. Loop MR : MC

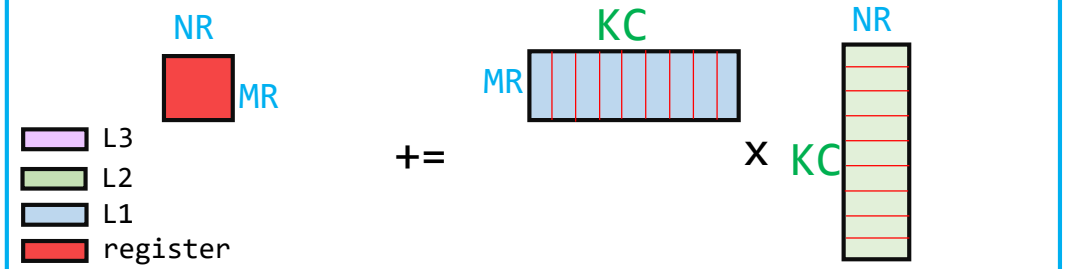


*transposed
-> col major

5. Loop NR : NC



Micro Kernel



col major

L1 consideration:

$$MR*KC + NR*KC + MR*NR + MR + MR*NR < L1\_size/(dtype\_size)$$

L2 consideration:

$$MC*KC + NR*KC + MC*NR + NR*KC + MC*NR < L2\_size/(dtype\_size)$$

L3 consideration:

$$MC*KC + NC*KC + MC*NC + NC*KC + MC*NC < L3\_size/(dtype\_size)$$

L1D TLB consideration:

$$2*TA + TB + TC < T\_entry\_total$$

TA: CEIL(MR*KC*d_size/PAGE_SIZE)+1
TB: CEIL(NR*KC*d_size/PAGE_SIZE)+1
TC: up to MR (assume ldc is every big)

row major

L1 consideration:

$$MR*KC + NR*KC + MR*NR + NR + MR*NR < L1\_size/(dtype\_size)$$

L2 consideration:

$$NC*KC + MR*KC + MR*NC + MR*KC + MR*NC < L2\_size/(dtype\_size)$$

L3 consideration:

$$MC*KC + NC*KC + MC*NC + NC*KC + MC*NC < L3\_size/(dtype\_size)$$

L1D TLB consideration:

$$TA + 2*TB + TC < T\_entry\_total$$

TA: CEIL(MR*KC*d_size/PAGE_SIZE)+1
TB: CEIL(NR*KC*d_size/PAGE_SIZE)+1
TC: up to MR (assume ldc is every big)

Further work:

1. Finetune difference MC/KC/NC… parameters for different M/K/C
2. Consider different micro kernel
3. Multiple CPU support
4. Different architecture support, like AMD Zen.