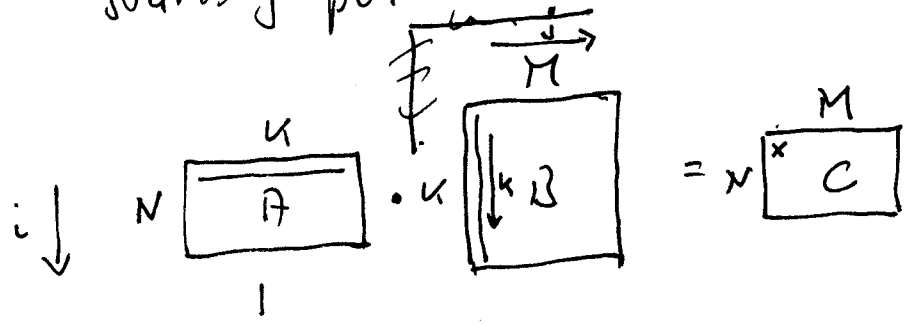
MMM - code generation with ATLAS

starting point: standard triple loop




```
for i = 0:1:N-1
    for j = 0:1:M-1
        c_ij = 0
        for k = 0:1:K-1
            C_ij = C_ij + A_ik * B_kj
```

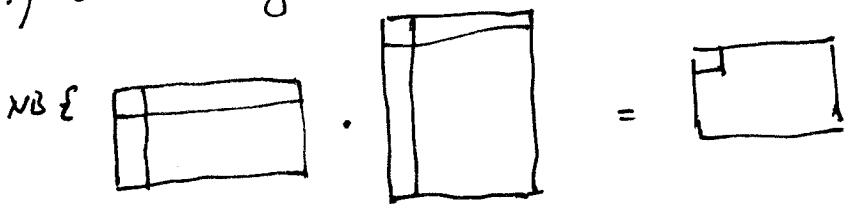
output locality } cost = $2NMK$!

1.) loop order ijk or jik?

ijk: B is reused
jik: A is reused

⇒ for locality: if $M < N$: ijk
if $N < M$: jik

2.) blocking for cache into square $NB \times NB$ blocks, for cache factor



assuming $NB \mid N, M, K$

```
for i = 0 : NB : N-1
    for j = 0 : NB : M-1
        for k = 0 : NB : K-1
            for i' = i : 1 : i+NB-1
                for j' = j : 1 : j+NB-1
                    for k' = k : 1 : k+NB-1
                        C_i'j' = C_i'j' + A_i'k' * B_k'j'
```

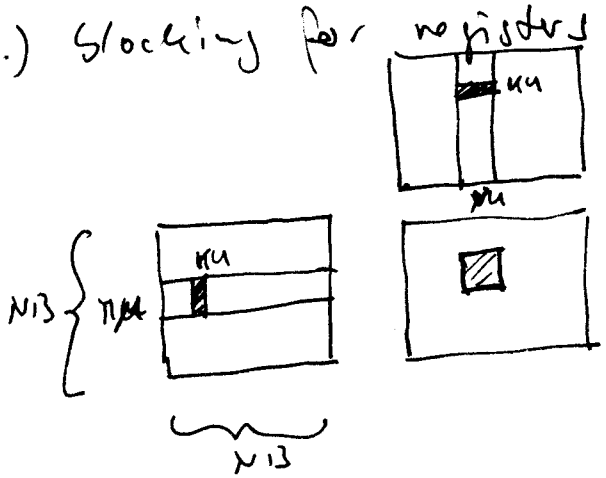
mini-MMM

⇒ for locality

~~NB is search parameter~~

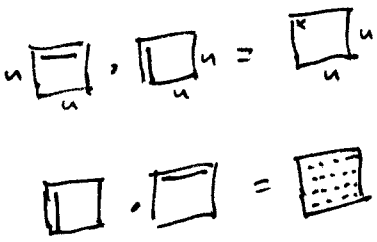
N_B search parameter

3.) Blocking for registers



now: - ij order
⇒ no output locality
⇒ but better instruction level parallelism

explain instruction level parallelism:



consecutive
$2u-1$ instructions:
- u parallel mults
- u-1 dependent adds (at least $\log_2(u-1)$ steps)

$2u^2$ instructions
- u^2 parallel mults
- u^2 parallel adds

but: $2u+1$ live vars
vs. u^2+2u live vars ← register pressure
" larger working set "

need: $u^2+2u \le N_R$ (# registers)

code:

```
for i = 0 : NB : N-1
  for j = 0 : MB : M-1
    for k = 0 : KB : K-1
      for i' = 0 : MU : i+NB-1
        for j' = 0 : NU : j+MB-1
          for k' = 0 : KU : k+KB-1
            for i'' = i' : 1 : i' + MU - 1
              for j'' = j' : 1 : j'' + NU - 1
                C[i'',j''] = C[i'',j''] + A[i'',k'] B[k',j'']
```

micro-MMM

unroll + scalar replacement

necessary: $M_u \cdot N_u + M_u + N_u \le N_R$ measured

M_u, N_u search parameters

unrolling 1: i", j" loop
- reduce loop overhead
- scalar replacement
  - enables register allocation + other optimn.
  - do not reuse variable names (artificial dependencies)

4.) Instruction interleaving/scheduling
- adds/mults alternately

$c_{ij} = c_{ij} + a_{ik} * b_{kj}$

↪ $t = a_{ik} * b_{kj}$
$c_{ij} = c_{ij} + t$ > dependency, problem with fmult latency
solution: skewing
increases register pressure

5.) unrolling 2: k" loop (U_K) is search parameter
- limited by size I-cache