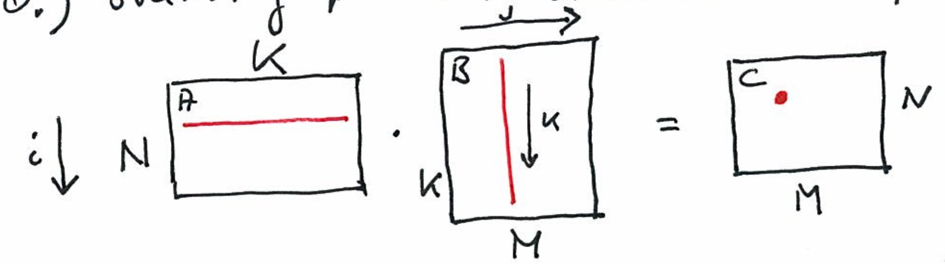
How to optimize MMM

The presentation follows the way the program generator ATLAS does it. ATLAS identifies optimization parameters (e.g., the blocking size) and uses search to find the best choices. Alternatively, a model can be used to determine each parameter (see paper on website). We discuss both.

0.) starting point: standard triple loop

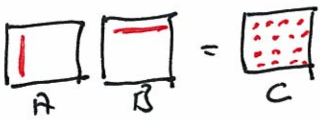


$$\left.\begin{array}{l} \text{for } i = 0 : N-1 \\ \quad \text{for } j = 0 : M-1 \\ \quad\quad \text{for } k = 0 : K-1 \\ \quad\quad\quad c_{ij} = c_{ij} + a_{ik} b_{kj} \end{array}\right\} \text{computes } C = C + AB$$

Important cases (from most to least): based on usage in LAPACK
- two out of N, K, M are small
- one "
- none "

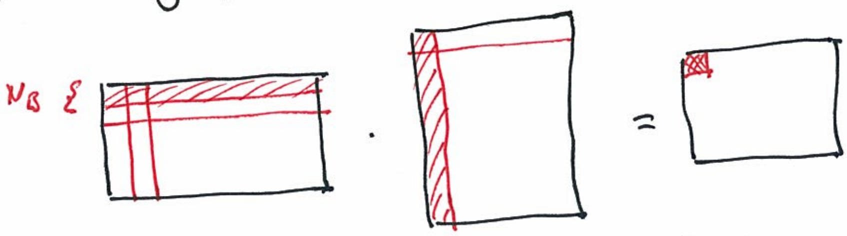
1.) Loop order: i, j, k loops can be permuted into any order
- i, j, k: B is reused, good if $M < N$
- j, i, k: A " , " $N < M$

ATLAS includes versions for both
- other choices are bad, e.g., k, i, j:



poor temporal locality w.r.t. C

2.) Blocking for cache



assume $N_B \mid N, K, M$

results in a six-fold nested loop

$$
\begin{array}{l}
\text{for } i = 0 : N_B : N-1 \\
\quad \text{for } j = 0 : N_B : M-1 \\
\quad\quad \text{for } k = 0 : N_B : K-1 \\
\quad\quad\quad \text{for } i' = i : i+N_B-1 \\
\quad\quad\quad\quad \text{for } j' = j : j+N_B-1 \\
\quad\quad\quad\quad\quad \text{for } k' = k : k+N_B-1 \\
\quad\quad\quad\quad\quad\quad c_{i'j'} = c_{i'j'} + a_{i'k'} b_{k'j'}
\end{array}
$$

Formally obtained from the triple loop through loop tiling & loop exchange

mini-MMM multiplies $N_B \times N_B$ blocks

ATLAS: uses search to find best N_B

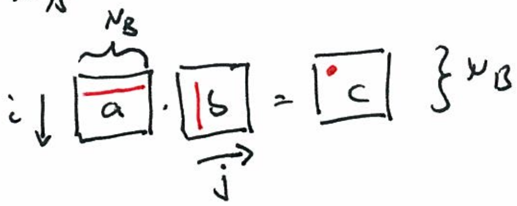
bound: $N_B^2 \leq C_1$ (cache size)

Model: explained next, model refined in steps

a.) Idea: working set has to fit in cache

easy bound: $|\text{working set}| = 3N_B^2$

$\Longrightarrow 3N_B^2 \leq C_1$



b.) Closer analysis:

$$\underbrace{N_B^2}_{\text{all of } b} + \underbrace{N_B}_{\text{row of } a} + \underbrace{1}_{\text{element of } c} \leq C_1$$

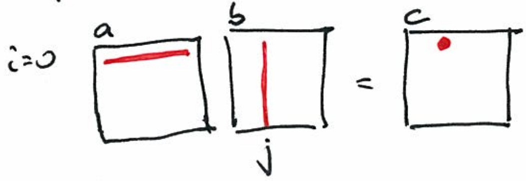
c.) Take into account cache block size B_1

$$\left\lceil \frac{N_B^2}{B_1} \right\rceil + \left\lceil \frac{N_B}{B_1} \right\rceil + 1 \leq \frac{C_1}{B_1}$$

(this just translates b.) into cache block units)

d.) Take into account LRU replacement

→ build a history of elements being accessed



$i=0$: $a_{00}\, b_{00} \, \ldots \, a_{0N_B-1}\, b_{N_B-1\,0} \;\; c_{00} \;\; (j=0)$

$a_{00}\, b_{01} \, \ldots \, {''} \;\; b_{N_B-1\,1} \;\; c_{01} \;\; (j=1)$

$\ldots \qquad \ldots$

$a_{00}\, b_{0N_B-1} \, \ldots \, {''} \;\; b_{N_B-1\,N_B-1} \;\; c_{0N_B-1} \;\; (j=N_B-1)$

corresponding history:

$b_{00} \, \ldots \, b_{N_B-1\,0} \;\; c_{00}$

$b_{01} \, \ldots \, b_{N_B-1\,1} \;\; c_{01}$

$\ldots$

$a_{00}\, b_{0N_B-1} \, \ldots \, a_{0N_B-1}\, b_{N_B-1\,N_B-1} \;\; c_{0N_B-1}$

more recent

Observations:

- all of b has to fit into cache for next iteration $i = 1$
- when $i = 1$, row 1 of a will not cleanly replace row 0 of a
- when $i = 1$, element of c will not cleanly replace previous elements of c

$\Rightarrow$ This has to fit:

- entire b
- 2 rows of a (here: a_{0*}, a_{1*})
- 1 row of c (here: c_{0*})
- 1 element of c (here: c_{10})

$$\Rightarrow \left\lceil \frac{N_B^2}{B_1} \right\rceil + 3\left\lceil \frac{N_B}{B_1} \right\rceil + 1 \le \frac{C_1}{B_1}$$

e.) Take into account blocking for registers (next opt.)

$$\left\lceil \frac{N_B^2}{B_1} \right\rceil + 3\left\lceil \frac{N_B M_u}{B_1} \right\rceil + \left\lceil \frac{M_u N_u}{B_1} \right\rceil \le \frac{C_1}{B_1}$$

Pick largest N_B that satisfies, shrink so $M_u, N_u \mid N_B$ (avoids clean-up code)

3.) Blocking mini-MMM for registers into micro-MMMs

revisit the question of loop order:

ijk:

for fixed i, j:
$2u$ instructions
- u independent mults
- u dependent adds ($\geq \log_2(u)$ steps)

kij:

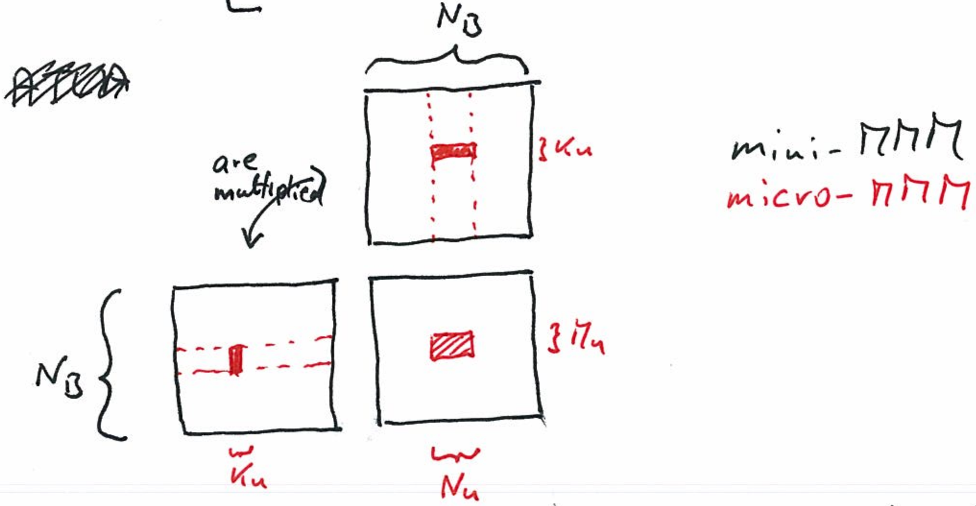
for fixed k:
$2u^2$ instructions
- u^2 independent mults
- u^2 " adds

$\Rightarrow$ good ILP (but larger working set)

Result: micro-MMMs with kij loop order for ILP

Code:

```
for i = 0 : N_B : N-1
  for j = 0 : N_B : M-1
    for k = 0 : N_B : K-1
      for i' = i : M_u : i+N_B-1                    } mini-MMM
        for j' = j : N_u : j+N_B-1
          for k' = k : K_u : k+N_B-1          <--- (1)
            for k'' = k' : k'+K_u-1           <--- (2)     } micro-MMM
              for i'' = i' : i'+M_u-1                       } (X)
                for j'' = j' : j'+N_u-1
                  c_{i''j''} = c_{i''j''} + a_{i''k''} b_{k''j''}
```



ATLAS: Uses search to find best M_u, N_u, K_u

Bound: $\underbrace{M_u + N_u + M_u N_u}_{\substack{\text{size of working set in } \otimes \\ = \text{no. of live variables}}} \leq N_R$ (#registers)

Model: Use largest M_u, N_u that satisfy this equation and $M_u \approx N_u$

4.) Basic block optimizations

step 1: unroll micro-MMM

c... = c... + a... b...
c... = c... + a... b...
⋮

step 2: scalar replacement
(all elements a..., b..., c... are reused)

$$
\left.\begin{array}{l}
t_0 = c\ldots \\
t_1 = a\ldots \\
t_2 = b\ldots \\
\vdots
\end{array}\right\} \text{loads}
$$

$$
\left.\begin{array}{l}
t_3 = t_0 + t_1 t_2 \\
\vdots
\end{array}\right\} \text{computation}
$$

$$
\left.\begin{array}{l}
c\ldots = t_3 \\
\vdots
\end{array}\right\} \text{stores}
$$

Note: The loads from c... ($M_u N_u$ many) are done at 1 in the above code.

The loads from a... and b... ($M_u + N_u$ many) are done at 2 in the above code

$\Rightarrow M_u + N_u + M_u N_u$ scalar variables

5.) Other optimizations

a.) "skewing": Separating dependent adds - mults for better ILP

b.) Software pipelining: Move loads from one iteration to previous iteration (to hide load latency)

c.) Buffering to avoid TLB misses (later)