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 or website). We discuss both.

0.) Starting point: standard triple loop

\[
\begin{align*}
N & \quad K \\
\downarrow & \quad K \\
M & \quad N
\end{align*}
\]

\[
\begin{align*}
\text{for } & \quad i = 0 : N-1 \\
\text{for } & \quad j = 0 : M-1 \\
\text{for } & \quad k = 0 : K-1 \\
& \quad c_{ij} = c_{ij} + a_{ik} b_{kj}
\end{align*}
\]

Important cases (from most to least): Based on usage in loops
- 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., \(k, i, j\):

\[
\begin{array}{ccc}
A & B & C \\
\hline
1 & 1 & 1 \\
\end{array}
\]

poor temporal locality w.r.t. \(C\)

2.) Blocking for cache

\[
\begin{align*}
N_{B} & \quad E \\
\downarrow & \quad E \\
N_{B} & \quad N_{B}
\end{align*}
\]

\[
\begin{align*}
\text{result in a six-fold nested loop}
\end{align*}
\]

assume \(N_{B} \mid N, K, M\)
for i = 0 : \( N_b : N - 1 \)
for j = 0 : \( N_b : N - 1 \)
for k = 0 : \( N_b : N - 1 \)
for i' = \( i + N_b - 1 \)
for j' = \( j + N_b - 1 \)
for k' = \( k + N_b - 1 \)
\[ C_{i',j',k'} = C_{i',j',k'} + a_{i',j',k'} \]

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

\[ \text{min}_i \text{min}_j \text{min}_k \]
\[ \text{multiply} \]
\[ N_b \times N_b \text{ stocks} \]

**ATLAS**: uses search to find best \( N_b \)

sound: \( N_b^2 \leq C_i \) (cache law)

**Model**: explained next, model refined in steps

a.) Idea: working set has to fit in cache
easy bound: \( |\text{working set}| = 3N_b^2 \)
\[ \Rightarrow 3N_b^2 \leq C_i \]

\[ \frac{N_b}{B_1} \cdot \frac{N_b}{B_1} \] and \( \frac{N_b}{B_2} \cdot \frac{N_b}{B_2} \)

b.) Closer analysis:
\[ N_b + N_b + 1 \leq C_i \]
\[ \forall a \text{ of } \begin{bmatrix} b \end{bmatrix} \text{ of } c \]

c.) Take into account cache block size \( B \):

\[ \left\lfloor \frac{N_b^2}{B_1} \right\rfloor + \left\lfloor \frac{N_b^2}{B_2} \right\rfloor + 1 \leq \frac{C_i}{B_i} \]

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

d.) Take into account LRU replacement

build a history of elements being accessed

\[ \begin{bmatrix} a & b \end{bmatrix} = \begin{bmatrix} c \end{bmatrix} \]

\[ i = 0 \]
\[ a_{00} b_{00} \ldots a_{N_b - 1} b_{N_b - 1} c_{00} \]
\[ (i = 0) \]
\[ a_{00} b_{01} \ldots a_{N_b - 1} b_{N_b - 1} c_{01} \]
\[ (i = 1) \]
\[ \vdots \]
\[ a_{00} b_{0N_b - 1} \ldots a_{N_b - 1} b_{N_b - 1} c_{0N_b - 1} \]
\[ (j = N_b - 1) \]

Corresponding history:
\[ b_{00} \ldots b_{N_b - 1} c_{00} \]
\[ b_{01} \ldots b_{N_b - 1} c_{01} \]
\[ \vdots \]
\[ b_{0N_b - 1} \ldots a_{N_b - 1} c_{0N_b - 1} \]

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 \( c \)

\[ \Rightarrow \text{This has to fit:} \]
- Entire \( b \)
- 2 rows of \( a \) (here: \( a_{0x}, a_{1x} \))
- 1 row of \( c \) (here: \( c_{0x} \))
- 1 element of \( c \) (here: \( c_{10} \))

\[ \Rightarrow \left[ \frac{N_{13}^2}{B_1} \right] + 3 \left[ \frac{M_{13}^3}{B_1} \right] + 1 \leq \frac{C_i}{B_1} \]

E.) Take into account blocking for registers (next opt.)
\[ \left[ \frac{N_{13}^2}{B_1} \right] + 3 \left[ \frac{M_{13}^3}{B_1} \right] + \left[ \frac{M_{13} M_{15}}{B_1} \right] \leq \frac{C_i}{B_1} \]

Pick largest \( N_{13} \) that satisfies, shrink so \( M_{13}, M_{15} / N_{13} \)
(avoides clean-up code)

3.) Blocking mini: \( M_{13} / 7 \) for registers into micro-\( M_{13} / 7 \)
revisit the question of loop order:

\[ ijk: \quad \]

for fixed \( i,j \):
- \( 2n \) instructions
- \( n \) independent multiplies
- \( n \) dependent adds
\( (\geq \log_2(n)) \) steps

\[ kij: \quad \]

for fixed \( k \):
- \( 2n^2 \) instructions
- \( n^2 \) independent multiplies
- \( n^2 \) adds
\( \Rightarrow \) good ILP (but Flanger working set)

Result: micro-\( M_{13} / 7 \) wih kij loop order for ILP
Code:

```
for \( i = 0 : \text{N}_{3} = \text{N}-1 \)
for \( j = 0 : \text{N}_{3} = \text{N}-1 \)
for \( k = 0 : \text{N}_{3} = \text{K}-1 \)
for \( i' = i : \text{M}_{i} = i + \text{N}_{3} - 1 \)
for \( j' = j : \text{N}_{i} = j + \text{N}_{3} - 1 \)
for \( k' = k : \text{K}_{i} = k + \text{N}_{3} - 1 \)
for \( i'' = i' : i'' = i' + \text{N}_{i} - 1 \)
for \( j'' = j' : j'' = j' + \text{N}_{i} - 1 \)
\( C_{i''j''} = C_{i'j'} + a_{i''k''} b_{i'j'} \)
```

**ATLAS**: Uses search to find best \( \text{Nu}, \text{Nu}, \text{Nu} \)

**Sound**: \( \text{Nu} + \text{Nu} + \text{Nu} \text{Nu} \leq \text{N}_{R} \) (# registers)

\( \text{size of working set in } \Theta \)

\( \text{no. of live variables} \)

**Model**: Use largest \( \text{Nu}, \text{Nu} \) that satisfies this equation

and \( \text{Nu} \leq \text{Nu} \)

4. Basic block optimizations

   **step 1**: unroll micro-\( \text{HLM} \)
   \( c_{...} = c_{...} + a_{...} b_{...} \)
   \( c_{...} = c_{...} + a_{...} s_{...} \)

   **step 2**: scalar replacement
   (all elements \( a_{...}, s_{...}, c_{...} \) are reused)
\[ \begin{aligned} \text{loa} & \quad \text{d} \\
\text{c} & = c \\
\text{y} & = a \\
\text{y} & = s \\
\vdots & \\
\text{y} & = t_0 + t_1 \text{c} \\
\vdots & \\
\text{c} & = \text{y} \\
\end{aligned} \]

Note: The loads from \( c \ldots \) (\( Nu \) many) are done at \( 1 \) in the above code.

The loads from \( a \ldots \) and \( s \ldots \) (\( Nu \) many) are done at \( 2 \) in the above code.

\( \Rightarrow \) \( Nu + Nu + Nu Nu \) scalar variables

5.) Other optimizations

a.) "skewing": separating dependent adds - multiplies for better ILP

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

c.) Buffering to avoid TCB misses (later)