## SESSION 7

# Matrix–matrix multiplication


MASSIMILIANO FASI

Durham University

# Matrix–matrix multiplication

Given n-by-n matrices A and B, we can compute $C = AB$

$$C_{ij} \leftarrow C_{ij} + \sum_k A_{ik} B_{kj}$$

with

```
for (int i = 0; i < n; i++)
  for (int j = 0; j < n; j++)
    for (int k = 0; k < n; k++)
      C[i*n + j] += A[i*n + k] * B[k*n + j];
```

# Two-level memory model ("fast" and "slow")

- $m$: # data elements moved
- $f$: # flops
- $t_m$: time per memory access
- $t_f \ll t_m$ time per flop

$q =: f/m$ average flops per slow memory access

hardware software

# Two-level memory model (“fast” and “slow”)

- $m$: # data elements moved
- $f$: # flops
- $t_m$: time per memory access
- $t_f \ll t_m$ time per flop

$q =: f/m$ average flops per slow memory access

Minimum time to solution

$$t_f \cdot f$$

hardware software

# Two-level memory model ("fast" and "slow")

- $m$: # data elements moved
- $f$: # flops
- $t_m$: time per memory access
- $t_f \ll t_m$ time per flop

$q =: f/m$ average flops per slow memory access

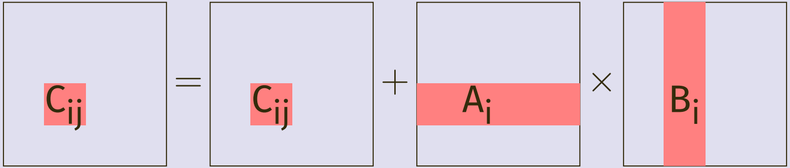
| Minimum time to solution | Typical time to solution |
| --- | --- |
| $t_f \cdot f$ | $f\,t_f + m\,t_m = f\,t_f \left(1 + \frac{t_m}{t_f} \cdot \frac{1}{q}\right)$ |

hardware software

# Naïve matrix-multiply

```
for (int i = 0; i < n; i++)
  for (int j = 0; j < n; j++)
    for (int k = 0; k < n; k++)
      C[i*n + j] = C[i*n + j] + A[i*n + k] * B[k*n + j];
```

- $2n^3 = \mathcal{O}(n^3)$ flops and $3 \cdot 8n^2$ bytes of memory
- q potentially $\mathcal{O}(n)$, arbitrarily large for large n



# Naïve matrix-multiply

```
for (int i = 0; i < n; i++)
  // Read row i of A into fast memory
  for (int j = 0; j < n; j++)
    // Read Cij into fast memory
    // Read column j of B into fast memory
    for (int k = 0; k < n; k++)
      C[i*n + j] = C[i*n + j] + A[i*n + k] * B[k*n + j];
      // Write Cij back to slow memory
```

# Number of slow memory references

$$
\begin{aligned}
m &= n^3 && \text{each column of B is read n times} \\
&\quad + n^2 && \text{each row of A is read once} \\
&\quad + 2n^2 && \text{each entry of C is read once and written once} \\
&= (n^3 + 3n^2)
\end{aligned}
$$

Hence

$$\lim_{n\to\infty} q = \frac{f}{m} = \frac{2n^3}{(n^3 + 3n^2)} = 2$$

# Two-level memory model (“fast” and “slow”)

- $m$: # data elements moved
- $f$: # flops
- $t_m$: time per memory access
- $t_f \ll t_m$ time per flop

$q =: f/m$ average flops per slow memory access

| Minimum time to solution | Typical time to solution |
| --- | --- |
| $t_f \cdot f$ | $f\, t_f + m\, t_m = f\, t_f \left(1 + \frac{t_m}{t_f} \cdot \frac{1}{q}\right)$ |

hardware software

# From model to prediction

- For three nested loops, predicted *typical* time to solution is:

$$T = f\, t_f \left(1 + \frac{t_m}{2\, t_f}\right)$$

# From model to prediction

- For three nested loops, predicted *typical* time to solution is:

$$T = f\, t_f \left(1 + \frac{t_m}{2\, t_f}\right)$$

- Memory *latency* of about 200 cycles per cache line (8 doubles)

# From model to prediction

- For three nested loops, predicted *typical* time to solution is:

$$T = f\, t_f \left(1 + \frac{t_m}{2\, t_f}\right)$$

- Memory *latency* of about 200 cycles per cache line (8 doubles)
- Approximating $t_m \approx 200/8 = 25$, and say $t_f = 1$:

$$T = f\, t_f \left(1 + \frac{25}{2}\right) = 13.5\, f\, t_f$$

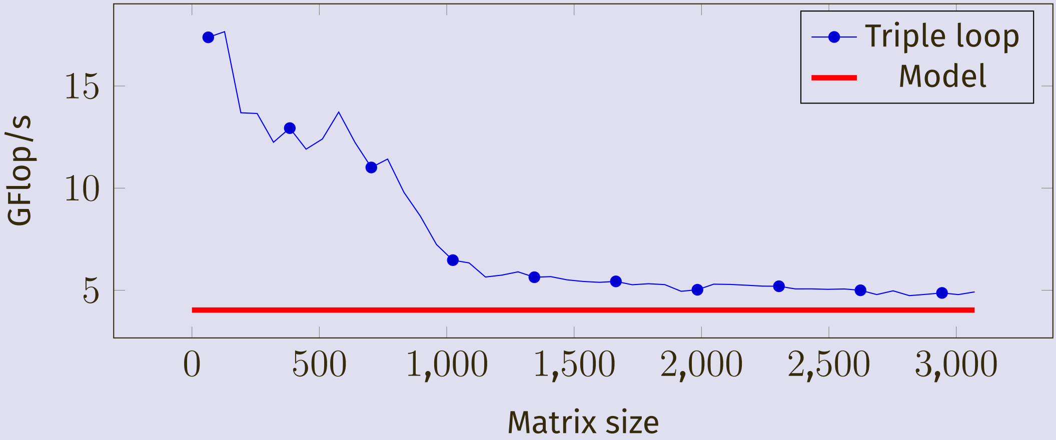
- **Estimate:** Maximum 7% flop peak.

# Measurement

- 2 4-wide FMAs per cycle $\Rightarrow$ 16 DP Flops/cycle
- Peak is $3.6 \cdot 16 = 57.6$ GFlops/s $\Rightarrow$ model predicts 4.03 GFlops/s

# Measurement

- 2 4-wide FMAs per cycle $\Rightarrow$ 16 DP Flops/cycle
- Peak is $3.6 \cdot 16 = 57.6$ GFlops/s $\Rightarrow$ model predicts 4.03 GFlops/s



# How to improve reuse?

```
// Treat $A, B, C \in \left(\mathbb{R}^{b \times b}\right)^{N \times N}$
// $N \times N$ matrices where each entry is a $b \times b$ matrix.
for (int i = 0; i < N; i++)
  for (int j = 0; j < N; j++)
    // Read block $C_{ij}$ into fast memory
    for (int k = 0; k < n; k++)
      // Read block $A_{ik}$ into fast memory
      // Read block $B_{kj}$ into fast memory
      // Do matrix multiply on the blocks
      C[i*N + j] = C[i*N + j] + A[i*N + k] * B[k*N + j];
      // Write block $C_{ij}$ back to slow memory
```

# How to improve reuse?

```
// Treat A, B, C ∈ (ℝ^(b×b))^(N×N)
// N × N matrices where each entry is a b × b matrix.
for (int ii = 0; ii < N; ii++)
  for (int jj = 0; jj < N; jj++)
    for (int kk = 0; kk < N; kk++)
      for (int i_ = 0; i_ < b; i_++)
        for (int j_ = 0; j_ < b; j_++)
          for (int k_ = 0; k_ < b; k_++) {
             const int i = ii*b + i_;
             const int j = jj*b + j_;
             const int k = kk*b + k_;
             C[i*n + j] = C[i*n + j] + A[i*n + k] * B[k*n + j];
          }
```

# What did that do to the data movement?

$$\begin{aligned}
m &= Nn^2 && \text{each block of B is read } N^3 \text{ times} \\
&+ Nn^2 && \text{each block of A is read } N^3 \text{ times} \\
&+ 2n^2 && \text{each block of C is read once and written once} \\
&= 2n^2(N+1)
\end{aligned}$$

Hence

$$\lim_{n\to\infty} q = \frac{f}{m} = \frac{2n^3}{2n^2(N+1)} \approx \frac{n}{N} = b \gg 2$$

# What did that do to the data movement?

$$\begin{aligned} m &= Nn^2 && \text{each block of B is read } N^3 \text{ times} \\ &+ Nn^2 && \text{each block of A is read } N^3 \text{ times} \\ &+ 2n^2 && \text{each block of C is read once and written once} \\ &= 2n^2(N+1) \end{aligned}$$

Hence

$$\lim_{n\to\infty} q = \frac{f}{m} = \frac{2n^3}{2n^2(N+1)} \approx \frac{n}{N} = b \gg 2$$

**Limit on** $b$: still must still fit in fast memory.

# From model to machine characteristics

- If algorithm is “fast” when $T \geq 50\%$ peak, then

$$f\, t_f \left(1 + \frac{t_m}{t_f}\frac{1}{q}\right) \leq 2\, f\, t_f \Leftrightarrow \frac{t_m}{t_f}\frac{1}{q} \leq 1 \Leftrightarrow q \geq \frac{t_m}{t_f}$$

- For $t_m = 25$, $t_f = 1 \Rightarrow b \approx q \geq 25$.
- To hold all three $b \times b$ matrices in cache we need

$$3b^2 = 3 \cdot 25^2 = 1875 \text{ doubles } \approx 14.6\text{KB of fast memory}$$

- This is smaller than L1, but too large for registers.

# Is this the best we can do?

## Hong and Kung (1981)

Any reorganization of this algorithm using only *associativity* has

$$q = \mathcal{O}(\sqrt{M_{\mathrm{fast}}})$$

and the number of data elements moved (slow $\Leftrightarrow$ fast) is

$$\Omega\left(\frac{n^3}{\sqrt{M_{\mathrm{fast}}}}\right)$$

- Exact values for the bounds are not known
- Best bounds by Smith and van de Geijn (2017)
- GotoBLAS/OpenBLAS approach approaches these bounds

# Exercise 8: tiled matrix–matrix multiplication

1. Split into small groups
2. Download the code (three versions, one source)
3. Measure Flop rate as matrix size changes
4. Try different tile sizes
5. Ask questions!