Ma221 Lecture 4 Segment 1

Goal: understand real cost (time) of running an algorithm, needed to design fast algorithms.

Traditionally: count # arithmetic ops (flops) but these are cheapest operations: it can be orders of magnitude more expensive to move data from wherever it is stored (main memory) and bring it to part that does arithmetic (CPU).

```
<table>
<thead>
<tr>
<th>CPU</th>
<th>CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>move data</td>
<td>fast</td>
</tr>
<tr>
<td>main mem</td>
<td>cache</td>
</tr>
<tr>
<td></td>
<td>slow</td>
</tr>
<tr>
<td></td>
<td>main mem</td>
</tr>
<tr>
<td></td>
<td>DRAM</td>
</tr>
</tbody>
</table>

minimize slow memory movement (DRAM → cache)
Notation: all data movement is called "communication".

Matrix Multiplication (Matmul) Theorem giving a lower bound on communication required between DRAM and cache, assuming usual $2n^3$ flops, but done in any order (Hong, Kung 1981). There is a widely known and widely used algorithm that attains this bound.
In 2011 we showed this lower bound extends to any algorithm that "smells like" 3-nested loops of matmul, covers Gaussian elim, least squares, eig, ... (Ballard, Ho, Schwartz, D.)

Usual algorithms cannot attain lower bound, just by doing flops in a different order. Need new mathematical algorithms

Widely used, often much faster O(CDx).

 Extends (lower bounds and optimal algs) to other computer architectures, multiple layer of cache, parallel processors.

Here, discuss lower bound and optimal alg for matmul. Later, just sketch optimal algs for more complicated Linear Algebra problems.
Many possible class projects.

Need simple model of cost of moving data to analyze alg.
Need to define: bandwidth, latency.

For intuition, describe but latency for freeway connecting Berkeley to Sacramento.

Bandwidth measures cars/hour that can get from B to S.

\[
\text{#cars/hour} = \text{density} \times \text{velocity} \times \text{lanes}
\]

Latency measures how long it takes 1 car to get from B to S.

\[
\text{time (hours)} = \frac{\text{distance (miles)}}{\text{velocity (miles/hour)}}
\]
So minimum time for $n$ cars to go from $B$ to $S$: when all travel in single "convoy", as close as possible:

time (hours) = time for first car to arrive + time for remaining cars

= latency + $n$/bandwidth

Same idea for moving data on wires (harder physics)

time to move $w$ words from DRAM to cache = latency + $w$/bandwidth assuming all packed into one "message"

More generally, time to move $w$ words in $m$ messages = $m \cdot$ latency + $w$/bandwidth

Notation: write cost as $m \cdot \alpha + w \cdot \beta$

$w$ = # words moved $m$ = # messages

$\beta$ = time per flop
Total time of time = \( m \cdot \alpha + w \cdot B + f \cdot y \)

\( f = \# \text{flops} \)

Reiterate why communication important.

1. \( y \ll B \ll \alpha \) on current computers
   (factor of 10x or 100x)

2. Gaps are increasing over time
   all of \( y, B, \alpha \) improving
   but \( y \) faster than \( B, \) faster than \( \alpha \)
   (show figure of \( \alpha, B, y \))

Similar story for energy costs.

---

*Ma221 Lecture 4 Segment 2*

Time = \( m \cdot \alpha + w \cdot B + f \cdot y \)

\( m = \# \text{messages}, \ \alpha = \text{latency} \)

\( w = \# \text{words moved}, \ \beta = \text{bandwidth} \)

\( f = \# \text{flops} \)

\( y \ll B \ll \alpha \)
\[ f \geq \max + w/B \]
\[ \Leftrightarrow \text{flips dominate within a factor of 2 of optimal} \]
\[ f \gg \max + w/B \]
\[ \Leftrightarrow \text{near machine peak} \]
\[ f \ll \max + w/B \]
\[ \Leftrightarrow \text{communication dominates} \]

**Notation:** Computational Intensity
\[ q = \frac{f}{w} = \text{"flips per word moved"} \]

q needs to be large to be fast
\[ f \gg w/B \quad q = \frac{f}{w} > \frac{B}{f} \gg 1 \]

How has this impacted algorithm design? A little history:

In the beginning, was the do-loop. Enough for libraries like EISPACK (eigenproblems, mid 1960s):
People didn’t worry about communication:
goal: get right answer in \( O(n^3) \) flips
BLAS-1 library (Basic Linear Algebra Subprograms)

Standard library of 15 operations mostly on vectors.

1. \( y = \alpha \cdot x + \gamma \cdot y \) vectors, \( \alpha, \gamma \) scalars
   "AXPY" for short
   Example: innermost loop of Gaussian elimination:
   for \( k = l + 1 \) to \( n \)
   \( A(j, k) = A(j, k) - A(j, l) \cdot A(l, k) \)

2. Dot product

3. \( \|x\|_2 = (\sum_{i=1}^{n} x_i^2)^{1/2} \)

4. Find largest entry in magnitude in a vector (for pivoting in GE)

Motivations for BLAS-1:

- Easing programming, readability, since common functions
- Robustness (avoiding overflow/underflow in \( \|x\|_2 \))
- Portability and efficiency (if optimized on each architecture)

No way to minimize communication using BLAS-1 as building blocks
comp intensity \( g \) for dot product
\[
= \frac{2n}{2n} = 1
\]
⇒ if \( \expf \) is used to implement GE,
\( g \leq 1 \) for GE

BLAS2 library (mid-1980s)
standard library of 25 operations
(mostly) on matrix-vector pairs
(1) \( y = \alpha y + \beta A x \) \( \alpha, \beta \) vectors \( A \) matrix
(\"GEMV\")
\( y \) vectors \( \alpha, \beta \) scalars
lots of variations:
\( A \) symmetric, banded, triangular, ...
can multiply by \( A \) or \( A^T \)
obvious optimization if \( \alpha, \beta = \pm 5, \pm 50 \)
(2) \( A = A + \alpha \cdot x \cdot y^T \) (rank-1 update of \( A \), \"GER\")
2 inner most loops of GE can be done by 1 call to \( G \)-\( ER \) (details later)
\[
A(i+i',j+i') = A(i+i',i+i') - A(i+i',j+i') \cdot A(i,i+i')
\]
(3) Solve $TX=B$ where $T$ triangular

"TRSV", used by GE

Motivation: similar to BLAST:
+ more opportunities to optimize on vector computers

Still not much improvement in communication:

$n \times n$ GEMV $q = \frac{t}{w} = \frac{2n^2}{n^2 + n \times n} \sim 2$ not large

BLAS 3 library (late 1980s)

9 operations on matrix-matrix pairs

1. $C = \beta C + \alpha A \cdot B$ ("GEMM" for short)
2. $C = \beta C + \alpha A \cdot A^T$ ("SYRK")
3. Solve $TX = B$, $T$ triangular

$X, B$ rectangular

$q$ finally large

$n \times n$ for GEMM: $q = \frac{t}{w} = \frac{2n^3}{3n^2 \text{ inputs} + n^2 \text{ outputs}} = \frac{n}{2}$
But straightforward, 3-nested loop version of GEMM are faster than BLAS1 or BLAS2—need a new algorithm.

Note: BLAS-k does $O(n^k)$ operations on inputs of dimension $n$.

Also supplied as optimized libraries on all platforms (e.g. MKL on Intel): use them as your building blocks!

documentation: www.netlib.org/blas

BLAS3 led community to design new algorithms for solving $Ax = b$, least squares, eig, SVD... that did as much work as possible using BLAS3. This led to LAPACK, later parallel version called ScalAPACK.
Later we learned that lower bound for matmul applied to these other Linear Algebra problems. But the algorithms in Sc/Lapack did not attain them in general. Set off search for new optimal algorithms. Some (argu speedups attained O(10x), but still work to do, constants matter!

Ma221 Lecture 4 Segment 3

Goal: Prove communication lower bound for matmul
This originally due to Hong, Kung 1981. We will present proof based on Irony, Tiskin, Tolosd, 2004 (generalizes to parallel case, 3-nested loops)

**Theorem:** To multiply \( C = A \cdot B \) all \( n \times n \) matrices using classical \( 2n^3 \) flops in any correct order, # words moved = \( \Omega \left( \frac{n^3}{V^2} \right) \)

Proof does not assume \( n \times n \) matmul, applies to rectangular, and sparse matmul:

\( \Omega \left( \frac{\# flops}{V^2} \right) \)

**Proof:** Sketch (not worry about constants)

Suppose we fill cache with \( M \) words, do as many flops as possible, store results back in DRAM. Suppose we could upper bound by \( G \) the number of flops possible using \( M \) words \( \Rightarrow \)
Doing $G$ flops costs at least $2M$ words moved. Since we do $2n^3$ flops total, we need to repeat at least $2n^3/G$ times. 
\[ \Rightarrow \text{Total # words moved } \geq \frac{2n^3}{G} \cdot 2M \]

Need an upper bound $G$. Geometry problem: 
\[ C(i,j) = C(i,j) + A(i,k) \cdot B(k,j) \]
represent this as a lattice point $(i,j,k)$ with $1 \leq i,j,k \leq n$ in a $n \times n \times n$ cube of lattice points.

How many lattice points can I have?

\[ V = \text{set of } (i,j,k) \]

To execute $V$, need $V_A, V_B, V_C$. Want to bound # entries in $V = |V|$ given $|V_A| \leq M$, $|V_B| \leq M$, $|V_C| \leq M$.
Thm (Loomis & Whitney, 1949)

\[ |V| \leq \sqrt[3]{|V_a| \cdot |V_b| \cdot |V_c|} \leq M^{\frac{3}{12}} = \Omega \]

# words moved \( \geq \frac{2v^3}{G} - 2M = \frac{2v^3}{M^{\frac{1}{12}}} - 2M = \Omega \left( \frac{v^3}{UM} \right) \)

QED

If careful with constants, get

# words moved \( \geq \frac{2v^3}{UM} - 2M \)

attainable!

---

Ma221 Lecture 4 Segment 4

Goal: Optimal Matrix Multiply

(attains comm. lower bound)

proof gives hint

what shape \( V \) has surface projection \( M \) in all 3 directions and volume \( M^{\frac{3}{12}} \)?

\( \text{cube}: \)

\[ \text{M}^{\frac{3}{2}} \]
Algorithm: break $A, B, C$, into submatrices that all fit in cache, as large as possible. Read each triple of submatrices into cache. Multiply submatrices. Put answer back in DRAM.

So that all 3 submatrices fit, we will pick them to be $\sqrt{\frac{M}{3}} \times \sqrt{\frac{M}{3}}$, $b = \sqrt{\frac{M}{3}}$

Notation: $A = \begin{bmatrix} b & b & b \\ b & b & b \\ b & b & b \end{bmatrix}$ $A[I, J]$ $b \times b$ block

$\frac{N}{b} = \# \text{ blocks}$
for $i = 1 \text{ to } n/b$
  for $j = 1 \text{ to } n/b$
    read $C[i,j]$ into cache ... $b^2$ words moved
    for $k = 1 \text{ to } n/b$
      read $A[i,k]$ and $B[k,j]$ into cache ... $2b^2$ words moved
      $C[i,j] = C[i,j] + A[i,k] \cdot B[k,j]$
      ... $b \times b$ matmul, all in cache
      ... 3 more nested loops
    endfor
  endfor
  write $C[i,j]$ back into memory ... $b^2$ words moved
endfor

Total words moved = $C \cdot (b)^2 \cdot b^2 = n^3$ ... for reading
  + $(n/b)^3 \cdot 2b^2 = 2n^3/b$ ... for reading $A, B$
  + $(n/b)^3 \cdot b^2 = n^3$ ... for writing $C$
  = $2n^3/b + 2n^2$

minimize by making $b$ as large as possible

$b = \sqrt[3]{n/3} \Rightarrow \text{ words moved} = O(\sqrt[n^3]{n/3})$
hitting lower bound.
"What about $A \cdot B$ when $B$ has a few columns, or just 1? Still 3 nested loops!"

If too few columns, can't break $B$ into $b \times b$ blocks. But all lower bounds and optimal algorithms extend to general loop bounds: $\Omega(\max(\frac{\text{flops}}{TM}, \text{size(input)} + \text{size(output)}))$.

This approach to finding a lower bound + optimal algorithm extends to any algorithm that can be written as nested loops (as many as you like) any number of arrays any "affine subscripts" $i, ij, i-2j+3k, \ldots$

Lower bound looks like $\Omega(\frac{\text{loop iterations}}{\sum})$ for natural

where $e$ depends on what code looks like (Hölder-Brascamp-Lieb inequality) - attainable by "blocking", general parallelograms
Another optimal matrix algorithm: "cache oblivious" because code does not depend explicitly on $M$, only $M$ for analysis.

```plaintext
function C = RMM(A, B)
    ... Recursive Matrix Multiply
    ... For simplicity assume A and B
    ... are square of size $n = 2^m$
    if $n = 1$
        C = A.B      ... scalar multiply
    else
        ... write $A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$ each $A_{ij}$ is $n/2 \times n/2$
        ... ditto for B and C
        \begin{align*}
        C_{11} &= RMM(A_{11}, B_{11}) + RMM(A_{12}, B_{21}) \\
        C_{12} &= \text{ditto} \\
        C_{21} &= \text{ditto} \\
        C_{22} &= \text{ditto}
        \end{align*}
    end if

Correctness by induction
```
Cost: $A(n) = \# \text{arithmetic ops on dimension } n$

$A(n) = 8A\left(\frac{n}{2}\right) ... 8 \text{ recursive calls} + n^2 \quad ... 4 \text{ additions of } \frac{n^3}{8} \text{ submatrices}$

Claim: $A(n) = 2n^3 - n^2 ... \text{ same as usual}$

Solve recurrence:

change variables $n = 2^m$ to $m$

$a(m) = 8a(m-1) + 2^{2m}$

divide by $8^m$

$a(m) = \frac{a(m-1)}{8^{m-1}} + 2^{-m}$

change to $b(m) = a(m)/8^m$

$b(m) = b(m-1) + 2^{-m} \quad b(0) = 1$

$= \sum_{k=1}^{m} 2^{-k} + b(0)$

$W(n) = \# \text{words moved}$

$W(n) = 8 \cdot W\left(\frac{n}{2}\right) ... 8 \text{ recursive calls} + (2\left(\frac{n}{2}\right))^2 \quad ... 8 \text{ reads, 4 writes of } \frac{n^3}{8} \text{ submatrices}$

$= 8W\left(\frac{n}{2}\right) + 3n^2 \quad \text{oops, look like more than } A(n)$
Base case not $W(1)$, rather

\[ W(b) = 3b^2 \text{ if } 3b^2 \leq M \]

\[ b = \sqrt[3]{M/3} \]

Solve as before: \( n = 2^m \) to \( m \):

\[ w(m) = 8w(m-1) + 3 \cdot 2^m \]

\[ \overline{m} = \log_2 \sqrt[3]{M/3} \] base case

divide by \( 8^m \) to get \( \nu(m) = w(m) / 8^m \)

\[ \nu(m) = \nu(m-1) + 3 \cdot (1/2)^m \]

\[ \nu(\overline{m}) = M / 8^{\overline{m}} = M / (1/3)^{3/2} = 3^{3/2} / M^{1/2} \]

Geometric sum: only done to \( m = \overline{m} \) not 1

\[ \nu(m) = \sum_{k=\overline{m}+1}^{m} 3(1/2)^k + \nu(\overline{m}) \]

\[ = 2 \cdot 3^{3/2} / \sqrt[3]{M} \]

\[ W(n) = w(\log_2 n) = 8^{\log_2 n} \nu(\log_2 n) = 2 \cdot 3^{3/2} \frac{n^3}{M} \]

as desired.

"Cache oblivious" works for any number of levels of cache of any size
Analysis extends to parallel case.

Each of \( P \) processors
store \( \frac{1}{P} \) of all data
does \( \frac{1}{P} \) of all work

"Fast memory" is memory local to each proc

"Slow memory" is on other procs

Same lower bound: \( \Omega \left( \frac{\#\text{flops per proc}}{\text{mem per proc}} \right) \)

\[
\Omega \left( \frac{n^3}{P} \right) = \Omega \left( \frac{n^2}{P} \right)
\]

Attainable by parallel alg (SUMMA)

topic in CS 267

Ma221 Lecture 4 Segment 5

One more topic: How to go asymptotically than \( O(n^3) \).
Reduce cost of rest of
dense linear algebra, communication.

Similarly: Many algorithms just discuss first one:

Strassen (1967) matrix possible
in $O(n^{\log_27})$ ops $\approx O(n^{2.81})$

Similar in spirit to RMM, but
do just 7 recursive calls, not 8

Write $A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$ each block $\frac{n}{2} \times \frac{n}{2}$

$p_1 = (A_{12} - A_{22}) \cdot (B_{21} + B_{22})$
$p_2 = (A_{11} + A_{22}) \cdot (B_{11} + B_{12})$
$p_3 = (A_{11} - A_{21}) \cdot (B_{11} + B_{12})$
$p_7 \ldots p >$ analogous

$c_{11} = p_1 + p_2 - p_4 - p_6$
$c_{12} = p_4 + p_5$
$c_{21}, c_{22}$ analogous
function $C = Strassen\ (A, B)$

... assume $n = 2^m$

if $n = 1$

$C = A \cdot B$

else

... use formulas for $P_1 \cdots P_7$

$P_1 = Strassen\ (A_{2 \times 2} - A_{2 \times 2}, B_{2 \times 2} + B_{2 \times 2})$

... 6 more analogous calls

$C_{11} = P_1 + P_2 - P_4 + P_6$

... 3 more analogous lines

end if

Analysis similar to RMM

$A(n) = \#\ arithmetic\ ops$

$= 7A(n/2) + \Omega(n^2)$

Solve as before:

$A(n) = A(2^m)$

Divide by $7^m\ instead\ of\ 8^m$

get $A(n) = \Theta(n \log_2 7)$

Similar Analysis for #words moved

$W(n) = 7W(n/2) + \Theta(n^2)$

Base case $W(n) = \Theta(n^2)$ when $n^2 = M$
\[ W(n) = O\left( \frac{n^{x}}{\mu^{x/2} - 1} \right) \quad x = \log_{2} 7 \]

Thm (2010) \( W(n) \) attains lower bound

Thm (2015) \( \Delta \) Extends to "all" Strassen like algorithms for matmul

Latest Record (LeGall 2014) \( x = 2.3728639 \) (not practical)

Error Analysis

Usual matmul (Q11.10)
\[ | \mathbf{f} \mathbf{c} (A \cdot B) - (A \cdot B) | \leq n \cdot \varepsilon |A| \cdot |B| \]

Strassen
\[ \| \mathbf{f} \mathbf{c} (A \cdot B) - (A \cdot B) \| = O(\varepsilon) \| A \| \cdot |B| \]

weaker but still useful
Strassen-like trick for complex matrix mul

\[(A + iB) \cdot (C + iD) = (A \cdot C - B \cdot D) + i(A \cdot D + B \cdot C)\]

only need 3 matrix:

\[T_1 = A \cdot C\]
\[T_2 = B \cdot D\]
\[T_3 = (A + B) \cdot (C + D)\]

\[(A + iB) \cdot (C + iD) = (T_1 - T_2) + i(T_3 - T_1 - T_2)\]

\[\text{cost: 3 matrix + 5 additions}\]
\[\text{vs: 4 matrix + 2 additions}\]

\[\text{cost about } \frac{3}{4} \text{ of traditional algorithm.}\]