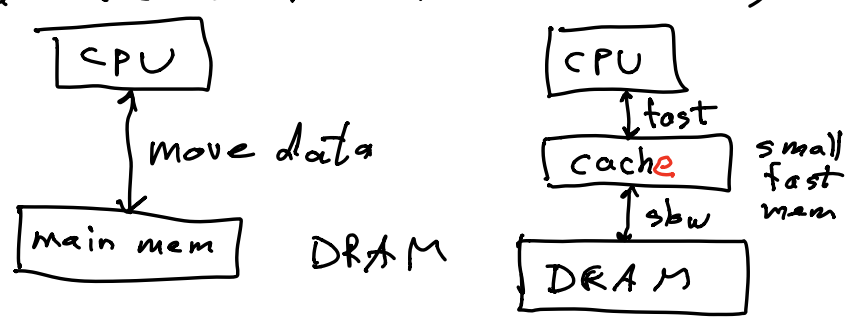
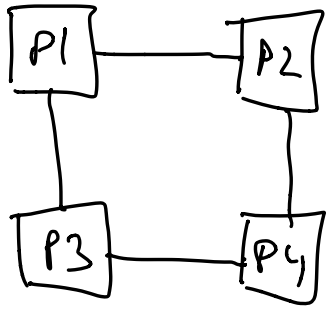
Ma221 Lecture 4 Segment 1

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)



minimize slow memory movement (DRAM $\leftrightarrow$ cache)



moving data from P_i to P_j slow

Notation: all data movement is called "communication"

Matrix Multiplication (Matmul)

Theorm 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, Holtz, 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 $O(10x)$.

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 algs:
Need to define: bandwidth, latency.

For intuition, describe bw + latency for freeway connecting Berkeley + Sacramento.

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

#cars/hour = density (#cars/mile/lane)
× velocity (miles/hour)
× lanes

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

time (hours) = distance (miles) / 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
γ = time per flop

Total time of time $= m \cdot \alpha + w \cdot \beta + f \cdot \gamma$

$f = \#$ flops

Reiterate why communication important.

(1) $\gamma << \beta << \alpha$ on current computers (factor of $10x$ or $100x$)

(2) Gaps are increasing over time

all of γ, β, α improving

but γ faster than β, faster than α

(show figure of α, β, γ)

Similar story for energy costs.

Ma221 Lecture 4 Segment 2

time $= m \cdot \alpha + w \cdot \beta + f \cdot \gamma$

$m = \#$ messages, $\alpha =$ latency

$w = \#$ words moved $\beta = 1/$ bandwidth

$f = \#$ flops, $\gamma =$ time per flop

$\gamma << \beta << \alpha$

$f\cdot\gamma \geq m\alpha + w\beta$

$\Leftrightarrow$ flops dominate

within a factor of 2 of optimal

$f\gamma \gg m\alpha + w\beta$

$\Leftrightarrow$ near machine peak

$f\gamma \ll m\alpha + w\beta$

$\Leftrightarrow$ communication dominates

Notation: Computational Intensity

$q = \frac{f}{w}$ = "flops per word moved"

q needs to be large to be fast

$f\gamma > w\beta \qquad q = \frac{f}{w} > \frac{\beta}{\gamma} \gg 1$

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 $O(n^3)$ flops

BLAS-1 library (Basic Linear Algebra Subprograms)

Standard library of 15 operations, mostly on vectors.

(1) $y = \alpha \cdot x + y$ x, y vectors, $\alpha \beta$ scalars
"AXPY" for short
Ex: innermost loop of GaussElim(GE)
for $k = i+1:n$ $A(j,k) = A(j,k) - A(j,i) \cdot A(i,k)$

(2) dot product

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

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

Motivations for BLAS1

easing programming, readability, since common functions

robustness (avoiding over/underflow in $\|x\|_2$)

portability + efficiency (if optimized on each architecture)

No way to minimize communication using BLAS-1 as building blocks

comp intensity q for dot product

$$= \frac{2n}{2n} = 1$$

$\Rightarrow$ if axpy is used to implement GE,

$q \le 1$ for GE

BLAS2 library (mid 1980s)

standard library of 25 operations

(mostly) on matrix-vector pairs

(1) $y = \alpha y + \beta A x$ $\quad x, y$ vectors, A matrix, α, β scalars

("GEMV")

lots of variations:

A symmetric, banded, triangular, ..

can multiply by A or A^T

obvious optimization if $\alpha, \beta = +1, -1, 0$

(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 GER (details later)

$$A(i+1:n, i+1:n) = A(i+1:n, i+1:n) - A(i+1:n, i) \cdot A(i, i+1:n)$$

(3) Solve $Tx=b$ when T triangular
"TRSV", used by GE

Motivation: similar to BLAS1:
+ more opportunities to optimize
on vector computers

Still not much improvement in communication:

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

BLAS 3 library (late 1980s)
q operations on matrix-matrix pairs

(1) $C = \beta C + \alpha \cdot A \cdot B$ ("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

for $n \times n$ GEMM: $q = \frac{f}{w} = \frac{2n^3}{3n^2 \text{ inputs} + n^2 \text{ outputs}}$

$= \frac{n}{2}$

But straightforward, 3-nested loop version of GEMM no faster than BLAS1 or BLAS2 - need a new algorithm

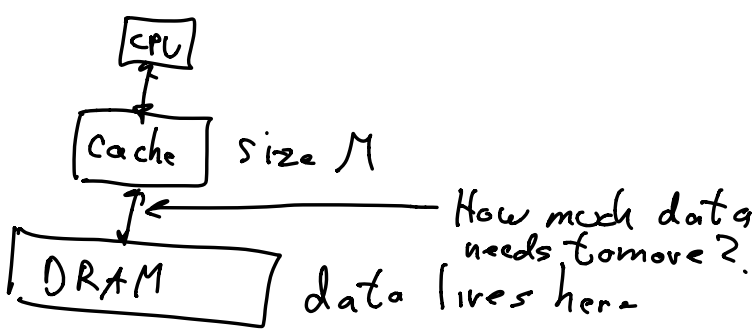
Note: BLAS-k does $O(n^k)$ operations on inputs of dimension n
Also supplied as optimized libraries on all platforms (eg MKL on Intel): use them as your building blocks!
documentation: www.netlib.org/blas

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 Sca/Lapack did not attain them in general. Set off search for new optimal algorithms. Some large speedups attained $O(10x)$, but still work to do, constants matter!

Ma221 Lecture 4 Segment 3

Goal: Prove communication lower bound for matmul



Thm originally due to Hong, Kung 1981.
We will present proof based on
Irony, Tiskin, Toledo, 2004 (generalizes
to parallel case, 3-nested loops)

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

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

$$\Omega\left(\frac{\#\text{ flops}}{\sqrt{M}}\right)$$

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, need to repeat at least $2n^3/G$ times.

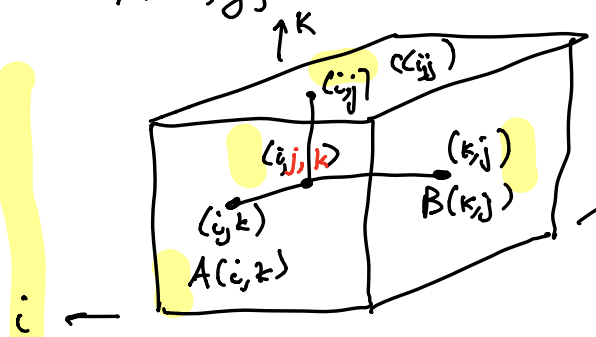
$\Rightarrow$ total # words moved $\geq \frac{2n^3}{G} \cdot 2M$

Need an upper bound G: Geometry problem

$$C(i,j) = C(i,j) + A(i,k) \cdot B(k,j)$$

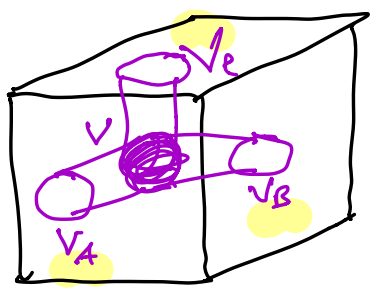
represent this as a lattice point (i,j,k)

$1 \leq i,j,k \leq n$ is a $n \times n \times n$ cube of lattice points



at most M projections ie entries of A, B, C are available.

How many lattice points can I have?



V = 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)

$$|V| \le \sqrt{|V_A|\cdot|V_B|\cdot|V_C|} \le M^{3/2} = G$$

$$\#\text{ words moved} \ge \frac{2n^3}{G} - 2M = \frac{2n^3}{M^{3/2}} - 2M = \Omega\left(\frac{n^3}{\sqrt{M}}\right)$$

QED

If careful with constants, get

$$\#\text{ words moved} \ge \frac{2n^3}{\sqrt{M}} - 2M$$

attainable!

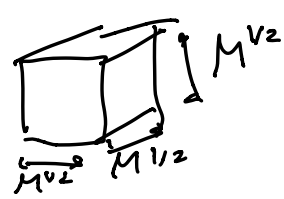
Ma221 Lecture 4 Segment 4

Goal: Optimal Matrix Multiply

(attains comm. lower bound)

proof gives hint

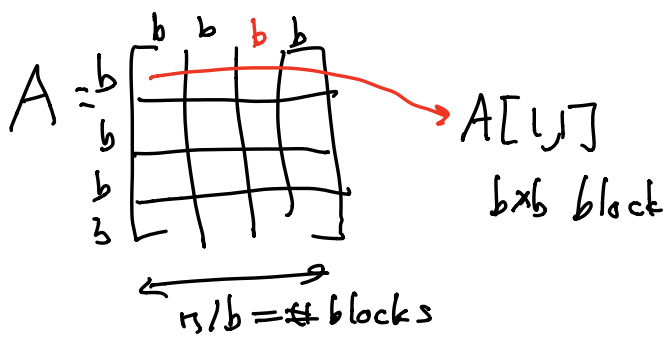
what shape V has surface projection M in all 3 directions and Volume $M^{3/2}$? cube:



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:



```
for i=1 to n/b
    for j=1 to n/b
        read C[i,j] into cache ... b^2 words moved
        for k=1 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]·B[k,j]
            ... b×b matmul, all in cache
            ... 3 more nested loops
        endfor
        write C[i,j] back into memory ... b^2 words moved
    endfor
endfor
```

Total words moved $= (n/b)^2 \cdot b^2 = n^2$ ... for reading C

$+ (n/b)^3 \cdot 2b^2 = 2n^3/b$ ... for reading A, B

$+ (n/b)^2 \cdot b^2 = n^2$ ... for writing C

$= 2n^3/b + 2n^2$

minimize by making b as large as possible

$b = \sqrt{M/3} \Rightarrow$ #words moved $= O\left(\frac{n^3}{\sqrt{M}}\right)$

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: $\Omega(\max(\frac{\#\text{flops}}{\sqrt{M}}, \text{size(input)} + \text{size(output)}))$

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, $i+j$, $i-2j+3k$, ...

Lower bound looks like

$$\Omega\left(\frac{\#\text{loop iterations}}{M^e}\right) \qquad e = \tfrac{1}{2} \text{ for matmul}$$

where e depends on what code looks like (Hölder-Brascamp-Lieb inequality)

– attainable by "blocking", general parallelograms

Another optimal matmul algorithm:
"cache oblivious" because code does not depend explicitly on M, only M for analysis.

```
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=[A11 A12; A21 A22] each Aij is n/2 x n/2
    ... ditto for B and C
    C11 = RMM(A11,B11) + RMM(A12,B21)
    C12 =  ditto
    C21 =  ditto
    C22 =  ditto
  endif
```

Correctness by induction

Cost: $A(n)$ = # arithmetic ops on dimension n

$$A(n) = 8A\left(\tfrac{n}{2}\right) \quad \ldots \text{ 8 recursive calls}$$
$$+ n^2 \quad \ldots \text{ 4 additions of } \tfrac{n}{2} \times \tfrac{n}{2} \text{ submatrices}$$

claim $A(n) = 2n^3 - n^2$ ... same as usual

Solve recurrence:

change variables $n = 2^m$ to m

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

divide by 8^m

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

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

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

$W(n)$ = # words moved

$$W(n) = 8 \cdot W\left(\tfrac{n}{2}\right) \quad \ldots \text{ 8 recursive calls}$$
$$+ 12\left(\tfrac{n}{2}\right)^2 \quad \ldots \text{ 8 reads, 4 writes of } \tfrac{n}{2} \times \tfrac{n}{2} \text{ submatrices}$$
$$= 8W\left(\tfrac{n}{2}\right) + 3n^2$$

oops look like more than $A(n)$

Base case not $W(1)$, rather

$$W(b) = 3b^2 \text{ if } 3b^2 \le M$$

$$b = \sqrt{M/3}$$

Solve as before: $n = 2^m$ to m:

$$w(m) = 8w(m-1) + 3 \cdot 2^{2m}$$

$\bar{m} = \log_2 \sqrt{M/3}$ base case

divide by 8^m to get $v(m) = w(m)/8^m$

$$v(m) = v(m-1) + 3 \cdot (1/2)^m$$

$$v(\bar{m}) = M/8^{\bar{m}} = M/(M/3)^{3/2} = 3^{3/2}/M^{1/2}$$

Geometric sum: only down to $m = \bar{m}$ not 1

$$v(m) = \sum_{k=\bar{m}+1}^{m} 3(1/2)^k + v(\bar{m})$$

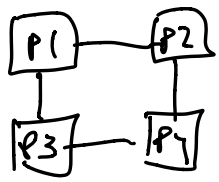
$$= 2 \cdot 3^{3/2}/\sqrt{M}$$

$$W(n) = w(\log_2 n) = 8^{\log_2 n} v(\log_2 n) = 2 \cdot 3^{3/2} \frac{n^3}{\sqrt{M}}$$

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: $\Omega\left(\frac{\#\text{flops per proc}}{\sqrt{\text{mem per proc}}}\right)$

$$= \Omega\left(\frac{n^3/P}{\sqrt{n^2/P}}\right) = \Omega\left(\frac{n^2}{\sqrt{P}}\right)$$

- attainable by parallel algs (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, comm. costs,
Similarly: Many algorithms just
discuss first one:

Strassen (1967) matmul possible
in $O(n^{\log_2 7})$ ops $\sim O(n^{2.81})$
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 $\frac{n}{2} \times \frac{n}{2}$

$P1 = (A_{12} - A_{22}) \cdot (B_{21} + B_{22})$
$P2 = (A_{11} + A_{22}) \cdot (B_{11} + B_{22})$
$P3 = (A_{11} - A_{21}) \cdot (B_{11} + B_{12})$
$P4 \ldots P7$ analogous

$C_{11} = P1 + P2 - P4 + P6$
$C_{12} = P4 + P5$
C_{21}, C_{22} analogous

```
function C = Strassen(A, B)
    ... assume n = 2^m
    if n = 1
        C = A · B
    else
        ... use formulas for P1 .. P7
        P1 = Strassen(A12 - A22, B21 + B22)
        ... 6 more analogous calls
        C11 = P1 + P2 - P4 + P6
        ... 3 more analogous lines
    end if
```

Analysis similar to RMM

$$A(n) = \#\text{arithmetic ops}$$
$$= 7A\left(\tfrac{n}{2}\right) + 18\left(\tfrac{n}{2}\right)^2$$

solve as before:

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

divide by 7^m instead of 8^m

get $A(n) = O(n^{\log_2 7})$

Similar Analysis for #words moved

$$W(n) = 7W\left(\tfrac{n}{2}\right) + O(n^2)$$

base case $W(\bar{n}) = O(\bar{n}^2)$ when $\bar{n}^2 = M$

$$W(n) = O\left(\frac{n^x}{M^{x/2-1}}\right) \quad x = \log_2 7$$

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

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

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

Error Analysis

Usual matmul (Q1.10)

$$|fl(A \cdot B) - (A \cdot B)| \leq n \cdot \varepsilon |A| \cdot |B|$$

Strassen

$$\| fl(A \cdot B) - (A \cdot B) \| = O(\varepsilon) \|A\| \cdot \|B\|$$

weaker but still useful

Strassen like trick for complex matmul

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

only need 3 matmuls:

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

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

cost: 3 matmuls + 5 additions

vs: 4 matmuls + 2 additions

cost about $\frac{3}{4}$ of traditional algorithm.