Welcome to Ma 221! Lec 6 Fall 24

Fact 10 about SVD of $A = U\Sigma V^T = \sum_i u_i \cdot \sigma_i \cdot v_i^T$

goal: data compression

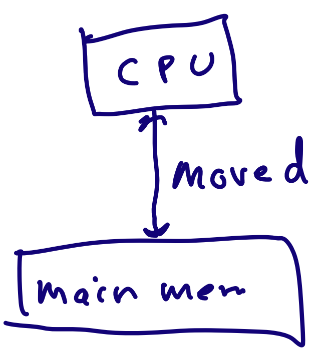
closest Matrix to A of rank $\le k$ is $A_k = \sum_{i=1}^{k} u_i \sigma_i \cdot v_i^T$, measured in $\|\cdot\|_2$

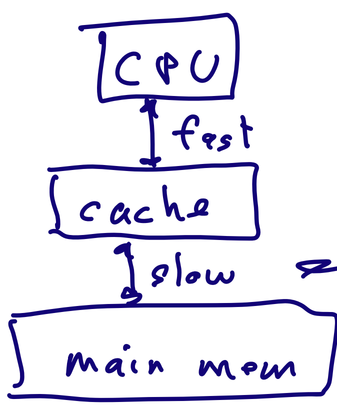
$\text{size}(A^{m \times n}) = m \cdot n$, $\text{size}(A_k) = k(m+n)$

(matlab demo, see typed notes for code)

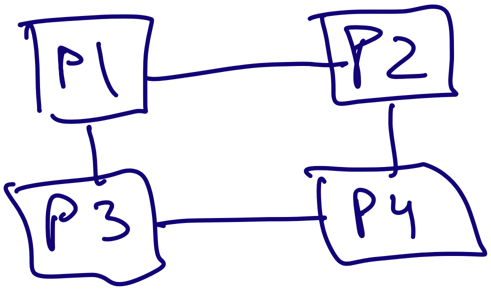
Goal: understand real cost (in time) of running an algorithm

Traditionally: count flops, but flops are cheapest ops, costs orders of magnitude more to move data





Goal: minimize this data movement



moving data between procs is slow

Notation: "communication"

Matmul: Thm: gives a lower bound on #words moved between main memory and cache of size M, for any matmul doing n^3 muls + adds in any order (Hong, Kung 1981)

2004: extended to parallel case (assuming load balanced, memory balanced)

2011: extended to any algorithm that "smells like 3 nested loops" including matmul, GE, LS, eig, ...

Usual algorithms for GE, LS, eig, .. cannot attain lower bounds, just by reordering ops, Need new ones, will sketch some of them

Extends to parallel case, etc

Need simple model of comm. costs:
Bandwidth (bw), Latency

Intuition: freeway from Berkeley → Sacramento

BW = # cars/hour that can go from B → S

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

Latency = how long it takes 1 car to go B → S

time (hours) = distance / velocity (miles/hour)

So minimum time to move n cars from B → S: need to drive packed together in "convoy" as close as possible

time (hrs) = time for first car
+ time for rest
= latency + n/bw

Same idea for moving data:

time to move w words from DRAM to Cache

$= \text{latency} + w/bw$

assuming all words packed into one "message"

Moving w words in m messages

$\text{cost} = m \cdot \text{latency} + w/bw$

Notation: $m \cdot \alpha + w \cdot \beta$

$\alpha = \text{latency}$

$\beta = 1/bw$

$\gamma = \text{time per flop}$

$f = \#\text{ flops}$

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

Today: $\gamma \ll \beta \ll \alpha$

growing apart exponentially

same for energy (see plot of α, β, γ over time, posted on web page) flops dominates cost if

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

comm dominates if

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

Notation: Computational Intensity

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

q needs to be large to be fast

$$f\gamma \geq w\beta \Rightarrow \frac{f}{w} = q \geq \frac{\beta}{\gamma} >> 1$$

History of how this has influenced algorithms:

In the beginning was the do loop
Enough for first libraries, eg
EISPACK (mid 1960s)

People didn't worry about comm,
just flops and accuracy

BLAS-1 Library (Basis Linear Algebra Sub programs)

Standard library of 15 ops mostly on vectors:

(1) $y = \alpha \cdot x + y$ $\quad$ x, y vectors, α scalar

"AXPY", inner loop of GE

(2) dot product

(3) $\|x\|_2 = \sqrt{\sum_i x_i^2}$

(4) find largest entry $|x_i|$ in x

Motivation: easier programming, readability, robustness (avoid over/underflow in $\|\cdot\|_2$)

portable + efficient

Can't minimize comm:

Computational intensity (for dot prod)

$= q = \frac{f}{w} = \frac{2n}{2n} = $)

BLAS-2 library (mid 1980s)
standard library of 25 ops
on pairs of matrix+vector

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

"GEMV"

lots of variations: A symm, triangular, could use A or A^T

(2) $A = A + \alpha \cdot x \cdot y^T$ — rank-one update

"GER" 2 inner most loops of GE

(3) Solve $Tx = b$, T triangular

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

not much improvement on q

GEMV: $q = \frac{f}{w} = \frac{2n^2}{n^2+n+n} \sim 2$

BLAS-3 library (late 1980s)

9 operations on pairs of matrices

(1) $C = \beta \cdot C + \alpha \cdot A \cdot B$ A, B, C matrices, α, β scalars

"GEMM"

(2) $C = \beta C + \alpha A \cdot A^T$ $A^{n \times k}$

"SYRK"

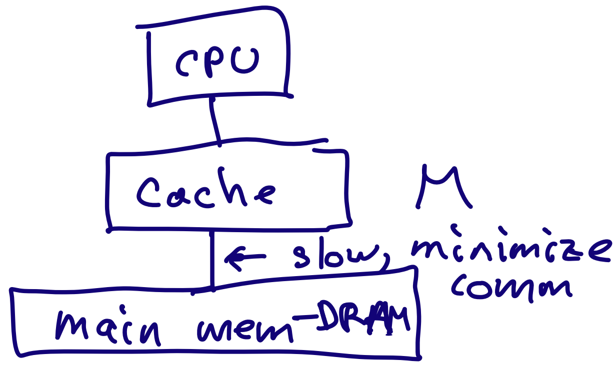
(3) Solve $TX = B$, T triangular, X, B rectangular

For GEMM: $q = \frac{f}{w} = \frac{2n^3}{4n^2} = \frac{n}{2}$ big at last!

But usual 3 nested loops for GEMM no help, actual q for "naive" GEMM $= 2$

Hint: BLAS-k does $O(n^k)$ operations on data of dimension n

Goal: Prove comm lower bound for mat mul for



Easy case: if $3n^2 \le M$ (all A, B, C fit in cache)

: read all data (A, B, C) into cache do work, write C back to DRAM

Hard case: $3n^2 > M$

Thm (Hong, Kung, 1981) to multiply $C = A \cdot B$ using usual $2n^3$ flops in any correct order,

\# words moved (cache $\leftrightarrow$ DRAM)

$$= \Omega\left(\frac{n^3}{\sqrt{M}}\right)$$

Modern proof, based on

Irony, Tiskin, Toledo (2004)

extends to rectangular, sparse matrices:

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

Proof sketch (ignore constants)

Suppose we fill cache with M words, do as many flops as possible, store results back to DRAM, repeat until done

Upper bound # flops possible by G

$\Rightarrow$ doing G flops costs $2M$ words moved

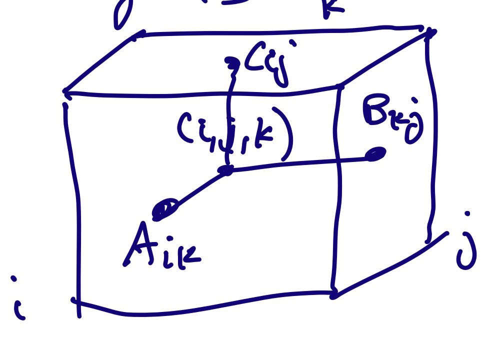
$\Rightarrow$ since we need to do $2n^3$ flops need to repeat $2n^3/G$ times

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

Need G (or an upper bound)

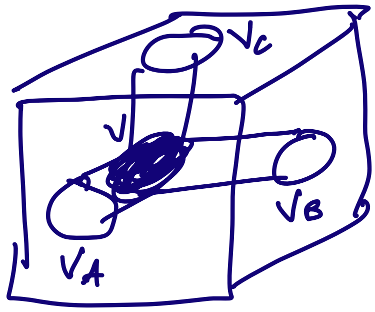
Use a geometric model to get G

Represent alg as $n \times n \times n$ lattice

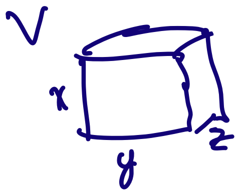


$(i,j,k) \equiv$

$C_{ij} \mathrel{+}= A_{ik} \cdot B_{kj}$



need to bound $|V|$ using
$|V_A|, |V_B|, |V_C|$

Intuition: 

$|V| = x \cdot y \cdot z$

$|V_A| = x \cdot y$

$|V_B| = x \cdot z$

$|V_C| = y \cdot z$

$$\sqrt{|V_A| \cdot |V_B| \cdot |V_C|} = \sqrt{x^2 \cdot y^2 \cdot z^2} = |V|$$

Thm (Loomis-Whitney, 1949)

$$\sqrt{|V_A| \cdot |V_B| \cdot |V_C|} \geq |V|$$

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

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

$$= \Omega\left(\frac{n^3}{\sqrt{M}}\right)$$

If careful with constants

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

attainable!

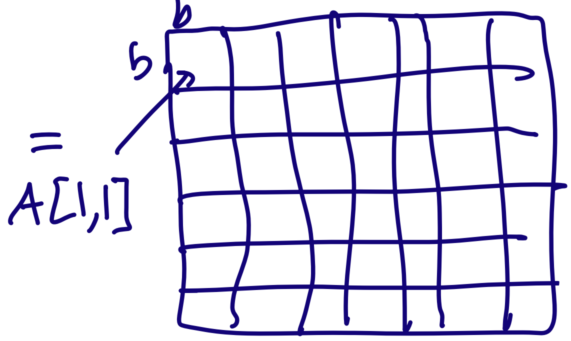
Goal: Optimal Matmul:

What shape does V need to have in order to attain bound?

$|V| = \sqrt{|V_A| \cdot |V_B| \cdot |V_C|}$ V a cube of size $M^{1/2} \times M^{1/2} \times M^{1/2}$

Algorithm: break A, B, C into square submatrices that all fit in cache, read each triple of submatrices into cache, multiply them, put answer back in DRAM

$A^{n\times n} =$



A[i,j] is a $b\times b$ submatrix

```
for i = 1 to n/b
  for j = 1 to n/b
    read C[i,j] into cache, ... b^2 words
    for k = 1 to n/b
      read A[i,k], B[k,j] into cache,
                              2b^2 words
      C[i,j] = C[i,j] + A[i,k] · B[k,j]
        bxb matmul, 3 more loops
        all in cache
    end for
    write C[i,j] to main mem,
                              b^2 words
  end for
end for
```

Total Words moved =

$$2\frac{n}{b}\cdot\frac{n}{b}\cdot b^2 + \left(\frac{n}{b}\right)^3\cdot 2b^2 = 2n^2 + \frac{2n^3}{b}$$

where $b = \sqrt{\frac{M}{3}}$

$$= O\left(\frac{n^3}{\sqrt{M}}\right)$$

General case: nonsquare, some matrix small

lower bound is $\Omega\left(\max\left(\frac{m \cdot n \cdot k}{\sqrt{M}}, \text{size(input)}, \text{size(output)}\right)\right)$