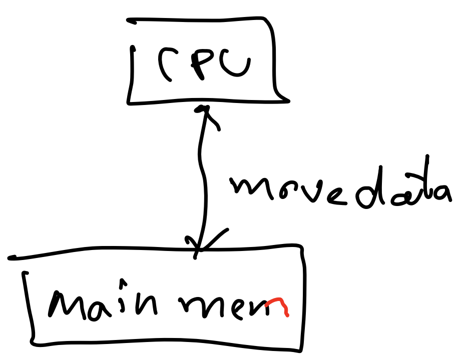
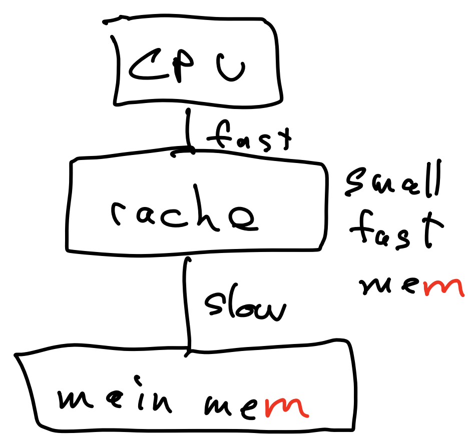
Welcome to Ma221 !

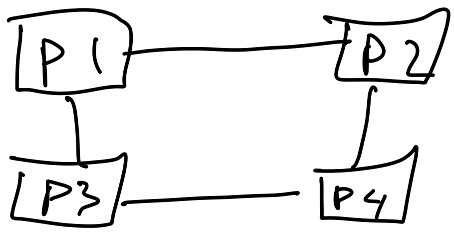
Goal: understand real cost in time of running an alg.

Traditionally: count flops but this is cheapest op, costs orders of magnitude more to move data:





Goal: minimize data movement cache $\leftrightarrow$ main mem



moving data between procs over network

slow

Notation: "communication"

Matmul: Theorem: gives a lower bound on # words moved between main memory and cache of size M assuming do usual $2n^3$ flops, done in any order (Hong, Kung 1981)
There is a widely known and used alg attaining lower bound

2004: extended to parallel case
2011: extended to any algorithm that "smells like 3 nest loops": includes matmul, GE, LS, eig, ...

Usual algorithms for GE, LS, eig, ..
cannot attain lower bound, just by
reordering ops. Will sketch some,
widely used, new algorithms

Extends to other computer architectures
(eg parallel)

(many possible class projects)

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 get from B → S

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

So minimum time to move n cars from B → S when they they all travel in one "convoy" as close as possible

time (hrs) = time for 1st car + time for remaining cars

= latency + n / bw

Same idea for moving data

time to move w words from DRAM to Cache = latency + w / bw

assuming all words in one "message"

Moving w words in m messages costs $m \cdot \text{latency} + w / bw$

Notation: $m \cdot \alpha + w \cdot \beta =$ comm cost

α = latency

β = 1/bw

γ = time per flop

f = # flops

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

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

growing apart exponentially

same model for energy

flops dominates if

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

comm dominates if

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

Notation: Computational Intensity:

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

q needs to be large to be fast

$$f \cdot \gamma \geq w \cdot \beta$$

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

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 (Basic Linear Algebra Subprograms)

Standard library of 15 ops mostly on vectors:

(1) $y = \alpha \cdot x + y$ x, y vectors, α scalar
"AXPY" for short
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)
portable + efficient

Can't minimize comm

$$\text{Comp. Intensity} = q = \frac{f}{w} = \frac{2n}{2n} = 1$$

dot product

BLAS-2 library (mid 1980's)
standard library of 25 ops
on pairs of matrices + vectors

(1) $y = \alpha y + \beta A x$

A matrix
x, y vectors
α, β scalars

"GEMV"
lots of variations A symmetric,
triangular, ...
A or A^T

(2) $A = A + \alpha x \cdot y^T$ rank-one update
"GER" : 2 innermost loops
of GE

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

Motivation: similar to BLAS1
+ more opportunities for
optimation 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) $C = \beta C + \alpha A \cdot B$ — 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

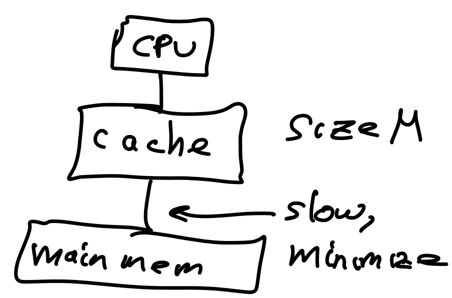
for GEMM $q = \frac{f}{w} = \frac{2n^3}{3n^2+n^2} = \frac{n}{2}$

But usual 3 nested loops for
matmul no help, actual $q = 2$

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

BLAS3 led community to rebuild all linear algebra to use GEMM etc as much as possible LAPACK, ScaLAPACK,

Goal: Prove comm lower bound for matmul for



Easy case: if $3n^2 \le M \Rightarrow$ read all data into cache, do all 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 $= \Omega\left(\frac{n^3}{\sqrt{M}}\right)$

More modern proof, based on
Irony, Tiskin, Toledo (2004)

Extends to rectangular, sparse matrices

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

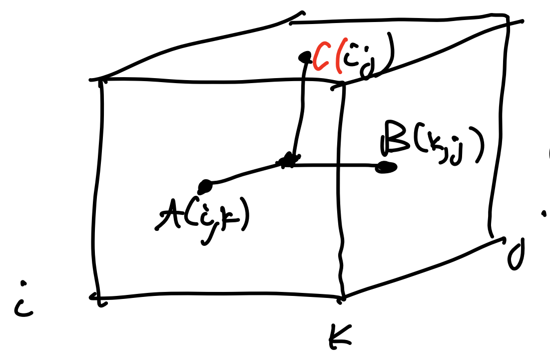
Proof sketch (ignore constants)

Suppose we fill cache with M words,
do as many flops as possible,
store results back to main mem
Upper bound # flops possible by G
$\Rightarrow$ Doing G flops costs 2M words moved
$\Rightarrow$ Since we do $2n^3$ flops, need to repeat $2n^3/G$ times
$\Rightarrow$ # words moved $\underline{\frac{2n^3}{G}\cdot 2M}$

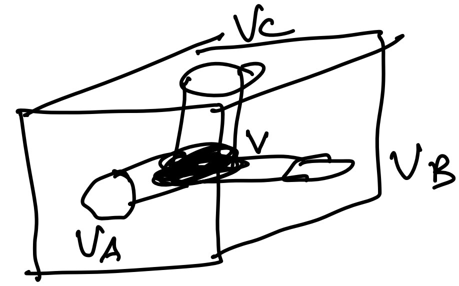
Need G (or an upper bound)

use geometric model to get G

represent alg as $n \times n \times n$ lattice
with axes i, j, k



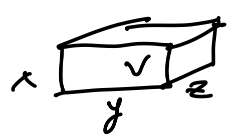
$(i,j,k) \equiv$
$C(i,j) \pm A(i,k) \cdot B(k,j)$



Bound $|V|$ using $|V_A|, |V_B|, |V_C|$

because $|V_A| + |V_B| + |V_C| \leq M$

Intuition



$|V| = x \cdot y \cdot z$
$|V_A| = xy$
$|V_B| = x \cdot z$
$|V_C| = y \cdot z$

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

Thus (Loomis + Whitney 1949)

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

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

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

If careful with constants:

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

attainable!

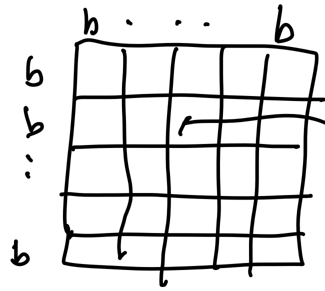
Goal: Optimal Matmul:

What shape of V has projections V_A, V_B, V_C all of size M, $|V| = M^{3/2}$?

Cube: V: $M^{1/2} \times M^{1/2} \times M^{1/2}$ cube

Algorithm: break A, B, C into square submatrices that all fit in cache, as large as possible

read each triple of submatrices into cache, multiply, put answer back into main mem

$A^{n \times n} =$ 

$A[i,j]$ $b \times b$ 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 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
    write $C[i,j]$ to main mem ... b^2 words moved
  endfor
endfor

Total words moved =

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

$$= 2n^2 + \frac{2n^3}{b} \qquad \text{where } b = \sqrt{\frac{M}{3}}$$

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

What about rectangular case?

What if some dimension $< b$?

Ex: Ax

General case for $m \times k \times n$ matmul:

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

Idea of using Loomis-Whitney enough for linear algebra, extends to any algorithm that looks like:

- nested loops
- any number of arrays
- "any" subscripts, eg i, $i+j$, $i-2j+k$, ..

Get a lower bound and optimal alg

$$\#\text{words moved} = \Omega\left(\frac{\#\text{loop iterations}}{M^e}\right)$$

e depends on details of alg, uses generalization of Loomis Whitney called Hölder-Brascamp-Lieb Inequality