CS410: Parallel Computing

Spring 2024

Nikhil Hegde
Milind Chabbi

Applications and Algorithms from Dense Linear Algebra

Lecture 18 and 19, IIT Dharwad
Feb/14/2024
Feb/16/2024

credits: James Demmel ([CS267]), Ananth Grama ([slides accompanying the text book])

This class and next..

- Applications and Algorithms from Dense Linear Algebra
  - Matrix multiplication, Matrix-Vector multiplication
  - Solving linear equations using Gauss Elimination

- “Thinking Parallel”
  - Synchronization requirement (have seen this)
  - Load balancing requirement (haven’t seen this much)
  - Minimizing communication overhead requirement (haven’t seen this much)

Matrix Multiplication

- Why study?
  - An important “kernel” in many linear algebra algorithms
  - Most studied kernel in high performance computing
  - Simple. Optimization ideas can be applied to other kernels
- Matrix representation
  - Matrix is a 2D array of elements. Computer memory is inherently linear
  - C++ and Fortran allow for definition of 2D arrays. 2D arrays stored row-wise in C++. Stored column-wise in Fortran. E.g.

```
// stores 10 arrays of 20 doubles each in C++
double** mat = new double[10][20];
```

Storage Layout - Example

- Matrix (**2D**): $A = \begin{bmatrix} A(0,0) & A(0,1) & A(0,2) \\ A(1,0) & A(1,1) & A(1,2) \\ A(2,0) & A(2,1) & A(2,2) \end{bmatrix}$

$A(i,j) = A(row, column)$ refers to the matrix element in the i^{th} row and the j^{th} column

- Row-wise (/Row-major) storage in memory:

$A(0,0)$	$A(0,1)$	$A(0,2)$	$A(1,0)$	$A(1,1)$	$A(1,2)$	$A(2,0)$	$A(2,1)$	$A(2,2)$

- Column-wise (/Column-major) storage in memory:

$A(0,0)$	$A(1,0)$	$A(2,0)$	$A(0,1)$	$A(1,1)$	$A(2,1)$	$A(0,2)$	$A(1,2)$	$A(2,2)$

Matrix Multiplication

- Three fundamental ways to think of the algorithm
  - Dot product / Inner product

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \times \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} = \begin{bmatrix} 1.5 + 2.7 & 1.6 + 2.8 \\ 3.5 + 4.7 & 3.6 + 4.8 \end{bmatrix}$$

  - Linear combination of left matrix columns

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \times \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} = \begin{bmatrix} 5 \begin{bmatrix} 1 \\ 3 \end{bmatrix} + 7 \begin{bmatrix} 2 \\ 4 \end{bmatrix} & 6 \begin{bmatrix} 1 \\ 3 \end{bmatrix} + 8 \begin{bmatrix} 2 \\ 4 \end{bmatrix} \end{bmatrix}$$

  - Sum of outer products

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \times \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} 1 \\ 3 \end{bmatrix} \begin{bmatrix} 5 & 6 \end{bmatrix} + \begin{bmatrix} 2 \\ 4 \end{bmatrix} \begin{bmatrix} 7 & 8 \end{bmatrix} \end{bmatrix}$$

Review: Matrix-Matrix Product

- Computing Matrix-Matrix product $C = C + AB, A \in \mathbb{R}^{m\times r}, B \in \mathbb{R}^{r\times n},\ C \in \mathbb{R}^{m\times n}$

$$\begin{bmatrix} c_{11} & c_{12} & .. & c_{1n} \\ c_{21} & c_{22} & .. & c_{2n} \\ & \vdots & & \\ c_{m1} & c_{m2} & .. & c_{mn} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & .. & c_{1n} \\ c_{21} & c_{22} & .. & c_{2n} \\ & \vdots & & \\ c_{m1} & c_{m2} & .. & c_{mn} \end{bmatrix} + \underbrace{\begin{bmatrix} a_{11} & a_{12} & .. & a_{1r} \\ a_{21} & a_{22} & .. & a_{2r} \\ & \vdots & & \\ a_{m1} & a_{m2} & .. & a_{mr} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} & .. & b_{1n} \\ b_{21} & b_{22} & .. & b_{2n} \\ & \vdots & & \\ b_{r1} & b_{r2} & .. & b_{rn} \end{bmatrix}}$$

$$\mathrm{A}\times\mathrm{B} = \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} + .. + a_{1r}b_{r1} & . & . & a_{11}b_{1n} + a_{12}b_{2n} + .. + a_{1r}b_{rn} \\ . & & . & \\ a_{m1}b_{11} + a_{m2}b_{21} + .. + a_{mr}b_{r1} & . & . & a_{m1}b_{1n} + a_{m2}b_{2n} + .. + a_{mr}b_{rn} \end{bmatrix}$$

Notice that:

- subscript on a varies from 1 to `m` in a column (i.e. m rows exist)
- subscript on a varies from 1 to `r` in a row (i.e. r columns exist)

If we treat a_i and b_i as a vector of size r and for such m vectors:

$$= \begin{bmatrix} a_1^T b_1 & . & . & a_1^T b_n \\ . & . & & \\ a_m^T b_1 & . & . & a_m^T b_n \end{bmatrix}$$

$a_i^T \in \mathbb{R}^{1\times r}, b_j \in \mathbb{R}^{r\times 1}$

i ranges from 1 to m

j ranges from 1 to n

Review: Matrix-Matrix Product using Dot Product Formulation

- Pseudocode - Matrix-Matrix product: $C = C + AB,\ A \in \mathbb{R}^{m\times r}, B \in \mathbb{R}^{r\times n}, C \in \mathbb{R}^{m\times n}$

```
..
for i=1 to m
    for j=1 to n
        //compute updates involving dot products
```

$$c_{ij} = c_{ij} + a_i^T b_j$$

Review: Matrix-Matrix Product using Dot Product Formulation

- Pseudocode - Matrix-Matrix product: $C = C + AB,\ A \in \mathbb{R}^{m\times r}, B \in \mathbb{R}^{r\times n}, C \in \mathbb{R}^{m\times n}$

```
..
for i=1 to m
    for j=1 to n
        //compute updates involving dot products
```

$$c_{ij} = c_{ij} + a_i^T b_j$$

- Expanded:

```
..
for i=1 to m
    for j=1 to n
        for k=1 to r
```

$$c_{ij} = c_{ij} + a_{ik} b_{kj}$$

Review: Matrix-Matrix Product using Dot Product Formulation

- Cost? (of $C = C + AB,\ A \in \mathbb{R}^{m\times r}, B \in \mathbb{R}^{r\times n}, C \in \mathbb{R}^{m\times n}$)
  - Per dot-product cost = 2r ($a_i\ , b_j \in \mathbb{R}^r$)
  - Total cost = **2mnr** or **O(mnr)**

```
..
for i=1 to m
    for j=1 to n
        for k=1 to r
```

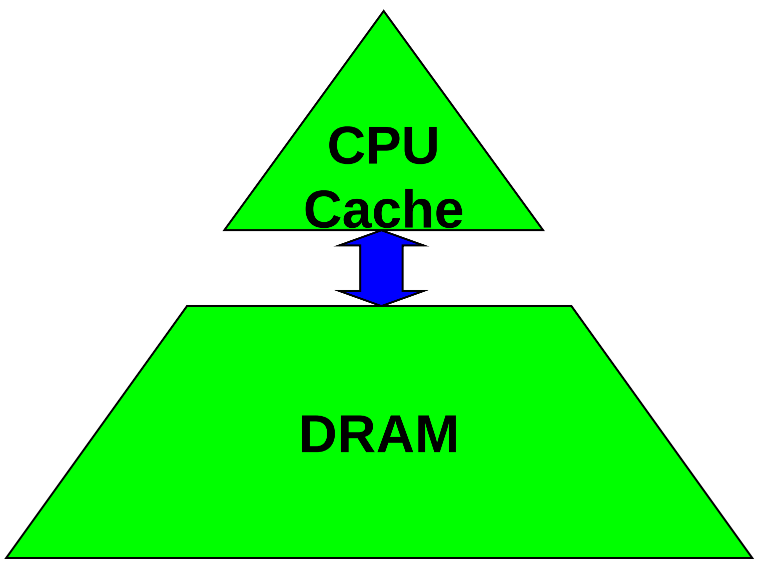
$$c_{ij} = c_{ij} + a_{ik}b_{kj}$$

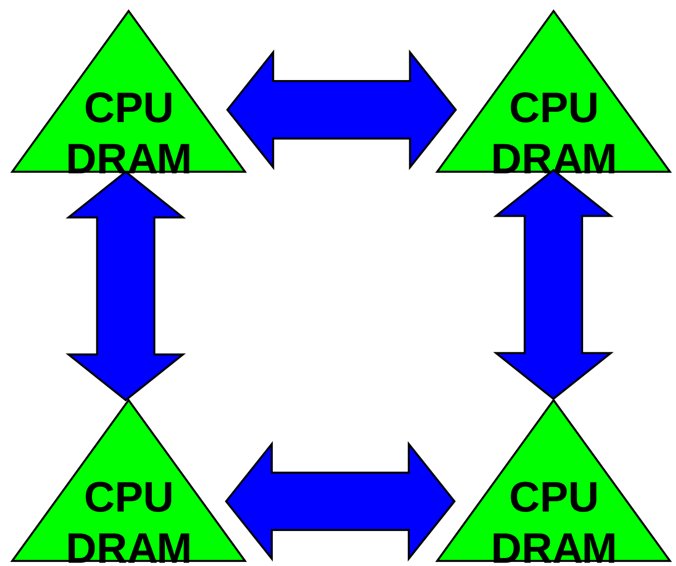
- The above is the arithmetic cost. What about other costs ?

Costs Involved

Algorithms have two costs:

1. Arithmetic (FLOPS)
2. Communication: moving data between
   - levels of a memory hierarchy (sequential case)
   - processors over a network (parallel case).





Costs Involved

- Running time of an algorithm is sum of 3 terms:
  1. # flops * time_per_flop
  2. # words moved / bandwidth
  3. # messages * latency

communication

Matrix-Matrix Addition

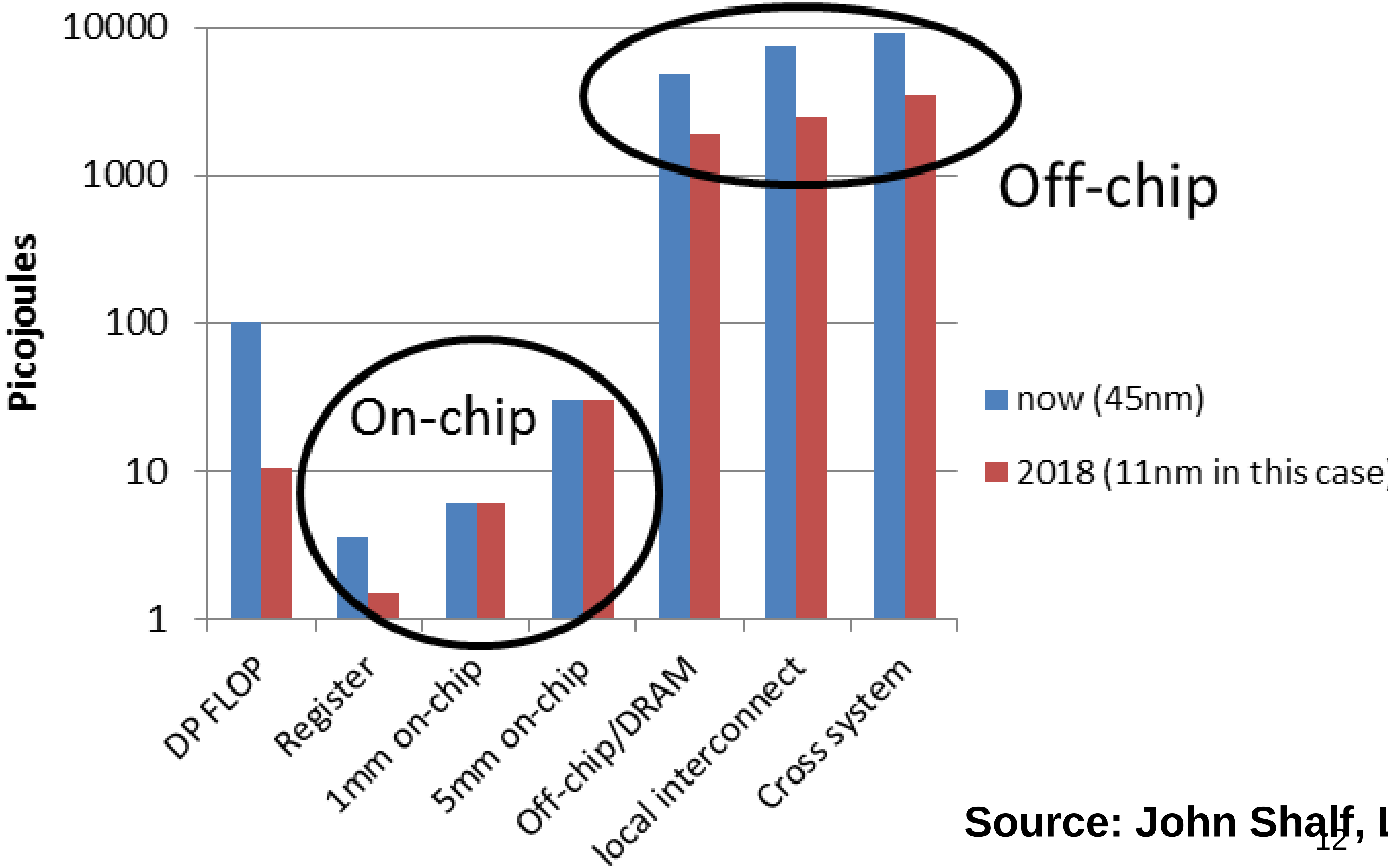
Minimize communication to save time

```
Run on (12 X 2592.01 MHz CPU s)
CPU Caches:
  L1 Data 32 KiB (x6)
  L1 Instruction 32 KiB (x6)
  L2 Unified 256 KiB (x6)
  L3 Unified 12288 KiB (x1)
Load Average: 0.07, 0.02, 0.07
------------------------------------------------------------------------------------
Benchmark                          Time             CPU   Iterations UserCounters...
------------------------------------------------------------------------------------
BM_AddByCol/64/64               3270 ns         3270 ns       212929 items_per_second=1.25254G/s
BM_AddByCol/128/128            39741 ns        39741 ns        17617 items_per_second=412.272M/s
BM_AddByCol/256/256           314880 ns       314878 ns         2241 items_per_second=208.132M/s
BM_AddByCol/512/512          1276733 ns      1276723 ns          545 items_per_second=205.326M/s
BM_AddByRow/64/64                693 ns          693 ns      1042737 items_per_second=5.91004G/s
BM_AddByRow/128/128             2464 ns         2464 ns       271766 items_per_second=6.64813G/s
BM_AddByRow/256/256            11134 ns        11133 ns        63210 items_per_second=5.88639G/s
BM_AddByRow/512/512            44353 ns        44353 ns        15576 items_per_second=5.91041G/s
```


(Source code and further reading)

Costs Involved

Minimize communication to save energy



Source: John Shalf, LBL

Computational Intensity

- Connection between computation and communication cost
- Average number of operations performed per data element (word) read/written from slow memory
  - E.g. Read/written m words from memory. Perform f operations on m words.
  - Computational Intensity q = f/m (*flops per word*).
- Goal: we want to *maximize* the computational intensity
  - We want to minimize words moved (read/written)
  - We want to minimize messages sent

Communication Cost – Matrix-Matrix Product

```
//Assume A, B, C are all nxn
for i=1 to n
 for j=1 to n
   for k=1 to n
     C(i,j)=C(i,j) + A(i,k)*B(k,j)
```

- n^2 words read: each row of A read once for each i.
- Assume that row i of A stays in fast memory during j=2, .. J=n
- Reading a row i of A

- loop k=1 to n: read C(i,j) into fast memory and update in fast memory
- End of loop k=1 to n: write C(i,j) back to slow memory

n^2 words read and n^2 words written (each entry of C read/written to memory once). = 2 n^2 words read/written

total cost = 3 n^2 +n^3 (if the cache size is n+n+1)

- Reading column j of B
- Suppose there is space in fast memory to hold only one column of B (in addition to one row of A and 1 element of C), then every column of B is read once in **inner two loops.**
- Each column of B read n times including **outer i loop =** n^3 words read

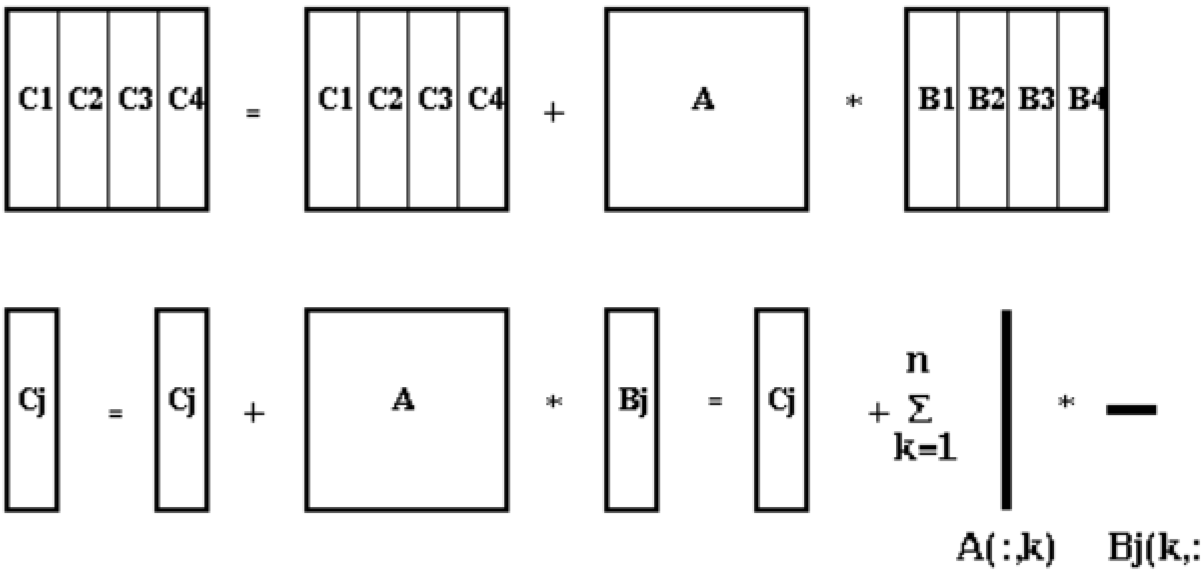
Computational Intensity – Matrix-Matrix Product

- Words moved = $n^3+3n^2 = n^3+O(n^2)$
- Number of arithmetic operations = $2n^3$
- computational intensity $q \approx 2n^3/n^3 = 2$. (computation to communication ratio)

- Can we do better?

Blocked Matrix Multiply

- For N=4:




```
for j=1 to N
//Read entire Bj into fast memory
//Read entire Cj into fast memory
  for k=1 to n
   //Read column k of A into fast memory
   Cj=Cj + A(*,k) * Bj(k,*)
   //Write Cj back to slow memory
```


source: http://people.eecs.berkeley.edu/~demmel/cs267/lecture02.html

Computational Intensity - Blocked Matrix Multiply

```
for j=1 to N
//Read entire Bj into fast memory
//Read entire Cj into fast memory
  for k=1 to n
   //Read column k of A into fast memory
   C(*,j)=C(*,j) + A(*,k)*Bj(k,*) //outer-product
        //Write Cj back to slow memory
```

n^2 words read: each column of B read once.

Nn^2 words read: each column of A read N times

$2n^2$ words read: read/write each entry of C to memory once.

- Number of arithmetic operations = $2n^3$
- $\boldsymbol{q = 2n^3/(N+3)n^2 \ = \ 2n/N}$. **Good!**

Blocked Matrix Multiply - General

$$C = \begin{bmatrix} C_{11} & C_{12} & .. & C_{1r} \\ C_{21} & C_{22} & .. & C_{2r} \\ & \vdots & & \\ C_{q1} & C_{q2} & .. & C_{qr} \end{bmatrix} \qquad A = \begin{bmatrix} A_{11} & A_{12} & .. & A_{1p} \\ A_{21} & A_{22} & .. & A_{2p} \\ & \vdots & & \\ A_{q1} & A_{q2} & .. & A_{qp} \end{bmatrix} \qquad B = \begin{bmatrix} B_{11} & B_{12} & .. & B_{1r} \\ B_{21} & B_{22} & .. & B_{2r} \\ & \vdots & & \\ B_{p1} & B_{p2} & .. & B_{pr} \end{bmatrix}$$

$\downarrow q \; \rightarrow r$ $\qquad$ $\downarrow q \; \rightarrow p$ $\qquad$ $\downarrow p \; \rightarrow r$

- $A, B, C \in \mathbb{R}^{n \times n}$
- We wish to update C block-by-block: $C_{ij} = C_{ij} + \Sigma_{k=1}^{p} A_{ik} B_{kj}$
  – Assume that blocks of A, B, and C fit in cache. C_{ij} is roughly n/q by n/r, A_{ij} is roughly n/q by n/p, B_{ij} is roughly n/p by n/r.
  – But how to choose block parameters p, q, r such that assumption holds for a cache of size M?
    - i.e. given the constraint that $\frac{n}{q} \times \frac{n}{r} + \frac{n}{q} \times \frac{n}{p} + \frac{n}{p} \times \frac{n}{r} \leq M$

Blocked Matrix Multiply - General

- Maximize $\frac{2n^3}{qrp}$ subject to $\frac{n}{q} \times \frac{n}{r} + \frac{n}{q} \times \frac{n}{p} + \frac{n}{p} \times \frac{n}{r} \leq M$
  - $q_{opt} = p_{opt} = r_{opt} \approx \sqrt{\frac{n^2}{3M}}$

- *Each block should roughly be a square matrix and occupy one third of the cache size*

i.e. for a bxb block matrix, $3b^2$ = cache size = M

Review: Blocked Matrix Multiply

- Blocked Matmul $C = A \cdot B$ breaks A, B and C into blocks with dimensions that depend on cache size

… Break A^{nxn}, B^{nxn}, C^{nxn} into bxb blocks labeled A(i,j), etc

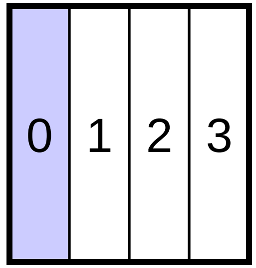
… b chosen so 3 bxb blocks fit in cache

for i = 1 to n/b, for j=1 to n/b, for k=1 to n/b

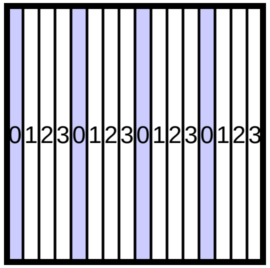
$C(i,j) = C(i,j) + A(i,k) \cdot B(k,j)$ … b x b matmul, $4b^2$ reads/writes

- When b=1, get "naïve" algorithm, want b larger …
- $(n/b)^3 \cdot 4b^2 = 4n^3/b$ reads/writes altogether
- Minimized when $3b^2$ = cache size = M, yielding $O(n^3/M^{1/2})$ reads/writes

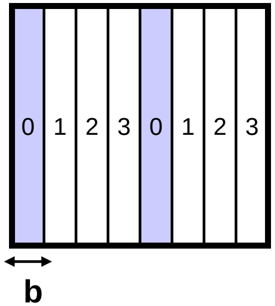
Different Parallel Data Layouts for Matrices (not all!)



1) 1D Column Blocked Layout

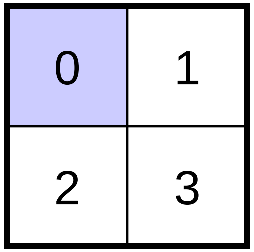


2) 1D Column Cyclic Layout

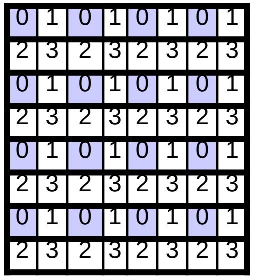


3) 1D Column Block Cyclic Layout

4) Row versions of the previous layouts



5) 2D Row and Column Blocked Layout



Generalizes others

6) 2D Row and Column Block Cyclic Layout

Communication Lower Bounds: Prior Work on Matmul

- Assume n^3 algorithm (i.e. not Strassen-like)
- Sequential case, with fast memory of size M
  - Lower bound on #words moved to/from slow memory = $\Omega(n^3 / M^{1/2})$ [Hong, Kung, 81]
  - Attained using blocked or cache-oblivious algorithms

- Parallel case on P processors:
  - Let M be memory per processor; assume load balanced
  - Lower bound on #words moved $= \Omega((n^3/p) / M^{1/2}))$ [Irony, Tiskin, Toledo, 04]
  - If $M = 3n^2/p$ (one copy of each matrix), then lower bound $= \Omega(n^2/p^{1/2})$
  - Attained by SUMMA, Cannon's algorithm

Load Balancing

- Let a task compute C_{ij}
- Divide the task among p_{row} x p_{col} processors
  - Lay them in 2D and represent a processor in x^{th} row and y^{th} column as Proc(x,y). $1 <= x <= p_{row}$ $1 <= y <= p_{col}$

- Load Balancing (balancing the arithmetic computation assigned to processors)
  1. 2D block distribution
     - Assigns continuous block updates
  2. 2D block cyclic distribution
     - Assigns blocks of C in strides of p_{row} (along y) and p_{col} (along x)

Load Balancing - Block Distribution

Proc(1,1)	Proc(1,2)	Proc(1,3)
$\begin{Bmatrix} C_{11} & C_{12} & C_{13} \\ C_{21} & C_{22} & C_{23} \\ C_{31} & C_{32} & C_{33} \\ C_{41} & C_{42} & C_{43} \end{Bmatrix}$	$\begin{Bmatrix} C_{14} & C_{15} & C_{16} \\ C_{24} & C_{25} & C_{26} \\ C_{34} & C_{35} & C_{36} \\ C_{44} & C_{45} & C_{46} \end{Bmatrix}$	$\begin{Bmatrix} C_{17} & C_{18} & C_{19} \\ C_{27} & C_{28} & C_{29} \\ C_{37} & C_{38} & C_{39} \\ C_{47} & C_{48} & C_{49} \end{Bmatrix}$
Proc(2,1)	**Proc(2,2)**	**Proc(2,3)**
$\begin{Bmatrix} C_{51} & C_{52} & C_{53} \\ C_{61} & C_{62} & C_{63} \\ C_{71} & C_{72} & C_{73} \\ C_{81} & C_{82} & C_{83} \end{Bmatrix}$	$\begin{Bmatrix} C_{54} & C_{55} & C_{56} \\ C_{64} & C_{65} & C_{66} \\ C_{74} & C_{75} & C_{76} \\ C_{84} & C_{85} & C_{86} \end{Bmatrix}$	$\begin{Bmatrix} C_{57} & C_{58} & C_{59} \\ C_{67} & C_{68} & C_{69} \\ C_{77} & C_{78} & C_{79} \\ C_{87} & C_{88} & C_{89} \end{Bmatrix}$

- $M=8$, $N=9$, $p_{row}=2$, $p_{col}=3$

Load Balancing - Block Cyclic Distribution

Proc(1,1) $\left\{\begin{matrix} C_{11} & C_{14} & C_{17} \\ C_{31} & C_{34} & C_{37} \\ C_{51} & C_{54} & C_{57} \\ C_{71} & C_{74} & C_{77} \end{matrix}\right\}$	Proc(1,2) $\left\{\begin{matrix} C_{12} & C_{15} & C_{18} \\ C_{32} & C_{35} & C_{38} \\ C_{52} & C_{55} & C_{58} \\ C_{72} & C_{75} & C_{78} \end{matrix}\right\}$	Proc(1,3) $\left\{\begin{matrix} C_{13} & C_{16} & C_{19} \\ C_{33} & C_{36} & C_{39} \\ C_{53} & C_{56} & C_{59} \\ C_{73} & C_{76} & C_{79} \end{matrix}\right\}$
Proc(2,1) $\left\{\begin{matrix} C_{21} & C_{24} & C_{27} \\ C_{41} & C_{44} & C_{47} \\ C_{61} & C_{64} & C_{67} \\ C_{81} & C_{84} & C_{87} \end{matrix}\right\}$	Proc(2,2) $\left\{\begin{matrix} C_{22} & C_{25} & C_{28} \\ C_{42} & C_{45} & C_{48} \\ C_{62} & C_{65} & C_{68} \\ C_{82} & C_{85} & C_{88} \end{matrix}\right\}$	Proc(2,3) $\left\{\begin{matrix} C_{23} & C_{26} & C_{29} \\ C_{43} & C_{46} & C_{49} \\ C_{63} & C_{66} & C_{69} \\ C_{83} & C_{86} & C_{89} \end{matrix}\right\}$

- M=8, N=9, p_{row} =2, p_{col} =3

Exercise

- Are the 2D block- and block-cyclic- distribution schemes load balanced for C=C+AB update?
- Scenario 1: A is lower-triangular and B is upper-triangular
  - Block distribution:
    - Load balance depends on number of processors (gets worse with increasing number of processors)
  - Block-cyclic distribution
    - Increasingly balanced as the problem size grows
- Scenario 2: First few rows of A are zeros. First few columns of B are zeros

Load Balance and C=AB

- When A is lower triangular, and B is upper triangular:

$$\begin{bmatrix} A11 & 0 & 0 & 0 \\ A21 & A22 & 0 & 0 \\ A31 & A32 & A33 & 0 \\ A41 & A42 & A43 & A44 \end{bmatrix} x \begin{bmatrix} B11 & B12 & B13 & B14 \\ 0 & B22 & B23 & B24 \\ 0 & 0 & B33 & B34 \\ 0 & 0 & 0 & B44 \end{bmatrix} = \begin{array}{c} P11 \\ \\ P21 \end{array} \left[\begin{array}{cc:cc} C11 & C12 & C13 & C14 \\ C21 & C22 & C23 & C24 \\ \hdashline C31 & C32 & C33 & C34 \\ C41 & C42 & C43 & C44 \end{array}\right] \begin{array}{c} P12 \\ \\ P22 \end{array}$$

C11 = A11xB11	C12 = A11xB12	. .
C21 = A21xB11	C22 = A21xB12+A22xB22	. .
C31 = A31xB11	C32 = A31xB12+A32xB22	. .
C41 = A41xB11	C42 = A41xB12+A42xB42	. C44= A41xB14+A42xB24+ A43xB34+A44xB44

If Pxy denotes a processor and 1<=x<= p_{row} 1<=y<= p_{col} ,
for prow=2, pcol=2, P22 does the most work and P11 does the least work if block-distribution is followed.

Load Balance and C=AB

- When A is lower triangular, and B is upper triangular:

$$\begin{bmatrix} A11 & 0 & 0 & 0 \\ A21 & A22 & 0 & 0 \\ A31 & A32 & A33 & 0 \\ A41 & A42 & A43 & A44 \end{bmatrix} \text{x} \begin{bmatrix} B11 & B12 & B13 & B14 \\ 0 & B22 & B23 & B24 \\ 0 & 0 & B33 & B34 \\ 0 & 0 & 0 & B44 \end{bmatrix} = \begin{bmatrix} C11 & C12 & C13 & C14 \\ C21 & C22 & C23 & C24 \\ C31 & C32 & C33 & C34 \\ C41 & C42 & C43 & C44 \end{bmatrix}$$

C11 = A11xB11   C12 = A11xB12   .   .
C21 = A21xB11   C22 = A21xB12+A22xB22   .   .
C31 = A31xB11   C32 = A31xB12+A32xB22   .   .
C41 = A41xB11   C42 = A41xB12+A42xB42   .   C44= A41xB14+A42xB24+ A43xB34+A44xB44

if block-cyclic distribution is followed:
P11 gets updates C11, C13, C31, and C33
P22 gets updates C22, C42, C24, and C44

Model for Communication Costs in MatMul

- Let there be p processors doing `C=C+AB (A=mxr, B=rxn and C=mxn. Also A=MxR, B=RxN, C=MxN)`
- Assume `block-cyclic` distribution (so that arithmetic cost is evenly spread among all processors (i.e. load balanced))
- Let individual processors perform `Cij=Cij + Aik * Bkj @ F` flops/sec
- Time to move `w` words into and out of processor's memory = `l + b*w (l=latency b=bandwidth)`

Time spent by each processor on doing the computation: T_{arith} (p) ~ `(2mnr/p) / F`

- `Let` T_{data}(p) be the time that each processor spends in acquiring the data (communication cost)

Speedup with p processors: $S(P) \approx \frac{T_{arith}(1)}{T_{arith}(p)+T_{data}(p)} = \frac{p}{1+\frac{T_{data}(p)}{T_{arith}(P)}}$

Communication to compute ratio

Model for Communication Costs in MatMul

- An individual processor performing `Cij=Cij + Aik * Bkj` would require blocks:
  - `Cij` (count=Num_{Cij})
  - `Ai1, Ai2, . ., AiR` (count=Num_{Aij})
  - `B1j, B2j, . ., BRj` (count=Num_{Bij})
- Suppose each block is uniformly subdivided as: `m = m1*M` (i.e. every submatrix in A and C (Aij and Cij) would have m1 rows.). Similarly, `r = r1*R` (i.e. every submatrix in A would have r1 columns and submatrix in B would have r1 rows.) and `n=n1*N`.
- Then:
  - Time required to bring one block of C (Cij) into fast memory = `l + b*m1*n1`
  - Time required to bring one block of B (Bij) into fast memory = `l + b*r1*n1`
  - Time required to bring one block of A (Aij) into fast memory = `l + b*m1*r1`
- Therefore, $T_{data}(p) = Num_{Cij}(l + b*m1*n1) + Num_{Bij}(l + b*r1*n1) + Num_{Aij}(l + b*m1*r1)$

Model for Communication Costs in MatMul

- The blocked MatMul:

```
for i=x:prow:M //means increment of prow along y direction
      for j=y:pcol:N //increment of pcol along x direction
            //Read Cij into fast memory
            for k=1:R
                  //Read Aik into fast memory
                  //Read Bkj into fast memory
                  //perform update:
                  Cij=Cij + Aik*Bkj
           //Write Cij back to slow memory
```

- This algorithm has Num_{Cij}=2MN/p, Num_{Aij}=RMN/p, Num_{Bij}=RMN/p
- Communication to compute ratio: $\frac{T_{data}(p)}{T_{arith}(P)} \approx \frac{F}{2}\left(l\,\frac{(2+2R)}{m1n1r} + b\,\left(\frac{2}{r} + \frac{1}{n1} + \frac{1}{m1}\right)\right)$
- Conclusion:
  - Speedup degrades as floprate F increases.
  - Speedup improves if latency and bandwidth (l and b) decrease or m1, n1, r1 increase

Model for Communication Costs in MatMul

- Observations:
  - Communication to compute ratio for the previous matmul method doesn't depend on number of processors
  - Fast Memory Capacity vs. Communication cost tradeoff
    - Each submatrix of A loaded N/pcol times
    - Each submatrix of B loaded M/prow times

What if each task computing Cij can get all the required submatrices in fast memory? i.e. the fast memory is big enough so that once loaded, submatrices stay there till end of computation

  - Each processor could access C, A, and B matrices.
    - Shared-memory model.

What if A, B, and C together don't fit in RAM?

Even if it fits, it takes enormous amount of time to do the computation?