CS601: Software Development for Scientific Computing

Autumn 2022

Week6: Motifs – Matrix Computations with Dense and Sparse Matrices, Accelerating computation with FFTs

Last week..

- Three fundamental ways to multiply two matrices
  - Comprising of dot products, linear combination of the left matrix columns, outer product updates
    - Commonly occurring algorithmic patterns / kernels :

      Dot product, AXPY and GAXPY, outer product, matrix-vector product, matrix-matrix product
- Linear algebra software (BLAS, LAPACK)
  - BLAS routines and categorization
- Algorithmic costs
  - Arithmetic cost
  - Data movement cost
- Computational intensity (examples: `axpy, matvec, matmul`)

Last week - Communication Cost

```
//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 from slow memory to fast memory once in **inner two loops.**
- Each column of B read n times including **outer i loop =** n^3 words read

Last week – Computational Intensity of Matmul (ijk)

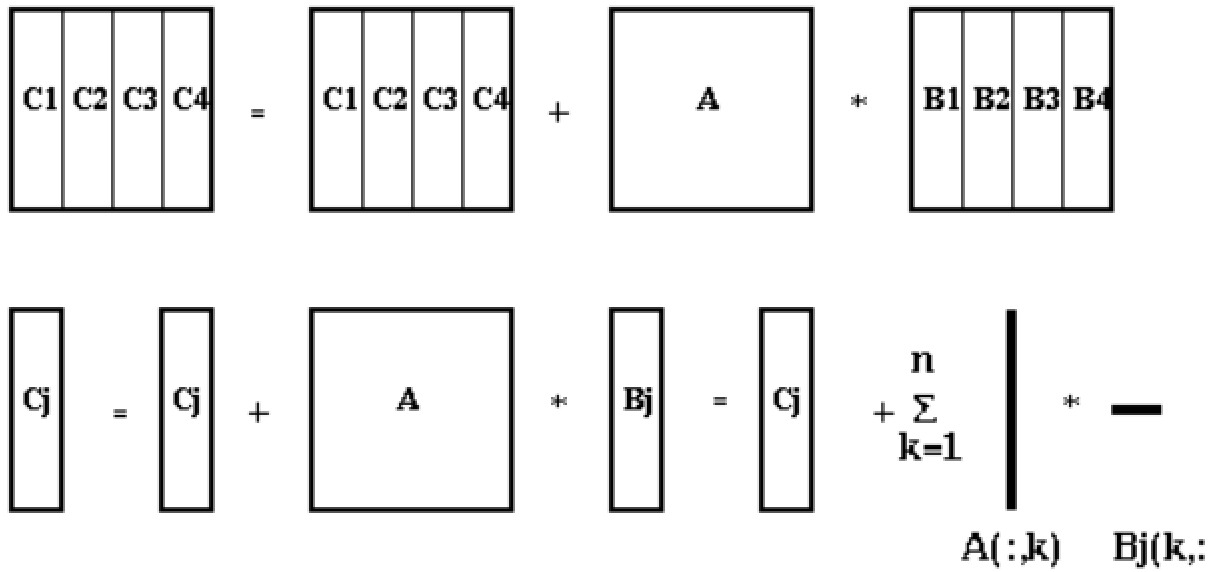
- Words moved = $n^3+3n^2 = n^3+O(n^2)$
- Number of arithmetic operations = $2n^3$ (from slide 35)
- computational intensity $q \approx 2n^3/n^3 = 2$. (computation to communication ratio)

  Same as q for matrix-vector?
  What if the fast memory has more space ? more than just two columns + one element space?

- Can we do better?

Last week - 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

Last week – Computational Intensity

```
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

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

$\downarrow q \; \xrightarrow{r}$ $\quad$ $\downarrow q \; \xrightarrow{p}$ $\quad$ $\downarrow p \; \xrightarrow{r}$

- $A, B, C \in \mathbb{R}^{n \times n}$
- 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
- Can we design algorithms that are independent of cache size?

Recursive Matrix Multiply

- Cache-oblivious algorithm
  - No matter what the size of the cache is, the algorithm performs at a near-optimal level
- Divide-conquer approach

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{bmatrix}$$

- Apply the formula recursively to $A_{11}B_{11}$ etc.
  - Works neat when n is a power of 2.
- What layout format is preferred for this algorithm?
  - Row-major or Col-major? Neither.

Recursive Matrix Multiply

- Cache-oblivious Data structure

$$\begin{bmatrix} 1 & 2 & 5 & 6 & 17 & 18 & 21 & 22 \\ 3 & 4 & 7 & 8 & 19 & 20 & 23 & 24 \\ 9 & 10 & 13 & 14 & 25 & 26 & 29 & 30 \\ 11 & 12 & 15 & 16 & 27 & 28 & 31 & 32 \\ 33 & 34 & 37 & 38 & 49 & 50 & 53 & 54 \\ 35 & 36 & 39 & 40 & 51 & 52 & 55 & 56 \\ 41 & 42 & 45 & 46 & 57 & 58 & 61 & 62 \\ 43 & 44 & 47 & 48 & 59 & 60 & 63 & 64 \end{bmatrix}.$$

- Matrix entries are stored in the order shown
  - E.g. row-major would have 1-8 in the first row, followed by 9-16 in the second and so on.