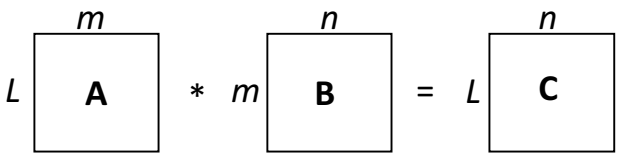
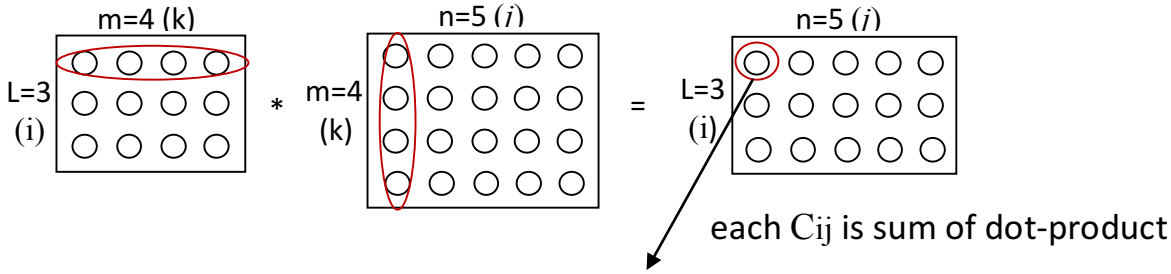
Matrix multiplication



ex) (3*4) * (4*5) = (3*5)



each Cij is sum of dot-product

$Cij = \sum_{k=0}^{m-1} A_{ik} * B_{kj}$

Sequential algorithm:

```
globals:  A[L][m], B[m][n], C[L][n]
. . . .
for (i=0 ~ L-1) do        //for each row in C
   for (j=0 ~ n-1) do     //for each ele. in a row
      temp = 0;   //flush sum of dot-product
      for (k=0 ~ m-1) do
          temp += A[i][k] * B[k][j];     //compute sum of dot-product
      C[i][j] = temp;
```

Parallel approaches:

1. rotation way thread assignment (scheduling)
2. block way thread assignment

Rotation way:

globals: A[L][m], B[m][n], C[L][n]

. . . .

for all P_q, where $1 \le q \le P$ do *//P is tot # of threads*

  for (i=q ~ L, step P) do *//for(i=my_rank; i<L; i+=P)*

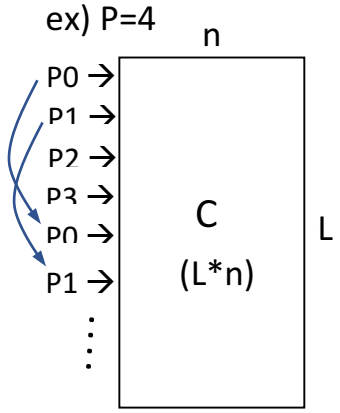
    for (j=0 ~ n-1) do

      temp = 0; //flush

      for (k=0 ~ m-1) do

        temp += A[i][k] * B[k][j];

      C[i][j] = temp;



Block division way: dividing L by num_threads

Each thread processes $\frac{L}{P}$ consecutive rows of C.

L is not evenly divisible by P case, e.g., L=10, P=4

  row index of C: [0] [1] [2] [3] [4] [5] [6] [7] [8] [9]

  P0 P1 P2 P3

  recall HW1:

    Q = L / P; *//quotient*

    R = L % P; *//remainder*

    If (my_rank < R) *//assign 1 more row*

      my_first_index = my_rank * (Q+1);

      my_last_index = my_first_index + Q;

    else

      my_first_index = my_rank * Q + R;

      my_last_index = my_first_index + (Q-1);

  then,

    for (i=my_first_index; i<=my_last_index; i++)

    . . . .