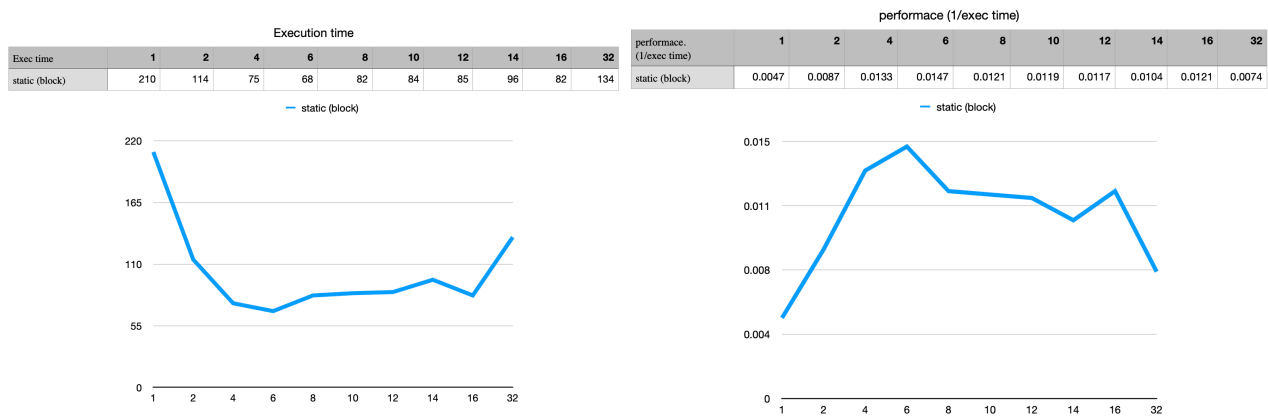


## problem 2. Parallel matrix multiplication with static load balancing approach

### (a) experiment environment

- CPU type : Apple M1
- number of cores : 8 core CPU
- memory size : 16 GB
- OS : macOS Monterey

### (b) execution time (ms) per the number of entire threads = {1,2,4,6,8,10,12,14,16,32}.



Because this experiment was done at 8-core cpu environment, performance improves until 6-8 threads. Interesting point is that more than 8 threads, the execution time rather increases. In my opinion, excessive number of threads doesn't always guarantee better performance.

At 8-core cpu environment, It seems that for example 32 threads (more than 8) make worse queueing delay time.

### (c) Analysis

```

3 import java.util.*;
4
5 class ComputeThread extends Thread {
6     int[][] a;
7     int[][] b;
8     BlockedMat c;
9     public int from;
10    public int to;
11    long timeDiff = 0;
12
13    ComputeThread(int[][] a, int[][] b, BlockedMat c, int f, int t) {
14        this.a = a;
15        this.b = b;
16        this.c = c;
17        from = f;
18        to = t;
19    }
20    long getTimeDiff() {
21        return timeDiff;
22    }
23    public void run() { // overriding, must have this type
24        long s = System.currentTimeMillis();
25        int d = c.col;
26        for (int i = from; i <= to; i++) {
27            int sum = 0;
28            for (int j = 0; j < b.length; j++) {
29                sum += a[i/d][j] * b[j][i%d];
30            }
31            c.setPartialSum(i/d, i%d, sum);
32        }
33        long e = System.currentTimeMillis();
34        timeDiff = e - s;
35    }
36 }
37 }
38
39 class BlockedMat {
40     public int row;
41     public int col;
42     public int[][] mat;
43
44     BlockedMat(int r, int c) {
45         row = r;
46         col = c;
47         mat = new int[row][col];
48     }
49     synchronized void setPartialSum(int r, int c, int sum) {
50         mat[r][c] = sum;
51     }
52     public void printMatrix() {
53         System.out.println("Matrix["+mat.length+"["+mat[0].length+"]");
54         int rows = mat.length;
55         int columns = mat[0].length;
56         int sum = 0;
57         for (int i = 0; i < rows; i++) {
58             for (int j = 0; j < columns; j++) {
59                 System.out.printf("%4d ", mat[i][j]);
60                 sum += mat[i][j];
61             }
62             System.out.println();
63         }
64         System.out.println();
65         System.out.println("Matrix Elements Sum = " + sum + "\n");
66     }
67 }

```

```

76 public class MatmultD {
77     private static Scanner sc = new Scanner(System.in);
78     private static BlockedMat multRes;
79
80     public static void main(String [] args) {
81         // set thread nums.
82         int thread_no=0;
83         if (args.length==1) thread_no = Integer.valueOf(args[0]);
84         else thread_no = 1;
85
86         // read matrix a, b from .txt file
87         int a[][]=readMatrix();
88         int b[][]=readMatrix();
89
90         multRes = new BlockedMat(a.length, b[0].length);
91         ComputeThread[] ct = new ComputeThread[thread_no];
92         int totalNeededComputation = a.length * b[0].length;
93
94         // if there's too much thread set appropriate thread num;
95         if (totalNeededComputation < thread_no) {
96             System.out.println("Too much thread.. downgrade");
97             thread_no = totalNeededComputation;
98         }
99         int d = totalNeededComputation / thread_no;
100
101         long startTime = System.currentTimeMillis();
102         for (int i = 0; i < thread_no; i++) {
103             if (i == thread_no - 1) {
104                 ct[i] = new ComputeThread(a, b, multRes, i*d, totalNeededComputation-1);
105             } else {
106                 ct[i] = new ComputeThread(a, b, multRes, i*d, (i+1)*d - 1);
107             }
108             ct[i].start();
109         }
110
111         // wait the other threads
112         try {
113             for (int i = 0; i < thread_no; i++) {
114                 ct[i].join();
115             }
116         } catch (InterruptedException e) { System.out.println("Error occurred!");}
117         long endTime = System.currentTimeMillis();
118
119         multRes.printMatrix();
120
121         System.out.printf("Total number of threads: %d\n", thread_no);
122         System.out.printf("Calculation Time: %d ms\n", endTime-startTime);
123
124         for (int i = 0; i < thread_no; i++) {
125             System.out.printf("[thread_no]:%2d, [Time]:%4d ms, [Range]:%4d...%4d\n", i, ct[i].getTimeDiff(), ct[i].from, ct[i].to);
126         }
127     }
128
129     public static int[][] readMatrix() {
130         int rows = sc.nextInt();
131         int cols = sc.nextInt();
132         int[][] result = new int[rows][cols];
133         for (int i = 0; i < rows; i++) {
134             for (int j = 0; j < cols; j++) {
135                 result[i][j] = sc.nextInt();
136             }
137         }
138         return result;
139     }
140 }

```

```

problem2 -- -zsh -- 84x25
30 536 629 599 589 490 514 549 517 577 505 501 534 534 561 493 558
562 517 513 532 474 512 567 524 541 589 566 523 514 502 574 559 561
585 576 549 568 517 556 535 526 563 557 569 517 555 507 536 518 499
546 540 502 584 547 565 549 543 529 502 552 546 587 589 552 563 52
7 540 541 563 506 514 512 545 564 532 511 542 551 524 532 569 539 5
37 525 468 538 509 529 573 573 514 505 530 507 551 548 493 518 560
524 538 564 514 487 545 513 506 513 527 520 517 537 531 488 497 566
539 528 536 516 549 511 550 552 503 545 514 541 529 537 514 554 511
675 546 486 542 552 526 536 533 577 570 541 521 517 506 557 547 52
3 554 559 554 544 521 512 533 544 529 542 529 572 521 574 545 542 5
45 540 556 519 517 593 529 476 546 528 509 559 518 535 529 548 567
511 540 558 581 537 504 540 498 517 546 557 570 514 515 512 537 560
525 554 554 554 533 550 529 551 541 561 511 510 544 495 519 537 552
567 537 496 538 533 519 479 556 535 511 502 527 553 540 546 548 54
8 509 585 540 510 530 569 524 529 530 514 517 535 534 561 547 536 5
52 538 552 546 567 551 548 516 578 526 512 497 520 502 558 528 548
532 529 531 536 535 535 580 534 528 524 561 484 536

Matrix Elements Sum = 125231132

Total number of threads: 2
Calculation Time: 113 ms
[thread_no]: 0, [Time]: 113 ms, [Range]: 0...124999
[thread_no]: 1, [Time]: 113 ms, [Range]: 125000...249999
(base) seokwonyoon@yunsog-won-ui-MacBookPro problem2 %

```

```

problem2 -- -zsh -- 84x25
505 576 549 560 517 556 535 526 563 557 569 517 555 507 536 510 499
546 540 502 504 547 565 549 548 529 502 552 546 587 589 552 563 52
7 540 541 563 506 514 512 545 564 532 511 542 551 524 532 569 539 5
37 525 468 538 509 529 573 573 514 505 530 507 551 548 493 518 560
524 538 564 514 487 545 513 506 513 527 520 517 537 531 488 497 566
539 528 536 516 549 511 550 552 503 545 514 541 529 537 514 554 511
575 546 486 542 552 526 536 533 577 570 541 521 517 506 557 547 52
3 554 559 554 544 521 512 533 544 529 542 529 572 521 574 545 542 5
45 540 556 519 517 593 529 476 546 528 509 559 518 535 529 548 567
511 540 558 581 537 504 540 498 517 546 557 570 514 515 512 537 560
525 554 554 554 533 550 529 551 541 561 511 510 544 495 519 537 552
567 537 496 538 533 519 479 556 535 511 502 527 553 540 546 548 54
8 509 585 540 510 530 569 524 529 530 514 517 535 534 561 547 536 5
52 538 552 546 567 551 548 516 578 526 512 497 520 502 558 528 548
532 529 531 536 535 535 580 534 528 524 561 484 536

Matrix Elements Sum = 125231132

Total number of threads: 4
Calculation Time: 65 ms
[thread_no]: 0, [Time]: 65 ms, [Range]: 0...62499
[thread_no]: 1, [Time]: 64 ms, [Range]: 62500...124999
[thread_no]: 2, [Time]: 65 ms, [Range]: 125000...187499
[thread_no]: 3, [Time]: 65 ms, [Range]: 187500...249999
(base) seokwonyoon@yunsog-won-ui-MacBookPro problem2 %

```

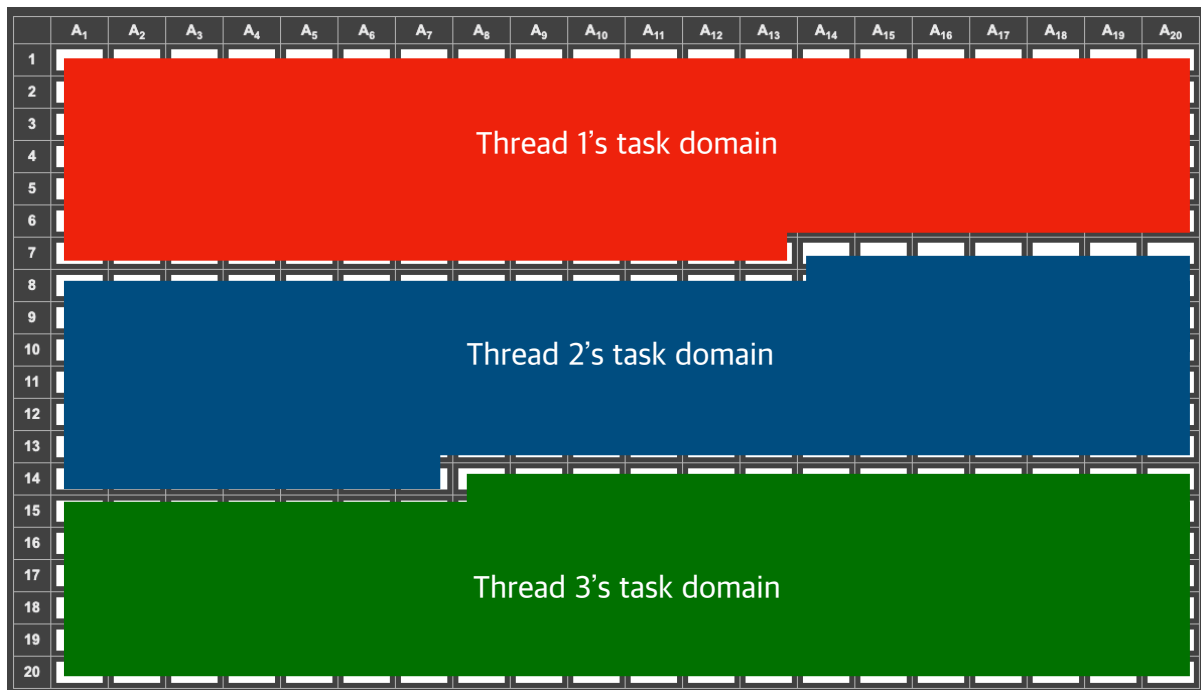
2 threads

4 threads

It seems that the loads allocated to each thread are very similar.

Let's say we multiply two matrices  $A(m * n)$  and  $B(n * k)$ . Then the result matrix  $C$  will be  $m * k$ . It means that we have to calculate  $m * k$  elements to get result  $C$ .

If you use " $t$ " number of threads, then divide  $C$ 's elements into  $t$  bunch of tasks for  $t$  threads.



For example, let's say the result matrix  $A$ 's shape is 20 by 20. If you use 3 threads. Each of their task domain will be like the image above. Each thread calculate their separated task domain.

#### (d) How to compile and execute the source code

At terminal,

(\*) `javac MatmultD.java`

After compilation execute the program.

(\*) `java MatmultD 6 < mat500.txt`

6 means the number of threads to use, `< mat500.txt` means the file that contains two matrices is given as standard input.

```
problem2 — -zsh — 89x5
(base) seekwonyoon@yunseog-won-ui-MacBookPro problem2 % java MatmultD 6 < mat500.txt
```