**LAB-3**

Prahalad Vijaykumar

2003128

Using mat .c to generate matrices of random size of m\*n and n\*k and then my\_matrix\_multiply.c first accepts three flags -a for matrix a and -b for matrix b and -p for number of threads and then computes the matrix multiplication of the two.

Using 1 D array to represent a 2d array and then using a thread to execute each row if the threads are greater than or equal to number of rows or else threads repeatedly execute the rows.

And the issue faced is when the multiplication between threads are ununiform this is because of access of all threads at the same time which will lead to skipping of the iteration.

Eg 0,1,1,3…this has happened because two threads have accessed 1 at nearly the same time the iter variable(reference of iter variable in the code.). so to avoid this there are two cases compare and swap principle and mutex lock. Since it involves using atomic variable and mutex lock is easier to implement so I implemented mutex lock.

Mutex lock works just like a lock in the sense if two threads come towards accessing the same function then it gives access to one of the thread and the other thread has to wait until the previous thread completes its execution in c mutex\_thread\_lock() and unlock() are used respectively to implement the same

Observation :

For smaller matrix it is better to work normally than performing multithreading

For larger matrix of size greater than 2500 I found a major difference when comes to execution time . though it takes a lot of time to executed.

prahaladvijaykumar$ ./my\_matrix\_multiply -a i -b o -t 1

213255720

123437373

prahaladvijaykumar$ ./my\_matrix\_multiply -a i -b o -t 100

207723403

56871058

Size of i and o is 2500\*2500

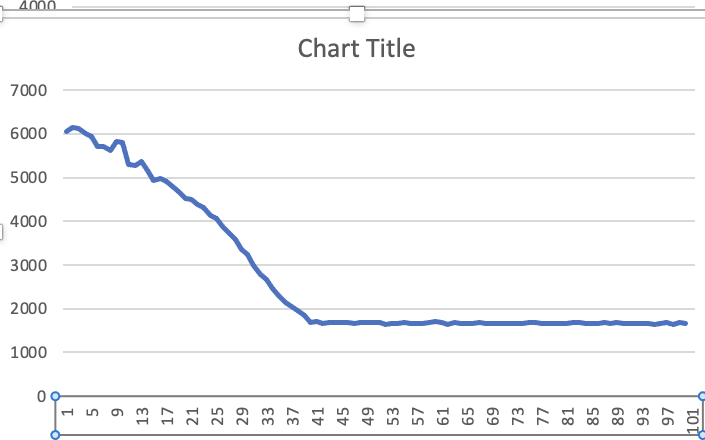
prahaladvijaykumar$ ./my\_matrix\_multiply -a i -b o -t 1

7

549

Size of i and o is 10\*10

The first output is the time taken without multithreading and the second is with multithreading



Plot for multiplying 40\*40 matrix for random threads from 1-100.

Chart, line chart

Description automatically generated

100\*100 matrix result on increasing threads from 1-100

Each time is taken by running each number of thread 100 times and then taking an average and reporting in the graph and plotted it using excel.

512

354

529

359

474

385

517

320

385

558

513

465

354

360

526

496

508

492

494

502

338

434

531

467

520

514

442

513

490

536

527

527

470

528

569

477

434

586

347

360

515

572

581

544

501

571

521

495

368

373

367

439

442

374

482

493

528

571

498

540

432

386

564

506

454

455

591

469

473

499

593

503

503

517

505

494

505

458

548

464

477

539

436

524

471

532

468

435

457

488

496

510

648

549

563

518

480

462

491

503

If u take a bigger matrix then it can be observed that a decreasing graph will be obtained