VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

Programme	:	**BTech. CSE Core**	Semester	:	**Fall 2022-23**
Course	:	**Parallel and Distributed Computing**	Code	:	**CSE4001**
Faculty	:	**Dr. Kumar R**	Slot	:	**D2**
Name	:	**Abhishek N N**	Register No.	:	**20BCE1025**

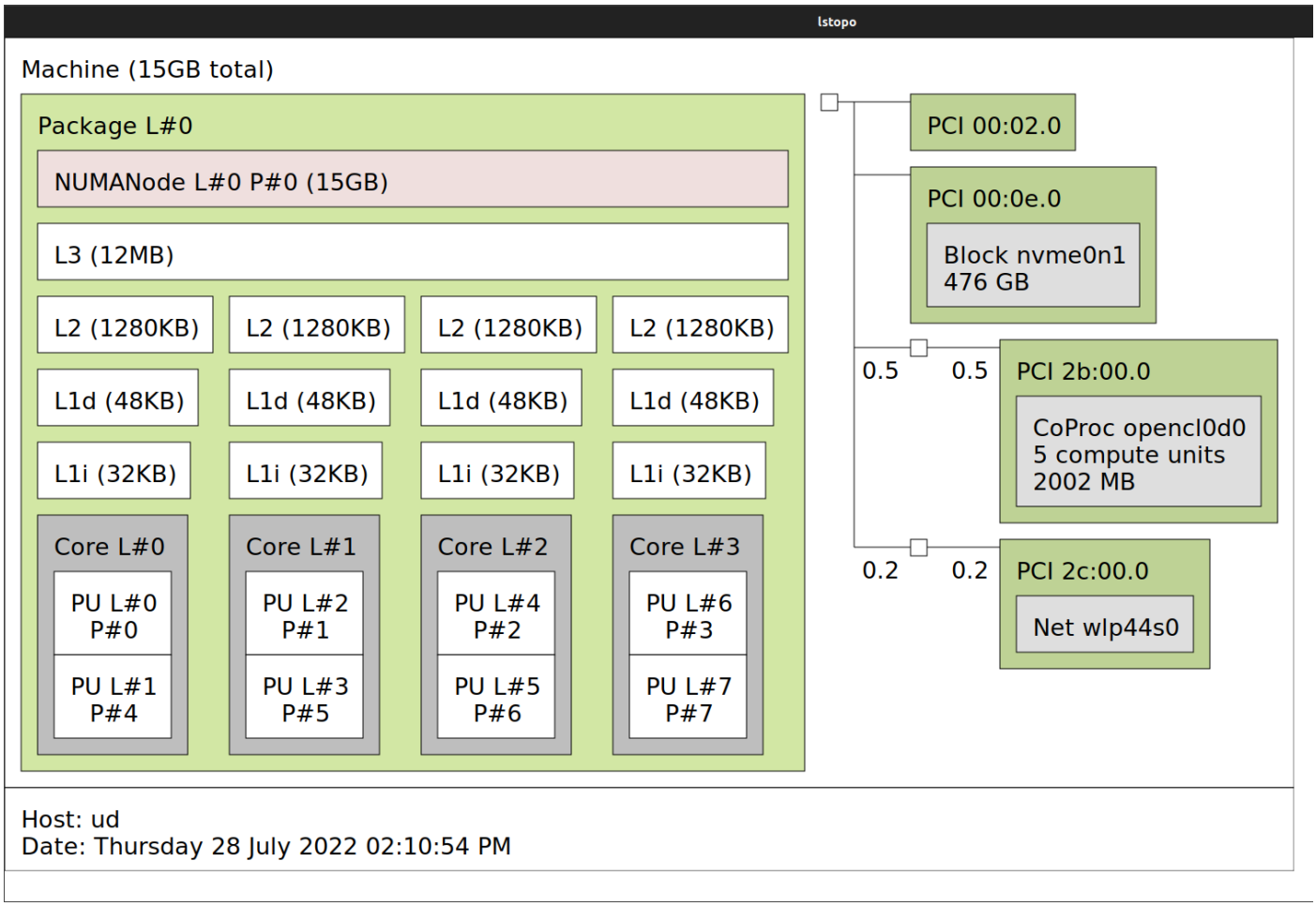
Name : Abhishek N N
Reg.no : 20BCE1025
Email : abhishek.nn2020@vitstudent.ac.in
Mob : 9482992712

CSE4001 Parallel and Distributed Computing

1. Display the processors layout of your system

```
abhishek_n_n_20bce1025@ud:~$ sudo apt install hwloc
```

```
abhishek_n_n_20bce1025@ud:~$ lstopo
hwloc/linux: Ignoring PCI device with non-16bit domain.
Pass --enable-32bits-pci-domain to configure to support such devices
```



2. Write a multithreaded-thread program in **C** to create 10k, 20k, and 50k threads and measure the time taken for each thread group.

```c
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

void *thread_function(void *arg) {}

int main() {
  int no[3] = {10000, 20000, 50000};

  for (int n = 0; n < 3; n++) {
    clock_t begin = clock();
    pthread_t threads[no[n]];
    int rc, i;

    for (i = 0; i < no[n]; i++) {
      rc = pthread_create(&threads[i], NULL, thread_function, NULL);

      if (rc) {
        printf("Error:unable to create thread, %d\n", rc);
        exit(-1);
      }
      pthread_join(threads[i], NULL);
    }
    clock_t end = clock();
    double time_spent = (double)(end - begin) / CLOCKS_PER_SEC;
    printf("%d threads: %f miliseconds\n", no[n], time_spent);
  }
}
```

```
abhishek_n_n_20bce1025@ud:/mnt/D/ccpp$ gcc thread_time.c -pthread
abhishek_n_n_20bce1025@ud:/mnt/D/ccpp$ ./a.out
10000 threads: 0.161338 miliseconds
20000 threads: 0.284242 miliseconds
50000 threads: 0.701180 miliseconds
abhishek_n_n_20bce1025@ud:/mnt/D/ccpp$
```

3. Write a program to create two threads. Thread1 has to print the print String1 “PDC” and Thread2 has to print the String2 “Lab”.
   Hint: Implement the message_function () is used as the start routine for the threads used it accepts a void pointer.
   Sample output:
   Thread1 prints: PDC
   Thread2 prints: Lab

```c
#include <pthread.h>
#include <stdio.h>

void *message_function(void *arg) {
	char *msg = (char *)arg;
	printf("%s", msg);
}

int main() {
	pthread_t thread1, thread2;
	char *msg1 = "PDC", *msg2 = " Lab\n";
	pthread_create(&thread1, NULL, message_function, (void *)msg1);
	pthread_join(thread1, NULL);
	pthread_create(&thread2, NULL, message_function, (void *)msg2);
	pthread_join(thread2, NULL);
	return 0;
}
```

```
abhishek_n_n_20bce1025@ud:/mnt/D/ccpp$ gcc thread_time.c -pthread
abhishek_n_n_20bce1025@ud:/mnt/D/ccpp$ ./a.out
PDC Lab
abhishek_n_n_20bce1025@ud:/mnt/D/ccpp$
```

20BCE1025
Abhishek N N
abhishek.nn2020@vitstudent.ac.in

Lab2 - OpenMP Basics & Vector addition

i) OpenMP Case1 1000 threads by using runtime lib routine

Code:

```
#include <omp.h>
#include <stdio.h>

int main(int argc, char *argcv[]) {
	int pid = 0, numthreads = 1000;
	omp_set_num_threads(numthreads);

	#pragma omp parallel private(pid)
	{
		printf("%d ", omp_get_thread_num());
		if (omp_get_thread_num() == pid) {
		printf("master");
	}
	}
	return 0;
}
```

Output:

```
abhishek_n_n_20bce1025@ud:/mnt/D/ccpp/Assignment$ gcc -fopenmp -o case1 case1.c
abhishek_n_n_20bce1025@ud:/mnt/D/ccpp/Assignment$ ./case1
13 57 295 751 693 394 695 305 20 753 395 696 309 93 756 21 397 699 311 96 895 393 398 313 336 98 391 317 399 335 104 390 325 400 334 1 3
89 109 333 401 332 3 388 110 357 402 331 4 387 365 114 403 2 386 330 367 116 404 5 384 329 373 405 117 328 6 382 375 406 327 120 380 7 3
77 407 326 379 124 8 381 408 324 378 9 383 126 410 323 376 10 385 411 322 127 374 11 392 412 321 128 372 14 396 413 320 371 129 15 580 4
14 319 370 585 131 415 17 318 369 588 416 133 368 316 18 591 417 315 366 134 19 596 418 314 364 24 144 599 419 312 363 22 601 145 420 31
0 362 23 605 421 308 146 361 26 615 422 307 360 147 27 623 424 306 359 28 149 629 426 304 358 30 633 150 303 356 427 29 639 355 428 154
302 31 641 354 429 33 301 156 353 645 430 32 300 159 352 649 431 34 299 142 653 160 432 35 298 661 161 433 162 36 959 164 664 435 961 16
5 37 167 708 964 437 38 168 737 968 170 438 39 171 971 694 439 172 41 173 440 973 742 176 40 441 975 178 42 180 745 442 977 43 181 748 1
82 443 980 44 750 183 185 444 982 45 752 445 119 984 46 754 158 101 446 700 647 825 652 197 646 821 701 827 654 644 194 819 668 829 655
200 643 818 199 656 831 642 816 25 202 657 833 640 814 679 835 658 196 638 812 837 204 637 810 205 660 839 636 808 713 206 662 841 806 6
63 209 210 804 665 212 211 802 666 213 214 800 707 215 798 901 216 709 203 903 796 217 220 898 794 905 222 223 224 896 792 908 225 226 8
94 227 790 911 228 229 788 913 893 231 218 786 914 891 234 784 235 916 889 236 782 237 887 920 781 238 239 922 886 779 221 242 923 884 2
43 778 244 246 882 925 776 247 248 254 880 774 927 232 878 772 258 929 259 876 770 930 260 261 874 768 932 262 873 264 766 934 266 871 2
33 764 198 936 268 761 460 869 272 938 271 219 867 457 273 940 267 275 866 453 276 277 941 278 279 864 284 449 285 942 287 862 288 436 2
89 944 290 860 291 293 425 946 294 858 253 274 337 423 948 856 208 339 297 108 409 950 854 338 250 952 191 281 850 680 296 954 184 848 6
73 169 846 676 735 990 153 845 736 988 341 85 151 843 992 986 682 148 714 721 152 842 983 105 711 141 724 188 981 342 95 715 201 979 139
 725 712 195 106 972 137 706 140 689 969 347 731 967 719 102 64 743 722 966 245 732 671 241 965 726 207 729 734 963 348 255 704 962 249
705 193 960 730 747 257 733 958 111 702 340 956 263 738 270 758 746 720 350 252 265 739 763 698 280 900 697 685 283 703 251 907 292 683
346 345 269 755 684 47 985 344 48 51 240 741 136 52 351 434 123 910 50 135 187 448 767 125 760 53 157 349 769 121 765 450 54 112 192 113
 771 451 155 717 97 452 56 775 58 16 777 59 780 115 49 60 994 783 61 63 62 690 787 66 789 65 186 692 495 785 68 795 70 55 797 174 497 67
 498 72 793 499 69 92 749 74 803 502 501 805 73 76 494 807 505 77 75 190 78 809 492 100 493 80 491 813 506 79 81 811 83 815 503 490 510
817 84 488 511 537 489 820 487 512 87 486 533 86 88 485 824 483 525 89 799 826 521 130 90 482 822 519 91 0 480 830 478 999 517 832 516 9
96 477 834 513 998 94 515 475 508 836 166 828 143 177 474 472 838 122 500 840 132 507 470 118 844 484 469 847 523 529 468 481 849 995 46
7 851 466 853 476 465 855 462 857 473 461 859 459 522 861 471 458 863 865 456 455 870 872 524 454 282 875 526 569 567 509 566 879 565 88
1 564 563 883 562 885 561 520 560 890 558 614 557 514 556 528 555 897 617 554 899 553 902 552 568 550 904 549 620 906 619 548 909 547 55
9 622 681 447 546 551 612 624 688 611 544 545 543 616 541 687 610 542 987 609 915 912 540 608 989 539 921 538 626 607 924 991 691 888 53
6 606 535 630 604 99 892 534 593 603 631 532 928 931 602 919 634 877 926 531 600 597 933 595 953 594 598 951 590 993 592 175 949 163 589
 587 947 945 586 584 179 103 943 957 583 581 939 937 12 582 935 723 710 578 716 579 740 627 686 575 577 744 757 632 674 576 759 675 670
677 678 574 648 650 667 823 672 659 256 621 343 669 230 978 625 718 976 651 727 628 852 970 107 138 728 762 791 773 496 518 974 801 504
570 479 618 613 71 463 955 571 918 464 527 82 868 286 997 917 573 572 635 189 530 abhishek_n_n_20bce1025@ud:/mnt/D/ccpp/Assignment$
```

ii) OpenMP Case2 1000 threads by using Compiler directive

Code:

```
#include <omp.h>
#include <stdio.h>

int main(int argc, char *argcv[]) {
	int pid = 0, numthreads = 1000;

	#pragma omp parallel num_threads(numthreads) private(pid)
	{
		printf("%d ", omp_get_thread_num());
		if (omp_get_thread_num() == pid) {
			printf("master");
		}
	}
	return 0;
}
```

Output:

```
abhishek_n_n_20bce1025@ud:/mnt/D/ccpp/Assignment$ gcc -fopenmp -o case2 case2.c
abhishek_n_n_20bce1025@ud:/mnt/D/ccpp/Assignment$ ./case2
0 1 4 2 3 6 5 9 10 12 7 15 14 13 16 17 20 11 8 18 22 21 24 19 25 26 27 28 23 30 32 29 33 31 35 34 36 38 39 37 42 43 40 41 44 45 47 46 50
 51 53 49 55 57 58 52 60 59 61 48 64 65 66 63 67 56 71 69 72 73 74 54 75 79 80 81 76 68 77 82 86 84 62 78 88 85 90 89 91 87 94 93 92 96
100 95 101 98 103 97 102 108 109 111 99 104 107 115 116 105 118 106 117 122 123 110 125 126 113 124 112 131 133 134 114 127 136 138 140
120 121 142 144 146 119 148 129 150 137 151 152 153 135 132 155 158 159 160 130 157 162 166 128 167 168 171 164 83 173 174 175 163 70 17
8 181 182 183 172 185 188 170 189 154 190 192 149 196 184 197 198 200 147 180 204 205 206 145 176 208 212 143 214 215 216 194 141 220 22
2 187 223 139 224 186 228 230 231 232 218 234 210 235 201 236 237 202 242 199 243 246 248 195 238 250 251 254 193 227 256 252 259 226 19
1 258 263 265 179 240 267 261 270 177 272 244 268 274 169 277 276 280 278 282 165 285 284 286 289 161 290 288 292 156 294 297 296 298 30
0 266 299 302 304 305 269 307 306 308 311 271 312 310 313 315 273 314 318 281 323 326 327 283 320 334 335 328 287 321 325 344 291 345 27
5 317 351 293 366 353 279 341 295 370 352 319 339 301 374 337 303 364 372 357 368 309 381 373 385 355 379 377 356 411 442 384 378 418 41
2 387 375 365 398 413 388 367 383 405 414 391 386 435 406 415 401 362 432 416 350 403 390 363 396 407 438 409 399 358 448 395 402 424 45
2 408 455 394 422 463 361 466 426 360 473 444 476 359 400 484 354 487 397 330 410 494 430 496 332 349 440 503 504 443 454 445 512 513 44
6 516 447 517 467 451 524 526 465 453 329 532 461 535 457 324 460 540 459 322 543 458 462 549 316 456 552 464 264 450 558 560 404 468 26
2 392 566 470 260 568 382 257 472 576 578 380 255 475 376 585 586 253 477 369 249 593 479 595 393 247 481 348 602 604 245 347 483 241 61
2 613 343 485 239 621 622 489 233 491 229 493 449 225 625 495 221 633 548 219 498 637 547 641 217 500 545 645 213 502 649 541 211 653 65
4 506 539 209 657 661 508 537 663 207 667 533 510 671 203 673 530 514 677 679 528 681 518 685 523 687 520 689 425 521 522 695 423 519 52
5 421 515 969 971 511 527 981 816 420 994 419 529 417 626 531 628 624 346 629 534 620 631 618 342 635 340 616 639 614 536 643 338 610 60
8 647 336 542 606 651 603 333 655 544 601 659 546 599 660 551 597 662 553 591 555 664 589 666 557 587 434 668 583 670 559 581 562 672 57
9 433 675 577 570 683 575 431 691 573 572 693 571 429 699 569 574 700 567 580 704 565 428 706 564 427 708 584 563 709 582 561 588 711 72
2 556 590 713 554 780 714 592 550 594 716 781 596 782 598 783 861 600 863 605 784 858 865 607 856 785 867 854 609 611 869 852 871 786 61
5 850 873 787 617 848 875 846 619 788 877 844 768 623 879 842 881 627 630 840 883 838 789 632 885 836 790 634 887 834 889 636 791 831 89
1 638 829 792 893 827 640 642 895 825 793 644 897 823 767 899 646 821 900 794 818 648 904 799 650 795 906 732 652 910 740 731 656 913 72
9 741 915 658 727 665 918 726 742 669 920 725 743 922 674 724 676 924 723 744 926 678 721 745 928 680 719 746 930 684 717 694 932 747 68
6 690 750 972 692 738 973 970 696 975 968 682 752 966 977 701 751 979 964 753 707 947 995 945 997 710 943 705 758 941 688 715 940 735 73
4 936 796 934 771 776 778 538 774 761 772 764 763 759 757 755 331 712 728 718 773 436 775 777 439 437 811 441 813 814 779 817 819 999 82
2 809 826 830 832 833 835 837 839 841 843 845 847 849 851 853 855 857 859 860 862 864 866 868 828 870 872 509 507 876 505 815 878 501 88
0 499 882 497 884 492 886 490 888 488 890 486 892 482 894 480 896 478 902 474 908 812 471 804 820 469 737 952 912 765 720 954 766 874 80
7 959 810 798 805 965 967 803 963 980 962 982 960 754 958 984 698 703 986 957 956 953 990 988 951 955 993 938 992 824 949 389 942 939 37
1 898 944 937 901 946 935 903 948 914 917 950 905 806 921 961 907 802 923 974 801 909 925 976 800 911 927 978 808 916 929 983 702 931 74
9 985 987 933 697 989 919 733 756 760 991 730 797 748 996 736 998 739 762 770 769 abhishek_n_n_20bce1025@ud:/mnt/D/ccpp/Assignment$
```

ii) OpenMP Case3 1000 threads by using Environment Variables

Code:

```c
#include <omp.h>
#include <stdio.h>

int main(int argc, char *argcv[]) {
	int pid = 0;

	#pragma omp parallel private(pid)
	{
		printf("%d ", omp_get_thread_num());
		if (omp_get_thread_num() == pid) {
			printf("master");
		}
	}
	return 0;
}
```

Output:

```
abhishek_n_n_20bce1025@ud:/mnt/D/ccpp/Assignment$ export OMP_NUM_THREADS=1000
abhishek_n_n_20bce1025@ud:/mnt/D/ccpp/Assignment$ gcc -fopenmp -o case3 case3.c
abhishek_n_n_20bce1025@ud:/mnt/D/ccpp/Assignment$ ./case3
0 1 2 3 4 6 7 5 10 11 9 13 8 14 16 19 21 12 22 24 17 26 23 15 18 29 30 31 28 20 27 35 36 37 38 33 25 43 32 44 45 46 39 49 50 51 52 34 42
 57 41 58 59 60 40 66 67 68 69 56 64 71 72 73 74 55 63 80 54 81 82 83 62 53 61 89 90 91 92 48 47 98 99 100 101 84 103 104 105 106 94 79
111 112 113 93 114 78 120 88 77 122 123 124 87 76 129 131 133 86 132 75 137 85 70 140 141 142 145 65 148 149 150 152 153 121 154 155 119
 157 158 139 159 160 118 138 165 166 117 167 168 136 116 173 174 135 175 176 115 180 134 181 110 183 184 130 187 109 189 128 192 193 194
 108 197 127 107 200 201 202 126 205 102 209 208 125 210 97 213 217 96 218 219 95 221 222 223 224 229 170 171 231 232 233 169 236 172 23
9 240 164 241 177 243 247 163 178 250 249 251 162 179 254 161 257 258 182 259 262 156 267 266 268 144 270 146 275 274 147 276 143 278 28
3 151 284 285 286 289 288 290 291 235 234 293 294 230 295 296 279 228 301 277 303 304 305 227 273 308 311 226 312 313 272 315 225 318 27
1 320 321 322 220 269 325 216 328 329 265 330 333 215 264 336 337 338 214 347 253 350 349 348 212 355 263 358 363 211 342 366 367 207 37
2 373 374 206 375 380 381 204 383 384 387 203 391 392 256 394 199 399 400 346 401 198 345 196 408 344 411 195 413 416 418 417 191 422 42
4 190 426 427 431 188 432 433 435 186 439 440 441 185 444 448 340 449 450 453 456 457 455 459 460 463 352 467 468 470 351 476 478 477 25
2 341 486 485 339 487 248 495 335 246 496 497 334 245 503 505 506 332 244 510 331 513 242 514 516 327 238 521 524 523 326 237 530 532 32
4 261 533 539 540 323 343 542 255 319 548 549 552 260 317 556 558 316 562 564 567 298 572 571 299 575 297 300 579 580 292 302 584 587 28
7 588 306 592 593 282 590 307 281 599 309 595 280 310 597 606 314 601 607 609 605 611 613 614 603 404 620 621 622 627 628 629 377 631 63
5 376 636 371 633 600 378 641 370 637 643 379 369 648 649 639 651 382 368 650 645 656 365 385 658 652 647 662 364 386 654 665 653 388 66
0 670 361 657 673 389 360 669 678 659 362 390 359 671 661 393 688 357 675 395 663 356 695 677 396 667 354 679 397 703 704 668 353 681 39
8 672 711 687 674 655 676 720 689 760 680 754 691 682 733 755 693 684 715 697 756 716 686 462 699 757 751 461 690 701 758 692 458 683 77
1 759 694 454 685 761 773 696 452 713 762 403 698 451 763 775 776 700 447 764 779 446 783 774 736 781 445 769 794 735 443 785 702 442 73
4 787 803 438 806 732 790 809 437 731 792 436 814 815 795 730 434 821 797 728 822 430 799 727 829 429 831 801 729 428 837 805 840 425 72
6 843 807 423 724 848 851 421 811 721 420 856 813 858 722 419 817 723 866 415 819 725 871 414 874 823 719 412 878 881 828 768 410 830 88
6 887 409 766 832 407 753 892 894 834 406 752 899 836 405 739 902 746 838 906 777 748 841 912 913 778 744 845 920 780 747 847 770 782 92
5 772 784 767 891 786 928 884 889 893 788 936 882 895 402 789 922 880 943 897 791 879 948 950 901 877 903 875 905 872 862 953 954 907 87
0 873 860 959 909 868 960 854 876 865 911 963 852 883 966 863 915 850 971 888 859 917 846 976 890 978 857 919 844 896 855 985 984 842 92
1 898 853 839 989 923 991 900 849 835 927 904 997 464 833 908 465 961 827 910 957 914 824 472 968 466 916 955 473 965 975 918 474 977 95
1 924 967 471 475 979 949 926 973 479 981 947 930 480 983 945 932 481 987 941 934 482 993 939 938 483 995 937 940 484 935 488 942 933 50
8 944 931 509 946 507 929 511 504 952 502 956 501 958 561 500 962 563 560 499 964 559 565 498 557 970 566 494 555 972 568 493 554 974 56
9 492 553 980 570 491 551 982 573 490 550 986 574 547 489 988 576 546 545 577 578 544 820 581 543 818 582 541 816 583 538 812 585 537 81
0 586 536 808 589 535 804 591 534 594 802 531 596 529 800 598 528 798 602 527 796 604 526 793 608 525 992 610 522 994 612 520 615 996 61
6 519 825 617 518 826 618 517 619 998 515 623 512 624 625 718 750 714 737 712 717 738 710 742 709 749 708 741 707 745 706 743 705 740 66
6 765 664 861 864 634 638 867 632 640 869 630 642 885 644 626 969 646 999 990 469 abhishek_n_n_20bce1025@ud:/mnt/D/ccpp/Assignment$
```

2) OpenMP vector addition of 1 lack elements with serial and parallel with 2, 4, 8, 16, 32, 100, 1000 threads to select the fastest

Serial Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define N 100000

void assignRandomValues(int a[], int b[]) {
	for (int i = 0; i < N; i++) {
		a[i] = rand() % 1000;
		b[i] = rand() % 1000;
	}
}

int main(int argc, char *argcv[]) {
	int a[N], b[N], c[N];
	assignRandomValues(a, b);

	clock_t begin = clock();

	for (int i = 0; i < N; i++) {
		c[i] = a[i] + b[i];
	}

	clock_t end = clock();
	double time_spent = (double)(end - begin) / CLOCKS_PER_SEC;
	printf("Serial: %f miliseconds\n", time_spent);

	return 0;
}
```

Serial Output Code:

```
abhishek_n_n_20bce1025@ud:/mnt/D/ccpp/Assignment$ gcc -fopenmp -o vecADD vecADD.c
abhishek_n_n_20bce1025@ud:/mnt/D/ccpp/Assignment$ ./vecADD
Serial: 0.000718 miliseconds
abhishek_n_n_20bce1025@ud:/mnt/D/ccpp/Assignment$
```

Parallel Code:

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define N 100000

void assignRandomValues(int a[], int b[]) {
	for (int i = 0; i < N; i++) {
		a[i] = rand() % 1000;
		b[i] = rand() % 1000;
	}
}
int main(int argc, char *argcv[]) {
	int a[N], b[N], c[N];
	assignRandomValues(a, b);

	int p[] = {2, 4, 8, 16, 32, 100, 1000};

	for (int i = 0; i < 7; i++) {
		clock_t begin = clock();

		#pragma omp parallel private(i) shared(a, b, c)
		{
			#pragma omp for
			for (int j = 0; j < N; j++) {
				c[j] = a[j] + b[j];
			}
		}
		clock_t end = clock();
		double time_spent = (double)(end - begin) / CLOCKS_PER_SEC;
		printf("%d threads: %f miliseconds\n", p[i], time_spent);
	}

	return 0;
}
```

Parallel Output:

```
abhishek_n_n_20bce1025@ud:/mnt/D/ccpp/Assignment$ gcc -fopenmp -o vecADD vecADD.c
abhishek_n_n_20bce1025@ud:/mnt/D/ccpp/Assignment$ ./vecADD
2 threads: 0.002884 miliseconds
4 threads: 0.000444 miliseconds
8 threads: 0.089227 miliseconds
16 threads: 0.020128 miliseconds
32 threads: 0.003935 miliseconds
100 threads: 0.008823 miliseconds
1000 threads: 0.078460 miliseconds
```

.'. 4 threads are best for my machine.

Extra: vector with 10 elements addtion to check OpenMP

Code:

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>

#define n 10

int main() {
	double a[n], b[n], c[n];
	float startTime, endTime, execTime;
	int i;
	int omp_rank;

	startTime = omp_get_wtime();
	#pragma omp parallel private(i) shared(a, b, c)
	{
	#pragma omp for
	for (i = 0; i < n; i++) {
		omp_rank = omp_get_thread_num();
		a[i] = i * (rand() % 1000);
		b[i] = i * (rand() % 1000);
		c[i] = a[i] + b[i];
		printf("The value of a[%d] = %lf and b[%d] = %lf and result
		c[%d] = %lf done by worker Thread ID = %d\n", i, a[i], i,
		b[i], i, c[i], omp_rank);
	}
	}
	endTime = omp_get_wtime();

	execTime = endTime - startTime;
	printf("%f \n", execTime);
	return (0);
}
```

Output:

```
abhishek_n_n_20bce1025@ud:/mnt/D/ccpp$ gcc -fopenmp -o sample sample.c
abhishek_n_n_20bce1025@ud:/mnt/D/ccpp$ ./sample
The value of a[5] = 3450.000000 and b[5] = 295.000000 and result c[5] = 3745.000000 done by worker Thread ID = 7
The value of a[9] = 5841.000000 and b[9] = 243.000000 and result c[9] = 6084.000000 done by worker Thread ID = 7
The value of a[6] = 2298.000000 and b[6] = 5316.000000 and result c[6] = 7614.000000 done by worker Thread ID = 1
The value of a[7] = 2702.000000 and b[7] = 2534.000000 and result c[7] = 5236.000000 done by worker Thread ID = 7
The value of a[4] = 3660.000000 and b[4] = 1968.000000 and result c[4] = 5628.000000 done by worker Thread ID = 7
The value of a[2] = 1586.000000 and b[2] = 842.000000 and result c[2] = 2428.000000 done by worker Thread ID = 7
The value of a[3] = 1620.000000 and b[3] = 1278.000000 and result c[3] = 2898.000000 done by worker Thread ID = 1
The value of a[0] = 0.000000 and b[0] = 0.000000 and result c[0] = 0.000000 done by worker Thread ID = 0
The value of a[1] = 172.000000 and b[1] = 736.000000 and result c[1] = 908.000000 done by worker Thread ID = 0
The value of a[8] = 6216.000000 and b[8] = 2680.000000 and result c[8] = 8896.000000 done by worker Thread ID = 7
time taken: 0.035156
```

20BCE1025
Abhishek N N
abhishek.nn2020@vitstudent.ac.in

Lab 3 - Loop sharing

1. Write a parallel program in openMP to create 8, 16 and 32 threads using runtime library routines. Construct an array of 10000 elements. Distribute the loop iterations to 32, 64, 128 concurrent threads with a chunk-size of 10, 20 and 50 using static, dynamic, guided and auto scheduling schemes. Find out the odd and even numbers global sum of 10K items. Record your execution times for the abovementioned schemes.

Code:

```
#include <omp.h>

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main() {
	int N;
	printf("Enter N: ");
	scanf("%d", &N);
	int tid, i;
	long int odd_sum = 0, even_sum = 0;
	int A[N];
	for (i = 0; i < N; i++) A[i] = i;

	clock_t t;
	t = clock();

#pragma omp parallel for shared(A, even_sum, odd_sum) num_threads(8)
schedule(static, 10)
	for (i = 0; i < N; i++) {
		if (A[i] % 2 == 0)
		even_sum += A[i];
		else
		odd_sum += A[i];
	}
#pragma omp barrier

	t = clock() - t;
	double time_taken = ((double)t) / CLOCKS_PER_SEC;
	printf("8 threads in static: Even sum:%li Odd sum:%li\n", odd_sum,
	even_sum);
```

```
    printf("Time for static method: %fs\n\n", time_taken);
    odd_sum = 0;
    even_sum = 0;

    t = clock();

#pragma omp parallel for shared(A, even_sum, odd_sum) num_threads(16)
schedule(dynamic, 20)
    for (i = 0; i < N; i++) {
        if (A[i] % 2 == 0)
        even_sum += A[i];
        else
        odd_sum += A[i];
    }
#pragma omp barrier

    t = clock() - t;
    time_taken = ((double)t) / CLOCKS_PER_SEC;
    printf("16 threads in dynamic: Even sum:%li Odd sum:%li\n",
    odd_sum, even_sum);
    printf("Time for dynamic method: %fs\n\n", time_taken);
    odd_sum = 0;
    even_sum = 0;

    t = clock();
#pragma omp parallel for shared(A, even_sum, odd_sum) num_threads(32)
schedule(guided, 50)
    for (i = 0; i < N; i++) {
        if (A[i] % 2 == 0)
        even_sum += A[i];
        else
        odd_sum += A[i];
    }
#pragma omp barrier
    t = clock() - t;
    time_taken = ((double)t) / CLOCKS_PER_SEC;
    printf("32 threads in guided: Even sum:%li Odd sum:%li\n",
    odd_sum, even_sum);
    printf("Time for guided method: %fs\n\n", time_taken);

    return 0;
}
```

Output:

```
abhishek_n_n_20bce1025@ud:/mnt/D/ccpp(main)$ gcc -fopenmp 1.c
abhishek_n_n_20bce1025@ud:/mnt/D/ccpp(main)$ ./a.out
Enter N: 1
8 threads in static: Even sum:0 Odd sum:0
Time for static method: 0.002574s

16 threads in dynamic: Even sum:0 Odd sum:0
Time for dynamic method: 0.113021s

32 threads in guided: Even sum:0 Odd sum:0
Time for guided method: 0.003357s

abhishek_n_n_20bce1025@ud:/mnt/D/ccpp(main)$ ./a.out
Enter N: 2
8 threads in static: Even sum:1 Odd sum:0
Time for static method: 0.001826s

16 threads in dynamic: Even sum:1 Odd sum:0
Time for dynamic method: 0.042402s

32 threads in guided: Even sum:1 Odd sum:0
Time for guided method: 0.003563s

abhishek_n_n_20bce1025@ud:/mnt/D/ccpp(main)$ ./a.out
Enter N: 4
8 threads in static: Even sum:4 Odd sum:2
Time for static method: 0.002797s

16 threads in dynamic: Even sum:4 Odd sum:2
Time for dynamic method: 0.009698s

32 threads in guided: Even sum:4 Odd sum:2
Time for guided method: 0.003269s

abhishek_n_n_20bce1025@ud:/mnt/D/ccpp(main)$ ./a.out
Enter N: 10
8 threads in static: Even sum:25 Odd sum:20
Time for static method: 0.002176s

16 threads in dynamic: Even sum:25 Odd sum:20
Time for dynamic method: 0.021007s

32 threads in guided: Even sum:25 Odd sum:20
Time for guided method: 0.003556s

abhishek_n_n_20bce1025@ud:/mnt/D/ccpp(main)$ ./a.out
Enter N: 10000
8 threads in static: Even sum:10748832 Odd sum:11235196
Time for static method: 0.219813s

16 threads in dynamic: Even sum:8029663 Odd sum:8266358
Time for dynamic method: 0.026199s

32 threads in guided: Even sum:10090835 Odd sum:9796274
Time for guided method: 0.004288s
```

2. Write a parallel program to sort N elements in an array using OpenMP
   i. Bubble Sort

Code:

```
#include <omp.h>

#include <stdio.h>
#include <stdlib.h>

void swap();

int main(int argc, char *argv[]) {
	int SIZE = 10;
	int A[SIZE];
	for (int i = 0; i < SIZE; i++) {
		A[i] = rand() % 100;
		printf(" %d", A[i]);
	}
	printf("\n");

	int N = SIZE;
	int i = 0, j = 0;
	int first;
	double start, end;
	start = omp_get_wtime();
	for (i = 0; i < N - 1; i++) {
		first = i % 2;
#pragma omp parallel for default(none), shared(A, first, N)
		for (j = first; j < N - 1; j += 1) {
			if (A[j] > A[j + 1]) {
				swap(&A[j], &A[j + 1]);
			}
		}
	}
	end = omp_get_wtime();
	for (i = 0; i < N; i++) {
	printf(" %d", A[i]);
	}

	printf("\n-------------------------\n Time Parallel= %f\n", (end -
	start));
}

void swap(int *num1, int *num2) {
	int temp = *num1;
	*num1 = *num2;
	*num2 = temp;
}
```

Output:

```
abhishek_n_n_20bce1025@ud:/mnt/D/ccpp(main)$ gcc -fopenmp 1.c
abhishek_n_n_20bce1025@ud:/mnt/D/ccpp(main)$ ./a.out
 83 86 77 15 93 35 86 92 49 21
 15 21 35 49 77 83 86 86 86 92
-------------------------
 Time Parallel= 0.129913
abhishek_n_n_20bce1025@ud:/mnt/D/ccpp(main)$
```

ii. Quick Sort

Code:

```
#include <stdio.h>

#include <omp.h>
#include <stdlib.h>

void swap(int* a, int* b) {
	int t = *a;
	*a = *b;
	*b = t;
}

int partition(int arr[], int start, int end) {
	int pivot = arr[end];
	int i = (start - 1);

	for (int j = start; j <= end - 1; j++) {
		if (arr[j] < pivot) {
			i++;
			swap(&arr[i], &arr[j]);
		}
	}
	swap(&arr[i + 1], &arr[end]);

	return (i + 1);
}

void quicksort(int arr[], int start, int end) {
	int index;

	if (start < end) {
		index = partition(arr, start, end);
```

```
#pragma omp parallel sections
          {
#pragma omp section
                {
                     quicksort(arr, start, index - 1);
                }
#pragma omp section
                {
                     quicksort(arr, index + 1, end);
                }
          }
     }
}

int main() {
int N = 10;
int arr[N];
for (int i = 0; i < N; i++) {
arr[i] = rand() % 100;
printf(" %d", arr[i]);
}
printf("\n");

quicksort(arr, 0, N - 1);

printf("Array after Sorting is: \n");

for (int i = 0; i < N; i++) {
printf(" %d", arr[i]);
}
printf("\n");
return 0;
}
```

Output:

```
abhishek_n_n_20bce1025@ud:/mnt/D/ccpp(main)$ g++ -fopenmp 1.cpp
abhishek_n_n_20bce1025@ud:/mnt/D/ccpp(main)$ ./a.out
 83 86 77 15 93 35 86 92 49 21
Array after Sorting is:
 15 21 35 49 77 83 86 86 92 93
abhishek_n_n_20bce1025@ud:/mnt/D/ccpp(main)$
```

Abhishek N N

20BCE1025

Programme	:	**B.Tech.(CSE)**	Semester	:	**Fall '22-23**
Course	:	**Parallel and Distributed Computing**	Code	:	**CSE4001**
Faculty	:	**R. Kumar**	Slot	:	**L9+L10**

1. Write a openMP program using section constructs

   Function 1

   Generate 100000 random numbers in an array X and find out the min value.

   Function 2

   Generate 1000 prime numbers using Sieve of Sundaram algorithm

   Record your run times using omp_get_wtime() routine for Function 1 & Function 2.

Code:

```
#include <omp.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
void function1() {
      int arr[100000];
      for (int i = 0; i < 100000; i++) arr[i] = rand() % 100000;
      int min = arr[0];
      for (int i = 1; i < 100000; i++)
      if (arr[i] < min) min = arr[i];
      printf("\nminimum in 100000 random elements is %d\n", min);
}
```

```
void SieveOfSundaram(int n) {
	int nNew = (n - 1) / 2;
	bool marked[nNew + 1];
	memset(marked, false, sizeof(marked));
	for (int i = 1; i <= nNew; i++)
		for (int j = i; (i + j + 2 * i * j) <= nNew; j++)
			marked[i + j + 2 * i * j] = true;

	int primeArr[1000], p = 0;
	if (n > 2) primeArr[p++] = 2;

	for (int i = 1; i <= nNew; i++)
		if (marked[i] == false)
			primeArr[p++] = 2 * i + 1;

	printf("\n some prime numbers are: ");
	for(int i=0;i<10;i++) printf("%d ",primeArr[i]);
	printf("\n");
}

int main() {
#pragma omp parallel sections
	{
#pragma omp section
		{
			double start = omp_get_wtime();
			function1();
			double end = omp_get_wtime();
			printf("time taken function1 in seconds: %f", end
		- start);
		}
#pragma omp section
		{
			double start = omp_get_wtime();
			SieveOfSundaram(1000);
			double end = omp_get_wtime();
			printf("time taken function2 in seconds: %f", end
		- start);
		}
	}
	return 0;
}
```

Output:

```
abhishek_n_n_20bce1025@ud:/mnt/D/ccpp$ gcc -fopenmp 1.c
abhishek_n_n_20bce1025@ud:/mnt/D/ccpp$ ./a.out

 some prime numbers are: 2 3 5 7 11 13 17 19 23 29
time taken function2 in seconds: 0.009828
minimum in 100000 random elements is 0
time taken function1 in seconds: 0.025735abhishek_n_n_20
```

2. Write a multithreaded program using OpenMP for computing a matrix-matrix product for a large dimension. Use the OMP_NUM_THREADS environment variable to control the number of threads and plot the performance with varying numbers of threads (4,8 and 16). Consider four cases in which

   i) Only the outermost loop is parallelized
   ii) The outer two loops are parallelized
   iii) All three loops are parallelized
   iv) Use collapse clause
   What is the observed result from these four cases?

   Record your run times using omp_get_wtime() routine.

Code:

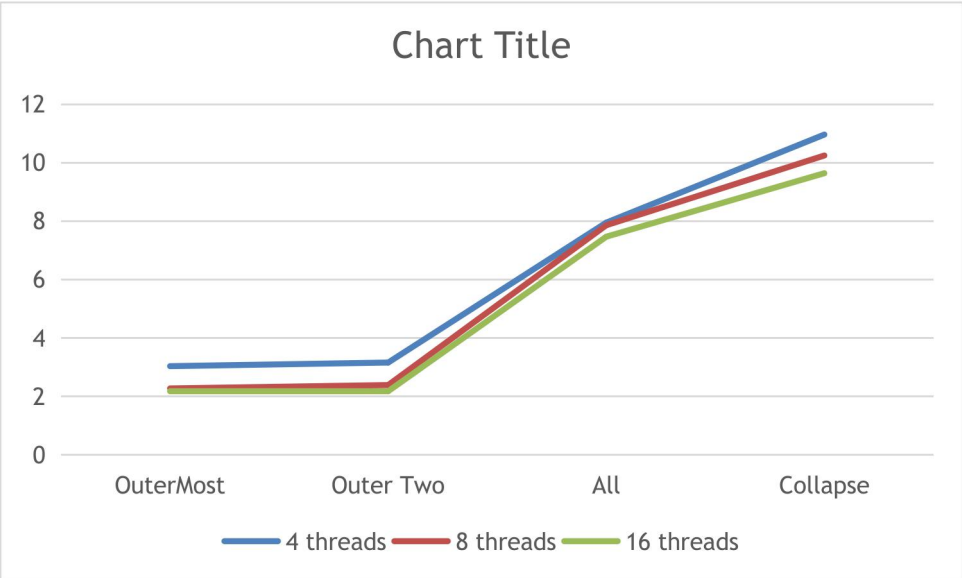
```
#include <omp.h>
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define N 1000

int A[N][N];
int B[N][N];
int C[N][N];

int main() {
     int i, j, k;
     for (i = 0; i < N; i++)
     for (j = 0; j < N; j++) {
     A[i][j] = 2;
     B[i][j] = 2;
     }

     int threadArr[] = {4, 8, 16};
     for (int t = 0; t < 3; t++) {
          omp_set_num_threads(threadArr[t]);
          double start = omp_get_wtime();
```

```
#pragma omp parallel for private(i, j, k) shared(A, B, C)
          for (i = 0; i < N; ++i) {
               for (j = 0; j < N; ++j) {
                    for (k = 0; k < N; ++k) {
                         C[i][j] += A[i][k] * B[k][j];
                    }
               }
          }
          double end = omp_get_wtime();
          printf("%d threads, outermost: %f seconds\n",
     threadArr[t], end - start);

          memset(C, 0, sizeof(C));

          start = omp_get_wtime();
#pragma omp parallel for private(i, j, k) shared(A, B, C)
          for (i = 0; i < N; ++i) {
#pragma omp parallel for private(i, j, k) shared(A, B, C)
               for (j = 0; j < N; ++j) {
                    for (k = 0; k < N; ++k) {
                         C[i][j] += A[i][k] * B[k][j];
                    }
               }
          }
          end = omp_get_wtime();
          printf("%d threads, outer two: %f seconds\n",
     threadArr[t], end - start);

          memset(C, 0, sizeof(C));

#pragma omp parallel for private(i, j, k) shared(A, B, C)
          for (i = 0; i < N; ++i) {
#pragma omp parallel for private(i, j, k) shared(A, B, C)
               for (j = 0; j < N; ++j) {
#pragma omp parallel for private(i, j, k) shared(A, B, C)
                    for (k = 0; k < N; ++k) {
                         C[i][j] += A[i][k] * B[k][j];
                    }
               }
          }
          end = omp_get_wtime();
          printf("%d threads, all loops : %f seconds\n",
     threadArr[t], end - start);

          memset(C, 0, sizeof(C));
```

```
        #pragma omp parallel for private(i, j, k) shared(A, B, C)
    collapse(3)
        for (i = 0; i < N; ++i) {
            for (j = 0; j < N; ++j) {
                for (k = 0; k < N; ++k) {
                    C[i][j] += A[i][k] * B[k][j];
                }
            }
        }
        end = omp_get_wtime();
        printf("%d threads, collapse : %f seconds\n",
    threadArr[t], end - start);
    }
    return 0;
}
```

Output

```
abhishek_n_n_20bce1025@ud:/mnt/D/ccpp$ gcc -fopenmp 1.c
abhishek_n_n_20bce1025@ud:/mnt/D/ccpp$ ./a.out
4 threads, outermost: 3.027410 seconds
4 threads, outer two: 3.151151 seconds
4 threads, all loops : 7.940060 seconds
4 threads, collapse : 10.953520 seconds
8 threads, outermost: 2.265373 seconds
8 threads, outer two: 2.380733 seconds
8 threads, all loops : 7.850639 seconds
8 threads, collapse : 10.237010 seconds
16 threads, outermost: 2.169480 seconds
16 threads, outer two: 2.170995 seconds
16 threads, all loops : 7.458266 seconds
16 threads, collapse : 9.631188 seconds
abhishek_n_n_20bce1025@ud:/mnt/D/ccpp$
```



20BCE1025

Abhishek N N

VIT UNIVERSITY
(Estd. u/s 3 of UGC Act 1956)
www.vit.ac.in
Vellore ▪ Chennai
CHENNAI CAMPUS
Vandalur - Kelambakkam Road, Chennai - 600127

Programme	:	**B.Tech.(CSE)**	Semester	:	**Fall '22-23**
Course	:	**Parallel and Distributed Computing**	Code	:	**CSE4001**
Faculty	:	**R. Kumar**	Slot	:	**L9+L10**

1. Write your own code snippet to demonstrate the following
   a. **Barrier**

Code:

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
        omp_set_num_threads(4);

#pragma omp parallel
        {
                printf("[Thread %d] I print my first message.\n",
                omp_get_thread_num());

#pragma omp barrier

                // One thread indicates that the barrier is complete.
#pragma omp single
                {
                        printf("The barrier is complete, which means all threads have
                        printed their first message.\n");
                }
                printf("[Thread %d] I print my second message.\n",
                omp_get_thread_num());
        }
        return EXIT_SUCCESS;
}
```

Output:

```
[Thread 2] I print my first message.
[Thread 3] I print my first message.
[Thread 0] I print my first message.
[Thread 1] I print my first message.
The barrier is complete, which means all threads have printed their first message.
[Thread 0] I print my second message.
[Thread 3] I print my second message.
[Thread 2] I print my second message.
[Thread 1] I print my second message.
```

b. Master

Code:

```
#include <omp.h>

#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
        omp_set_num_threads(4);

#pragma omp parallel
        {
                printf("[Thread %d] I print my first message.\n",
                omp_get_thread_num());

                // master only runs (default argument is nowait)
                #pragma omp master
                {
                printf("[Thread %d] Message by master.\n", omp_get_thread_num());
                }
                printf("[Thread %d] I print my second message.\n",
                omp_get_thread_num());
        }
        return EXIT_SUCCESS;
}
```

Output:

```
[Thread 0] I print my first message.
[Thread 0] Message by master.
[Thread 3] I print my first message.
[Thread 3] I print my second message.
[Thread 2] I print my first message.
[Thread 2] I print my second message.
[Thread 0] I print my second message.
[Thread 1] I print my first message.
[Thread 1] I print my second message.
```

c. **Single**

Code:

```
#include <omp.h>

#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
	omp_set_num_threads(4);

#pragma omp parallel
	{
		printf("[Thread %d] I print my first message.\n",
		omp_get_thread_num());

		// single: only one thread runs the block(with default barrier after block)
#pragma omp single
		{
			printf("[Thread %d] messge by single construct.\n",
			omp_get_thread_num());
		}
		printf("[Thread %d] I print my second message.\n",
		omp_get_thread_num());
	}
	return EXIT_SUCCESS;
}
```

Output:

```
[Thread 1] I print my first message.
[Thread 1] messge by single construct.
[Thread 3] I print my first message.
[Thread 2] I print my first message.
[Thread 0] I print my first message.
[Thread 1] I print my second message.
[Thread 2] I print my second message.
[Thread 0] I print my second message.
[Thread 3] I print my second message.
```

d. Critical

Code:

```
#include <omp.h>

#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
        omp_set_num_threads(4);

#pragma omp parallel
        {
                printf("[Thread %d] I print my first message.\n",
                omp_get_thread_num());

// critical: only one thread executes the block at a perticular time
#pragma omp critical
                {
                        printf("[Thread %d] messge by critical construct.\n",
                        omp_get_thread_num());
                }
                printf("[Thread %d] I print my second message.\n",
                omp_get_thread_num());
        }

        return EXIT_SUCCESS;
}
```

Output:

```
[Thread 3] I print my first message.
[Thread 3] messge by critical construct.
[Thread 3] I print my second message.
[Thread 2] I print my first message.
[Thread 2] messge by critical construct.
[Thread 0] I print my first message.
[Thread 1] I print my first message.
[Thread 2] I print my second message.
[Thread 0] messge by critical construct.
[Thread 0] I print my second message.
[Thread 1] messge by critical construct.
[Thread 1] I print my second message.
```

e. Ordered

Code:

```
#include <omp.h>

#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
	omp_set_num_threads(4);

#pragma omp parallel for ordered
	for(int i=0;i<10;i++){
		printf("[%d unordered] [%d thread]\n",i,omp_get_thread_num());

#pragma omp ordered
		printf("[%d ordered] [%d thread]\n",i,omp_get_thread_num());
	}

	return EXIT_SUCCESS;
}
```

Output:

```
[6 unordered] [2 thread]
[0 unordered] [0 thread]
[0 ordered] [0 thread]
[1 unordered] [0 thread]
[1 ordered] [0 thread]
[2 unordered] [0 thread]
[2 ordered] [0 thread]
[3 unordered] [1 thread]
[3 ordered] [1 thread]
[4 unordered] [1 thread]
[4 ordered] [1 thread]
[5 unordered] [1 thread]
[5 ordered] [1 thread]
[6 ordered] [2 thread]
[7 unordered] [2 thread]
[7 ordered] [2 thread]
[8 unordered] [3 thread]
[8 ordered] [3 thread]
[9 unordered] [3 thread]
[9 ordered] [3 thread]
```

2. Write a parallel program in OpenMP API to implement readers-writers problem. Ensure the usage of suitable low-level synchronization constructs. Document the result of synchronization overhead incurred in your experiment.

Code:

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <unistd.h>
omp_lock_t lock;

void Reader(int tid) {
	int a;
	// Wait till there is lock
	while (1) {
		if (!omp_test_lock(&lock)) {
			printf("\nReader : %d is waiting ", tid);
			sleep(1);
		}
		else break;
	}
	omp_unset_lock(&lock); // No need of lock for reading
	printf("\nReader : %d is reading", tid);

	sleep(3);
}

void Writer(int tid) {
	// First get the lock
	while (!omp_test_lock(&lock)) {
		if (!omp_test_lock(&lock)) {
			printf("\nWriter : %d is waiting ", tid);
			sleep(2);
		}
		else break;
	}
	printf("\nWriter : %d acquired lock", tid);
	printf("\nWriter : %d is writing", tid);
	// perform operation on the shared data
	sleep(0.5);
```

```
        omp_unset_lock(&lock); // release the lock after writing
        printf("\nWriter : %d released lock", tid);
        sleep(2);
}
int main() {
        int no, tid;
        omp_init_lock(&lock);
        srand((unsigned)time(NULL));
        int totalthreads;
#pragma omp parallel private(tid) shared(totalthreads)
        {
                totalthreads = omp_get_num_threads();

                while (1) {
                        int randomno = rand() % 100;
                        // Getting thread ID
                        tid = omp_get_thread_num();
                        // half of threads are readers and half are
                        writers

                        if (randomno < 50) { // calling reader
                                printf("\nProcess no. %d is a reader ",
                                randomno);
                                Reader(randomno);
                        }

                        else { // calling writer
                                printf("\nProcess no. %d is a writer ",
                                randomno);
                                Writer(randomno);
                        }
                }
        }
}
```

Output:

```
Process no. 42 is a reader
Reader : 42 is reading
Process no. 25 is a reader
Reader : 25 is reading
Process no. 81 is a writer
Writer : 81 acquired lock
Writer : 81 is writing
Process no. 68 is a writer
Writer : 68 is waiting
Process no. 39 is a reader
Reader : 39 is waiting
Process no. 71 is a writer
Writer : 71 is waiting
Process no. 67 is a writer
Writer : 67 is waiting
Process no. 44 is a reader
Reader : 44 is waiting
Writer : 81 released lock
Reader : 39 is reading
Reader : 44 is reading
Writer : 68 acquired lock
Writer : 68 is writing
Writer : 68 released lock
Writer : 71 acquired lock
Writer : 71 is writing
Writer : 71 released lock
Writer : 67 acquired lock
Writer : 67 is writing
Process no. 89 is a writer
Writer : 89 acquired lock
Writer : 89 is writing
Writer : 67 released lock
Writer : 89 released lock
Process no. 81 is a writer
Writer : 81 acquired lock
Writer : 81 is writing
Writer : 81 released lock
Process no. 93 is a writer
Writer : 93 acquired lock
Writer : 93 is writing
Writer : 93 released lock
Process no. 59 is a writer
Writer : 59 acquired lock
Writer : 59 is writing
Process no. 37 is a reader
Reader : 37 is waiting
Writer : 59 released lock
Process no. 8 is a reader
Reader : 8 is reading
```

20BCE1025

Abhishek N N

VIT UNIVERSITY
(Estd. u/s 3 of UGC Act 1956)
www.vit.ac.in
Vellore ▪ Chennai
CHENNAI CAMPUS
Vandalur - Kelambakkam Road, Chennai - 600127

Programme	:	**B.Tech.(CSE)**	Semester	:	**Fall '22-23**
Course	:	**Parallel and Distributed Computing**	Code	:	**CSE4001**
Faculty	:	**R. Kumar**	Slot	:	**L9+L10**

1. Write a program in OpenMP to find out the largest number in an array of 1000000 randomly generated numbers from 1 to 100000 using reduction clause. Compare the versions of serial, parallel for and reduction clause.

Code:

```
#include <limits.h>
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>

int main(void) {
	omp_set_num_threads(100);

	int arr[1000000];
	for (int i = 0; i < 1000000; i++) arr[i] = rand() % 100000;

	// serial
	double start = omp_get_wtime();
	int m = INT_MIN;
	for (int i = 0; i < 1000000; i++) {
		if (arr[i] > m) {
			m = arr[i];
		}
	}
```

```
    printf("serial max: %d in %f seconds\n", m, omp_get_wtime() -
start);

    // parallel
    start = omp_get_wtime();
    m = INT_MIN;
    #pragma omp parallel for shared(arr, m)
    for (int i = 0; i < 1000000; i++) {
        if (arr[i] > m) {
            m = arr[i];
        }
    }
    printf("parallel max: %d in %f seconds\n", m, omp_get_wtime()
- start);

    // reduction
    start = omp_get_wtime();
    m = INT_MIN;
    #pragma omp parallel for reduction(max : m)
    for (int i = 0; i < 1000000; i++) {
        if (arr[i] > m) {
            m = arr[i];
        }
    }
    printf("reduction max: %d in %f seconds\n", m, omp_get_wtime()
- start);

    return 0;
}
```

Output:

on first run

```
serial max: 100000 in 0.004459 seconds
parallel max: 100000 in 0.033015 seconds
reduction max: 100000 in 0.001992 seconds
```

on second run

```
serial max: 100000 in 0.004726 seconds
parallel max: 100000 in 0.034484 seconds
reduction max: 100000 in 0.002650 seconds
```

on third run

```
serial max: 100000 in 0.004332 seconds
parallel max: 100000 in 0.033231 seconds
reduction max: 100000 in 0.002358 seconds
```

Comparision:

To get better results I runned code 3 times

It is evident that **serial** is slowest in all times and

parallel is faster but we got lucky particularly here and there was no synchronization issue (usually **parallel** gives wrong output when no synchronization constraints such as critical or reduction)

reduction clause was the fastest taking full advantage of parallelism

2. Write a program in OpenMP to find out the standard deviation of 1000000 randomly generated numbers using reduction clause. Document the development versions of serial, parallel for and reduction clause.

Documentation is done in code as comments

Code:

```
#include <limits.h>
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
#include<math.h>
// count of randomly generated numbers
#define N 1000000

int main(void) {
	// setting no of omp threads
	omp_set_num_threads(100);

	// creating array with random numbers
	int arr[N];
	for (int i = 0; i < N; i++) arr[i] = rand() % 100000;

	// serial part
	double start = omp_get_wtime();
	double mean = 0;
	// summing up
	for (int i = 0; i < N; i++) mean+=arr[i];
	// now dividing by N to get mean
	mean/=N;
	// lets now find standard deviation
	double sd=0;
	// summing up squares of difference
	for (int i=0;i<N;i++) sd+=pow(arr[i]-mean,2);
	// dividing by N
	sd/=N;
	// taking square root
	sd=sqrt(sd);

	printf("serial standard deviation: %f in %f seconds\n", sd,
omp_get_wtime() - start);

	// parallel part
	start = omp_get_wtime();
	mean = 0;
	// summing up with mean and arr as shared
```

```
    #pragma omp parallel for shared(arr,mean)
    for (int i = 0; i < N; i++) mean+=arr[i];
    // now dividing by N to get mean
    mean/=N;
    // lets now find standard deviation
    sd=0;
    // summing up squares of difference
    // with mean and arr as shared
    #pragma omp parallel for shared(arr,mean)
    for (int i=0;i<N;i++) sd+=pow(arr[i]-mean,2);
    // dividing by N
    sd/=N;
    // taking square root
    sd=sqrt(sd);

    printf("parallel standard deviation: %f in %f seconds\n", sd,
omp_get_wtime() - start);
    // reduction part
    start = omp_get_wtime();
    mean = 0;
    // summing up with arr as shared and mean with reduction
    #pragma omp parallel for shared(arr) reduction(+:mean)
    for (int i = 0; i < N; i++) mean+=arr[i];
    // now dividing by N to get mean
    mean/=N;
    // lets now find standard deviation
    sd=0;
    // summing up squares of difference
    // with arr as shared and mean with reduction
    #pragma omp parallel for shared(arr) reduction(+:sd)
    for (int i=0;i<N;i++) sd+=pow(arr[i]-mean,2);
    // dividing by N
    sd/=N;
    // taking square root
    sd=sqrt(sd);

    printf("parallel standard deviation: %f in %f seconds\n", sd,
omp_get_wtime() - start);

    return 0;
}
```

Output:

on first run

```
serial standard deviation: 28875.866538 in 0.059834 seconds
parallel standard deviation: 13486.613913 in 0.082146 seconds
parallel standard deviation: 28875.866538 in 0.018416 seconds
```

on Second run

```
serial standard deviation: 28875.866538 in 0.056498 seconds
parallel standard deviation: 22702.105537 in 0.093420 seconds
parallel standard deviation: 28875.866538 in 0.018237 seconds
```

on third run

```
serial standard deviation: 28875.866538 in 0.058602 seconds
parallel standard deviation: 19682.337686 in 0.091663 seconds
parallel standard deviation: 28875.866538 in 0.016612 seconds
```

Comparision:

To get better results I runned code 3 times

It is evident that **serial** is slowest and gave correct output in all times and

parallel is faster but we got wrong output when no synchronization constraints such as critical or reduction is used

reduction clause was the fastest taking full advantage of parallelism and giving correct output

3. Write a multithreaded program using OpenMP to implement sequential and parallel version of the Monte Carlo algorithm for approximating Pi. Compare the results of sequential, loop-level parallelism and reduction clause with 10000000 samples.

Code :

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <omp.h>
// for rand() function
#define SEED 35791246
#define niter 100000
int main() {
     // serial
     double start = omp_get_wtime();
     srand(SEED); // initialize random numbers
     int count = 0;
     for (int i = 0; i < niter; i++) {
     double x = (double)rand() / RAND_MAX;
     double y = (double)rand() / RAND_MAX;
     double z = x * x + y * y;
     if (z <= 1) count++;
     }
     double pi = (double)count / niter * 4;
     printf("serial pi : %g with time %f seconds\n", pi,
omp_get_wtime() - start);

     // parallel (loop level)
     start = omp_get_wtime();
     srand(SEED); // initialize random numbers
     count = 0;
     #pragma omp parallel for shared(count)
     for (int i = 0; i < niter; i++) {
     double x = (double)rand() / RAND_MAX;
     double y = (double)rand() / RAND_MAX;
     double z = x * x + y * y;
     if (z <= 1) count++;
     }
     pi = (double)count / niter * 4;
     printf("parallel pi : %f with time %f seconds\n", pi,
omp_get_wtime() - start);

     // reduction
```

```
    start = omp_get_wtime();
    srand(SEED); // initialize random numbers
    count = 0;
    #pragma omp parallel for reduction(+ : count)
    for (int i = 0; i < niter; i++) {
    double x = (double)rand() / RAND_MAX;
    double y = (double)rand() / RAND_MAX;
    double z = x * x + y * y;
    if (z <= 1) count++;
    }
    pi = (double)count / niter * 4;
    printf("reduction pi : %f with time %f seconds\n", pi,
omp_get_wtime() - start);
    return 0;
}
```

Output:

on first run

```
serial pi : 3.1424 with time 0.006635 seconds
parallel pi : 3.073200 with time 0.107288 seconds
reduction pi : 3.142640 with time 0.143944 seconds
```

on second run

```
serial pi : 3.1424 with time 0.006874 seconds
parallel pi : 3.079760 with time 0.080502 seconds
reduction pi : 3.141480 with time 0.071293 seconds
```

on third run

```
serial pi : 3.1424 with time 0.006667 seconds
parallel pi : 3.075840 with time 0.089968 seconds
reduction pi : 3.143480 with time 0.075784 seconds
```

Comparision:

To get better results I runned code 3 times

It is evident that **serial(sequential)** is fastest(due to excessive overhead in parallel systems) and gave correct output in all times and

parallel is slowest and we got wrong output when no synchronization constraints such as critical or reduction is used

reduction clause was the faster taking advantage of parallelism and giving correct output near to serial

20BCE1025
Abhishek N N

Parallel and Distributed Computing(CSE4001)

Lab 7 -Profiling

profiling of 2,4,8 threads in matrix multiplication
CODE:

```
#include <omp.h>
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define N 10
int A[N][N];
int B[N][N];
int C[N][N];
int main() {
	int i, j, k;
	for (i = 0; i < N; i++) {
		for (j = 0; j < N; j++) {
			A[i][j] = 2;
			B[i][j] = 2;
		}
	}

	int arr[] = {2, 4, 8};
	for (int t = 0; t < 3; t++) {
		double start = omp_get_wtime();
#pragma omp parallel for private(i, j, k) shared(A, B, C)
num_threads(arr[t])
		for (i = 0; i < N; i++) {
			for (j = 0; j < N; j++) {
				for (k = 0; k < N; k++) {
					C[i][j] += A[i][k] * B[k][j];
				}
			}
		}
		double end = omp_get_wtime();
		printf("%d threads, : %f seconds\n", arr[t], end -
	start);
	}

	return 0;
}
```

Commands:

```
abhishek_n_n_20bce1025@ud:/mnt/D/ccpp/lab7$ kinst-ompp gcc -fopenmp lab7.c -o lab7
abhishek_n_n_20bce1025@ud:/mnt/D/ccpp/lab7$ ./lab7
2 threads, : 0.000180 seconds
4 threads, : 0.000101 seconds
8 threads, : 0.062241 seconds
```

output:

##BEG header separator=									
Start Date	Fri Sep 16 00:26:14 2022								
End Date	Fri Sep 16 00:26:14 2022								
Duration	0.062575	sec							
Application Name	unknown								
Type of Report	final								
Thread Count	8								
ompP Version Major	0								
ompP Version Minor	8								
ompP Version Revision	99								
ompP Build Date	Sep 15 2022 22:54:34								
PAPI Support	not available								
##END header									
##BEG region overview									
PARALLEL LOOP	1								
R00001	PARALLEL LOOP	lab7.c	23	30					
##END region overview									
##BEG ompp callgraph									
Inclusive	(%)	Exclusive	(%)						
0.062575	100	0.000165	0.264041	0	[unknown: 8 threads]				
0.06241	99.735959	0.06241	99.735959	1	R00001	PARALLEL LOOP	lab7.c	23	30
##END ompp callgraph									

##BEG flat region profiles								
##BEG R00001 flat profile								
R00001	PARALLEL LOOP	lab7.c	23	30				
TID	execT	execC	bodyT	exitBarT	startupT	shutdwnT	taskT	
0	0.06241	3	0.000016	0.000008	0.030384	0.031999	0	
1	0.06241	3	0.000013	0.015993	0.014398	0.032001	0	
2	0.06241	3	0.000007	0.015992	0.014335	0	0	
3	0.06241	3	0.000005	0.015993	0.014337	0	0	
4	0.06241	3	0.000003	0.01599	0.014246	0	0	
5	0.06241	3	0.000003	0.01599	0.014246	0	0	
6	0.06241	3	0.000003	0.014597	0.015639	0	0	
7	0.06241	3	0.000002	0.031991	0.014245	0	0	
SUM	0.499279	24	0.000051	0.126554	0.13183	0.064	0	
##END R00001 flat profile								
##END flat region profiles								
##BEG callgraph region profiles								
##BEG R00001 callgraph profile								

[*00]	unknown							
[=01]	R00001	PARALLEL LOOP	lab7.c	23	30			
TID	execT	execC	bodyT/I	bodyT/E	exitBarT	startupT	shutdwnT	taskT
0	0.06241	3	0.000015	0.000015	0.000006	0.030386	0.031997	0
1	0.06241	3	0.000013	0.000013	0.015992	0.0144	0.031999	0
2	0.06241	3	0.000007	0.000007	0.015992	0.014337	0	0
3	0.06241	3	0.000007	0.000007	0.015991	0.014337	0	0
4	0.06241	3	0.000003	0.000003	0.015989	0.014247	0	0
5	0.06241	3	0.000003	0.000003	0.015989	0.014247	0	0
6	0.06241	3	0.000003	0.000003	0.014597	0.01564	0	0
7	0.06241	3	0.000002	0.000002	0.031991	0.014246	0	0
SUM	0.499279	24	0.000052	0.000052	0.126547	0.13184	0.063996	0
##END R00001 callgraph profile								
##END callgraph region profiles								
##BEG overhead analysis report								
Total runtime (wallclock) [secs]	0.062575							
Number of threads	8							
Number of parallel regions	1							
Parallel coverage [secs]	0.06241							
Parallel coverage [percent]	99.735959							
##BEG parallel regions sorted by wallclock time								
	Type	Wallclock	(%)					
R00001	PARLOOP	0.06241	99.735959					
SUM	0.06241		99.735959					
##END parallel regions sorted by wallclock time								

##BEG overheads for parallel region												
	Total	Ovhds	(%)	Synch	(%)	Imbal	(%)	Limpar	(%)	Mgmt	(%)	
R00001	0.499279	0.322383	64.569692	0	0	0.126547	25.346015	0	0	0.195836	39.223677	
##END overheads for parallel region												
##BEG overheads for whole program												
	Total	Ovhds	(%)	Synch	(%)	Imbal	(%)	Limpar	(%)	Mgmt	(%)	
R00001	0.499279	0.322383	64.399201	0	0	0.126547	25.279091	0	0	0.195836	39.12011	
SUM	0.499279	0.322383	64.399201	0	0	0.126547	25.279091	0	0	0.195836	39.12011	
##END overheads for whole program												
##END overhead analysis report												

20BCE1025
Abhishek N N

Parallel and Distributed Computing(CSE4001)

Lab 8 - MPI Basics

Code:

```
#include <mpi.h>
#include <stdio.h>

int main(int argc, char** argv) {
   // Initialize the MPI environment
   MPI_Init(NULL, NULL);

   // Get the number of processes
   int world_size;
   MPI_Comm_size(MPI_COMM_WORLD, &world_size);

   // Get the rank of the process
   int world_rank;
   MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

   // Get the name of the processor
   char processor_name[MPI_MAX_PROCESSOR_NAME];
   int name_len;
   MPI_Get_processor_name(processor_name, &name_len);

   // Print off a hello world message
   printf("Hello world from processor %s, rank %d out of %d
processors\n", processor_name, world_rank, world_size);

   // Finalize the MPI environment.
   MPI_Finalize();
}
```

Running:

```
abhishek_n_n_20bce1025@ud:/mnt/D/ccpp$ mpicc mpi_example.c -o mpi_example
abhishek_n_n_20bce1025@ud:/mnt/D/ccpp$ mpirun -np 1 ./mpi_example
hwloc/linux: Ignoring PCI device with non-16bit domain.
Pass --enable-32bits-pci-domain to configure to support such devices
(warning: it would break the library ABI, don't enable unless really needed).
Hello world from processor ud, rank 0 out of 1 processors
abhishek_n_n_20bce1025@ud:/mnt/D/ccpp$ mpirun -np 2 ./mpi_example
hwloc/linux: Ignoring PCI device with non-16bit domain.
Pass --enable-32bits-pci-domain to configure to support such devices
(warning: it would break the library ABI, don't enable unless really needed).
Hello world from processor ud, rank 0 out of 2 processors
Hello world from processor ud, rank 1 out of 2 processors
abhishek_n_n_20bce1025@ud:/mnt/D/ccpp$
```

Name : Abhishek N N

Reg.No : 20BCE1025

Email : abhishek.nn2020@vitstudent.ac.in

VIT UNIVERSITY
(Estd. u/s 3 of UGC Act 1956)
www.vit.ac.in
Vellore • Chennai
CHENNAI CAMPUS
Vandalur - Kelambakkam Road, Chennai - 600127

Programme	:	**B.Tech.(CSE)**	Semester	:	**Fall '22-23**
Course	:	**Parallel and Distributed Computing**	Code	:	**CSE4001**
Faculty	:	**R. Kumar**	Slot	:	**L9+L10**

```
abhishek_n_n_20bce1025@ud:~$ ifconfig
lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 3234  bytes 402338 (402.3 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 3234  bytes 402338 (402.3 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

virbr0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 192.168.122.1  netmask 255.255.255.0  broadcast 192.168.122.255
        ether 52:54:00:d3:0d:be  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

wlp44s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.20.115.247  netmask 255.255.248.0  broadcast 172.20.119.255
        inet6 fe80::8553:cd3a:236d:e426  prefixlen 64  scopeid 0x20<link>
        ether a4:97:b1:aa:fb:05  txqueuelen 1000  (Ethernet)
        RX packets 299921  bytes 311385781 (311.3 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 109973  bytes 19903605 (19.9 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

abhishek_n_n_20bce1025@ud:~$
```

```
  GNU nano 6.2
127.0.0.1       localhost
127.0.1.1       ud

#MPI CLUSTERS
172.20.115.247 manager
192.168.96.21 worker1
172.20.116.236 worker2
172.20.116.203 worker3

# The following lines are desirable for IPv6
::1     ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
```

```
  GNU nano 6.2
127.0.0.1 localhost
127.0.1.1 us1

172.20.115.247 manager
172.20.116.203 worker3

# The following lines are desirable for IPv6
::1     ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
```

```
ujjwal@kali: ~
File  Actions  Edit  View  Help
  GNU nano 5.9                    /etc/hosts *
127.0.0.1       localhost
127.0.1.1       kali.ujjwalkali.com     kali

172.20.115.247 manager
172.20.116.236 worker2
# The following lines are desirable for IPv6 capable hosts
::1     localhost ip6-localhost ip6-loopback
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters

^G Help        ^O Write Out   ^W Where Is    ^K Cut         ^T Execute
^X Exit        ^R Read File   ^\ Replace     ^U Paste       ^J Justify
```

```
  GNU nano 6.2                    /etc/hosts
127.0.0.1       localhost
127.0.1.1       sreyas-VirtualBox

# The following lines are desirable for IPv6 capable hosts
172.20.115.247 manager
192.168.96.21 worker1

::1     ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
```

```
abhishek_n_n_20bce1025@ud:~$ sudo adduser 20BCE1025_Abhishek_N_N --force-badname
Allowing use of questionable username.
Adding user `20BCE1025_Abhishek_N_N' ...
Adding new group `20BCE1025_Abhishek_N_N' (1002) ...
Adding new user `20BCE1025_Abhishek_N_N' (1001) with group `20BCE1025_Abhishek_N_N' ...
Creating home directory `/home/20BCE1025_Abhishek_N_N' ...
Copying files from `/etc/skel' ...
New password:
Retype new password:
passwd: password updated successfully
Changing the user information for 20BCE1025_Abhishek_N_N
Enter the new value, or press ENTER for the default
        Full Name []:
        Room Number []:
        Work Phone []:
        Home Phone []:
        Other []:
Is the information correct? [Y/n] y
abhishek_n_n_20bce1025@ud:~$
```

```
us1@us1:~$ sudo adduser 20BCE1025_Abhishek_N_N_worker --force-badname
Allowing use of questionable username.
Adding user `20BCE1025_Abhishek_N_N_worker' ...
Adding new group `20BCE1025_Abhishek_N_N_worker' (1001) ...
Adding new user `20BCE1025_Abhishek_N_N_worker' (1001) with group `20BCE1025_Abhishek_N_N_worker' ...
Creating home directory `/home/20BCE1025_Abhishek_N_N_worker' ...
Copying files from `/etc/skel' ...
New password:
Retype new password:
passwd: password updated successfully
Changing the user information for 20BCE1025_Abhishek_N_N_worker
Enter the new value, or press ENTER for the default
        Full Name []:
        Room Number []:
        Work Phone []:
        Home Phone []:
        Other []:
Is the information correct? [Y/n] Y
us1@us1:~$
```

1. Write a program in MPI to create two processes in two different machines. Process 0 pings Process 1 and awaits for return ping using Non-blocking message passing routines. Execute your code on MPI cluster.

```
#include < stdio.h>
#include < unistd.h>

#include "mpi.h"
int main(int argc, char *argv[]) {
int num, r, dest, src, rc, c, tag = 1;
char inmsg, outmsg = 'x';
MPI_Request send_msg, recv_msg;
MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &num);
MPI_Comm_rank(MPI_COMM_WORLD, &r);
if (r == 1) {
dest = 0;
src = 0;
rc = MPI_Isend(&outmsg, 1, MPI_CHAR, dest, tag, MPI_COMM_WORLD,
&send_msg);
printf("Process %d: Pinged Process %d\n", r, dest);
rc = MPI_Irecv(&inmsg, 1, MPI_CHAR, src, tag, MPI_COMM_WORLD,
&recv_msg);
printf("Process %d: Received a ping from Process %d\n", r, src);
else {
dest = 1;
src = 1;
rc = MPI_Isend(&outmsg, 1, MPI_CHAR, dest, tag, MPI_COMM_WORLD,
&send_msg);
printf("Process %d: Pinged Process %d\n", r, dest);
sleep(1);
rc = MPI_Irecv(&inmsg, 1, MPI_CHAR, src, tag, MPI_COMM_WORLD,
&recv_msg);
printf("Process %d: Received a ping from Process %d\n", r, src);
}
MPL Finalize();
}
```

```
20BCE1025_Abhishek_N_N@ud:~$ mpicc ping_lab9.c -o ping_lab9
20BCE1025_Abhishek_N_N@ud:~$ mpiexec -np 2 ./ping_lab9
Process 1: Pinged Process 0
Process 0: Pinged Process 1
Process 1: Received a ping from Process 0
Process 0: Received a ping from Process 1
```

2. Write a program in MPI to create 10 tasks. Construct a ring topology to exchange message to its nearest neighbour in the ring using blocking massage passing routines. Execute your code on MPI cluster.

```
#include "mpi.h" #include < stdio.h>
int main(int argc, char *argv[]) {
int num, r, dest, src, rc, c, tag = 1;
char in_msg, out_msg = 'x';
MPI_Status Stat;
MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &num);
MPI_Comm_rank(MPI_COMM_WORLD, &r);
dest = r + 1;
src = r - 1;
if (dest > 9) {
dest = 0;
}
if (src < 0) {
src = 9;
}
rc = MPI_Send(&out_msg, 1, MPI_CHAR, dest, tag, MPI_COMM_WORLD);
printf("Node %d: Pinged Process %d\n", r, dest);
rc = MPI_Recv(&in_msg, 1, MPI_CHAR, src, tag, MPI_COMM_WORLD,
&Stat);
printf("Node %d: Recieved a ping from Node %d\n", r, src);
MPI_Finalize();
}
```

```
20BCE1025_Abhishek_N_N@ud:~$ mpicc ring_topology_lab9.c -o ring_topology_lab9
20BCE1025_Abhishek_N_N@ud:~$ mpiexec -np 10 ./ring_topology_lab9
Node 4: Pinged Process 5
Node 8: Pinged Process 9
Node 6: Pinged Process 7
Node 0: Pinged Process 1
Node 1: Pinged Process 2
Node 1: Recieved a ping from Node 0
Node 5: Pinged Process 6
Node 6: Recieved a ping from Node 5
Node 9: Pinged Process 0
Node 9: Recieved a ping from Node 8
Node 3: Pinged Process 4
Node 5: Recieved a ping from Node 4
Node 2: Pinged Process 3
Node 2: Recieved a ping from Node 1
Node 4: Recieved a ping from Node 3
Node 7: Pinged Process 8
Node 7: Recieved a ping from Node 6
```

Name : Abhishek N N

Reg.No : 20BCE1025

Email : abhishek.nn2020@vitstudent.ac.in

VIT UNIVERSITY
(Estd. u/s 3 of UGC Act 1956)
www.vit.ac.in
Vellore • Chennai
CHENNAI CAMPUS
Vandalur - Kelambakkam Road, Chennai - 600127

Programme	:	**B.Tech.(CSE)**	Semester	:	**Fall '22-23**
Course	:	**Parallel and Distributed Computing**	Code	:	**CSE4001**
Faculty	:	**Prof. R. Kumar**	Slot	:	**L9+L10**

1. Write a program in MPI to generate 'n' random float numbers and send' k' of those to each node and make them compute the average and send it back to the master which computes the average of those averages.

```
#include <mpi.h>

#include <stdio.h>
#include <stdlib.h>

#define max_n 10000
#define send_data_tag 2001
#define return_data_tag 2002

float array[max_n];
float array2[max_n];
int main(int argc, char** argv) {
     float sum, partial_sum;
     float avg, partial_avg;
     MPI_Status status;
     int my_id, root_process, ierr, i, num_procs, an_id, sender,
     num_rows_to_send, hum_rows_to_receive, num_rows_received;
     int n, k;

     int start_n, end_n;
```

```
ierr = MPI_Init(&argc, &argv);
root_process = 0;
ierr = MPI_Comm_rank(MPI_COMM_WORLD, &my_id);
ierr = MPI_Comm_size(MPI_COMM_WORLD, &num_procs);

if (my_id == root_process) {
     printf("Please enter the number of elements: ");
     scanf("%d", &n);

     if (n > max_n) {
          printf("Too many numbers.\n");
          exit(1);
     }

     /* Initialize an array */

     for (i = 0; i < n; i++) {
          array[i] = (float)(rand() % 100) / 7;
     }

     k = n / num_procs;
     /* Distribute a portion of the vector to each child
     process */

     for (an_id = 1; an_id < num_procs; an_id++) {
          start_n = an_id * k + 1;
          end_n = (an_id + 1) * k;
          if ((n - end_n) < k)
          end_n = n - 1;

          num_rows_to_send = end_n - start_n + 1;

          ierr = MPI_Send(&num_rows_to_send, 1, MPI_INT,
          an_id, send_data_tag,
          MPI_COMM_WORLD);

          ierr = MPI_Send(&array[start_n],
          num_rows_to_send, MPI_INT, an_id, send_data_tag,
          MPI_COMM_WORLD);
     }

     /* Calculate the sum and average of the values in the
     segment assigned to the root
     process */

     sum = 0;
```

```
        for (i = 0; i < k + 1; i++) {
             sum += array[i];
        }
        avg = sum / (k + 1);
        printf("Average calculated by Root Process = %.3f\n",
        avg);
        /* and, finally, I collect the partial sums from the
        slave processes, print them, and add them to the grand
        sum, and print it */

        for (an_id = 1; an_id < num_procs; an_id++) {
             ierr = MPI_Recv(&partial_avg, 1, MPI_FLOAT,
             MPI_ANY_SOURCE, return_data_tag, MPI_COMM_WORLD,
             &status);
             sender = status.MPI_SOURCE;
             printf("Partial Average returned from process %i
             = %.3f\n", sender,
             partial_avg);
        }
        avg += partial_avg;
        avg = avg / num_procs;
        printf("The Overall Average is: %f\n", avg);
    }

    else {
        /* I must be a slave process, so I must receive my
        array segment, storing it in a "local" array, array1.
        */

        ierr = MPI_Recv(&num_rows_to_receive, 1, MPI_INT,
        root_process, send_data_tag, MPI_COMM_WORLD, &status);
        ierr = MPI_Recv(&array2, num_rows_to_receive, MPI_INT,
        root_process, send_data_tag, MPI_COMM_WORLD, &status);
        num_rows_received = num_rows_to_receive;

        /* Calculate the sum and avergae of my portion of the
        array */ partial_sum = 0;
        for (i = 0; i < num_rows_received; i++) {
             partial_sum += array2[i];
        }
        partial_avg = (float)partial_sum / num_rows_received;
        /* and finally, send my partial sum to the root
        process */
        ierr = MPI_Send(&partial_avg, 1, MPI_FLOAT,
        root_process, return_data_tag, MPI_COMM_WORLD);
    }
```

```
    ierr = MPI_Finalize();
    return 0;
}
```

```
20BCE1025_Abhishek_N_N@ud:~$ mpicc sum_avg_lab10.c -o sum_avg_lab10
20BCE1025_Abhishek_N_N@ud:~$ mpiexec -np 4 ./sum_avg_lab10
Please enter the number of elements: 8
Average calculated by Root Process = 2.000
Partial Average returned from process 1 = 4.500
Partial Average returned from process 2 = 6.500
Partial Average returned from process 3 = 8.000
The Overall Average is: 5.250000
```

```
20BCE1025_Abhishek_N_N@ud:~$ mpiexec -np 5 ./sum_avg_lab10
Please enter the number of elements: 25
Average calculated by Root Process = 3.500
Partial Average returned from process 2 = 14.000
Partial Average returned from process 4 = 23.500
Partial Average returned from process 3 = 19.000
Partial Average returned from process 1 = 9.000
The Overall Average is: 13.800000
```

```
20BCE1025_Abhishek_N_N@ud:~$ mpiexec -np 20 ./sum_avg_lab10
Please enter the number of elements: 100
Average calculated by Root Process = 3.500
Partial Average returned from process 2 = 14.000
Partial Average returned from process 5 = 29.000
Partial Average returned from process 14 = 74.000
Partial Average returned from process 16 = 84.000
Partial Average returned from process 17 = 89.000
Partial Average returned from process 1 = 9.000
Partial Average returned from process 3 = 19.000
Partial Average returned from process 4 = 24.000
Partial Average returned from process 6 = 34.000
Partial Average returned from process 7 = 39.000
Partial Average returned from process 10 = 54.000
Partial Average returned from process 12 = 64.000
Partial Average returned from process 13 = 69.000
Partial Average returned from process 8 = 44.000
Partial Average returned from process 9 = 49.000
Partial Average returned from process 11 = 59.000
Partial Average returned from process 15 = 79.000
Partial Average returned from process 18 = 94.000
Partial Average returned from process 19 = 98.500
The Overall Average is: 51.450001
```

2. Write a MPI program to compute PI using “dartboard” technique for 1000 rounds by using reduction collective computation.

```
#include <stdio.h>
#include <stdlib.h>

#include "mpi.h"
void srandom(unsigned seed);
double pi_board(int darts);
#define NUM_DARTS 5000
#define NUM_ROUNDS 1000
#define MASTER 0
int main(int arc, char *argv[]) {
	double homepi, pisum, pi, avepi;
	int t_id, num_tasks, rc, i;
	MPI_Status status;
	MPI_Init(&argc, &argv); // Initializing MPI
	MPI_Comm_size(MPI_COMM_WORLD,&num_tasks);
	MPI_Comm_rank(MPI_COMM_WORLD,&t_id);
	printf("MPI task %d has started...\n", t_id);
	srandom(t_id);
	avepi = 0;
	for (i = 0; i < NUM_ROUNDS; i++) {
		homepi = pi_board(NUM_DARTS);
		rc = MPI_Reduce(&homepi, &pisum, 1, MPI_DOUBLE, MPI_SUM,
		MASTER, MPI_COMM_WORLD); // MPI_Reduce is used to sum
		the values of pi across all tasks if (rc != MPI_SUCCESS)
		printf("%d: mpc_reduce failed ! \n", t_id);
		if (t_id == MASTER) {
			pi = pisum / num_tasks;
			avepi = ((avepi * i) + pi) / (i + 1);
			printf("Round : %d , average value of pi = %10.8f\
			n", i, avepi);
		}
	}
	if (t_id == MASTER)
	printf("\nReal value of PI: 3.1415926535897 \n");
	MPI_Finalize();
	return 0;
}

double pi_board(int d) // Function for cal PI
```

```
{
      #define sqr(x) ((x) * (x)) long random(void);
      double x_coord, y_coord, pi, r;
      int score, n;
      unsigned int cconst;
      cconst = 2 << (31 - 1);
      score = 0;
      for (n = 1; n <= d; n++) {
            r = (double)random() / cconst;
            x_coord = (2.0 * r) - 1.0;
            r = (double)random() / cconst;
            y_coord = (2.0 * r) - 1.0;
            if ((sqr(x_coord) + sqr(y_coord)) <= 1.0) score++;
      }
      pi = 4.0 * (double)score / (double)d; // Calculating PI
      return(pi);
      return (pi);
}
```

```
20BCE1025_Abhishek_N_N@ud:~$ mpicc dartboard_reduction_lab10.c -o dartboard_reduction_lab10
20BCE1025_Abhishek_N_N@ud:~$ mpiexec -np 5 ./dartboard_reduction_lab10
MPI task 0 has started...
MPI task 1 has started...
MPI task 3 has started...
MPI task 4 has started...
MPI task 2 has started...
Round : 0 , average value of pi = 3.13984000
Round : 1 , average value of pi = 3.13640000
Round : 2 , average value of pi = 3.14288000
Round : 3 , average value of pi = 3.14444000
Round : 4 , average value of pi = 3.14118400
Round : 5 , average value of pi = 3.14264000
Round : 6 , average value of pi = 3.14560000
Round : 7 , average value of pi = 3.14452000
Round : 8 , average value of pi = 3.14535111
Round : 9 , average value of pi = 3.14523200
Round : 10 , average value of pi = 3.14462545
Round : 11 , average value of pi = 3.14326667
Round : 12 , average value of pi = 3.14395077
Round : 13 , average value of pi = 3.14274286
Round : 14 , average value of pi = 3.14184533
Round : 15 , average value of pi = 3.14300000
Round : 16 , average value of pi = 3.14211765
Round : 17 , average value of pi = 3.14261333
Round : 18 , average value of pi = 3.14304842
Round : 19 , average value of pi = 3.14244000
Round : 20 , average value of pi = 3.14157714
Round : 21 , average value of pi = 3.14108364
Round : 22 , average value of pi = 3.14103652
```

```
Round : 90 , average value of pi = 3.14187253
Round : 91 , average value of pi = 3.14176696
Round : 92 , average value of pi = 3.14195613
Round : 93 , average value of pi = 3.14195404
Round : 94 , average value of pi = 3.14208000
Round : 95 , average value of pi = 3.14202333
Round : 96 , average value of pi = 3.14207010
Round : 97 , average value of pi = 3.14160163
Round : 98 , average value of pi = 3.14159677
Round : 99 , average value of pi = 3.14141120
Round : 100 , average value of pi = 3.14135446
Round : 101 , average value of pi = 3.14124863
Round : 102 , average value of pi = 3.14124738
Round : 103 , average value of pi = 3.14138308
Round : 104 , average value of pi = 3.14139429
Round : 105 , average value of pi = 3.14151547
Round : 106 , average value of pi = 3.14168224
Round : 107 , average value of pi = 3.14170222
Round : 108 , average value of pi = 3.14192440
Round : 109 , average value of pi = 3.14199273
Round : 110 , average value of pi = 3.14216216
Round : 111 , average value of pi = 3.14251143
Round : 112 , average value of pi = 3.14230655
Round : 113 , average value of pi = 3.14235088
Round : 114 , average value of pi = 3.14254330
Round : 115 , average value of pi = 3.14252690
Round : 116 , average value of pi = 3.14243966
```

```
Round : 555 , average value of pi = 3.14167942
Round : 556 , average value of pi = 3.14168330
Round : 557 , average value of pi = 3.14170753
Round : 558 , average value of pi = 3.14169989
Round : 559 , average value of pi = 3.14166400
Round : 560 , average value of pi = 3.14164848
Round : 561 , average value of pi = 3.14165950
Round : 562 , average value of pi = 3.14165826
Round : 563 , average value of pi = 3.14161730
Round : 564 , average value of pi = 3.14161501
Round : 565 , average value of pi = 3.14163194
Round : 566 , average value of pi = 3.14164346
Round : 567 , average value of pi = 3.14163437
Round : 568 , average value of pi = 3.14160815
Round : 569 , average value of pi = 3.14158428
Round : 570 , average value of pi = 3.14161233
Round : 571 , average value of pi = 3.14159692
Round : 572 , average value of pi = 3.14157682
Round : 573 , average value of pi = 3.14157268
Round : 574 , average value of pi = 3.14153739
Round : 575 , average value of pi = 3.14152167
Round : 576 , average value of pi = 3.14153428
Round : 577 , average value of pi = 3.14156429
Round : 578 , average value of pi = 3.14154003
Round : 579 , average value of pi = 3.14154593
Round : 580 , average value of pi = 3.14152372
```

```
Round : 975 , average value of pi = 3.14170230
Round : 976 , average value of pi = 3.14169695
Round : 977 , average value of pi = 3.14170094
Round : 978 , average value of pi = 3.14172846
Round : 979 , average value of pi = 3.14175102
Round : 980 , average value of pi = 3.14175739
Round : 981 , average value of pi = 3.14174143
Round : 982 , average value of pi = 3.14173624
Round : 983 , average value of pi = 3.14173447
Round : 984 , average value of pi = 3.14171744
Round : 985 , average value of pi = 3.14170353
Round : 986 , average value of pi = 3.14169062
Round : 987 , average value of pi = 3.14169279
Round : 988 , average value of pi = 3.14170402
Round : 989 , average value of pi = 3.14171442
Round : 990 , average value of pi = 3.14171189
Round : 991 , average value of pi = 3.14171806
Round : 992 , average value of pi = 3.14172681
Round : 993 , average value of pi = 3.14171525
Round : 994 , average value of pi = 3.14170436
Round : 995 , average value of pi = 3.14168546
Round : 996 , average value of pi = 3.14170078
Round : 997 , average value of pi = 3.14168754
Round : 998 , average value of pi = 3.14169049
Round : 999 , average value of pi = 3.14170816

Real value of PI: 3.1415926535897
```

3. Write a MPI program to perform matrix multiplication (1000x1000) using scatter and gather routines.

```
#define N 200
#include <math.h>
#include <mpi.h>
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>

void print_results(char *prompt, int a[N][N]);

int main(int argc, char *argv[]) {
     int i, j, k, rank, size, tag = 99, blksz, sum = 0;
     int a[N][N] = {{1, 2, 3, 4}, {5, 6, 7, 8}, {9, 1, 2, 3},
     {4, 5, 6, 7}};

     int b[N][N]={{1,2,3,4},{5,6,7,8},{9,1,2,3},{4,5,6,7}};
     int c[N][N];
     int a[N][N], b[N][N];
     int aa[N], CC[N];
     for (int i = 0; i < N; i++) {
          for (int j = 0; j < N; j++) {
               a[i][j] = 1; b[i[j]=2;
          }
     }
     MPI_Init(&argc, &argv);
     MPI_Comm_size(MPI_COMM_WORLD, &size);
     MPI_Comm_rank(MPI_COMM_WORLD, &rank);
     // Scatter rows of first matrix to different processes
     MPI_Scatter(a, N * N / size, MPI_INT, aa, N * N / size,
     MPI_INT, 0, MPI_COMM_WORLD);

     // Broadcast second matrix to all processes MPI_Bcast(b,
     N*N, MPI_INT, 0, MPI_COMM_WORLD);

     MPI_Barrier(MPI_COMM_WORLD);

     // Perform vector multiplication by all processes
     for (i = 0; i < N; i++)
     {
     for (j = 0; j < N; j++) {
     sum = sum + aa[j] * b[j][i];
     }
     cc[i] = sum;
```

```
        sum = 0;
        }

        MPI_Gather(cc, N * N / size, MPI_INT, c, N * N / size,
        MPI_INT, 0, MPI_COMM_WORLD);

        MPI_Barrier(MPI_COMM_WORLD);
        MPI_Finalize();
        if (rank == 0)
        print_results("C = ", c);
}

void print_results(char *prompt, int a[N][N]) {
        int i, j;

        printf("\n\n%s\n", prompt);
        for (i = 0; i < N; i++) {
                for (j = 0; j < N; j++) {
                        printf(" %d", a[i][j]);
                }
                printf("\n");
        }
        printf("\n\n");
}
```

```
20BCE1025_Abhishek_N_N@ud:~$ mpicc matrix_lab10.c -o matrix_lab10
20BCE1025_Abhishek_N_N@ud:~$ mpiexec -np 4 ./matrix_lab10


C =
 54 37 47 57
 130 93 119 145
 44 41 56 71
 111 79 101 123
```

```
20BCE1025_Abhishek_N_N@ud:~$ mpiexec -np 100 ./matrix_lab10


C =
 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200
0 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200
 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200
0 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200
 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200
0 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200
 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200
0 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200
 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200
0 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200
 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200
0 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200
 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200
0 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200
 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200
0 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200
 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200
0 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200
 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200
0 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200
 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200
0 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200
 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200
0 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200
 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200
0 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200
```

```
20BCE1025_Abhishek_N_N@ud:~$ mpiexec -np 200 ./matrix_lab10


C =
 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 4
0 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400
00 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400
400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 40
 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 4
0 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400
00 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400
400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 40
 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 4
0 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400
00 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400
400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 40
 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 4
0 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400
00 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400
400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 40
 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 4
0 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400
00 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400 400
```

20BCE1025_Abhishek_N_N_Lab_11_CUDA_Parallel and Distributed Computing(CSE4001)

November 5, 2022

[1]:
```
!nvcc --version
```

```
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2021 NVIDIA Corporation
Built on Sun_Feb_14_21:12:58_PST_2021
Cuda compilation tools, release 11.2, V11.2.152
Build cuda_11.2.r11.2/compiler.29618528_0
```

[2]:
```
!nvidia-smi
```

```
Sat Nov  5 15:17:09 2022
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 460.32.03    Driver Version: 460.32.03    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla T4            Off  | 00000000:00:04.0 Off |                    0 |
| N/A   65C    P0    29W /  70W |      0MiB / 15109MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

[3]:
```
!pip install git+https://github.com/andreinechaev/nvcc4jupyter.git
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
wheels/public/simple/
Collecting git+https://github.com/andreinechaev/nvcc4jupyter.git
```

```
  Cloning https://github.com/andreinechaev/nvcc4jupyter.git to /tmp/pip-req-
build-uulyfjs_
  Running command git clone -q https://github.com/andreinechaev/nvcc4jupyter.git
/tmp/pip-req-build-uulyfjs_
Building wheels for collected packages: NVCCPlugin
  Building wheel for NVCCPlugin (setup.py) … done
  Created wheel for NVCCPlugin: filename=NVCCPlugin-0.0.2-py3-none-any.whl
size=4306
sha256=121337c267896d72b8ae53a2667b8287a5e374ceaad47723d37496a3223ee975
  Stored in directory: /tmp/pip-ephem-wheel-cache-5jgnm9vx/wheels/ca/33/8d/3c86e
b85e97d2b6169d95c6e8f2c297fdec60db6e84cb56f5e
Successfully built NVCCPlugin
Installing collected packages: NVCCPlugin
Successfully installed NVCCPlugin-0.0.2
```

[4]:
```
%load_ext nvcc_plugin
```

```
created output directory at /content/src
Out bin /content/result.out
```

[5]:
```
%%cu
#include<stdio.h>
#include<cuda.h>

int main()
{
    cudaDeviceProp p;
    int device_id;
    int major;
    int minor;


    cudaGetDevice(&device_id);
    cudaGetDeviceProperties(&p,device_id);

    major=p.major;
    minor=p.minor;

    printf("Name of GPU on your system is %s\n",p.name);

    printf("\n Compute Capability of a current GPU on your system is %d.
 ↪%d",major,minor);

    return 0;
}
```

```
Name of GPU on your system is Tesla T4
```

```
Compute Capability of a current GPU on your system is 7.5
```

1 example 1 matrix mul

[6]:
```
%%cu
#include <cassert>
#include <cstddef>
#include <cstdint>
#include <iomanip>
#include <iostream>
#include <random>
#include <stdexcept>
#include <vector>

#define BLOCK_DIM 32

#define checkCuda(val) check((val), #val, __FILE__, __LINE__)
template <typename T>
void check(T err, const char* const func, const char* const file,
           const int line)
{
    if (err != cudaSuccess)
    {
        std::cerr << "CUDA Runtime Error at: " << file << ":" << line
                  << std::endl;
        std::cerr << cudaGetErrorString(err) << " " << func << std::endl;
        std::exit(EXIT_FAILURE);
    }
}

template <typename T>
std::vector<T> create_rand_vector(size_t n)
{
    std::random_device r;
    std::default_random_engine e(r());
    std::uniform_int_distribution<int> uniform_dist(-256, 256);

    std::vector<T> vec(n);
    for (size_t i{0}; i < n; ++i)
    {
        vec.at(i) = static_cast<T>(uniform_dist(e));
    }

    return vec;
}
```

```
// mat_1: m x n
// mat_2: n x p
// mat_3: m x p
template <typename T>
void mm(T const* mat_1, T const* mat_2, T* mat_3, size_t m, size_t n, size_t p)
{
    // Compute the cells in mat_3 sequentially.
    for (size_t i{0}; i < m; ++i)
    {
        for (size_t j{0}; j < p; ++j)
        {
            T acc_sum{0};
            for (size_t k{0}; k < n; ++k)
            {
                acc_sum += mat_1[i * n + k] * mat_2[k * p + j];
            }
            mat_3[i * p + j] = acc_sum;
        }
    }
}

// mat_1: b x m x n
// mat_2: b x n x p
// mat_3: b x m x p
template <typename T>
void bmm(T const* mat_1, T const* mat_2, T* mat_3, size_t b, size_t m, size_t n,
         size_t p)
{
    // Iterate through the batch dimension.
    for (size_t i{0}; i < b; ++i)
    {
        mm(mat_1 + i * (m * n), mat_2 + i * (n * p), mat_3 + i * (m * p), m, n,
           p);
    }
}

template <typename T>
__global__ void mm_kernel(T const* mat_1, T const* mat_2, T* mat_3, size_t m,
                          size_t n, size_t p)
{
    // 2D block and 2D thread
    // Each thread computes one cell in mat_3.
    size_t i{blockIdx.y * blockDim.y + threadIdx.y};
    size_t j{blockIdx.x * blockDim.x + threadIdx.x};

    // Do not process outside the matrix.
    // Do not forget the equal sign!
```

```
    if ((i >= m) || (j >= p))
    {
        return;
    }

    T acc_sum{0};
    for (size_t k{0}; k < n; ++k)
    {
        acc_sum += mat_1[i * n + k] * mat_2[k * p + j];
    }
    mat_3[i * p + j] = acc_sum;
}

// It should be straightforward to extend a kernel to support batching.
template <typename T>
__global__ void bmm_kernel(T const* mat_1, T const* mat_2, T* mat_3, size_t b,
                           size_t m, size_t n, size_t p)
{
    // 2D block and 2D thread
    // Each thread computes one cell in mat_3.
    size_t i{blockIdx.y * blockDim.y + threadIdx.y};
    size_t j{blockIdx.x * blockDim.x + threadIdx.x};

    // Do not process outside the matrix.
    // Do not forget the equal sign!
    if ((i >= m) || (j >= p))
    {
        return;
    }

    // Process the cell of the same index along the batch dimension.
    for (size_t l{0}; l < b; ++l)
    {
        T acc_sum{0};
        for (size_t k{0}; k < n; ++k)
        {
            acc_sum +=
                mat_1[l * m * n + i * n + k] * mat_2[l * n * p + k * p + j];
        }
        mat_3[l * m * p + i * p + j] = acc_sum;
    }
}

template <typename T>
void mm_cuda(T const* mat_1, T const* mat_2, T* mat_3, size_t m, size_t n,
             size_t p)
{
```

```
    dim3 threads_per_block(BLOCK_DIM, BLOCK_DIM);
    dim3 blocks_per_grid(1, 1);
    blocks_per_grid.x = std::ceil(static_cast<double>(p) /
                                  static_cast<double>(threads_per_block.x));
    blocks_per_grid.y = std::ceil(static_cast<double>(m) /
                                  static_cast<double>(threads_per_block.y));
    mm_kernel<<<blocks_per_grid, threads_per_block>>>(mat_1, mat_2, mat_3, m, n,
                                                      p);
}

template <typename T>
void bmm_cuda(T const* mat_1, T const* mat_2, T* mat_3, size_t b, size_t m,
              size_t n, size_t p)
{
    dim3 threads_per_block(BLOCK_DIM, BLOCK_DIM);
    dim3 blocks_per_grid(1, 1);
    blocks_per_grid.x = std::ceil(static_cast<double>(p) /
                                  static_cast<double>(threads_per_block.x));
    blocks_per_grid.y = std::ceil(static_cast<double>(m) /
                                  static_cast<double>(threads_per_block.y));
    bmm_kernel<<<blocks_per_grid, threads_per_block>>>(mat_1, mat_2, mat_3, b,
                                                       m, n, p);
}

template <typename T>
bool allclose(std::vector<T> const& vec_1, std::vector<T> const& vec_2,
              T const& abs_tol)
{
    if (vec_1.size() != vec_2.size())
    {
        return false;
    }
    for (size_t i{0}; i < vec_1.size(); ++i)
    {
        if (std::abs(vec_1.at(i) - vec_2.at(i)) > abs_tol)
        {
            std::cout << vec_1.at(i) << " " << vec_2.at(i) << std::endl;
            return false;
        }
    }
    return true;
}

template <typename T>
bool random_test_mm_cuda(size_t m, size_t n, size_t p)
{
    std::vector<T> const mat_1_vec{create_rand_vector<T>(m * n)};
```

```
    std::vector<T> const mat_2_vec{create_rand_vector<T>(n * p)};
    std::vector<T> mat_3_vec(m * p);
    std::vector<T> mat_4_vec(m * p);
    T const* mat_1{mat_1_vec.data()};
    T const* mat_2{mat_2_vec.data()};
    T* mat_3{mat_3_vec.data()};
    T* mat_4{mat_4_vec.data()};

    mm(mat_1, mat_2, mat_3, m, n, p);

    T *d_mat_1, *d_mat_2, *d_mat_4;

    // Allocate device buffer.
    checkCuda(cudaMalloc(&d_mat_1, sizeof(T) * mat_1_vec.size()));
    checkCuda(cudaMalloc(&d_mat_2, sizeof(T) * mat_2_vec.size()));
    checkCuda(cudaMalloc(&d_mat_4, sizeof(T) * mat_4_vec.size()));

    // Copy data from host to device.
    checkCuda(cudaMemcpy(d_mat_1, mat_1, sizeof(T) * mat_1_vec.size(),
                         cudaMemcpyHostToDevice));
    checkCuda(cudaMemcpy(d_mat_2, mat_2, sizeof(T) * mat_2_vec.size(),
                         cudaMemcpyHostToDevice));

    // Run matrix multiplication on GPU.
    mm_cuda(d_mat_1, d_mat_2, d_mat_4, m, n, p);
    cudaDeviceSynchronize();
    cudaError_t err{cudaGetLastError()};
    if (err != cudaSuccess)
    {
        std::cerr << "CUDA Matrix Multiplication kernel failed to execute."
                  << std::endl;
        std::cerr << cudaGetErrorString(err) << std::endl;
        std::exit(EXIT_FAILURE);
    }

    // Copy data from device to host.
    checkCuda(cudaMemcpy(mat_4, d_mat_4, sizeof(T) * mat_4_vec.size(),
                         cudaMemcpyDeviceToHost));

    // Free device buffer.
    checkCuda(cudaFree(d_mat_1));
    checkCuda(cudaFree(d_mat_2));
    checkCuda(cudaFree(d_mat_4));

    return allclose<T>(mat_3_vec, mat_4_vec, 1e-4);
}
```

```
template <typename T>
bool random_test_bmm_cuda(size_t b, size_t m, size_t n, size_t p)
{
    std::vector<T> const mat_1_vec{create_rand_vector<T>(b * m * n)};
    std::vector<T> const mat_2_vec{create_rand_vector<T>(b * n * p)};
    std::vector<T> mat_3_vec(b * m * p);
    std::vector<T> mat_4_vec(b * m * p);
    T const* mat_1{mat_1_vec.data()};
    T const* mat_2{mat_2_vec.data()};
    T* mat_3{mat_3_vec.data()};
    T* mat_4{mat_4_vec.data()};

    bmm(mat_1, mat_2, mat_3, b, m, n, p);

    T *d_mat_1, *d_mat_2, *d_mat_4;

    // Allocate device buffer.
    checkCuda(cudaMalloc(&d_mat_1, sizeof(T) * mat_1_vec.size()));
    checkCuda(cudaMalloc(&d_mat_2, sizeof(T) * mat_2_vec.size()));
    checkCuda(cudaMalloc(&d_mat_4, sizeof(T) * mat_4_vec.size()));

    // Copy data from host to device.
    checkCuda(cudaMemcpy(d_mat_1, mat_1, sizeof(T) * mat_1_vec.size(),
                         cudaMemcpyHostToDevice));
    checkCuda(cudaMemcpy(d_mat_2, mat_2, sizeof(T) * mat_2_vec.size(),
                         cudaMemcpyHostToDevice));

    // Run matrix multiplication on GPU.
    bmm_cuda(d_mat_1, d_mat_2, d_mat_4, b, m, n, p);
    cudaDeviceSynchronize();
    cudaError_t err{cudaGetLastError()};
    if (err != cudaSuccess)
    {
        std::cerr << "CUDA Matrix Multiplication kernel failed to execute."
                  << std::endl;
        std::cerr << cudaGetErrorString(err) << std::endl;
        std::exit(EXIT_FAILURE);
    }

    // Copy data from device to host.
    checkCuda(cudaMemcpy(mat_4, d_mat_4, sizeof(T) * mat_4_vec.size(),
                         cudaMemcpyDeviceToHost));

    // Free device buffer.
    checkCuda(cudaFree(d_mat_1));
    checkCuda(cudaFree(d_mat_2));
    checkCuda(cudaFree(d_mat_4));
```

```
    return allclose<T>(mat_3_vec, mat_4_vec, 1e-4);
}

template <typename T>
bool random_multiple_test_mm_cuda(size_t num_tests)
{
    std::random_device r;
    std::default_random_engine e(r());
    std::uniform_int_distribution<int> uniform_dist(1, 256);

    size_t m{0}, n{0}, p{0};
    bool success{false};

    for (size_t i{0}; i < num_tests; ++i)
    {
        m = static_cast<size_t>(uniform_dist(e));
        n = static_cast<size_t>(uniform_dist(e));
        p = static_cast<size_t>(uniform_dist(e));
        success = random_test_mm_cuda<T>(m, n, p);
        if (!success)
        {
            return false;
        }
    }

    return true;
}

template <typename T>
bool random_multiple_test_bmm_cuda(size_t num_tests)
{
    std::random_device r;
    std::default_random_engine e(r());
    std::uniform_int_distribution<int> uniform_dist(1, 256);

    size_t b{0}, m{0}, n{0}, p{0};
    bool success{false};

    for (size_t i{0}; i < num_tests; ++i)
    {
        b = static_cast<size_t>(uniform_dist(e));
        m = static_cast<size_t>(uniform_dist(e));
        n = static_cast<size_t>(uniform_dist(e));
        p = static_cast<size_t>(uniform_dist(e));
        success = random_test_bmm_cuda<T>(b, m, n, p);
        if (!success)
```

```
        {
            return false;
        }
    }

    return true;
}

template <typename T>
float measure_latency_mm_cuda(size_t m, size_t n, size_t p, size_t num_tests,
                              size_t num_warmups)
{
    cudaEvent_t startEvent, stopEvent;
    float time{0.0f};

    checkCuda(cudaEventCreate(&startEvent));
    checkCuda(cudaEventCreate(&stopEvent));

    T *d_mat_1, *d_mat_2, *d_mat_4;

    // Allocate device buffer.
    checkCuda(cudaMalloc(&d_mat_1, sizeof(T) * m * n));
    checkCuda(cudaMalloc(&d_mat_2, sizeof(T) * n * p));
    checkCuda(cudaMalloc(&d_mat_4, sizeof(T) * m * p));

    for (size_t i{0}; i < num_warmups; ++i)
    {
        mm_cuda(d_mat_1, d_mat_2, d_mat_4, m, n, p);
    }

    checkCuda(cudaEventRecord(startEvent, 0));
    for (size_t i{0}; i < num_tests; ++i)
    {
        mm_cuda(d_mat_1, d_mat_2, d_mat_4, m, n, p);
    }
    checkCuda(cudaEventRecord(stopEvent, 0));
    checkCuda(cudaEventSynchronize(stopEvent));
    cudaError_t err{cudaGetLastError()};
    if (err != cudaSuccess)
    {
        std::cerr << "CUDA Matrix Multiplication kernel failed to execute."
                  << std::endl;
        std::cerr << cudaGetErrorString(err) << std::endl;
        std::exit(EXIT_FAILURE);
    }
    checkCuda(cudaEventElapsedTime(&time, startEvent, stopEvent));
```

```
    // Free device buffer.
    checkCuda(cudaFree(d_mat_1));
    checkCuda(cudaFree(d_mat_2));
    checkCuda(cudaFree(d_mat_4));

    float latency{time / num_tests};

    return latency;
}

template <typename T>
float measure_latency_bmm_cuda(size_t b, size_t m, size_t n, size_t p,
                               size_t num_tests, size_t num_warmups)
{
    cudaEvent_t startEvent, stopEvent;
    float time{0.0f};

    checkCuda(cudaEventCreate(&startEvent));
    checkCuda(cudaEventCreate(&stopEvent));

    T *d_mat_1, *d_mat_2, *d_mat_4;

    // Allocate device buffer.
    checkCuda(cudaMalloc(&d_mat_1, sizeof(T) * b * m * n));
    checkCuda(cudaMalloc(&d_mat_2, sizeof(T) * b * n * p));
    checkCuda(cudaMalloc(&d_mat_4, sizeof(T) * b * m * p));

    for (size_t i{0}; i < num_warmups; ++i)
    {
        bmm_cuda(d_mat_1, d_mat_2, d_mat_4, b, m, n, p);
    }

    checkCuda(cudaEventRecord(startEvent, 0));
    for (size_t i{0}; i < num_tests; ++i)
    {
        bmm_cuda(d_mat_1, d_mat_2, d_mat_4, b, m, n, p);
    }
    checkCuda(cudaEventRecord(stopEvent, 0));
    checkCuda(cudaEventSynchronize(stopEvent));
    cudaError_t err{cudaGetLastError()};
    if (err != cudaSuccess)
    {
        std::cerr << "CUDA Matrix Multiplication kernel failed to execute."
                  << std::endl;
        std::cerr << cudaGetErrorString(err) << std::endl;
        std::exit(EXIT_FAILURE);
    }
```

```
    checkCuda(cudaEventElapsedTime(&time, startEvent, stopEvent));

    // Free device buffer.
    checkCuda(cudaFree(d_mat_1));
    checkCuda(cudaFree(d_mat_2));
    checkCuda(cudaFree(d_mat_4));

    float latency{time / num_tests};

    return latency;
}

int main()
{
    constexpr size_t num_tests{10};

    assert(random_multiple_test_mm_cuda<int32_t>(num_tests));
    assert(random_multiple_test_mm_cuda<float>(num_tests));
    assert(random_multiple_test_mm_cuda<double>(num_tests));
    assert(random_multiple_test_bmm_cuda<int32_t>(num_tests));
    assert(random_multiple_test_bmm_cuda<float>(num_tests));
    assert(random_multiple_test_bmm_cuda<double>(num_tests));

    constexpr size_t num_measurement_tests{100};
    constexpr size_t num_measurement_warmups{10};
    size_t b{128}, m{1024}, n{1024}, p{1024};

    float mm_cuda_int32_latency{measure_latency_mm_cuda<int32_t>(
        m, n, p, num_measurement_tests, num_measurement_warmups)};
    float mm_cuda_float_latency{measure_latency_mm_cuda<float>(
        m, n, p, num_measurement_tests, num_measurement_warmups)};
    float mm_cuda_double_latency{measure_latency_mm_cuda<double>(
        m, n, p, num_measurement_tests, num_measurement_warmups)};

    float bmm_cuda_int32_latency{measure_latency_bmm_cuda<int32_t>(
        b, m, n, p, num_measurement_tests, num_measurement_warmups)};
    float bmm_cuda_float_latency{measure_latency_bmm_cuda<float>(
        b, m, n, p, num_measurement_tests, num_measurement_warmups)};
    float bmm_cuda_double_latency{measure_latency_bmm_cuda<double>(
        b, m, n, p, num_measurement_tests, num_measurement_warmups)};

    std::cout << "Matrix Multiplication CUDA Latency" << std::endl;
    std::cout << "m: " << m << " "
              << "n: " << n << " "
              << "p: " << p << std::endl;
    std::cout << "INT32: " << std::fixed << std::setprecision(5)
              << mm_cuda_int32_latency << " ms" << std::endl;
```

```
    std::cout << "FLOAT: " << std::fixed << std::setprecision(5)
              << mm_cuda_float_latency << " ms" << std::endl;
    std::cout << "DOUBLE: " << std::fixed << std::setprecision(5)
              << mm_cuda_double_latency << " ms" << std::endl;

    std::cout << "Batched Matrix Multiplication CUDA Latency" << std::endl;
    std::cout << "b: " << b << " "
              << "m: " << m << " "
              << "n: " << n << " "
              << "p: " << p << std::endl;
    std::cout << "INT32: " << std::fixed << std::setprecision(5)
              << bmm_cuda_int32_latency << " ms" << std::endl;
    std::cout << "FLOAT: " << std::fixed << std::setprecision(5)
              << bmm_cuda_float_latency << " ms" << std::endl;
    std::cout << "DOUBLE: " << std::fixed << std::setprecision(5)
              << bmm_cuda_double_latency << " ms" << std::endl;
}
```

```
Matrix Multiplication CUDA Latency
m: 1024 n: 1024 p: 1024
INT32: 4.12928 ms
FLOAT: 3.46946 ms
DOUBLE: 9.34288 ms
Batched Matrix Multiplication CUDA Latency
b: 128 m: 1024 n: 1024 p: 1024
INT32: 512.25275 ms
FLOAT: 508.87042 ms
DOUBLE: 1337.29053 ms
```

2 Example 2

[7]:
```
%%cu
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

// CUDA kernel. Each thread takes care of one element of c
__global__ void vecAdd(double *a, double *b, double *c, int n)
{
    // Get our global thread ID
    int id = blockIdx.x*blockDim.x+threadIdx.x;

    // Make sure we do not go out of bounds
    if (id < n)
```

```
        c[id] = a[id] + b[id];
}

int main( int argc, char* argv[] )
{
    // Size of vectors
    int n = 100000;

    // Host input vectors
    double *h_a;
    double *h_b;
    //Host output vector
    double *h_c;

    // Device input vectors
    double *d_a;
    double *d_b;
    //Device output vector
    double *d_c;

    // Size, in bytes, of each vector
    size_t bytes = n*sizeof(double);

    // Allocate memory for each vector on host
    h_a = (double*)malloc(bytes);
    h_b = (double*)malloc(bytes);
    h_c = (double*)malloc(bytes);

    // Allocate memory for each vector on GPU
    cudaMalloc(&d_a, bytes);
    cudaMalloc(&d_b, bytes);
    cudaMalloc(&d_c, bytes);

    int i;
    // Initialize vectors on host
    for( i = 0; i < n; i++ ) {
        h_a[i] = sin(i)*sin(i);
        h_b[i] = cos(i)*cos(i);
    }

    // Copy host vectors to device
    cudaMemcpy( d_a, h_a, bytes, cudaMemcpyHostToDevice);
    cudaMemcpy( d_b, h_b, bytes, cudaMemcpyHostToDevice);

    int blockSize, gridSize;

    // Number of threads in each thread block
```

```
    blockSize = 1024;

    // Number of thread blocks in grid
    gridSize = (int)ceil((float)n/blockSize);

    // Execute the kernel
    vecAdd<<<gridSize, blockSize>>>(d_a, d_b, d_c, n);

    // Copy array back to host
    cudaMemcpy( h_c, d_c, bytes, cudaMemcpyDeviceToHost );

    // Sum up vector c and print result divided by n, this should equal 1␣
 ↪within error
    double sum = 0;
    for(i=0; i<n; i++)
        sum += h_c[i];
    printf("final result: %f\n", sum/n);

    // Release device memory
    cudaFree(d_a);
    cudaFree(d_b);
    cudaFree(d_c);

    // Release host memory
    free(h_a);
    free(h_b);
    free(h_c);

    return 0;
}
```

```
final result: 1.000000
```

[ ]:
```
!apt-get install texlive texlive-xetex texlive-latex-extra pandoc
!pip install pypandoc
from google.colab import drive
drive.mount('/content/drive')
!cp "/content/drive/MyDrive/5th_fall_sem_22_23/cse4001_pdc_l/
 ↪20BCE1025_Abhishek_N_N_Lab_11_CUDA_Parallel and Distributed␣
 ↪Computing(CSE4001).ipynb" ./
!jupyter nbconvert --to PDF "20BCE1025_Abhishek_N_N_Lab_11_CUDA_Parallel and␣
 ↪Distributed Computing(CSE4001).ipynb"
```

```
Reading package lists… Done
Building dependency tree
Reading state information… Done
```

```
pandoc is already the newest version (1.19.2.4~dfsg-1build4).
texlive is already the newest version (2017.20180305-1).
texlive-latex-extra is already the newest version (2017.20180305-2).
texlive-xetex is already the newest version (2017.20180305-1).
The following package was automatically installed and is no longer required:
  libnvidia-common-460
Use 'apt autoremove' to remove it.
0 upgraded, 0 newly installed, 0 to remove and 5 not upgraded.
```

Name : Abhishek N N
Reg.No : 20BCE1025
Email : abhishek.nn2020@vitstudent.ac.in

Parallel and Distributed Computing(CSE4001)

Lab12- RMI

MulI.java

```
import java.rmi.Remote;
public interface MulI extends Remote{
	public int mul(int x, int y) throws Exception;
}
```

MulC.java

```
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

public class MulC extends UnicastRemoteObject implements MulI{
	protected MulC() throws RemoteException{
		super();
	}
	public int mul(int x, int y) throws Exception{
		return x*y;
	}
}
```

Server.java

```
import java.rmi.Naming;

public class Server{
	public static void main(String[] args) throws Exception{
		MulC obj= new MulC();
		Naming.rebind("MUL",obj);
		System.out.println("server started");
	}
}
```

Client.java

```
import java.rmi.Naming;

public class Client{
	public static void main(String[] args) throws Exception{
		MulI obj= (MulI)Naming.lookup("MUL");
		int n= obj.mul(2,3);
		System.out.println("Multiplication: "+n);
	}
}
```

```
● abhishek_n_n_20bce1025@ud:/mnt/D/5th_fall_sem_22_23/private_5th_fall_sem_22_23/cse4001_pdc_l/lab_12_rmi(main)$ javac *.java
● abhishek_n_n_20bce1025@ud:/mnt/D/5th_fall_sem_22_23/private_5th_fall_sem_22_23/cse4001_pdc_l/lab_12_rmi(main)$ rmic MulC
  Warning: generation and use of skeletons and static stubs for JRMP
  is deprecated. Skeletons are unnecessary, and static stubs have
  been superseded by dynamically generated stubs. Users are
  encouraged to migrate away from using rmic to generate skeletons and static
  stubs. See the documentation for java.rmi.server.UnicastRemoteObject.
● abhishek_n_n_20bce1025@ud:/mnt/D/5th_fall_sem_22_23/private_5th_fall_sem_22_23/cse4001_pdc_l/lab_12_rmi(main)$ rmiregistry &
  [1] 16469
○ abhishek_n_n_20bce1025@ud:/mnt/D/5th_fall_sem_22_23/private_5th_fall_sem_22_23/cse4001_pdc_l/lab_12_rmi(main)$ java Server
  server started
```

```
● abhishek_n_n_20bce1025@ud:/mnt/D/5th_fall_sem_22_23/private_5th_fall_sem_22_23/cse4001_pdc_l/lab_12_rmi(main)$ java Client
  Multiplication: 6
○ abhishek_n_n_20bce1025@ud:/mnt/D/5th_fall_sem_22_23/private_5th_fall_sem_22_23/cse4001_pdc_l/lab_12_rmi(main)$
```