

Comparing Performance of Merge Sort in Serial and Parallel Environments

Manish V , Mohammad Rahil K A, Moulidhar B, Manoja U
USN: 1MS18CS067 , 1MS18CS072 , 1MS18CS075 , 1MS18CS068

Abstract

We must utilize all of the CPU's resources in order to get the most out of it. Sorting is the process of arranging a collection of numbers in ascending or descending order. To activate parallel programming, we apply a merging strategy in merge sort. To assess parallel programming's performance, we compare it to sequential merge sort, which does not make efficient use of the CPU. To determine whether an algorithm is more efficient, we evaluate the performance of sequential and parallel sorting algorithms. In a parallel sorting algorithm, we use a message passing interface (MPI).

Introduction

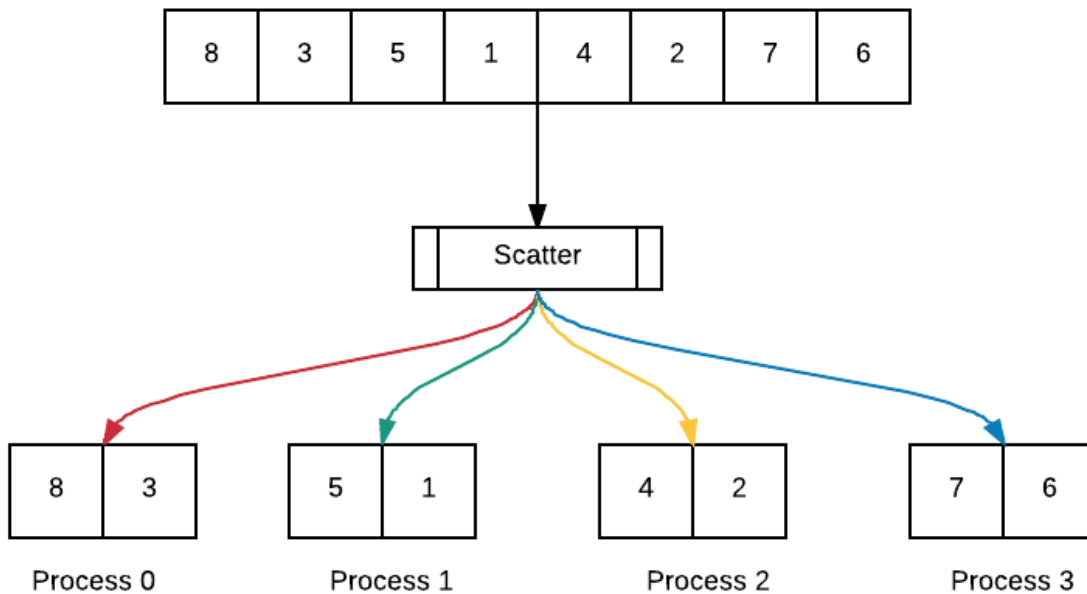
New applications always need faster processors in today's world to operate efficiently. Parallel systems are employed in a lot of business applications nowadays. These systems can handle massive amounts of data in a variety of ways, resulting in excellent efficiency. Parallel systems are all about breaking out distinct pieces of a program's instructions so that they can be executed on many CPUs at the same time.

Parallel systems, also known as tightly connected systems, are designed to reduce the execution time of programmes by dividing them into multiple fragments and processing them all at the same time. By constructing a parallel processing bundle or a combination of both entities, a parallel system can deal with many processors, machines, computers, or CPUs, among other things.

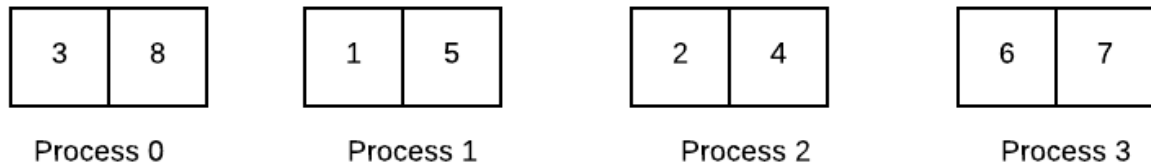
Parallel processing is commonly used to perform complex tasks and computations. Data scientists will commonly make use of parallel processing for compute and data-intensive tasks.

To parallelize this algorithm, we will use a mixed strategy in which the sublists are sorted by a sequential sorting algorithm and the merging of sublists is done in parallel between processes. We chose to stick with cases in which the number of processes is a power of two so that all processes are doing roughly the same amount of work.

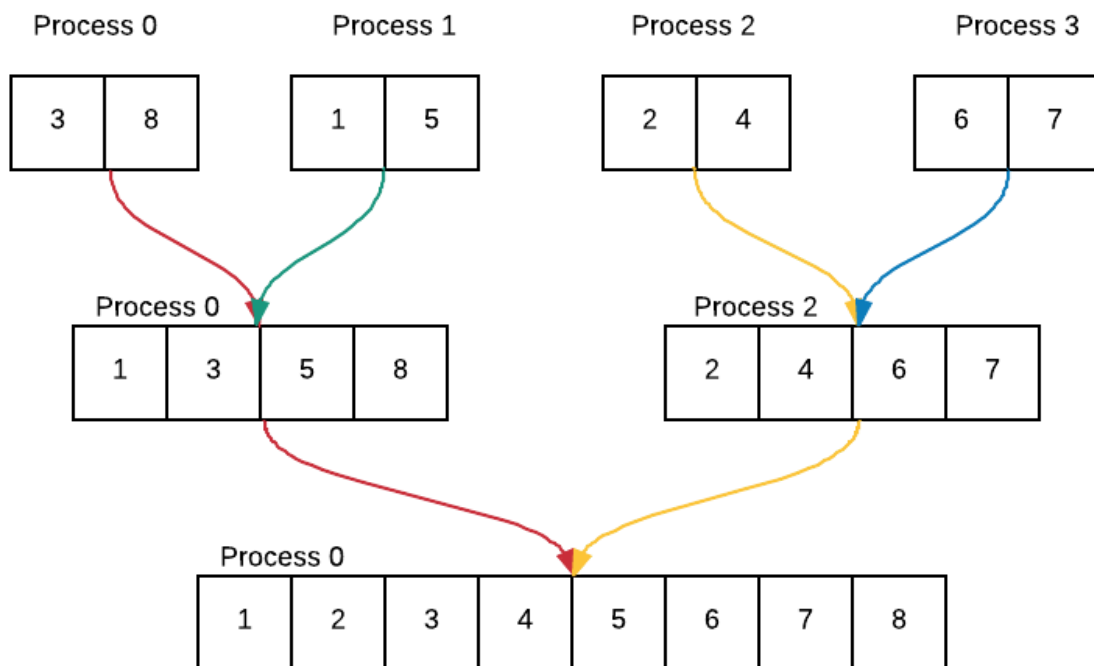
Part I: Divide list into unsorted sublists : For this portion of the problem, we begin with a single unsorted list. This list is scattered to all of the processes such that each process has an equal chunk of the list. Suppose we have 4 processes and a list containing 8 integers. The code is executing as follows:



Part II: Sort sublists : We can sort these sublists by applying a serial sorting algorithm. We use the C library function `qsort` on each process to sort the local sublist. After sorting the processes have the following sorted sublists:



Part III: Merge sublists : The merging of the sublists to form a single list is done by sending and receiving sublists between processes and merging them together. Each initial sorted sublist (with a size of 2) provides the sorted result to the parent process. That process combines the two sublists to generate a list of size 4, and then sends that result to its parent process. Lastly, the root process merges the two lists to obtain a list of size 8 that is fully sorted.



Code :

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

```

#include <time.h>
#include <mpi.h>

void parallelMerge(int *, int *, int, int, int);
void parallelMergeSort(int *, int *, int, int);

void merge(int arr[], int l, int m, int r)
{
    int i, j, k;
    int n1 = m - l + 1;
    int n2 = r - m;
    /* create temp arrays */
    int L[n1], R[n2];
    /* Copy data to temp arrays L[] and R[] */
    for (i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[m + 1 + j];
    /* Merge the temp arrays back into arr[l..r]*/
    i = 0; // Initial index of first subarray
    j = 0; // Initial index of second subarray
    k = l; // Initial index of merged subarray
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        }
        else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }
    /* Copy the remaining elements of L[], if there are any */
    while (i < n1) {

```

```

        arr[k] = L[i];
        i++;
        k++;
    }
    /* Copy the remaining elements of R[], if there are any */
    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}

```

/* l is for left index and r is right index of the sub-array of arr to be sorted */

```

void mergeSort(int arr[], int l, int r)
{
    if (l < r) {
        // Same as (l+r)/2, but avoids overflow for
        // large l and h
        int m = l + (r - l) / 2;
        // Sort first and second halves
        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);
        merge(arr, l, m, r);
    }
}

```

```

void printArray(int A[], int size)
{
    int i;
    for (i = 0; i < size; i++)
        printf("%d ", A[i]);
    printf("\n");
}

```

```

int main(int argc, char** argv) {
    /***** Create and populate the array *****/
    int n = atoi(argv[1]);
    int *original_array = malloc(n * sizeof(int));
    int c;
    srand(time(NULL));
    printf("This is the unsorted array: ");
    for(c = 0; c < n; c++) {
        original_array[c] = rand() % n;
        printf("%d ", original_array[c]);
    }
    printf("\n");
    printf("\n");
    /***** Initialize MPI *****/
    int world_rank;
    int world_size;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);

    /***** Divide the array in equal-sized chunks *****/
    int size = n/world_size;
    /***** Send each subarray to each process *****/
    int *sub_array = malloc(size * sizeof(int));
    MPI_Scatter(original_array, size, MPI_INT, sub_array, size, MPI_INT, 0,
MPI_COMM_WORLD);
    /***** Perform the merge sort on each process *****/
    int *tmp_array = malloc(size * sizeof(int));
    parallelMergeSort(sub_array, tmp_array, 0, (size - 1));
    /***** Gather the sorted subarrays into one *****/
    int *sorted = NULL;
    if(world_rank == 0) {
        sorted = malloc(n * sizeof(int));
    }
}

```

```

    MPI_Gather(sub_array, size, MPI_INT, sorted, size, MPI_INT, 0,
MPI_COMM_WORLD);
    /***** Make the final mergeSort call *****/
    clock_t begining,end;
    if(world_rank == 0) {
        int *other_array = malloc(n * sizeof(int));
        begining = clock();
        parallelMergeSort(sorted, other_array, 0, (n - 1));
        end = clock();
        /***** Display the sorted array *****/
        printf("This is the sorted array: ");
        for(c = 0; c < n; c++) {
            printf("%d ", sorted[c]);
        }
        printf("\n");
        printf("\n");
        /***** Clean up root *****/
        free(sorted);
        free(other_array);
    }
    /***** Finalize MPI *****/
    MPI_Barrier(MPI_COMM_WORLD);
    MPI_Finalize();
    float elapsed = (float)(end - begining) / CLOCKS_PER_SEC * 1000;
    printf("Parallel needed %f miliseconds \n\n", elapsed);

    printf("Given array is \n");
    printArray(original_array, n);
    clock_t s_begining = clock();
    mergeSort(original_array, 0, n - 1);
    clock_t s_end = clock();
    printf("\nSorted array is \n");
    printArray(original_array, n);

    float s_elapsed = (float)(s_end - s_begining) / CLOCKS_PER_SEC * 1000;

```

```

printf("Serial needed %f miliseconds \n\n", s_elapsed);
free(original_array);
free(sub_array);
free(tmp_array);
}

```

```

/***** Merge Function *****/
void parallelMerge(int *a, int *b, int l, int m, int r) {
    int h, i, j, k;
    h = l;
    i = l;
    j = m + 1;

    while((h <= m) && (j <= r)) {

        if(a[h] <= a[j]) {

            b[i] = a[h];
            h++;
        }
        else {
            b[i] = a[j];
            j++;
        }
        i++;
    }
    if(m < h) {
        for(k = j; k <= r; k++) {
            b[i] = a[k];
            i++;
        }
    }
    else {
        for(k = h; k <= m; k++) {
            b[i] = a[k];
            i++;
        }
    }
}

```



```

        for(k = l; k <= r; k++) {
            a[k] = b[k];
        }
    }

/***** Recursive Merge Function *****/
void parallelMergeSort(int *a, int *b, int l, int r) {
    int m;
    if(l < r) {
        m = (l + r)/2;
        parallelMergeSort(a, b, l, m);
        parallelMergeSort(a, b, (m + 1), r);
        parallelMerge(a, b, l, m, r);
    }
}

```

Output:

This is the unsorted array: 42 44 0 81 63 10 60 27 41 40 13 33 32 30 1 55 95 4 59 33 25 28 40 70 39 76 45 97 23 69 58 17 66 58 50 81 68 10 61 61 51 26 47 35 56 48 43 3 4 54 88 30 83 29 52 22 5 98 71 80 19 29 97 85 39 99 19 59 9 80 20 12 6 67 0 14 67 43 17 72 49 58 2 32 87 6 6 92 4 29 24 24 10 73 61 49 72 32 8 81

This is the sorted array: 0 0 1 2 3 4 4 4 5 6 6 6 8 9 10 10 10 12 13 14 17 17 19 19 20 22 23 24 24 25 26 27 28 29 29 29 30 30 32 32 32 33 33 35 39 39 40 40 41 42 43 43 44 45 47 48 49 49 50 51 52 54 55 56 58 58 58 59 59 60 61 61 61 63 66 67 67 68 69 70 71 72 72 73 76 80 80 81 81 81 83 85 87 88 92 95 97 97 98 99

Parallel needed 0.038000 milliseconds

Given array is

42 44 0 81 63 10 60 27 41 40 13 33 32 30 1 55 95 4 59 33 25 28 40 70 39 76 45 97 23 69 58 17 66 58 50 81 68 10 61 61 51 26 47 35 56 48 43 3 4 54 88 30 83 29 52 22 5 98 71 80 19 29 97 85 39 99 19 59 9 80 20 12 6 67 0 14 67 43 17 72 49 58 2 32 87 6 6 92 4 29 24 24 10 73 61 49 72 32 8 81

Sorted array is

0 0 1 2 3 4 4 4 5 6 6 6 8 9 10 10 10 12 13 14 17 17 19 19 20 22 23 24 24 25 26 27
28 29 29 29 30 30 32 32 32 33 33 35 39 39 40 40 41 42 43 43 44 45 47 48 49 49
50 51 52 54 55 56 58 58 58 59 59 60 61 61 61 63 66 67 67 68 69 70 71 72 72 73
76 80 80 81 81 81 83 85 87 88 92 95 97 97 98 99
Serial needed 0.071000 milliseconds

```
File Edit View Search Terminal Help
~/MCA ~./output 1000
This is the unsorted array: 498 918 95 856 791 601 509 375 257 166 739 836 955 959 352 644 119 858 673 954 120 913 316 60 544 876 267 941 80 991 494 578 262 5
89 787 53 543 648 780 800 814 520 637 121 479 341 765 950 199 790 256 319 703 572 731 600 448 350 893 880 342 387 811 956 329 598 361 872 598 141 24 764 661 1
3 885 492 354 3 442 553 793 698 224 849 623 956 449 71 658 694 304 352 434 115 308 763 65 669 987 663 163 11 427 824 25 664 669 379 667 463 933 813 514 157 66
2 137 465 463 560 124 157 864 828 591 979 137 354 396 806 341 411 321 705 838 146 730 855 815 109 522 630 394 687 144 904 349 281 721 812 842 197 970 58 26 91
3 390 163 620 786 321 313 198 643 18 388 141 748 243 308 210 118 938 956 805 83 212 155 364 934 319 558 483 641 617 509 555 7 24 175 145 346 488 343 341 507 7
32 834 607 327 142 169 445 432 478 603 515 690 110 232 976 429 142 460 71 111 969 626 118 994 153 264 692 641 959 385 500 43 219 460 371 713 629 168 145 459 7
71 13 150 881 597 126 311 739 586 734 851 908 360 321 902 513 937 946 506 897 683 7 292 902 467 663 967 448 184 464 260 955 477 410 189 426 888 852 166 475 58
6 369 383 946 690 637 811 980 935 317 229 618 676 521 872 495 537 839 296 721 303 556 28 133 318 217 559 206 69 77 33 655 798 768 953 489 405 116 821 692 786
50 310 814 571 182 310 108 373 958 181 29 514 210 514 184 779 425 390 849 503 424 856 653 192 162 142 950 278 315 642 416 365 305 231 937 839 893 397 213 851
579 594 717 141 460 901 920 885 291 769 740 67 978 394 612 492 888 562 122 204 204 539 569 861 122 858 701 15 256 266 866 187 860 935 328 320 188 248 557 831
370 298 899 348 692 511 192 932 425 314 136 629 853 58 843 975 916 544 342 172 810 560 359 22 495 687 694 683 288 251 515 658 901 766 358 945 277 550 878 702
864 366 683 70 424 878 397 341 422 740 865 584 300 225 958 148 264 4 831 552 608 698 210 509 464 920 807 93 470 685 147 687 403 183 757 828 61 154 521 836 246
386 420 547 963 731 695 228 735 878 132 343 577 695 205 393 615 12 839 86 49 986 125 452 521 882 280 935 388 153 771 635 540 543 534 503 274 229 83 362 459 2
16 705 36 911 910 782 526 922 621 964 971 959 89 776 833 323 408 768 712 562 891 699 102 434 233 957 61 462 41 423 921 609 480 310 520 391 444 398 665 65 363
989 376 804 117 209 128 525 329 192 87 220 891 541 7 124 851 68 938 244 843 211 853 323 873 373 66 317 771 84 382 486 73 111 643 190 672 771 67 2 963 155 222
206 48 229 330 899 649 620 143 492 831 348 816 705 721 234 22 845 318 757 683 743 868 326 933 540 449 353 894 412 508 117 970 908 698 300 160 348 920 655 840
104 4 8 161 725 243 183 922 561 292 958 657 160 284 942 53 734 295 299 498 155 416 469 64 467 121 224 815 394 231 7 850 235 368 11 313 963 194 235 876 839 545
533 999 830 476 404 916 771 704 766 927 472 587 343 291 61 567 106 807 150 466 657 386 186 668 699 149 214 286 25 53 832 559 405 14 35 161 282 806 217 400 85
42 340 428 333 401 347 792 208 498 258 217 884 444 885 935 945 451 221 970 505 405 529 262 771 916 423 405 723 641 806 160 683 498 589 368 899 936 160 459 43
4 770 676 318 566 913 605 511 364 179 482 869 584 363 131 708 280 907 465 355 548 271 515 583 121 104 951 20 41 112 831 827 234 507 498 801 420 103 312 785 28
2 146 6 219 510 490 927 142 397 392 497 297 16 364 232 137 821 535 510 214 999 341 41 234 201 539 35 973 995 347 110 277 494 117 496 356 959 775 498 708 520 3
47 357 888 63 589 25 884 476 887 450 476 581 492 62 782 383 97 755 730 796 866 8 642 335 856 350 646 632 848 354 152 547 711 392 611 652 769 847 480 657 298 3
08 590 142 370 372 877 819 479 608 968 697 616 610 384 824 961 30 456 161 384 960 61 447 352 672 451 122 519 932 131 169 240 721 311 963 445 189 782 276 149 7
50 974 765 713 358 941 674 389 398 187 125 710 248 573 415 272 376 889 144 308 372 313 901 93 625 864 890 166 646 166 315 749 140 432 462 851 373 488 240 123
675 717 186 276 290 953 900 19 842 44 327 566 358 580 11 335 796 901 501 443 67 168 192 560 952 654 763 677 142 3 801 169 720 339 797 363

This is the sorted array: 2 3 3 4 4 6 7 7 7 7 8 8 11 11 11 12 13 13 14 15 16 18 19 20 22 22 24 24 25 25 25 26 28 29 30 33 35 35 36 41 41 41 42 43 44 48 49 50
53 53 53 58 58 60 61 61 61 62 63 64 65 65 66 67 67 67 68 69 70 71 71 73 77 80 83 83 84 85 86 87 89 93 93 95 97 102 103 104 104 106 108 109 110 110 111 111
112 115 116 117 117 117 118 118 119 120 121 121 121 122 122 122 123 124 124 125 125 126 128 131 131 132 133 136 137 137 137 140 141 141 141 142 142 142 142 14
2 142 143 144 144 145 145 146 146 147 148 149 149 150 150 152 153 153 154 155 155 155 157 157 160 160 160 160 161 161 161 162 163 163 166 166 166 166 168 168
169 169 169 172 175 179 181 182 183 183 184 184 186 186 187 187 188 189 189 190 192 192 192 192 194 197 198 199 201 204 204 205 206 206 208 209 210 210 210 21
1 212 213 214 214 216 217 217 217 219 219 220 221 222 224 224 225 228 229 229 229 231 231 232 232 233 234 234 234 235 235 240 240 243 243 244 246 248 248 251
256 256 257 258 260 262 262 264 264 266 267 271 272 274 276 276 277 277 278 280 280 281 282 282 284 286 288 290 291 291 292 292 295 296 297 298 298 299 300 30
0 303 304 305 308 308 308 308 310 310 310 311 311 312 313 313 313 314 315 315 316 317 317 318 318 318 319 319 320 321 321 321 323 323 326 327 327 328 329 329
330 333 335 335 339 340 341 341 341 341 341 342 342 343 343 343 346 347 347 347 348 348 348 349 350 350 352 352 352 353 354 354 354 355 356 357 358 358 358 35
9 360 361 362 363 363 363 364 364 364 365 366 368 368 369 370 370 371 372 372 373 373 373 375 376 376 379 382 383 383 384 384 385 386 386 387 388 388 389 390
390 391 392 392 393 394 394 394 396 397 397 397 398 398 400 401 403 404 405 405 405 405 408 410 411 412 415 416 416 420 420 422 423 423 424 424 425 425 426 42
7 428 429 432 432 434 434 434 442 443 444 444 445 445 447 448 448 449 449 450 451 451 452 456 459 459 459 460 460 460 462 462 463 463 464 464 465 465 466 467
467 469 470 472 475 476 476 476 477 478 479 479 480 480 482 483 486 488 488 489 490 492 492 492 492 494 494 495 495 496 497 498 498 498 498 498 500 501 50
3 503 505 506 507 507 508 509 509 509 510 510 511 511 513 514 514 514 515 515 515 519 520 520 520 521 521 521 522 525 526 529 533 534 535 537 539 539 540 540
541 543 543 544 544 545 547 547 548 550 552 553 555 556 557 558 559 559 560 560 560 561 562 562 566 566 566 567 569 571 572 573 577 578 579 580 581 583 584 584 58
```

```
immanish19@Alien:/MCA
File Edit View Search Terminal Help
0 303 304 305 306 308 308 308 310 310 310 311 311 312 313 313 313 314 315 315 316 317 317 318 318 318 319 319 320 321 321 321 322 323 323 326 327 327 328 329 329
330 333 335 335 339 340 341 341 341 341 342 342 343 343 343 346 347 347 347 348 348 349 350 350 352 352 352 353 354 354 354 355 356 357 358 358 358 35
9 360 361 362 363 363 363 364 364 364 366 366 368 368 369 370 370 371 372 372 373 373 373 375 376 376 379 382 383 383 384 384 385 386 386 387 388 388 389 390
390 391 392 392 393 394 394 394 396 397 397 397 398 398 400 401 403 404 405 405 405 405 408 410 411 412 415 416 416 420 420 422 422 423 423 424 424 425 425 426 42
7 428 429 432 432 434 434 434 442 443 444 444 445 445 447 448 448 449 449 450 451 451 452 452 455 455 459 459 459 459 460 460 460 462 462 463 463 464 464 465 465 466 467
467 469 470 472 475 476 476 476 477 478 479 479 480 480 482 483 486 488 488 489 490 492 492 492 492 494 494 495 495 496 497 498 498 498 498 498 500 501 50
3 503 505 506 507 507 508 509 509 510 510 511 511 513 514 514 514 515 515 515 519 520 520 520 521 521 521 522 525 526 529 533 534 535 537 539 539 540 540
541 543 543 544 544 545 547 547 548 550 552 553 555 556 557 558 559 559 560 560 560 561 562 562 566 566 567 569 571 572 573 577 578 579 580 581 583 584 584 58
6 586 587 589 589 590 591 594 597 598 598 600 601 603 605 607 608 608 609 610 611 612 615 616 617 618 620 620 621 623 625 626 629 629 630 632 635 637 637
641 641 641 642 642 643 643 644 646 646 648 649 652 653 654 655 655 657 657 657 658 658 661 662 663 663 664 665 667 668 669 669 672 672 673 674 675 676 676 67
7 683 683 683 683 685 687 687 687 690 690 692 692 692 694 694 695 695 697 698 698 698 699 699 701 702 703 704 705 705 705 708 708 710 711 712 713 713 717
717 720 721 721 721 721 723 725 730 730 731 731 732 734 734 735 739 739 740 740 743 748 749 750 755 757 757 763 763 764 765 765 766 766 768 768 769 769 770 77
1 771 771 771 771 771 775 776 779 780 782 782 782 785 786 786 787 790 791 792 793 796 796 797 798 800 801 801 804 805 806 806 806 807 807 810 811 811 812 813
814 814 815 815 816 819 821 821 824 824 827 828 828 830 831 831 831 831 832 833 834 836 836 838 839 839 839 839 840 842 842 843 843 845 847 848 849 849 850 85
1 851 851 851 852 853 853 855 856 856 856 858 858 860 861 864 864 864 865 866 866 868 869 872 872 873 876 876 877 878 878 878 880 881 882 884 884 885 885 885
887 888 888 888 889 890 891 891 893 893 894 897 899 899 899 900 901 901 901 902 902 904 907 908 908 910 911 913 913 913 916 916 916 918 920 920 920 921 92
2 922 927 927 932 932 933 933 934 935 935 935 935 936 937 937 938 938 941 941 942 945 945 946 946 950 950 951 952 953 953 954 955 955 956 956 957 958 958
958 959 959 959 959 960 961 963 963 963 963 964 967 968 969 970 970 970 971 973 974 975 976 978 979 980 986 987 989 991 994 995 999 999

Parallel needed 0.247000 milliseconds

Given array is
498 918 95 856 791 601 509 375 257 166 739 836 955 959 352 644 119 858 673 954 120 913 316 60 544 876 267 941 80 991 494 578 262 589 787 53 543 648 780 800 81
4 520 637 121 479 341 765 950 199 790 256 319 703 572 731 600 448 350 893 880 342 387 811 956 329 598 361 872 598 141 24 764 661 13 885 492 354 3 442 553 793
698 224 849 623 956 449 71 658 694 344 352 434 115 308 763 65 669 987 663 163 11 427 824 25 664 669 379 667 463 933 813 514 157 662 137 465 463 560 124 157 86
4 828 591 979 137 354 396 806 341 411 321 705 838 146 730 855 815 109 522 630 394 687 144 904 349 281 721 812 842 197 970 58 26 913 390 163 620 786 321 313 19
8 643 18 388 141 748 243 308 210 118 938 956 805 83 212 155 364 934 319 558 483 641 617 509 555 7 24 175 145 346 488 343 341 507 732 834 607 327 142 169 445 4
32 478 603 515 690 110 232 976 429 142 460 71 111 969 626 118 994 153 264 692 641 959 385 500 43 219 460 371 713 629 168 145 459 771 13 150 881 597 126 311 73
9 586 734 851 908 360 321 902 513 937 946 506 897 683 7 292 902 467 663 967 448 184 464 260 955 477 410 189 426 888 852 166 475 586 369 383 946 690 637 811 98
0 935 317 229 618 676 521 872 495 537 839 296 721 303 556 28 133 318 217 559 206 69 77 33 655 798 768 953 489 405 116 821 692 786 50 310 814 571 182 310 108 3
73 958 181 29 514 210 514 184 779 425 390 849 503 424 856 653 192 162 142 950 278 315 642 416 365 305 231 937 839 893 397 213 851 579 594 717 141 460 901 920
885 291 769 740 67 978 394 612 492 888 562 122 204 204 539 569 861 122 858 701 15 256 266 866 187 860 935 328 320 188 248 557 831 370 298 899 348 692 511 192
932 425 314 136 629 853 58 843 975 916 544 342 172 810 560 359 22 495 687 694 683 288 153 771 635 540 543 534 503 274 229 83 362 459 216 705 36 911 910 782 526 22
41 422 740 865 584 300 225 958 148 264 4 831 552 608 698 210 509 464 920 807 93 470 685 147 687 403 183 757 828 61 154 521 836 246 386 420 547 963 731 695 228
7 735 878 132 343 577 695 205 393 615 12 839 86 49 986 125 452 522 882 280 935 388 153 771 635 540 543 534 503 274 229 83 362 459 216 705 36 911 910 782 526 22
2 621 964 971 959 89 776 833 323 408 768 712 562 891 699 102 434 233 957 61 462 41 423 921 609 480 310 520 391 444 398 665 65 363 989 376 804 117 209 128 525
379 192 831 348 816 705 721 234 22 845 318 757 683 743 868 326 933 540 449 353 894 412 508 117 970 908 698 300 160 348 920 655 840 104 4 8 161 725 243 813 922
561 292 958 657 160 284 942 53 734 295 299 498 155 416 469 64 467 121 224 815 394 231 7 850 235 368 11 313 963 194 235 876 839 545 533 999 830 476 404 916 771
7 704 766 927 472 587 343 201 61 567 106 807 150 466 657 386 186 668 699 149 214 286 25 53 832 559 405 14 35 161 282 806 217 400 85 42 340 428 333 401 347 792
208 498 258 217 884 444 885 935 945 451 221 970 505 405 529 262 771 916 423 405 723 641 806 160 683 498 589 368 899 936 160 459 434 770 676 318 566 913 605 51
1 364 179 482 869 584 363 131 708 280 907 465 355 548 271 515 583 121 104 951 20 41 112 831 827 234 507 498 801 420 103 312 785 282 146 6 219 510 490 927 142
397 392 497 297 16 364 232 137 821 535 510 214 999 341 41 234 201 539 35 973 995 347 110 277 494 117 496 356 959 775 498 708 520 347 357 888 63 589 25 884 476

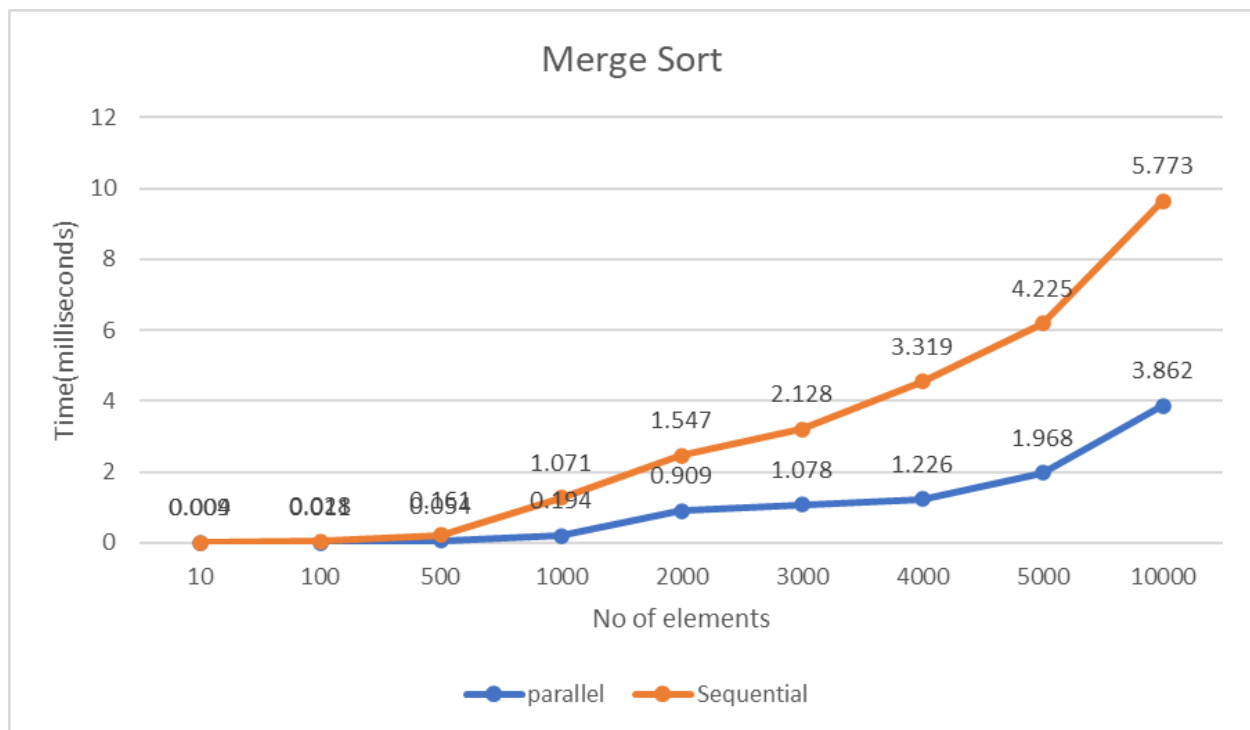
Sorted array is
2 3 3 4 4 6 7 7 7 7 8 8 11 11 11 12 13 13 14 15 16 18 19 20 22 22 24 24 25 25 25 26 28 29 30 33 35 35 36 41 41 41 42 43 44 48 49 50 53 53 53 58 58 60 61 61 61
61 62 63 64 65 65 66 67 67 67 68 69 70 71 71 73 77 80 83 83 84 85 86 87 89 93 95 97 102 103 104 104 106 108 109 110 110 111 111 112 115 116 117 117 117 117
9 118 119 120 121 121 122 122 122 123 124 124 125 125 126 128 131 131 132 133 136 137 137 137 140 141 141 141 141 142 142 142 142 143 144 144 145 145
146 146 147 148 149 149 150 150 152 153 153 154 155 155 155 157 157 160 160 160 160 161 161 161 162 163 163 166 166 166 166 168 168 169 169 169 172 175 179 18
1 182 183 183 184 184 186 186 187 188 189 189 190 192 192 192 192 194 197 198 199 201 204 204 205 206 206 208 209 210 210 210 211 211 212 213 214 216 217 217
217 217 219 219 220 221 222 224 224 225 228 229 229 229 231 231 232 232 233 234 234 234 235 235 240 240 243 243 244 246 248 248 251 256 256 257 258 260 262 26
2 264 264 266 267 271 272 274 276 276 277 277 278 280 280 281 282 282 284 286 288 290 291 291 292 292 295 296 297 298 298 299 300 300 303 304 305 308 308 308
308 310 310 310 311 311 312 313 313 313 314 315 315 316 317 317 318 318 318 319 319 320 321 321 321 323 323 326 327 327 328 329 329 330 333 335 335 339 340 34
1 341 341 341 341 342 342 343 343 346 347 347 347 348 348 348 349 350 350 352 352 352 353 354 354 354 355 356 357 358 358 358 359 360 361 362 363 363 363
364 364 364 365 366 368 368 369 370 370 371 372 372 373 373 373 376 376 376 379 382 383 383 384 384 385 386 386 387 388 388 389 390 390 391 392 392 393 394 39
4 394 396 397 397 397 398 398 400 401 403 404 405 405 405 408 410 411 412 415 416 416 420 420 422 423 423 424 424 425 425 426 427 428 429 429 432 432 434 434
434 442 443 444 444 445 445 447 448 448 449 449 450 451 451 452 452 459 459 460 460 460 462 462 463 463 464 464 465 465 466 467 467 469 470 472 475 476 47
6 476 477 478 479 479 480 480 482 483 486 488 488 489 490 492 492 492 492 494 494 495 495 496 497 498 498 498 498 498 500 501 503 503 505 506 507 507 508
509 509 509 510 510 511 511 513 514 514 514 515 515 515 510 520 520 520 521 521 521 522 525 526 529 533 534 535 537 539 539 540 540 541 543 543 544 544 545 54
7 547 548 550 552 553 555 557 558 559 559 560 560 561 562 562 566 566 567 569 571 572 573 577 578 579 580 581 583 584 586 586 587 589 589 589 590
591 594 597 598 598 600 601 603 605 607 608 608 609 610 611 612 615 617 618 620 620 621 623 625 626 629 629 630 632 635 637 637 641 641 641 642 642 643 64
3 644 646 646 648 649 652 653 654 655 655 657 657 657 658 658 661 662 663 663 664 665 667 668 669 669 672 672 673 674 675 676 676 677 683 683 683 683 685
687 687 687 690 690 692 692 692 694 694 695 695 697 698 698 698 699 699 701 702 703 704 705 705 705 708 708 710 711 712 713 713 717 717 720 721 721 721 721
73 725 730 731 731 732 734 734 735 739 740 740 743 748 749 750 755 757 757 763 763 764 765 765 766 766 768 768 769 769 770 771 771 771 771 771 775
776 779 780 782 782 782 785 786 786 787 790 791 792 793 796 796 797 798 800 801 801 804 805 806 806 806 807 807 810 811 811 812 813 814 814 815 815 816 819 82
1 821 824 827 828 828 830 831 831 831 831 832 833 834 836 836 838 839 839 839 839 840 842 842 843 843 845 847 848 849 849 850 851 851 851 851 852 853 853
855 856 856 856 858 858 860 861 864 864 864 865 866 866 868 869 872 873 876 876 877 878 878 878 880 881 882 884 884 885 885 885 887 888 888 888 889 890 89
1 891 893 893 894 897 899 899 899 900 901 901 901 901 902 902 902 904 907 908 908 910 911 913 913 913 916 916 916 918 920 920 920 922 922 927 927 932 932 933
933 934 935 935 935 935 936 937 937 938 938 941 941 942 945 945 946 946 950 950 951 952 953 953 954 955 955 956 956 956 957 958 958 959 959 959 959 960 96
1 963 963 963 964 967 968 969 970 970 970 971 973 974 975 976 978 979 980 986 987 989 991 994 995 999 999

Serial needed 0.630000 milliseconds
```

Similarly we ran the algorithm in serial and parallel environments and obtained the table as shown below.

n (Number of elements in the array)	Time taken for the Parallel Algorithm in ms	Time taken for the Serial Algorithm in ms
10	0.009	0.004
100	0.011	0.028
500	0.054	0.161
1000	0.194	1.071
2000	0.909	1.547
3000	1.078	2.128
4000	1.226	3.319
5000	1.968	4.225
10000	3.826	5.773

Comparison of serial and parallel merge sort algorithms :



This is the output graph we obtained after running the merge sort algorithm in serial and parallel environments. We observed as the number of elements of the array increases the sequential run time increases exponentially. But in case of parallel the run time is increasing very slowly which shows that parallel merge sort algorithm is much more efficient than the serial merge sort algorithm. We are using a dual core machine and still we got great results during parallel execution. If we had a system with multiple cores then we would have got more better results. So we can conclude that Merge Sort Algorithm is more efficient in parallel environment.

References

1. Jeon, Minsoo & Kim, Dongseung. (2003). Parallel Merge Sort with Load Balancing. International Journal of Parallel Programming. 31. 21-33. 10.1023/A:1021734202931.
2. https://stanford.edu/~rezab/classes/cme323/S16/notes/Lecture04/cme323_lec4.pdf
3. https://www.tutorialspoint.com/parallel_algorithm/parallel_algorithm_sorting
4. <https://pediaa.com/what-is-the-difference-between-serial-and-parallel-processing-in-computer-architecture>