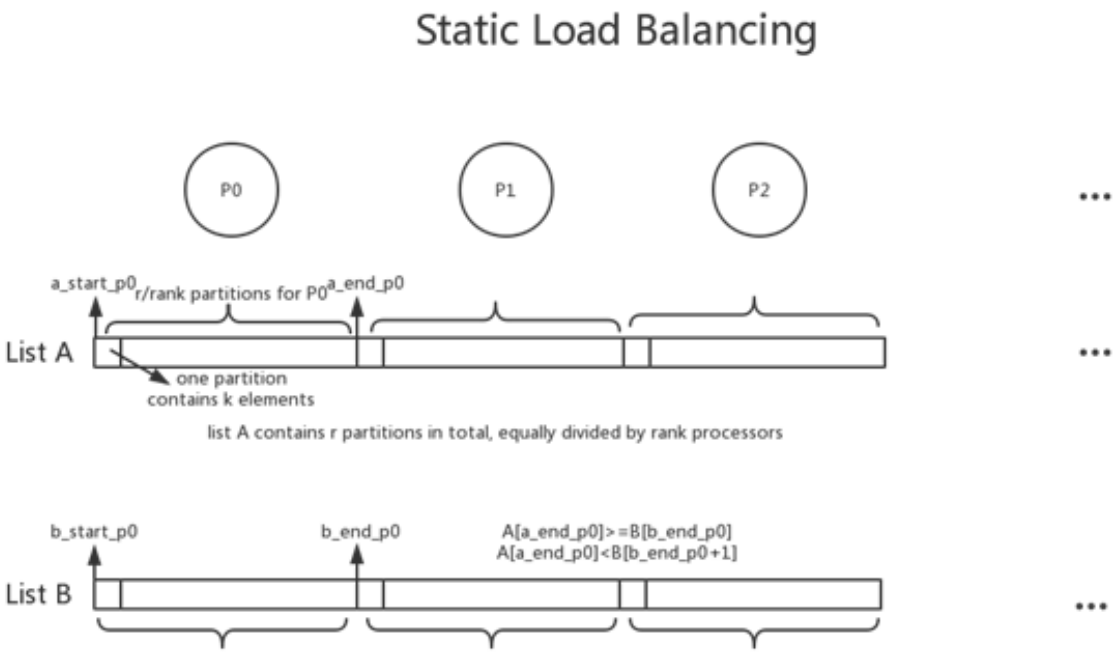
CP 431/631 Assignment 2

By group 2 (Omer Tal, Elizabeth Gorbonos, Tianran Wang)

1. Assignment description:

The assignment is to write the parallel merging algorithm to merge two large randomly generated sorted arrays.
To accomplish this, we have written a MPI python 3 program (appendix 1).
We assume the arrays are of the same length which is a power of two ($n = 2^k$). *k* is a command line parameter.

First, we calculate the number of partitions $r = \left\lceil \frac{n}{k} \right\rceil$. Each partition consists of *k* elements, besides possibly the last one. We then generate two random sorted arrays *A* and *B* of length *n* in process 0 and broadcast them to all other processes.
The work division is done by a method of static load balancing, each process calculates the range of partitions in array *A* to works on according to: $r_p = \left\lfloor \frac{r}{P} \right\rfloor + \begin{cases} 1 & if\ p < mod(r,P) \\ 0 & else \end{cases}$.
Each process's range starts from: $r_{start,p} = r * \left\lfloor \frac{r}{P} \right\rfloor + min(p, mod(r,P))$.



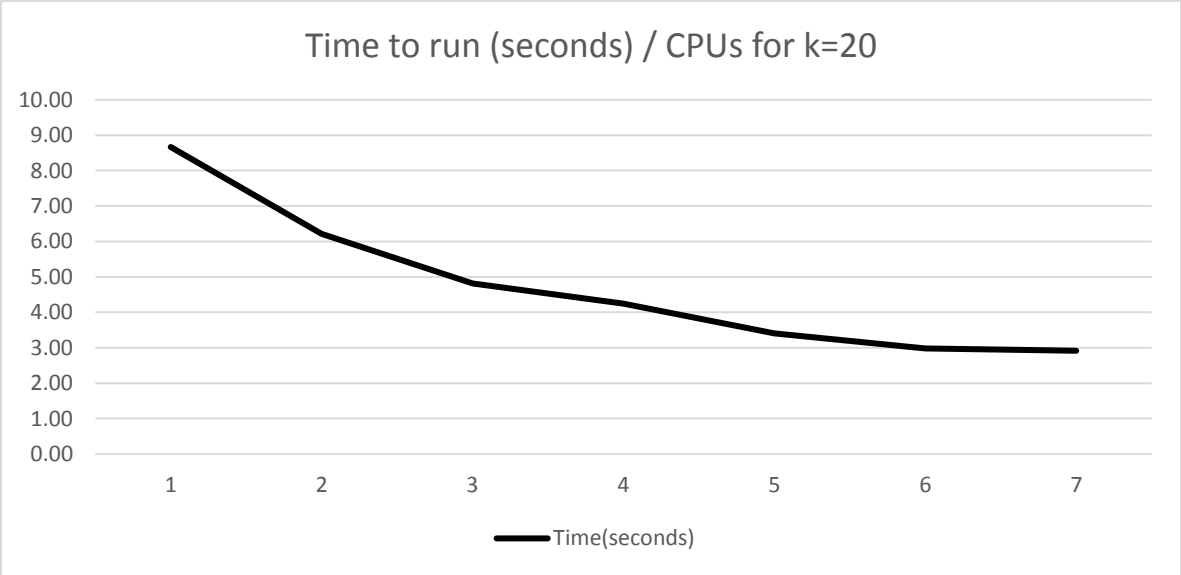
For each partition A_i we perform a binary search to find the index *j* in array *B* which stores the largest element (in B) that is smaller than the largest element in A_i. *j* is the upper bound of the corresponding partition B_i. The lower bound is equal to the upper bound of $B_{i-1} + 1$. The lower bound of $r_{start,p}$ for all *p* > 0 is received from process *p*-1 (upper bound +1 of $r_{end,p-1}$).

Every partition is then merged by its process and added to a list in an ascending order. The merged lists are gathered by process 0, resulting in a sorted list of lists.
The final product is tested for correctness and written to a file by process 0.

2. Results:

We benchmarked the parallel section of the program, neglecting the array generation and output. The following benchmarks are based on merging arrays *A* and *B* of size 2^{20} each.

Due to the large size of the output files we are only attaching the logs of Orca's sqsub (appendix 2) and a print screen of a sample output. The full output files are available in `/scratch/otwluq1/a2/.`



Data size (n)	Number of CPUs	Time (seconds)
2^{20}	2	8.67
	3	6.21
	4	4.82
	5	4.25
	6	3.41
	7	2.98
	8	2.92

Appendix 1 : Program code

```
"""
-------------------------------------------------------------------------------
assingment2.py
A solution in python for CP431/CP631 assignment 2
The program generates 2 n-size  arrays where n is the power of input k in base 2
It then merges the two files using parallel merge and writes the output to a file
Tested for up to k=22 -> two arrays with 4,194,304 32 bit integers each
-------------------------------------------------------------------------------
Authors: Elizabeth Gorbonos, Omer Tal, Tianran Wang
-------------------------------------------------------------------------------
"""

import sys
import datetime
import math
import os
import random
import numpy as np
from mpi4py import MPI

def binary_search(array,i,j,value):
    """
    -------------------------------------------------------------------------------
    Performing a binary search for the biggest item smaller then value on array
    -------------------------------------------------------------------------------
    Preconditions:
        array - a list of data
        i - start index for the search
        j - end index for the search
        value - the item to find
    Postconditions:
        returning the index of the closest item to value, smaller or equal
    -------------------------------------------------------------------------------
    """
    # Stop condition for when the range is up to 2 values
    if (j-i<=1):
        # If the value is lower than the minimum value, return the index before it
        if (value<array[i]):
            return i-1;
        # If the value is higher than the small value but lower than the higher
        elif(value<=array[j]):
            return i;
        # The value is higher than the two indexes
        else:
            return j;
    # Comparing to the item found in the middle of the range
    k=i+math.floor((j-i)/2)
    if (value<=array[k]):
        # Recursion call for the lower half of the range
        return binary_search(array,i,k,value)
    else:
        # Recursion call for the higher half of the range
        return binary_search(array,k+1,j,value)
```

```
def get_input(rank,comm,n):
    """
    ------------------------------------------------------
    Generating two lists of random n numbers
    ------------------------------------------------------
    Preconditions:
       rank - the id of the current processsor
       comm - MPI communicator instance
    Postconditinos:
       a - sorted list of random items in size n
       b - sorted list of random items in size n
    ------------------------------------------------------
    """
    # Maximum number to generate
    LIM=3999999999
    # Generate the two lists
    a = generate_randoms(n,LIM)
    b = generate_randoms(n,LIM)

    return a,b

def generate_randoms(n,lim):
        """
        ------------------------------------------------------------
        Generating a list of n random numbers between 0 and lim
        ------------------------------------------------------------
        Preconditions:
                n - number of items to generate
                lim - highest value to generate
        Postconditions:
                returns a sorted list of n random items between 0-lim
        ------------------------------------------------------------
        """
        a=np.empty(n,dtype=np.uint32)
        increase=int(lim/n)
        last_value=1
    # Generate each new number as a random between the previous_value
    # and a relative increase to ensure a sorted order
        for i in range(n):
                a[i] = random.randint(0,increase) + last_value
                last_value = a[i]
        return a

def merge(a,b,a_start,a_end,b_start,b_end):
    """
    ------------------------------------------------------------------------
    Performing a merge of two local sorted lists
    ------------------------------------------------------------------------
    Preconditions:
       a,b - two sorted lists to be merged
       a_start,b_start - the indexes of a and b to start merging from
       a_end, b_end - the indexes of a and b to stop merging at
    Postconditions:
       merged and sortedlist c in size (a_end-a_start) + (b_end-b_start)
    ------------------------------------------------------------------------
    """
    c = []
    # As long as we haven't reached to the end of the range for either of the lists
    # we continue to merge the smallest item available
    while (a_end>=a_start and b_end>=b_start):
       if (a[a_start]<=b[b_start]):
          c.append(a[a_start])
```

```
        b_start+=1

    return c

def output_file(array,filename):
    """
    ------------------------------------------------------------------------------------------
-----
    Write the two generated lists and the merged list into a given output file name
    ------------------------------------------------------------------------------------------
-----
    Preconditions:
        array - merged list, given as an array of arrays of arrays
        (first array - each process, second array - each partition, third array -
values)
        filename - the output file directory
    Postconditions:
        List in array are written to the file named filename
    ------------------------------------------------------------------------------------------
-----
    """
    # Wrtie to fle with  a buffer of 20MB
    file = open(filename,'w',20000)
    file.write("Merged list:")
    # Iterator over all processes
    for process in range(len(array)):
        # Iterate over all partitions
        for partition in range(len(array[process])):
            # Iterate over items
            for item in range(len(array[process][partition])):
                if (process==0 and partition==0 and item==0):
                    prefix=""
                else:
                    prefix=","
                file.write("{}{}".format(prefix,array[process][partition][item]))
    file.close()
    print("Sucessfully wrote results to file {}".format(filename))

def test(a,b,c):
    """
    ----------------------------------------------------------------------
    Testing the function by comparing lists a and b to the merged list
    ----------------------------------------------------------------------
    """
    index_a = 0
    index_b = 0
    count_c = 0
    for arr_process in c:
        for arr_partition in arr_process:
            for item in arr_partition:
                if (index_a<len(a) and a[index_a] == item):
                    index_a += 1
                elif (index_b<len(b) and b[index_b] == item):
                    index_b +=1
                else:
                    print("discrepancy found when comparing the two lists to the third. a
size={}, b size={}, c size={}, c_value={}"
                          .format(len(a),len(b),count_c,item))
                    return False
                count_c+=1
    if (index_a == len(a) and index_b==len(b) and count_c==len(a) + len(b)):
        print("Tested and found correct")
```

```
            return True
        else:
            print("Discrepancy found when comparing the lists length. a size={}, b size={},
c size={}".format(len(a),len(b),count_c))
            return False

def main():
    # Get process size and rank
    comm = MPI.COMM_WORLD
    size = comm.Get_size()
    rank = comm.Get_rank()
    # Get the number k representing a power of 2 from the user
    if (len(sys.argv)<2 or not sys.argv[1].isdigit()):
        print("Please supply the program with the number of items")
        exit(1)
    k = int(sys.argv[1])
    n = 2**k
    r = int (n/k)
    # When n is not divided by k, increase the number of partitions by 1
    if (n%k>0):
        r+=1
    #Barrier to measure start time afte input
    comm.Barrier()
    if (rank==0):
        # Get the two random arrays
        a,b = get_input(rank,comm,n)
        print("Done with input stage")
        wt = MPI.Wtime()
    else:
        a = np.empty(n,dtype=np.uint32)
        b = np.empty(n,dtype=np.uint32)
    # Broadcasting the two random lists to size-1 processes
    comm.Bcast(a,root=0)
    comm.Bcast(b,root=0)
    # Dividing the partitions between the processes
    if(rank<(r%size)):
        partitions = int(math.floor(r/size))+1
    else:
        partitions = int(math.floor(r/size))

    # Calculate the range dealt by the current process by multiplying the process id
and it's k partition
    partitions_start = int(rank*math.floor(r/size) + min(rank,r%size))
    partitions_end = partitions_start+partitions-1

    #print("Process {} has {} partitions: {}-
{}".format(rank,partitions,partitions_start,partitions_end))

    a_start = []
    a_end = []
    b_start = []
    b_end = []

    # For each of the partitions, setting the values for the index in a and the last
item to partition in b
    for index in range(partitions):
        a_start.insert(index,int(partitions_start*k + k*index))
        # For the last partition, the highest item in b is the last
        if (a_start[index]+k<n):
            a_end.insert(index,int(a_start[index] + k-1))
            # Finding the local upper limit in list b, by using binary search to find the
minimum value bigger then the maximum local value in a
```

```
            index_b = binary_search(b,0,len(b)-1,a[a_end[index]])
            b_end.insert(index,index_b)
         else:
            a_end.insert(index,n-1)
            b_end.insert(index,n-1)
         # The index to start iterating over b is 1 after the last index of the previous
partition
         if (index>0):
                     b_start.insert(index,b_end[index-1]+1)

      # Sending the process last found b_end to be the neighbor's first b_start, for all
but the last process
      if (rank<size-1):
         comm.send(b_end[partitions-1]+1,dest=(rank+1),tag=1)

      # Process 0 starts from index 0 in b
      if (rank==0):
         b_start.insert(0,0)
      # All other processes receive the first position from their left neighbor
      else:
         b_start.insert(0,comm.recv(source=(rank-1),tag=1))

      # Array of merged lists
      merged_lists=[]

      # Merging each partition using a loop in ascending order
      for index in range(partitions):
         # Merging lists a and b in relevant positions into a local list c

merged_lists.append(merge(a,b,a_start[index],a_end[index],b_start[index],b_end[index])
)

      # Process 0 gathers all merged lists into a new list merged, ordered by ranking
index
      merged = comm.gather(merged_lists,root=0)

      # Process 0 now has the merged list and needs to write it to input
      if (rank==0):
         # Print the total time the program took before writing to output
         print("Total time to compute: {:2f} seconds".format(MPI.Wtime() - wt))
         # Testing if the merged list is equal to the merged input lists serially
         test(a,b,merged)
         # Writing to output file
         output_file(merged,"/scratch/otwluq1/a2/output_{}.txt".format(os.getpid()))


main()
```

Appendix 2: Program logs

```
2 Processor:

Done with input stage
Total time to compute: 8.667797 seconds
Tested and found correct
Sucessfully wrote results to file /scratch/otwluq1/a2/output_12098.txt
--- SharcNET Job Epilogue ---
             job id: 10927619
         exit status: 0
            cpu time: 50s / 2.0h (0 %)
        elapsed time: 33s / 1.0h (0 %)
      virtual memory: 784.9M / 1.0G (76 %)

Job completed successfully

--------------------------------------------------------------------------------
3 Processor:

Done with input stage
Total time to compute: 6.214778 seconds
Tested and found correct
Sucessfully wrote results to file /scratch/otwluq1/a2/output_12165.txt
--- SharcNET Job Epilogue ---
             job id: 10927620
         exit status: 0
            cpu time: 64s / 3.0h (0 %)
        elapsed time: 32s / 1.0h (0 %)
      virtual memory: 263.3M / 1.0G (25 %)

Job completed successfully

--------------------------------------------------------------------------------
4 Processor:

Done with input stage
Total time to compute: 4.816678 seconds
Tested and found correct
Sucessfully wrote results to file /scratch/otwluq1/a2/output_11544.txt
--- SharcNET Job Epilogue ---
             job id: 10927621
         exit status: 0
            cpu time: 79s / 4.0h (0 %)
        elapsed time: 32s / 1.0h (0 %)
      virtual memory: 401.4M / 1.0G (39 %)

Job completed successfully

--------------------------------------------------------------------------------
5 Processor:

Done with input stage
Total time to compute: 4.247634 seconds
Tested and found correct
Sucessfully wrote results to file /scratch/otwluq1/a2/output_9748.txt
--- SharcNET Job Epilogue ---
             job id: 10927622
         exit status: 0
            cpu time: 104s / 5.0h (0 %)
        elapsed time: 31s / 1.0h (0 %)
```

```
      virtual memory: 520.8M / 1.0G (50 %)

Job completed successfully

--------------------------------------------------------------------------------
6 Processor:
Done with input stage
Total time to compute: 3.405965 seconds
Tested and found correct
Sucessfully wrote results to file /scratch/otwluq1/a2/output_27651.txt
--- SharcNET Job Epilogue ---
              job id: 10927623
         exit status: 0
            cpu time: 105s / 6.0h (0 %)
        elapsed time: 24s / 1.0h (0 %)

Job completed successfully


--------------------------------------------------------------------------------
7 Processor:

Done with input stage
Total time to compute: 2.980946 seconds
Tested and found correct
Sucessfully wrote results to file /scratch/otwluq1/a2/output_27727.txt
--- SharcNET Job Epilogue ---
              job id: 10927624
         exit status: 0
            cpu time: 121s / 7.0h (0 %)
        elapsed time: 28s / 1.0h (0 %)
      virtual memory: 613.4M / 1.0G (59 %)

Job completed successfully


--------------------------------------------------------------------------------
8 Processor:

Done with input stage
Total time to compute: 2.916491 seconds
Tested and found correct
Sucessfully wrote results to file /scratch/otwluq1/a2/output_27803.txt
--- SharcNET Job Epilogue ---
              job id: 10927625
         exit status: 0
            cpu time: 136s / 8.0h (0 %)
        elapsed time: 27s / 1.0h (0 %)
      virtual memory: 117.2M / 1.0G (11 %)

Job completed successfully
```

Output File Sample:

```
Merged
list:802,950,1012,3295,3430,4026,6657,7278,7891,10220,11359,11427,12965,1500
7,16044,16175,16789,18775,19774,19950,22158,22696,22951,23840,25272,25463,26
649,27313,27481,30381,30810,32423,33121,33669,34945,35002,35638,36621,38767,
39054,40158,41042,41404,41858,44120,45388,46622,48725,49932,50497,51247,5138
2,52547,52653,53323,53507,54911,55025,57623,58564,58713,60287,60590,61063,61
970,63203,64322,64374,64979,65016,66697,67909,68899,70219,72064,72555,72965,
73927,74114,77487,77514,79448,81103,82368,82939,82958,83078,84027,85199,8570
3,87684,88147,88683,88774,90869,90983,91295,92812,93939,94035,97167,97429,98
142,98233,98735,101301,101535,102170,103712,103731,106961,107126,107187,1074
28,109393,110753,111013,111577,114148,114382,114814,116542,116728,116730,118
864,119019,119872,120255,121903,122446,123211,124200,125326,126204,126366,12
7587,127667,128018,129509,131232,132822,133521,134442,137261,137856,138505,1
38844,139124,140151,141322,142720,143290,145254,146441,147870,148596,151281,
151399,154466,154852,156312,156415,158042,158344,160275,160710,162185,163767
,163854,164400,165129,166319,167047,167687,168170,168422,169701,170446,17253
8,172664,172723,173339,173761,174940,175118,177701,177769,179412,179505,1830
72,183190,185233,185971,186225,187542,188320,188424,191238,191414,192289,193
335,194709,195905,196638,197052,198268,199145,199848,200081,201727,202186,20
3668,204679,206324,208304,208329,208959,211463,212271,213845,214831,215124,2
```