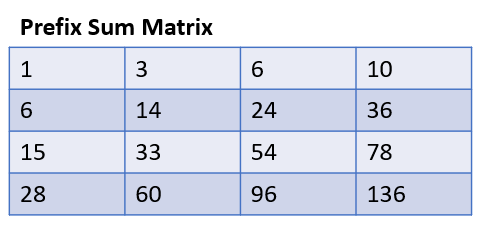
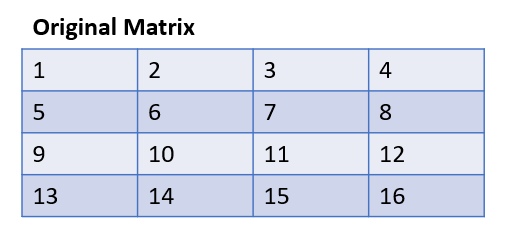
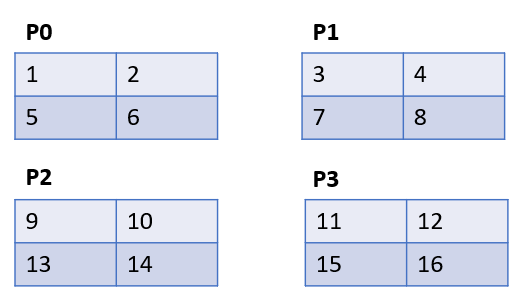
Homework 4

Due: Apr 29th @ 11:59pm (as a typed, pdf file)

1. (20 points) **MPI:** The prefix sum of a particular cell in a NxN matrix is the sum of its value and the values of all cells above and/or to the left of it. An example of a matrix and its prefix sum matrix is as follows:



We want to compute the prefix sum of any such square matrix in parallel using MPI. If we divide the NxN matrix amongst P^2 processes such that each process owns a (N/P)x(N/P) submatrix, the processes can be arranged in a PxP 2D topology to represent the original matrix. For example, we can divide the above 4x4 matrix amongst P^2 = 4 processes arranged in a 2x2 grid:



Given such an arrangement, complete the following MPI program to compute the prefix sum of the original matrix using collective operations within sub-communicators (i.e. no point-to-point communication). You **do not** need to reconstruct the original matrix on a single process afterwards, it is enough to compute the submatrices only. In other words, it is okay to leave the above prefix sum matrix divided amongst the processes.

MPI\_Comm\_size(MPI\_COMM\_WORLD, &num\_pes);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &my\_rank);

int dim\_size[2] = {(int)sqrt(num\_pes), (int)sqrt(num\_pes)};

int periodic[2] = {0, 0};

int my\_coords[2];

MPI\_Comm comm2D;

MPI\_Cart\_create(MPI\_COMM\_WORLD, 2, dim\_size, periodic, 0, &comm2D);

MPI\_Cart\_coords(comm2D, my\_rank, 2, my\_coords);

int my\_row = my\_coords[0], my\_col = my\_coords[1];

MPI\_Comm commR, commC;

MPI\_Comm\_split(comm2D, my\_row, my\_col, &commR);

MPI\_Comm\_split(comm2D, my\_col, my\_row, &commC);

int N = ...;

int\* my\_matrix = (int\*)malloc(N\*N\*sizeof(int));

initializeMatrix(my\_matrix, N);

**/\*\***

**\* a. compute local prefix sum serially**

**\*/**

int\* col\_sums = (int\*)malloc(N\*sizeof(int));

int\* row\_sums = (int\*)malloc(N\*sizeof(int));

int\* prefix\_col\_sums = (int\*)calloc(N, sizeof(int));

int\* prefix\_row\_sums = (int\*)calloc(N, sizeof(int));

for (int i = 0; i < N; i++)

col\_sums[i] = my\_matrix[(N-1)\*N+i];

**/\*\***

**\* b. communicate local column sums to other processes in same grid column using**

**\* a collective operation.**

**\*/**

**/\*\***

**\* c. update local matrix with received column data**

**\*/**

for (int i = 0; i < N; i++)

row\_sums[i] = my\_matrix[i\*N+N-1];

**/\*\***

**\* d. communicate local row sums to other processes in same grid row using a**

**\* collective operation.**

**\*/**

**/\*\***

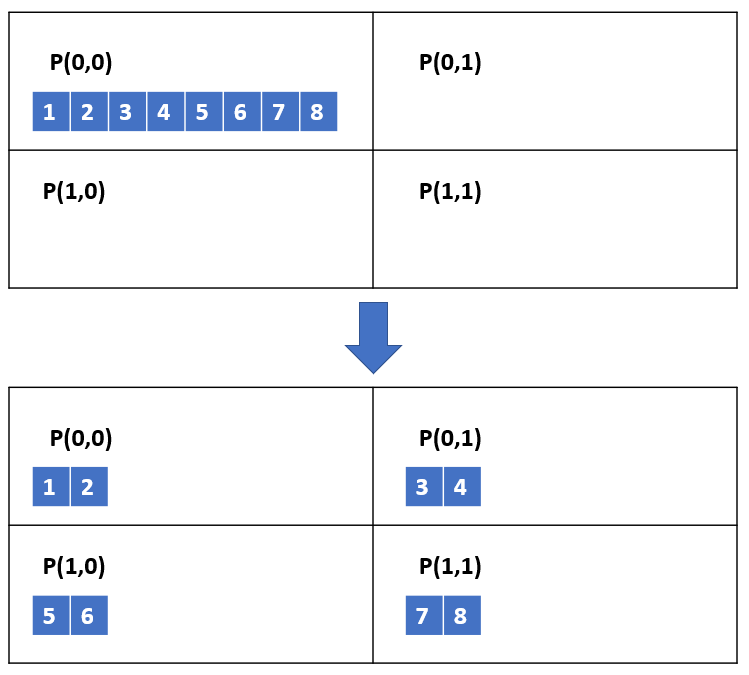
**\* e. update local matrix with received row data**

**\*/**

**Reminder:** you will **not** receive full credit if you use point-to-point communication in your solution.

2. (15 points) **Cost Model:** Consider the following algorithm for the scatter (e.g. MPI\_Scatter) operation, where NP words of data are being scattered from process 0 to each of the P processes, such that each process ends up with N words of data.

Organize the processors in a 2D grid, so each process has 2 coordinates (you may assume the number of processes P is a square). Process 0 with coords (0,0) sends the portion of the data that needs to be scattered among the processes in row i to process (i,0) via a point-to-point messages. After every process (i,0) in the first column receives its data from (0,0), each then sends data that needs to go to each process in its row via individual point-to-point message. The following images illustrate the results of such a scatter for P=4 and N=2:



1. Write an expression for the completion time of this algorithm using the alpha-beta model learned in class. Assume that a process cannot start sending a second message until the first message has been sent out.
2. Write an expression for the completion time of a simple algorithm where process 0 sends a separate message to each process with its N-word data. As with part a), use the alpha-beta model.
3. Compare the two algorithms: under what condition(s) is the grid-based algorithm preferable to the simple algorithm?

3. (15 points) **One-sided MPI:** Consider P processes each with local arrays containing N <int, int> key-value pairs where the keys are integers from 0 to P-1 and the values are integers. Write an MPI program that sorts all the elements in the processes such that process i contains only values whose key is i (i.e. has all values that had key 0, has all values that had key 1, …, has all values that had key P-1). Each process’ final array should contain only the values of the pairs.

1. Complete the following code to perform this operation.

int myRank, numProcesses;

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &myRank);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &numProcesses);

std::pair<int,int>\* local\_kv\_pairs = ...;

int num\_local\_vals = ...;

int num\_per\_process\_local[numProcesses];

for(int i = 0; i < num\_local\_vals; i++) {

num\_per\_process\_local[local\_kv\_pairs[i].first] += 1;

}

int num\_per\_process\_global[numProcesses];

MPI\_Allreduce(&num\_per\_process\_local, &num\_per\_process\_global, numProcesses,

MPI\_INT, MPI\_ADD, MPI\_COMM\_WORLD);

int num\_from\_processes\_before[numProcesses];

MPI\_Exscan(&num\_per\_process\_local, &num\_from\_processes\_before, numProcesses,

MPI\_INT, MPI\_ADD, MPI\_COMM\_WORLD);

**// a. Allocate an array `final\_data` of the appropriate size and expose it as a**

**// window using MPI\_Win\_create.**

int\* final\_data;

MPI\_Win win;

MPI\_Win\_create(/\* fill in arguments \*/);

**MPI\_Win\_fence(0, win);**

int count[numProcesses];

for (int i = 0; i < N; i++) {

int key = local\_kv\_pairs[i].first;

int value = local\_kv\_pairs[i].second;

**// b. Complete the following Put call to put this value in the right**

**// processor’s window.**

MPI\_Put(/\* fill in arguments \*/);

count[key] += 1;

}

**MPI\_Win\_fence(0, win);**

for (int i = 0; i < num\_per\_process\_global[myRank]; i++)

printf(“%s\n”, tostring(final\_data[i]));

1. How could you improve the efficiency of this code? (Hint: reduce the number of MPI\_Put calls). Just a high level description/approach is fine; no code is needed.
2. What can happen if we remove the two MPI\_Win\_fence() calls (in bold)?

4. (10 points) **Charm++:** Write parts of a short Charm++ program and then answer the question below:

A 1D Chare array A has an entry method called sendData that is shown below:

void A::sendData(int key, int requester) {

value = localLookup(key);

thisProxy[requester].response(value);

}

You should complete the entry methods getAllData(), and response(), partially implemented below. Assume main chare has created an instance of A with many elements. Also assume functions hashProc, calculateKey, and f exist. Make any other reasonable assumptions and ***state them***.

void A::getAllData() {

for (int i = 0; i < N; i++) {

key = calculateKey(thisIndex, i); // you don’t need to write this function

// use a hash-like function (provided to you) hashProc(key) which returns the

// index of chare in A which holds the value for the given key

index = hashProc(key);

// fill in 1-line code to request value via sendData.

}

}

void A::response(int value) { // most of the code to be written is here.

w += f(value); // w is a double variable initialed in the constructor to 0.0;

// When all the responses have been received, contribute w into a sum reduction

// going to Main::calcDone. Assume mainProxy is available to you.

}

(b) Will the response entry method always be the last entry method that a chare executes?