Notes on Parallel Sort

Parallel Computer Architecture and Programming

CMU 15-418/15-618, Spring 2014

Parallel sort API

```
void parallelSort(
    float *data, float *&sortedData,
    int procs, int procId,
    size_t dataSize, size_t &localSize ) {
  // Implement parallel sort algorithm as
  // described in assignment 3 handout.

  localSize = 0;
  return;
}
```

Inputs:

data: Input array (a[n/p])

procs: Number of proces

procId: This process id (i)

dataSize: Aggregate data size (n)

localSize: Size of data on process i (~n/p)

Outputs:

sortedData: Sorted array (sorted)

localSize: Size of sorted data on process I

Important: set localSize to sortedData array size to pass the result checking, 0 to skip.

Parallel sort using MPI

Step 1: Choosing pivots to define buckets

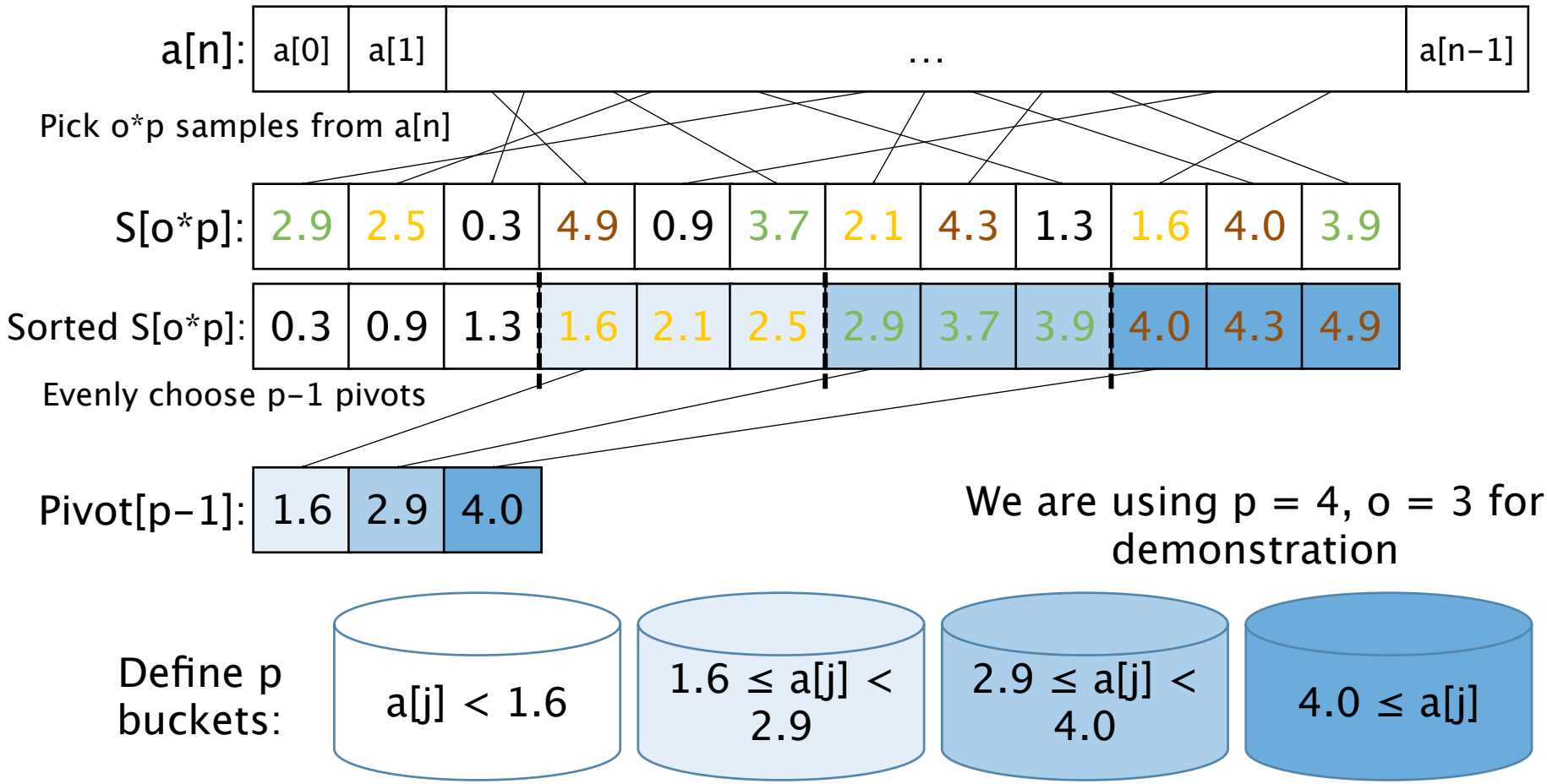
Step 2: Bucketing elements of the input array

Step 3: Redistributing elements

Step 4: Final local sort

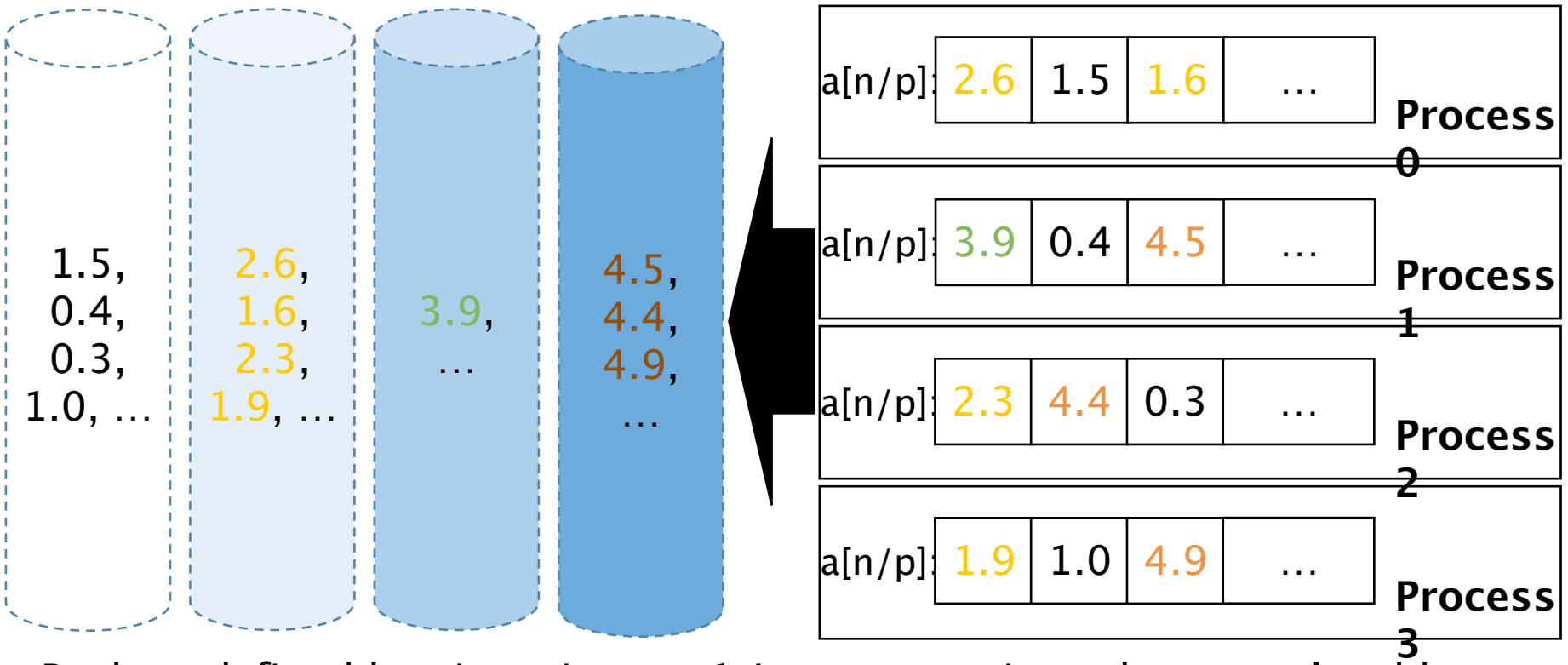
Note: This is only the idea (a sketch) of the algorithm, not it's implementation
Think of how you will implement this with MPI

Step 1: Choosing pivots to define buckets



a[n]: Input array S[o*p]: Sample array o: Oversample n = dataSize p = procs
Tip for o: our reference solution uses o = 12 * lg(n)

Step 2: Bucketing elements of the input array

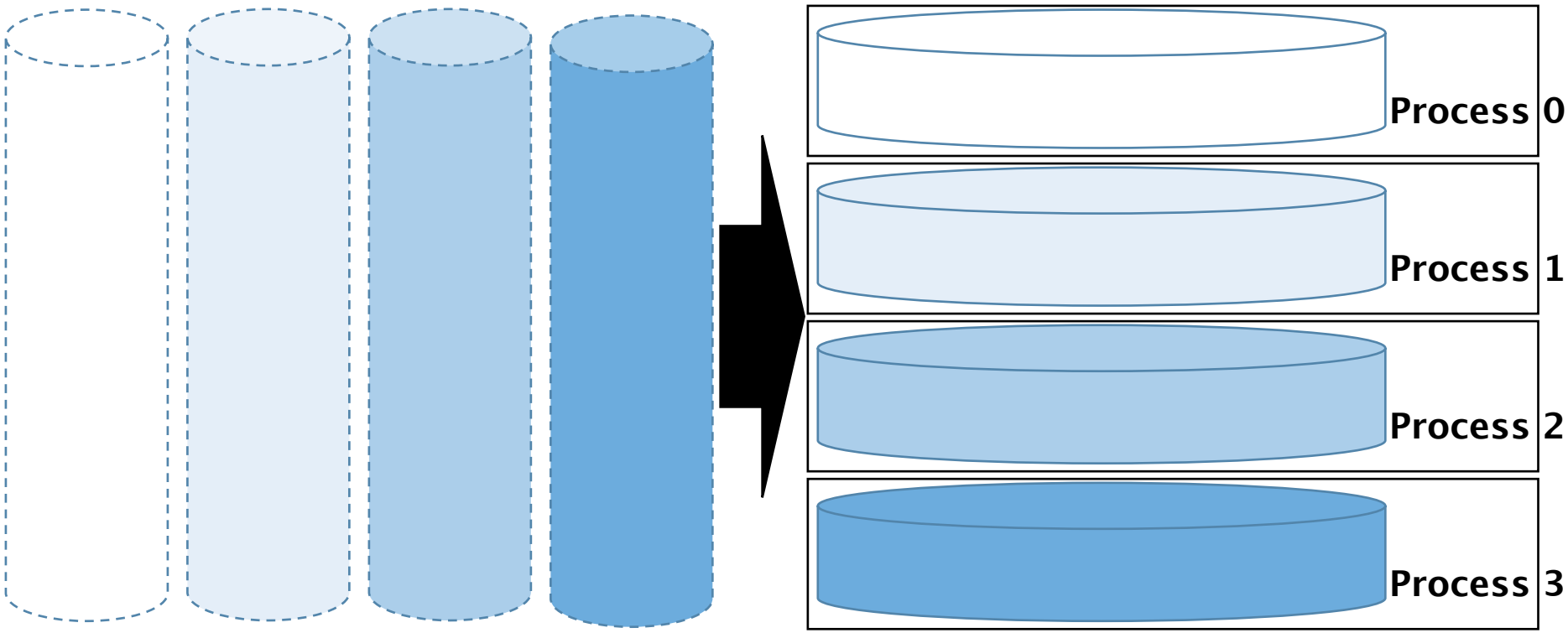


Buckets defined by pivots in step 1

Input arrays in each process's address space

Put all the elements into their corresponding bucket (as defined in step 1)
Note that all processes have to agree on their bucket definition

Step 3: Redistributing elements

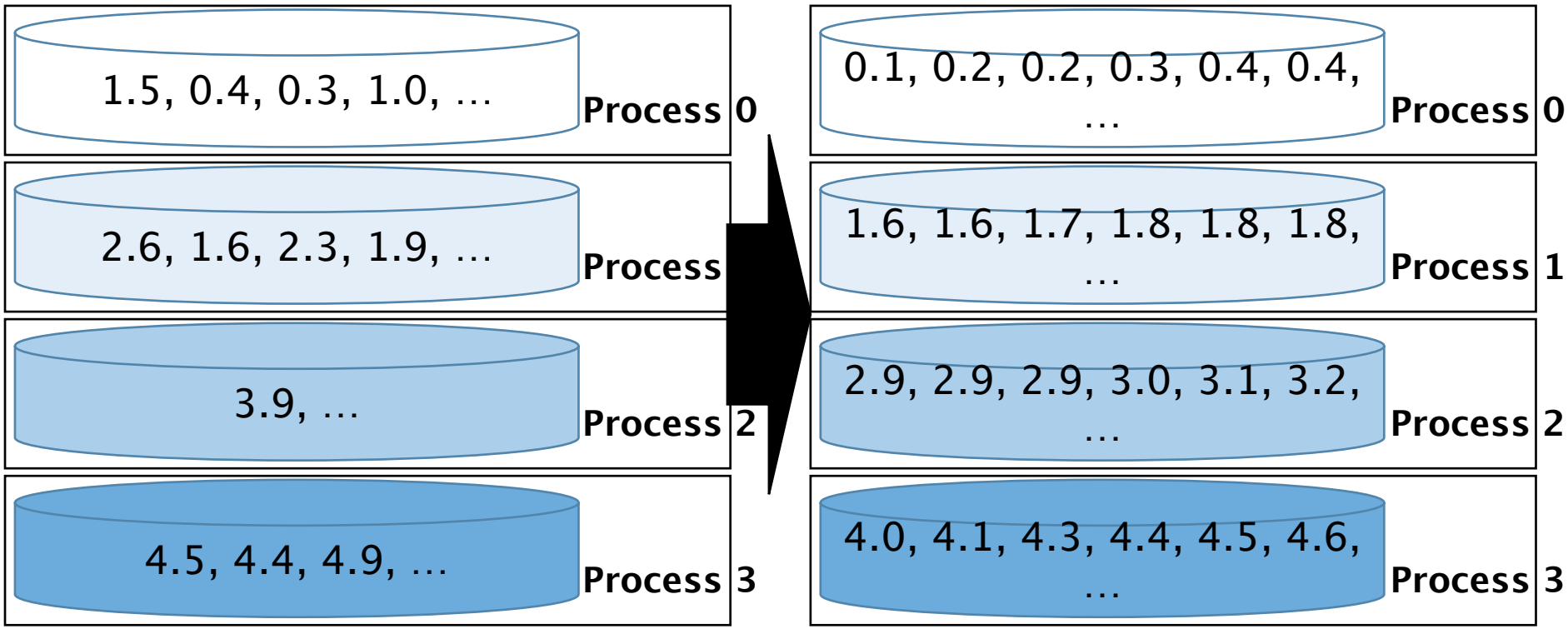


Virtual buckets from step 2

Buckets after redistribution

Redistribute the elements such that elements on each process are now separate,
i.e., elements on process i < elements on process j

Step 4: Final local sort



Unsorted buckets from step 3

Sorted buckets after step 4

Sequentially sort each bucket using a fast sequential sort algorithm
The distributed array is now sorted!

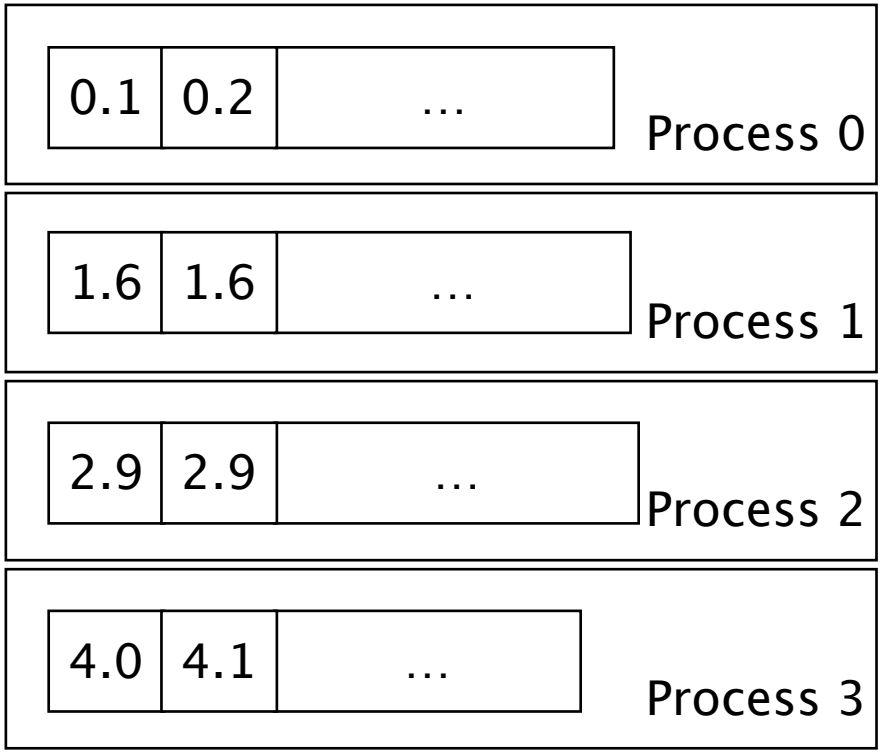
Step 4: Final local sort

Notes for the final step:

Buckets should not overlap so that all elements on process i should be less than elements on process j.

Bucket size on each process can be different, but,

Update localSize to the bucket size on each process!



Sorted buckets from step 4

Tips for parallel sort

Compile and run parallel sort

Makefile and job script

Helper functions

Useful STL functions

General tips

Compile and run parallel sort

Compile parallelSort on a ghc machine

```
[ghc70 starter]$ cd asst3_part1/
[ghc70 asst3_part1]$ make
mkdir -p objs
/usr/lib64/openmpi/bin/mpic++  -O3 -std=c++0x  src/main.cpp -c -o objs/main.o
/usr/lib64/openmpi/bin/mpic++  -O3 -std=c++0x  src/parallelSort.cpp -c -o objs/parallelSort.o
/usr/lib64/openmpi/bin/mpic++  -O3 -std=c++0x  src/dataGen.cpp -c -o objs/dataGen.o
/usr/lib64/openmpi/bin/mpic++  -O3 -std=c++0x  src/stlSort.cpp -c -o objs/stlSort.o
/usr/lib64/openmpi/bin/mpic++ -O3 -std=c++0x  -lpthread -lmpi -lmpi_cxx objs/main.o objs/paral
[ghc70 asst3_part1]$ /usr/lib64/openmpi/bin/mpirun -np 1 parallelSort --help
Usage: parallelSort [options]
  Sort a random or the input dataset

Program Options:
  -s  --size <N>           Size of dataset to sort
  -d  --dist exp|norm|bad1 Distribution to generate dataset
  -p  --par <pram>         Use <pram> as distribution parameter
  -a  --almost <swap>      use <swap> comparisons to generate almost sorted dataset
  -i  --input <file>       Use <file> instead of generated dataset
  -?  --help               This message
```

Run parallelSort on a ghc machine

```
[ghc70 asst3_part1]$ make run
/usr/lib64/openmpi/bin/mpirun -np 4 parallelSort -s 10000000 -d norm -p 5
@@@ Skipping check results for processor 1 because of zero localSize!
@@@ Skipping check results for processor 3 because of zero localSize!
@@@ Skipping check results for processor 2 because of zero localSize!
@@@ Skipping check results for processor 0 because of zero localSize!
Serial sort             took 1.1620s on 1 processors
Parallel merge sort     took 0.4102s on 4 processors    Speedup: 2.8325x
Solution                took 0.0000s on 4 processors    Speedup: infx
```

Compile and run parallel sort

Compile parallelSort on blacklight

```
asst3> scp -r ghc70.ghc.andrew.cmu.edu:~/asst3_part1 ./
yixinluo@ghc70.ghc.andrew.cmu.edu's password:
main.cpp
dataGen.cpp
stlSort.cpp
parallelSort.h
parallelSort.cpp
dataGen.h
stlSort.h
generate_job.sh
example.job
Makefile
asst3> cd asst3_part1/
asst3_part1> module load openmpi/1.6/gnu
asst3_part1> make jobs
mkdir -p objs
mpic++  -O3 -std=c++0x  src/main.cpp -c -o objs/main.o
mpic++  -O3 -std=c++0x  src/parallelSort.cpp -c -o objs/parallelSort.o
mpic++  -O3 -std=c++0x  src/dataGen.cpp -c -o objs/dataGen.o
mpic++  -O3 -std=c++0x  src/stlSort.cpp -c -o objs/stlSort.o
mpic++ -O3 -std=c++0x  -lpthread -lmpi -lmpi_cxx objs/main.o objs/para
cd jobs && ./generate_job.sh 1
cd jobs && ./generate_job.sh 2
cd jobs && ./generate_job.sh 4
cd jobs && ./generate_job.sh 8
cd jobs && ./generate_job.sh 16
cd jobs && ./generate_job.sh 32
cd jobs && ./generate_job.sh 64
cd jobs && ./generate_job.sh 128
```

Compile and run parallel

Submit jobs on blacklight

make jobs creates .job files in jobs/<username>_<cores>.job

Submit job with qsub jobs/<username>_<procs>.job

View job status with qstat -u <username>

Delete job with qdel <jobid>

```
asst3_part1> qsub jobs/yixinluo_128.job
318016.tg-login1.blacklight.psc.teragrid.org
asst3_part1> qsub jobs/yixinluo_2.job
318017.tg-login1.blacklight.psc.teragrid.org
asst3_part1> qstat -u yixinluo

tg-login1.blacklight.psc.teragrid.org:

                                                                  Req'd  Req'd   Elap
Job ID               Username Queue    Jobname    SessID  NDS  TSK  Memory Time  S Time
-------------------- -------- -------- ---------- ------- ---- ---- ------ ----- - -----
318016.tg-login1     yixinluo batch_r1 prog1.job   706802  --   128    --  00:10 R   --
318017.tg-login1     yixinluo batch_r1 prog1.job      --   --    16    --  00:10 R   --

Total cpus requested from running jobs: 144
asst3_part1> qdel 318016
asst3_part1> cat prog1.job.o318017
cp: cannot create regular file `parallelSort': Text file busy
@@@ Skipping check results for processor 1 because of zero localSize!
@@@ Skipping check results for processor 0 because of zero localSize!
Serial sort            took 1.4867s on 1 processors
Parallel merge sort    took 0.7810s on 2 processors    Speedup: 1.9035x
Solution               took 0.0000s on 2 processors    Speedup: infx
```

Please do delete mis-submitted/useless jobs quickly! Especially large ones!

Makefile and job script

You may need to change makefile and job script to test with different parameters

```
[ghc70 asst3_part1]$ /usr/lib64/openmpi/bin/mpirun -np 1 parallelSort --help
Usage: parallelSort [options]
  Sort a random or the input dataset

Program Options:
  -s  --size <N>           Size of dataset to sort
  -d  --dist exp|norm|bad1 Distribution to generate dataset
  -p  --par <pram>         Use <pram> as distribution parameter
  -a  --almost <swap>      use <swap> comparisons to generate almost sorted dataset
  -i  --input <file>       Use <file> instead of generated dataset
  -?  --help               This message
```

e.g., mpirun -np 2 parallelSort -s 10000000 -d norm -p 1
e.g., mpirun -np 4 parallelSort -s 1000000 -d exp -p 5

Tips: test and debug your program with smaller data size, ghc machines usually have little free memory space, which may cause your program to segmentation fault (or you can test if your malloc/new succeeded)
Important! DO NOT run your program on blacklight!

Makefile and job script

```
34 .PHONY: jobs
35
36 # all should come first in the file, so it is the
37 all : parallelSort
38
39 run : parallelSort
40         $(MPIRUN) -np 4 parallelSort -s 10000000 -d norm -p 5
41
42 parallelSort: $(OBJS)
43         $(CXX) $(CXXFLAGS) $(LDFLAGS) $^ -o $@
44
45 jobs: parallelSort
46         cd jobs && ./generate_job.sh 1
47         cd jobs && ./generate_job.sh 2
48         cd jobs && ./generate_job.sh 4
49         cd jobs && ./generate_job.sh 8
50         cd jobs && ./generate_job.sh 16
51         cd jobs && ./generate_job.sh 32
52         cd jobs && ./generate_job.sh 64
53         cd jobs && ./generate_job.sh 128
54
55 $(OBJS): | $(OBJDIR)
56 $(OBJDIR):
57         mkdir -p $@
58
59 $(OBJDIR)/%.o: $(SRCDIR)/%.cpp $(SRCDIR)/*.h Makefile
60         $(CXX) $(CPPFLAGS) $(CXXFLAGS) $< -c -o $@
61
62 clean:
63         rm -rf $(OBJDIR) parallelSort $(TOOLS) jobs/$(USER)_*.job
```

Makefile:
Change this line to whatever argument you want when make run

<- This generates your job files in jobs/ folder as jobs/<username>_<procs>.job

Makefile and job script

```
1  #!/bin/bash
2  #ncpus must be a multiple of 16
3  #PBS -l ncpus=ROUNDCORES
4  #PBS -l pmem=8gb
5  #PBS -l walltime=10:00
6
7  # Merge stdout and stderr into one output file
8  #PBS -j oe
9
10 #PBS -q batch
11
12 # use the name prog1.job
13 #PBS -N prog1.job
14
15 # Load mpi.
16 source /usr/share/modules/init/bash
17 module load openmpi/1.6/gnu
18
19 # Move to my $SCRATCH directory.
20 cd $SCRATCH
21
22 # Set this to the important directory.
23 execdir=PROGDIR
24 exe=parallelSort
25 args="-s 10000000 -d exp -p 5"
26
27 # Copy executable to $SCRATCH.
28 cp $execdir/$exe $exe
29
30 # Run my executable
31 mpirun -np NCORES ./$exe $args
```

<- Important! Add this line to your script

job script: jobs/example.job
<- Change this line to whatever argument you want blacklight to run

Helper functions

```
void printArr(const char* arrName, int *arr, size_t size, int procId);
void printArr(const char* arrName, float *arr, size_t size, int procId);
```

e.g., `printArr("pivot", pivot, procs-1, procId);`

Helps you debug your program, can be easily turned off by

```
#define NO_DEBUG
```

in parallelSort.h

```
void randomSample(float *data, size_t dataSize,
                  float *sample, size_t sampleSize) {
  for (size_t i=0; i<sampleSize; i++) {
    sample[i] = data[rand()%dataSize];
  }
}
```

e.g., `randomSample( data, localSize, sample, 12*log(dataSize) );`

Uniform-randomly pick samples from data and put in sample array

Useful STL functions

std::sort(first, last)
e.g. `sort(data, data + localSize);`
Comments: a very decent sequential sort

std::inplace_merge(first, middle, last)
e.g. `inplace_merge(data, data + 5, data + 10);`
Comments: merge two sorted arrays between
(1) first to middle-1, and
(2) middle to last-1

std::lower_bound(first, middle, val)
e.g. `int bucketId = lower_bound(pivot, pivot+procs-1, data[i]) - pivot;`
Comments: useful to find buckets for each elements

Examples can be found in src/stlSort.cpp
References: http://www.cplusplus.com/

General tips

Start early! You may have to wait days for the results to come back from blacklight, especially close to deadline.

Use small input sizes and printArr to debug your program.

Again, start early!

Challenges

Choose pivots that can divide the workload evenly.

Experiment your code with different inputs we provided: norm, exp, bad1

How to deal with different input patterns?

What are the inputs that can break your sampling scheme?

Thought experiment:

What if the input array is an integer array?

What are the new challenges induced by integer array?