Notes on Parallel Sort

Parallel Computer Architecture and Programming
CMU 15–418/15–618, Spring 2014
Parallel sort API

Inputs:
data: Input array (a[n/p])
procs: Number of processes (p)
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.

```c
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;
}
```
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 its 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 = data Size**

**p = procs**

**Tip for o:** our reference solution uses o = 12 * lg(n)

---

Pick o*p samples from a[n]

**S[o*p]:**

| 2.9 | 2.5 | 0.3 | 4.9 | 0.9 | 3.7 | 2.1 | 4.3 | 1.3 | 1.6 | 4.0 | 3.9 |

**Sorted S[o*p]:**

| 0.3 | 0.9 | 1.3 | 1.6 | 2.1 | 2.5 | 2.9 | 3.7 | 3.9 | 4.0 | 4.3 | 4.9 |

Evenly choose p-1 pivots

**Pivot[p−1]:**

| 1.6 | 2.9 | 4.0 |

Define p buckets:

- a[j] < 1.6
- 1.6 \leq a[j] < 2.9
- 2.9 \leq a[j] < 4.0
- 4.0 \leq a[j]

We are using p = 4, o = 3 for demonstration
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

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**

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!

<table>
<thead>
<tr>
<th></th>
<th>Process 0</th>
<th>Process 1</th>
<th>Process 2</th>
<th>Process 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>0.2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.6</td>
<td>1.6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.9</td>
<td>2.9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.0</td>
<td>4.1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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
[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|badl 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: infinity
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>
```

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

```
<table>
<thead>
<tr>
<th>Job ID</th>
<th>Username</th>
<th>Queue</th>
<th>Jobname</th>
<th>SessID</th>
<th>NDS</th>
<th>TSK</th>
<th>Memory</th>
<th>Time</th>
<th>S Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>318016.tg-login</td>
<td>yixinluo</td>
<td>batch_r1</td>
<td>progl.job</td>
<td>706802</td>
<td>--</td>
<td>128</td>
<td>--</td>
<td>00:10</td>
<td>R</td>
</tr>
<tr>
<td>318017.tg-login</td>
<td>yixinluo</td>
<td>batch_r1</td>
<td>progl.job</td>
<td>--</td>
<td>--</td>
<td>16</td>
<td>--</td>
<td>00:10</td>
<td>R</td>
</tr>
</tbody>
</table>
```

Total cpus requested from running jobs: 144

```
asst3_part1> qdel 318016
asst3_part1> cat progl.job.0318017
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 10000000 -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

```makefile
.PHONY: jobs

# all should come first in the file, so it is the  
all : parallelSort

run : parallelSort
    $(MPIRUN) -np 4 parallelSort -s 10000000 -d norm -p 5

parallelSort: $(OBJ)
    $(CXX) $(CXXFLAGS) $(LDFLAGS) $^ -o $@

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

$(OBJ): | $(OBJDIR)

$(OBJDIR):
    mkdir -p $@

$(OBJDIR)/%.o: $(SRC)/%.cpp $(SRC)/*.h Makefile
    $(CXX) $(CXXFLAGS) $(CXXFLAGS) $< -c -o $@

clean:
    rm -rf $(OBJDIR) parallelSort $(TOOLS) jobs/$(USER) *.job
```

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

```makefile
<-- This generates your job files in jobs/ folder as jobs/<username>_ <procs>.job
```
Makefile and job script

```bash
#!/bin/bash
# ncpus must be a multiple of 16
#PBS -l ncpus=ROUNDCORES
#PBS -l pmem=8gb
#PBS -l walltime=10:00

# Merge stdout and stderr into one output file
#PBS -j oe

#PBS -q batch

# use the name progl1.job
#PBS -N progl1.job

# Load mpi.
source /usr/share/modules/init/bash
module load openmpi/1.6/gnu

# Move to my $SCRATCH directory.
cd $SCRATCH

# Set this to the important directory.
execdir=PROGDIR
exe=parallelSort
args="-s 10000000 -d exp -p 5"

# Copy executable to $SCRATCH.
cp $execdir/$exe $exe

# Run my executable
mpirun -np NCORES ./$exe $args
```

<-- Important! Add this line to your script

<-- Change this line to whatever argument you want blacklight to run

job script: jobs/example.job

Thursday, February 27, 14
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?