Super Computing on the go: MapReduce for mobile GPUs

Super Computing on the go: MapReduce for mobile GPUs

Background

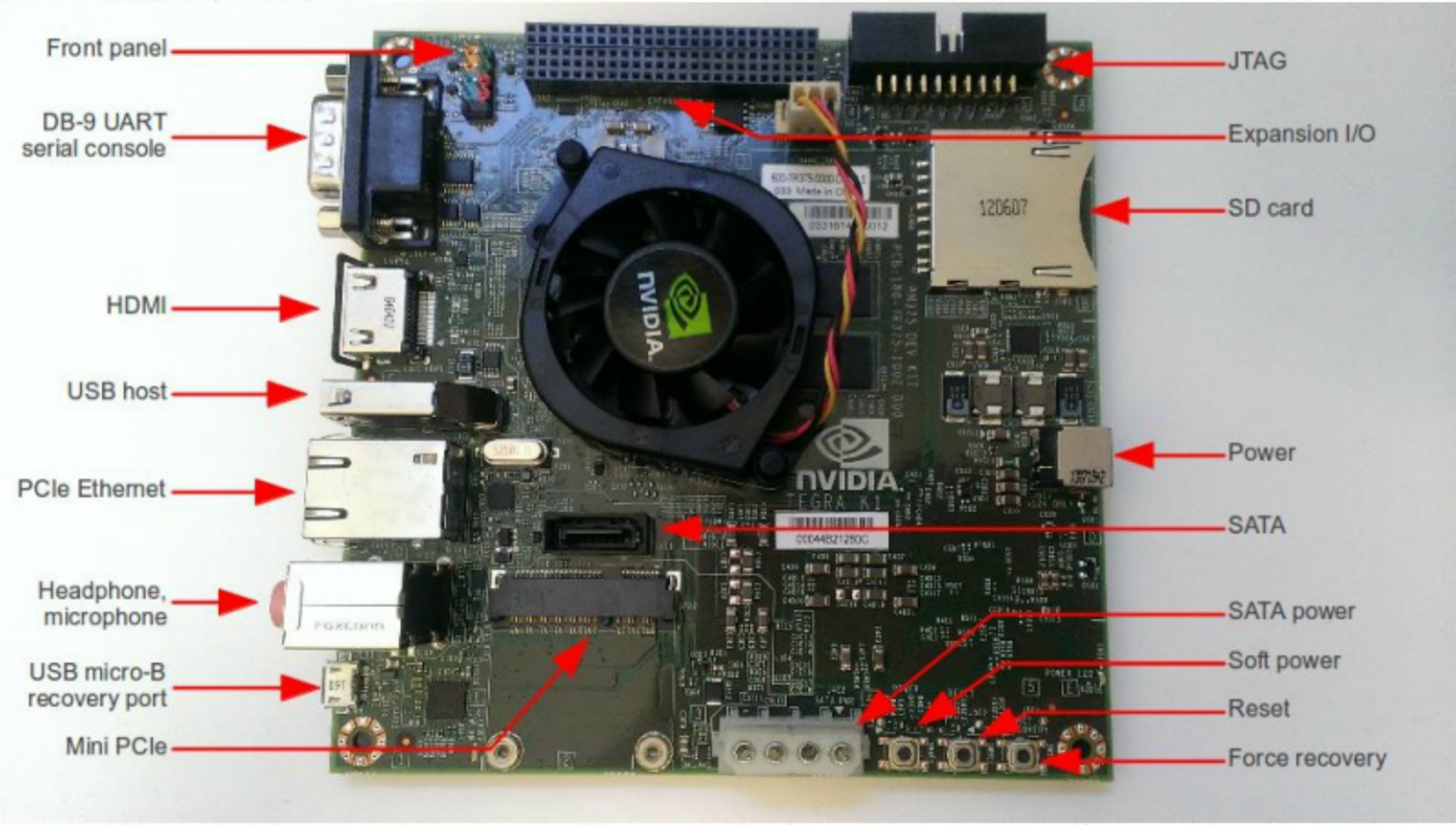
- Data Driven Applications
- MapReduce
- GPGPUs

Jetson TK1

- Tegra K1 - first mobile GPU
- $192 USD
- 192 cores
- Quad-core ARM processor
- unified memory

Jetson TK1

- Tegra K1 - first mobile GPU
- $192 USD
- 192 cores
- Quad-core ARM processor
- unified memory

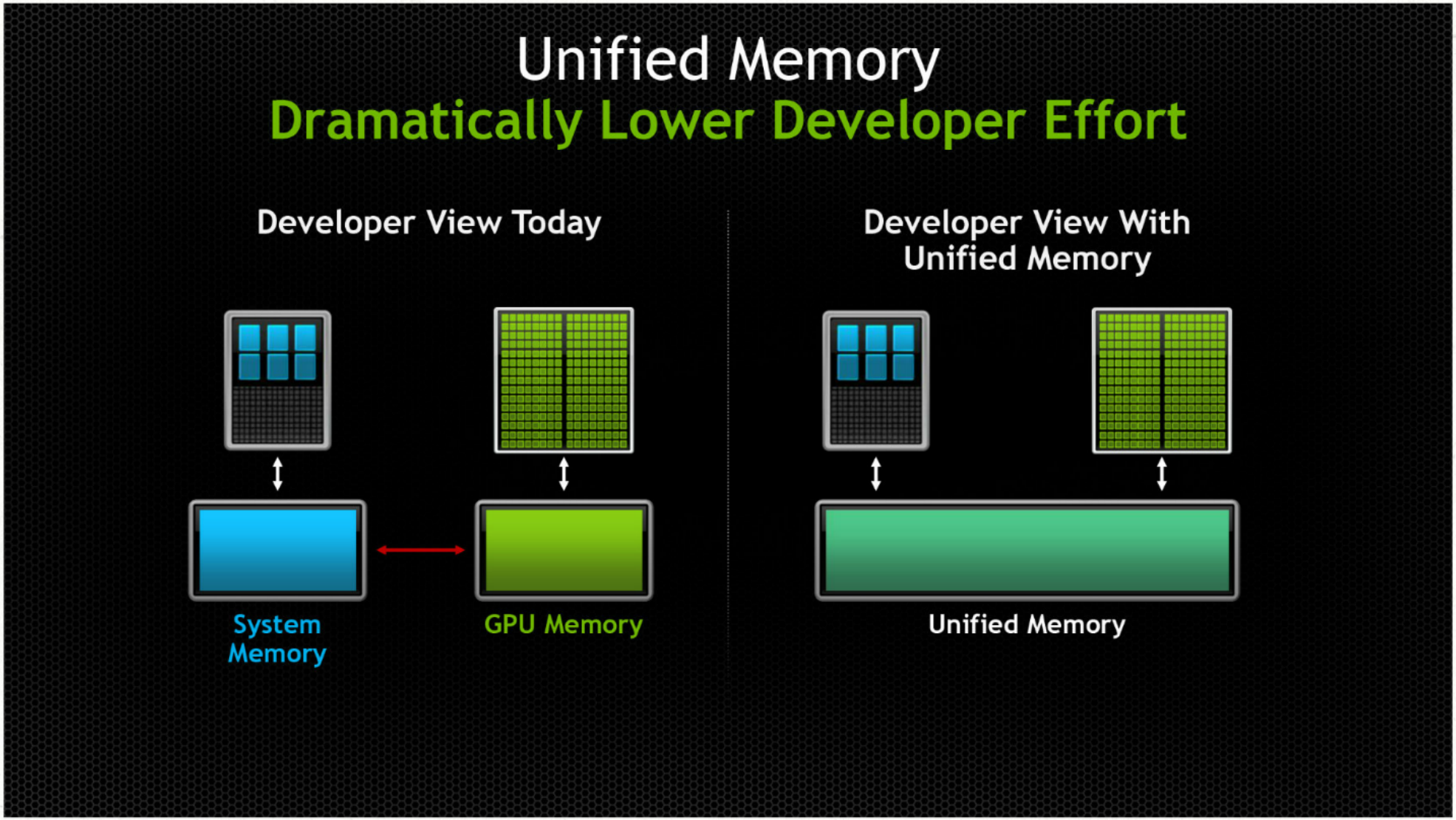

Front panel
DB-9 UART serial console
HDMI
USB host
PCIe Ethernet
Headphone, microphone
USB micro-B recovery port
Mini PCIe
JTAG
Expansion I/O
SD card
Power
SATA
SATA power
Soft power
Reset
Force recovery

Jetson TK1

- Tegra K1 - first mobile GPU
- $192 USD
- 192 cores
- Quad-core ARM processor
- unified memory

Unified Memory

Dramatically Lower Developer Effort



Background

MapReduce for Jetson TK1

Proof of Concepts

- Hadoop on VM
- Hadoop on Jetson TK1-single node
- MARS on Jetson

Significance

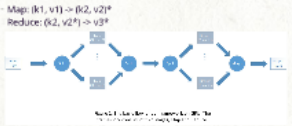
- Mobility
- Power Consumption
- Performance

Features

- Highly parallel computation
- Map Reduce Programming with ease
- Scale-out Architecture
- Mobility
- Super Computing on a power budget
- First of its kind

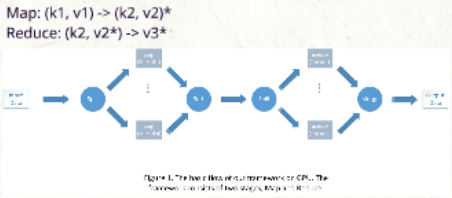
Features

- Highly parallel computation
- Map Reduce Programming with ease
- Scale-out Architecture
- Mobility
- Super Computing on a power budget
- First of its kind



Features

- Highly parallel computation
- Map Reduce Programming with ease
- Scale-out Architecture
- Mobility
- Super Computing on a power budget
- First of its kind



Map: (k1, v1) -> (k2, v2)*

Reduce: (k2, v2*) -> v3*

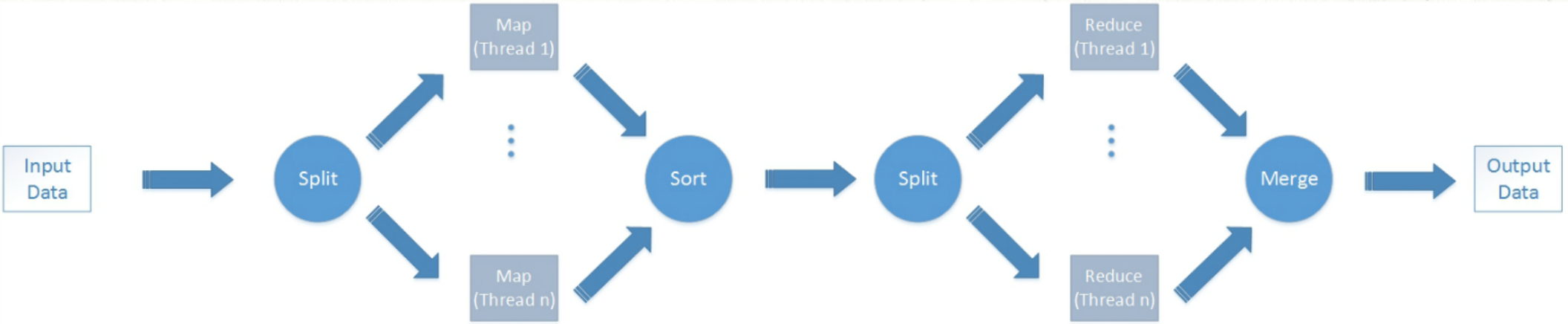


Figure 1. The basic flow of our framework on GPU. The framework consists of two stages, Map and Reduce.

Significance

- Mobility
- Power Consumption
- Performance

Proof of Concepts

- Hadoop on VM
- Hadoop on Jetson TK1-single node
- MARS on Jetson

Literature Review

- Remote Access - modmypi.com/blog
- Big Data Analytics - "World's largest artificial neural networks with GPUs"
- NVIDIA performance primitives
- CUDA Data Parallel Primitives Library
- oscpp a header-only C++11 library for constructing and parsing OpenSoundControl packets
- MPI Cluster System
- OpenCL technology - Intel Developer Zone
- CUDA Zone - NVidia Developer
- History and Evolution of GPU Architecture
- GPU Clusters for High-Performance Computing
- Current and next-generation GPUs for accelerating CT reconstruction: quality, performance, and tuning
- Whitepaper NVIDIA Tegra K1
- GPGPU Processesing in CUDA Architecture
- MapReduce: Simplified Data Processing on Large Clusters
- Hadoop and HDFS: - Cloudera
- Jetson TK1 Embedded Development Kit | NVIDIA
- CUDA 6 Unified Memory explained - Blog - StreamComputing

Literature Review

- Remote Access - modmypi.com/blog
- Big Data Analytics - "World's largest artificial neural networks with GPUs"
- NVIDIA performance primitives
- CUDA Data Parallel Primitives Library
- oscpp a header-only C++11 library for constructing and parsing OpenSoundControl packets
- MPI Cluster System
- OpenCL technology - Intel Developer Zone
- CUDA Zone - NVidia Developer
- History and Evolution of GPU Architecture
- GPU Clusters for High-Performance Computing
- Current and next-generation GPUs for accelerating CT reconstruction: quality, performance, and tuning
- Whitepaper NVIDIA Tegra K1

- MPI Cluster System
- OpenCL technology - Intel Developer Zone
- CUDA Zone - NVidia Developer
- History and Evolution of GPU Architecture
- GPU Clusters for High-Performance Computing
- Current and next-generation GPUs for accelerating CT reconstruction: quality, performance, and tuning
- Whitepaper NVIDIA Tegra K1
- GPGPU Processesing in CUDA Architecture
- MapReduce: Simplified Data Processing on Large Clusters
- Hadoop and HDFS: - Cloudera
- Jetson TK1 Embedded Development Kit | NVIDIA
- CUDA 6 Unified Memory explained - Blog - StreamComputing

Challenges

- Released lately
- Modified version of CUDA
- Linux for Tegra
- Memory Architecture
- Concurrent Reads and Writes
- Efficient Load Balancing Scheme
- Low Synchronization Overhead

- Released lately
- Modified version of CUDA
- Linux for Tegra
- Memory Architecture
- Concurrent Reads and Writes
- Efficient Load Balancing Scheme
- Low Synchronization Overhead

What we've done so far

- Designing - single node
- Coding - single node
- Testing - single node

Designing

GPU

- copy the input data to the GPU
- make Map, Sort, and Reduce kernel calls
- copy the output data back to the CPU
- Each data item is assigned to one GPU thread
- we process many items (a “Chunk”) with a single kernel call

CPU

- Take a set of indivisible work units (items)
- Map them, generating key-value pairs
- Sort these pairs by key and Reduce all like keyed pairs and gather the final output

Optimizations

Motto

focus on reducing communication times at the expense of more computation time

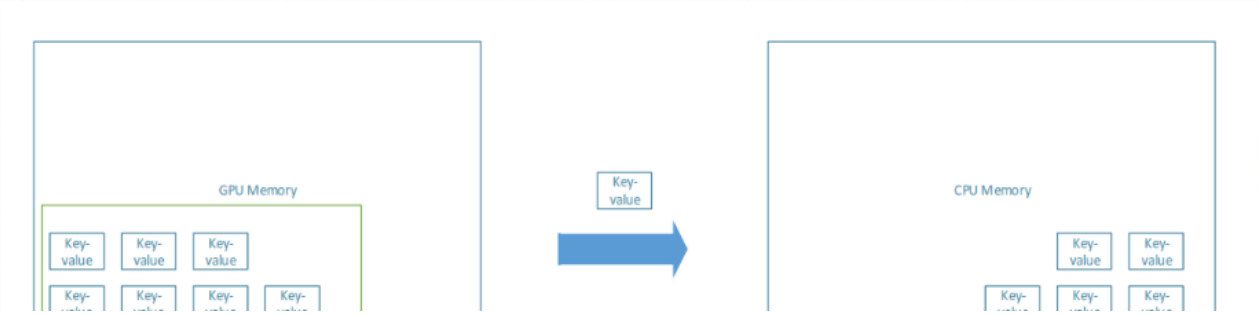
Optimizations

Motto

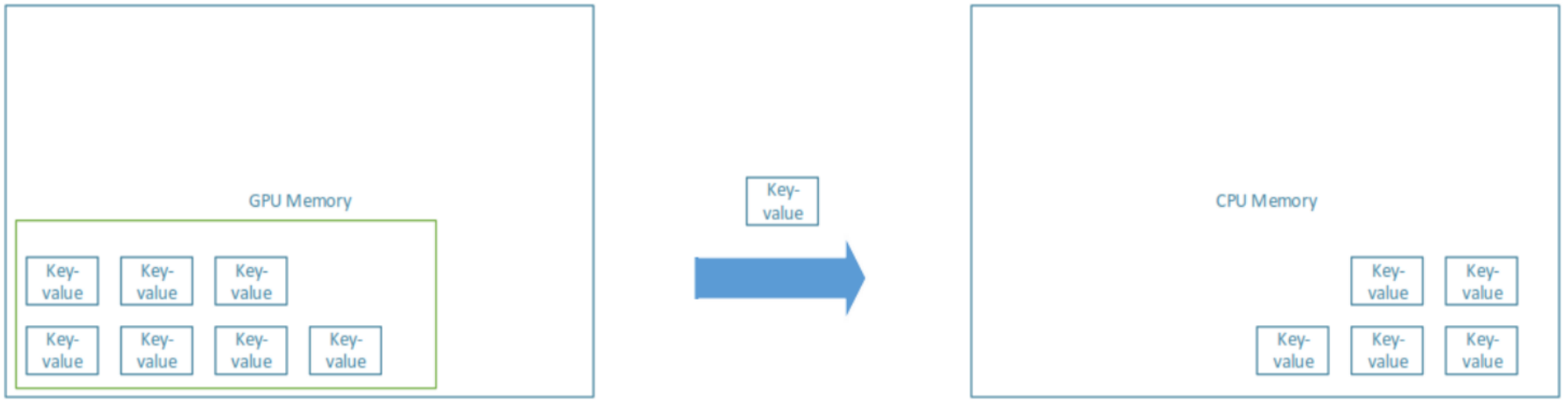
focus on reducing communication times
at the expense of more computation time

- Chunking
- relaxed mapping, more flexibility and more higher-level operations efficient
- Combine - not new, but needed GPU implementation
- Partial Reduction - final key-value set is large
- Accumulation - final key-value set is small

- Chunking
- relaxed mapping, more flexibility and more higher-level operations efficient
- Combine - not new, but needed GPU implementation
- Partial Reduction - final key-value set is large
- Accumulation - final key-value set is small



ccumulation - final key-value set is s



The polymorphic Map

- Map itself
- Accumulation
- Partial Reduction
- Combination
- one chunk at a time

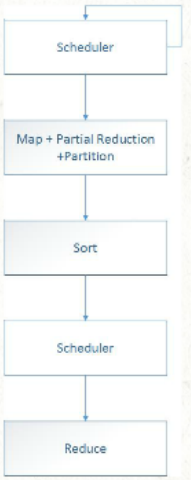
Sort

radix sort from CUDPP (CUDA Data Parallel Primitives Library)

Reduce

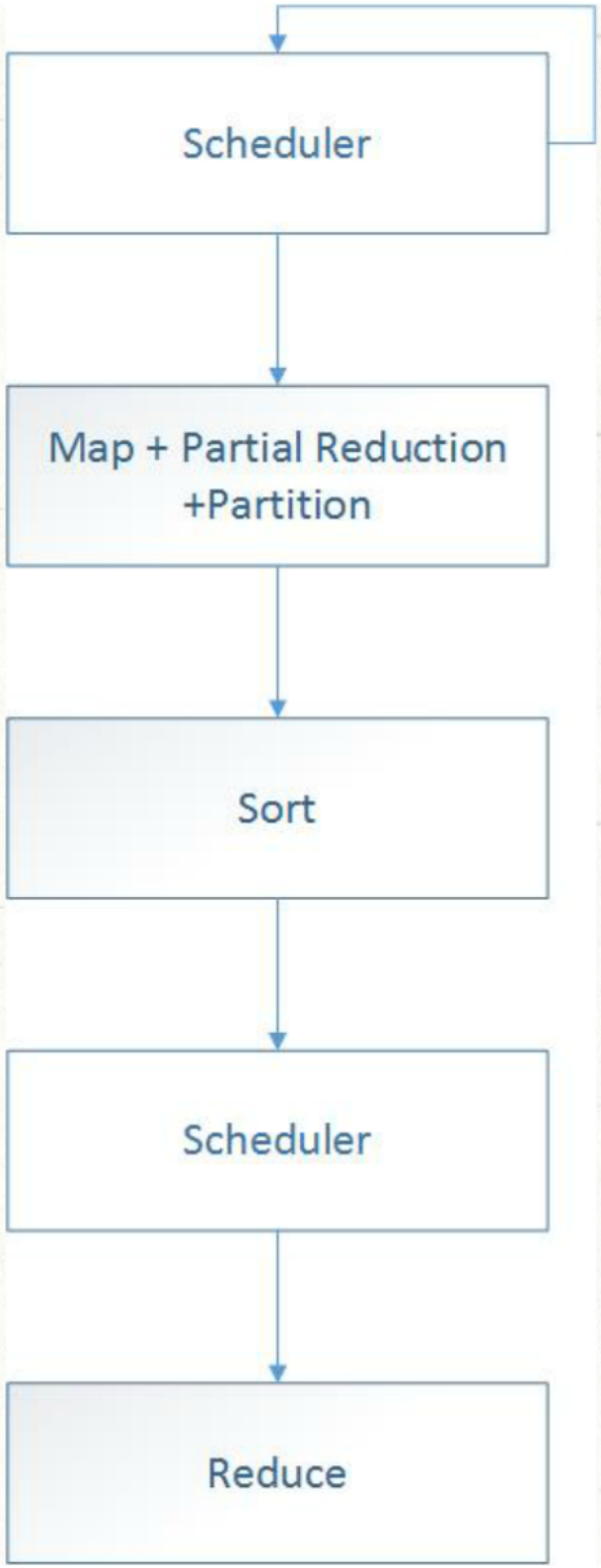
callback to asks how many value sets should be copied for the next reduction

Pipeline Architecture

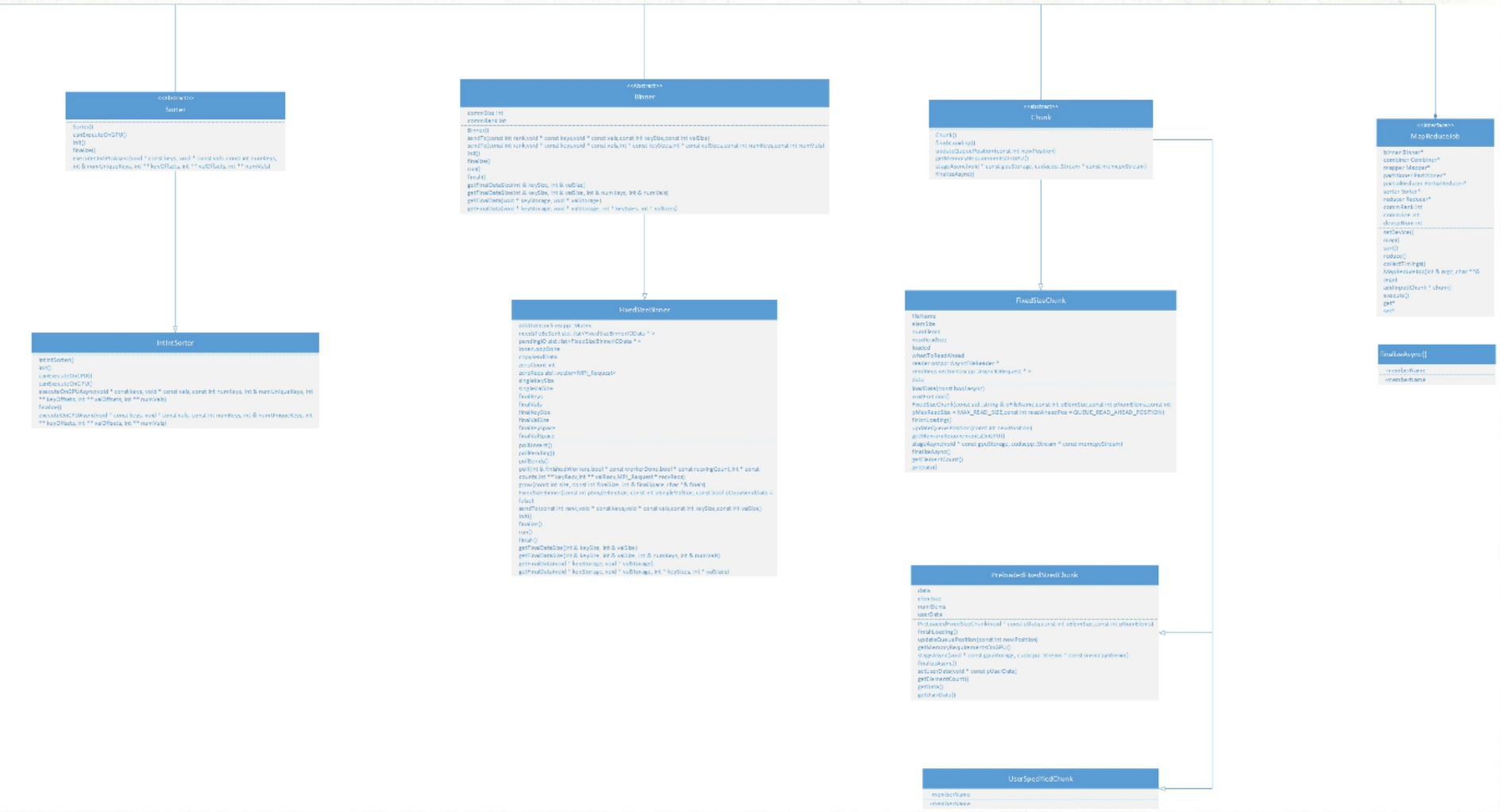


Class diagram

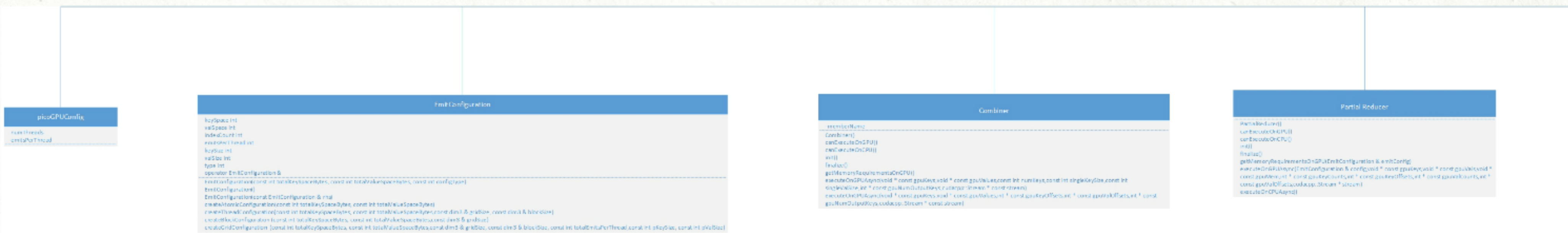
Sequence diagrams


Scheduler
Map + Partial Reduction
+Partition
Sort
Scheduler
Reduce

Class diagram

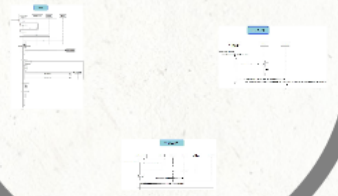

Sorter
IntIntSorter
Binner
HostBinnerBase
HostBinner
Chunk
FixedSizeChunk
PreloadedHostSizedChunk
UserSpecifiedChunk
MapReduceJob
FinalizeAsync[]


Reducer
Mapper
Partitioner
WordCountReducer
WordCountMapper


picoGPUConfig
numThreads
emitsPerThread
EmitConfiguration
keySpace int
valSpace int
indexCount int
keySize int
valSize int
type int
operator EmitConfiguration &
EmitConfiguration()
createAtomicConfiguration(const int totalKeySpaceBytes, const int totalValueSpaceBytes)
Combiner
memberName
Combiner()
canExecuteOnGPU()
canExecuteOnCPU()
init()
finalize()
getMemoryRequirementsOnGPU()
Partial Reducer
PartialReducer()
canExecuteOnGPU()
canExecuteOnCPU()
init()
finalize()
getMemoryRequirementsOnGPU(EmitConfiguration & emitConfig)
executeOnCPUAsync()

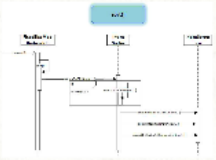
Sequence diagrams

Map

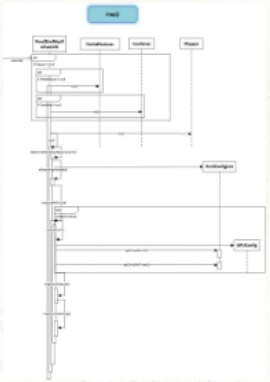


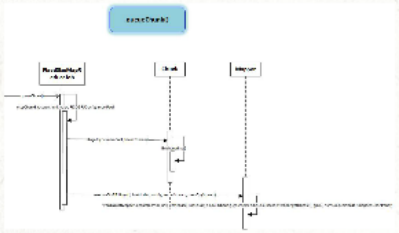
Reduce

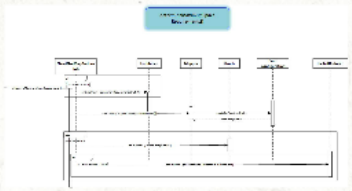
Sort



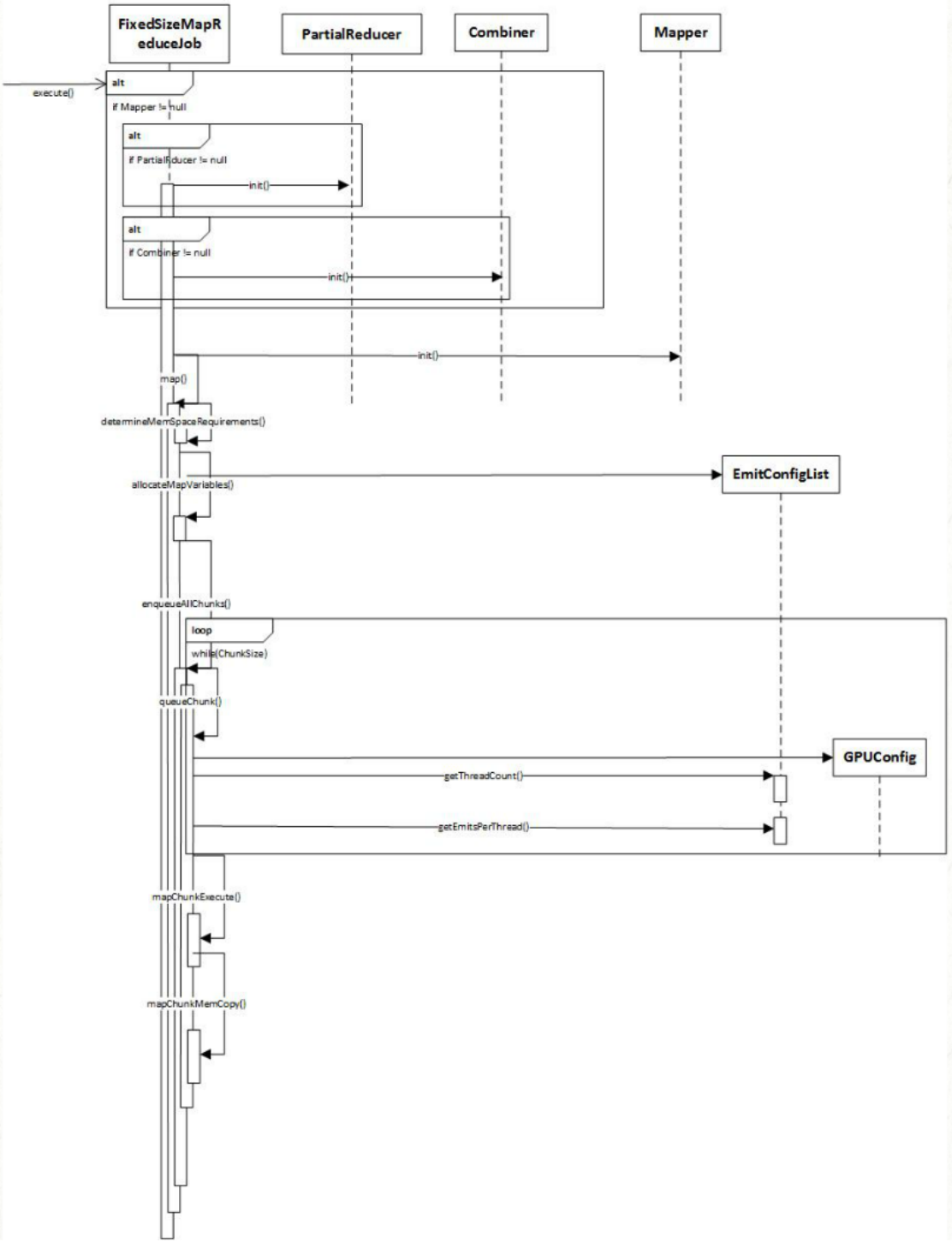
Map

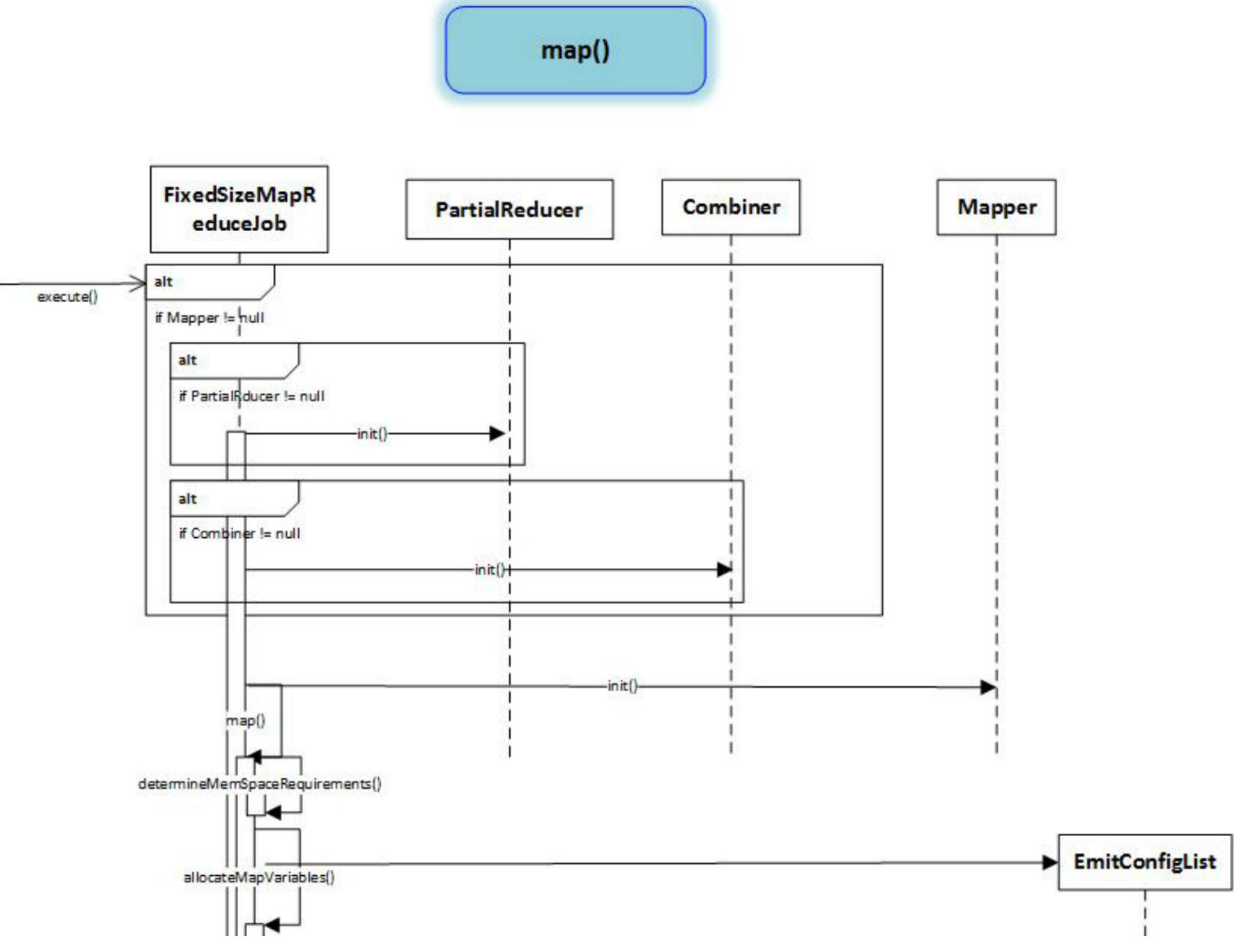






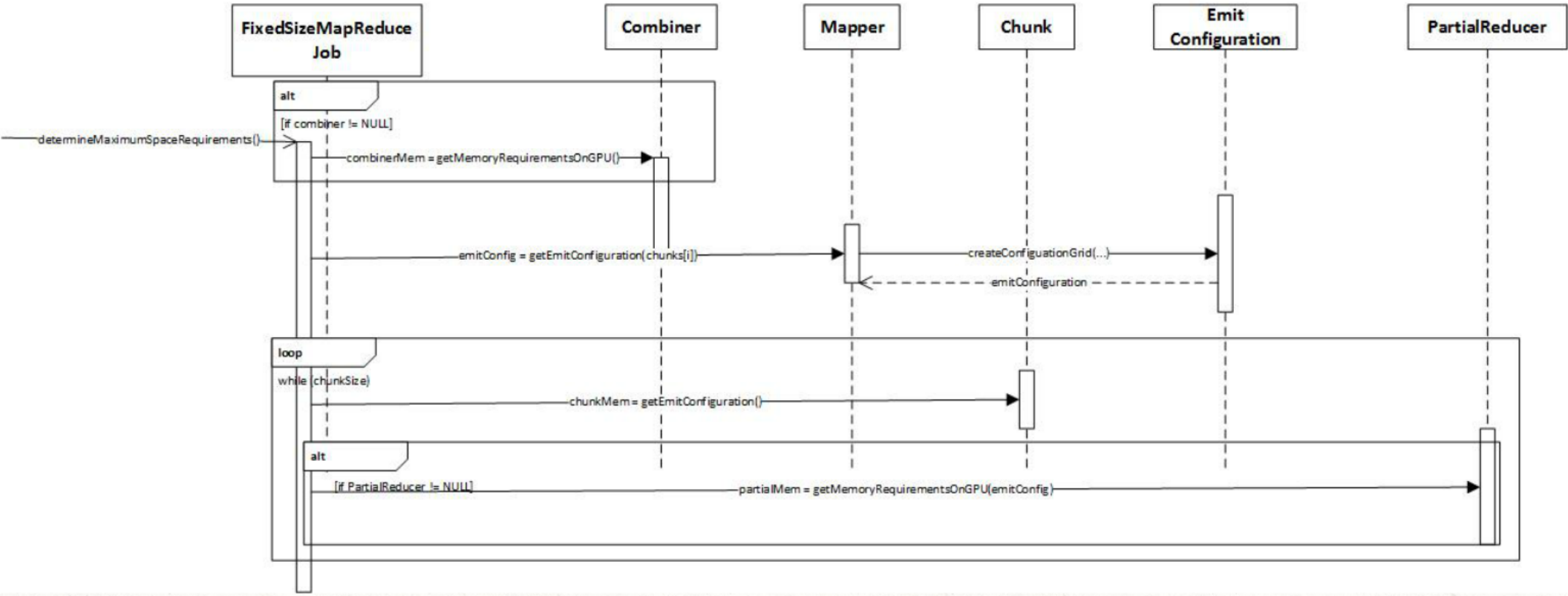
map()

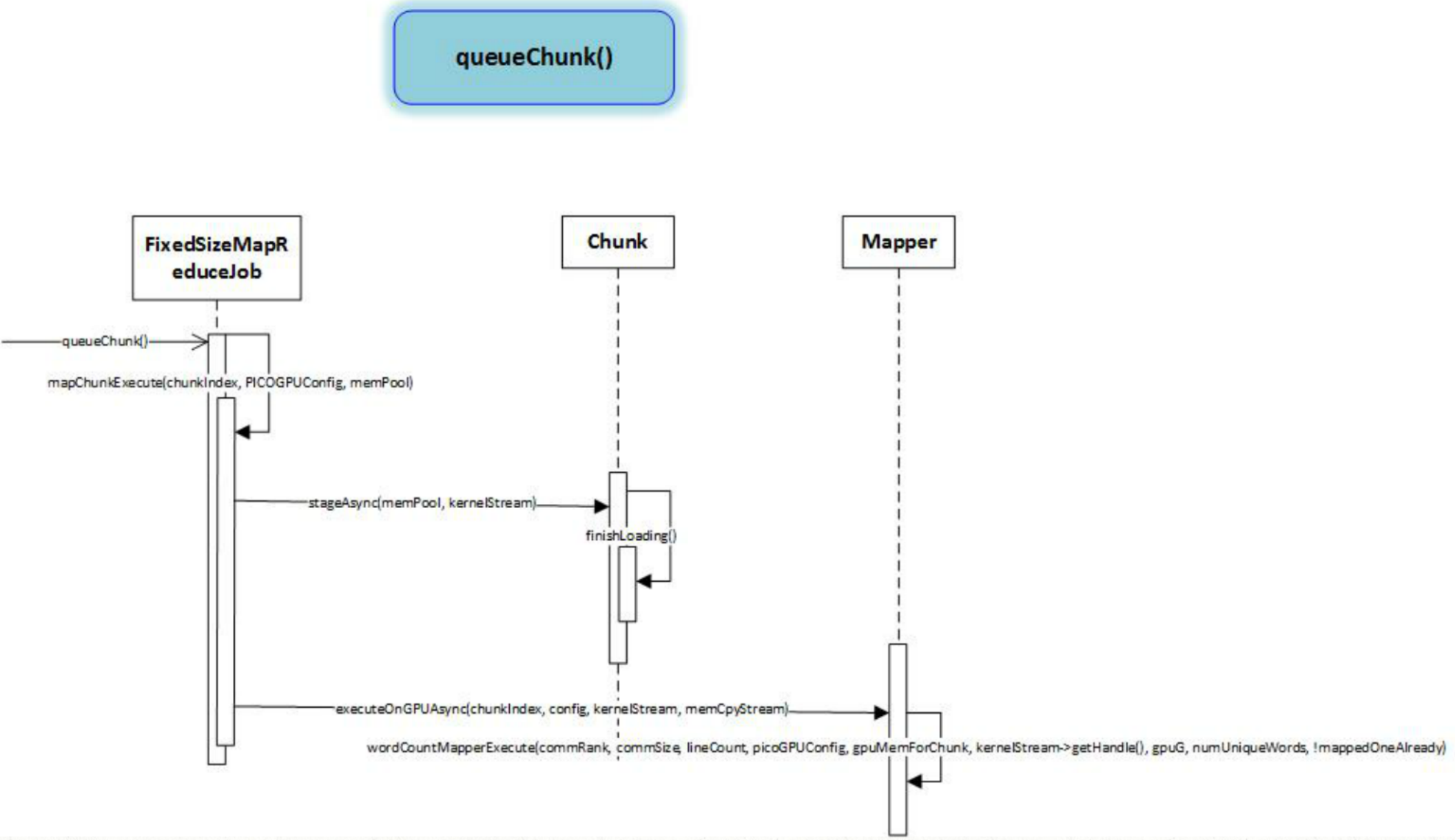

FixedSizeMapReduceJob
PartialReducer
Combiner
Mapper
execute()
alt
if Mapper != null
alt
if PartialReducer != null
init()
alt
if Combiner != null
init()
init()
map()
determineMemSpaceRequirements()
allocateMapVariables()
EmitConfigList
enqueueAllChunks()
loop
while(ChunkSize)
queueChunk()
GPUConfig
getThreadCount()
getEmitsPerThread()
mapChunkExecute()
mapChunkMemCopy()


map()
FixedSizeMapReduceJob
PartialReducer
Combiner
Mapper
execute()
alt
if Mapper != null
alt
if PartialReducer != null
init()
alt
if Combiner != null
init()
init()
map()
determineMemSpaceRequirements()
allocateMapVariables()
EmitConfigList

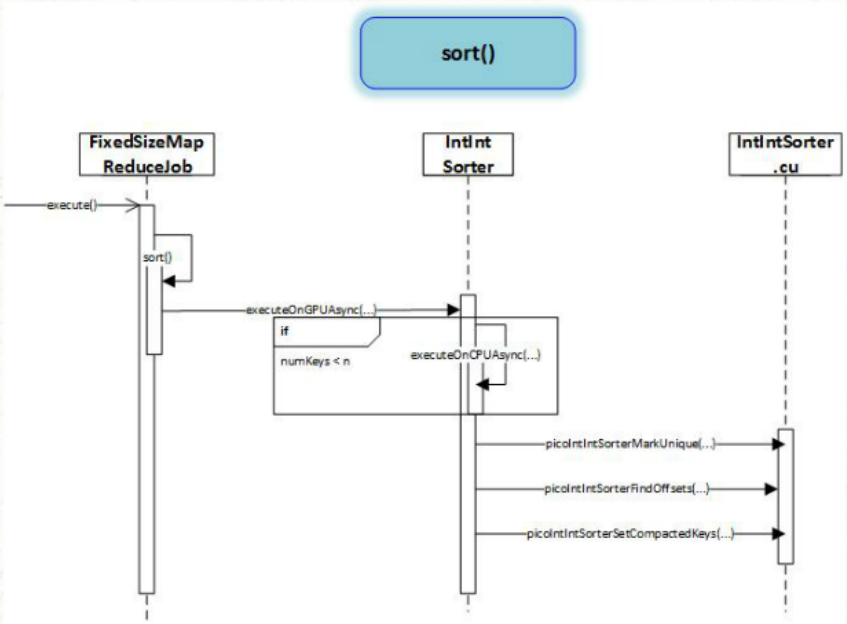

allocateMapVariables()
EmitConfigList
enqueueAllChunks()
loop
while(ChunkSize)
queueChunk()
GPUConfig
getThreadCount()
getEmitsPerThread()
mapChunkExecute()
mapChunkMemCopy()

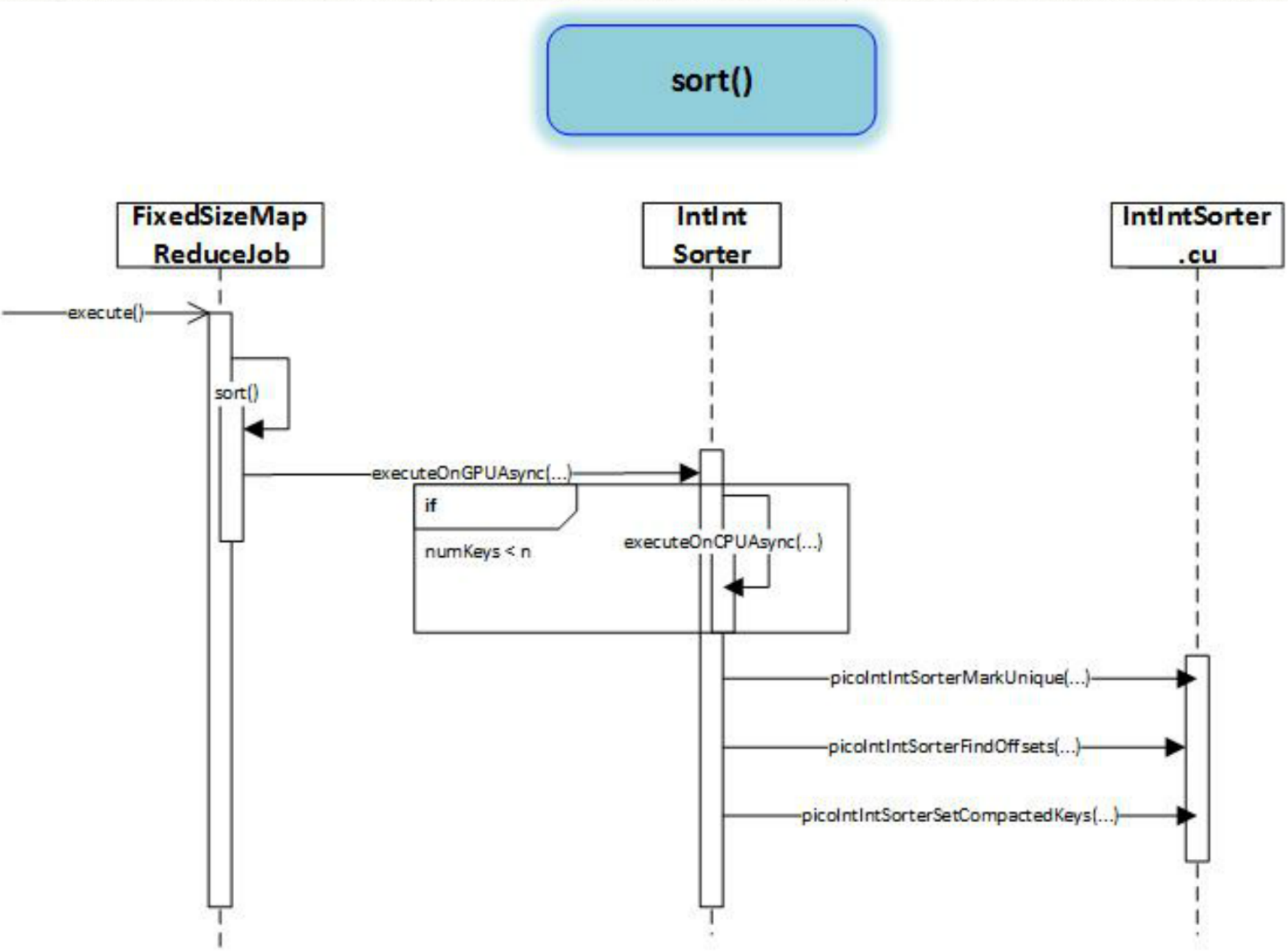
determineMaximumSpace Requirements()



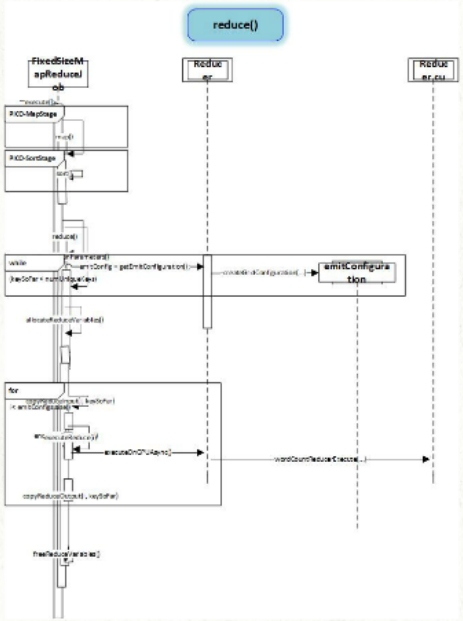

queueChunk()
FixedSizeMapReduceJob
Chunk
Mapper
queueChunk()
mapChunkExecute(chunkIndex, PICOGPUConfig, memPool)
stageAsync(memPool, kernelStream)
finishLoading()
executeOnGPUAsync(chunkIndex, config, kernelStream, memCpyStream)
wordCountMapperExecute(commRank, commSize, lineCount, picoGPUConfig, gpuMemForChunk, kernelStream->getHandle(), gpuG, numUniqueWords, !mappedOneAlready)

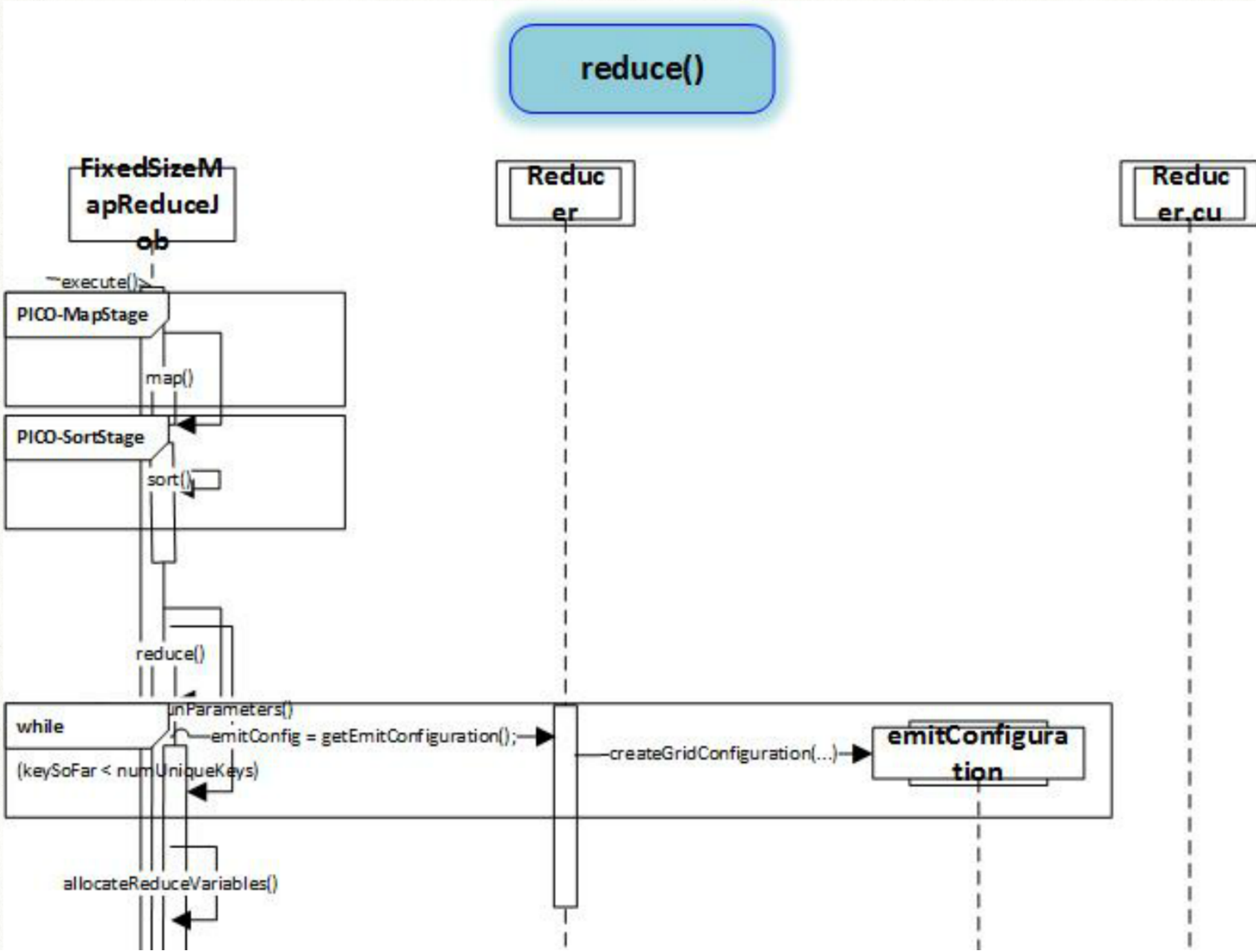
Sort

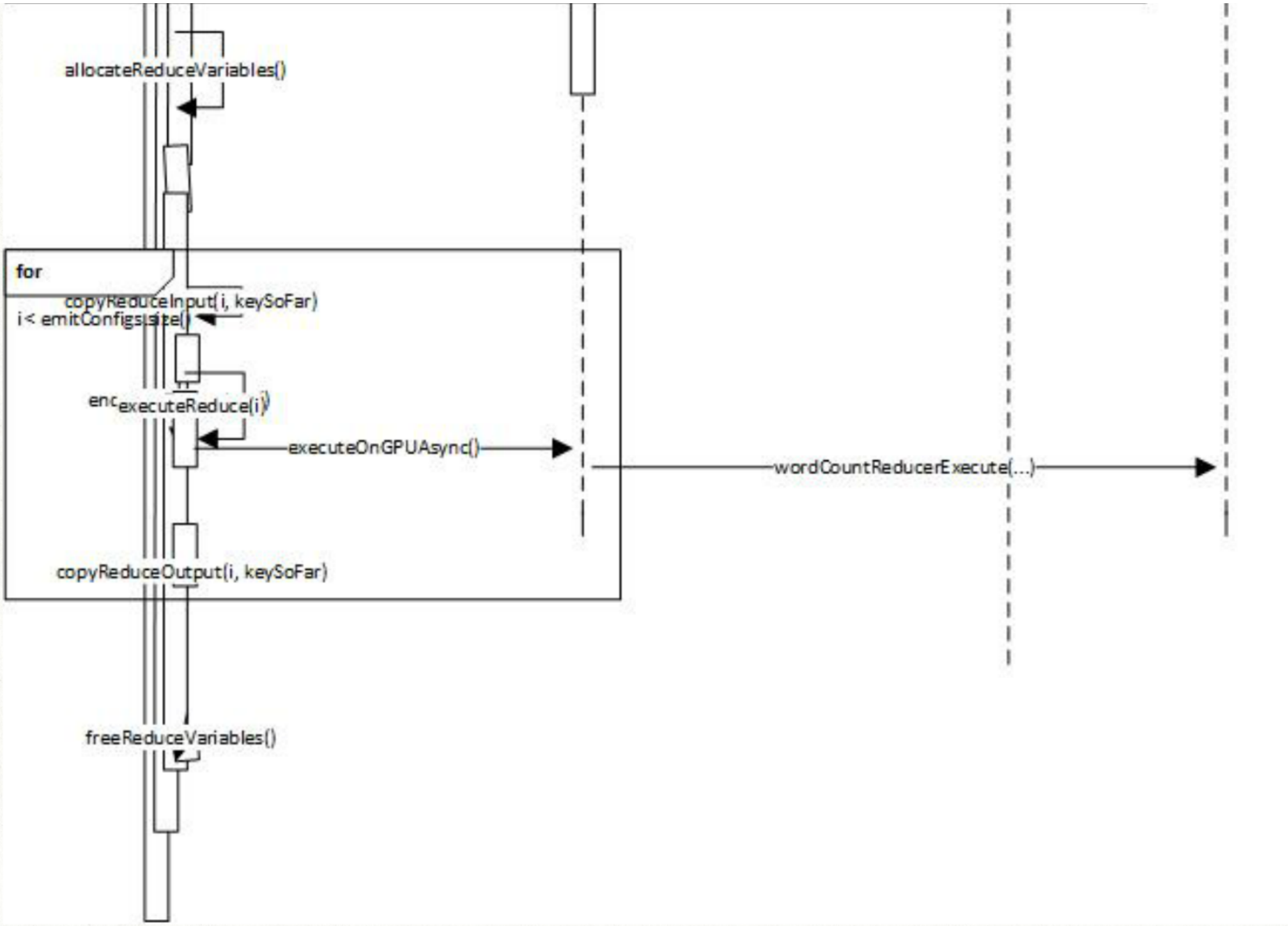

sort()
FixedSizeMap ReduceJob
IntInt Sorter
IntIntSorter .cu
execute()
sort()
executeOnGPUAsync(...)
if
numKeys < n
executeOnCPUAsync(...)
picoIntIntSorterMarkUnique(...)
picoIntIntSorterFindOffsets(...)
picoIntIntSorterSetCompactedKeys(...)


sort()
FixedSizeMap ReduceJob
IntInt Sorter
IntIntSorter .cu
execute()
sort()
executeOnGPUAsync(...)
if
numKeys < n
executeOnCPUAsync(...)
picoIntIntSorterMarkUnique(...)
picoIntIntSorterFindOffsets(...)
picoIntIntSorterSetCompactedKeys(...)

Reduce

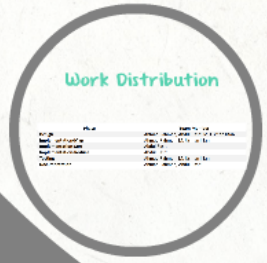

reduce()
FixedSizeMapReduceJob
Reducer
Reducer.cu
PKDMapStage
PKDSortStage
reduce()
while
emitConfiguration
for
executeOnCPUAsync()
wordCountReduceKernel...()
copyReduceOutput(..., keysSoFar)
freeReduceVariables()

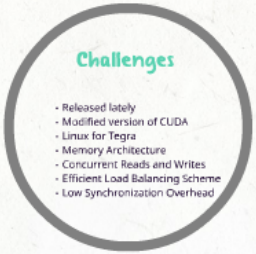

reduce()
FixedSizeMapReduceJob
Reducer
Reducer.cu
execute()
PICO-MapStage
map()
PICO-SortStage
sort()
reduce()
unParameters()
while
(keySoFar < numUniqueKeys)
emitConfig = getEmitConfiguration();
createGridConfiguration(...)
emitConfiguration
allocateReduceVariables()

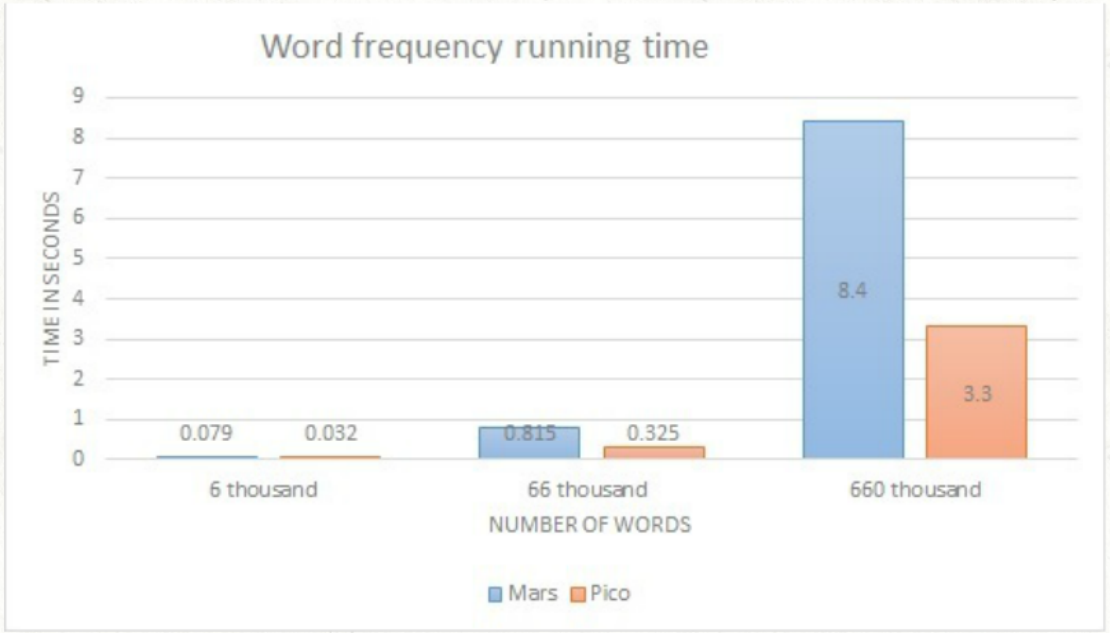

allocateReduceVariables()
for
copyReduceInput(i, keySoFar)
i< emitConfigs.size()
enc executeReduce(i)
executeOnGPUAsync()
wordCountReducerExecute(...)
copyReduceOutput(i, keySoFar)
freeReduceVariables()

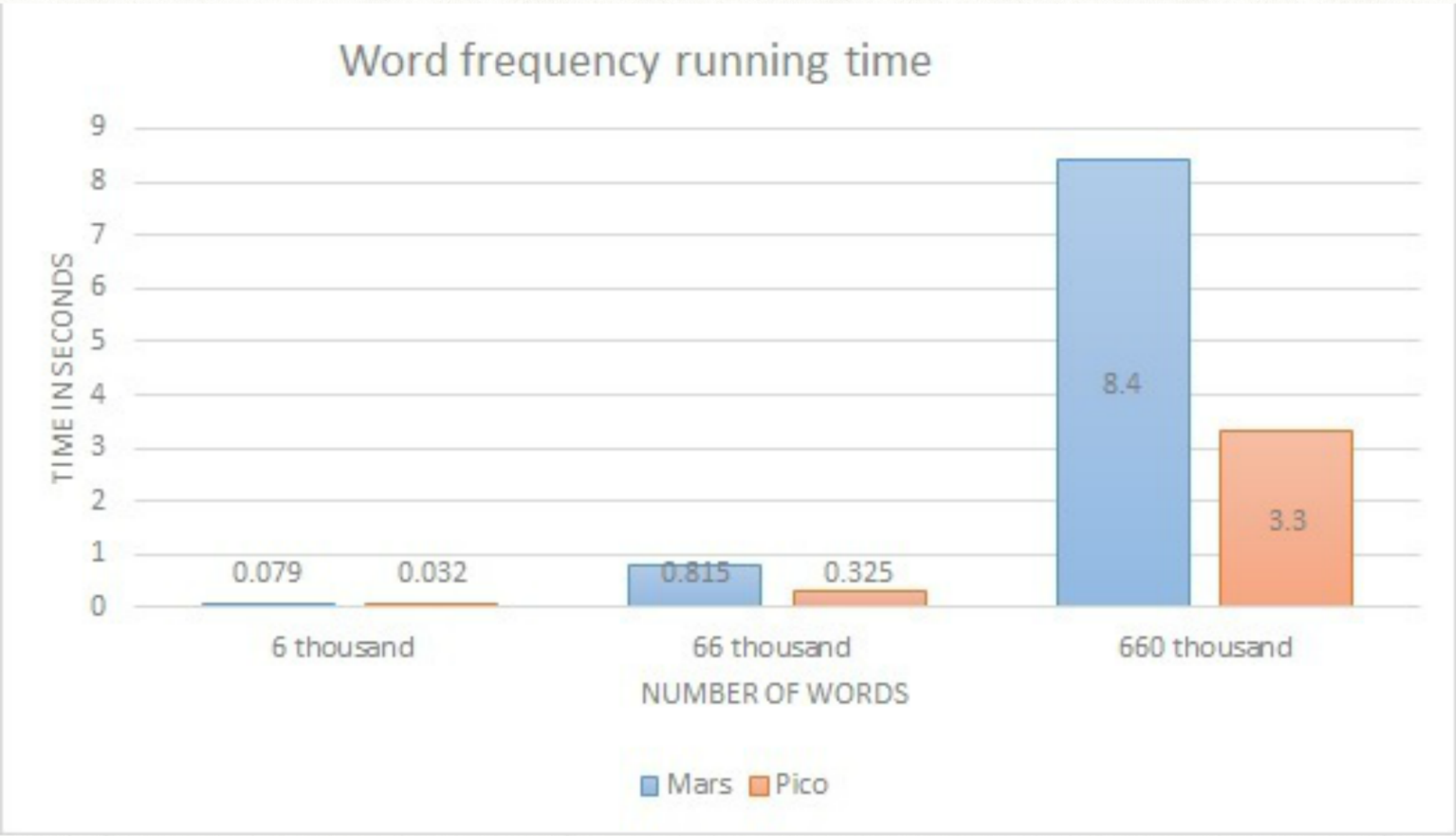
Results

- Word frequency Benchmark
- 3 subsets of dataset
- 2.47 Speed-up over Mars






Word frequency running time
TIME IN SECONDS
9
8
7
6
5
4
3
2
1
0
0.079
0.032
0.815
0.325
8.4
3.3
6 thousand
66 thousand
660 thousand
NUMBER OF WORDS
Mars
Pico


Word frequency running time
TIME IN SECONDS
9
8
7
6
5
4
3
2
1
0
0.079
0.032
0.815
0.325
8.4
3.3
6 thousand
66 thousand
660 thousand
NUMBER OF WORDS
Mars
Pico

Work Distribution

Phase	Team Member
Design	Ahmad Rahman, Abdul Basit, M. Usman Irfan
Implementation-Map	Ahmad Rahman, M. Usman Irfan
Implementation-Sort	Abdul Basit
Implementation-Reduce	Abdul Basit
Testing	Ahmad Rahman, M. Usman Irfan
Documentation	Ahmad Rahman, Abdul Basit

Work Distribution

Phase	Team Member
Design	Ahmad Rahman, Abdul Basit, M. Usman Irfan
Implementation-Map	Ahmad Rahman, M. Usman Irfan
Implementation-Sort	Abdul Basit
Implementation-Reduce	Abdul Basit
Testing	Ahmad Rahman, M. Usman Irfan
Documentation	Ahmad Rahman, Abdul Basit

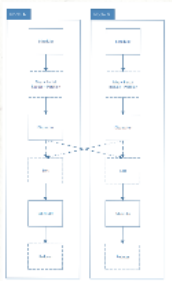
Future Direction

Multilevel Parallelism on
a cluster of Jetson TK1

Hardware

- NVidia Jetson TK1*
- Gigabit Switch
- SATA Hard Drive

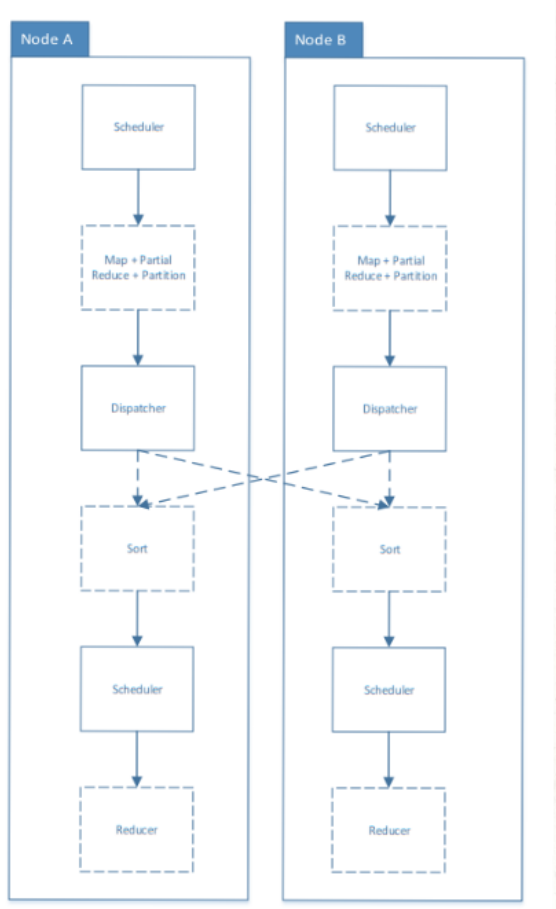
PICO - Multinode

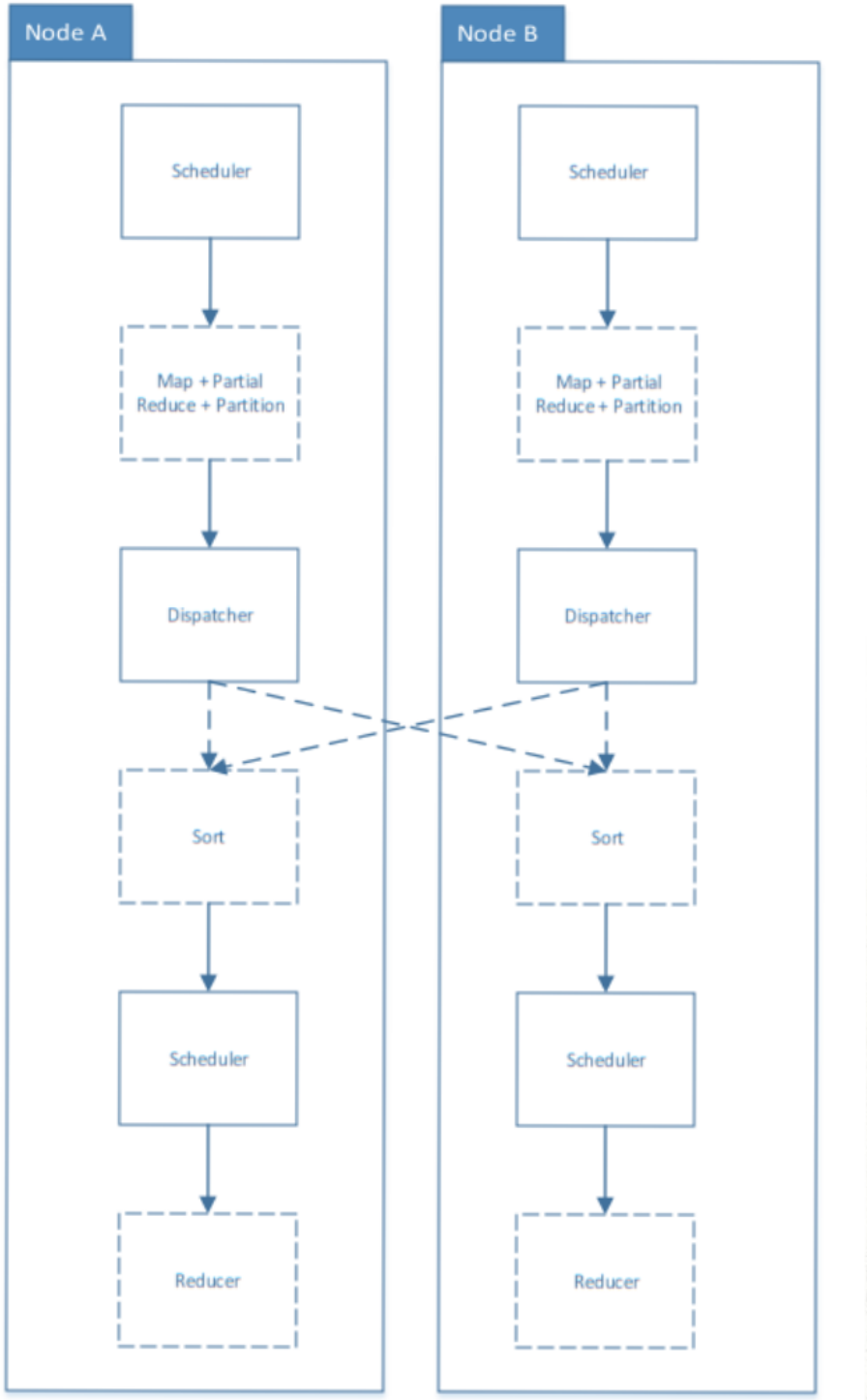


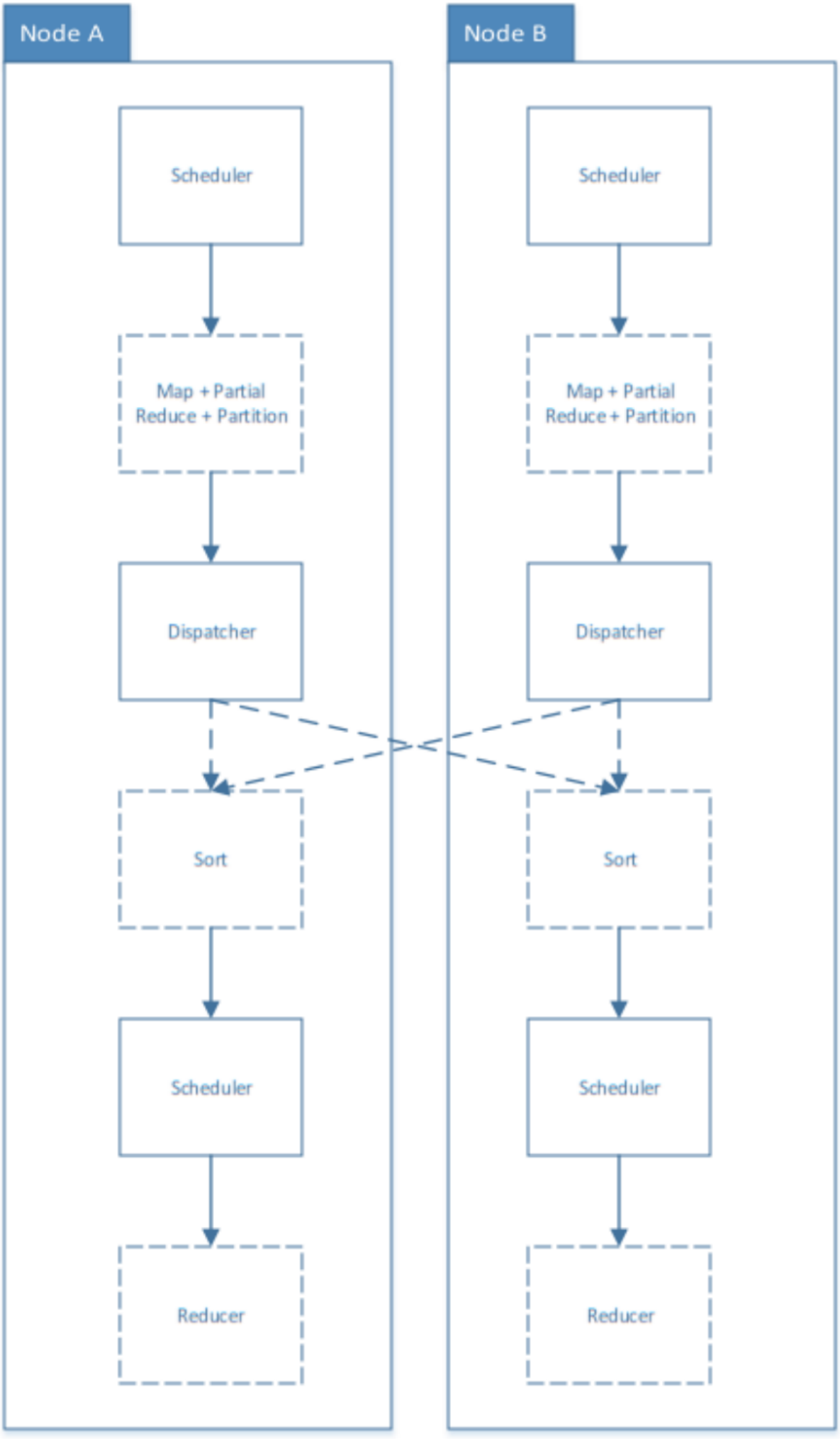
Hardware

- NVidia Jetson TK1*
- Gigabit Switch
- SATA Hard Drive

PICO - Multinode




Node A
Node B
Scheduler
Scheduler
Map + Partial Reduce + Partition
Map + Partial Reduce + Partition
Dispatcher
Dispatcher
Sort
Sort
Scheduler
Scheduler
Reducer
Reducer


Node A
Node B
Scheduler
Scheduler
Map + Partial Reduce + Partition
Map + Partial Reduce + Partition
Dispatcher
Dispatcher
Sort
Sort
Scheduler
Scheduler
Reducer
Reducer

References

- http://blog.kurtosys.com/12-big-facts-about-big-data/#.VBszDBY4S-J
- http://www.datanami.com/2014/05/29/hadoop-market-grow-58-2020-report-says/
- http://www.developer.com/db/10-facts-about-hadoop.html
- http://www.itbusinessedge.com/blogs/integration/five-facts-for-understanding-big-data-and-hadoop.html
- elinux.org/Jetson_TK1
- jetsonhacks.com/2015/06/03/battery-power-for-nvidia-jetson-tk1/
- infosys.com - A successful data center migration

Super Computing on the go: MapReduce for mobile GPUs