HeteroDoop: A MapReduce Programming System for Accelerator Clusters

Amit Sabne, Putt Sakdhnagool,
Rudolf Eigenmann

School of Electrical and Computer Engineering, Purdue University

Download : http://bit.ly/1BwxGER

PURDUE UNIVERSITY

Motivation - Why MapReduce?

- Data explosion
- Needs **distributed** programming models and frameworks
- MapReduce offers ease of programming:
  Underlying framework (e.g. Hadoop), not the user, is responsible for
  - Parallelization (intra and inter node)
  - Ensuring data locality
  - Assuring fault tolerance
- MapReduce requirement: Only two operations Map (completely parallel), Reduce (partially parallel)

Motivation - Why Accelerators?

- Massive parallelism → good fit for Map
- High memory bandwidth (~10x of CPUs)
- High perf/watt (~5x of CPUs)
- Commonplace, e.g. GPUs come built-in

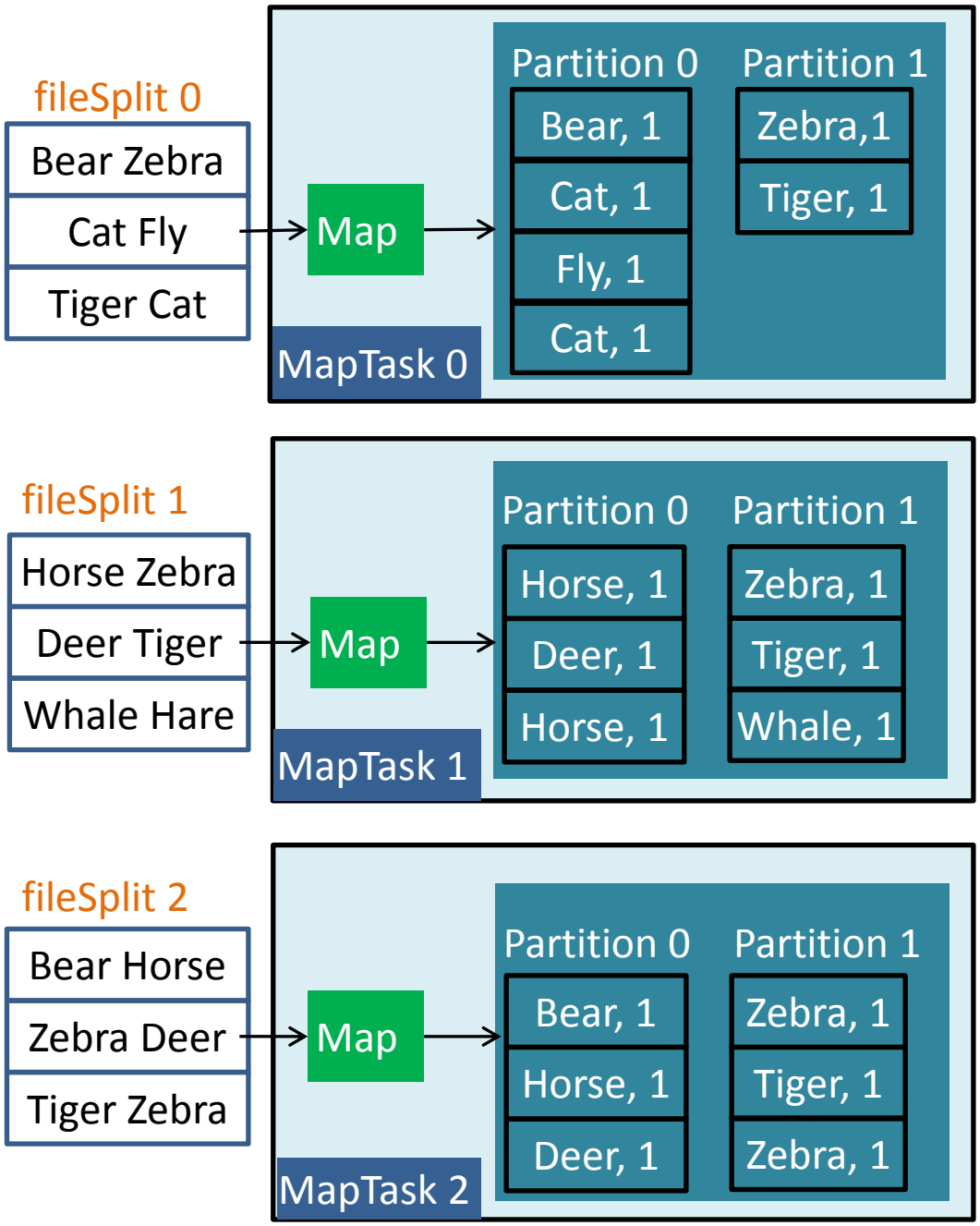
Hadoop

- Most popular, open-source MapReduce implementation

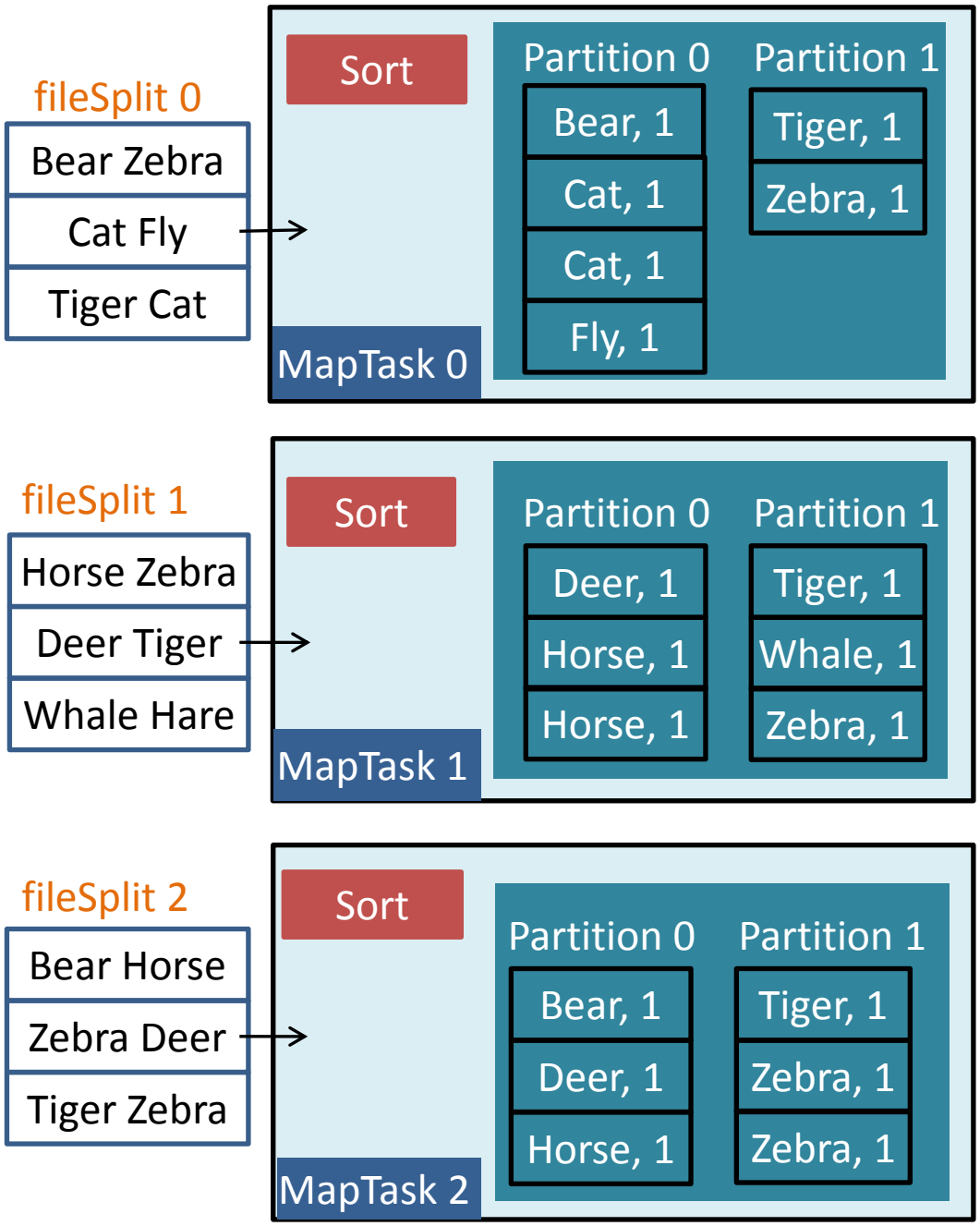
- HDFS

- Hadoop Streaming
  - Requires just the executables for map, combine and reduce
  - User may write the program in any language adhering to Unix filter style (IO via STDIN/STDOUT)

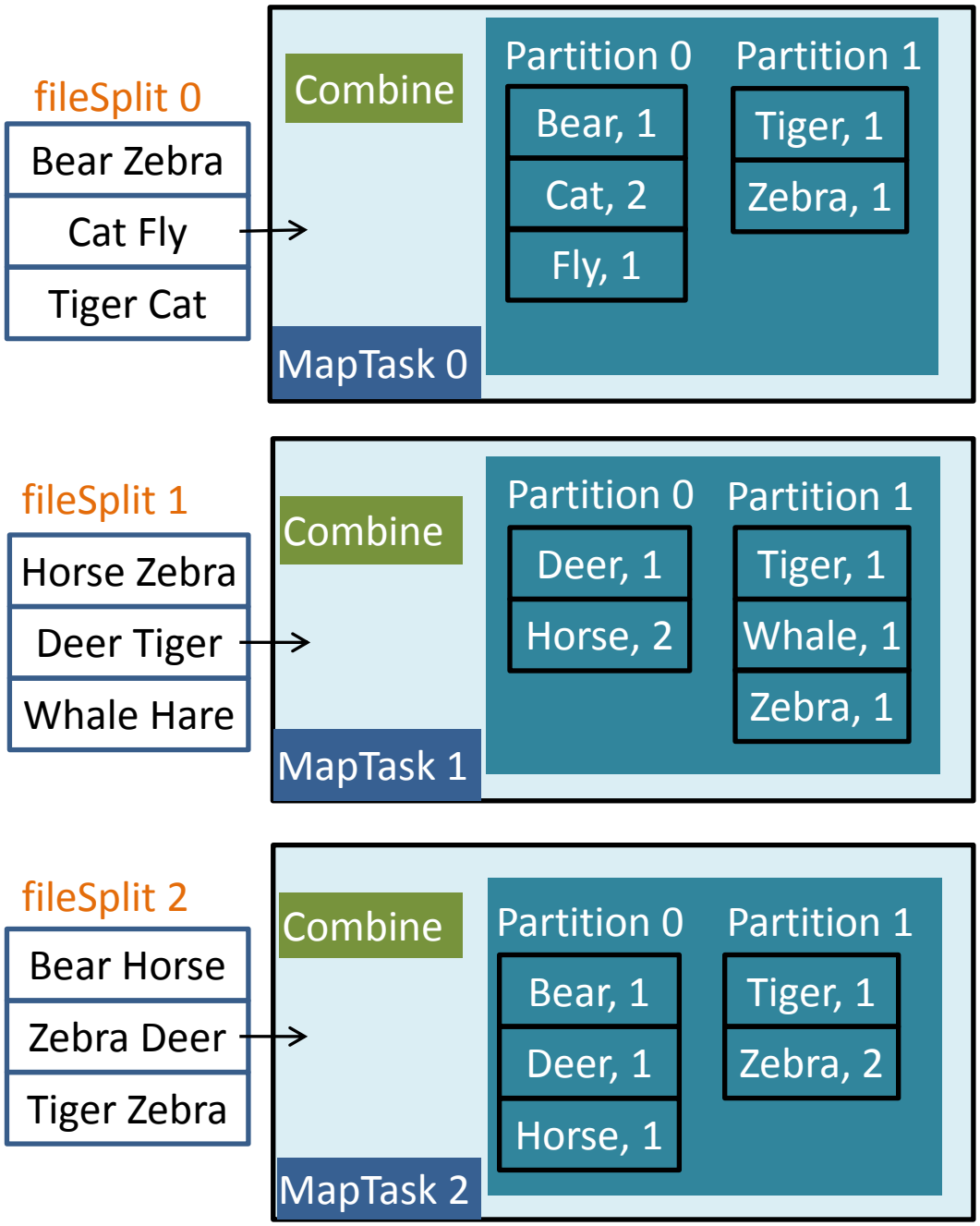
MapReduce – WordCount Example



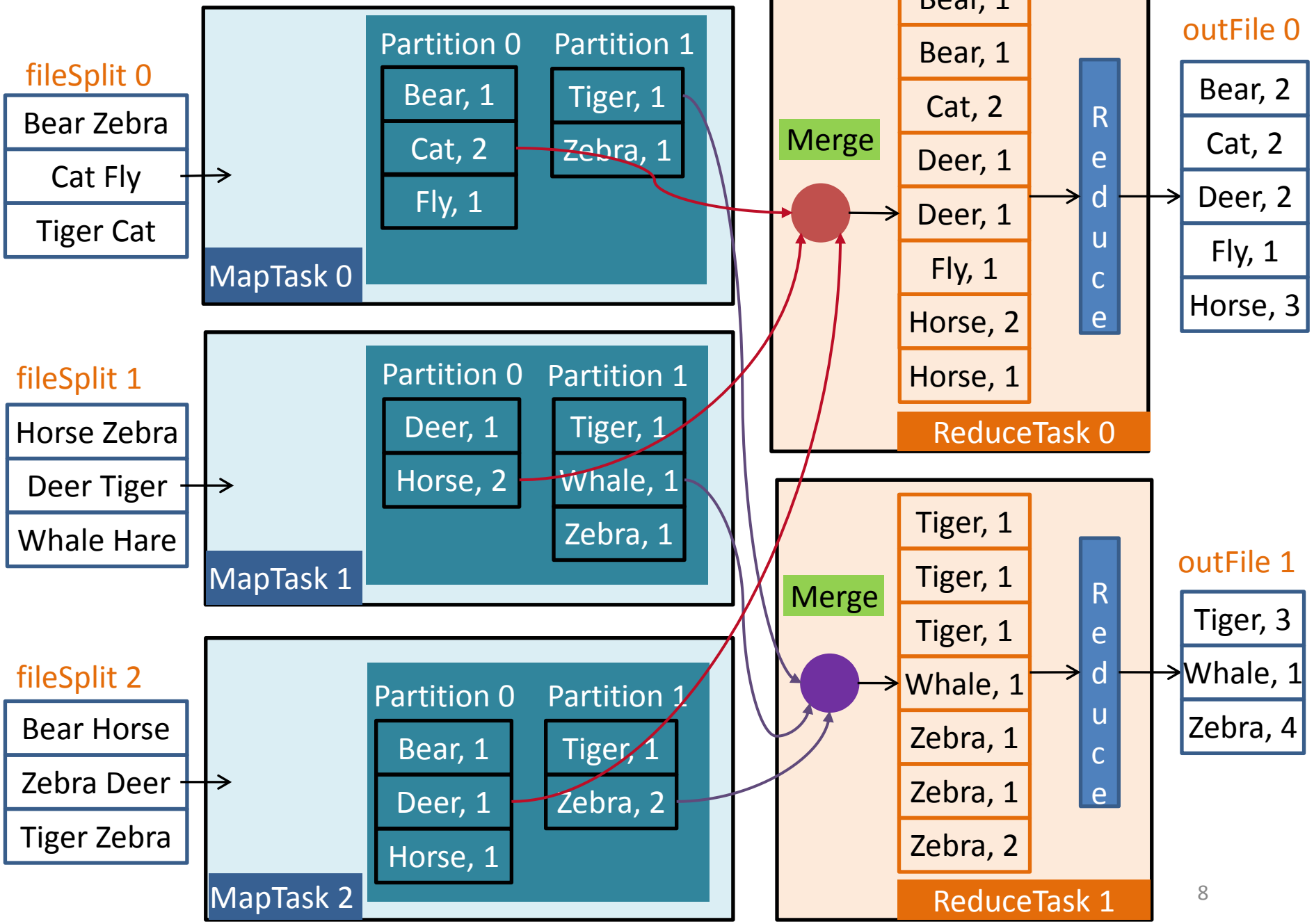
MapReduce – WordCount Example



MapReduce – WordCount Example



MapReduce – WordCount Example



HeteroDoop: Challenges and Approach

- **Accelerator programming models differ from those of CPUs. To exploit both, the user would have to write two program source codes**
  - HeteroDoop offers **program constructs** so that a CPU-only program can now run on accelerators as well
  - An **optimizing compiler** translates codes to accelerator programs

- **Parallelism exploitation in Hadoop : One fileSplit per core – would exceed available GPU memory**
  - Use **record-level parallelism** on the GPU, retain fileSplit-level parallelism for the CPU.

- **Lack of MapReduce semantics for Accelerators**

  e.g. Intermediate sort
  - HeteroDoop contains a GPU-side runtime system

- **Load Balancing : Accelerators are faster than CPUs**
  - We present a **tail scheduling scheme** to optimize the execution

HeteroDoop Program Constructs

```
int main() {
  char word[30], *line;
  size_t nbytes = 10000;
  int read, linePtr, offset, one;
  line = (char*) malloc(nbytes*sizeof(char));


  //read input file line by line
  while ((read = getline(&line, &nbytes, stdin)) != -1) {
    linePtr = 0;
    offset = 0;
    one = 1;
    while( (linePtr = getWord(line, offset, word,
      read, 30)) != -1) {//read words in the line
      printf("%s\t%d\n", word, one); //emit <word, 1>
      offset += linePtr;
    }
  }
  free(line);
  return 0;
}
```

WordCount Map Code

CPU-only

Independent Iterations

HeteroDoop Program Constructs

```
int main() {
  char word[30], *line;
  size_t nbytes = 10000;
  int read, linePtr, offset, one;
  line = (char*) malloc(nbytes*sizeof(char));
  #pragma mapreduce mapper key(word) value(one) \\
  keylength(30) vallength(1)
  //read input file line by line
  while ((read = getline(&line, &nbytes, stdin)) != -1) {
    linePtr = 0;
    offset = 0;
    one = 1;
    while( (linePtr = getWord(line, offset, word,
      read, 30)) != -1) {//read words in the line
      printf("%s\t%d\n", word, one); //emit <word, 1>
      offset += linePtr;
    }
  }
  free(line);
  return 0;
}
```

WordCount Map Code

CPU + Accelerator

HeteroDoop Program Constructs

```
int main() {
  char word[30], prevWord[30];
  int count, val, read;


  //read map emitted KV pairs, which are already sorted
  while( (read = scanf("%s %d", word, &val)) == 2 ) {
    if(strcmp(word, prevWord) == 0 ) {
      count += val; //sum up occurrences of the same word
    } else {
      if(prevWord[0] != '\0')
        printf("%s\t%d\n", prevWord, count);
      strcpy(prevWord, word);
      count = val;
    }
  }
  if(prevWord[0] != '\0')
    printf("%s\t%d\n", prevWord, count);

  return 0;
}
```

CPU-only

WordCount Combine Code

Data-dependence is present

HeteroDoop Program Constructs

CPU + Accelerator

WordCount Combine Code

```
int main() {
  char word[30], prevWord[30];
  int count, val, read;
  #pragma mapreduce combiner key(prevWord) value(count)
  keyin(word) valuein(val) keylength(30) vallength(1)
  firstprivate(prevWord, count) {
  //read map emitted KV pairs, which are already sorted
  while( (read = scanf("%s %d", word, &val)) == 2 ) {
    if(strcmp(word, prevWord) == 0 ) {
      count += val; //sum up occurrences of the same word
    } else {
      if(prevWord[0] != '\0')
        printf("%s\t%d\n", prevWord, count);
      strcpy(prevWord, word);
      count = val;
    }
  }
  if(prevWord[0] != '\0')
    printf("%s\t%d\n", prevWord, count);
  }
  return 0;
}
```

HeteroDoop Program Constructs : Why?

- Low-level models (CUDA/OpenCL) : Programmer has to
  - Manage CPU-Accelerator data transfers
  - Identify the parallelism and launch “kernels”
  - Exploit intricate memory hierarchy (e.g. shared, textures memories)

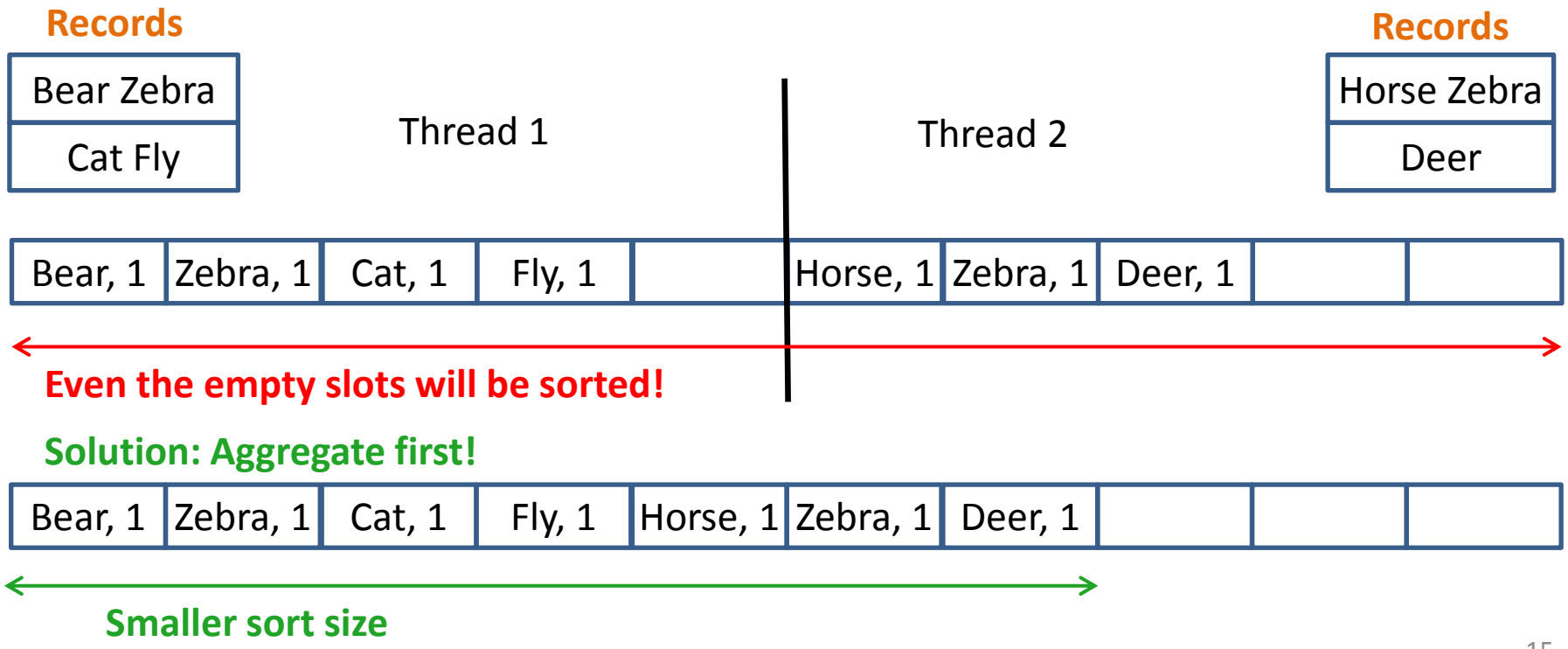
- High-level programming models (OpenMPC, OpenACC, OpenMP 4.0) : Programmer has to
  - Identify parallelism
  - Lose out on architecture-specific optimizations

- Why not use OpenACC etc. for MapReduce?
  - Inherent mismatch : MapReduce programmers write only a **serial** code, OpenACC programmers write a **parallel** code
  - OpenACC does not understand MapReduce semantics, restricting the scope of optimizations.

Parallelization Strategies for the GPU Code

Map

- Explicitly parallel
- Each thread works on different records and places output key-value (KV) pairs in its portion of a **global KV store**
  - Each thread has a portion in the global KV store for each **partition**
- Global KV store is conservatively over allocated

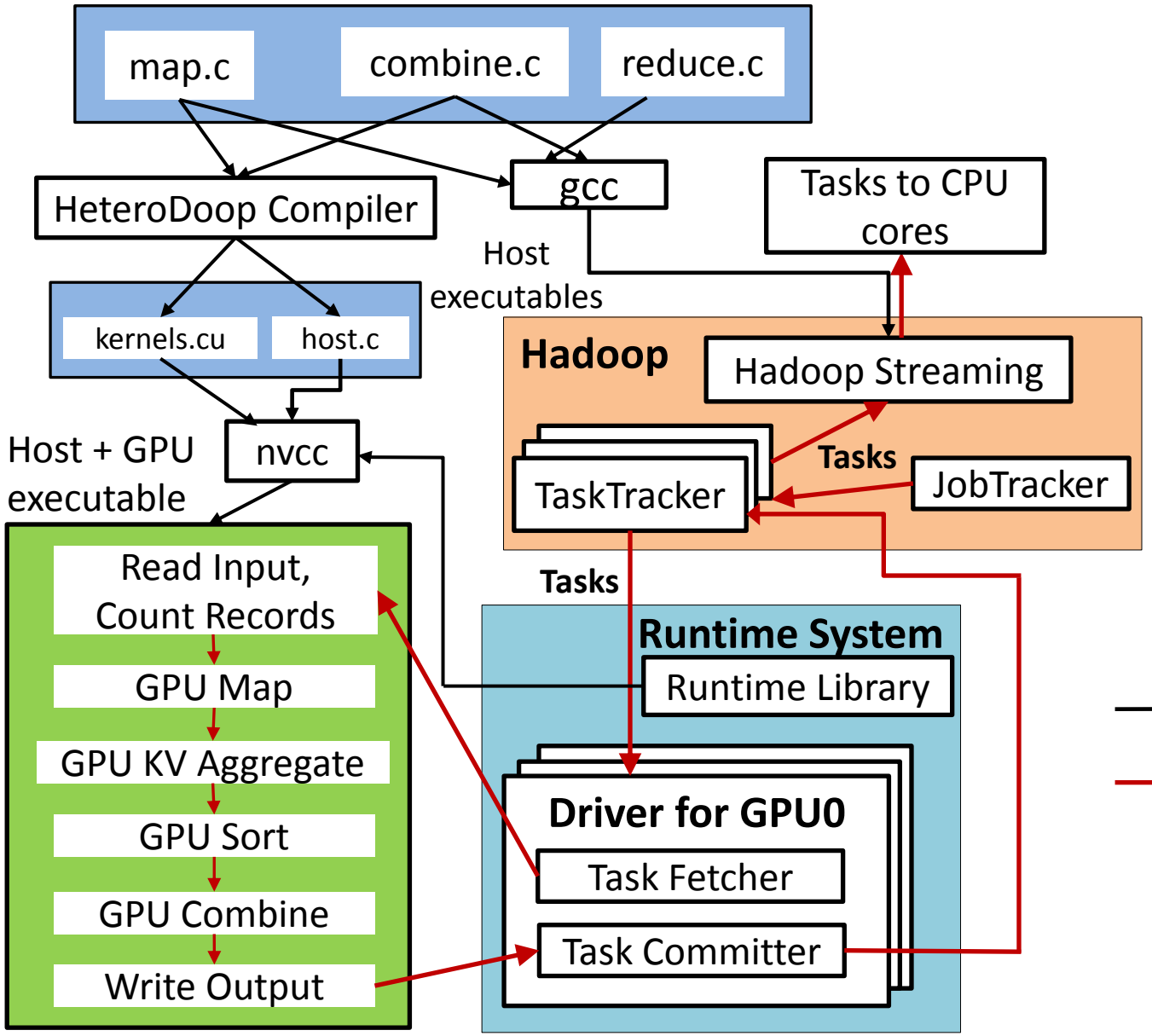


Parallelization Strategies for the GPU Code

- **Combine**
  - Not explicitly parallel, but parallel across partitions
  - We exploit reduction-style parallelism **inside** a partition
  - Beneficial to run on the GPU since data is already present in the GPU memory

- **Reduce**
  - Only on the CPU
    - Data is NOT already on the GPU
    - Number of reducers is typically low → barely any parallelism to use GPUs

HeteroDoop Execution Scheme



- Hadoop exploits all CPU cores of a node by running multiple **slots** on the TaskTracker of the node
- HeteroDoop scheme runs one extra slot per GPU on the TaskTracker

HeteroDoop Compiler

- Auto-translator from annotated C to CUDA
- Built with Cetus
- Generates host (launcher) + device (kernel) code
- Generated code contains calls to functions of the **runtime system**

Compiler Optimizations

Map

- No shared writeable data → All variables can be privatized
- Optimizations :
  - Dynamic record fetching (record stealing)
  - Use of CUDA vector data types (e.g. char4)

Combine

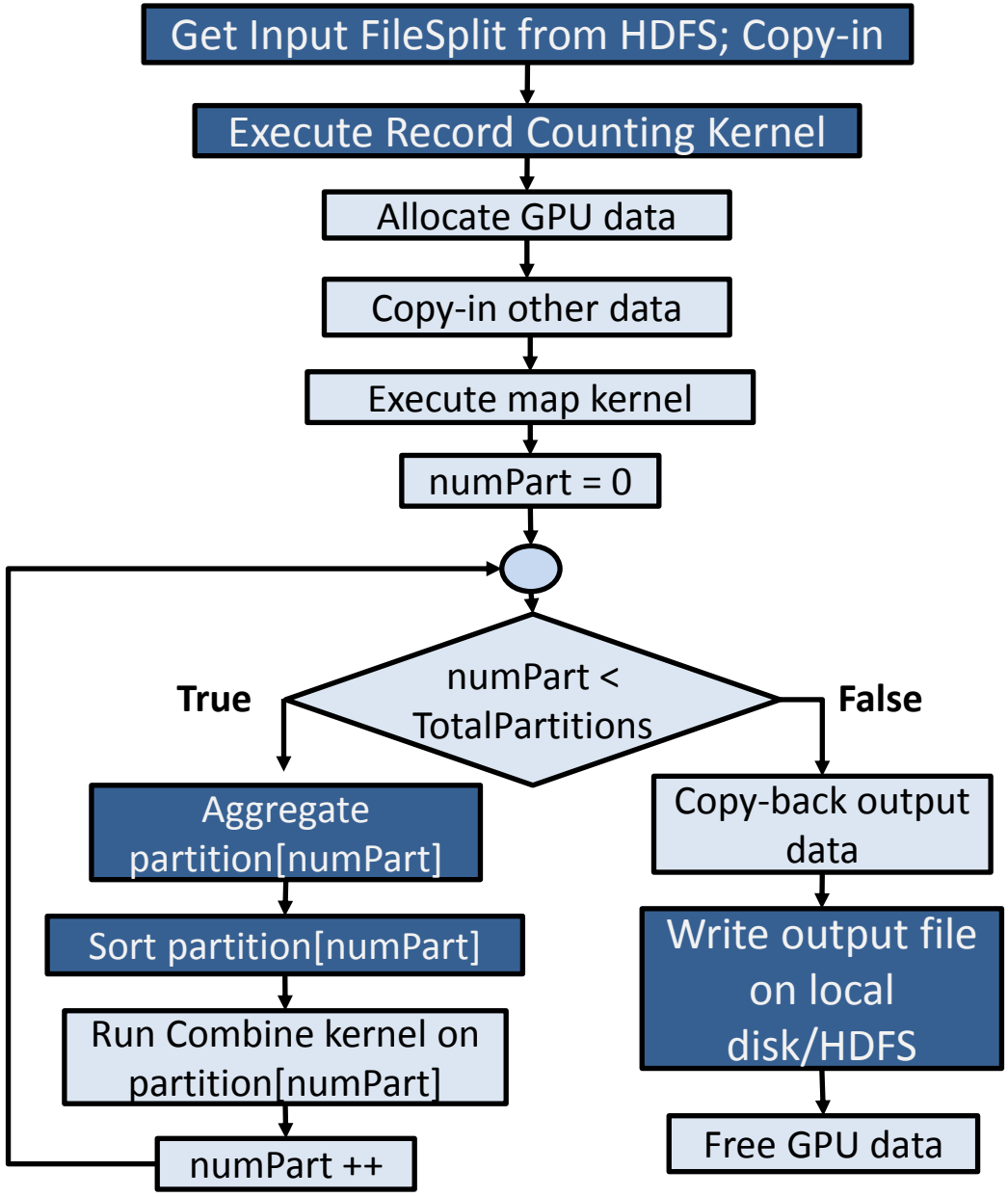
- Optimizations :
  - Less parallelism → Use 1 thread per warp → **no warp divergence**
  - Use remaining warp threads to vectorize certain operations e.g. KV read/write, strcpy etc.
  - Private arrays are placed in the GPU shared memory → faster access

Host Code Flowchart

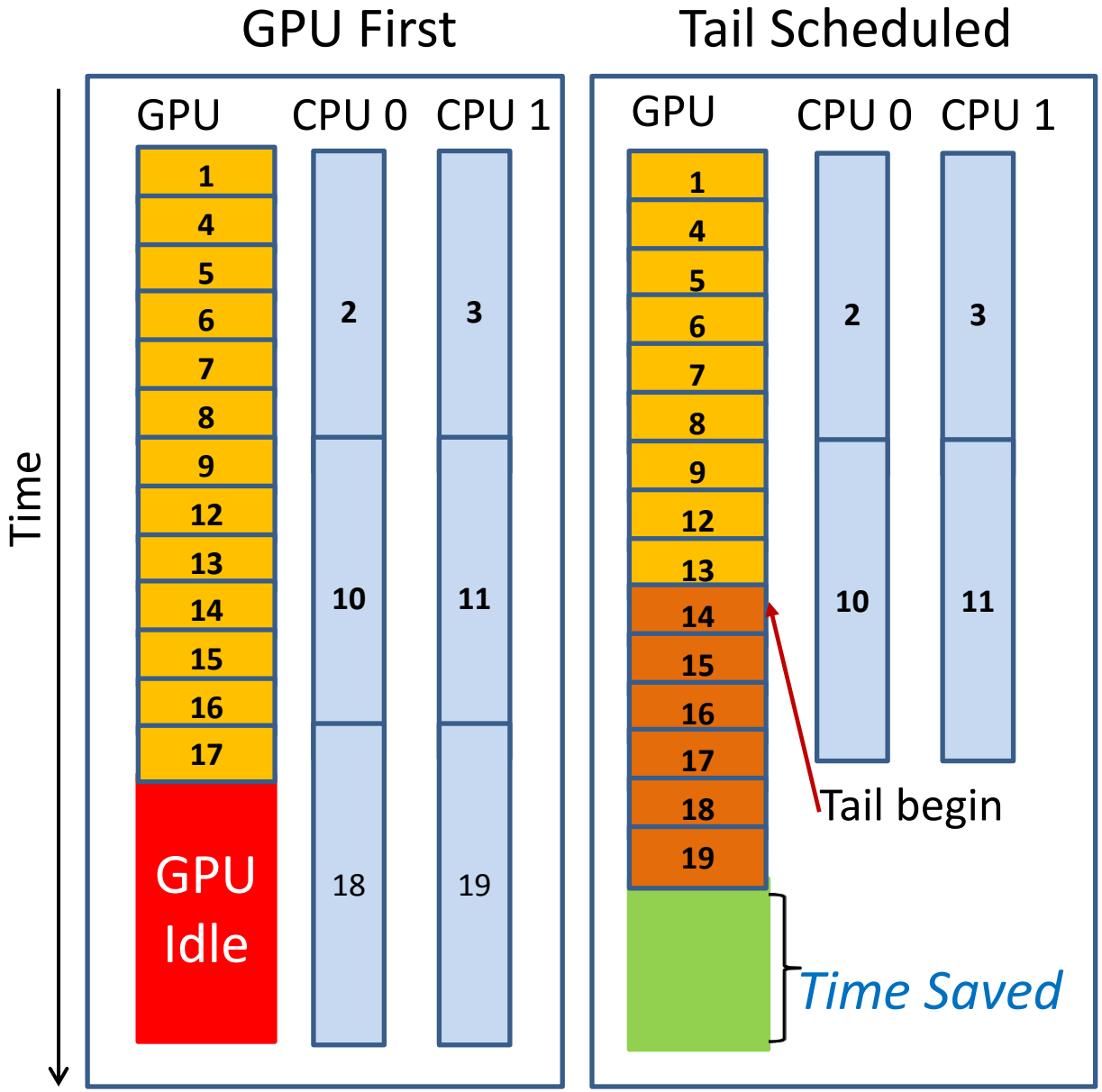
Runtime System Call

Host Code

- Runtime system supports kernel code by providing functions for:
  - read/write KV pairs
  - standard library functions for the GPU e.g. strcmp



Tail Scheduling



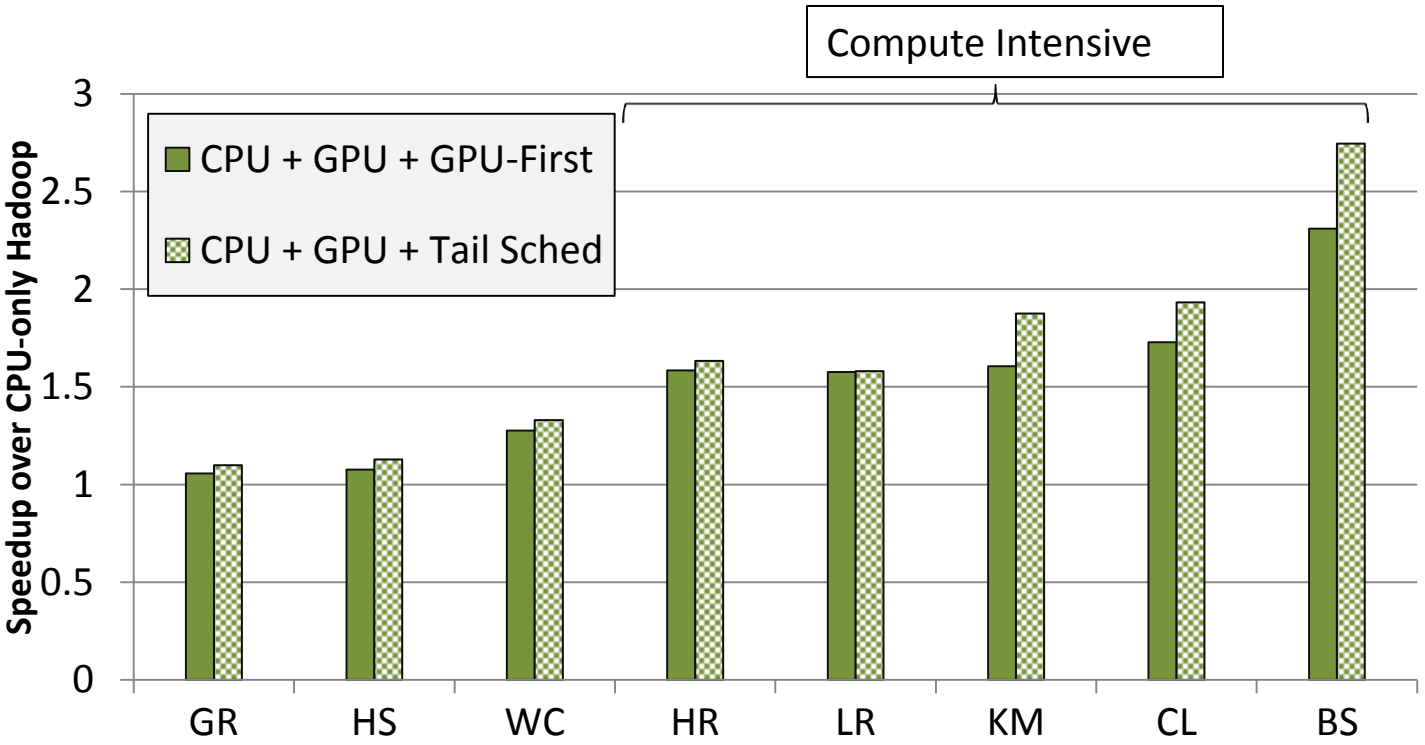
- **GPU First** – naïve strategy where a GPU slot is preferred over a CPU slot
- **Tail Scheduling**: Force tailing tasks on the GPU
- Tail size = GPU slot speedup over the CPU slot (6x in this example)

Evaluation : Setup

	Cluster1	Cluster2
#nodes	48 (+1 master)	32 (+1 master)
CPU	Intel Xeon E5-2680, 20 cores	Intel Xeon X5560, 12 cores
GPU	Tesla K40 (Kepler) **New HW**	3 x Tesla M2090 (Fermi) **Old HW**
RAM	256GB	24GB
Disk	500GB	**None (in-memory system)**
Network	FDR Infiniband	QDR Infiniband
HDFS Block Size	256 MB	256 MB
HDFS Replication Factor	3	1
Max. Map Slots	20 (+1 for GPU runs)	4 (+1 per GPU for GPU runs)
Max. Reduce Slots	2	2

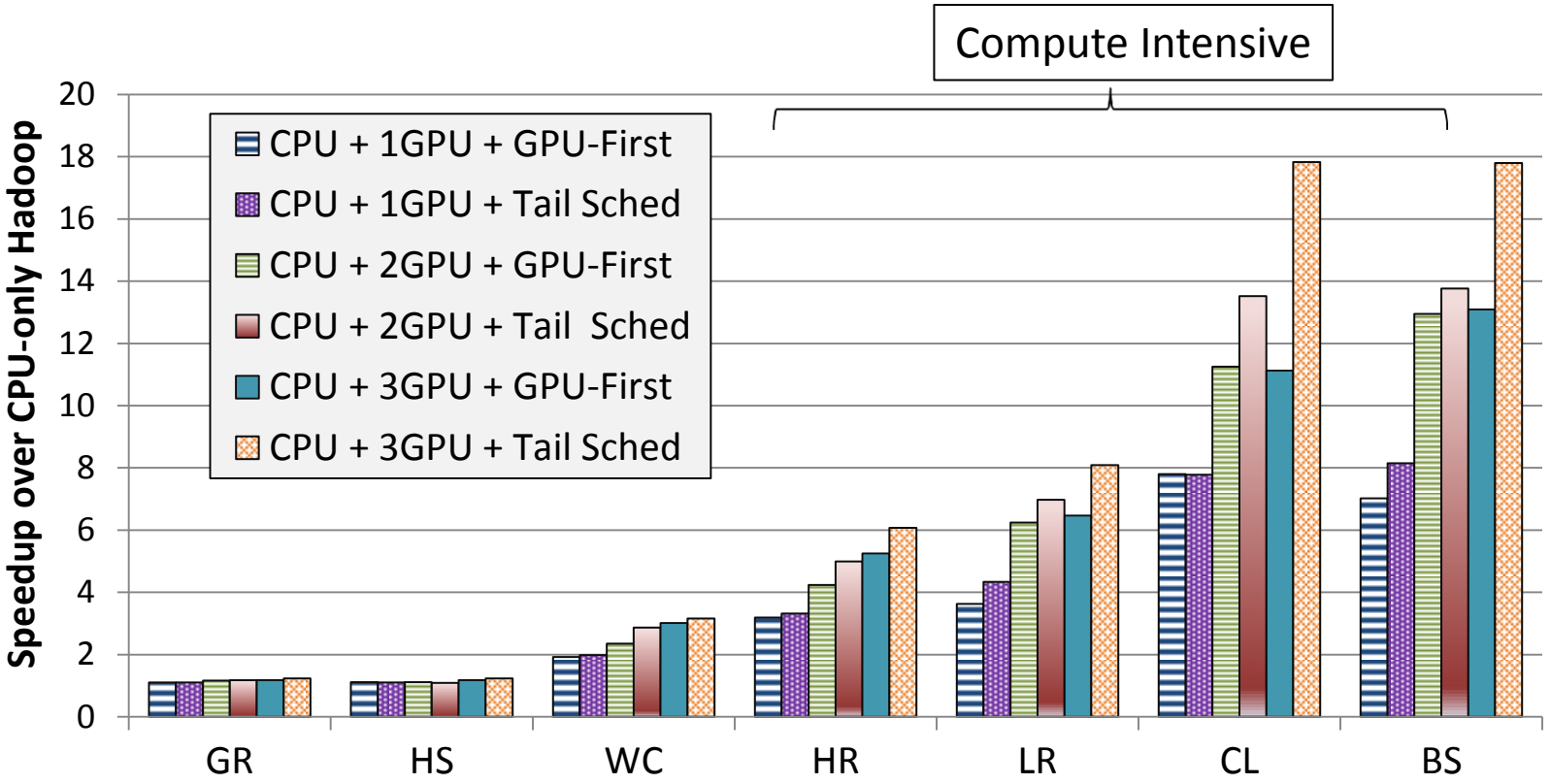
- Speculative execution was disabled
- Reduce phase was started after 20% map execution
- Cluster2 : Used for evaluating multi-GPU scalability

Evaluation : Overall Benefits – Cluster1



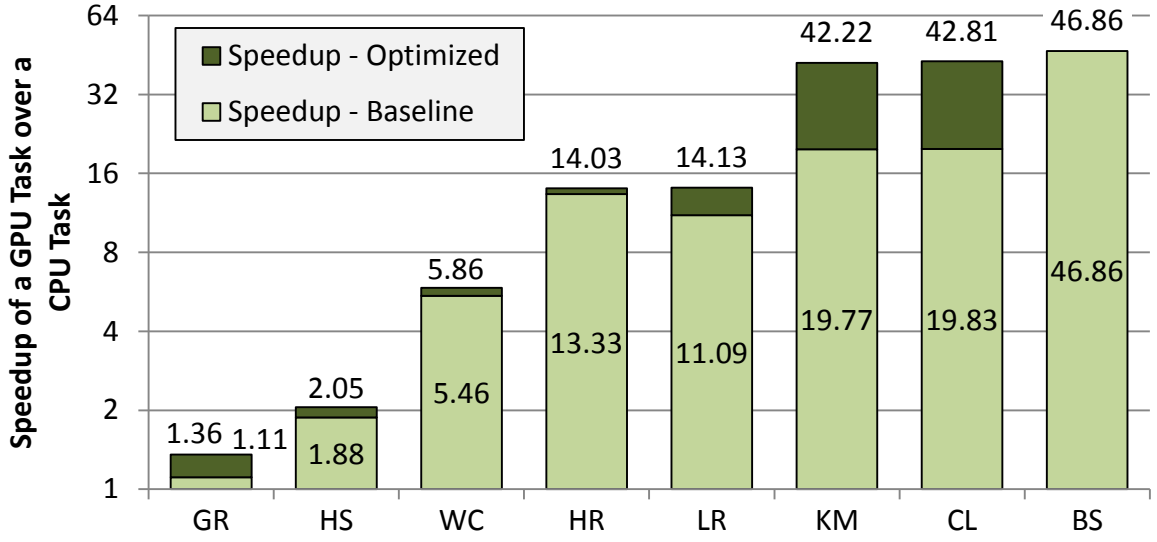
- Speedup with Tail Scheduling 1.6x (geo mean)
- Speedup with GPU-First 1.48x (geo mean)
- Higher speedups for compute-intensive applications

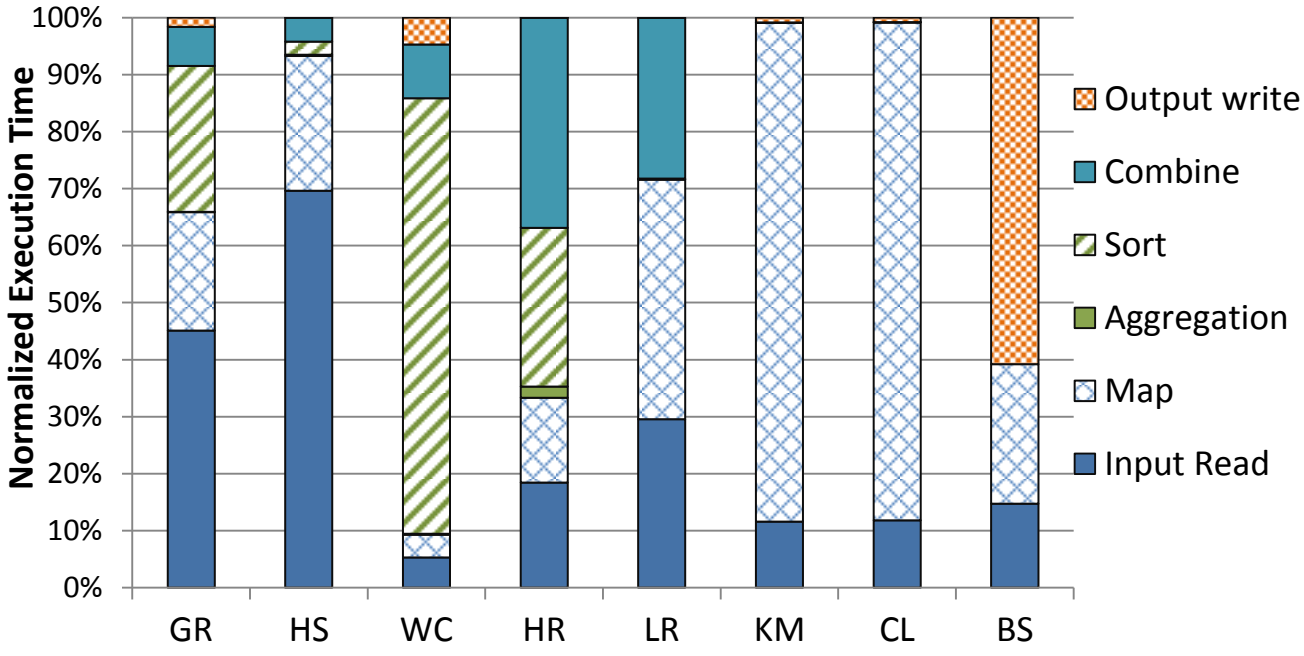
Evaluation : Overall Benefits – Cluster2



- Larger speedups than Cluster1 since Cluster2 uses less CPU cores
- Scalable performance with #GPUs

Individual Task Performance





- Single task speedup: max 47x
- Optimizations can have a high impact
- Bottlenecks are application dependent

Conclusion

HeteroDoop is a MapReduce programming system for accelerator clusters that features

- Single input source for the CPUs and GPUs
- Optimizing compiler to generate GPU program
- Runtime system to handle MapReduce semantics on GPUs
- Tail scheduling scheme that optimizes execution on an intra-node heterogeneous cluster

Download : http://bit.ly/1BwxGER

PURDUE UNIVERSITY