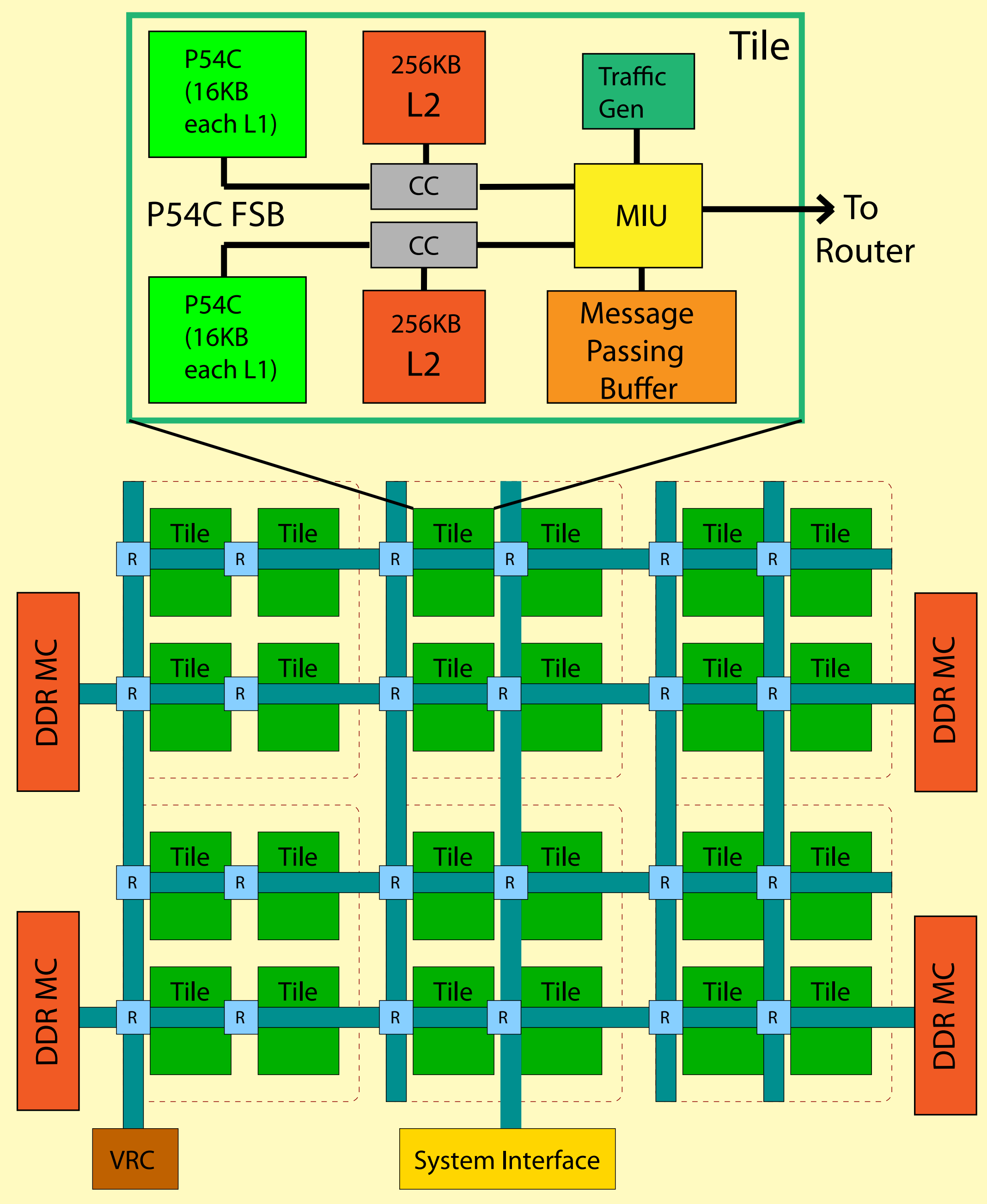
MapReduce for the Single-Chip-Cloud Architecture

Anastasios Papagiannis and Dimitrios S. Nikolopoulos
{apapag, dsn}@ics.forth.gr

Abstract

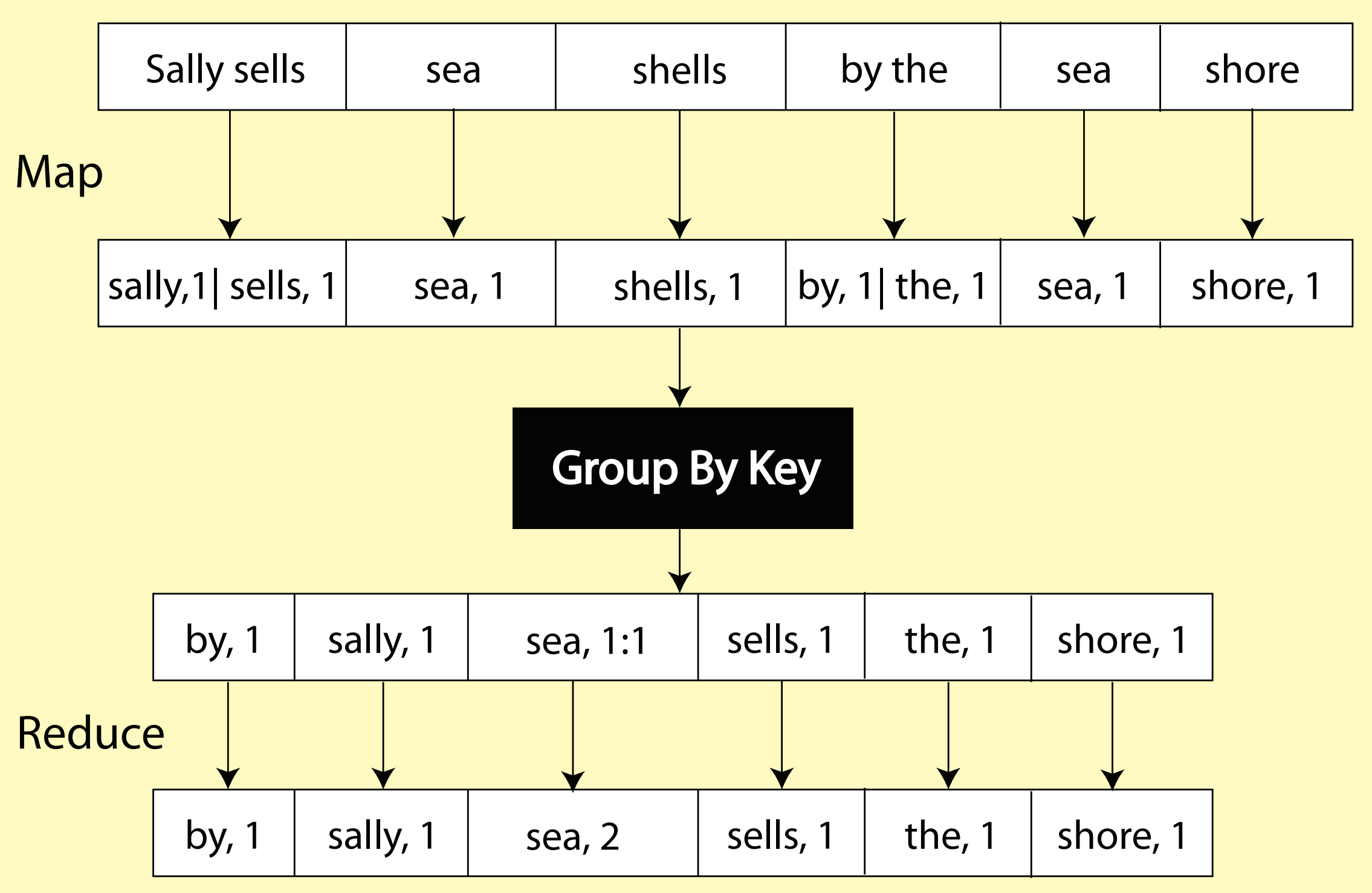
Many-core processors, due to their complexity and diversity, will necessitate high-productivity, domain-specific approaches to parallel programming. These approaches should hide architectural details and low-level parallel programming constructs while enabling scalability. We implement MapReduce on Intel SCC. MapReduce is a large scale data processing framework and SCC is a 48 core research processor.

Background - Intel SCC



Intel SCC is a many-core processor with 24 tiles and 2 IA cores per tile. Tiles organized in a 4×6 mesh network with 256 GB/s bisection bandwidth. Each core has a private L1 instruction cache of 16 KB, a private L1 data cache of 16 KB and a private unified L2 cache of 256 KB. Also each tile conatins 16 KB message passing buffer (MPB) (only on-chip memory shared between cores).

Background - MapReduce



MapReduce is a framework for large-data scale processing, based on functional programming language primitives. Runtime has to deal with fault-tolerance, parallelization, scheduling, synchronization and communication.

References

[1] Dean, Jeffrey and Ghemawat, Sanjay. MapReduce: Simplified Data Processing on Large Clusters In *Commun. ACM 2008*

[2] Mattson, Timothy G. and et al. The 48-core SCC Processor: the Programmer's View In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*

Design

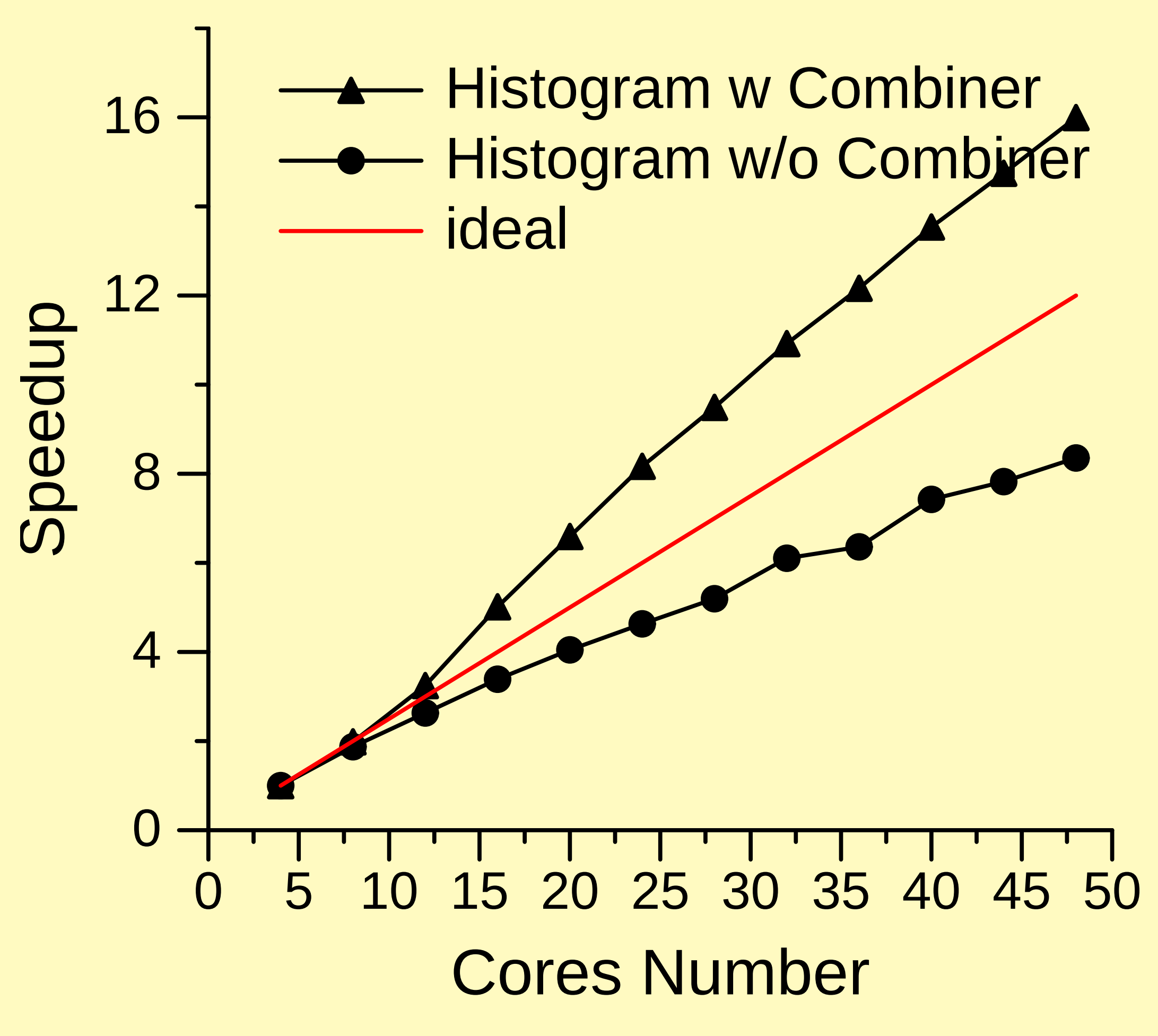
We implement a seven-stage runtime system for MapReduce. The seven stages are *map*, *combine*, *partition*, *group*, *reduce*, *sort* and *merge*. The *combine* and *merge* stages are optional in typical MapReduce applications. Our constribution is the partition and group stages that is optimized for SCC architecture.

The partitioning stage requires an all-to-all exchange between cores. Data partitions generated during the map stage may be different in size. We implement a custom all-to-all exchange algorithm for the SCC to achieve scalable data partitioning. The algorithm first executes an all-to-all exchange of the intermediate partition's sizes, followed by an all-to-all exchange of the intermediate data. We implement the all-to-all exchange using pairwise exchanges. Let p be the number of available cores and $rank$ the core ID. This algorithm uses $p-1$ steps and in each step k the $rank$ core receives data from core $rank-k$ and sends data to core $rank+k$.

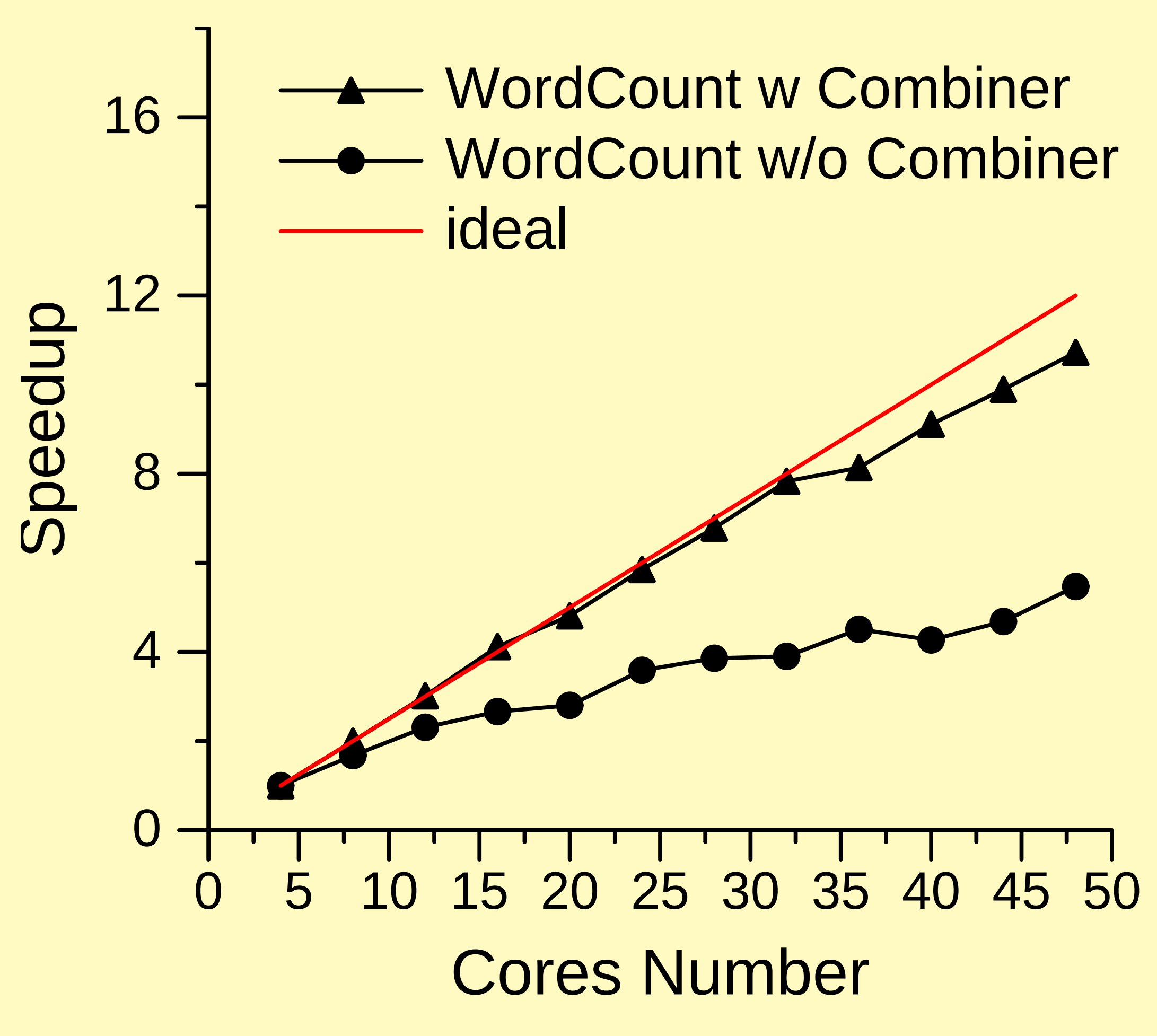
The group stage groups together all key-value pairs with the same key, taken across all intermediate data partitions. In previous works, a generic sorting scheme with a user-defined comparator was used to perform grouping. We replace this scheme with a radix sorting algorithm for grouping on the SCC. The quicksort algorithm employed in prior MapReduce implementations on multi-core systems has complexity $O(nlogn)$, whereas radix sort has complexity $O(kn)$ where k is the size of the key in bytes.

Results

We use Histogram and Word Count applications to evaluate MapReduce. Histogram counts the frequency of occurrences of each RGB color component in an image file. Word Count counts the number of occurrences of each word in a text file.

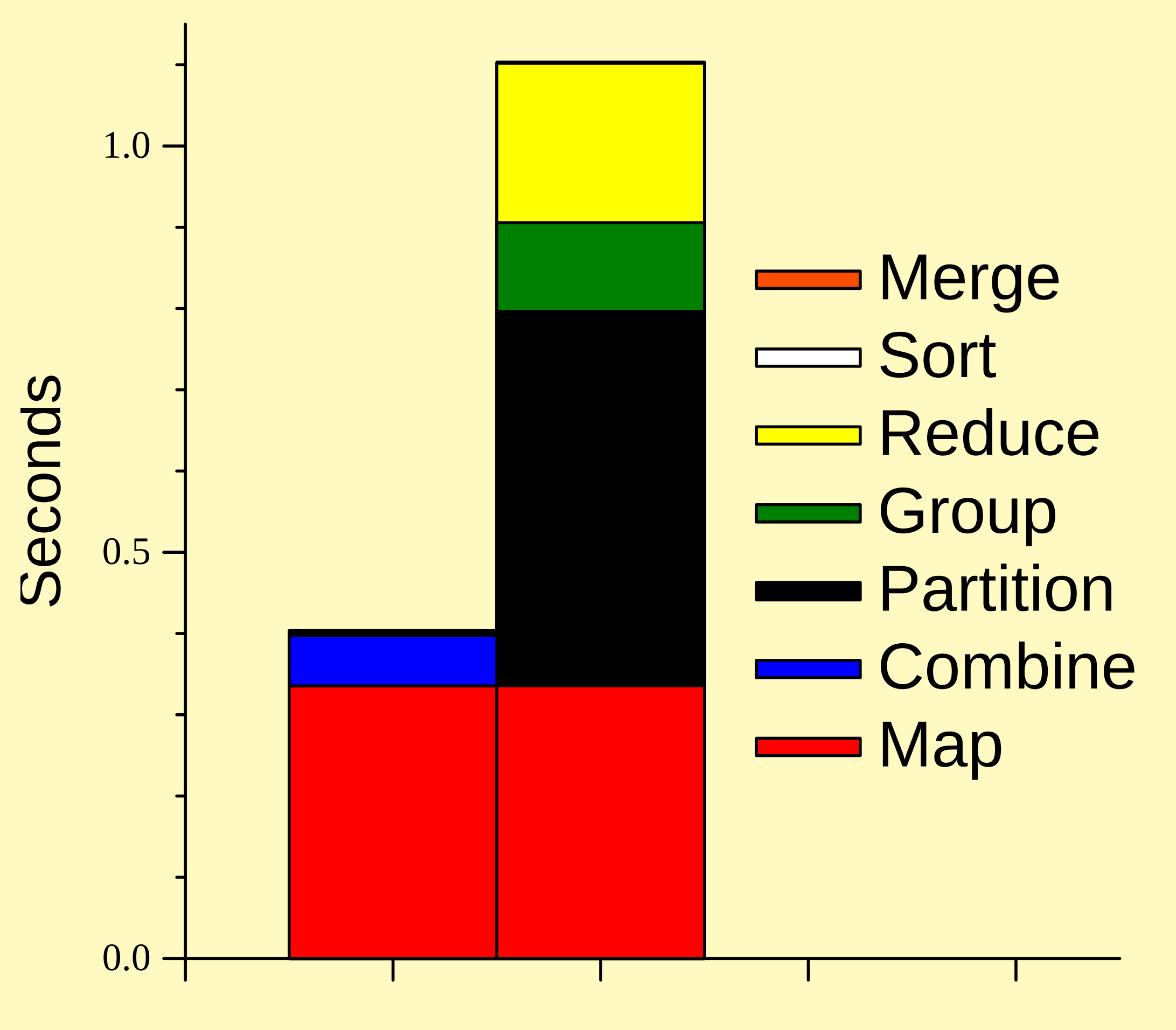


Histogram Speedup

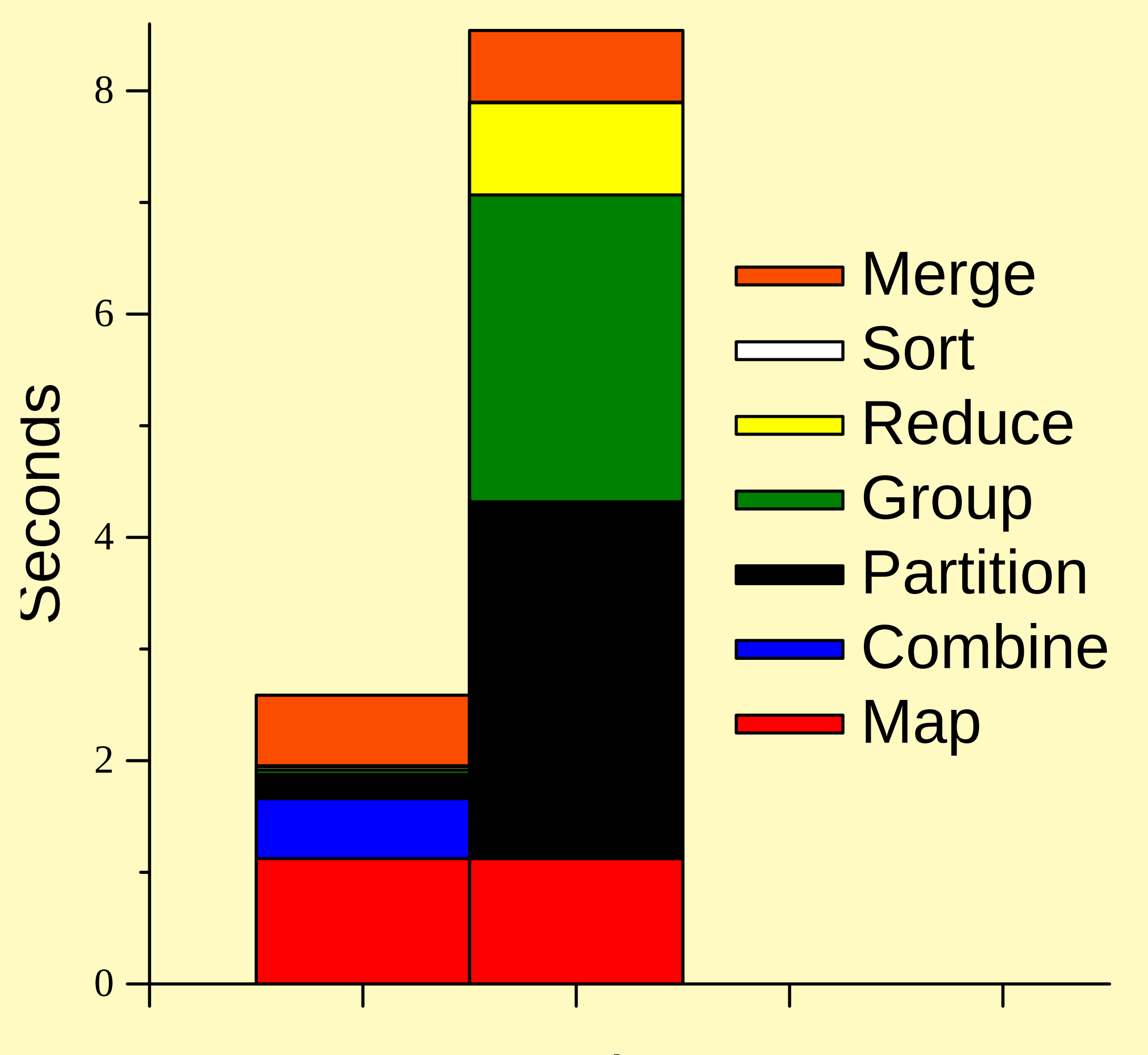


WordCount Speedup

Combiner stage reduces the intermediate buffers size. This results in improving the scalability. Superlinear speedup because complexity of the group stage decreases exponentially with the number of cores.



Histogram



WordCount

Partition stage does not scale. Combiner minimizes total partition time and group time.

Funding

The research leading to these results has received funding from the European Community's Seventh Framework. Programme [FP7/2007-2013] under the I-CORES project, grant agreement n^o 224759.