Scalable Runtime Support for Data-Intensive Applications on the Single-Chip Cloud Computer

Anastasios Papagiannis and Dimitrios S. Nikolopoulos,
FORTH-ICS

Institute of Computer Science (ICS)
Foundation for Research and Technology – Hellas (FORTH)
GR–70013, Heraklion, Crete, GREECE
{apapag,dsn}@ics.forth.gr

3rd MARC Symposium, 2011

Outline

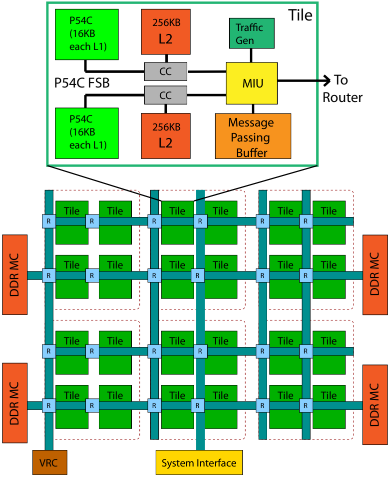
Motivation and Contributions

- We are on the transition from multi-core processors to many-core processors
- Programmers have to deal with:
  - many cores
  - many forms of implicit or explicit communication
  - many forms of synchronization
  - potential lack of cache coherence
- Contributions of this work:
  - First implementation of a high-level domain-specific parallel programming model (Google's MapReduce) on a cache-based many-core processor with no cache coherence, based on explicit communication (SCC)
  - Evaluation showing that the Intel SCC supports effectively:
    - High-level programming models that hide communication, synchronization, parallelization under the hood
    - Scalable execution of data-intensive applications

Outline

Intel SCC

- Many-core processor with 24 tiles, 2 IA cores per tile
- Tiles organized in a 4×6 mesh network with 256 GB/s bisection bandwidth
- Private L1 instruction cache of 16 KB, private L1 data cache of 16 KB, private unified L2 cache of 256 KB, per core
- 16 KB message passing buffer (MPB) per tile (only on-chip memory shared between cores)

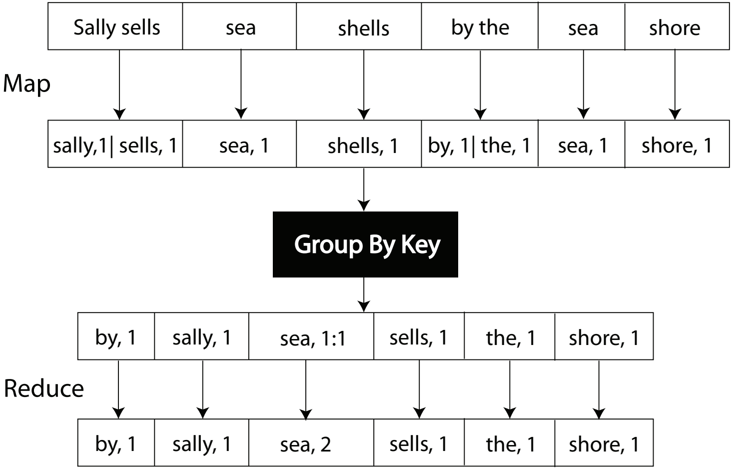


Outline

MapReduce

- A framework for large-scale data processing
  - Programming model (API) and runtime system for a variety of parallel architectures
    - Clusters, SMPs, multi-cores, GPUs, among others
  - Based of functional programming language primitives
- Used extensively in real applications
  - Indexing system, distributed grep, document clustering, machine learning, statistical machine translation
- Relies heavily on a scalable runtime system
  - Fault-tolerance, parallelization, scheduling, synchronization and communication

Example



Counting word occurrences in a set of documents

Outline

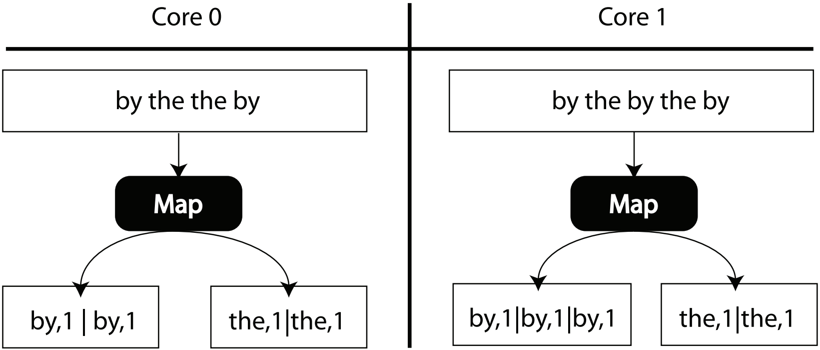
Design

Seven-stage runtime system for MapReduce:

- Map
- Combine (optional)
- Partition
- Group
- Reduce
- Sort (optional)
- Merge (optional)

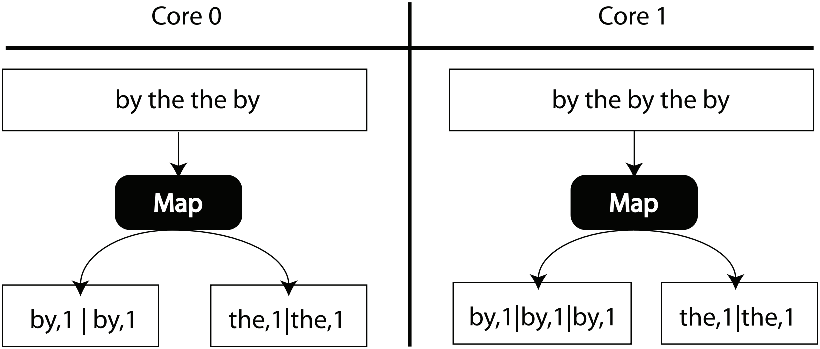
Outline

MapReduce
Map



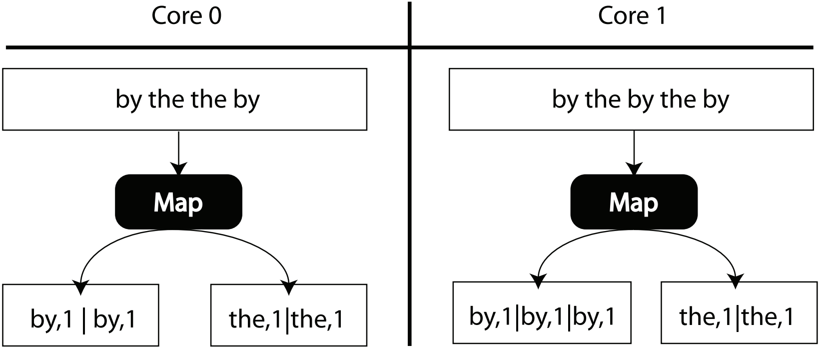
- Each core executes the user-defined map function on chunks of input data, located in local memory
- Map function emits one or more intermediate key-value pairs

MapReduce
Map



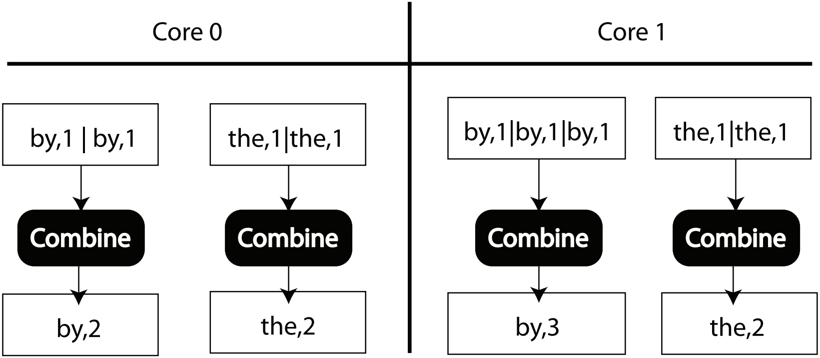
- Intermediate key-value pairs stored in a contiguous buffer
  - Runtime preallocates large chunks of memory (64 MB) for intermediate data buffers
  - More buffering space allocated on demand, if needed
  - Allocation strategy reduces memory management overhead

MapReduce
Map



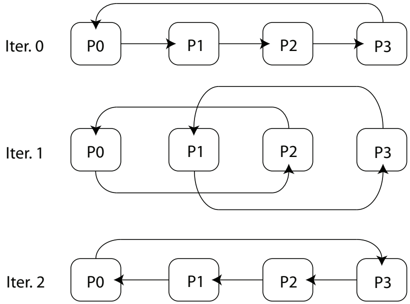
- Each core produces as many intermediate data partitions as the total number of cores

MapReduce
Combine



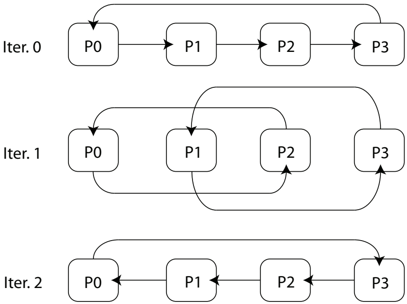
- Optional stage executed if user provides a combiner function
- Reduces locally the size of each partition produced during the map stage

MapReduce
Partition



- Requires an all-to-all exchange between cores
- Data partitions generated during the map stage may be different in size
  - First execute an all-to-all exchange of the sizes of each partition
  - Knowing the size of each partition, execute a second all-to-all exchange with the actual data

MapReduce
Partition

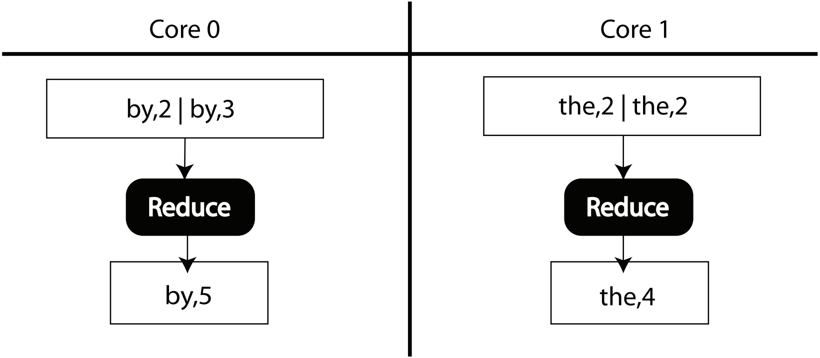


- Let p be the number of available cores and *rank* the core ID. This algorithm uses $p-1$ steps and in each step k, core *rank* receives data from core $rank-k$ and sends data to core $rank+k$.

MapReduce
Group

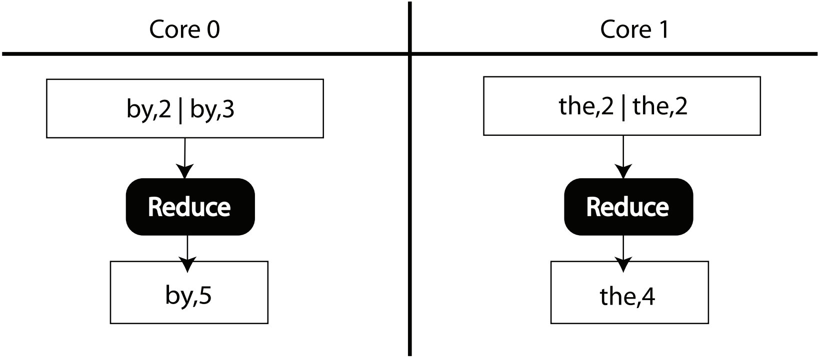
- Groups all (key, value) pairs with the same key
- Use radix sort instead of conventional merge sort
  - Radix sort sorts strings of bytes and can not use a user-defined comparator for sorting
  - If radix sort does not sort native application type, sort the output using a user-specified compare function
  - Conventional sorting algorithms have complexity $O(nlogn)$. Radix sort has complexity $O(kn)$ where k is the size of the key in bytes.

MapReduce
Reduce



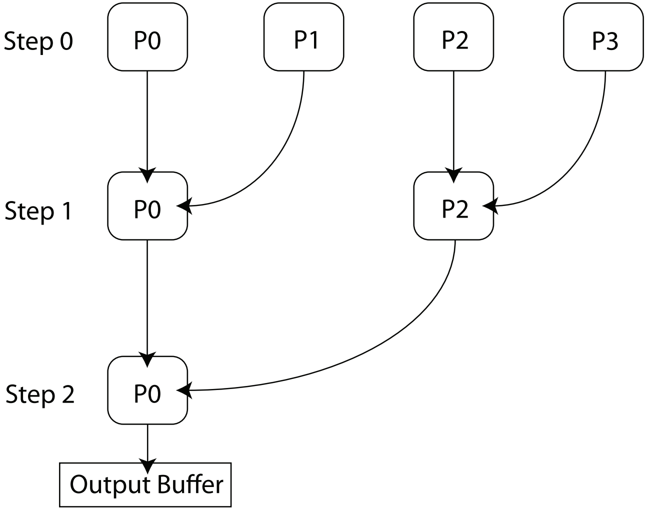
- Group stage exports distinct keys with a list of corresponding values
- Reduce stage executes user-defined aggregation function on each key-list(of values) pair

MapReduce
Reduce



- Reduce function emits one or more output key-value pairs
  - Total output size known prior to reduction, therefore output buffer is preallocated
  - Minimizes memory management overhead

MapReduce
Sort and Merge



Outline

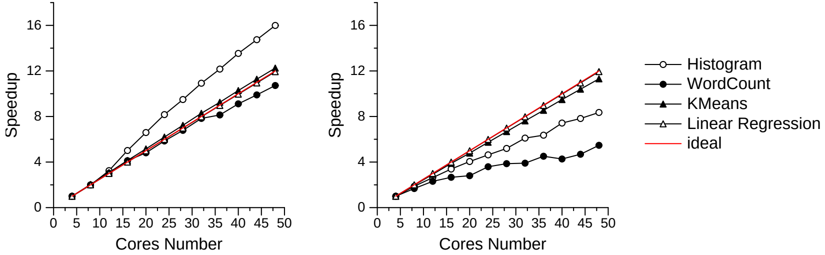
Benchmarks

- Histogram (partition-dominated) counts the frequency of occurrences of each RGB color component in an image file
- Word Count (partition-dominated) counts the number of occurrences of each word in a text file
- Kmeans (map-dominated) creates clusters from a set of data points
- Linear Regression (map-dominated) computes a line of best fit for a set of points, given their 2D coordinates

Configuration:

- Tiles run at 533MHz
- Mesh interconnect runs at 800MHz
- DRAM runs at 800MHz

Outline

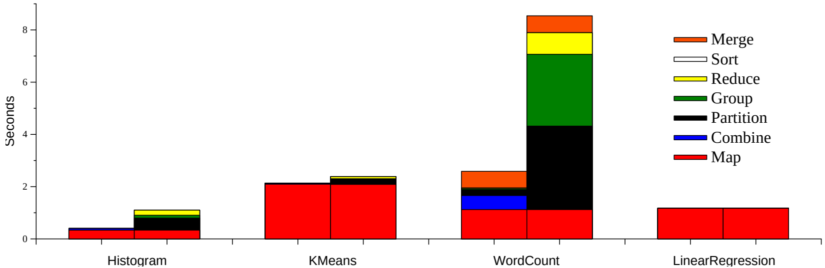


Speedup with Combiner

Speedup without Combiner

- Combiner function improves scalability
  - Kmeans and Linear Regression are map-dominated benchmarks
- Superlinear speedup because complexity of the group stage decreases exponentially with the number of cores

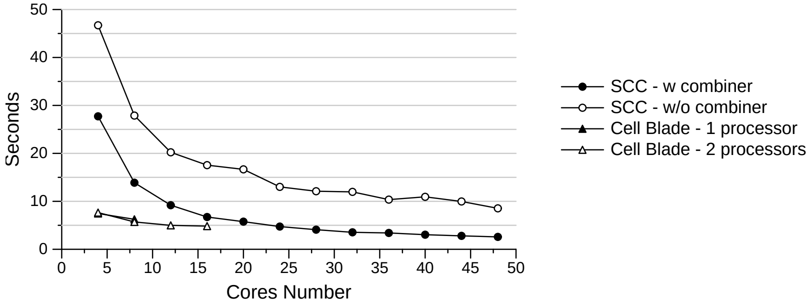
Outline



Left bars with combiner, right without combiner

- Using a combiner function reduces execution time
  - Partition stage does not scale
  - Combiner minimizes total partition time and group time

Outline



- QS22 Blade consists of 2 Cell BE Processors at 3.2 GHz
- Each processor has 8 SPEs (accelerators)
- WordCount benchmark with 60MB input size
- Single-SCC nodes outperforms dual-Cell blade by up to 1.87×

Related Work

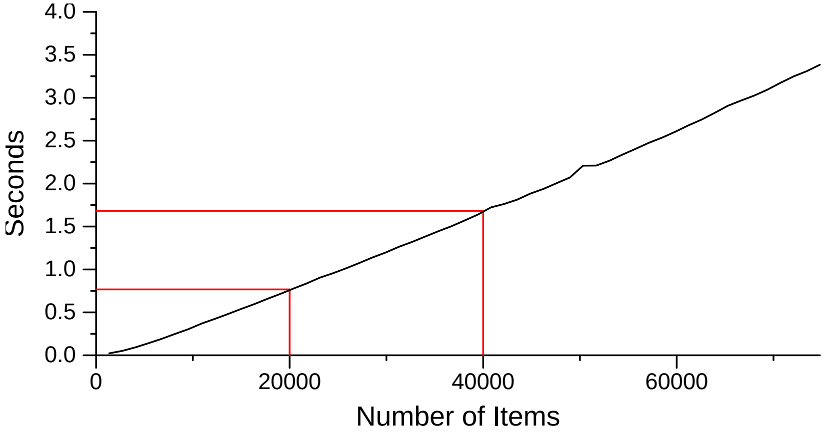
- Other ports of MapReduce on clusters, SMPs, multicores and GPUs (HPCA07,PACT08,IISWC09,ICPP10)
- Shared-memory ports based on shared data structures in cache-coherent address space
  - SCC port based on scalable exchange algorithms, while utilizing caches for fast message exchanges
- Distributed-memory ports based on generic sorting algorithms
  - SCC port based on combiner and radix sort algorithm

Conclusions

- Our implementation of MapReduce on the Intel SCC demonstrates:
  - Feasibility of implementing high-level, domain-specific parallel programming models that hide explicit communication
  - SCC chip scalability when using optimized chip-specific global communication algorithms
  - Good adaptivity to diverge workloads: map-dominated, partition-dominated

Thank you!

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.


Seconds
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
20000
40000
60000
Number of Items