HPI Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

Parallel Data Processing

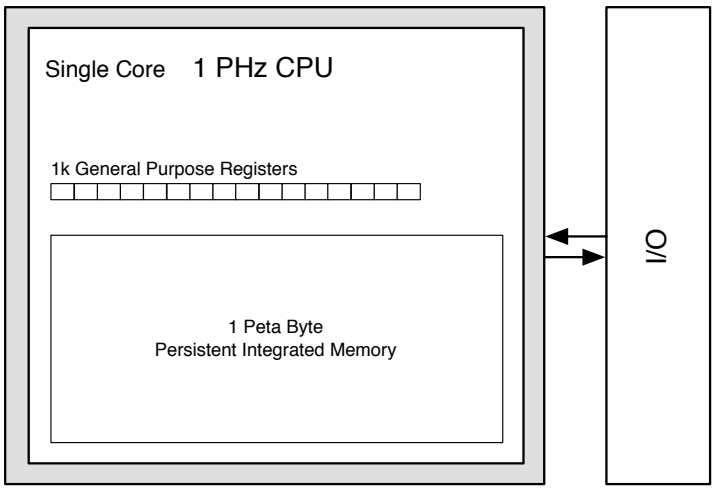
Prof. Hasso Plattner

Motivation: Why parallelization?

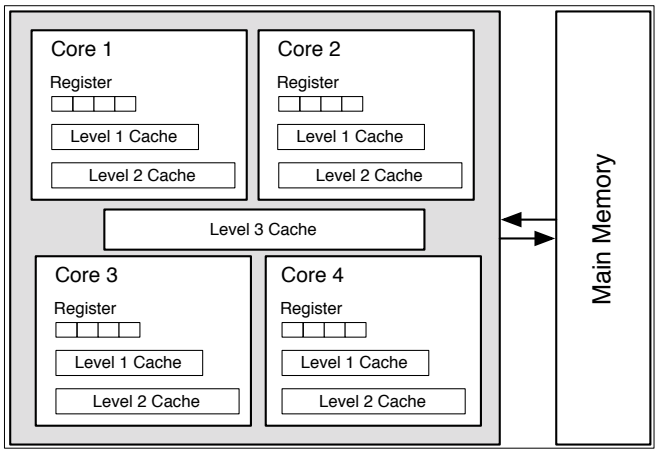
HPI Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

Parallel Hardware

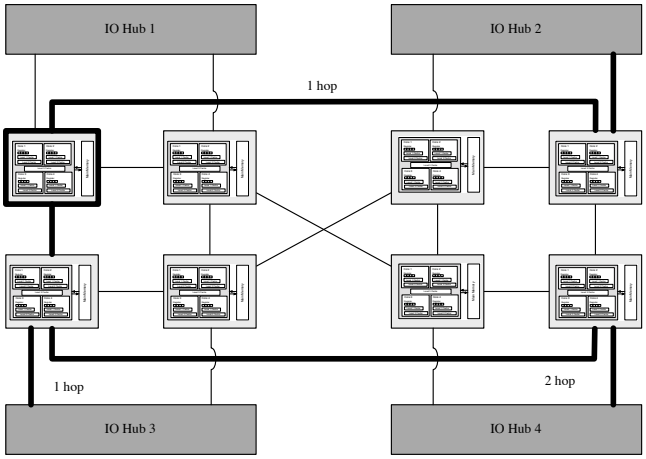
The Ideal Hardware?



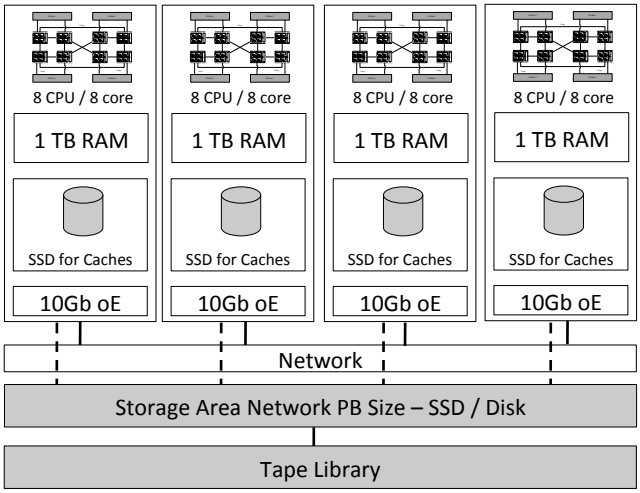
The Reality: A Multi-Core Processor



A Server with Multiple Processors



A System with Multiple Servers



Observations

- Parallelism is inherent for today's computer architecture
- Degree of parallelism in future computer systems will increase
- Parallelism needs to be leveraged in all layers
  - Multi-Core
  - Main Memory – NUMA
  - Parallel Systems

Amdahl's Law

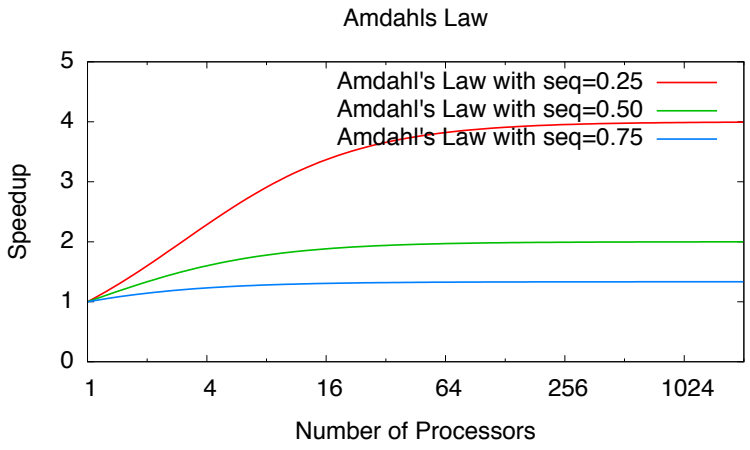
How can we determine the effect of parallelizing an algorithm?

- The runtime is bound by the longest sequential fraction (*seq*) of the algorithm. For an unlimited number of processors, speedup can be calculated as: $speedup = \frac{1}{seq}, \quad seq = (1 - P)$
- To compute the speedup for a given number of processors, the following equation can be used:

$$speedup = \frac{1}{S + \frac{P}{N}} = \frac{1}{(1-P) + \frac{P}{N}}$$

P: Parallel Fraction
S: Sequential Fraction
N: Number of processors

Amdahl's Law



Parallel Programming Models

Techniques to program parallel hardware:

- Shared memory/threads
- Message passing
- Data parallelism
- Combinations (hybrid)

Shared Memory

- Concurrent tasks share common (logical) address space (does not imply physical shared memory)
- No explicit communication required
- Locks/semaphores are required to coordinate access
- Challenges:
  - Data locality is hard to manage (which data belongs to which process)
  - Scaling

Threads

- An OS process can start multiple threads
- Each thread has local data, but shares the resources of the parent OS process
- Threads communicate through global memory
- Threads are often associated with shared memory programming
- Require synchronization, e.g., locks/semaphores

- Example: POSIX Threads

Message Passing

- Each concurrent task has own local memory
- Tasks can reside on one or on different machines
- State is kept and exchanged via message
- Tasks communicate and transfer data by sending/receiving messages

- Example: OpenMPI

Data Parallel

- Data resides in shared data structure (array, table, cube, ...)
- Concurrent tasks work independently on data partitions
- All tasks perform the same operation
- Sometimes, merge is required to create final result
- Task scheduling is done by execution framework

- Example: OpenMP ParallelFor, Map/Reduce

Parallelization with MapReduce

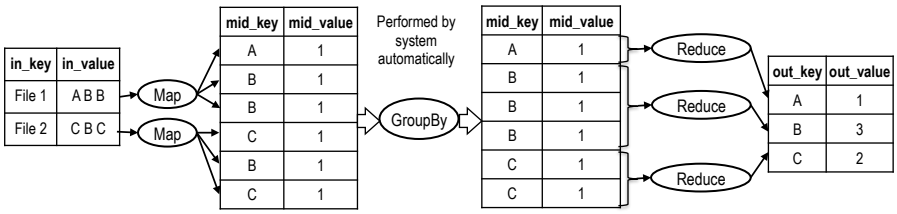
- MapReduce is a programming paradigm for shared-nothing cluster
  - Developed by Google/Yahoo to analyze large (petabyte) data-sets
  - Allows for automatic parallelization and linear scaling of MapReduce programs
- In a narrow sense, MapReduce describes a programming model based on a **map** and a **reduce** function (both well known in functional programming):

  map(*key1, value1*) → list of <key2, *value2*>
  reduce(*key2*, list of <value2>) → list of <*key3, value3*>

  map and reduce functions process <*key, value*> pairs independently and thus can be parallelized easily.
- In a wider sense, MapReduce describes an execution framework for distributing map and reduce tasks in a shared-nothing cluster

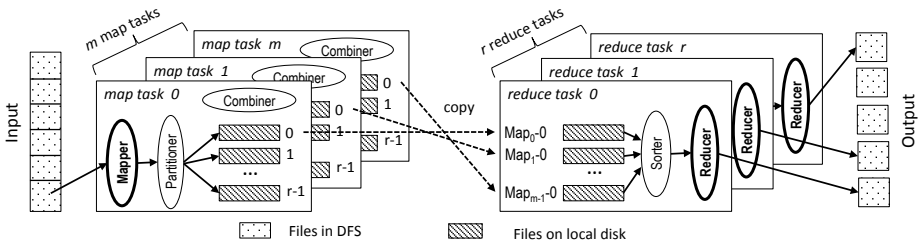
MapReduce Programming Model

Example: Word Count



1. Input is a list of key value pairs (e.g. keys="*file names*", values="*file content*")
2. A map function produces zero or more <*mid_key*, *mid_value*> pairs
3. The system performs a group-by operation on *mid_key*
4. A reduce function processes the list of *mid_values* belonding to a *mid_key* and produces a list of <*out_key*, *out_value*> pairs, here aggregate over values

MapReduce Execution Engine



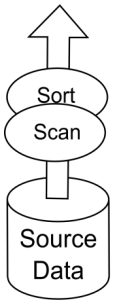
- Engine schedules map and reduce tasks, i.e. the execution of the map or reduce function on a <*key, value*> pair
- Programmer can focus on algorithm and has to implement only a map and reduce function
- The execution engine takes care of:
  - Task distribution: takes co-location of data into account
  - Shipping data between nodes: communication happens via disk!

Drawbacks and Problems

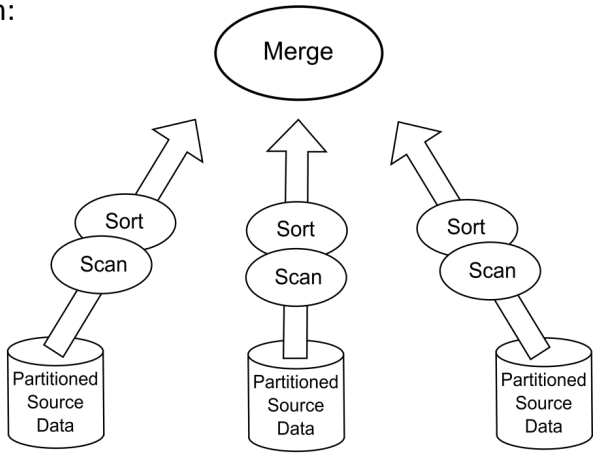
- Not all problems can be implemented efficiently with map and reduce functions (e.g. iterative algorithms)
- Batch-oriented execution model, thus not suited for interactive data analysis
- Communication via files is great for fault-tolerance, but inefficient for small data sets
- Good performance is only achieved if additional components e.g. *combiner*, *partitioner*, and *sorter* are also tuned by the programmer → programmer cannot focus on algorithm only

Parallelism in In-Memory Databases

Types of Parallelism:

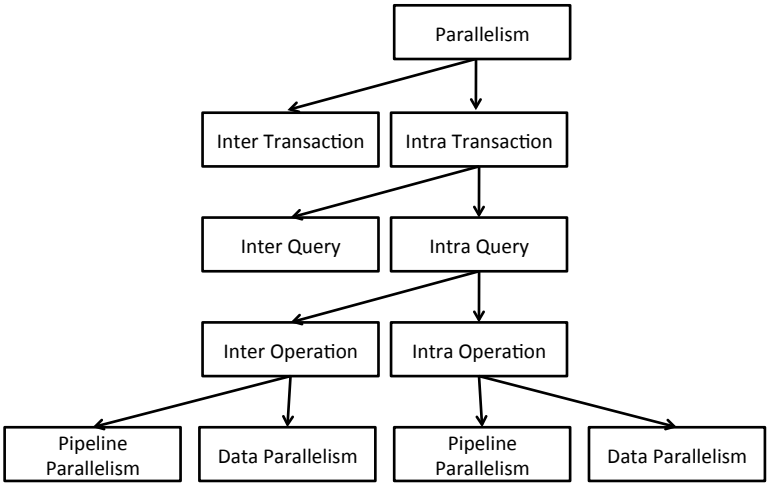


Pipeline Parallelism



Data Parallelism

Approaches to Parallelism

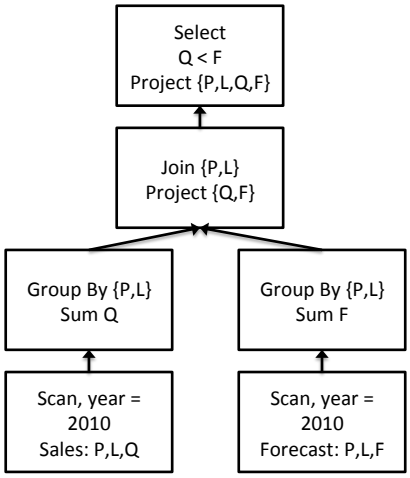


Example (Single Blade)

- Table “Sales”
  Columns: Product P, Location L, Quantity Q, Year Y
- Table “Forecast”
  Columns: Product P, Location L, Forecast F, Year Y
- Query: “Which product at which location had lower quantity forecasted overall sales in 2010?”

```
SELECT Sales.P, Sales.L, SUM(Sales.Q) AS QTY, SUM(Forecast.F) AS FCST
FROM Sales, Forecast
WHERE    Sales.P  = Forecast.P AND
         Sales.L = Forecast.L AND
         Sales.Y = 2010 AND
         Forecast.Y = 2010
GROUP BY Sales.P, Sales.L
HAVING QTY < FCST
```


Inter-Operator Parallelism (One Blade)

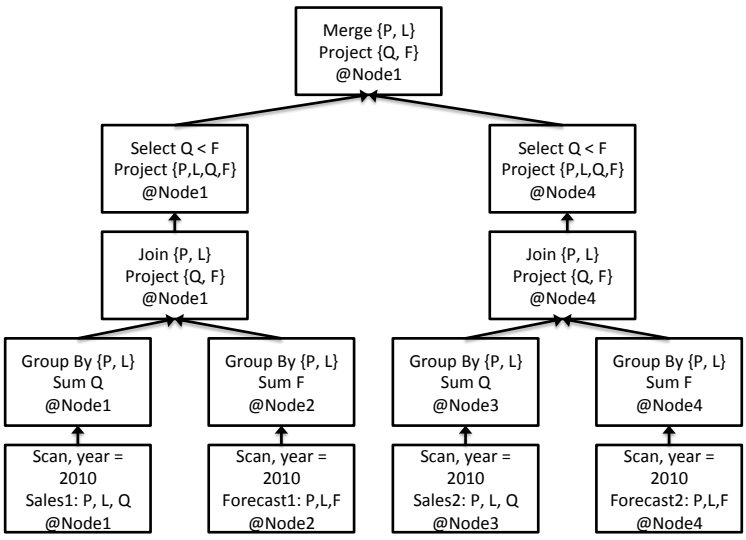


Example (Four Blades)

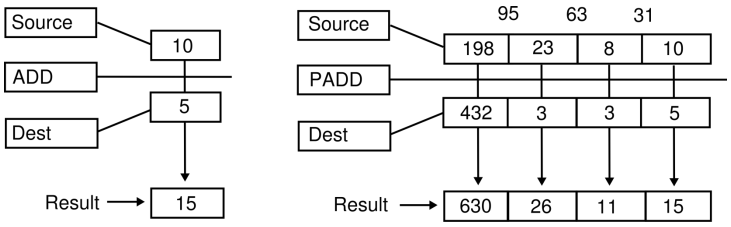
- Table "Sales"
  Columns: Product P, Location L, Quantity Q, Year Y
  Split into two parts: "Sales1" and "Sales 2" based on P and L
  "Sales 1" stored at "node1"
  "Sales 2" stored at "node3"
- Table "Forecast"
  Columns: Product P, Location L, Forecast F, Year Y
  Split into two parts: "Forecast1" and "Forecast2" based on P and L
  "Forecast1" stored at "node2"
  "Forecast2" stored at "node4"
- Query (remains the same; distribution transparent to the user):
  "Which product at which location had lower than forecasted overall sales in 2010?"

```
SELECT Sales.P, Sales.L, SUM(Sales.Q) AS QTY, SUM(Forecast.F) AS FCST
FROM Sales, Forecast
WHERE Sales.P  = Forecast.P AND Sales.L = Forecast.L
    AND Sales.Y = 2010 AND Forecast.Y = 2010
GROUP BY Sales.P, Sales.L
HAVING QTY < FCST
```


Inter-Operator Parallelism (Four Blades)



SIMD
Multiple Data Operations per Cycle



Scalar Add

Parallel Add
(Low-Level Parallel Column Access)

Key Observations

- Algorithm
  - 2-stage pipeline (pipeline parallelism)
  - Programming Model: shared memory/threads, data parallel
- Synchronization: every thread writes into its private data structure → no synchronization required for that
- Data skew handled by small input work package size (compared to fact table size)
- Cache-awareness due to fixed size of local hash tables
- Main-memory consumption bounded by buffer size
- Number of threads can be adjusted
- Similar algorithm for parallel join computation