Fork-join

Técnicas de Programación Concurrente

Fork-join

Paper de **1963**

"... in an internal **merge sort** all strings in any pass may be created at the same time."

En honor al paper original, implementemos un merge sort concurrente

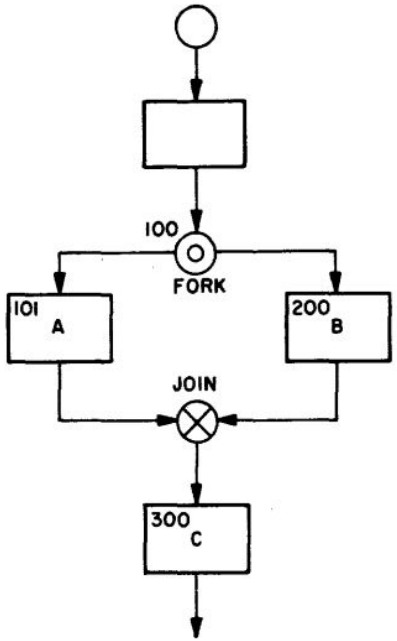
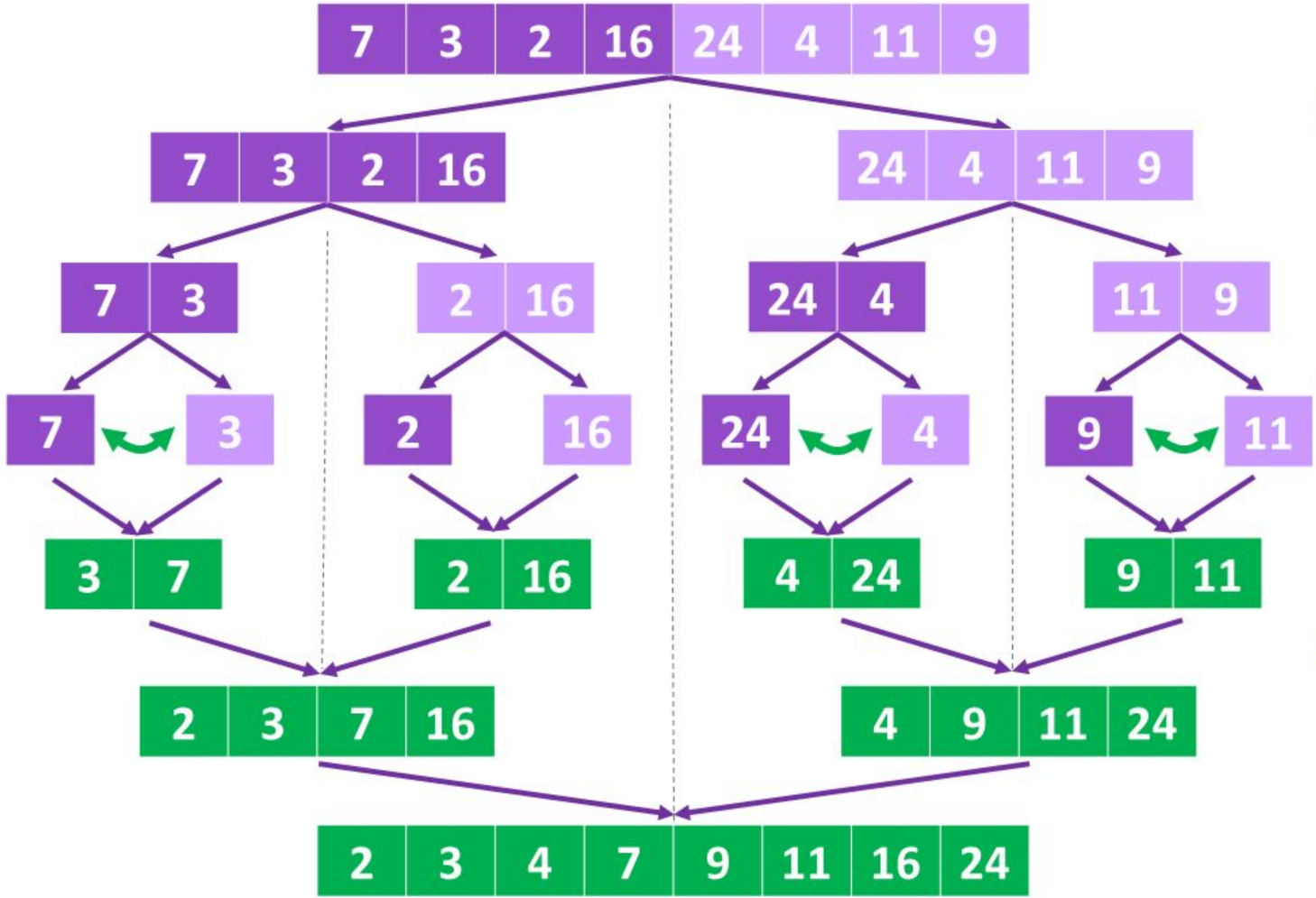


Figure 1. Conventions for drawing fork and join.

Melvin Conway

Merge Sort



Step 1:
Split sub-lists in two until you reach pair of values.

Step 3:
Sort/swap pair of values if needed.

Step 4:
Merge and sort sub-lists and repeat process till you merge to the full list.

MapReduce

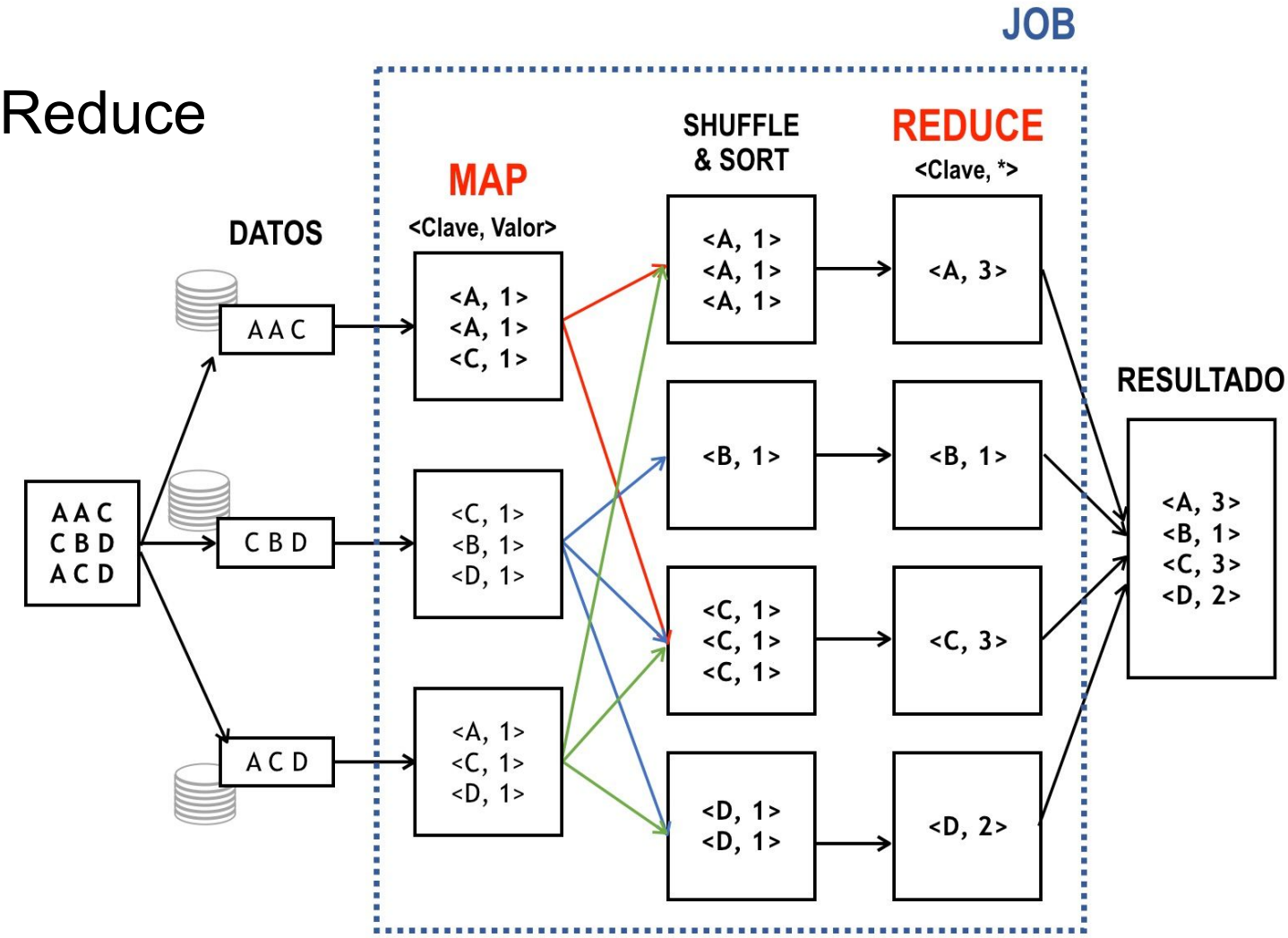
Paper del año 2004

MapReduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key.

Muchas implementaciones, Hadoop siendo la primera open source

"Hijos" más modernos incluyen a Spark

MapReduce


JOB
DATOS
MAP
<Clave, Valor>
SHUFFLE
& SORT
REDUCE
<Clave, *>
RESULTADO
A A C
C B D
A C D
A A C
C B D
A C D
<A, 1>
<A, 1>
<C, 1>
<C, 1>
<B, 1>
<D, 1>
<A, 1>
<C, 1>
<D, 1>
<A, 1>
<A, 1>
<A, 1>
<B, 1>
<C, 1>
<C, 1>
<C, 1>
<D, 1>
<D, 1>
<A, 3>
<B, 1>
<C, 3>
<D, 2>
<A, 3>
<B, 1>
<C, 3>
<D, 2>

Word count

Word count es el "Hello World" en el mundo del map reduce

- Map
  - Tomar de potencialmente varios archivos de texto, las lineas
  - De las líneas, las palabras
  - De las palabras sus frecuencias locales
- Reduce
  - Combinar las frecuencias locales en una única tabla con todas las frecuencias

Implementemos el word count!

Ryon

- Rayon is a data-parallelism library for Rust. It is extremely lightweight and makes it easy to convert a sequential computation into a parallel one. It also guarantees data-race freedom.
- Rayon makes it drop-dead simple to convert sequential iterators into parallel ones: usually, you just change your `foo.iter()` call into `foo.par_iter()`, and Rayon does the rest

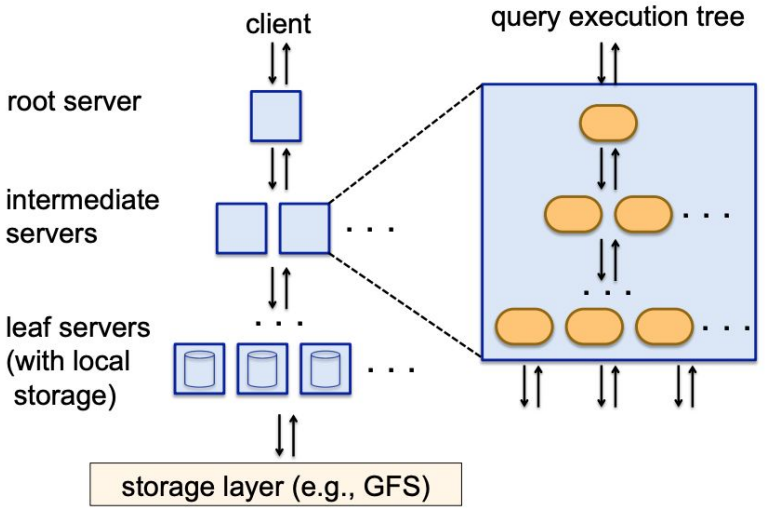
  Comparar wordcount secuencial vs paralelo usando ryon.

- Performance de mergesort con Threadpool

Dremel

Sistema de consultas en tiempo casi real, expuesto al público como BigQuery

"A query gets pushed down the tree and is rewritten at each step. The result of the query is assembled by aggregating the replies received from lower levels of the tree"



Vectorización

- Queremos realizar un cómputo "simple" sobre un conjunto grande de datos.
  - Ejemplos: procesar sonido, imágenes, video, entrenar redes neuronales, etc, etc.
- Cada dato es independiente o un pequeño subconjunto lo es.
- Fork join?

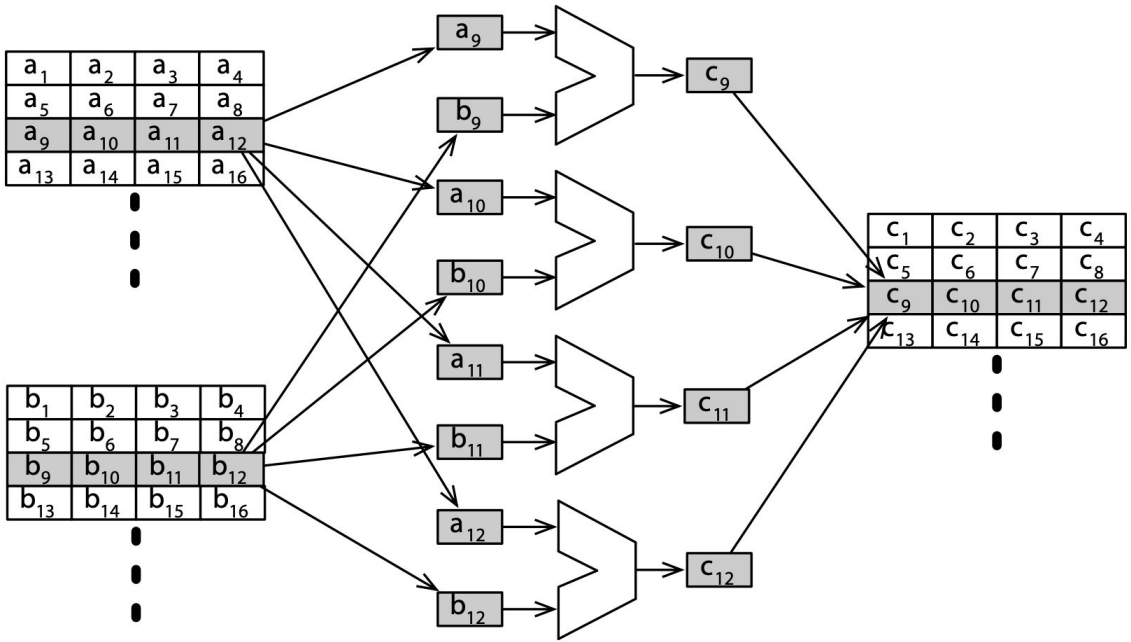
Vectorización

- Lanzar incluso una task tiene un costo en cómputo y accesos a memoria (localidad)
- Las CPU son limitadas (de hecho alguna vez fue una sola!)

Operaciones vectoriales

Al "detenerse" la ley de Moore, el aumento de transistores no se tradujo en un aumento de velocidad. En su lugar se agregaron múltiples ALUs para operar sobre los mismos registros de forma concurrente. Esto dió lugar a la introducción de juegos de instrucciones SIMD (Single Instruction Multiple Data) como ser MMX, SSE, AVX en x86 y NEON en ARM.

Operaciones Vectoriales

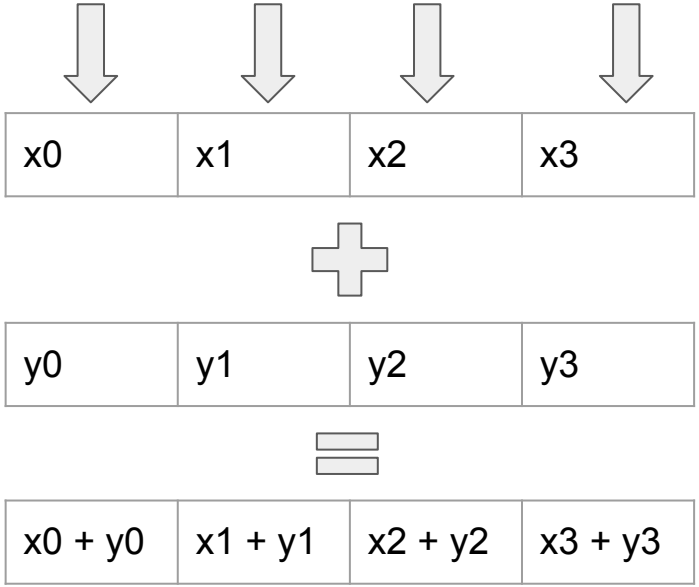


Operaciones vectoriales

- Cada variable (registro) es un vector de tamaño fijo, generalmente de entre 128 y 512 bits
- Se divide en N "lanes" (carriles) según el tamaño de los datos. Por ejemplo un registro de 512 bits puede alojar 16 enteros de 32 bits, o bien 8 flotantes de 64.

Operaciones vectoriales

- Operaciones "verticales" son entre distintos registros en el mismo lane

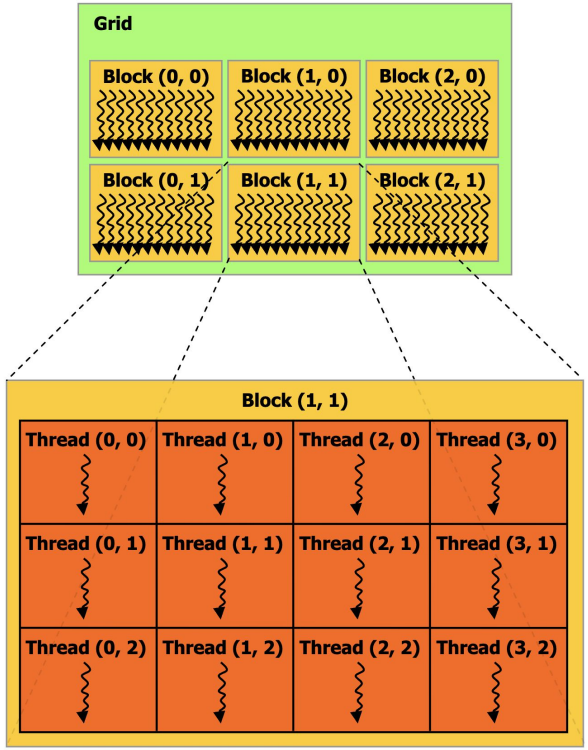


Operaciones vectoriales

- Operaciones "horizontales" reducen un vector a un escalar. Son más lentas.

CUDA

- Compute Unified Device Architecture
- Standard de facto de NVDIA para trabajar sobre GPUs
- Modela "threads" en bloques que trabajan sobre una pequeña porción de memoria independiente, y se pueden direccionar en 1D, 2D, 3D.
- Permite hasta 2048 threads por "Streaming Processor"



CUDA

Ejemplo de código de un "thread" para sumar dos vectores

```
pub unsafe fn add(a: &[f32], b: &[f32], c: *mut f32) {
    let idx = thread::index_1d() as usize;
    if idx < a.len() {
        let elem = &mut *c.add(idx);
        *elem = a[idx] + b[idx];
    }
}
```

Referencias

- A Multiprocessor System Design, M.E. Conway, 1963 - https://ia600703.us.archive.org/30/items/AMultiprocessorSystemDesignConway1963/A%20Multiprocessor%20System%20Design%20%28Conway%2C%201963%29.pdf
- MapReduce: Simplified Data Processing on Large Clusters, Jeffrey Dean and Sanjay Ghemawat, 2004 - https://static.googleusercontent.com/media/research.google.com/es//archive/mapreduce-osdi04.pdf
- Dremel: Interactive Analysis of Web-Scale Datasets S. Melnik et al., 2010 - https://storage.googleapis.com/pub-tools-public-publication-data/pdf/36632.pdf
- Seven Concurrency Models in Seven Weeks: When Threads Unravel (The Pragmatic Programmers), Paul Butcher