Bitonic Sort

OCTOBER 2014 RUD

CIRCUIT DE SPA-FRANCORCHAMPS

Definitions

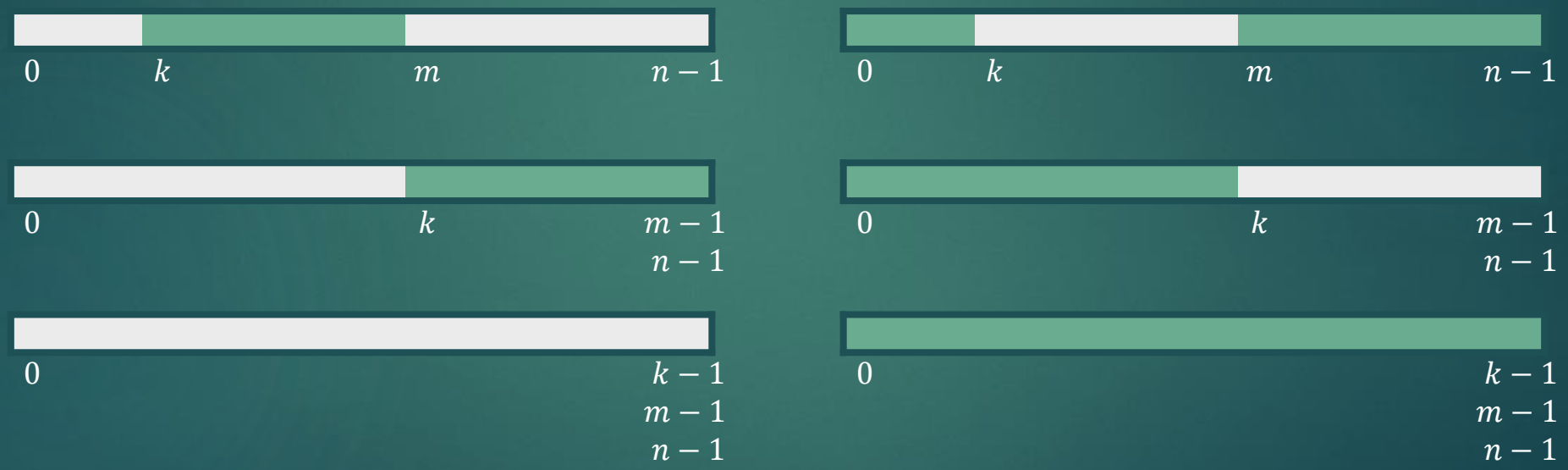
LA SOURCE

Bitonic sequence

- A sequence $a = a_0, \dots, a_{n-1}$ with $a_i \in \{0,1\}, i = 0, \dots, n-1$ is called a 0-1 sequence

- A 0-1 sequence is called bitonic if it is composed of one sequence of 0s and one sequence of 1s when both ends are connected

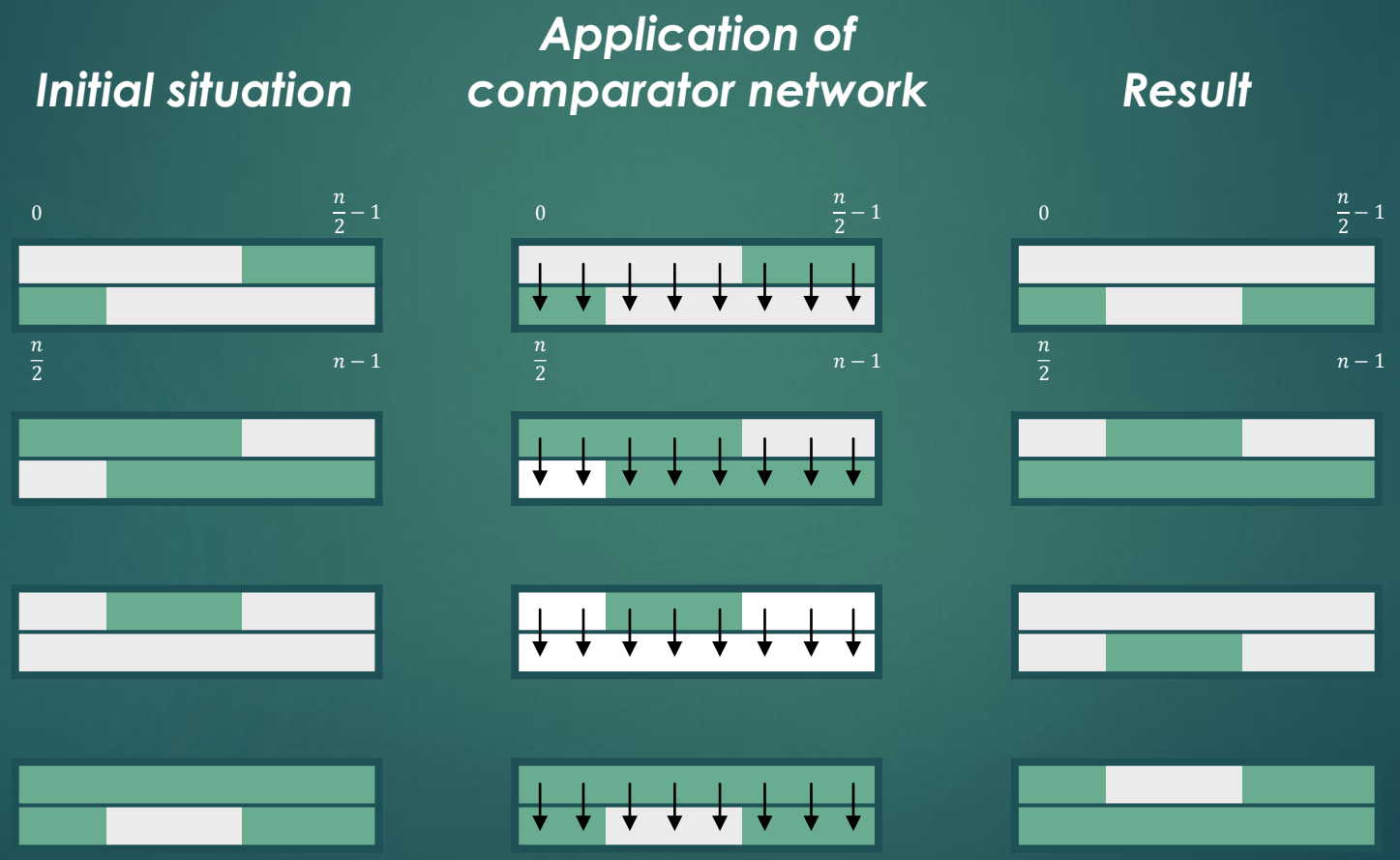
Example



Comparator network B_n

- A comparator $[i : j]$ is an operator which rearranges elements i and j of a sequence a such that $a_i = \min(a_i, a_j)$ and $a_j = \max(a_i, a_j)$
- A comparator stage is a composition of comparators such that no element is involved in more than one comparator (i.e., all comparators can be executed simultaneously)
- A comparator network is a composition of comparator stages
- B_n is defined as $B_n = \left[0 : \frac{n}{2}\right], \left[1 : \frac{n}{2} + 1\right], \ldots, \left[\frac{n}{2} - 1 : n - 1\right]$

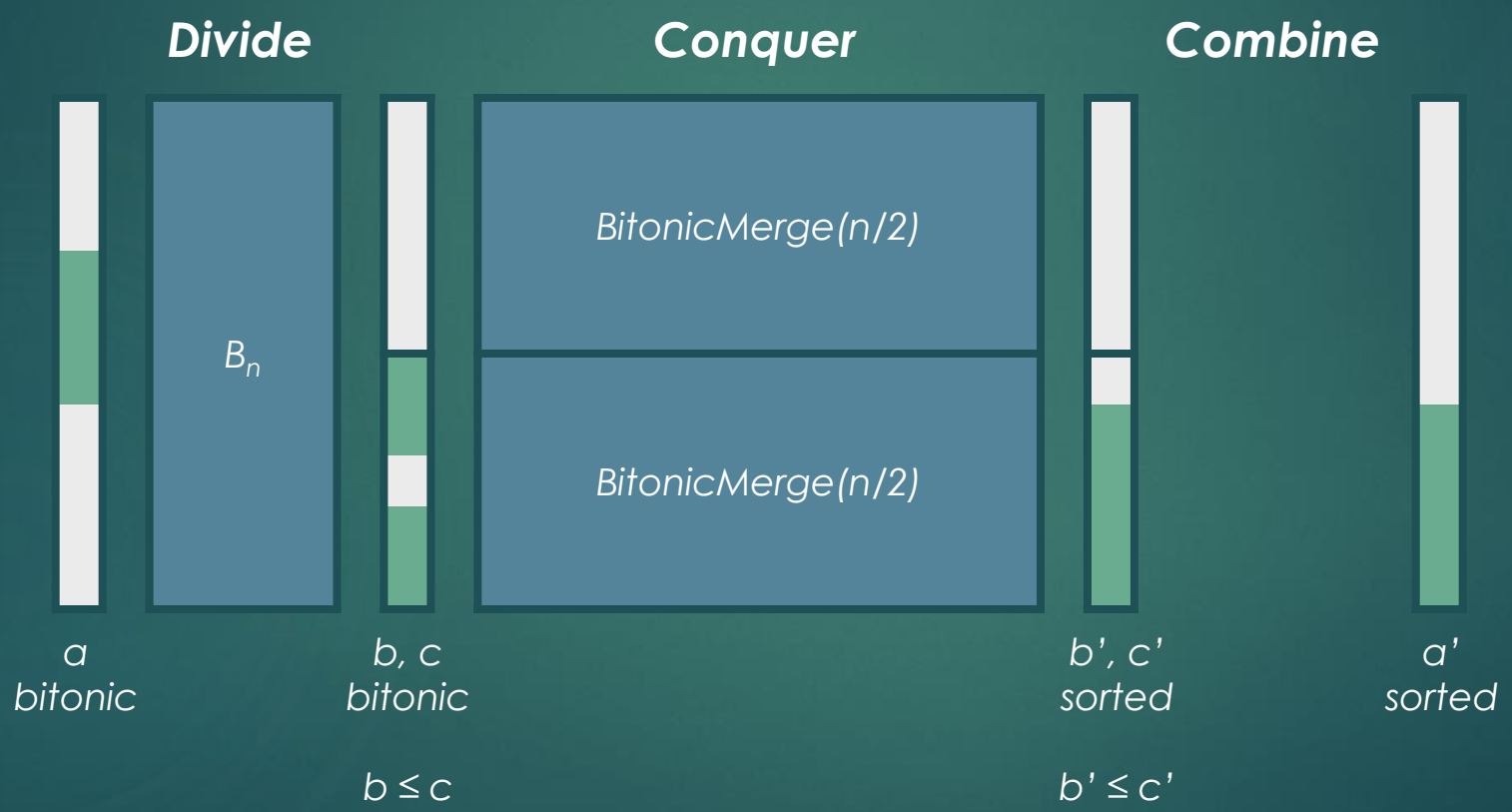
Applying bitonic sequences to B_n



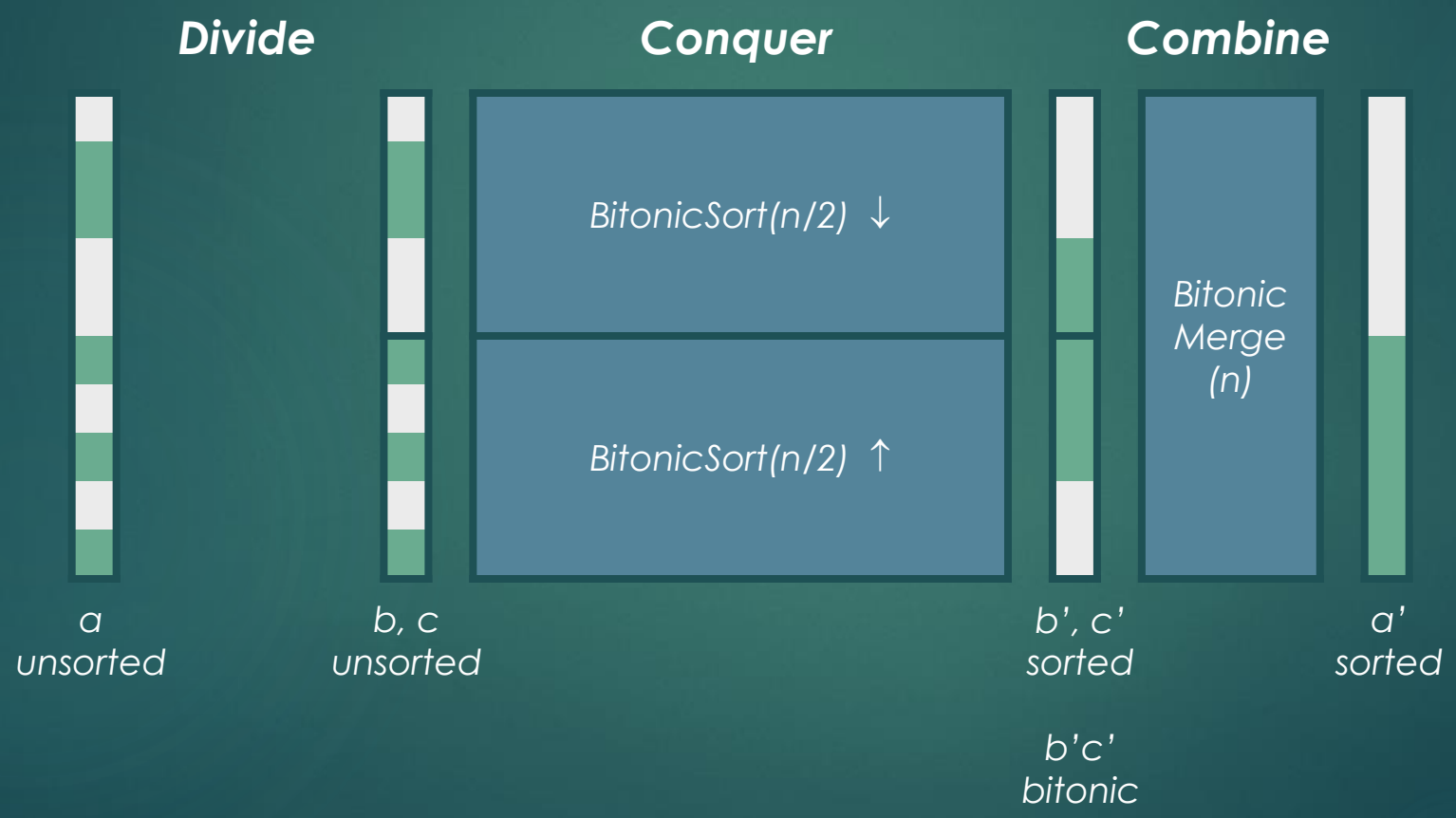
Algorithm

EAU ROUGE

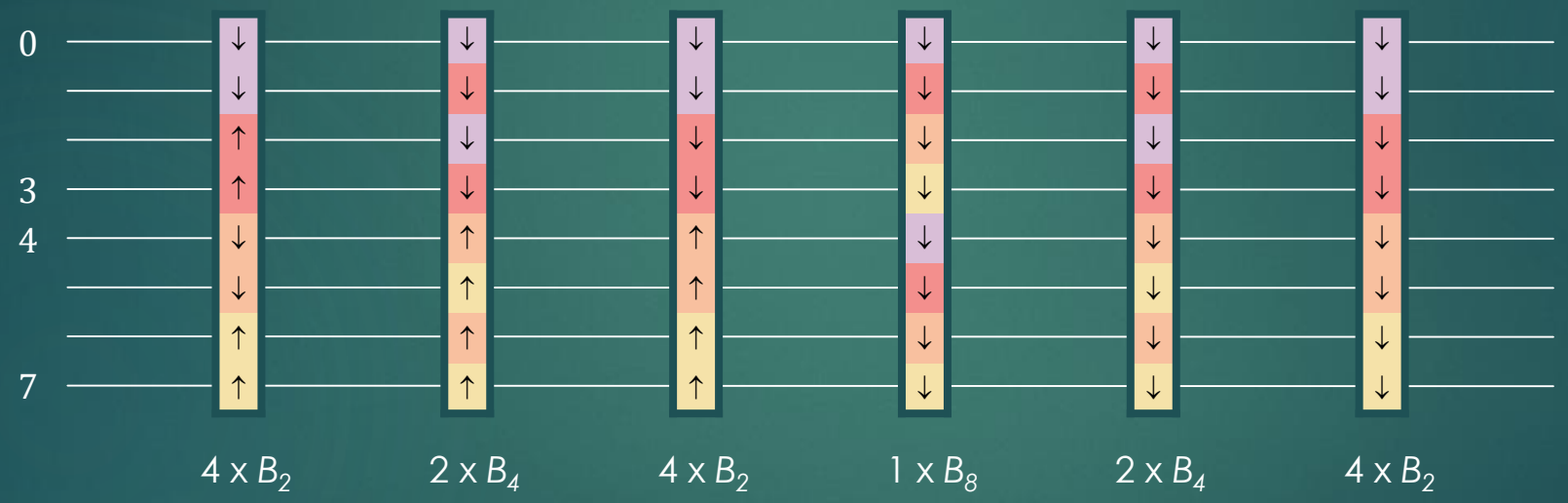
BitonicMerge(n)



BitonicSort(n)



Bitonic sorting network (n = 8)



Complexity

- $\frac{1}{2}\log n\,(\log n + 1)$ stages
- $\frac{1}{2}n$ comparators per stage
- $O(n\log^2 n)$ sequential time
- $O(\log^2 n)$ parallel time

Implementation

KEMMEL STRAIGHT

CUDA C++ implementation (1)

```
void bitonic_sort(nat* val, real* key, nat N)
{
  nat threads = N >> 1;
  nat blocks = RDIV(threads, THREADS_PER_BLOCK);

  for (nat size = 2; size <= N; size <<= 1)
    for (nat stride = size >> 1; stride > 0; stride >>= 1)
      bitonic_sort_K<<<blocks, THREADS_PER_BLOCK>>>(val, key,
threads, size, stride);
}
```

CUDA C++ implementation (2)

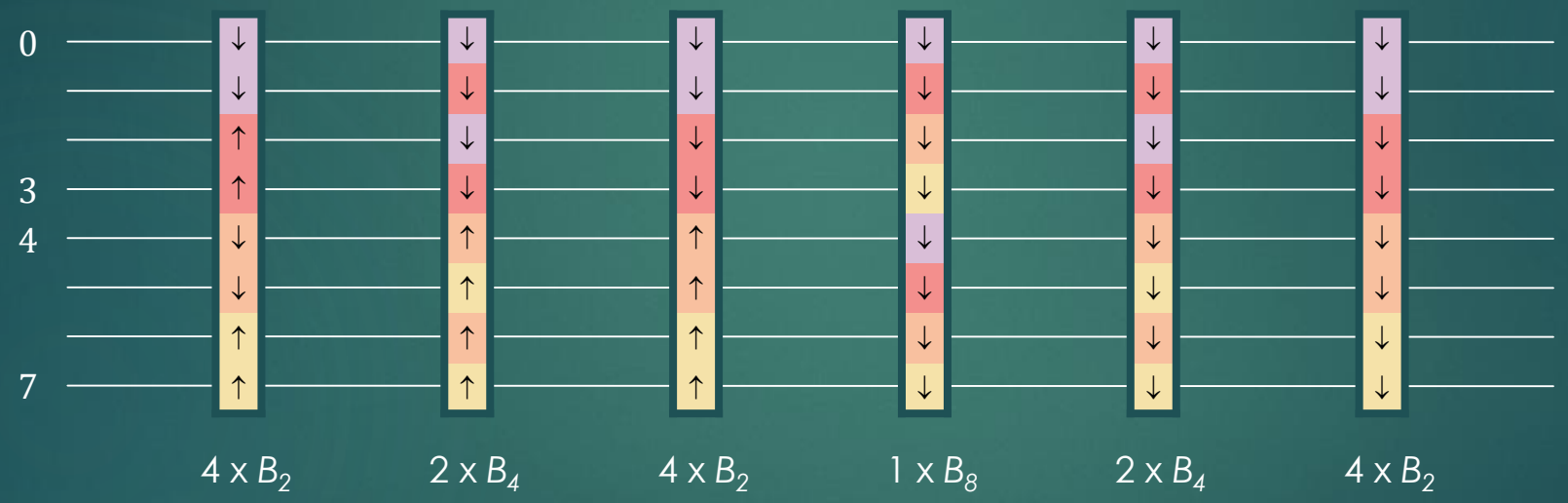
```
__global__ void bitonic_sort_K(nat* val, real* key, nat
threads, nat size, nat stride)
{
  nat x = threadIdx.x + blockIdx.x*blockDim.x;


  if (x < threads)
    COMPARATOR(x + (x & ~(stride - 1)),            // i
               x + (x & ~(stride - 1)) + stride,   // j
               x & size);                          // direction
}
```

Optimization tips

- Move loops inside kernel as soon as size is small enough to fit inside individual blocks and use `__syncthreads()` to avoid kernel launch overhead

- Start by reading all data required for subsequent stages by block into shared memory, execute stages, then write data back to global memory (also avoids uncoalesced access to global memory)

Bitonic sorting network (n = 8)



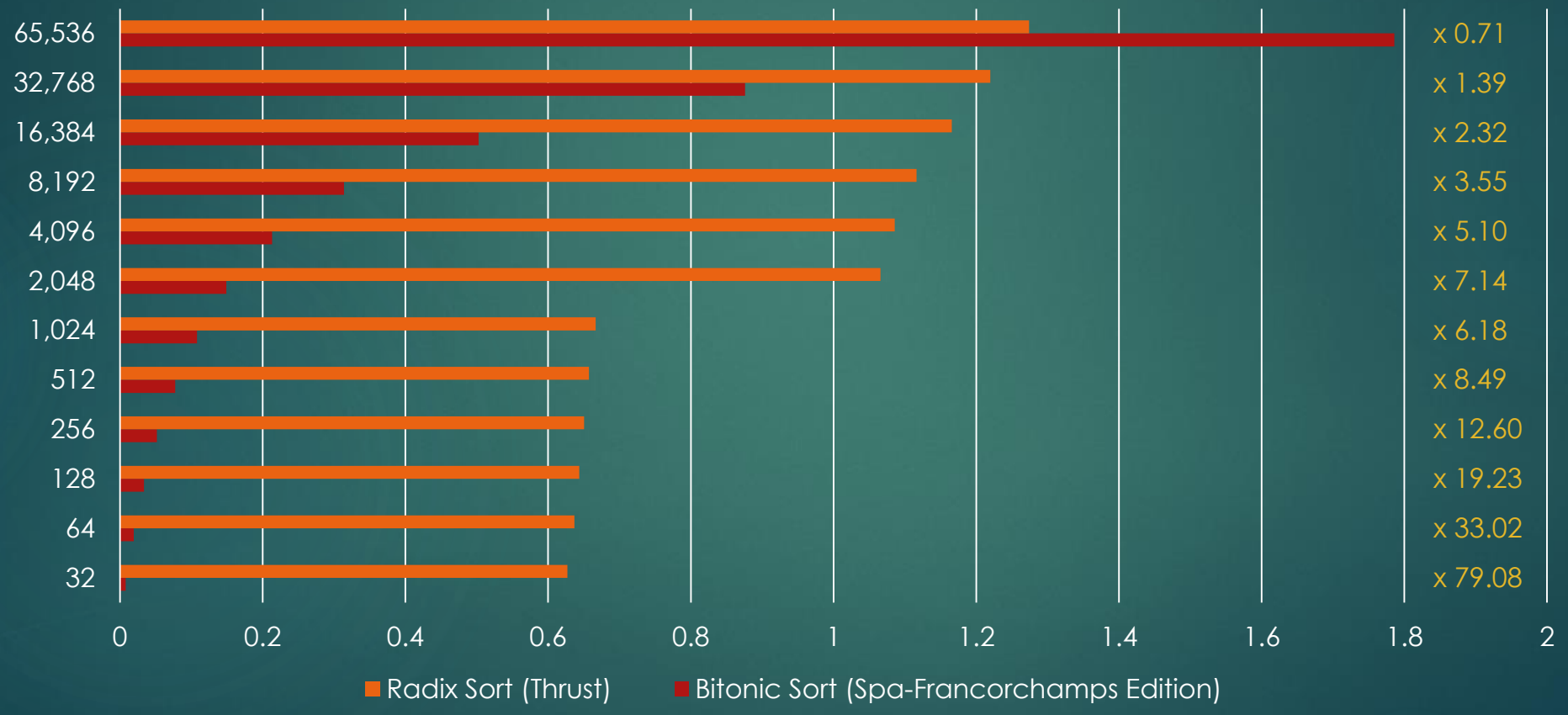
Application

POUHON

Used in a parallel cascade-correlation implementation

- Evolutionary optimization methods (genetic algorithms, cross-entropy, …) with small population size (less than 10,000)
  - GPU implementation
  - Data resides in graphics memory
  - Copying data to system memory is expensive (travels on PCIe bus)

Sorting time (ms) by array size on a GeForce GTX 460 (and speedup)



Closing Words

CHICANE

Wrap-up

- Interesting choice for small arrays
- Easy to implement in case a custom implementation is needed
- Seamless parallelization
- Free $O(\log^2 n)$ complexity with enough cores and bounded N

Perspectives

- GeForce GTX 460
  - 336 CUDA cores
  - Sublinear complexity up to 30,000 elements
- GeForce GTX TITAN Z
  - 5,760 CUDA cores
  - Sublinear complexity up to 500,000 elements ???
- Send me the card to find out ☺

References

- Sorting networks
  - http://www.iti.fh-flensburg.de/lang/algorithmen/sortieren/bitonic/bitonicen.htm
- NVIDIA CUDA Toolkit Samples
  - http://docs.nvidia.com/cuda/cuda-samples/

Thanks for listening!

AND GOODBYE!