Parallelization of sorting algorithms

Julius Plehn
Camila Roa

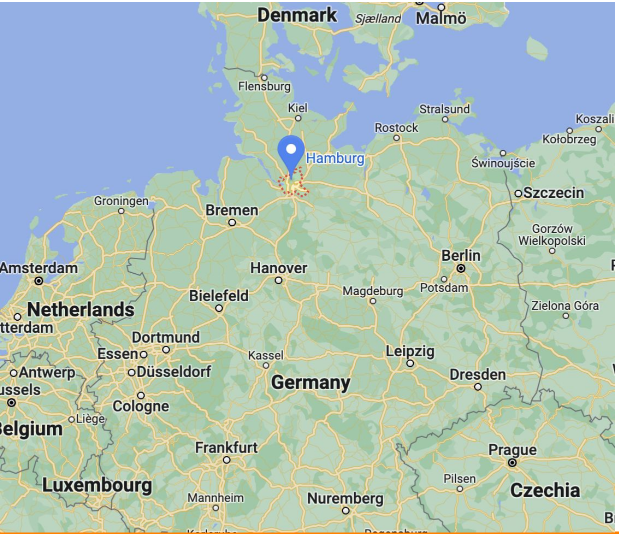
THE UNIVERSITY OF
TENNESSEE
KNOXVILLE

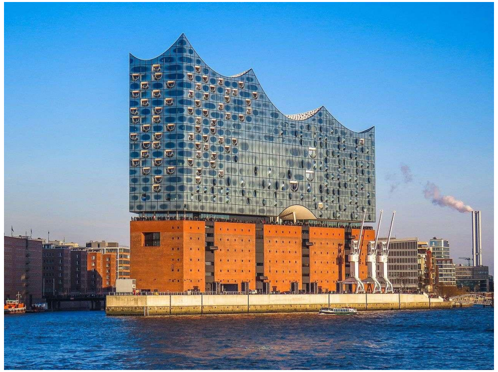
Questions

1. Why were the first implementations of radix sort considered memory inefficient?
2. What is the difference between OpenMP and MPI?
3. Why is radix sort difficult to parallelize?

Julius Plehn

- Master student, Computer Engineering, Graduate Research Assistant @ GCLab
  - Advisor: Michela Taufer
  - Research Interest: High Performance Computing
- B.S., Software Systems Development (University of Hamburg, Germany)
- M.S., Computer Science (University of Hamburg, Germany)

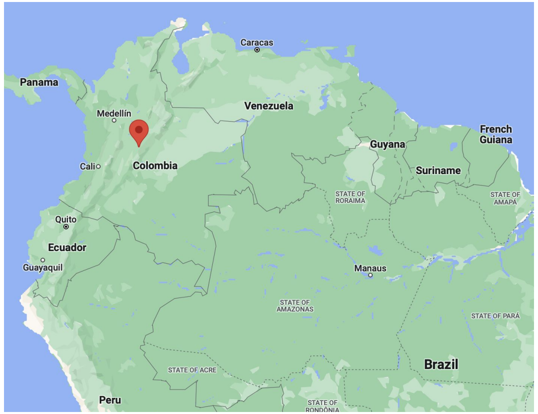




- Hiking, Traveling + Road trips, Cooking, Biking

Camila Roa

- Master student, Computer Engineering, Graduate Research Assistant @ GCLab
  - Advisor: Michela Taufer
  - Research Interests: Machine Learning, High Performance Computing
- B.S., Electronics Engineering (Pontificia Universidad Javeriana, Bogotá, Colombia)



Bogotá: Capital of Colombia
Part of the Andes: 2,640 m (8,660 ft)
Average temp: 14.5 °C (58 °F)

- Food, Traveling, Music, Tenis, Climbing, Hiking, Diving

Outline

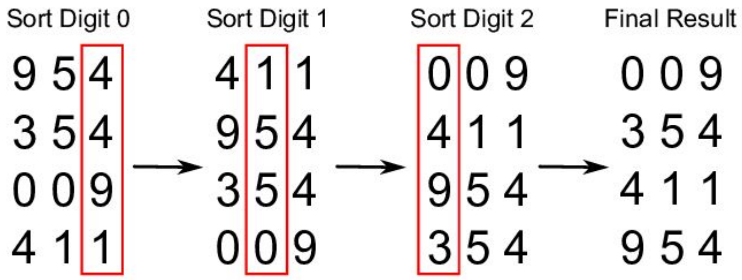
1. Overview:
   a. Brief review on Radix Sort
   b. Why is Radix Sort difficult to parallelize?
2. History
3. Algorithm:
   a. State of the art parallelization on GPU
   b. Parallelization on multi-core system (OpenMP and MPI)
4. Implementation:
   a. Our implementation of parallel Radix Sort
   b. Benchmarks
5. Applications
6. Open Issues

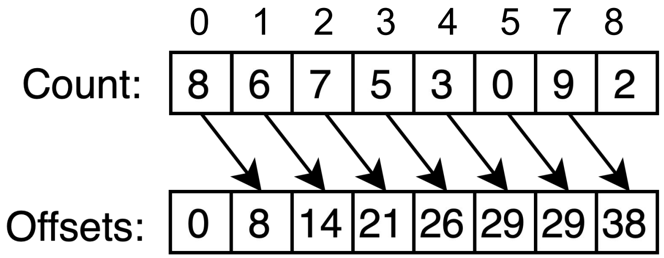
Overview: Radix sort

- Sorting takes place from LSD to HSD distributing elements into buckets
- Number of times sorting is performed depends on the radix
- Stable algorithm (required), not comparison-based sort, uses a version of counting sort
- Time complexity: O(d*n), d = # of digits, n = # of elements

Some definitions:

- Prefix sum: cumulative sum, used to compute bin-relative offsets
- GPU kernel: function that is executed in parallel (SIMD)





Tristram, Dale & Bradshaw, Karen. (2014). Identifying attributes of GPU programs for difficulty evaluation. South African Computer Journal. 53. 10.18489/sacj.v53i0.195.

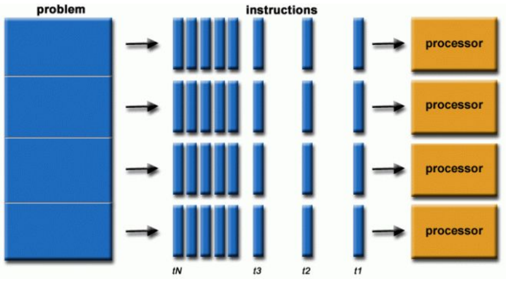
Overview: Radix sort

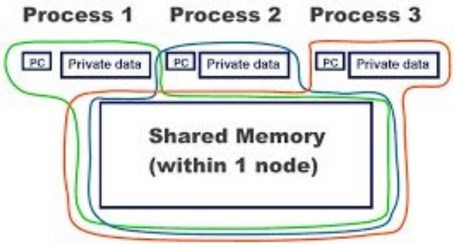
Why is Radix Sort hard to parallelize?

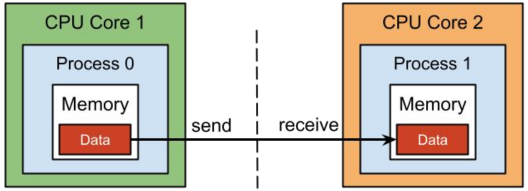
- Data dependencies: inherent to the algorithm
    - Thread cooperation
    - Ratio of computation to communication
- Irregular memory access patterns: moving records
- Parallelization involves overhead: dataset needs to be large enough to compensate

General challenges of parallelization:

- Amount of parallelizable work / Amdahl's Law
- Task granularity
- Load balancing
- Memory allocation and garbage collection (synchronization)
- Cache coherence
- Locality
- Control flow divergence



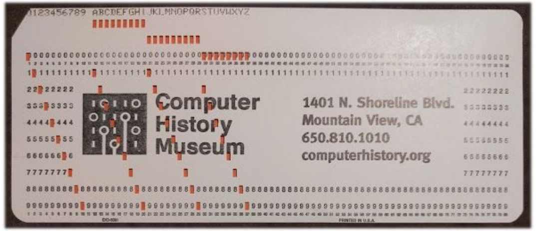




Tristram, Dale & Bradshaw, Karen. (2014). Identifying attributes of GPU programs for difficulty evaluation. South African Computer Journal. 53. 10.18489/sacj.v53i0.195.

History

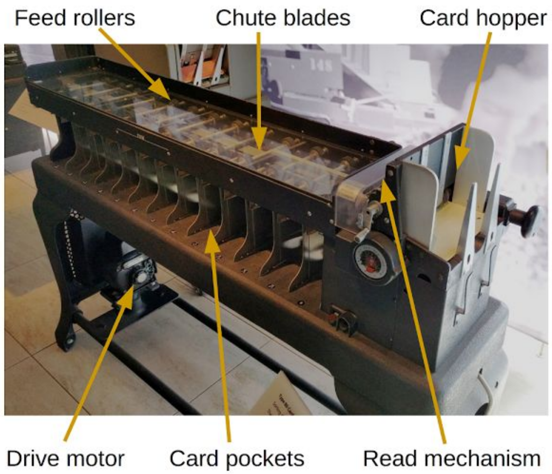
- 1887: **Herman Hollerith** (American Inventor)
  - Hollerith Machine for population count of 1890 U.S census.
  - 8 years to process (census every 10 years) vs 3 months.
- 1896: Hollerith founded the Tabulating Machine Company (TMC)
- 1900’s: Tabulating Machines went mainstream for accounting and tracking inventory
- 1911: After merger, TMC became Computing Tabulating Recording Company (CTR)
- 1924: CTR became International Business Machines Corporation (**IBM**).
- 1954: **Harold H. Seward** (MIT)
  - Creates first memory-efficient radix sort algorithm
  - Previously: allocated space for buckets of unknown size
  - Initial scan to get bucket sizes and offsets to do allocation
  - Invented counting sort and applied it to radix sort

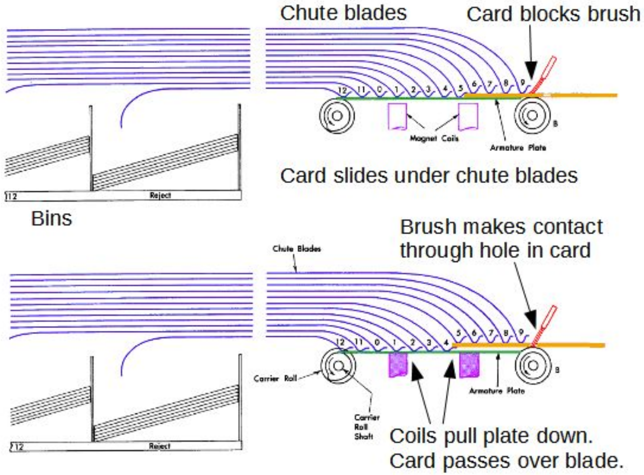


F. da Cruz, “Hollerith 1890 Census Tabulator,” Hollerith 1890 census tabulator. [Online]. Available: http://www.columbia.edu/cu/computinghistory/census-tabulator.html. [Accessed: 18-Apr-2023].

History

1. Wire brush (red) detects the presence or absence of a hole
2. If there is a hole in the card, the brush makes contact with the roller through the hole, completing the circuit
3. A stack of metal guides (chute blades) are used to direct the card into the appropriate bin





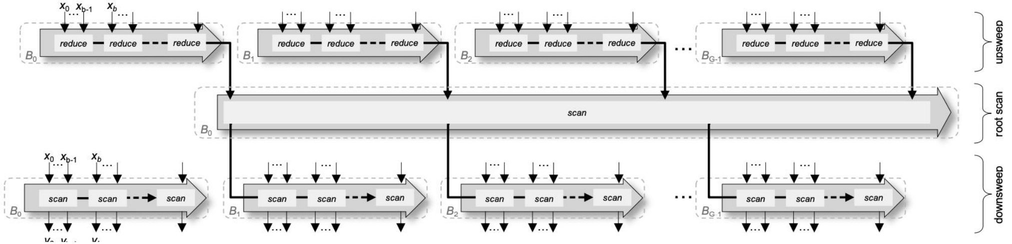
K. Shirriff, *Inside card sorters: 1920s data processing with punched cards and relays*. [Online]. Available: http://www.righto.com/2016/05/inside-card-sorters-1920s-data.html. [Accessed: 18-Apr-2023].

Onesweep: Radix Sort for GPUs

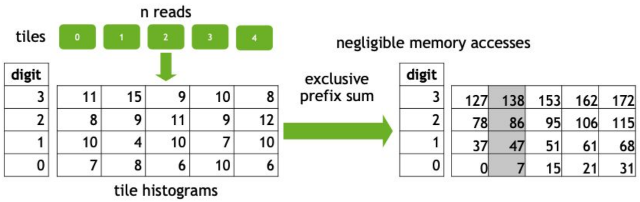
- Adinets and Merrill (2022) written in CUDA for NVIDIA GPUs.
- Uses single-pass prefix sum: reduces last-level memory traffic
  - **~2*n* vs 3*n* global memory operations**

Previous implementation: *reduce-then-scan,* 3 kernels and 2 full passes through the data

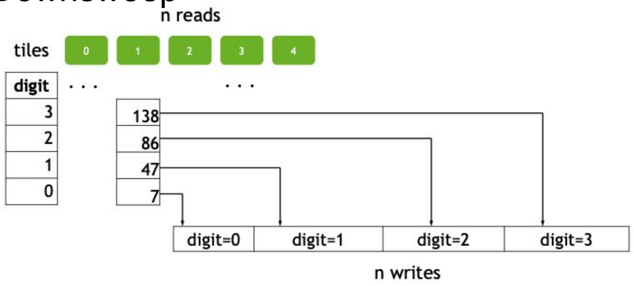
1. “upsweep” pass to compute per-block digit histograms
2. Prefix sum of block counts
3. “downsweep” pass to relocate keys



Upsweep



Downsweep

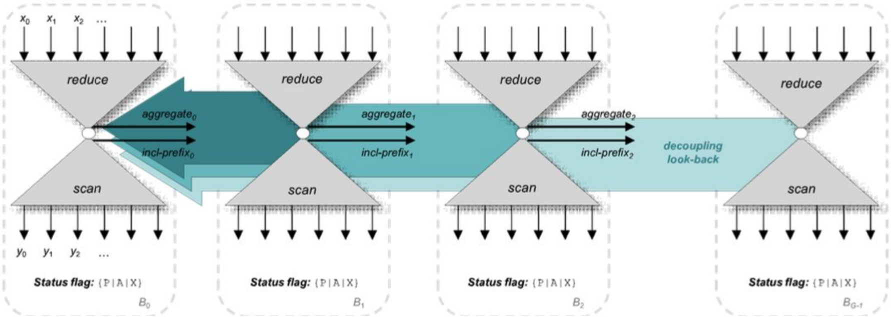


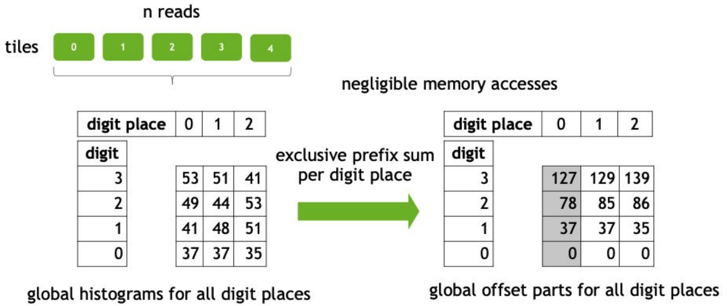
Adinets, Andy, and Duane Merrill. “Onesweep: A Faster Least Significant Digit Radix Sort for GPUs.” arXiv, June 3, 2022. https://doi.org/10.48550/arXiv.2206.01784.

Onesweep: Radix Sort for GPUs

1. Histograms of global digit counts
2. Prefix sums of global digit counts
3. p = # of digits iterations of *chained scan* digit binning: each thread block reads its tile of elements, decodes key digits, participates in a chained scan of block-wide digit counts and scatters its elements into their global output bins (using its digit counts and global digit counts)

~(2*p*+1)*n vs* ~3*pn* memory operations for each digit-binning iteration.





digit place	0	1	2
digit			
3	53	51	41
2	49	44	53
1	41	48	51
0	37	37	35

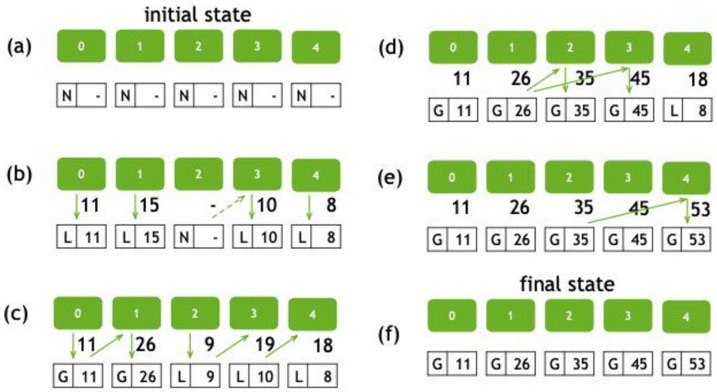
digit place	0	1	2
digit			
3	127	129	139
2	78	85	86
1	37	37	35
0	0	0	0

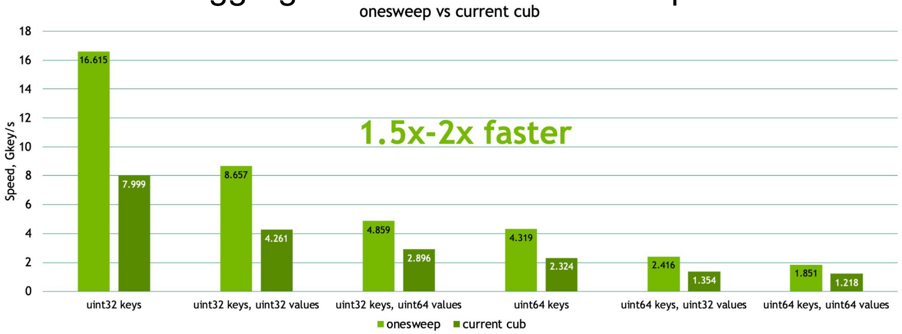
Adinets, Andy, and Duane Merrill. "Onesweep: A Faster Least Significant Digit Radix Sort for GPUs." arXiv, June 3, 2022. https://doi.org/10.48550/arXiv.2206.01784.

Onesweep: Radix Sort for GPUs

Chained scan:

- Each block is assigned a tile of the input and computes their local tile aggregate and the running prefix propagates from block to block.
- Latency in the propagation can be hidden via “decoupled lookback”: each block progressively consumes the per-tile totals of its predecessors until it discovers one that has also the global inclusive prefix.
- It must only wait on the processor to produce its local tile aggregate and all blocks are expected to





“We also reduce the number of binning iterations by using a larger digit size”

Adinets, Andy, and Duane Merrill. “Onesweep: A Faster Least Significant Digit Radix Sort for GPUs.” arXiv, June 3, 2022. https://doi.org/10.48550/arXiv.2206.01784.

Parallelization Techniques: MPI

- Message Passing Interface (MPI)
- Standard for passing messages across processes in a **distributed memory system**
- Standardisation started back in 1991, most recent: MPI 4
- Implemented by various vendors and open source efforts: OpenMPI, MPICH, IBM Spectrum, …
- High level: Run application multiple times concurrently and assign work to specific processes:

```
MPI_Comm_size(MPI_COMM_WORLD, &mpi_size);
MPI_Comm_rank(MPI_COMM_WORLD, &mpi_rank);
```

- Communicate data in case another process needs it:

Rank 0: `MPI_Send(buf, 1, MPI_INT, mpi_rank + 1, 0, MPI_COMM_WORLD);`
Rank 1: `MPI_Recv(buf, 1, MPI_INT, mpi_rank - 1, 0, MPI_COMM_WORLD);`

Parallelization Techniques: MPI - RDMA

- Remote direct memory access (RDMA) allows to access the memory of another process while bypassing both operating systems
- Instead of copying data from the network to some kernel space and then to the application memory, the data is copied directly to the memory region of the application
- Requires compatible networking hardware: Infiniband, advanced Ethernet
- No explicit synchronization: Application needs to be aware of where memory is modified
- Collectively allocate memory on each process:

```
MPI_Win_allocate(length * sizeof(int), sizeof(int), MPI_INFO_NULL,
MPI_COMM_WORLD, &shared_sorting, &win);
```

- Access other processes memory if needed:

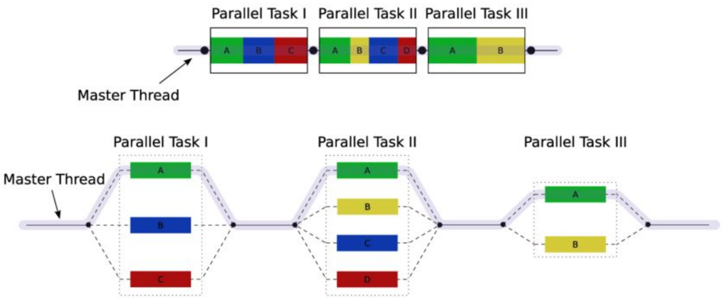
```
if (move_to_rank == mpi_rank)
    shared_sorting[rank_local_position] = output[i];
else {
    MPI_Put(&output[i], 1, MPI_INT, move_to_rank, rank_local_position, 1,
    MPI_INT, win); }
```

Parallelization Techniques: OpenMP

- Open Multi-Processing (OpenMP)
- API for **shared memory** parallelism and implemented within the compiler (GCC, Clang, …)
- Parallelism is achieved by the fork-join model where for specific parts of the application threads are created
- Implemented using compiler directives:

```
$ export OMP_NUM_THREADS=12

#pragma omp parallel for
for (int i = 0; i < length; i++) {
     output[i] = shared_sorting[i];
}
```



Heller, Thomas. (2019). Extending the C++ Asynchronous Programming Model with the HPX Runtime System for Distributed Memory Computing. Available: https://opus4.kobv.de/opus4-fau/frontdoor/index/index/docId/11078 [Accessed: 18-Apr-2023].

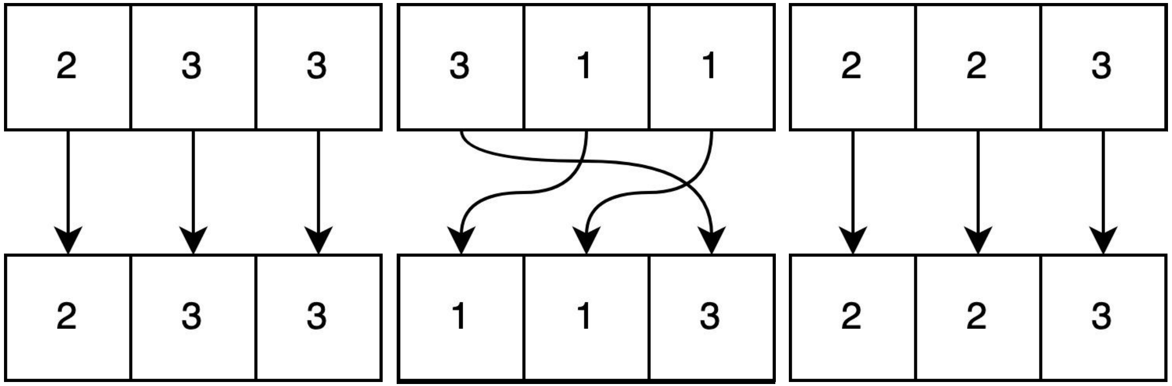
Parallelization Scheme

Bin

- # of Occurrences
- Cumulative count

Step 1: Process-local sorting

Process 1 | **Process 2** | Process 3



Counting

Bin	# of Occurrences
0	0
1	2
2	0
3	1

Cumulative

Bin	Cumulative count
0	0
1	0
2	2
3	2

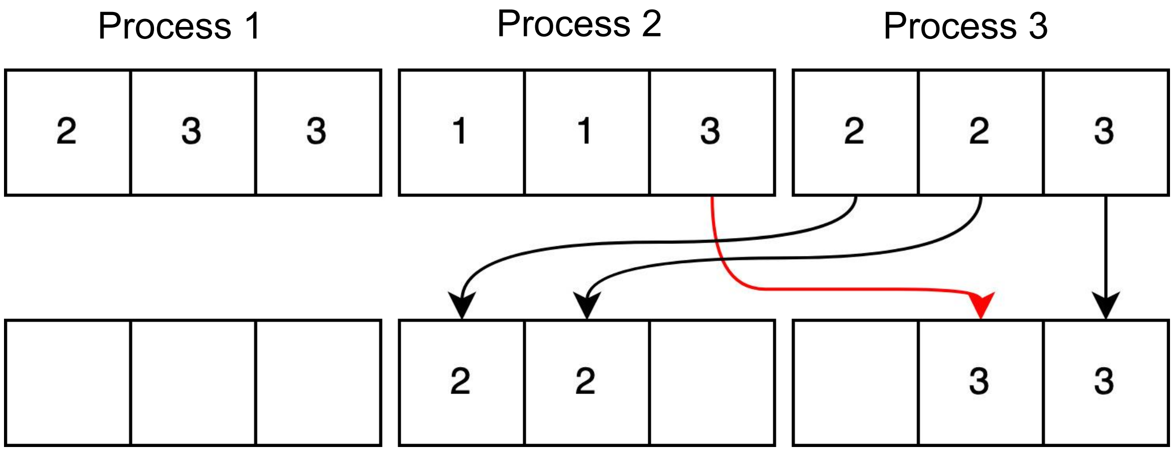
Step 2: Redistribute globally

Parallelization Scheme

Bin

- # of Occurrences
- Cumulative count

Step 2: Redistribute globally - **Non-parallelized**



Global Cumulative

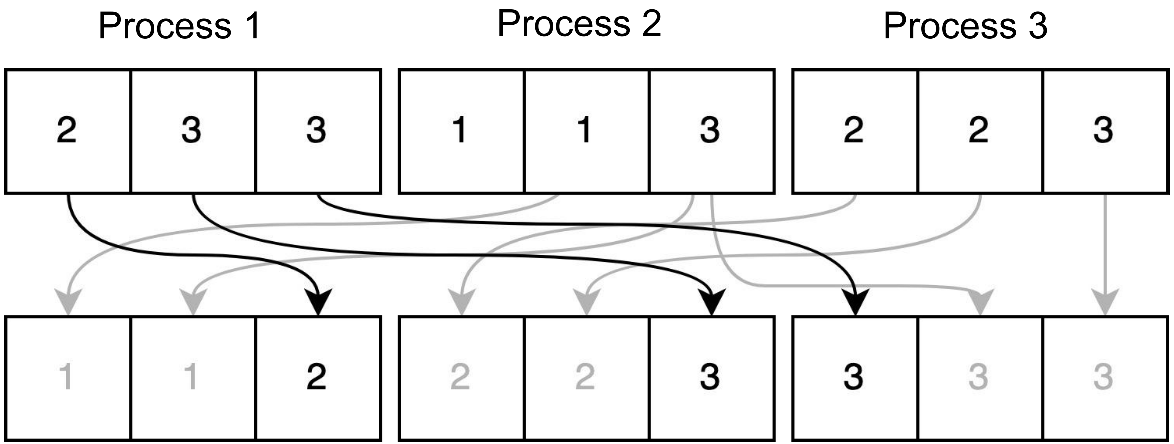
Bin	Cumulative count
0	0
1	2
2	5
3	9

Global Cumulative **Process 2**

Bin	Cumulative count
0	0
1	2
2	3
3	8

Parallelization Scheme

Step 2: Redistribute globally - **Parallelized**



Bin

- # of Occurrences
- Cumulative count

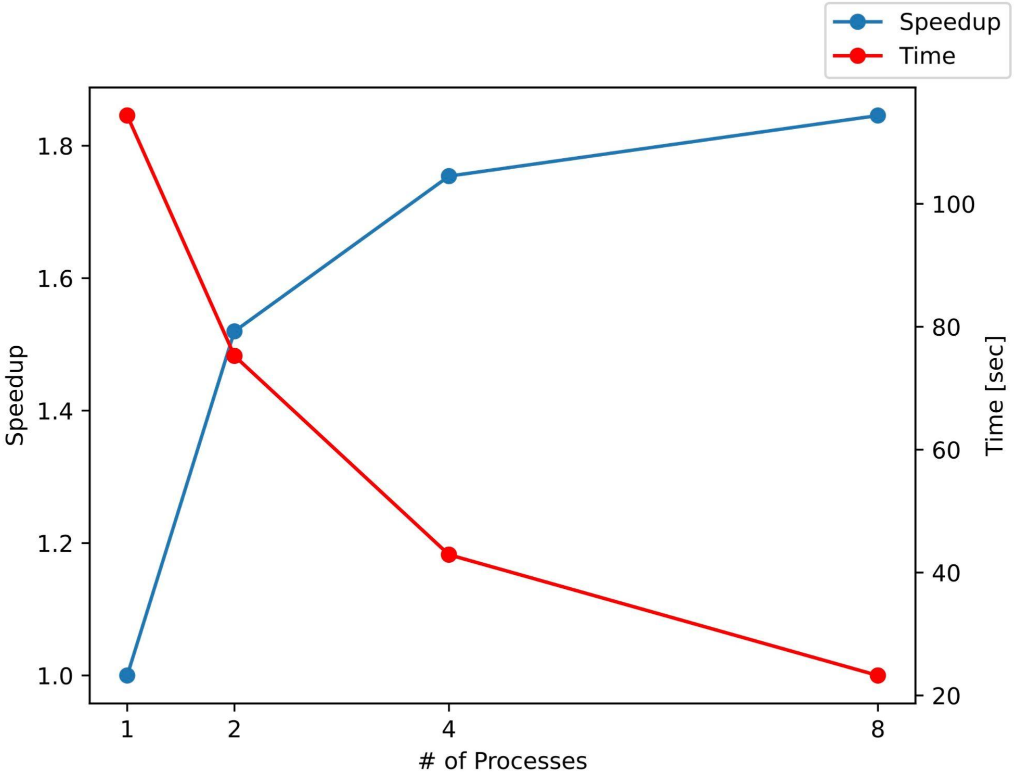
Offsetted Cumulative

Bin	Cumulative count
0	0
1	0
2	3
3	7

Strong Scaling

$$Speedup = t(1)/t(N)$$

- Strong scaling on 1, 2, 4 and 8 processes on Apple M2 Pro with OpenMPI@4.1.4
- Sorting 400.000.000 numbers
- Speedup trend noticeable to at least 8 processes



THE UNIVERSITY OF TENNESSEE KNOXVILLE

Applications

- **In general:**
  - Search and query problems (to search DB efficiently)
  - Event-driven simulations
  - Schedulers
  - Huffman compression
  - Kruskal’s algorithm
  - Dijkstra's algorithm
  - Construction and manipulation of data structures, especially those modeling relationships in sparse systems
- **More common for radix sort:**
  - Sorting large datasets of integers
  - Text indexing (string processing)
  - Genome sequencing: “On the tandem duplication-random loss model of genome rearrangement”
    - Duplication loss step: equivalent to one step of radix sort.

Open Issues

- Challenge of achieving high levels of utilization from irregular workloads: which yield a very inconsistent performance response that is highly dependent upon the key distribution and problem size.
- Reducing communication overhead
- GPU parallelization: atomic read-modify-write operations can provide similar utility as prefix sum with less overhead, especially on architectures where they are implemented at the memory-level.
  But there are problems associated with it:
    - Their update-order is non-deterministic, preventing them from being used in stable LSD digit binning outside of simple digit-counting tasks
    - Performance can be unstable in sorting problems having low digit diversity due to contended accesses to the same counters
    - Their hardware support may be non-existent or have insufficient throughput for use at scale

References

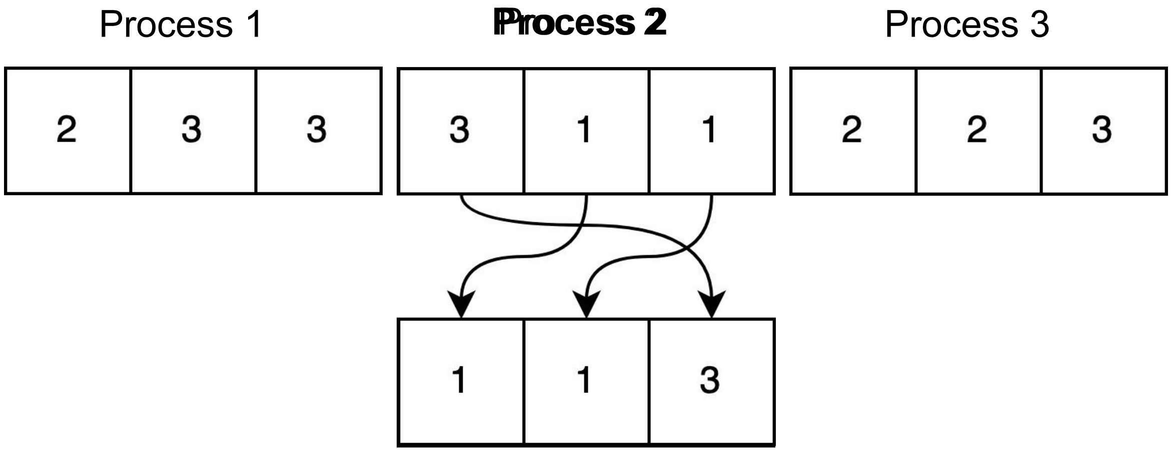
- Tristram, Dale & Bradshaw, Karen. (2014). Identifying attributes of GPU programs for difficulty evaluation. South African Computer Journal. 53. 10.18489/sacj.v53i0.195.
- K. Shirriff, Inside card sorters: 1920s data processing with punched cards and relays. [Online]. Available: http://www.righto.com/2016/05/inside-card-sorters-1920s-data.html. [Accessed: 18-Apr-2023].
- F. da Cruz, “Hollerith 1890 Census Tabulator,” Hollerith 1890 census tabulator. [Online]. Available: http://www.columbia.edu/cu/computinghistory/census-tabulator.html. [Accessed: 18-Apr-2023].
- “Radix sort,” Wikipedia, 30-Jan-2023. [Online]. Available: https://en.wikipedia.org/wiki/Radix_sort. [Accessed: 19-Apr-2023].
- Adinets, Andy, and Duane Merrill. “Onesweep: A Faster Least Significant Digit Radix Sort for GPUs.” arXiv, June 3, 2022. https://doi.org/10.48550/arXiv.2206.01784.
- Chaudhuri, Kamalika & Yong Syuan, Chen & Mihaescu, Radu & Rao, Satish. (2006). On the Tandem Duplication-Random Loss Model of Genome Rearrangement. Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms. 564-570. 10.1145/1109557.1109619.
- Axtmann, Michael, Timo Bingmann, Peter Sanders, and Christian Schulz. “Practical Massively Parallel Sorting.” arXiv, February 25, 2015. https://doi.org/10.48550/arXiv.1410.6754.

Discussion

Questions

1. Why were the first implementations of radix sort considered memory inefficient?
2. What is the difference between OpenMP and MPI?
3. Why is radix sort difficult to parallelize?

Parallelization Scheme



Bin

- # of Occurrences
- Cumulative count

Counting

Bin	# of Occurrences
0	0
1	2
2	0
3	1

Cumulative

Bin	Cumulative count
0	0
1	0
2	2
3	2