Parallel Sorting

Samplesort

Pedro Costa

University of Minho
Department of Informatics

Braga, May 2012

Index

Index

PSRS

Why?

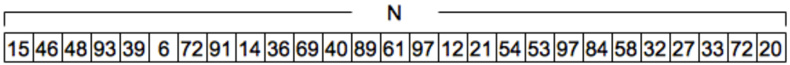
1. From [Quinn, 2004]:

   > ***Parallel sorting by regular sampling*** *(PSRS), developed by Li et al. [Li* et al.*, 1993], has three advantages over hyperquicksort. It keeps list sizes more balanced among the processes, it avoids repeated communications of the keys, and it does not require that the number of processes be a power of 2.*

2. Very high "score" in [Kale and Solomonik, 2010];
3. Quicksort does not offer a challenge;
4. It's awesome.

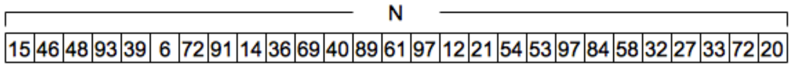
Index

Partition

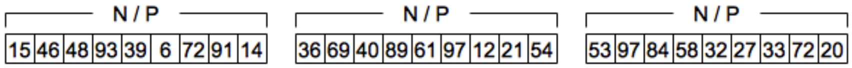


- A set of n keys to order.

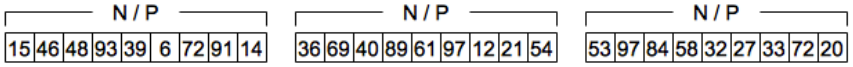
Partition



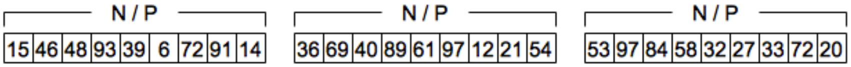
- A set of n keys to order.
- Divide over p processes.



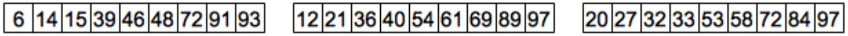
Local sorting



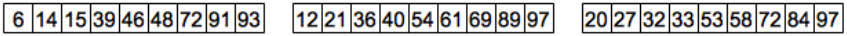
Local sorting



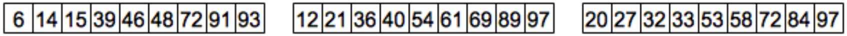
- Each process uses the *quicksort* algorithm.



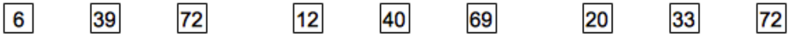
Sampling


6 14 15 39 46 48 72 91 93
12 21 36 40 54 61 69 89 97
20 27 32 33 53 58 72 84 97

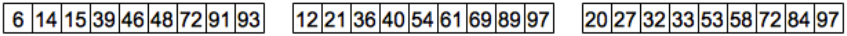
Sampling



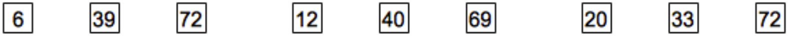
- Each process selects *p* samples:
  - Samplesort variants focus on the sampling rule;
  - Regular Sampling = samples taken at regular intervals.



Sampling

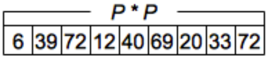


- Each process selects p samples:
  - Samplesort variants focus on the sampling rule;
  - Regular Sampling = samples taken at regular intervals.

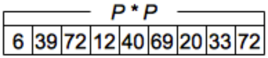


- Each process sends its samples to the master.

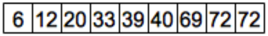
Sort samples



Sort samples



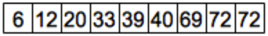
- Master uses *quicksort* to sort the collected samples.



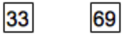
Pivots

6	12	20	33	39	40	69	72	72

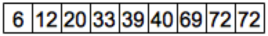
Pivots



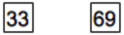
- Master selects $p - 1$ pivots:
  - Rule depends on the algorithm variant;
  - Regular Sampling = regular intervals.



Pivots



- Master selects $p - 1$ pivots:
  - Rule depends on the algorithm variant;
  - Regular Sampling = regular intervals.



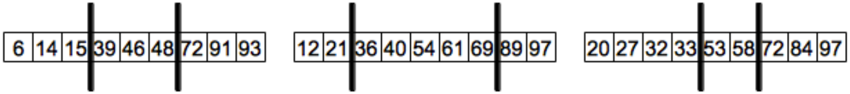
- Master broadcasts the selected pivots.

Slices

$\boxed{33}$ $\boxed{69}$

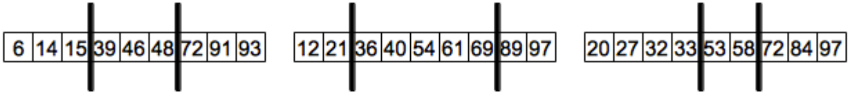
Slices

- Each process divides its partition according to the pivots:
  - $\leq$ to the left;
  - $>$ to the right;



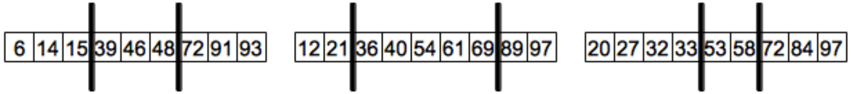
Slices

- Each process divides its partition according to the pivots:
    - $\leq$ to the left;
    - $>$ to the right;

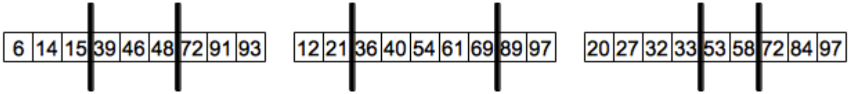


- Each process ends up with p slices.

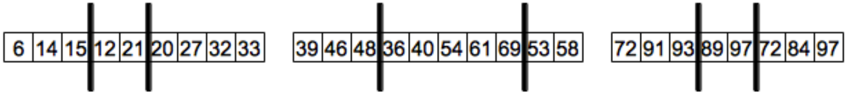
Trade


6 14 15 39 46 48 72 91 93
12 21 36 40 54 61 69 89 97
20 27 32 33 53 58 72 84 97

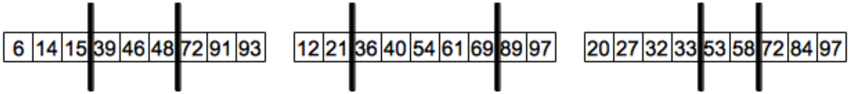
Trade



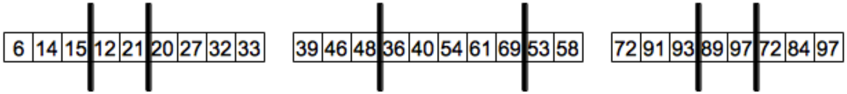
- Each process sends each k^{th} slice to the k^{th} process.



Trade

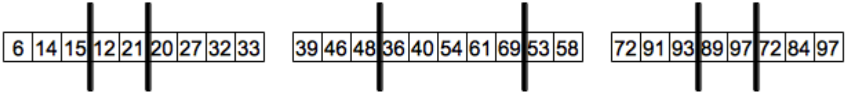


- Each process sends each k^{th} slice to the k^{th} process.

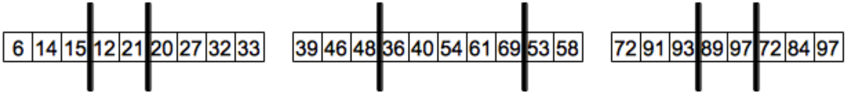


- The resulting partitions reflect the division according to the pivots.

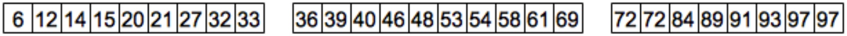
Local sort (part 2)



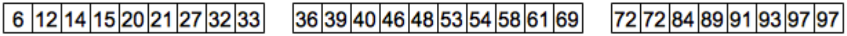
Local sort (part 2)



- Each process uses the *quicksort* algorithm once more.

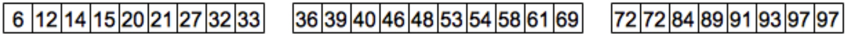


Gather

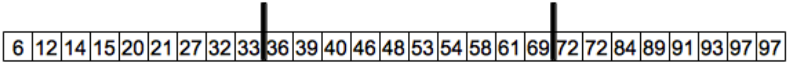


- Each process sends its partition to the master.

Gather



- Each process sends its partition to the master.



- The concatenation of the partitions gives the sorted set.

Index

MPI

- Process 0 reads keys from file, calculates partition sizes and sends the partitions to each other process;

MPI

- Process 0 reads keys from file, calculates partition sizes and sends the partitions to each other process;
- Local sorting with quicksort;

MPI

- Process 0 reads keys from file, calculates partition sizes and sends the partitions to each other process;
- Local sorting with quicksort;
- Sampling;

MPI

- Process 0 reads keys from file, calculates partition sizes and sends the partitions to each other process;
- Local sorting with quicksort;
- Sampling;
- Process 0 sorts samples, collects pivots and broadcasts;

MPI

- Process 0 reads keys from file, calculates partition sizes and sends the partitions to each other process;
- Local sorting with quicksort;
- Sampling;
- Process 0 sorts samples, collects pivots and broadcasts;
- Each process calculates where to slice;

MPI

- Process 0 reads keys from file, calculates partition sizes and sends the partitions to each other process;
- Local sorting with quicksort;
- Sampling;
- Process 0 sorts samples, collects pivots and broadcasts;
- Each process calculates where to slice;
- Trade: first sizes, then content;

MPI

- Process 0 reads keys from file, calculates partition sizes and sends the partitions to each other process;
- Local sorting with quicksort;
- Sampling;
- Process 0 sorts samples, collects pivots and broadcasts;
- Each process calculates where to slice;
- Trade: first sizes, then content;
- Local sorting, again;

MPI

- Process 0 reads keys from file, calculates partition sizes and sends the partitions to each other process;
- Local sorting with quicksort;
- Sampling;
- Process 0 sorts samples, collects pivots and broadcasts;
- Each process calculates where to slice;
- Trade: first sizes, then content;
- Local sorting, again;
- Every process sends its partition to process 0;

MPI

- Process 0 reads keys from file, calculates partition sizes and sends the partitions to each other process;
- Local sorting with quicksort;
- Sampling;
- Process 0 sorts samples, collects pivots and broadcasts;
- Each process calculates where to slice;
- Trade: first sizes, then content;
- Local sorting, again;
- Every process sends its partition to process 0;
- Process 0 outputs the result;

OpenMP

- Master thread reads keys from file and calculates partitions (size and offset);

OpenMP

- Master thread reads keys from file and calculates partitions (size and offset);
- Local sorting with quicksort;

OpenMP

- Master thread reads keys from file and calculates partitions (size and offset);
- Local sorting with quicksort;
- Collect samples and save in a global array;

OpenMP

- Master thread reads keys from file and calculates partitions (size and offset);
- Local sorting with quicksort;
- Collect samples and save in a global array;
- Master sorts samples, collects pivots and saves in a global array;

OpenMP

- Master thread reads keys from file and calculates partitions (size and offset);
- Local sorting with quicksort;
- Collect samples and save in a global array;
- Master sorts samples, collects pivots and saves in a global array;
- Each thread calculates where to slice;

OpenMP

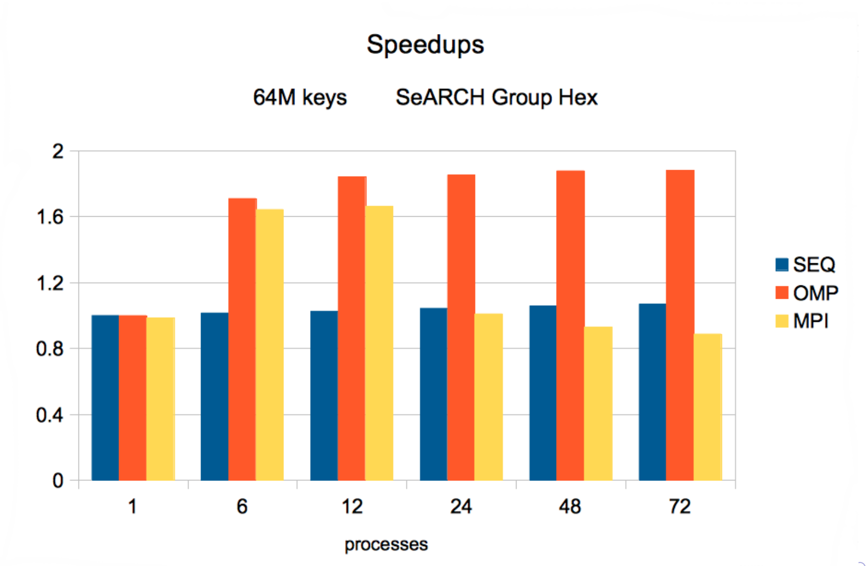
- Master thread reads keys from file and calculates partitions (size and offset);
- Local sorting with quicksort;
- Collect samples and save in a global array;
- Master sorts samples, collects pivots and saves in a global array;
- Each thread calculates where to slice;
- Trade-Sort-Gather:
    - Master reserves the memory for the sorted array;
    - Each thread calculates the size of the new partition and copies from old array;
    - Each thread sorts its new partition;

OpenMP

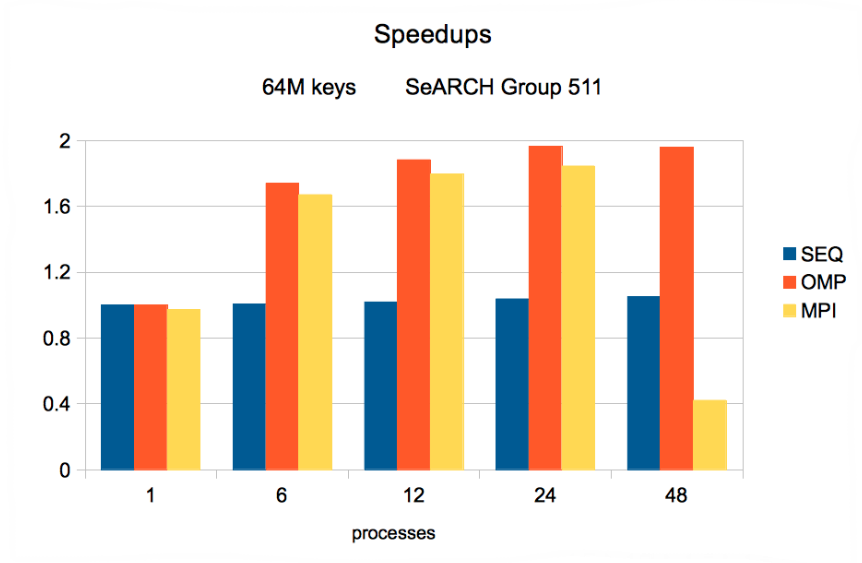
- Master thread reads keys from file and calculates partitions (size and offset);
- Local sorting with quicksort;
- Collect samples and save in a global array;
- Master sorts samples, collects pivots and saves in a global array;
- Each thread calculates where to slice;
- Trade-Sort-Gather:
    - Master reserves the memory for the sorted array;
    - Each thread calculates the size of the new partition and copies from old array;
    - Each thread sorts its new partition;
- Master thread outputs the result;

Index

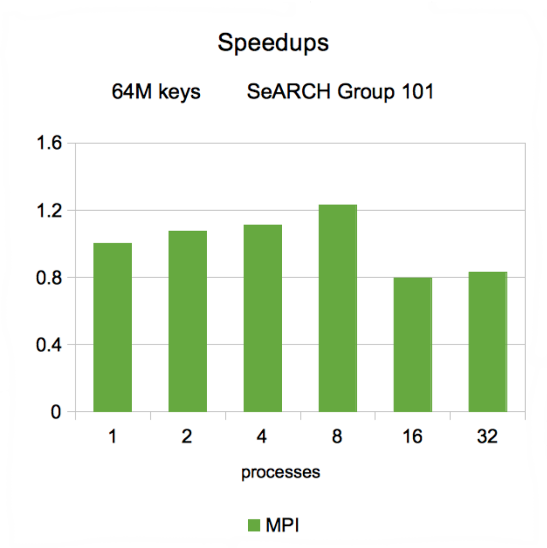
Results@hex


Speedups
64M keys SeARCH Group Hex
2
1.6
1.2
0.8
0.4
0
1
6
12
24
48
72
processes
SEQ
OMP
MPI

Results@511


Speedups
64M keys
SeARCH Group 511
2
1.6
1.2
0.8
0.4
0
1
6
12
24
48
processes
SEQ
OMP
MPI

Results@101


Speedups
64M keys
SeARCH Group 101
1.6
1.2
0.8
0.4
0
1
2
4
8
16
32
processes
🟩 MPI

Conclusion

- The algorithm is better suited for shared memory
  - The number of keys to sort must be a lot larger for distributed memory to stand out.
- HyperThreading reduces the performance to values near MPI.
- OpenMP
  - Merge parallel zones;
  - More threads vs. more partitions/thread;

  MPI
  - Increase n;
  - Optimized directives;

Bibliography

Kale, V. and Solomonik, E. (2010).
Parallel sorting pattern.
In Proceedings of the 2010 Workshop on Parallel Programming Patterns ParaPLoP '10 pp. 10:1–10:12, ACM, New York, NY, USA.

Li, X., Lu, P., Schaeffer, J., Shillington, J., Wong, P. S. and Shi, H. (1993).
On the versatility of parallel sorting by regular sampling.
Parallel Comput. *19*, 1079–1103.

Quinn, M. (2004).
Parallel Programming in C With Mpi and Openmp.
McGraw-Hill Higher Education, McGraw-Hill Higher Education.

Parallel Sorting

Samplesort

Pedro Costa

University of Minho
Department of Informatics

Braga, May 2012

– ? –