CS301 project report

Team members:
(1) Aayush Kapadia (201401407)
(2) Mit Naria (201401448)

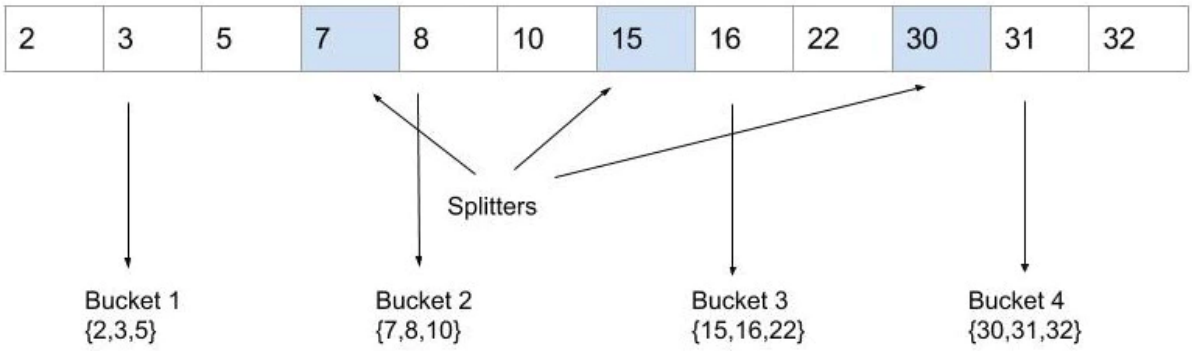
Project Title: Sample Sort

Introduction:

- Sample sort is a parallel sorting algorithm.
- Sample sort is a comparison based sorting algorithm.
- Sample sort follows PSRS algorithm design strategy. [Parallel Sorting by Regular Sampling] [explained in upcoming sections]

Definition of Splitters:

Splitters: Splitters are the elements which divides sorted array into small blocs[chunks or buckets].
Example:
Here, a sorted array of 12 elements is splitted in 4 blocs[chunks or buckets] by 3 splitters.



PSRS algorithm design strategy

This algorithm design consists of 5 stages.

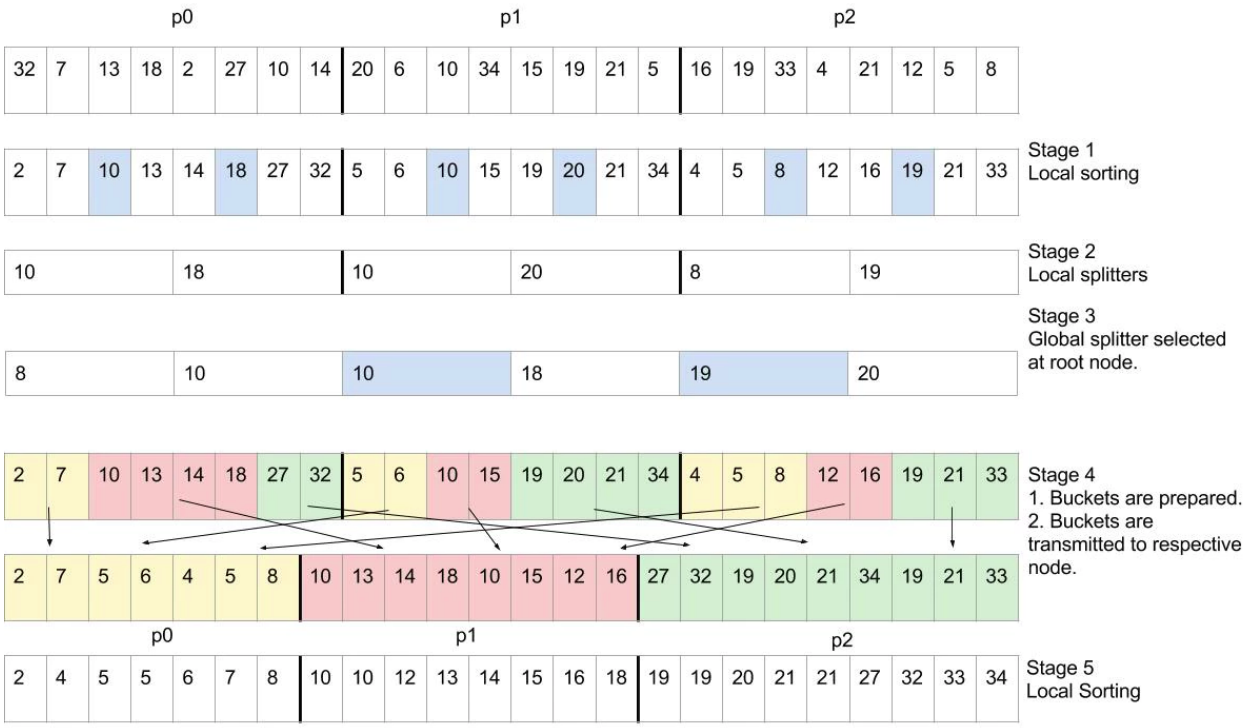
1. Local sorting
2. Generation of local splitters by using a sampling algorithm
3. Generation of global splitters
4. Distributing buckets to nodes
5. Local sorting of a bucket

Input for sample sort is unsorted array and output is sorted array.

Sample sort algorithm:

1. Local sorting (Quicksort) .
2. Generation of local splitters. (each node selects (p-1) splitters) .
3. Generation of global splitters. (root node gathers p*(p-1) splitters and sorts them using quicksort and selects (p-1) splitters from total p*(p-1) splitters. They are transmitted to each node.)
4. Distributing buckets to nodes. (buckets are prepared by using new (p-1) splitters and send to respective node. Bucket[i] goes node with rank=i. After that each node merges the buckets received.)
5. Local sorting of a bucket (Quicksort).

Example:



We have assumed that communication latency is O`(log(p))` .

Theoretical analysis of parallel algorithm:

Step 1: O`((n/p)*log(n/p))` [Sorting]
Step 2: O`(p)` [Choosing local splitters]
Step 3: O`(p*log(p))` [Collecting local splitters] + O`((p^2)*log(p))` [Sorting local splitters]
+ O`(p)` [Choosing global splitters] + O`(p*log(p))` [Distributing global splitters]
Step 4: O`(n/p)` [Buckets generation] + O`((n/p)*log(p))` [Buckets transmission]
+ O`((n/p)*log(p))` [Merging buckets]
Step 5: O`((n/p)*log(n/p))` [Sorting]

As noted in reference [2] and proved in reference [3],
Computational complexity: O`((n/p)*log(n/p))` + O`((p^2)*log(p))` + O`((n/p)*log(p))` .
Since `n >> p` , `log(n/p) ~ log(n)` ,
Overall computational complexity:O`( (n/p) * ( log(n) + log(p) ) )` .

Communication complexity: O`(p*log(p))` + O`((n/p)*log(p))` .

Since `(n/p)>>log(p)` and `(n/p)>>p` ,
Overall communication complexity: O`(n/p)` .

Total bound: O`(n/p)` +O`( (n/p) * ( log(n) + log(p) ) )` .

We compare this parallel algorithm to serial randomized quicksort algorithm. [most widely used comparison-based sorting algorithm] .

Theoretical analysis of serial algorithm:

Total bound: O`(n*log(n))` .

Theoretical speedup:

Theoretical Speedup = `O(n*log(n)) ) / { O(n/p) + O( (n/p) * (log(n)+log(p)) ) }` .

Expected speedup:

Expected speedup (predication based approach):
To calculate approximate speedup, we simply say that parallel algorithm actually sorts twice.[Local sorting]. Thus, on p processors calculation is p times faster but 2 times slower.
Thus, predicated speedup = `p/2` . [This speedup is just based on observation.]

Scalability:

From theoretical analysis of parallel algorithm:
k(n,p) = O`(p*log(p))` + O`((n/p)*log(p))` .
We can ignore first term in k(n,p).
Thus, k(n,p) = O`((n/p)*log(p))`
Therefore To(n,p) = O`(n*log(p))`
T(n,1) = O`(n*log(n))` .

Isoefficiency function:

```
T(n,1)      >= C * To(n,p)
n * log(n)  >= C * n * log(p)
log(n)      >= C * log(p)
n           >= p^C                        ....equation(1)
```

Scalability function:

```
Since M(n) = n , Scalability function M(f(p))/p = p^(C-1)
```

If C = 1 then system will perfectly scalable.

```
C = (1 - efficiency)/efficiency .
Suppose x = maximum allowed efficiency which results into perfectly
scalable system.
Then C = (1-x)/x = 1 (because system is perfectly scalable.)
This only happens when x = 1/2.
```

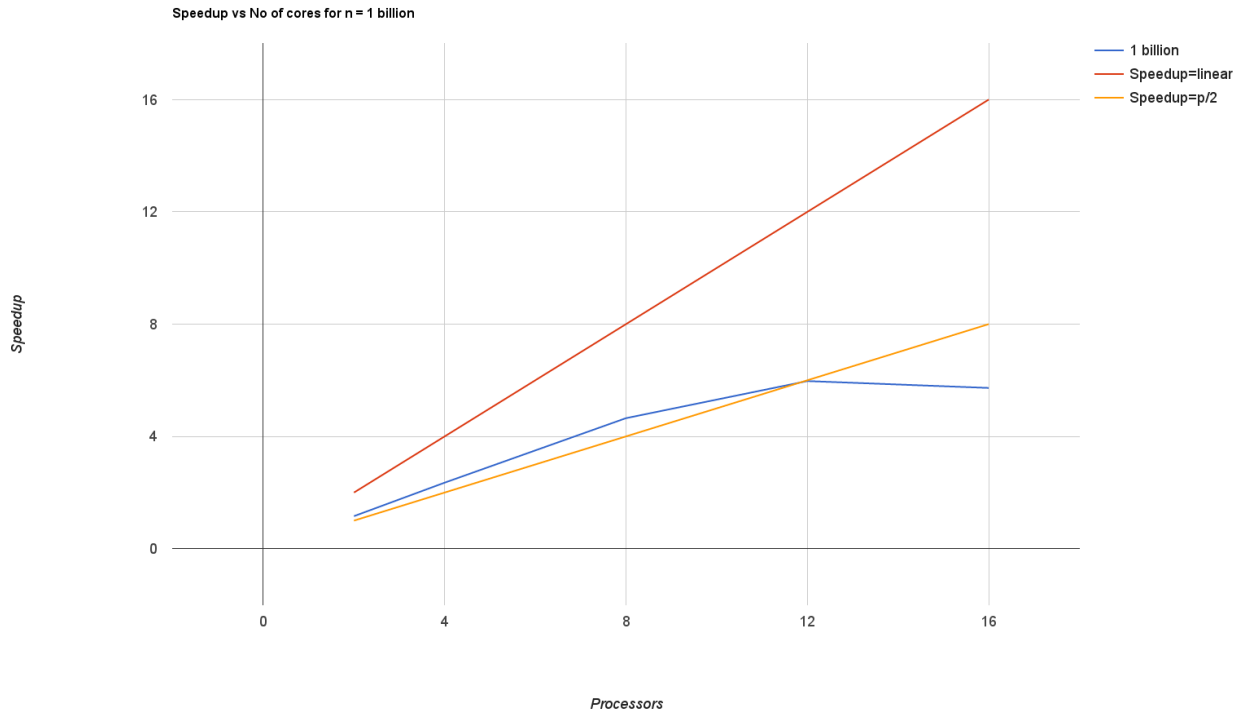
Thus we can maintain efficiency of 1/2 and speedup of p/2 by increasing n linearly with p.
(i.e. put C=1 in equation(1) thus isoefficiency function becomes n >= p).

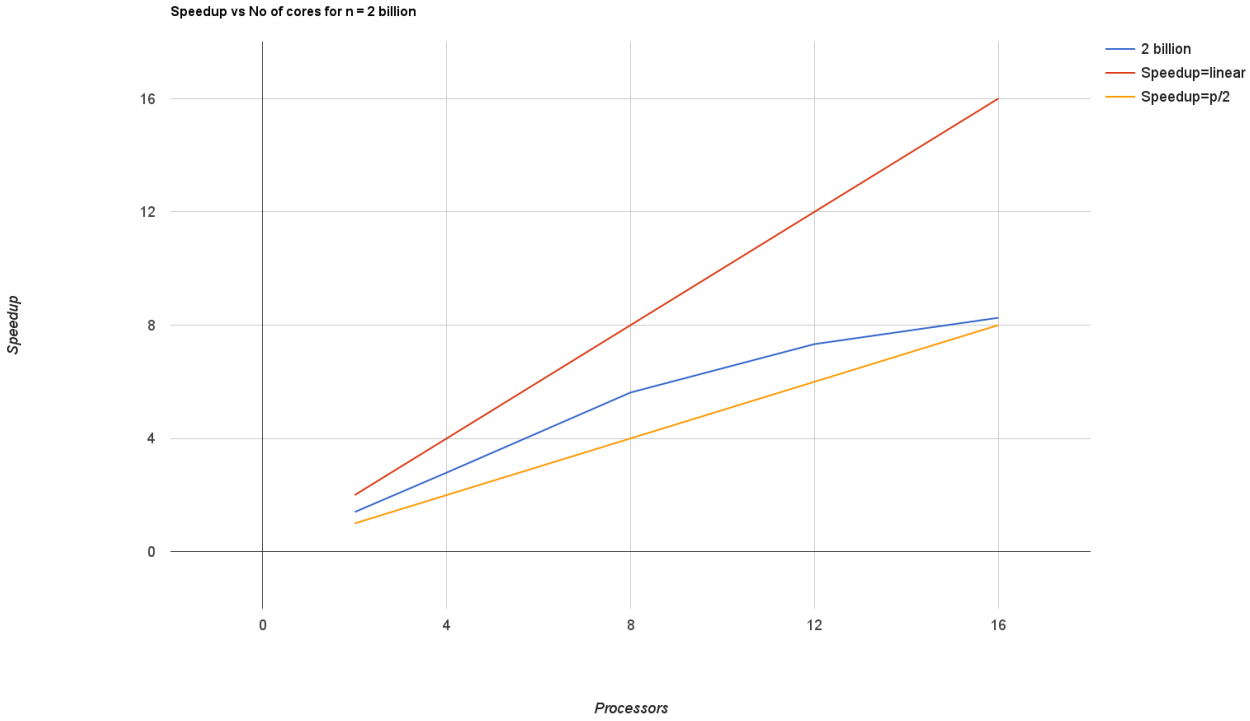
Practical speedup:

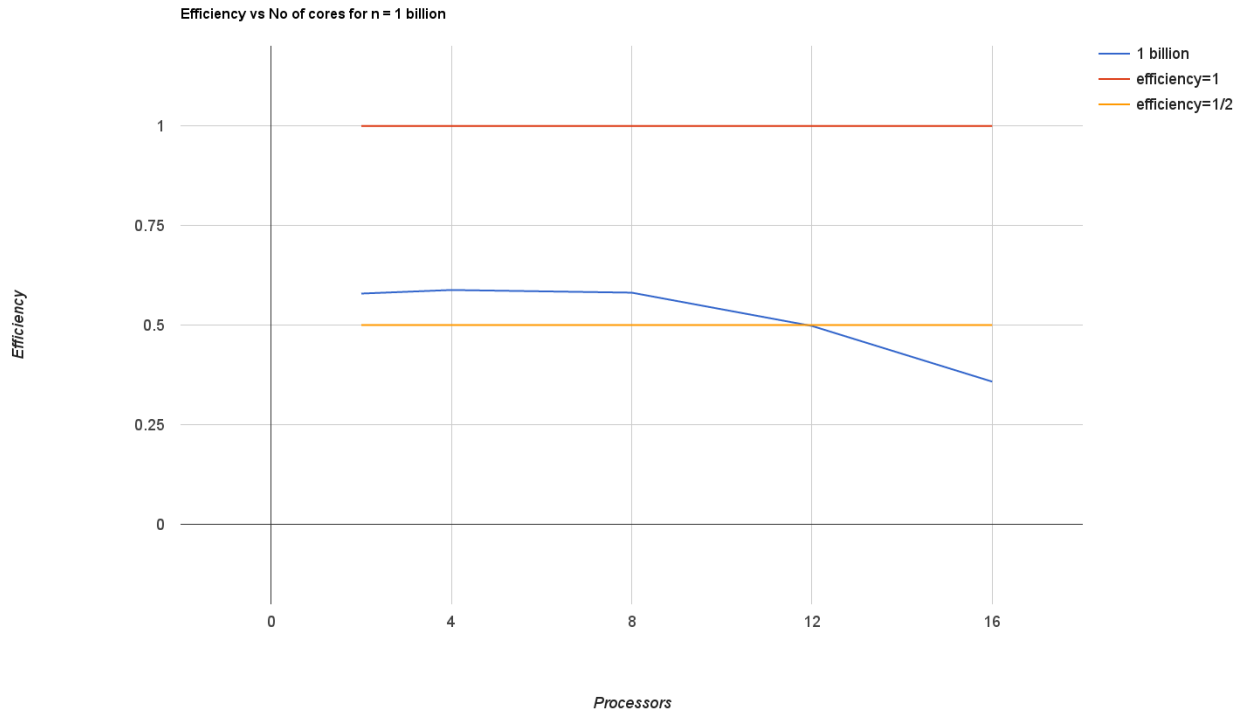
Hardware details:

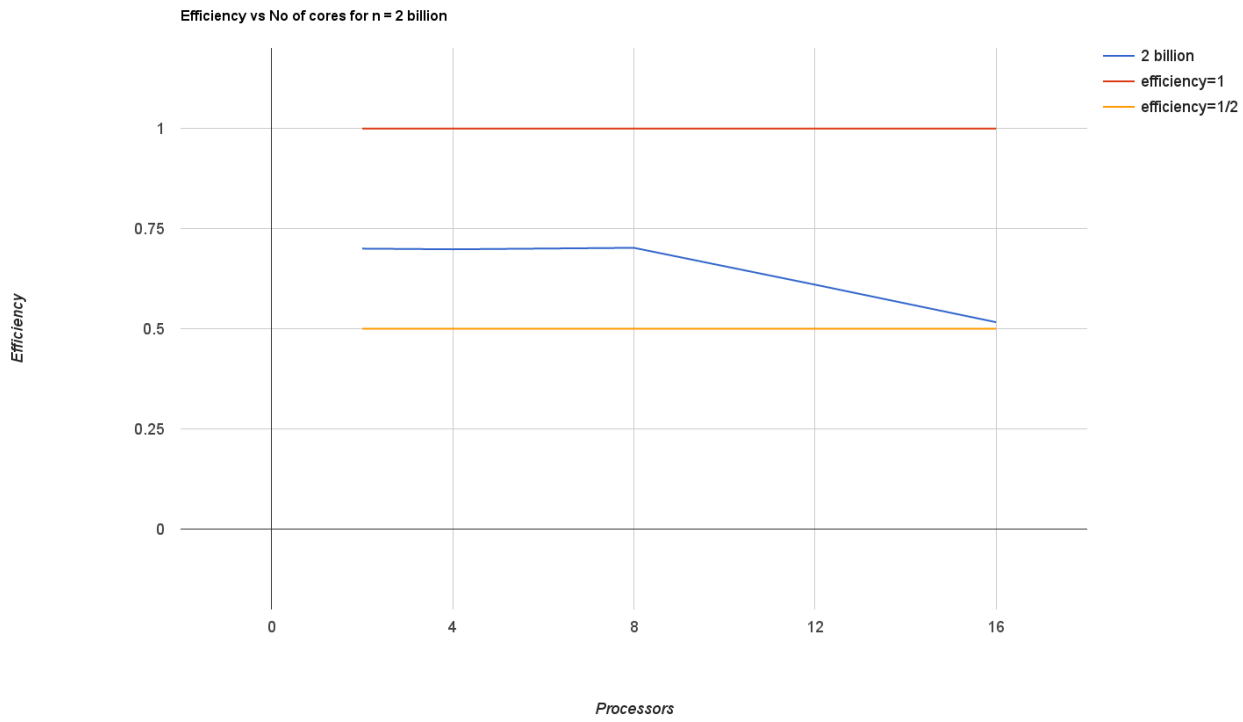
```
Architecture:            x86_64
CPU op-mode(s):          32-bit, 64-bit
Byte Order:              Little Endian
CPU(s):                  16
On-line CPU(s) list:     0-15
Thread(s) per core:      1
Core(s) per socket:      8
Socket(s):               2
NUMA node(s):            2
Vendor ID:               GenuineIntel
CPU family:              6
Model:                   62
Stepping:                4
CPU MHz:                 1200.000
BogoMIPS:                3999.45
Virtualization:          VT-x
L1d cache:               32K
L1i cache:               32K
L2 cache:                256K
L3 cache:                20480K
NUMA node0 CPU(s):       0-7
NUMA node1 CPU(s):       8-15
```

Amdahl's law:

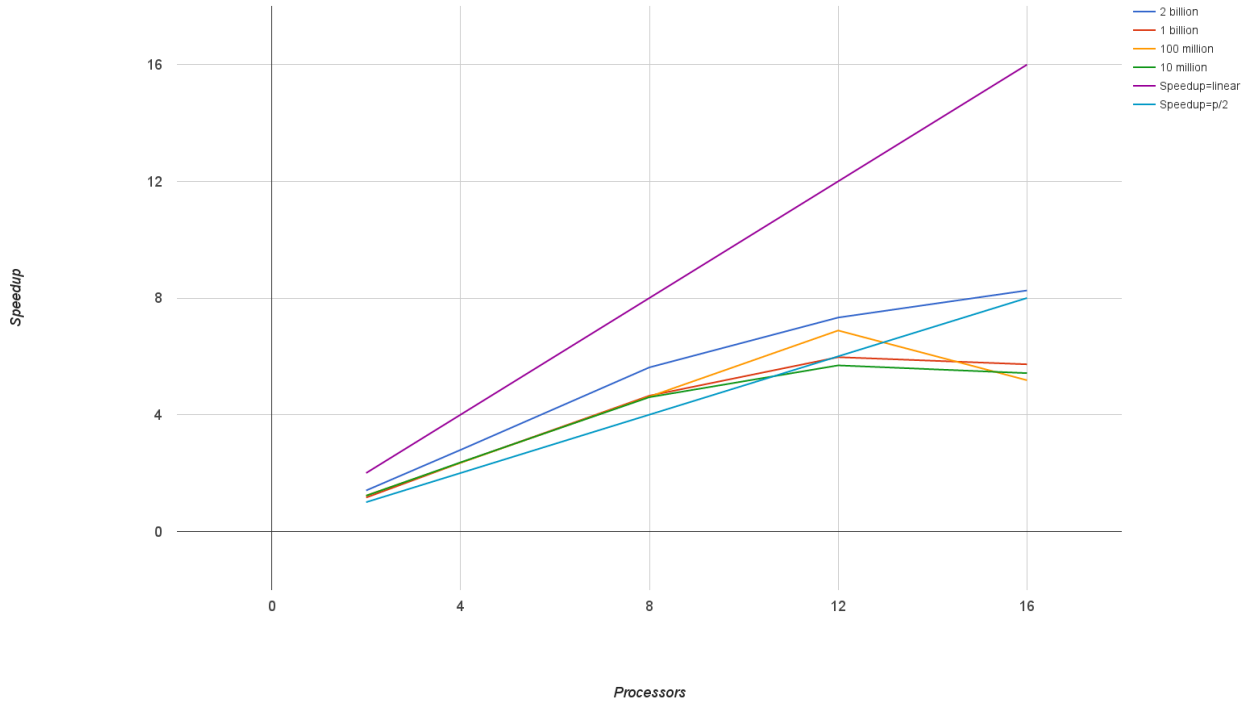

Speedup vs No of cores for n = 1 billion
1 billion
Speedup=linear
Speedup=p/2
Speedup
16
12
8
4
0
0
4
8
12
16
Processors

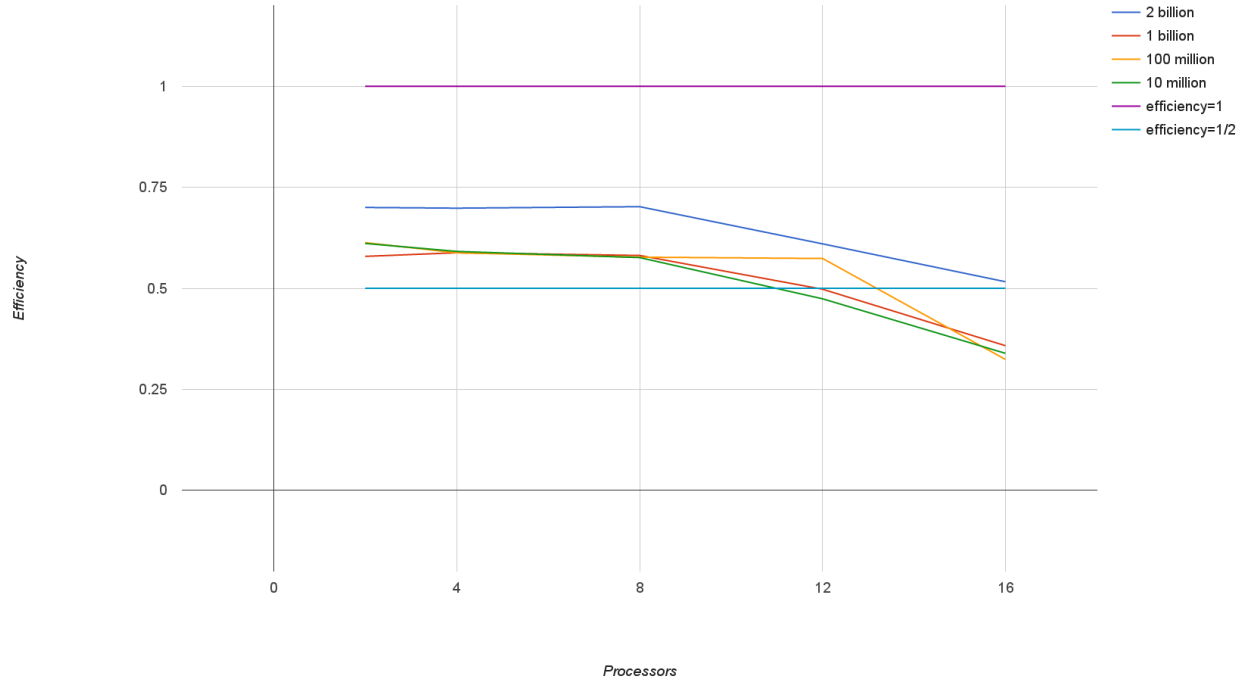

Speedup vs No of cores for n = 2 billion
2 billion
Speedup=linear
Speedup=p/2
Speedup
16
12
8
4
0
0
4
8
12
16
Processors


Efficiency vs No of cores for n = 1 billion
1 billion
efficiency=1
efficiency=1/2
Efficiency
1
0.75
0.5
0.25
0
0
4
8
12
16
Processors


Efficiency vs No of cores for n = 2 billion
2 billion
efficiency=1
efficiency=1/2
Efficiency
1
0.75
0.5
0.25
0
0
4
8
12
16
Processors

Amdahl's effect:





Further Improvements:

Based on design strategy of sample sort (PSRS algorithm design strategy), several new sorting algorithms have been developed. Most of them focus on 2 things:

(i) Better Sampling algorithm

In sample sort, we first sort and then choose local splitters.

A better way is to use a good sampling algorithm which eliminates need of this local sorting.

Parallel Histogramsort and Psort [older world record holders of fastest sorting algorithms] have focused on this point.

(ii) Decrease communication time

As number of cores in sample sort increases [greater than 32k as mentioned in reference[4]], communication time becomes bottleneck and hardware for communication starts to reach its saturation capacity. There are 2 ways to decrease communication time. First, by using shared memory programming model and second, by improving communication patterns.

a. By using shared memory programming model:
   In shared memory model , sharing of data is faster than sending data to another core in same node. We can improve performance and decrease communication time by using shared memory parallelism. A node with many cores uses a shared memory based local sorting algorithm. cloudRAMsort uses this approach.
b. By improving communication patterns:
   By observing sample sort , we can say all the communication happens in small amount time which saturates the hardware capacity of communication. We can decrease time taken in communication by doing communication on various stages rather than one single stage. cloudRAMsort uses this approach.

By following above 2 improvements, many sorting algorithms have been designed which have been appeared in fastest sorting algorithms as mentioned in reference [5] and [6].

Conclusion:

Through sample sort may not be able to sort terabytes data, it can certainly be used to sort gigabytes of data efficiently. Variants of sample sort with above 2 improvements have been proposed and proved to sort terabytes of data. Example: Psort, cloudRAMsort, Parallel Histogram sort. New variant of sample sort with improvements named HykSort have been proposed to sort exabytes of data.
Thus , we can say that PSRS algorithm design strategy [PSRS - Parallel Sorting by Regular Sampling] which was introduced by sample sort continue to be main driver in parallel sorting algorithms.

References:

1. “Introduction to Parallel Computing” (2nd edition) by Ananth Grama, Anshul Gupta, George Karypis, Vipin Kumar. (Topic 9.5) [http://parallelcomp.uw.hu/ch09lev1sec5.html]
2. “Parallel Programming in C with MPI and OpenMP” by Michael Quinn (Topic 14.5 Pg 346-349).
3. “On the versatility of parallel sorting by regular sampling.” by Li. X., P. Lu., J. Schaeffer, J. Shillington, P. S. Wong and H. Shi. Parallel Computing 19: Pg 1079-1193, 1993
   [https://pdfs.semanticscholar.org/3307/61cfa0d31d184e3583e41c8ba3d19e8ea929.pdf]
   [https://scholar.google.com/citations?view_op=view_citation&hl=en&user=W6WwvbQAAAAJ&cstart=280&sortby=pubdate&citation_for_view=W6WwvbQAAAAJ:YsMSGLbcyi4C]
4. HykSort: A New Variant of Hypercube Quicksort on Distributed Memory Architectures by Hari Sundar, Dhairya Malhotra, George Biros. Proceedings of the 27th international ACM conference on International conference on supercomputing
   [https://www.cs.utah.edu/~hari/files/pubs/ics13.pdf]
   [https://scholar.google.com/citations?view_op=view_citation&hl=en&user=equOxc0AAAAJ&sortby=pubdate&citation_for_view=equOxc0AAAAJ:ZeXyd9-uunAC]
   [https://pdfs.semanticscholar.org/c04f/f62fd8366fa57fb9a039a52e590470066f43.pdf]

5. A Comparison of Sorting Algorithms for the Connection Machine CM-2 by Guy E. Blelloch, Charles E. Leiserson, Bruce M. Maggs, C. Gregory Plaxton, Stephen J. Smith, and Marco Zagha. SPAA '91 Proceedings of the third annual ACM symposium on Parallel algorithms and architectures. Pages 3-16. [http://dl.acm.org/citation.cfm?id=113380] [https://courses.cs.washington.edu/courses/cse548/06wi/files/benchmarks/radix.pdf]
6. Sorting benchmarks [http://sortbenchmark.org/]
7. Phases of PSRS [http://csweb.cs.wfu.edu/bigiron/LittleFE-PSRS/build/html/PSRSalgorithm.html]