# Bitonic Sort

Amrutha Mullapudi
CSE633: Parallel Algorithms (Spring 2014)

# Sorting

Arrange an unordered collection of items into a meaningful order.

Sorting can be

- Comparison Based. Example: Bubble Sort, Selection Sort
- Non Comparison Based. Example: Bucket Sort or a Count Sort

# Parallel Sorting Algorithms

Perform many comparisons in parallel.

Data distributed among multiple processors.

- How many per processor?

Communication between processors and data exchange.

Find a trade-off between local vs global processing time.

# Applications of sorting algorithm

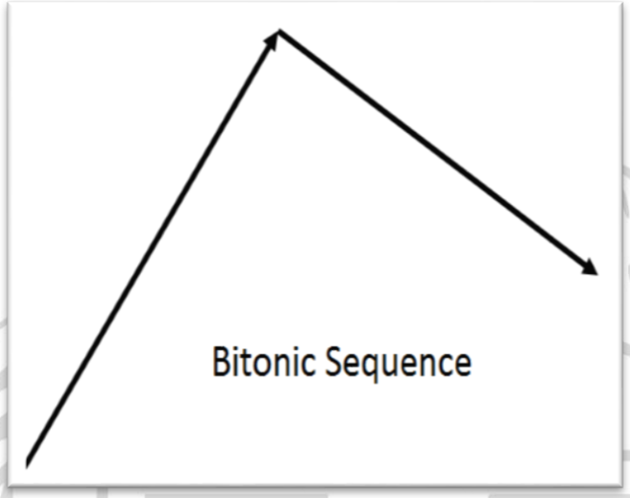
- Organize an MP3 library
- Display Google PageRank results
- Find the median
- Identify statistical outliers
- Find duplicates in a mailing list
- Data compression
- Computer graphics

Every system needs (and has) a system sort!

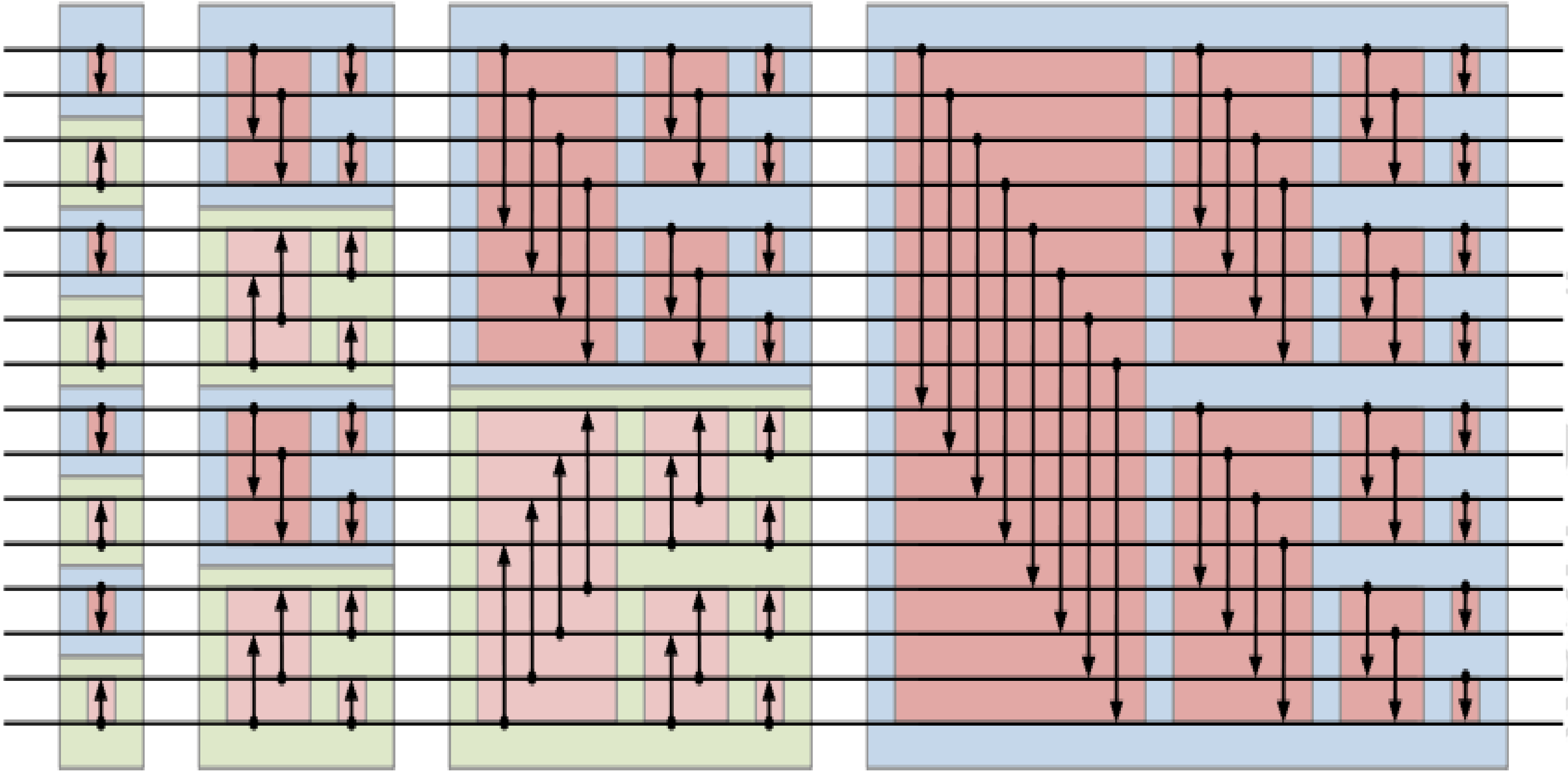
# Bitonic Sequence

A sequence $a = (a1, a2, . . ., ap)$ of $p$ numbers is said to be *bitonic* if and only if

1. $a1 \leq a2 \leq . . . \leq ak \geq . . . \geq ap$, for some $k$, $1 < k < p$, or
2. $a1 \geq a2 \geq . . . \geq ak \leq . . . \leq ap$, for some $k$, $1 < k < p$, or
3. $a$ can be split into two parts that can be interchanged to give either of the first two cases.



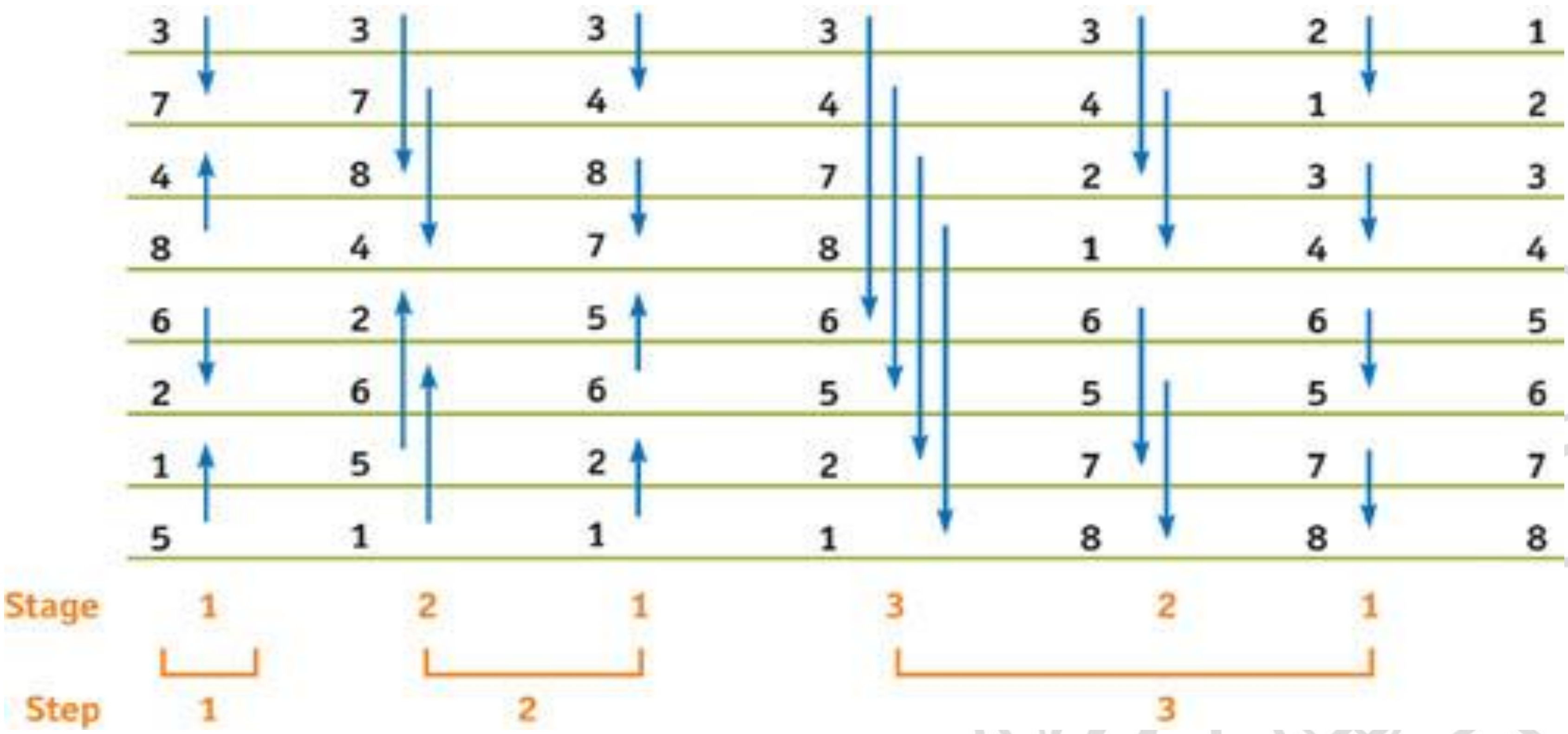
# Bitonic Sorting Network



# Algorithm

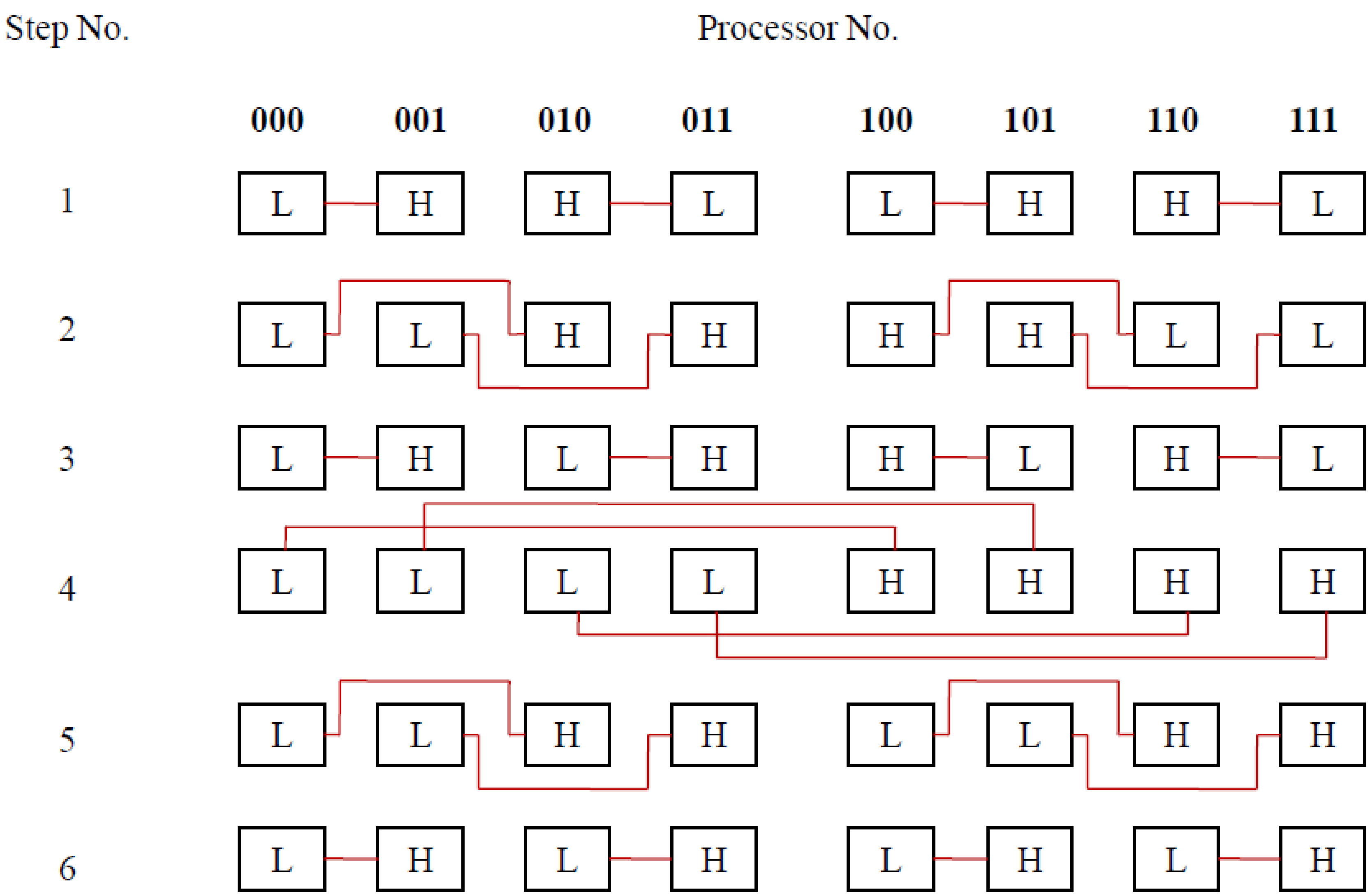
- Input: Random set of $2n=2^k$ (k is some positive integer) numbers. Note that every pair of elements is bitonic.
- Bitonic sequences of size 2 are merged to create ordered lists of size 2. At the end of this first stage of merging, we actually have $n/4$ bitonic sequences of size 4.
- Bitonic sequences of size 4 are merged into sorted sequences of size 4, alternately into increasing and decreasing order, so as to form $n/8$ bitonic sequences of size 8 and so on.
- Given an unordered sequence of size $2n$, exactly $\log_2 2n$ stages of merging are required to produce a completely ordered list.
- Output : Ordered list of size 2n
- $\Theta(\log^2 n)$ levels of comparators are required to sort completely an initially unordered list of size $2n$ *when done in parallel*.

# Bitonic Merge Sort



# Implementation

- Input: Number of processors, Data length
- Find the ranks of each processor
- Generate data in each processor using randomize function
- Sort the lists generated in the processor
- Compare and exchange data with a neighbor whose (d-bit binary) processor number differs only at the jth bit to merge the local subsequences
- The above steps use comparison functions to compare and exchange


Step No.
Processor No.
000 001 010 011 100 101 110 111
1 L H H L L H H L
2 L L H H H H L L
3 L H L H H L H L
4 L L L L H H H H
5 L L H H L L H H
6 L H L H L H L H

# Test Strategy

Parameters: Number of Processors, Number of data elements per processor

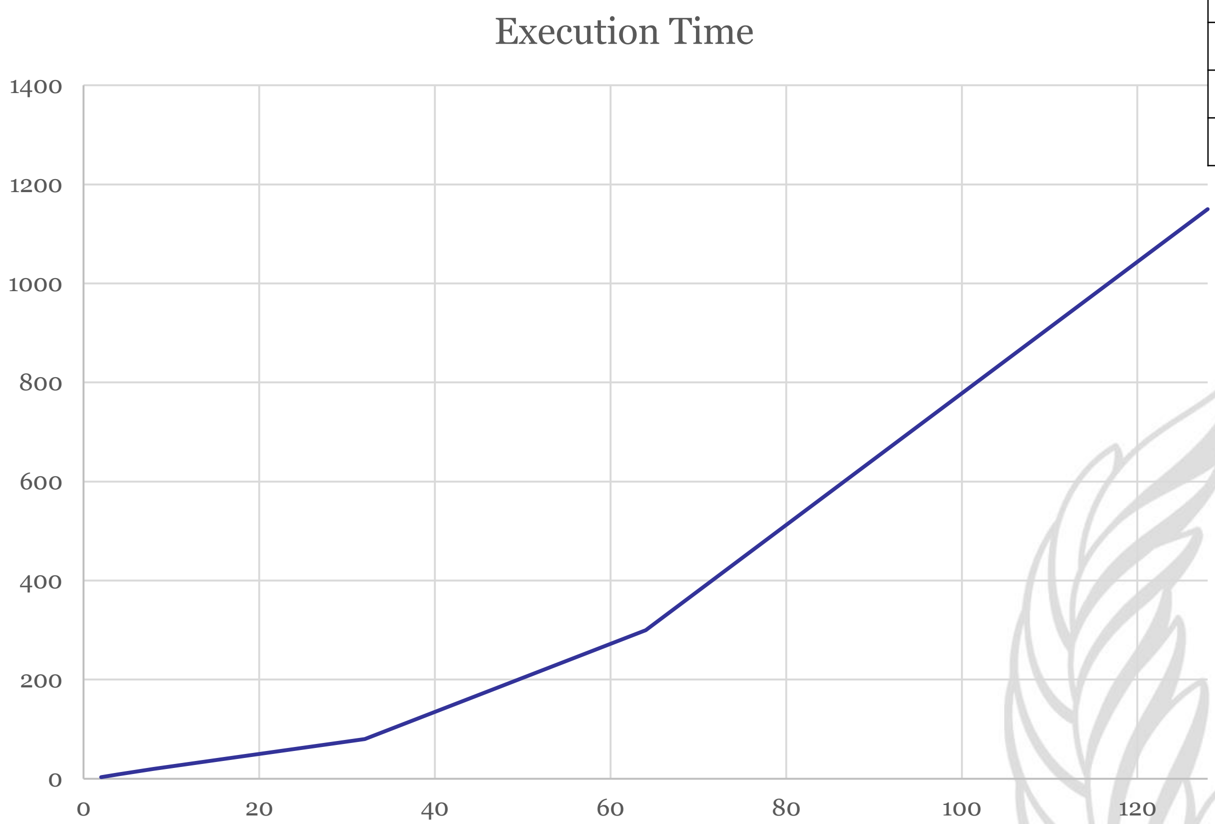
Keeping number of processors constant- Plot number of data elements per processor vs execution time

Keeping number of data elements constant- Plot number of processors vs execution time

Exchanging the complete dataset chunks between processors.

# Number of Processors vs Execution Time with increase in data elements per processor

| No of Processors | Execution Time |
|---|---|
| 2 | 3 |
| 4 | 9 |
| 8 | 20 |
| 16 | 40 |
| 32 | 80 |
| 64 | 300 |
| 128 | 1150 |




| No. of data elements per processor |
|---|
| 10000 |
| 20000 |
| 30000 |
| 40000 |
| 50000 |
| 60000 |
| 100000 |

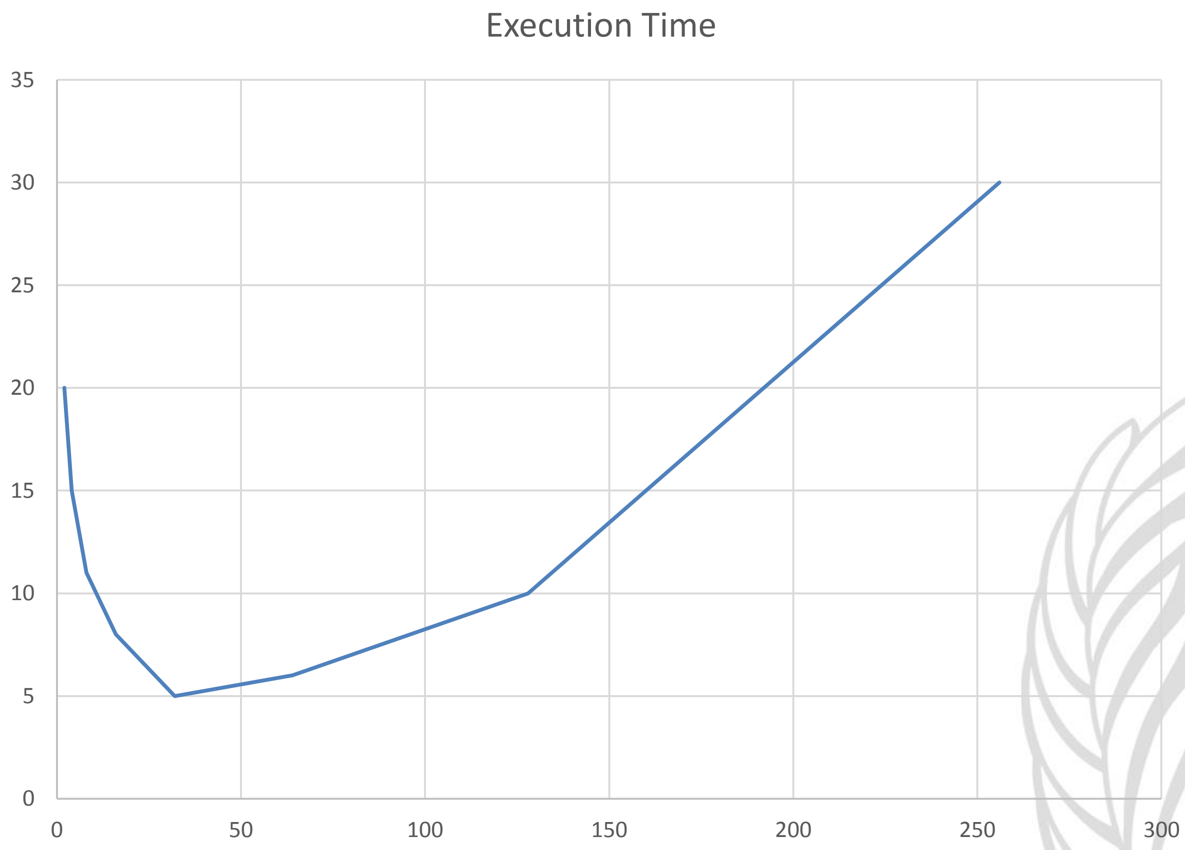
University at Buffalo The State University of New York | REACHING OTHERS

# Number of data elements per processor vs Execution time with constant number of processors

| Data per Processor | Execution Time |
|---|---|
| 8000 | 40 |
| 16000 | 70 |
| 32000 | 170 |
| 64000 | 270 |
| 128000 | 550 |
| 256000 | 1320 |

# Number of processors vs Execution time with constant data element set

| No. of Processors | Execution Time |
|---:|---:|
| 2 | 20 |
| 4 | 15 |
| 8 | 11 |
| 16 | 8 |
| 32 | 5 |
| 64 | 6 |
| 128 | 10 |
| 256 | 30 |



# Dataset Chunk Exchange

Exchanging the processor’s complete dataset with another did not make any significant difference in running time.

Time taken to sort is dominating the running time.

Communication time is negligible in comparison to sorting time.

# Future Scope

Implement parallel quicksort and hyper quicksort algorithms and compare their running times.

Analyze the advantages and disadvantages of bitonic sort when compared to other sorting algorithms

# References


- **Algorithms Sequential and Parallel: A Unified Approach by Russ Miller and Laurence Boxer**
- http://en.wikipedia.org/wiki/Bitonic_sorter
- http://www.cs.rutgers.edu/~venugopa/parallel_summer2012/bitonic_overview.html

# THANK YOU