Implementing Parallel Quick Sort on OTIS Hyper Hexa-Cell (OHHC) interconnection network

Mohamad Fasha & Esam Al-Nsour

Outline

* Introduction.
  * Sorting algorithms and Quick Sort algorithm.
  * Interconnection Networks; HHC and OHHC.
  * Parallel Algorithms.
* Problem statement.
* The proposed parallel QS algorithm.
* Algorithm analytical assessment.
* Algorithm simulation results.
* Conclusion.

Introduction

* Sorting algorithms.
* Quick Sort algorithm.
* Interconnection Networks.
* HHC interconnection network.
* OTIS optoelectronic architecture.
* OHHC optoelectronic interconnection network.
* Parallel Algorithms.

Introduction

Sorting algorithms:

* Sorting is defined as arranging data consisting of items of the same kind in a prescribed sequence.
* Sorting is a fundamental process considered as an elementary operation in computer science.
* Sorting is needed in too many cases where the sequence of input data is of high importance
* Used as an initial step for different purpose algorithms.

Introduction

Quick Sort algorithm:

* Well-known sorting algorithm first introduced by Hoare in 1962.
* Sorts in place a set of data elements using the divide-and-conquer process.
* An array A [*p..r*] is partitioned into two non-empty sub arrays; A[*p..q*] and A[*q+1..r*].
* Sub arrays are then recursively sorted by calls to QS.
* No combining step is required since the two sub arrays form an already-sorted array

Introduction

Interconnection networks:

* Networks that connect a group of processors and memory units to construct either a shared address space computers or message passing computers
* May be classified as being static or dynamic networks.
* An interconnection network can be represented as an undirected graph G(V,E):
  * processor is represented as a node $u \in V(G)$,
  * Edge $(u, v) \in E(G)$ between corresponding nodes u and v represents a communication channel between processors.
* These networks importance rise from its being the heart of the parallel processing systems.

Introduction

Hyper Hexa-Cell Interconnection network (HHC):

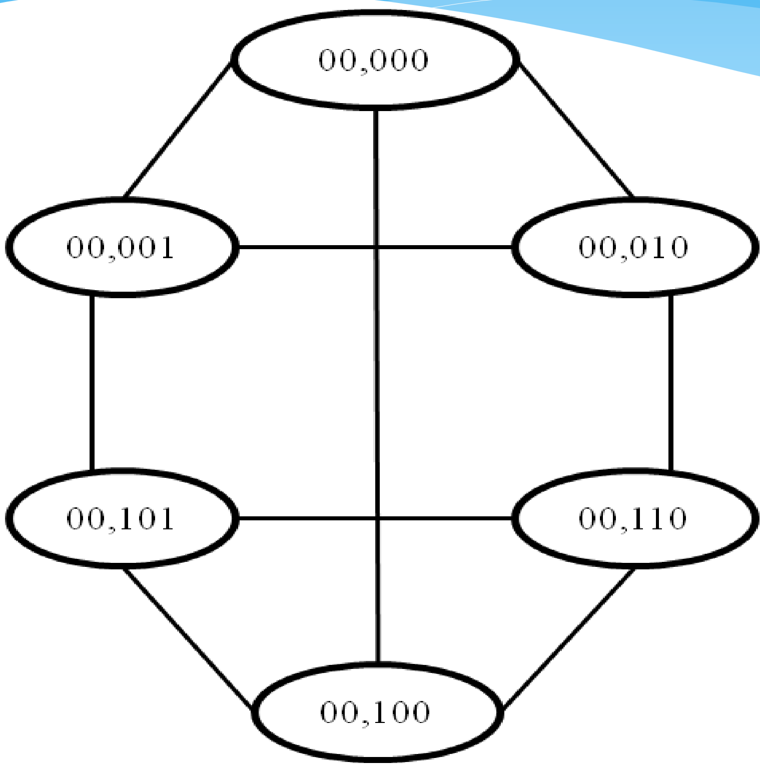
* Static interconnection network.
* Consists of six connected processors arranged in hexagon shape.
* The hexagon shape is arranged into two triangles where each contains three processors that are fully connected.
* Each node in each triangle is connected to the node corresponding to (facing) it in the other triangle

Introduction

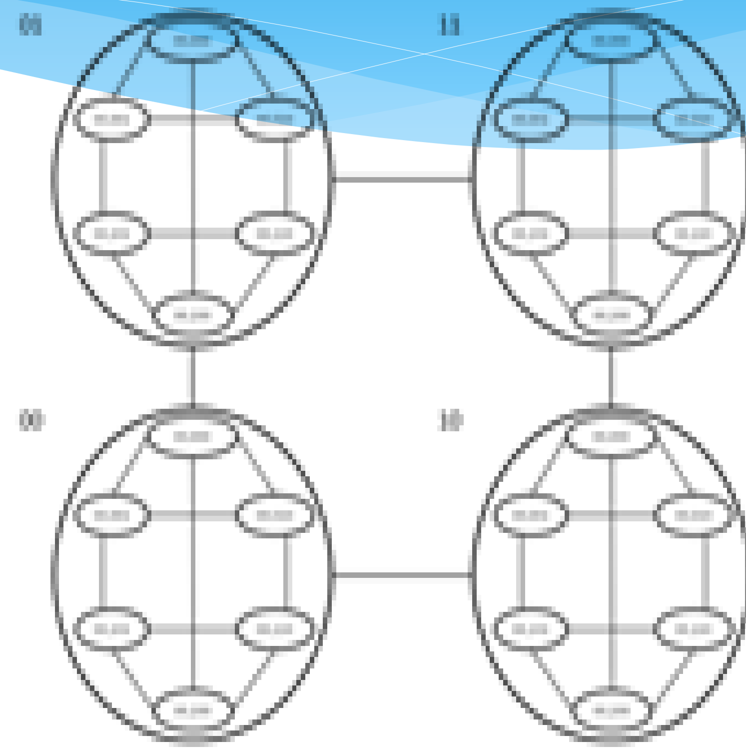
Hyper Hexa-Cell Interconnection network (HHC) *(Cont.)*:

* These six-processors grouped in two triangles represents (forms) a one-dimensional HHC.
* A d_h-dimensional HHC is built by replacing each zero-dimension in a (d_h-1)-dimensional hyper-cube interconnection network by a one-dimensional HHC undirected graph.

HHC interconnection network



1-dimensional HHC.



Each zero dimension in a 2-dimensional hyper cube is replaced by one-dimensional HHC to form 3-dimensional HHC

Introduction

Optical transpose interconnection system (OTIS):

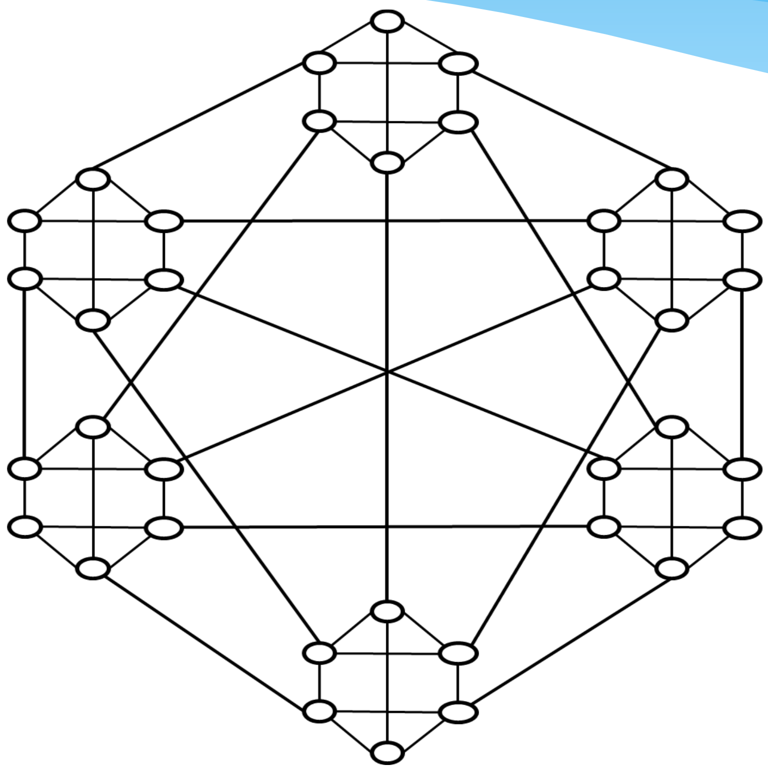
* An interconnection network with hybrid properties:
  * Uses optical and electrical communication links between processors
  * Benefit from the advantages of both types of links.
  * Electrical links are used for neighbor processors which have short distance between them
  * Processors that are apart and its communication links are distanced will get an optical links in order to benefit from its speed.
* This type of hybrid interconnection network is referred to by “optoelectronic architectures”

Introduction

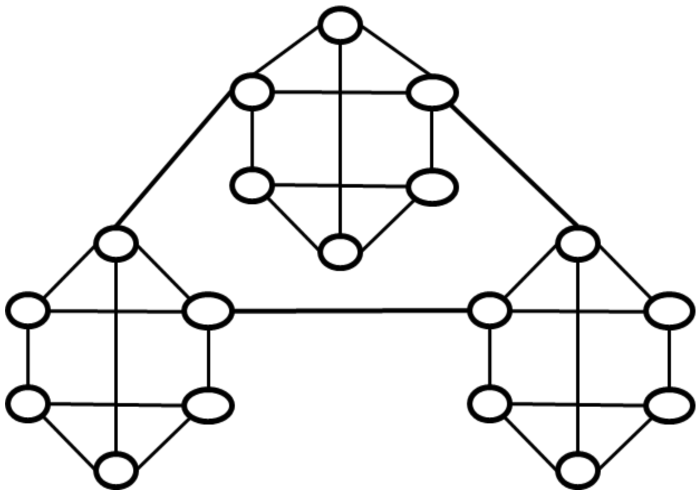
OTIS Hyper Hexa-Cell (OHHC):

* Optoelectronic architecture that connects a group of HHC using optical communication links between groups while keeping electrical communication links inside each group.
* Number of groups in the OHHC is determined by the number of processors in the HHC.:
  * When the number of groups G is equal to the number of processors P at each group; i.e. $G = P$.
  * When number of groups G is equal to half number of processors P at each group; i.e. $G = P/2$.

OTIS Hyper Hexa-Cell (OHHC)



One dimensional OHHC optoelectronic interconnection network when G=P.



One dimensional OHHC optoelectronic interconnection network when G=P/2.

Introduction

Parallel Algorithms:

* Parallel computing carries out calculations using multiple processing elements simultaneously.
* This is accomplished by breaking the problem into independent parts so that each processing element can execute its part of the algorithm simultaneously with the others.
* Operating on the principle that large problems can often be divided into smaller ones, which are then solved concurrently ("in parallel").

Introduction

Parallel Algorithms (Cont.):

* The development of parallel computing is encouraged by the evolving computational speed and power along with decreasing costs of memory and processor components.
* The research on parallel or multithreaded versions of sorting algorithm has gained an increasing share in the last decades.
* Many versions of parallel and multithreaded Quick Sort versions were introduced either for general or specific architecture type.

Problem Statement

* ***Applying*** Parallel Quick Sort (**QS**) Algorithm on ***OHHC*** interconnection network considering:
  * OHHC two cases; when ***G=P***, and when ***G=P/2***.
  * ***OHHC dimensions form 1 to 4***, for both OHHC cases.
* Evaluate the parallel QS version ***analytically***.
* Evaluate the Parallel QS version by ***simulation***.
* Simulation include running the Parallel QS version on input arrays with ***different sizes*** and ***different data distributions.***

The proposed parallel QS algorithm

* Major steps of the algorithm
  1. Split Main Array.
  2. Sort Sub Arrays.
  3. Data Accumulation
     a. HHC Level.
     b. Hypercube Level.
     c. OTIS Level.

The proposed parallel QS algorithm

* Splitting Main Array:
  1. Find minimum value; e.g. 100,000.
  2. Find maximum value; e.g. 900,000.
  3. Based on OHHC dimension; e.g. 1-d (G=6) → 36 processors
  4. Calculate divider = (Max-Min) / Number of Threads.
  5. Iterate main array, examining each element

  TargetArr = ArrElement / SubDivider; e.g. (354,234 / 22,222) == 15

  6. Send element to target sub array (15)
* All elements are already in the correct sub array.
* No additional measures is needed to merge after sort.

The proposed parallel QS algorithm

* After Sub-Arrays Creation :
  1. Each node sorts its designated sub-array.
  2. After sorting, start message passing.
  * Message passing is static, deterministic.
    * Based on the (OHHC) dimension.
  * Basically, each node waits then sends.
    * Based on its location in the architecture.

The proposed parallel QS algorithm

* Message Passing :
  1. Phase I

     All nodes except (OTIS) group (0).
  2. Phase II

     (OTIS) group (0) nodes.

The proposed parallel QS algorithm

* Message Passing *(Cont.)* :
  1. Phase I:
     a. Inner HHC MP.
     b. Intra HHC MP (Hypercube).
     c. OTIS Based MP.
  2. Phase II:

     OTIS Group (1) Starts Finalizing Data Accumulation.
     * Inner HHC MP.
     * Intra HHC MP (Hypercube).

  Done, all data at head node.

The proposed parallel QS algorithm

* Message Passing *(Cont.)* :
  * Based on HHC Group First Set Bit:
    * Calculate amount of sub-arrays.
    * Calculate Destination Node.
  * The same simple concept for all nodes.

processor->WaitForSubArrays(Number);

processor->(Destination);

Analytical Assessment

* Time Complexity.
* Number of computation steps.
* Number of communication steps.
* Speedup.
* Efficiency.
* Message delay.
* Key comparisons.

Analytical Assessment

* Time Complexity:
  * Complexity represents time spent by a given algorithm to complete its work measured by the number of steps needed.
  * Sequential version Of QS:
    * Worst case: $\Theta(n^2)$
    * Best and average $\Theta(n \log n)$
  * Average parallel time complexity Of QS:
    * $\Theta(n/P \log n/P)$.
  * Assuming:
    * All processors get almost the same share $(t = n/P)$.
    * Not considering the splitting ,distributing, and gathering of the array chunks

Analytical Assessment

- Number of communication steps:
  - Communication Steps represent the number of steps required by algorithm to:
    - Spread the array chunks from the head node to its destination at a specific processor.
    - Sending chunks back to the head processor after being sorted.
  - Steps inside HHC dimensions (each group) = $6*d_h-1$.
  - Sum of steps for all groups = = $G*(6*d_h - 1) = 6*G*d_h - G$.
  - Optical link steps = $G-1$.
  - Spreading steps sum = $6*G*d_h - G + G-1 = 6*G*d_h -1$
  - Spreading and gathering steps sum = $2(6*G*d_h -1)=$ $\mathbf{12*G*d_h -2}$

Analytical Assessment

* Speedup:
  * Measures the percentage of improvement in time complexity when using a parallel version of an algorithm by comparing it with the sequential version complexity.
  * The ratio of improvement is measured by the equation:

    Speedup (S) = Serial run time (T_S) / Parallel run time (T_P).
  * $T_S = \Theta(n \log n)$, and $T_P = \Theta(n/P \log n/P)$
  * Then: $T_S/T_P = \Theta((n \log n) / (n/P \log n/P))$

    $= \Theta((P/n)*n \log n / (\log n/P))$

    $= \Theta(P \log n / \log n/P)$

    $= \boldsymbol{\Theta(P \log n / (\log n - \log P))}$

Analytical Assessment

* Efficiency:
  * Presents the effectiveness of the parallel algorithm.
  * Ratio of the algorithm serial time to the time taken by all processors in parallel accumulated .
  * Efficiency (E) = Speedup (S) / number of processors (P).
  * Can be rewritten as E= Serial run time (T_S) / P * Parallel run time (T_P)

Analytical Assessment

- Efficiency (*Cont.*):
  - $E_f = T_S/P*T_P$. Which equals :

    $= \Theta((n \log n) / P*(n/P \log n/P))$

    $= \Theta((n \log n) / (n \log n/P))$

    $= \Theta((\log n) / (\log n/P))$

    $= \boldsymbol{\Theta((\log n) / (\log n - \log P))}$

Analytical Assessment

- Message Delay:
  - Time for a chunk of array data to be sent from the head node to the processor that will sort it sequentially or its way back
  - Assuming the average case where all chunks are almost equal, the message size (t) will equal n/P.
  - The longest path for a message is the diameter of the source group plus the diameter of the destination group plus one optical connection link from the source group to the destination group.

Analytical Assessment

- Message Delay (*Cont.*):
  - Total number of links is the diameter of the source group plus the diameter of the destination group plus one optical connection link
  - Number of links (L) = (2 * d_h +3)
  - Assuming we are using the cut through routine, the cost of Θ(t + L) is applied and the message delay will be Θ (t + (2 * d_h +3))
  - The worst case of partitioning , t ≈ n.
  - The average case of partitioning; t ≈ n/P

Analytical Assessment

- Message Delay (*Cont.*):
  - Total number of links is the diameter of the OHHC.
  - Number of links (L) = $(2 * d_h + 3)$
  - Assuming we are using the cut through routine, the cost of $\Theta(t + L)$ is applied and the message delay will be $\Theta(t + (2 * d_h + 3))$
  - The worst case of partitioning ; $t \approx n$: $\boldsymbol{\Theta (n + ((2 * d_h +3)}$.
  - The average case of partitioning; $t \approx n/P$: $\boldsymbol{\Theta (n/P + ((2 * d_h +3)}$

Analytical Assessment

Summary:

- Time Complexity: $\boldsymbol{\Theta (n/P \log n/P)}$.
- Number of communication steps: $\boldsymbol{12*G*d_h -2}$.
- Speedup: $\boldsymbol{\Theta(P \log n / (\log n - \log P))}$.
- Efficiency: $\boldsymbol{\Theta((\log n) / (\log n - \log P))}$
- Message delay:
  - Worst case of partitioning ; t ≈ n: $\boldsymbol{\Theta (n + ((2 * d_h +3)}$.
  - Average case of partitioning; t ≈ n/P: $\boldsymbol{\Theta (n/P + ((2 * d_h +3)}$

Simulation Environment

* Hardware
  * Intel Core i 7 @ 2.2Ghz dual (quad cores) machine.
  * 6 GB Ram.
* Software
  * Windows 7 OS 64 bit version.
  * GNU C++ version 4.7 using Code Blocks IDE.
  * Pthreads.

Simulation Environment

* Two main architectures of (OHHC) were examined:
  * The full OHHC optoelectronic architecture where the number of groups is equal to the number of processors in each group.
  * A minimized version of the OHHC optoelectronic architecture where the number of groups is half the number of the processors in each group.

Simulation Environment

* Using the two OHHC architectures, several experiments where performed on different OHHC dimensions, the 1-d, 2-d, 3-d and the four dimension.
* Different 216 runs where examine using different combinations

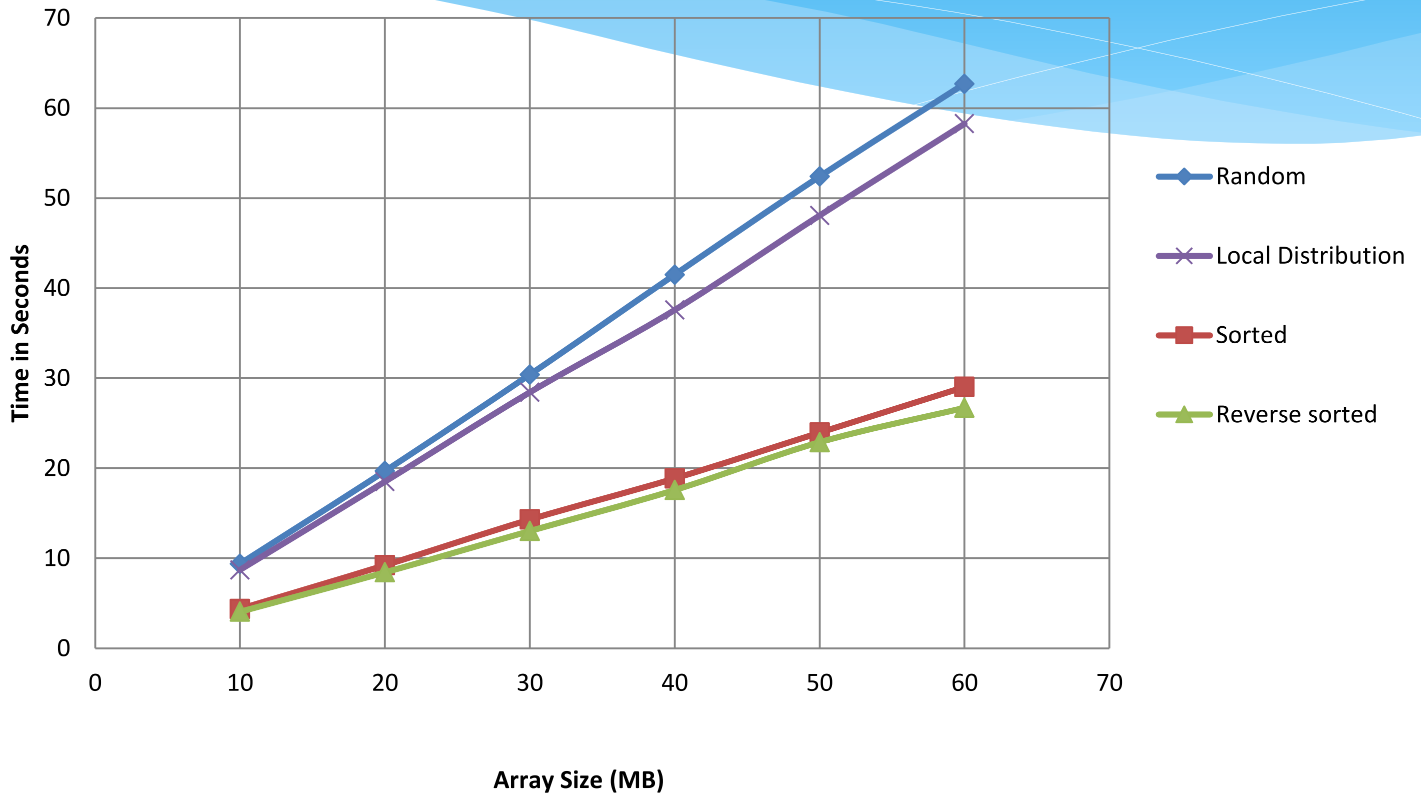
Simulation Environment

- Integer arrays were used as input data to the algorithm.
- Combination of the following variances where experimented:
  - Using different types of integer arrays (random generated arrays, sorted arrays, reverse sorted arrays, local distribution version of the input array).
  - Using different array sizes (10, 20, 30, 40, 50 and 60) MB arrays.

Simulation results

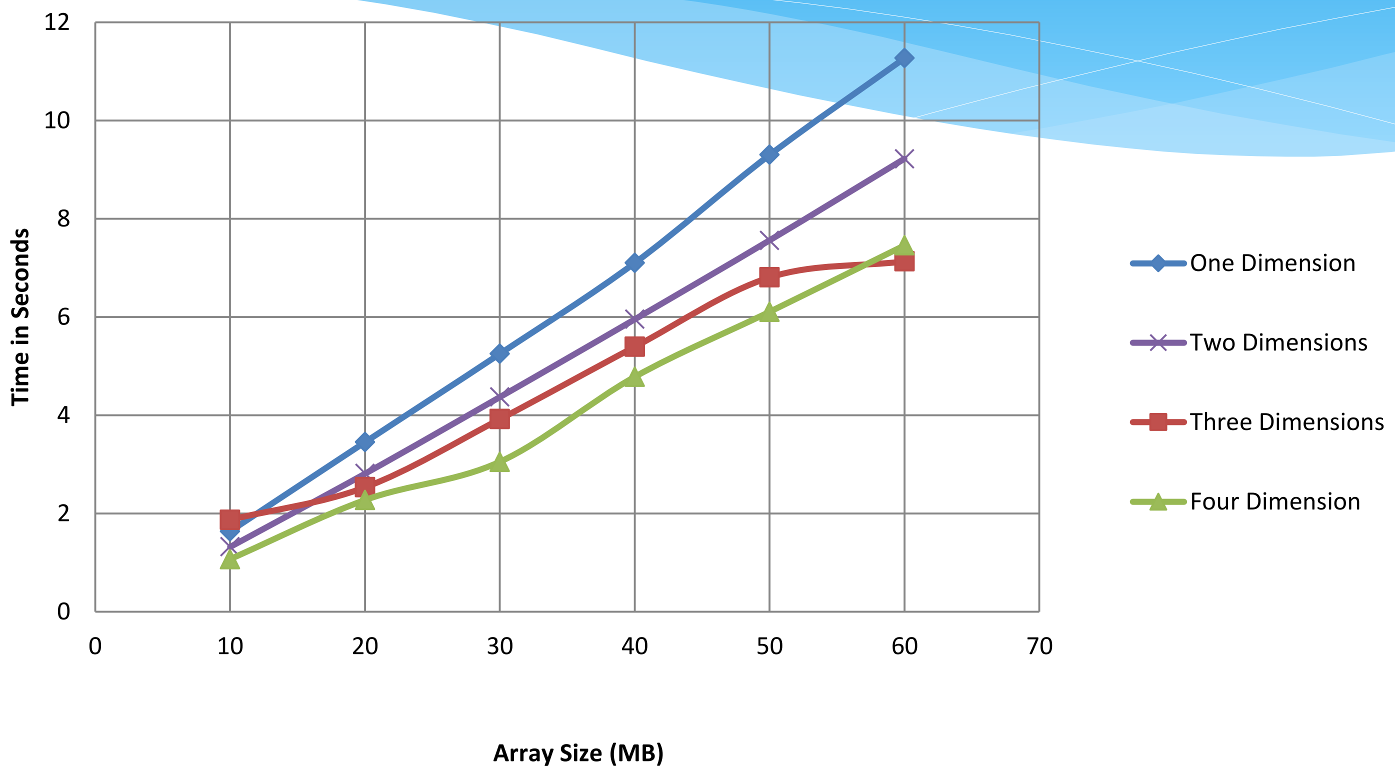
* Execution time.
* Speedup.
* Efficiency.
* Message delay.
* Key comparisons.

Simulation results



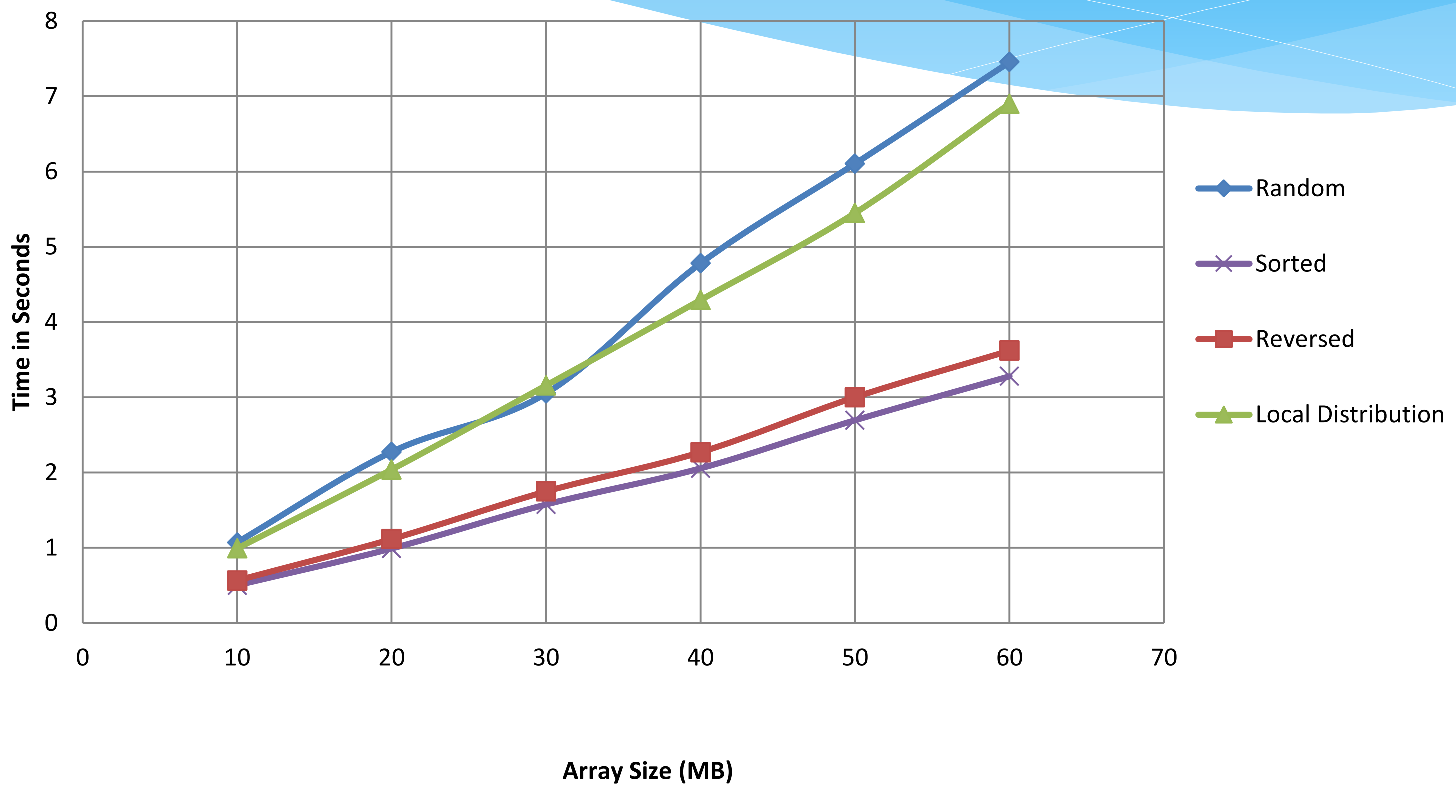
Running the sequential version of the algorithm over arrays of different types and different sizes

Simulation results



Run time of the Parallel Quick Sort algorithm over different OHHC dimensions using Random distribution

Simulation results



A Parallel run using a four-dimension network sorting integer arrays of different types and different sizes

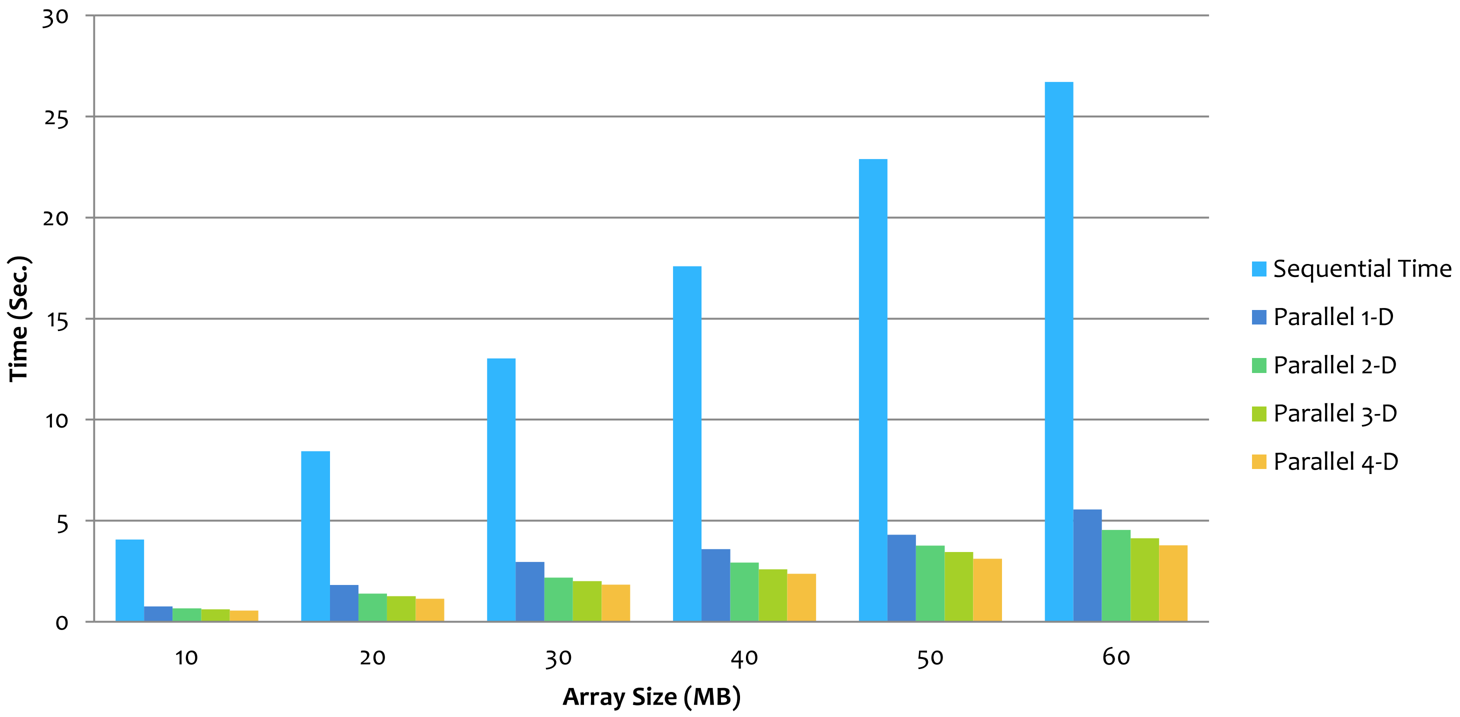
Simulation results

* Execution time when **G=P**

Random distribution					
Array Size (MB)	Sequential Time	Parallel 1-D	Parallel 2-D	Parallel 3-D	Parallel 4-D
10	9.39	1.636	1.32	1.87	1.069
20	19.68	3.45	2.81	2.53	2.272
30	30.4	5.25	4.369	3.919	3.05
40	41.5	7.1	5.951	5.392	4.78
50	52.4	9.3	7.559	6.807	6.105
60	62.7	11.27	9.219	7.132	7.456

Simulation results

Time comparison for Random distribution data input, G=P



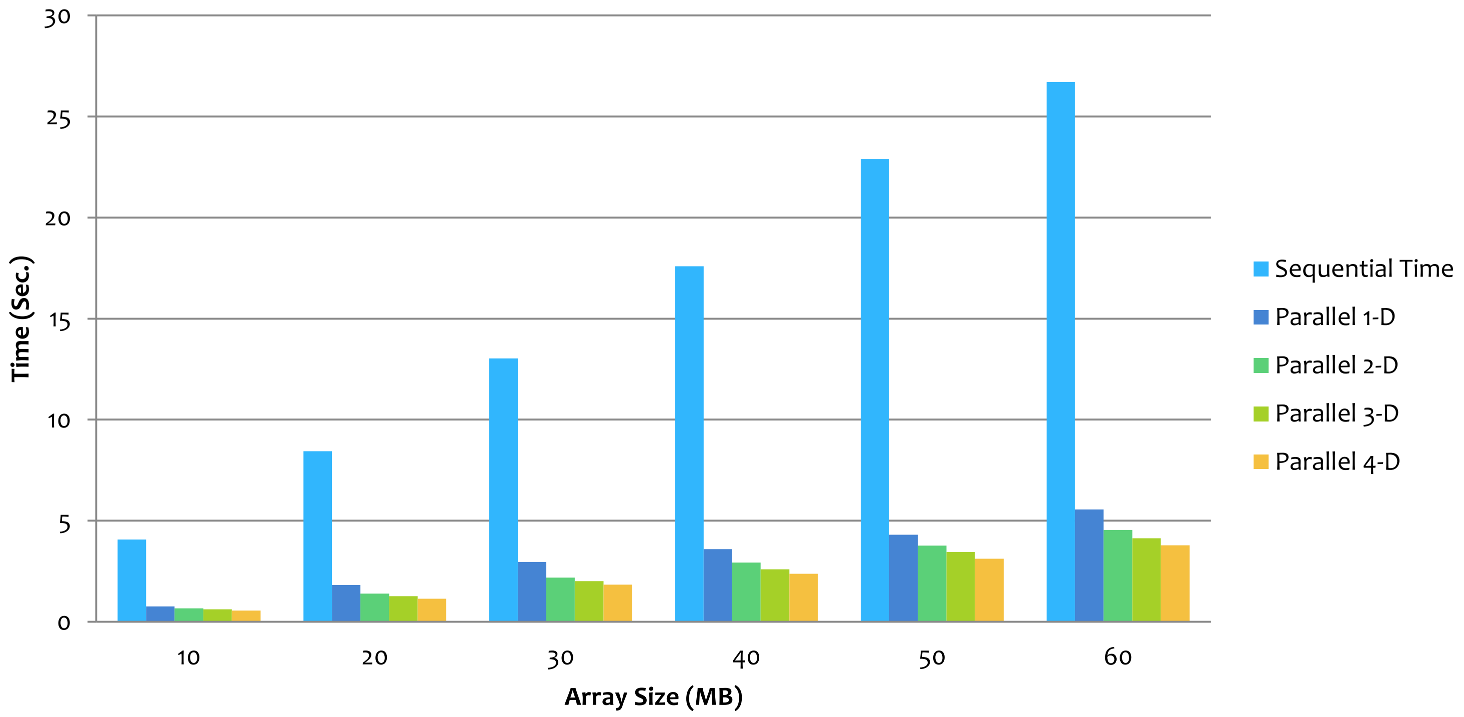
Simulation results

- Execution time when **G=P**

Sorted distribution					
Array Size (MB)	Sequential Time	Parallel 1-D	Parallel 2-D	Parallel 3-D	Parallel 4-D
10	4.377	0.699	0.588	0.528	0.501
20	9.227	1.511	1.205	1.094	0.99
30	18.873	2.363	1.927	1.74	1.575
40	18.873	3.184	2.506	2.256	2.056
50	23.958	3.899	3.272	2.98	2.695
60	29.03	4.676	3.987	3.645	3.279

Simulation results

Time comparison for Sorted distribution data input, G=P

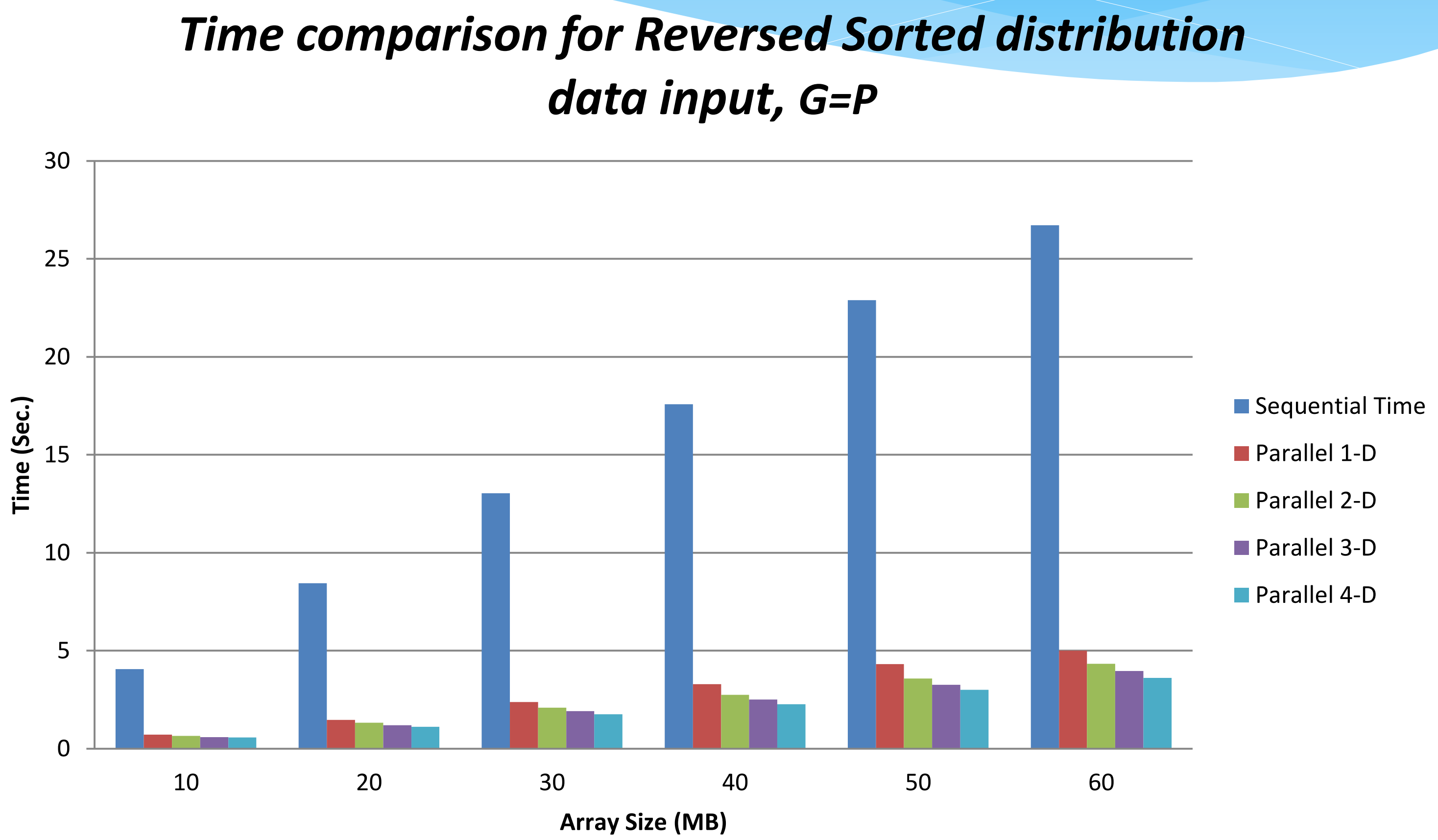


Simulation results

* Execution time when **G=P**

Reversed Sorted distribution					
Array Size (MB)	Sequential Time	Parallel 1-D	Parallel 2-D	Parallel 3-D	Parallel 4-D
10	8.441	1.461	1.323	1.199	1.114
20	13.034	2.378	2.093	1.915	1.748
30	17.586	3.297	2.744	2.5	2.27
40	22.897	4.311	3.582	3.26	3.001
50	26.716	5.01	4.332	3.966	3.619
60	8.441	1.461	1.323	1.199	1.114

Simulation results

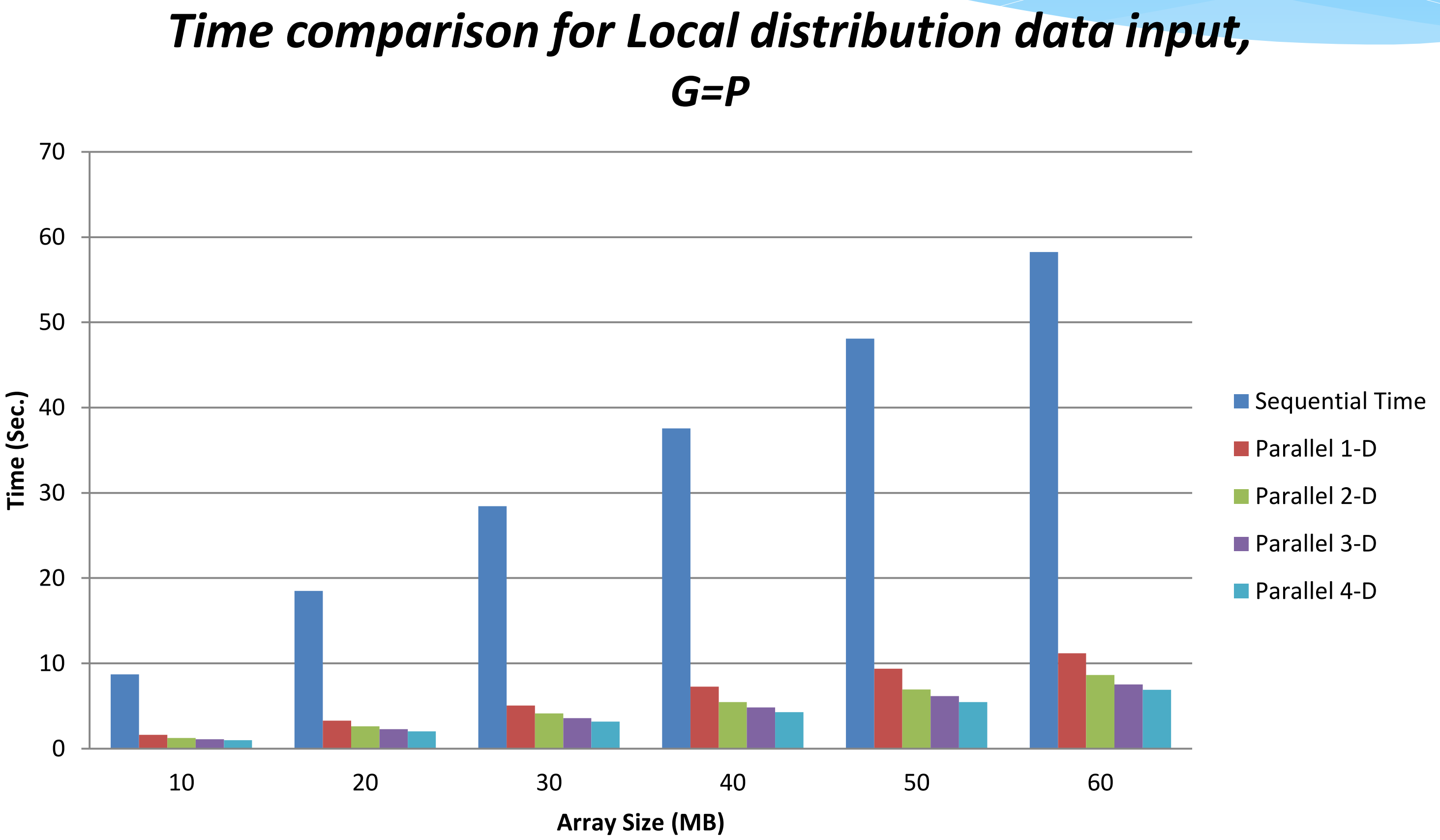


Simulation results

- Execution time when **G=P**

Local distribution					
Array Size (MB)	Sequential Time	Parallel 1-D	Parallel 2-D	Parallel 3-D	Parallel 4-D
10	8.716	1.619	1.252	1.097	0.992
20	18.511	3.303	2.613	2.302	2.04
30	28.432	5.057	4.139	3.574	3.158
40	37.579	7.26	5.47	4.854	4.292
50	48.073	9.38	6.927	6.177	5.448
60	58.258	11.188	8.629	7.525	6.9

Simulation results



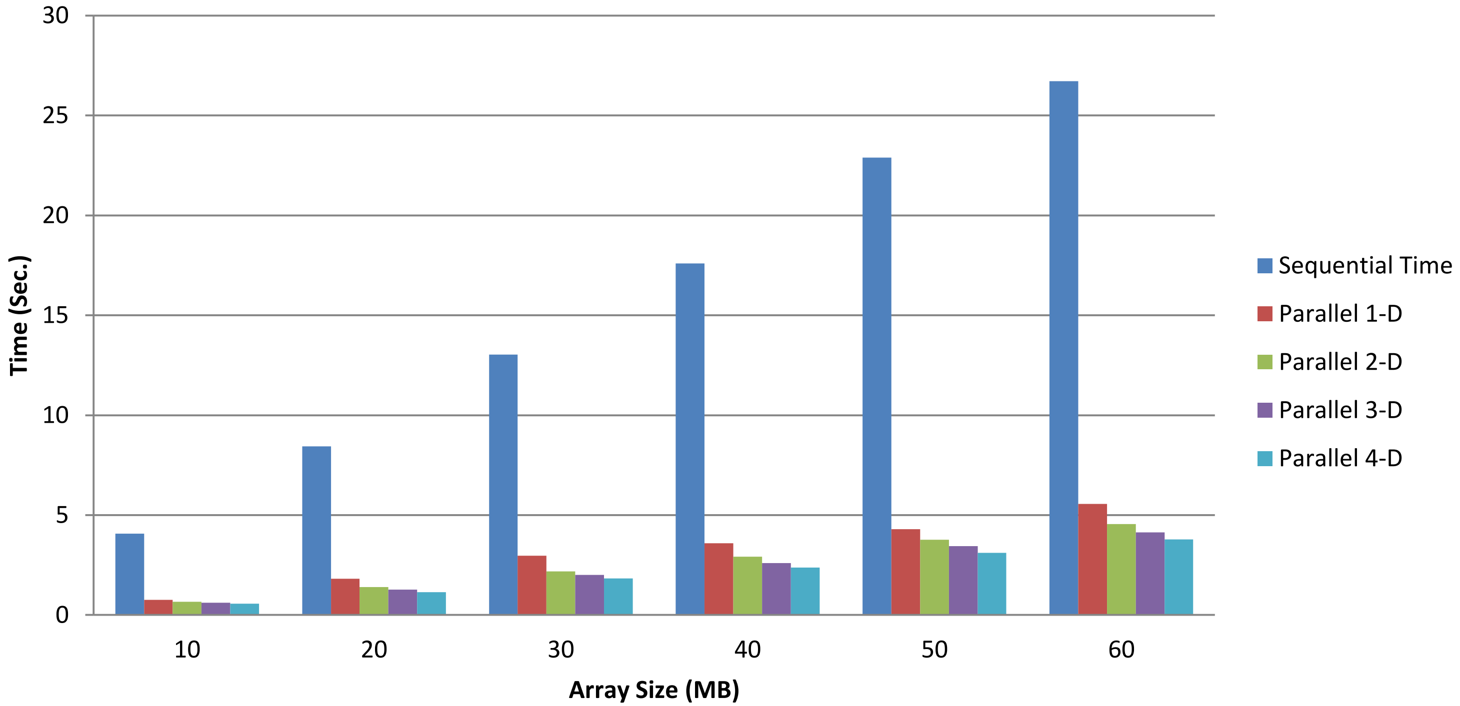
Simulation results

* Execution time when **G=P/2**

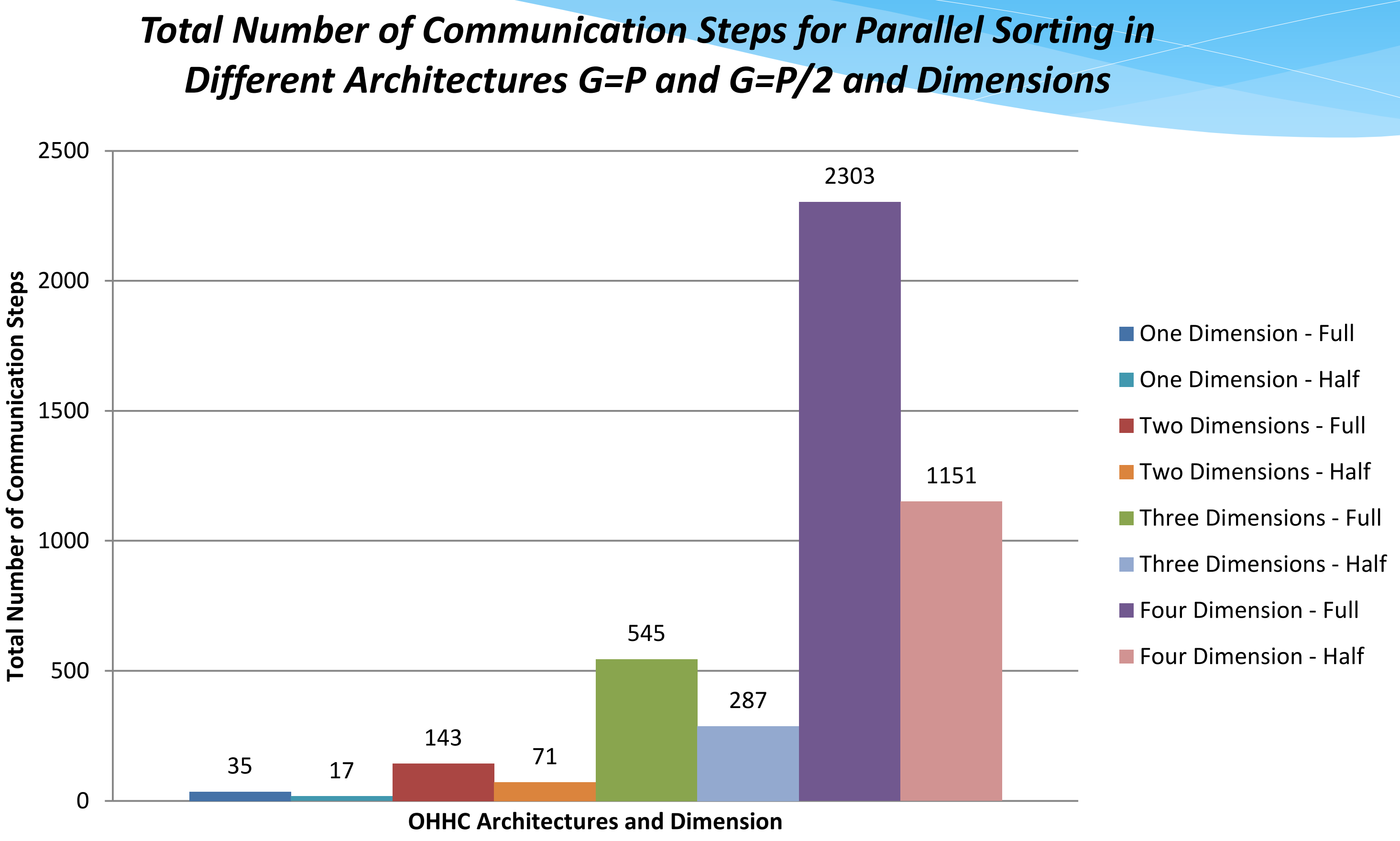
Local distribution					
Array Size (MB)	Sequential Time	Parallel 1-D	Parallel 2-D	Parallel 3-D	Parallel 4-D
10	4.067	0.765	0.669	0.613	0.561
20	8.441	1.819	1.396	1.27	1.145
30	13.034	2.963	2.186	2.006	1.834
40	17.586	3.596	2.926	2.607	2.372
50	22.897	4.303	3.77	3.453	3.117
60	26.716	5.563	4.55	4.13	3.779

Simulation results

Time comparison for Reversed Sorted data input, G=P/2

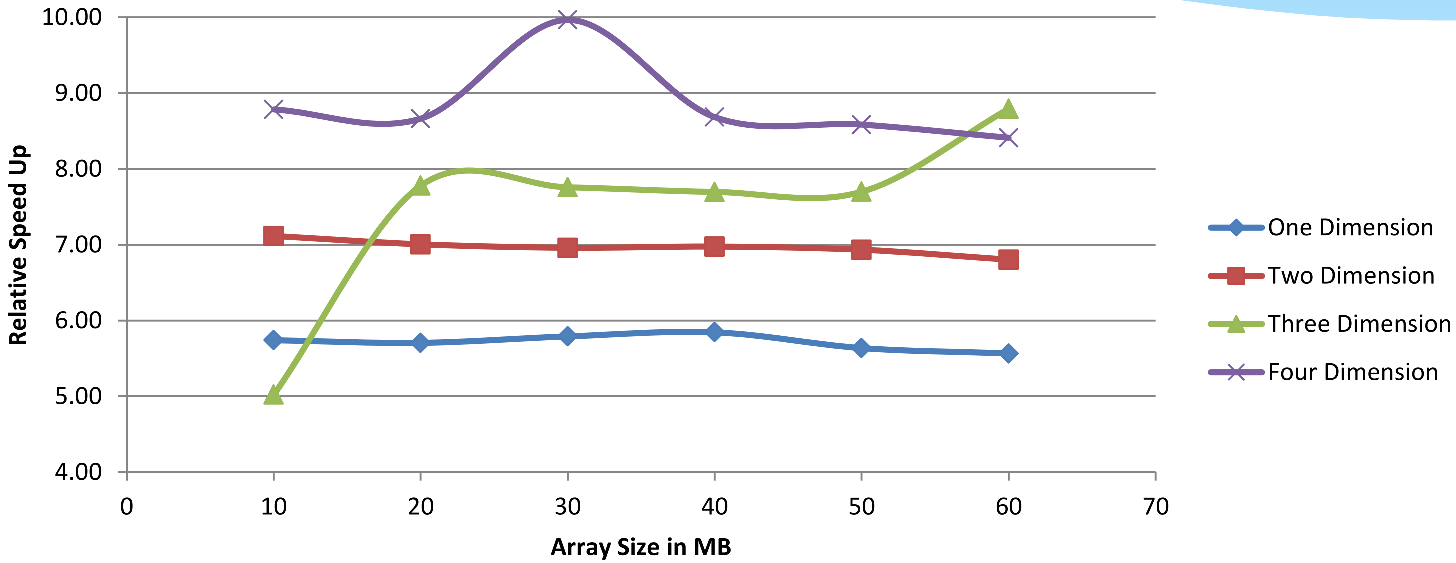


Simulation results



Simulation results

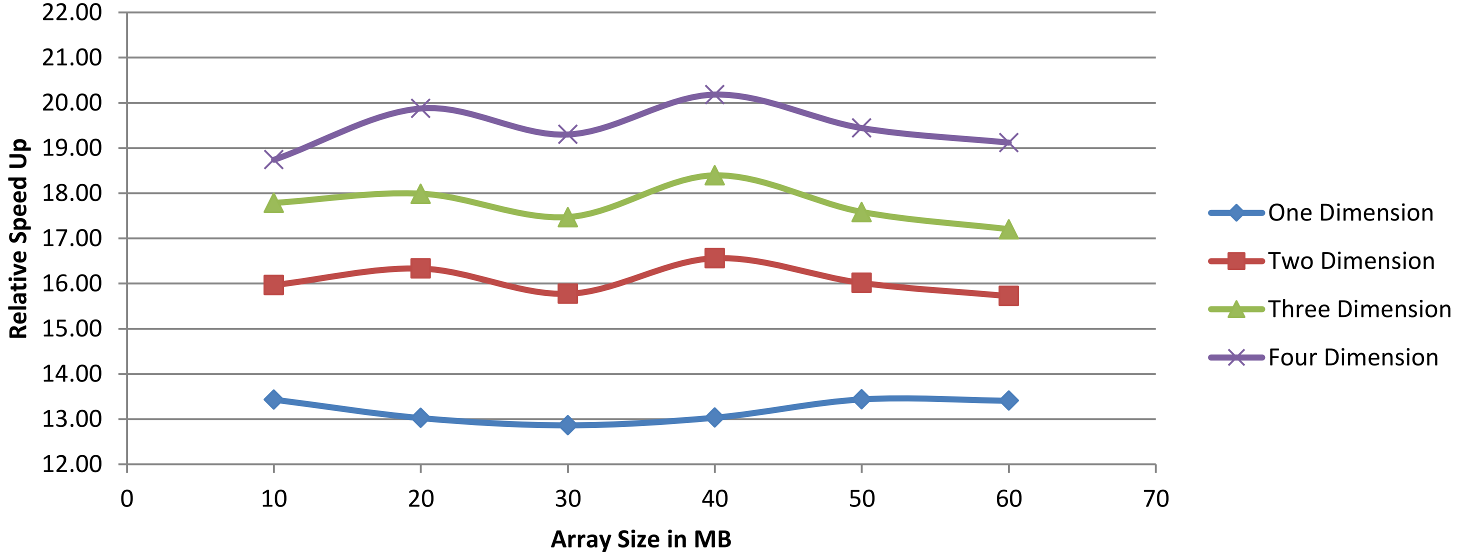
* Relative speed Up when $G = P$



Relative Speedup when G=P using Random distribution for different OHHC dimensions

Simulation results

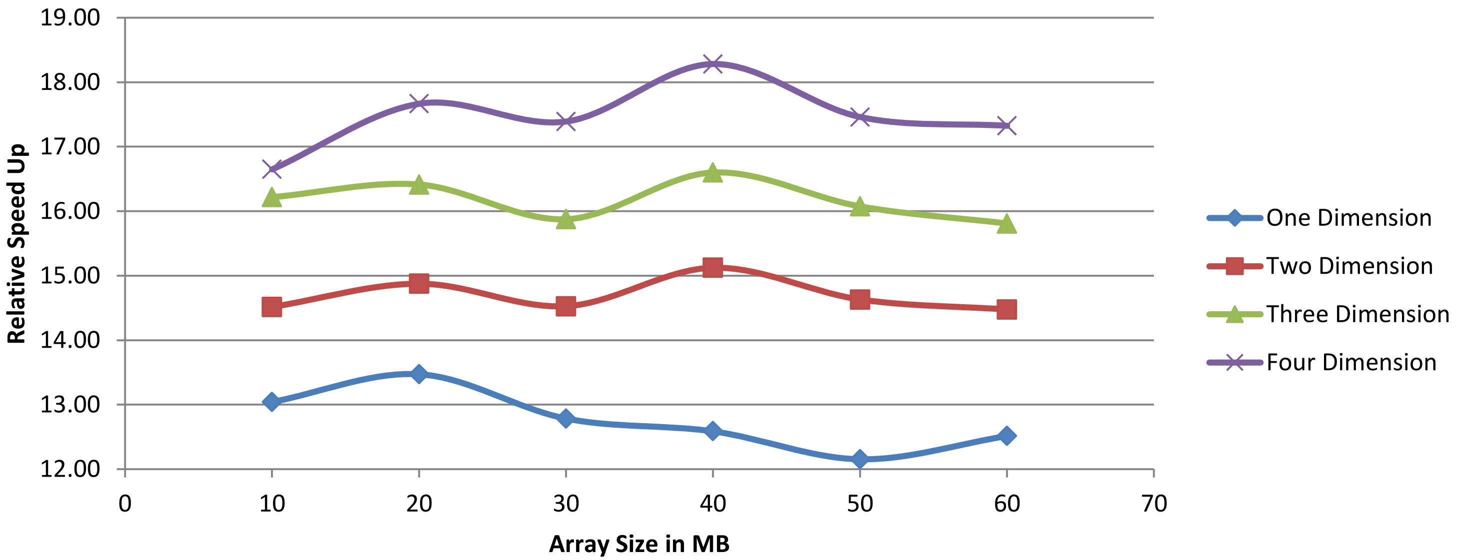
* Relative speed Up when *G = P*



Relative Speedup when G=P using sorted distribution for different OHHC dimensions

Simulation results

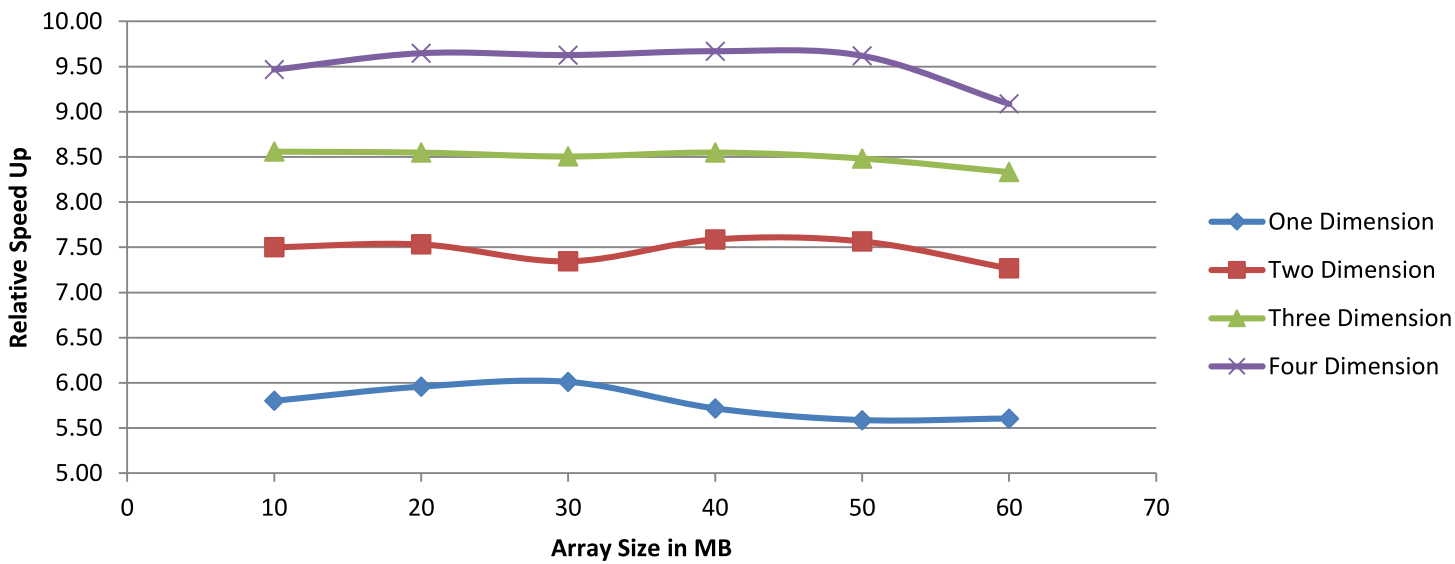
- Relative speed Up when $G = P$



Relative Speedup when G=P using reversed sorted distribution for different OHHC dimensions

Simulation results

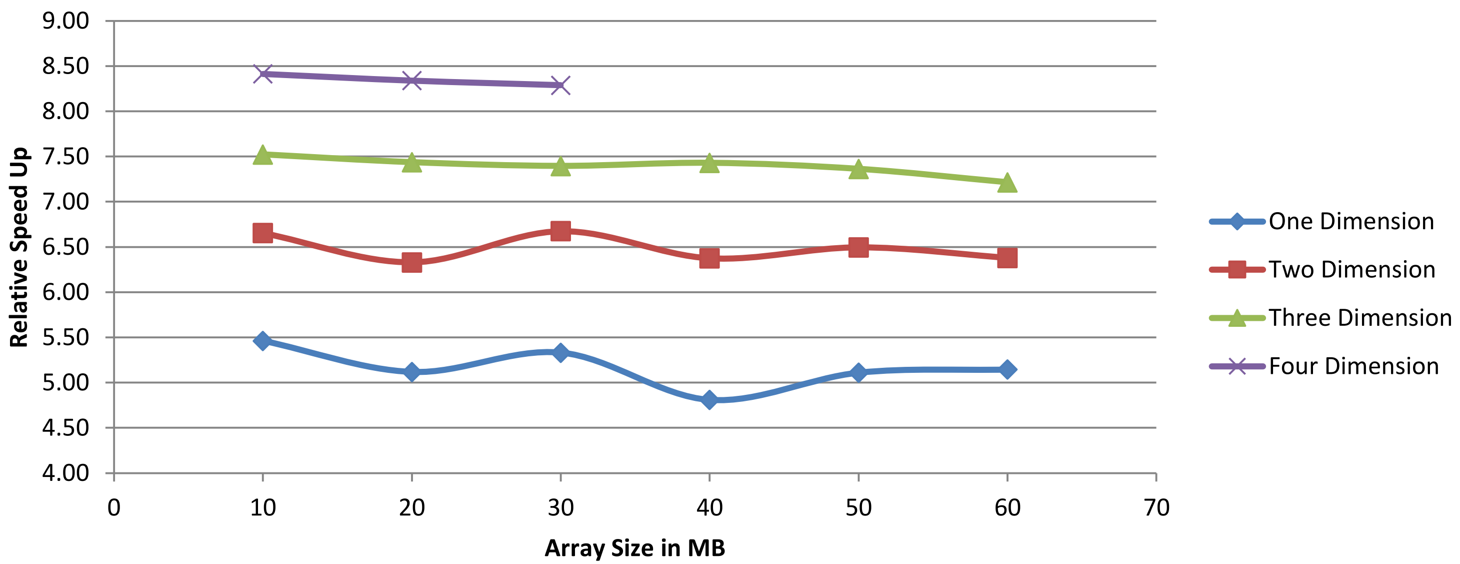
* Relative speed Up when $G = P$



Relative Speedup when G=P using local distribution for different OHHC dimensions

Simulation results

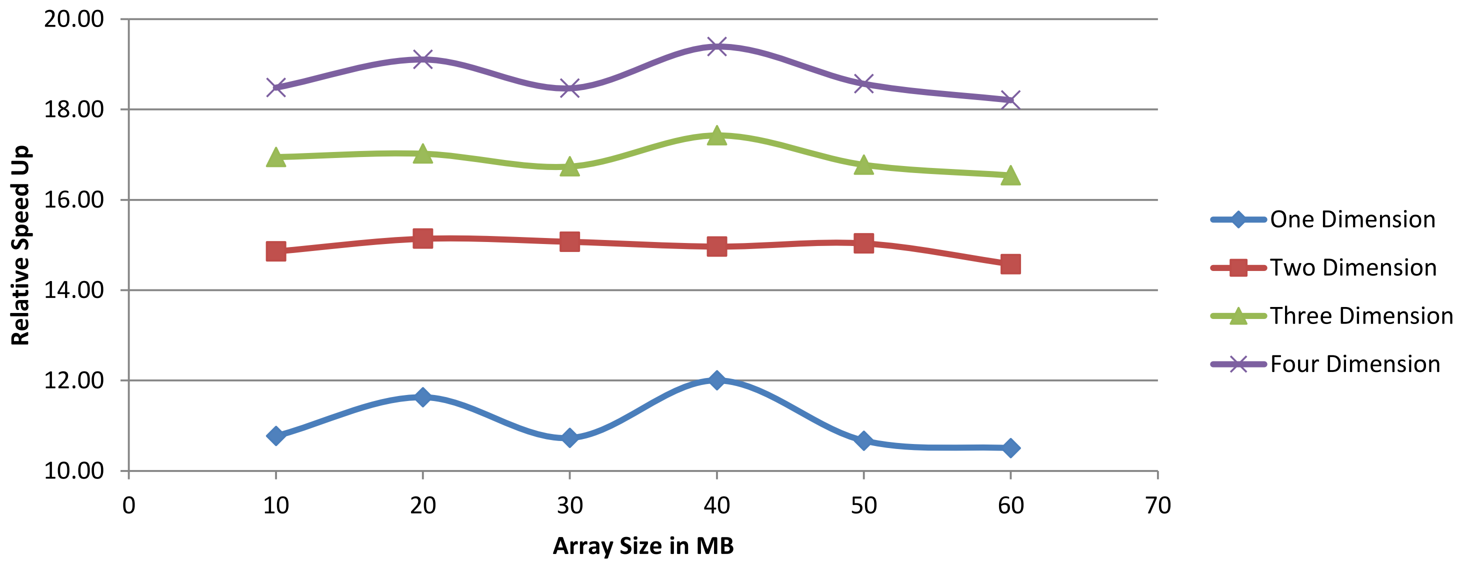
* Relative speed Up when G = *P*/2



Relative Speedup when G=P/2 using Random distribution for different OHHC dimensions

Simulation results

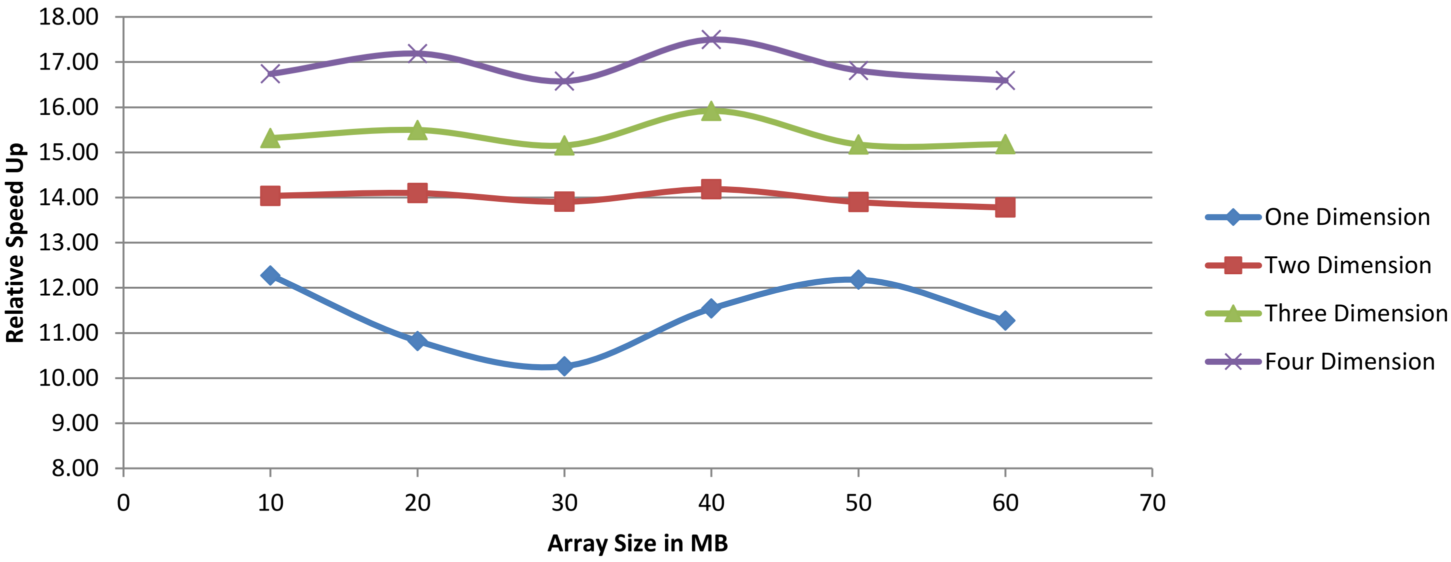
- Relative speed Up when G = *P*/2



Relative Speedup when G=P/2 using Sorted distribution for different OHHC dimensions

Simulation results

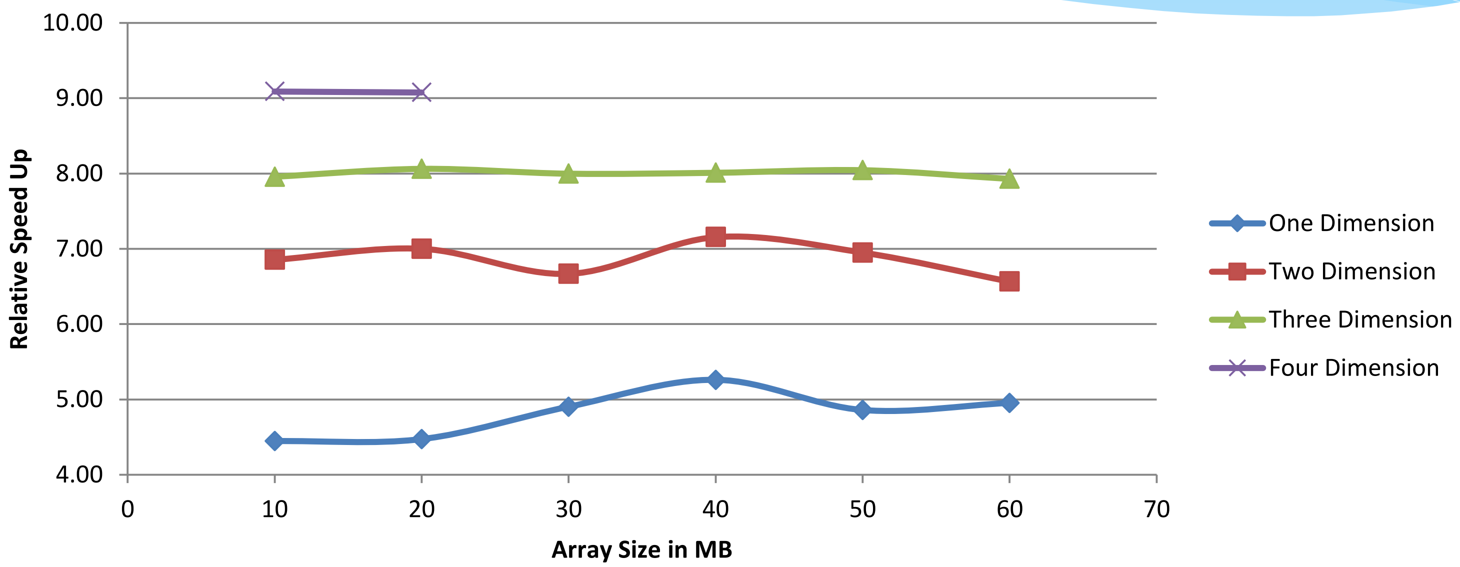
* Relative speed Up when G = P/2



Relative Speedup when G=P/2 using Reversed Sorted distribution for different OHHC dimensions

Simulation results

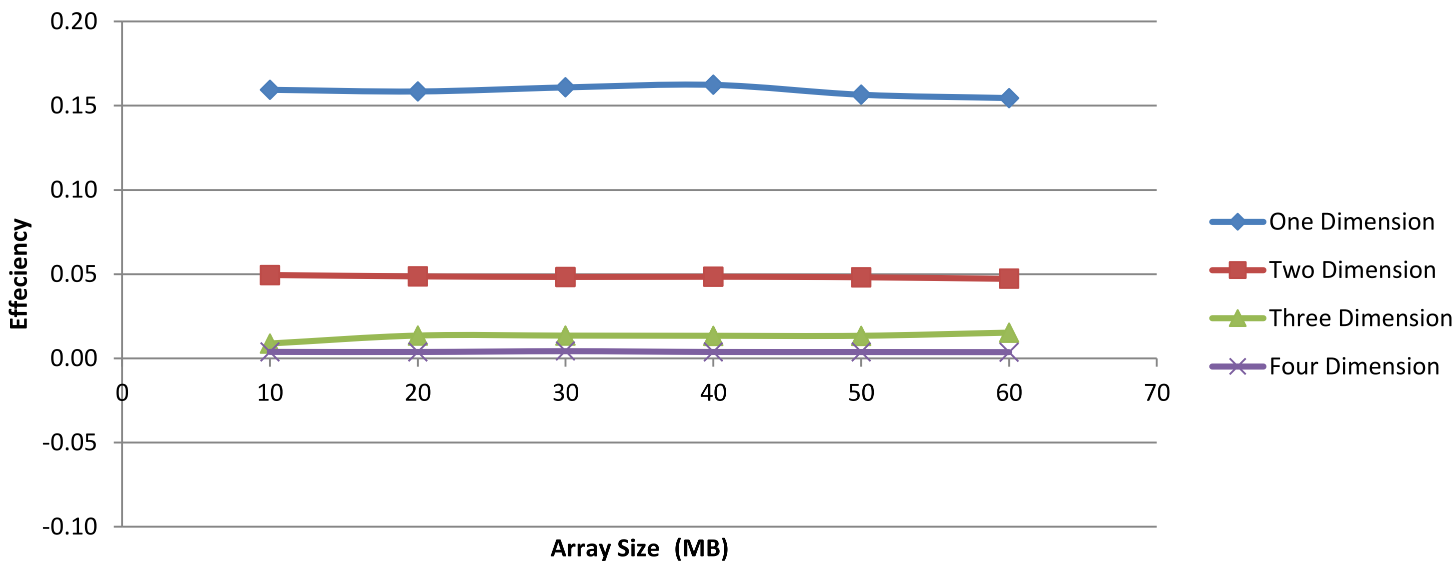
* Relative speed Up when G = P/2



Relative Speedup when G=P/2 using Local distribution for different OHHC dimensions

Simulation results

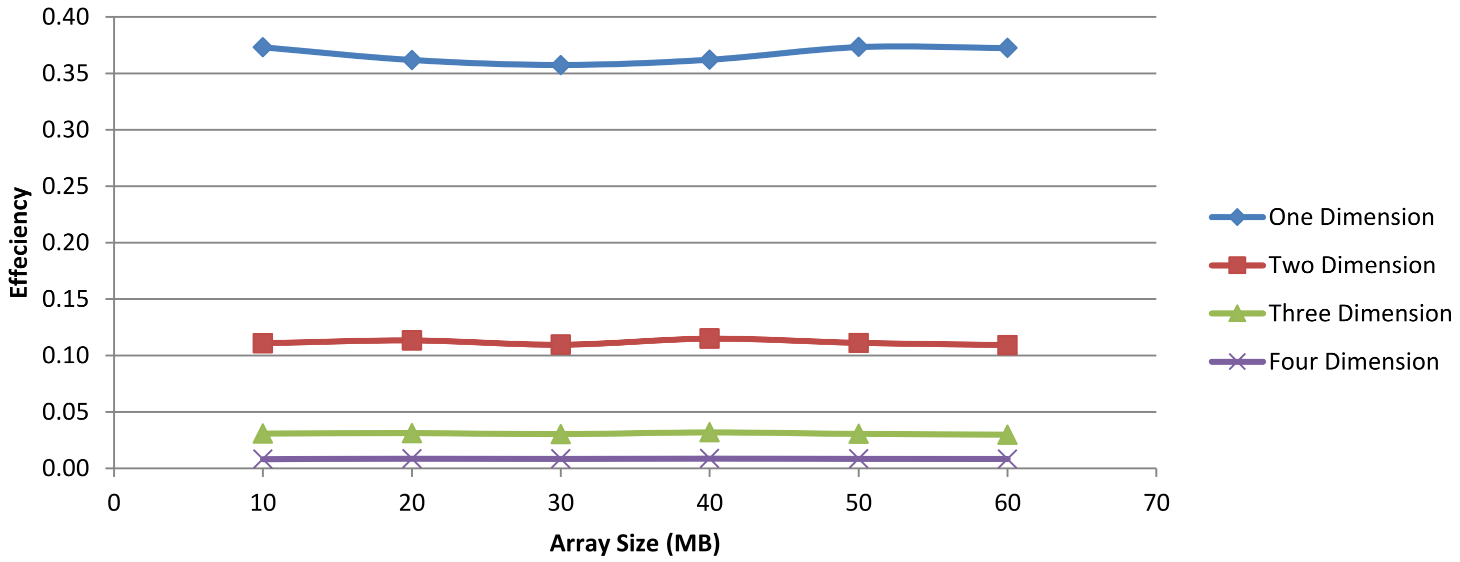
* Efficiency when *G* = *P*



Efficiency ratio when G=P using Random distribution for different OHHC dimensions.

Simulation results

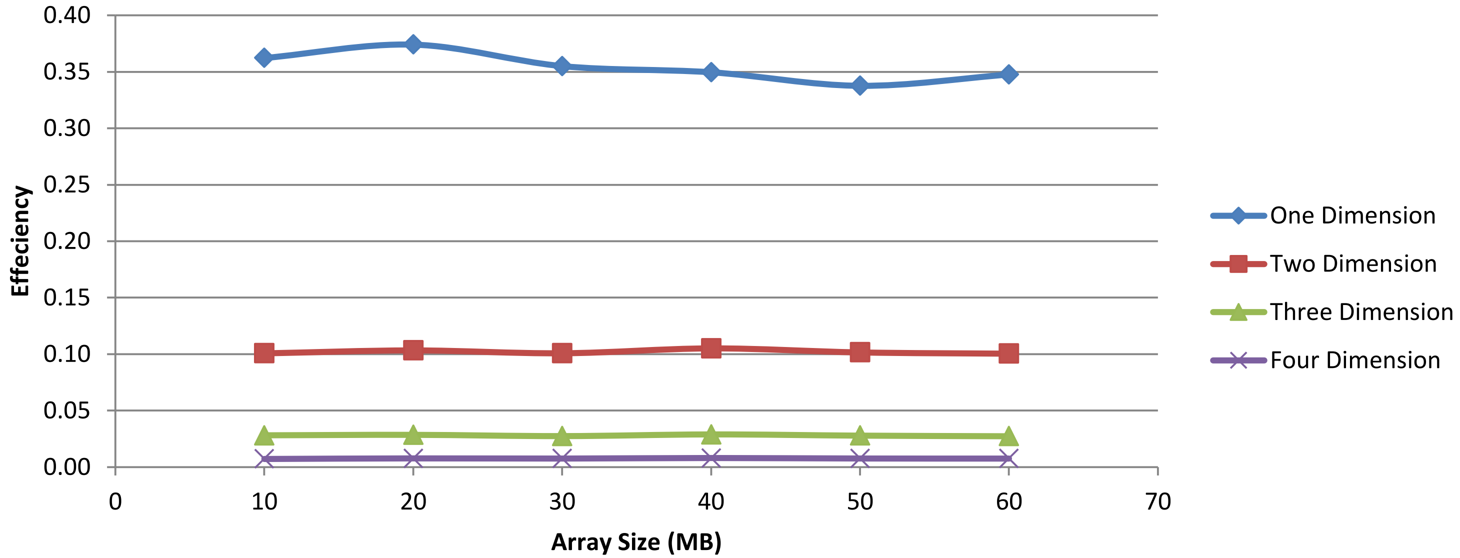
* Efficiency when *G* = *P*



Efficiency ratio when G=P using Sorted distribution for different OHHC dimensions.

Simulation results

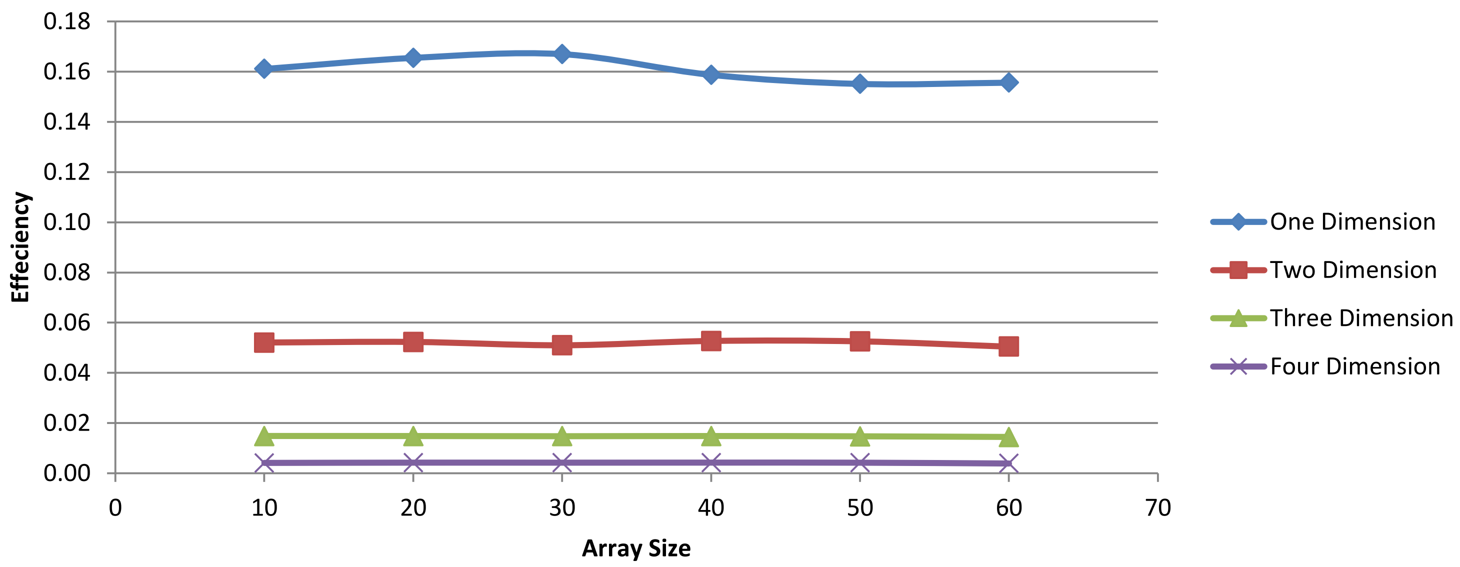
* Efficiency when *G* = *P*



Efficiency ratio when G=P using Reversed Sorted distribution for different OHHC dimensions.

Simulation results

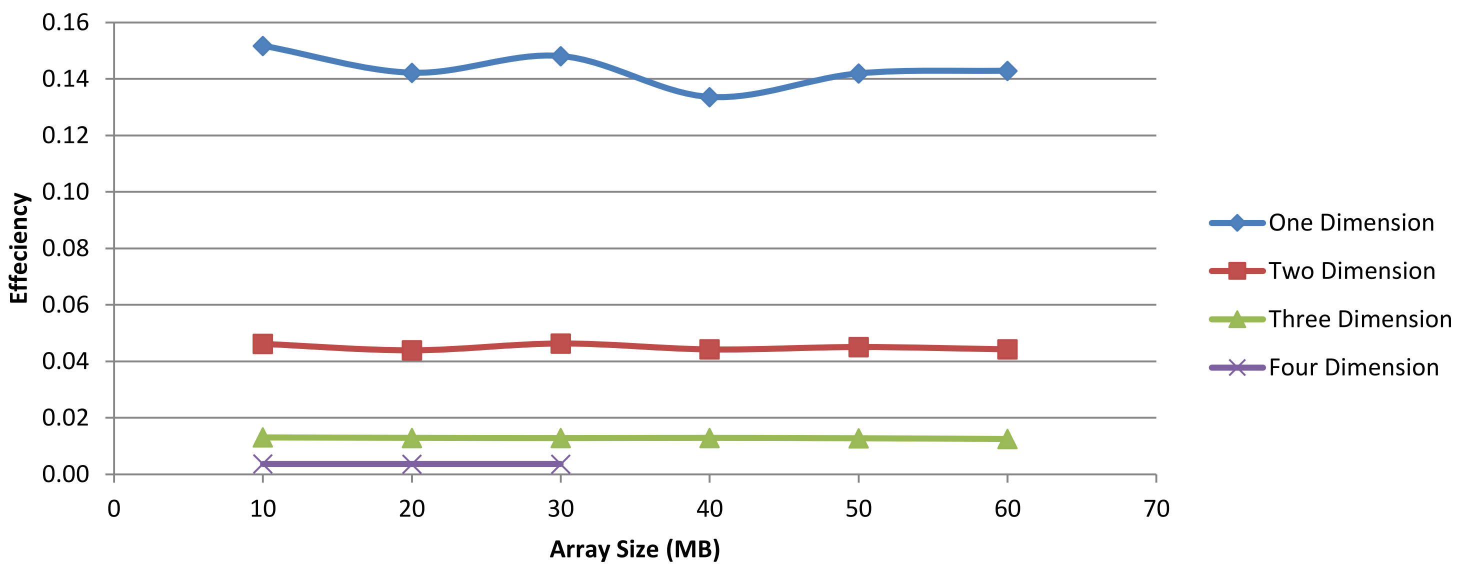
- Efficiency when *G = P*



Efficiency ratio when G=P using Local distribution for different OHHC dimensions.

Simulation results

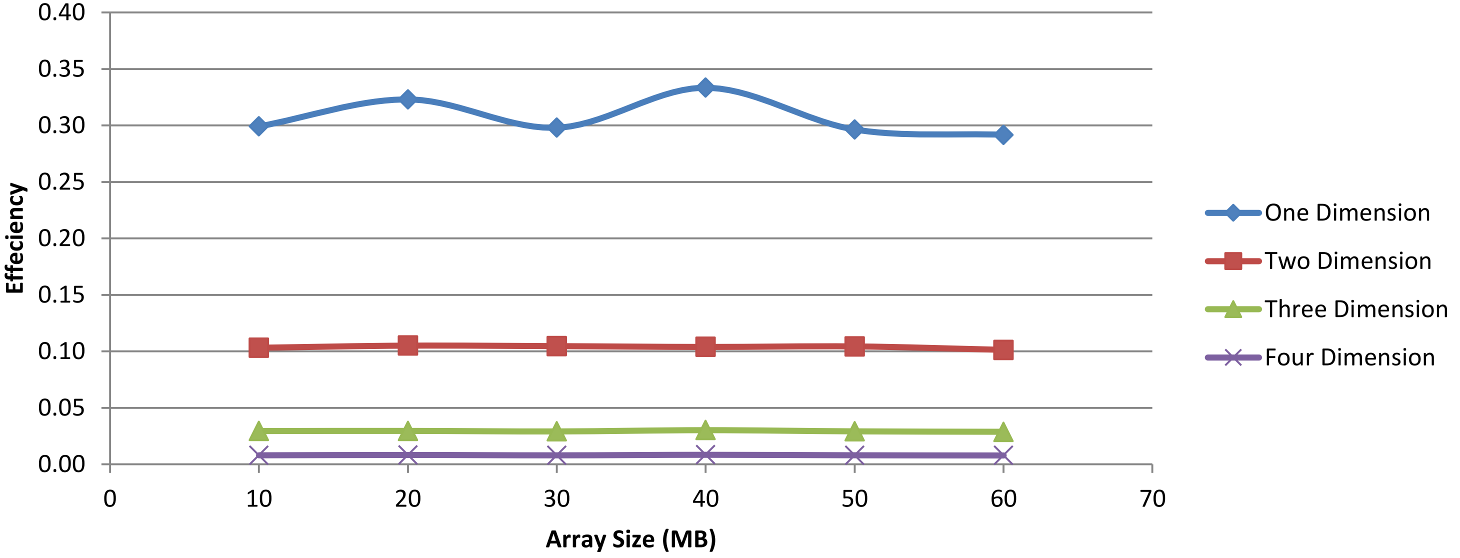
* Efficiency when G = *P*/2



Efficiency ratio when G=P/2 using Random distribution for different OHHC dimensions.

Simulation results

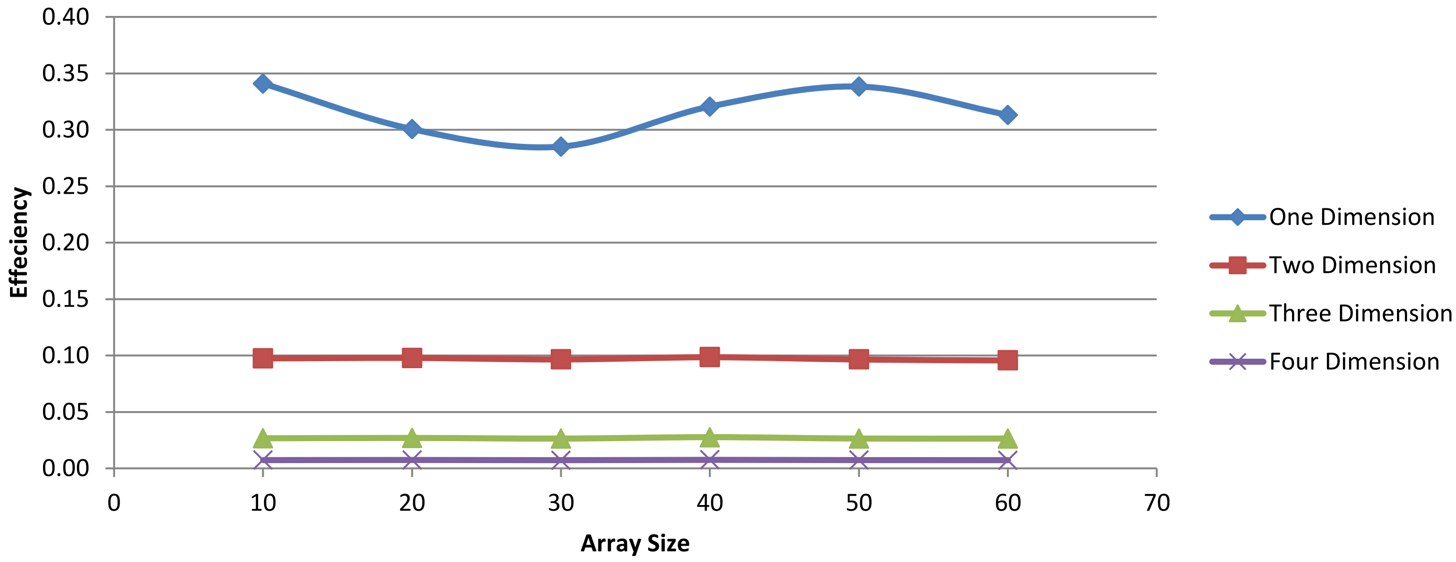
* Efficiency when G = *P*/2



Efficiency ratio when G=P/2 using Sorted distribution for different OHHC dimensions.

Simulation results

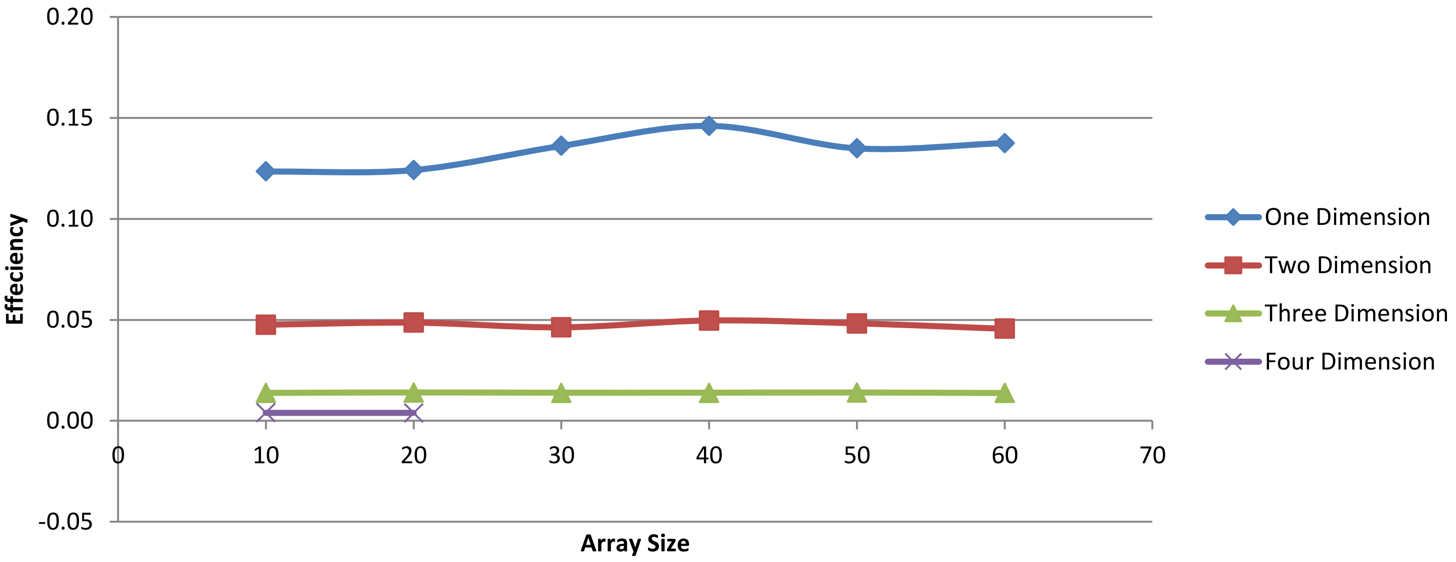
* Efficiency when G = *P*/2



Efficiency ratio when G=P/2 using Reversed Sorted distribution for different OHHC dimensions.

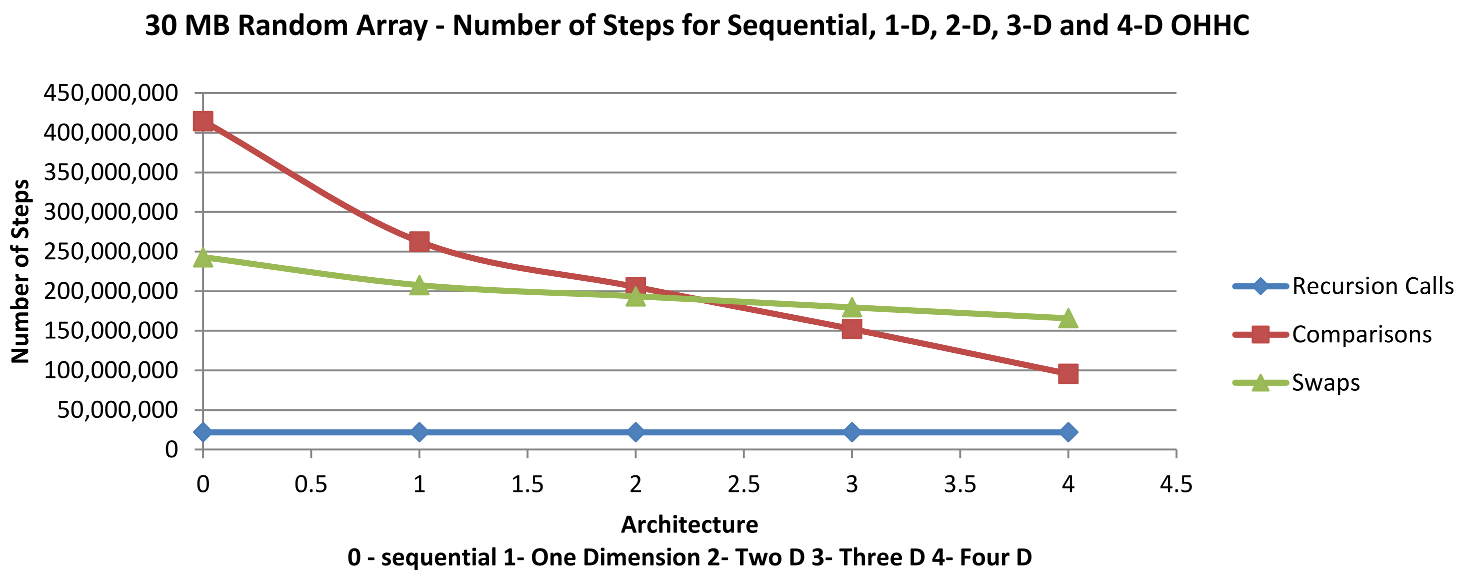
Simulation results

* Efficiency when G = *P*/2



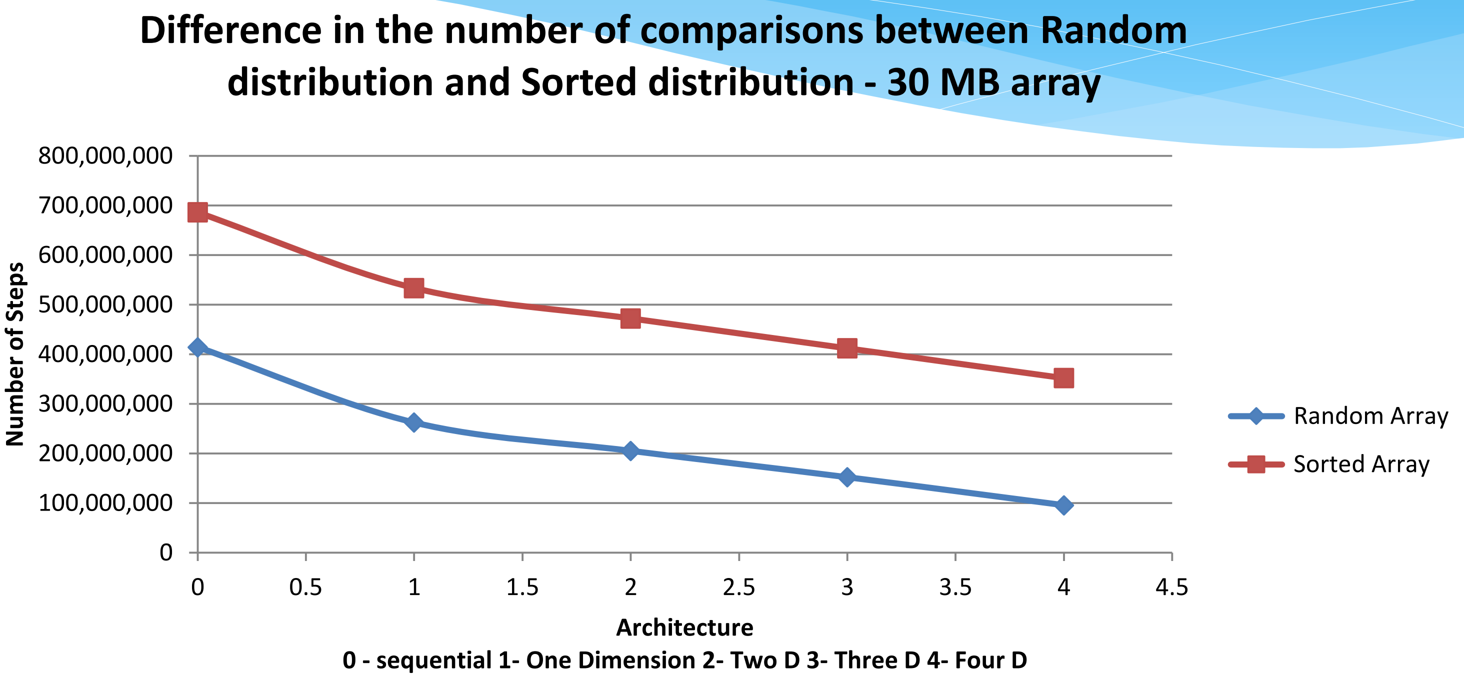
Efficiency ratio when G=P/2 using Local distribution for different OHHC dimensions.

Simulation results



Comparisons, Swaps and Recursion Calls

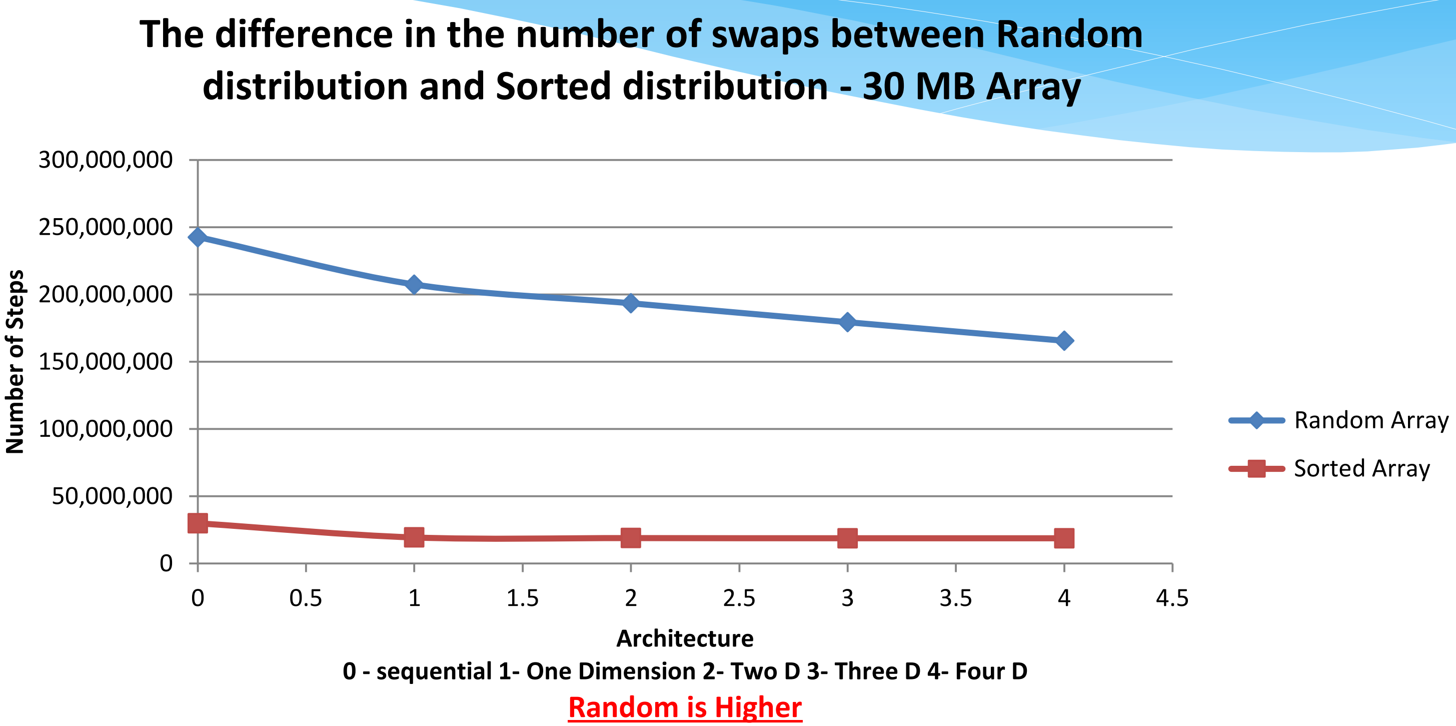
Simulation results



Random is lower

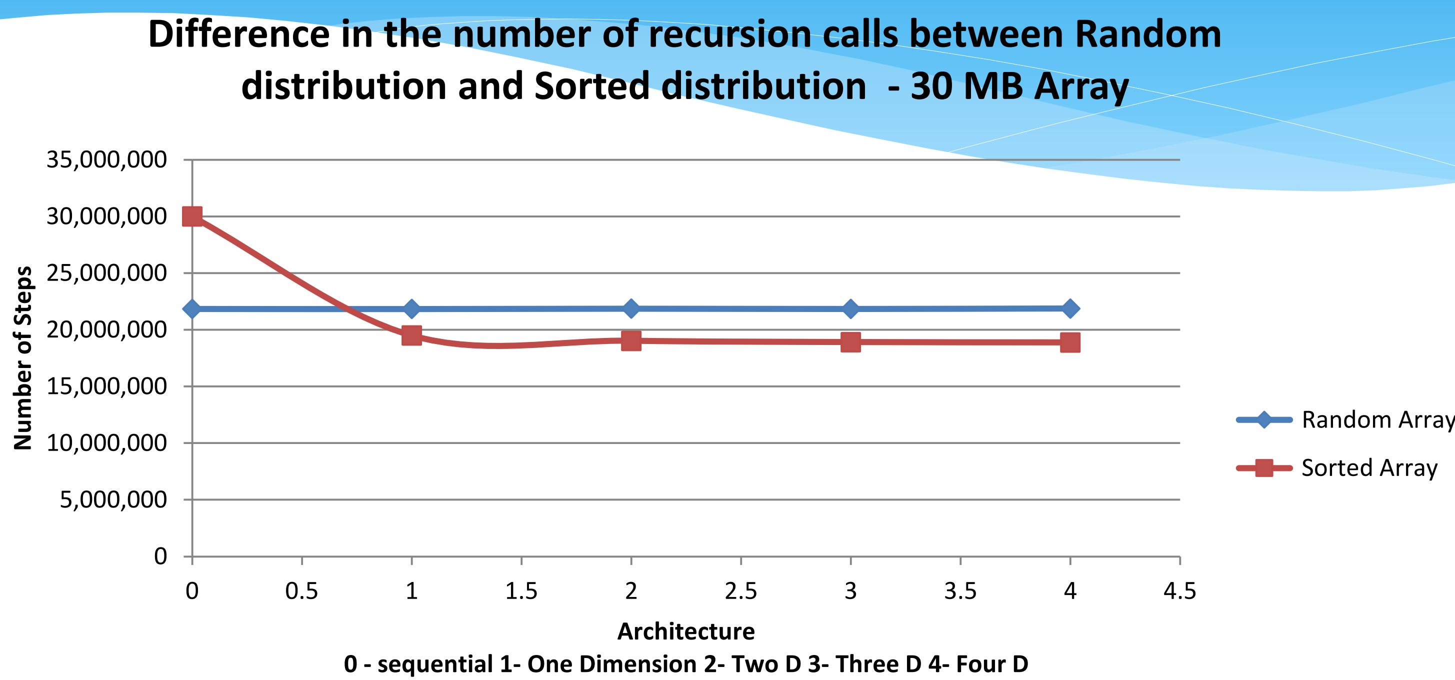
Comparing Random and Sorted distributions for the number of comparisons.

Simulation results



Comparing Random and Sorted distributions for the number of swaps.

Simulation results



Both are Close

Comparing Random and Sorted distributions for the number of recursions.

Implementation Observations

- Vector vs. int * performance.
- Thread barrier limitations.
- 32-bit environment limitations.
- Speedup and Efficiency should increase more in real parallel environments.

Conclusions

* Increasing the number of processors will decrease the time needed to sort the data arrays, thus increasing the speedup.
* However, the efficiency gets very low when the number of processors increase (moving to higher dimensions).
* Sorted and reversed sorted distribution showed similar run times for all dimension, Random and Local distributions are similar too.

Conclusions

* For both the sequential and parallel algorithm versions, Sorted and reversed sorted distributions took only about half the time taken the Local and Random distributions