



**VIT<sup>®</sup>**

**Vellore Institute of Technology**

(Deemed to be University under section 3 of UGC Act, 1956)

# **IMPLEMENTING DISCRETE CUCKOO SEARCH ALGORITHM FOR TSP USING MPI AND BEOWULF CLUSTER**

**HIGH PERFORMANCE COMPUTING (CSE4014)  
PROJECT**

By,

**Mahale Shantanu**

**17BCE1161**

Submitted To,

**S. Harini**

**November, 2019**

## CERTIFICATE

This is to certify that the Project Based Learning entitled “**Implementing Discrete Cuckoo Search Algorithm for TSP using MPI and Beowulf Cluster**” that is being submitted for CAL in B. Tech High Performance Computing (CSE4014) is a record of bonafide work done under my supervision. The contents of this Project, in full or in parts, have neither been take from any other source nor have been submitted for any other CAL course.

**Place:** Chennai

**Date:** 08<sup>th</sup> November 2019

**Signature of Students:**

Kumar Shikhar	17BCE1198
Abhijit Pingle	17BCE1254
Vinay C Shekhar	17BCE1317

**Signature of Faculty:**

**Ayesha**

School of Computer Science and Engineering (SCSE)

## ABSTRACT

This project is an implementation of a research paper based on High Performance Computing which was published on 8<sup>th</sup> of June, 2019 at International Journal of Innovative Technology and Exploring Engineering (IJITEE) by

Bhavana V, Varshini Ramesh (students of SCSE, VIT Chennai) and Sivagami M (Professor at VIT Chennai).

The main aim of the project is to implement the proposed idea from the paper.

The main aim of the paper was to find a better-quality solution for Travelling Salesman Problem using Cuckoo Search Algorithm efficiently. Parallelizing to be done to increase the quality of result. To implement parallelism across the nodes, MPI to be used. A Beowulf cluster to be created to achieve parallelism. Combinatorial optimization problems are typically NP-hard, and thus very challenging to solve. In the paper, the random key cuckoo search (RKCS) algorithm for solving the famous Travelling Salesman Problem (TSP) is discussed. A simplified random-key encoding scheme to pass from a continuous space (real numbers) to a combinatorial space is to be used. The Discrete Cuckoo Search optimization code is to be run on a single system with multiple cores and on the Beowulf cluster.

A thorough analysis has been performed to assess the efficacy of this optimization and the results are compared from the results in the paper and the reasons for such deviations is concluded.

## INTRODUCTION

Many combinatorial optimization problems are NP-hard, and thus very challenging to solve. In fact, they cannot be solved efficiently by any known algorithm in a practically short time scale when the size of the problem is moderately large. The main difficulty arises with the number of combinations which increases exponentially with the size of the problem. Searching for every possible combination is extremely computationally expansive and unrealistic. An example of these problems is the travelling salesman problem in which a salesperson has to visit a list of cities exactly once, and returning to the departure city, with the aim of minimizing the total travelled distance or the overall cost of the trip.

For these problems, optimization is used to find the 'best' solution. Optimization plays an important role in solving various engineering problems which are Non-deterministic in nature. The goal of the optimization process is to determine either a maximum or a minimum value of the problem being solved, generally known as the objective function. These problems include, but not limited to, systems design, electricity network operation, electricity generation, wireless communications routing and minimization of energy losses during electricity transmission. Proper measures of efficiency of optimization algorithms require assessment of computational time and convergence rate in addition to the accuracy to determine the minimum or maximum values.

Approximate algorithms such as metaheuristics [2] are actually the best choice to solve many combinatorial optimization problems. They are characterized by their 1 simplicity and flexibility while demonstrating remarkable effectiveness. Metaheuristics are usually simple to implement; however, they are often capable to solve complex problems and can thus be adapted to solve problems with diverse objective function properties, either continuous, discrete or mixed, including many real-world optimization problems, from engineering to artificial intelligence.

Brute force often has high computational complexity and is unsuitable for large datasets. Greedy algorithm makes greedy choices at every step. It makes the choice that is best at that moment. This approach often gives only an average solution and not the optimal solution. The greedy algorithms fail especially when global optimal solution is required. Heuristics are used to solve and quickly find solutions by performing optimized search.

So, the question is how to treat combinatorial problems properly without losing the good performance of these metaheuristics. In this paper, we propose the random-key cuckoo search (RKCS) algorithm using the random-key encoding scheme to represent a position, found by the cuckoo search algorithm, in a combinatorial space.

Cuckoo Search Optimization (CSO) is one such meta-heuristic algorithm which replicates brood parasitism found cuckoos. The eggs of the bird represent the potential solutions. Gradually, the algorithm replaces the poor solutions with new and better solutions. The CSO has several applications in various fields such as neural networks, job scheduling, etc. One such problem which can be solved with Cuckoo Search Optimization is Travelling Salesman Problem

A distributed system is a system whose components are located on different networked computers, which communicate and coordinate their actions by passing messages to one another. Cluster computing is a type of distributed system. A computer cluster is a set of connected computers that work together to perform a single task. They can be tightly coupled or loosely coupled.

## CUCKOO SEARCH ALGORITHM

Some cuckoo species can have the so-called brood parasitism as an aggressive reproduction strategy. This most studied and discussed feature is that cuckoos lay eggs in a previously observed nest of another species to let host birds hatch and brood their young cuckoo chicks. From the evolutionary point of view, cuckoos aim to increase the probability of survival and reduce the probability of abandoning eggs by the host birds. The behavior of cuckoos is mimicked successfully in the cuckoo search algorithm, in combination with L'evy flights to effectively search better and optimal survival strategy. L'evy flights, named by the French mathematician Paul L'evy, represent a model of random walks characterized by their step lengths which obey a power-law distribution. Several scientific studies have shown that the search for preys by hunters follows typically the same characteristics of L'evy flights. This model is commonly presented by small random steps followed occasionally by large jumps.

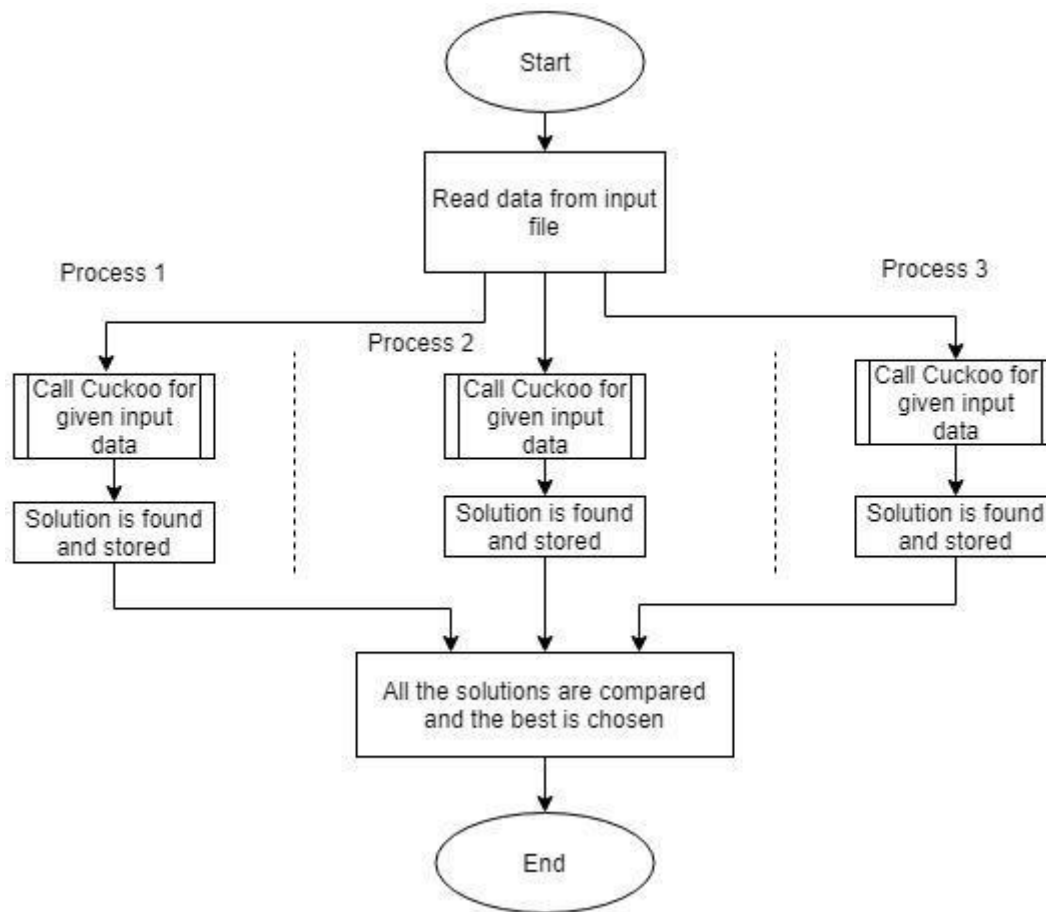
CS is summarized as the following ideal rules:

1. Each cuckoo lay one egg at a time and selects a nest randomly.
2. The best nest with the highest quality of egg can pass onto the new generations.
3. The number of host nests is fixed, and the egg laid by a cuckoo can be discovered by the host bird with a probability  $p_a \in [0, 1]$ .

## PROPOSED MODEL

In this approach we have taken as a basis an improved version of CS. This improvement considers a new category of cuckoos that can engage a kind of surveillance on nests likely to be a host. These cuckoos use mechanisms before and after brooding such as the observation of the host nest to decide if the chosen nest is the best choice or not. So, from the current solution, this portion  $pc$  of cuckoos searches in the same area a new better solution via L'evy flights.

PSO suffers from partial optimization. The algorithm can get trapped in local optimum, which will lead to non-optimal solution. Probability distribution changes by iteration in PSO, hence the algorithm will be not be efficient. PSO also has a weak local search ability. Hence, Cuckoo Search Optimization is chosen to find the optimum solution in this proposed work. Since the Travelling Salesman Problem is a combinatorial optimization problem and Cuckoo Search Optimization is designed to apply for continuous optimization problem, this research proposes Discrete Cuckoo Search Optimization for solving TSP. Cuckoo search is a non-deterministic algorithm. Therefore, getting one solution may not be the best solution. The algorithm needs to be run multiple times.



Cuckoo Search Optimization executed parallelly



## DISCRETE CUCKOO SEARCH OPTIMIZATION (DCSO)

Discrete Cuckoo Search Optimization algorithm uses random walks with Levy flights. Cuckoo Search is an Optimization algorithm developed by Xin-she Yang and Suash Deb in 2009. The algorithm was derived from nature of cuckoo species which lay their eggs in the nests of other birds. If the host bird finds alien eggs, i.e., cuckoo eggs, the host bird will throw these eggs or abandon the nest and create a new nest

In the Cuckoo Search Optimization an egg in a nest represents a solution, and a cuckoo egg represents a new and better solution. The final goal of the algorithm is to throw the bad solutions and replace them with better solutions. Each nest has one egg. Complex cases include multiple eggs in every nest.

Basic rules for the algorithm:

1. Cuckoo can lay only one egg in an iteration, and discards its egg in a random nest.
2. Only the best solutions will carry on to the next generation.
3. Probability of the host bird finding the egg by the cuckoo is higher if the solution is bad. If an egg is detected, it is thrown to a faraway nest.

Each node selects a different starting city and computes a traversal path. Randomizing the cities selected helps in a better optimized solution. Each node where this program is run will result in different cuckoo solutions. Finally, all the solutions across the nodes are compared based on their cost and the best solution is selected and presented as the final solution.

## ALGORITHM

**Algorithm:** Discrete Cuckoo Search Optimization

**Input:** Distance Matrix

**Output:** Optimized path and cost

```
1: numNest = sizeof(InputMatrix)

2: Pa = 0.2 * numNest

3: MaxGen = 50
4: Generate initial population with randomly chosen city and
   store in nests.
5: while (MaxGen)
6: Assign random index i from nests to Cnest

7: Calculate Levy Flight

8: if Levy Flight>2:
   Perform Double Bridge for Cnest with random values
9: else:
   Perform 2-Opt-Move for Cnest with random values

   end if
10: Get random index j from nests

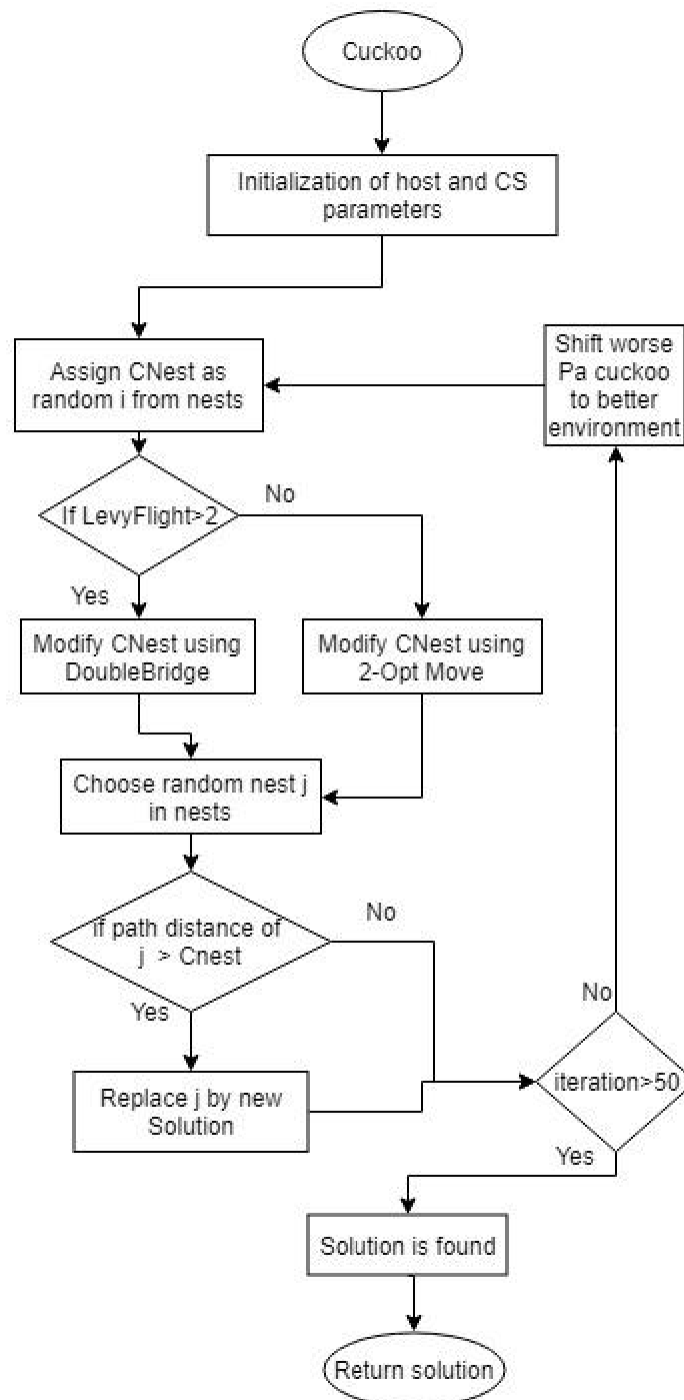
11: if cost(nest_j) > cost(Cnest)

   Replace j by the new solution;
   end if
12: A fraction(Pa) of the worse nests are abandoned.

13: Sort the nests based on cost

   end while
14: nest[0] is the solution
```

## FLOWCHART



Cuckoo function

## IMPLEMENTATION DETAILS AND RESULT ANALYSIS

In this proposed work, a new approach is proposed to solve the TSP optimization using multiple nodes and performing the computation in parallelism. By parallelizing we reduce the computational load on each of the individual machines thus improving performance of the algorithm. We have achieved communication between the nodes using Message Passing Interface (MPI, specifically mpi4py) as it is highly suitable for distributed memory models and supports all high-performance computing platforms. Parallelizing is done by splitting the work among different nodes of the cluster.

Beowulf cluster was established for parallelizing. Beowulf cluster is a multi-computer architecture which uses identical computers connected via a network and libraries, which produces a high-performance parallel cluster thus allowing for faster computation. It supports a Non uniform memory access (NUMA) architecture. The cluster follows a client-server architecture. Beowulf clusters often use cheap commodity grade computers, resulting in an inexpensive supercomputer.

First, a sequential algorithm was implemented in python, and it was then converted into a parallelized using MPI libraries designed for python (mpi4py) and then was executed on the Beowulf cluster, with the server node running the driver program. The experimental results are efficacious and are presented to demonstrate the benefits of parallelizing the discrete cuckoo search algorithm for the optimization of the travelling salesman problem.

## OUTPUT

Accessing the master node: ssh master

```
mpiuser@abhishek-VirtualBox:~$ ssh master
Welcome to Ubuntu 18.04.3 LTS (GNU/Linux 5.0.0-23-generic x86_64)

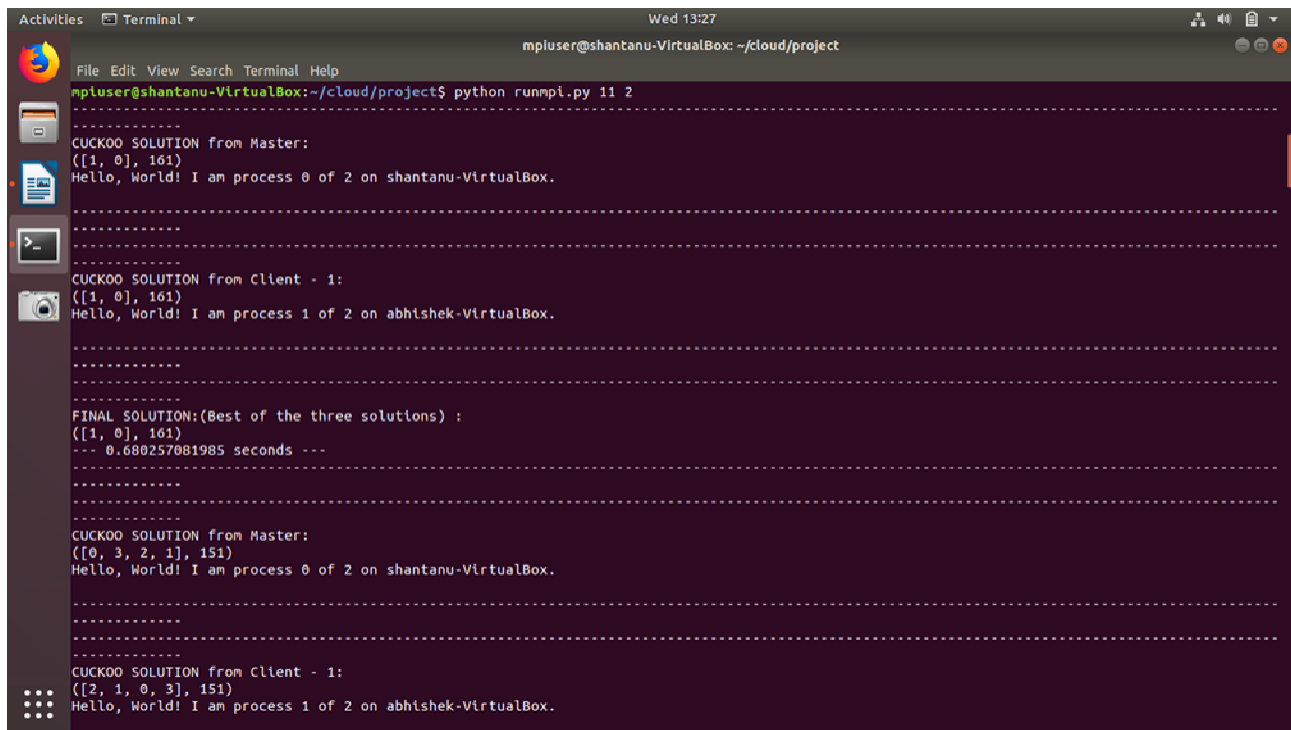
 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

 * Canonical Livepatch is available for installation.
   - Reduce system reboots and improve kernel security. Activate at:
     https://ubuntu.com/livepatch

191 packages can be updated.
109 updates are security updates.

Your Hardware Enablement Stack (HWE) is supported until April 2023.
Last login: Wed Nov  6 12:49:08 2019 from 192.168.43.202
mpiuser@shantanu-VirtualBox:~$
```

Serial Implementation: 10 matrices with size =  $2^n$  where  $n = 1$  to 10



```
Activities Terminal
Wed 13:27
mpiuser@shantanu-VirtualBox: ~/cloud/project

File Edit View Search Terminal Help
mpiuser@shantanu-VirtualBox:~/cloud/project$ python runmpl.py 11 2
-----
CUCKOO SOLUTION from Master:
([1, 0], 161)
Hello, World! I am process 0 of 2 on shantanu-VirtualBox.
-----
CUCKOO SOLUTION from Client - 1:
([1, 0], 161)
Hello, World! I am process 1 of 2 on abhishek-VirtualBox.
-----
FINAL SOLUTION:(Best of the three solutions) :
([1, 0], 161)
--- 0.680257081985 seconds ---
-----
CUCKOO SOLUTION from Master:
([0, 3, 2, 1], 151)
Hello, World! I am process 0 of 2 on shantanu-VirtualBox.
-----
CUCKOO SOLUTION from Client - 1:
([2, 1, 0, 3], 151)
Hello, World! I am process 1 of 2 on abhishek-VirtualBox.
```

```
mpluser@shantanu-VirtualBox: ~/cloud/project
File Edit View Search Terminal Help
CUCKOO SOLUTION from Master:
([3, 2, 0, 7, 1, 6, 14, 10, 8, 5, 15, 4, 12, 9, 13, 11], 491)
Hello, World! I am process 0 of 2 on shantanu-VirtualBox.
-----
-----
CUCKOO SOLUTION from Client - 1:
([4, 7, 8, 15, 2, 6, 0, 14, 13, 9, 10, 3, 12, 1, 5, 11], 472)
Hello, World! I am process 1 of 2 on abhishek-VirtualBox.
-----
-----
FINAL SOLUTION:(Best of the three solutions) :
([4, 7, 8, 15, 2, 6, 0, 14, 13, 9, 10, 3, 12, 1, 5, 11], 472)
--- 0.649039030075 seconds ---
-----
-----
CUCKOO SOLUTION from Master:
([1, 30, 3, 20, 9, 6, 14, 29, 5, 24, 7, 11, 12, 22, 28, 15, 23, 17, 8, 27, 13, 21, 4, 16, 18, 25, 26, 19, 10, 0, 2, 31], 1191)
Hello, World! I am process 0 of 2 on shantanu-VirtualBox.
-----
-----
CUCKOO SOLUTION from Client - 1:
([15, 3, 16, 0, 4, 20, 6, 7, 29, 9, 8, 11, 12, 22, 14, 27, 2, 17, 19, 18, 5, 21, 13, 23, 1, 25, 26, 24, 10, 28, 30, 31], 1291)
Hello, World! I am process 1 of 2 on abhishek-VirtualBox.
-----
-----
```

```
mpluser@shantanu-VirtualBox: ~/cloud/project
File Edit View Search Terminal Help
73, 74, 75, 76, 77, 383, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 110, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 10
6, 107, 108, 109, 93, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134,
135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 16
3, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 932, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191,
192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 2
20, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 684, 248
, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271, 272, 273, 274, 275, 276,
277, 278, 279, 280, 281, 282, 283, 284, 285, 286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298, 299, 300, 301, 302, 303, 304, 30
5, 306, 307, 308, 309, 310, 311, 312, 313, 469, 315, 316, 317, 318, 319, 320, 321, 322, 323, 324, 325, 326, 327, 328, 329, 330, 331, 332, 333,
334, 335, 336, 337, 338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 348, 349, 350, 351, 352, 353, 354, 355, 356, 357, 358, 359, 360, 361, 3
62, 363, 364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377, 378, 379, 380, 381, 382, 895, 384, 385, 386, 387, 388, 389, 390
, 391, 392, 393, 394, 395, 396, 397, 398, 399, 404, 401, 402, 403, 400, 405, 406, 407, 408, 409, 410, 411, 412, 413, 414, 415, 497, 417, 418,
419, 420, 421, 422, 423, 424, 425, 426, 427, 428, 429, 430, 431, 432, 433, 434, 435, 436, 437, 438, 439, 440, 441, 442, 443, 444, 445, 446, 44
7, 448, 449, 450, 451, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461, 462, 463, 464, 465, 466, 467, 468, 314, 470, 471, 472, 473, 474, 475,
476, 477, 478, 479, 480, 481, 482, 483, 484, 485, 486, 487, 799, 489, 490, 491, 492, 493, 494, 495, 496, 416, 498, 499, 500, 501, 502, 503, 5
04, 505, 506, 507, 508, 509, 510, 511, 512, 513, 514, 515, 516, 517, 518, 519, 520, 521, 522, 523, 524, 525, 526, 527, 528, 529, 530, 531, 532
, 533, 534, 535, 536, 537, 538, 539, 540, 541, 542, 543, 544, 545, 546, 547, 548, 549, 550, 551, 552, 553, 554, 555, 556, 557, 558, 559, 560,
561, 562, 563, 564, 565, 566, 567, 568, 569, 570, 571, 572, 573, 574, 575, 576, 577, 578, 579, 580, 581, 582, 583, 584, 585, 586, 587, 588, 58
9, 590, 591, 592, 593, 594, 595, 596, 597, 598, 599, 600, 601, 602, 603, 604, 605, 606, 607, 608, 609, 610, 611, 612, 613, 614, 615, 616, 617,
618, 619, 620, 621, 622, 623, 624, 625, 626, 627, 628, 629, 630, 631, 632, 633, 634, 635, 636, 637, 638, 639, 640, 641, 642, 643, 644, 645, 6
46, 647, 648, 649, 650, 651, 652, 653, 654, 655, 656, 657, 658, 659, 660, 661, 662, 663, 664, 665, 666, 667, 668, 669, 670, 671, 672, 673, 674
, 675, 676, 677, 678, 679, 680, 681, 682, 683, 247, 685, 686, 687, 688, 689, 690, 691, 692, 693, 694, 695, 696, 697, 698, 699, 700, 701, 702,
703, 704, 705, 706, 707, 708, 709, 710, 711, 712, 713, 714, 715, 716, 717, 718, 719, 720, 721, 722, 723, 724, 725, 726, 727, 728, 729, 730, 73
1, 732, 733, 734, 735, 736, 737, 738, 739, 740, 741, 742, 743, 744, 745, 746, 747, 748, 749, 750, 751, 752, 753, 754, 755, 756, 757, 758, 759,
760, 761, 762, 763, 764, 765, 766, 767, 768, 769, 770, 771, 917, 773, 774, 775, 776, 777, 778, 779, 780, 781, 782, 783, 784, 785, 786, 787, 7
88, 789, 790, 791, 792, 793, 794, 795, 796, 797, 798, 488, 800, 801, 802, 803, 804, 805, 806, 807, 808, 809, 810, 811, 812, 813, 814, 815, 816
, 817, 818, 819, 820, 821, 822, 823, 824, 825, 826, 827, 828, 829, 830, 831, 832, 833, 834, 835, 836, 837, 838, 839, 840, 841, 842, 843, 844,
845, 846, 847, 848, 849, 850, 851, 852, 853, 854, 855, 881, 857, 858, 859, 860, 861, 862, 863, 864, 865, 866, 867, 868, 869, 870, 871, 872, 87
3, 874, 875, 876, 877, 878, 879, 880, 856, 882, 883, 884, 885, 886, 887, 888, 889, 890, 891, 892, 893, 894, 78, 896, 897, 898, 899, 900, 901,
902, 903, 904, 905, 906, 907, 908, 909, 910, 911, 912, 913, 914, 915, 916, 772, 918, 919, 920, 921, 922, 923, 924, 925, 926, 927, 928, 929, 93
0, 931, 177, 933, 934, 935, 936, 937, 938, 939, 940, 941, 942, 943, 944, 945, 946, 947, 948, 949, 950, 951, 952, 953, 954, 955, 956, 957, 958,
959, 960, 961, 962, 963, 964, 965, 966, 967, 968, 969, 970, 971, 972, 973, 974, 975, 976, 977, 978, 979, 980, 981, 982, 983, 984, 985, 986, 9
87, 988, 989, 990, 991, 992, 993, 994, 995, 996, 997, 998, 999, 1000, 1001, 1002, 1003, 1004, 1005, 1006, 1007, 1008, 1009, 1010, 1011, 1012,
1013, 1014, 1015, 1016, 1017, 1018, 1019, 1020, 1021, 1022, 1023], 51811)
--- 8.85648179054 seconds ---
-----
-----
mpluser@shantanu-VirtualBox:~/cloud/projects
```



1 2	0.0028781890869
2 4	0.0029230117798
3 8	0.0024788379669
4 16	0.0024328231812
5 32	0.0024781227112
6 64	0.0025179386139
7 128	0.0132279396057
8 256	0.0113651752472
9 512	0.0353300571442
10 1024	0.0672559738159

Parallel Implementation: 10 matrices with size =  $2^n$  where  $n = 1$  to 10

```

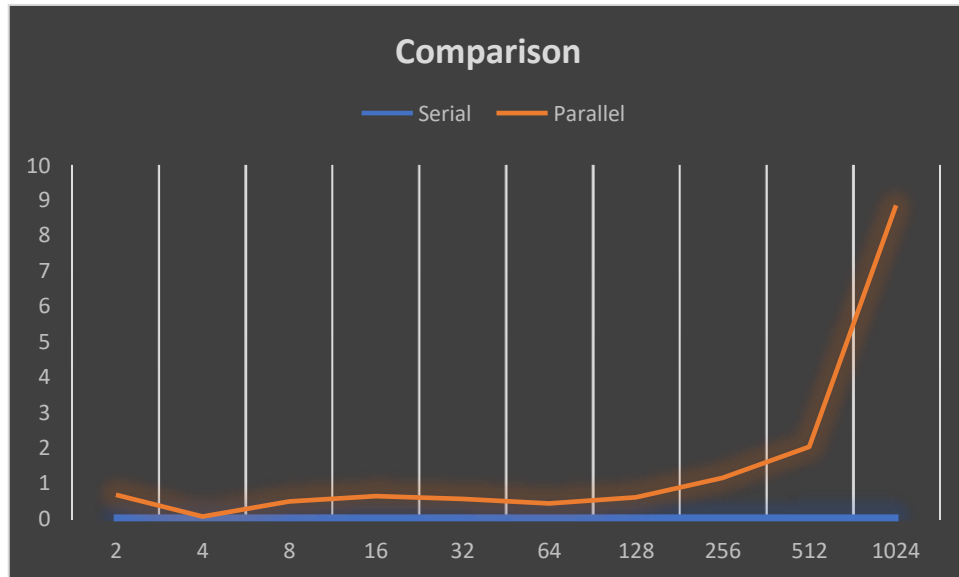
mpluser@shantanu-VirtualBox: ~/cloud/project
File Edit View Search Terminal Help
mpluser@shantanu-VirtualBox:~/cloud/project$ python runserial.py 11 2
CUCKOO's SOLUTION
([1, 0], 161)
--- 0.00287818908691 seconds ---
CUCKOO's SOLUTION
([1, 0], 161)
--- 0.00292301177979 seconds ---
CUCKOO's SOLUTION
([1, 0], 161)
--- 0.00247883796692 seconds ---
CUCKOO's SOLUTION
([1, 0], 161)
--- 0.00243282318115 seconds ---
CUCKOO's SOLUTION
([1, 0], 161)
--- 0.00247812271118 seconds ---
CUCKOO's SOLUTION
([1, 0], 161)
--- 0.00251793861389 seconds ---
CUCKOO's SOLUTION
([19, 2, 3, 4, 5, 6, 0, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 1, 20, 21, 22, 23, 24, 25, 26, 27, 96, 29, 30, 31, 32, 33, 88, 35, 36, 37, 38, 90, 40, 41, 100, 43, 44, 45, 46, 47, 48, 49, 50, 54, 52, 53, 51, 55, 56, 57, 58, 59, 60, 78, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 61, 79, 80, 81, 82, 83, 84, 117, 86, 87, 34, 89, 39, 91, 92, 93, 94, 95, 28, 97, 98, 99, 42, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 85, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127], 6252)
--- 0.0132279396057 seconds ---
CUCKOO's SOLUTION
([1, 2, 3, 93, 118, 6, 58, 8, 0, 9, 10, 124, 12, 13, 14, 82, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 40, 35, 3, 6, 37, 38, 39, 34, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 111, 56, 57, 7, 11, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 15, 83, 84, 85, 86, 87, 88, 89, 90, 114, 92, 4, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 55, 112, 113, 91, 115, 116, 117, 5, 119, 120, 121, 122, 123, 59, 125, 126, 127], 5794)
--- 0.0113651752472 seconds ---
CUCKOO's SOLUTION
([1, 0, 2, 221, 4, 5, 6, 7, 8, 9, 10, 319, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 439, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 472, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 448, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 211, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 476, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 224, 158, 159, 160, 161, 16, 2, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190,

```

```
File Edit View Search Terminal Help
mpiuser@shantanu-VirtualBox: ~/cloud/project
([1, 2, 3, 4, 5, 6, 0, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37,
, 38, 39, 40, 41, 42, 43, 44, 957, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72,
, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106
, 661, 108, 109, 110, 111, 112, 113, 242, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134,
135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 16
3, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191,
, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 823, 218, 219, 2
20, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 581, 243, 244, 245, 246, 247, 248
, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271, 272, 273, 274, 275, 276,
277, 278, 279, 280, 281, 282, 283, 284, 285, 286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298, 299, 300, 301, 302, 303, 304, 30
5, 306, 307, 308, 309, 310, 311, 312, 313, 314, 315, 316, 317, 318, 319, 320, 321, 322, 323, 324, 325, 326, 327, 328, 329, 330, 331, 332, 333,
, 334, 335, 336, 337, 338, 339, 340, 341, 342, 343, 344, 345, 493, 347, 348, 349, 350, 351, 352, 353, 354, 355, 356, 357, 358, 359, 360, 361, 3
62, 363, 364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377, 378, 379, 380, 381, 382, 383, 384, 385, 386, 387, 388, 389, 390
, 391, 392, 393, 394, 395, 396, 397, 398, 399, 400, 401, 402, 403, 404, 405, 406, 407, 408, 409, 410, 411, 412, 413, 414, 415, 416, 417, 418,
419, 677, 421, 422, 423, 424, 425, 426, 427, 428, 429, 430, 431, 432, 433, 434, 435, 436, 437, 438, 439, 440, 441, 442, 443, 444, 445, 446, 44
7, 448, 449, 450, 451, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461, 462, 463, 464, 465, 466, 467, 468, 469, 470, 471, 472, 473, 474, 475,
476, 477, 478, 479, 480, 481, 482, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492, 578, 494, 495, 496, 497, 498, 499, 500, 501, 502, 503, 5
04, 505, 506, 507, 508, 509, 510, 511, 512, 513, 514, 515, 516, 517, 518, 519, 520, 521, 522, 523, 524, 525, 526, 527, 528, 529, 530, 531, 532
, 533, 534, 535, 536, 537, 538, 539, 540, 541, 542, 543, 544, 545, 546, 547, 548, 549, 550, 551, 552, 553, 554, 555, 556, 557, 558, 559, 560,
561, 562, 563, 564, 565, 566, 567, 568, 569, 570, 571, 572, 573, 574, 575, 576, 577, 346, 579, 580, 114, 582, 583, 584, 585, 586, 587, 588, 58
9, 590, 591, 592, 593, 594, 595, 596, 597, 598, 599, 600, 601, 602, 603, 604, 605, 606, 607, 608, 609, 610, 611, 612, 613, 614, 615, 616, 617,
, 618, 619, 913, 621, 622, 623, 624, 625, 626, 627, 628, 629, 630, 631, 632, 633, 634, 635, 636, 637, 638, 639, 640, 641, 642, 643, 644, 645, 6
46, 647, 648, 649, 650, 651, 652, 653, 654, 655, 656, 657, 658, 659, 660, 107, 662, 663, 664, 665, 666, 667, 668, 669, 670, 671, 672, 673, 674
, 675, 676, 420, 678, 679, 680, 681, 682, 683, 684, 685, 686, 687, 688, 689, 690, 691, 692, 693, 694, 695, 696, 697, 698, 699, 700, 701, 702,
703, 704, 705, 706, 707, 708, 709, 710, 711, 712, 713, 714, 715, 716, 717, 718, 719, 720, 721, 722, 723, 724, 725, 726, 727, 728, 729, 730, 73
1, 732, 733, 734, 735, 736, 737, 738, 739, 740, 741, 742, 743, 744, 745, 746, 747, 748, 749, 750, 751, 752, 753, 754, 755, 756, 757, 758, 759,
760, 761, 762, 763, 764, 765, 766, 767, 768, 769, 770, 771, 772, 773, 774, 775, 776, 777, 778, 779, 780, 781, 782, 783, 784, 785, 786, 787, 7
88, 789, 790, 791, 792, 793, 794, 795, 796, 797, 798, 799, 800, 801, 802, 803, 804, 805, 806, 807, 808, 809, 810, 811, 812, 813, 814, 815, 816
, 817, 818, 819, 820, 821, 822, 217, 824, 825, 826, 827, 828, 829, 830, 831, 832, 833, 834, 835, 836, 837, 838, 839, 840, 841, 842, 843, 844,
845, 846, 847, 848, 849, 850, 851, 852, 853, 854, 855, 856, 857, 858, 859, 860, 861, 862, 863, 864, 865, 866, 867, 868, 869, 870, 871, 872, 87
3, 874, 875, 876, 877, 878, 879, 880, 881, 882, 883, 884, 885, 886, 887, 888, 889, 890, 891, 892, 893, 894, 895, 896, 897, 898, 899, 900, 901,
902, 903, 904, 905, 906, 907, 908, 909, 910, 911, 912, 620, 914, 915, 916, 917, 918, 919, 920, 921, 922, 923, 924, 925, 926, 927, 928, 929, 9
30, 931, 932, 933, 934, 935, 936, 937, 938, 939, 940, 941, 942, 943, 944, 945, 946, 947, 948, 949, 962, 951, 952, 953, 954, 955, 956, 45, 958,
959, 960, 961, 950, 963, 964, 965, 966, 967, 968, 969, 970, 971, 972, 973, 974, 975, 976, 977, 978, 979, 980, 981, 982, 983, 984, 985, 986, 9
87, 988, 989, 990, 991, 992, 993, 994, 995, 996, 997, 998, 999, 1000, 1001, 1002, 1003, 1004, 1005, 1006, 1007, 1008, 1009, 1010, 1011, 1012,
1013, 1014, 1015, 1016, 1017, 1018, 1019, 1020, 1021, 1022, 1023], 51757)
--- 0.0672559738159 seconds ---
mpiuser@shantanu-VirtualBox: ~/cloud/projects
```

1 2	0.6802570819855
2 4	0.0684709548950
3 8	0.4936518669128
4 16	0.6490390300751
5 32	0.5639431476593
6 64	0.4355430603027
7 128	0.6131639480591
8 256	1.1653029918671
9 512	2.0407080650330
10 1024	8.8564817905426





X-axis: matrix size Y-axis: seconds

## CONCLUSION

The graph plotted for the data obtained is abrupt and far from normal as well as misleading. However, if the matrix size increases beyond a certain point, the serial line will definitely quickly exponentiate, even faster than the parallel line. The above difference between the lines, which is huge, is due to the fact that only two nodes with dual-core processors each were used and the communication overhead between them is too high. If even powerful machines were to be used, we would be able to get outputs for the larger matrix inputs.

Thus, we have successfully implemented the research paper chosen even if we got deviated and illusioned results.