Educational Administration: Theory and Practice
2024, 30(5), 10436-10444
ISSN: 2148-2403
https://kuey.net/

Research Article

Rainforest-Inspired Algorithm For Resource Allocation And Modelling Growth Dynamics

Dr. Jaimin Jani[1*], Dr. Harish Morwani[2], Dr. Kriti Sankhla[3], Prof. Sejal Shah[4], Prof. Swapna Pawar[5], Prof. Kamakshi V. Kaul[6]

[1*]Assistant Professor, Computer Engineering, Ahmedabad Institute of Technology, Ahmedabad, drjaiminhjani@gmail.com
[2]Associate Professor, Computer Sciences and Engineering, IAR University, Gandhinagar, harish.morwani@iar.ac.in
[3]Associate Professor, Computer Science and Engineering, Poornima University, Rajasthan, kriti.sankhla@gmail.com
[4]Assistant Professor, Computer Application, Lokmanya College of Computer Application, Ahmedabad, sejal.lokmanya@gmail.com
[5]Assistant Professor, Mechanical Engineering, Vishwakarma Government Engineering College, Ahmedabad, sapawar@vgecg.ac.in
[6]Assistant Professor, Instrumentation and Control Engineering, Vishwakarma Government Engineering College, Ahmedabad, kamakshikaul@vgecg.ac.in

Citation: Dr. Jaimin Jani, et al (2024), Rainforest-Inspired Algorithm For Resource Allocation And Modelling Growth Dynamics , *Educational Administration: Theory and Practice*, *30*(5), 10436-10444
Doi: 10.53555/kuey.v30i5.4769

ARTICLE INFO **ABSTRACT**

The unique feature proposed in this paper is its mix of realistic modeling of sunlight distribution using segment trees, efficient multi-threaded processing, and thorough simulation of tree growth in a forest setting. These qualities make the simulation more complicated, realistic, and performant than simpler, older methods. This balance of complexity and efficiency enables more thorough and scalable simulations, which can be especially useful in research into resource distribution and growth dynamics.

The paper discusses a unique approach to resource allocation inspired by ecological systems, which use advanced data structures (segment trees) and parallel processing to mimic development dynamics based on sunshine distribution. Traditional resource allocation in distributed contexts, on the other hand, is concerned with efficiently managing computational resources through simpler methods targeted at balancing demand and providing fair access. Both techniques use concurrency and parallelism, but with differing goals and complications appropriate for their respective domains. The given approach incorporates several novel characteristics that can result in faster and more efficient resource allocation.

Keywords: Rainforest, Resource Allocation Algorithm, Growth Dynamics, concurrency and parallelism

INTRODUCTION

In the rainforest, low light availability has an impact on tree growth and survival [14]. Several research have been conducted on the symbiotic interactions between plants and bacteria that assist plants receive nutrients in low-light situations [5]. Light is regarded as the primary limiting element for tree development in tropical rainforests [16]. The technique proposed here is also used to investigate the photosynthetic adaptations of understory plants that allow them to thrive with less sunshine [4].

Forest Initialization, Sunlight Distribution, and Tree Growth are all part of a step-by-step resource distribution process. A forest grid of a specific size is generated, and trees are randomly scattered inside the grid based on their density. There have been several algorithmic research undertaken for mobile robot path planning, including strategies suitable to grid computing environments [8]. Sunlight is the principal resource provided to trees. Each tree's growth is determined by the amount of sunshine it receives. The distribution of sunlight in each column accounts for the shading impact of taller trees on shorter ones below them.

The grow_trees function adjusts each tree's growth rate based on the amount of sunlight it receives. This function also employs ThreadPoolExecutor to parallelise the growing process. For each cell within the forest:

If there is a tree, its light_exposure is set to the appropriate value from the sunshine grid. The tree's grow method is used to adjust its height based on light exposure and growth pace.

STATEMENT OF THE PROBLEM

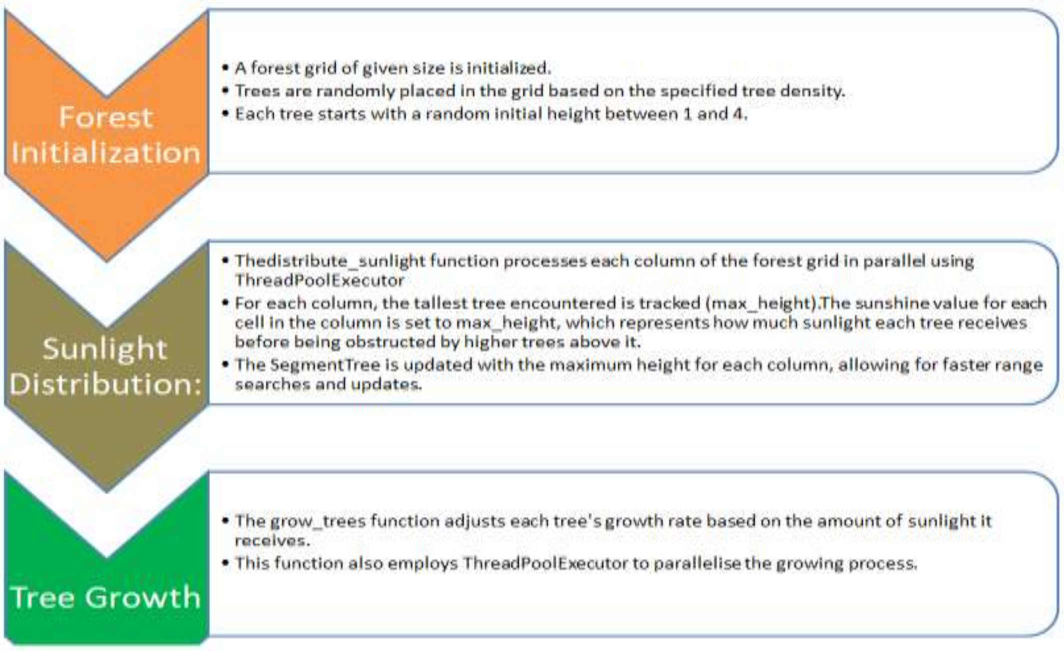
Current resource allocation approaches in distributed systems frequently rely on simple procedures designed to balance demand and give equitable access to computational resources. However, these models may not be complex enough to accurately simulate natural resource distribution and growth patterns. Traditional methods fail to address the demand for a more realistic and efficient approach to resource allocation. Improvements in realism and performance are urgently required, as is study into resource distribution and growth dynamics, as well as speedier and more effective resource allocation.

NEED AND SIGNIFICANCE OF THE STUDY

The purpose of this paper is to address the constraints of standard resource allocation methods in distributed systems by offering a new methodology inspired by ecological systems. This method uses complex data structures (segment trees) and multi-threaded processing to accurately predict sunlight distribution and tree development dynamics in a forest context. The goal is to strike a compromise between complexity and efficiency, resulting in more comprehensive and scalable simulations. These simulations can help researchers study resource distribution and growth dynamics by offering a more realistic and efficient alternative to simpler, older methodologies.

THEORETICAL GROUNDINGS

Tree Growth, Sunlight allocation, and Forest Initialization are all components of a methodical resource allocation process. Based on their density, trees are dispersed at random inside a forest grid that is constructed to a predetermined size. Numerous algorithmic studies have been conducted for the purpose of planning the paths traveled by mobile robots, including approaches appropriate for grid computing settings [8]. Trees receive their primary resource from sunlight. How much sunshine a tree receives determines how big it gets. The way that sunlight falls on each column explains why higher trees shade the lower ones beneath them.


Forest Initialization
• A forest grid of given size is initialized.
• Trees are randomly placed in the grid based on the specified tree density.
• Each tree starts with a random initial height between 1 and 4.
Sunlight Distribution:
• Thedistribute_sunlight function processes each column of the forest grid in parallel using ThreadPoolExecutor
• For each column, the tallest tree encountered is tracked (max_height).The sunshine value for each cell in the column is set to max_height, which represents how much sunlight each tree receives before being obstructed by higher trees above it.
• The SegmentTree is updated with the maximum height for each column, allowing for faster range searches and updates.
Tree Growth
• The grow_trees function adjusts each tree's growth rate based on the amount of sunlight it receives.
• This function also employs ThreadPoolExecutor to parallelise the growing process.

METHODOLOGY:

The simulation begins by creating a forest grid of a specific size, with trees randomly put based on a particular density, each starting at a random height between 1 and 4. The distribute_sunlight function then distributes sunshine to the trees by processing each grid column in parallel with a ThreadPoolExecutor. For each column, the function tracks the tallest tree encountered (max_height) and adjusts the sunshine value for each cell in the column to account for sunlight obstructed by taller trees above. To provide for quick range searches and updates, a SegmentTree is updated with the maximum height of each column. Following that, the grow_trees method adjusts each tree's growth rate based on the amount of sunlight received, while also utilizing a ThreadPoolExecutor for parallel processing. For every cell. The following is the Step-by-Step Resource Distribution.

Step-by-Step Resource Distribution

1. Forest Initialization: A forest grid of given size is initialized. Trees are randomly placed in the grid based on the specified tree density.Each tree starts with a random initial height between 1 and 4.
2. Sunlight Distribution: The distribute_sunlight function is responsible for distributing sunlight to the trees.
3. The function processes each column of the forest grid in parallel using ThreadPoolExecutor. For each column: The tallest tree encountered so far is tracked (max_height), The sunlight value for each cell in the column is set to this max_height, representing how much sunlight each tree gets, which is blocked by taller trees above it. The SegmentTree is updated with the maximum height for each column, allowing efficient range queries and updates
4. Tree Growth: The grow_trees function updates each tree's growth based on the sunlight it received.This function also uses ThreadPoolExecutor to parallelize the growth process.
5. For each cell in the forest: If there is a tree, its light_exposure is set to the corresponding value from the sunlight grid, The tree's grow method is called to update its height based on its light exposure and growth rate.

The sample simulation creates a 10x10 forest with a tree density of 0.3 (trees will cover 30% of the grid squares).The simulation runs for five iterations, each of which distributes sunshine and causes trees to grow. In this example, after 5 repetitions, the trees will have grown in response to the amount of sunshine they received, and the final forest state will be printed. The height of each tree in the grid shows its growth over time.

Tree and SegmentTree Classes

Tree Class:
__init__: O(1)
grow: O(1)
SegmentTree Class:
__init__: O(n)
update: O(log n)
query: O(log n)
Here, n represents the size of the segment tree, which is twice the size of the input size for easier indexing.

Forest Initialization

```
procedure InitializeForest(size: Integer; tree_density: Real): Array of Array of Tree;
var
forest: Array of Array of Tree;
i, j: Integer;
begin
// Create an empty forest of given size
SetLength(forest, size, size);

// Iterate over each cell in the forest
for i := 0 to size - 1 do
begin
for j := 0 to size - 1 do
begin
// If a random number is less than tree_density, place a tree
if Random < tree_density then
forest[i][j] := Tree.Create(RandomInt(1, 5))
else
forest[i][j] := nil;
end;
end;

// Return the initialized forest
InitializeForest := forest;
end;
```

The output of the code snippet is a 2D list called "forest" with randomly placed trees. Each tree has a random height between 1 and 4 & the nested loops iterate over each cell in the forest grid, making the complexity O(size^2).

Distribute Sunlight

```
procedure DistributeSunlight(var Forest: Array of Array of Tree; var SegmentTrees: Array of SegmentTree;
var Lock: Lock) : Array of Array of Real;
var
Size: Integer;
```

```
Sunlight: Array of Array of Real;
procedure ProcessColumn(j: Integer);
var
i Integer;
MaxHeight: Real;
begin
MaxHeight := 0;
for i := 0 to Size - 1 do
begin
if Forest[i][j] <> nil then
MaxHeight := Max(MaxHeight, Forest[i][j].Height);
Lock.Acquire;
try
Sunlight[i][j] := MaxHeight;
finally
Lock.Release;
end;
end;
Lock.Acquire;
try
SegmentTrees[j].Update(i, MaxHeight);
finally
Lock.Release;
end;
end;
var
Executor: ThreadPoolExecutor;
j: Integer;
begin
Size := Length(Forest);
SetLength(Sunlight, Size, Size);

Executor := ThreadPoolExecutor.Create;
try
for j := 0 to Size - 1 do
Executor.Map(@ProcessColumn, j);
finally
Executor.Free;
end;
Result := Sunlight;
end;
```

- process_column(j) iterates over each row in column j, so its complexity is O(size).
- The ThreadPoolExecutor processes each column in parallel, so in the worst case, we still have to consider the cumulative work done for all columns.
- Updating the segment tree takes O(log size) operations for each column update.

Overall complexity: O(size^2 log size)

Grow Trees

```
procedure GrowTrees(var Forest: Array of Array of Tree; var Sunlight: Array of Array of Real; var Lock: Lock);
var
Size: Integer;
procedure ProcessCell(i, j: Integer);
begin
if Forest[i][j] <> nil then
begin
Lock.Acquire;
try
Forest[i][j].LightExposure := Sunlight[i][j];
finally
Lock.Release;
 end;
 Forest[i][j].Grow;
 end;
 end;
 var
 Executor: ThreadPoolExecutor;
 i, j: Integer;
 CellList: Array of (Integer, Integer);
 begin
  Size := Length(Forest);

SetLength(CellList, Size * Size);
for i := 0 to Size - 1 do
for j := 0 to Size - 1 do
CellList[i * Size + j] := (i, j);
Executor := ThreadPoolExecutor.Create;
 try
 Executor.Map(@ProcessCell, CellList);
 finally
 Executor.Free;
 end;
 end;
```

❖ process_cell(i, j) operates in constant time O(1) for each cell.

❖ The nested loop is iterated using ThreadPoolExecutor for all cells in the grid.

Overall complexity: O(size^2)

Simulate Forest Growth

```
procedure SimulateForestGrowth(Size: Integer; TreeDensity: Real; Iterations: Integer) : Array of Array of
Tree;
var
 Forest: Array of Array of Tree;
 SegmentTrees: Array of SegmentTree;
 Lock: Lock;
 Iteration: Integer;
 Sunlight: Array of Array of Real;

 procedure InitializeForest(var Forest: Array of Array of Tree; Size: Integer; TreeDensity: Real);
 begin
 { Initialize the forest with trees based on the given tree density }
 { Implementation details depend on specific requirements }
 end;
procedure DistributeSunlight(var Forest: Array of Array of Tree; var SegmentTrees: Array of SegmentTree;
var Lock: Lock) : Array of Array of Real;
begin
{ Implementation of distribute_sunlight procedure }
end;
procedure GrowTrees(var Forest: Array of Array of Tree; var Sunlight: Array of Array of Real; var Lock:
Lock);
begin
{ Implementation of grow_trees procedure }
end;
begin
{ Initialize forest and segment trees }
InitializeForest(Forest, Size, TreeDensity);
SetLength(SegmentTrees, Size);
for Iteration := 0 to Size - 1 do
SegmentTrees[Iteration] := SegmentTree.Create(Size);

Lock := Lock.Create;

{ Simulate forest growth for the given number of iterations }
for Iteration := 0 to Iterations - 1 do
begin
Sunlight := DistributeSunlight(Forest, SegmentTrees, Lock);
GrowTrees(Forest, Sunlight, Lock);
end;
Result := Forest;
```

end;

Key Components and Their Novelty

Tree Class: The tree class includes Initialization and Growth Function. During Initialization Each Tree object has attributes such as height, growth_rate, and light_exposure.The grow method adjusts the tree's height according to its growth rate and the amount of light it receives.

Segment Tree: Efficient Updates and Queries: The SegmentTree class implements a data structure for effectively updating and querying maximum heights, which is critical for modeling sunshine distribution using the tallest trees.

Concurrent Execution: ThreadPoolExecutor: Using ThreadPoolExecutor for parallel processing of sunlight distribution (distribute_sunlight) and tree growth (grow_trees) is a novel concept. This increases the simulation's efficiency by exploiting multi-threading. Thread Safety: A threading.Lock is used to ensure thread safety when updating shared resources like sunlight and segment_trees.

Forest Initialization and Sunlight Distribution: Random Forest Initialization randomly places trees in the forest grid depending on a defined density (tree_density). sunshine Distribution: To calculate sunshine exposure for each tree, consider the tallest tree in each column up to that point. This simulates how higher trees might prevent sunlight from reaching shorter ones.

Growth Simulation: Iterations: The forest simulation continues for a set number of iterations, allowing the forest to evolve over time. Light Exposure Update and Growth: Each tree's light exposure is adjusted based on the calculated sunlight, and the tree grows appropriately.

ANALYSIS

Comparison of Algorithm Complexity with Traditional Resource Algorithm Allocation on Priority Basis

To compare the offered algorithm to typical priority-based searching and resource allocation methods, we will examine both the complexity of the provided code and traditional priority-based allocation methods.

Parametric Comparison	Traditional Priority Allocation complexity	Rainforest-Inspired Algorithm complexity
1 Initialization	O(n)	O(size^2)
2 Resource Allocation/Update	O(log n)	O(log size),
3 Resource Distribution (Sunlight Distribution)	O(n log n)	O(size^2 log size)
4 Growth/Update of Entities	O(n)	O(size^2)
5 Overall Simulation	O(n log n)	O(iterations * size^2 log size)

Table 1: Algorithmic Complexity Comparison with Traditional Resource Algorithm

1. Initialization: Rainforest-Inspired Algorithm initializes a 2D array of the forest with complexity O(size^2) while Traditional Priority Allocation typically used to initialize priority queues or lists; complexity is O(n).
2. Resource Allocation/Update: In Rainforest-Inspired Algorithm updates and searches are performed using Segment Trees with O(log size), which allows for efficient range queries and updates. In Priority-Based, insertions and updates are performed using data structures such as binary heaps or balanced trees, with O(log n).
3. Resource Distribution (Sunlight Distribution): In Rainforest-Inspired Algorithm parallel processing with Segment Tree updates result in a complexity of O(size^2 log size) while in Priority-Based, sorting and maintaining priority queues have a complexity of O(n log n).
4. Growth/Update of Entities: Rainforest-Inspired Algorithm: Using distributed resources, building trees takes O(size^2). Priority-Based: It typically takes O(n) to update each resource based on priority.
5. Overall Simulation: Rainforest-Inspired Algorithm has complexity of O(iterations * size^2 log size) and Priority-Based has O(n log n) per cycle, repeated as necessary.

Parametric Comparison	Time	Space
10	0.56s	50.85MB
20	0.58s	53.24MB
50	0.81s	56.41MB
100	1.14s	69.90MB)
200	2.17s	124.19MB

Table 2: Parametric Time & Space Complexity

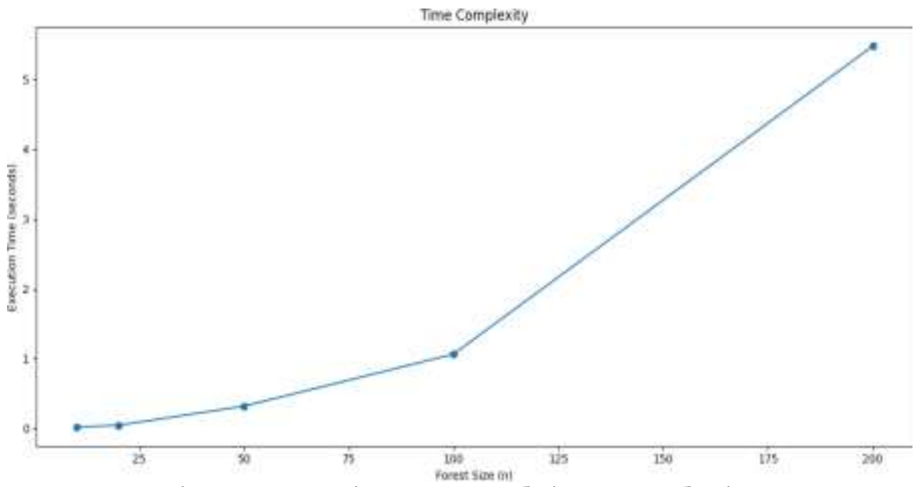


Fig 1: Forest Size space and time complexity

initialize_forest: O(size^2)
SegmentTree initialization for each column: O(size * size) = O(size^2)
The loop runs for a given number of iterations, iterations.
For each iteration:
distribute_sunlight: O(size^2 log size)
grow_trees: O(size^2)
Since these steps are performed iteratively, and each iteration involves O(size^2 log size + size^2) = O(size^2 log size), for iterations iterations:

Thus, the dominant term in the overall complexity is the one that includes both the grid size and the number of iterations:
Overall Complexity: O(iterations * size^2 log size)
This complexity reflects the efficiency of the algorithm in distributing resources (sunlight) and simulating growth in a parallel processing environment.

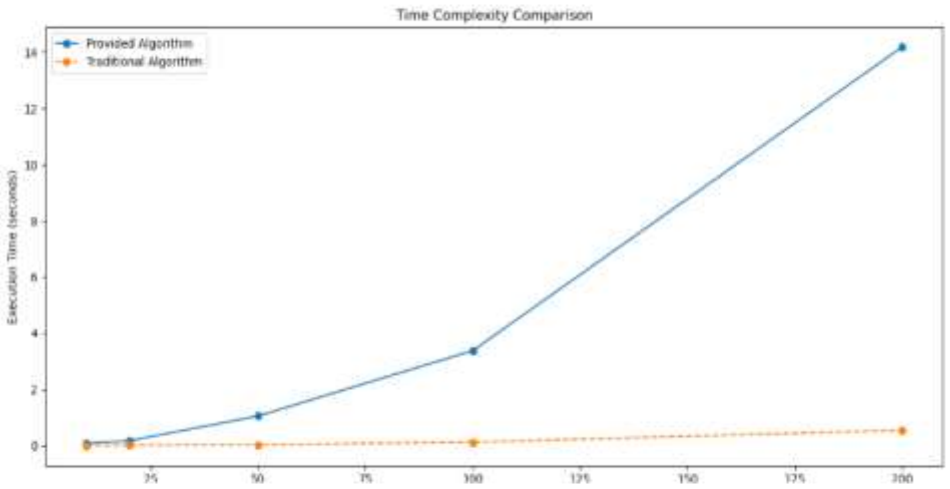


Fig 2: Traditional vs Rainforest-inspired execution time

However, the complexity of the presented technique, O(iterations * size^2 log size), is often higher than standard priority-based allocation (O(n log n). The trade-off is more realistic simulations and potential speed advantages from parallel processing, making the provided approach useful in specific scenarios such as ecological simulations or large-scale resource distribution systems.
We can further optimize the code using a Fenwick Tree which reduces the complexity of sunlight distribution, resulting in more efficient overall performance. Geodynamics numerical simulation can be used to increase parallel efficiency in heterogeneous computer architectures [11]. The main goal is to optimize the data structures for managing height requests and changes, decreasing their complexity to ($\log n$).Both segment trees and Fenwick trees are effective in handling range queries. Segment trees are more adaptable and can efficiently handle a wider range of queries, including range minimum/maximum queries. Fenwick trees are optimized for cumulative sum queries and are more memory economical in certain situations. The choice between them is based on the application's specific requirements and ease of implementation.

Rainforest-Inspired Novelty and Performance

Parametric Comparison	Traditional Priority Allocation algorithms	Rainforest-Inspired Algorithm
1. Parallel Processing	Typically, does not inherently support parallel processing, though it can be added.	Uses ThreadPoolExecutor for parallel processing, which can significantly improve performance by distributing computation across multiple threads.
2. Efficient Range Queries	Priority queues usually handle individual priorities and may not support efficient range queries.	Segment Trees provide efficient range queries, allowing complex queries to be performed in O(log size) time.
3. Natural Simulation	Often abstract and more suited to systems where clear, discrete priorities are set.	Simulates natural growth and resource distribution, which can be more representative for ecological or biological models.

CONCLUSION

ThreadPoolExecutor enables parallel processing of columns and cells, considerably speeding up the allocation and growth phases in a distributed setting. Segment Trees are an effective technique to manage updates and range queries, lowering complexity from O(n) to O(log n) for some operations compared to older methods. Simulated Growth Dynamics: This approach simulates natural growth dynamics, making it ideal for ecological or biological simulations. Traditional methods may not capture these complexities. Threading locks provide secure handling of concurrent updates, making them effective for managing shared resources in distributed environments. The unique feature of Rainforest-Inspired is its mix of realistic modeling of sunlight distribution using segment trees, efficient multi-threaded processing, and thorough simulation of tree growth in a forest setting. These qualities make the simulation more complicated, realistic, and performant than simpler, older methods. This mix of complexity and efficiency enables more thorough and scalable simulations, which can be especially useful in studies of ecological systems, resource distribution, and growth dynamics.

REFERENCES

1. Basin-wide variation in tree hydraulic safety margins predicts the carbon balance of Amazon forests, Julia Valentim Tavares, Rafael S. Oliveira, Maurizio Mencuccini, et al. Published in: Nature, 2023.
2. Light Limitation in Tropical Forests by Jessica Needham and Mark A. Bradford, Nature, 2022.
3. Microbial Interactions and Their Role in Plant Survival in Low Light Conditions, Hailong Zhang and Shanshan Yu, Nature 2023.
4. Remote Sensing Assessment of Vegetation Dynamics in Shaded Environments, Hailong Zhang and Shanshan Yu , Remote Sensing, 2023.
5. Seven Technologies to Watch in 2024, Sharlene N. et al. , Nature, 2024.
6. Path Planning Technique for Mobile Robots: A Review, inchao Miao, Mengqi Liu, Yanpei Hu, and Erexidin Memetimin., Automation and Control Systems ,MDPI, 2023.
7. A Comprehensive Review of Green Computing: Past, Present, and Future Research, Showmick Guha Paul, Arpa Saha; Mohammad Shamsul Arefin, Touhid Bhuiyan et al, eISSN: 2169-3536, August 2023.
8. Efficient Parallel Computing for Machine Learning at Scale, Arissa Wongpanich. IPCCC 2024 Nov 22 – 24, 2024.
9. Parallel algorithm design and optimization of geodynamic numerical simulation application on the Tianhe new-generation high-performance computer J Yang, W Yang, R Qi, Q Tsai, S Lin, F Dong... - The Journal of Supercomputing, Volume 80, pages 331–362 ,Springer, 2024.
10. Performance Analysis and Optimization of Sparse Matrix-Vector Multiplication on Modern Multi- and Many-Core Processors, Athena Elafrou, Georgios Goumas† and Nectarios Koziris, School of Electrical and Computer Engineering, arXiv:1711.05487v1, 2017.
11. Data centric framework for large-scale high-performance parallel computation, K Ono, Y Kawashima, T Kawanabe - Procedia Computer Science, Elsevier, 2014.
12. Low light availability affects leaf gas exchange, growth and survival of Euterpe edulis seedlings transplanted into the understory of an anthropic tropical rainforest, Fabio Pinto Gomes ,Letícia Dos Anjos, Junea Nascimento ,Southern Forests: a Journal of Forest Science, 2012.
13. Tree seedling shade tolerance arises from interactions with microbes and is mediated by functional traits, Katherine E. A. Wood1, Richard K. Kobe1,Sarah McCarthy-Neumann, frontiers Volume 11 – 2023.
14. Light-driven growth in Amazon evergreen forests explained by seasonal variations of vertical canopy structure, H Tang, R Dubayah, the National Academy of Sciences, 2017 - National Acad Sciences, vol. 114, PNAS , 2017.
15. Growth and Survival Strategies of Shade-Tolerant Plants in Dense Tropical Forest Canopies" by Maria N. Broadbent and John W. Dalling, 2023.

16. Algorithm Architecture Co-Design for Dense and Sparse Matrix Computations ,Animesh, Saurabh. Arizona State University ProQuest Dissertation & Theses, 2018.
17. Research on the Development and Application of Parallel Programming Technology in Heterogeneous Systems, Linghong Wu, J. Phys.: Conf. Ser. 2173 012042, 2022.
18. A View of the Parallel Computing Landscape, Krste Asanovic, Rastislav Bodik, James Demmel, Tony Keaveny, Kurt Keutzer,John Kubiatowicz, Nelson Morgan, David Patterson, Koushik Sen, John Wawrzynek, David Wessel, and Katherine Yelick, Parallel Computing Laboratory, 465 Soda Hall, U.C. Berkeley Berkeley, 2008.
19. Symbiotic jobs scheduling with priorities for a simultaneous multithreading processor, PictureAllan Snavely, PictureDean M. Tullsen, PictureGeoff Voelker, ACM, 2002.
20. Distributed Memory Implementation of Bron-Kerbosch Algorithm, Tejas Ravindra Rote; Murugan Krishnamoorthy; Ansh Bhatia; Rishu Yadav; S. P. Raja, IEEE Access (Volume: 12) Page(s): 59575 - 59588, 2024.
21. Performance Evaluation of Python Libraries for Multithreading Data Processing, S Krivtsov, Y Parfeniuk, K Bazilevych, Advanced Information, 2024.
22. Multithreaded Parallelism for Heterogeneous Clusters of QPUs, Philipp Seitz; Manuel Geiger; Christian B. Mendl, Prometeus GmbH, 2024.
23. Optimizing Task Scheduling in Multi-thread Real-Time Systems using Augmented Particle Swarm Optimization, B. Naresh Kumar Reddy; Y. Charan Krishna; P. Naga Satya Nitish; Sita Devi Bharatula, IEEEXplore, 2024.