Parallel Implementation of Multiple Interdependent Lindenmayer Systems

Jose Javier Gonzalez Ortiz[*]
University of Michigan

April 13, 2016

Abstract

This paper introduces a botanical model to realistically simulate the growth behavior of different types of forests. The model relies on L-Systems for the individual behavior, and local interaction for the interdependence. We provide both a serial implementation and parallel implementation designed for a shared memory machine supporting OpenMP. The algorithm is modified in order to minimize thread communication and lock operations. The algorithm successfully parallelizes the time dimension to minimize the runtime. The forest model obtained is satisfactory and the parallelization techniques used are not bound to the problem at hand, and can be extended to a number of simulations with local dependency.

Keywords: *L-Systems, forest simulation, botanical models, parallel processing, OpenMP*

1 Introduction

Modern techniques for botanic and arboreal growth simulation rely in specific and dedicated models to correctly recreate the behavior of different types of plants. This creates a segmented and disconnected variety of models and techniques. As A. Lindenmayer and Przemyslaw described in [PL96], almost all botanical structures can be successfully modeled by the so called Lindenmayer Systems.

However, raw L-Systems are not complex enough to model most botanical structures, and therefore we need to include a parametric implementation to successfully model these types of structures. Moreover, parametric L-Systems portray a deterministic behavior in spite of depending on a number of initial parameters and conditions. To cope with this limitation, randomness is employed in the traditional form described in [PL96], by using Stochastic-Parametric L-Systems. These systems will have production rules that will not only rely on several parameters to correctly scalate the complexity of the system, but also apply these rules using a source of randomness to choose between a number of specified rules.

Another aspect that should be taken into account is the fact that botanical growth almost never occurs in a isolated fashion. Therefore, we will be interested in looking at a framework that can accommodate the simultaneous derivation of multiple L-Systems. Having multiple stochastic L-Systems produces a non-realistic result, as will be discussed later. To compensate for this caveat the concept of Interdependent L-Systems will need to be defined.

[*]jjgo@umich.edu

In this paper we investigate the use of Interdependent L-System to realistically model the growth of a forest. We analyze several parallel implementations of this problem in a shared memory machine using OpenMP. L-Systems are parallel rewriting grammars, which initially would seem to simplify the problem due to the inherent parallel nature of these systems. Nevertheless, this derivation step leads to highly uneven amounts of work in a great number of scenarios. Furthermore, as we shall see later, adding the interdependence to the systems serializes the problem greatly, since after every timestep each system will need to communicate a number of metrics to a variable number of neighboring systems. This problem will produce a scenario experienced by a great amount of local simulation techniques.

We will introduce several parallel algorithms with increasing complexity and increasing speedup, and compare their behavior under different types of datasets. From a simulation perspective, this would translate into different ecosystems. The final algorithm will calculate the connected components of the forest graph to initially subdivide the problem in equally complex albeit not equally balanced subproblems, and introduce a general approach to tackle this subproblems. To accomplish this, the algorithm will parallelize the time dimension by allowing some L-Systems to carry out further iterations as long as the correctness constraints are met. We will also analyze the bigger bottleneck of the algorithm and the way it tries to cope with the variability of input systems.

Overview: We will start by providing a background in canonical, parametric and stochastic L-Systems in Section 2. Section 3 will elaborate the model used to realistically simulate the problem and will discuss the serial implementation. The parallel algorithms designed to solve this problem are discussed in Section 4 and their associated results are shown in Section 5 and analyzed in Section 6.

1.1 Previous Work

General L-Systems: Prusinkiewicz and Lindenmayer gave the basic definition to the L-System algorithm and structure in [PL96]. This was further extended by the work carried out in [PM01, PJM94, PMKL01]

Parallelizing L-Systems: Lacz and Hart introduced the use of manually written pixel shaders to compute L-Systems [LH04]. A distributed memory approach making use up to 8 CPUs and the Message Passing Interface (MPI) was described in [YHL+07]. An algorithm for the Parallel Generation of L-Systems was introduced by [LWW09]. The approach was appropriate for both GPU and multi-core CPUs, parallelizing both the derivation and the interpretation of given L-Systems. The implementation provided was generic and supported parametric, stochastic and context sensitive productions. The work was further extended in [LWW10] to make the algorithm work with multiple L-Systems.

Forest Simulation: L-Systems are within the most popular grammars to satisfactory model botanical structures. The definitions introduced by [PL96] were later improved to more realistic three dimensional trees in [AK84]. Since then, both models have been used in a number of forest growth models.

The model proposed by [KS02] provides a wonderful framework to model the interdependence of L-Systems. Geometrical interpretation of parameters was used to calculate shadow cones and a carbon allocation economy was used to successfully control the growth of the trees. A later paper introduced a quasi-physical simulation of large-scale dynamic forest scenes by introducing a similar growth model and a wind field to account for environmental factors. Finally, [RLP07] introduced a new model by employing a three dimensional version of the space colonization algorithm.

2 Background

2.1 Definition

The work presented in this paper is based on L-systems. This modeling structure is briefly introduced in this section.

L-Systems are parallel rewriting systems and a type of formal grammar. They are now commonly referred to as *parametric* L-systems and defined as a tuple:

$$\mathbf{G} = (V, \omega, P) \tag{2.1}$$

where

- **V** (*alphabet*) is a set of symbols containing the elements in the string that can be replaced.
- $\boldsymbol{\omega}$ (*axiom*) is a string of symbols from V defining the initial state of the system.
- **P** is a set of *production rules* which define the way variables can be replaced with combinations of constants and variables. Each rule is composed by a *predecessor* and a *successor*. The successor will consist of a list of symbols that will replace the predecessor. The predecessor usually contains only one symbol, if it involves more, the system is called *context sensitive*. If a production rule is not specified for a symbol, the identity production is assumed.

Parametric L-systems: We can further expand the definition of L-systems to accommodate parameters. In a parametric grammar, each symbol in the alphabet has a list of parameters associated. Parameters are usually real valued, but there is no constraint in the mathematical structures that can be used. Production rules need to be extended to deal with parametric symbols. Parameters can be both *local* or *global*, depending on whether they belong to the current predecessor or not. Parameters are used both in conditional statements to choose between different production rules, and for modifying the parameters of the symbols in the successor.

In the following example (2.2) we can see the behavior, where g is a global parameter.

$$A(x,y) : x < 2 \quad \rightarrow \quad A(g, x+1)CB(3,y) \tag{2.2}$$

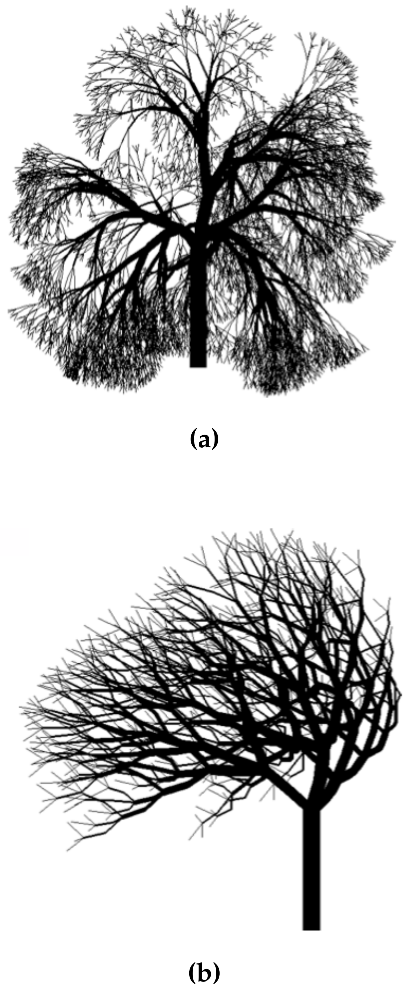
Stochastic L-systems: In order to deal with nondeterministic models, randomness can be introduced in the model by defining for each production rule $p \in P$ a set of production rules $\overline{p} = \{p_1, p_2 \ldots p_k\}$ with associated probabilities $\overline{q} = \{q_1, q_2 \ldots q_k\}$ and a random variable R that will evaluate the rule p as p_i with probability q_i. This kind of grammars are tremendously useful when generating great amounts of L-systems, because otherwise all the elements in the group would look identical.

2.2 Application

After defining the specific type of grammar or combination of grammars, to generate the geometry described by a L-System we need to execute two stages: the *derivation* of the final symbol string, and the *interpretation* of such string to a particular geometry or structure.

Derivation: this phase involves a number of successive interpretation iterations. In each iteration, all symbols in the current state are translated in parallel by using the production rules. For each symbol, a production rule with a matching predecessor and condition is searched. Once found, the symbol will be substituted by the parametric symbols in the successor. These parametric symbols are evaluated using the global parameters at the current iteration and the local parameters of the symbol that is being substituted. The initial state is the axiom of the system. The state string is updated after each iteration, so the final state is specified by the arbitrary number k of iterations that we have defined.

Interpretation: after obtaining the final state string of symbols, a set of interpretation rules will have to be applied to extract the information of the system. Most commonly, a geometric representation is generated by using two or three dimensional Turtle Geometry [AD86]. This translates the commands into modifications of the turtle state, which is represented by a position in space and a angle orientation in such space. Most modifications can be usually associated with euclidean affine transformations. However, *branching* (or *bracketed*) systems will also use push and pop commands to input the current *turtle state* into a stack for later recovery.


(a)

(b)

Figure 2.1: *Examples of trees with ternary branching*[PL96]

3 Interdependence

Three dimensional parametric and stochastic L-Systems are a really powerful tool when modeling complex botanical structures such as monopodial, sympodial and ternary trees. By varying global parameters we can regulate some fixed parameters, such as the way gravity affects the tree or the branching angle that the tree will follow. We can visualize these ideas in the ternary trees shown in Figure 2.1.

However, whereas we need stochastic systems as a source of variability to realistically emulate the structure of forests, undesired behavior arises when using it carelessly as shown in Figure 3.1. Having independent systems growing in parallel usually produces unrealistic and odd looking results, where one of the plants will have grown bigger after a few iterations (and will keep doing so); whereas other plants have carried average iterations resulting in similar systems. Therefore we need some mechanism to synchronize the growth of the systems.

We can employ both the global parameters to the system, and universal parameters to the whole group of systems, to thwart this undesired behavior. This was done in a similar fashion by [KS02]. These parameters display the relation between different systems or different parts of the same system, and are usually updated after each derivation step with the information provided by different metrics on the current string state. We can differentiate two types of relations depending on the locality of the parameters:

a) **Selfdependence** - Relation between different parts of the same system. For example, we can consider a tree whose production rules depend on the amount of light received by each branch. This system would need to update the growth parameters after each timestep, considering the shadows projected by its own branches.

b) **Interdependence** - A more complex case that involves relations between potentially different types of L-Systems, and that reflects a reciprocal interaction between them. We can extend the previous example to consider a number of trees that not only consider the shadow of each to itself, but also the shadows projected from other trees.

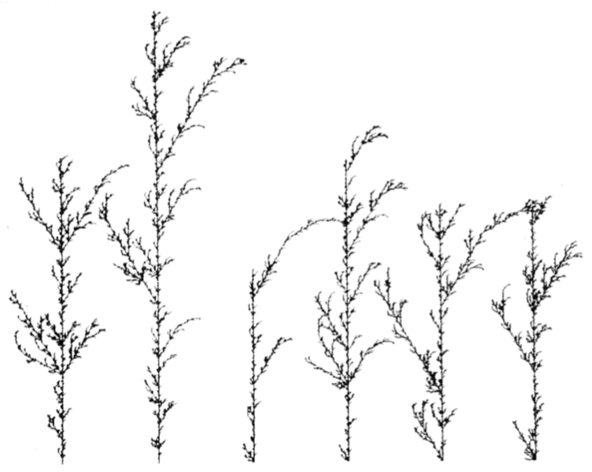


Figure 3.1: *Simulated growth of several simple stochastic L-Systems*[PL96]

Independent and selfdependent systems provide a embarrassingly parallel problem. Since each system can be analyzed without considering the others, they can be treated as individual problems. Load balancing can be achieved with a simple manager-worker scheme using a shared queue. However, interdependent groups of L-Systems can pose a nontrivial parallelization given appropriate constraints. For this paper we have used a local resource economy which translates in the following behavior. A tree will only be affected by the trees within a radius R, which from now onwards will referred to as his *neighbors*. If a tree is bigger than the average size of their neighbors then it will be harder for it to grow at the next iteration, preventing the overgrowth behavior depicted in Figure 3.1. If a tree is smaller than the average size of its neighbors, then it will be more likely for it to grow in the next iteration, compensating the randomness in previous iterations. These constraints need to be used since, whereas randomness is a good source of variability for the population, it deeply alters the growth rate of different systems.

We still need to define what the *size* of a tree means from an analytical viewpoint. We will use the concept of *metrics*: functions that take as input a list of parametric symbols, and output a list of real valued parameters. The output will reflect different properties of the system. The metric used in our implementation measured both the number of nodes (points where branching occurs) and leaves (terminal points of the system) to account for the dimensions of the tree.

A serial implementation of the described scenario was carried outA fixed number of different L-Systems composed by a given probability distribution of monopodial, sympodial and ternary trees was randomly distributed in a square surface. All these tree systems where slightly modified from the Honda canonical definitions given in [PL96] to accommodate for randomness in the growth. Next, a neighbor look-up list was initialized, since neighboring relations do not change throughout the simulation. For a fixed number k of iterations, each system was applied the derivation procedure and its metric recalculated and stored in a global array. Then, before starting the next iteration, the growth parameter of each system was updated using the metrics from its neighbors.

This implementation allows for huge variability of environments. Modifying the probability distribution of trees, the density and the number of iterations allow us to model from old, dense boreal forests to a widely spaced savanna.

4 Parallelization

When looking at a decently sized group of L-Systems ($N \approx 100$) and a considerable number of iterations ($k \approx 14$), the serial implementation starts taking seconds on most modern machines. Trying to increase any of the parameters goes to the domain of minutes and even hours. Therefore, the problem can use parallelization techniques in order to reduce the computation time by introducing more processing power. For this analysis we have chosen a shared memory architecture using the OpenMP application programming interface in C++. In this paper we are going to focus in the parallelization of the derivation step for the interdependant system described in Section 3.

The interpretation step of multiple L-Systems is an inherently embarrassingly parallel problem since all of the systems are independent when it comes to interpretation. To further optimize the parallelization of the interpretation step, one can look at parallelizing the interpretation of an individual L-System. This can be easily done for *non-branching* systems by replacing each symbol with its associated affine transformation (4x4 matrix). Since matrix multiplication is an associative operation we can easily solve the problem by using a *reduce* function. *Branching* systems can follow a similar approach but the complexity escapes the scope of the current paper. A more detailed explanation can be found in [LWW10].

4.1 Dynamic scheduling

An initial naive parallelization can be used by just simply parallelizing each one of the derivation iteration steps. Using a dynamic scheduling algorithm, the processors will evenly split the work and solve each iteration in parallel. We consider dynamic scheduling instead of static scheduling to get a better load balancing scheme. Even though we know the amount of subproblems from the start, we cannot evenly divide them into the processors since the complexity of each system will be variable due to the stochastic formulation that the trees are following. Dynamic scheduling will be dealt in a *first-come, first-served* manner to optimize allocation

As we shall see later in Section 5, this approach decreases the overall time but is far from perfect. The main disadvantage associated with this approach comes from the fact that two barriers need to be set up at each iteration to ensure correctness. This will imply that for p processors and N similarly sized trees, at the end of each iteration approximately $r \equiv N \mod p$ processors will be still working and $p - r$ will be idle, waiting for the rest to finish and wasting processing power. On average, this means that half of the processors will be idle for approximately $\frac{1}{\lceil N/p \rceil}$ part of the time. As p grows larger this fraction also becomes larger which is undesirable. Furthermore, the presence of the two implicit *barriers* per iteration produces a great overhead since all threads will be synchronized after each iteration. Therefore, we will need to reduce all of this communication to a minimum.

4.2 Lookahead strategy

Given the constraints of the problem we can enhance the parallelization by taking into account two factors:

Local dependence - As we explained before, the simulation has only a local dependence from a tree to its neighbors. Therefore, we need not wait for every tree to finish iteration i to start iteration $i + 1$. A given tree t at iteration i only needs the metrics of its neighbors at iteration i to start the derivation phase. By applying this mechanism we can make use of every processor available all the time; however, we will need to introduce series of constraints into the implementation.

An additional verification step must be performed to test if all the neighbors have in fact reached at least the current iteration, and proceed only if the minimum of their iterations is bigger or equal the iteration of the system at hand.

The shared array that previously stored the metrics at each iteration will need another dimension for all the iterations, otherwise, we could potentially be reading a metric from a future iteration of a given neighbor, and correctness would be violated. Therefore, we will need to store for each tree t the metric at every iteration i.

Since no two processors will try to update the same metric or iteration variable during the entire program and reading not yet updated iteration variables does not affect the correctness; we will not need mutual exclusion to modify the metric variables. Accordingly, the communication is clearly lessened to only the neighbor checking and the work scheduling.

Geometry of the subproblems - Since the simulation is running in parallel over all the N trees for k iterations we know in advance that $N \cdot k$ derivation steps will need to be done. We can dynamically schedule all this subproblems via an ordered work queue, so that the processors can take one subproblem at a time and if the derivation is possible (the neighbors have reached all the previous iteration) they will proceed, otherwise they will return the work to the queue and try the next available subproblem.

Since we are using an approach that relies on the connectivity of the trees in the forest, we can express the neighboring relations as a undirected graph. An arbitrary queue will render a great amount of the neighbor checking steps unsuccessful, since we cannot predict which vertices have been derived yet and which not.

However, we can make use of the inherent geometry of the problem by employing a graph traversing algorithm such as *Breadth First Search*. By performing an initial *BFS* traversal into the graph and storing this precalculated order we can ensure that if at every iteration, the nodes are interpreted in the same order, then we will be maximizing vertices whose neighbors all have the same iteration value.

Consequently, the algorithm will start deriving at a given starting point of the graph and traversing it following BFS. When the other end is reached, a number of processors may not have work to do in said end, so they can go back to the starting point of the graph and with a really high probability, they will be able to derive the next iteration of the vertices there, maximizing the processing power.

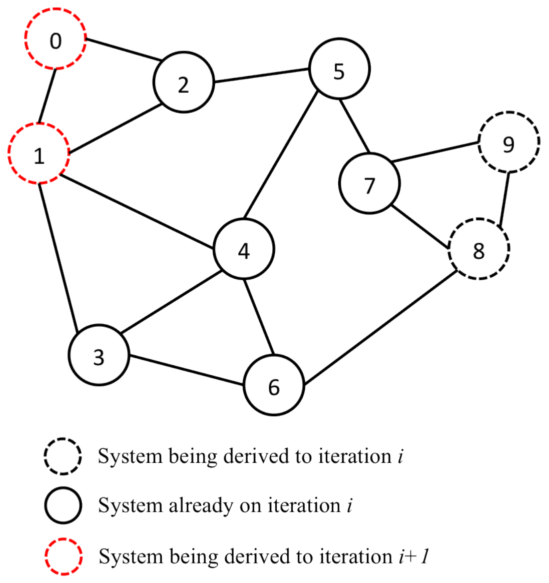

0
1
2
3
4
5
6
7
8
9
System being derived to iteration i
System already on iteration i
System being derived to iteration i+1

Figure 4.1: *Example of the algorithm behavior between iterations i and $i + 1$ with $p = 4, N = 10$*

We can see a diagram that reflects this process for a small input in Figure 4.1. With $p = 4$ and $N = 10$, at the end of iteration i, two processors would be idle in the barrier.Now, we can allocate them with work in the opposite side of the graph so they can start deriving iteration $i + 1$.

With all this optimizations, the program works as fast as before when $p \mid N$, and much faster when $N \mod p$ is really small. Since the previous implementation involved a big fraction of the processors being idle for an entire derivation step and the complexity of the derivation steps grows exponentially, the new algorithm enhances the simulation runtime significantly.

4.3 Connected components optimization

A further optimization can be made. The main disadvantage of the solution proposed in section 4.2 comes from the fact that it introduces a whole neighbor checking phase that can consume a non-negligible amount of resources if the graph presents great connectivity. Also, the simulation uses as inputs dense and sparse groups of trees in order to satisfactorily model different types of environments. This implies that in some cases we will have one single highly connected graph, and in other cases we may have several smaller subgraphs or even individual vertices.

Moreover, as briefly discussed in the previous section, the neighbor checking may be unnecessary in some cases. Specifically, when all processors are executing trees in the same iteration, neighbor checking needs not be done. By using a series of flag variables we can check if all the processors are in the same iteration and skip the neighbor checking.

In addition, to improve the algorithm we can also make use of the fact that we are performing a initial BFS to compute an order of visit to the vertices and also identify the different connected components of the graph. This is useful, since connected components in the graph render completely independent subgroups of L-Systems that will not share any information whatsoever during all the iterations. Using this information we can split the main queue into smaller queues, which would reduce the overhead of scheduling the work. We just have to make sure that for a connected component with n_i nodes, the number of processors in that queue is always $p_i < n_i$. A simple way to solve this problem is allowing idle processors to dynamically jump from queue to queue until they find work they can process.

4.4 Splitting L-Systems

To try to extend the parallelization further, an attempt was made to parallelize the derivation of individual L-Systems. This rendered a problem when either N was really small or k was exceedingly big. To successfully divide a L-System into smaller work items, we can use the fact that L-Systems are parallel rewriting grammars. Since all the L-Systems covered in this implementation are context free, we can evenly divide the number of symbols in the system into m different pieces and add them into the dynamic queue. It is important that they are not dispersed in the queue, since as we analyzed before, the neighbor checking pattern works better when given a BFS order for the nodes. The L-Systems were slightly modified to cope with the multiple updates that now a derivation step needs to do. The iteration value will not change until the last derivation part is finished.

Following this scheme, we successfully reached a correct implementation that splits the work for L-Systems into several subproblems. This strategy is only applied when there are idle processors that depend on a concrete L-System to continue their work. Unfortunately, the overhead involved in checking if one L-System is bottlenecking the derivation and maintaining a queue with partial L-System states rendered the solution inefficient. By adding the splitting L-Systems technique, the serial was increased by an order of magnitude for almost all inputs. Consequently, this implementation was not used for the final results.

5 Results

To thoroughly test the parallelization, the parallelization techniques described in section 4, were implemented incrementally and tested independently. As data inputs we used a variety of tree models including monopodial, sympodial and ternary trees depending on the density properties of the forest. They all shared the same defining parameters, since forests are usually made of trees with similar characteristics. Randomness was employed to place the points in a plane. A square surface was used, whose size was randomly established at the beginning. The value for the size oscillated within the range $\sqrt{\sqrt{N}} \leq s \leq \sqrt{2N}$, since bigger sizes led to very sparse systems and smaller systems produced highly concentrated systems, and neither of them resembled a realistic forest model. Therefore, by randomizing the area we are getting a random density for each testcase.

As we also explained in previous section randomness is a crucial part in the derivation of stochastic L-Systems. At the interpretation of every nontrivial symbol, randomness is used to decide between using the default production rule or the plain identity. This coupled with the interdependence strategy models the forest behavior as desired.

We made an analysis with numerous values for N and p and realized that in general, the strategies outlined in Section 4.1 and 4.2 have significant improvement but the optimization outlined in section 4.3 was only marginally better than 4.2. The latter version worked in less time for a really specific type of inputs and in general, the speedup was compensated by the added overhead.

Therefore, in the figures and plots we are going to differentiate between the initial algorithm with just sequential dynamic scheduling, which will be denominated "*dynamic*", and the one that takes the rest of the optimizations into account, which will be referred to as the "*lookahead*" algorithm. Since adult trees require between 11 and 14 iterations to "grow" we have used $k = 12$ for the forests. Now, for the input sizes, we have $N = 65, 110, 220$ which were reasonable values for formulated model. Smaller sizes did not involve enough work, whereas bigger sizes led to almost identical results as $N = 220$

5.1 Timing results

The obtained timing results are gathered in Tables 5.1a and 5.1b. A great variety of number processor amounts were used $(1, 2, 4, 8, 16, 32)$. Averages were taken in order to get uniform results for the different random inputs and to eliminate potential machine inefficiencies introduced in the execution of the program.

For a better understanding of the behavior of both algorithms for the data covered in both tables, the figures depicting timing results, SpeedUp and Efficiency are provided (Figures 5.1, 5.2a, 5.2b).

$N \setminus p$	**1**	**2**	**4**	**8**	**16**	**32**
65	1.060	0.767	0.411	0.262	0.121	0.104
110	1.881	1.241	0.675	0.412	0.203	0.152
220	3.462	2.418	1.324	0.682	0.388	0.279

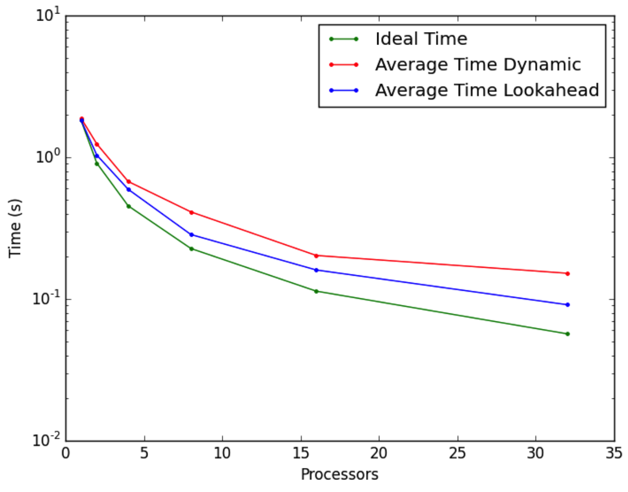
(a) *Times for the dynamic algorithm (k = 12)*

$N \setminus p$	**1**	**2**	**4**	**8**	**16**	**32**
65	1.150	0.734	0.363	0.191	0.112	0.077
110	1.817	1.036	0.594	0.285	0.160	0.094
220	3.295	2.380	1.073	0.538	0.306	0.174

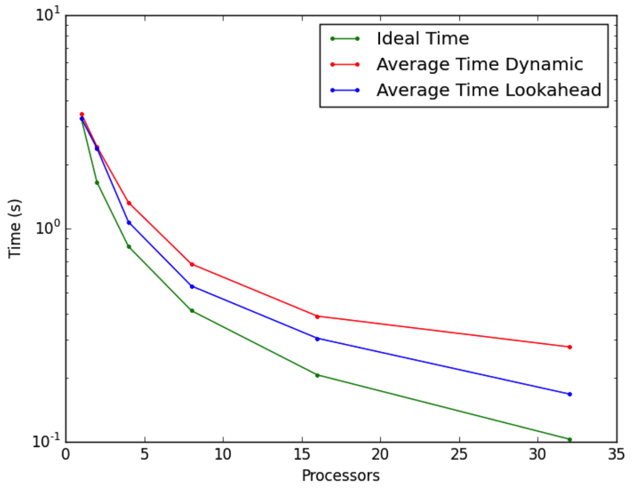
(b) *Times for the lookahead algorithm (k = 12)*

Table 5.1: *Timing results (in seconds) for both algorithms*

As we see in Figure 5.1, both algorithms successfully parallelize the problem. The *lookahead* algorithm runs on average faster than the *dynamic* algorithm, roughly by a factor of two.

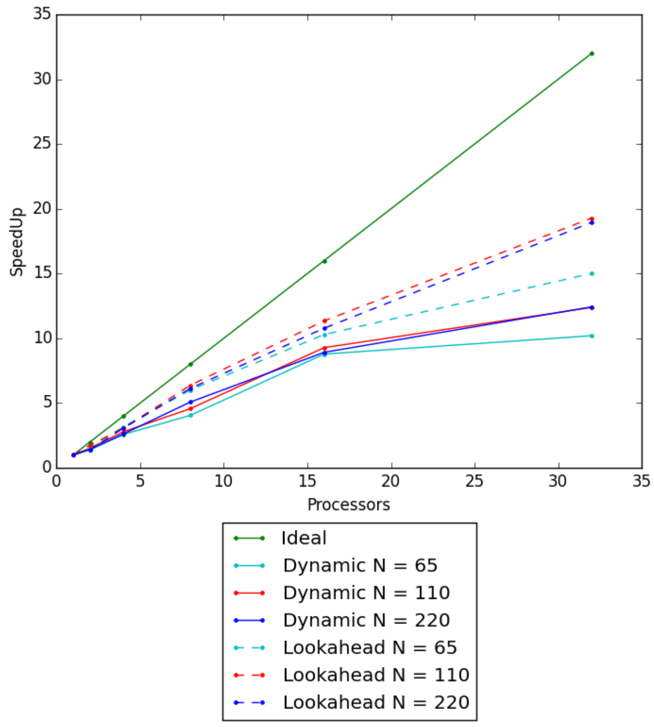


(a) $N = 110$

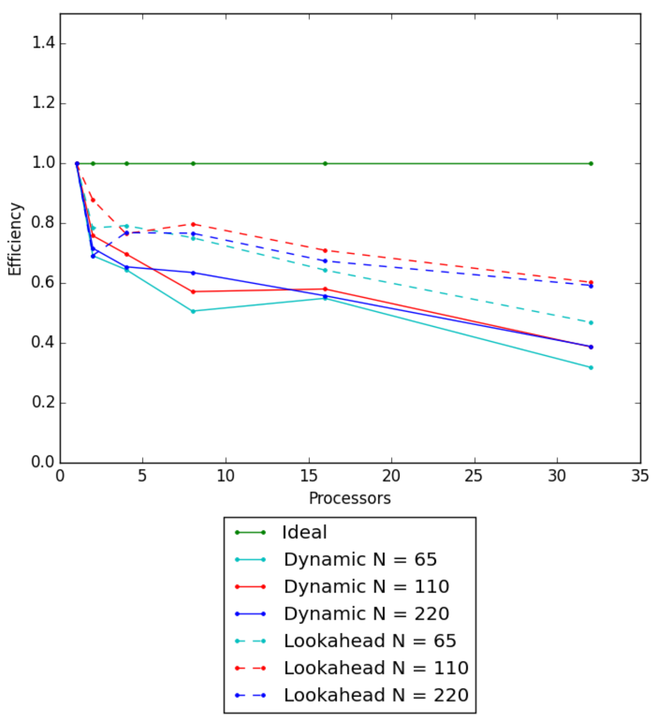


(b) $N = 220$

Figure 5.1: *Logarithmic plot of times for different number of processors and inputs*



(a) *Speed-Up*



(b) *Efficiency*

Figure 5.2: *Speed-Up and Efficiency plots for the both algorithms and different input sizes*

p	ideal	average	min	max
1	1320.0	1320.0	1320.0	1320.0
2	660.0	660.0	647.0	672.0
4	330.0	330.0	315.0	345.0
8	165.0	165.0	131.0	201.0
16	82.5	82.5	47.0	137.0
32	41.2	41.2	18.0	100.0

(a) *Load balancing of systems*

p	ideal	average	min	max
1	1553837.0	1553837.4	1553837.0	1553837.0
2	734333.5	734333.8	731044.0	737623.0
4	365782.5	365782.7	360182.0	371576.0
8	184617.5	184617.5	177446.0	191610.0
16	96045.2	96045.3	88506.0	104027.0
32	45396.5	45396.5	38811.0	54650.0

(b) *Load balancing of symbols*

Table 5.2: *Results for the load balancing analysis for $N = 110$ and $k = 12$*

Looking at the SpeedUp and Efficiency (Figure 5.2a and 5.2b), we see that in fact the SpeedUp of the *lookahead* algorithm behaves not only better than the *dynamic* one, but also for a large enough input N, it gives an almost linear speedup. Bigger input sizes were also analyzed but the results obtained were almost identical to those for $N = 220$. For $N = 65$ the SpeedUp is worse because the amount of work is not large enough to compensate for the communication overhead that the dynamic scheduling involves. The SpeedUp for the *lookahead* algorithm is almost linear whereas the *dynamic* algorithm presents a linear SpeedUp for $p < 16$ and greatly decreases for $p = 32$.

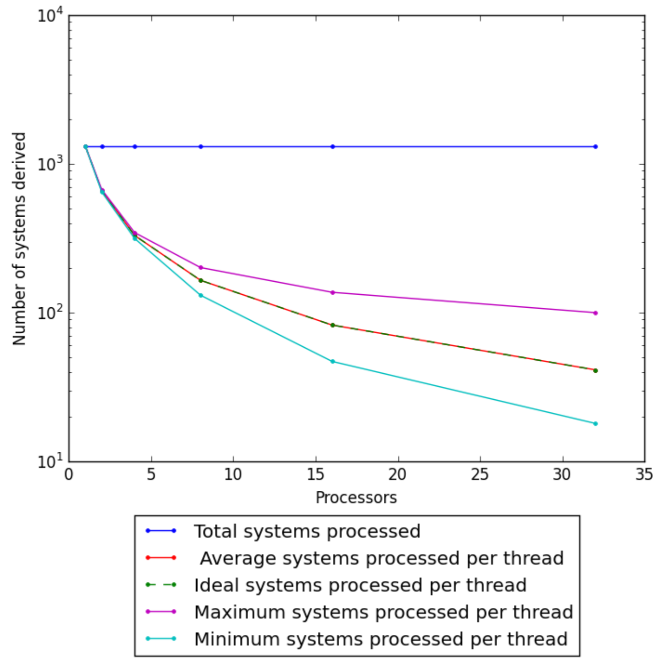
Efficiency analysis gives some interesting results. First for almost all of the testcases the efficiency is not strictly decreasing, which means that the algorithms are able to parallelize better for some specific values of p. However, we can see a decreasing trend for all of the cases, with the *dynamic* decreasing faster than the *lookahead* which agrees with the results of the SpeedUp.

5.2 Load Balancing

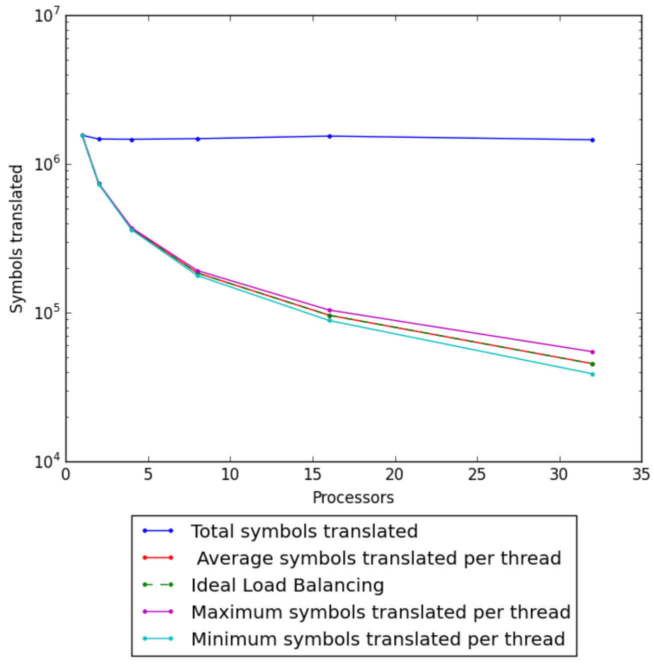
Load balancing measurements were taken for both algorithms, however, with the given inputs the results were practically indistinguishable (they both load balance correctly, the second algorithm is just able to do it faster), so we can just focus in whether the load balancing is happening correctly or not. We considered both the load balancing of systems and the load balancing of symbols and got averages of the average, minimum and maximum case in each scenario. We also computed the ideal load balancing by dividing the total number of symbols/systems by the amount of processors. This was all done for different values of N but as before, the results were identical for the different values, so only $N = 110$ is provided as a matter of simplicity. We can see the results in Table 5.2.

Again, for a better understanding of the gathered results, accompanying Figures are provided in 5.3a and 5.3b that depict both the load balancing of symbols and systems in a logarithmic scale.

Using this figures and tables we can confirm that the dynamic scheduling approach proposed in Section 4.1 is satisfactory. Figure 5.3a reveals that systems are not being load balanced. This was expected, since one of the problems we faced in this implementation was that parametric stochastic L-Systems have a variable complexity and therefore dividing them evenly between the processors via static scheduling would give subpar performance. Due to this fact we can observe the inmense variability that minimum and maximum amount of systems offer, in order to accomodate for the variable complexity of L-Systems.



(a) *Load Balancing of systems*



(b) *Load Balancing of symbols*

Figure 5.3: *Results for the load balancing analysis for $N = 110$ and $k = 12$*

Figure 5.3b truly indicates that load balancing is happening as expected and that all processors are doing an equal amount of work. In both figures we can appreciate that the ideal case is coincidental with the average case, but for symbol balancing we can see that maximum and minimum are extremely close to the ideal, suggesting that no processor is having a substantially larger or smaller amount of work.

6 Discussion

Multiple interdependent stochastic and parametric Lindenmayer systems are able to realistically simulate the growth behavior of a forest. Varying the L-Systems employed and the density of the trees allow us to model different types of environments, ranging from dense boreal forests to widely spread savannas. However, as with many procedural simulation techniques, the deriving cost comes at a price and if we want to simulate environments of considerable size, parallelization must be made to reduce the computing time needed.

From the results obtained in section 5 we can conclude that the parallelization was successful and linear speedup was achieved for the final algorithm outlined in section 4.3. It is important to note that as $N = 65$ showed, the speedup will only hold if the input size is big enough for the amount of processors used. This is a fundamental characteristic of this problem. As with almost any type of parallelization, the communication will be the biggest burden to speedup and efficiency. In this particular problem, the communication is dealt via the dynamic scheduling algorithm described in section 4.1. This involves a nontrivial amount of interaction between the processors.

Furthermore, time profiling for the execution revealed that we cannot easily increase the Speedup shown in Figure 5.2a because time needs to be spent correctly scheduling the work. Otherwise, in a static scheduling, the complexity variability of the L-Systems completely ruins the load balancing and the results are far worse.

It is also important to mention that several other values of k were tried for different input sizes and number of processors. As we explored in section 4.2, the plain dynamic scheduling behaves worse for larger values of k. However, for the cases when the difference was significantly bigger ($k > 16$), the model was overconvoluted and did not reflect any more the true appearance of a forest. Because of this, the results have not been analyzed. Albeit helping the case for the look-ahead algorithm, they do not represent a realistic scenario.

In conclusion, both the dynamic scheduling and the look-ahead strategy based in the BFS traversal have proved to be successful strategies to tackle the parallelization of the simulation. The connected components derivation just enhances marginally the behavior without adding a significant amount of overhead into the algorithm.

7 Conclusion

This report explores the viability of using parametric stochastic interdependent Lindenmayer systems to model the growth behavior of a forest, and a nontrivial way of parallelizing the problem. The final parallelization scheme involved a number of components which include a dynamically scheduled queue, an initial traversal search to determine the order of the nodes, a neighbor checking algorithm that deserialized the iteration steps allowing multiple iterations to happen at once, and a decomposition of the graph into smaller subgraphs to further optimize the results.

An important conclusion that we can gather from the analysis given is the fact that the nature of the L-Systems structure was not directly used in the parallelization (except for Section 4.4, which was unsuccessful). This means that the obtained parallelization algorithm can adequately model similar problems of the same characteristics. Namely, given a simulation problem with fixed local dependence (does not change over time), and problems that grow with the same complexity after each iteration; the present algorithm should render similar results with comparable inputs, even if the internal elements are not precisely L-Systems.

Future work: The main caveat of the produced outputs was that all trees had the same “age”, which translates in similar complexities (within one order of magnitude). This scenario is not unrealistic, (forest after fires develop in this fashion), but is far from general. We would like to analyze how the system would grow if new trees had the possibility to develop at any given iteration when certain circumstances are met, like the presence of neighboring trees. This would be an interesting problem to analyze, since adding this constraint breaks the main approach of the current parallelization.

In addition, the neighboring relation was an absolute one: either two trees were neighbors or they were not. Reality is not discrete in this regard so we could further enhance the simulation by taking into account the distance as part of the relation and defining a maximum distance for neighbors. Furthermore, as trees grow larger we could increase this radius to satisfactorily model the interdependence between the trees. Both of this additions would affect the parallelization scheme which would need to be rederived for these considerations.

References

[AD86] H. Abelson and A.A. DiSessa. *Turtle Geometry: The Computer as a Medium for Exploring Mathematics*. Artificial Intelligence Series. AAAI Press, 1986.

[AK84] M. Aono and T.L. Kunii. Botanical tree image generation. *Computer Graphics and Applications, IEEE*, 4(5):10–34, May 1984.

[KS02] Winfried Kurth and Branislav Sloboda. Growth grammars simulating trees - an extension of l-systems incorporating local variables and sensitivity. *Silva Fennica*, 31(3):285 – 295, 2002.

[LH04] Patrick Lacz and John C. Hart. Procedural geometry synthesis on the gpu, 2004.

[LWW09] Markus Lipp, Peter Wonka, and Michael Wimmer. Parallel generation of l-systems. In Holger Theisel Marcus Magnor, Bodo Rosenhahn, editor, *Vision, Modeling, and Visualization Workshop (VMV) 2009*, pages 205–214, November 2009.

[LWW10] Markus Lipp, Peter Wonka, and Michael Wimmer. Parallel generation of multiple l-systems. *Computers & Graphics*, 34(5):585–593, October 2010.

[PJM94] Przemyslaw Prusinkiewicz, Mark James, and Radomír Měch. Synthetic topiary. In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '94, pages 351–358, New York, NY, USA, 1994. ACM.

[PL96] Przemyslaw Prusinkiewicz and Aristid Lindenmayer. *The Algorithmic Beauty of Plants*. Springer-Verlag New York, Inc., New York, NY, USA, 1996.

[PM01] Yoav I. H. Parish and Pascal Müller. Procedural modeling of cities. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, pages 301–308, New York, NY, USA, 2001. ACM.

[PMKL01] Przemyslaw Prusinkiewicz, Lars Mündermann, Radoslaw Karwowski, and Brendan Lane. The use of positional information in the modeling of plants. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, pages 289–300, New York, NY, USA, 2001. ACM.

[RLP07] Adam Runions, Brendan Lane, and Przemyslaw Prusinkiewicz. Modeling trees with a space colonization algorithm. In *Eurographics Workshop on Natural Phenomena*, 2007.

[YHL$^+$07] Tingjun Yang, Zhengge Huang, Xingsheng Lin, Jianjun Chen, and Jun Ni. A parallel algorithm for binary-tree-based string rewriting in l-systems. In *Computer and Computational Sciences, 2007. IMSCCS 2007. Second International Multi-Symposiums on*, pages 245–252, Aug 2007.

[ZST$^+$06] Long Zhang, Chengfang Song, Qifeng Tan, Wei Chen 0001, and Qunsheng Peng. Quasi-physical simulation of large-scale dynamic forest scenes. In Tomoyuki Nishita, Qunsheng Peng, and Hans-Peter Seidel, editors, *Computer Graphics International*, volume 4035 of *Lecture Notes in Computer Science*, pages 735–742. Springer, 2006.