Journal of Zhejiang University-SCIENCE C (Computers & Electronics)
ISSN 1869-1951 (Print); ISSN 1869-196X (Online)
www.zju.edu.cn/jzus; www.springerlink.com
E-mail: jzus@zju.edu.cn

A GPU-based multi-resolution algorithm for simulation of seed dispersal*

Jing FAN†, Hai-feng JI, Xin-xin GUAN, Ying TANG†‡

(*School of Computer Science and Technology, Zhejiang University of Technology, Hangzhou 310023, China*)

†E-mail: fanjing@zjut.edu.cn; tangying@gmail.com

Received May 23, 2012; Revision accepted Aug. 9, 2012; Crosschecked Oct. 12, 2012

Abstract: In forest dynamics models, the intensive computation and load involved in the simulation of seed dispersal can become unbearably huge for large-scale forest analysis. To solve this problem, we propose a multi-resolution algorithm to compute seed dispersal on GPU. By exploiting the computation parallelism of seed dispersal, the computation of the whole forest plot is divided into multiple small plot cells, which are computed independently by parallel threads on GPU. To further improve the calculation efficiency with limited threads scale for GPU computation, we propose a hierarchical method to cluster the plot cells into a multi-resolution form according to the biological curves of tree seed dispersal. Experimental results show that our algorithm not only greatly reduces computational time but also obtains comparably correct results as compared to the naive GPU algorithm, which makes it especially suitable for large-scale forest modeling.

Key words: GPU, Seed dispersal, Large-scale, Multi-resolution, Data clustering
doi:10.1631/jzus.C1200147 **Document code:** A **CLC number:** TP391.9

1 Introduction

Forest dynamics models are powerful tools for effectively simulating forest evolution processes on the computer without the need to perform real-world experiments. In forest dynamics models tree development undergoes several stages from seed to death, which are seed, seedling, youth, and adult (Pacala *et al.*, 1993; Govindarajan *et al.*, 2004; Kunstler, 2011). During all these stages, the initial establishment step, i.e., seed dispersal, determines population spread and local abundance across landscape and is very important for the accuracy of the following developmental stages. In spatially explicit forest dynamics models (Bugmann, 2001; Govindarajan *et al.*, 2004), adult trees produce seeds and distribute them across the uniformly divided forest plot of explicit coordinates during the seed dispersal stage. The plot cell is regarded as an independent calculation target and used to index the tree positions (Clark *et al.*, 2001; Govindarajan *et al.*, 2007). The computation of dispersed seeds at a specific location is closely related to the sizes of the adult trees which produce seeds and the distances between those adult trees and the target location. As the vital spatio-temporal component of forest simulation, the calculation of seed dispersal is very time-consuming since to calculate the quantity of seeds distributed across the forest we have to traverse all adult trees for each target location. The time complexity is proportional to the product of the plot size and the number of adult trees. When the number of trees increases or the size of the forest becomes larger, the amount of data and computation involved in seed dispersal becomes enormous and unacceptable; e.g., it requires 5 h to simulate the 100 m×100 m forest plot with 2500 trees with pure CPU implementation and without acceleration.

‡ Corresponding author

* Project supported by the National Natural Science Foundation of China (Nos. 61173097 and 61003265), the Natural Science Foundation of Zhejiang Province, China (No. Z1090459), the Science and Technology Planning Project of Zhejiang Province, China (No. 2010C33046), and Tsinghua–Tencent Joint Laboratory for Internet Innovation Technology

To efficiently compute the seed dispersal of adult trees, we design a novel GPU (graphics processing unit) algorithm which uses the spatial coherence of adult trees and target plot cells to achieve higher computational efficiency and consume fewer resources. We organize the data including trees and plot cells into multi-resolution form by adopting the hierarchical data clustering method. The hierarchical data and computation are properly mapped to the CUDA (NVIDIA Corporation, 2007) parallel computing structure, and the reasonable threads scheduling is proposed to optimize the running process of seed dispersal on CUDA. Compared with our previous work of GPU implementation without data clustering (Tang *et al*., 2011), the method in this paper effectively increases the simulation efficiency at the same scale of thread and forest, which makes our algorithm scalable for the computation of large-scale forest. The experiments performed for forest plots of various scales show that our method not only achieves a higher speed-up ratio than the previous GPU method, but also obtains results with comparable accuracy and less resource waste.

2 Related work

Since the calculation of seed dispersal is a time consuming process, Govindarajan *et al.* (2007) organized the plot and trees as quad trees to reduce the amount of data required for acceleration. They implemented the quad-tree data structure to approximately calculate the seed dispersal on CPU and achieved a speed-up ratio of 1–20 compared with the direct CPU algorithm for different plot scales. Although the result of acceleration is quite good, there is still much room to improve the speed-up ratio by taking advantage of parallelism in seed distribution computation. With the rapid development of GPU and the high-performance parallel computing architecture CUDA, the parallel computing strategy has been used in more and more general purpose applications to improve their computing speed (Stone *et al*., 2007; Nickolls *et al*., 2008; Du *et al*., 2010; Mielikainen *et al*., 2011; Xia *et al*., 2011). In the calculation of seed dispersal, each plot cell is not influenced by the surrounding plot cells. Besides, for a certain target plot cell each adult tree independently contributes their seeds without inter-influence. By exploiting the above computation independence of each plot cell and of each adult tree, the parallel seed dispersal algorithm is implemented on a GPU and the computation speed is greatly improved compared with previous CPU implementation (Tang *et al*., 2011). With a forest plot of 250 m×400 m, it is able to complete the process of seed dispersal calculation about 1000 times faster than the CPU calculation. The above GPU implementation shoots threads for each adult tree and plot cell to make the computation parallel. However, when the plot becomes larger and the number of trees increases greatly, the number of threads exceeds the limited capacity of the GPU's resource and reduces simulation efficiency.

To further improve the GPU algorithm of dispersal and more effectively use the limited threads resource, the arrangement of threads in the GPU should be optimized. Considering the seed number produced by adult trees based on the biological curve of the seed dispersal model, there are some calculations on GPU which can be simplified by exploiting the spatial coherence of adult trees and the target plot cells. To reduce the computational load of the GPU, we can cluster the adult trees which are far from the target cell as a node to make an approximate calculation. This method is very similar to the Barn-Hut approximate algorithm (Barnes and Hut, 1986) which is applied in the astronomical N-body simulation. The simulation of N-body is a system of bodies (particles) in which each body interacts with every other body continuously. If there are N bodies, it needs to iterate over other $N-1$ bodies to complete the one target body's force calculation, like the situation of our simulation of seed dispersal. In the implementation of simulation of N-body on GPU, there are several kinds of optimization algorithms derived from the Barn-Hut algorithm. The first algorithm was proposed by Nyland *et al.* (2007), called the i-parallel method. Also, there are other improved algorithms proposed based on this method, such as j-parallel (Hamada *et al.*, 2007), w-parallel (Hamada *et al.*, 2009), and jw-parallel (Wang *et al.*, 2011). All the GPU methods mentioned above have better performance than the direct GPU algorithm for N-body simulation.

However, all of these algorithms reduce the thread number by increasing the number of time steps (loops) for unit thread, which increases the total time

for simulation when the number of bodies is large. For simulation of *N*-body this may be acceptable, since the execution time does not increase sharply and can be tolerated. But the time required increases dramatically when this method is applied in the simulation of seed dispersal because of the poor capability of dealing with branches on GPU. There is no conditional branch for the calculation of the *N*-body computational kernel. However, the simulation of seed dispersal has many branching operations. So, our method does not increase the number of time steps to reduce the thread number like *N*-body.

Furthermore, for *N*-body simulation all the bodies are equal in computational position and the information of the body stored can be reused in the implementation. In our seed dispersal simulation, the plot cells and adult trees play different roles in calculation. Plot cells are divided into different resolutions for indexing and recording the density of the seed, while adult trees play the role of devoting seeds to the forest in all directions. We need only to store results and the plot cell index. So, the strategies of defining the thread block and grid size on a GPU in our simulation are different from those in *N*-body calculation. However, the shared memory allocation methods for source bodies calculated in the threads of these algorithms can be applied for the allocation of locations of trees or clustered nodes and these allocation methods are used in the multi-resolution algorithm.

3 Seed dispersal model

In forest simulation, the plot is modeled as a fixed rectangle and divided into uniform grid cells (Fig. 1). We use *X*-*Y* coordinates to represent the plot rectangle *M*. Positive *X* coordinate increases to the east, while positive *Y* coordinate increases to the north. Each plot cell is defined as $M(i, j)$. The side length of the grid cell determines the resolution of the plot grid and affects the computing precision of forest dynamics. A grid with higher resolution leads to higher precision but longer computation time and more computation resource.

In our forest dynamics models, the dispersal of seeds initiates the formation of tree individuals, and the seeds are produced by the adult trees. For computing simplicity and without loss of generality, a tree individual is represented as two cylinders (the trunk and the crown) and has an explicit location *P* (Fig. 2). We refer to the diameter at the breast height of an adult tree as DBH, which is the most commonly measured tree growth parameter. The crown radius, crown depth, or tree height can be calculated as a function of DBH.

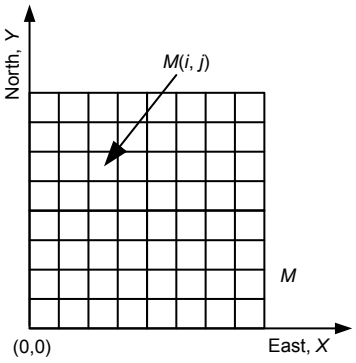


Fig. 1 Forest plot grid

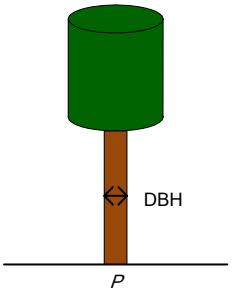


Fig. 2 Geometric model of a tree individual (DBH: diameter at the breast height of an adult tree)

To compute the seed dispersal of the whole plot, we calculate the seed dispersal of each plot cell. Fig. 3 describes the calculation of seed dispersal.

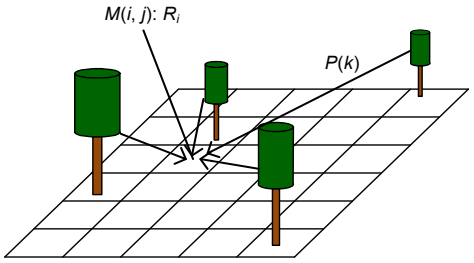


Fig. 3 The seed dispersal model

As Fig. 3 shows, the seed number produced by the *k*th reproductive adult tree is $P(k)$. If we want to

calculate the seed density of all plot cells, we need 2D loops as follows:

1. For R_i of unit plot cell $M(i, j)$, iterate over all the adult trees of the scene that are located in different distances to the target plot cell $M(i, j)$.

2. Repeat step 1 and loop over all cells of the landscape.

Adult trees that produce and disperse seeds to a plot cell are independent of each other. Seeds dispersed in different plot cells are also computed separately. The computation of the seed dispersal of one adult tree to a given plot cell can be viewed as a fine-grained sub-task, with simple logic. The computation time is closely related to the plot size and tree population. It takes a long time to calculate the seed dispersal because of a large amount of sub-tasks. We therefore adopt the GPU-based parallel algorithm by partitioning the double loops into sub-tasks and mapping these sub-tasks into threads to reduce computing time, which is introduced in Section 6 in detail.

To calculate the number of seeds dispersed per plot cell, we apply the model of the Weibull function (Lepage *et al*., 2000):

$$R_i = \sum_{k=1}^{N} P(k), \tag{1}$$

where R_i denotes the total seed number of the plot cell i, N is the number of adult trees in the forest, referring to those adult trees whose DBH values are larger than the minimum reproductive DBH of a certain species, and $P(k)$ is the seed number produced by the kth adult tree, which is measured by the change in its DBH and the distance to the ith plot grid cell. $P(k)$ is calculated as follows:

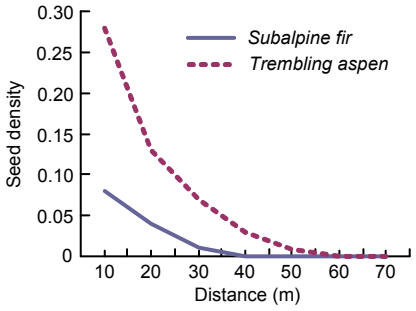
$$P(k) = \frac{1}{\eta}\mathrm{STR} \times \left(\frac{\mathrm{DBH}(k)}{30}\right)^{\beta} \times \exp(-Ud(k)^{\theta}), \tag{2}$$

where STR, η, θ, β, and U are all constant parameters related with specific tree species. The value of $d(k)$ denotes the distance between the kth adult tree and the ith plot grid cell. Eq. (2) shows that the quantity of seeds produced by one adult tree is proportional to the tree's DBH and in inverse ratio to the exponent of distance d.

4 Multi-resolution data clustering strategy for seed dispersal

4.1 Analysis of the dispersal model

From the seed dispersal model introduced in Section 3, we know that the time complexity of computing the forest plot's seed dispersal is $O(N\times G)$, where N is the plot size and G is the number of adult trees. When N and G increase largely, the calculation load becomes excessively heavy. So, simplifying the amount of data by clustering data into different resolutions is an important optimization strategy (Gelbard *et al*., 2007). To prove the feasibility of clustering data we first analyze curves of the Weibull function for two different trees. In Fig. 4, the vertical axis is the survived seed number per square meter of a certain grid cell ζ produced by one tree. The horizontal axis is the distance between the adult trees and the grid cell ζ. The dashed curve is the seed density per plot cell of species *Trembling aspen* and the solid curve is the seed density of species *Subalpine fir*. The detailed parameters (Astrup *et al*., 2007) of the Weibull function for these two species are shown in Fig. 4.



Species	DBH (cm)	STR	β	θ	D	η
Trembling aspen	30	0.2	2	3	0.000038	1
Subalpine fir	30	0.09768	2	3	0.000132	1

Fig. 4 Seed density (survived seed number per square meter) curves for two tree species *Trembling aspen* and *Subalpine fir*

Fig. 4 shows that the larger the distance between the tree and the grid cell, the fewer seeds the tree contributes. In *Trembling aspen* the seed density approaches 0 when the distance reaches about 60 m. The same occurs for *Subalpine fir* when the distance is nearly 40 m. According to the characteristic of the curves, we find that the seeds produced by trees would become negligible when they are located far

away from the plot cell. For the adult trees whose distances to the plot cells are large enough and thus their produced seeds are negligible, they can be clustered to one node to save computation time. The computation of this clustered node can be used to replace the respective calculations of the adult trees it contains. In the next subsection we will introduce the method of clustering in detail.

4.2 Multi-resolution data clustering for seed dispersal

In the forest dynamics model, we divide the plot into uniform cells until the unit plot cell contains at most one tree. Thus, the plot is transformed to a high-resolution grid, of which the size of the unit cell is usually 1 m×1 m and each adult tree can be explicitly indexed by the cell. So, in the implementation, the cells are clustered according to their distances to the target tree. As the plot can be evenly divided into $2^k \times 2^k$ plot cells easily, 2×2 cells are clustered to form the parent node. The center of the node is defined as the center of the adult trees it contains. We call the cells leaf nodes and the clustered node the super node. These nodes are hierarchically clustered to build the multi-resolution data structure for seed dispersal. The clustering strategy is shown in Fig. 5, where $M(i, j)$ denotes the target plot cell which collects seeds from all adults in the forest. In the left figure, the dots denote trees of different locations. Regions r_1 and r_2 represent the different regions which are located at different distances to the target plot cell $M(i, j)$. Region r_2 is farther than region r_1, and thus more unit plot cells are clustered in r_2 than in r_1. As the figure shows, the four sub-regions in r_2 (which are all of the same size as r_1) are merged. In the right half of the figure, the dots represent the nodes after clustering whose size is proportional to the number of child cells clustered. n_1 and n_2 represent different resolution cluster nodes. n_1 consists of four leaf nodes (plot cells)

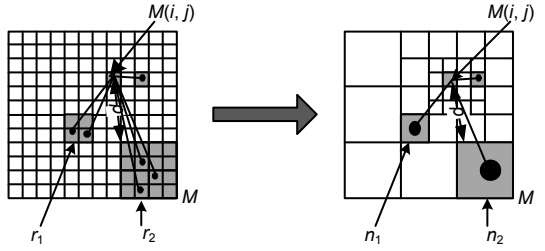


Fig. 5 The multi-resolution clustering strategy of seed dispersal

and we define its multi-resolution level as layer 1. We define n_2 as the cluster node of layer 2 and node n_2 consists of the four sub-nodes of layer 1. We can expect that the farther the distance, the larger the cluster node.

In the following we describe how to calculate seed dispersal from super nodes to target plot cells. In a node, the distance from each tree to the target cell is approximately equal and is defined as the distance d from the super node to the target plot cell. So, for the trees of one node, $d(k)$ in Eq. (2) can be expressed as d. As U and θ are constant in Eq. (2), the value of $\exp(-Ud^{\theta})$ is equivalent for each tree in the same node. The parameters STR, η, θ are all species-specific constants, so for a node that contains m adult trees, the seed density of the ith target plot cell can be expressed as

$$R_i = L \cdot \sum_{k=1}^{m} \left(\frac{\mathrm{DBH}(k)}{30} \right)^{\beta}. \tag{3}$$

Here the value of L has the following form:

$$L = \frac{1}{\eta} \cdot \mathrm{STR} \cdot \exp(-Ud^{\theta}). \tag{4}$$

All of the factors in Eq. (4) are determined for the trees of each species in a certain node. Thereby, L is constant for a certain clustered node and can be pre-computed.

To clearly and easily represent the value of a clustered node, we define variable $\mathrm{NODE}(p, q)$ as

$$\mathrm{NODE}(p,q) = \sum_{k=1}^{m} \left(\frac{\mathrm{DBH}(k)}{30} \right)^{\beta}, \tag{5}$$

where the value of $\mathrm{NODE}(p, q)$ will be used as the DBH of the clustered node p, which contains the m adult trees and belongs to layer q, and q is related with the value of d in L.

The number of adult trees m in a clustered node is also dependent on d. The larger the d, the more adult trees the node contains.

The calculation of seed density for a clustered node to a certain plot cell takes Eq. (3) only when the distance from the node to the target plot cell is larger than the threshold of clustering, which is based on the analysis in Section 4.1. For each species, the

threshold for clustering is different. For example, 40 m is set as the threshold to cluster for *Subalpine fir*, and 60 m is set as the threshold for *Trembling aspen*.

5 Data storage

5.1 Storage of tree individuals data

An object-oriented programming framework requires programmers define the class data structure which contains all related tree attributes to represent the tree entity. However, for the computation of seed dispersal, not all the attributes included in the class are involved. For example, the canopy radius of a certain target tree is not involved in dispersal computation. During the computation of seed dispersal, unused tree data wastes the very limited device memory. Besides, the tree data can be organized as a list (see the top half of Fig. 6) or multi-branch trees (i.e., quad-tree) by pointers in the main memory for CPU implementation. But the namespaces of the memory address for host (CPU) and device (GPU) are different and we should pre-process the data on a host, so it is not appropriate to use pointers. Because that array has high efficiency for access and transfer, we therefore isolate the attributes into different arrays (see the bottom half of Fig. 6). The values with the same index in an array compose a tree individual. Additionally, trees in different growth stages are stored separately (see the right part of Fig. 6). During the computation of seed dispersal, we just need to transfer the related attribute arrays, such as DBHs, ages, and locations of adults, to the global memory of GPU.

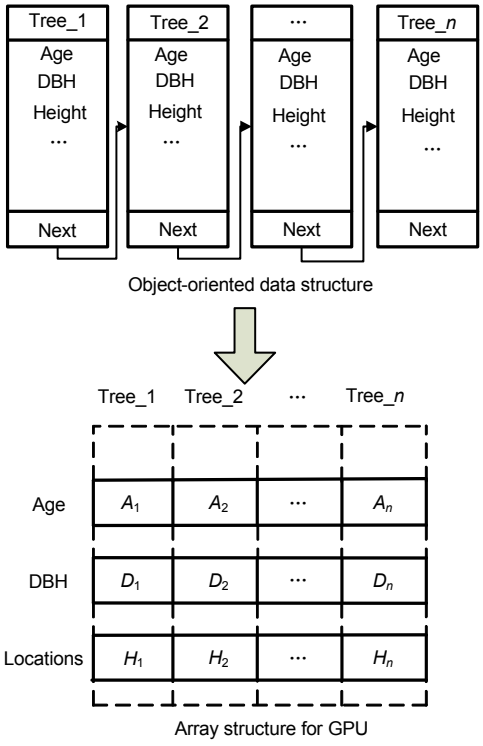


Fig. 6 Data storage applied in GPU

5.2 Storage of multi-resolution data

Arrays of different sizes are used to represent different levels of tree clustered nodes for storage of multi-resolution data based on the data structure for individual trees introduced above. The array's subscripts are used to index data. Fig. 7 shows how we organize the array structure and describe the relationship of two layers (K and K+1). Each element of the array stores the NODE (defined in Section 4) of one clustered node of a certain species. While there are n nodes in layer K+1, there are $4n$+3 nodes in layer K. The value of each parent node is the sum of the values of four sub-nodes for a certain species. For example, NODE(0, k+1) is the sum of NODE(0, k), NODE(1, k), NODE(2, k), and NODE(3, k). The above array structure is based on a single species. In actual scene applications, the arrays can be proportionally expanded for multi-species by explicitly adding the clustered nodes as NODE at the end of the array at each level for new tree species.

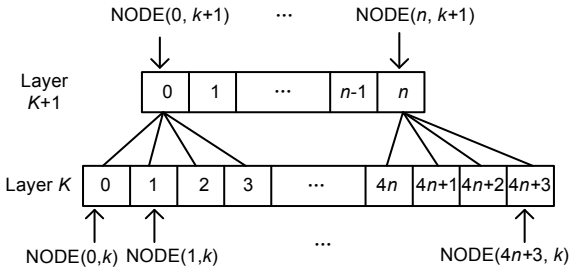


Fig. 7 Structure of multi-resolution array in GPU

6 Implementation of direct calculation for the GPU kernel

To calculate the seed density for the target plot cell, each tree in the forest is iterated according to the Weibull function. For GPU implementation the loop

iteration is replaced by CUDA threads. We have launched a 2D grid of blocks on a naive GPU to split the double loop of computation in previous work (Tang *et al*., 2011), where each block is a 1D array of threads. For clear explanation, we apply our algorithm to a single species and map one tree to one thread to calculate the seed dispersal. The implementation of multiple species is on the same principle and easy to extend. Fig. 8 shows the design of the seed dispersal kernel of the naive GPU algorithm without a clustered node. On the left side is the plot grid where dots denote the trees. On the right is the corresponding thread grid, where the X dimension indicates the trees and the Y dimension indicates the plot cells. The S-shaped curves in the thread grid are the unit threads which compute one tree in Fig. 8. The seed density of plot (i, j) is computed by the row of threads whose Y-coordinate is $i+j\times$PW (PW denotes the width of the plot grid), and the seed dispersal of plot (i, j) is the sum of these threads' computation results.

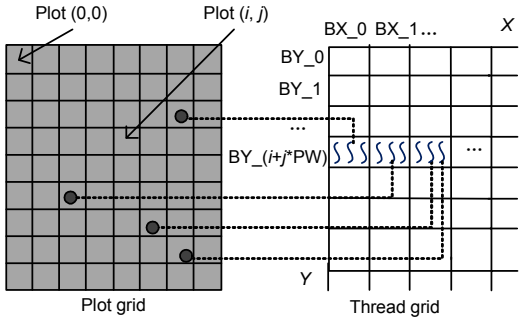


Fig. 8 The naive GPU algorithm kernel for seed dispersal

The naive GPU algorithm computes each tree's contribution to the target cell in parallel and achieves a high speed-up ratio compared with sequential loop implementation on a CPU. When the number of trees increases greatly, however, the threads in the X dimension become so numerous that they exceed the limit of the GPU computing resources.

To reduce the thread scale of the X dimension, we schedule one thread to treat t adult trees as an adult set; for example, we can group a row of trees or a column of trees as one thread. Since there are some conditional branches for the calculation of dispersal (such as the branches based on the condition whether one adult tree's DBH is larger than the minimum reproductive DBH of a certain species), the value of t should not be too large in order to ensure that each thread would not cost excessive time in the execution. Otherwise, the time of the entire simulation would increase sharply due to the limited and weaker logic processing capability of GPU compared with CPU. In the implementation, the experimental results show that setting the initial value of t to 10 is an appropriate choice considering the balance between the thread scale and the efficiency. When the value of t is doubled, the total execution time grows to 3.3 times the original cost of the initial t=10 at the scale of 10^5 trees.

7 Implementation of multi-resolution calculation for the GPU kernel

7.1 Design of multi-resolution calculation on GPU

To better solve the problem of the excessive amount of threads in the X dimension for large-scale forests, the trees can be clustered according to the strategy introduced in Section 4 to reduce the trees involved in the computation of seed dispersal. To compute the seed dispersal for clustered data on GPU, we need to design a new CUDA-based kernel to implement the multi-resolution seed dispersal calculation. Fig. 9 shows the design of the seed dispersal kernel of the multi-resolution approximation algorithm. The left figure is the plot grid and the right figure is the corresponding thread grid. The S-shaped curves represent the unit threads and dots denote the clustered nodes.

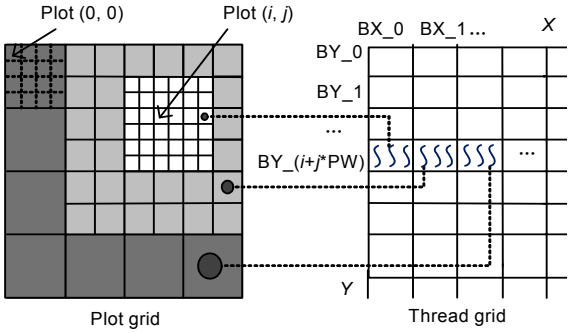


Fig. 9 The multi-resolution GPU algorithm kernel for seed dispersal

We use different clustered data nodes to replace those adult trees located at a far distance from the target cell. For example, in Fig. 9, the region with

small-sized sub-divided cells includes the trees nearest to the target cell plot (i, j). In this region, unit thread computes seed contribution of one tree for the target plot to keep computation accuracy, corresponding to the computation of the highest resolution. The region with middle-sized sub-divided cells is farther than the region with small-sized cells, and the corresponding unit threads compute the clustered node including several trees. For the farthest region with large-sized sub-divided cells in Fig. 9, unit thread computes a larger clustered node including more trees, corresponding to lower resolution computation. In our implementation, multiple nodes can be scheduled per thread. However, the number of nodes per thread is limited as we have explained in the implementation of the naive GPU algorithm. The final seed density of plot (i, j) is the accumulation of results of those unit threads.

7.2 Implementation of multi-resolution calculation on CUDA

In this subsection we compare the scale and computational complexity of the threads in our algorithm with those in the naive GPU algorithm. In the naive GPU algorithm, assume the number of plot cells is n and there are k adult trees in the forest. In our implementation the size of the Y dimension is n. When t adults are computed for unit thread, the size of X dimension is k/t. Thus, the total amount of threads for the calculation of seed dispersal is nk/t. For multi-resolution clustering, the adult trees which are computed by the thread of X dimension are replaced by the clustered nodes. For the dispersal calculation of a certain plot cell, the total number of clustered nodes computed in the X dimension is about the log value of the original number of plot cells since the original plot cells are organized into a quad tree (Section 4.2). We define the number of nodes of the quad tree as m. When t clustered nodes are computed for unit thread, which is the same as in the naive GPU algorithm, the total amount of threads for the calculation of seed dispersal is nm/t. The amount of reduced threads compared with naive GPU implementation is $n(k-m)/t$, where k is linearly related to n and m is log-related to n. The computational complexity of the naive GPU algorithm is $O(n^2)$, while the computational complexity of threads to compute one plot cell's seed density is $O(\log n)$. Therefore, the whole computational complexity of computing all plot cells is $O(n\log n)$ for a multi-resolution algorithm.

Kernel steps:

1. Get the block index k of the Y dimension which indicates the kth plot (denoted by BY_k). Get the thread's index of the X dimension which indicates the index of a cluster node.

2. According to the block index k of the Y dimension, get the location of plot k and calculate the distance d between the node and the target plot. According to the different criteria of distance (Section 4), determine whether to use the higher multi-resolution clustered node. For example, if the criterion of layer 2 is ρ_2 and d is greater than it, then the clustered node is not divided; otherwise, it is divided.

3. If the node is a leaf node or need not be divided in step 2, get the data (i.e., the value of NODE) of the node through the X dimension index. Afterwards, compute the seed density of the target plot depending on the distance d and the model function.

4. If in step 2 the node is divided, the thread gets the data of the node's children and repeats step 2.

Since the clustering has the feature of spatial locality, it is highly possible that the neighbor clustered nodes are in the same layer as the target cell. Thus, it greatly reduces the frequency of threads switching among different levels in one warp (In the architecture of Tesla on GPU, one warp contains 32 threads) (Ryoo *et al*., 2008), which can increase the thread executing efficiency.

7.3 Device memory access and allocation

The GPU performance is dependent on not only parallel computation of tremendous threads but also high memory bandwidth. Optimal organization of memory access has great impact on performance. We should take advantage of each type of memory to maximize the throughput of memory access. In this subsection, we introduce the access and allocation of three kinds of GPU memory (constant, global, and shared) in our implementation.

Constant memory resides in device memory (GPU memory). It is cached to increase the throughput of device memory and used to store constant parameters which are frequently required during calculation. Arrays for the species-specific parameters STR, U, η, θ, β (Fig. 10) are built and the size of each constant parameter array is the same as the number of the

species of tree. In Fig. 10, T_k means the kth thread in the block.

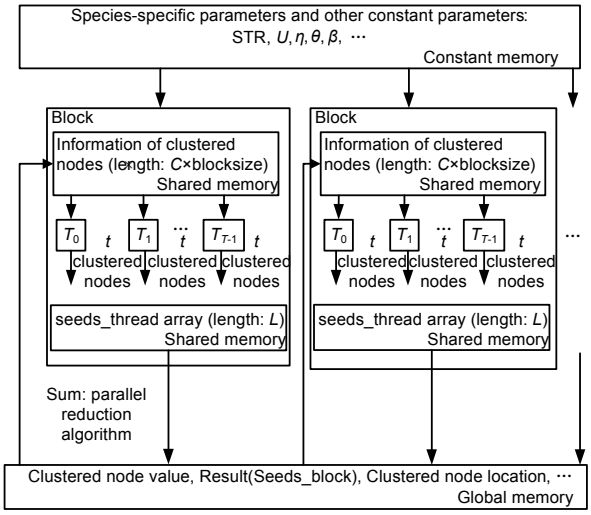


Fig. 10 Device memory access design

Global memory is the main memory of GPU which occupies a large space of linear memory. The value and location of clustered nodes and result arrays are stored in global memory. The size of the memory allocated to the clustered node is the total size of arrays of all layers. The structure of multi-resolution arrays is introduced in Section 5.2. In our system, the percentage of data residing in global memory reaches about 19% when the scale is 5×10^4 and 51% when the scale is 5×10^5. Considering the large amount of data, it is a time consuming procedure to transfer data between device (GPU) and host (CPU) memory. For a more efficient data transfer, we use the 2D array copy function to transfer data and define row length in the clustered node array as the pitch parameter of the function.

Besides, some other measures are taken to optimize global memory access. For fast access to the global memory, the first thread of each block in the X dimension is scheduled to calculate the element of the array (such as the clustered node array) whose index is an integral multiple of the block size, which causes merging access. Under merging access it costs only one transfer clock cycle for half-warp (In the architecture of Tesla on GPU, half-warp contains 16 threads); otherwise, it would cost 16 transfer clock cycles (Zhang *et al*., 2009).

Shared memory is visible to all threads within the block and with the same lifetime as the block. It has limited memory space but much faster access than the global memory. Since each block corresponds to the computation of one target plot cell and a certain number of clustered nodes, we use shared memory to load the information of target clustered nodes from global memory to improve memory access efficiency, which is similar to the method proposed by Hamada and Titala (2007) and Hamada *et al.* (2009) for N-body simulation. In Fig. 10 the size of shared memory is defined as C×blocksize, where C is the length of a clustered node which contains location coordinates, value of node DBH, and species. After loading operation, the CUDA synchronization function is called on to make sure that each thread can read the node information safely.

In addition, we allocate another array in shared memory, called seeds_thread, to keep record of seed density accumulated by t clustered nodes in each thread (Fig. 10). To minimize the bank-conflict, the array length is set to L, which is the same as the number of threads per block, i.e., 128. During the iterative computation of t clustered nodes, each thread accumulates the seed density into a register and writes it to the shared memory according to the index of thread within a block. The CUDA synchronization function is also called on to make sure that the array of seeds_thread is completely filled. After that we sum the element values of the seeds_thread array through the parallel reduction algorithm (Lin *et al*., 2010) to obtain the final seed density contributed by the block, and write it to the array of seeds_block in the global memory. The parallel reduction algorithm builds a tree structure over the partial sum array (Fig. 11). As the figure shows, the accumulate

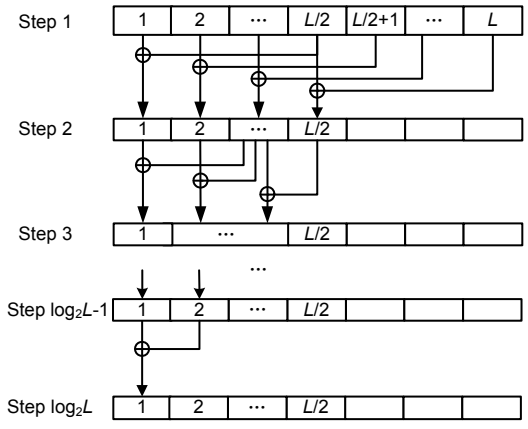


Fig. 11 Parallel reduction algorithm

operation is divided into $\log_2 L$ steps, where L denotes the length of seeds_thread. In each step, pairs of the array's elements are summed to obtain the new half-sized array which is to be processed in the next step. For example, in step k, elements of the seeds_thread in each block whose subscripts are smaller than $L/2^{k-1}$ do summing operations and $L/2^k$ threads are needed to complete the seed number accumulation. At the end of the loop, thread 0 holds the final result, and the seed density of each block is stored in seeds_thread[0].

8 Experimental results

We tested our algorithm on realistic forest data to simulate the seed dispersal of four tree species. All of our experiments were performed on a PC with the following configurations: 2.13 GHz Intel® Xeon® CPU E5506, 4 GB memory, NVIDIA Quadro 600 graphic card with 2 GB memory, and Win7 32-bit OS.

8.1 Execution time

In the kernel, the different block sizes (thread number per block) cause different numbers of resident blocks to be loaded in one stream multiprocessor, which affects the computational time of seed dispersal. First we compared the execution time of the different block sizes of our algorithm. Table 1 shows the comparison of execution time of different numbers of threads per block under different scales of plot cells and adult trees.

Table 1 shows that when the block size is 256, the performance of the computation of seed dispersal on GPU is maximized. So, under this block size, we compared our algorithm to the CPU and naive GPU. Table 2 shows the execution time and speed-up ratios of different algorithms for the simulation of seed dispersal.

Table 1 Comparison of runtime in different block sizes (128, 256, and 512 threads per block)

Number of plot cells	Number of adult trees	Time (ms)		
		128	256	512
5×10^2	10^2	105	102	107
5×10^3	10^3	187	177	198
5×10^4	10^4	771	735	796
5×10^5	10^5	20 439	19 367	21 542

As shown in Table 2, the speed-up ratio of our algorithm to CPU improves dramatically when the scale of plot and the number of adults increase. The transfer time between the host memory and device memory is negligible compared to the intensive computation time when the scale of plot and the number of adult trees are large. The table also shows that the speed-up ratio of our multi-resolution GPU can reach about 15 times that of the naive GPU algorithm when the number of plot cells is over 5×10^5. This is because when the scale of the forest increases sharply, the increasing amount of data has been simplified more evidently.

Table 2 Comparison of execution time and speed-up ratio of different algorithms when the block size is 256

Number of plot cells	Number of adult trees	Time (ms)		Speed-up ratio 1
		CPU algorithm	Multi-resolution GPU algorithm	
5×10^2	10^2	620	102	6.07
5×10^3	10^3	52 250	177	295.20
5×10^4	10^4	5 257 500	735	7153.06
5×10^5	10^5	526 500 000	19 367	27 185.42
Number of plot cells	Number of adult trees	Time (ms)		Speed-up ratio 2
		Naive GPU algorithm	Multi-resolution GPU algorithm	
5×10^2	10^2	157	102	1.54
5×10^3	10^3	294	177	1.66
5×10^4	10^4	2953	735	4.02
5×10^5	10^5	290 125	19 367	14.98

Both naive GPU and multi-resolution GPU algorithms deal with 10 trees or clustered nodes per thread

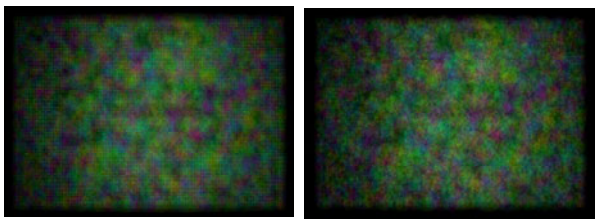
However, in Table 2 the initial scale of threads in the naive GPU algorithm is larger than that in our multi-resolution algorithm under the condition that both of them deal with 10 trees or clustered nodes per thread. For further comparison of these two GPU algorithms, under the same thread scale of 10^6 threads we tested the execution time of different GPU algorithms for the scene of 5×10^5 trees and 10^5 plot cells. In the naive GPU algorithm it cost about 320 s while the multi-resolution algorithm required only about 19 s. The speed-up ratio increased obviously and was up to 16 times. In this situation, under the same constant thread scale of 10^6, the number of trees treated per thread in the naive GPU algorithm was increased to more than the initial value in Table 2, i.e., 10 trees per thread. As discussed in Section 6, the more the clustered nodes or trees calculated by per thread, the

lower the simulation efficiency.

Therefore, our multi-resolution GPU algorithm is more effective in computing seed dispersal and useful for forests with larger scales.

8.2 Correctness of simulation for seed dispersal

To verify the results of the simulation for seed dispersal we visualized the density of the seeds by colored density maps. All the initial data of trees was collected from the 52 m×89 m forest which is located at a moist cold subzone near the town of Smithers in central British Columbia. The original forest data was provided by Dave Coates, a leading researcher of British Columbia Forest Service. In the pictures, different species are represented by different colors, and the deeper the color, the greater the seed density. Fig. 12 shows the density maps of seeds produced by the adults for two different algorithms after a growth cycle for hybrid species, including *Subalpine fir*, *Lodgepole pine*, *Tnterior spruce*, and *Trembling aspen*. From these visualized results in Fig. 12, we observe that the color distribution is nearly the same, which means that the results of seed densities computed by the two methods are very close.



(a) (b)

Fig. 12 The seed dispersal results obtained by the naive GPU algorithm (a) and the multi-resolution GPU algorithm (b)

In addition to perceptual similarity, we calculated the relative error between the accurate result of naive GPU and our approximation result. For each plot cell of the landscape, the scope of the relative errors ranged from 0 to 4.1%. The average error of all plot cells was 1.72%. Compared with the naive result, our multi-resolution algorithm had a slight loss in accuracy which is acceptable for fast simulation. To further confirm the correctness of our algorithm, we computed the number of young trees growing from seeds each year. Since the surviving seeds grow directly into youths, the number of young trees increasing each year is a credible indicator of the number of seeds. Fig. 13 shows the curves depicting how the number of young trees increases over 30 years. The horizontal axis is the time step in year. The vertical axis is the number of young trees that grow from new-born seeds. The range of errors was from 0 to 3.1%. The average relative error of youth number was 1.9%. The errors are acceptable and reasonable considering that our algorithm obtains a speed-up ratio of up to 16 compared with the naive GPU algorithm.

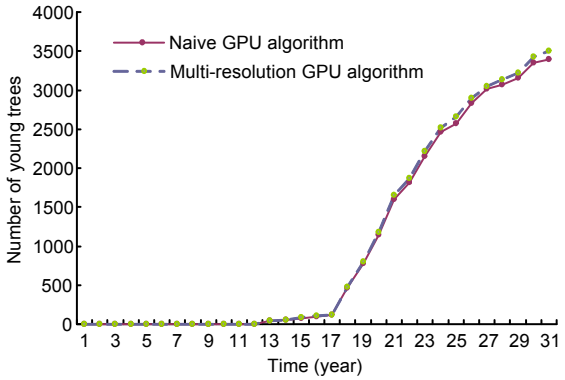


Fig. 13 The number of young trees growing from seeds over 30 years

9 Conclusions

In this paper we propose a GPU-based multi-resolution algorithm to compute seed dispersal in forest dynamics models. We cluster the forest data into different layers according to biology models and design the GPU algorithm to calculate the dispersal. Experimental results show that the algorithm has a good performance compared to naive GPU parallel computation. Our algorithm not only optimizes the utilization of limited thread computing resource, but also greatly improves the speed of calculation with a reasonable error. In the future, we will extend the multi-resolution GPU algorithm to the computation of other growth stages of forest dynamics models that meet the criterion of parallel acceleration.

References

Astrup, R., Coates, D.K., Hall, E., Trowbridge, A., 2007. Documentation for the SOTIE-ND SBS Research Parameter File Version 1.0. Natural Resources Research and Management Report, Bulkley Valley Centre. Available from http://www.bvcentre.ca/files/SORTIE-ND_SBS_

Research_Parameter_File_Version_1.0.pdf [Accessed on May 5, 2012].

Barnes, J., Hut, P., 1986. A hierarchical $O(n\log n)$ force calculation algorithm. *Nature*, **324**(6096):446-449. [doi:10.1038/324446a0]

Bugmann, H., 2001. A review of forest gap models. *Climate Change*, **51**(3/4):259-305. [doi:10.1023/A:1012525626267]

Clark, J.S., Lewis, M., Horvath, L., 2001. Invasion by extremes: population spread with variation in dispersal and reproduction. *Am. Nat.*, **157**(5):537-554. [doi:10.1086/319934]

Du, Z.H., Yin, Z.M., Bader, D.A., 2010. A Tile-Based Parallel Viterbi Algorithm for Biological Sequence Alignment on GPU with CUDA. IEEE Int. Symp. on Parallel & Distributed Processing Workshops and PhD Forum, p.1-8. [doi:10.1109/IPDPSW.2010.5470903]

Gelbard, R., Goldman, O., Spiegler, I., 2007. Investigating diversity of clustering methods: an empirical comparison. *Data. Knowl. Eng.*, **63**(1):155-166. [doi:10.1016/j.datak.2007.01.002]

Govindarajan, S., Dietze, M., Agarwal, P.K., Clark, J., 2004. A Scalable Simulator for Forest Dynamics. Proc. 20th Annual Symp. on Computational Geometry, p.106-115. [doi:10.1145/997817.997836]

Govindarajan, S., Dietze, M.C., Agarwal, P.K., Clark, J.S., 2007. A scalable algorithm for dispersing population. *J. Intell. Inf. Syst.*, **29**(1):39-61. [doi:10.1007/s10844-006-0030-z]

Hamada, T., Titala, I., 2007. The Chamomile Schema: an Optimized Algorithm for *N*-Body Simulations on Programmable Graphics Processing Units. Available from http://arxiv.org/abs/astro-ph/0703100 [Accessed on June 25, 2012].

Hamada, T., Narumi, T., Yokota, R., Yasuola, K., Nitadori, K., Taiji, M., 2009. 42 TFlops Hierarchical *N*-Body Simulations on GPUs with Applications in Both Astrophysics and Turbulence. Proc. Conf. on High Performance Computing Networking, Storage and Analysis, p.14-20. [doi:10.1145/1654059.1654123]

Kunstler, G., Allen, R.B., Coomes, D.A., Canham, C.D., Wright, E.F., 2011. SORTIE/NZ Model Development. Landcare Research New Zealand Ltd. Available from http://www.Landcareresearch.co.nz/publications/researchpubs/sortie_nz_model_dev.pdf [Accessed on May 5, 2012].

Lepage, P.T., Canham, C.D., Coates, K.D., Bartemucci, P., 2000. Seed abundance versus substrate limitation of seedling recruitment in northern temperate forests of British Columbia. *Can. J. Forest Res.*, **30**(3):415-427. [doi:10.1139/x99-223]

Lin, J., Tang, M., Tong, R.F., 2010. GPU accelerated biological sequence alignment. *J. Comput.-Aided Des. Comput. Graph.*, **22**(3):420-427 (in Chinese).

Mielikainen, J., Huang, B., Huang, H.L.A., 2011. GPU-accelerated multi-profile radiative transfer model for the infrared atmospheric sounding Interferometer. *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, **4**(3):691-700. [doi:10.1109/JSTARS.2011.2159195]

Nickolls, J., Buck, I., Garland, M., Skadron, K., 2008. Scalable parallel programming with CUDA. *Queue*, **6**(2):40-53. [doi:10.1145/1365490.1365500]

NVIDIA Corporation, 2007. CUDA Programming Guide, Version 3.0. NVIDIA Corporation. Available from http://developer.nvidia.com/nvidia-gpu-programming-guide [Accessed on May 5, 2012].

Nyland, L., Harris, M., Prins, J., 2007. Fast *N*-Body Simulation with CUDA. *In*: Nguyen, H. (Ed.), GPU Gems 3. Addison-Wesley, London, p.677-795.

Pacala, S.W., Canham, C.D., Silander, J.A.Jr., 1993. Forest models defined by field measurements: I. The design of a northeastern forest simulator. *Can. J. Forest Res.*, **23**(10):1980-1988. [doi:10.1139/x93-249]

Ryoo, S., Rodrigues, C.I., Banghsorkhi, S.S., Stone, S.S., Kirk, D.B., Hwu, W.W., 2008. Optimization Principles and Application Performance Evaluation of a Multithreaded GPU Using CUDA. Proc. 13th ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming, p.73-82. [doi:10.1145/1345206.1345220]

Stone, J.E., Phillips, J.C., Freddolino, P.L., Hardy, D.J., Trabuco, L.G., Schulten, K., 2007. Accelerating molecular modeling applications with graphics processors. *J. Comput. Chem.*, **28**(16):2618-2640. [doi:10.1002/jcc.20829]

Tang, Y., Guan, X.X., Fan, J., 2011. Design and Implementation of Seeds Dispersion on Graphic Processor Unit. Proc. 10th Int. Conf. on Virtual Reality Continuum and Its Applications in Industry, p.403-406. [doi:10.1145/2087756.2087828]

Xia, Y.J., Kuang, L., Li, X.M., 2011. Accelerating geospatial analysis on GPUs using CUDA. *J. Zhejiang Univ.-Sci. C (Comput. & Electron.)*, **12**(12):990-999. [doi:10.1631/jzus.C1100051]

Zhang, S., Chu, Y.L., Zhao, K.Y., Zhang, Y.B., 2009. High Performance GPU Computing of CUDA. China Water Publishing House, Beijing, China, p.155-157 (in Chinese).