Parallel Multiple Observer Siting on Terrain

Wenli Li, W. Randolph Franklin, Daniel N. Benedetti, Salles V. G. Magalhães
Rensselaer Polytechnic Institute, Troy, NY, USA
liw9@rpi.edu, mail@wrfranklin.org, daniel.n.benedetti@gmail.com, vianas2@rpi.edu

ABSTRACT

This paper presents the optimization and parallelization of the multiple observer siting program, originally developed by Franklin and Vogt. Siting is a compute-intensive application with a large amount of inherent parallelism. The advantage of parallelization is not only a faster program but also the ability to solve bigger problems. We have parallelized the program using two different techniques: OpenMP, using multi-core CPUs, and CUDA, using a general purpose graphics processing unit (GPGPU). Experiment results show that both techniques are very effective. Using the OpenMP program, we are able to site tens of thousands of observers on a 16385×16385 terrain in less than 2 minutes, on our workstation with two CPUs and one GPU. The CUDA program achieves the same in about 30 seconds.

Categories and Subject Descriptors

I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling—*geometric algorithms, languages, and systems*

Keywords

Terrain, siting, parallelization

1. INTRODUCTION

The purpose of multiple observer siting [2, 4, 5] is to place a number of observers to cover some targets on or above a terrain represented as a digital elevation map (DEM). Assuming observers and targets are placed only at terrain points, the number of possible positions is the terrain size. The objective is either to cover as many targets as possible using a certain number of observers, or to cover a certain number of targets using as few observers as possible. Usually, an observer covers a target that is visible, or has direct line of sight (LOS) from the observer, within a certain radius of interest. If the targets are all points of the terrain, then the targets visible from an observer constitute the viewshed of the observer. Define the area of the terrain to be the terrain size. The number of targets covered by an observer is the area of the terrain covered by the observer, which is the viewshed area of the observer if the targets are all the terrain points.

SIGSPATIAL '14, November 04 - 07 2014, Dallas/Fort Worth, TX, USA

http://dx.doi.org/10.1145/2666310.2666486

Multiple observer siting is useful in the placement of radio transmission towers, mobile ad hoc networks, environmental monitoring sites, and various other applications.

2. RELATED WORK

In parallel terrain visibility, Llobera et al. [9] proposed a method of computing the viewshed at all points of a terrain, called the inherent viewshed or total viewshed, using a culster of computers. Tabik et al. [18, 19] proposed a parallel horizon algorithm based on the algorithm of Stewart [16] that computes the horizon at all points of a terrain, and an algorithm to compute the total viewshed at all points of a terrain. Lu et al. [10] proposed a multi-core LOS and viewshed algorithm on the Cell Broadband Engine (CBE) processor. Many tried to accelerate LOS or viewshed computations using traditional or general purpose computing on the GPU. Salomon et al. [6, 14, 15, 20] proposed a method to accelerate LOS computations on TINs using the GPU for computer generated forces (CGF). The algorithm combines GPU rendering into depth buffer for conservative visibility culling and CPU ray casting for exact visibility computations. Later, they precompute visibility by dividing the terrain into square regions and computing a region-to-region visibility map for each region. Finally, they use dynamic bounding volume hierarchies (D-BVH) for fast refitting to support moving and deforming objects, and axis aligned bounding boxes (AABB) as bounding volumes for fast LOS computations. Fang et al. [1] proposed an algorithm for real-time computing and rendering of the viewshed using a depth buffer and vertex and pixel shaders. Strnad [17] proposed two implementations of viewshed computation on the GPU, which support multiple viewpoints and the composition of viewsheds.

On multiple observer siting, Lebeck et al. [7] proposed an algorithm for computing highly occluded paths on a terrain with unknown observer positions. They define the path cost as the sum of point visibility along a path and compute the minimum cost path using Dijkstra's algorithm. They compute the approximate viewshed of an observer by simplifying the terrain adaptively and accelerate it using the GPU. To guess observer positions, they use a topology based approach for maxima and saddle points, a coverage based approach for good coverage points, or a combination of the two. Magalhães et al. [11] and Pena et al. [12, 13] proposed a local search heuristic to increase the percentage of a terrain covered by a given number k of observers. Given a set of candidate observers, each subset of k observers is a solution S, and a neighbor of S can be created by replacing an observer in S with one not in S. Starting from an initial solution, the local search heuristic repeatedly replaces the current solution with its largest coverage neighbor, until there is no larger neighbor. In this way, it finds a local maximum solution.

3. MULTIPLE OBSERVER SITING

Franklin and Vogt [5] developed a software package that places observers to maximally cover the area of a terrain. Define the *visibility index* of a point to be the viewshed area of an observer at the point, divided by the total number of points within radius of interest. It first computes an approximate visibility index for each point, and then selects some points with high visibiliby indexes as candidate positions for observers, called *tentative observers*. After that, it computes the viewshed of each tentative observer, and incrementally selects observers from the tentative observers to cover the terrain. As a result, it has four programs that run in sequence: *VIX*, *FINDMAX*, *VIEWSHED* and *SITE*.

VIX computes the approximate visibility index, the value of which is in [0, 255]. The input parameters are: the number of rows and columns of the terrain, *nrows*; the radius of interest, *roi*; the observer and target height above the terrain, *ht*; and the number of random visibility tests for each point, *ntests*. Taking each point as an observer, *VIX* randomly picks *ntests* targets within *roi* and computes their visibility, and calculates the ratio of visible targets times 255 as the approximate visibility index. *FINDMAX* selects a set of tentative observers with high visibility indexes. Points with the highest visibility indexes are often close together. To space tentative observers out over the terrain, *FINDMAX* divides it into square blocks and selects a fixed number of most visible points in each block as tentative observers. The additional input parameters are the number of tentative observers, *nwanted*, and the width of a block, *blocksize*. The program adjusts the values of *nwanted* and *blocksize* so that each block has the same number of tentative observers. *VIEWSHED* computes the viewshed of each tentative observer. The algorithm is based on the R2 program of Franklin and Ray [3]. *SITE* selects a set of observers from the tentative observers to cover the terrain. It uses a greedy algorithm that selects one observer at a time. In each step, *SITE* computes the area of the union of each unused observer viewshed and the cumulative viewshed. Then it finds the unused observer whose viewshed has the largest union area and adds it to the selected observers. Finally, it updates the cumulative viewshed as its union with the added observer viewshed. The program stops when it has reached a specified number of observers or coverage of the terrain (the area of the cumulative viewshed over the size of the terrain), or no more observers can be added to increase the area of the cumulative viewshed.

4. PARALLEL MULTI-OBSERVER SITING

Multiple observer siting is a compute-intensive application with a lot of inherent parallelism. It is possible to reduce the running time greatly by implementing it in parallel, the advantages of which include not only a faster program, but also the abilities to use higher resolution terrains and compute more accurate visibility indexes, and to select more tentative observers. We have used two different techniques to parallelize the program: OpenMP, an API for shared memory parallel programming, using multi-core CPUs, and CUDA, a parallel computing architecture, using the general purpose programming capability of NVIDIA GPUs.

4.1 Improved Sequential Program

Before parallelization, we have made a few improvements to the sequential program. First, we combine the four programs into one to eliminate intermediate I/O's. The resulting program has four functions: *VIX*, *FINDMAX*, *VIEWSHED*, and *SITE*. *VIX* is very time consuming but, since the visibility index is an approximation, we can approximate more. Instead of computing the exact visibility of each random target around an observer, we compute its approximate visibility by evaluating a subset of points along the line of sight, with a distance *stride* between adjacent points. This is similar to volumetric ray marching in Computer Graphics, where the values of equidistant sample points along a viewing ray are evaluated and combined to produce the value of a pixel. $stride = 1$ is exact visibility. How we select *stride* and how it affects the result of siting will be elaborated in section 5.1. To represent observer and the cumulative viewsheds, we use one bit per pixel and pack each row to complete words. If rows are not packed, some operations are easier but boundary detection is harder. A word of an observer viewshed is usually not aligned with a word of the cumulative viewshed. *SITE* is also very time consuming. Two modifications to the algorithm greatly reduce its time complexity. First, the time to compute the union of a viewshed V and the cumulative viewshed C is $O(nrows^2)$. However, to find the top observer, we can just search for the viewshed that adds the largest extra area to the cumulative viewshed. The extra area of V can be computed as the set difference $V - C_V$, where C_V is the corresponding area on the cumulative viewshed. It is implemented as (V or C_V) xor C_V and the time is $O(roi^2)$. Second, not all tentative observers need to have the extra area recomputed in each iteration. The ones that need update are within $2 \times roi$ of the last added observer. If the distance of a tentative observer to the last addition is larger than $2 \times roi$, then its extra area remains the same. The OpenMP adaptation of the program mainly has a few compiler directives more than the sequential program.

4.2 CUDA Program

To work in CUDA, a kernel function has to be defined and executed by a large number of threads on the GPU. It is natural to assign a thread to each point in *VIX* and a kernel function is defined to compute the visibility index of a point. We assign a thread block to each terrain block in *FINDMAX*. At first, we defined a kernel function to sort points by visibility index. Then we found that if we only want a few tentative observers in each terrain block, it is faster to search for them one by one. In *VIEWSHED*, we use a CUDA thread block to compute the viewshed of a tentative observer. A kernel function is defined that computes a sector of a viewshed. In *SITE*, the section that updates extra areas is implemented first. The initial implementation assigns a thread block to an observer and performs poorly due to unbalanced workload. Assigning multiple observers to a thread block improves the situation, but does not solve the problem. To separate the two tasks of the section, finding observers for update and updating them, we implement them in two kernels and launch one kernel from another using dynamic parallelism. Whenever an observer that needs update is found, a kernel is launched with a single thread block to compute its extra area. Then we implement a kernel for the section that searches for the top observer, using loop unrolling for parallel reduction. Finally, we implement a kernel to update the cumulative viewshed. Like the kernel that computes the extra area, it only needs as many threads as the number of rows of a viewshed and is far from having enough parallelism. However, updating

Table 1: Running time of *VIX* (in seconds), coverage (%) and number of observers of the CUDA program, with different values of *stride* and *ntests*

stride	Time	Cov.	Obs.	Time	Cov.	Obs.
	ntests = 20			*ntests* = 50		
1	58.82	95.04	25279	143.16	95.93	25128
2	34.02	95.09	25294	81.99	96.00	25155
4	21.11	95.14	25313	49.08	95.97	25229
8	13.89	95.11	25385	30.89	95.91	25279
2^i	8.16	95.65	25417	16.72	96.42	25282
	ntests = 120			*ntests* = 255		
1	339.64	96.45	25108	722.97	96.79	25008
2	193.16	96.49	25130	410.78	96.81	25047
4	114.39	96.48	25137	241.94	96.80	25070
8	70.76	96.37	25214	148.66	96.71	25129
2^i	36.61	96.84	25239	75.01	97.06	25197

the cumulative viewshed on the CPU and transferring it to the GPU is even slower. As a result, there are three kernel calls in each iteration of *SITE*, but data transfers between the GPU and the CPU are mostly eliminated.

5. EXPERIMENTS

The test machine has dual Intel Xeon E5-2687W CPUs with a total of 16 cores, 128GB of memory, and an NVIDIA Tesla K20Xm GPU accelerator with 14 streaming multiprocessors and 2688 CUDA cores. The operating system is Ubuntu 12.04. The programs are compiled with `g++ 4.6.4` optimization level 2 (`-O2`). The test terrains are 1025×1025, 2049×2049, 4097×4097, 8193×8193 and 16385×16385 Puget Sound terrains downsized from a 16385×16385 Puget Sound terrain of Lindstrom and Pascucci [8], which was extracted from Washington State 10-meter DEMs from the USGS. The unit of value is integer decimeter.

5.1 The Visibility Index

We use the CUDA program to test the effect of *stride* because it is the fastest. We run the program until no more observers can be added to increase the area of the cumulative viewshed, and record the running time of *VIX*, the coverage, and the number of selected observers. We first tested constant values of *stride* from 1 to 16 using different *ntests* and found that *stride* $= 1, 2, 4, 8, 16$ are more effective than other values. We also found the first point along the line of sight very important to the accuracy of visibility. In fact, points nearer to the observer are more important so that evaluation points should be denser towards the observer (the same is true for points nearer to the target). Therefore, we also tested *stride* that grows exponentially: *stride* $= c^i$, where c is a consant and $i = 0, 1, 2, \ldots$ for points $1, 2, 3, \ldots$ from the observer. Table 1 shows the results with *stride* $= 1, 2, 4, 8, 2^i$ and *ntests* $= 20, 50, 120, 255$. The terrain is 16385×16385 and the parameters are the same as in the next subsection. In general, a larger *stride* produces less accurate visibility indexes in less time, and obtains less coverage using more observers. As *ntests* and the accuracy of visibility indexes increase, the coverage increases and the number of observers decreases. We choose *stride* $= 2^i$ and *ntests* $= 30$.

5.2 The Programs

We test the sequential program on the test terrains with parameters fixed at *roi* $= 200$, *ht* $= 20$, *ntests* $= 30$ and *blocksize* $= 100$, and one tentative observer per block. The radius of interest is twice the width of a block, so there is a fair amount of overlap between viewsheds. *roi* and *blocksize* are fixed so that the visible range decreases and the number of tentative observers increases as the terrain resolution increases. The number of terrain blocks and the number of tentative observers is *nwanted* $= 100, 400, 1681, 6724, 26896$ respectively. The program is run until the area of the cumulative viewshed can not be increased. Table 2 shows the running time of the four parts and the total time, as well as coverage and number of selected observers. The total time includes initialization, mostly reading data and creating data structures, which is the same for all the programs. *VIX* and *VIEWSHED* are the most time-consuming parts.

Table 2: Running time of the four parts and total time of the sequential program (in seconds)

roi=200, *ht*=20, *ntests*=30, *blocksize*=100					
nrows	1025	2049	4097	8193	16385
nwanted	100	400	1681	6724	26896
Initialization	0.04	0.15	0.58	2.40	9.24
VIX	2.10	8.67	37.28	163.29	704.63
FINDMAX	0.00	0.02	0.06	0.27	1.11
VIEWSHED	0.56	2.51	11.47	54.35	232.91
SITE	0.03	0.18	0.90	4.64	27.57
Total time	2.73	11.53	50.30	224.98	975.52
Coverage (%)	73.5	81.2	88.5	92.9	96.1
Observers	100	400	1681	6678	25374

Table 3: Running time of the four parts and total time of the OpenMP program (in seconds)

roi=200, *ht*=20, *ntests*=30, *blocksize*=100					
nrows	1025	2049	4097	8193	16385
VIX	0.12	0.41	1.78	8.79	41.03
FINDMAX	0.00	0.00	0.01	0.02	0.08
VIEWSHED	0.03	0.11	0.49	2.19	9.61
SITE	0.04	0.23	0.49	2.65	15.52
Total time	0.23	0.92	3.41	16.12	75.67

We test the OpenMP program with the same tests as the sequential program and Table 3 shows the running time of each part and the total time. It is much faster than the sequential program and *VIX* and *SITE* are the most time-consuming parts. A CUDA kernel is called with an execution configuration that specifies the number of thread blocks, *dimgrid*, and the number of threads per block, *dimblock*. We tested various values as *dimblock* and selected a relatively good execution configuration for each kernel. Table 4 shows the running time of the CUDA program. It is much faster than the sequential program except for the first test. Compared to the OpenMP program, it is slower for the three smaller tests but is faster for the two larger tests. We can see that it requires a reasonable amount of parallelism and a sufficient problem size to be cost-effective. As the OpenMP program, *VIX* and *SITE* are the most time-consuming parts.

Table 4: Running time of the four parts and total time of the CUDA program (in seconds)

roi=200, *ht*=20, *ntests*=30, *blocksize*=100					
nrows	1025	2049	4097	8193	16385
VIX	2.36	2.36	2.59	3.93	11.11
FINDMAX	0.00	0.00	0.01	0.03	0.11
VIEWSHED	0.01	0.04	0.17	0.66	2.84
SITE	0.02	0.08	0.36	1.42	6.08
Total time	2.45	2.67	3.84	8.56	29.61

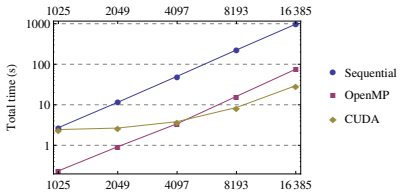

Figure 1: Total running time of the programs.

Table 5: Speedup of the running time of the OpenMP program and the CUDA program

The OpenMP program					
nrows	1025	2049	4097	8193	16385
VIX	17.7	21.0	20.9	18.6	17.2
FINDMAX	2.1	3.3	12.7	11.2	14.1
VIEWSHED	20.8	22.5	23.2	24.8	24.2
SITE	0.8	0.8	1.8	1.7	1.8
Total time	11.7	12.5	14.8	14.0	12.9
Four parts	14.1	14.9	17.9	16.3	14.6
The CUDA program					
nrows	1025	2049	4097	8193	16385
VIX	0.9	3.7	14.4	41.6	63.4
FINDMAX	2.9	5.1	6.6	9.7	10.4
VIEWSHED	46.1	60.3	67.8	82.8	82.0
SITE	1.5	2.1	2.5	3.3	4.5
Total time	1.1	4.3	13.1	26.3	32.9
Four parts	1.1	4.6	15.9	36.9	48.0

5.3 OpenMP and CUDA Program Speedups

Finally, we compare the running time of the programs. Figure 1 shows the line chart of the total time with the vertical axis in logarithmic scale. The lines of the sequential program and the OpenMP program are almost straight, which indicates the exponential growth of time with the terrain width, or linear growth with the terrain size. The line of the CUDA program is nearly flat at first, then rises to the slope of the other lines. Table 5 shows the speedups of the OpenMP program and the CUDA program relative to the sequential program. The OpenMP program has about 11–15 times speedup in total time. Because initialization is not parallelized, we also compare the sum of four part time. The CUDA program has about 1–33 times speedup in total time. Unlike the OpenMP program, the speedup increases with the terrain width, which makes us speculate that higher speedups are possible with larger tests. However, the test size is limited by the memory of the GPU.

6. CONCLUSIONS

We have improved and parallelized the siting program using OpenMP and CUDA. The results show that both techniques are effective ways to accelerate it, which also shows the inherent parallelism of multiple observer siting. The OpenMP program is easy to implement and achieves a speedup of about 13 times with the 16385×16385 terrain. The CUDA program is much more complex to implement and requires properly selected execution configurations, but it achieves a speedup of about 33 times. We expect the CUDA program to achieve higher speedups as the ratio of computation to memory transfer increases, for example, when the radius of interest or the number of tentative observers increases.

Acknowledgement This research was partially supported by NSF grant IIS-1117277 and CAPES (Ciência sem Fronteiras).

7. REFERENCES

[1] C. Fang, C. Yang, Z. Chen, X. Yao, and H. Guo. Parallel algorithm for viewshed analysis on a modern GPU. *Int. J. Digital Earth*, 4(6):471–486, 2011.

[2] W. R. Franklin. Siting observers on terrain. In *ISPRS Commission IV, Symposium 2002, Geospatial Theory, Processing and Applications*, 2002.

[3] W. R. Franklin and C. Ray. Higher isn't necessarily better: Visibility algorithms and experiments. In *Proc. 6th Int. Symp. Spat. Data Handl.*, pp. 751–770, 1994.

[4] W. R. Franklin and C. Vogt. Multiple observer siting on terrain with intervisibility or lo-res data. In *XXth Congress, ISPRS*, pp. 12–23, 2004.

[5] W. R. Franklin and C. Vogt. Tradeoffs when multiple observer siting on large terrain cells. In *Proc. 12th Int. Symp. Spat. Data Handl.*, pp. 845–861, 2006.

[6] C. Lauterbach, M. C. Lin, D. Manocha, S. Borkman, E. LaFave, G. Peele, and M. Bauer. Accelerating line-of-sight computations in large OneSAF terrains with dynamic events. In *Proc. I/ITSEC*, 2008.

[7] N. Lebeck, T. Mølhave, and P. K. Agarwal. Computing highly occluded paths on a terrain. In *Proc. 21st ACM SIGSPATIAL GIS*, pp. 14–23, 2013.

[8] P. Lindstrom and V. Pascucci. Visualization of large terrains made easy. In *Proc. IEEE Visualization*, pp. 363–371, 2001.

[9] M. Llobera, D. Wheatley, J. Steele, S. Cox, and O. Parchment. Calculating the inherent visual structure of a landscape (inherent viewshed) using high-throughput computing. In *Beyond the artefact: Proc. CAA2004*, pp. 146–151, 2004.

[10] L. Lu, B. Paulovicks, M. Perrone, and V. Sheinin. High performance computing of line of sight viewshed. In *Proc. IEEE ICME*, pp. 1–6, 2011.

[11] S. V. G. Magalhães, M. V. A. Andrade, and R. S. Ferreira. Using GPU to accelerate heuristics to site observers in DEM terrains. In *Proc. IADIS Applied Computing*, pp. 127–133, 2011.

[12] G. C. Pena, S. V. G. Magalhães, M. V. A. Andrade, W. R. Franklin, W. Li, D. N. Benedetti, and C. R. Ferreira. An efficient GPU multiple-observer siting method based on sparse-matrix multiplication. Manuscript submitted for publication.

[13] G. C. Pena, M. V. A. Andrade, S. V. G. Magalhães, W. R. Franklin, and C. R. Ferreira. An improved parallel algorithm using GPU for siting observers on terrain. In *Proc. 16th ICEIS*, pp. 367–375, 2014.

[14] B. Salomon, N. Govindaraju, A. Sud, R. Gayle, M. Lin, D. Manocha, B. Butler, M. Bauer, A. Rodriguez, L. Eifert, A. Rubel, and M. Macedonia. Accelerating line of sight computation using graphics processing units. In *Proc. 24th Army Science Conference*, 2004.

[15] B. Salomon, D. Tuft, S. Hanlon, and D. Manocha. Accelerating line-of-sight queries for terrain processing using region based visibility. Technical Report TR06-019, Dep. of Comp. Sci., Univ. of North Carolina - Chapel Hill, 2006.

[16] A. J. Stewart. Fast horizon computation at all points of a terrain with visibility and shading applications. *IEEE Trans. Vis. Comp. Graph.*, 4(1):82–93, 1998.

[17] D. Strnad. Parallel terrain visibility calculation on the graphics processing unit. *Concurr. Comput. : Pract. Exper.*, 23(18):2452–2462, 2011.

[18] S. Tabik, L. Romero, and E. Zapata. High-performance three-horizon composition algorithm for large-scale terrains. *Int. J. Geogr. Inf. Sci.*, 25(4):541–555, 2011.

[19] S. Tabik, E. L. Zapata, and L. F. Romero. Simultaneous computation of total viewshed on large high resolution grids. *Int. J. Geogr. Inf. Sci.*, 27(4):804–814, 2013.

[20] D. Tuft, B. Salomon, S. Hanlon, and D. Manocha. Fast line-of-sight computations in complex environments. Technical Report TR05-025, Dep. of Comp. Sci., Univ. of North Carolina - Chapel Hill, 2005.