Parallel Multiple Observer Siting on Terrain

Wenli Li †  W. Randolph Franklin †  Daniel Benedetti †

ABSTRACT
Multiple observer siting has much inherent parallelism. We have parallelized the siting package using OpenMP and CUDA. The results show that both methods are effective for parallelizing the program. Although the overall speedups of the two methods are similar, which are about 16, the speedups of the constituent parts of the program are quite different.

1. INTRODUCTION
The objective of multiple observer siting is to place a number of observers on a terrain, which together cover as many targets on the terrain as possible. If the targets are all the terrain points, then the targets covered by an observer are the visible terrain points, or the viewshed of the observer. Franklin and Vogt [2] developed a software package that sits a number of observers to cover a terrain. It first computes an approximate visibility index at each point, then finds some points with high visibility indexes as positions of tentative observers. Then it computes the viewshed of each tentative observer and incrementally selects observers that cumulatively cover the most area of the terrain. Correspondingly, it has four programs, VIX, FINDMAX, VIEWSHED and SITE, which execute in order.

VIX computes the approximate visibility index. The parameters to it are the number of rows and columns of the terrain, nrows, the radius of interest, roi, the observer and target height, ht, and the number of tests for each point, nt. For each point, VIX picks nt random targets within roi of it and computes the ratio of the visible targets as its approximate visibility index. FINDMAX finds some tentative observers that are most visible. To prevent the tentative observers from congesting, the terrain is divided into equal-sized blocks and a same number of tentative observers are found in each block. The parameters to FINDMAX are the number of wanted tentative observers, nwanted, and the size length of a block, blocksize. VIEWSHED computes the viewshed of each tentative observer with roi and ht. The algorithm is based on [1]. SITE selects a set of observers that cover the terrain. It incrementally selects the next unused tentative observer whose viewshed adds the most area to the cumulative viewshed, until the area cannot be increased.

2. PARALLEL MULTIPLE OBSERVER SITING
Multiple observer siting is compute-intensive and has much inherent parallelism. Therefore, it is a good candidate for parallelization. Two methods are used to parallelize the package: OpenMP, an API for shared-memory parallel programming, and CUDA, a parallel computing architecture created by NVIDIA. CUDA-enabled GPUs are widely available even in laptops. As related work, Lu et al. [4] proposed a parallel line-of-sight algorithm based on the Cell Broadband Engine processor. Strnad [5] proposed GPU-based implementations of viewshed computation for multiple viewpoints. But before parallelization, changes are made to make the sequential package faster. One is merging the four programs into one to eliminate intermediate file I/O’s. The resulting program still has four parts or functions: VIX, FINDMAX, VIEWSHED, and SITE. Another change is using a slightly different algorithm for SITE. Instead of computing the area of the union of a tentative viewshed with the cumulative viewshed, SITE directly computes the extra area that would be added by it and reduces the computation from $O(nrows^2)$ to $O((2 \times roi + 1)^2)$.

2.1 OpenMP Program
The OpenMP program adds a few compiler directives to the sequential program: a #pragma omp parallel for in VIX before the for loop that computes the approximate visibility index of every point, a #pragma omp parallel for in FINDMAX before the for loop that finds the top points of every block, a #pragma omp parallel for in VIEWSHED before the for loop that computes the viewshed of every tentative observer, and a #pragma omp parallel for in SITE before the for loop that computes the extra area of every unused tentative viewshed.

2.2 CUDA Program
The CUDA adaptation of the program consists of CUDA adaptations of the four parts. Each part has a kernel function that does a fraction of the work, and a large number of threads executing the kernel are created to complete the work. In VIX, a thread computes the approximate visibility index of one point. In FINDMAX, a thread block sorts the points of a terrain block and finds the top points. Each thread sorts a fraction of the points using quicksort, then the thread block merges the results as a parallel merge sort. In VIEWSHED, a thread block computes the viewshed of every tentative observer. In SITE, a thread block computes the extra area of the viewshed of a tentative observer, with each thread processing several rows of the viewshed.

3. EXPERIMENTS
The test machine has dual Intel Xeon E5-2687 CPUs with 16 cores and 32 threads, 128GB of memory, and an NVIDIA Tesla K20x GPU accelerator with 2688 CUDA processing cores and 6GB of memory. The test terrains are $1025 \times 1025, 2049 \times 2049, 4097 \times 4097, 8193 \times 8193$ and $16385 \times 16385$. Puget Sound terrains derived from a $16385 \times 16385$ Puget Sound terrain of Lindstrom and Pascucci [3]. First we test the running times of the sequential program with different terrains, the same roi, ht, nt and blocksize, and one tentative observer per block. The running times are shown in Table 1.

---

1Rensselaer Polytechnic Institute. Email: liw9@rpi.edu, mail@wrfranklin.org, benedd@rpi.edu
2This research was partially supported by NSF grant IIS-1117277.
The total time includes initialization. From the results, VIX is roughly linear to nrows$^2$. FINDMAX is linear to nrows$^2$. VIEWSHED is nearly linear to nwanted. SITE is roughly linear to nwanted$^2$. SITE and VIX are time-consuming but FINDMAX and VIEWSHED take little time. The OpenMP program is tested using the same tests and the running times are shown in Table 2. The OpenMP program is much faster and SITE is the most time-consuming part.

A kernel function is called with a specified execution configuration that defines the number of thread blocks, dimgrid, and the number of threads per block, dimblock. A thread block executes on a multiprocessor and a thread executes on a CUDA core. The NVIDIA Tesla K20x GPU has 14 multiprocessors and 192 CUDA cores per multiprocessor. A multiprocessor executes threads in warps of 32 threads, therefore dimblock is usually a multiple of 32. The execution configuration is selected based on the GPU and the data, and has a great impact on performance. We have tested 32, 64, 128, 256, 512 and 1024 as dimblocks for the four parts and selected 128, 512, 512 and 32 for the four parts respectively, which are good for most test terrains. The running times of the CUDA program are shown in Table 3. The CUDA program is comparable in speed to the OpenMP program and SITE is even more relatively time-consuming.

Finally, we compare the running times of the programs. The relative speedups of the OpenMP program and the CUDA program to the sequential program are shown in Table 4. Although the speedups of the total time are similar between the two programs for larger terrains, the speedups of the four parts are quite different. The CUDA program is much faster in VIX and VIEWSHED, but is much slower in FINDMAX and a little slower in SITE. For FINDMAX, there is a lack of parallelism in sorting, and for SITE, there is an overhead to transfer data back and forth between CPU and GPU. For both parts, the implementation may not be efficient enough. Because SITE is the dominant part of the total time, the CUDA program is not much faster than the OpenMP program.

### 4. CONCLUSIONS

We have parallelized the siting package using OpenMP and CUDA. The results show the effectiveness of both ways. It also demonstrates the inherent parallelism of siting. The OpenMP program is easy to write and achieves a speedup of about 16. The CUDA program is more complex to write and configure, and achieves comparable speedups, but it has more potential. An interesting fact is that two Xeon E5-2687 cost about the same as a Tesla K20x. How much speedup a program can actually achieve, however, depends on the specifications of the available hardware and the input data and parameters.

### 5. REFERENCES


