ERNEST ORLANDO LAWRENCE
BERKELEY NATIONAL LABORATORY

Acceleration of the matrix multiplication of Radiance three phase daylighting simulations with parallel computing on heterogeneous hardware of personal computer

Wangda Zuo
University of Miami

Andrew McNeil
Lawrence Berkeley National Laboratory

Michael Wetter
Lawrence Berkeley National Laboratory

Eleanor S. Lee
Lawrence Berkeley National Laboratory

Windows and Envelope Materials Group
Building Technology and Urban Systems Department
Environmental Energy Technologies Division

April 2013

DISCLAIMER

Acceleration of the matrix multiplication of Radiance three phase daylighting simulations with parallel computing on heterogeneous hardware of personal computer

Wangda Zuo*[+], Andrew McNeil, Michael Wetter, Eleanor S. Lee

Building Technology and Urban Systems Department, Environmental Energy Technologies Division, Lawrence Berkeley National Laboratory, Berkeley, CA 94720, USA

[+]Current Employer: Department of Civil, Architectural and Environmental Engineering, University of Miami, Coral Gables, FL, 33146

Abstract

Building designers are increasingly relying on complex fenestration systems to reduce energy consumed for lighting and HVAC in low energy buildings. Radiance, a lighting simulation program, has been used to conduct daylighting simulations for complex fenestration systems. Depending on the configurations, the simulation can take hours or even days using a personal computer. This paper describes how to accelerate the matrix multiplication portion of a Radiance three-phase daylight simulation by conducting parallel computing on heterogeneous hardware of a personal computer. The algorithm was optimized and the computational part was implemented in parallel using OpenCL. The speed of new approach was evaluated using various daylighting simulation cases on a multicore central processing unit and a graphics processing unit. Based on the measurements and analysis of the time usage for the Radiance daylighting simulation, further speedups can be achieved by using fast I/O devices and storing the data in a binary format.

Keywords: parallel computing, OpenCL, daylighting simulation, multicore central processing unit, graphics processing unit

1. Introduction

Building fenestration systems impact building energy usage through daylighting, solar heating and shading, and natural ventilation. Designing low energy buildings requires a strategic optimization of fenestration systems to balance these impacts. This can only be done by conducting integrated simulation of fenestration systems with whole building energy simulation. Unfortunately, current building energy simulation tools are not able to simulate complex fenestration systems (CFS) that are often used by low energy buildings, such as sculptural shading layers and sunlight redirecting systems. To study the energy impact of innovative fenestration systems for a whole building, it is necessary to use a sophisticated daylight simulation tool capable of simulating CFS, such as Radiance (Larson, 1998), in conjunction with the building energy simulation tool.

Recent enhancements to Radiance enable the simulation of CFS using a three-phase method (McNeil, 2012, Ward, 2011). This method pre-calculates coefficients that relate the luminance of discrete sky regions to resulting illuminance at work plane sensors. Once these coefficients are computed, the work plane illuminance can be calculated for any time, sky condition or fenestration system with relatively fast matrix multiplication. Using the three-phase method, Radiance can pre-calculate the annual daylight illuminance profile for a room which can then be input into building energy simulation tools. The matrix multiplication of the three-phase method takes about a few minutes for a single annual daylighting simulation for a small building on a personal computer. The annual daylighting simulation may take hours if it is applied to a large commercial building. Furthermore, the simulation time can become days or weeks when we optimize the design of CFS by performing parametric studies for different fenestration systems, weather conditions, window sizes and orientations, and

* Corresponding Author: 305-284-5993 (Phone), 305-284-3492 (Fax), w.zuo@miami.edu

building shapes. Considering the limited time available for the CFS design in industrial practice, it is important to further reduce the simulation time of Radiance daylighting simulations.

A common approach to accelerate the simulation is parallel computing. Parallel computing with multi-CPUs on supercomputers is the most commonly used parallel technology in building simulation. The examples are computational fluid dynamics for indoor environment (Mazumdar, 2008, Hasama, 2008), uncertainty and sensitivity decomposition of building energy models (Eisenhower, 2011), and massive building energy simulations (Hopkins, 2011). However, purchasing and maintaining the supercomputers is usually too expensive for small businesses that make up the majority of the building industry. Even with the newest cloud computing, companies still have to pay a fair amount of fee for the computing service. Considering modern personal computers are equipped with parallel computing hardware, such as the multi-core CPU and GPU, it is an economic approach to conduct building simulation in parallel using them. For example, researchers have conducted parallel computing for indoor airflow simulation (Zuo, 2010, Wang, 2011) and solar radiation (Jones, 2012) using the GPU.

Parallel programming languages may work only for specific hardware. For instance, the CUDA language used in the study of Zuo and Chen (2010) was only for the NVIDIA GPU. Since Radiance is publicly released, it is important that it is parallelized using a language supported by various hardware. Thus, this study selected OpenCL that is the first open standard for parallel programming on heterogeneous hardware, including CPUs, GPUs, embedded processors and other processors (http://www.khronos.org/opencl/).

In the following parts of the paper, we will first introduce the three-phase method for daylighting calculation in Radiance and the basic concepts of OpenCL. Then we will show the optimization and implementation of the Radiance daylighting simulation algorithm using OpenCL for parallel computing. Numerical experiments will be presented to evaluate acceleration of simulation on heterogeneous hardware. Detailed analysis on the performance barriers will also be given through the cases studies. At the end, methods to further accelerate the simulation will be discussed.

2. Daylighting calculation in Radiance

The three-phase method enables users to perform annual daylight simulations for complex and/or dynamic fenestration systems (McNeil, 2012, Ward, 2011). It is based on a daylight coefficient method (Tregenza, 1983) that calculates contribution coefficients for discretized sky patches to sensor scenes. The three-phase method splits the daylight simulation into three phases: sky contribution to the exterior of the fenestration, transmission through the fenestration, and window contribution to the sensor scene. A Radiance ray tracing simulation generates luminous energy transfer coefficients relating the directional output of the sky to the directional input of the window and the directional output of the window to the illuminance contributions at the sensors. These coefficients are stored in three independent matrices, one for each phase of the simulation, termed daylight matrix M_D for exterior transfer from the sky to the window, transmission matrix M_T for window transmission, and view matrix M_V for interior transfer from the window to the sensor.

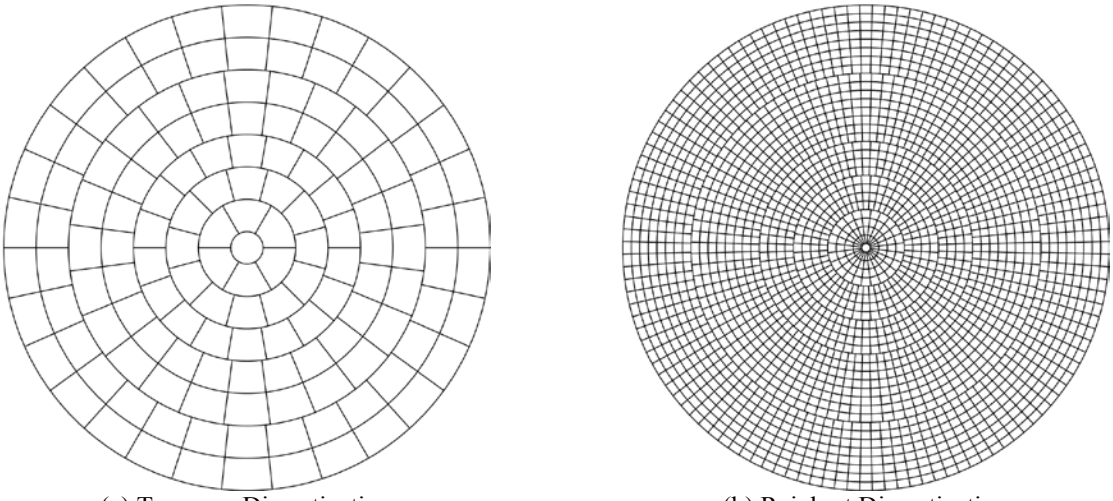
To generate work plane illuminance, users first create a sky vector $V_S(t)$ from diffuse horizontal and direct normal irradiance data using Radiance's *genskyvec* program. As shown in Figure 1, the sky vector has either 146 elements using Tregenza sky discretization (Tregenza, 1987) or 2306 elements using Reinhart sky discretization (Reinhart, 2001, Reinhart, 2011), representing the average luminance of the corresponding regions for a given time, location, sky type and color of light. The simulation using the Reinhart vector can provide more accurate results, but is more time consuming during matrix multiplication stage than that using the Tregenza vector. Then, the sky vector is multiplied by matrices M_D, M_T and M_V to generate an illuminance vector $V_I(t)$ for all of the sensor points as follows:

$$V_R(t) = M_D V_S(t), \tag{1}$$

$$V_C(t) = M_T V_R(t), \tag{2}$$

$$V_I(t) = M_V V_C(t), \tag{3}$$

where $V_R(t)$ and $V_C(t)$ are temporary vectors.



(a) Tregenza Discretization (b) Reinhart Discretization

Figure 1. Sky discretization methods

The matrix multiplications in equations (1) to (3) are performed by a program called *dctimestep* in Radiance (Version 4R0). The algorithm was designed to calculate the daylighting effects for only one time step. To conduct a daylighting simulation for a period of time, we will need another program to invoke *dctimestep* at each time step (Figure 2). Similarly, parametric studies with various daylighting configurations can be performed by invoking the daylighting simulation with different combinations of coefficient matrices and sky vectors.

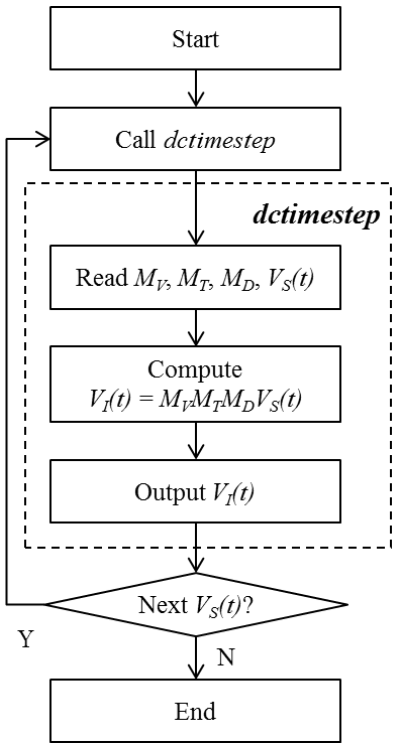


Figure 2. Workflow of the current Radiance daylighting simulation program

3. OpenCL

This study parallelized the daylighting simulation using OpenCL. OpenCL adopts a host-device platform model (Figure 3). The *host* is a commander that connects to one or more *devices*. The device contains one or more *compute units*. The compute unit can be further divided into one or more *processing elements*. The processing element is the basic unit for computing on the device.

In the execution mode of an OpenCL program, a host program runs on the host and one or more kernels runs on the devices. The host program initializes OpenCL supported hardware, creates OpenCL environment, and launches kernels. The parallel computing is conducted through kernels on devices. For more details of OpenCL, see the OpenCL specification (Munshi, 2010).

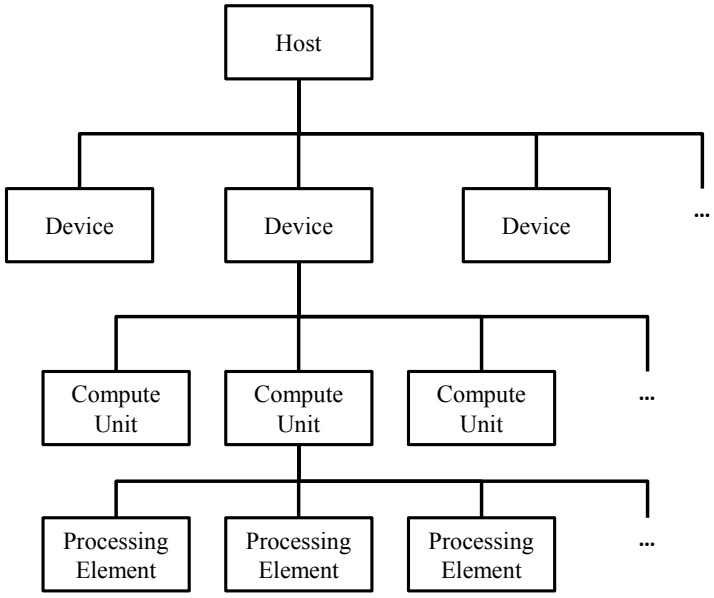


Figure 3. Platform model in OpenCL

4. Implementation

Our original plan was to parallelize the matrix multiplication in equations (1) to (3) without changing the algorithm. However, the numerical experiments showed that this approach slowed down the simulation instead of speeding up it. The reason was that the current algorithm was designed for sequential computing, not parallel computing. As a result, we conducted a rigorous analysis on the current algorithm and proposed a new algorithm that is more efficient and more suitable for parallel computing. The details of analysis and optimization are given in the Appendix.

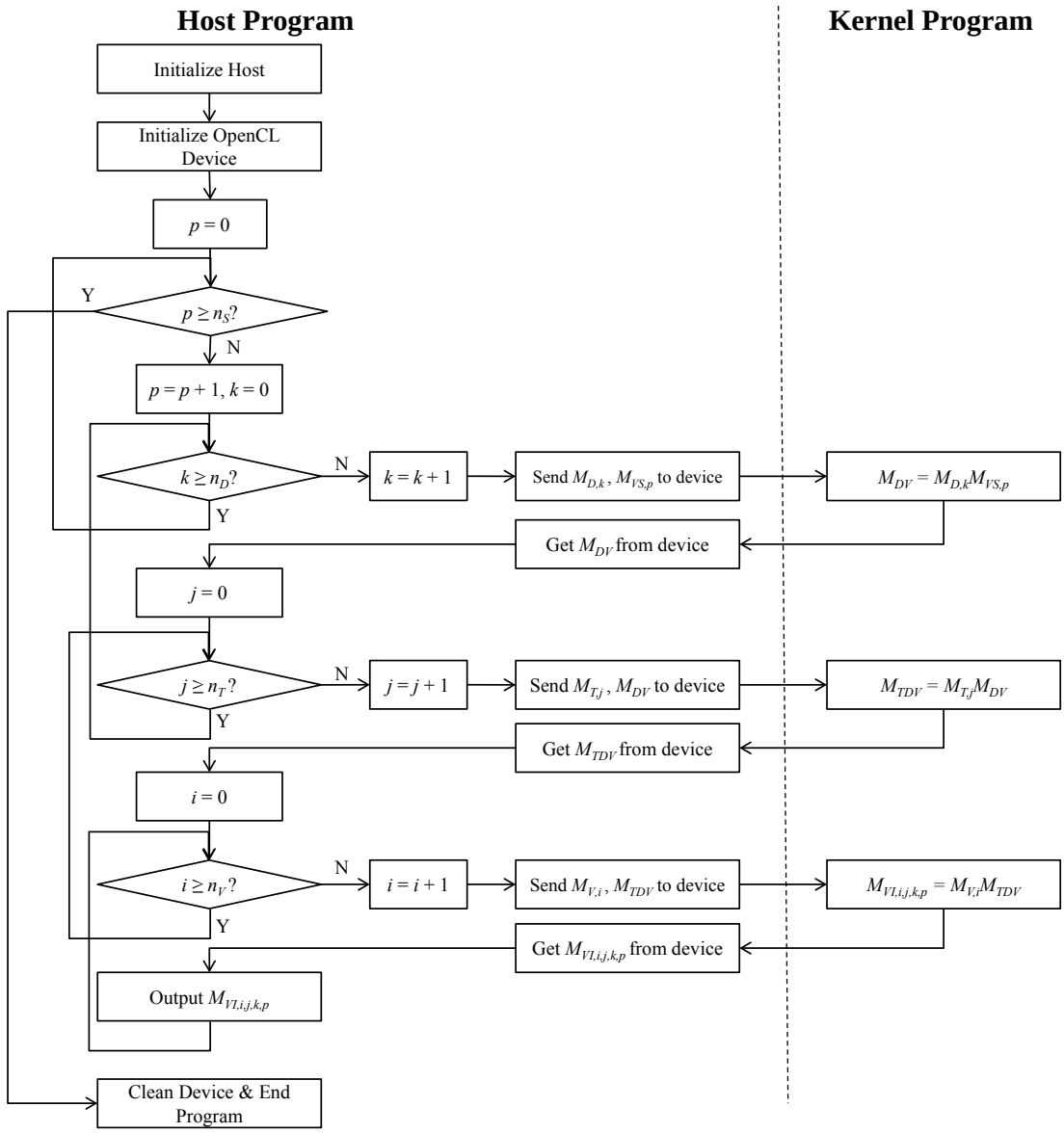


Figure 4. Flow diagram of parallelized Radiance daylighting simulation using OpenCL.

To validate the new algorithm, we first implemented it into a sequential code named *dctime*. After that, we implemented the new algorithm in parallel into a program named *dctime_ocl* using OpenCL 1.1 (Munshi, 2010). Figure 4 shows the flow diagram of *dctime_ocl*. The host program is a C code modified from the *dctime*. Compared to *dctime*, the host program of *dctime_ocl* has additional routines to initialize the OpenCL device, to create an OpenCL environment, to allocate device memory and transfer the data between the host memory and the device memory, and to define input parameters needed by the kernel functions. The matrix multiplication part is implemented in the kernel program. For simplification, current implementation assigns one thread to compute only one element of the matrix. There are approaches to achieve higher performance for matrix calculations on GPUs or multi-core CPUs (Kirk, 2010). But the current approach is the simplest and the results also show that it is sufficient for this application.

5. Case Studies

5.1. Experimental Settings

To evaluate the perforamnce of the new approach proposed in section 4, we measured the performance of three programs, including *dctimestep* of the current Radiance release (Version 4R0), *dctime* and *dctime_ocl* that are the sequential and parallel versions of the new algorihtm, respectively. Both the Tregenza and Reinhart vectors

were used for the same sky condition. The computer used in this study has two hardware supporting OpenCL, an Intel Xeon CPU and a NVIDIA GeForce GTX 460 GPU. The CPU has 4 processors and 6 cores on each processor, which corresponds to 4 OpenCL compute units and 24 processing elements in total. The GPU has 42 streaming multiprocessors and 8 streaming processors on each streaming multiprocessor, which corresponds to 42 compute units and 336 processing elements in total. The clock frequency is 2.67GHz for the CPU and 1350MHz for the GPU. This CPU is used as the host and the device is either the CPU or the GPU.

We denote by "simulation time" the real elapsed time between program invocation and termination. It is the real time measured by *time* command in Linux and includes the time for system overhead, data I/O, and computing. As we will see in the results, the simulation time in this study is significantly different than the "computing time" that is the real elapsed time used for floating point operations.

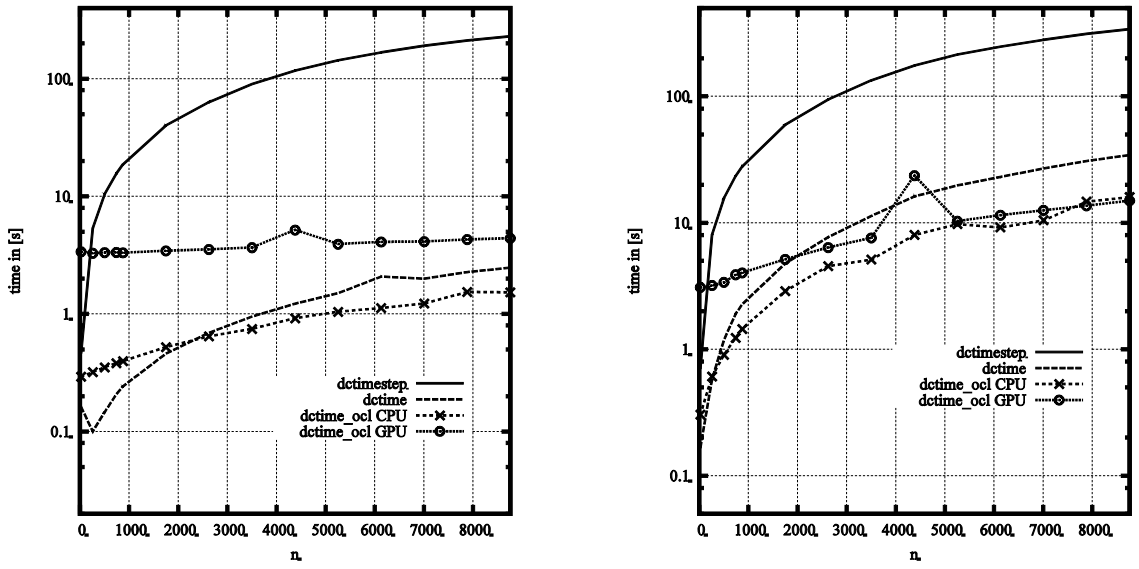
5.2. Simulation of one building for different periods

A common use of daylighting simulation is to simulate daylight for many points in time to understand the aggregate daylighting effect over a period of time. Figure 5 compares the simulation time of the current and new approaches for different simulation time periods. The time step size is one hour and n is the number of time steps. There are preprocesses for all the simulations that exclude the time steps without daylighting (see Appendix A for details). Optimizing the algorithm speeds up the simulation so *dctime* is always faster than *dctimestep*. Parallel computing doesn't always further speed up the simulation and the performance also highly depends on the parallel computing hardware. When the device is CPU, *dctime_ocl* is faster than *dctime* for $n >$ 2500 using the Tregenza vector (Figure 5a) and for $n >$ 500 using the Reinhart vector (Figure 5b). When the device is GPU, *dctime_ocl* is slower than *dctime* for all the n using the Tregenza vector (Figure 5a) and faster than *dctime* for $n >$ 2000 using the Reinhart vector (Figure 5b).

To understand why parallel computing did not always speed up the simulation, we measured the detailed time usages of *dctime_ocl* on both the CPU and GPU. The simulation time t_{sim} is then separated into five parts:

$$t_{sim} = t_{os} + t_{sky} + t_{cl} + t_{com} + t_{res}, \quad (4)$$

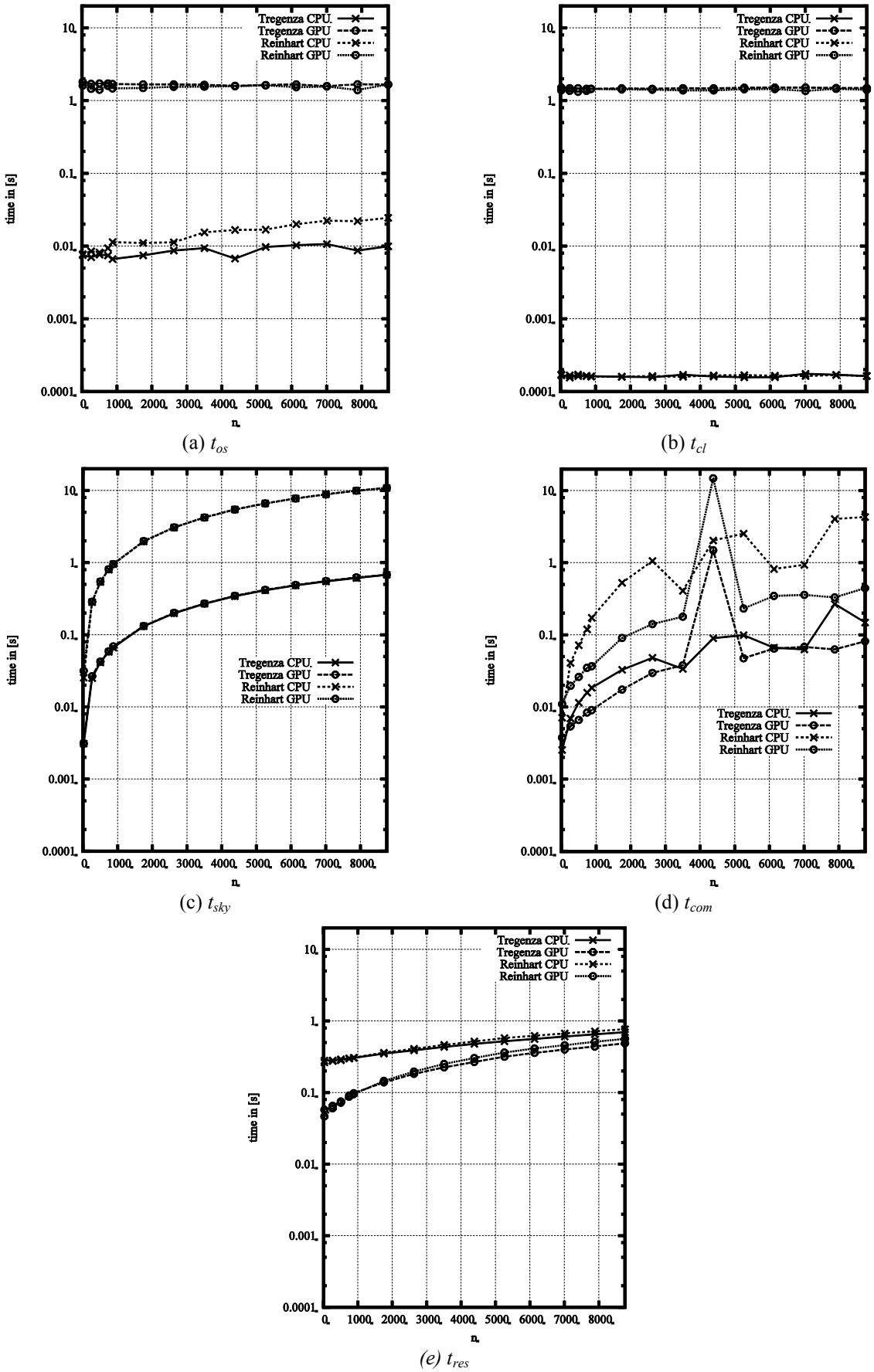
where the variable t_{os} is the time for Linux system to invoke and terminate *dctime_ocl*. The variable t_{sky} is the time to read the sky matrix M_{VS} from hard disk drive to CPU memory. The variable t_{cl} is the time to create an OpenCL context for the device by running an OpenCL function *clcreateContext*( ). Creating the OpenCL context is part of the initialization process for computing on the device. The OpenCL runtime uses the context for managing objects such as command-queues, memory, program and kernel objects and for executing kernels on devices specified in the context. The computing time t_{com} is the time to conduct floating-point operations for matrix multiplications on the device. The variable t_{res} is the time used by the rest parts of the simulation, including reading and writing other matrices, filtering the zero vectors, transferring data between the host and the device, initializing the OpenCL device and the OpenCL programming environment except creating OpenCL context.



(a) Tregenza Vector *(b) Reinhart Vector*

Figure 5. Comparison of the simulation time used by three programs.

As shown in Figure 6(a), the time t_{os} for the operating system to invoke and terminate the OpenCL program varies from about 0.01s for the CPU to about 1s for the GPU. Figure 6(b) compares the time t_{cl} for executing the function *clcreateContext*( ) to create the OpenCL context on the device. It took about 0.001s on the CPU and about 1.5s on the GPU. The t_{os} and t_{cl} can be considered time for overheads. When using the GPU as device with small n, the time for the overhead can be the dominating part of the simulation time. For instance, they accounted for 90% of the simulation time when $n = 24$ using the Tregenza vector. It is worth to mention that the values of t_{os} and t_{cl} were independent of matrix sizes and random for different program runs. Thus, the time reported in this paper was an averaged value of multiple program runs.



(a) t_{os} (b) t_{cl}

(c) t_{sky} (d) t_{com}

(e) t_{res}

Figure 6. Detailed time usages by *dctime_ocl* on the CPU and GPU.

The time t_{sky} for loading the sky matrix M_{VS} depends on the sky discretization methods and the number of time steps (Figure 6c). With finer discretization, the data set of the Reinhart vector is larger than that of the Tregenza vector. Thus, t_{sky} of simulations using the Reinhart vector is larger than that of using the Tregenza vector. The value of t_{sky} increases with n since the dimension of M_{VS} is $N_3 \times n$. As a result, t_{sky} is significant when n is large. For instance, using the Reinhart vector for $n = 8760$, t_{sky} was 10.85s and accounted for about 70% of the simulation time.

Compared to t_{sky}, the computing time t_{com} is less significant (Figure 6d) in this case. For $n = 8760$ with the Reinhart vector, t_{com} was 0.45s using the GPU and 4.33s using the CPU compared to $t_{sky} = 10.85$s. The t_{com} increases with n since the number of floating-point operations increases with n. Sudden increases of t_{com} for GPU and decreases of t_{com} for CPU can be observed in Figure 6d. The changes of t_{com} are repeatable and related to n. They may be caused by the mapping between the data structures and processing elements on the device. Similar phenomenon was also observed in other work (Zuo, 2010).

Figure 6(e) compares the time t_{res} used by other features. It also increases with n because larger n means larger data set to pre-process, transfer and post-process. Compared to other parts of the simulation time, t_{res} is not significant for a single annual simulation since it is less than 1s.

To identify the speedup due to parallel computing alone, Table 1 compares the computing time t_{com} of sequential and parallel program with $n = 8760$ and the speedups due to parallel computing. Parallel computing on the multicore CPU could reduce t_{com} by about 13 times using the Tregenza vector and 5 times using the Reinhart vector. The GPU can provide further speedup of about 24 times using the Tregenza vector and 52 times using the Reinhart vector. However, using GPU as device needs significant time for overheads that is significantly more than the computing time. Thus, the simulation time of parallel program on GPU is longer than that on the CPU in this case.

Table 1. Performance enhancement on matrices multiplication due to parallel computing.

Sky Vector	*dctime*	*dctime_ocl* on CPU	*dctime_ocl* on GPU
	Computing time (t_{com}) in seconds		
Tregenza	1.985	0.149	0.082
Reinhart	23.153	4.326	0.447
	Speedup compared to *dctime*		
Tregenza	1	13	24
Reinhart	1	5	52

5.3. Annual simulations for different buildings

The previous case used different simulation periods for the same building. This case is to evaluate the optimized algorithm and parallel computing on different hardware using 7 different commercial buildings for an hourly simulation of a year. The areas of the buildings were estimated based on the commercial reference buildings developed by the U.S. Department of Energy (Deru, 2011) and the number of sensor points for interior illuminance computation were based on the area of the space (Table 2). The same Reinhart sky vector, daylight matrix and transmission matrix were used for all the cases.

Table 2. Specifications of the view matrices for the 7 studied buildings

Case	Number of sensor points for illuminance computation	Description
1	25	A 10 × 10 feet small office
2	150	A 15 × 10 feet office
3	500	A 25 × 20 feet office
4	2,220	A 15 × 148 feet zone of a medium building
5	7,200	One floor of a small office building
6	17,000	One floor of a medium office building
7	38,400	One floor of a large office building

Figure 7 shows the speedup of the proposed sequential and parallel programs compared to the current Radiance method. By optimizing the algorithm, the new sequential code (*dctime*) can speed up the simulation by 1.4 to 3 times. Parallel computing on the GPU (*dctime_ocl GPU*) had high performances (about 50 times speedup) with small numbers of interior illuminances and low performances (about 4 times speedup) with large numbers of interior illuminances. Due to the low overhead, parallel computing on the multicore CPU provided the best performance and the speedup was from 70 to 110 times.

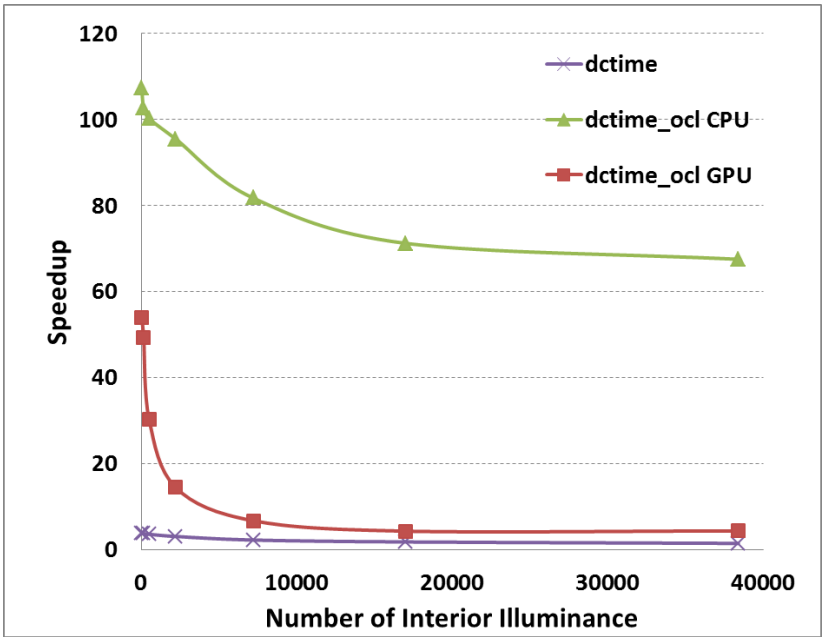


Figure 7. Simulation time of annual daylighting simulation for 7 buildings using different approaches.

To show the impact of accelerating matrix multiplication on the overall annual daylighting simulation time, Table 3 and Table 4 show full simulation time for annual 3-phase simulations similar to what would be required by the IESNA spatial daylight autonomy calculation (Illuminating Engineering Society of North America, 2011). The simulations for case 2 (Table 3) assumed two shading states (deployed and retracted) and a single window orientation so that two annual matrix multiplications are required. The simulations for case 6 (Table 4) assumed four window orientations, two window shade groups per orientation, and two window shade conditions (deployed and retracted). This required 16 annual matrix multiplications. The results clearly show when the number of sensor points increases to encompass a full floor of a medium sized office building, the view matrix generation becomes more time consuming. View matrix generation is a parallel process, so acceleration over multi-core desktop machines is possible and currently implemented.

Table 3. Full 3-phase simulation duration in seconds for case 2 (150 sensor points, 10×15 office)

	Simulation Time [s]			
	dctimestep	*dctime*	*dctime_ocl_cpu*	*dctime_ocl_gpu*
View Matrix	75			
Transmission Matrix (2 shading states)	120 (60×2)			
Daylight Matrix	45			
Matrix Multiplication (2 runs)	664 (332×2)	179 (89.5×2)	6.4 (3.2×2)	13.4 (6.7×2)
Total	904	419	246.4	253.4
Speedup		**2.2×**	**3.7×**	**3.6×**

Table 4. Full 3-phase simulation duration in seconds for case 6 (17,000 sensor points, one floor of a medium building)

	Simulation Time [s]			
	dctimestep	*dctime*	*dctime_ocl_cpu*	*dctime_ocl_gpu*
View Matrix (2 shade groups, computed in parallel)	3230			
Transmission Matrix (2 shading states)	120 (60×2)			
Daylight Matrix (4 orientations)	180 (45×4)			
Matrix Multiplication (16 runs)	11874 (742.1×16)	6752 (422.0×16)	166 (10.4×16)	2782 (173.9×16)
Total	15404	10282	3696	6312
Speedup		**1.5×**	**4.2×**	**2.4×**

5.4. Parametric study for optimization of fenestration systems

The previous two cases are annual simulations for a single condition (climate and fenestration system). This case is to evaluate the performance for multiple annual simulations using parametric studies for CFS design optimization. The design choices included 8 fenestration systems, 4 window sizes, 6 window orientations, and 8 interior building shapes. This corresponded to 1536 simulations with a combination of 8 transmission matrices M_T, 32 view matrices M_V, and 6 daylight matrices M_D. For this case (unlike the previous two cases) we measured performance of parallel program on the same CPU by using different numbers of CPU processors as devices.

To conduct the same parametric study, the current Radiance code needed about 337,827s and 500,792s for the cases with Tregenza and Reinhart vectors, respectively. As shown in the Figure 8, the new approaches could significantly reduce the simulation time and the speedups were from 310 to 800 times for cases using Tregenza vector and from 410 to 720 for Reinhart vectors. The large contribution of the speedup was due to the redesign of the algorithm. Parallel computing can also further reduce the simulation time for this parametric study.

Among the parallel computing approaches, using more processing elements did not necessarily reduce the simulation time (Figure 8) although it did reduce the computing time (Figure 9). The reason is that adding more processing elements will also increase the time for overhead. As a result, the simulation time, that includes both the time for computing and overhead, may increase as the number of processing elements increases. Thus, it is critical to find an optimal number of processing elements to balance the time of computing and overhead for the maximum performance. For this case, using 2 CPU cores required the least simulation time although its computing time was the longest.

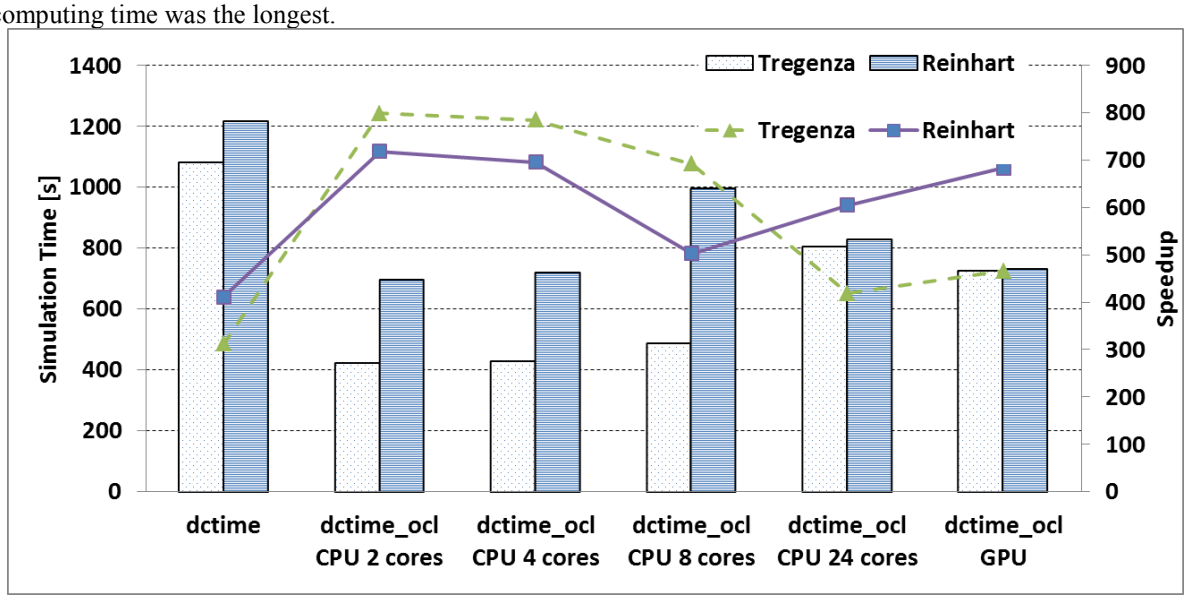


Figure 8. Simulation time of a parametric study using difference approaches. Bars: simulation time; lines: speedup compared to *dctimestep*

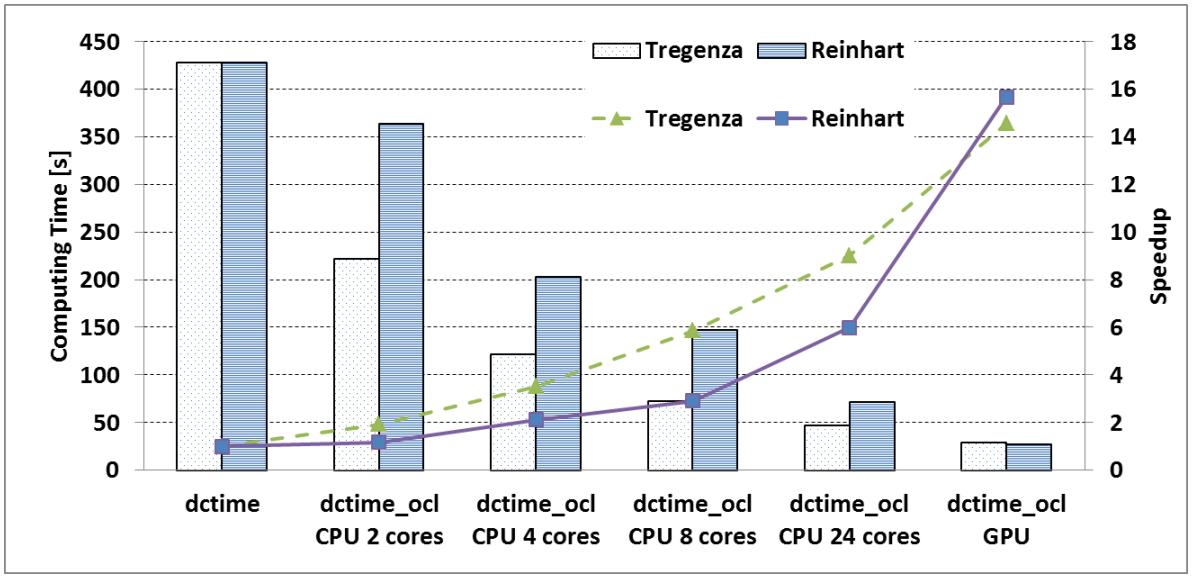


Figure 9. Computing time of a parametric study using difference approaches. Bars: simulation time; lines: speedup compared to *dctime*

6. Discussion

The fact that speedup comes from the optimization of numerical algorithm was unexpected. Originally, we planned to only parallelize the computational part without changing the algorithm. However, it turned out that our first parallelized code was much slower than the existing code. We then did research to understand why the code performed so badly and found that we had to optimize the algorithm first. The design of new algorithm was based a large amount numerical experiments and data analysis although it seems straight forward afterwards. For instance, the large overhead on the operating system and OpenCL program was also unexpected and only found after detailed measurement of time usage for each simulation step. Combining all those finding, we proposed the new algorithm.

The same principle also applies for the further speed up of the simulation. According to the Amdahl's law (Amdahl, 1967), the maximum speedup of a program by using parallel computing is decided by the ratio of sequential parts in the entire simulation. Originally, the data I/O time was not an issue since it was much shorter than the computing time. After we significantly reduced the computing time, the data I/O became a dominant barrier for further speedup. For instance, the parametric study wrote 1536 M_{VT} matrices with about 37.3 GB data to the hard disk drive. Writing this amount of data accounted for about 95% of the simulation time of *dctime_ocl* on the GPU.
To further speed it up, we used more efficient data I/O, such as storing the matrices in binary format instead of ASCII (American National Standards Institute, 1968) and storing them in fast storage devices. Our experiments showed that the simulation time of *dctime_ocl* on GPU for the above parametric study using Reinhart vector was further reduced to about 40s, which made the overall speedup about 12,000 times. However, users may not benefit from this approach if they need to post-process the data in ASCII format. In that case, the data has to be converted from binary to ASCII and the format conversion takes a similar amount of time as writing the data directly in ASCII format.

It is worth to mention that the data I/O issue may also apply for many other building simulation tools, such as EnergyPlus that also stores data in ASCII format. Due to the same reason, storing data in binary format or using fast storage devices is urgently needed in the past. It is likely that the data I/O will become a challenge when a large number of building simulations are performed.

As mentioned in Figure 6(d), there is sudden time increase or decrease in computing time for OpenCL devices. To avoid this, optimization of the data flow for a specific OpenCL device is needed. Furthermore, it is possible to achieve higher performance by optimizing the implementation of OpenCL code for specific hardware to utilize their full capacity (Volko, 2008). However, two issues should be considered. First, the performance enhancement due to optimization highly depends on the nature of the applications. For the parametric study of annual daylighting simulations discussed in this paper, the computing time only account for a small portion

(about 5%) of the simulation time for the *dctime_ocl* if the data is in ASCII format and stored in the hard disk drive. In that case, further optimization for parallel computing is not critical since it can only reduce a small amount of the simulation time. Second, Radiance is widely used software and may run on different hardware. The cost of optimizing the code for various hardware platforms can be too large compared to the benefits.

7. Conclusion

As a proof of concept study, this work is one of the first papers that implementing and evaluating the cross-platform parallel computing technology on different hardware for building simulation. Using the matrix multiplication of Radiance daylighting simulation as an example, we have conducted detailed analysis on this new technology and identified the advantages and limitations. The program we developed was among the first building simulation programs that could run on both multi-core CPU and GPU. Parallel computing with OpenCL on multi-core CPU or GPU could speedup the daylighting simulations for large buildings and parametric studies although the largest speedup came from the algorithm optimization due to the inefficiency of current algorithm. For our application, there was a significant difference between the computing time and simulation time and using multi-core CPU could provide better performance than the GPU since the prior had significantly lower overhead than the latter.

Acknowledgement

This work was supported by the Assistant Secretary for Energy Efficiency and Renewable Energy, Building Technologies Program of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231 and by the California Energy Commission through its Public Interest Energy Research (PIER) Program on behalf of the citizens of California.

Appendix 1: Analysis and optimization of single annual daylighting simulation

The current Radiance *dctimestep* algorithm calculates the daylighting effects using equations (1) to (3) for one time step. Table 1 gives the dimensions of matrices used in the three-phase method. To calculate an element $v_{Ri,j}(t)$ of $V_R(t)$ using equation (1), the number of floating-point operations is $2N_3 - 1$ with N_3 multiplications and $N_3 - 1$ additions. Considering $2N_3 >> 1$, one can approximate that $2N_3 - 1 \approx 2N_3$. Then the number of floating-point operations for calculating N_2 elements of $V_R(t)$ is about $2N_2N_3$. Similarly, the numbers of floating-point operations are about $2N_2^2$ for computing equation (2) and about $2N_1N_2$ for equation (3) since $N_2 >> 1$. As a summation, the number of floating-point operations for computing equations (1) to (3) is about $2N_2$ ($N_1 + N_2 + N_3$) for a single time step and about $2nN_2$ ($N_1 + N_2 + N_3$) for n time steps.

Table 5. Dimensions of matrices and vectors in the Radiance daylighting simulation.

M_V	M_T	M_D	$V_S(t)$
$N_1 \times N_2$	$N_2 \times N_2$	$N_2 \times N_3$	$N_3 \times 1$

N_1 is the number of computed interior illuminances defined by users, $N_2 = 145$, $N_3 = 146$ for Tregenza vector, $N_3 = 2306$ for Reinhart vector

For a single daylighting simulation, matrices M_V, M_T and M_D are constant during the simulation. As shown in Figure 10(a), a better approach is to calculate the product $M_VM_TM_D$ only once at the first time step then reuse it for the rest of the simulation:

$$M_{VTD} = M_VM_TM_D, \tag{5}$$

(6)

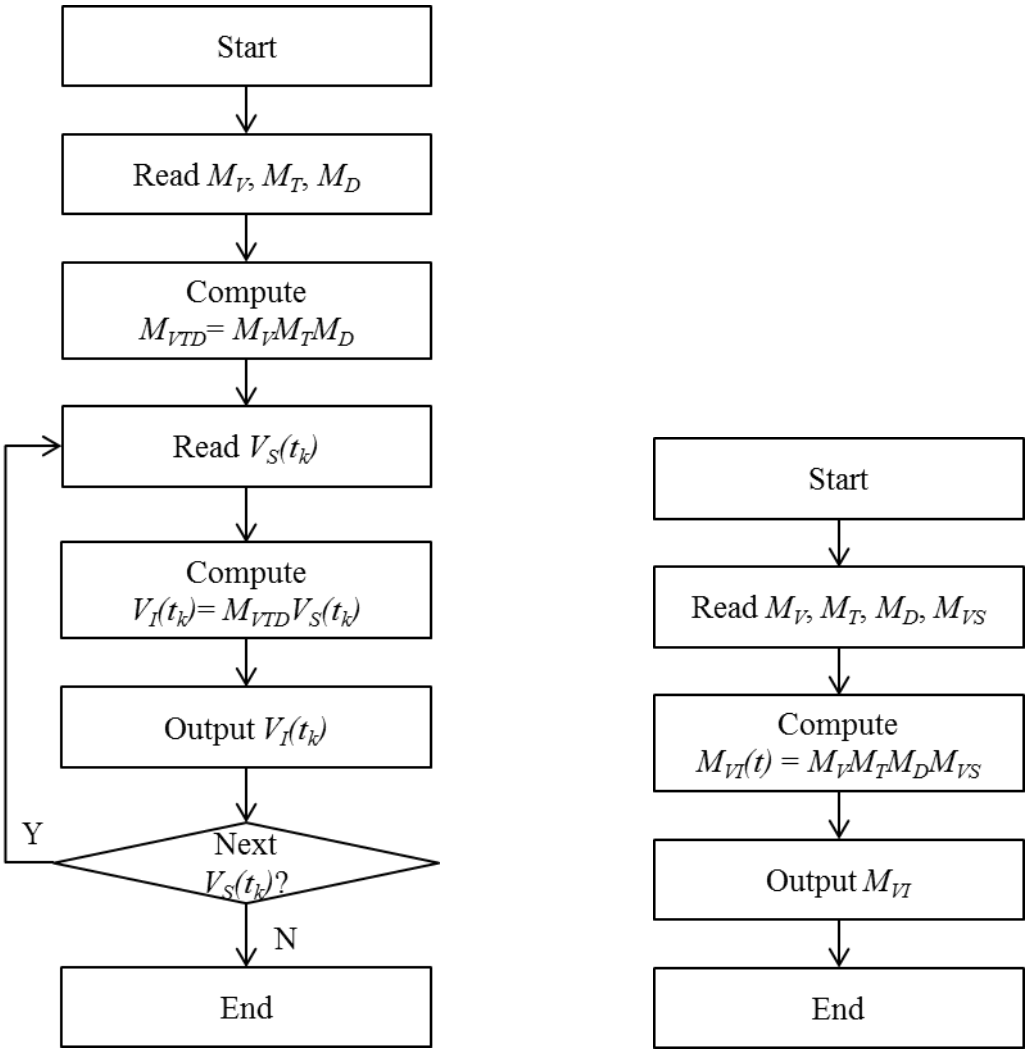


Figure 10. Optimization of the Radiance daylighting simulation algorithm

For n time steps, the number of floating-point operations is about $2N_2^2(N_1 + N_3) + 2nN_1N_3$ with $2N_2^2(N_1 + N_3)$ for computing equation (4) and $2nN_1N_3$ for equation (5). The ratio of number of floating-point operations of the new method described by equations and to that of the current method by equations (1) to (3) is

$$R_{flop} = \frac{N_2^2(N_1 + N_3) + nN_1N_3}{nN_2(N_1 + N_2 + N_3)} \tag{7}$$

For an annual daylighting simulation with a time step of one-hour, n is equal to 8760. Using the matrix dimensions given in Table 1 and assuming $N_1 = 64$, the value of R_{flop} is about 0.19 using the Tregenza vector and about 0.42 using the Reinhart vector. This indicates that the new method can reduce the number of floating-point operations by at least the half.

In addition, the current approach results in redundant data I/O operations when repeatedly calling *dctimestep* during the annual simulation. At each time step, *dctimestep* reads $N_1N_2 + N_2N_2 + N_2N_3 + N_3$ floating-point data for the M_V, M_T, M_D and $V_S(t)$. It also writes N_1 number of floating-point data for the $V_I(t)$. For a simulation with n time steps, the total number of I/O operations is n $(N_1N_2 + N_2N_2 + N_2N_3 + N_1 + N_3)$. As a comparison, the number of I/O operations required by the new approach is $N_1N_2 + N_2N_2 + N_2N_3 + nN_1 + nN_3$ since it only reads M_V, M_T and M_D once. The ratio of number of I/O operations of the new method to that of the current method is

$$R_{IO} = \frac{N_1N_2 + N_2N_2 + N_2N_3 + nN_1 + nN_3}{n(N_1N_2 + N_2N_2 + N_2N_3 + N_1 + N_3)} \tag{8}$$

Assuming n = 8760 and N_1= 64, the value of R_{IO} is about 0.0042 using the Tregenza vector and 0.0065 using the Reinhart vector. With less I/O operations, the new method will need less time for the simulation.

Furthermore, the current approach has to invoke *dctimestep* n times for a simulation with n time steps since it only computes the results for a single time step. It takes time to invoke *dctimestep* by the operating system and to initialize the program for parallel computing. Although they are usually small, the accumulated invocation and initialization time can be large if the program is invoked for millions of times in the parametric study. To reduce the time for program invocations, a better approach is to merge the n sky vectors $V_S(t_k)$, into a single sky matrix $M_{VS} = [V_S(t_1), ..., V_S(t_n)]$ so that the program only needs to be invoked once.

Combining the optimization strategies mentioned above, we proposed an optimized algorithm for the single daylighting simulation over n time steps as follows:

$$M_{VI} = M_V M_T M_D M_{VS}, \tag{9}$$

where $M_{VI} = [V_I(t_1), ..., V_I(t_n)]$. The workflow the optimized algorithm is shown in Figure 10(b).

In addition, the daylighting calculation is not needed when there is no daylighting. Thus, when $V_S(t_k) = 0$, a filtering-inserting procedure can be used to set the corresponding $V_I(t_k) = 0$ without the calculation process. The process is shown in Figure 11. First, all zero sky vectors are removed from M_{VS}, which reduces the $N_3 \times n$ matrix M_{VS} to an $N_3 \times n_1$ matrix M_{VS1} where n_1 is the number of non-zero sky vectors. Then, the equation is computed using M_{VS1} and the result is an $N_1 \times n_1$ matrix M_{VI1}. Finally, the $N_1 \times n$ matrix M_{VI} is generated by inserting zero vectors at corresponding columns in M_{VI1}. This paper applied the filtering method on all programs for all numerical experiments.

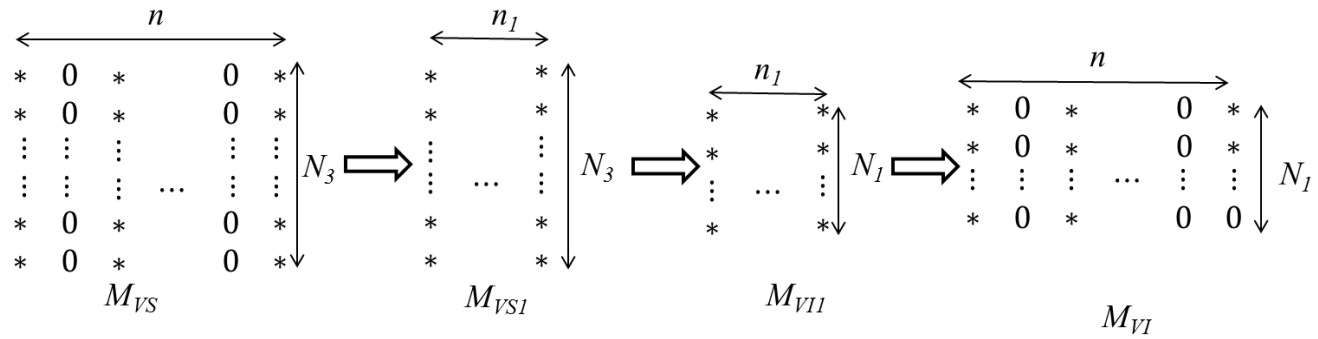


Figure 11. Process of filtering and inserting zero vectors

Appendix 2: Analysis and optimization of multiple annual daylighting simulations

This section discusses how to accelerate a parametric study of multiple annual daylighting simulations. Based on equation (8), the multiple annual simulations can be defined as

$$M_{VI,i,j,k,p} = M_{V,i}M_{T,j}M_{D,k}M_{VS,p,}, \quad (10)$$

where $i = 1, ..., n_V$, $j = 1, \ldots, n_T$, $k = 1, \ldots, n_D$, and $p = 1, \ldots, n_S$. To perform multiple annual daylighting simulations in the parametric study, the current approach is to conduct a single annual daylighting simulation for each different combination of the coefficient and sky matrices. If the number of matrices M_V, M_T, M_D and M_{VS} are n_V, n_T, n_D, and n_S, then the current approach needs $n_V n_D n_T n_S$ single annual daylighting simulations.

To reduce the data I/O operations and floating-point operations, we proposed a new algorithm as follows:

Loop p=1 to n_S
    Loop k=1 to n_D
        $M_{DV} = M_{D,k}M_{VS,p}$
        Loop j=1 to n_T
            $M_{TDV} = M_{T,j}M_{DV}$
            Loop i=1 to n_V
                $M_{VI,i,j,k,p} = M_{V,i}M_{TDV}$
            End Loop i
        End Loop j
    End Loop k
End loop p

Figure 12. Pseudo code of the new approach for parametric studies.

When $n_V = n_D = n_T = n_S = 1$, equation is equivalent to equation. Thus, the above algorithm is also valid for the single daylighting simulation. As shown in Table 6, the current approach loads the same matrices for multiple times during the parametric study and the new approach only loads the matrices once. The difference in matrices loading between the current and the new method increases with the number of cases. The new algorithm can also reduce the number of matrix calculations by reusing the calculated results. For instance, the current approach calculates the M_{DV} for $n_V n_D n_T n_S$ times and the new approach only calculates it for $n_S n_D$ times.

Table 6. Comparison of numbers of loading operations for matrices in the current and new approaches.

Matrices	$M_{V,i}$	$M_{T,j}$	$M_{D,k}$	$M_{VS,p}$
Number of matrix loading by current approach	$n_D n_T n_S$	$n_V n_D n_S$	$n_V n_T n_S$	$n_V n_D n_T$
Number of matrix loading by new approach	1	1	1	1

References

Amdahl, G.M. (1967) "Validity of the single processor approach to achieving large scale computing capabilities". In: Proceedings of AFIPS spring joint computer conference, Vol. 30, pp. 483-385.

American National Standards Institute (1968) American National Standard Code for Information Interchange (ASCII), *Standard No. X3.4*.

Deru, M., Field, K., Studer, D., Benne, K., Griffith, B., Torcellini, P., Liu, B., Halverson, M., Winiarski, D., Rosenberg, M., Yazdanian, M., Huang, J. and Crawley, D. (2011) U.S. Department of Energy Commercial Reference Building Models of the National Building Stock.

Eisenhower, B., O'neill, Z., Fonoberov, V.A. and Mezic´, I. (2011) "Uncertainty and Sensitivity Decomposition of Building Energy Models", *Journal of Building Performance Simulation*.

Hasama, T., Kato, S. and Ooka, R. (2008) "Analysis of wind-induced inflow and outflow through a single opening using LES & DES", *Journal of Wind Engineering and Industrial Aerodynamics*, **96**, 1678-1691.

Hopkins, A.S., Lekov, A., Lutz, J., Rosenquist, G. and Gu, L. (2011) Simulating a Nationally Representative Housing Sample Using EnergyPlus.

Illuminating Engineering Society of North America (2011) *The Light Handbook,* New York, Illuminating Engineering Society of North America.

Jones, N.L., Greenberg, D.P. and Pratt, K.B. (2012) "Fast computer graphics techniques for calculating direct solar radiation on complex building surfaces", *Journal of Building Performance Simulation*, **5**, 300-312.

Kirk, D.B. and Hwu, W.-M.W. (2010) *Programming Massively Parallel Processors: A Hands-on Approach (Applications of GPU Computing Series)*, Morgan Kaufmann.

Larson, G.W. and Shakespeare, R.A. (1998) *Rendering With Radiance: The Art And Science Of Lighting Visualization*, Morgan Kaufmann Publishers.

Mazumdar, S. and Chen, Q. (2008) "Influence of cabin conditions on placement and response of contaminant detection sensors in a commercial aircraft", *Journal of Environmental Monitoring*, **10**, 71-81.

Mcneil, A. and Lee, E.S. (2012) "A validation of the Radiance three-phase simulation method for modelling annual daylight performance of optically complex fenestration systems", *Journal of Building Performance Simulation*.

Munshi, A. (2010) The OpenCL Specification Version 1.1, Khronos OpenCL Working Group.

Reinhart, C.F. (2001) Daylight availability and manual lighting control in office buildings: simulation studies and analysis of measurement. Ph.D., Technical University of Karlsruhe, Department of Architecture.

Reinhart, C.F. and Wienold, J. (2011) "The daylighting dashboard - A simulation-based design analysis for daylit spaces", *Building and Environment*, **46**, 386-396.

Tregenza, P.R. (1983) "Daylight coefficients", *Lighting Research and Technology*, **15**, 65-71.

Tregenza, P.R. (1987) "Subdivision of the sky hemisphere for luminance measurements", *Lighting Research & Technology*, **19**, 13-14.

Volko, V. and Demmel, J.W. (2008) "Benchmarking GPUs to Tune Dense Linear Algebra". In: Proceedings of 2008 ACM/IEEE Conf. on Superconducting (SC '08), Vol. 31, pp. 1-11.

Wang, Y., Malkawi, A. and Yi, Y. (2011) "Implementing CFD (Computational Fluid Dynamics) in OpenCL for Building Simulation". In: Proceedings of The 12th International Building Performance Simulation (Building Simulation 2011).

Ward, G., Mistrick, R., Lee, E.S., Mcneil, A. and Jonsson, J. (2011) "Simulating the Daylight Performance of Complex Fenestration Systems Using Bidirectional Scattering Distribution Functions within Radiance", *Leukos*, **7**, 241-261.

Zuo, W. and Chen, Q. (2010) "Fast and informative flow simulations in a building by using fast fluid dynamics model on graphics processing unit", *Building and Environment*, **45**, 747-757.