Virtual prototyping of complex optical systems on multiprocessor workstations

Andrey Zhdanov,* **Dmitry Zhdanov**, **Maksim Sorokin**, and **Igor Potemin**
ITMO University, St. Petersburg, Russia

Abstract. Nowadays, optical designers often use multiprocessor workstations for the virtual prototyping of complex optical systems. Modern workstations may have several CPUs with up to 128 virtual cores each and a non-uniform access speed to different memory areas. Effective implementation of virtual prototyping methods and algorithms requires the development of special methods for efficient algorithm parallelization and shared memory access. As a basis for the virtual prototyping, we proposed a progressive backward photon mapping method that allows for reducing the amount of data used by photon maps, speeding up the luminance calculation process, and estimating the luminance errors for the resulting image. The main algorithmic complexity of this method is the need to synchronize data when calculating and accumulating the luminance of indirect and caustic illumination. The authors propose the three-level semi-synchronous parallelization method, which consists of fully synchronous, semi-synchronous, and asynchronous levels with illumination processing effectively distributed among the computation threads. The main benefit of the developed method is that it does not require additional synchronization when accumulating luminance, thus increasing the image synthesis speed. The designed three-level method can also be used in distributed systems with good scalability. The results obtained with virtual prototypes of complex optical systems are presented. [DOI: 10.1117/1.OE.62.2.021006]

Keywords: virtual prototyping; ray tracing; parallel computing; realistic rendering; backward photon mapping; multiprocessor workstations.

Paper 20220840SS received Jul. 27, 2022; accepted for publication Nov. 16, 2022; published online Dec. 7, 2022.

1 Introduction

Virtual prototyping techniques are growing increasingly in demand as new manufacturing technologies are developed. When attempting to solve a wide variety of practical issues, such as the creation of photorealistic images, the modeling and design of optical effects, the virtual prototyping of intricate optical systems, etc., the task of physically accurate light propagation modeling and luminance calculation is becoming more in demand. By incorporating virtual prototyping into the production process, it is possible to speed up research and development of the final product and, as a result, increase its competitiveness in the consumer market. Real prototyping of systems, such as virtual, augmented, or mixed reality systems, is becoming increasingly resource-intensive. There is demand for complicated optical device virtual prototyping in a variety of fields, including:

1. Examining how virtual or augmented reality technologies affect how people experience them in the real and virtual worlds. The comfort of visual perception is a focus of research in virtual and augmented reality systems, which include both personal wearable gadgets and systems of head-mounted displays or head-up displays on a windshield.[1]
2. Investigating the viability of deploying sophisticated optical systems in virtual environments that mimic real-world environments. Such gadgets might be either a system created

*Address all correspondence to Andrey Zhdanov, adzhdanov@itmo.ru

for use in a particular area or consumer optical gadgets, such as 3D scanning systems or photography lenses. Real-world prototype manufacturing for intricate optical devices is frequently an expensive and time-consuming procedure. Additionally, when the prototype has been tested, it can be discovered that the original design needs to be changed, necessitating the time- and money-consuming process of creating a new prototype. It should be noted that real-world testing can occasionally be challenging, particularly if the proposed equipment needs a particular setting, such as deep space, deep water, or an open area.

The physical accuracy of the synthesized picture is a crucial prerequisite for an approach being employed for the outcomes from the virtual prototype of an identical optical system to match those from the actual physical prototype. Realistic rendering, which is an integral part of modern realistic visualization, is one of the methods widely used for virtual prototyping of complex optical systems. However, not every rendering method is appropriate for virtual prototyping needs.

To be used in a virtual prototype, the visualization method should not only visualize it but also calculate the correct scene objects luminance at each point of the image; this is a challenge that can be solved by realistic rendering. The ray-tracing method-based algorithms are the primary ones employed when developing realistic rendering for the purposes of virtual prototyping of complex optical systems. The modern realistic rendering methods are mainly based on James Kajia's formula for calculation of the visible luminance.[2] This was called the rendering equation and is used to compute the luminance at point $\vec{p}$ in the direction of observation $\vec{v}_r$ as

$$L(\vec{p},\vec{v}_r)=\tau(\vec{p})\frac{n_r^2}{n_0^2}(L_0(\vec{p},\vec{v}_r)+\int_\omega L_i(\vec{p},\vec{v}_i)f(\vec{p},\vec{v}_i,\vec{v}_r)\cos(\vec{n},\vec{v}_i)\mathrm{d}\omega). \tag{1}$$

$L_0(\vec{p},\vec{v}_r)$ in Eq. (1) denotes a surface's inherent luminance at point $\vec{p}$; $\tau(\vec{p})$ is the medium transmission factor along the path from the observer to the surface being viewed; n_r is the medium refractive index at the point of observation; n_0 is the medium's refractive index on the observer; $L_i(\vec{p},\vec{v}_i)$ is the luminance incident on the surface in direction $\vec{v}_i$; $f(\vec{p},\vec{v}_i,\vec{v}_r)$ is the bidirectional scattering distribution function (BSDF) value for the direction of the incident light $\vec{v}_i$ and the direction of observation $\vec{v}_r$4; and $\vec{n}$ is the surface normal at the point of incidence.

Meanwhile, in 1996, Jensen[3] presented an approach to compute global illumination using photon maps. Three basic phases make up his photon mapping rendering algorithm. The process begins with light sources emitting rays that scatter photons over the scene's surfaces. Next, photon maps are created from the distribution of the photons, and corresponding K-dimensional tree (KD-tree) acceleration structures are created. Finally, the observer gathers the luminance distribution from the photons seen with the camera by backward ray tracing. The photon from the light source has flux $\Delta\Phi(\vec{v}_i)$, and if it passes through the integrating sphere with radius r, which is the field of view of a backward ray, its flux is changed to luminance[4] by the following equation:

$$L_{\mathrm{idc}}(\vec{p},\vec{v}_r)\approx\frac{1}{\pi r^2}\sum_{i=1}^{K}f(\vec{p},\vec{v}_i,\vec{v}_r)\Delta\Phi(\vec{v}_i). \tag{2}$$

The multiple importance sampling method[5] is used to estimate the luminance of a direct illumination with N samples:

$$L_d(\vec{p},\vec{v}_r)\approx\frac{1}{N}\sum_{i=1}^{N}\left(w_{i,1}\cdot L(\vec{p},\vec{v}_i)+w_{i,2}\cdot f(\vec{p},\vec{v}_i,\vec{v}_r)\frac{I(l_i)\cos(\vec{n}_i,\vec{v}_i)}{d^2}\right). \tag{3}$$

The photon mapping method is precise and allows for the independent calculation of the various illumination components, such as direct light visibility, direct illumination, indirect illumination, and caustic illumination. This feature is very helpful for virtual prototyping needs because it allows for the independent evaluation of the various illumination components, such as for stray light analysis.[6]

2 Backward Photon Mapping Rendering Method

For the purpose of virtual prototyping of complex optical systems, we employ the rendering method based on bidirectional stochastic ray tracing with backward photon mapping.[7] When backward photon maps are created, the photon map's physical significance is altered. The light flux that was transmitted from the light sources was kept in the forward photon map. Conversely, the backward photon map saves a filter on the light path from the camera to the matching luminance accumulation point. In terms of the methodology, calculations resembling those used for forward photon mapping in Eqs. (2) and (3) are employed to compute the luminance of the direct illumination:

$$L_{\mathrm{idc}}(\vec{p},\vec{v}_r) \approx \frac{\tau(\vec{p})}{\pi r^2}\sum_{i=1}^{K} f(\vec{p},\vec{v}_i,\vec{v}_r)\Delta\Phi(\vec{v}_i), \tag{4}$$

$$L_d(\vec{p},\vec{v}_r) \approx \frac{\tau(\vec{p})}{N}\sum_{i=1}^{N}\left(w_{i,1}\cdot L(\vec{p},\vec{v}_i) + w_{i,2}\cdot f(\vec{p},\vec{v}_i,\vec{v}_r)\frac{I(l_i)\cos(\vec{n}_i,\vec{v}_i)}{d^2}\right). \tag{5}$$

The transmission factor of the medium, which contains the compensatory component of the BSDF irreversibility on the backward ray path, is the primary distinction of Eqs. (4) and (5).

We utilized the mean-square error across the entire image using the following formula to measure the accuracy of the computations carried out using the progressive backward photon mapping method after the completion of N calculation phases:

$$\delta_N = \sqrt{\frac{\sum_{i=1}^{w\cdot h}\mathrm{SEM}_{i,N}^2}{\sum_{i=1}^{w\cdot h}L_i^2}}, \tag{6}$$

where the image's width and height are w and h, respectively; L_i is the total luminance at the image's i'th pixel; and $\mathrm{SEM}_{i,N}^2$ is the standard error mean at pixel after execution of N calculation phases, which is calculated using the following formula:

$$\mathrm{SEM}_{i,N}^2 = \frac{1}{N}\left(\frac{\sum_{j=1}^{N}L_{i,j}^2}{N} - \left(\frac{\sum_{j=1}^{N}L_{i,j}}{N}\right)^2\right), \tag{7}$$

where $L_{i,j}$ is the luminance accumulated at the image's i'th pixel during phase j and N is the number of previously completed computation phases. For estimating accuracy, this approach does not need to store any intermediate rendering results; instead, it simply has to collect values for the sums $L_{i,j}^2$ and $L_{i,j}$ after finishing each computation phase.

These concepts are applied in our implementation of the backward photon mapping-based rendering method for the accuracy estimation. Because the method accumulates luminance over separate phases, it can be implemented in a progressive manner. Figure 1 shows the flow chart

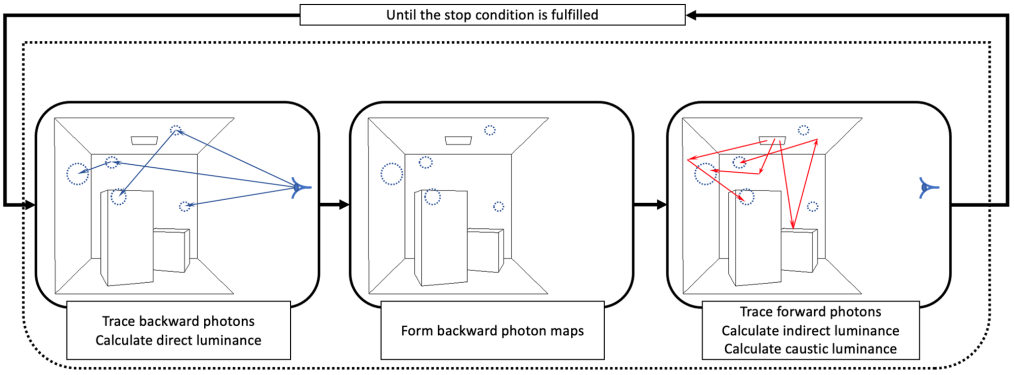

Fig. 1 The bidirectional ray tracing with a progressive backward photon mapping flowchart diagram.

diagram for the progressive backward photon mapping rendering process. It comprises the following four steps in each of its recurring phases:

1. The backward ray tracing with the direct illumination and direct light source visibility components calculation. The backward photons are created by storing the traced ray's endpoints.
2. The creation of a backward photon map using the information obtained from the backward ray tracing. For each scene geometry item, distinct backward photon maps are built, and associated KD-tree acceleration structures are constructed.
3. The forward ray tracing with the calculation of caustic illumination, indirect illumination, and direct light source visibility components.
4. The formation of the final image and accuracy estimation. If the achieved accuracy level is insufficient, the computation process goes back to step 1.

To be able to estimate the complexity of the bidirectional ray tracing with the progressive backward photon mapping algorithm, first, we need to introduce some parameters to define the basic complexity of the input data. Let w and h be the image width and height, respectively, and O_S be the ray-tracing complexity of scene S. The scene complexity estimation depends on the scene geometry and geometry intersection optimization, surface and media properties, light sources and their types, and other scene parameters. Then each of the rendering steps has the following complexity:

1. The backward ray-tracing step traces rays from the camera and intersects them with the scene geometry. Its algorithmic complexity is roughly estimated as $O(w \cdot h \cdot O_S)$. As a result, $O(w \cdot h)$ backward photons are created to be intersected with forward rays in the next steps.
2. The creation of a backward photon map algorithmic complexity depends on the number of backward photons created in the previous step, so its algorithmic complexity is estimated as $O(w \cdot h \cdot \log(w \cdot h))$. The created acceleration structure requires $O(w \cdot h)$ additional memory.
3. The forward ray-tracing step traces rays from the light sources and intersects them with the backward photon maps. Because the total number of forward rays traced at this step, in general, depends on the image resolution only, $O(w \cdot h)$ rays should be traced and intersected both with scene objects and backward photon maps. For scenes containing virtual prototypes of real optical systems, in general, the scene complexity is of a higher order than the complexity of photon maps. So, the total algorithmic complexity of this step is $O(w \cdot h \cdot O_S)$. All data is accumulated in the backward photon maps, so no additional memory is required.
4. The formation of the final image algorithmic complexity depends only on the image resolution and is equal to $O(w \cdot h)$.

As a result, the total algorithmic complexity of the single calculations phase is roughly estimated as $O(w \cdot h \cdot O_S)$ with memory consumption $O(w \cdot h)$. Then, the algorithmic complexity of p phases equals $O(p \cdot w \cdot h \cdot O_S)$. The algorithm's total memory consumption does not depend on the number of phases and is equal to $O(w \cdot h)$.

It should be emphasized that these estimations are very rough. In general, the complexity of the bidirectional ray tracing with a progressive backward photon mapping algorithm is more complex, e.g., the number of rays traced at each calculation phase depends on the data accumulated in previous phases and cannot be estimated in advance. However, this should allow us to understand the complexity of the jobs designated to each parallel thread when parallelizing the algorithm.

The progressive backward photon mapping method creates an image that is made up of numerous layers, which are necessary to visualize various components separately and to resume computations from the last saved state if the optical system designer feels that the achieved accuracy is insufficient. These layers are utilized to keep the following parameters for each pixel of the rendered image:

1. the accumulated luminance of directly visible light sources illumination;
2. the accumulated luminance of direct illumination;
3. the accumulated luminance of indirect illumination;
4. the accumulated luminance of caustic illumination;
5. the number of backward rays that were traced through a corresponding image pixel;
6. the number of samples per ray made to estimate the luminance of direct illumination;
7. the sum of the products of forward and backward traced ray counts;
8. the sum of squares of luminances of accumulated directly visible light sources illumination;
9. the sum of squares of luminances of accumulated direct illumination;
10. the sum of squares of luminances of accumulated indirect illumination;
11. the sum of squares of luminances of accumulated caustic illumination;
12. the extra information needed to build the image's statistics.

The type sizes of each of these values can range from float and int to double and INT64, depending on the type of image being rendered in the virtual prototype. The intermediate image data may be stored with smaller size types, whereas the final accumulated data requires a higher data range because the number of traced rays and accuracy values increase over time. Thus, it takes around 512 MB to store a single RGB image with a size of 1920×1080 pixels. This amount increases to 1 GB when data are stored in double and INT64. If a spectral color model is used in the virtual prototype, then an order of magnitude more data may need to be saved just to store the luminances of one layer of one image pixel. In general, it takes about 4 GB for an intermediate image and 8GB for the final one to keep an image with a resolution of 1920×1080 pixels because spectral visualization is often carried out at 30 wavelengths.

A multiple of the image resolution also applies to the size of the backward photon maps, which are created at each backward ray-tracing step. Studies have shown that increasing the depth of the diffuse ray trace does not enhance the quality of the final image,[8] so the depth of a backward diffuse ray trace is restricted by the second diffuse scattering event to reduce the amount of memory needed to store the backward photon maps. However, when rendering with a single compute thread, the backward photon map at HD resolution in the spectral mode with 30 wavelengths might be larger than 2 GB, necessitating a total of 6 to 10 GB of RAM to keep both the image and the photon maps.

In addition to having processors with dozens of computational cores, modern workstations may also have multiple processors with non-uniform memory access (NUMA) speeds to various regions of the physical memory. Therefore, it is required to create effective shared memory access techniques and build specialized methods and algorithms for successful parallel processing of scene data to execute realistic rendering algorithms on these systems.

3 Traditional Parallelization Methods

In the context of the current study, three traditional parallel data processing methods, which are frequently used to parallelize the rendering algorithms based on backward photon mapping, were implemented and examined. These are synchronous and asynchronous parallelization and their combination. The results of testing the implementations of these traditional methods are described in Sec. 6.

3.1 *Synchronous Parallelization*

A fully synchronous parallelization paradigm uses all available workstation cores. Synchronous parallelization is a conventional way of parallelization on multi-core computers when each of the rendering stages is executed in parallel on shared data. All threads have the common scene data, the common data for forward and backward photon maps, so the amount of RAM needed to store intermediate computation data almost does not change as the number of computational threads increases. It makes this method the most effective among conventional parallelization

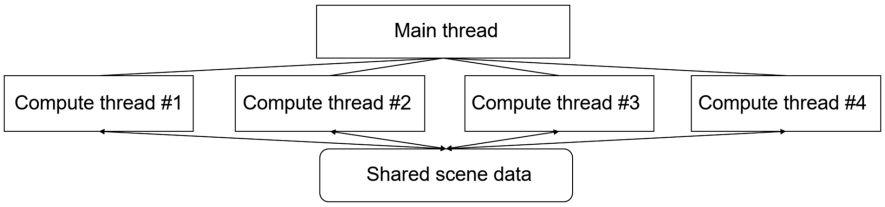

Fig. 2 The synchronous parallelization threads.

approaches in terms of using the workstation's RAM resources. The algorithmic complexity of synchronously parallelized algorithm is not changed and is equal to $O(p \cdot w \cdot h \cdot O_S)$, a memory consumption $O(w \cdot h)$, where p is the number of rendering phases, w and h are the rendered image width and height, respectively, and O_S is the scene complexity. The theoretical speedup of synchronous parallelization is limited by Amdahl's law.[9] The synchronous parallelization is schematically shown in Fig. 2.

The non-parallelized portions of the rendering algorithm, the need to synchronize threads, and the need to synchronize the memory access are the limits of the synchronous parallelization approach, which cause a delay of the virtual prototyping as the number of the computation cores employed rises. This slowdown is consistent with Amdahl's law.[9] Additionally, synchronous parallelization may create an extra delay in the case of nonuniform memory access times due to the utilization of shared data by computational threads executing on cores of several separate NUMA nodes.

3.2 *Asynchronous Parallelization*

A fully asynchronous parallelization paradigm is the second common parallelization technique. It is easy to organize a fully asynchronous rendering using a distributed computing model that uses one main thread and a group of computational threads because the stochastic ray tracing with progressive backward photon mapping method sums up the intermediate images obtained at separate phases of the calculation.[10] Each computational thread independently synthesizes the whole displayed image, whereas the main thread compiles the local calculation results from all independent computational threads to create the final image. The asynchronous parallelization is schematically shown in Fig. 3.

This method uses the workstation's RAM resources the least efficiently because each computational thread creates its own local image and forward and backward photon maps, even though all threads may share the same memory space and the same scene data. As a result, the amount of RAM needed to store the intermediate computation data is a multiple of the number of computational threads and is estimated as $O(w \cdot h \cdot n)$. At the same time, because the asynchronous parallelization method splits the rendering task by phases to be executed in parallel threads, the algorithmic complexity of each separate thread is equal to $O(\frac{1}{n} \cdot p \cdot w \cdot h \cdot O_S)$, where p is the number of rendering phases, w and h are the rendered image width and height, respectively, O_S is the scene complexity, and n is the number of workstation compute cores. In its turn, even if theoretical speedup of asynchronous parallelization looks linear, because RAM resources are rigorously constrained, the size of the backward photon map created at each stage of backward ray tracing is also constrained in addition to the image resolution. So, fewer backward and forward rays are traced during each computing phase, which ultimately results in a relative increase

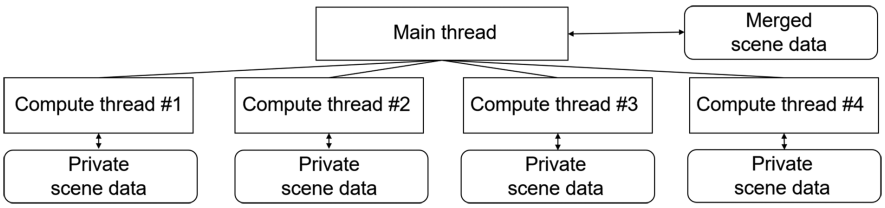

Fig. 3 The asynchronous parallelization threads.

in the overhead related to processing the intermediate image data produced after each calculation phase. This results in the fact that the number of forward ray intersections with backward photon maps is lower in the asynchronous calculation model, even though the ray-tracing efficiency may be higher. This happens because each computational thread's backward photon maps are private and cannot be intersected by the forward rays of other computational threads. Finally, in most cases, this leads to a lower virtual prototype image accuracy, while the number of traced rays is higher and the ray-tracing scalability looks better.

When employing the asynchronous parallelization method on a workstation with just 12 computing cores, it would take around 100 GB of RAM merely to store spectral images of a scene and backward photon maps due to the quantity of memory needed to store the image and backward photon maps data. Rendering at a resolution of 4 K, or 3840×2160, will take roughly 400 GB of RAM, which is too much even for modern workstations.

3.3 *Combination of Synchronous and Asynchronous Parallelizations*

A natural option on a workstation with NUMA is to attempt to combine the high scalability provided by asynchronous parallelization methods with the regional effectiveness of synchronous parallelization techniques.[11] This is accomplished by forming distinct, autonomous groups of synchronous threads, each of which renders a full image and runs exclusively on a separate NUMA node's core.[12] The dedicated control thread is also required; its job is to periodically compile the state of computations from every group of synchronous computation threads into a single final image and evaluate the achieved accuracy. Synchronous computations were carried out on each group of four cores with a shared memory, which matched the distribution of CPU cores across NUMA nodes and offers the quickest memory access for threads operating in synchronous mode.[13] The combination of synchronous and asynchronous parallelizations is schematically shown in Fig. 4.

The combination of synchronous and asynchronous parallelization methods splits the rendering task by phases for each phase to be executed in a separate group of threads running on a single NUMA node, so the algorithmic complexity of each of these separate groups of threads is equal to $O(\frac{1}{N} \cdot p \cdot w \cdot h \cdot O_S)$, where p is the number of rendering phases, w and h are the rendered image width and height, respectively, O_S is the scene complexity, and N is the number of workstation NUMA nodes. As can be seen, the benefit of this approach is the expected linear scalability of ray tracing and processing speed with an increase in the number of the utilized workstation's NUMA nodes. Combining synchronous and asynchronous parallel computations uses RAM resources at a rate that is a multiple of the number of NUMA nodes in the workstation, which, along with the binding of each synchronous group's threads to cores of a single NUMA node,[14] eliminates the need to access memory of other NUMA nodes in all cases except when collecting the intermediate visualization results. The memory consumption of this method is $O(w \cdot h \cdot N)$. It should be pointed out that the number of NUMA nodes grows much more slowly than the number of compute cores, and even for a high-performance workstation, the number of NUMA nodes does not exceed 8. So, the memory consumption of this method is significantly better than purely asynchronous calculations and is acceptable for use in the rendering algorithm.

Because synchronously executed sections of the algorithm impair the effectiveness of ray tracing, the overall number of rays traced and processed is substantially smaller when compared

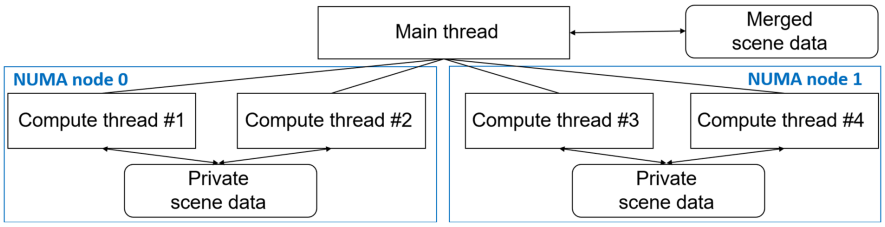

Fig. 4 The combination of synchronous and asynchronous parallelization threads distributed among NUMA nodes.

with utilizing the entirely asynchronous technique. Therefore, the primary objective of the current study was to accelerate, as much as possible, the synchronously executed part of the parallelization of rendering using the backward photon mapping approach with keeping the RAM resources consumption almost intact.

4 Two-Level Semi-synchronous Parallelization Method

In our research, we propose a new two-level semi-synchronous parallelization method for the realistic rendering algorithms that are based on the progressive backward photon mapping method. The designed parallelization method consists of two levels: completely synchronous and semi-synchronous. This method does not directly utilize asynchronous calculations to make effective use of the RAM and CPU resources of a workstation with uniform memory access. The approach combines the benefits of synchronous calculations when utilizing a limited number of cores with the natural scalability of asynchronous calculations when utilizing a large number of cores.

4.1 *Two Levels of the Threads Hierarchy*

Fully synchronous parallel calculations constitute the first level. The entire image is rendered using a synchronous group of first-level threads, using shared scene data and backward photon maps. Each of the first-level thread groups uses a random mask built with repeating 32×32 pixels tiles to render an image that only contains a fragment of the complete scene image. A first-level synchronous group of threads is schematically shown in Fig. 5.

As was stated in the previous section, increasing the number of synchronously running threads too much results in a significant calculation slowdown. In our research, we found that, even if the synchronous parallelization of the rendering algorithm can be implemented effectively, a noticeable slowdown appears if the number of parallel threads exceeds four. This result is consistent with Amdahl's law[9] and is expected. So, we used either 2 or 4 threads as the size of the synchronous group of threads.

Semi-synchrony is added at the second parallelization level. The second-level threads group is made up of first-level thread groups, and each of them individually renders a sub-image of the scene's image using a 32×32 pixels random mask. As a result, the second-level thread group visualizes the entire image and distributes masks for the first-level groups, whereas the first-level thread groups each create a random portion of the final image on their own. First and second-level thread groups are schematically shown in Fig. 6.

The second-level's thread group synchronizes and forms a whole image after a certain number of calculation phases. Following synchronization, the computations proceed with the first-level threads' random masks being re-randomized. This strategy is used to decrease the potential delay brought on by thread synchronization. This thread synchronization is performed in the following steps:

1. The main thread of a semi-synchronous level receives the signal when one of the groups of synchronous computation threads completes the execution of a predetermined number of phases of the rendering algorithm.
2. The main thread sends notification signals to stop all other synchronous first-level thread group calculations.

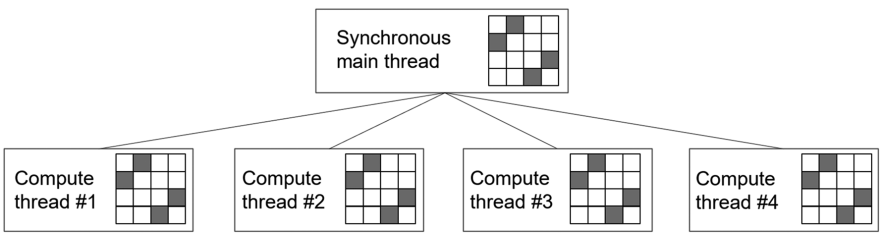

Fig. 5 The group of synchronous threads that executes the realistic rendering of the sub-image defined by a given mask.

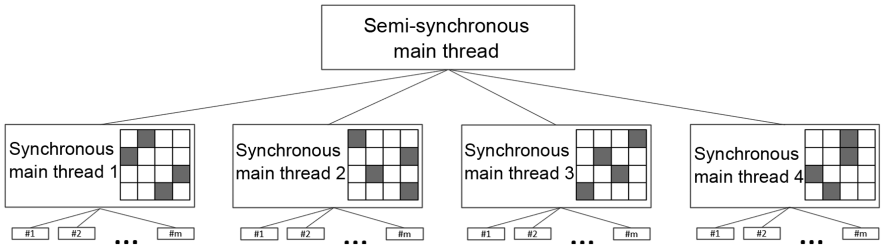

Fig. 6 The group of semi-synchronous threads rendering the entire image and consisting of groups of synchronous threads with their own masks.

3. Upon receiving a stop notification from a parent semi-synchronous level main thread, a first-level thread group tries to stop its execution. If calculations are in the forward ray-tracing stage, they can be interrupted immediately without losing any data. If not, the thread group waits until the forward ray tracing begins and interrupts the computations.
4. The semi-synchronous level's main thread creates a whole scene image and distributes new random masks to groups of synchronous threads.
5. The computations of the synchronous thread groups are carried out by the main thread for the next number of phases as stated.

The time diagram of interaction between threads of the semi-synchronous level are shown in Fig. 7.

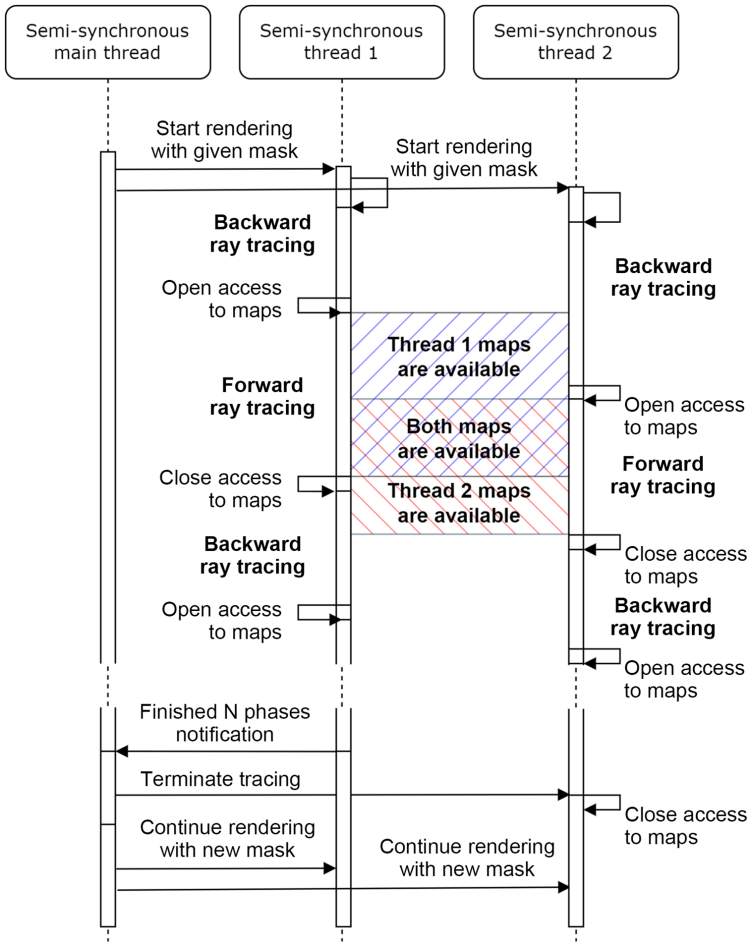

Fig. 7 The time diagram for a group of semi-synchronous threads rendering the entire image.

It should be emphasized that, when compared with the conventional synchronous computation model, the employment of a pseudo-asynchronous computation model at the semi-synchronous computing level does not result in a considerable increase in the amount of used RAM. This is because there is no overlap of ray-tracing operations across the various synchronous thread groups, allowing each group to render its own part of the whole image while still using the same image data. Additionally, the backward photon maps created by each thread are only created based on their masks, which reduces the size of each map by the number of semi-synchronously running groups of threads and prevents an increase in RAM use.

Such task distribution among groups of threads at the semi-synchronous computing level allows us to lower the algorithmic complexity of calculations performed by each group of synchronous threads by the number of groups at this level. Each synchronous group traces $O(\frac{1}{m}w \cdot h)$ rays at each rendering phase, where m is a number of synchronous groups at the semi-synchronous level and w and h are the rendered image width and height, respectively. Because the scene complexity is of a higher order than the photon maps complexity, it is possible to estimate the algorithmic complexity of calculations performed by each synchronous group of threads as $O(\frac{1}{m} \cdot p \cdot w \cdot h \cdot O_S)$, where p is the number of rendering phases and O_S is the scene complexity. At the same time, the memory consumption of this thread group is estimated as $O(\frac{1}{m} \cdot w \cdot h)$. Then, for a semi-synchronous group of m synchronous groups, the total memory consumption would be equal to one of purely synchronous calculations, which is $O(w \cdot h)$. The task homogeneity between several groups of synchronous threads should allow for obtaining the linear acceleration with increasing the number of groups of threads of the semi-synchronous level.

The achievement of task homogeneity for backward ray tracing between several groups of synchronous threads is made possible by the random selection of masks. The 32×32 pixels mask showed good performance on up to 16-core uniform memory access CPUs; however, if their number is higher, the mask's size can be expanded up to 128×128 pixels or more. In the case of an unequal speed of the algorithm in multiple processing threads, the periodic update of random masks is especially necessary to ensure the homogeneity of the distribution of the backward rays over the scene. Because all threads share the same shared memory space, they all have access to the same scene and picture data while also having separate backward photon mappings. Moreover, taking into account that synchronous thread groups render independent image areas, they can share same intermediate and final image objects as they operate on non-intersecting memory areas.

Our tests show that, if a semi-synchronous computational level's synchronization interval is between 4 and 8, then individual groups of synchronous computational threads operate almost synchronously, and the interruption of computations happens when all threads are at the step of forward ray tracing, which does not necessitate additional CPU time nor distort the final image.

4.2 *Asynchronous Access to Backward Photon Maps at the Semi-synchronous Parallelization Level*

When indirect and caustic luminance values are accumulated at image points during the forward ray-tracing phase, it is possible that the forward ray will not intersect the backward photon map of the corresponding synchronous group. However, it may be able to intersect backward photons of another first-level group of synchronous threads that are located in the same shared memory and, as a result, add luminance to image pixels. For this need, an algorithm for asynchronous access to backward photon maps of other first-level groups of synchronous threads that only employ atomic operations was proposed to increase the effectiveness of using the results of the forward ray tracing at the stage of the accumulation of indirect and caustic luminance values. To prevent shared memory access conflicts, the atomic addition and atomic compare-and-swap operations are used while accounting for the luminance values of the indirect and caustic lighting at the same points of the image from different threads.

Additionally, to avoid memory leaks and accuracy loss, the thread must be forbidden from recycling its backward photon maps when some thread is accessing them. To achieve this, a solely atomic operations-based technique for opening and closing access to backward photon

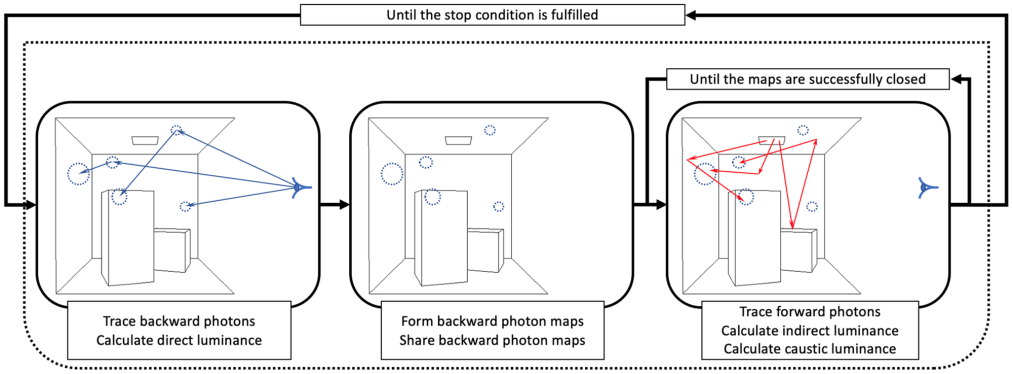

Fig. 8 The flowchart for the parallel modification of the progressive backward photon mapping rendering algorithm.

maps was created. Due to the fact that the proposed algorithm does not require synchronization, the ray-tracing process is not slowed down by the access to backward photon maps, and the efficiency of searching for the intersection of rays with backward photon maps is increased. The main idea of the proposed approach is that each backward photon map keeps a flag that notifies other threads when the data on this map is available, allowing other threads to access the backward photon maps of a certain synchronous threads group when these maps exist. The progressive backward photon mapping method should thus include two extra steps: opening access to one's backward photon maps before beginning the forward ray-tracing phase and shutting access to one's own backward photon maps when the forward ray-tracing step is complete. At the same time, the maps owner thread should continue forward ray tracing until all other threads release the relevant map because backward photon maps cannot be closed while they are being updated by other threads. Only then should the owner thread go on to the next rendering stage. Figure 8 shows the required modification of the flowchart for the progressive backward photon mapping rendering method to be implemented in the semi-synchronous parallel computation environment.

The algorithm's core point is that each backward photon map is maintained with a counter indicating how many threads are actively interacting with the relevant backward photon map's data. This counter is a flag indicating that maps exist and are open for an update by other threads. A counter value of 0 denotes the absence of the map, whereas a value larger than 0 denotes the existence of the map and the ability to utilize it to accumulate luminance values at corresponding image pixels.

Opening access to photon maps is executed by an atomic increment of the corresponding flag counter by 1. As for the closing, the photon maps are considered to be no longer in use by other threads, access to them is closed, and the rendering process may go on to the next stage of the rendering algorithm after the thread that created them successfully completed the compare-and-swap operation with the swap of the corresponding flag counter from 1 to 0. As a result, after the backward photon maps and corresponding acceleration structures were created, the following modifications are introduced in the forward ray-tracing stage of the first-level synchronous threads group:

1. The atomic increment operation is used to increase the flag counter for the relevant backward photon map by 1, indicating that the map is present and available for an update by other threads.
2. The forward ray tracing and luminance accumulation are started with two stopping criteria: the first is the number of forward rays traced and the second is the successful completion of the atomic compare-and-swap operation that reduces the number of active threads from 1 to 0 after completing the tracing the required number of rays.

The flowchart in Fig. 9 shows this procedure.

A similar procedure applies when accessing the backward photon maps of other synchronous thread groups: if the corresponding thread flag counter is positive and is successfully

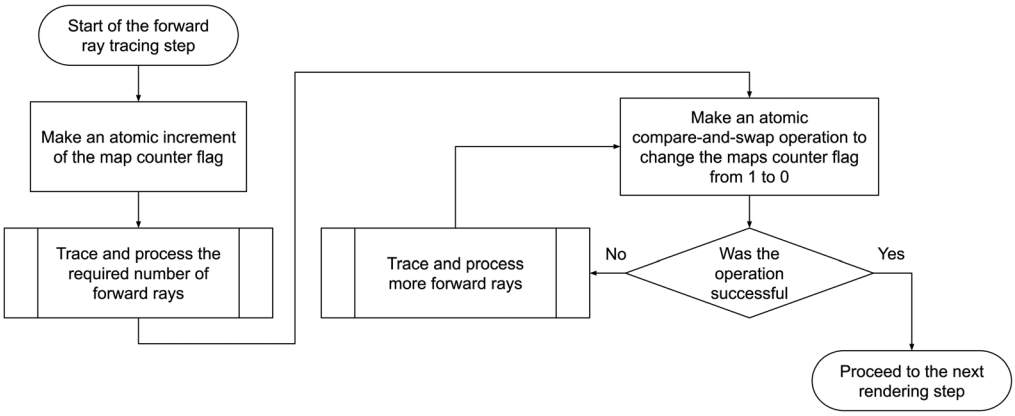

Fig. 9 The flowchart for the forward ray-tracing phase with granting other threads access to the backward photon maps.

incremented using a compare-and-swap operation, then access to the maps is granted, and maps can be used to account for indirect and caustic luminances. The counter is atomically decreased by 1 upon the completion of updating the maps. Atomic addition operations are also employed when adding up luminance values. The following steps are taken by a thread if it wants to access the backward photon maps of another first-level synchronous thread group while tracing a forward ray:

1. If the next first-level thread group photon maps flag counter is positive, then attempt is made to increment it using the atomic compare-and-swap operation. In the case of failure, the thread proceeds to the next available photon maps.
2. If the compare-and-swap operation was successful, the forward ray's intersection with the backward photons of the backward photon map is executed. If the intersection is found, then the luminance value is accumulated at the corresponding pixels of the intermediate image using atomic addition operations. The corresponding map statistics are also updated for correct accuracy estimation.
3. After the work with the backward photon map is finished, the corresponding flag counter is atomically decremented by 1 to release the maps.

The statistics that are accumulated when accessing backward photon maps include the number of forward rays processed with this photon map. This ray counter is required to calculate the final luminance distribution in image pixels and achieved accuracies.

The flowchart in Fig. 10 shows the modification to the algorithm of the forward ray tracing to use all backward photon maps that are available at the moment when calculating the caustic and indirect luminance values.

As a result, a thread from a different synchronous thread group uses the backward photon maps data, and the group that originally generated it continues the forward ray tracing step and luminance calculation. The synchronous threads group will cease the forward ray tracing as soon as no thread utilizes the backward photon maps of this threads group and the stopping requirement is satisfied, at which point it will go on to the subsequent rendering stage. Access to the memory of other synchronous thread groups is restricted by the criterion set by the time needed to finish the tracing of a preset number of rays to prevent the possibility of mutual blocking of threads during the forward ray tracing phase.

5 Three-Level Semi-synchronous Parallelization Method

A three-level rendering parallelization method was created as part of the current study to handle multiprocessor workstations with NUMA. This approach's fundamental concept is to integrate the two-level semi-synchronous parallelization method with the previously studied traditional methods of combining synchronous and asynchronous parallel computations. The three levels of this approach are synchronous, semi-synchronous, and asynchronous:

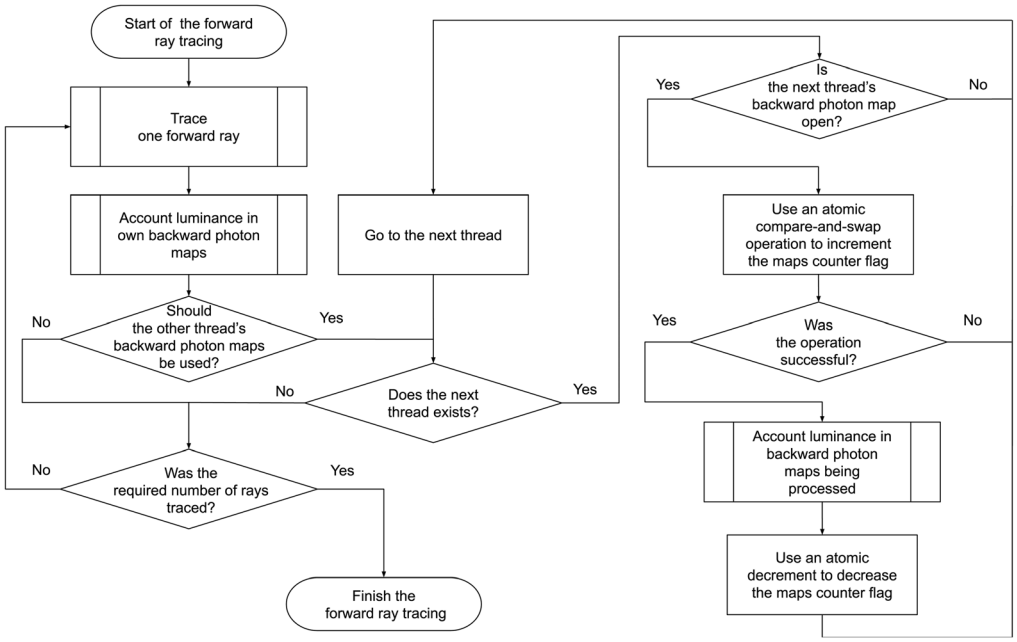

Fig. 10 The flowchart for the single ray-tracing phase with accessing backward photon maps of all available first-level threads of the same semi-synchronous level.

1. Similar to two-level parallelization, the fully synchronous parallel computations form the first parallelization level. Each synchronous thread group contributes to the overall scene image, which is defined by a random mask. The second level combines several synchronous thread groups with access to one another's first-level groups' backward photon maps into a semi-synchronous parallel calculations level.
2. The second level's main thread constructs masks, synchronizes group calculations, builds a whole scene image, and then creates new random masks.
3. The third level is a completely asynchronous calculation without shared memory.

To operate the asynchronous calculation on the third level, we propose using the distributed computation model. So, the third parallelization level consists of a group of control processes running a two-level parallel rendering. The number of processes is equal to the number of NUMA nodes of the workstation. One of them is the main rendering process that controls the whole rendering process and launches additional local rendering processes, one for each NUMA node.

TCP/IP and a loopback interface are used to arrange communication with local rendering processes. This enables the three-level parallel rendering interfaces to be used in a distributed environment and launch remote rendering processes to add them to the same rendering threads and processes hierarchy.

On systems with several NUMA nodes, the third level of parallelization organizes effective computations using an asynchronous distributed computing architecture. The execution of each of the third-level top threads is tightly restricted to the cores of a single NUMA node and is not permitted on the cores of any other NUMA nodes.

Two-level parallel renderings are performed by both the main and computational processes. The main thread regularly gathers the status of both local and distant computations to update the state of the main rendering result without pausing the computations. Based on the connection speed, the data collection frequency is chosen such that local intermediate rendering data is gathered more frequently and remote intermediate rendering data is gathered less frequently.

As described in previous sections, the two top levels of the parallelization hierarchy effectively distribute the rendering tasks between groups of synchronous threads with high homogeneity. The asynchronous level distributes rendering phases among NUMA nodes and remote servers, whereas the semi-synchronous level distributes random parts of the image among

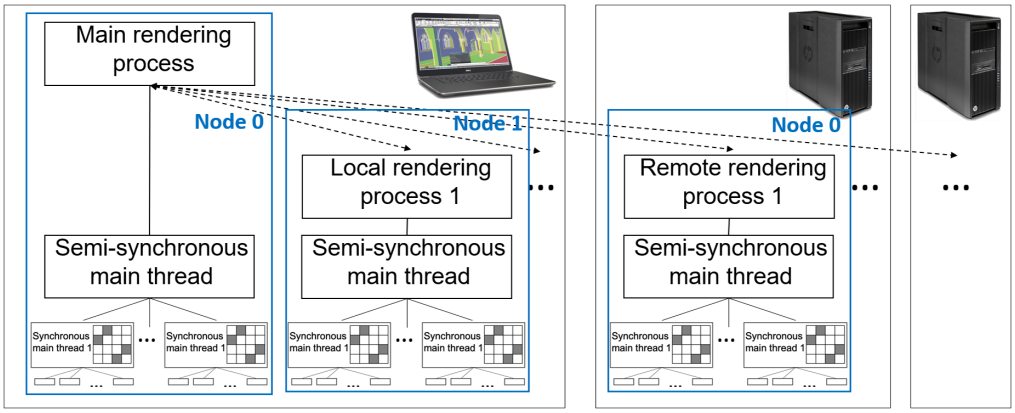

Fig. 11 The three-level hierarchy of local and remote processes and threads.

synchronous groups of threads. So it is possible to estimate the complexity and memory consumption of each parallelization level.

The algorithmic complexity of calculations performed by a synchronous group of threads of the first level is $O(\frac{1}{m \cdot N} \cdot p \cdot w \cdot h \cdot O_S)$, where p is the number of rendering phases, w and h are the rendered image width and height, respectively, O_S is the scene complexity, m is the number of synchronous thread groups, and N is the number of NUMA nodes. At the same time, the memory consumption of such thread group is estimated as $O(\frac{1}{m} \cdot w \cdot h)$. The synchronous parallelization is limited to 4 threads per group, which results in an almost linear calculation speedup.

The second-level group of threads executes calculations with complexity $O(\frac{1}{N} \cdot p \cdot w \cdot h \cdot O_S)$ and memory consumption $O(w \cdot h)$. Due to the task homogeneity between different groups of synchronous threads, this level also results in almost linear acceleration.

The third-level group of threads executes the whole calculations with complexity $O(p \cdot w \cdot h \cdot O_S)$ and memory consumption $O(N \cdot w \cdot h)$. The asynchrony of calculation at the top level with a relatively low number of asynchronous groups allows us to execute calculations without lowering the sizes of the photon maps and keeping the acceleration almost linear.

A three-level structure of threads and processes is shown in Fig. 11. The time diagram of the interaction between the main third-level process and single local rendering process is shown in Fig. 12. Each of these processes is running a two-level semi-synchronous group of threads, as described in the previous section.

The three-level rendering threads structure allows us to efficiently utilize all local resources as well as manage the various rendering models, such as rendering using multiple multiprocessor workstations or rendering using a basic machine (laptop) and resources from a distant server.

By binding each of the synchronous group's threads to cores on a single NUMA node and using all available NUMA nodes of the workstation, the three-level semi-synchronous parallelization method effectively uses RAM resources similarly to combining synchronous and asynchronous parallel computations. This approach eliminates the third-level thread groups' need to access the memory of other NUMA nodes, except for the case of collecting intermediate rendering results.

6 Simulation Results

The two-level and three-level semi-synchronous parallelization methods described in this article were tested on two scenes containing virtual prototypes of complex optical devices.

Testing was performed on two workstations. The first workstation was equipped with a single Intel Xeon 6230 2.1GHz 12-core processor and 128GB RAM running at 2933 MHz with uniform memory access and a single NUMA node. Because this workstation has uniform memory access, it has a constant memory idle latency of 92 ns and memory bandwidth of 77.5 GB/s.

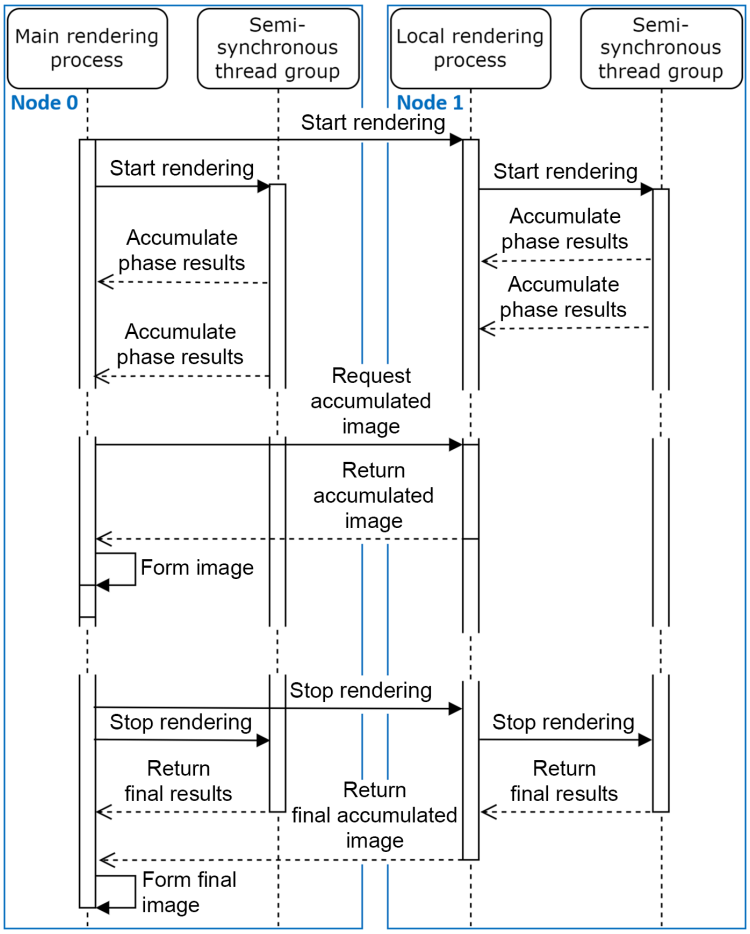

Fig. 12 The time diagram for a top level of three-level threads hierarchy for the main process and a single local process, each of them running its own set of semi-synchronous threads as described in Sec. 4.

The second workstation was equipped with two AMD EPYC 7281 2.1GHz 16-core processors and 256GB RAM running at 2666 MH. The specificity of the AMD EPYC processor series is that each processor has nonuniform memory access with several NUMA nodes. In the case of AMD EPYC 7281, there are 4 NUMA nodes with 4 cores in each CPU. So in the case of two CPUs, it results in a complex workstation with 8 NUMA nodes and 32 cores total. Moreover, the memory latency and memory bandwidth significantly depend on the mutual location of cores inside the workstation. For cores of the same NUMA node, the memory idle latency is about 89 ns, and the memory bandwidth is about 19.8 GB/s. If cores are located at different NUMA nodes of the same CPU, then the memory latency increases to about 136 ns, and the memory bandwidth drops to about 17.4 GB/s. If the cores are located in different CPUs, then the change is more significant: the memory latency increases up to 240 ns, and memory bandwidth drops down to 8.3 GB/s.

The results section follows the order of the experiments that were used when evaluating the parallelization methods described in this paper and are grouped by the method type. The following combinations of parallelization methods and workstations were evaluated for virtual prototyping:

1. Traditional parallelization methods. These methods were evaluated on both test workstations. Traditional methods include:

    - Synchronous parallelization method.
    - Asynchronous parallelization method.

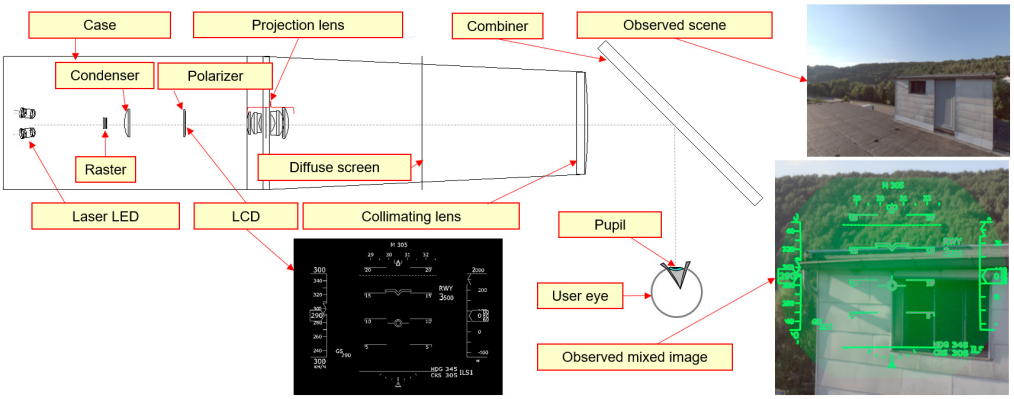

Fig. 13 The principal scheme of the head-up display test scene.

- Two-level semi-synchronous parallelization method. This method was evaluated only on the workstation with uniform memory access (the workstation with a single Intel Xeon CPU).

2. Parallelization methods that take the information about the distribution of cores among NUMA nodes of the workstation. These methods were evaluated only on the workstation with NUMA (the workstation with dual AMD EPYC CPUs). These methods include:

- Combination of traditional synchronous and asynchronous parallelization methods.
- Three-level semi-synchronous parallelization method.

6.1 *Test Scenes Setup*

Each combination of a parallelization method and workstation listed above was tested on scenes containing virtual prototypes of two real optical devices.

The first scene is a model of the optical system of the head-up display (HUD).[15] The principal scheme of the HUD system is shown in Fig. 13. The LCD matrix, with a dynamic slide displaying some information is illuminated by laser LEDs and projected onto a diffuse screen, located at the focus of the collimation system projecting the image of the slide through the combiner to the observer's eye. At the same time, the observer sees the surrounding space through the same combiner. The eye model is defined by the real eye characteristics such as pupil size, field of view, and focal length.

The second scene is a model of a virtual prototype of the image-forming optical system of the lens camera placed in a virtual environment.[16] The test scene setup and image formed on the camera sensor are shown in Fig. 14. The camera lens has a field of view of 40 deg and is directed

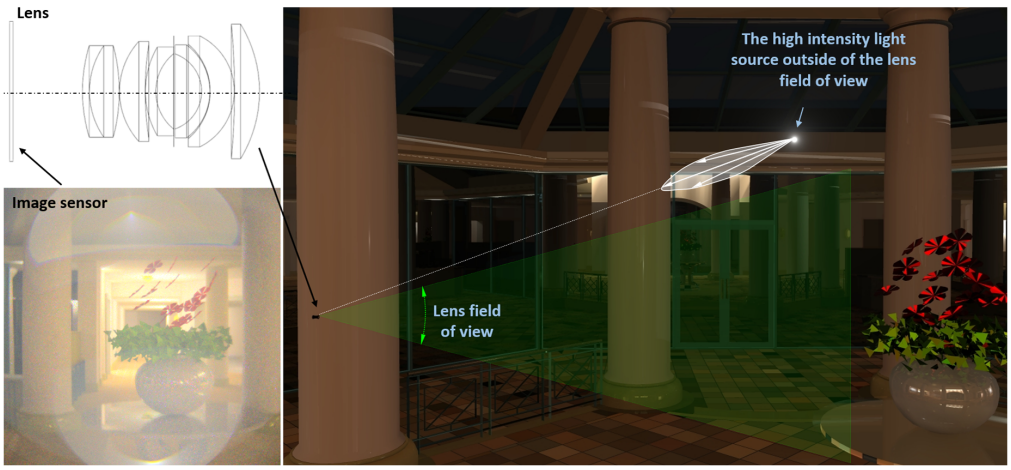

Fig. 14 The setup of the lens system test scene. The rendered image is shown in the left part. The right part shows the whole scene setup displayed with the OpenGL visualization.

at the vase with flowers on the table. The scene is illuminated by a set of light sources, with one high-intensity light source being located near the lens field of view. Due to the camera lens surfaces have Fresnel surface properties, the interreflections between the lens surfaces cause ghosts to appear on the camera sensor.

6.2 *Traditional Parallelization Methods*

At first, traditional parallelization methods that include synchronous and asynchronous parallelizations, were tested with both workstations, first, using the head-up display system test scenes and then using the lens system scene. Each test scene was rendered for 30 min, and the number of traced forward and backward rays was recorded. The ray tracing speedup was calculated as the total number of traced rays and normalized to the number of rays traced with a single core when using the corresponding method. This speedup value allows us to estimate the scalability of each method when increasing the number of cores without taking the number of intersections of forward and backward rays into account. Achieved accuracies were evaluated only on the maximum load of each workstation, i.e., when using all available cores, to compare the rendering efficiency of each of the tested methods. Test results are shown using tables and speedup graphs.

Tables 1 and 2 give the results of testing the traditional synchronous and asynchronous parallelization methods applied to the head-up display system scene on workstations with uniform and NUMA, respectively. The corresponding graph of dependence of the ray tracing speedup from the number of used computation cores is shown in Fig. 15

Table 1 Testing results of the application of the synchronous and asynchronous parallelization methods to the simulation of the virtual prototype of the head-up display system scene on the workstation with uniform memory access with 30 min simulation time.

	Synchronous parallelization			Asynchronous parallelization		
Number of used cores	Forward rays	Backward rays	Tracing speedup	Forward rays	Backward rays	Tracing speedup
1 core	60.8M	27.7M	1.0	64.3M	28.9M	1.0
2 cores	87.8M	50.9M	1.6	128.9M	37.6M	1.8
4 cores	170.3M	83.6M	2.9	252.3M	60.9M	3.4
8 cores	278.3M	142.2M	4.8	626.0M	97.7M	7.8
12 cores	406.4M	215.7M	7.0	972.4M	118.5M	11.7

Table 2 Testing results of the application of the synchronous and asynchronous parallelization methods to the simulation of the virtual prototype of the head-up display system scene on the workstation with NUMA with 30 min simulation time.

	Synchronous parallelization			Asynchronous parallelization		
Number of used cores	Forward rays	Backward rays	Tracing speedup	Forward rays	Backward rays	Tracing speedup
1 core	41.4M	22.3M	1.0	44.6M	23.6M	1.0
4 cores	132.4M	71.4M	3.2	215.3M	49.3M	3.9
8 cores	223.6M	120.4M	5.4	472.2M	78.4M	8.1
16 cores	388.2M	196.3M	9.2	982.6M	116.9M	16.1
24 cores	545.3M	282.1M	13.0	1381.4M	182.9M	22.9
32 cores	681.3M	345.1M	16.1	1968.1M	233.9M	32.3

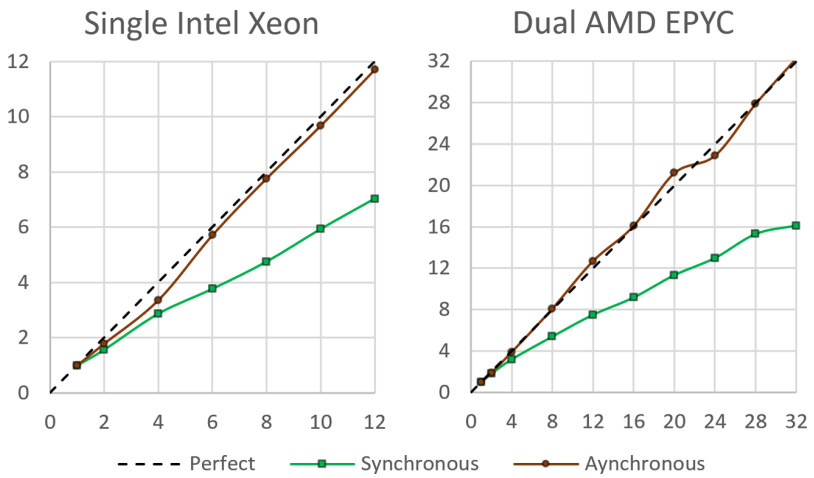

Fig. 15 Ray-tracing speedup achieved when using the synchronous and asynchronous parallelization methods to parallelize the simulation of the virtual prototype applied to the head-up display system scene on workstations with uniform and NUMA.

Table 3 Testing results of the application of the synchronous and asynchronous parallelization methods to the simulation of the virtual prototype of the lens system scene on the workstation with uniform memory access with 30 min simulation time.

	Synchronous parallelization			Asynchronous parallelization		
Number of used cores	Forward rays	Backward rays	Tracing speedup	Forward rays	Backward rays	Tracing speedup
1 core	45.9M	35.1M	1.0	56.7M	41.1M	1.0
2 cores	85.5M	57.9M	1.8	87.3M	60.6M	1.5
4 cores	147.6M	99.6M	3.1	266.7M	110.4M	3.9
8 cores	186.6M	121.8M	3.8	616.2M	150.9M	7.8
12 cores	169.2M	114.3M	3.5	923.7M	167.7M	11.2

Tables 3 and 4 give the results of testing the traditional synchronous and asynchronous parallelization methods applied to the lens system scene on workstations with uniform and NUMA, respectively. The corresponding graph of dependence of the ray-tracing speedup from the number of used computation cores is shown in Fig. 16

After 30 min of calculations on the workstation with uniform memory access, the synchronous parallelization method achieved the accuracy of 31.1%, and the asynchronous parallelization method achieved the accuracy of 34.2% on a head-up display system scene. A lens system scene achieved the accuracy of 19% with synchronous parallelization and 20.4% with asynchronous parallelization. On the workstation with NUMA, the synchronous parallelization method achieved the accuracy of 27%, and the asynchronous parallelization method achieved the accuracy of 28.2% on a head-up display system scene. At the same time, a lens system scene achieved the accuracy of 14.1% with synchronous parallelization and 15.9% with asynchronous parallelization.

The main drawbacks of the synchronous parallelization approach are brought on by the presence of non-parallelized parts of the rendering algorithm and the requirement to synchronize threads, which causes a rendering slowdown as the number of compute cores increases. This slowdown was anticipated due to Amdahl's law.[9] The NUMA of the second test workstation led to an additional slowdown of the synchronous parallelization method, which was a result

Table 4 Testing results of the application of the synchronous and asynchronous parallelization methods to the simulation of the virtual prototype of the lens system scene on the workstation with NUMA with 30 min simulation time.

	Synchronous parallelization			Asynchronous parallelization		
Number of used cores	Forward rays	Backward rays	Tracing speedup	Forward rays	Backward rays	Tracing speedup
1 core	38.1M	28.5M	1.0	42.0M	30.6M	1.0
4 cores	122.7M	83.4M	3.1	216.3M	90.0M	4.2
8 cores	151.8M	101.7M	3.8	518.1M	122.7M	8.8
16 cores	156.3M	107.7M	4.0	988.2M	162.3M	15.8
24 cores	248.7M	172.2M	6.3	1299.6M	250.2M	21.3
32 cores	244.8M	173.1M	6.3	1691.7M	279.9M	27.1

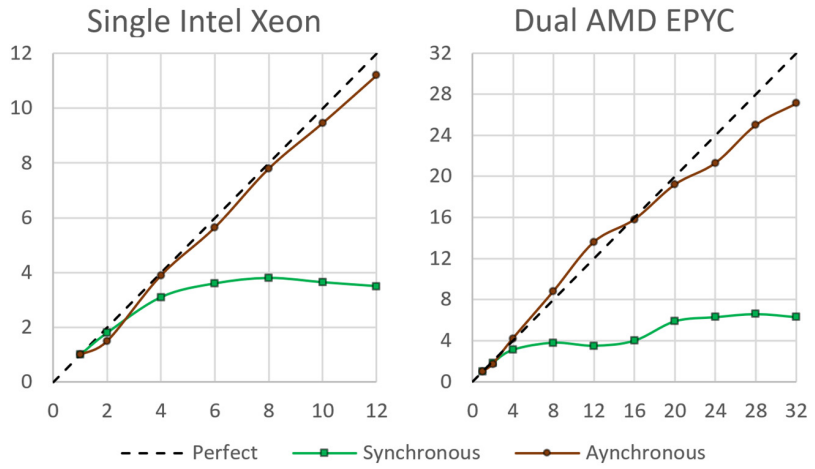

Fig. 16 Ray-tracing speedup achieved when using the synchronous and asynchronous parallelization methods to parallelize the simulation of the virtual prototype of the lens system scene on workstations with uniform and NUMA.

of synchronous computational threads running on different NUMA nodes' cores utilizing the shared data.

As for the asynchronous parallelization approach, it should be noted that, despite the ray-tracing speedup nearly linearly depending on the number of used computation cores, the achieved accuracy is noticeably lower. The reason for this is the decrease in the number of intersections of forward rays with backward photons of backward photon maps because each computation thread has its own independent photon maps, and these maps are not available for intersection by forward rays of other computation threads. In addition, full thread independence increases the use of memory resources by a multiple of the number of threads, which results in excess memory usage, especially in the case of rendering being performed in the spectral color model.

6.3 *Two-level Semi-synchronous Parallelization Method*

Because the designed two-level semi-synchronous parallelization method is intended to be used in a shared memory environment and is not expected to give significant calculation speedup on a workstation with NUMA, it was only tested on a first test workstation with a single 12-core

Table 5 Testing results of the application of the two-level semi-synchronous parallelization methods to the simulation of the virtual prototype of the head-up display system and lens system scenes on the workstation with uniform memory access with 30 min simulation time.

	HUD system scene			Lens system scene		
Number of used cores	Forward rays	Backward rays	Tracing speedup	Forward rays	Backward rays	Tracing speedup
1 core	56.3M	29.7M	1.0	46.2M	41.1M	1.0
2 cores	134.8M	51.9M	2.2	139.5M	60.3M	2.3
4 cores	275.0M	92.3M	4.3	243.9M	103.2M	4.0
8 cores	529.0M	186.1M	8.3	497.1M	204.9M	8.0
12 cores	813.4M	212.9M	11.9	740.7M	309.0M	12.0

Intel Xeon CPU. The number of threads running synchronously in the groups of the first level was 2.

Table 5 gives the test results after 30 min of computation using the two-level semi-synchronous parallelization method with the progressive backward photon mapping on the workstation with the uniform memory access (single 12-core Intel Xeon processor). The matching graph of the acceleration for the total number of traced and processed rays and the number of computing threads employed is shown in Fig. 17.

After 30 min of calculation with the two-level semi-synchronous parallelization methods an accuracy of 30.1% was achieved for the head-up display system scene and 15.6% for the lens system scene. The combination of higher computation asynchrony and absence of fully asynchronous threads accounts for the improved ray processing efficiency. By granting mutual asynchronous access to backward photon maps of various thread groups, the computation accuracy significantly increased.

The obtained test results demonstrate that the two-level semi-synchronous parallelization method scales similarly with a fully asynchronous computation while being more efficient in rendering. It also uses roughly the same amount of memory as the straightforward synchronous parallelization method and does not call for any additional private memory.

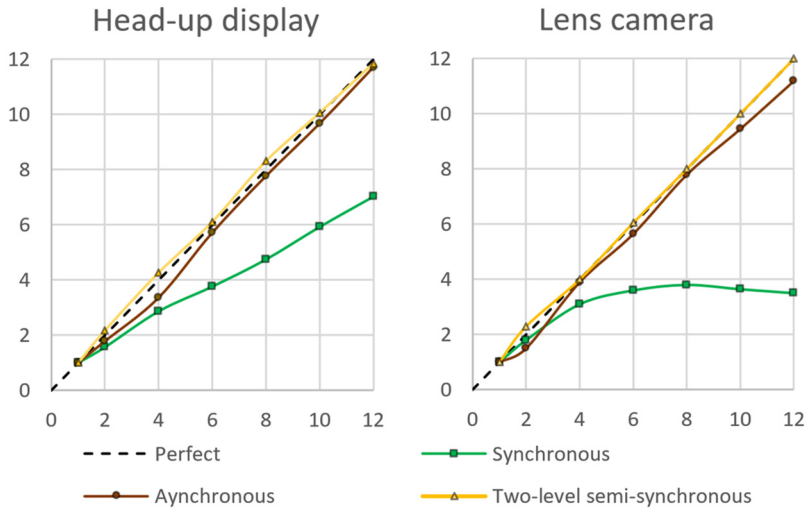

Fig. 17 Ray-tracing speedup achieved when using the two-level semi-synchronous parallelization method to parallelize rendering with backward photon mapping applied to the test scenes on workstation with uniform memory access in comparison with traditional parallelization methods.

6.4 *Combination of Synchronous and Asynchronous Parallelizations, and Three-level Semi-synchronous Parallelization*

Both the combination of traditional synchronous and asynchronous parallelization methods and the designed three-level semi-synchronous parallelization method were only tested on a workstation with NUMA (dual 16-core AMD EPYC CPUs). The reason for this is that both of these methods are designed to distribute rendering tasks among NUMA nodes, so if only one NUMA node is present, then the method is reduced to a simple synchronous or two-level semi-synchronous parallelization and does not give any extra benefits. In this test, the number of threads running in synchronous groups of the first level of the three-level parallelization method was 2.

Tables 6 and 7 give the test results obtained after 30 min of rendering using the combination of synchronous and asynchronous parallelization methods and the three-level semi-synchronous parallelization method when rendering the test scenes using the progressive backward photon mapping method on the workstation with the NUMA (dual 16-core AMD EPYC processors). The matching graph of the acceleration for the total number of traced and processed rays and the number of computing threads is shown in Fig. 18.

Table 6 Testing results of the application of the combination of synchronous and asynchronous and three-level semi-synchronous parallelization methods to rendering with backward photon mapping applied to the HUD system scene on the workstation with NUMA for 30 min.

	Combination of synchronous and asynchronous parallelization			Three-level semi-synchronous parallelization		
Number of used cores	Forward rays	Backward rays	Tracing speedup	Forward rays	Backward rays	Tracing speedup
1 core	33.1M	17.8M	1.0	41.4M	22.3M	1.0
4 cores	115.9M	62.4M	3.5	236.5M	80.3M	4.2
8 cores	229.7M	123.8M	6.9	457.1M	155.6M	8.1
16 cores	471.0M	253.7M	14.2	877.7M	298.9M	15.5
24 cores	673.4M	363.0M	20.3	1253.7M	427.4M	22.2
32 cores	904.0M	488.4M	27.3	1647.3M	561.2M	29.2

Table 7 Testing results of the application of the combination of synchronous and asynchronous, and three-level semi-synchronous parallelization methods to rendering with backward photon mapping applied to the lens system scene on the workstation with NUMA for 30 min.

	Combination of synchronous and asynchronous parallelization			Three-level semi-synchronous parallelization		
Number of used cores	Forward rays	Backward rays	Tracing speedup	Forward rays	Backward rays	Tracing speedup
1 core	34.8M	26.1M	1.0	35.1M	30.6M	1.0
4 cores	119.4M	81.0M	3.3	205.2M	85.5M	4.4
8 cores	231.9M	157.5M	6.4	406.5M	172.2M	8.8
16 cores	407.7M	285.3M	11.4	764.1M	324.0M	16.6
24 cores	703.8M	481.8M	19.5	1107.6M	468.9M	24.0
32 cores	963.3M	666.9M	26.8	1539.0M	651.6M	33.3

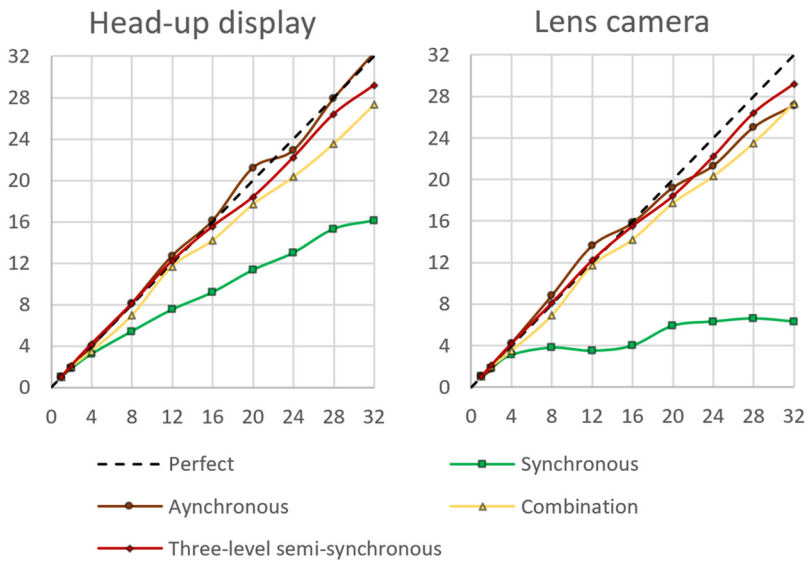

Fig. 18 Ray-tracing speedup achieved when using the combination of synchronous and synchronous and three-level semi-synchronous parallelization methods to rendering with backward photon mapping applied to test scenes in comparison with traditional methods.

Additionally, tests showed that the combination of traditional synchronous and asynchronous paralellization methods has the same memory load with three-level parallelization; however, it is less efficient because it works in a completely synchronous mode on a single NUMA node.

After rendering for 30 min with the combination of synchronous and asynchronous parallelization methods, an accuracy of 23.5% was achieved for an HUD system scene and accuracy of 13.1% was achieved for a lens system scene, which is better than utilizing entirely synchronous or asynchronous parallelization. The efficiency of ray processing is lower than that of completely asynchronous calculations because of the presence of the completely synchronously parts of the algorithm, so the overall number of traced and processed rays is fewer than when utilizing the fully asynchronous approach.

The three-level semi-synchronous parallelization method after 30 min achieved the accuracy of 33% for a head-up display system scene and 11.7% for a lens system scene. The accuracy is higher than other methods because of the semi-synchronous approach, which allowed for achieving the scalability of asynchronous calculation with a better rate of intersections between forward rays and backward photon maps, which is specific to traditional synchronous calculations.

Tables 8 and 9 give the cumulative results of the achieved speedup and corresponding accuracies when utilizing all available computation cores for both of the test workstations.

The acquired results show the benefits of the designed three-level method, which are in the linear scaling of the performance of ray processing and tracing with an increase in the workstation's number of active CPU cores. The amount of RAM used when synchronous and asynchronous parallel computations are combined is a multiple of the number of NUMA nodes in the

Table 8 Summary of the test results for the workstation with the single Intel Xeon CPU and uniform memory access with utilization of all 12 computation cores.

	HUD system scene		Lens system scene	
Parallelization method	Tracing speedup	Achieved accuracy (%)	Tracing speedup	Achieved accuracy (%)
Synchronous	7.0	31.1	3.5	19
Asynchronous	11.7	34.2	11.2	20.4
Two-level	11.9	30.1	12.0	15.6

Table 9 Summary of the test results for the workstation with dual AMD EPYC CPUs and NUMA with utilization of all 32 computation cores.

Parallelization method	HUD system scene		Lens system scene	
	Tracing speedup	Achieved accuracy (%)	Tracing speedup	Achieved accuracy (%)
Synchronous	16.1	27	6.3	14.1
Asynchronous	32.3	28.2	27.1	15.9
Combination	27.3	23.5	26.8	13.2
Three-level	29.2	22	33.3	11.7

workstation. This, along with the fact that each synchronous group's threads are bound to a single NUMA node's cores, eliminates the need to access the memory of other NUMA nodes, except for the gathering of rendering results.

7 Conclusion

In the scope of the current study, the traditional parallel data processing methods for the backward photon mapping method were implemented and examined. The drawbacks of using the traditional synchronous and asynchronous parallel data processing approaches with backward photon mapping algorithm, including the synchronous method's poor scalability and the asynchronous method's excessive memory usage, were discussed. Their combination helped to mitigate their drawbacks, but the rendering scalability was still inferior to the straightforward asynchronous parallelization approach.

On workstations with uniform memory access, the designed two-level semi-synchronous backward photon mapping parallelization method showed good results, and ray-tracing speedup almost linearly depended on the number of used computation cores. At the same time, the memory usage was kept at the level comparable to the traditional synchronous parallelization method.

On workstations with NUMA, the designed three-level semi-synchronous backward photon mapping parallelization method showed promising results. It offered almost a linear scalability of completely asynchronous calculations and memory usage that was a multiple of the number of NUMA nodes. Due to the asynchronous nature of calculation on a third computation level, it allowed for a straightforward integration of the distributed rendering by utilizing remote server resources with little modification of an intermediate data fetching method.

Acknowledgments

This work was supported by the Russian Science Foundation, project no. 22-11-00145. The results shown in the current paper are partly based on previous research that the authors presented at a SPIE conference and published in SPIE Proceedings.[17]

References

1. I. S. Potemin et al., "An application of the virtual prototyping approach to design of VR, AR, and MR devices free from the Vergence-accommodation conflict," *Proc. SPIE* **10694**, 19–27 (2018).
2. J. T. Kajiya, "The rendering equation," *SIGGRAPH Comput. Graph.* **20**(4), 143–150 (1986).
3. H. W. Jensen, "Global illumination using photon maps," in *Eurograph. Workshop on Rendering Tech.*, pp. 21–30 (1996).
4. H. W. Jensen and P. Christensen, "High quality rendering using ray tracing and photon mapping," in *ACM SIGGRAPH 2007 Courses, SIGGRAPH '07*, pp. 1–es (2007).

5. E. Veach and L. J. Guibas, "Optimally combining sampling techniques for Monte Carlo rendering," in *Proc. 22nd Annu. Conf. Comput. Graph. and Interactive Tech., SIGGRAPH '95*, pp. 419–428 (1995).
6. D. D. Zhdanov et al., "Use of computer graphics methods for efficient stray light analysis in optical design," *Proc. SPIE* **10690**, 523–534 (2018).
7. A. Zhdanov and D. Zhdanov, "Progressive backward photon mapping," *Program. Comput. Softw.* **47**, 185–193 (2021).
8. S. Ershov and A. Voloboy, "Calculation of MIS weights for bidirectional path tracing with photon maps in presence of direct illumination," *Math. Montisnigri* **48**, 86–102 (2020).
9. G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *Proc., Spring Joint Comput. Conf., AFIPS '67 (Spring)*, 18–20 April, pp. 483–485 (1967).
10. J. Günther, I. Wald, and P. Slusallek, "Realtime caustics using distributed photon mapping," in *Eurograph. Workshop on Rendering*, A. Keller and H. W. Jensen, eds., pp. 111–121 (2004).
11. B. Nouanesengsy et al., "Revisiting parallel rendering for shared memory machines," in *Eurograph. Symp. Parallel Graph. and Visualization*, T. Kuhlen and R. Pajarola and K. Zhou, eds. (2011).
12. E. W. Bethel and M. Howison, "Multi-core and many-core shared-memory parallel raycasting volume rendering optimization and tuning," *Int. J. High Perform. Comput. Appl.* **26**(4), 399–412 (2012).
13. L. Bouganim, D. Florescu, and P. Valduriez, "Load balancing for parallel query execution on numa," *Multiprocessors Distrib. Parallel Databases* **7**, 99–121 (1999).
14. T. Ogasawara, "Numa-aware memory manager with dominant-thread-based copying GC," *Int. J. High Perform. Comput. Appl.* **44**(10), 0362–1340 (2009).
15. I. S. Potemin et al., "Hybrid ray tracing method for photorealistic image synthesis in head-up displays," *Proc. SPIE* **1690**, 75–86 (2018).
16. I. S. Potemin et al., "Analysis of the visual perception conflicts in designing mixed reality systems," *Proc. SPIE* **10815**, 181–194 (2018).
17. A. Zhdanov, D. Zhdanov, and M. Sorokin, "The virtual prototyping of complex optical systems on multiprocessor workstations," *Proc. SPIE* **11875**, 41–60 (2021).

Andrey Zhdanov is an associate professor at ITMO University. He received his BS and MS degrees in applied math and informatics from ITMO University in 2006 and 2008, respectively, and his PhD in computer systems from ITMO University in 2020. He is the author of more than 70 journal papers. His current research interests include realistic rendering, virtual prototyping, virtual and mixed reality, and optical systems design. He is a member of SPIE.

Dmitry Zhdanov is an associate professor at ITMO University. He received his specialist degree in optics from ITMO University in 1985 and his PhD in computer systems from the Keldysh Institute of Applied Mathematics of Russian Academy of Sciences in 2010. He is the author of more than 100 journal papers. His current research interests include realistic rendering, virtual prototyping, virtual and mixed reality, and optical systems design.

Maksim Sorokin is a senior researcher at ITMO University. He received his PhD in computer systems from ITMO University in 2021. He is the author of more than 20 papers in peer-reviewed scientific journals. His current research interests include virtual and mixed reality, deep learning, and computer vision.

Igor Potemin received his PhD in optical and optoelectronic instruments from Saint-Petersburg Electrotechnical University in 2015. He is an associate professor at ITMO University. Also, he cooperates with the KIAM and Vavilov State Optical Institute. His areas of scientific activity are photonics, optical design and testing, computer graphics, virtual prototyping, illumination optics design, and stray light analysis. He is the author of more than 50 scientific publications in the areas of photonics, optical design, and computer graphics.