

1. Introduction

In parallel computing systems computations are executed simultaneously, wholly or in part. This approach is based on the partitioning of a big task into smaller subtasks that can run in parallel. There are four basic forms of parallelization: bit-level, instruction-level, data and task levels. The first two belong to low-level programming. This article focuses on the higher —task and —data levels of parallelization:

- Data parallelism focuses on distributing the data across different parallel computing nodes. Data parallelism occurs when each node performs the same task on different pieces of distributed data. For example, the task of adding two matrices is suitable for data parallelization. Matrices are easily divided into sub-matrices that can be added by different processors. Two parallel processors will perform this task twice as fast than one.
- Task parallelism involves the distribution of tasks across different parallel computing nodes. This is achieved when each processor executes a different thread on the same or different data. The threads can execute the same or different code. Threads can communicate by passing data portions needed for next steps of the work-flow. For example, one processor can perform calculations and another one can validate and optimize the next portion of input values.

Most real programs operate somewhere between task and data parallelism. Our solution also lies in this set.

Parallel computing programs are more complicated than sequential ones. Concurrency among threads creates conditions for software bugs. Competition for computing resources can create bottlenecks for threads trying to access the same resource (for example, a results file). Therefore, synchronization and communication between threads is important for achieving effective performance improvement.

Competent implementation of parallel computing improves software effectiveness and conserves computational resources. However, before starting

parallelization of sequential software, it is advisable to evaluate the maximum expected speedup of a program. In accordance with Amdahl's law, if the proportion of code that can be parallelized is not significant, then increasing the number of processors will only provide a slight speed boost, at a high cost of data exchange.

Now that the cost of parallelization and the complexity of its implementation have significantly decreased, modern software and hardware commonly use parallel computing methods. Software and hardware providers have made it possible to use the power of existing IT methods to create a parallel computing environment with off-the-shelf servers and high-speed interconnections. These systems provide computing power for significantly lower costs of entry and ownership. Microsoft has deployed two powerful services for parallel computing: the Windows Azure platform and HPC cluster (to be discussed in detail later). The main purpose of our project was to explore ways to parallelize existing open-source software and adapt it to these two Microsoft parallel computing services.

For this project we chose LuxRender, an open-source software solution for rendering realistic 3D-scenes with photographic quality. Its engine is based on a state-of-the-art algorithm of ray-tracing, which has great potential for parallelization. LuxRender makes the rendering process far more efficient and productive even for users who are not professional 3D scene modelers. Its network mode enables simultaneous deployment of many computers (IP addresses required) to render 3D scenes, providing higher performance.

We selected LuxRender because it is open-source, and adaptation for Azure and HPC requires changes in the source code. Also, rendering tasks are suitable for visual evaluation; even a novice user can identify changes in performance and detect errors.

2. Windows Azure

a. Platform Overview

Windows Azure is a platform developed by Microsoft for applications deployment and data storage that utilizes the —cloud computing concept. —Cloud refers to a set of software and hardware resources available to a client via an internet connection. End users need to know nothing about technical implementation of the cloud. Data and applications stored in the Windows Azure cloud platform are highly reliable, available in 99.99% of all requests.

The key functions of Windows Azure are:

- Windows Azure – data storage and computing;
- SQL Azure – relational databases support;
- Windows Azure AppFabric – access control and communications support (Service Bus);
- Windows Azure Marketplace – service for purchasing applications and data.

Windows Azure enables users to rent computing resources, instead of purchasing hardware and operational systems. Customers can easily increase or decrease the capacity of —parked (powered down) virtual machines. This approach is highly flexible and commercially profitable because it:

- eliminates the cost of purchasing and installing equipment;
- reduces time for software deployment;
- eliminates need for manual update of installed software;
- provides the latest virus protection;
- reduces power consumption;
- saves work space.

Windows Azure Storage service utilizes binary large objects, or —blobs. A blob is a file that contains any kind of information — photos, videos, spreadsheets, text documents, etc. Parallel rendering uses project blobs to store both image files and files containing intermediate results.

Windows Azure supports the following roles:

- Web role – provides the web interface for the Azure application;
- Worker role – runs the tasks initiated by the web role;
- VM (virtual machine) – runs applications with specific claims to the environment.

Parallel rendering requires two types of roles, both of which are supported by Windows Azure: Web roles (front-end) and Worker roles (middle layer). In our application, Worker roles are used for both rendering and the merging of rendering results.

The Windows Azure table utilizes a structured data storage mechanism. The Windows Azure queue is a mechanism for asynchronous message delivery from one role to another. In our project, queues are used to deliver messages from the Web role to the rendering Worker roles and from the rendering role to the Merger role.

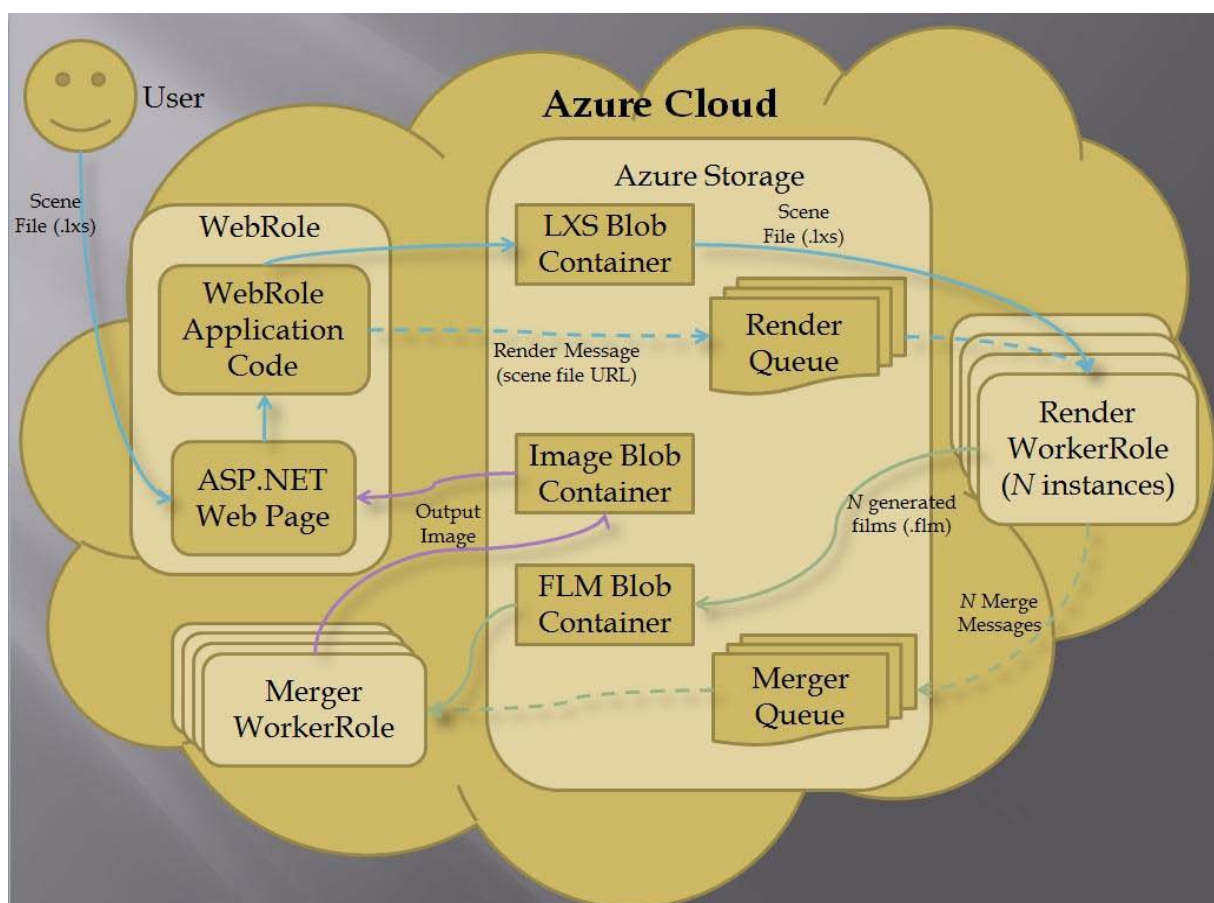
Microsoft Visual Studio 2008 provides a broad set of tools for the development and deployment of Windows Azure applications. Our project was implemented under Visual Studio on C# language. Open-source rendering functionality on C++ was compiled into a dynamic link library and attached to the project.

Details about Windows Azure are available at:
<http://www.microsoft.com/windowsazure/whitepapers>.

b. Adapting LuxRender for Windows Azure

In our implementation of parallel rendering on the Windows Azure cloud, we utilized a feature of LuxRender that saves a current state of the rendering process. The process can later be resumed. This means that independent processes can run separately, saved to FLM files, and subsequently gathered to a single machine for merging. Performance remains unaffected; it is the same as in network mode.

Our concept is based on creating a number of rendering Worker role instances which render the scene in parallel, and a merging Worker role for collecting the FLM files and producing the resulting image.



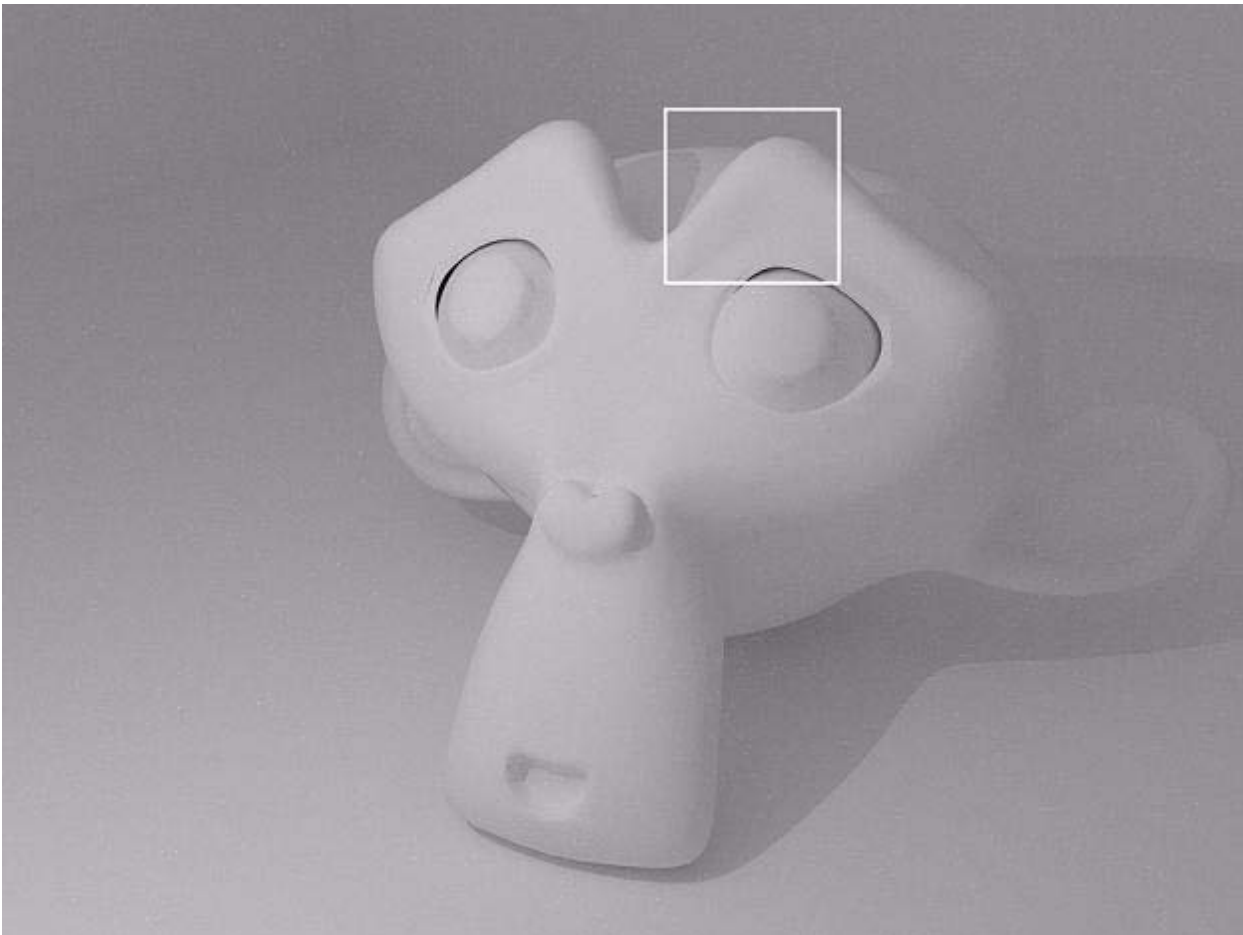
LuxRender functionality can be used inside applications with the help of LuxRender API along with the source code. The main problem we confronted using LuxRender in Azure was the absence of a file system on the cloud. Despite being a cross-platform application, LuxRender stores all its files (FLMs, temporary files) in a folder which contains the scene file. Azure stores files in blobs, which are accessible by HTTP. In order to solve this issue, we had to modify the source code for LuxRender to load and save files in Azure blob storage.

In our implementation of cloud rendering, every rendering Worker role runs an independent process and stores its results in blob storage. The merger Worker roles process FLM files from an FLM blob container in the order that they are received, and generate the resulting image. The Web role is responsible for loading the originating scene file into blob storage. All roles use LuxRender API functions. The core of LuxRender was compiled into a dynamic-link library (DLL) and linked to the Azure application.

c. Test Results: Local and Cloud Rendering

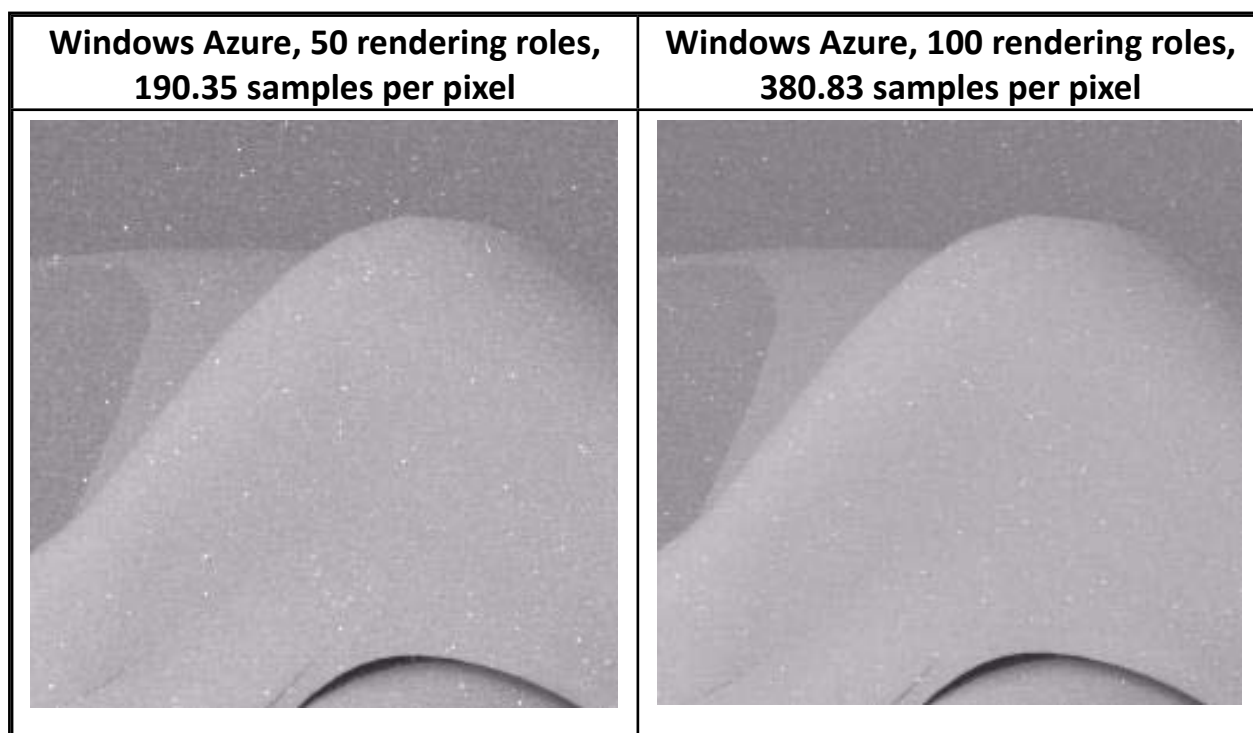
We ran the rendering process for a single scene four times: once on local PC, and three times on Azure Cloud. Each measurement on Windows Azure was executed with a different number of rendering roles, deployed on the Windows Azure Extra Small compute instances (CPU 1.0 GHz, Memory 768 MB, Instance Storage 20 GB). The allotted rendering time was the same for all measurements (one hour). To compare performance, we used the ratio of the number of generated ray traces (samples) to the number of pixels in the resulting image (S/px). The higher the ratio, the higher the quality of the resulting image.

Below is the image obtained after rendering on 100 Extra Small Windows Azure compute instances (image scale was reduced to fit the page; its real size is 2000x1500). On the next page is a table with four image fragments of real size (no zoom). These fragments contain the area outlined within the white border on the image below. These fragments are taken from the source images obtained after measurement. The first demonstrates the quality level after rendering on a local PC, and the three others reflect the quality level obtained from Windows Azure Cloud with 25, 50 and 100 rendering instances, respectively.

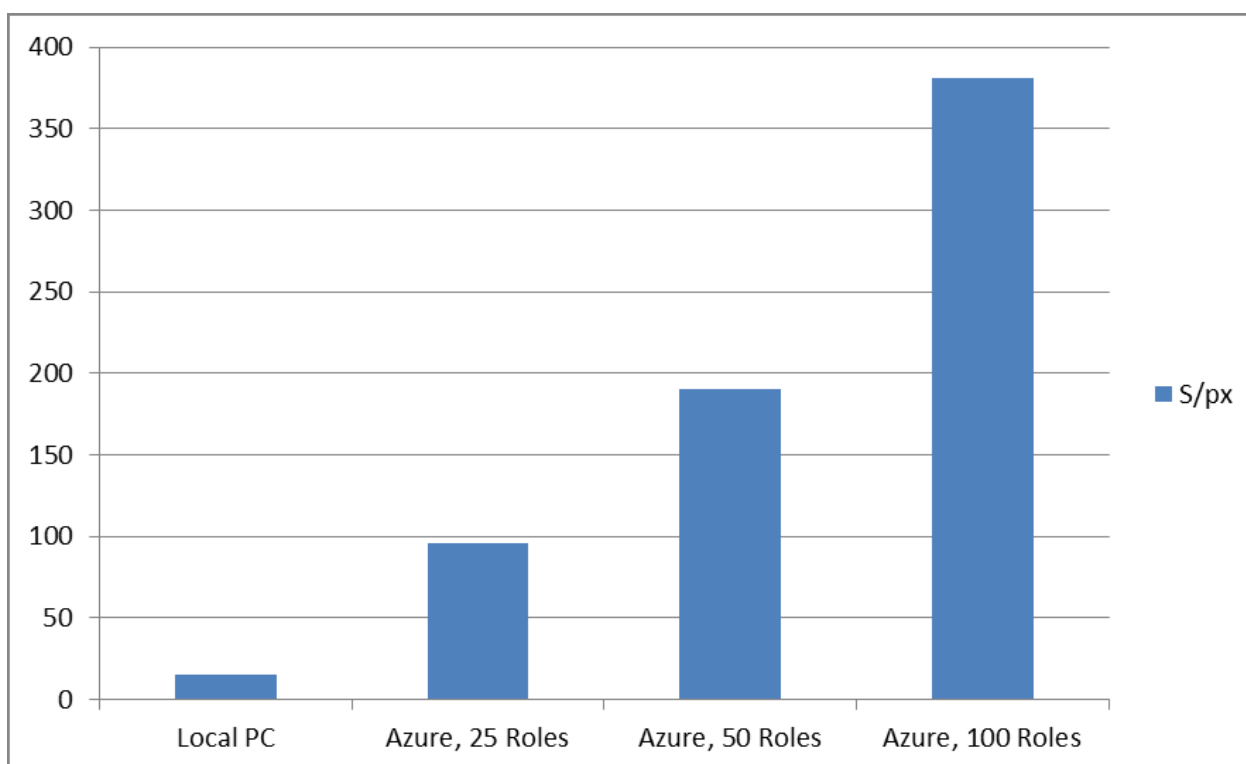


Fragments of resulting images obtained after rendering on local PC and Windows Azure Cloud.

Local PC, 15.40 samples per pixel	Windows Azure, 25 rendering roles, 95.67 samples per pixel



The following chart reflects the quality level of the four measurements.



Vertical axis (quality level – ratio of the number of generated ray traces (samples) to the number of pixels in the resulting image, S/px):

1. 15.40;
2. 95.67;
3. 190.35;
4. 380.83.

As the chart demonstrates, the number of samples per pixel generated on Windows Azure depends directly on the number of rendering cores. Because the local PC used in this test is much more powerful than the Azure Extra Small compute instance, the difference between results from the local PC and the 25 Extra Small Windows Azure compute instances is not very large. It should also be noted that the improvement in visual quality does not continue to increase as a direct function of the ratio S/px , but slows (i.e., the vertical axis values do not directly reflect the perceived visual quality). This is caused by peculiarities in the LuxRender algorithm: random generation inevitably creates overlaps in ray traces. This leads to a reduced rate of improvement in visual quality as the number of renderings (or rendering time) increases.

3. HPC Cluster

a. Platform Overview

A cluster is a group of computers connected via a network, working together like a single supercomputer. The cluster nodes are interconnected through fast local area networks. Clusters are usually deployed to achieve high performance for low price. Costs for the purchase and installation of cluster components are lower than the cost of a supercomputer of similar power.

Clusters are often utilized to perform tasks that can be divided effectively into low-level dependent subtasks. The results of these subtasks are then merged into a general solution. For example, calculating a function for which each step value does not depend on the previous ones ($f(t) \neq g(f(t-1))$), where $f(t), g(t)$ can be done in parallel on cluster nodes.

There are four types of clusters:

- High-availability clusters (systems with redundant servers to replace failed nodes on the fly);
- Load balancing clusters (systems with dynamic load distribution);
- Computing clusters (used for computational purposes);

- Grid-systems (distributed networks of loosely coupled computers).

Computing clusters are used for calculation purposes: scientific research, engineering and economic calculations. Clusters of this type require high performance for operations with floating points (flops) and high-speed network connections. The calculation time on this kind of cluster is significantly faster than on a single PC. This is especially true for tasks that can be divided into smaller subtasks; for example, calculations with matrices.

Microsoft Windows HPC Server 2008 is an operating system for high-performance computing clusters. This system is well scalable to large number of nodes, has service-oriented architecture, and provides useful features for task scheduling and monitoring. Each node in an HPC cluster performs one or more tasks. Cluster nodes can take the roles of a head node, compute node, scheduler or broker. Compared with specialized supercomputers, an HPC cluster provides similar computing power for lower cost. In addition, it can be quickly set up and is easy to maintain.

b. Adapting LuxRender for HPC Cluster

LuxRender's algorithm for ray casting is based on pseudo-random numbers. Parallel LuxRender tasks running on HPC cluster nodes generate their own separate files of results (FLMs). Merging all the FLMs generated independently on HPC cluster nodes creates the final image. Since image quality is directly proportional to the number of different rays cast, the picture quality obtained from this combined union of FLMs is much better than that obtained from each individual FLM file. As a result, the rendering time required to achieve a high quality image is reduced.

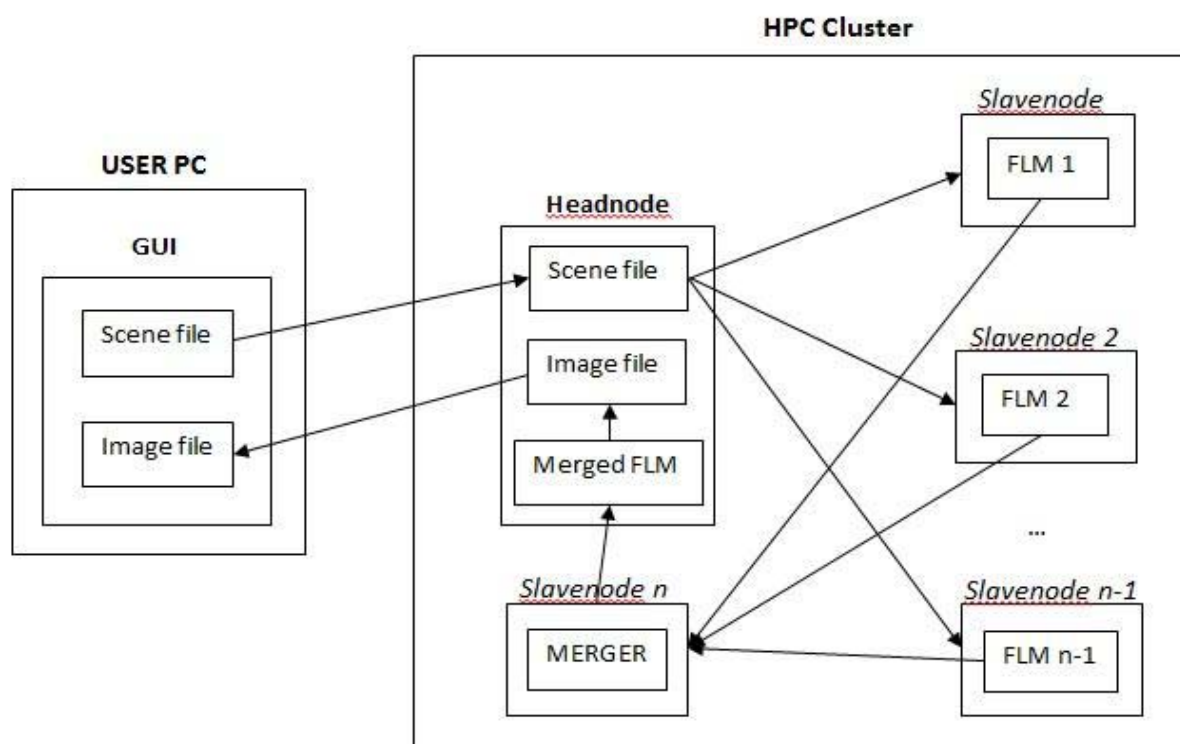
In implementing this adaptation we faced the following issues:

- Random number algorithm generation depends on the start time. Running rendering processes with the same start times simultaneously would lead to a set of files with identical rendering results. This problem can be solved in two ways: 1) delay the time between start of rendering tasks; 2) change the random base (replace dependence on start time with dependence on some other value). We used the delayed start method, which eliminated the need to invent our own random number generator.
- The name of the output FLM file is declared in the scene file. This means that all rendering processes that use the same scene will

rewrite each others' work from scratch when saving the results to the project folder on the head node. To resolve this issue, we implemented generation of unique file names.

- The LuxMerger utility merges FLM files using file names passed via command line. The file names are selected from project folder by the mask: `—*.flm` . This FLM internal LuxRender format is used for storing information about traced rays and the resulting pixels. We extended LuxMerger with a format converter from FLM into PNG (image).

Our approach is illustrated in the diagram below.



1. User submits scene file to HPC Cluster head node (via GUI*).
2. Rendering and merging tasks are started on cluster nodes via GUI. The head node can also run rendering or merging tasks.
3. The merger task checks for updated files to be merged. It then merges them and converts the result to an image file.
4. The resulting image file is updated periodically. The user can force-quit the rendering process or wait until the rendering time runs out. The image file containing the current results will be saved in both cases.

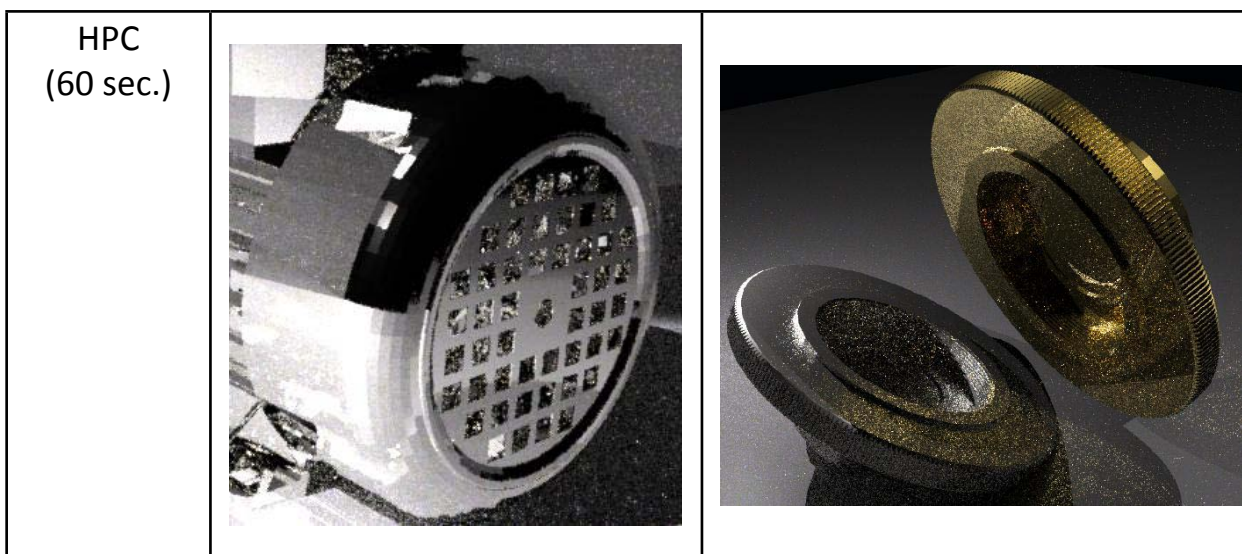
* We implemented a simple GUI to interact with LuxRender tasks on the HPC cluster.

c. Test Results: LuxRender on HPC cluster with local and network modes

To compare the performance of LuxRender on an HPC cluster with local and network modes, we again used the ratio of the number of generated ray traces (samples) to the number of pixels in the resulting image (S/px). The higher the ratio, the higher the quality of the resulting image.

Scene-file	working time (sec.)	Local (2 cores) samples/pixel	net-mode(2x2) samples/pixel	HPC (2x4) samples/pixel
motor.lxs	60	1.09544	3.42477	14.48688
	120	2.12954	7.82698	34.64708
magnifier.lxs	60	0.16214	0.32299	3.19696
	120	0.29637	1.50787	6.438

	motor.lxs	magnifier.lxs
Local mode (60 sec.)		
Network mode (60 sec.)		



As the data demonstrates, the number of samples per pixel generated on the HPC cluster is significantly higher than in local and network modes. As with the previous test, this corresponds to a much higher quality image.

4. Summary Discussion

Based on the tests results, it is clear that increasing the number of rendering processes reduces the time needed to achieve high image quality. As a result, users with access to a large quantity of computational resources can significantly speed up rendering time for LuxRender or similar rendering software. The parallel processing techniques described here demonstrate how:

- HPC Cluster owners can use it for rendering, with the resulting savings in payable rendering software;
- Windows Azure platform users can rent the computational resources to build their own rendering farm, with the resulting savings in equipment, as well as in payable rendering software.

Both projects are demo versions. Adopting them for use in practice requires additional changes, such as the development of converters from popular 3D modeling software into LuxRender format, etc.