GPGPU for Cheaper 3D MMO Servers

ASAVEI VICTOR, MOLDOVEANU ALIN DRAGOS BOGDAN, MOLDOVEANU FLORICA, MORAR ANCA, EGNER ALEXANDRU
Faculty of Automatic Control and Computers
University "POLITEHNICA" of Bucharest
Splaiul Independentei 313, Bucuresti
ROMANIA
victor.asavei@cs.pub.ro, alin.moldoveanu@cs.pub.ro, florica.moldoveanu@cs.pub.ro, anca.morar@cs.pub.ro, alexandru.egner@cs.pub.ro http://csite.cs.pub.ro

Abstract: Massive Multiplayer Online (MMO) applications have become extremely popular in the last years and this has led to an increase in the numbers of users that 3D MMO servers need to cope with. The current Client-Server architecture that is used by the majority of MMOs introduces a severe bottleneck regarding the performance and scalability of the virtual spaces. In order to achieve the necessary level of performance for the 3D MMO servers the operators of such virtual worlds are faced with a high financial cost to maintain the system infrastructure. In this paper we describe the challenges that 3D MMO servers face, analyze the general architecture of 3D MMO servers, identify key operation that can be optimized as GPGPU (General Purpose computation on Graphical Processing Units) programs and propose adaptations for the server architecture to run GPGPU tasks. The tests that we have conducted using a prototype implementation have shown encouraging results proving that it is feasible for a 3D MMO server to offload tasks as GPGPU programs and thus reducing the overall costs.

Key-Words: GPGPU, Massive Multiplayer Online, CUDA, Virtual Worlds, Parallel processing

1 Introduction

Massive Multiplayer Online (MMO) games and also other types of virtual spaces such as virtual museums, virtual expositions, etc, are already attracting a great number of users. The technology and the number of these applications are in a continuous evolution and it is likely that in the near future, 3D virtual spaces will become the standard for communication over the Internet, replacing or integrating current technologies (web browsers, instant messengers, etc).

MMO applications are deployed online over the Internet and have as support a persistent virtual world which is accessed at the same time by hundreds or even thousands of users.

A **persistent** world is an online virtual world that continues to "function" even when the user is disconnected and it is not involved anymore in the events of the world. The user can reconnect back at any moment of time and resume his interaction from the point where he left the world.

One of the main goals of a 3D MMO application is to create a high degree of immersion for the users of the virtual space. Many MMOs implement modern 3D real-time graphics engines, stereo and spatial audio creating a very interactive and immersive virtual space. The main advantage and feature that a MMO provides is that it accommodates and allows the interaction of a great number of users at the same time and this makes MMOs different from other single-user / multiplayer applications that simulate a virtual world.

However, providing this feature also rises the main problem that all 3D MMO virtual spaces must confront and that is the huge load that the virtual world must be able to cope with.

This load is given not only by the sheer number of tasks that is directly proportional with the number of online users but also by the complexity of the tasks that is affected very much by the particularities of the virtual world.

In this paper we first describe the challenges that 3D MMO Servers must overcome, give an overview of a general 3D MMO Server Architecture, identify the operations that can benefit from a GPGPU implementation and propose adaptations for the server architecture in order to support GPGPU.

2 Challenges for 3D MMO Servers

Although some necessary requirements in order to use a MMO application in optimal circumstances such as high bandwidth, low latency, are becoming more and more accessible for the regular user, this is not sufficient to solve the scalability problems that MMOs face in their attempt to accommodate as many users as possible.

The scalability problems are mainly due to the necessity to maintain the consistency of the virtual world. Breaking the consistency requirements can lead for

example to visual artifacts (e.g. : the avatar of an user entering a building wall) that don't have long term consequences but also can cause more serious problems such as the loss or duplication of object during financial transactions.

Traditionally, the majority of MMO applications are implemented using a *client-server* architecture that has the following main advantages :

- Centralized control
- Increased security
- Relatively simple implementation

Although this architecture, that is widely spread, is appropriate for many distributed applications when talking about MMOs it also introduces the following disadvantages / challenges :

1. **Scalability** : The performance of the central system of servers that simulates the virtual world may represent a bottleneck and thus imposing a superior limit for the total number of users that can access a virtual world
2. **Redundancy :** To make sure that a certain server can handle periods of peak usage, it is necessary to provide a high degree of hardware redundancy for the equipment
3. **Reliability :** This architecture does not have a high degree of reliability because the servers represent possible failure / break-down nodes of the system
4. **Cost** : Usually, the development of a regular MMO application takes 2-3 years and the initial cost for the launch is a high one. Another aspect to consider is that MMO applications need to simulate virtual worlds that have a large geographical area and this leads to the employment of a large number of servers (sometimes even hundreds) to fully host the virtual medium and the facilities that it must provide. This means, that after the launch of a MMO, the maintenance costs get as high as 80% of the total income [4].

Taking into account all the challenges mentioned, it is obvious why there is a necessity to explore new ideas and solutions for the architectures used by 3D MMOs and also to find new methods to optimize the operations used inside the simulations of the virtual world in order to eliminate or at least to reduce to a certain degree the problems that current MMOs face.

3 General 3D MMO Server Architecture

As mentioned previously, an important aspect when designing the architecture is that MMOs must simulate virtual worlds that have large physical span. This has lead to the development of methods to split the virtual space into several distinct sub-spaces.

3.1 Zoning

Currently there are two different approaches to split the virtual world into sub-spaces. The simplest, from the implementation viewpoint, is to strictly partition the entire world into static zones that are small enough to be managed by a single server.

The borders for these zones are very well determined, logically connected and an user that crosses into another zone will also connect to the server that manages the destination zone.

There are obvious disadvantages for this approach, all of them leading to the decrease of realism:

- there is a loading time when an user switches zones
- lack of possibility to see inter-zone objects

The second approach has from the start the goal to provide a "seamless" simulation where the existence of borders and server zones are transparent to the user. This will create for the user the sensation that he is in a "continuous" world that functions without "interruptions".

Although it is not explicitly visible to the user, the logical separation in zones for the virtual world still exists. Like the previous approach, important neighboring zones are managed by different servers. When a user exits from the zone that is managed by zone 1 server, there will be a transition zone that is synchronized with the adjacent zones. The user management take over by the zone 2 server will not happen immediately but only when the user will cross the transition zone.

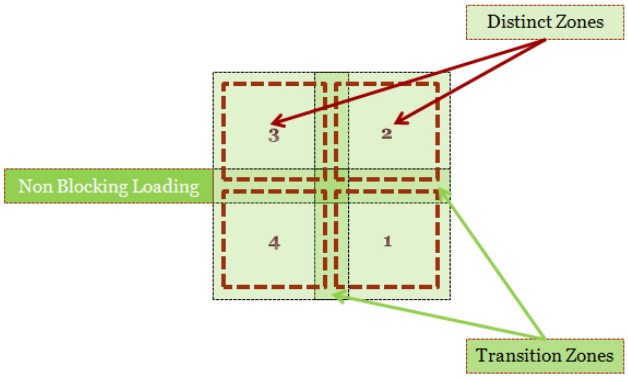

Fig.1 : "Seamless" zoning for a virtual world

The disadvantages for this approach are the complexity , from the implementation viewpoint, and also the possibility of an increased data traffic when there is a high number of events in the transition areas, both

leading to a high number of operations that need to be done by the zone servers.

3.2 Basic functionalities

The architecture for a MMO application needs to provide the following basic functionalities/features:

- User authentication/accounting
- Manage access rights for the users inside the virtual world
- Virtual world consistency
- Event order management
- Dispatch events to the entities of the virtual world
- Secure storage of the characters of the users and their possessions
- Schedule the computational operations
- Low latency
- Security for the virtual world preventing eventual fraud attempts

We can identify the following main components for a general 3D MMO server architecture :

1) **Authentication component** : it is responsible with access control for the virtual space
2) **Communication component** : it is responsible with the message exchange and events management. The main tasks for this component are to maintain the correct order of events at the level of the entire virtual space, to achieve a latency as low as possible and to provide security mechanisms in order to prevent cheating attempts
3) **Storage component** : provides the long-term persistent storage of the virtual world data
4) **Computational component** : schedules and execution of the computational operations that are linked to the virtual space logic.
5) **Control component :** this component provides the high level decision mechanism and has the role of managing, supervising and that of arbiter for all the operations executed by the other components of the system

3.3 Client-Server Model

Most of the MMO applications use a Client-Server architecture. The clients first access the virtual space through a *Connection Server* that redirects them to a *Shard Server*.

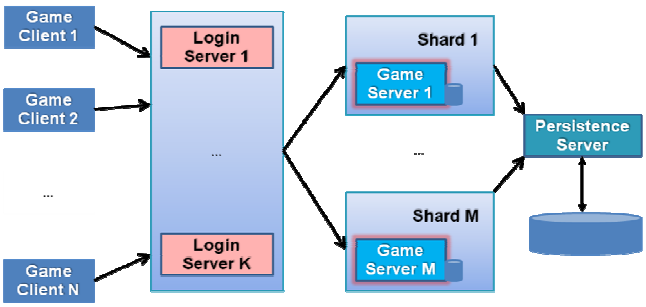

Fig.2 : Client-Server Shard Architecture for MMO Servers

The shards represent independent versions of the same virtual world and they are used to improve the scalability of the application. Usually, independent shards are not synchronized between them and the users of a shard cannot interact with the users of another shard.

Although this solution provides a certain degree of scalability for MMOs, it also limits the degree of realism preventing the interaction of a high number of users. There are some MMO applications that try to find new solutions that don't use shards, who split the virtual space into many independent worlds, and to allow the interaction of **all** the users inside of a single huge virtual simulation [6]. However this attempts are for the moment very strongly dependent of the particularities of the virtual world.

4 Computations that can be moved as GPGPU

The quantity of information and data that a MMO application needs to process is growing constantly. Usually, a high profile MMO monitors and executes a number of tasks that account for more than 1 million operations per second in periods of peak usage [3].

It is clear that scalability for the number of events processed is a necessity for MMO applications.

First generations of MMO event processing systems were centralized ones, but lately there have been solutions that try to achieve this scalability by using a distributed approach.

However, the partitioning of the processes proves to be very difficult due to the many dependencies that occur between the processes and also the agents that manage them.

Usually, the partitioning is done vertically, having a single set of local partition data that then uses a distribution mechanism that is centralized.

4.1 Low-level virtual space logic

The core of a MMO application is represented by a simulation loop that executes synchronized with a fixed frame rate.
From a conceptual viewpoint, the state of a virtual world inside a MMO can be seen as a data table that contains the "objects" of the virtual space including the users and the entities with which they interact.
To accommodate the strict demands for real time simulation, the active state of the virtual space is kept directly in the memory of the core servers of the virtual world. Physical disks are used usually for persistency and to store auxiliary data.
On each iteration of the simulation loop (usually called a *tick*), portions of this state are updated by the virtual world simulation logic. These updates can be triggered by user actions, timers or other virtual world events.

Following, we present some key operations that are done at the level of the simulation logic and are usually executed using a distributed mechanism and thus susceptible to optimizations using GPGPU.

4.1.1 Collision detection

Collision detections are one of the more frequent operations that are executed by the MMO virtual space servers and they require a high computational time.
The virtual space needs to assure the correctness of the physical interactions between the entities that populate the world and so it needs to execute a large number of collision detection computations.
Usually, at every movement / update of the position of a character inside the virtual space there are computation executed to assure that the movement / update is valid.
Because collision detection computations are very costly, in practice there is a limitation regarding the number and the entities that support collision detection (characters with the terrain, characters with certain buildings, etc).

4.1.2 Zoning operations

As explained previously, zoning is one of the most important mechanisms of a 3D MMO server architecture. The zoning system executes a large number of operations for the management of the users that are inside in a geographical area of the virtual world.
Also, another category of operations are executed to secure the persistency of the virtual space because a perfect synchronization is necessary between adjacent zones / transition zones and this has as effect an increased information and data flow.

4.1.3 Instancing

In a MMO application, an instance represents a special zone that creates a replica of itself, hence the name, for a certain group of users that want to access it. The group of users accesses the replica and thus is isolated from the rest of the users from the virtual space because the content of the instance they are in, is visible only in the context of those users.
Usually, there are dedicated servers for instances because they represent special areas that are not fit for a large number of users.
Some of the advantages of using instances are :

- Some of the operations that need to be processed by the virtual world servers are transferred to the instance servers
- Network traffic for the users is reduced, being practically isolated to events inside the instance and this leads to a decrease in latency
- Designing a user experience targeted specifically to fixed number of users
- Competition elimination , that can sometimes cause problems , on objects/resources of the virtual space

5 Adaptation to architecture to support GPGPU

GPGPU (General Purpose computation on Graphical Processing Units) is the method of using the graphical processing unit to process computations/programs that are normally executed by CPUs. This has been possible with the addition of programmable stages to the GPUs and with libraries and development toolkits from the vendors.

5.1 GPUs vs. CPUs

In the last years, the computing hardware , CPU and GPU, have evolved dramatically in terms of computing power and architecture. [5]
The modern GPU has transformed into a very versatile hardware using parallel multi-core architectures. These architectures that include GPUs, multi-core CPUs from Intel and AMD, CELL processors, SUN UltraSparc processors differentiate from the classical CPU architecture in the following way: they are designed to prioritize operations that can be executed in parallel over a large quantity of data compared to single task operations that have low latency.

Type	Processor	Cores/Chip	ALUs/Core
GPU	AMD Radeon HD 4870	10	80
	NVIDIA GeForce GTX 280	30	8
CPU	Intel Core2Quad	4	8
	Cell	8	4
	Sun UltraSparc	8	1

Fig. 3 : Modern CPU and GPU architectures

Although there are differences of implementations between vendors, all the modern GPUs try to remain as efficient as possible by using multi-core designs that use hardware multithreading and SIMD processing.
These techniques are not unique for the graphical processing units, but when compared with CPUs , the GPUs design take these architectures to the extreme.
As an example, the NVIDIA GeForce 280GTX has a total of 240 Streaming Processors that operate at a frequency of 1.3 GHz, and has a peak rate of 933 GFLOPS. In comparison, a high-end processor from Intel, a Core2Quad that operates at 3.0 GHz and has 4 cores has a peak rate of 96 GFLOPS [1].

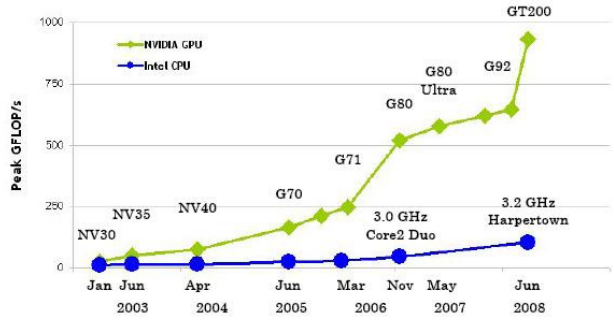

Fig.4: Comparison between NVidia GPUs and Intel CPUs. Source: NVidia CUDA SDK 2.0

5.2 CUDA

For our implementation we have used NVidia GPU hardware with CUDA development.
CUDA (Compute Unified Device Architecture) is a hardware and software platform for GPGPU programs that makes the graphical unit available as a parallel computation device for the developer without the need to use a graphics API.
CUDA uses three important abstractions:

- Hierarchy of thread groups
- Shared memory
- Barrier Synchronization

Using CUDA, it is possible to partition a computing problem into different sub-problems that can be processed independently in parallel.
Using this partitioning the group of threads can cooperate in solving each sub-problem achieving a natural scalability because computations for each sub-problem can be scheduled to run on any of the available streaming multiprocessors.

5.3 Implementation and results

For our solution we have decided to implement the following operations as GPGPU programs :

- Zone client computations using a spatial heuristic allocation on GPU for the users (number of users in a certain zone, workload balancing between zones)
- World Physics
  - Collision detection between entities
  - Computations regarding forces involved in the virtual world (friction, gravity, ground collision)

In order to accommodate the GPGPU computations in the MMO server architecture the following modules have been implemented and are used by the virtual world server :

- Client Allocation Module : this module is responsible with the heuristic allocation of the client on the GPU streaming multiprocessors
- Task Scheduler Module : this module is responsible with the creation of the CUDA tasks, scheduling the tasks and with the propagation of the results
- CUDA Processing Module : this module executes the tasks as GPGPU programs on the graphics hardware

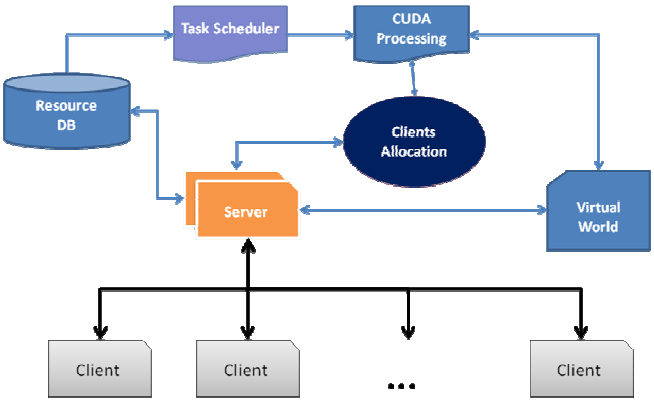

Fig. 5: Adaptation to the server architecture to run GPGPU tasks

Using the above prototype architecture and modules we have achieved encouraging results, running simulations that behaved with real-time response for a number of 4096 users on a single GPU NVidia card (9600M GT).

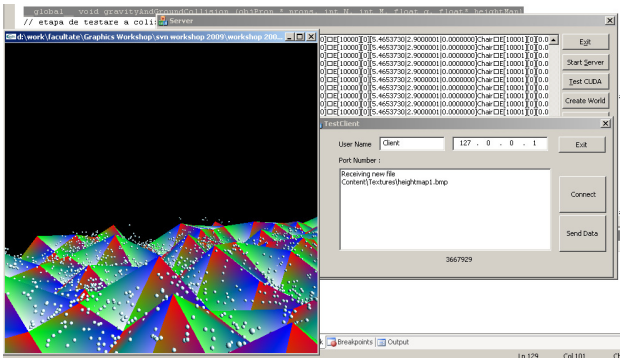

Fig. 6 : Server running GPGPU physics tasks for 4096 users simulated as spheres

6 Conclusion

The growing popularity of 3D MMO virtual spaces has lead also to an increase in the numbers of users that access such spaces. This has pushed to the extreme the load that the architecture of these spaces must face.

Currently, Client-Server architectures provide the necessary functionalities required by a MMO application but with a high cost that limits practically the scalability of the virtual space and thus being forced to split the world into several instances / shards.

In order to achieve a performance level that will satisfy a large number of users, the producer behind virtual spaces are confronted with a significant financial cost to maintain the infrastructure of the system

Another aspect to take into account is the fact that currently GPUs evolve at increased rate when compared with CPUs and regarding future scalability we can mention the following facts :

- Current workstations motherboards can accommodate up to 4 GPUs and this number could increase in the future
- Current NVidia GPUs have a maximum of 480 cores and this number will continue to grow in the future
- As a conclusion we can have now in a workstation up to 2000 programmable cores and the next generations of GPUs might push this number up to hundreds of thousands

Our approach has proved that it is feasible to offload heavy computational operations from the virtual world servers as GPGPU programs and so reducing the overall effort that is necessary to run 3D MMO applications.

References:

[1] Fathalian K., Houston M., A closer look at GPUs, *Communications of the ACM*, ACM Press, 2008

[2] Chris Chambers, Wu-chang Feng, Wu-chi Feng, Towards public server MMOs, *Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games*, 2006

[3] Geetika T. Lakshmanan, Yuri G. Rabinovich, Opher Etzion, A stratified approach for supporting high throughput event processing applications, *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems*, 2009

[4] J. Kesselman, Server Architectures for Massively Multiplayer Online Games, *In Session TS-1084, Javaone conference*, SUN, 2005

[5] Breitbart J., *Case studies on GPU usage and data structure design*, Dept. of Computer Science and Electrical Engineering, Universitat Kassel, 2008

[6] *Eve Online*
Available at: http://www.eveonline.com

[7] Leigh Achterbosch,Robyn Pierce,Gregory Simmons, Massively multiplayer online role-playing games: the past,present, and future, *Computers in Entertainment (CIE)*,Volume 5 Issue 4, 2008