Dissertation for Master's Degree in Informatics Engineering

An evaluation of the GAMA framework for heterogeneous platforms: the progressive photon mapping algorithm

Miguel Palhas
(Student)
pg19808@alunos.uminho.pt

Alberto Proença
(Advisor)
aproenca@di.uminho.pt

Luis Paulo Santos
(Co-Advisor)
psantos@di.uminho.pt

Braga, November 2012

Abstract

High performance computing has suffered several recent changes, with the evolution of Heterogeneous Platforms, where multiple devices with different architectures are able to share workload of increase performance. Several frameworks have been in development, to aid the programmer in using these platforms, by dynamically distributing workload. One of these is the GAMA framework, designed to deal with irregular applications, by unifying the multiple execution and memory models of each device into a single, hardware independent model. GAMA handles the workload distribution across multiple computational resources, and attempts to balance them based on measurements from previous executions, with the main goal of maximizing efficiency. So far, this framework has only been tested for small benchmark kernels, for comparison with similar alternatives using standardized algorithms, but still lacks a deeper study, using a more realistic case study, to better understand its limitations.

The subject of this dissertation is to provide a more in-depth evaluation of GAMA, using the progressive photon mapping algorithm as a case study. The goal is to validate its effectiveness with a full sized application, and identify points of improvement, not only with the analysis of the case study, but also via comparison with similar frameworks.

1. Context

Current high performance computing platforms are composed of different architectures working together to deliver a higher computational power. These platforms, usually called Heterogeneous Platforms (HetPlats), typically consist of one or more multi-core CPU, and other types of devices, such as massively parallel architectures like Graphical Processing Units (GPUs), or even more specialized architectures like Digital Signal Processors (DSPs), Field-Programmable Gate Arrays (FPGAs), or the recent Intel Many Integrated Architecture (MIC).

Using different devices to share the workload and achieve a higher performance can be a challenging task, since each platform has very specific characteristics, and usually requires specific programming and memory models to effectively take advantage of the specialized hardware. Algorithms often have to be adapted as well, or even replaced, to run efficiently on a specific platform. Input data may also play a role, as the same algorithm applied to different inputs may also have completely different behavior.

When dealing with a HetPlat, an additional concern arises, related to the memory space. In a HetPlat, each device usually has its own local memory space, and requires communication operations to transfer between the local memory and the global system memory. This means that the programmer must be careful not only to adapt the programming model to each device, but it also to be concerned about the

cost to move data back and forth, which can introduce an additional bottleneck in the execution, and even hide it completely.

Due to these issues, enabling an application to correctly take advantage of all the available resources of a HetPlat can become a challenging problem. A balance must be achieved between the tasks and the resources, assigning each available task to the platform that will most likely achieve better performance at any given moment.

To tackle this problem, several frameworks have been proposed which attempt to hide these difficulties. Some of these include StarPU [1], Harmony [2] and GAMA [3]. These frameworks attempt to address problems such as memory management, or the scheduling of the multiple tasks issued for execution.

Most of these frameworks however, provide mechanisms suited mostly for regular applications. When dealing with irregular applications, problems like scheduling become even more difficult, in particular because memory usage patterns are less predictable. This directly affects the decisions of the scheduler, and as such must be taken into account by the performance model employed by the framework, as explained in [4]. Irregular applications are one of the targets of the GAMA framework, which will be the main focus of this dissertation, and is explained more deeply in Section 2.

The GAMA framework is currently still in development, and supports only x86-64 processors and CUDA capable GPU's. Although it has been deeply tested for a wide variety of kernels in order to validate the correctness and efficiency of its memory and execution model, it currently lacks a more intensive usage, for a more robust application.

Small kernels have a wide range of applications, are deeply studied and optimized, and are a good source for an initial analysis on the performance results of the execution model. However, when considering a real, more resource intensive applications, where possibly multiple kernels must share the available resources, other problems may arise. Therefore, an evaluation of the framework with a real application, as opposed to the previously used synthetic benchmarks, is needed to understand its real effectiveness.

2. GAMA

The GPU And Multi-core Aware (GAMA) framework is a tool that provides the mechanisms required to efficiently execute data-parallel applications in a Heterogeneous Platform. It works as an abstraction layer above the hardware, employing an execution and memory model that allows the developer to remain agnostic of the available devices and other resources, focusing more on the algorithm itself, while also allowing lower level tuning of the algorithms for each specific device, if required.

The main goal of GAMA is not to provide a replacement to the current set of different programming models that are found on a HetPlat, but to create a layer above them to automatically deal with the more generic aspects such as task scheduling, memory management and coordination.

2.1. Execution Model

GAMA employs a task-based Unified Programming and Execution Model (UPEM) to create an abstraction of the different models of the underlying devices on a HetPlat. This model organizes available hardware using three concepts:

Computational Unit (CU) Name given to an individual processing element, capable of performing generic computations;

Device The group of all Computational Units with a common programming model and a common private memory address space;

Host The group of all Devices within a single computing node.

This organization of the execution model ensures that each device contains only CU's with the same memory address space, allowing for the usage of device-specific synchronization mechanisms to manage the coordination of concurrent executions within that device.

A basic implementation of an algorithm with GAMA requires at least two methods: a kernel and a dice. These two method together form what is known by GAMA as a job. The kernel is analogous to a

CUDA kernel, as it defines the code that will be executed in each thread. A dice method is a way to split the input domain across multiple tasks. It is somewhat analogous the ability of defining task granularity when using OpenMP, but it can employ more flexible solutions, to account for the irregularity of the algorithms.

2.2. Memory Model

While at the device level, synchronization is allowed via device-specific mechanics, that is not the case at the host level. When considering the whole host, or the entire group of available devices, a problem arises as each device possesses its own private memory address space, creating an obstacle to efficiently take advantage of the computational resources. Thus, a Global Memory System (GMS) is employed, using a relaxed consistency model requiring explicit memory fences to ensure full data consistency.

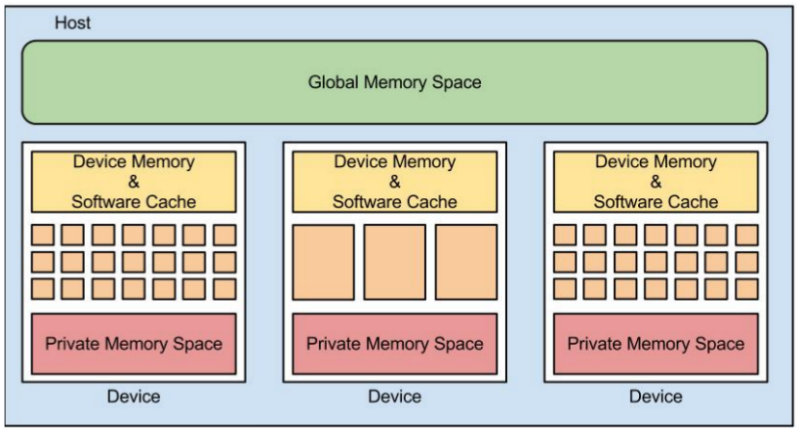

Figure 1: GAMA Memory and Execution Model

Figure 1 provides an overview of the memory model employed in GAMA. It is based on a hierarchy, composed of three levels:

- The first level is private and refers to a memory space addressable only by a single task
- The second level is shared between all Computational Units of a single device, and thus can be shared across tasks that are running on the same device. It is not addressable by other devices, however.
- Finally, the global level is addressable by the entire host system. However, due to implementation details of atomic operations being undefined, this level uses a relaxed consistency model, and requires explicit synchronization barriers to be used in order to ensure data consistency.

Besides the memory hierarchy, GAMA also employs a software cache to reduce latency in global memory space accesses and to exploit potential temporal and spatial locality. This cache is typically implemented over the private memory address space of each device, and can also operate as a pre-fetch mechanism, as it has the ability of copying memory blocks to the device in which they are required, prior to the execution of the corresponding task.

3. Progressive Photon Mapping

The realistic simulation of illumination of an environment is a complex problem. In theory, a simulation is truly realistic when it completely simulates, or closely approximates, the rendering equation [5]. This equation is based on the laws of conservation of energy, and describes the total amount of emitted light from any given point, based on incoming light and reflection distribution, and is described as follows:

In short, the equation defines the surface radiance $L_s(x, \Psi_r)$, leaving the point x in the direction Ψ_r This is given by L_e, which represents light emitted by a surface, and L_i, which is the radiance along the

$$L_s(x, \Psi_r) = L_e(x, \Psi_r) + \int_{\Omega} f_r(x, \Psi_i; \Psi_r) L_i(x, \Psi_i) cos\Theta_i d\omega_i \quad (1)$$

direction given by Ψ_i. f_r is the Bidirectional Reflectance Distribution Function (or BRDF) and Θ is the domain of incoming directions, which is given by a sphere centered on the point x.

Prior to Photon Mapping, typical approaches would include more than one method to approximate the rendering equation, such as Ray Tracing or Radiosity. Each method attempts to simulate the travel of light particles across the scene, and model the various interactions with the environment, but with different approaches and limitations.

Ray Tracing methods work by simulating light particles traveling from the eye into the scene, being reflected and refracted until they reach a light source.

Radiosity follows an opposite approach, and simulates the path light takes from the light source, until it reaches the eye. It only deals with diffuse interactions, however, and is commonly used in combination with other techniques.

Photon Mapping is another method of approximating the rendering equation [6], and works in a two pass way. In a first pass, a photon map is constructed by emitting photons from the light sources in the scene. This follows a method similar to Ray Tracing, but every interaction of a photon with the scene is stored in the photon map, creating a structure that roughly represents the entirety of light within the scene. In the second pass, the final image is rendered by using common Ray Tracing techniques (for example, Monte Carlo ray tracing). The photon map is then used to aid in the computation of the total radiance. The usage of the photon map is useful not only to increase performance, mostly by allowing a reduction in the number of samples to cast without affecting the final render, but also to allow the modeling of some light effects that are not present, or are inefficient, in other rendering methods, such as caustics or subsurface scattering.

Lastly, Progressive Photon Mapping is an extension to the previous method, that allows arbitrary accuracy not limited by memory [7]. In this method, a multi-pass algorithm is employed, where the first pass consists of a ray tracing method, and all subsequent passes use photon tracing to compute a photon map which will contribute to the global illumination.

The main advantage of this approach is that there is no actual need to store the entire photon map as it is created. Unlike standard photon mapping, it is possible to progressively arbitrarily increase the accuracy of the global illumination without being limited by the amount of memory.

Rendering methods such as photon mapping are a common example of resource demanding, irregular applications, due to the amount of information required to accurately describe a three-dimensional scene, and to realistically simulate all of its lighting effects. Therefore, it should serve a suitable case study for the GAMA framework

4. Objectives

The main goal of this dissertation is to perform a quantitative and qualitative analysis of the GAMA framework, applied to a large scale irregular algorithm, in order to validate its effectiveness, and identify possible soft-spots, especially when compared to other similar frameworks. The work will be focused on fully understanding GAMA, how to better take advantage of it, and possibly how to improve it even further.

This will be accomplished by using GAMA to implement a computational intensive, irregular application. The results will be used to evaluate the overall efficiency of the GAMA framework, applied to a large scale application, instead of only a single computational kernel. The selected test case for this dissertation was the Progressive Photon Mapping Algorithm, which was described in Section 3

A comparative analysis will be made between the developed implementation of the algorithm against existing implementations, to assert whether the GAMA framework does a good job at handling a resource intensive application like the progressive photon mapping algorithm.

The work will also be useful to understand how to better take advantage of GAMA, and where it should be improved. An analysis of the execution results of the application should provide insight about the way

GAMA is handling the existing jobs and data structures, and from there, identify possible bottlenecks or situations where an improvement could be possible. The comparative analysis will also be of extreme usefulness in this process, as it will make it possible to know whether or not different implementations perform better than the GAMA implementation, and why.

5. Methodology

In a first stage, the main focus will be in the research and study of the progressive photon mapping algorithm to better comprehend its details, and in experimental work with GAMA to fully understand the framework. Some time will also be spent studying existing implementations of the case study.

The second stage will focus on implementing the test case using GAMA. This is expected to be the most time consuming task, and a good understanding of the algorithm from the previous stage will be essential. Other implementations will also be studied, not only to assist in the development, but also to serve as a comparison basis for future analysis.

In the third stage, a deeper analysis will be done, which should be assisted by the previously gained knowledge. This stage will be essential to determine the effectiveness of GAMA, and to understand how to develop it further.

It will also be interesting, if time allows it, to study the possibilities and limitations of extending GAMA by providing support for additional architectures other than GPU's.

6. Timings

- **November 2012 to December 2012:**

  Literature search on Progressive Photon Mapping;

  Initial studies and hands-on of the GAMA framework.

- **January 2013 to March 2013:**

  Implementation of the Progressive Photon Mapping algorithm.

- **April 2013 to May 2013:**

  Comparative evaluation with other implementations;

  Quantitative and Qualitative analysis of GAMA.

- **June 2013**

  Dissertation writing.

References

[1] C. Augonnet, S. Thibault, R. Namyst, and P.A. Wacrenier, "Starpu: A unified platform for task scheduling on heterogeneous multicore architectures", *Concurrency and Computation: Practice and Experience*, vol. 23, no. 2, pp. 187–198, 2011.

[2] G.F. Diamos and S. Yalamanchili, "Harmony: an execution model and runtime for heterogeneous many core systems", in *Proceedings of the 17th international symposium on High performance distributed computing*. ACM, 2008, pp. 197–200.

[3] João Barbosa, "Gama framework: Hardware aware scheduling in heterogeneous environments", Tech. Rep., September 2012.

[4] Artur Mariano, Ricardo Alves, Joao Barbosa, Luis Paulo Santos, and Alberto Proenca, "A (ir)regularity-aware task scheduler for heterogeneous platforms", in *Proceedings of the 2nd International Conference on High Performance Computing*, Kiev, October 2012, pp. 45–56.

[5] J.T. Kajiya, "The rendering equation", *ACM SIGGRAPH Computer Graphics*, vol. 20, no. 4, pp. 143–150, 1986.

[6] H.W. Jensen, "Global illumination using photon maps", *Rendering Techniques*, vol. 96, pp. 21–30, 1996.

[7] T. Hachisuka, S. Ogaki, and H.W. Jensen, "Progressive photon mapping", in *ACM Transactions on Graphics (TOG)*. ACM, 2008, vol. 27, p. 130.

[8] A. Mariano, "Scheduling (ir)regular applications on heterogeneous platforms", Master thesis, University of Minho, Gualtar, 4715 Braga, September 2012.

[9] Ricardo Daniel Queirós Alves, "Distributed shared memory on heterogeneous cpus+gpus platforms", 2012.

[10] M.F. Cohen and J.R. Wallace, *Radiosity and realistic image synthesis*, Morgan Kaufmann, 1993.

[11] E. Johansson-Evegård, "Monte carlo path tracing".

[12] S.E. Chen, H.E. Rushmeier, G. Miller, and D. Turner, "A progressive multi-pass method for global illumination", *ACM SIGGRAPH Computer Graphics*, vol. 25, no. 4, pp. 165–174, 1991.

[13] P. Shirley, "A ray tracing method for illumination calculation in di use-specular scenes", in *Proceedings of Graphics Interface*, 1990, vol. 90, pp. 205–212.

[14] J.R. Wallace, M.F. Cohen, and D.P. Greenberg, *A two-pass solution to the rendering equation: A synthesis of ray tracing and radiosity methods*, vol. 21, ACM, 1987.

[15] C. Knaus and M. Zwicker, "Progressive photon mapping: A probabilistic approach", *ACM Transactions on Graphics (TOG)*, vol. 30, no. 3, pp. 25, 2011.

A. Signatures

(Student) Miguel Branco Palhas

(Advisor) Alberto Proença

(Co-Advisor) Luis Paulo Santos