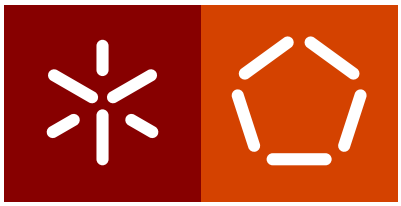
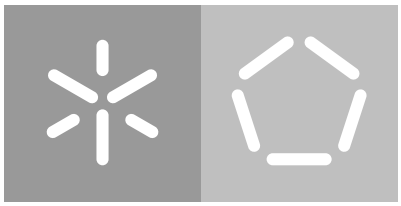


Universidade do Minho
Escola de Engenharia
Departamento de Informática

André David Gomes Monteiro Oliveira

Exploring Heterogeneous Computing with Advanced Path Tracing Algorithms

March 2017



Universidade do Minho
Escola de Engenharia
Departamento de Informática

André David Gomes Monteiro Oliveira

Exploring Heterogeneous Computing with Advanced Path Tracing Algorithms

MSc Dissertation
MSc Degree in Computer Science

Supervised by
Alberto José Proença
Luís Paulo Peixoto dos Santos

March 2017

ACKNOWLEDGEMENTS

To my supervisors, Alberto Proença, for this incredible opportunity and the outstanding rigor, dedication and encouragement, and Luís Paulo Santos, for asking the right questions with striking precision and relevance.

To César Perdigão and Roberto Ribeiro for the help navigating the immense sea of code, and other technical difficulties in the the lab.

To Janine Freitas for all the support and care, specially in this last difficult times, without her this dissertation would have not been possible.

To Albertina Gomes and Américo Oliveira, my parents, for always encouraging me to make my own choices and move forwards past their own interests, and to my sister, Catarina Oliveira, to whom I can only wish to be a small inspiration in life.

To the countless friends in this university, where I have grown and learned, for often being family and to the remaining family for always being friends.

ABSTRACT

Currently, most computing systems have access to more than one type of processing unit, typically a multicore CPU device and a computing accelerator, such as a GPU. However, the vast majority of the existing implementations of advanced path tracing algorithms only take advantage of one of these processing units. The implementation of these algorithms in such heterogeneous platforms while efficiently using both types of computing units already proved to provide improved performance results.

This dissertation examines four path tracing algorithms (Path Tracing *aka* PT, Bidirectional Path Tracing *aka* BPT, Bidirectional Photon Mapping *aka* BPM and Vertex Connection and Merging *aka* VCM) and extends previous work by exploring a richer heterogeneous environment with more GPU accelerators and with manycore x86 devices (i.e., Xeon Phi Knights Corner), complemented with an insight into the challenges introduced by each computing architecture and their programming environment. It also shows how these are combined together to perform heterogeneous computation managed by a simple scheduling algorithm, created to take advantage of each device's features.

This work proved that a fully heterogeneous approach to these four path tracing algorithms is feasible and the performance results are significantly improved.

RESUMO

Atualmente, muitos dos sistemas de computação conseguem tirar proveito de mais do que um tipo de processador (tipicamente o *multicore* e o GPU). Contudo, a maioria das implementações de algoritmos de *Path Tracing* avançados aproveitam apenas um destes processadores. A implementação destes algoritmos de *Path Tracing* em plataformas heterogéneas tem resultados comprovados que se mostram mais eficientes.

Esta dissertação analisa quatro algoritmos de *Path Tracing* avançados: o *Path Tracing* (PT), o *Bidirectional Path Tracing* (BPT), o *Bidirectional Photon Mapping* (BPM) e o *Vertex Connection and Merging* (VCM). Expande também o trabalho previamente desenvolvido explorando um ambiente heterogéneo mais rico, com mais GPUs e com *manycore*s (i.e., Xeon Phi Knights Corner), e apresenta os desafio que estas arquiteturas e os seus ambientes de programação podem trazer. Mostra ainda como estas contribuem em conjunto para o mesmo sistema heterogéneo com um simples algoritmo de escalonamento, implementado para tirar partido do melhor de cada arquitetura.

No final mostra-se que uma abordagem heterogénea para estes quatro algoritmos de *Path Tracing* avançado consegue ser viável e ainda trazer ganhos significativos de performance.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

B

BPM Bidirectional Photon Mapping.

BPT Bidirectional Path Tracing.

C

CG Computer graphics.

CPU Central Processing Unit.

CUDA Compute Unified Device Architecture.

D

DLP Data Level Parallelism.

F

FIFO First In First Out.

G

GPU Graphics Processing Unit.

H

HPC High-Performance Computing.

HT Hyper-Threading.

I

ICC Intel C Compiler.

ILP Instruction Level Parallelism.

K

KNC Knights Corner.

KNL Knights Landing.

M

MIC Many Integrated Core.

MIMD Multiple Instruction Multiple Data.

MIS Multiple Importance Sampling.

O

OS Operating System.

P

PCAM Partitioning-Communication-Agglomeration-Mapping.

PM Photon Mapping.

PPM Progressive Photon Mapping.

PT Path Tracing.

R

RMSE Root Mean Squared Error.

S

SIMD Single Instruction Multiple Data.

SMT Simultaneous multithreading.

STL Standard Template Library.

T

TBB Thread Building Blocks.

TLP Task Level Parallelism.

V

VCM Vertex Connection and Merging.

1

INTRODUCTION

One of the main challenges in *Computer graphics (CG)* is physically based rendering, the synthetic creation of images that are perceptually indistinguishable from real world views based on a geometric description of the scene, materials and light sources. There are several algorithms that try to solve this problem, although none of them is yet robust enough to handle every possible situation.

The current most popular advanced algorithms include *Bidirectional Path Tracing (BPT)* (Lafortune and Willems (1993) and Veach (1998)), *Bidirectional Photon Mapping (BPM)* (Vorba, 2011) and *Vertex Connection and Merging (VCM)* (Georgiev et al., 2012).

There are few implementations of these advanced algorithms which have been reported to work in computing systems built with both multicore *Central Processing Unit (CPU)* devices and computing accelerator units, such as *Graphics Processing Unit (GPU)* or Intel *Many Integrated Core (MIC)*, where both types of devices are simultaneously used in an efficient way. Previously, Perdigão (2015) dissertation work addressed some of them, namely Intel devices (CPU), with the Embree (Wald et al., 2014) and *Thread Building Blocks (TBB)* (Reinders, 2007) libraries, and the NVIDIA GPU, with OptiX (Parker et al., 2010).

This work delivered a version operating on both multicore and GPU devices. However, due to some limitations with the OptiX computation model, multiple GPU's could not be supported. This means that the use of more GPUs or other accelerators had not yet been evaluated.

Heterogeneous Computing can be defined as the parallel computation of a collection of devices with different characteristics. When a system contains several devices with different tradeoffs, and this devices are tuned for a specific situation, there is a better distribution of work that can be archived by sending the right kind of work to the right device.

Additionally the performance of the system will largely depend on the efficient exploitation of each device resources. Each architecture will have its strengths and requirements (eg., large *Single Instruction Multiple Data (SIMD)* registers, non-divergent code execution paths, etc.) in order to archive its ideal utilisation.

Developing on heterogeneous platforms, such that all different devices are effectively and efficiently used by the application, raises a number of challenges. Besides all issues commonly associated with parallel processing (e.g., workload decomposition, communication

overheads, load balancing), the heterogeneity of the different devices often implies maintaining different implementations for each architecture as well as dealing with separate memory address spaces. Frameworks, such as StarPU (Augonnet et al., 2011) and DICE (Barbosa et al., 2015) have been proposed to alleviate the application programmer from the challenges risen by heterogeneous systems.

1.1 MOTIVATION & GOALS

A recent MSc dissertation (Perdigão, 2015) implemented advanced physical based algorithms, but it only explored Embree for multicore devices and OptiX for a single NVIDIA GPU device, where the DICE framework takes care of workload scheduling among all computing devices/cores and data transfers between work-shared memories.

The main goals of this dissertation are:

- To expand the range of supported accelerators, namely the Intel Xeon Phi
- To analyse and propose solutions to overcome the limitations imposed by the use of OptiX when controlling more than one GPU
- To allow for all supported architectures to execute simultaneously

1.2 CONTRIBUTION OF THE WORK

This project expands the previous proposed solution with the addition of the MIC architecture. It additionally proposes a two level workload decomposition for the advanced *Path Tracing (PT)* algorithms, in order to allow the full exploitation of multiple heterogeneous parallel devices and assesses the performance and scalability of this solution on two highly parallel heterogeneous servers.

1.3 OUTLINE

This dissertation will be organised as follows. The next chapter (2) presents a background in computation light transport, namely a review of the various selected light transport algorithms with their characteristics and relevance in the research community. Chapter 3 gives insight into the main problems and challenges of dealing with heterogeneous computing ending with an overview of the computer hardware architectures used in the project. Chapter 4 explain the challenges of Heterogeneous Computing when applied to the Advanced Path Tracing Algorithms at different levels and describes the solution employed to archive the goals of this dissertation. Chapter 5 presents the performance analysis results of this

solution and it precisely defines the methodology used when studying this performance. And finally, the last chapter (6) summarises the conclusions and results of this dissertation, leaving suggestions for future work on relevant topics.

2

COMPUTATIONAL LIGHT TRANSPORT

The artificial creation of realistic images is one of the main challenges in CG. Generating images that can be perceptually indistinguishable from the real world involves many components that contribute to solving the problem of physically based rendering. A realistic definition of materials and lights, a sufficiently accurate geometric description of the scene and many other characteristics have to be studied and defined for the result to be convincing. The use of this information to simulate the propagation of light across a scene is called the light transportation problem.

2.1 PATH TRACING

By applying the rendering equation, Kajiya (1986) was one of the first to give a solution to the global illumination problem. This equation defining the problem of rendering as the resolution of an integral equation (1) represents how much light reaches a certain point given a direction. This equation is calculated through Monte Carlo Integration since it can not be fully solved analytically.

$$I(x, x') = g(x, x') \left[e(x, x') + \int_S \rho(x, x', x'') I(x', x'') dx'' \right] \tag{1}$$

where:

$I(x, x')$	is related to the intensity of light passing from point x' to point x
$g(x, x')$	is a "geometry" term
$e(x, x')$	is related to the intensity of emitted light from x' to x
$\rho(x, x', x'')$	is related to the intensity of light scattered from x'' to x by a patch of surface at x'

There are two ways this equation can be used: by tracing a light transport path from the light source to camera lens (forwards PT) or from the camera lens to the light source (backwards PT). By tracing the rays from the light sources, a lot of those rays would never reach the camera (bounced in an opposite direction, etc.) and most computation would not contribute to the actual image plane. Therefore, the second one (figure 1) is the more common method. For each sample on the image plane, a random walk is performed, from

the camera into the scene, passing through it, thus gathering radiance for every single point. The path starts with the primary ray and, at each intersection point, the next ray's direction is selected according to some probability density function. The path then finishes, using a Russian Roulette approach due to its unbiased nature.

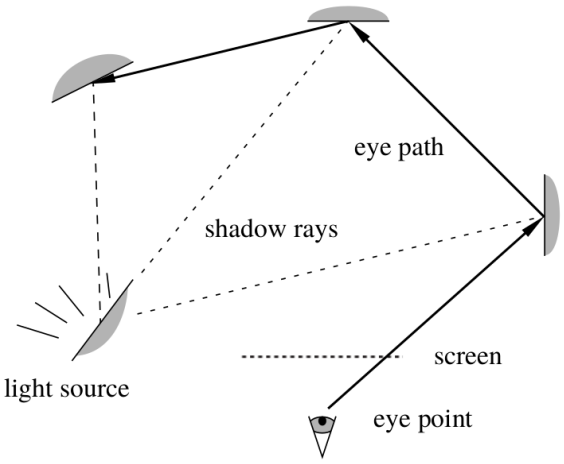


Figure 1: A path in the Path Tracing algorithm (Lafortune, 1996)

2.2 BIDIRECTIONAL PATH TRACING

This algorithm is the direct combination of both forward and backwards PT (figure 2). Suggested by Lafortune and Willems (1993) and Veach (1998) the combination of this techniques handles indirect light problems far more efficiently and robustly than simple PT.

It works by paring the tracing of a sub-path that originates from the light source with the one that originates from the camera. Both sub-paths are shot and perform their respective tracing just like the PT algorithm. A sub-path can then be connected to the intersection of the other sub-path in order to form more paths, that can add their contribution to the amount of light that reach a point in the camera, with less computational effort. Propagating radiance from the light sources allows for the efficient simulation of specular to diffuse light transport phenomena, such as caustics.

For a given path the final amount of contributed light might be obtained by either the light or the camera sub-paths, therefore Veach (1998) also introduces *Multiple Importance Sampling (MIS)* in order to control the weighting of the contribution of these sub-paths. MIS uses an heuristic to achieve a combination of all sampling techniques resulting in reduced variance in the sampling of values. This increases the convergence rate, hence the number of samples needed to obtain a image with better quality is reduced.

The heuristic is a simple and robust way to calculate these weights and is demonstrated (Veach, 1998) to be the almost optimal solution.

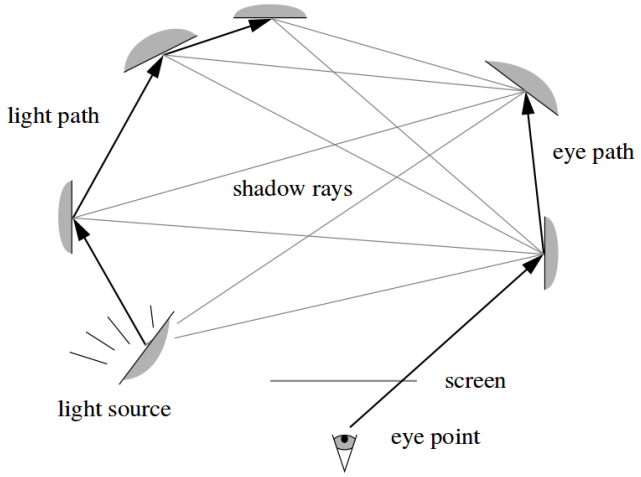


Figure 2: A path in the Bidirectional Path Tracing algorithm (Lafortune and Willems, 1993)

2.3 BIDIRECTIONAL PHOTON MAPPING

Photon Mapping (PM) works differently from PT algorithms, firstly because it will introduce some bias. This means that, as the image converges, the final result might not be the exact result of the rendering equation. It also uses a different approach by not using the tracing of rays going from lights to the camera.

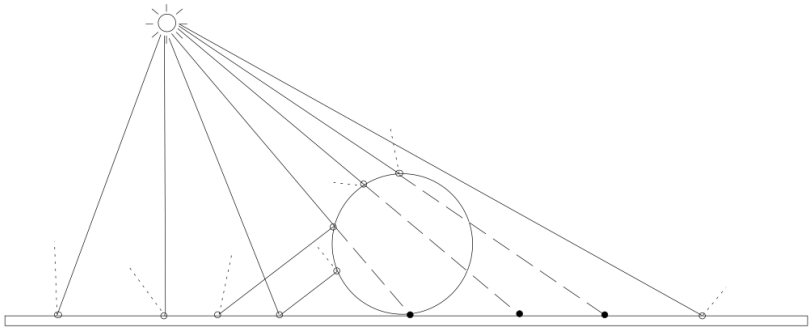


Figure 3: The photon tracing in the first phase of the Photon Mapping algorithm (Jensen, 1996)

As Jensen (1996) proposes this technique is divided into two phases. The photon map construction and the rendering (which consults the photon map).

The first phase (figure 3) is where the tracing of photons is performed. This means that every time a ray interacts with a non-specular surface the location and incoming direction of the hit is stored in a range search acceleration structure to facilitate spatial searching called a photon map.

This structure is then used in the second phase. By simply tracing rays from the camera, one can use the photon map to search for neighbouring photons and scan their density around each intersection to calculate the contribution amount.

With *Progressive Photon Mapping (PPM)* algorithm Hachisuka et al. (2008) reorganises the PM algorithm into a multi-pass (figure 4) algorithm. The first path, as opposed to storing the position of each photon in the photon map, stores all the hit points in an acceleration structure in order to find all the surfaces in the scene visible from the camera (ray tracing pass) through each pixel in the image.

Then, on consecutive photon tracing passes, a photon map is generated from the light sources. For every hit point in the acceleration structure the created photon map is sampled and the photon density is used to calculate the contribution to be added to the global estimation. At the end of each pass, once the contribution of the photons is recorded, the photons can be discarded and the search radius is decreased according to a constant parameter.

This reduction guarantees that in the limit the final estimate will converge to the correct solution of the rendering equation, turning the algorithm back to its unbiased form.

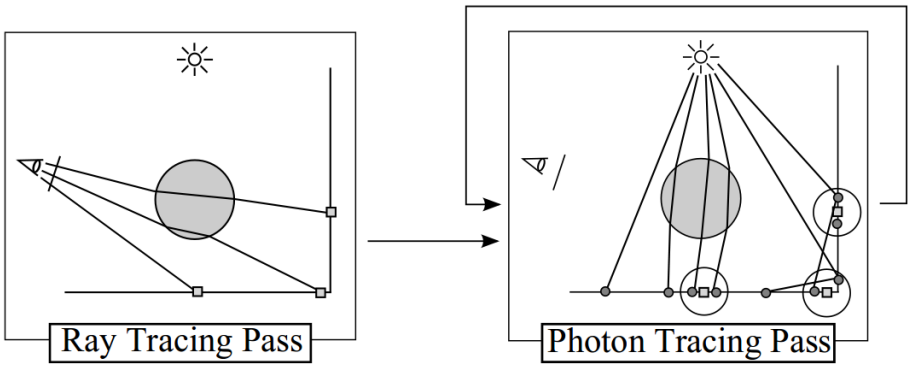


Figure 4: Phases of the Progressive Photon Mapping algorithm (Hachisuka et al., 2008)

As an improvement to the PM algorithm Vorba (2011) proposed the BPM where MIS is used in the same way as the BPT. This translates to two major differences. First, the probability of each generated photon is stored with it, and second, instead of only using simple tracing of primary rays, a path just like in PT is used where the photon map is consulted on every bounce. By reducing the search radius of the photon map, a progressive, unbiased algorithm can also be archived.

2.4 VERTEX CONNECTION AND MERGING

The VCM algorithm combines BPT and PPM through MIS to create an algorithm that is more robust than either of the algorithms alone. Georgiev et al. (2012) uses the advan-

tages of each algorithm, high convergence rate of the BPT and the quality of the caustics generated by PM, and incorporates it into VCM.

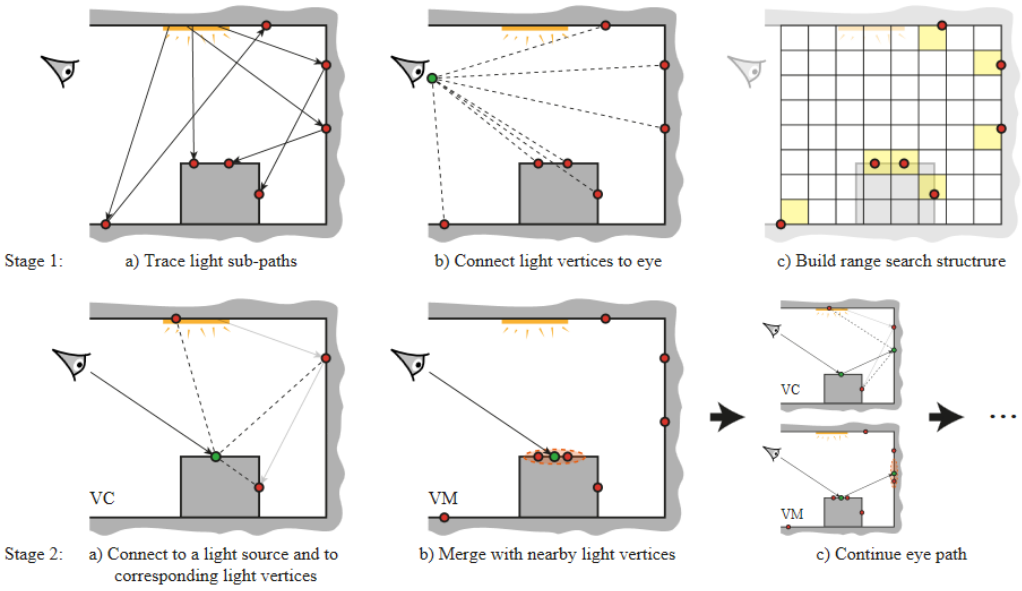


Figure 5: Phases of the Vertex Connection and Merging algorithm (Georgiev, 2012)

The algorithm is divided in two stages (figure 5). The first behaves like a PM algorithm would. It starts by tracing from the light source, connecting it to the camera lens and storing in an range search acceleration structure for every non-specular surface intersection.

On the second stage, it uses a BPT like technique, and it traces camera sub-paths across the scene, connecting it with the respective light sub-path (Vertex Connect). Then it performs a range search to determine if there is any neighbouring light sub-path viable for merging (Vertex Merging).

(a) PT — Kitchen

(b) PT — Living Room

(c) BPT — Kitchen

(d) BPT — Living Room

(e) BPM — Kitchen

(f) BPM — Living Room

(g) VCM — Kitchen

(h) VCM — Living Room

Figure 6: Comparison of advanced Path Tracing algorithms for two different scenes rendering for the same amount of time

3

CHALLENGES OF HETEROGENEOUS COMPUTING

In today's computing solutions there is an emerging trend towards Heterogeneous Parallel Computing. With an increasing demand for more computation power, new opportunities for parallelism can be unlocked with the use of Heterogeneous Computing.

The advantage of heterogeneous systems relies on the exploitation of different hardware capabilities to solve specific problems. This allows for work to be broken and distributed between devices according to their ability to handle the problems at hand. Algorithms that can take advantage of such characteristics have therefore great potential gains in exploring this style of parallelism.

However, there are great a number of challenges that go into making use of Heterogeneous Computing. Being aware of this challenges is crucial to guide the creation of new algorithms that can fully take advantage of the underlying heterogeneous system.

3.1 HETEROGENEITY

A clear understanding of the problem and the types of parallelism available within it is key to a successful exploitation of heterogeneous parallelism, but heterogeneity can exist in more that one form on computational systems. Sunderam and Geist (1999) defines that heterogeneity can occur in several different forms:

SYSTEM ARCHITECTURE SIMD, *Multiple Instruction Multiple Data (MIMD)*, scalar, and vector computers

MACHINE ARCHITECTURE Different instruction sets and/or data representation

MACHINE CONFIGURATIONS Differences such as clock speeds and memory

EXTERNAL INFLUENCES Dynamic variations in interconnection network capacity.

INTERCONNECTION NETWORKS Optical or electrical, local or wide-area, high or low speed, and may employ several different protocols

SOFTWARE Different underlying operating systems and/or programming models, languages, and support libraries

In order for a problem to take advantage of Heterogeneous Computing there needs to be some parallelism inherent to the problem. This parallelism must be well analysed and identified. The *Partitioning-Communication-Agglomeration-Mapping (PCAM)* design methodology (Foster, 1995) helps by defining four steps to approach the design of parallel algorithms.

3.1.1 *Work Decomposition*

The work decomposition can be determined by following the first two steps: partition and communication. At this level the hardware issues are ignored and the attention is in the conceptual division of work.

The first step is the partition of the problem into small tasks, in fact, the smallest possible (fine-grained decomposition). A problem can be divided by its data (domain decomposition) or its computation (functional decomposition).

Secondly, the communication needs between the tasks are determined. This will establish dependencies between tasks along with data and it is a crucial to the design of parallel algorithms since it is where the available parallelism will be restricted.

These two steps ensure that the opportunities for parallel execution are fully recognised. This kind of reasoning about the decomposition can only be obtained with good knowledge of the problem at hand.

3.1.2 *Scheduling*

The last two steps in the PCAM methodology are agglomeration and mapping. To make an efficient solution the fine-grained decomposition might not be the best, so the agglomeration can reduces the communication costs and task creation costs, where there might be overhead, with a bigger granularity.

The mapping step assigns the decomposed work to the actual hardware and is where the scheduling occurs. Typical homogeneous environments already have some form of scheduling in-device. But with heterogeneous systems there will be the need for extra scheduling between devices. Different scheduling policies, such as *First In First Out (FIFO)*, round-robin, shortest-job-first, and shortest-remaining-time, can be employed at each level of scheduling.

3.1.3 *Programming Environments*

Making all the different platforms and frameworks work together can already be hard due to such a wide range of possible hardware, all with different programming paradigms, development tools and levels of support.

This directly impacts code portability and maintainability and is often the main factor in dismissing such solutions to a wider audience.

3.1.4 *Interconnect Requirements*

Furthermore, devices need to communicate in order to co-operate. The exact amount and frequency of it is relative to each problem but generally devices have disjoint address spaces. This forces the communications to be made over a bus interconnect with limited bandwidth where programs that move a lot of data between devices will have a performance bottleneck. The careful distribution of work between devices is essential in order to obtain gains with heterogeneous architectures.

3.2 COMPUTER HARDWARE ARCHITECTURE

In all computing the performance of an application is highly bounded by the physical system where it is deployed. Each device has a specific design characteristics that might fit best with an algorithm or an implementation of it. Knowing this characteristics and having them in mind when writing a peace of software is crucial for making sure the hardware is utilised to its fullest potential. Further, in Heterogeneous Computing the device capabilities will determine the type and the amount of work computed by each device.

This project choose the conventional Intel CPU architecture along with two other computational accelerators, the NVidia GPU and the Intel MIC. Therefore each architecture capabilities and the kind of parallelism they can employ need to be evaluated.

3.2.1 *Multicore*

The multicore processor is the most accessible processor currently available. In these type of processors there are always more than one computation unit, coined as "core" by Intel. Intel multicore processor devices started with 2-cores in 2005 and on average the number of cores is growing by two every year.

Modern processor devices have a multicore architecture, multiple processing units (cores), and are able to explore the parallelism, either with *Instruction Level Parallelism (ILP)*, *Data Level Parallelism (DLP)* or *Task Level Parallelism (TLP)*.

Using ILP techniques such as out-of-order execution, instruction pipelining and superscalar execution, the performance of a single execution stream can greatly be improved inside a single core. With two or more execution streams TLP can be explored to map those streams to the increasing number of cores or in the same core with *Hyper-Threading (HT)*, Intel's proprietary *Simultaneous multithreading (SMT)* solution.

Intel® Xeon® CPU E5-2695 v2	
Codename	Ivy Bridge
Number of Cores	12
SMT	2 way
Max Turbo Frequency	3.20 GHz
Memory Size	64 GB
Max Memory Bandwidth	59.7 GB/s
L1 data cache	32K
L1 instruction cache	32K
L2 cache	256K
L3 cache	30720K (shared)
Instruction Set	64 bits
SIMD Instructions Set	AVX (256-bit)

Table 1: Hardware specs of an Intel Multicore device

The main form of DLP present on multicore processors are SIMD instructions. This means that the same operation can be applied to various elements at the same time. The more elements it can process at the same time, the more it can take advantage of the DLP. SIMD has seen increasing attention in *High-Performance Computing (HPC)* due to the growing support for bigger registers available in modern processors, where more elements can be processed. To reduce the memory latency the multicore device uses a memory hierarchy which separates each level of memory by response time, from the lowest to the highest times, starting from registers and going to L1..Ln cache, Main Memory (RAM), Disk Storage, etc.. This means that data (and instruction) locality has a great impact in the performance of an algorithm and it has to be taken into account when having HPC in mind.

3.2.2 *GPU - Graphics Processing Unit*

The most abundant hardware for coprocessing is the GPU. It is called coprocessor because it extends the functionality of the CPU which in this case would be considered a host device. The host device is therefore where the main execution of the program is located and it is responsible to offload and retrieve the work performed by the coprocessor.

The use of the GPU is very common in CG, image processing, etc.. The GPU started with no programmable interface but later developed some ability to have programmability that

Figure 7: NVidia Kepler schematic view

was, however, still limited to the graphics pipeline (e.x.: shaders). It was only recently that it has been applied to General Purpose programming and has seen a surge of interest in HPC with the rise of APIs than can use the GPU directly, such as *Compute Unified Device Architecture (CUDA)* and OpenCL.

Without having to map the computing to graphics primitives like textures or vertexes, the GPU could now be used for any problem that fits into its streaming processing paradigm. This paradigm explores the DLP in a sequence of data (stream) applying similar set of operations (kernel functions) to all elements.

The GPU assumes that a vast number of parallel work is going to be processed (eg.: rays, vertexes, pixels), enough to maintain it highly occupied. This results in a design that greatly favours the throughput, i.e. we care a lot more how fast the collective work is finished than how fast we can run any individual piece. This is a fundamental departure from the multicore design where great care is given to reduce latency, something the GPU does not have to worry about because it assumes there is always work to hide that latency.

3.2.3 *Manycore*

MIC is a recent architecture by Intel that can also be a coprocessor. Much like a GPU, it tries to simplify its cores while having many (dozens) of them, rather than having only a few powerful cores. One of the advantages of these Intel manycore devices is that, unlike GPU, it can run x86 code: with very little changes and by recompiling the code, some

1 The 64 KB memory can be configured as either 48 KB of Shared memory with 16 KB of L1 cache, or 16 KB of Shared memory with 48 KB of L1 cache.

NVidia® Tesla K20m	
Codename	Kepler K110
Number of SMX	13
Max Turbo Frequency	706 MHz
Graphical Memory Size	5 GB
Max Memory Bandwidth	208 GB/s
L1 data cache	48K (read-only)
L1 cache	64K [1]
L2 cache	1536KB (shared)

Table 2: Hardware specs of a NVidia GPU device

existent applications can already take advantage of the device. Commercially there are two generations of MIC devices called Xeon Phi, *Knights Corner (KNC)* for the 1st generation (only sold as a coprocessor device) and *Knights Landing (KNL)* for the 2nd generation (only sold as a processor, the coprocessor will be launched soon). At the time of development only the KNC was available so the details in this section are of the 1st generation.

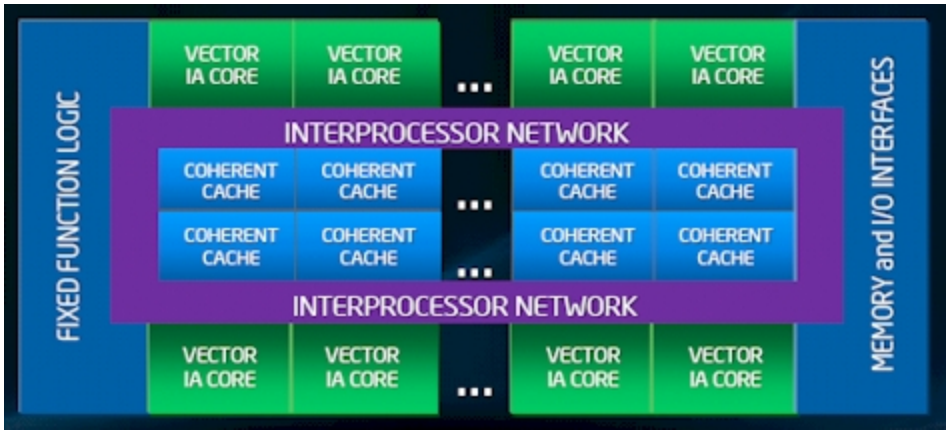


Figure 8: Intel Many Integrated Core architecture

Coding for the Xeon Phi coprocessor can be achieved by using either one of two available execution modes: the native mode and the offloading mode.

Native execution occurs when an application runs completely on the MIC coprocessor. A native application is a way to get existing software running with minimal code changes. An application with a lot of parallelism and very low memory footprint is likely to benefit from the large number of cores available with native execution.

In the offloading model, the programmer designates code sections to run on the MIC target device. This model uses simple directives and language extensions to specify code sections and data to offload to the MIC. In this model, the application is run on the host.

When a offload region is found, either the offload region and data is transferred and run on the target device, or the code region is run on the CPU if no coprocessor is found.

Intel® Xeon Phi™ Coprocessor 7120	
Codename	Knights Corner
Number of Cores	61
SMT	4 way (time-multiplexed)
Max Turbo Frequency	1.33 GHz
Memory Size	16 GB
Max Memory Bandwidth	352 GB/s
L1 data cache	32K
L1 instruction cache	32K
L2 cache	512K (coherent cache)
Instruction Set	64 bits
SIMD Instructions Set	IMCI [2] (512-bit w/ FMA)

Table 3: Hardware specs of an Intel Manycore device

The cores in the KNC device are based in an old `P56` architecture, with no support for out-of-order execution but with TLP (time multiplexed hardware threads) and a bigger emphasis on DLP, perhaps in order to approximate it to the GPU model. It heavily relies on its increased vector registers size at each core, supporting 4-way simultaneous multithreading and bringing the device to a total of 240 threads available to run simultaneously.

2 Intel® Initial Many Core Instructions

4

THE PROBLEM AND ITS CHALLENGES

With each leap in hardware compute capability there is a potential to unlock new and more complex CG techniques. This was the case with path space integration based rendering algorithms. What previously was not feasible to perform in a timely manner was now given attention due to new hardware advances, bringing with it a surge of interest within the physically-based global illumination research community.

Modern graphics applications demand more of their hardware. These demands include increased image resolution and stereoscopic rendering, along with more realism, where both a more accurate simulation of light transport and more detailed scenes are essential.

In order to meet these requirements the efficient exploitation of the computational resources available on current heterogeneous systems is mandatory. Well known algorithms have thus to be rethought and their performance on such heterogeneous systems assessed.

4.1 HETEROGENEITY

The main level of heterogeneity (section 3.1) in this problem is at the System Architecture level with the presence of multicore, manycore and GPU devices. Different machines of the same architecture do not vary, so at the level of Machine Architecture and Machine Configuration there is no heterogeneity.

All devices are run inside the same node so the External Influences due to network also do not exist. However, inside the node the devices are interconnected by a PCIe bus and even if there is only one type of connection, i.e. no Interconnect Network heterogeneity, this communication will highly influence the performance of the system when communication is needed.

Lastly, and also with vast impact, there is heterogeneity at the Software level. Multicore and manycore architectures use the Embree ray-tracing library with the *Intel C Compiler (ICC)* while the GPU uses OptiX in the CUDA programming platform. Additionally the manycore runs a stripped down Linux *Operating System (OS)* that manages the devices and also causes a difference in the programming environment that can be limiting.

4.2 PREVIOUS WORK

Perdigão (2015) work evaluated the suitability of the advanced light transport algorithms in heterogeneous systems using a combination of a single Intel multicore and Nvidia GPU with the use of the Nvidia OptiX (Parker et al., 2010) and Intel Embree (Wald et al., 2014) ray tracing engines.

The two approaches that could be used for work decomposition were: by algorithm iteration or by image space. For the PPM and VCM algorithms, since both use a photon map, decomposition at the image plane involves heavy communication between the multicore and GPU since on every iteration the photon map needs to be consulted. The decomposition by algorithm iteration can be sent to the multicore or GPU independently. However an iteration cannot be given to every core in the multicore because a photon map would need to be available on each one and this would increase the memory requirements to impossible levels. This was not the case for the PT and BPT since no such communication is needed.

To handle heterogeneity and work distribution the multicore (and not its individual cores) and the GPU were considered two work units. Then work (iterations) would be sent independently to each device by DICE. From the implementation side of the application a shared (across all cores) photon map would have to be used to address the memory issues. However this had to be managed from the application side without DICE's intervention.

With great results this work shows that these algorithms can scale very well on heterogeneous systems and therefore are ideal for this use case of a single Intel multicore with a single Nvidia GPU.

4.3 PROGRAMMING ENVIRONMENT

Building on Perdigão (2015) work the first concern was porting the current algorithms to the MIC architecture. While both the native and offloading approach were considered, for the final solution offloading was chosen since it was the most convenient and more viable to integrate with the other architectures since some code, like the scene loading, needs to run on the host.

The first limitation of the implementation was that the CUDA 7.5 platform was only compatible with ICC up to version 15 of Intel compiler. Therefore, in order to support both devices at the same time these version needed to be used. The limitation emerged because ICC 15 does not fully support `C++`'s *Standard Template Library (STL)* when offloading to the Xeon Phi. Since most core code was using STL containers or other STL components extensively these constructs needed to be replaced with appropriate alternatives, such as raw pointers/references, dynamic arrays, etc..

The MIC code also had to be reorganised to not rely on any CUDA primitives because they were not available on the Xeon Phi. This required the creation of some math libraries (ex.: vec2, vec3, vec4 classes) as well as other definitions (ex.: sample_hemisphere, luminance, dot, cross, etc.) which where only available to CUDA code.

Another major hurdle was the fact that the external libraries needed to be compiled specifically for the MIC architecture. With the exception of Intel's Embree which already distributes the MIC compiled binaries, all libraries (DevIL, Assimp, libpng, libjpeg, libtif, etc.) needed to be compiled from source thus opening its own set of problems with incompatibilities of build systems (some needed to be patched to work).

After rewriting, some performance tuning was necessary due to the lack of performance of the MIC architecture. With the switching of flags (such as ensuring `-fastmath` and vectorisation) was very important to bring the performance somewhat on pair with the remaining architectures.

4.4 Work Decomposition

The work associated with all algorithms is decomposed into iterations. An iteration involves evaluating all paths spawned by shooting one primary ray per pixel. At the end of an iteration there is effectively a low quality image, with a sampling rate of 1 sample per pixel. Only a single iteration at a time is performed on any device.

In order to avoid communications, all devices maintain a local frame buffer where the accumulated result of every iteration performed on each device is stored. The final image is then the result of the integration of all the local frame buffers for each device. Given the reduced number of devices, this operation is performed sequentially by the multicore when all iterations finish their execution avoiding extra synchronisation between devices.

The synchronisation between the different tasks inside each device might, however, be unavoidable depending on the rendering algorithm. PT does not need any sharing of data structure since the tree of paths spawned by one primary ray contributes only to the corresponding pixel. Each task writes only to the pixels of the frame buffer corresponding to the image space tile assigned to it. In BPT and VCM algorithms the frame buffer needs to be protected. The connection of the light path vertexes to the camera results on potential contributions to any pixel in the image plane, therefore any tasks can write anywhere in the device frame buffer. To control the concurrent access among the cores the GPU uses atomic operations and the Intel devices use a spin mutex across the whole image plane (using a single mutex proved to be as efficient, performance wise, as using multiple mutexes to protect different regions of the image plane). BPM and VCM also require building a photon map per iteration. On the photon tracing stage, each thread is responsible for a subset of the light paths. An hash grid is used to speedup range searches during the rendering stage,

therefore upon creation of a new photon the respective voxel photon counter is atomically incremented. When all photons have been created voxels are linked to photons using an offset vector and atomic operations whenever required. Since the photon map is built on the multicore for all devices the structure is at the end of this stage copied to the respective device.

4.5 SCHEDULING

Another major concern was to overcome the limitation imposed by NVidia Optix where the scheduler was not able to take control of multiple GPU devices.

The system was changed from a thread centric design to a process centric one, where the different devices are represented by a separate process that interfaces with the scheduler. This required not only that the scheduler had to manage processes instead of threads, but also that launching and merging these processes' output had to be handled by an external program that was created. Another implication was that DICE could not be used, since it was not adaptable to this situation where processes were being used.

This scheduler program, built in Rust (Matsakis and Klock, 2014), can be described as having three phases: setup, spawn and merge. The setup is used for detecting devices and selecting the devices to be used in the computation. The next steps are where the computation is performed. The first one is responsible for spawning the processes that control the selected devices and distribute the different iterations by these processes. The second is initiated when all processes finish their local computation and merges their output into a final image sequentially.

There are two levels of scheduling. The higher level is where iterations are assigned to devices on a demand driven basis and, at any given time, there is only a single iteration being computed per device. At this level the definition of device is closely related to the memory address spaces. Computational resources sharing the same address space (cores within the central processing unit, cores within the Xeon Phi, elementary processors within the GPU) are assigned a single iteration, such that memory consumption, i.e., the number of photon maps, does not increase with the number of such resources per device (we have in fact one photon map per device).

The lower level of scheduling depends on the particular device. NVidia's OptiX, used in the GPUs, is responsible for managing the workload within that device and performs it at the ray level, beyond the programmer's control.

For the remaining devices the workload decomposition is achieved by partitioning the image plane into a large number of tiles and assigning them to the processing cores on a demand driven basis with Intel's TBB. For the BPM and VCM algorithms this is preceded

by the photon shooting stage, with the number of light paths uniformly distributed among the cores.

All the physical/logical cores of multi-socket central processing units are treated as a single device and assigned the same iteration. Also all the physical/logical cores of each Xeon Phi device are assigned a single iteration. This is due to the fact that the BPM and VCM algorithm photon map requires approximately 0.5 GBs of memory (1024x768 light paths with an average of 10 photons per light path), hindering the possibility of assigning an iteration (and consequently a different photon map) to each of these devices' cores.

5

PERFORMANCE ANALYSIS

5.1 METHODOLOGY

A comparative evaluation among rendered images has two key components: how fast they are generated (a measurement of the algorithm and architecture performance) and how good we perceive them (a measurement of the image quality). To accurately evaluate advanced path tracing algorithms on different architectures or heterogeneous system as a whole, all previously selected algorithms were executed in two servers in the SeARCH6 [1] cluster, both with an Intel Xeon dual-multicore `E5-2695 v2` (specs. in table 1) and either with two NVidia Tesla `K20m` GPUs (specs. in table 2) or Intel Xeon Phi Coprocessors `7120` (specs. in table 3). This allowed for the following heterogeneous setups combinations:

- multicore
- multicore + 1 GPU
- multicore + 2 GPU
- multicore + 1 manycore
- multicore + 2 manycore

The multicore will be abbreviated in the images below as "mcore" and the manycore as "MIC". On each of the combinations the experimental results were obtained by executing 4 path space integration algorithms (PT, BPT, BPM and VCM) with 4 different scenes (*Cornell Box*, *Kitchen*, *Sponza* and *Living Room*) as seen on Figure 9. Each of these algorithms was chosen either by its relevance in the light transport simulation study, namely BPT, BPM and VCM, or for some historical relevance, namely PT, which provides a nice base case with low memory overhead and low complexity of implementation.

1 Search-ON2: Revitalization of HPC infrastructure of UMinho, (NORTE-07-0162-FEDER-000086), co-funded by the North Portugal Regional Operational Programme (ON.2-O Novo Norte), under the National Strategic Reference Framework (NSRF), through the European Regional Development Fund (ERDF).

(a) Cornell Box (b) Kitchen

(c) Sponza (d) Living Room

Figure 9: Rendered test scenes used as reference images

It was defined that the programs would be measured by a total of 5 minutes and at the end the number of iterations from each device would be considered. The performance metric is thus the number of iterations per 5 minutes. The number of iterations correlates with image quality, since it is well known that variance reduces with the number of total samples, since all algorithms are unbiased or consistent.

The quality metric will be the *Root Mean Squared Error (RMSE)* and is also evaluated for 5 minute executions relatively to the reference images. The reference image is a very important step in order to evaluate the quality of the images/algorithms since it establishes the baseline to compare the other options against. The reference images used to validate the correctness of the implementations of the algorithms and to verify convergence with the number of iterations were obtained by running the VCM algorithm during 4 hours on a dual 12-core Intel Xeon `E5-2695` server (see Figure 9).

The reported values are obtained using the K-best methodology out of a maximum of 10 executions, with a tolerance of 3%. All images were rendered at a resolution of 1024x768 pixels.

The complexity of each algorithm is different starting with the PT which is the simplest of the four. The BPT builds on the PT with increasing implementation complexity and increased computation complexity. BPM with a different approach need a spatial representation of the light in the scene, and as such requires a higher memory footprint. VCM tries to bring the best of both worlds and is based on techniques pulled from both BPM and BPT to improve the convergence rate. The complexity and memory footprint are noticeably increased for each iteration but the difference in quality can generally outweigh the increase in complexity.

This difference also means that comparing different algorithms by their iterations is not desirable since different amount of work is being done per iteration. Therefore the selected scenes are used to showcase the differences between the algorithms and were chosen to make these differences more apparent allowing for relevant comparison.

The canonical scene in ray tracing is the *Cornell Box* scene, a simple scene that can serve has a base of comparison for all algorithms. The *Sponza* scene can be considered an outdoor scene where little to no reflexion exists, mainly diffuse, and lighten by daylight. The *Kitchen* and the *Living Room*, on the other hand, are indoor scenes. The first one exhibits a powerful light source bouncing around the scene coming from the exterior, and the second with an interior light source with a specular reflective and transmissive materials, causing reflected caustics, well captured by the VCM algorithm. These scenes are also always ordered by geometric complexity (number of vertexes and triangles).

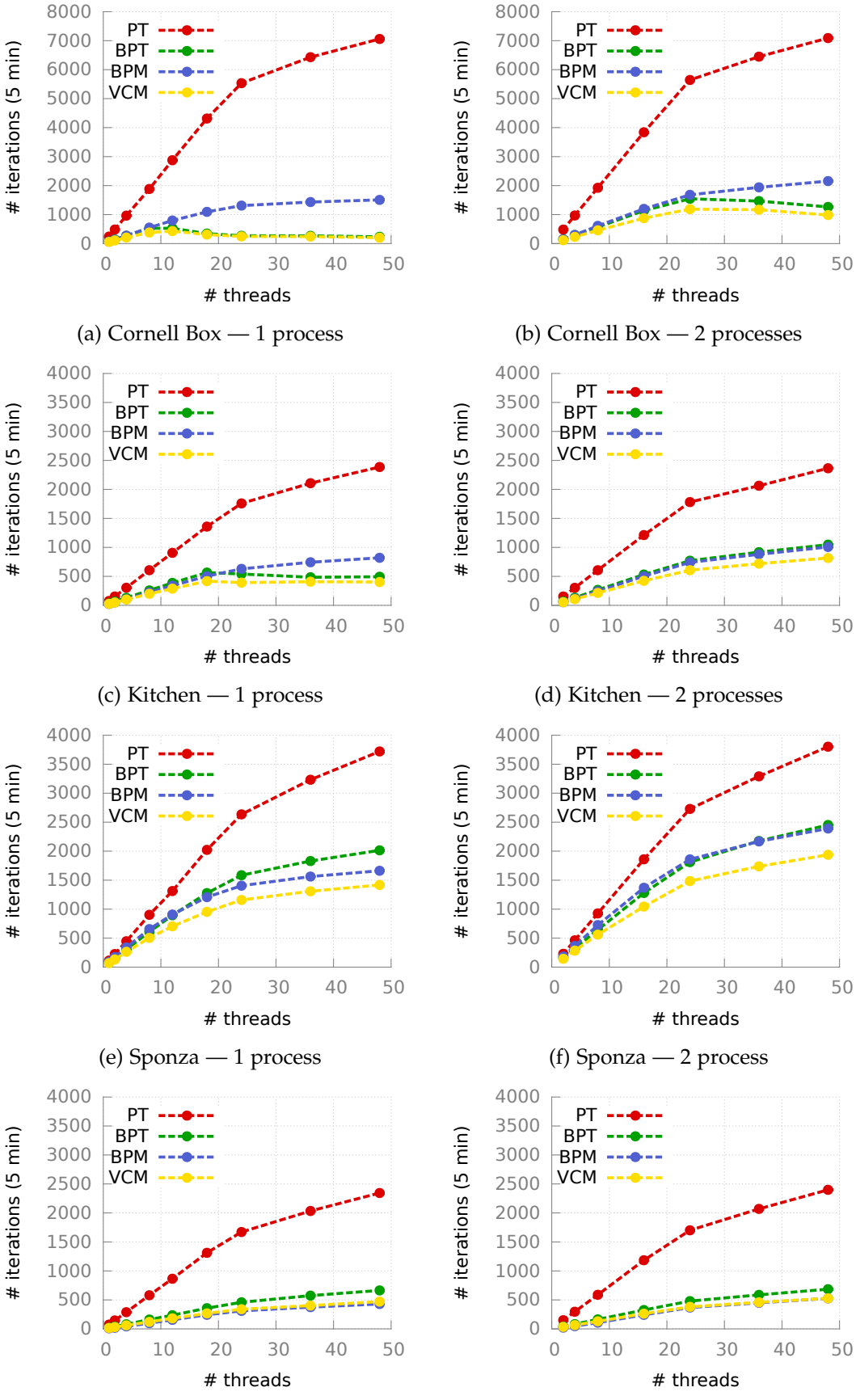
5.2 Results Analysis

5.2.1 *Scalability on Multicore Devices*

These scalability results portray the number of iterations completed within 5 minutes of execution time for different numbers of threads on a computing node with two multicore devices (Intel Xeon `E5-2695`).

Figure 10 shows, for four scenes, the scalability of two different parallel approaches to four algorithms: when a single process is used, using up to 48 threads on both devices, or when two process are used, each using up to 24 threads. This means that although in the first case the threads are allowed to migrate between devices in the second one, this migration is not allowed and no data is transferred across devices.

All algorithms scale well with the number of threads for *Sponza* and *Living Room*.



(a) Cornell Box — 1 process

(b) Cornell Box — 2 processes

(c) Kitchen — 1 process

(d) Kitchen — 2 processes

(e) Sponza — 1 process

(f) Sponza — 2 process

(g) Living Room —1 process

(h) Living Room — 2 process

Figure 10: Scalability on Intel Xeon muticore devices

PT is noticeably better that the others as expected, since no written data structure is ever shared among threads. Also attention to scale of the y axis being different for the *Cornell Box* figures.

However in BPT and VCM there is a sharing of the frame buffer since a light path can write onto any pixel. While on the most complex scene this appears to not make a difference, on less complex scenes such as the *Cornell Box* and *Kitchen* in figures 10a and 10c there is a maximum at 12 and 18 respectively.

This shows that in this more simple scenes there is an overhead in the synchronisation of the frame buffer and there is not enough work to compensate. This becomes even more evident in figures 10b and 10d when, with 2 processes, there is a clear improvement of both scenes when limiting the frame buffer access to a single device. While the *Cornell Box* still has a negative slope above the 24 threads the *kitchen* scene is able to have continuous improvement with the number of threads.

For BPM and VCM there is also a sharing of the photon map for both writing and reading. Therefore the slope of the performance curve of each algorithm is dependent on the level of shared data structures.

There is also a clear decrease in steepness beginning after 24 threads. This is the point where HT starts being used and the device resources being shared. This form of SMT does not allow for a doubling in performance.

Where no inflection point has been reached and the performance keeps increasing with the number of threads it would be interesting to analyse when and which algorithms would reach this inflection point first, if more cores were available.

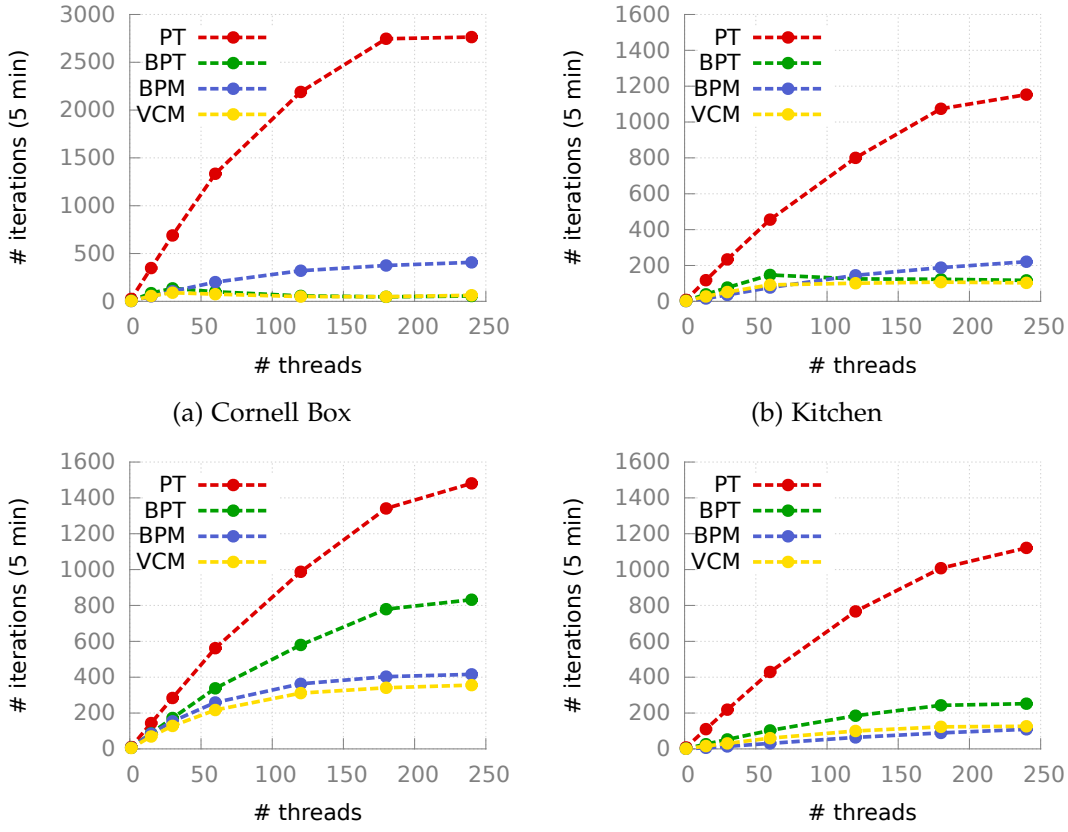
So generally, having 2 processes exhibits a significant impact on performance relatively to the 1 process case. Threads are not allowed to migrate between sockets, and data re-utilisation on the caches shared among cores within the same device is larger. Locality of data accesses is therefore a relevant issue to take into consideration.

5.2.2 *Scalability on Manycore Devices*

Figure 11 portrays the number of iterations completed within 5 minutes of execution time for different numbers of threads on a manycore device (Intel Xeon Phi Knights Corner).

Since this is a 60 physical cores device where each core is capable of up to 4-way SMT, the results are depicted for 1, 15, 30 and 60 threads (no HT), 120 threads (HT 2-way), 180 threads (HT 3-way) and 240 threads (HT 4-way). PT scales far better than the other three algorithms.

As in the previous case the last two scenes have no inflexion point (i.e., performance does not decrease with the number of threads). It is also clear that maximum performance has been achieved for both BPT and VCM. For instance, in figures 11c and 11d VCM's total



(a) Cornell Box

(b) Kitchen

(c) Sponza

(d) Living Room

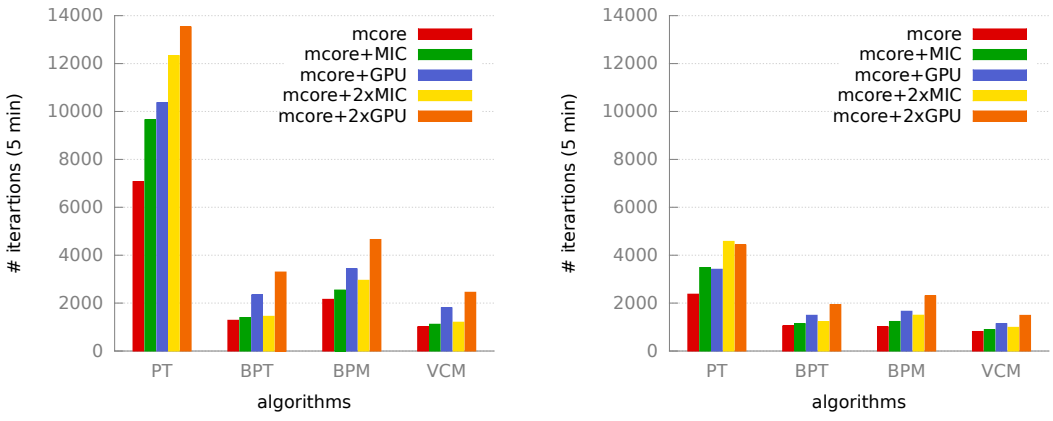
Figure 11: Scalability on Intel Xeon Phi manycore devices

number of iterations is roughly the same for 180 and 240 threads. Adding computational resources no longer results on significantly increased performance due to data sharing.

Despite remaining the same for the last two scenes, for the first two scenes the overhead problem is severely amplified with the use of the manycore. The increase in cores aggravates immensely the amount of communication produced and BPT and VCM have a disastrous preformance in the *Cornell Box* and *Kitchen* scenes.

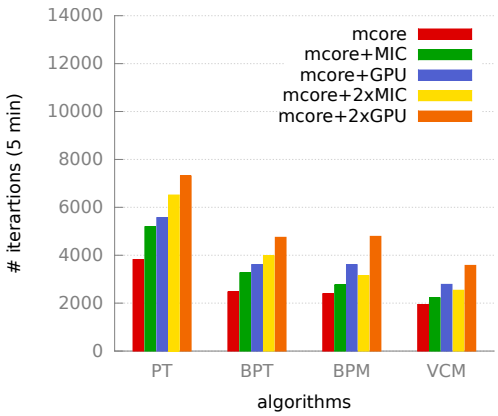
5.2.3 *Heterogeneous Scalability*

Figure 12 depicts the number of iterations completed within 5 minutes of execution time for different combinations of accelerator devices. The dual multicore Xeon E5 is, for all reported scenes and algorithms, the device that contributes with a larger fraction of iterations than any other single device. GPUs clearly outperform the Intel Xeon Phi devices (except for PT in figures 12b and 12d).

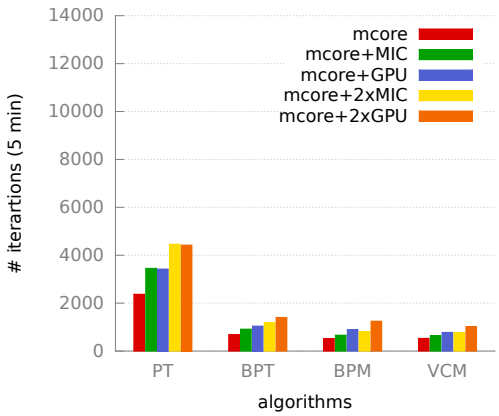


(a) Cornell Box

(b) Kitchen



(c) Sponza



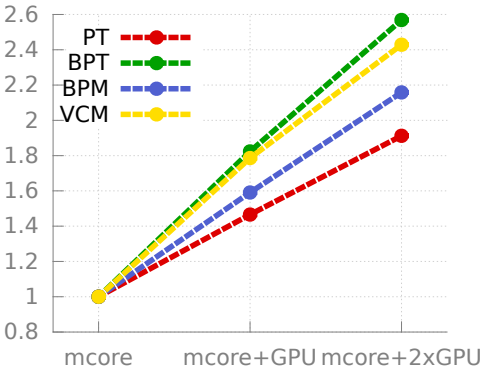
(d) Living Room

Figure 12: Number of iterations on different heterogeneous systems

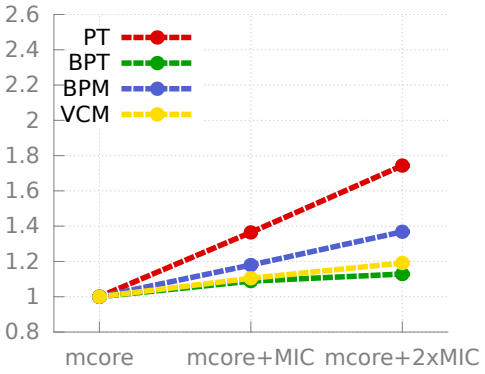
The most important conclusion which can be drawn from this set of results is that, with up to two additional GPUs or Xeon Phi, performance increases for all reported path space integration algorithms.

In order to better illustrate how this performance gain behaves figure 13 presents speedup over the multicore configuration, i.e., for each scene and algorithm, the ratio between the number of iterations obtained including the multicores plus the accelerators and the number of iterations using the multicore alone. Two additional results become now more evident. First, for each algorithm and for each kind of device speedup always increases with the number of such devices; this raises the question of up to which number of devices will speedup maintain this increasing trend. Second, all four algorithms scale with the added heterogeneous computing power.

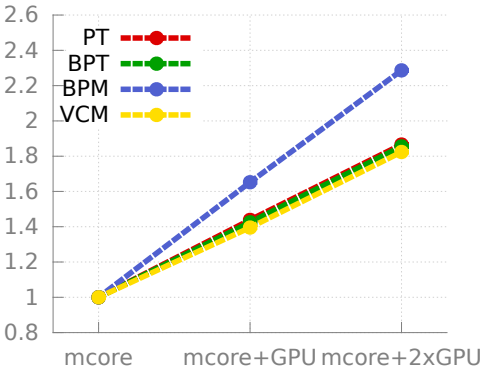
Whereas figure 12 could lead to the conclusion that PT scales much better than the other two, this is not true. The speedups obtained with GPUs are impressive for all algorithms, and, in fact, PT seems to be slightly less scalable than the alternatives – a more thorough



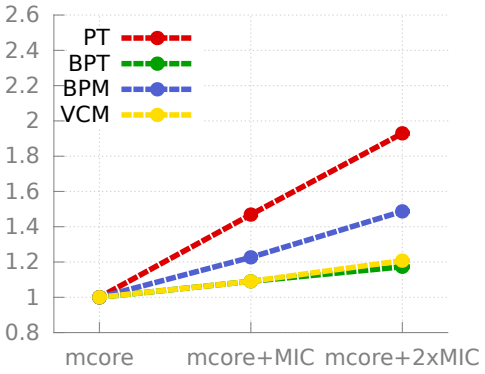
(a) Cornell - GPU



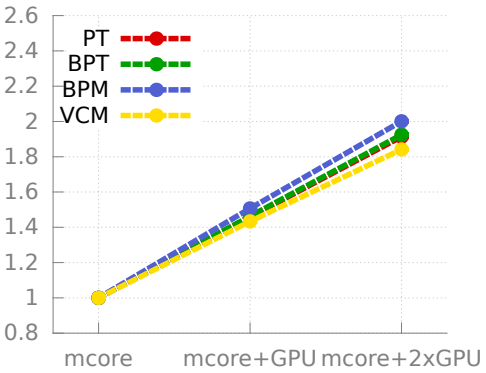
(b) Cornell - MIC



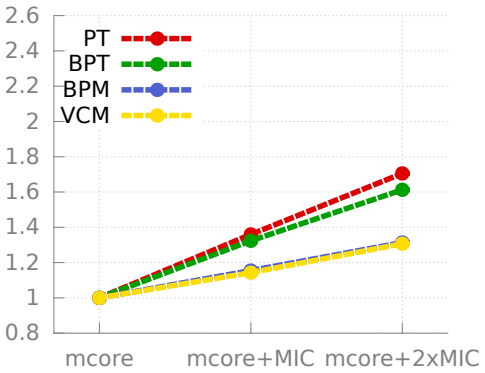
(c) Kitchen - GPU



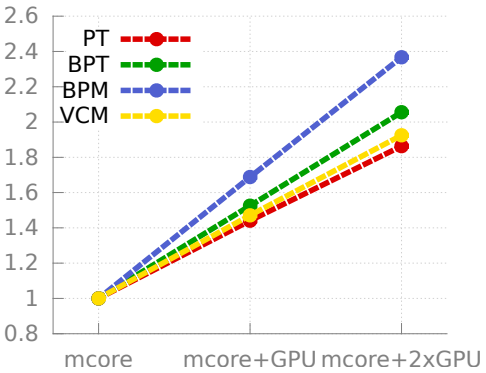
(d) Kitchen - MIC



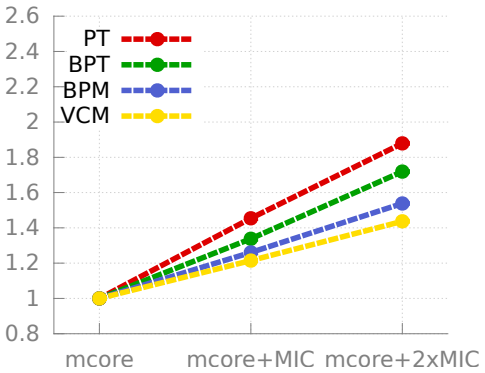
(e) Sponza - GPU



(f) Sponza - MIC



(g) Living Room - GPU



(h) Living Room - MIC

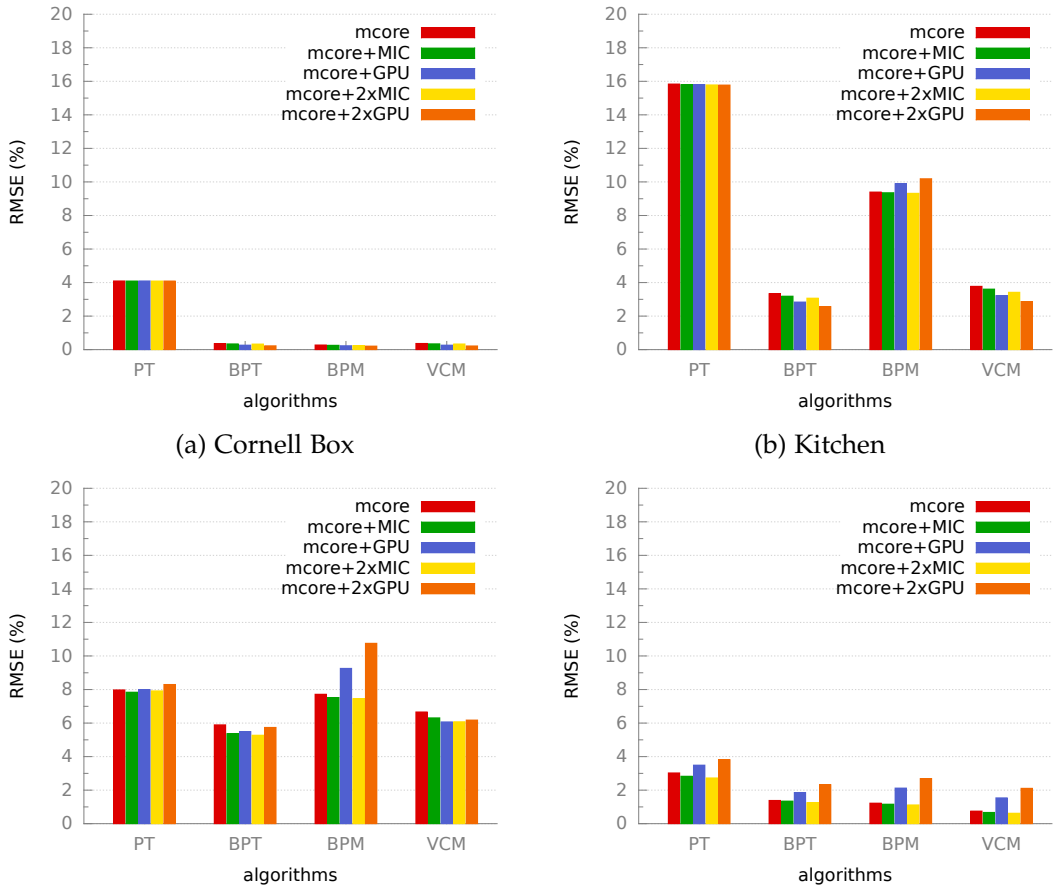
Figure 13: Speedup with different accelerator devices

study of this phenomenon is required, to understand whether this results from these particular data sets and/or servers and whether this trend would be maintained with a larger number of accelerator boards.

5.2.4 *Quality Results*

Figure 14 presents the RMSE obtained with all algorithms for each scene and different hardware configurations. The RMSE is evaluated for 5 minutes executions relatively to the reference images.

The Cornell Box Scene (Figure 14a) due to its simplicity can get a RMSE lower that 1%. This shows the high convergence rate for this scene since after 5 minutes of rendering we get an image that is within 1% of the ref image that rendered for 4 hours.



(a) Cornell Box

(b) Kitchen

(c) Sponza

(d) Living Room

Figure 14: Quality of the 5-min rendered scenes *vs.* reference images

The techniques to improve convergence on BPT, BPM and VCM always generate a lower RMSE than PT which is the worst in every case. The BPM has a bigger error than the other

two and generates an image that is less like the reference image (generated with VCM) but is an algorithm that generates considerably less noise compared to the other algorithms. The BPM results might be suffering due to the algorithm using a different method from the ref image.

Interestingly the VCM only has an advantage over the other algorithms in the Living Room Scene (Figure 14d) that contains reflected caustics, but is not the best choice in all other where BPT is better. Therefore VCM is better suited for more complex scenes with many reflections, including caustics.

In figure 14 the quality clearly correlates with the number of iterations even though it is slightly biased towards VCM due to the actual reference images being generated with the same algorithm.

6

CONCLUSIONS

This dissertation presents the evaluation of Heterogeneous Parallel Computing for different combinations of three computer hardware architectures (Intel multicore with either up to two Intel manycore or NVidia GPU) on the context of Advanced Path Tracing Algorithms. These rendering algorithms include Path Tracing, Bidirectional Path Tracing, Bidirectional Photon Mapping and Vertex Connect and Merge.

The previous proposed solution (Perdigão, 2015), due to some limitations with the OptiX ray-tracing engine, could not take control of more that one GPU. This in conjunction with some limitations porting the algorithms to the MIC architecture, lead to the creation of a new scheduler that manages different devices through processes rather than threads. Each device available on the heterogeneous server is represented by a different process and the workload is distributed among them in a demand driven basis.

This workload is comprised by the different iterations of the algorithms but only an iteration will be running on any single device to avoid contention on the local frame buffer of each device. Inside the each device the managing of the work is specific to the architecture.

Scalability results show that most combination scale well and no inflection point is reached for the number of used devices, i.e., performance increases with additional computational resources. However, scalability is clearly related to the degree of shared data among threads, with PT exhibiting the best results and BPT and VCM suffering from this specially in the manycore where the higher number of cores exacerbates it even more. Results also show that assuring locality of data accesses impacts significantly on overall performance. Using two processes, instead of one, to avoid the migration of inter-socket on the dual multicore (i.e., assuring thread affinity) increases performance, by better exploiting each multicore caches.

It is possible to conclude that these heterogeneous servers have an interesting potential for the efficient execution of path tracing based algorithms. Even though scalability for the more sophisticated algorithms, requiring shared data structures, seems to be approaching the limit.

6.1 FUTURE WORK

The implementation suffered considerable setbacks due to duplication of code and maintainability of different versions of basically the same code in each specific platform. There is technology (eg., OpenCL) that promises to allow a single application to run on different devices without diverging the implementation and it would be interesting to determine the maturity and viability of this technology within the context of advanced path tracing algorithms.

ReadonRays (former FireRays) was considered and later abandoned due to some lack of features, namely ray filtering support, as well as for its hardware compatibility reasons in the OpenCL back-end drivers where only the CPU would be compatible with the OpenCL 2.x version while the GPU and MIC are stuck with OpenCL 1.2. In the meantime the new RadeonRays version support Embree and Vulkan as a back-end. It remains to be determined if these changes mitigate some of the restriction that prevented RadeonRays to be used in as a ray tracing engine in this project.

To expand on the number of supported device architectures is also something that is always expected to bring good insight to capabilities of heterogeneous computing. For example the MIC architecture has seen the new release of the Xeon Phi codenamed KNL during the elaboration of this work, which succeeds the KNC. Accessing how this new hardware compares to its predecessor as well as the others studied architectures is an interesting challenge since a great deal of architectural changes were made (integrated fabric and memory, 2D mesh interconnect, etc.).

In the application itself some small improvements can still be made. For example, with any number of devices selected in a given run, the initial scene setup is being performed on all devices. Some way of sharing or caching this data so that it can be computed only once and then used on all devices could give a nice loading time improvement.

BIBLIOGRAPHY

Cédric Augonnet, Samuel Thibault, Raymond Namyst, and Pierre-André Wacrenier. Starpu: A unified platform for task scheduling on heterogeneous multicore architectures. *Concurr. Comput. : Pract. Exper.*, 23(2):187–198, feb 2011. ISSN 1532-0626. doi: 10.1002/cpe.1631. URL http://dx.doi.org/10.1002/cpe.1631.

J. Barbosa, R. Ribeiro, and L. P. Santos. A framework for efficient execution of data parallel irregular applications on heterogeneous systems. *Parallel Processing Letters*, 25(02): 1550004, 2015. doi: 10.1142/S0129626415500048. URL http://www.worldscientific.com/doi/abs/10.1142/S0129626415500048.

Ian Foster. *Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995. ISBN 0201575949.

Iliyan Georgiev. Implementing vertex connection and merging. Technical report, Saarland University, 2012. URL http://www.iliyan.com/publications/ImplementingVCM.

Iliyan Georgiev, Jaroslav Křivánek, Tomáš Davidovič, and Philipp Slusallek. Light transport simulation with vertex connection and merging. *ACM Trans. Graph.*, 31(6):192:1–192:10, nov 2012. ISSN 0730-0301. doi: 10.1145/2366145.2366211. URL http://doi.acm.org/10.1145/2366145.2366211.

Toshiya Hachisuka, Shinji Ogaki, and Henrik Wann Jensen. Progressive photon mapping. In *ACM SIGGRAPH Asia 2008 Papers*, SIGGRAPH Asia '08, pages 130:1–130:8, New York, NY, USA, 2008. ACM. ISBN 978-1-4503-1831-0. doi: 10.1145/1457515.1409083. URL http://doi.acm.org/10.1145/1457515.1409083.

Henrik Wann Jensen. Global illumination using photon maps. In *Proceedings of the Eurographics Workshop on Rendering Techniques '96*, pages 21–30, London, UK, UK, 1996. Springer-Verlag. ISBN 3-211-82883-4. URL http://dl.acm.org/citation.cfm?id=275458.275461.

James T. Kajiya. The rendering equation. *SIGGRAPH Comput. Graph.*, 20(4):143–150, August 1986. ISSN 0097-8930. doi: 10.1145/15886.15902. URL http://doi.acm.org/10.1145/15886.15902.

Eric Lafortune. Mathematical models and monte carlo algorithms for physically based rendering. Technical report, 1996.

Eric P. Lafortune and Yves D. Willems. Bi-directional path tracing. In *Proceedings of Third International Conference on Computational Graphics and Visualization Techniques (COMPUGRAPHICS '93)*, pages 145–153, 1993.

Nicholas D. Matsakis and Felix S. Klock, II. The rust language. *Ada Lett.*, 34(3):103–104, October 2014. ISSN 1094-3641. doi: 10.1145/2692956.2663188. URL http://doi.acm.org/10.1145/2692956.2663188.

Steven G. Parker, James Bigler, Andreas Dietrich, Heiko Friedrich, Jared Hoberock, David Luebke, David McAllister, Morgan McGuire, Keith Morley, Austin Robison, and Martin Stich. Optix: A general purpose ray tracing engine. *ACM Trans. Graph.*, 29(4):66:1–66:13, jul 2010. ISSN 0730-0301. doi: 10.1145/1778765.1778803. URL http://doi.acm.org/10.1145/1778765.1778803.

César Perdigão. Advanced light transport algorithms on heterogeneous platforms: Evaluation of the dice framework. Master's thesis, University of Minho, Portugal, 2015.

James Reinders. *Intel Threading Building Blocks*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, first edition, 2007. ISBN 9780596514808.

V.S. Sunderam and G.A. Geist. Heterogeneous parallel and distributed computing. *Parallel Computing*, 25(13–14):1699 – 1721, 1999. ISSN 0167-8191. doi: http://dx.doi.org/10.1016/S0167-8191(99)00088-5. URL //www.sciencedirect.com/science/article/pii/S0167819199000885.

Eric Veach. *Robust Monte Carlo Methods for Light Transport Simulation*. PhD thesis, Stanford, CA, USA, 1998. AAI9837162.

Jiri Vorba. Bidirectional photon mapping. In *Proceedings Central European Seminar on Computer Graphics (CESCG'11)*, 2011.

Ingo Wald, Sven Woop, Carsten Benthin, Johnson Gregory S., and Manfred Ernst. Embree: A kernel framework for efficient cpu ray tracing. *ACM Trans. Graph.*, 33(4):143:1–143:8, jul 2014. ISSN 0730-0301. doi: 10.1145/2601097.2601199. URL http://doi.acm.org/10.1145/2601097.2601199.