IA Algorithm Acceleration Using GPUs

Antonio Seoane
University of A Coruña, Spain

Alberto Jaspe
University of A Coruña, Spain

INTRODUCTION

Graphics Processing Units (GPUs) have been evolving very fast, turning into high performance programmable processors. Though GPUs have been designed to compute graphics algorithms, their power and flexibility makes them a very attractive platform for general-purpose computing. In the last years they have been used to accelerate calculations in physics, computer vision, artificial intelligence, database operations, etc. (Owens, 2007).

In this paper an approach to general purpose computing with GPUs is made, followed by a description of artificial intelligence algorithms based on Artificial Neural Networks (ANN) and Evolutionary Computation (EC) accelerated using GPU.

BACKGROUND

General-Purpose Computation using Graphics Processing Units (GPGPU) consists in the use of the GPU as an alternative platform for parallel computing taking advantage of the powerful performance provided by the graphics processor (*General-Purpose Computation Using Graphics Hardware Website*; Owens, 2007).

There are several reasons that justify the use of the GPU to do general-purpose computing (Luebke, 2006):

- Last generation GPUs are very fast in comparison with current processors. For instance, a NVIDIA 8800 GTX card has computing capability of approximately 330 GFLOPS, whereas an Intel Core2 Duo 3.0 GHz processor has only a capability of about 48 GFLOPS.
- GPUs are highly-programmable. In the last years graphical chip programming capacities have grown very much, replacing fixed-programming engines with programmable ones, like pixel and vertex engines. Moreover, this has derived in the appearance of high-level languages that help its programming.
- GPUs evolution is faster than CPU's one. The increase in GPU's performance is nowadays from 1.7x to 2.3x per year, whereas in CPUs is about 1.4x. The pressure exerted by videogame market is one of the main reasons of this evolution, what forces companies to evolve graphics hardware continuously.
- GPUs use high-precision data types. Although in the very beginning graphics hardware was designed to work with low-precision data types, at the present time internal calculations are computed using 32 bits float point numbers.
- Graphics cards have low cost in relation to the capacities that they provide. Nowadays, GPUs are affordable for any user.
- GPUs are highly-parallel and they can have multiple processors that allow making high-performance parallel arithmetic calculations.

Nevertheless, there are some obstacles. First, not all the algorithms fit for the GPU's programming model, because GPUs are designed to compute high-intensive parallel algorithms (Harris, 2005). Second, there are difficulties in using GPUs, due mainly to:

- GPU's programming model is different from CPU's one.
- GPUs are designed to graphics algorithms, therefore, to graphics programming. The implementation of general-purpose algorithms on GPU is quite different to traditional implementations.
- Some limitations or restrictions exist in programming capacities. Most functions on GPU's programming languages are very specific and dedicated to make calculations in graphics algorithms.

Copyright © 2009, IGI Global, distributing in print or electronic forms without written permission of IGI Global is prohibited.

- GPU's architectures are quite variable due to their fast evolution and the incorporation of new features.

Therefore it is not easy to port an algorithm developed for CPUs to run in a GPU.

Overview of the Graphics Pipeline

Nowadays GPUs make their computations following a common structure called *Graphics Pipeline*. The Graphics Pipeline (Akenine-Möller, 2002) is composed by a set of stages that are executed sequentially inside the GPU, allowing the computing of graphics algorithms. Recent hardware is made up of four main elements. First, the *vertex processors*, that receive vertex arrays from CPU and make the necessary transformations from their positions in space to the final position in the screen. Second, the *primitive assembly* build graphics primitives (for instance, triangles) using information about connectivity between different vertex. Third, in the *rasterizer*, those graphical primitives are discretized and turned into fragments. A fragment represents a potential pixel and contains the necessary information (color, depth, etc.) to generate the final color of a pixel. Finally, in the *fragment processors*, fragments become pixels to which final color is written in a target buffer, that can be the screen buffer or a texture.

In the present, GPUs have multiple vertex and fragment processors that compute operations in parallel. Both are programmable using little pieces of code called vertex and fragment programs, respectively. In the last years different high-level programming languages have released like Cg/HLSL (Mark, 2003; HLSL Shaders) or GLSL (*OpenGL Shading Language Information Site*), that make easier the programming of those processors.

The GPU Programming Model

There is a big difference between programming CPUs and GPUs due mainly to their different programming models. GPUs are based on the *stream programming model* (Owens, 2005a; Luebke, 2006; Owens, 2007), where all data are represented by a *stream* that can be defined as a sorted set of data of the same type. A *kernel* operates on full streams, and takes input data from one or more streams to produce one or more output streams. The main characteristic of a kernel is that it operates on the whole stream, instead individual elements. The typical use of a kernel is the evaluation of a function over each element from an input stream, calling this a *map* operation. Other operations of a kernel are expansions, reductions, filters, etc. (Buck, 2004; Horn, 2005; Owens, 2007). The kernel generated outputs are always based on their input streams, what means that inside the kernel, the calculations made on an element never depends of the other ones. In stream programming model, applications are built connecting multiple kernels. An application can be represented as a dependency graph where each graph node is a kernel and each edge represents a data stream between kernels (Owens, 2005b; Lefohn, 2005).

The behavior of graphic pipeline is similar to the stream programming model. Data flows through each stage, where the output feeds the next one. Stream elements (vertex or fragment arrays) are processed independently by kernels (vertex or fragment programs) and their output can be received again by another kernels.

The stream programming model allows an efficient computation, because kernels operate on independent elements from a set of input streams and can be processed using hardware like GPU, that process vertex or fragments streams in parallel. This allows making parallel computing without the complexity of traditional parallel programming models.

Computational Resources on GPU

In order to implement any kind of algorithm on GPU, there are different computational resources (Harris, 2005; Owens, 2007). By one side, current GPUs have two different parallel programmable processors: vertex and fragment processors. Vertex processors compute vertex streams (points with associated properties like position, color, normal, etc.). A vertex processor applies a vertex program to transform each input vertex to its position on the screen. Fragment processors compute fragment streams. They apply a fragment program to each fragment to calculate the final color of the pixel. In addition of using the attributes of each fragment, those processors can access to other data streams like textures when they are generating each pixel. Textures can be seen as an interface to access to read-only memory.

Another available resource in GPU is the rasterizer. It generates fragments using triangles built in from vertex and connectivity information. The rasterizer

4 more pages are available in the full version of this document, which may be purchased using the "Add to Cart" button on the publisher's webpage: www.igi-global.com/chapter/algorithm-acceleration-using-gpus/10346

Related Content

User Relevance Feedback in Semantic Information Retrieval
Antonio Picarielloand Antonio M. Rinaldi (2007). *International Journal of Intelligent Information Technologies (pp. 36-50).*
www.irma-international.org/article/user-relevance-feedback-semantic-information/2417

Generative AI Applications for Enhancing Medical Training
Quratulain Sial, Imdad Ali Shahand N. Z. Jhanjhi (2025). *Generative AI Techniques for Sustainability in Healthcare Security (pp. 161-174).*
www.irma-international.org/chapter/generative-ai-applications-for-enhancing-medical-training/363499

Transforming Student Assessment in Higher Education: The Role of Artificial Intelligence Tools
Arun Agrawal, Saivya Bhadhouriya, Anand Kumar Pandey, Smriti Bhadoriya, Deshdeepak Shrivastava, Gaurav Dubey, Madhukar Dubey, Honey Sengar, Khemchand Shakywarand Vineet Shrivastava (2025). *Improving Student Assessment With Emerging AI Tools (pp. 363-386).*
www.irma-international.org/chapter/transforming-student-assessment-in-higher-education/363058

From Existential Graphs to Conceptual Graphs
John F. Sowa (2013). *International Journal of Conceptual Structures and Smart Applications (pp. 39-72).*
www.irma-international.org/article/from-existential-graphs-to-conceptual-graphs/80382

Global Digital Networks Enabling Cross-Cultural Collaborative Learning Through Dialogue and Co-Creation
Pooja Rani, Jagjit Kaurand Sukhwinder Kaur (2026). *AI, Digital Learning, and Human-Centered Innovation in Education (pp. 91-120).*
www.irma-international.org/chapter/global-digital-networks-enabling-cross-cultural-collaborative-learning-through-dialogue-and-co-creation/411543