Attaining High Performance in General-Purpose Computations on Current Graphics Processors

Francisco D. Igual, Rafael Mayo, and Enrique S. Quintana-Ortí

Depto. Ingeniería y Ciencia de los Computadores, Universidad Jaume I, 12.071–Castellón, Spain
{figual, mayo, quintana}@icc.uji.es

Abstract. The increase in performance of the last generations of graphics processors (GPUs) has made this class of hardware a coprocessing platform of remarkable success in certain types of operations. In this paper we evaluate the performance of linear algebra and image processing routines, both on classical and unified GPU architectures and traditional processors (CPUs). From this study, we gain insights on the properties that make an algorithm likely to deliver high performance on a GPU.

Keywords: Graphics processors (GPUs), general purpose computing on GPU, linear algebra, image processing, high performance.

1 Introduction

During the last years, since the emergence of the first generation of programmable graphics processors (GPUs), many studies have evaluated the performance of these architectures on a large number of applications. Thus, linear algebra operations [10,6], medical image processing [9,12], or database querying [8] are just a few examples of different arenas in which GPU computation has been successfully applied.

Recently, the design of GPUs with unified architecture and the development of general-purpose languages which enable the use of the GPU as a general-purpose coprocessor has renewed and increased the interest in this class of processors. Unfortunately, the rapid evolution of both the hardware and software (programming languages) of GPUs has outdated most of the performance studies available to date.

In this paper, we design and implement a reduced collection of “benchmark” routines, composed of four linear algebra operations (matrix-matrix product, matrix-vector product, saxpy, and scaling of a vector) and an image processing kernel (convolution filter). These routines are employed to evaluate the impact of the improvements introduced in the new generation of GPUs (Nvidia G80), comparing the results with those obtained on a GPU from a previous generation (Nvidia NV44) and current general-purpose processors (AMD Athlon XP 2400+ and Intel Core 2 Duo). The ultimate purpose of this evaluation is to characterize
the properties that need to be present in an algorithm so that it can be correctly and efficiently adapted into the GPU execution model.

The rest of the paper is organized as follows. Section 2 describes the basic architecture and execution model of both the old and new generations of GPUs. Section 3 characterizes the routines in the benchmark collection. Sections 4 and 5 evaluate the performance of the benchmark routines on the Nvidia NV44 and the Nvidia G80, respectively, comparing the results with those obtained on a CPU, and identifying a set of properties that must be present in an algorithm to deliver high performance on that GPUs. Finally, Section 6 summarizes the conclusions that can be extracted from our analysis.

2 GPU Architecture and Execution Model

2.1 GPU Graphics Pipeline

The graphics pipeline consists of a set of sequential stages, each one with a specific functionality and operating on an specific type of data. The process transforms original graphical information (vertices) into data suitable for being shown on display (pixels). Figure 1 illustrates the usual stages (or phases) that form the graphics pipeline.

Current GPUs implement this pipeline depending on the generation they belong to. Thus, classical GPUs have specific hardware units, known as shaders (or processors), for each one of the stages of the graphics pipeline. On the other hand, GPUs from the latest generation have a unified shader (or unified processor), with the ability to both execute any of the stages of the pipeline and work with any type of graphical data.

2.2 Classical Architecture

Until 2006 GPUs were based on a design where each pipeline stage was executed on a specific hardware unit or processor inside the pipeline. Thus, e.g., vertices are processed by vertex processors while pixels (also called fragments) are transformed by fragment processors. In practice, general-purpose algorithms implemented on these classical architectures exploit fragment processors only, due to their larger number and broader functionality. Fragment processors operate in SIMD mode, taking a fragment as input, and processing its attributes;