Aquila: An Open-Source GPU-Accelerated Toolkit for Cognitive and Neuro-Robotics Research

Martin Peniak, Anthony Morse, Christopher Larcombe, Salomon Ramirez-Contla and Angelo Cangelosi

Centre for Robotics and Neural Systems, Plymouth University, UK

Abstract

This paper presents a novel open-source software application, Aquila, developed as a part of the ITALK and RobotDoC projects. The software provides many different tools and biologically-inspired models, useful for cognitive and developmental robotics research. Aquila addresses the need for high-performance robot control by adopting the latest parallel processing paradigm, based on the NVidia CUDA technology. The software philosophy, implementation, functionalities and performance are described together with three practical examples of selected modules.

Introduction

Recent approaches that attempt to understand the nature of cognition have shifted their focus from emphasising formal operations on abstracts symbols to a rather different approach where cognition is seen as an embodied or situated activity and therefore largely determined by the physical form of an embodied system [1][2]. Artificial intelligence, developmental psychology, neuroscience, and dynamical systems theory have directly inspired a completely novel approach called developmental robotics, which is a highly interdisciplinary subfield of robotics also known as epigenetic or epigenetic robotics [3][4]. Artificial cognitive systems based on the developmental robotics approach need to undergo an autonomous and gradual mental development from ”infancy” to ”adulthood”. Interaction with their environments in an autonomous manner with little or no human intervention is necessary where no tasks or goals are pre-defined. This aids the process of achieving a good level of environmental openness where an artificial cognitive system is able to cope with different and previously unexpected environments. Another important aspect of the systems based on developmental robotics is that they are able to learn from their previous experience and use it to assist the acquisition of new skills. Developmental robotics attempts to understand how the control system’s organisation of a single robot develops through various experiences over time. Embodiment plays a significant part for achieving the goal to develop autonomous artificial embodied agents capable of acquiring complex behavioural, cognitive and linguistic skills through individual and social learning. For example, it has been demonstrated that research in action and language learning in natural and artificial cognitive systems can directly benefit from this approach, inspired by the developmental models and phenomena studied in children, allowing re-enactment of gradual process that have the potential to bootstrap various cognitive capabilities and integrate them into a unified interactive cognitive system [4][5]. The main theoretical hypothesis is based on the assumption that the parallel development of action, conceptualisation and social interactions permits the bootstrapping of language capabilities, which leads to the enhancement of cognitive development [6].

The developmental robotics approach to action and language learning is consistent with recent brain-inspired approaches to mental development since computational neuroscience considers the neural development constrains on embodiment, as well as on cognition [7][8][9]. It is not surprising that an overwhelming number of studies from various fields suggest that actions and language are closely integrated together. This highlights the importance of embodiment as well as supports the hypothesis of usage based language as opposed to classical explanations of language development that assumes an extensive language-specific cognitive hardwiring. The modelling of the integration of various cognitive skills and modalities requires complex and computationally intensive algorithms running in parallel while controlling high-performance systems. The processing requirements are increasing with every added feature and it is not uncommon that at the end of the software development stage a particular system is unable to cope with fast-response robotcontrol tasks. This is very likely when a system requires applying filters to millions of pixels from a robot's cameras, running large-size neural networks with millions of synaptic connections and using multiple self-organising maps while controlling the robot in the real-time. Around the year 2003, to overcome the energy consumption and heat-dissipation problems of standard PC processors, manufacturers started to produce computers with multiple cores. This had a strong impact on the software developer community [10]. In the meanwhile, manufacturers have been looking into new technologies that would increase the number of transistors per wafer. However, reducing these dimensions comes at a price since the current leakage becomes a problem. Since 2003, the production of semiconductors has been divided into multicore and manycore design trajectories [11]. Manycore design aims to increase the processing power by increasing the number of cores in a processor. This number was doubling with each semiconductor process generation starting with dual-core chips and reaching hyper-threaded hexa-core systems. A manycore system is fundamentally different with regards to its design philosophy. While CPUs (Computer Processing Units) are optimised for the processing of sequential code and feature sophisticated control logic and large cache memories, the GPU (Graphic Processing Units) design philosophy emerged from the fast growing video industry where massive numbers of floating point operations are required to render every single frame. As a result, a GPU chip has most of its area dedicated to processing of the floating point operations and features only tiny cache memories. In 2006, NVidia released GeForce 8800 GPU, which was capable of mapping separate programmable graphics processes to an array of GPUs, which paved the way to first general purpose computing using parallel GPU processors. GPGPU was an intermediate step where graphics card programmers had to use the OpenGL or DirectX API to implement their programs. Using the GPGPU technique many different applications have achieved dramatic speed improvements. For example, Kruger and Westermann developed a framework for solving linear algebra [12], Harris and colleagues designed a cloud dynamics simulation based on partial differential equations [13], Rodrigues and colleagues implemented molecular dynamics simulation [14] and Nyland and colleagues N-body simulation [15]. More recent GPU developments, based on the NVidia Tesla GPU architecture, provide clusters of GPUs used as individual programmable processors and allow more efficient parallel processing tools. The CUDA (Compute Unified Device Architecture) programming tool has been designed on purpose to support mutual CPU/GPU application execution. Parallel computing using CUDA and GPU cards is being increasingly taken up by industry and academies. Many commercial

and research applications have migrated from using solely standard CPU processors to a collaborative CPU/GPU use where each architecture does what is best at. In general, most of these applications can achieve tremendous speed-ups in performance, which is anything between 1.3x to 2,600x [16]. Since quantum computing is still in its infancy and CPUs are approaching the processing limits constrained by the physical laws, it seems that parallel computing using GPU devices is the next paradigm that is yet to become fully recognised and widely used.
CUDA has been employed in a wide variety of applications, however, only a handful of these have any relevance for the cognitive and neuro-robotics domains. Only few studies have to date applied CUDA to neural networks and visual processing (e.g. [17][18]), as this field requires further investigation in applying the CUDA technology to neural computation and robotics research. This paper presents a novel software tool, named Aquila, suitable for GPU-based cognitive and neuro-robotics applications and that makes use of the latest parallel processing paradigm based on the CUDA technology. This software has been developed as a part of the ITALK and RobotDoC projects as well as the open-source project on the iCub humanoid robot1.
In the next sections we first provide an overview of Aquila's philosophy, its software implementation and its functionalities and performance. We then discuss three different examples on the use of Aquila for various cognitive robotics studies.