A Pipelined Hardware Implementation of Genetic Programming Using FPGAs and Handel-C

Peter Martin
Department of Computer Science, University of Essex, Wivenhoe Park, Colchester, CO4 3SQ, UK.

Abstract. A complete Genetic Programming (GP) system implemented in a single FPGA is described in this paper. The GP system is capable of solving problems that require large populations and by using parallel fitness evaluations can solve problems in a much shorter time that a conventional GP system in software. A high level language to hardware compilation system called Handel-C is used for implementation.

1 Introduction

The motivation for this work is that as problems get harder, the performance of traditional computers can be severely stretched despite the continuing increase in performance of modern CPUs. By implementing a GP system directly in hardware the aim is to increase the performance by a sufficiently large factor to be able to tackle harder problems and to make investigations into the operation of GP easier. This paper is an update to the work presented in [9] which describes how a GP system that includes initial population creation, fitness evaluation, selection, and breeding operators can be implemented in a Field Programmable Gate Array (FPGA) using a high level language to hardware compilation technique. Two major areas were singled out for further work in order to improve the performance: 1) extend the implementation to handle larger populations; 2) the use of pipelining to improve the parallelism of the hardware.

The changes needed and the results of implementing the changes are described in this paper. The paper begins with a brief survey of previous work using FPGAs for evolutionary techniques and a short summary of the Handel-C language and the target hardware. This is followed by a description of the revised design that stores the population in off-chip Static Random Access Memory (SRAM) and that also uses pipelining. The experimental setup is presented together with some results that illustrate the effect of the changes. The changes are then discussed and areas for further work are suggested.

2 Previous Work Using FPGAs in Evolutionary Computing

FPGAs have featured in the field of evolutionary computing under three distinct headings:
1) as a means of implementing the fitness functions of Genetic Algorithms or Genetic Programming [6,17];
2) as a platform for implementing a Genetic or Evolutionary Algorithm [3-4, 12,13,14];
3) in relation to evolving hardware by means of an evolutionary technique [2,8,15,16].
A more detailed review of this and other work can be found in [9].

3 Description of Handel-C and the Target Hardware

Handel-C is a high level language that is at the heart of a hardware compilation system known as Celoxica DK1 [1] which is designed to compile programs written in a C-like high level language into synchronous hardware. Since Handel-C targets hardware, there are some programming restrictions when compared to using ISO-C, and these need to be considered when designing code that can be compiled by Handel-C. Some of these restrictions particularly affect the building of a GP system. Firstly, there is no stack available, so recursive functions cannot be directly supported by the language. Secondly, there is a severe limit to the size of memory that can be implemented using standard logic cells on an FPGA because implementing memory is expensive in terms of silicon real estate. However, some FPGAs have internal RAM that can be used by Handel-C which is supported by the `ram` storage specifier.

The target hardware for this work is a Celoxica RC1000 FPGA development board fitted with a Xilinx XCV2000E Virtex-E FPGA having 43,200 logic cells and 655,360 bits of block ram, a PCI bridge that communicates between the RC1000 board and the host computer’s PCI bus, and four banks of Static Random Access Memory (SRAM). Logic circuits isolate the FPGA from the SRAM, allowing both the host CPU and the FPGA to access the SRAM, though not concurrently.

4 System Architecture

The lack of a stack in Handel-C means that a standard tree based representation is difficult to implement because recursion cannot be handled by the language. Instead, a linear program representation is used [11], though other compact representations such as Cartesian Genetic Programming [10] in which programs are represented as graphs, are also worth considering. Using a linear representation, a program consists of a sequence of words which are interpreted by the problem specific fitness function. To ease the design, each program has a fixed maximum length that it can grow to. Crossover is performed by selecting crossover points at random in two individuals and swapping the nodes after the crossover points. If the length of a program would exceed the maximum, it is simply truncated to the maximum. Mutation is performed on an individual by replacing a word with a new randomly generated word which has the potential effect of changing