Automatic Machine Code Generation for a Transport Triggered Architecture Using Cartesian Genetic Programming

James Alfred Walker, Department of Electronics, University of York, York, UK
Yang Liu, Department of Electronics, University of York, York, UK
Gianluca Tempesti, Department of Electronics, University of York, York, UK
Jon Timmis, Departments of Electronics and Computer Science, University of York, York, UK
Andy M. Tyrrell, Department of Electronics, University of York, York, UK

ABSTRACT

Transport triggered architectures are used for implementing bio-inspired systems due to their simplicity, modularity and fault-tolerance. However, producing efficient, optimised machine code for such architectures is extremely difficult, since computational complexity has moved from the hardware-level to the software-level. Presented is the application of Cartesian Genetic Programming (CGP) to the evolution of machine code for a simple implementation of transport triggered architecture. The effectiveness of the algorithm is demonstrated by evolving machine code for a 4-bit multiplier with three different levels of parallelism. The results show that 100% successful solutions were found by CGP and by further optimising the size of the solutions, it’s possible to find efficient implementations of the 4-bit multiplier. Further analysis of the solutions showed that use of loops within the CGP function set could be beneficial and was demonstrated by repeating the earlier 4-bit multiplier experiment with the addition of a loop function.

INTRODUCTION

In the past decade, evolvable hardware has attracted interest from both circuits design and evolutionary computation. Generally there are two main branches of applications: optimising the elementary parameters of a circuit (Hilder, Walker & Tyrrell, 2009), and more interestingly, creating a circuit from smaller compositional units (Miller, Job & Vassilev, 2000, Walker & Miller, 2008). In the latter, Cartesian Genetic Programming (CGP) (Miller & Thomson, 2000) has demonstrated great potential to create combinatorial circuits. However, previous evolutionary digital circuit designs are limited by...
the fine-grained nature of the devices. Gate level programmability provides the system with high flexibility but also enlarges the overall search space and thus increases the degree of complexity tending to reduce the overall functionality that can be achieved. Lifting the granularity from gate level to processor architecture level provides another form of evolutionary medium (substrate). A typical coarse-grained programmable machine is a general purpose processor (GPP). At this level, Nordin (1997) proposed a graph-based genetic programming technique, known as Linear Genetic Programming (LGP), to automatically generate machine code for GPPs. The behaviour of a processor is expressed by executing the evolved instructions.

Between the fine-grained and coarse-grained architectures, another architecture exists, which is known as Application Specific Instruction Processors (ASIP). As the name suggests, an ASIP usually performs application oriented functionalities and the instructions are coded for particular operations. In order to reduce the complexity of the decoder logic and the length of the execution pipeline, an operation of an ASIP is usually an atomic control of the internal data flow of the processor. An example of a simple implementation of an ASIP is the MOVE processor, which is based on a transport triggered architecture (TTA) and has been used in bio-inspired systems, such as the POEtic Project (Rossier, Thoma, Mudry & Tempesti, 2006, Mudry, 2009) and the SABRE project (Liu, Timmis, Qadir, Tempesti & Tyrrell, 2010) because of its simplicity, modularity and intrinsic fault tolerance. However, one disadvantage of the TTA architecture is that the computational complexity is transferred from the hardware-level to the software-level, which makes it much harder to program. Compilers and synthesis tools for the MOVE processor do exist (Mudry, 2009), but they are still in the development stages and do not yet produce efficient, optimised machine code.

This paper is an extension of the earlier work reported in (Walker, Liu, Tempesti & Tyrrell, 2010, Liu et al, 2011), which focuses on the automatic generation of machine code using Cartesian Genetic Programming (CGP) for a MOVE processor. First, we review the transport triggered architecture (TTA) and describe the CGP algorithm for code generation. After, we present a demonstration of how to create a particular function using a sequence of evolved instructions and describe an extension to the CGP algorithm that incorporates the use of a loop function. Following that we demonstrate the effectiveness of the modified CGP algorithm compared with the earlier method. Finally, we conclude the paper and proposes directions for future work.

**TRANSPORT TRIGGERED ARCHITECTURE**

The transport triggered architecture (TTA) was created in the 1980’s, as a development of the Very Long Instruction Word (VLIW) architecture (Corporaal, 1998). The TTA usually contains an instruction decoder, an interconnection network and a number of functional units (FU), as shown in Figure 1. Functional units and the decoder are connected through data and address buses. A single data/address bus pair is called a slot. A processor can have multiple slots to allow parallelism. An input/output interface of a functional unit is called a port. Ports are usually globally addressable. The connection of a port to a slot is called a socket and the number of conjunctions in a socket is flexible.

In the TTA, the instruction decoder is a unit that fetches code from the memory, decodes the instructions and controls the interconnecting network. The decoder has a program counter (PC) which points to the address of next instruction in the memory. The structure of a decoder is generally very simple, because instructions of a TTA only contain two types of information: the source port (SRC) address (or intermediate value) and the destination port (DST) address, as shown in Figure 2. A source address and a destination address together indicate a data flow from one port to another. From this perspective, the TTA has similar
Mixing Workflows and Components to Support Evolving Services

www.igi-global.com/article/mixing-workflows-components-support-evolving/48263?camid=4v1a