Customisable Multi-Processor Acceleration of Inductive Logic Programming

Andreas K. Fidjeland, Wayne Luk, and Stephen H. Muggleton

Imperial College London,
180 Queen's Gate,
London SW7 2AZ, UK
{andreas.fidjeland,w.luk,s.muggleton}@imperial.ac.uk

Abstract. Parallel approaches to Inductive Logic Programming (ILP) are adopted to address the computational complexity in the learning process. Existing parallel ILP implementations build on conventional general-purpose processors. This paper describes a different approach, by exploiting user-customisable parallelism available in advanced reconfigurable devices such as Field-Programmable Gate Arrays (FPGAs). Our customisable parallel architecture for ILP has three elements: a customisable logic programming processor, a multi-processor for parallel hypothesis evaluation, and an architecture generation framework for creating such multi-processors. Our approach offers a means of achieving high performance by producing parallel architectures adapted both to the problem domain and to specific problem instances.

Keywords: ILP, FPGA, multi-processor

1 Introduction

Inductive Logic Programming (ILP) is a powerful paradigm for symbolic machine learning, since it can incorporate existing theories and produce human-readable output. However, ILP systems are computationally demanding. Several approaches to speeding up ILP have been developed with parallelisation taking place at different levels [1, 3, 7, 9]. Common to these approaches is the reliance on conventional general-purpose processors, with the parallel processing units either being nodes in a distributed computer or cores in a multi-core processor.

This paper describes a different approach, based on advanced reconfigurable hardware such as Field-Programmable Gate Arrays (FPGAs), to provide multi-processors with a customisable architecture. Similar approaches have been used in speeding up various demanding applications, such as those in financial modelling [4] and in medical imaging [11]. Our approach is developed for speeding up the ILP system Progol [6], using customised instruction processors and multi-processors. Its unique features include:

1. Arvand, an instruction processor for logic programming, which can be customised to particular classes of data sets for learning (Section 2);

2. a Progol multi-processor, which exploits data parallelism in hypothesis evaluation (Section 3); and
3. a multi-processor architecture generation framework for logic programming, which is used to create customised multi-processors (Section 4).

Reconfigurable hardware has been used in emulating Intel architectures [8]. Such emulation, however, does not exploit the reconfigurability of FPGAs to provide a customisable architecture. Moreover, our research demonstrates how fine-grained parallelism on an FPGA can significantly enhance ILP performance.

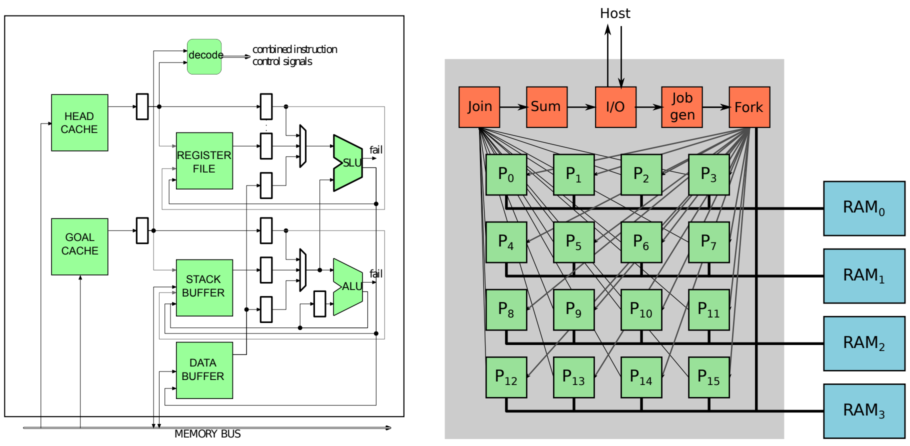
2 Parallel ILP and the Arvand Processor

Inductive Logic Programming systems come in several variations, but in general they take as input a set of positive and negative examples, some background knowledge, and a language bias defining the hypothesis space. An ILP system produces a theory explaining the examples. ILP algorithms employ different strategies for constructing and searching through the hypothesis space, and for assessing the quality of each hypothesis.

Parallel approaches to ILP exploit parallelism at different levels. Fonseca et al. [3] identify three main levels. First, search parallelism performs search through the hypothesis space in parallel [1, 7]. Second, data parallelism splits the example set and performs learning based on the subsets [9]. Third, evaluation parallelism splits the coverage test, and evaluates the candidate with respect to the example set in parallel. These approaches are not necessarily mutually exclusive. The work described below exploits mainly evaluation parallelism.

Our approach to accelerating ILP is built around a customisable processor, Arvand. The Arvand processor is based on the Vienna Abstract Machine (VAM), a two-pointer abstract machine for Prolog [5]. The abstract machine is realised in hardware as a two-issue four-stage pipelined processor (Figure 1a) which directly executes VAM instructions. The processor performs unification on the two instruction streams, which correspond to the goal and head of the current predicate. A general-purpose register set supports both this unification and arithmetic. The stack contains both determinate and non-determinate activation records, supporting backtracking. The complex control instructions handle operations on this mixed stack, with the support of a special-purpose register set. The processor contains logic for handling indexing, which is a common Prolog implementation technique to avoid redundant computation.

The Arvand processor can be customised for a particular program type, or to exploit different run-time characteristics of a program. The processor customisations affect the usage of both programmable fabric (used for processor logic) and embedded memory (used for caches and buffers), and can additionally affect execution time, and the class of programs supported (Figure 2 left). Customisations come in four different forms. (a) Microarchitecture customisations reduce the processor instruction set to the minimal that supports specific programs. Examples include simplifying the control logic for programs containing only ground unit clauses, and removing the ALU for purely symbolic programs.



(a) (b)

Fig. 1. Multi-processor architecture for ILP. (a) Arvand processor in a typical configuration. (b) Multiple Arvand processors for parallel evaluation of hypotheses.

(b) Memory interface customisations make some memory units (goal program code, stack, or heap) purely local units rather than being caches/buffers backed by off-processor data. (c) Memory size customisations vary the capacity of on-processor memory units; they can have a large effect on memory usage and on program execution time, due to variations in cache performance. (d) Data width customisations change the default word width of the processor. Figure 2 (right) shows the usage of both programmable fabric and memory for processors in various configurations. The programmable fabric usage (measured in device-independent flip-flops) varies by a factor of around three, whereas the memory usage varies by a factor of around five. These resource savings translate into increased parallelism, since more small processors can fit on a particular chip.

3 Multi-Processor Architecture for ILP

The Arvand processor requires only a fraction of the resources found on a modern reconfigurable device, even when using one of the more demanding processor configurations. It is therefore possible to place a large number of processors on a single chip, creating a customised multi-core processor for ILP. Our multi-processor (Figure 1b) acts as an accelerator for a host system running the learning algorithm, directing the search through the hypothesis space. The multi-processor receives a stream of candidate hypotheses from which a number of queries for the example tests are generated. These queries are distributed to the available processors. The results (successes or failures) from all the queries related to a hypothesis are aggregated and returned to the host system.

Customisation	Space	Memory	Other
Microarchitecture	X		programs
Memory interface	x	X	
Memory size	x	X	run time
Data width	x	x	

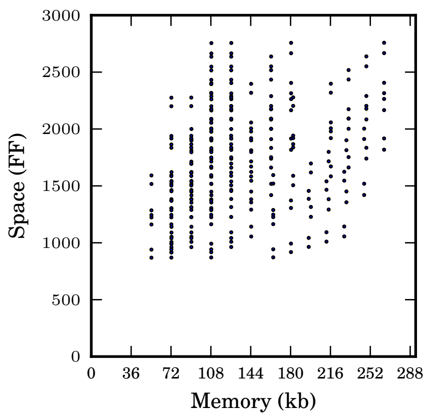


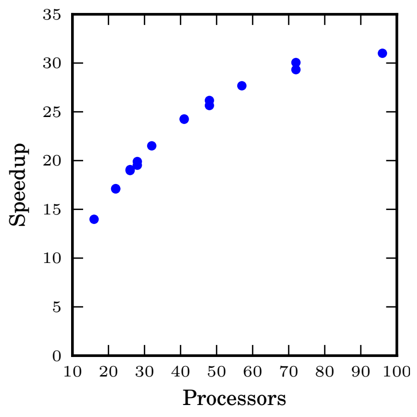
Fig. 2. Arvand processor configuration: (left) major (X) and minor (x) effects of different configuration options; (right) resource usage for a number of different processor configurations.

The single-processor customisations affect the configuration of the multi-processor. For a given device, there is a fixed amount of resources (both programmable fabric and memory). There is thus a general trade-off between the resource usage of a single processor and the number of constituent processors in a multi-processor. For a given data set, the processor architecture, memory interface, and data width can be fixed. The amount of local memory dedicated to caches and buffers can be varied, however. The trade-off is therefore in practice between the number of processors and the cache sizes of each processor; the single-processor performance will typically be better if caches are larger.

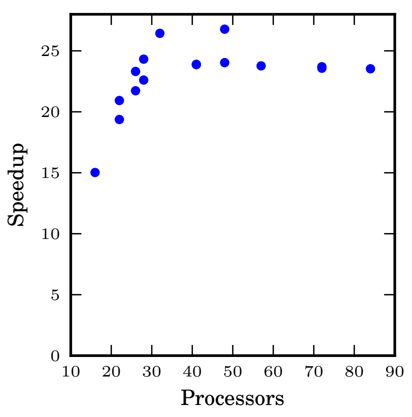
The effect of this trade-off can be observed in Figure 3, which shows the speedup with respect to a single processor, for multi-processor configurations optimised for two ILP data sets: mutagenesis [10] and protein folding [12]. The mutagenesis dataset has a small working set, so it suffers little adverse effect from having small caches; the highest-performing configurations, with more than 30 times speedup, are therefore those which maximise parallelism. The protein folding data set requires a more complex processor, and also has a larger working set. A large number of small-cache processors is therefore not advantageous. The highest-performing configurations achieve around 27 times speedup.

4 Multi-Processor Arcitecture Generation

To facilitate generation of multi-processor systems based around Arvand, we have developed an architecture-description language, Archlog, which ties together software compilation, processor configuration and instantiation, and multi-processor configuration and generation [2]. Archlog includes a domain-specific language in Prolog that covers multi-processor architectures using a number of primitives (including Arvand) and communication streams between them.



(a) Mutagenesis



(b) Protein folding

Fig. 3. Speedup of optimal multi-processor configurations for two different datasets.

The Archlog system (Figure 4) takes an architecture description and a Prolog program, and generates a hardware configuration tailored for this combination of architecture and software. Analysis of the input Prolog program(s) provides part of the processor configuration, by specifying the minimal processor configuration that can support the program. An architecture description may not fully specify all parameters of the design, for example the level of parallelism in a multi-processor design. The system explores the space of possible designs, and can return a number of pareto-optimal designs.

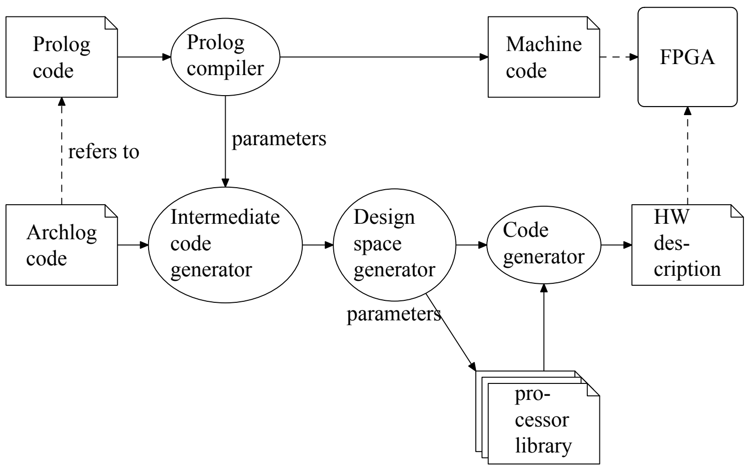


Fig. 4. The Archlog system for multi-processor architecture generation.

5 Concluding Remarks

Customisable architectures show good promise in speeding up demanding ILP applications. Our building blocks and architecture generation framework can be adapted to enable ILP systems other than Progol to exploit this technology, without the need for hardware design expertise. Moreover, to realise the full potential of our approach, we are integrating the multi-processor design tools seamlessly with Progol, targeting the latest high-performance FPGA systems.

Another theme of ongoing research involves studying how variations in run-time characteristics of inductive logic programming can be used in optimising performance and energy consumption. Such variations can be exploited by adapting the resources in a multi-processor system to match the run-time characteristics, making use of hardware reconfigurability. A key challenge is to automate such exploitation for realistic applications while minimising overheads in run-time reconfiguration, such that efficient designs can be produced cost-effectively.

References

1. Dehaspe, L., Raedt, L.D.: Parallel inductive logic programming. In: Proc. MLnet Familiarization Workshop on Statistics, Machine Learning and Knowledge Discovery in Databases. pp. 112–117. Heraklion, Crete (January 1995)
2. Fidjeland, A., Luk, W.: Archlog: High-level synthesis of reconfigurable multiprocessors for logic programming. In: Proc. Int. Conf. Field-Programmable Logic and Applications. pp. 335–340. IEEE (August 2006)
3. Fonseca, N.A., Srinivasan, A., Silva, F., Camacho, R.: Parallel ILP for distributed-memory architectures. Mach. Learn. 74, 257–279 (March 2009)
4. Jin, Q., Thomas, D.B., Luk, W., Cope, B.: Exploring reconfigurable architectures for tree-based option pricing models, ACM Trans. on Reconfig. Tech. and Sys., 2(4), Article 21 (2009)
5. Krall, A., Neumerkel, U.: The Vienna Abstract Machine. In: Proc. Int. Workshop Programming Language Implementation and Logic Programming. pp. 121–135. No. 456 in LNCS, Springer-Verlag (August 1990)
6. Muggleton, S.H.: Inverse entailment and Progol. New Generation Computing 13, 245–286 (1995)
7. Ohwada, H., Nishiyamai, H., Mizoguchi, F.: Concurrent execution of optimal hypothesis search for inverse entailment. In: Proc. Int. Conf. Inductive Logic Programming. pp. 165–173 (2000)
8. Schelle, G. et al: Intel Nehalem processor core made FPGA synthesizable. In: Proc. Int. Symp. on FPGA, 3–12 (2010)
9. Skillicorn, D.B., Wang, Y.: Parallel and sequential algorithms for data mining using inductive logic. Knowledge and Information Systems 3, 405–421 (2001)
10. Srinivasan, A., Muggleton, S.H., King, R., Sternberg, M.: Mutagenesis: ILP experiments in a non-determinate biological domain. In: Proc. Int. Inductive Logic Programming Workshop (1994)
11. Tsoi, K.H., Rueckert, D., Ho, C.H., Luk, W.: Reconfigurable acceleration of 3D image registration. In: Proc. South. Conf. on Prog. Logic, 95–100 (2009)
12. Turcotte, M., Muggleton, S.H., Sternberg, M.J.E.: Automated discovery of structural signatures of protein fold and function. Journal of Molecular Biology 306, 591–605 (2001)