Science of Computer Programming ••• (••••) •••–•••

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

Logic/Constraint Programming and Concurrency: The hard-won lessons of the Fifth Generation Computer project

Kazunori Ueda

Department of Computer Science and Engineering, Waseda University, 3-4-1, Okubo, Shinjuku-ku, Tokyo 169-8555, Japan

ARTICLE INFO

Article history:
Received 10 July 2016
Received in revised form 29 May 2017
Accepted 5 June 2017
Available online xxxx

Keywords:
Logic Programming
Concurrent Logic Programming
Constraint-Based Concurrency
Fifth Generation Computer Systems project

ABSTRACT

The technical goal of the Fifth Generation Computer Systems (FGCS) project (1982–1993) was to develop Parallel Inference technologies, namely systematized technologies for realizing knowledge information processing on top of parallel computer architecture. The Logic Programming paradigm was adopted as the central working hypothesis of the project. At the same time, building a large-scale Parallel Inference Machine (PIM) meant to develop a novel form of general-purpose computing technologies that are powerful enough to express various parallel algorithms and to describe a full operating system of PIM. Accordingly, the research goal of the Kernel Language was set to designing a concurrent and parallel programming language under the working hypothesis of Logic Programming. The aim of this article is to describe the design process of the Kernel Language (KL1) in the context of related programming models in the 1980s, the essence of Concurrent Logic Programming and Constraint-Based Concurrency, and how the technologies we developed in those days evolved after their conception.

1. Introduction and background

The grand challenge of the Fifth Generation Computer Systems (FGCS) project (1982–1993) was to develop "truly general-purpose computers for the 1990s,[1]" and it was envisaged that key technologies for this goal included

1. symbolic (as opposed to numerical) computation,
2. knowledge (as opposed to data) processing, and
3. large-scale parallel computer architecture,

as well as a new form of Software Science and Engineering.

More specifically, the technical goal of the project was set to developing *Parallel Inference* technologies, namely systematized technologies for realizing knowledge information processing on top of parallel computer architecture [15] (Fig. 1).

The design space of methodologies for bridging parallel computers and knowledge information processing is immense. For this reason, it was considered necessary to set up a *working hypothesis* to conduct research in a coherent manner,

E-mail address: ueda@ueda.info.waseda.ac.jp.

[1] The FGCS project was often referred to as an Artificial Intelligence project, but its overall goal was literally about general-purpose computer systems rather than AI, which was clearly stated from the beginning of the project (see, for example, [15], p. 100).

http://dx.doi.org/10.1016/j.scico.2017.06.002
0167-6423/

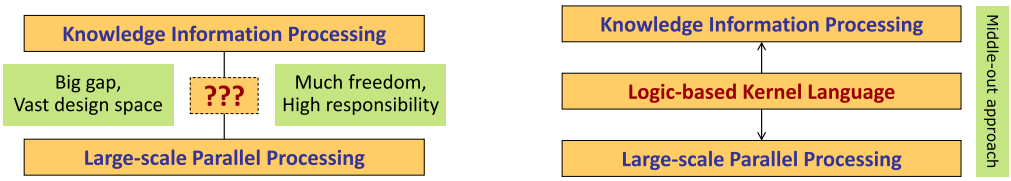

Fig. 1. Goal of the FGCS project (left) and its working hypothesis (right).

and the Logic Programming paradigm was adopted as the working hypothesis. The FGCS project decided to develop a *Kernel Language*[2] based on Logic Programming as the core of systematized technologies for bridging the architecture layer and the application layer. The outstanding feature of the FGCS project lay in this *middle-out*—as opposed to top-down and bottom-up—approach to bridging the big gap between large-scale parallel processing and knowledge information processing. People who worked on the Kernel Language had both much freedom in putting it into shape and high responsibility for the whole project.

1.1. Logic Programming as working hypothesis

There were several reasons why Logic Programming was chosen as the working hypothesis towards the Kernel Language. Firstly, Logic came out as a formalism for representing human knowledge and reasoning. (Note that AI in the early 1980's was under strong influence of Newell's and Simon's Physical Symbol Hypothesis [24]. The mainstream technology of natural language processing in the 1980's was also based on logic.) Secondly, Logic as a declarative formalism was considered promising for exploiting parallelism of underlying computer architecture, say, by not imposing unnecessary sequentiality between the left and the right operands of conjunction and disjunction. The third reason was the connection of Logic Programming to databases and knowledge bases. Finally, Logic was a major language for formal approaches to software engineering including specification, verification, program analysis and transformation.

The project was well aware of, and paid close attention to, Functional Programming, especially Lisp and Lisp machines which were a hot topic in the early 1980's. Compared with Lisp and Functional Programming, Logic Programming did not have full-fledged higher-order constructs, but instead it featured incomplete (partially instantiated) data structures and unification as the basic operation on incomplete data structures. The belief of the project leaders was that the latter was more fundamental than the former and worth exploring in the first place. Potential for various forms of parallel execution was another concern in choosing Logic rather than Functional Programming.

1.2. Kernel Language for parallel inference

When the FGCS project started, the language specification and the implementation techniques of Prolog was reasonably well established already, and the Warren Abstract Machine (WAM), which became the de-facto standard of the implementation technique of Prolog, was under design. However, as mentioned before, the understanding of the project leaders was that realizing a system of technologies for Parallel Inference necessarily meant to develop a new form of *general-purpose* computing technologies that encompass (but are not limited to) knowledge information processing. In particular, being able to describe a *full operating system* for the Parallel Inference Machine (PIM) to be developed in the project and to express and execute various *parallel algorithms* was considered to be a fundamental requirement on the Kernel Language.

Consequently, the research goal of the Kernel Language was set to designing a concurrent and parallel programming language under the working hypothesis of Logic Programming, and soon after I joined the project in 1983, the Kernel Language Task Group started the overall language design with international collaboration (Section 3.1, Fig. 4). After many discussions on the requirement specification of the language, we became convinced by the end of 1983 that Concurrent Logic Programming—more specifically, Concurrent Prolog introduced to us by Ehud Shapiro [27]—was basically the right choice as the basis of the Kernel (as opposed to user-level) Language of the project for its simplicity and expressive power I will describe later.

1.3. Concurrent Logic Programming

The aim of this article is (i) to describe how the Concurrent Logic Programming paradigm was developed as the core principle of the Fifth-Generation Kernel Language, (ii) to convey the essence of the Concurrent Logic Programming paradigm, and (iii) to describe how their research and development evolved after their conception.

Some remarks on the second point are appropriate here. Both Logic Programming (and Constraint Programming as its relative) and concurrent programming (and concurrency theory as its foundation) build upon basic concepts quite different from mainstream programming models such as imperative and functional programming. This requires researchers and

[2] The FGCS project designed and implemented two Kernel Languages, KL0 and KL1, of which this article focuses on KL1 for the Parallel Inference Machine. KL0 was a Kernel Language for the Sequential Inference Machine developed for quick startup of the project.

Please cite this article in press as: K. Ueda, Logic/Constraint Programming and Concurrency: The hard-won lessons of the Fifth Generation Computer project, Sci. Comput. Program. (2017), http://dx.doi.org/10.1016/j.scico.2017.06.002