Parallel Object-Oriented Descriptions of Graph Reduction Machines
(extended abstract)

David Bolton*, Chris Hankin** and Paul Kelly **

*Dept of Computer Science, City University, Northampton Square, LONDON EC1V 0HB.
**Dept of Computing, Imperial College, 180 Queen's Gate, LONDON SW7 2BZ.

ABSTRACT

Abstract machine descriptions of parallel computer architectures must capture communications and concurrency characteristics at a high level. Current design techniques and notations are weak in this respect. We present a layered method for refinement of a requirements specification through to a detailed systems architecture design.

This paper concentrates on the two highest layers, the logical model, which is a requirements statement, and the systems architecture, which specifies logical processes and explicit communications. While requirements are expressed in a language that matches the problem domain, we suggest that a parallel object-oriented notation is most appropriate for the systems architecture layer. Refinements within this layer reflect implementation details (eg. structure sharing and distribution of work among processing elements). We introduce a parallel object-oriented notation based on rewriting systems concepts and use it to refine the design of a parallel graph reduction machine to execute functional programs.

The notation used is a natural extension of a graph rewriting language and the work forms the basis for a structured explication of parallel graph rewriting in which all communications are made explicit.

Keywords: Parallel combinator reduction, graph rewriting systems, object-oriented programming

1. Introduction

In this paper we propose a formally-based methodology for the design of parallel computing systems. The method applies to the logical model which involves a requirements statement and its refinements, and the systems architecture which involves a specification of logical processes and explicit inter-process communication. The systems architecture can be further refined towards a silicon design, but in this paper we focus on the top two layers.
The logical model

The logical model begins with a requirements statement. This and any refinements of the logical model are expressed in a language which matches the problem domain.

Our demonstrator is a parallel combinator reduction machine, and in this particular example, the language of term and graph rewriting systems is the most appropriate. The requirements are expressed as a term rewriting system which describes the normal-order reduction of combinator expressions. The lowest level of the logical model layer uses an annotated graph rewriting system to describe a parallel, packet-based graph reduction mechanism. Verification of the refinements employs the well-understood theory of graph rewriting systems.

The systems architecture

The role of the systems architect is to identify logical processes within the architecture, and the control and data flows between them. Here we introduce a parallel object-oriented notation for use across all applications. The top-level of the layer recasts the lowest level of the logical model in the object-oriented notation; for our example the objects will be packets and processing agents. Lower levels of this layer might incorporate processors as objects allowing us to reflect scheduling and process placement design decisions, and we sketch an example refinement to model load balancing. The correctness proofs at this lower level are not treated in the paper.

The rest of this paper is structured to reflect the different layers of design activity. In the next section we present the requirements statement that the example design is meant to meet. This represents the top-level (most abstract) design within the logical model. In Section 3 we present a graph rewriting system which describes the lowest level of description within the logical model; we identify the correctness criterion for this description. Section 4 contains a description of an object-oriented notation and Section 5 illustrates its use in the description of the top two levels of the systems architecture. We conclude with some suggestions for future work.

2. The Requirements

The example architecture is the Cobweb machine [And87, And88]. This parallel architecture aims to execute functional programs using a large array of processing elements fabricated on a single wafer of silicon.

The arguments for using functional languages in parallel systems have been well-rehearsed elsewhere [Gla84, Pey87, Kel89]. Functional programming systems must implement function application efficiently. During application, some representation of the argument (either a value or closure/suspension) becomes associated with the formal parameters of the function. These associations are used in the evaluation of the body of the function. There are two standard approaches to functional language implementation: environment-based machines (e.g. the SECD machine) and combinator-based machines [Gla84, Pey87]. In environment-based machines the binding of actual parameters to formal parameters is represented by an association list which is