An Approach to Programming Process Interconnection Structures: Aggregate Rewriting Graph Grammars*

Duane A. Bailey
Janice E. Cuny
Computer and Information Sciences
University of Massachusetts
Amherst, Massachusetts 01003

Abstract

We describe a mechanism for generating families of process interconnection structures. Parallel programming environments that support individually programmed processor elements should allow the programmer to explicitly specify the necessary channels of communication at the level of logical abstraction of the algorithm. For highly parallel processors, the specification of this structure with traditional methods can be tedious and error-prone. Aggregate rewriting graph grammars provide a framework for describing families of regular graphs. Using this scheme, the difficulty of specifying an algorithm's communication structure is independent of its size. In addition, we note that scripts of derivation sequences generating different members of a family of structures can suggest an intra-family contracting map.

1. Introduction

When the processing elements of a parallel processor are individually programmed, explicit description of the necessary channels of communication often aids in the correct and efficient implementation of an algorithm. Knowledge of the underlying communication structure of an algorithm can provide redundancy needed for automatic error detection and correction, and it can provide structural information useful in mapping to a target architecture. Only a few parallel programming environments, however, support the explicit specification of communication structures[8,11].

Programmers are most effective when they work at the level of abstraction required by the algorithm. For programmers of parallel algorithms, this means communication structures should be logically depicted as graphs, as these representations usually accompany informal presentations of parallel algorithms. Graphical representations of communication structures serve as a basis for the display of mapping, control and debugging information. Programming environments that support graphical specification are currently very rudimentary: they rely on the programmer

*The Parallel Programming Environments Project at the University of Massachusetts is supported by the Office of Naval Research, contract N00014-84-K-0647. Duane Bailey was also supported by an American Electronics Association ComputerVision fellowship.
to draw each interconnection. This manual process is tedious, error prone and not feasible for large architectures. In addition, there is currently no support for the abstraction of families of communication structures. Description of graph families is necessary because most algorithms are designed in-the-small but are intended for arbitrarily large machines.

In this paper, we present a new form of graph grammar – called an aggregate rewriting graph grammar – and demonstrate its use in the specification of families of regular communication structures. This type of grammar facilitates description of regular structures at the programmer’s level of abstraction. The resulting description is natural, compact and, in the case of recursively constructed graphs, a description that suggests contracting quotient maps[4].

In the next section, we informally describe aggregate rewriting graph grammars. The third section demonstrates the use of these grammars in describing a number of common network structures. The fourth section suggests mapping techniques naturally induced by recursive descriptions. Our final section discusses the use of aggregate rewriting grammars within a programming environment for highly parallel computation.

2. Aggregate Rewriting Graph Grammars

The use of graph grammars in Computer Science has been largely restricted to describing transformations on structures that are easily represented by graphs: databases, derivation trees of a compiler, operations on abstract data types, etc. These systems do not, in general, have the regularity that we would expect to find in process interconnection structures. For our domain, we have been able to define a restricted form of graph grammar that introduces and preserves regularity and thus forms a natural basis for our descriptions.

An aggregate rewriting graph grammar (subsequently, an AR grammar) is a sequential graph rewriting mechanism[3,10]. The subgraphs to be rewritten at each production step are aggregates of nodes – the union of occurrences of a production’s left side – allowing massive, but regular, changes in the structure of the transformed graph.

\[ A_0 \rightarrow A_1 \rightarrow A_2 \rightarrow A_3 \]

Figure 1: The two graphs shown above are equivalent; nodes generated with equivalent labels are identified.

The set of strings, along with valid operations on those strings (such as the operations of concatenation and addition we use in this paper), is determined by the designer of the grammar.