Chapter 1
Algebras of Algorithms, Parallel Computing, and Software Auto-Tuning

ABSTRACT

This chapter gives an overview of programming methods (algebraic, parallel, adaptive, and other) related to the approach of the program design proposed in the book. Algorithm algebras intended for formalized description of algorithms in the form of high-level schemes are considered: Dijkstra's algebra associated with technology of structured programming; Kaluzhnin's algebra for graphical description of non-structured schemes of algorithms; Glushkov's algebra for description of structured schemes, including the facilities for computation process prediction and design of parallel algorithms; the algebra of algorithmics, which is based on the mentioned algebras. The signature of each algebra consists of predicate and operator constructs conforming to a specific method of algorithm design, that is, structured, non-structured, and other. Basic notions related to software auto-tuning are considered, and the classification of auto-tuners is given.

INTRODUCTION

In the following subsections, the programming methods (formal, algebraic, functional, logic, parallel and adaptive) associated with the methodology of software design being proposed in the book are considered.

DOI: 10.4018/978-1-5225-9384-3.ch001

Algebraic programming is a special form of programming activity in which programs are constructed in terms of algebraic objects and their transformations. Objects of a subject domain and reasoning about these objects are represented by algebraic expressions (e.g., terms, predicate formulas, algorithm schemes) in many-sorted algebra (Sannella & Tarlecki, 2012). The transformation of expressions is provided by application of equalities or rewriting rules. A number of systems are known, which support various forms of algebraic programming, such as Casl (Mossakowski, Haxthausen, Sanella, & Tarlecki, 2008), Maude (Meseguer, Olveczky, Stehr, & Talcott, 2002), APS (Letichevsky, Kapitonova, & Konozenko, 1993).

Functional and logic programming (Harrison, 1997; Kowalski, 2014; Kowalski & Hogger, 1992; Paulson & Smith, 1991) belong to the declarative programming paradigm, under which a program describes what result must be received instead of a description of a sequence of its obtaining. A functional program defines a system of rewriting rules that can evaluate the desired function. A logic program defines a search space of problem reductions that can solve all instances of the desired goal.

The development of modern software is characterized by dynamic expansion in construction and usage of parallel computing models, which became ubiquitous and permeate the most aspects of architecture and programming tools of computer systems. Network technologies and Internet facilities, operating systems and application software in present circumstances more or less are based on concepts of parallel and distributed computing.Solving complex scientific and technological tasks requires significant computing resources; sustainable use of these resources has always been one of the main problems in software development. Programming efficient algorithms has become more complicated after the transition to multicore processor architecture. For achieving the maximum program performance, it is necessary to adapt a program to a computing environment in which it will be executed. The modern methodology of software auto-tuning (Durillo & Fahringer, 2014; Naono, Teranishi, Cavazos, & Suda, 2010) allows automating this process. The idea of auto-tuning consists in empirical evaluation of several versions of a program and selection of the best one. Traditionally, the selection is made by a separate tuner program which generates various program modifications. In this chapter, an overview of the main concepts and tools related to software auto-tuning is given.