An XML Specification for Automatic Parallel Dynamic Programming*

Ignacio Peláez, Francisco Almeida, and Daniel González

Departamento de Estadística, I. O. y Computación
Universidad de La Laguna, c/ Astrofísico F. Sánchez s/n
38271 La Laguna, Spain
ignacio.pelaez@gmail.com, falmeida@ull.es, dgonmor@ull.es

Abstract. Dynamic Programming is an important problem-solving technique used for solving a wide variety of optimizations problems. Dynamic programs are commonly designed as individual applications and, software tools are usually tailored for specific classes of recurrences and methodologies. We presented in [9] a methodological proposal that allowed us to develop a generic tool [DPSKEL] that supports a wide range of Dynamic Programming formalizations for different parallel paradigms. In this paper we extent this work by including a new layer between the end users and the tool in order to reduce the development complexity. This new layer consists in a XML specification language to describe Dynamic Programming problems in an easy manner.

1 Introduction

As stated in [2] Skeletal programming has revealed as an alternative and contributes to simplify programming, to enhance portability and to improve performance. Such systems hide the parallelism to the programmer and have been characterized for been embedded entirely into a functional programming language or for integrating imperative code within a skeletal framework in a language or library. Some of these approaches can be seen at [4], [7], [3], [1], [6], [8], [5], [2]. The underlying idea of separating the specification of a problem, or an algorithm, from implementation details that are hidden to the user is present in all the proposals.

In [9] we presented a methodological proposal that allowed us to develop a generic tool (DPSKEL) that supports a wide range of Dynamic Programming formalizations for different parallel paradigms. Parallelism is supplied to the user in a transparent manner through a common sequential interface of C++ classes. The user provides the functional equation as a sequential C++ method. DPSKEL has been developed and a release for shared memory architectures has been validated using a set of tests problems representative of different classes of Dynamic Programming functional equations.

* This work has been partially supported by the EC (FEDER) and the Spanish MEC (Plan Nacional de I+D+I, TIN2005-09037-C02).

V.N. Alexandrov et al. (Eds.): ICCS 2006, Part I, LNCS 3991, pp. 872–875, 2006.

We propose to introduce a new abstraction layer between the user and the skeletons (DPSPEC from now on). This layer separates the fundamental logic behind a problem specification (the functional equation) from the specifics of the particular middleware that implements it, the instantiation using DPSKEL. This allows rapid developments and delivery of new applications. The benefits of the approach are significant to scientists:

- Reduced development time for new applications
- Improved application quality
- Increased use of parallel architectures by non expert users.
- Rapid inclusion of emerging technology benefits into their systems

We will show how this methodology can be applied without any loss of efficiency. The W3C recommendations have also been a requirement of the project. The software architecture of our transformation tools is presented in table 1. The paper has been structured as follows, we present in section 2 the XML specification for DP (DPSPEC), and in section 3 the transformation between DPSPEC and DPSKEL is stated. The paper finalizes with some concluding remarks and future lines of work.

Table 1. Software architecture of the Dynamic Programming transformation tool

Mathematics Editor (Graphical)
MathML, OpenMath, etc.
Transformer to DPSPEC
DPSPEC
Transformer to DPSKEL
DPSKEL
OpenMP, MPI, OpenMP + MPI, etc.
Parallel Architecture

2 DPSPEC a XML Specification for Dynamic Programming Problems

Although some of the XML specifications already existing (MathML, OpenMath, OMDoc, XDF, ...) could be used to specify the DP functional equation, we decided to develop our own specification adapted to DP problems. The main reasons for that decision were to reduce the elements to the minimum and to introduce some changes in the structure on specific elements appearing in classical specifications. These design issues make easier the later parsing and allow a better semantical analysis to detect data dependencies, while keeping as closer as possible to the user defined equation at the same time. The semantical analysis determines the traversing mode of the DP table. DPSPEC brings together the elements to describe piecewise defined functions, simple variables and vectors, arithmetic, logical, relational and max, min operators, and iterators. Table 2 summarizes elements available in the DPSPEC language.

Table 2. Current elements available in DPSPEC

Element	Operation
Main element	`<problem>`
Logical functions	`<and>, <or>, <not>`
Conditional	`<cond>`
Relational operators	`<lt>, <le>, <gt>, <ge>, <eq>, <ne>`
Binary mathematical operators	`<minus>, <divide>, <power>`
N-ary mathematical operators	`<plus>, <times>, <max>, <min>`
N-ary iterative operators	`<imax>, <imin>`
States	`<state>`
Variables	`<ci>`
Constants	`<cn>`
User defined functions	`<functiondef>, <function>`
User defined vectors	`<vectordef>, <vector>`

3 Automatic Parallelization of Dynamic Programming Problems

The automatic parallelization of Dynamic Programming problems is achieved by making explicit transformations on the DPSPEC data file holding the Dynamic Programming equation. The XML description of the formula is converted into a specific instantiation of the DPSKEL C++ skeleton to solve the problem considered. However, scientists typically represent the functional equation as mathematics expressions, using their favorite equation editor (latex, OpenOffice, etc.). Therefore, two transformations are implicitly involved in the process, the conversion of the mathematical equation to the XML specification and the transformation of the XML equation specification into the set of C++ classes. The first transformation step is currently provided automatically for many popular MathML software editors, which generate a MathML document for a given equation. DPSPEC is compatible with MathML and the conversion between a MathML document into a DPSPEC document is achieved through a XSLT preprocessing step.

The second transformation step involves a deeper analysis of the functional equation, we perform a DOM parsing of the XML functional equation to produce the proper C++ required classes of DPSKEL, the parser has been developed using the Xerces-C++ library.

4 Conclusions and Future Work

As a conclusions we can say that we have developed a XML specification language to describe Dynamic Programming problems (DPSPEC). We present a methodology that allows to generate automatically parallel applications. The methodology is based in the existence of general parallel programs (DPSKEL) that can be generated from the XML specification. The code generated is efficient

since no overhead is introduced during the transformation steps. The technique has been validated with a wide range of cases of study. Several extensions to DPSPEC are in the agenda. A natural extension to the language is to support new data types. Once a XML specification has been stated, transformations from/to many other languages can be developed at a reasonably cost. We are also interested in the development of new transformation tools from other languages to DPSPEC, that is the case for example OpenMath and from DPSPEC to other languages such as WSDL to provide the interface for a web service application.

This work has been partially supported by the EC (FEDER) and the Spanish MEC (Plan Nacional de I+D+I, TIN2005-09037-C02).

References

1. M. Aldinucci, S. Gorlatch, C. Lengauer, and S. Pelagatti. Towards parallel programming by transformation: The FAN skeleton framework. *Parallel Algorithms and Applications*, 16(2–3):87–122, 2001.
2. Murray Cole. Bringing skeletons out of the closet: a pragmatic manifesto for skeletal parallel programming. *Parallel Comput.*, 30(3):389–406, 2004.
3. Marco Danelutto and Massimiliano Stigliani. Skelib: Parallel programming with skeletons in c. In *Euro-Par '00: Proceedings from the 6th International Euro-Par Conference on Parallel Processing*, pages 1175–1184, London, UK, 2000. Springer-Verlag.
4. John Darlington, A. J. Field, Peter G. Harrison, Paul H. J. Kelly, D. W. N. Sharp, and Q. Wu. Parallel programming using skeleton functions. In *PARLE '93: Proceedings of the 5th International PARLE Conference on Parallel Architectures and Languages Europe*, pages 146–160, London, UK, 1993. Springer-Verlag.
5. A. J. Dorta, J. A. González, C. Rodríguez, and F. de Sande. llc: A parallel skeletal language. *Parallel Processing Letters*, 13(3):437–448, 2003.
6. E. Alba et al. MALLBA: A library of skeletons for combinatorial optimisation (research note). In *Proceedings of the 8th International Euro-Par Conference*, volume 2400 of *LNCS*, pages 927–932, 2002.
7. Daniel González-Morales, Francisco Almeida, F. Garcia, J. Gonzalez, Jose; L. Roda, and Casiano Rodríguez. A skeleton for parallel dynamic programming. In *Euro-Par '99: Proceedings of the 5th International Euro-Par Conference on Parallel Processing*, pages 877–887, London, UK, 1999. Springer-Verlag.
8. Herbert Kuchen. A skeleton library. In *Euro-Par'02: Proceedings of the 8th Euro-Par Conference on Parallel Processing*, pages 620–629, London, UK, 2002. Springer-Verlag.
9. Ignacio Peláez, Francisco Almeida, and Daniel González. High level parallel skeletons for dynamic programming. *Parallel Processing Letters*, To appear, 2006.