PRETSEL: A PARALLEL REAL TIME SPECIFICATION LANGUAGE*

ALOK CHOUDHARY, V. GEHLOT and B. NARAHARI
Department of ECE & CASE Center, Syracuse University
Syracuse, NY 13244, USA
Email:narahari@seas.gwu.edu

MILISSA BENINCASA, RICHARD METZGER
Rome Laboratory Griffis AFB, Rome, NY, USA
Email:benincas@sybil.se.rl.af.mil

Abstract

In recent years, the research in software support for parallel computers has mainly addressed scientific and information processing applications. Very little attention, if at all, has been paid to real-time embedded system requirements. This research investigates important issues related to designing real-time system software for parallel computers. In this paper, we discuss a formal model for specification of real-time systems on parallel architectures. Towards this end we propose PRETSEL - Parallel Real Time Specification Language - and discuss the computation model, its syntax, and the informal semantic model. PRETSEL extends existing algebraic models by providing structured timing constructs, and explicit parallelism constructs.

1 Introduction

Parallel and distributed computing offers a high speed computing platform for many applications, including those in science, engineering, and command and control. However, lack of software support both in the design as well as in the implementation phases has resulted in slower acceptance of parallel computing than originally expected. Real-time (reactive) systems put even greater requirements on parallel computing software because design criteria must also consider "performance", "guarantees of deadlines", "adaptability to external events", and "fault-tolerance".

In recent years, the research in software support for parallel computers has mainly addressed scientific and information processing applications. Very little

*This research is supported by a grant, contract No. F30602-94-C-0073, from the Rome Laboratory.

attention, if at all, has been paid to real-time embedded system requirements. The objective of this research is to investigate some important issues related to designing real-time system software for parallel computers. In this paper we address formal software specification models for real-time computing in a parallel environment.

A real-time reactive system implemented on a parallel computer would consist of many tasks (each of which can be a parallel task) communicating with each other, some handling input data, others processing data and events and some producing output, and each task using the appropriate number of processors for required computation and performance requirements. These tasks will communicate with each other to exchange information and communicate with the external environment to input (/output) data. Each of these tasks, and the overall computations, must be completed to meet the time constraints demanded by the application.

Much of the past work on formal specification models for real-time systems has dealt mainly with the specification of structural and temporal properties of interacting sequential tasks [14]. The consideration of parallel tasks, which has not been addressed adequately, introduces an extra level of complexity into the problem of specification due to various new issues and factors introduced by parallelism. These include the degree of parallelism, scalability of the parallel algorithm, resource allocation and mapping of the data, and communication and synchronization overheads and requirements. Therefore, in order to specify and verify a real-time system running on a parallel computer, a formal specification model that defines the temporal and structural properties of parallel tasks, and not just sequential tasks, is required. Towards this end we need a formal notation with rigorous semantics for specifying these properties (of real-time parallel processes), and a formalism for verification of
the properties. These will provide the basis for automated/mechanized verification tools.

In this paper we propose a formal specification language called PRETSEL – Parallel REal Time Specification Language – and provide some details of its syntax and a discussion of its underlying semantics. The semantics of PRETSEL operators is based on a parallel computation model, where a system consists of interacting parallel tasks with each task being (for the purposes of this paper) a data parallel algorithm. The next section provides a brief summary of formal models of real-time computation. In section 3 we outline the issues introduced into the problem of specification of real-time computations by parallelism. Section 4 introduces the PRETSEL model, and we show how the operators of PRETSEL model some of the requirements for parallel real-time computing.

2 Formal Specification of Real-Time Systems

A specification language for real-time processes must express hard timing constraints and, ideally, the possible structures of real-time processes. In addition, it should be abstract enough to represent top-level (prescriptive) as well as implementation-level (descriptive) specifications. It should have rigorously defined semantics that reflect the execution of real-time processes. It should model a number of different types of timing constraints that are placed on real-time systems. These timing constraints include deadline times of a process, periodic processes, timeout, duration between instantiation of two processes [3, 8]. The specification of processes normally treats them as consisting of events and actions – an action may be some computations.

Typically, in the area of real-time computing, it is assumed that an event marks an instant in time and has no duration while an action refers to some computation which the real-time process engages. In terms of parallel real-time systems, both events and actions in a more realistic sense may have durations.

A number of promising paradigms for formal specification and verification of real-time systems, in which each computation is executed on a sequential computer, have been proposed and studied. These include CSP (communicating sequential processes), temporal logic, process algebras, Petri-nets, CCS (calculi of communicating systems) and very high level languages (such as SETL, PSETL, and Z) [7, 1, 6, 2, 10, 5, 13]. These formalisms were originally designed for specification and verification of concurrent processes, but since have been augmented to include timing specification which allow them to be used for real-time processes [4, 9, 6, 11, 12].

Depending on the formalism, one can use it either descriptively or prescriptively or both. The former means to give the details of an actual system such as the number of subprocesses involved, their repetitive behaviours and the ways in which the subprocesses interact. The latter means specifying the desired behaviour of a system without specifying how that behaviour is to be obtained. Temporal logic has been use prescriptively, CSP, process algebra, Petri nets, and SETL (and like) can be used both descriptively and prescriptively. In this regard, SETL (and like) have advantage in that it is a real programming language from which an actual implementation can be easily derived. This is usually done by compiling such a language to an application language. Next we give a brief description of some of the existing methods for specification and verification of real-time systems.

Temporal logic [1, 12] is a modal logic that allows one to reason about the truth of statements over time. Temporal logic has been used for specification and verification of program properties. Extensions to temporal logic have been proposed for specification and verification or real-time properties [11].

Timed CSP is an extension of Communicating Sequential Processes (CSP) [7, 4]. In this extension, the prefix operator of CSP is decorated with a time value. Timed CSP assumes existence of a global clock and that events occurrence have zero duration!

A process algebra consists of an algebraic language that is a collection of function symbols or combinators, and a semantic interpretation of this language [6]. The algebraic language can be used as a specification language, and the verification can be performed by using an equational proof system. Timed Process Algebra extends the standard process algebra model by including a distinguished action among its set of actions [6]. This action represents a “clock tick”, and a clock itself is assumed to a process generating ticks.

Extensive work has been done in using Petri-Nets to model concurrent and real-time systems [2]. Timed Petri Nets are again extensions of standard Petri Nets which have been used in the modelling of control flow in asynchronous parallel systems [2]. In the timed version of Petri nets, a time duration is associated with a place and/or transition.

The language SETL belongs to the class of “very high level languages” and is based on general finite sets and maps [13]. SETL is well suited for specifical-
tion and rapid prototyping because it is built on data structures that are powerful aggregates, and thus allows one to say much using very few statements.

The formal methods proposed and developed in the real-time literature are seen to closely model many real-time systems. However, since they were not designed for the explicit purpose of implementing real-time systems on parallel computers, they all do not adequately address issues raised introduced by parallelism into the specification problem. Some of these issues are briefly discussed in the next section.

3 Issues Introduced by Parallel Computers

Parallel computing adds complexities to a real-time system which are normally absent in uniprocessor systems. In real-time systems, performance correctness (i.e., meeting deadlines etc.) is as important as functional correctness. However, performance on parallel computers, now also depends on a number of architectural and algorithmic properties such as number of processors, communication, scalability of algorithms, overhead of scheduling parallelism and synchronization. These characteristics must be adequately specifiable in the formal specification model. The formalism must also provide features to recognize changes in the computations (due to changes in the input data) and thereby respond by allocating additional (or reallocating) resources to meet timing requirements. The necessity of specifying some of the parameters described above is illustrated by the following example which shows how scalability behaviour may require multiple versions of an algorithm to be provided for the same computation.

The performance of a parallel algorithm depends on a number of factors such as degree of parallelism, the data characteristics and data size, and the system characteristics (such as the number and type of processors and the communication channels). Typically the speedup per processor, also called the efficiency, of a parallel algorithm decreases with increasing number of processors (due to more communication). This scalability parameters must be included in the program specification. Different parallel algorithms for the same computation can have different efficiency functions. Consider two different algorithms for the same problem with different performance characteristics - Algorithm 1 and Algorithm 2. As the scalability parameters vary, the type of algorithm to use to meet the performance requirements may vary.

Thus, there is a need for multiple versions/algorithms to carry out a given computation and a specification model must capture the scenario described above by specifying these multiple versions and the precise metrics used for selecting each version. The specification must model the performance (of the algorithm) as a function of the data size, system size, degree of parallelism and other factors. Depending on the state of the system, the properties of the data, and the performance requirement the appropriate algorithm is selected. This ensures predictability of performance.

4 An Overview of PRETSEL

The specification language PRETSEL is an algebraic model, similar to CSP, CCS, and process algebra (and their various timed extensions). It provides constructs for structured timing requirements, parallelism specification and requirements, and functional requirements. It allows specification of scalability attributes, and also provides a "choice" construct to select different algorithms (for the same computation) based on the scalability, resource availability, and performance requirements. The PRETSEL model itself is based on a two level computation model of parallel computing which is described next.

4.1 The Computation Model

A parallel computation is, in general, a collection of interacting tasks, each of which is a parallel algorithm. At this stage in our research, we consider the case where each task is a data parallel algorithm - i.e., a real-time parallel computation is a set of interacting data parallel algorithms. Modelling parallel computations in this manner naturally leads to a two-level specification model. At level 1 we provide constructs for specifying data parallel algorithms, and at level 2 we provide constructs to combine such tasks in a variety of ways. Thus, for example parallelism occurs at two levels - within a task (data parallelism) and among tasks (functional or task parallelism).

A data parallel algorithm consists of three activity phases: (1) input and distribution of data, including a external synchronization step, (2) compute-communicate cycles, and (3) output of data and external synchronization. The distribution of data across the processors, and the time taken by the algorithm is a function of the number of processors and the size of the data. These (number of processors and data size) factors themselves can be specified as part of the al-
algorithm. It is noted that the compute-communicate cycle is a synchronous activity.

The data parallel tasks can be combined (to get a Level 2 process) using a number of operators which reflect different conditions and dependencies among such data parallel tasks. For example, two data parallel tasks may be executed concurrently (pure parallel composition) or may be executed sequentially (a sequential composition) due to some data dependence. To model real-time processes PRETSEL provides timed versions of various operators (contrasts) which are defined in the next subsection.

To be able to model time, we assume all actions are recorded with reference to a global clock. This obviates the need to model clock synchronization at the specification level.

### 4.2 Syntax and Informal Semantics of PRETSEL

The following grammar defines the syntax of PRETSEL expressions. According to this syntax, a real-time parallel process specified in PRETSEL is a string derivable from the variable \( \text{proc} \).

#### Level 2 Syntax

\[
\begin{align*}
\text{proc} & ::= (\text{task}) \\
& \quad | \ (\text{proc})(\text{op2})(\text{proc}) \\
\text{op2} & ::= (\text{basic}\_\text{op2}) \\
& \quad | \ (\text{timed}\_\text{op2}) \\
\text{basic}\_\text{op2} & ::= \mid \mid \\
& \quad | \ + \\
& \quad | \ \triangleright \\
\text{timed}\_\text{op2} & ::= (\text{basic}\_\text{op2})(\text{time}\_\text{spec}) \\
\text{time}\_\text{spec} & ::= < \text{max}\_\text{time} \\
& \quad | \ > \ \text{min}\_\text{time} \\
& \quad | \ [\text{duration}] \\
\end{align*}
\]

A \( \text{task} \) represents a data parallel algorithm/task which at level 2 are combined to form a real-time parallel process. The meaning of various operators at this level is as follows. The parallel composition \( P_1 \parallel P_2 \) denotes a process where two components \( P_1 \) and \( P_2 \) proceed in time independently of each other except for synchronisations. The sequential composition \( P_1 \triangleright P_2 \) denotes a process where the initiation of the second component \( P_2 \) takes place only after the successful termination of the first component \( P_1 \). The choice operator \(+\) in the expression \( P_1 + P_2 \) allows the computation to proceed according to either \( P_1 \) or \( P_2 \) depending on the environmental factors (the effect of which will become clear after we describe the timed operators).
The timed versions of the above operators specify of various timing constraints associated with the computations. Currently, we have three types of timing constraints: (1) \( > \min\text{.time} \), (2) \(< \max\text{.time} \), and (3) \([\text{duration}] \). The first one specifies the minimum time, i.e., lower bound requirement, for the computation. The second specifies the maximum time, i.e., upper bound, for the computation. The third specifies an exact duration interval for the computation.

The timed operators are interpreted as follows. The parallel timed operator \( P_1 \parallel^{<t} P_2 \) (respectively \( > t \)) specifies that the maximum (respectively minimum) that the two processes will take to complete. The parallel timed operator \( P_1 \parallel^{[t_1,t_2]} P_2 \) specifies the earliest \((t_1)\) that one of the components can start and the latest \((t_2)\) that both can terminate. The timed sequential composition \( P_1 \gg^{<t} P_2 \) denotes the maximum time requirement to complete both \( P_1 \) and \( P_2 \). Similarly for the remaining two constructs. The timed choice expression \( P_1 +^{<t} P_2 \) denotes the maximum time \( t \) allowed for the computation. Thus, the choice (of \( P_1 \) or \( P_2 \)) is based on their individual timing requirements. For example, if \( P_2 \) can meet the deadline but \( P_1 \) cannot, then \( P_2 \) will be selected and vice-versa. If both can meet the deadline, then the choice is resolved non-deterministically. This choice operator allows us to specify different versions of an algorithm to perform the same computation, such that the algorithm that meets the deadlines will be selected.

Now consider the level 1 syntax which specifies the data parallel algorithms/task. At this level a task may be parameterised by the system specification. This will allow, for example, scalability parameters to be captured by the model. The system specification can include system specific information such as the architecture characteristics (number, type and speed of processors), input characteristics (size and type of data), the mapping function to illustrate how data is distributed across the processors, and the execution time characteristics which can be the execution time as a function of the scalability parameters. (If the execution time is omitted, then it is implied that this information can be derived from the specification of the task.)

At level 1 our basic unit of computation is an action. Actions may be combined in several ways to form a composite action or a task, as described later. To model real-time behaviour we distinguish between a basic action and a timed action. The later have time specifications associated with them using the same timing constraints defined for the level 2 syntax. For example, send may be one of the basic actions for communication and its timed version send\(<t\) would specify that this communication must be completed within time \( t \).

Basic actions can be categorized as pure computations, pure internal communication (communication within the algorithm), external communication (i.e., for synchronization), and in addition some special actions such as termination and null action. The first three form the three phases of data parallel algorithms defined by our model of computation. The computations can be arithmetic operations. The internal communications would include communication primitives such as the send-receive primitive, barrier synchronization, broadcast, etc.. The external communications would include communication needed with other tasks to exchange data, for I/O activities, and pure synchronization with other tasks.

The basic actions can be combined in parallel using the synchronous parallel operator \& or in sequence using the ; operator. The if operator allows a deterministic choice to be made based on the boolean expression. The while operator allows iterative computations. The time taken by a while operator is derived from the length of the iterations.

4.3 An Example

As an example we consider the following vector computations which must be completed to meet some maximum time deadline \( d \). Let \( X, Y \) and \( Z \) be vectors of length \( n \), and suppose we must compute the two vectors \((X+Y+Z)\) and \((X+Y)\ast Z\). Further, assume that the process \( X+Y \) is computed first and then the result is passed on to two concurrent processes which compute the two desired equations. Let \( Q \) denote the process that computes \( X+Y \), \( S \) denote \( Z+(X+Y) \) and \( T \) denote \( Z\ast(X+Y) \). Let \( p \) be the total number of processors in the system. The level 2 process \( P \) may be specified as follows: \( P ::= Q \gg^{<t} R, R ::= S \parallel T \). Note that \( R \) is simply the two processes in parallel and its job may be to only initiate the two processes. The time to complete task \( R \) is the maximum time to complete \( S \) and \( T \). If \( Q \) takes more than \( d \) time then the process fails to meet the time constraints. If \( Q \) takes \( d_1 < d \) time, then \( R \) must complete within \( d - d_1 \) time. The task \( Q \) may be specified as the data parallel algorithm: \( Q ::= (n,p,\pi,kn/p)Q' \) where \( \pi \) is the mapping function and \( kn/p \) is the execution time given as a function of \( n \) and \( p \) (thus modelling the scalability of \( Q \)). The main body of \( Q' \) will have a loop \( \text{"while } i < n/p; (\text{icom.m_event}); (\text{ext.comm_event})\text{"} \) and is composed sequentially with an internal communication event which in this case can be a barrier syn-
chronization to signal the end of all processors. The external communication will signal completion of \( Q \) and initiate the process \( R \). The process \( Q' \) performs the arithmetic operations of adding the vector elements. Suppose that we had two different algorithms \( Q_1 \) and \( Q_2 \) for computing \( X + Y \), with different execution times. If \( Q \) had to complete within time \( d_1 \), then the level 2 process \( Q := Q_1 +^d_4 Q_2 \) specifies that two different algorithms are available to compute \( X + Y \). Depending on the values of \( n \) and \( p \), the algorithm that meets the deadline, if at all, will be selected as per the meaning of this operator. This illustrates how the effect of scalability of parallel algorithms is accounted for in the PRETSEL model.

In this section we have discussed the informal semantic model for PRETSEL. A formal semantic model is being developed, and this would be an extension of the semantic model underlying CCS and other process algebraic approaches.

5 Conclusions

This paper discussed the problem of formal specification of real-time systems implemented on a parallel machine. Towards this end we proposed the PRETSEL model which allows specification of functional, timing, and performance requirements. PRETSEL is a formal model which incorporates features from CCS, CSP and Process algebra. It deviates from these approaches in the sense that it takes a two level approach and explicitly addresses parallelism issues at a higher and more realistic level. This is to reflect the computation model used commonly in the parallel processing community. We outlined the basic syntax and informal semantics of PRETSEL and illustrated its use by means of some examples. Our ongoing efforts are towards providing a formal semantic model based on operational rules. A formal semantic model together with the operational rules would initiate the development of a semi-automated verification tool based on PRETSEL.

References


