TRANSPUTER-BASED PROGRAMMING OF PARALLEL SYSTEMS WITH DYNAMICALLY VARIABLE STRUCTURE

A. V. Anisimov and Yu. E. Boreisha

The family of transputer-implemented parallel languages (Parallel C, Parallel FORTRAN, etc.) is augmented with dynamic-parallelism software tools. The proposed tools are implemented on a transputer ring with an arbitrary number of transputers. Resource allocation and scheduling problems are solved automatically by a program switch. The capabilities of parallel programming in the proposed languages are similar to Ada.

Parallel systems are characterized by parallel operation and interaction of the component objects. In parallel systems with dynamically variable structure, objects may be dynamically created and destroyed and the connections between the objects are dynamically switchable.

Programming of parallel systems with dynamically variable structure requires software tools for adequate description of dynamic parallelism. An interesting range of such tools is provided by the parallel programming language Ada [1]. Transputer-implemented parallel programming languages (Occam, Parallel C, Parallel FORTRAN, etc.), on the other, virtually do not contain any such tools. The main focus in these languages is on solution of problems with static parallelism. The simplest tools of dynamic parallelism are supported by the apparatus of Parallel Execution Threads and Processor Farms [2, 3].

The PARUS programming technology (PARUS is the Russian acronym for Parallel Asynchronous Recursively-Controlled Systems) has been developed at the Department of Mathematical Information of the Kiev State University for problems of dynamic parallelism. The support environment of the PARUS programming technology is provided by the PARUS programming system [4, 5]. The PARUS programming system (like a transputer-based parallel programming system) extends the basic programming languages (C, FORTRAN, etc.) with parallel programming tools. The emphasis is on software tools for dynamic parallelism.

For the CSP model of parallel computation, which is the basis of Occam and Ada, these tools involve dynamic creation and destruction of parallel processes (using data recursion and control recursion) and dynamic switching of process connections. The PARUS programming system has been implemented in full under RSX 11M OS on PDP11 computers.

The combination of the PARUS programming system with transputer parallel programming makes it possible to augment the parallel execution threads and processor farms of Parallel C, Parallel FORTRAN, and other languages with the new powerful apparatus of dynamic parallelism PARUS (PARCS).

CHARACTERISTIC FEATURES OF TRANSPUTER ARCHITECTURE AND PROGRAMMING TECHNOLOGY

The transputer architecture and programming technology have been developed by INMOS for the design of high-performance parallel computing systems. A transputer is a computer on a chip, which includes a processor, a memory, and communication lines for connection to other transputers. The most common are 32-bit transputers IMS T414 and IMS T800.

By interconnecting transputers, we can create powerful parallel computer systems, which may be used as add-ons for IBM PC/AT, converting ordinary personal computers into high-performance parallel super-minicomputers.

The transputers are interconnected either through four duplex communication channels or by a programmable switch (IMS C004), which ensures complete switching of 32 outputs and 32 inputs. The transputer RAM usually ranges from 256 K to 2 M.

Transputer systems are programmed in high-level languages, such as Occam, Parallel C, Parallel FORTRAN, and others. Virtually all popular high-level languages have been implemented for transputers, with appropriate extension by parallel programming tools. In this sense, we can speak of a parallel Occam programming system.

Let us consider the basic parallel programming techniques supported by the Occam programming system in the main algorithmic languages (e.g., Parallel C).

1. **Cooperating Sequential Processes**

   The parallel system is assembled from nodes with a given number of inputs and outputs. Each node is implemented by a main procedure in C, which uses special library functions supporting input and output of messages.

   Each main procedure is compiled into a set of transputer instructions and is edited by the linkage editor. The result is an executable module (a problem).

   A parallel system is assembled using a configuration description language. This language describes the hardware topology of the transputer system, which is easily adjusted manually, the list of problems used, input and output switching, and the allocation of problems to transputers.

   In the configuration description language, a special program (configurator) constructs an executable file, which can be loaded and executed on the given transputer system.

2. **Parallel Threads**

   Parallel threads can be dynamically created in each problem using reentrant C functions. Each problem has its own domain of local variables and a shared domain of global (static and external) variables. The access to the shared domain is controlled by semaphores. The parallel threads are executed in a pseudoparallel manner on one transputer.

3. **Processor Farms**

   The processor farm formalism automatically provides configuration description and resource allocation for a specialized class of problems with the following features. The parallel system is specified in the form of one master problem and a set of homogeneous worker problems. The master problem generates the stream of tasks for the worker problems, submits it for execution (any free worker problem captures the current task from the stream, executes it, and passes the result to the master problem), and accepts the results.

   A typical example of a parallel problem solved by such an algorithm is image processing.

   Scheduling and resource allocation (the master problem is loaded into the transputer root; one worker problem is loaded into each remaining transputer in the network) are handled automatically on a transputer network of arbitrary topology.

**TRANSPUTER-BASED PARUS PROGRAMMING SYSTEM**

The PARUS tools extend the basic algorithmic language with dynamic parallelism tools, tools for recursive creation of problems, and dynamic switching of problem connections.

Resource allocation and routing are handled automatically by a software switch. It relieves the user of the need to resort to the configuration description language.

By analogy, we can say that the parallel programming capabilities of PARUS using the basic algorithmic languages are comparable with the parallel programming capabilities in Ada.

The PARUS system is used in the following way (we consider the example of Parallel C). A finite number of base algorithms are identified in the parallel system being developed, and these algorithms are realized in the form of procedures in a specific algorithmic language. These algorithms are then applied to dynamically (possibly recursively) create parallel processes as copies of a given template. The number of parallel processes is not limited. Once the corresponding procedure ends, the parallel process stops.