Parsec's Parallel Programming System for transputers

A complete development system

Parsec's parallel C system offers the software needed to make full use of the flexibility in the hardware offered by the Inmos transputer. The system is based on the C language, with a small number of powerful and easy-to-use extensions to enable concurrent processes and inter-process communications. These are explained on the following pages.

The Par.C System contains an intelligent loader, able to deal with any network of transputers. This loader makes it possible to change the size and configuration of the transputer network and run the same program, without having to re-compile or re-link it. Also, no configuration of the software is needed before running the same program on a different network.

The use of resident libraries for floating point primitives, which are linked at runtime, makes it possible to load the same program on T414 and T800 transputers. Where needed, special code for a T800 can be generated, using faster interleaved code for the CPU and FPU.

Each concurrent process is given full access to host system services through the Par.C file I/O system. It is also possible to use the link directly when no special file support is needed.

No special hardware requirements

The system can be purchased in a native version running on the transputer, using a loader/server on the host system. Servers for a variety of host systems are available. The Par.C System can also be purchased in a cross-system version, running on the host system and merely downloading programs to the transputer network.

The Par.C Parallel C Compiler

Parsec's Par.C System makes it possible to write parallel programs in an extended version of the C programming language. The extensions are few, but very powerful and easy to use. The flexibility of standard C is maintained in the extensions, allowing the programmer to use the properties of the transputer in a dynamic way.

The Par.C compiler recognizes K&R C, including the extensions described in "C: a reference manual" by Harbison and Steele. The runtime libraries conform to the ANSI C standard and have been implemented to support concurrency.

The language extensions in parallel C are explained on the following pages.

```
channel ChannelName;                         /* declaration of channel */
channel *ChannelPointer;                     /* declaration of channel */
channel ChannelArray[SIZE];                     /* array of channels */

ChannelName = VariableName;                   /* output using channel */
VariableName = ChannelName;                    /* input using channel */
*ChannelPointer = VariableName;               /* output using pointer */
VariableName = *ChannelPointer;                /* input using pointer */

ChannelName2 = (type)ChannelName1;          /* input directly to output */
```

Figure 1. Syntax of the channel datatype.

1. The channel datatype

A channel is a point-to-point connection between two processes, which can only be used for synchronous communication. This means, that both processes must have arrived at the corresponding communication statements before the datatransfer takes place. Both processes are descheduled until the message has been properly transferred.

A channel can be used in both directions and for different messages. Any type of variable can be transferred using the assignment operator. The size of the message is determined by the size of the variable sent. The programmer should take care to check on correspondance of the sent and expected messages on both sides. Assigning one channel directly to another channel will cause the message input from the channel on the right to be transferred over the channel on the left immediately. In that case a typecast is needed to determine the size of the message.

2. The par construct

The par construct is used to start a number of concurrently running processes on the same transputer. Each (compound) statement can be any piece of code, including other par constructs. Each (compound) statement is started as a separate process. The process containing a par construct will await termination of all started processes, before proceeding execution of the code following the par. In this way, the closing brace of the par construct functions as a synchronization point, and the results of all started processes in that par can be used after this point.

The header of the par construct may include a replicator of the form used in a for statement. This replicator will cause each (compound) statement within the par body to be started a number of times as separate concurrent processes.

```
par [replicator] {                           /* the replicator is optional */
   (compound) statement;                 /* concurrent process number 1 */
   ......
   (compound) statement;                 /* concurrent process number n */
}                                                /* synchronisation point */
```

Figure 2. Syntax of the par construct.

```
select [within <expression>] {                      /* optional "within"-part */
   alt guard ChannelPtr : <code>                    /* alternative number 1 */

   .........
   alt guard ChannelPtr : <code>                    /* alternative number n */
   alt cond <boolean expression> : <code>              /* default alternative */
   alt timeout : <code>                               /* timeout alternative */
}

alt <replicator> guard &ChannelArray[i] :                  /* replicated alt */
alt cond <expression> guard &Channel :                    /* conditional alt */
```

Figure 3. Syntax of the select construct.

3. The select construct

The most complicated of the extensions to the standard C language in Par.C is the select construct. In short, it is used to wait for the first in a number of specified possible events to occur. The event that triggers the select determines which piece of code will be executed next.

In each alt(ernative) clause, a replicator can be used to set a number of guards. Also, it is possible to 'switch' alternatives on and off, using a conditional expression. A special case of this is the use of a default alternative, in which the guard-part of the alt clause is left out. This default alternative will cause the select to be triggered even when no events have occurred.

The select will wait for one of a number of events to occur. If one wants to limit this waiting, this should be done using the "within expression" part in the header of the select. The result of the expression (which may be a constant) is taken as the maximum waiting time for the select, measured in processor clockticks. If a timeout alternative is defined, the code specified in this alternative will be executed when no other events have occurred before expiration of the specified maximum waiting time. The timeout clause may contain a conditional, but no replicator.

Example program

In the example program listed on the next page, a par construct is used to start a number of transmitting processes on the one hand, and one receiving process on the other. The receiving process uses a select construct to wait for one of the channels in the array to become active, after which the value is read and displayed, and the boolean switching this channel in the replicated alternative is set to FALSE. When no channel becomes active within a specified amount of time, a timeout-message will be displayed. When all messages have been received, the program will terminate.

Support for parallelism

Parsec's Par.C System contains everything one needs to actually run the program on a network of transputers. The Par.C Loader-Server, before loading the program into the network, first does a fast analysis of that network, and thereby makes it possible to load the same program into each kind of transputer network, without having to recompile or relink the code. The information, including type and processorspeed, the amount of external memory, the link-connections to other transputers and to the host system, is available to the program. In this way, it is possible to have programs adjust themselves to changes in the hardware.

```
/*********** EXSEL.C: Demonstration program for the Par.C System **********/

#include <stddef.h>                                      /* General definitions */
#define MX          10
#define Delay       1000
#define TimeOut     10

main() {
   int        i,j=3,N=MX,Again=1,nC=0;
   channel    C[MX];                                     /* Array of channels */
   int        NotYet[MX];                                /* Array of booleans */

   for (i=0 ;i<MX ; i++)                                 /* Set all to TRUE */
         NotYet[i] = TRUE ;
   par
   {{                                                    /* 1st process outer par */
         int   i;
         par (i=0 ; i<MX ;i++)
         {{
               int   r = i * i ;
               printf("Parallel process nr %d started\n",i) ;
               wait ( (Delay/MX) * (MX+Extra-i) );
               C[i] = r ;
               printf("Sent %d over channel at address%p\n", r, &C[i]);
         }}
         printf("End of replicated PAR\n");

   }{                                                    /* 2nd process outer par*/
         while (Again)
         select within TimeOut * (j+1)
         {
               alt (j=0;j<MX;j++) cond NotYet[j] guard &C[j] :
                     printf("Got %d from C[%d] at %x\n",C[j],j,&C[j]);
                     NotYet[j] = 0;
                     nC++;                               /* Number of channels ready */
                     break;
               alt timeout :
                     printf("Time Out in Select\n");
                     break ;
               alt cond (nC >= MX) :
                     printf("All messages have been received\n");
                     Again=0;
         }
         printf("End SELECT loop\n");
   }}
   printf("Exiting program\n");
}
```

Figure 4. Listing of an example Parallel C program.

Availability of the Par.C System

The Par.C System is designed to enable working with transputers from a wide variety of host systems and for a wide variety of transputer systems.

As for the host systems, basically two different configurations are available:

Par.C running on the transputer

The Par.C System as native system running on the transputer connected to the host system. The only component of the Par.C System that has to be ported to different hosts is the Loader/Server.

Available versions: IBM-PC, SUN 3, Harris and a transputer running HELIOS or Parsytec's Megatool. Other versions can be procuded on demand.

It is possible for users to obtain the standard C source code for the Loader/Server, to be able to adjust it to special demands.

Par.C running on the host system

Working with the Par.C System as cross-compiler on the host system and downloading the produced programs to a transputer system.

Available versions: IBM-PC, SUN 3 and Harris. Other versions can be produced on demand.

Prices for the Par.C System (in Dutch Guilders)

Par.C native System, excl. Loader/Server	ƒ 3175.00
Par.C Loader/Server for IBM XT/AT compatibles	ƒ 725.00
Par.C Loader/Server for other host systems	(on demand)
Par.C cross System for IBM XT/AT compatibles	ƒ 3900.00
Par.C cross System for SUN 3	ƒ 5900.00
Par.C cross System for other systems	(on demand)
(The Par.C cross System includes the Loader/Server)	

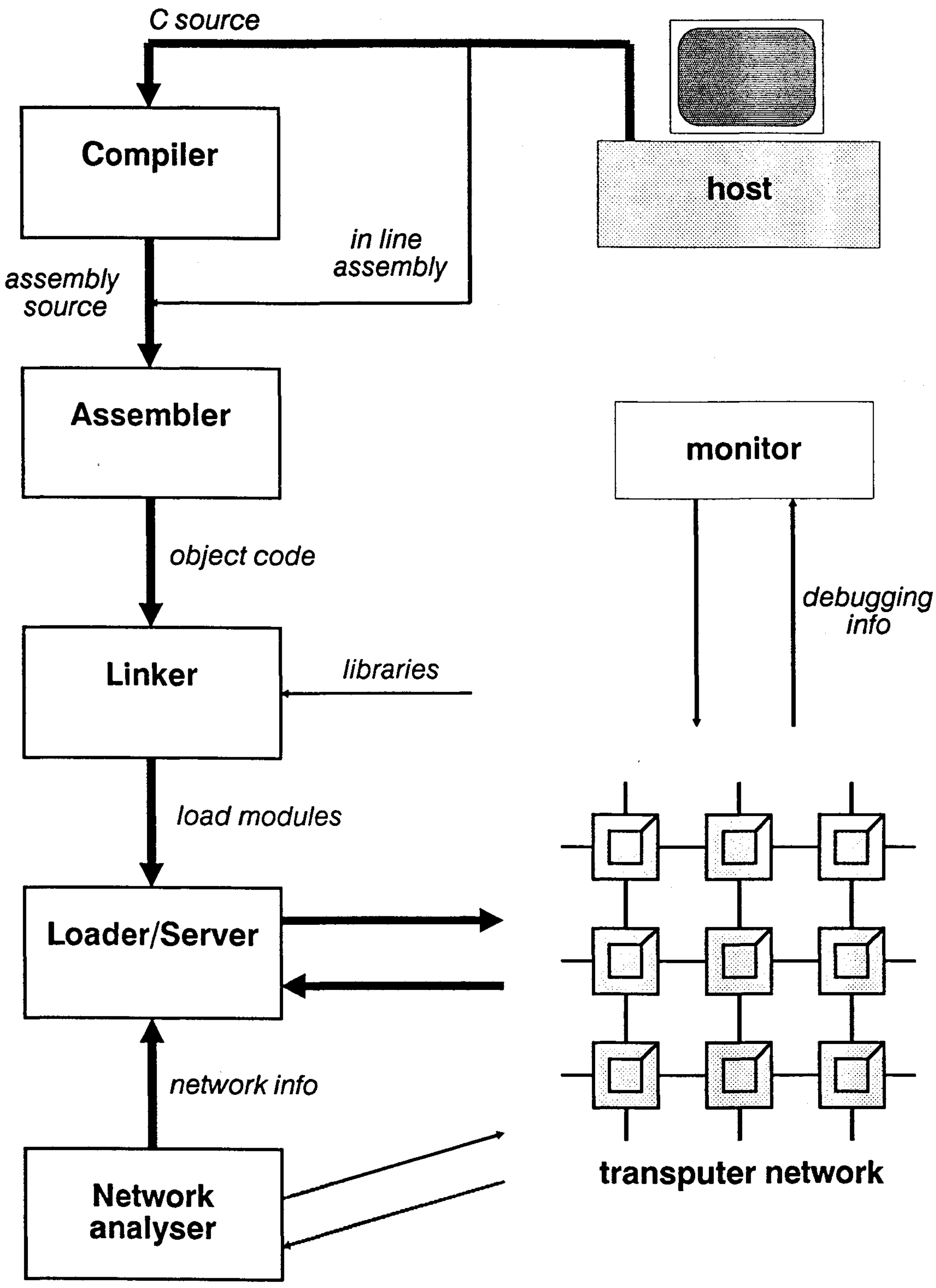
All prices are subject to change without notice.

More information

Anyone who wants to receive more information on the Par.C parallel programming System, and on future developments by Parsec, is welcome to contact Parsec at the address given in the box. You will then receive information on updates on a regular basis. You can also call us.

Parsec Developments
P.O. Box 782
2300 AT Leiden
Netherlands
Phone: (+31) 71 142 142
Telefax: (+31) 71 134 449

Overview of the Par.C Parallel Programming System



The 'Par.C' Parallel C compiler is a product of Parsec Developments, Leiden, the Netherlands.