Parallel C Version 2.2.2
Release Note

3L Ltd.

August 11, 2000

1 Introduction

This Release Note accompanies version 2.2.2 of 3L Parallel C for the Inmos transputer, and outlines the changes in Parallel C since version 2.1. The most important changes are:

- The compiler can now generate programs for 16-bit transputers such as the IMS T212.
- The compiler has been drawn closer to the ANSI standard.
- Changes have been made both to the compiler and the run-time libraries to increase the speed of generated programs.
- The master task of a processor farm may "broadcast" a message to all the worker tasks.
- The linker has been completely rewritten.
- The General Purpose Configurer is faster, and generates much smaller application image files.
- The configurer also supports sub-networks within the larger main network.
- Changes have been made to correct various errors.

V2.2.2 is released with a new edition of the *Parallel C User Guide*. This Release Note should be read in conjunction with that publication.

1.1 Notes

1.1.1 Stricter Type-Checking

Certain expressions involving non-standard combinations of types were previously wrongly accepted by the compiler. These will now be flagged as errors at compile time. This change reduces the chance of run-time errors caused by type incompatibilities. A few programs which took advantage of the previous version's laxer policy may now need to be corrected.

1.1.2 Integral Promotions

The compiler now follows the ANSI standard in performing the "integral promotions" before using integer-type values in an expression. In a very small number of cases, this may lead to changes in the value of expressions. This is discussed further in section 3.2 of this Release Note.

2 Installation Procedure

This version of Parallel C is supported by a new interactive installation program. This ensures that installation will be possible for all versions of DOS.

Details of the installation procedure will be found in chapter 1 of the new edition of the *User Guide*. It is important to follow this procedure, rather than simply copying the files. It is worthwhile performing the confidence test described in chapter 2 as well.

3 The Compiler

3.1 General Changes

- Variables of type `short` and `unsigned short` are now 16 bits wide by default. A new compiler switch, `/Gs`, will cause the compiler to generate 32-bit `short` variables, as before.

  Users should note that if program modules which were compiled with different length `short` variables attempt to communicate using `short` variables, problems will arise. In particular, the current versions of library functions, such as `printf` and `scanf`, expect short variables to be 16 bits wide. If problems of this sort arise, the user is advised to recompile programs, including function libraries, with the new version of the compiler.

  A new built-in macro `_3L_SHORT_BITS` has the value 16 or 32, depending on the width of `short` variables in the current compilation.

- The maximum length of a source line, once macro expansions have been performed, has been increased from about 1 KB to about 4 KB.

- The compiler can now generate object files for 16-bit transputers, such as the T212. A new command, `t2c`, is provided to compile T2 programs; full details can be found in chapter 8 of the *User Guide*.

- Previously, the limit on the total number of dimension specifications that may be used in all the array declarations in a module was about 1000. This limit has now been increased to more than 32,000.

- By default, previous versions of the compiler output the information required by Tbug and the decoder to associate source lines with the right parts of the binary code. In V2.2.1, this information will only be output if the compiler's `/Zd` switch is used. The `/Zi` switch has the same effect, as well as causing the compiler to output information for Tbug relating to variables.

Previous versions of the Parallel C compiler recognised the `/Zd` switch, but it had no effect.

3.2 ANSI features

- Previous versions of the compiler only allowed string literals with less than 256 characters. This restriction has been removed. The length of string literals is now limited by available memory and logical line length but will be about 4 KB at least.
- Expressions of the form `z++->a` are now allowed. Previously, the compiler followed K&R C in prohibiting them.
- The compiler now recognises the `signed char` data type.
- The `defined` unary operator, as specified in section 3.8.1 of the ANSI standard, is now recognised by the preprocessor.
- The ANSI standard prescribes an important change in the way in which the types of values are converted for use in expressions.

  The compiler now follows the standard by performing the "integral promotions" on integer-type values before they are used in an expression. This means that if the whole range of values of the type can be represented by an `int` it is converted into an `int`; otherwise it is converted into an `unsigned int`.

  Only after this has been done are the necessary conversions for evaluating the expression performed. In particular, if at this stage one of the operands is an `unsigned int` and the other an `int`, the `int` will be converted to `unsigned int`.

  This will make a difference to the value of expressions only in a small number of cases. For example,

  ```
  unsigned char c = 5;
  int a, b = -1;
  a = (c > b);
  ```

  With previous versions of the compiler, `a` was assigned the value 0 (false). Now it will be assigned the value 1 (true).

  Full details of these conversions may be found in section 3.2.1 of the standard. Users may also be interested in the corre-

sponding section of the *Rationale* for the standard, where this change is described as "the most serious semantic change made by the Committee to a widespread current practice".

Note that the integral promotions will treat `unsigned short` values differently for the T2 and T4/T8; for the former, they will become `unsigned int`, and for the latter, `int`.

- The `#elif` preprocessor directive has been implemented.
- Preprocessor directives may now be preceded on a line by white space, and the initial '`#`' character need not fall in the first column.
- The suffixes to floating constants ('`f`', '`l`, '`F`', '`L`') and to integer constants ('`u`', '`l`, '`U`', '`L`') are accepted by the compiler, although such suffixed constants are not at present treated differently from unsuffixed ones.
- The ANSI trigraph sequences, as described in section 2.2.1.1 of the standard, are now accepted by the compiler.
- Previous versions of the compiler followed K&R C by performing all floating-point arithmetic in double precision by default. The current version follows ANSI by performing floating-point arithmetic which involves only `float` values in single precision. (The user should bear in mind that a floating-point constant is always double-precision.)

  A new switch `/Gd` is provided to make the compiler use double-precision arithmetic as before. The old switch `/S`, which used to make the compiler use single precision where possible, is still recognised but has no effect.
- Adjacent string literals are now treated as a single literal. For example, the following two statements have the same effect:

```
p = "Hallo, world";
p = "Hallo, ""world";
```

  Note, however, that although ANSI allows white space to occur between the two literals, the compiler does not at present accept this.

- If a function-like macro name appears and is not followed by a left parenthesis, the macro will not be expanded. For example, in the following, `sqr` will not be expanded:

```
#define sqr(a) a*a
b = sqr+1;
```

- The compiler wrongly assumed that the result of the '`~`' (tilde) operator is always `int`. This is not always true, for example if the operand is an `unsigned int`.
- The compiler now allows expressions of type `float` to be used as control expressions; for example, in `if` statements. Expressions of type `double` were already permitted.
- It is now possible to select the elements of a structure value returned by a function.
- The compiler now allows the second and third operands of the `?:` conditional operator to be both of type `void`.
- Both K&R and the standard specify that the result of a cast operator is not an lvalue; it cannot, for example, appear as the left operand of an assignment operator. This is now enforced by the compiler.

3.3 Performance Enhancements

The speed and size of the code generated have been improved in many ways. The following changes are the most obvious to the user.

- The compiler's treatment of `switch` statements has been improved.
- The amount of diagnostic and other "red tape" information output by the compiler has been reduced, especially when a module contains no static variables. Recompiling the run-time library in this way has reduced the size of the minimal linked program from about 10 KB to about 8 KB.
- Further improvements in the compilation of `switch` statements.

- The compiler will now generate in-line code instead of function calls for the following functions whenever possible. This "in-lining" will not happen, however, if the compiler `/Gi` switch is used, or if the appropriate header file is not included.

```
abs                 ceil                fabs
floor               fabs                modf
memcpy
chan_in_byte        chan_in_message
chan_in_word
chan_init           chan_out_byte
chan_out_message
chan_out_word       chan_reset
thread_deschedule   thread_priority
thread_restart
thread_stop
timer_after         timer_delay
timer_now
timer_wait
```

4 Run-Time Libraries

4.1 General Changes

- Certain headers could not be included more than once without multiple definitions arising. This has been corrected.
- Certain low-level functions (e.g. `lseek`) used to output error messages to the console on their own account. This has been stopped.
- A stand-alone run-time library for supporting T2 programs has been implemented.
- The method used to invoke exit handlers registered by `atexit` (see section 4.10.4.2 of the ANSI standard) has been improved to cope with the possibility that an exit handler might itself call `atexit`.

- The function provided by the startup routine for exiting a program has been renamed `_exit`. This function should not normally be used directly; instead, the function `exit`, which is held in another module, should be used.

  This change has been made so that vendors of pre-processors to the C compiler may replace `exit` with their own alternative `exit` functions, which should finish by calling `_exit`.

- The library header files have been altered so that all follow the same style. In particular, all use ANSI function prototypes and do not include dummy identifiers for the arguments, only type specifications.

4.2 ANSI Features

- The facilities defined by the ANSI `stdarg.h` header are supported, along with the more traditional `varargs.h` facilities.
- The `printf`, `fprintf` and `sprintf` functions now follow the standard.
- The standard `strtod` function has been implemented.
- The `offsetof` macro, as described in section 4.1.5 of the standard, has been implemented.
- The `EXIT_SUCCESS` and `EXIT_FAILURE` macros are now defined in `stdlib.h`, as described in section 4.10 of the standard.
- The `tmpnam` function, described in section 4.9.4.4 of the standard, has been implemented.
- The `div_t` and `ldiv_t` types, as described in section 4.10 of the standard, and the `div` and `ldiv` functions, as described in section 4.10.6, have been implemented.
- The `tmpfile` function, described in section 4.9.4.3 of the standard, has been implemented.
- The `vprintf`, `vsprintf` and `vfprintf` functions described in section 4.9.6 of the standard have been implemented.
- Definitions of the `FOPEN_MAX` and `FILENAME_MAX` macros have been added to `stdio.h`.

- The `scanf`, `sscanf` and `fscanf` functions now comply with the ANSI standard.

  Note that in ANSI C `%E` is defined to be equivalent to `%e`, and so returns a `float`. In previous versions of the compiler, `%E` used to return a `double`. This may cause problems in existing programs. `%le` or `%lE` should now be used instead to return a `double`.

  Similarly `%D`, `%O`, `%X` return `int` where they used to return `long`. This will not be a problem for existing programs as these types are currently equivalent.

  `%F` is not defined by ANSI and so continues to return a `double`. This code is included for compatibility with previous versions and is not recommended for new programs.

- A definition for the standard macro `FLT_ROUNDS` has been added to `<float.h>`. The appropriate value for the transputer is 1, which means "round to nearest".

- The `fgetpos` and `fsetpos` functions defined in section 4.9.9 of the standard have been implemented. Users should note, however, that at present they support only binary files.

  The `setvbuf` function defined in section 4.9.5.6 of the standard has been implemented, but always returns a non-zero value to indicate that the request to change the buffering method cannot be honoured.

  The necessary prototypes and macros to support these functions have been added to `stdio.h`, as have the `SEEK_SET`, `SEEK_CUR` and `SEEK_END` macros needed by `fseek`.

- The `strcoll` and `strxfrm` functions defined in section 4.11.4 of the standard have been implemented.

- Implement support for multi-byte characters. At present, every multi-byte character is one byte in length. There is no state-dependent encoding.

  The following functions, described in section 4.10.7 of the standard, are available: `mblen`, `mbtowc`, `wctomb`, `mbstowcs` and `wcstombs`. The macros `MB_CUR_MAX` and `MB_LEN_MAX` are both defined with the value 1.

- A minimal `locale` functionality, as described in section 4.4 of the standard, has been implemented. At present, only the locales `"C"` and `""` are available. For both, the values for the members of the `lconv struct` returned by `localeconv` are as defined in section 4.4 of the standard.

  The `setlocale` and `localeconv` functions have been implemented, as required by section 4.4.

- A definition of the type `ptrdiff_t` has been added to `stddef.h`, as required by section 4.1.5 of the standard.

- A basic version of the `signal` facility, as defined in section 4.7 of the standard, has been implemented. The only signals are those listed on page 121 of the standard; that is, `SIGABRT`, `SIGFPE`, `SIGILL`, `SIGINT`, `SIGSEGV` and `SIGTERM`. None of these happen automatically, but only as a result of a call to `raise` or, in the case of `SIGABRT`, to `abort`. The default action selected by using the macro `SIG_DFL` as a parameter to the `signal` function is to ignore the signal.

- The standard "wide characters" facility has been implemented, although at present the type `wchar_t` (see section 4.1.5 of the standard) is defined as equivalent to `char` and each wide character occupies one byte.

  Wide character constants (e.g., `L'a'`) and wide string literals (e.g., `L"hello"`) are accepted, but are treated like the corresponding non-wide elements.

- The standard specifies that if a null pointer is passed to `free`, no action should occur. If a null pointer is passed to `realloc`, the equivalent of a call to `malloc` should be performed (see section 4.10.3 of the standard). These have been implemented.

- The `assert` macro has been changed so that it does not make implied reference to the contents of `<stdio.h>`.

4.3 Performance Enhancements

- The speed of the `realloc` function has been improved, in most cases by a factor of more than 10.

- The speed (in terms of CPU use) of the functions `fread` and `fwrite` has been improved.
- The speed of the `strcmp` function has been greatly improved.
- The module `atexit` is no longer included in the linked program unless it is used.
- Following section 4.10.1.1, the `atof` function is now implemented by a call to `strtod`, instead of using its own code. This reduces the size of the libraries.
- Similarly, `atoi` and `atol` are now implemented by calls to `strtol`.
- The `strtod` function has been altered so that it makes use of `ldexp` rather than `pow`. This requires the addition of fewer run-time library modules to the program, and is faster.

5 The Linker

The linker has been entirely rewritten. Amongst the new facilities are:

- Support for the interactive source-level debugger, Tbug.
- Binary files and libraries containing T2 object code can now be linked.
- Modules can be selected by reference to external names they contain for placing at the beginning of the image, so that they may be placed on on-chip RAM, if available.
- The files in the link list may contain more than one definition of a single external name. In this case, the linker selects the first occurring. In this way, files in a library can be overridden by routines with the same name appearing earlier in the link list.
- The command line has been simplified by permitting space as a connecting character in addition to '`+`' and assuming filename extensions for the various types of files accessed.

- At least 150000 external references may now be made within a single image.

The batch files which call the linker have also been enhanced to allow more than one object file to be specified. You may also specify linker switches.

6 The General Configurer

The General Configurer, `config`, has been improved in several ways.

- If a user declares multiple tasks all of which have the same image file, the contents of that file will appear only once in the application file.
- The number of copies of the loading software included in the application file has been much reduced.
- If the user places more than one identical task on the same processor, they will now share one copy of the code for the task.
- The PROCESSOR statement has a new BOOT attribute. This enables the user to specify sub-networks within a larger main network, which may, for example, contain T2 processors.
- The PROCESSOR statement also has a new RAM attribute, which forces the loader to assume the specified size.
- Filenames specified in configuration files are now treated as case-significant.
- The configurer has been changed so that, by default, the information needed by Tbug is not stored in the application file. This results in smaller application files, faster loading and less use of memory.

  The information is not normally needed because the debugger loads configured applications from the original task files, not from the application file. However, an option switch, `/K`, has

been provided, which will cause the debugging information to be passed to the application file as before, if the need arises.

7 Broadcasts in Processor Farms

The `net_broadcast` function has been added to the `net` package to enable the master task of a processor farm application to send a message to every worker task in the processor farm. It should not be used by any worker task. A synopsis for this function, using the format used in the *User Guide*, would look like this:

```
#include <net.h>
int net_broadcast(int n_bytes, char *packet);
```

`nbytes` is the number of bytes of data in the buffer pointed to by `packet`. This function is unlike `net_send` in that the `nbytes` parameter is *not* restricted to `NET_MAX_PACKET_LENGTH`. This means that the programmer does not have to split the message up into packets; this is done by `net_broadcast`. The worker tasks receive the message by calling `net_receive` in the usual way, possibly several times; `net_broadcast` ensures that the when the last packet is read, the `complete` argument is set to 1 as usual.

`net_broadcast` can only be used when all the worker tasks are known to be idle. Typically, this would be at the beginning of the program run, before any work packets have been sent out. Later, the master task can broadcast new data, provided a result packet has been received corresponding to every work packet sent out.

8 Miscellaneous

- The stub filer task did not support `SetReturnCode.Cmd`. This has been corrected.
- The `install` program and the external driver program have been compressed using `LZEXE`.

- The `decode` program now displays the contents of the debug areas in a more easily understood way.