C* 7.2 Release Notes

Part I: Overview

1 Introduction

Version 7.2 is a new release of the CM-5 C* compiler. It provides several new features, summarized in Section 4, as well as performance enhancements and bug fixes.

These release notes replace all previous C* release notes.

2 Hardware and Software Requirements

2.1 Hardware Required

Version 7.2 runs on the CM-5 and CM-5E Connection Machine systems, with our without vector units, and on Sun-4 workstations.

These features, new in Version 7.2, require a CM-5 with vector units:

- support for the global/local programming model
- support for 64-bit integers
- math-library intrinsic functions

Copyright © 1995 Thinking Machines Corporation

2.2 Software Required

Version 7.2 requires CMOST Version 7.3 or higher.

Printing or displaying 64-bit integers in Prism requires Prism Version 2.2 or later. Otherwise, you can debug C* 7.2 programs using Prism 2.0.

3 Porting Information

Object files compiled under previous releases of the compiler cannot be linked with object files from Version 7.2. All source files making up a C* executable program must be recompiled and relinked with the latest compiler.

4 New Features

4.1 New Functionality

Version 7.2 contains these new features:

- support for 64-bit integers; see Section 7
- support for the global/local programming model; see Section 8
- new functions, specially recognized by the compiler, that perform bit-level operations such as **`leadz`** and **`popcount`**; see Section 9
- several new compiling/linking options; see Section 10

Copyright © 1995 Thinking Machines Corporation

4.2 Performance Enhancements

Version 7.2 contains the performance enhancements listed below. All except the improved performance of contextualization operations apply only to C* programs compiled to run on systems with vector units.

- Many functions in the C* math library have been made intrinsics; that is, the compiler specially recognizes calls to these functions and produces more efficient code that eliminates function-call overheads. This increases the speed of these functions considerably when they are used with parallel operands. These functions are affected:

```
acos
asin
atan
atan2
cos
cosh
exp
fabs
fmod
log
log10
pow
sin
sinh
sqrt
tan
tanh
```

  You need not do anything different to obtain this speedup; it happens automatically when you include **<math.h>** and call one of the listed functions.

- The performance of all load operations of 4-byte data (**int** and **float** types) is faster. The performance of store operations of 4-byte data inside an **everywhere** statement is also faster. This improves the performance of both parallel computation and communication. Note that in both cases, however, the importance of using **everywhere** (when possible) to increase performance persists.

- The **-O** compiler option is supported with the 7.2 release. When **-O** is specified, the compiler performs some optimizations of parallel expressions and, in particular, optimizes parallel computation inside an **everywhere** statement.

Copyright © 1995 Thinking Machines Corporation

- Contextualization operations are faster in many cases. This affects **`where`** statements and context through the **`&&`**, **`||`**, and **`?:`** operators (when they have parallel operands). Contextualization remains slow when a function call, a communication operation, or a **`pcoord`** operation is made inside the **`where`** statement or as a child of the other operators. For more information, see Section 11.

5 Documentation for This Release

Documentation for Version 7.2, in addition to these release notes, consists of the following documents:

- *CM-5 C* Programming Guide*, Version 7.2
- *CM-5 C* User's Guide,* Version 7.2
- *Getting Started in C**, May 1993

Sample programs are available in **`/usr/cstar-7.2/examples/cs`** by default. If they aren't there, check with your system administrator for their location at your site.

6 Errors, Feedback, and Assistance

6.1 Bug Update File

To learn about restrictions in this release, see the on-line bug-update file, which by default is in **`/usr/doc/cstar-7.2.bugupdate`**. If this file doesn't exist on your system, check with your system administrator.

6.2 Request for Feedback

Users are encouraged to communicate with Thinking Machines as fully as possible. Please report any errors you find in the software (or documentation) and suggest ways to improve it.

Copyright © 1995 Thinking Machines Corporation

Part II: Detailed Information about New and Changed Features

7 Support for 64-Bit Integers

C* Version 7.2 supports the use of the 64-bit integer data type in programs compiled with the **-vu** option (that is, compiled to run on systems with vector units). Both parallel and scalar 64-bit **int**s are supported.

The preprocessor symbol **__LONG_LONG__** is predefined for programs compiled with **-vu**. It indicates that the 64-bit integer types are allowed.

For complete information on 64-bit integers, see Section 5.5 of the *CM-5 C* Programming Guide*, Version 7.2.

8 Global/Local Programming

C* Version 7.2 provides support for a programming model in which a global C* program can call local C* functions running on the individual nodes of a partition. This is known as the *global/local* programming model. The local functions can communicate via calls to the CMMD message-passing library.

The global/local programming model lets you obtain the advantages of both data parallel and message-passing programming.

You might want to do local programming

- if you want explicit control (via CMMD) over the communication in your program, because use of the global data parallel model is too restrictive.

Copyright © 1995 Thinking Machines Corporation

- if use of the standard data parallel model is creating too many code blocks (and thereby degrading performance) because your program has an intricate but local control flow.

You might want to use C* global/local programming rather than simply using C and CMMD in order to take advantage of the vector units.

Note these points with regard to global/local programming in C*:

- Global/local programs can run only on CM-5s with vector units.
- Global/local programming is also available in CM Fortran. Issues involved in C* global/local programming are similar to those involved in CM Fortran, but the interface is not identical.

For complete information on global/local programming, see Appendix D of the *CM-5 C* Programming Guide*, Version 7.2.

9 Integer Inquiry Functions

C* Version 7.2 includes new inquiry functions that provide bit-level information about integers. The functions are

Function	Use
`leadz`	Returns the number of leading 0 bits in the bit-level representation of a value.
`leadz_nz`	Like **`leadz`**, except its behavior is identical to the DPEAC **`ffb`** call.
`dimulh`	Returns the high 64 bits of the multiplication of two signed integer operands.
`dumulh`	Returns the high 64 bits of the multiplication of two unsigned integer operands.

Version 7.2, May 1995
Copyright © 1995 Thinking Machines Corporation

Function	Use
`popcnt`	Returns the number of 1 bits (the population count) in the bit-level representation of an integer.
`poppar`	Returns 0 if the population parity of the argument value is even, 1 if odd.

All functions are overloaded for 32-bit and 64-bit integers, and for scalar and parallel types. (Overloadings for 64-bit integers are available only when compiling for execution on the vector units.)

If you use any of these functions, include the header file **`<bitop.h>`**.

For complete information on these functions, see Appendix E of the *CM-5 C* Programming Guide*, Version 7.2.

10 New Compiler Options

C* Version 7.2 provides several new options to the **`cs`** command for compiling and linking.

10.1 Compiling Local Functions: The −local Option

Use the **`-local`** option in the global/local programming model before each file that is to be compiled as a local function. See Appendix D of the *CM-5 C* Programming Guide*, Version 7.2, for more information.

10.2 Creating Optimized Code: The −O Option

Use the **`-O`** option when compiling to produce optimized code. When **`-O`** is specified, the compiler performs some optimizations of parallel expressions and, in particular, optimizes parallel computation inside an **`everywhere`** statement.

Copyright © 1995 Thinking Machines Corporation

10.3 Don't Link with ANSI Library: The −noansilibs Option

C* Version 7.2 by default links with an extra library that provides support for some Standard C functions that aren't available in the Sun libraries. See Section 5.5 of the *CM-5 C* Programming Guide*, Version 7.2, for more information.

If you don't want to link with this library, use the option **`-noansilibs`** when linking with C*.

10.4 Don't Link with the New printf Library: The −noprintflibs Option

When compiling for the vector units, C* Version 7.2 brings in library support that allows the **`printf`** and **`scanf`** family of functions to work with 64-bit integers; see Section 5.5 of the *CM-5 C* Programming Guide*, Version 7.2. It does this by replacing the internal functions **`_doprnt`** and **`_doscan`** in the C libraries with ones that have this new support. If you want the ordinary internal routines instead, use the **`-noprintflibs`** option when linking.

11 Performance Improvements for Contextualization

As mentioned in Section 4.2, C* Version 7.2 has improved performance for **`where`** statements and the **`?:`**, **`&&`**, and **`||`** operators. This section goes into more detail about how to achieve the best performance for these operations.

Version 7.2, May 1995
Copyright © 1995 Thinking Machines Corporation

11.1 The ?:, &&, and || Operators

The following factors affect the performance of these operators:

- Whether the compiler can tell that the surrounding context is **everywhere**. A surrounding **everywhere** improves performance in most cases because it lets the compiler generate code that doesn't read the current context from memory.
- Whether the new context has to be computed. An assignment operation in the second child of the **||** or **&&** operator or in the second or third child of the **?:** operator causes context to be computed.
- Whether the new context has to be stored to parallel memory. These operations, as children of the operators, require the context to be stored to parallel memory:
  - function calls
  - communication operations (parallel left or right indexing)
  - **pcoord**

Thus,

- *Worst performance* occurs when there is an assignment, function call, communication operation, or **pcoord** in the RHS of one of these operators.
- *Better performance* occurs when there are no such operations in the RHS.
- *Best performance* occurs when, in addition, the operator is in a surrounding **everywhere** block.

In the "best performance" cases, the **||** and **&&** operations do not compute context, and the **?:** operation performs a vector merge operation.

11.2 Where Statements

The performance characteristics of **where** statements are similar to those of the **&&**, **||**, and **?:** operators. Thus, you should

- Put the statement in an **everywhere** block, if possible.

Copyright © 1995 Thinking Machines Corporation

- Avoid **pcoord**.
- Avoid communication operations.
- Avoid assignments that are not at the statement level. For example, avoid code like this:

```
where (condition) {
    x = (y = z + 1);
}
```

  Instead write

```
where (condition) {
    y = z + 1;
    x = z + 1;
}
```

If you observe these constraints, the **where** compiles as a vector merge operation.

Further constraints apply to **where** statements with an **else** clause; once again, if you observe them, the **where** compiles as a series of vector stores of merges. The constraints are

- No scalar assignments except in the last statement of the **else** clause.
- The lvalues of the assignments in each path must be syntactically identical.
- The **where** clause and the **else** clause must coincide in *every* lvalue. Thus, code like this will achieve the best performance:

```
where (condition) {
    i1 = i2 + i3;
    i3 = i2 + i4;
    i2 = 7 + i1;
    i1 += 3;
} else {
    i1 = i3 * 6;
    i3 = i2;
    i2 *= 4;
    i1 += 3;
}
```


Version 7.2, May 1995
Copyright © 1995 Thinking Machines Corporation