The
Connection Machine
System

Paris Release Notes

Version 5.0
February 1989

Thinking Machines Corporation
Cambridge, Massachusetts

First printing, February 1989

Thinking Machines Corporation
245 First Street
Cambridge, Massachusetts 02142-1214
(617) 876-1111

Contents

Customer Support

Thinking Machines Customer Support encourages customers to report errors in Connection Machine operation and to suggest improvements in our products.

When reporting an error, please provide as much information as possible to help us identify and correct the problem. A code example that failed to execute, a session transcript, the record of a backtrace, or other such information can greatly reduce the time it takes Thinking Machines to respond to the report.

To contact Thinking Machines Customer Support:

U.S. Mail:	Thinking Machines Corporation Customer Support 245 First Street Cambridge, Massachusetts 02142-1214
Internet Electronic Mail:	customer-support@think.com
Usenet Electronic Mail:	harvard!think!customer-support
Telephone:	(617) 876-1111

For Symbolics users only:

The Symbolics Lisp machine, when connected to the Internet network, provides a special mail facility for automatic reporting of Connection Machine system errors. When such an error occurs, simply press Ctrl-M to create a report. In the mail window that appears, the To : field should be addressed as follows:

```
To:  bug-connection-machine@think.com
```

Please supplement the automatic report with any further pertinent information.

1. About Paris Version 5.0

Connection Machine System Software Version 5.0 is built on a new and significantly expanded parallel instruction set. Several important design goals have been achieved in this release:

- *Better performance.* For most applications, the new software achieves faster performance, more efficient memory management, and better memory utilization. Many instructions have been reimplemented to improve overall system performance.

- *More sophisticated memory management.* User-allocated data sets are managed through abstract *fields* that maintain information about the location and size of data elements residing in CM processors' memory. This form of memory management uses CM memory more efficiently and can thus handle problems with even larger data sets than before. Data access through field-ids also permits more safety checks and more flexibility in managing memory.

- *Better utilization of CM-2 hardware.* New instructions have been added that make better use of the CM-2 hardware, including the router and the floating-point accelerator.

- *Easier programming.* New classes of instructions have been implemented that make Paris programming easier and provide a higher level of programming support. Consistent and regular naming conventions for instructions and arguments also simplify Paris programming.

1.1. Summary of New Features

These major new features of Paris Version 5.0 distinguish it from earlier versions:

- *More flexible virtual processor architecture.* The new virtual processor architecture supports multiple sets of virtual processors within a program. Virtual processor sets of different sizes can coexist in a program and can be created and destroyed under program control.

- *Support for* n*-dimensional NEWS addressing.* Static, 2-dimensional NEWS addressing has been replaced by dynamic, *n*-dimensional addressing in a rectilinear coordinate system of up to 31 dimensions specified by axes ordered from **0** through **30**, inclusive. The number of dimensions and the size of each dimension in the coordinate system can be changed under program control.

- *New classes of communication instructions.* New classes of instructions combine structured communication between processors with arithmetic and logical operations. Multiple-direction **scan**-like instructions called **spread**s can replicate data or perform reductions along several dimensions simultaneously; also, **scan** instructions now support all combining operations.

- *Expanded transcendental math library.* Many basic transcendental functions (including sine, cosine, and others) have been added to Paris.

- *Alphanumeric instruction names.* Paris instruction names no longer contain special symbols that conflict with the naming conventions of high-level language interfaces. There is now a simple and direct correspondence between Lisp names and those used by other languages.

- *Faster intraprocessor array access.* The new software exploits existing hardware support for fast array-indexing operations on arrays that are allocated each within a single processor. The new operations simultaneously access a possibly different element of each array within each processor.

- *Enhanced routing instructions.* The routing instructions (e.g., **send** and **get**) have more combiners and use special hardware features that enhance their overall performance for high virtual processor ratios.

- *Compatibility with earlier releases.* Version 5.0 supports all documented instructions provided in Version 4*.x*. Certain Version 4*.x* instructions have been superseded by newer instructions with more logically-chosen names and enhanced behavior, but the old names are still supported for this release.

- *Fortran/Paris interface.* The Paris object library includes a Fortran-callable interface to the Paris instructions. The instructions can be called from CM Fortran on a VAX front end, VAX Fortran on a VAX front end, and UNIX Fortran 77 on a Sun front end.

1.2. Status of Layered Products

- *Lisp fully supports and uses the new virtual processor architecture, *n*-dimensional NEWS, and the new Paris instructions.
- The C* compiler generates code for Version 4*x* Paris instructions only. C* programs must run in back-compatibility mode, as C* does not take advantage of either the new virtual processor architecture or *n*-dimensional NEWS.
- The CM Fortran compiler generates calls to some Version 5.0 Paris instructions, but CM Fortran programs must be executed in back-compatibility mode as they must use the previous virtual processor system. CM Fortran programs use *n*-dimensional NEWS for efficient operations on multidimensional arrays, but the relevant instructions are not the documented Paris NEWS instructions. (The operations that CM Fortran generates use masks rather than geometries to describe the coordinate system .) These restrictions will be removed in the next release of CM Fortran.
- The DataVault mass storage system uses the new Paris features. Programs that use the DataVault may be run in Version 5 mode or in back-compatibility mode (the lowest level interface to the DataVault depends only on processor cube addresses).
- The CM graphic display system uses the new Paris features. Whether programs that use display instructions run in Version 5 mode or in back-compatibility mode depends on the language from which the display instructions are called.

1.3. Porting Paris Code

Back-Compatibility Mode

Any existing programs that call Paris instructions must be recompiled and relinked with the new Paris object library and must be run in back-compatibility mode. Back-compatibility mode implements the 4*x* stack discipline by allocating the stack in field zero and making stack addresses offsets into this field. See the *Front-End Systems Release Notes*, Version 5.0, for information on executing programs in back-compatibility mode.

1.4. New and Changed Paris Instructions

Most Paris Version 4.*x* instructions have Version 5.0 equivalents; some do not. The 4.*x* names will work in 5.0 programs, with the exception of instructions based on the old NEWS scheme and of those based on the old memory management scheme. See Appendix A of the *Paris Reference Manual*, Version 5.0, for a list of 4.*x* instructions and their corresponding 5.0 equivalents. The same appendix describes the naming conventions introduced with Version 5.0, highlights new instructions, and identifies all obsolete 4.*x* instructions.

1.5. Obsolete Memory Management Instructions

In Version 5.0, Paris manages memory by means of a field allocation mechanism rather than through direct manipulation of a stack. The following Version 4.*x* stack manipulation instructions are supported in back compatibility mode only. Their continued use is strongly discouraged.

CM:push-space
CM:pop-and-discard
CM:set-stack-pointer
CM:set-stack-limit
CM:set-stack-upper-bound
CM:get-stack-pointer
CM:get-stack-limit
CM:get-stack-upper-bound
CM:reset-stack-pointer

Any existing code that uses the above 4.*x* instructions should be converted as soon as possible to use the 5.0 replacement instructions. The obsolete 4.*x* instructions will be removed in a future release of the software.

To allocate space on the stack, use the 5.0 instruction

CM:allocate-stack-field

To deallocate stack space previously allocated, use the 5.0 instruction

CM:deallocate-stack-through

See the Concepts section of the *Paris Reference Manual*, Version 5.0, for a discussion of fields. See Appendix A of the *Paris Reference Manual*, Version 5.0 for a complete list of obsolete Version 4.*x* Paris instructions.

1.6. Changes Unique to C/Paris

New C/Paris Header File

The C/Paris functions and types are now declared in the header file **cm/paris.h**. All function and type declarations previously declared in **cm/cm.h** are still supported (and are still available by including that header file).

C/Paris Type Names Changed

The following C/Paris types have changed with Version 5.0.

Version 4.*x* type	Version 5.0 type
CM_memaddr_t	**CM_field_id_t**
CM_cubeaddr_t	**CM_sendaddr_t**

The old types are still defined and the compiler will accept them (although lint will complain), but this may not be true in the future.

1.7. Changes Unique to Lisp/Paris

Elimination of Stack-Relative Addressing

The Lisp/Paris function **CM:stack**, which performed stack-relative addressing, is no longer supported under the new virtual processor field-addressing scheme.

2. Implementation Restrictions

2.1. Maximum Message Length

The constant **CM:*maximum-message-length*** has been defined to be 128. This constant is an upper bound on the number of bits transferred between processors by certain **send** instructions.

- The limit on message length applies to the following Version 4.*x* **send** instructions:

  CM:send
  CM:send-with-overwrite
  CM:send-with-logior
  CM:send-with-logxor
  CM:send-with-logand
  CM:send-with-add
  CM:send-with-max
  CM:send-with-min
  CM:send-with-unsigned-max
  CM:send-with-unsigned-min

- The maximum message length restriction also applies to the following Version 5.0 router instructions:

  CM:send-with-f-max-1L
  CM:send-with-f-min-1L
  CM:send-with-f-add-1L
  CM:send-aset32-overwrite-1L
  CM:send-aset32-u-add-1L
  CM:send-aset32-logior-1L
  CM:get-aref32

- The following new Version 5.0 **send** instructions have *no* message length restriction; their message size is limited only by available memory:

  CM:get-1L
  CM:send-1L
  CM:send-with-overwrite-1L
  CM:send-with-logxor-1L
  CM:send-with-logior-1L
  CM:send-with-logand-1L

CM:send-with-u-min-1L
CM:send-with-u-max-1L
CM:send-with-s-min-1L
CM:send-with-s-max-1L
CM:send-with-u-add-1L
CM:send-with-s-add-1L

2.2. Incomplete Support for IEEE Floating-Point

Support for IEEE floating-point instructions and flags is somewhat lacking in Version 5.0. In particular:

- the five floating-point flags are not supported
- denormalized numbers are not supported
- **Infinity** and **NaN** values are only partially supported

Also, all Version 5.0 floating-point instructions:

- set the *integer* overflow flag if overflow occurs
- set the test flag in response to an invalid operation
- produce a zero result on underflow, with no other indication

3. Implementation Errors

In Parallel contains descriptions of known Paris implementation errors. Published monthly between releases, *In Parallel* provides up-to-date information including bug reports and programming hints. Please see the Paris section in each *In Parallel* issue published since the release of Version 5.0.

4. Documentation Discrepancies

4.1. Unimplemented Instructions Documented

The instructions listed below, although they are not implemented, are documented in the accompanying *Paris Reference Manual*, Version 5.0.

CM:u-add-carry-3-1L
CM:u-add-carry-3-3L
CM:s-s-power-3-3L
CM:{u,s}-move-const-always-1L
CM:{f,u,s}-rank-2L
CM:s-f-signum-2-2L
CM:s-s-signum-1-1L
CM:u-isqrt-1-1L
CM:aref-2L
CM:aset-2L

Implementation of all but the last two instructions listed above is anticipated for Paris Version 5.1.

CM:aref-2L and **CM:aset-2L** are not implemented, except under the 4.*x* names **CM:aref** and **CM:aset**. These instructions are very slow and should be avoided. As a consequence of the poor performance of these general array instructions, it is unlikely that **CM:aref-2L** and **CM:aset-2L** will be implemented in any future release.

Although more strictly restricted in allowable array parameters, the new, faster array instructions, **CM:aref32** and **CM:aset32**, should be used for array manipulation. To make this easier, an array transposition instruction, which would convert arrays from and to a sideways representation, is planned for Version 5.1.

4.2. Documented and Implemented Names Differ

The following list shows the correspondence between Paris instructions that, with the release of Version 5.0, are documented under one name and implemented under an-

other. They work as documented, although called by a different name. These inconsistencies are expected to be resolved with Version 5.1.

Implemented	Documented
CM:my-send-address-1L	**CM:my-send-address**
CM:swap-2-1L	**CM:swap-1L**
CM:send-aset32-logior-1L	**CM:send-aset32-logior-2L**
CM:send-aset32-overwrite-1L	**CM:send-aset32-overwrite-2L**
CM:send-aset32-u-add-1L	**CM:send-aset32-u-add-2L**
CM:float-move-decoded-constant	**CM:f-move-decoded-constant-1L**

4.3. Operand Order Switched in VP Set Memory Allocation Instructions

The order in which operands to **CM:allocate-stack-field-vp-set** and **CM:allocate-heap-field-vp-set** are to be specified is documented as *vp-set-id, len*. However, as implemented, these instructions expect their arguments in the opposite order.

4.4. Zero Length Operands

All Paris operations on unsigned integers are documented to permit *length* operands of value zero. However, as implemented, some do support zero *length* operands and some do not. Giving an unsigned instruction a *length* operand of value zero will cause obvious errors in some cases, will cause subtle errors in other cases, and will work correctly in still other cases. It is therefore inadvisable to pass zero *length* operands to operations on unsigned integers.

Zero *length* operands are generally not useful and therefore this inconsistency should not prove troublesome. If a workaround is needed, provide a one-bit field containing zero in each processor. It is uncertain whether this restriction will persist in the future.