The
Connection Machine
System

Paris Release Notes

Version 5.2
October 1989

These release notes
replace all previous
Paris release notes

Thinking Machines Corporation
Cambridge, Massachusetts

First printing, October 1989

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

Contents

About Version 5.2 Paris Documentation

Intended Audience and Objectives

The Paris language and its documentation are intended for experienced developers of Connection Machine system software and applications. Version 5.2 Paris documentation is published to inform Paris programmers about new and modified Paris features.

Revision Information

The Version 5.2 Paris release notes and supplementary documentation supersede all previous Paris release notes and all past editions of the Paris sections of *In Parallel* software bulletin.

- Release notes distributed with Versions 5.0, 5.1A Field Test, 5.1, 5.2A Field Test, and with any earlier releases are now obsolete and should be removed from the *Programming in Paris* binder.
- *In Parallel* editions published before October 1989 should also be discarded.

Although these release notes replace the Version 5.1 Paris release notes, they do *not* replace the supplementary documentation that was published with the Paris release notes Version 5.1.

- The *Paris Reference Manual Supplement* Version 5.1, behind the tab labeled "Supplement," is not replaced by any 5.2 Paris documentation.
- The *Scientific Subroutines* Version 5.1 document, behind the tab labeled "CMSSL,"is not replaced by any 5.2 Paris documentation.

  Release of a Version 5.2 CM Scientific Subroutines Library is anticipated soon after installation of the CM System Software Version 5.2. At that time, new CMSSL documentation will replace the CMSSL documentation shipped with Version 5.1 Paris documentation.

Organization of Version 5.2 Paris Documentation

Paris Release Notes Version 5.2

The release notes broadly describe new and changed Paris features and detail language restrictions. Outstanding implementation and documentation errors are also reported.

Change Pages to Paris Reference Manual Supplement, Update for Version 5.2
The supplement change pages correct previous documentation errors. Directions for inserting the change pages into the current Paris dictionary are included.

Change Pages to Paris Reference Manual, Update for Version 5.2
The reference manual change pages correct previous documentation errors and document Paris instructions that are new or changed with Version 5.2 Directions for inserting the change pages into the current Paris dictionary are included.

Related Documents

- *Paris Reference Manual* Version 5.0, printed February 1989
- *Paris Reference Manual Supplement* Version 5.1, printed June 1989
- *Introduction to Programming in C/Paris* Version 5.0, printed June 1989
- *In Parallel*, Software Bulletin, Volume III, to be published after October, 1989

Notation Conventions

The table below displays the notation conventions used in the Paris documentation.

Convention	Meaning
boldface	Language elements, such as keywords and instruction names, when they appear in instruction formats or embedded in text.
italics	Operand names and placeholders, when they appear in instruction formats or embedded in text.
`typewriter`	Code examples and code fragments.

All Paris Version 5.2 documentation follows the conventions for alphabetizing, syntax, and pseudocode established at the beginning of Chapter 9 of the *Paris Reference Manual* Version 5.0. One new convention has been introduced.

In the Formats portion of dictionary entries, brackets, [and], enclose arguments that are either not provided, optional, or keywords in the Lisp/Paris interface. Wherever this notation is used, the Operands list states explicitly whether the brackets enclose unprovided, optional, or keyword arguments. For example, in the format line

In the Formats portion of dictionary entries, brackets, [and], enclose arguments that are either not provided, optional, or keywords in the Lisp/Paris interface. Wherever this notation is used, the Operands list states explicitly whether the brackets enclose unprovided, optional, or keyword arguments. For example, in the format line

Formats result ← **CM:intern-geometry** *dimension-array, [rank]*

the *rank* operand is not provided when calling Paris from Lisp.

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:	ames!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 5.2

The Paris Language is the Connection Machine assembly language. Paris is currently the lowest-level instruction set available for programming the Connection Machine. It provides a large number of operations similar to the machine-level instruction set of a serial computer. Paris is intended primarily as the basis for higher-level Connection Machine languages such as *Lisp, C*, and CM Fortran. It may nonetheless be called directly from standard Lisp, C, or Fortran or from *Lisp, C*, or CM Fortran code.

Paris Version 5.2 supports 64-bit floating point Connection Machine hardware, provides an expanded parallel instruction set, and corrects a number of implementation errors present in Versions 5.0 and 5.1.

1.1 New Hardware Supported

The primary purpose of Connection Machine System Software Version 5.2 is to support new hardware features for the Connection Machine model CM-2. These hardware features include larger CM memory chips and 64-bit floating-point accelerator chips.

Larger Memories

In the past, CM-2 configurations included 64K bits of memory per physical processor. Now, with the release of CM System Software Version 5.2, it is possible to configure a CM-2 with 256K bits of memory per physical processor.

The impact of software support for larger memories is seen in a small performance loss under certain circumstances when using Version 5.2.

An overall loss in execution speed of approximately 10% may be expected at low VP ratios. A low VP ratio is defined as 4 or fewer virtual processors per physical processor, but varies depending on the speed of the front end. With a fast front-end computer, no degradation of performance occurs at a VP ratio of 4; with a slower front end, using a VP ratio of 4 does result in a slight performance loss.

The performance cost of larger CM memories is the result of the increased front-end overhead that is required to manage the larger address space. This cost is incurred even by Connection Machine systems without the 256K memories.

64-Bit Floating-Point

Previously, the CM-2 model Connection Machines included one 32-bit floating-point accelerator for every 32 CM physical processors. Now, the Connection Machine model CM-2 may optionally be configured with 64-bit floating point accelerator units (fpu's). This new hardware feature is supported by the CM System Software Version 5.2.

The impact of software support for 64-bit floating-point accelerators is seen in both performance gains and performance losses when using Version 5.2. These changes in performance are best analyzed for the Paris instruction set. Be aware, however, that the high-level Connection Machine programming languages manifest corresponding performance changes because they are built on Paris.

For the vast majority of Connection Machine Paris instructions, performance gains and losses are in the 1% to 2% range. The substantial performance gains and losses exhibited under Version 5.2 are reported below.

Double-Precision Floating-Point Performance

Before Version 5.2, all CM double-precision floating-point operations were simulated in software and were consequently quite slow. With the introduction of 64-bit floating-point hardware, these operations are very fast.

On machines configured with 64-bit fpu's and running CM System Software Version 5.2, double-precision floating-point instructions are approximately 20 times faster than on machines without 64-bit fpu's.

On machines configured with 64-bit fpu's and running CM System Software Version 5.2, double-precision floating-point operations are approximately half as fast as single-precision floating-point operations. The only exceptions are the double-precision tangent, arctangent, arcsine, and arccosine instructions, which are approximately a third as fast as their single-precision counterparts.

Single-Precision Floating-Point Performance

The performance impact of CM System Software Version 5.2 on single-precision floating-point operations is as follows:

- Basic arithmetic without constants is 5–8% slower with 64-bit fpu's than with 32-bit fpu's
- Basic arithmetic with constants is 8–25% faster with 64-bit fpu's than with 32-bit fpu's
- Transcendental functions are about as fast with 64-bit fpu's as with 32-bit fpu's, with a 5% variance in either direction
- Inverse transcendental functions are about half as fast in Version 5.2 as in previous versions

1.2 Summary of New Language Features

In addition to support for the new hardware, these new language features distinguish Paris Version 5.2 from earlier versions:

- *Available memory check.* The new instruction **CM:available-memory** reports how much memory remains available, per virtual processor, for allocation on either the CM heap or stack.
- *An instruction to transpose data.* The new instructions **CM:transpose32-1-1L** and **CM:transpose32-2-1L** turn data slicewise.
- *Geometry serial numbers.* The new instruction **CM:geometry-serial-number** assigns to the specified geometry a unique number suitable for use as a hashing key.
- *More math functions.* Several previously unimplemented functions have been added to round out the instruction set. In particular, new truncate, floor, ceiling, round, compare, and signum functions are included with this release.

Paris features new with Version 5.2 are documented in the packet entitled *Change Pages to Paris Reference Manual,* Update for Version 5.2.

Paris Math Functions New with Version 5.2

With the release of Version 5.2, the following mathematical functions are newly added to the Connection Machine parallel instruction set:

CM:{s,u}-ceiling-3-3L
CM:{s,u}-f-ceiling-2-2L

CM:{f,s,u}-compare-3-2L

CM:{s,u}-floor-3-3L
CM:{s,u}-f-floor-2-2L

CM:u-round-3-3L
CM:u-f-round-2-2L

CM:f-s-scale-{2,3}-2L
CM:f-s-scale-constant-{2,3}-1L

CM:{s,u}-f-signum-2-2L

CM:u-f-truncate-2-2L

Dictionary entries for these functions are included in the change pages packet for Version 5.2. The change pages should be inserted into the Paris dictionary.

1.3 Summary of Changed Language Features

These Paris features existed in Version 5.1 and have been modified in Version 5.2:

- *Access to small portions of shared array elements.* For **CM:aref32-shared**, the lowerbound restriction of 32-bits on the destination field length is lifted. It is now possible to extract 8- and 16-bit portions of shared array elements.

- *Communication now uses* **:send-order** *axes.* Prior to the release of Version 5.2, the instructions **CM:get-from-news{-always}-1L**, **CM:get-from-power-two{-always}-1L**, and **CM:send-to-news{-always}-1L** did not work on send-ordered axes. With Version 5.2, this restriction is lifted.

Version 5.1 Paris features modified in Version 5.2 are documented by the packet entitled *Change Pages to Paris Reference Manual*, Update for Version 5.2.

1.4 Status of Layered Products

High-Level Languages

- CM Fortran fully supports all Paris features, including instructions that are new or changed with Version 5.2.
- *Lisp supports all Paris features, including instructions that are new or changed with Version 5.2.
- The C* compiler generates code for Version 4*x* Paris instructions only. C* programs must run in back-compatibility mode as C* takes advantage of neither the virtual processor scheme nor the *n*-dimensional NEWS facility, which were both introduced with Version 5.0.

Peripherals

- The DataVault may be operated by programs running under Version 5.2, or by programs running in back-compatibility mode.
- The CM Graphic Display System may be run under Version 5.2 features, or in back-compatibility mode.

Libraries

- Standard UNIX Math Library

  As of CM System Software Version 5.1, the standard UNIX math library is not automatically linked with Paris. This remains true in Version 5.2. When linking 5.2 C/Paris code, it is therefore necessary to use the `-lm` switch.

  Updated pages for the C/Paris interface chapter are included in the change pages packet for Version 5.2.

- CM Scientific Subroutines Library

  Release of an expanded CM Scientific Subroutines Library is anticipated soon after installation of the CM System Software Version 5.2.

1.5 Back Compatibility

Version 5.2 supports all documented instructions provided in Versions 4*x* and 5*x* to date.

Back-Compatibility Mode

Any existing programs that call Paris 4*x* 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 5.2 for information on executing programs in back-compatibility mode.

2 Implementation Restrictions

The following restrictions apply to Version 5.0 and all Version 5*x* point releases.

2.1 Maximum Message Length

The constant **CM:*maximum-message-length*** has been defined as 128. This constant is an upper bound on the number of bits transferred between processors by certain router instructions (**sends** and **gets**).

- The maximum message length restriction also applies to the following Version 5*x* 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 Version 5*x* router 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

- The limit on message length applies to the following Version 4*x* router 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

2.2 Incomplete Support for IEEE Floating-Point

Support for IEEE floating-point instructions and flags is incomplete in Version 5.1. In particular:

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

Also, all Version 5.1 floating-point instructions:

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

When overflow occurs, the value stored in the destination field varies depending on the floating-point hardware present. The result may be `0.0`, it may be a quiet **NaN**, or it may be the biased adjusted result specified by IEEE. Similarly, using a **NaN** as an operand to a floating-point instruction yields indefinite results.

3 Implementation Errors

Known implementation errors in Paris Version 5*x* have been reported, most recently in the *Paris Release Notes* Version 5.1 and in the *In Parallel* Software Bulletin Vol. II, No.1, August 1989. Most of the known errors are corrected in Version 5.2. The outstanding bugs are reported again in these release notes. The *Paris Release Notes* Version 5.1 and all past issues of Programming in Paris *In Parallel* may therefore be discarded.

3.1 Known Errors Corrected

The following Paris implementation errors, reported in the *Paris Release Notes* Version 5.1, are fixed in Paris Version 5.2:

cm-get-1l-runs-out-of-mem
no-segment-bits-for-scans
send-to-news-wrong-context

The following Paris implementation errors, reported in *In Parallel* Vol. II, No. 1, August 1989, are fixed in Paris Version 5.2:

array-format-incorrect-for-complex-and-float
cm:f-divinto-const-always-conditional
cm:global-min-inactive-bug
compress-heap-wrong-vp-set
cross-vp-move-errors
deposit-news-constant
doc-bug-CPG-file-missing
doc-bug-CPG-page23
float-const-ops-now-error-on-ints
fortran-paris-only-bugs
get-from-power-two-bug
ieee-to-vax-large-negative-exponents
my_news_coordinate_big_field_vp
rel-2-cube-array-transfers-broken
tanh-bug

3.2 Known Errors Outstanding

Version 5.2 of Paris has some known implementation errors, most of which have been previously reported. The following Paris implementation errors, reported in the *Paris Release Notes* Version 5.1, remain outstanding in Paris Version 5.2:

> **cm-time-overflows**
> **lintlib**
> **negative-field-length**

In addition, there is one previously unreported limitation called **no-segment-bits-for-enumerate-or-rank.**

All known unintentional restrictions for Version 5.2 Paris instructions are reported here in alphabetical order by bug report ID. If new bugs are discovered, they will be reported during the coming months in the *In Parallel* software bulletin, Vol. III.

ID cm-initialize-random-number-generator-name

Environment

Paris, Versions 5.0, 5.0.1, 5.1, and 5.2; any front-end/CM configuration.

Description

This operation is documented under the name **CM:initialize-random-generator**. It is, however, implemented as **CM:initialize-random-number-generator**, which violates the Paris name length limit of 32 characters.

Workaround

Use the implemented name.

ID cm-time-overflows

This was originally reported in *In Parallel* No. 2, February 1989. It was reported again in *Paris Release Notes* Version 5.2.

Environment

Paris, Versions 5.0, 5.0.1, and 5.1; any front-end/CM configuration.

Description

The result returned by **CM:time** can become too large to fit into the 32 bits that are allocated to accumulate and store the total time. When this happens in Lisp/Paris, control is transferred to the Lisp debugger; in C/Paris, **CM_time** returns an incorrect result.

ID lintlib

This was originally reported in *In Parallel* No. 1, January 1989. It was reported again in *Paris Release Notes* Version 5.2.

Environment

Paris, Versions 5.0, 5.0.1, and 5.1; any front-end/CM configuration.

Synopsis

The **lint** version of the Paris library does not work on the VAX front end. There is an ULRIX bug that prevents our **lint** library from working.

ID negative-field-length

This was originally reported in *In Parallel* No. 2, February 1989. It was reported again in *Paris Release Notes* Version 5.2.

Environment

Paris, Versions 5.0, 5.0.1, and 5.1; any front-end/CM configuration.

Description

The field allocation routines, **CM:allocate-stack-field** and **CM:allocate-heap-field**, successfully return when passed negative lengths as arguments—even if safety is on. The negative lengths can later cause a CM exception.

ID **no-segment-bits-for-enumerate-or-rank**

This restriction has not been previously reported.

Environment

Paris, Versions 5.0, 5.0.1, 5.1, and 5.2; any front-end/CM configuration.

Description

The Paris **CM:enumerate-1L** and **CM:{f,u,s}rank** instructions do not accept the **:segment-bit** value for the *smode* operand.

4 Debugging Hint

Here is a hint for effective C/Paris debugging.

ID **paris-safety-hint**

This was originally reported in *In Parallel* Vol. II, No.1, August 1989.

Environment

Paris, Version 5*x*; UNIX front end, using the c-shell; any CM configuration

Description

Paris safety checking can be turned on by default. When the C/Paris library is linked with C code, Paris safety checking is turned off by default. To speed the debugging process, turn safety checking on.

To turn Paris safety checking on by default, add the following line to your **.cshrc** file:

```
if ($?CMDEVICE) cmsetsafety on
```

This line turns Paris safety on each time you use a **cmattach** subshell.

5 Documentation Errors

Outstanding Documentation Errors

A number of documentation errors in the *Paris Reference Manual*, Version 5.0, remain outstanding. A corrected edition of the manual will be published in the future. Meanwhile, Paris programmers are strongly urged to add the corrections suggested here to their manuals by hand.

5.1 Instruction Set Overview

Omissions

The charts in Chapter 5, "Instruction Set Overview," do not include the following operation names. However, these operations are implemented and they are documented in the dictionary.

CM:extract-multi-coordinate
CM:field-vp-set
CM:move-decoded-constant
CM:{s,u,f}-rank-2L

Inaccuracies

The charts in Chapter 5, "Instruction Set Overview," include the following operation names. However, these operations are neither included in the dictionary nor are they implemented.

CM:invert-bit
CM:s-round
CM:deposit-multi-coordinate

5.1.1 Dictionary: General Problems

This section describes general problems with the Paris reference documentation. These are errors that occur in many instruction definitions.

C/Paris Types

Chapter 6, "The C/Paris Interface," is quite vague about the types of various Paris operands. In previous releases, the header files **cmtypes.h** and **paris.h** were not entirely accurate either. In the future, the C/Paris type information will be more explicitly described in the *Paris Reference Manual*. Meanwhile, the **cmtypes.h** and **paris.h** header files have been corrected for the release of Version 5.1. While we apologize for the inconvenience, C/Paris users are encouraged to use these header files as their definitive source of information about C/Paris operand and return value types.

Field ID Type

The dictionary section of the *Paris Reference Manual*, Version 5.0, defines a field-id as an unsigned integer. Although field-id's are currently implemented as unsigned integers, this may not be true in future Connection Machine System Software versions.

This error occurs throughout Version 5.0 of the *Paris Reference Manual*. For instance, definitions for all the field allocation instructions should define the return values as field-id's rather than as the field-id's of unsigned integer fields. Similarly the *dest* and *send–address* arguments to instructions such as **CM:deposit-news-coordinate** should be defined simply as field-id's—not necessarily as field-id's of unsigned integer fields.

User code should not depend on the type of a field-id. C/Paris and Fortran/Paris code should conform to the language-specific field-id types given in "The C/Paris Interface," Chapter 6, and "The Fortran/Paris Interface," Chapter 7. Lisp/Paris code may rely on automatic coercion.

Zero Length Operands

In Version 5.0 of the *Paris Reference Manual*, 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.

Integer Immediate Operands

For all Paris instructions that take signed and unsigned integer immediate operands, which become constant operands once broadcast to the CM processors, the constant *must* be representable in the number of bits specified by the *len* argument.

The statement "The constant need not be representable in the number of bits specified by *len*" is, in the current implementation, false. This discrepancy between the documentation and the implementation applies to all binary arithmetic and integer constant operations such as, for example,

```
CM:{s,u}-add
CM:{s,u}-max
CM:{s,u}-min
CM:{s,u}-mod
CM:{s,u}-multiply
CM:{s,u}-subtract
cm:{s u}-{lt, le, eq, ne, ge, gt}-constant-1L
```

Integer Division

Division on signed or unsigned integers is accomplished with the truncation operations, **CM:s-truncate**, **CM:s-f-truncate**, **CM:u-truncate**, and **CM:u-f-truncate**. Chapter 5, "Instruction Set Overview," does not make this clear.

CM Floating Point

The CM System Software currently does not fully support the IEEE standard for floating point operations. For every Paris floating-point instruction, the flags section of the dictionary entry should read:

Flags	*test-flag* is set if division by zero occurs; otherwise it is unaffected. *overflow-flag* is set if floating-point overflow (including division by zero) occurs; otherwise it is unaffected. Underflow sets the result field to all zeros.

5.2 Corrected Documentation Errors

All the documentation errors reported in the *Paris Release Notes*, Version 5.1, are corrected by the change pages distributed with these release notes for Version 5.2. The nature of the corrections are described here, listed alphabetically by instruction name.

Unless otherwise noted, these errors appear in the dictionary portion of the *Paris Reference Manual*, Version 5.0—as already updated by change pages issued with Version 5.1.

CM:f-abs

If the source operand is a **NaN**, then the output is also a **NaN**. The dictionary entry erroneously claims that a **NaN** source is copied unchanged.

A new, corrected dictionary entry is included in the 5.2 packet of change pages.

CM:aref32-shared-2L and CM:aset32-shared-2L

The dictionary descriptions of these instructions are incomplete. They do not describe the data layout for shared arrays, nor do they mention the restriction that the *array* field operand must be contiguous in CM memory.

New, corrected dictionary entries are included in the 5.2 packet of change pages.

CM:allocate-stack-field-vp-set and CM:allocate-heap-field-vp-set

The order in which operands to **CM:allocate-stack-field-vp-set** and **CM:allocate-heap-field-vp-set** are to be specified is *len, vp-set-id*. The dictionary entry lists these arguments in the opposite order.

New, corrected Paris Dictionary pages for these instructions are included in the 5.2 packet of change pages.

CM:change-field-alias and CM:make-field-alias

An offset into a field may not be used as the value of the *field-id* operand to either of these instructions. This restriction was not included in the Version 5.1 Supplement documentation for these instructions.

The last paragraph of the dictionary entry for **CM:make-field-alias** uses the term "physical length" where it should use the term "field length." Notice that this appears in the *Paris Reference Manual Supplement*, Version 5.1, June 1989.

The third paragraph of the description of **CM:change-field-alias** makes reference to *vp-set* as though that were an operand; this reference should be to *field-id*.

New, corrected Supplement pages are included in the 5.2 packet of change pages.

CM:create-detailed-geometry

The most recent documentation for **CM:create-detailed-geometry** (sent as change pages with Paris Version 5.1) included, on page 117b, a warning that read:

> **Be Careful:** All grid-oriented operations may be used only on axes with :news-order ordering. This includes scans, spreads, reductions, and the **get-from-news** and **send-to-news** instructions.

This warning was partially incorrect: scans, spreads, and reductions *do* work on send-ordered axes (and always have). The warning is now completely unnecessary.

Prior to the release of Version 5.2, the instructions **CM:get-from-news{-always}-1L**, **CM:get-from-power-two{-always}-1L**, and **CM:send-to-news{-always}-1L** did not work on send-ordered axes. With Version 5.2, this restriction is lifted.

A new, corrected dictionary entry is included in the 5.2 packet of change pages.

CM:deposit-news-coordinate-1L

The *coordinate* operand definition is misleading. It has been changed to emphasize that this is a field.

A new, corrected dictionary entry is included in the 5.2 packet of change pages.

CM:extract-news-coordinate and CM:extract-multi-coordinate

The *send-address* operand definition is wrong for both operations. It has been changed to emphasize that this is a field.

A new, corrected dictionary entry is included in the 5.2 packet of change pages.

CM:get-1L and **CM:get-aref32-2L**

The initial descriptions use the phrase "from the same address," which should read "from the same memory address." Also the *send-address* operand definition is wrong for both operations.

A new, corrected dictionary entry is included in the 5.2 packet of change pages.

CM:global-u-max-1L and **CM:global-u-min-1L**

The dictionary entry for **CM:global-u-max1L** says that $2^{len}-1$ is returned when no processors are selected; it should say that 0 is returned. Conversely, the entry for **CM:global-u-min-1L** says that 0 is returned when no processors are active; it should say that $2^{len}-1$ is returned.

New, corrected dictionary entries are included in the 5.2 packet of change pages.

CM:global-f-min-1L, **CM:global-u-min-1L**, and **CM:global-u-min-1L**

The Version 5.0 descriptions for each of these instructions incorrectly state that they return the *largest* instead of the *smallest* of the source field arguments.

New, corrected dictionary entries are included in the 5.2 packet of change pages.

CM:load-*flag*

CM:load-overflow-always and **CM:-load-test-always** are implemented. They are the unconditional versions of **CM:load-overflow** and **CM:load-test** and should be among the **CM:load-*flag*** instructions listed in the dictionary.

A new, corrected dictionary entry is included in the 5.2 packet of change pages.

CM:multispread

The definition formula for most of the **CM:multispread** operation dictionary entries contains the following errors. The statement "let r = rank( )" should read "let r = rank(g)." The statement "where *scan-subclass* is as defined on page 36" should read "where *hyperplane* is as defined on page 36."

New, corrected dictionary entries are included in the 5.2 packet of change pages.

CM:my-send-address

The definition of the *dest* operand fails to mention that the lower bound on this value is the value returned by **CM:geometry-send-address-length.**

A new, corrected dictionary entry is included in the 5.2 packet of change pages.

CM:s-s-power

CM:s-s-power-constant-3-2L is implemented. It should be among the **CM:s-s-power** instructions listed in the dictionary.

A new, corrected dictionary entry is included in the 5.2 packet of change pages.

CM:rank

For all the **CM:rank** instructions, the *dlen* operand definition fails to mention that the upper bound on this value is the value returned by **CM:geometry-coordinate-length.**

A new, corrected dictionary entry is included in the 5.2 packet of change pages.

CM:send-to-news

The context description for this instruction erroneously refers to the *context-flag* of the destination rather than to that of the source.

A new, corrected dictionary entry is included in the 5.2 packet of change pages.

CM:store-*flag*

CM:store-overflow-always and **CM:store-test-always** are implemented. They are the unconditional versions of **CM:store-overflow** and **CM:]store-test** instructions and should be among the **CM:store-*flag*** instructions listed in the dictionary.

A new, corrected dictionary entry is included in the 5.2 packet of change pages.

6 Performance Notes

This section discusses general router communication, broadly explaining the factors that determine execution speed for this class of instructions. A table with test timings for most Paris Version 5.2 instructions is also included.

6.1 General Router Communication

The Paris **send** and **get** instructions are among the most powerful operations available on the CM. The **send** and **get** instructions without **-news** in their names are collectively referred to as *general router communications* (or sometimes simply *sends*). They allow any processor to send or receive data from any other processor.

The general router communication instructions currently supported are:

CM:get-1L
CM:get-aref32-2L
CM:send-1L
CM:send-aset32-(u-add, logior, overwrite}-2L
CM:send-with-{f, s, u}-add-1L
CM:send-with-logand-1L
CM:send-with-logior-1L
CM:send-with-logxor-1L
CM:send-with-{f,s,u}-{max, min}-1L
CM:send-with-overwrite-1L

While powerful, general router communications are among the longest-running Paris operations provided. Furthermore, in some circumstances, **get** instructions use a significant amount of temporary CM memory. CM programmers are therefore encouraged to use general router communications judiciously. If possible, NEWS communication (accomplished with instructions whose names include the term **-news**) should be used.

The time required to execute a general router communication instruction depends primarily on the degree of router "traffic congestion" induced by a particular instruction invocation. Router congestion is caused by complex communication patterns and by high VP ratios.

Guidelines helpful in predicting the performance of general router communication instructions are provided below.

Send Speed

To a first approximation, the time required to do a Paris **send** operation is controlled by the following factors:

- Communication pattern complexity
- VP ratio
- Message length
- Specific instruction implementation

Communication pattern complexity. The relative locations of the source and destination processors for all messages sent determine the degree of congestion. If many messages must travel over the same path, communication is slower than if message paths are evenly distributed across the machine.

The congestion induced within the CM by a particular communication pattern is quantified by the number of internal router cycles (termed *petit cycles*) required to complete a send. In general, most patterns are low congestion patterns and take some small number of petit cycles—less than a random pattern takes.

An example of an extremely low congestion pattern is one that emulates NEWS, i.e., a pattern in which each virtual processor sends a message to one of its neighbors. Low congestion patterns involve many-to-many communication.

A high congestion pattern involves many–to–few communication. That is, a pattern in which all or many processors send messages to different virtual processors on the same physical chip takes many petit cycles to complete. For instance, matrix transposition at a VP ratio greater than one tends to create high congestion. If the matrix rows are stored across a set of physical processors and the columns are stored in virtual processor banks within these processors, then sending a whole row to a single column is many-to-few communication.

It is interesting to note that while most regular patterns are low congestion patterns, most high congestion patterns are regular patterns.

VP ratio. The higher the VP ratio, the more messages are likely to be sent across the same router paths. The number of petit cycles required to perform any given send instruction increases in roughly linear proportion with the VP ratio.

Message length. The duration of each petit cycle is a fixed overhead plus a certain amount per bit of data sent, so doubling a message length less than doubles the router time required to send the message. The minimum message length handled is approximately 25 bits; fewer bits may be sent, but this is no faster than sending 25. Messages over approximately 128 bits long are transferred as multiple messages, which can substantially slow and complicate a send operation.

Generally, for message lengths within the range of approximately 25 to 128 bits, it takes less time to send one long message than to send several short ones.

Specific instruction implementation. The exact operations performed by a specific send instruction affect execution time.

For example, the instruction **CM:send-with-f-add-1L** take longer than its integer counterparts. Before the floating-point data is transmitted by the router, it is denormalized to a fixed-point format. This denormalization takes time and also increases the message length. (This is not the case for the **CM:send-with-f-{min,max}-1L** instructions, which are implemented in a manner that avoids denormalization.)

As another example, router time generally decreases with increased enroute combining. Instructions whose names contain the term **-with** perform combining. While executing a combining instruction, the router attempts to combine any two messages headed for the same destination processor. Enroute combining reduces congestion and speeds up router execution because, as messages are combined, their number is reduced.

An exceptional case is the instruction **CM:send-with-logxor**, for which there is no hardware support. In contrast to other combining instructions, enroute combining is not done for a **CM:send-with-logxor** operation. Therefore, the time required to accomplish a **CM:send-with-logxor** operation is bounded below by a constant times the maximum number of items sent to any one destination.

Get Speed

A Paris **get** operation is accomplished by a process known as *backwards routing*. First, a **send** is done by the processors that are requesting data, and state information is saved. Then, the **send** is reversed, using the saved state information. Although this second phase is slightly faster than the first, one may assume that, for any given communication pattern, a **get** takes twice as long as a **send**.

Depending on the degree of congestion induced by the communication pattern, a **get** operation may use a substantial amount of CM temporary memory. This is especially likely if many processors attempt to retrieve data from different virtual processors on the same chip. If router congestion becomes severe during a **get**, an alternative algorithm is automatically used to avoid terminating with an out-of-memory condition. While this type of **get** operation takes up to 8 times as long as a normal **get**, it supports many processors retrieving data from one or a few chips without overrunning temporary memory.

1.3 General Timing Information

The following pages contain timing tables for most Paris Version 5.2 instructions.

These timings were done running Connection Machine System Software, Version 5.2, from a Sun-4/280 front end connected to an 8K CM-2 with 64-bit floating-point accelerator chips and 256K bits of memory per processor. Timings taken using different hardware configurations will vary.

Each instruction was tested both at a VP ratio of 1 and at a VP ratio of 16, within a 1-dimensional geometry.The timings for all router communication instructions were taken using random message communication patterns. Although data lengths are given for all instructions, these do not apply to instructions that do not take arguments. Timings taken using different software configurations will vary.

The reported "real time" measures the total execution time, including both front-end and CM execution time. The reported "CM time" measures the amount of time the Connection Machine spent executing the instruction. Differences between these numbers indicate how much time the CM spent idle while the front end completed its execution.

The timing numbers reported here were empirically derived; they are reliable within a 10% margin of accuracy. Use these numbers to compare the relative performance of different Paris instructions.

Instruction	Vpr 1 real time	Vpr 1 CM time	Vpr16 real time	Vpr16 CM time	field width
CM_f_abs_1_1L	12	7	28	28	32
CM_f_abs_2_1L	41	35	419	419	32
CM_f_c_abs_2_1L	375	227	3000	2998	32
CM_s_abs_1_1L	89	80	774	750	32
CM_s_abs_2_1L	75	73	903	903	32
CM_s_abs_2_2L	104	63	903	903	32
CM_c_acos_1_1L	5375	2912	39799	37236	32
CM_c_acos_2_1L	5249	2963	38800	36605	32
CM_f_acos_1_1L	2163	1432	17983	16039	32
CM_f_acos_2_1L	2153	1393	17612	15769	32
CM_f_acosh_1_1L	1704	1213	12790	11853	32
CM_f_acosh_2_1L	1718	1164	12741	12257	32
CM_c_add_2_1L	87	87	1199	1199	32
CM_c_add_3_1L	99	99	1000	999	32
CM_f_add_2_1L	71	64	532	532	32
CM_f_add_3_1L	82	67	532	532	32
CM_s_add_2_1L	53	37	470	470	32
CM_s_add_3_1L	150	81	1225	1225	32
CM_s_add_3_3L	81	76	774	774	32
CM_u_add_2_1L	48	47	451	451	32
CM_u_add_3_1L	136	92	1209	1209	32
CM_u_add_3_3L	90	73	741	741	32
CM_c_add_always_2_1L	146	104	854	854	32
CM_c_add_always_3_1L	165	142	1048	1048	32
CM_f_add_always_2_1L	82	57	435	435	32
CM_f_add_always_3_1L	80	80	451	451	32
CM_s_add_carry_2_1L	33	28	483	483	32
CM_s_add_carry_3_1L	146	108	1338	1338	32
CM_s_add_carry_3_3L	188	106	1435	1435	32
CM_u_add_carry_2_1L	52	36	467	467	32
CM_f_add_const_always_2_1L	73	37	322	322	32
CM_f_add_const_always_3_1L	68	53	338	338	32
CM_f_add_const_mult_1L	84	62	725	651	32
CM_f_add_const_mult_const_1L	90	61	612	612	32
CM_f_add_constant_2_1L	78	57	677	444	32
CM_f_add_constant_3_1L	65	55	435	435	32
CM_s_add_constant_2_1L	66	50	564	564	32
CM_s_add_constant_3_1L	128	89	1016	1016	32
CM_u_add_constant_2_1L	66	46	548	548	32
CM_u_add_constant_3_1L	130	108	967	967	32
CM_s_add_flags_2_1L	38	28	467	467	32
CM_u_add_flags_2_1L	56	37	451	451	32
CM_f_add_mult_1L	76	65	661	661	32
CM_f_add_mult_const_1L	94	58	580	580	32
CM_f_asin_1_1L	1625	1189	13799	13798	32
CM_f_asin_2_1L	1900	1346	17000	13876	32
CM_f_asinh_1_1L	1562	1274	12000	10821	32
CM_f_asinh_2_1L	1612	1570	10599	10599	32
CM_f_atan_1_1L	1262	942	10999	10422	32
CM_f_atan_2_1L	1587	917	12199	10998	32
CM_f_atan2_3_1L	1537	1144	13399	10683	32
CM_f_atanh_1_1L	2237	1360	17799	13299	32
CM_f_atanh_2_1L	2050	1307	17599	13907	32
CM_f_f_ceiling_1_1L	1034	658	11467	8368	32
CM_f_f_ceiling_2_1L	1157	694	12097	9065	32

Instruction	Vpr 1 real time	Vpr 1 CM time	Vpr16 real time	Vpr16 CM time	field width
CM_clear_all_flags	28	10	387	387	32
CM_clear_all_flags_always	33	8	48	48	32
CM_clear_bit	18	6	48	48	32
CM_clear_bit_always	16	9	67	23	32
CM_clear_context	7	2	19	19	32
CM_clear_overflow	13	5	48	41	32
CM_clear_test	23	9	51	39	32
CM_c_conjugate_1_1L	12	8	209	209	32
CM_c_conjugate_2_1L	89	68	870	870	32
CM_c_cos_1_1L	5237	3577	47200	43005	32
CM_c_cos_2_1L	5774	3506	41999	37983	32
CM_f_cos_1_1L	399	399	5200	5199	32
CM_f_cos_2_1L	425	424	5200	5199	32
CM_f_cosh_1_1L	1850	1300	12600	12598	32
CM_f_cosh_2_1L	1837	1385	14600	14361	32
CM_create_detailed_geometry	629	0	629	0	32
CM_create_geometry	822	0	822	0	32
CM_deposit_news_coordinate_1L	95	73	766	747	32
CM_c_divide_2_1L	1012	759	6800	6798	32
CM_c_divide_3_1L	1075	917	6000	5999	32
CM_f_divide_2_1L	300	293	2000	1268	32
CM_f_divide_3_1L	87	87	1399	1398	32
CM_c_divide_always_2_1L	962	725	6800	6798	32
CM_c_divide_always_3_1L	1249	1098	6000	5999	32
CM_f_divide_always_2_1L	87	87	1200	1199	32
CM_f_divide_always_3_1L	199	199	1200	1198	32
CM_f_divide_const_always_3_1L	75	74	1000	998	32
CM_f_divide_constant_2_1L	87	87	1200	1198	32
CM_f_divide_constant_3_1L	87	87	1200	1199	32
CM_c_divinto_2_1L	637	509	6599	6598	32
CM_f_divinto_2_1L	87	87	1200	1199	32
CM_c_divinto_always_2_1L	950	806	6599	6598	32
CM_f_divinto_always_2_1L	199	199	1200	1198	32
CM_f_divinto_const_always_2_1L	199	199	1000	999	32
CM_f_divinto_const_always_3_1L	87	84	1000	998	32
CM_f_divinto_constant_2_1L	75	74	1200	1199	32
CM_f_divinto_constant_3_1L	199	199	1200	1199	32
CM_enumerate_1L	1122	978	2768	2604	32
CM_f_eq_1L	58	38	322	322	32
CM_s_eq_1L	51	48	435	435	32
CM_s_eq_2L	52	51	435	435	32
CM_u_eq_1L	34	25	435	435	32
CM_u_eq_2L	61	34	435	435	32
CM_f_eq_constant_1L	43	27	209	209	32
CM_s_eq_constant_1L	62	61	548	548	32
CM_u_eq_constant_1L	69	64	548	548	32
CM_f_eq_zero_1L	46	24	225	225	32
CM_s_eq_zero_1L	79	67	548	548	32
CM_u_eq_zero_1L	59	55	532	532	32
CM_f_exp_1_1L	589	575	5403	5403	32
CM_f_exp_2_1L	602	602	5403	5403	32
CM_extract_news_coordinate_1L	113	90	884	878	32
CM_fe_deposit_news_coordinate	63	0	63	0	32
CM_fe_extract_news_coordinate	58	0	58	0	32
CM_fe_from_gray_code	28	0	28	0	32

Instruction	Vpr 1 real time	Vpr 1 CM time	Vpr16 real time	Vpr16 CM time	field width
CM_fe_make_news_coordinate	65	0	65	0	32
CM_fe_make_news_mask	246	0	246	0	32
CM_field_vp_set	18	0	18	0	32
CM_f_s_float_2_2L	404	314	6612	5122	32
CM_f_u_float_2_2L	383	272	6000	4366	32
CM_f_f_floor_1_1L	941	554	11645	9151	32
CM_f_f_floor_2_1L	1017	586	12564	9489	32
CM_f_ge_1L	56	39	322	322	32
CM_s_ge_1L	46	25	435	435	32
CM_s_ge_2L	56	46	451	451	32
CM_f_ge_constant_1L	51	31	225	222	32
CM_s_ge_constant_1L	80	64	612	602	32
CM_s_ge_zero_1L	43	42	596	596	32
CM_geometry_axis_length	6	0	6	0	32
CM_geometry_axis_off_chip_bits	6	0	6	0	32
CM_geometry_axis_on_chip_bits	6	0	6	0	32
CM_geometry_axis_ordering	6	0	6	0	32
CM_geometry_axis_vp_ratio	6	0	6	0	32
CM_geometry_coordinate_length	6	0	6	0	32
CM_geometry_rank	6	0	6	0	32
CM_geometry_send_address_length	6	0	6	0	32
CM_geometry_total_processors	10	0	10	0	32
CM_geometry_total_vp_ratio	6	0	6	0	32
CM_get_1L	4310	3589	27256	26234	32
CM_get_aref32_2L	7917	3650	82608	26994	32
CM_get_from_news_1L	173	146	2086	1452	32
CM_get_from_news_always_1L	162	144	1155	1064	32
CM_global_c_add_1L	380	279	800	485	32
CM_global_count_bit	199	186	400	332	32
CM_global_count_bit_always	232	198	400	333	32
CM_global_count_context	230	199	350	332	32
CM_global_count_overflow	232	201	350	335	32
CM_global_count_test	230	199	358	339	32
CM_global_f_add_1L	172	139	352	235	32
CM_global_f_max_1L	339	318	2177	2124	32
CM_global_f_min_1L	339	318	3482	3191	32
CM_global_logand_1L	167	126	1848	1713	32
CM_global_logand_bit	49	18	112	92	32
CM_global_logand_bit_always	46	15	96	91	32
CM_global_logand_context	49	12	112	93	32
CM_global_logand_overflow	500	17	135	131	32
CM_global_logand_test	500	18	129	115	32
CM_global_logior_1L	109	77	1135	1004	32
CM_global_logior_bit	19	2	40	30	32
CM_global_logior_bit_always	19	2	23	19	32
CM_global_logior_context	19	2	41	23	32
CM_global_logior_overflow	19	2	41	24	32
CM_global_logior_test	22	5	40	22	32
CM_global_logxor_1L	7420	6549	11904	10719	32
CM_global_s_add_1L	1216	1131	4625	4441	32
CM_global_s_max_1L	372	230	2029	1664	32
CM_global_s_min_1L	331	274	2112	1695	32
CM_global_u_add_1L	610	549	1118	1022	32
CM_global_u_max_1L	291	227	2759	2548	32
CM_global_u_max_s_intlen_1L	311	230	3278	3108	32

Instruction	Vpr 1 real time	Vpr 1 CM time	Vpr16 real time	Vpr16 CM time	field width
CM_global_u_max_u_intlen_1L	300	240	3395	3224	32
CM_global_u_min_1L	372	328	2883	2589	32
CM_f_gt_1L	48	39	322	322	32
CM_s_gt_1L	43	27	451	451	32
CM_s_gt_2L	46	24	435	435	32
CM_f_gt_constant_1L	39	25	532	445	32
CM_s_gt_constant_1L	86	83	596	596	32
CM_f_gt_zero_1L	36	34	209	209	32
CM_s_integer_length_2_2L	546	546	8709	8709	32
CM_invert_context	3	2	16	16	32
CM_invert_overflow	3	2	16	16	32
CM_invert_test_always	3	2	19	19	32
CM_is_field_in_heap	7	0	7	0	32
CM_is_field_in_stack	7	0	7	0	32
CM_s_isqrt_1_1L	693	693	11048	11048	32
CM_s_isqrt_2_1L	693	693	11048	11048	32
CM_s_isqrt_2_2L	791	736	11080	11080	32
CM_u_isqrt_2_1L	711	711	11339	11339	32
CM_u_isqrt_2_2L	811	766	11386	11386	32
CM_f_le_1L	44	31	322	322	32
CM_s_le_1L	40	25	451	451	32
CM_s_le_2L	60	27	435	435	32
CM_u_le_1L	39	38	435	435	32
CM_u_le_2L	38	25	435	435	32
CM_f_le_constant_1L	39	28	225	225	32
CM_s_le_constant_1L	79	66	612	612	32
CM_u_le_constant_1L	72	53	548	548	32
CM_f_le_zero_1L	58	41	225	225	32
CM_s_le_zero_1L	79	53	596	596	32
CM_u_le_zero_1L	77	74	548	548	32
CM_f_ln_1_1L	522	513	4645	4645	32
CM_f_ln_2_1L	508	500	4645	4645	32
CM_load_context	7	2	32	32	32
CM_load_overflow	4	4	48	48	32
CM_load_overflow_always	4	2	16	16	32
CM_load_test	4	4	48	48	32
CM_load_test_always	4	2	32	32	32
CM_f_log10_1_1L	517	517	4645	4645	32
CM_f_log10_2_1L	524	511	4645	4645	32
CM_logand_2_1L	47	33	419	419	32
CM_logand_3_1L	147	97	1282	1281	32
CM_logand_constant_2_1L	92	47	692	692	32
CM_logand_constant_3_1L	142	97	1157	1150	32
CM_logand_context	8	3	25	25	32
CM_logand_context_with_test	4	2	22	22	32
CM_logand_overflow	7	4	67	38	32
CM_logand_overflow_always	9	3	48	25	32
CM_logand_test	7	4	76	40	32
CM_logand_test_always	9	3	25	25	32
CM_logandc1_2_1L	45	41	419	419	32
CM_logandc1_3_1L	147	89	1317	1317	32
CM_logandc1_constant_2_1L	96	51	717	690	32
CM_logandc1_constant_3_1L	140	78	1209	1124	32
CM_logandc2_2_1L	38	25	419	419	32
CM_logandc2_3_1L	155	99	1290	1290	32

	Vpr 1		Vpr16		field
Instruction	real time	CM time	real time	CM time	width
CM_logandc2_constant_3_1L	71	68	1128	1128	32
CM_s_logcount_2_2L	163	161	2419	2419	32
CM_u_logcount_2_2L	163	160	2419	2419	32
CM_logeqv_2_1L	24	23	419	419	32
CM_logeqv_3_1L	117	88	1290	1290	32
CM_logeqv_constant_2_1L	86	50	693	693	32
CM_logeqv_constant_3_1L	140	76	1112	1112	32
CM_logior_2_1L	48	24	419	419	32
CM_logior_3_1L	142	89	1274	1274	32
CM_logior_constant_2_1L	96	52	693	693	32
CM_logior_constant_3_1L	144	91	1274	1123	32
CM_logior_context	9	3	25	25	32
CM_logior_overflow	4	4	48	48	32
CM_logior_overflow_always	14	3	16	16	32
CM_logior_test	4	3	48	48	32
CM_logior_test_always	6	3	32	32	32
CM_lognand_2_1L	37	24	419	419	32
CM_lognand_3_1L	167	84	1290	1290	32
CM_lognand_constant_2_1L	96	45	741	691	32
CM_lognand_constant_3_1L	132	87	1193	1193	32
CM_lognor_2_1L	51	24	419	419	32
CM_lognor_3_1L	146	76	1274	1274	32
CM_lognor_constant_2_1L	86	53	693	693	32
CM_lognor_constant_3_1L	128	94	1112	1112	32
CM_lognot_1_1L	62	52	354	354	32
CM_lognot_2_1L	33	24	419	419	32
CM_logorc1_2_1L	63	33	419	419	32
CM_logorc1_3_1L	127	91	1274	1274	32
CM_logorc1_constant_2_1L	95	54	693	693	32
CM_logorc1_constant_3_1L	132	102	1112	1112	32
CM_logorc2_2_1L	54	52	419	419	32
CM_logorc2_3_1L	148	84	1274	1274	32
CM_logorc2_constant_2_1L	79	51	693	693	32
CM_logorc2_constant_3_1L	117	71	1177	1177	32
CM_logxor_2_1L	44	26	419	419	32
CM_logxor_3_1L	142	80	1290	1290	32
CM_logxor_constant_2_1L	93	55	693	693	32
CM_logxor_constant_3_1L	132	90	1161	1123	32
CM_f_lt_1L	67	39	322	322	32
CM_s_lt_1L	44	42	451	451	32
CM_s_lt_2L	36	34	435	435	32
CM_u_lt_1L	44	34	435	435	32
CM_u_lt_2L	45	44	435	435	32
CM_f_lt_constant_1L	52	35	225	225	32
CM_s_lt_constant_1L	80	70	596	596	32
CM_u_lt_constant_1L	70	58	548	548	32
CM_f_lt_zero_1L	37	35	225	225	32
CM_s_lt_zero_1L	76	57	596	596	32
CM_u_lt_zero_1L	68	51	548	548	32
CM_make_news_coordinate_1L	115	86	946	914	32
CM_f_max_2_1L	89	70	677	677	32
CM_f_max_3_1L	83	74	677	677	32
CM_s_max_2_1L	75	45	838	838	32
CM_s_max_3_1L	188	140	1693	1693	32
CM_s_max_3_3L	203	143	1774	1774	32

Instruction	Vpr 1 real time	Vpr 1 CM time	Vpr16 real time	Vpr16 CM time	field width
CM_u_max_2_1L	76	45	822	822	32
CM_u_max_3_1L	180	134	1709	1692	32
CM_u_max_3_3L	188	146	1822	1822	32
CM_f_max_constant_3_1L	47	46	580	580	32
CM_s_max_constant_2_1L	77	77	1096	1096	32
CM_s_max_constant_3_1L	167	113	1532	1532	32
CM_u_max_constant_2_1L	186	102	919	919	32
CM_u_max_constant_3_1L	213	104	1612	1501	32
CM_f_min_2_1L	105	75	677	677	32
CM_f_min_3_1L	100	74	677	677	32
CM_s_min_2_1L	90	58	838	838	32
CM_s_min_3_1L	172	134	1693	1693	32
CM_s_min_3_3L	195	134	1774	1774	32
CM_u_min_2_1L	86	62	822	822	32
CM_u_min_3_1L	192	136	1677	1677	32
CM_u_min_3_3L	174	123	1806	1806	32
CM_f_min_constant_2_1L	93	69	596	596	32
CM_f_min_constant_3_1L	94	91	596	596	32
CM_s_min_constant_2_1L	121	76	1096	1096	32
CM_s_min_constant_3_1L	176	156	1516	1516	32
CM_u_min_constant_2_1L	167	81	1032	1032	32
CM_u_min_constant_3_1L	213	107	2032	1926	32
CM_f_mod_2_1L	3432	2166	31870	29109	32
CM_f_mod_3_1L	3419	2438	32338	29441	32
CM_s_mod_2_1L	1704	1415	27000	22253	32
CM_s_mod_3_1L	1810	1536	28467	22855	32
CM_u_mod_2_1L	1564	1279	23452	20624	32
CM_u_mod_3_1L	1619	1317	24274	21198	32
CM_f_mod_constant_2_1L	3521	2356	33241	30215	32
CM_f_mod_constant_3_1L	3549	2525	33339	30265	32
CM_s_mod_constant_2_1L	1803	1462	26806	23167	32
CM_s_mod_constant_3_1L	1857	1506	28838	23648	32
CM_u_mod_constant_2_1L	1617	1351	24886	20199	32
CM_u_mod_constant_3_1L	1669	1440	24160	20446	32
CM_f_move_1L	40	27	435	435	32
CM_f_move_2L	48	39	419	419	32
CM_s_move_1L	41	27	483	420	32
CM_s_move_2L	56	50	467	467	32
CM_u_move_1L	47	27	435	435	32
CM_u_move_2L	63	53	500	499	32
CM_f_move_always_1L	29	19	403	403	32
CM_s_move_always_1L	36	24	322	322	32
CM_u_move_always_1L	40	39	403	403	32
CM_f_move_const_always_1L	27	17	161	161	32
CM_f_move_constant_1L	45	26	403	372	32
CM_s_move_constant_1L	38	38	451	451	32
CM_u_move_constant_1L	33	19	274	274	32
CM_move_reversed_1L	65	42	516	516	32
CM_f_move_zero_1L	35	31	274	274	32
CM_s_move_zero_1L	39	28	258	258	32
CM_u_move_zero_1L	29	20	258	258	32
CM_f_move_zero_always_1L	22	12	177	177	32
CM_s_move_zero_always_1L	22	12	354	234	32
CM_u_move_zero_always_1L	22	12	177	177	32
CM_f_mult_add_1L	103	68	741	741	32

	Vpr 1		Vpr16		field
Instruction	real time	CM time	real time	CM time	width
CM_f_mult_add_const_1L	86	56	693	689	32
CM_f_mult_const_add_1L	94	67	661	573	32
CM_f_mult_const_add_const_1L	73	63	516	516	32
CM_f_mult_const_sub_1L	106	50	661	577	32
CM_f_mult_const_sub_const_1L	81	62	612	516	32
CM_f_mult_const_subf_const_1L	43	42	516	516	32
CM_f_mult_sub_1L	70	61	677	677	32
CM_f_mult_sub_const_1L	95	79	564	564	32
CM_f_mult_subf_1L	100	72	677	677	32
CM_f_mult_subf_const_1L	90	64	580	580	32
CM_f_multiply_2_1L	83	59	528	527	32
CM_f_multiply_3_1L	67	65	532	532	32
CM_s_multiply_2_1L	109	76	790	790	32
CM_s_multiply_3_1L	97	75	790	790	32
CM_s_multiply_3_3L	1113	1113	17758	17758	32
CM_u_multiply_2_1L	956	956	15225	15225	32
CM_u_multiply_3_1L	959	956	15242	15242	32
CM_u_multiply_3_3L	956	956	15242	15242	32
CM_f_multiply_always_2_1L	41	41	435	435	32
CM_f_multiply_always_3_1L	41	41	435	435	32
CM_f_multiply_const_always_2_1L	68	52	338	338	32
CM_f_multiply_const_always_3_1L	76	60	338	338	32
CM_f_multiply_constant_2_1L	70	48	451	451	32
CM_f_multiply_constant_3_1L	79	47	435	435	32
CM_s_multiply_constant_2_1L	102	90	709	709	32
CM_s_multiply_constant_3_1L	87	83	693	693	32
CM_u_multiply_constant_2_1L	1040	1021	15515	15515	32
CM_u_multiply_constant_3_1L	1197	1101	15935	15935	32
CM_multispread_f_add_1L	690	216	2608	1756	32
CM_multispread_f_max_1L	720	228	2773	2231	32
CM_multispread_f_min_1L	717	241	3413	2398	32
CM_multispread_logand_1L	650	200	2733	1839	32
CM_multispread_logior_1L	608	147	2799	1934	32
CM_multispread_logxor_1L	605	167	2746	2035	32
CM_multispread_s_add_1L	604	139	2480	1733	32
CM_multispread_s_max_1L	631	178	3613	2362	32
CM_multispread_s_min_1L	672	228	3520	2727	32
CM_multispread_u_add_1L	597	138	2786	1897	32
CM_multispread_u_max_1L	601	169	3319	2564	32
CM_multispread_u_min_1L	650	197	3333	2119	32
CM_my_news_coordinate_1L	169	143	685	685	32
CM_my_send_address_1L	121	45	843	701	32
CM_f_ne_1L	45	41	435	435	32
CM_s_ne_1L	48	26	435	435	32
CM_s_ne_2L	36	25	435	435	32
CM_u_ne_1L	48	39	435	435	32
CM_u_ne_2L	36	30	435	435	32
CM_f_ne_constant_1L	48	35	403	403	32
CM_s_ne_constant_1L	55	48	548	548	32
CM_u_ne_constant_1L	65	56	548	548	32
CM_f_ne_zero_1L	60	32	225	225	32
CM_s_ne_zero_1L	77	67	548	548	32
CM_u_ne_zero_1L	70	55	532	532	32
CM_f_negate_1_1L	20	7	16	16	32
CM_f_negate_2_1L	30	29	419	419	32

	Vpr 1		Vpr16		field
Instruction	real time	CM time	real time	CM time	width
CM_s_negate_1_1L	48	38	370	370	32
CM_s_negate_2_1L	49	26	435	435	32
CM_s_negate_2_2L	96	63	919	919	32
CM_u_negate_1_1L	48	35	370	370	32
CM_u_negate_2_1L	60	32	435	435	32
CM_u_negate_2_2L	90	54	935	935	32
CM_f_news_add_2_1L	249	148	1007	601	32
CM_f_news_add_always_2_1L	172	126	704	514	32
CM_f_news_add_always_3_1L	249	138	736	509	32
CM_f_news_add_const_3_1L	250	122	655	531	32
CM_f_news_add_const_a_3_1L	237	164	944	602	32
CM_f_news_add_mult_4_1L	283	188	1679	1326	32
CM_f_news_mult_2_1L	245	153	1376	945	32
CM_f_news_mult_3_1L	257	167	1168	639	32
CM_f_news_mult_add_4_1L	247	177	1327	1092	32
CM_f_news_mult_always_2_1L	234	145	1136	653	32
CM_f_news_mult_always_3_1L	280	181	1104	708	32
CM_f_news_mult_const_3_1L	241	138	1040	637	32
CM_f_news_mult_const_a_3_1L	246	126	1184	870	32
CM_f_news_mult_sub_4_1L	287	148	1536	1047	32
CM_f_news_sub_2_1L	247	125	1327	909	32
CM_f_news_sub_3_1L	260	142	1263	789	32
CM_f_news_sub_always_2_1L	249	143	1088	673	32
CM_f_news_sub_always_3_1L	254	153	991	699	32
CM_f_news_sub_const_3_1L	234	146	1152	566	32
CM_f_news_sub_const_a_3_1L	243	133	1168	856	32
CM_f_c_phase_2_1L	2037	1309	16799	14635	32
CM_physical_vp_set	3	0	3	0	32
CM_c_c_power_2_1L	5975	3605	39999	39994	32
CM_c_c_power_3_1L	5750	4501	41999	41999	32
CM_f_f_power_2_1L	1625	1010	12000	11998	32
CM_f_f_power_3_1L	1799	996	12000	11997	32
CM_f_s_power_2_2L	7650	5077	57999	49401	32
CM_f_s_power_3_2L	7699	5360	59000	47258	32
CM_f_u_power_2_2L	7174	4348	51500	48806	32
CM_f_u_power_3_2L	7100	4462	57500	48911	32
CM_s_s_power_2_1L	101980	83856	1091500	1091500	32
CM_s_s_power_3_1L	100589	81964	1092000	1092000	32
CM_s_s_power_3_3L	98624	83347	1092499	1092499	32
CM_s_u_power_3_3L	95299	82147	1070500	1070500	32
CM_u_s_power_3_3L	94025	73638	976750	976750	32
CM_u_u_power_2_1L	92249	78870	957000	957000	32
CM_u_u_power_3_1L	92799	74577	957499	957499	32
CM_u_u_power_3_3L	92937	72356	957499	957499	32
CM_f_f_power_constant_2_1L	1724	1169	11500	11494	32
CM_f_f_power_constant_3_1L	1200	826	11500	11497	32
CM_f_s_power_constant_2_1L	125	101	1000	997	32
CM_f_s_power_constant_3_1L	99	86	1499	1497	32
CM_f_u_power_constant_2_1L	49	33	399	399	32
CM_f_u_power_constant_3_1L	125	124	600	599	32
CM_s_s_power_constant_2_1L	2450	2449	35000	34997	32
CM_s_s_power_constant_3_1L	2474	2333	35500	35202	32
CM_s_s_power_constant_3_2L	2700	2569	35000	34997	32
CM_s_u_power_constant_2_1L	2224	2171	34499	34495	32
CM_s_u_power_constant_3_1L	2524	2412	35000	34997	32

Instruction	Vpr 1 real time	Vpr 1 CM time	Vpr16 real time	Vpr16 CM time	field width
CM_s_u_power_constant_3_2L	2374	2374	35000	34997	32
CM_u_s_power_constant_2_1L	2600	2280	30500	30495	32
CM_u_s_power_constant_3_1L	2550	2192	30500	30497	32
CM_u_s_power_constant_3_2L	2224	2031	30500	30497	32
CM_u_u_power_constant_2_1L	2300	2095	30500	30496	32
CM_u_u_power_constant_3_1L	2099	1937	30500	30497	32
CM_f_random_1L	1338	1146	15355	15355	32
CM_u_random_1L	2916	2909	46371	46371	32
CM_f_read_from_news_array_1L	36899	34976	485000	483300	32
CM_s_read_from_news_array_1L	45899	41749	709999	704280	32
CM_u_read_from_news_array_1L	42199	40189	670000	668479	32
CM_f_read_from_processor_1L	247	60	112	61	32
CM_s_read_from_processor_1L	190	74	177	61	32
CM_u_read_from_processor_1L	192	64	129	64	32
CM_reduce_with_f_add_1L	3500	2700	5319	4648	32
CM_reduce_with_f_max_1L	3301	2632	5493	4350	32
CM_reduce_with_f_min_1L	3263	2448	5306	4106	32
CM_reduce_with_logand_1L	2629	2085	4347	3641	32
CM_reduce_with_logior_1L	2529	1805	3752	3207	32
CM_reduce_with_logxor_1L	2565	1935	4004	2660	32
CM_reduce_with_s_add_1L	2548	1868	3409	3010	32
CM_reduce_with_s_max_1L	3100	2547	5373	4554	32
CM_reduce_with_s_min_1L	3197	2480	5373	4279	32
CM_reduce_with_u_add_1L	2581	2027	3746	3024	32
CM_reduce_with_u_max_1L	3052	2350	5186	4494	32
CM_reduce_with_u_min_1L	3216	2434	5493	4409	32
CM_f_rem_2_1L	3022	2158	25032	25032	32
CM_f_rem_3_1L	2994	2086	25144	25033	32
CM_s_rem_2_1L	1438	1269	21516	19558	32
CM_s_rem_3_1L	1725	1416	20741	19889	32
CM_u_rem_2_1L	1493	1196	24645	20805	32
CM_u_rem_3_1L	1644	1399	24547	21029	32
CM_f_rem_constant_2_1L	3110	2172	25210	25210	32
CM_f_rem_constant_3_1L	3153	2208	26048	25981	32
CM_s_rem_constant_2_1L	1658	1317	23790	20968	32
CM_s_rem_constant_3_1L	1861	1509	22112	20682	32
CM_u_rem_constant_2_1L	1612	1378	24152	20503	32
CM_u_rem_constant_3_1L	1683	1426	23581	20710	32
CM_f_f_round_1_1L	1091	682	15242	12021	32
CM_f_f_round_2_1L	1178	795	16612	12550	32
CM_scan_with_copy_1L	3314	2661	7757	7693	32
CM_scan_with_f_add_1L	4869	3913	8924	8139	32
CM_scan_with_f_max_1L	2391	1714	8077	6560	32
CM_scan_with_f_min_1L	2390	1723	8237	6687	32
CM_scan_with_f_multiply_1L	4952	4110	9427	7701	32
CM_scan_with_logand_1L	2998	2477	5652	5041	32
CM_scan_with_logior_1L	911	849	2334	2332	32
CM_scan_with_logxor_1L	2955	2407	4965	4370	32
CM_scan_with_s_add_1L	880	818	2334	2333	32
CM_scan_with_s_max_1L	1457	1298	3501	3500	32
CM_scan_with_s_min_1L	1454	1271	3638	3638	32
CM_scan_with_u_add_1L	894	845	2334	2333	32
CM_scan_with_u_max_1L	1069	1042	3341	3340	32
CM_send_1L	1111	1085	41464	41462	32
CM_send_aset32_logior_1L	6411	3091	44027	43444	32

Instruction	Vpr 1 real time	Vpr 1 CM time	Vpr16 real time	Vpr16 CM time	field width
CM_send_aset32_overwrite_1L	4771	2214	35744	29007	32
CM_send_aset32_u_add_1L	6774	3261	43958	43336	32
CM_send_to_news_1L	271	173	1043	817	32
CM_send_to_news_always_1L	165	140	588	588	32
CM_send_with_f_add_1L	6814	3554	137529	136600	32
CM_send_with_f_max_1L	4137	3389	90663	90597	32
CM_send_with_f_min_1L	3952	3039	91167	91110	32
CM_send_with_logand_1L	1382	1381	58924	58924	32
CM_send_with_logior_1L	1367	1342	57941	57939	32
CM_send_with_overwrite_1L	1214	1192	37230	37230	32
CM_send_with_s_add_1L	1355	1343	81418	81418	32
CM_send_with_s_max_1L	1781	1781	87573	87573	32
CM_send_with_s_min_1L	1900	1900	88740	88740	32
CM_send_with_u_add_1L	1342	1327	81395	81395	32
CM_send_with_u_max_1L	1781	1781	87071	87070	32
CM_send_with_u_min_1L	1892	1892	89930	89930	32
CM_set_bit	11	8	64	64	32
CM_set_bit_always	11	6	32	32	32
CM_set_context	3	1	16	16	32
CM_set_overflow	11	9	48	48	32
CM_set_test	9	8	48	48	32
CM_set_vp_set	3	0	3	0	32
CM_set_vp_set_geometry	3	0	3	0	32
CM_c_c_signum_1_1L	991	612	6967	6761	32
CM_c_c_signum_2_1L	1067	744	7758	7545	32
CM_f_f_signum_1_1L	47	44	354	354	32
CM_f_f_signum_2_1L	44	37	370	370	32
CM_s_s_signum_2_1L	69	47	596	596	32
CM_s_s_signum_2_2L	76	68	596	596	32
CM_f_sin_1_1L	425	424	4999	4998	32
CM_f_sin_2_1L	425	424	5200	5199	32
CM_f_sinh_1_1L	1712	1288	14200	12222	32
CM_f_sinh_2_1L	2062	1154	15200	13337	32
CM_spread_with_copy_1L	2212	1624	3180	2429	32
CM_spread_with_f_add_1L	3209	2353	6155	4401	32
CM_spread_with_f_max_1L	2984	2325	5972	4577	32
CM_spread_with_f_min_1L	3038	2376	5766	5211	32
CM_spread_with_logand_1L	2374	1690	4691	4311	32
CM_spread_with_logior_1L	2284	1714	4565	3496	32
CM_spread_with_logxor_1L	2277	1731	4714	3884	32
CM_spread_with_s_add_1L	2337	1747	4370	3249	32
CM_spread_with_s_max_1L	2778	2046	5514	4904	32
CM_spread_with_s_min_1L	2935	2427	6064	4377	32
CM_spread_with_u_add_1L	2291	1859	4004	3378	32
CM_spread_with_u_max_1L	2835	2342	5652	4826	32
CM_spread_with_u_min_1L	2897	2342	6384	4836	32
CM_f_sqrt_2_1L	112	112	1799	1799	32
CM_store_context	5	3	22	22	32
CM_store_overflow	5	5	51	51	32
CM_store_overflow_always	7	2	19	19	32
CM_store_test	9	5	54	54	32
CM_store_test_always	12	2	19	19	32
CM_f_sub_const_mult_1L	94	74	580	580	32
CM_f_sub_const_mult_const_1L	72	60	516	516	32
CM_f_sub_mult_1L	100	53	661	661	32

Instruction	Vpr 1 real time	Vpr 1 CM time	Vpr16 real time	Vpr16 CM time	field width
CM_f_sub_mult_const_1L	81	70	564	564	32
CM_f_subf_const_mult_1L	88	56	564	564	32
CM_f_subf_const_mult_const_1L	75	52	532	532	32
CM_f_subfrom_2_1L	82	62	532	532	32
CM_s_subfrom_2_1L	167	78	1338	1338	32
CM_u_subfrom_2_1L	136	107	1306	1306	32
CM_f_subfrom_always_2_1L	91	58	467	467	32
CM_f_subfrom_const_always_2_1L	63	53	338	338	32
CM_f_subfrom_const_always_3_1L	68	47	338	338	32
CM_f_subfrom_constant_2_1L	78	57	532	532	32
CM_f_subfrom_constant_3_1L	83	65	435	435	32
CM_s_subfrom_constant_2_1L	127	83	1225	1158	32
CM_s_subfrom_constant_3_1L	153	90	1241	1168	32
CM_u_subfrom_constant_2_1L	153	108	1225	1225	32
CM_u_subfrom_constant_3_1L	136	100	1241	1241	32
CM_f_subtract_2_1L	95	67	532	532	32
CM_f_subtract_3_1L	65	55	645	645	32
CM_s_subtract_2_1L	48	36	467	467	32
CM_s_subtract_3_1L	151	71	1225	1225	32
CM_s_subtract_3_3L	97	80	774	774	32
CM_u_subtract_2_1L	57	55	451	451	32
CM_u_subtract_3_1L	150	98	1225	1225	32
CM_u_subtract_3_3L	89	62	758	758	32
CM_f_subtract_always_2_1L	91	60	435	435	32
CM_f_subtract_always_3_1L	74	54	435	435	32
CM_f_subtract_const_always_2_1L	75	46	338	338	32
CM_f_subtract_const_always_3_1L	68	56	435	435	32
CM_f_subtract_constant_2_1L	80	48	500	499	32
CM_f_subtract_constant_3_1L	88	76	532	532	32
CM_s_subtract_constant_2_1L	63	39	564	564	32
CM_s_subtract_constant_3_1L	138	94	1112	1112	32
CM_u_subtract_constant_2_1L	68	46	532	532	32
CM_u_subtract_constant_3_1L	136	102	983	983	32
CM_swap_2_1L	144	98	1290	1290	32
CM_f_tan_1_1L	362	362	5599	5598	32
CM_f_tan_2_1L	449	363	5799	5799	32
CM_f_tanh_1_1L	1674	1037	10400	10397	32
CM_f_tanh_2_1L	1625	1108	10400	10398	32
CM_u_to_gray_code_1_1L	114	94	1193	1193	32
CM_u_to_gray_code_2_1L	89	78	870	870	32
CM_f_f_truncate_1_1L	643	387	4983	4821	32
CM_f_f_truncate_2_1L	673	425	5532	5010	32
CM_s_truncate_2_1L	1634	1468	25387	22289	32
CM_s_truncate_3_1L	1699	1471	25000	23151	32
CM_s_truncate_3_3L	1627	1459	25177	23044	32
CM_s_f_truncate_2_2L	531	428	8306	7135	32
CM_u_truncate_2_1L	1545	1315	23629	20593	32
CM_u_truncate_3_1L	1532	1291	23547	20400	32
CM_s_truncate_constant_2_1L	1887	1593	24790	22548	32
CM_s_truncate_constant_3_1L	1827	1602	25016	23560	32
CM_u_truncate_constant_2_1L	1679	1416	23531	21105	32
CM_u_truncate_constant_3_1L	1631	1346	23902	21881	32
CM_vp_set_geometry	5	0	5	0	32
CM_f_write_to_news_array_1L	32900	31507	561670	558200	32
CM_s_write_to_news_array_1L	33100	30674	548330	548030	32

Instruction	Vpr 1 real time	Vpr 1 CM time	Vpr16 real time	Vpr16 CM time	field width
CM_u_write_to_news_array_1L	33500	32770	524999	524580	32
CM_f_write_to_processor_1L	425	42	428	87	32
CM_s_write_to_processor_1L	58	55	161	161	32
CM_u_write_to_processor_1L	45	44	32	32	32
CM_u_write_to_processor_1L	45	44	32	32	32

Instruction	Vpr 1 real time	Vpr 1 CM time	Vpr16 real time	Vpr16 CM time	field width
CM_f_abs_1_1L	11	7	25	25	64
CM_f_abs_2_1L	75	67	794	794	64
CM_f_c_abs_2_1L	687	572	6000	5997	64
CM_s_abs_1_1L	157	108	1354	1354	64
CM_s_abs_2_1L	136	112	1661	1661	64
CM_s_abs_2_2L	148	108	1661	1661	64
CM_c_acos_1_1L	9399	6126	67199	67198	64
CM_c_acos_2_1L	9624	6933	67199	67198	64
CM_f_acos_1_1L	3887	2758	27710	27365	64
CM_f_acos_2_1L	3905	2824	27644	27385	64
CM_f_acosh_1_1L	2708	1832	19386	19386	64
CM_f_acosh_2_1L	2717	1784	19290	19290	64
CM_c_add_2_1L	274	270	2199	2199	64
CM_c_add_3_1L	300	299	2000	1999	64
CM_f_add_2_1L	150	99	1064	1064	64
CM_f_add_3_1L	148	114	1064	1064	64
CM_s_add_2_1L	84	72	846	846	64
CM_s_add_3_1L	234	156	2274	2274	64
CM_s_add_3_3L	125	110	1403	1403	64
CM_u_add_2_1L	82	72	822	822	64
CM_u_add_3_1L	228	155	2258	2257	64
CM_u_add_3_3L	115	91	1096	1096	64
CM_c_add_always_2_1L	251	178	1871	1870	64
CM_c_add_always_3_1L	273	236	1887	1887	64
CM_f_add_always_2_1L	132	99	935	935	64
CM_f_add_always_3_1L	136	119	935	935	64
CM_s_add_carry_2_1L	96	63	854	854	64
CM_s_add_carry_3_1L	257	178	2483	2483	64
CM_s_add_carry_3_3L	269	177	2451	2451	64
CM_u_add_carry_2_1L	68	66	854	854	64
CM_f_add_const_always_2_1L	93	91	596	596	64
CM_f_add_const_always_3_1L	99	79	596	596	64
CM_f_add_const_mult_1L	151	140	1112	1112	64
CM_f_add_const_mult_const_1L	130	109	919	919	64
CM_f_add_constant_2_1L	127	115	790	790	64
CM_f_add_constant_3_1L	138	114	806	806	64
CM_s_add_constant_2_1L	127	124	1064	1064	64
CM_s_add_constant_3_1L	214	184	1854	1854	64
CM_u_add_constant_2_1L	134	115	1032	1032	64
CM_u_add_constant_3_1L	214	175	1838	1838	64
CM_s_add_flags_2_1L	79	73	854	854	64
CM_u_add_flags_2_1L	81	67	838	838	64
CM_f_add_mult_1L	163	115	1370	1370	64
CM_f_add_mult_const_1L	180	158	1096	1096	64
CM_f_asin_1_1L	3525	2197	27400	27398	64
CM_f_asin_2_1L	4000	2849	27400	27398	64
CM_f_asinh_1_1L	2300	1784	17599	17597	64
CM_f_asinh_2_1L	2400	1894	17599	17598	64
CM_f_atan_1_1L	3100	2357	21800	21797	64
CM_f_atan_2_1L	3062	2116	21800	21798	64
CM_f_atan2_3_1L	3637	2749	24399	24398	64
CM_f_atanh_1_1L	3074	2071	23199	22092	64
CM_f_atanh_2_1L	3262	2172	24199	22552	64
CM_f_f_ceiling_1_1L	1674	1248	16774	12327	64
CM_f_f_ceiling_2_1L	1777	1195	17596	13868	64

Instruction	Vpr 1 real time	Vpr 1 CM time	Vpr16 real time	Vpr16 CM time	field width
CM_clear_all_flags	25	10	80	80	64
CM_clear_all_flags_always	44	18	48	48	64
CM_clear_bit	15	5	48	48	64
CM_clear_bit_always	18	8	64	64	64
CM_clear_context	6	2	19	19	64
CM_clear_overflow	20	9	57	42	64
CM_clear_test	18	5	60	42	64
CM_c_conjugate_1_1L	18	9	48	48	64
CM_c_conjugate_2_1L	161	139	1628	1628	64
CM_c_cos_1_1L	8399	5906	64400	62964	64
CM_c_cos_2_1L	8425	6156	61000	60290	64
CM_f_cos_1_1L	524	524	8000	7999	64
CM_f_cos_2_1L	537	537	8000	7999	64
CM_f_cosh_1_1L	2712	2004	19600	19598	64
CM_f_cosh_2_1L	2937	2035	21999	19810	64
CM_create_detailed_geometry	629	0	629	0	64
CM_create_geometry	882	0	882	0	64
CM_deposit_news_coordinate_1L	94	71	771	744	64
CM_c_divide_2_1L	1612	1214	13199	13198	64
CM_c_divide_3_1L	1787	1627	12000	11998	64
CM_f_divide_2_1L	162	162	2199	2199	64
CM_f_divide_3_1L	250	249	2400	2399	64
CM_c_divide_always_2_1L	1862	1853	14200	14198	64
CM_c_divide_always_3_1L	1850	1391	13000	12999	64
CM_f_divide_always_2_1L	262	262	2199	2199	64
CM_f_divide_always_3_1L	274	274	2199	2198	64
CM_f_divide_const_always_3_1L	137	137	1799	1798	64
CM_f_divide_constant_2_1L	137	137	2000	1999	64
CM_f_divide_constant_3_1L	137	137	2000	1998	64
CM_c_divinto_2_1L	1462	1282	13199	13198	64
CM_f_divinto_2_1L	187	187	2199	2198	64
CM_c_divinto_always_2_1L	1700	1407	14200	14197	64
CM_f_divinto_always_2_1L	262	152	2199	2199	64
CM_f_divinto_const_always_2_1L	250	249	1799	1799	64
CM_f_divinto_const_always_3_1L	250	136	1799	1798	64
CM_f_divinto_constant_2_1L	349	349	2000	1999	64
CM_f_divinto_constant_3_1L	150	149	2000	1999	64
CM_enumerate_1L	1589	1531	4896	4895	64
CM_f_eq_1L	83	69	596	596	64
CM_s_eq_1L	55	47	806	806	64
CM_s_eq_2L	74	52	806	806	64
CM_u_eq_1L	78	56	806	806	64
CM_u_eq_2L	83	56	806	806	64
CM_f_eq_constant_1L	71	58	322	322	64
CM_s_eq_constant_1L	138	122	1080	1080	64
CM_u_eq_constant_1L	138	136	1080	1080	64
CM_f_eq_zero_1L	68	48	322	322	64
CM_s_eq_zero_1L	134	114	1080	1080	64
CM_u_eq_zero_1L	142	121	1080	1080	64
CM_f_exp_1_1L	960	827	9693	9693	64
CM_f_exp_2_1L	953	788	9677	9677	64
CM_extract_news_coordinate_1L	174	147	1395	1395	64
CM_fe_deposit_news_coordinate	63	0	63	0	64
CM_fe_extract_news_coordinate	58	0	58	0	64
CM_fe_from_gray_code	28	0	28	0	64

Instruction	Vpr 1 real time	Vpr 1 CM time	Vpr16 real time	Vpr16 CM time	field width
CM_fe_make_news_coordinate	65	0	65	0	64
CM_fe_make_news_mask	246	0	246	0	64
CM_field_vp_set	18	0	18	0	64
CM_f_s_float_2_2L	730	548	11500	9014	64
CM_f_u_float_2_2L	738	565	10854	8385	64
CM_f_f_floor_1_1L	1674	1168	18129	15258	64
CM_f_f_floor_2_1L	1784	1278	20047	16225	64
CM_f_ge_1L	97	50	596	596	64
CM_s_ge_1L	76	60	822	822	64
CM_s_ge_2L	86	44	822	822	64
CM_f_ge_constant_1L	63	56	322	322	64
CM_s_ge_constant_1L	140	124	1128	1128	64
CM_s_ge_zero_1L	78	78	1128	1128	64
CM_geometry_axis_length	6	0	6	0	64
CM_geometry_axis_off_chip_bits	6	0	6	0	64
CM_geometry_axis_on_chip_bits	6	0	6	0	64
CM_geometry_axis_ordering	6	0	6	0	64
CM_geometry_axis_vp_ratio	6	0	6	0	64
CM_geometry_coordinate_length	6	0	6	0	64
CM_geometry_rank	6	0	6	0	64
CM_geometry_send_address_length	6	0	6	0	64
CM_geometry_total_processors	10	0	10	0	64
CM_geometry_total_vp_ratio	6	0	6	0	64
CM_get_1L	4230	4611	33225	32604	64
CM_get_aref32_2L	9227	4608	56338	31794	64
CM_get_from_news_1L	271	237	950	907	64
CM_get_from_news_always_1L	256	231	787	784	64
CM_global_c_add_1L	600	538	1170	998	64
CM_global_count_bit	209	210	477	362	64
CM_global_count_bit_always	232	214	419	337	64
CM_global_count_context	230	199	357	336	64
CM_global_count_overflow	230	199	353	327	64
CM_global_count_test	230	209	350	332	64
CM_global_f_add_1L	300	268	529	504	64
CM_global_f_max_1L	660	514	5687	5520	64
CM_global_f_min_1L	660	619	6454	5230	64
CM_global_logand_1L	300	238	4000	3333	64
CM_global_logand_bit	49	18	111	92	64
CM_global_logand_bit_always	49	12	114	96	64
CM_global_logand_context	49	12	113	86	64
CM_global_logand_overflow	49	18	129	123	64
CM_global_logand_test	49	18	129	115	64
CM_global_logior_1L	220	148	2271	2006	64
CM_global_logior_bit	20	2	40	30	64
CM_global_logior_bit_always	20	2	32	19	64
CM_global_logior_context	19	2	41	23	64
CM_global_logior_overflow	19	2	41	23	64
CM_global_logior_test	22	5	42	24	64
CM_global_logxor_1L	14968	12570	23613	21916	64
CM_global_s_add_1L	2110	1963	72290	7178	64
CM_global_s_max_1L	577	426	4948	4188	64
CM_global_u_add_1L	609	539	1377	1012	64
CM_global_u_max_1L	505	424	4177	3943	64
CM_global_u_max_s_intlen_1L	502	441	6806	6643	64
CM_global_u_max_u_intlen_1L	500	440	6753	6535	64

Instruction	Vpr 1 real time	Vpr 1 CM time	Vpr16 real time	Vpr16 CM time	field width
CM_global_u_min_1L	699	651	5533	5198	64
CM_f_gt_1L	52	49	580	580	64
CM_s_gt_1L	75	73	822	822	64
CM_s_gt_2L	75	48	822	822	64
CM_f_gt_constant_1L	69	56	322	322	64
CM_s_gt_constant_1L	140	114	1112	1112	64
CM_f_gt_zero_1L	72	66	322	322	64
CM_s_integer_length_2_2L	1986	1986	31838	31838	64
CM_invert_context	3	2	16	16	64
CM_invert_overflow	2	2	32	32	64
CM_invert_test_always	3	2	19	19	64
CM_is_field_in_heap	7	0	7	0	64
CM_is_field_in_stack	7	0	7	0	64
CM_s_isqrt_1_1L	2279	2279	36451	36451	64
CM_s_isqrt_2_1L	2279	2279	36451	36451	64
CM_s_isqrt_2_2L	2287	2287	36499	36499	64
CM_u_isqrt_2_1L	2309	2309	36887	36887	64
CM_u_isqrt_2_2L	2314	2314	36935	36935	64
CM_f_le_1L	46	45	580	580	64
CM_s_le_1L	44	43	806	806	64
CM_s_le_2L	57	47	822	822	64
CM_u_le_1L	75	67	806	806	64
CM_u_le_2L	81	53	822	822	64
CM_f_le_constant_1L	62	61	338	338	64
CM_s_le_constant_1L	148	122	1128	1128	64
CM_u_le_constant_1L	136	113	1080	1080	64
CM_f_le_zero_1L	49	48	322	322	64
CM_s_le_zero_1L	140	126	1128	1128	64
CM_u_le_zero_1L	138	125	1064	1064	64
CM_f_ln_1_1L	811	753	8048	8048	64
CM_f_ln_2_1L	829	733	8048	8048	64
CM_load_context	7	3	48	45	64
CM_load_overflow	4	4	64	64	64
CM_load_overflow_always	4	2	16	16	64
CM_load_test	6	5	48	48	64
CM_load_test_always	6	3	32	32	64
CM_f_log10_1_1L	822	777	8048	8048	64
CM_f_log10_2_1L	791	745	8048	8048	64
CM_logand_2_1L	81	62	796	796	64
CM_logand_3_1L	252	169	2410	2410	64
CM_logand_constant_2_1L	162	116	1339	1339	64
CM_logand_constant_3_1L	247	186	2157	2156	64
CM_logand_context	7	3	25	25	64
CM_logand_context_with_test	5	2	19	19	64
CM_logand_overflow	12	4	41	41	64
CM_logand_overflow_always	7	3	25	25	64
CM_logand_test	8	4	38	38	64
CM_logand_test_always	8	3	25	25	64
CM_logandc1_2_1L	78	66	798	798	64
CM_logandc1_3_1L	251	172	2410	2410	64
CM_logandc1_constant_2_1L	162	122	1339	1339	64
CM_logandc1_constant_3_1L	253	188	2145	2145	64
CM_logandc2_2_1L	76	55	806	806	64
CM_logandc2_3_1L	250	208	2419	2419	64
CM_logandc2_constant_3_1L	165	142	2145	2145	64

	Vpr 1		Vpr16		field
Instruction	real time	CM time	real time	CM time	width
CM_s_logcount_2_2L	326	326	5322	5322	64
CM_u_logcount_2_2L	326	326	5306	5306	64
CM_logeqv_2_1L	44	43	790	790	64
CM_logeqv_3_1L	151	146	2403	2403	64
CM_logeqv_constant_2_1L	159	115	1338	1338	64
CM_logeqv_constant_3_1L	237	174	2145	2145	64
CM_logior_2_1L	77	62	806	806	64
CM_logior_3_1L	246	196	2419	2419	64
CM_logior_constant_2_1L	182	139	1338	1338	64
CM_logior_constant_3_1L	243	191	2145	2145	64
CM_logior_context	8	3	25	25	64
CM_logior_overflow	4	4	48	48	64
CM_logior_overflow_always	4	3	32	32	64
CM_logior_test	4	4	32	32	64
CM_logior_test_always	4	3	32	32	64
CM_lognand_2_1L	82	61	790	790	64
CM_lognand_3_1L	253	152	2419	2419	64
CM_lognand_constant_2_1L	157	116	1338	1338	64
CM_lognand_constant_3_1L	246	173	2161	2161	64
CM_lognor_2_1L	79	73	806	806	64
CM_lognor_3_1L	255	178	2419	2419	64
CM_lognor_constant_2_1L	163	134	1338	1338	64
CM_lognor_constant_3_1L	251	163	2177	2177	64
CM_lognot_1_1L	70	63	645	645	64
CM_lognot_2_1L	74	69	790	790	64
CM_logorc1_2_1L	67	62	806	806	64
CM_logorc1_3_1L	262	224	2419	2419	64
CM_logorc1_constant_2_1L	163	129	1338	1338	64
CM_logorc1_constant_3_1L	230	195	2145	2145	64
CM_logorc2_2_1L	75	53	806	806	64
CM_logorc2_3_1L	241	188	2419	2419	64
CM_logorc2_constant_2_1L	165	115	1338	1338	64
CM_logorc2_constant_3_1L	234	178	2145	2145	64
CM_logxor_2_1L	84	56	806	805	64
CM_logxor_3_1L	241	191	2419	2419	64
CM_logxor_constant_2_1L	167	120	1354	1354	64
CM_logxor_constant_3_1L	247	169	2145	2145	64
CM_f_lt_1L	101	65	580	580	64
CM_s_lt_1L	87	53	806	806	64
CM_s_lt_2L	83	57	806	806	64
CM_u_lt_1L	79	65	806	806	64
CM_u_lt_2L	91	67	822	822	64
CM_f_lt_constant_1L	66	52	338	338	64
CM_s_lt_constant_1L	144	114	1128	1128	64
CM_u_lt_constant_1L	134	121	1080	1080	64
CM_f_lt_zero_1L	65	57	322	322	64
CM_s_lt_zero_1L	144	127	1128	1128	64
CM_u_lt_zero_1L	146	119	1064	1064	64
CM_make_news_coordinate_1L	114	84	949	928	64
CM_f_max_2_1L	144	114	1258	1258	64
CM_f_max_3_1L	174	157	1274	1274	64
CM_s_max_2_1L	142	107	1580	1580	64
CM_s_max_3_1L	297	209	3193	3193	64
CM_s_max_3_3L	319	273	3177	3177	64
CM_u_max_2_1L	138	130	1580	1580	64

Instruction	Vpr 1 real time	Vpr 1 CM time	Vpr16 real time	Vpr16 CM time	field width
CM_u_max_3_1L	311	267	3193	3193	64
CM_u_max_3_3L	320	250	3161	3161	64
CM_f_max_constant_3_1L	81	81	1032	1032	64
CM_s_max_constant_2_1L	218	185	2128	2128	64
CM_s_max_constant_3_1L	320	242	2935	2935	64
CM_u_max_constant_2_1L	246	154	1725	1725	64
CM_u_max_constant_3_1L	390	233	2903	2802	64
CM_f_min_2_1L	159	116	1274	1274	64
CM_f_min_3_1L	171	154	1274	1274	64
CM_s_min_2_1L	143	113	1580	1580	64
CM_s_min_3_1L	304	250	3193	3193	64
CM_s_min_3_3L	311	231	3161	3161	64
CM_u_min_2_1L	130	108	1564	1564	64
CM_u_min_3_1L	318	258	3193	3193	64
CM_u_min_3_3L	311	211	3161	3161	64
CM_f_min_constant_2_1L	148	148	1064	1064	64
CM_f_min_constant_3_1L	150	126	1016	1016	64
CM_s_min_constant_2_1L	224	179	2128	2128	64
CM_s_min_constant_3_1L	316	250	2919	2919	64
CM_u_min_constant_2_1L	277	177	1725	1725	64
CM_u_min_constant_3_1L	357	292	2725	2666	64
CM_f_mod_2_1L	9215	7162	90999	90976	64
CM_f_mod_3_1L	9010	7002	90984	90938	64
CM_s_mod_2_1L	3914	3909	63274	63196	64
CM_s_mod_3_1L	3961	3956	64080	64080	64
CM_u_mod_2_1L	3721	3721	60065	60063	64
CM_u_mod_3_1L	3770	3764	60871	60789	64
CM_f_mod_constant_2_1L	8910	6837	91289	91289	64
CM_f_mod_constant_3_1L	9400	7643	92983	92983	64
CM_s_mod_constant_2_1L	3957	3953	64209	63750	64
CM_s_mod_constant_3_1L	4000	3995	64644	64548	64
CM_u_mod_constant_2_1L	3765	3762	60564	60554	64
CM_u_mod_constant_3_1L	3807	3807	61402	61351	64
CM_f_move_1L	44	43	790	790	64
CM_s_move_1L	80	76	806	806	64
CM_s_move_2L	73	57	838	838	64
CM_u_move_1L	81	73	806	806	64
CM_u_move_2L	80	66	870	870	64
CM_f_move_always_1L	63	62	629	629	64
CM_s_move_always_1L	63	42	629	629	64
CM_u_move_always_1L	39	32	629	629	64
CM_f_move_const_always_1L	59	46	354	354	64
CM_f_move_constant_1L	61	51	532	532	64
CM_s_move_constant_1L	81	74	548	548	64
CM_u_move_constant_1L	62	54	548	548	64
CM_move_reversed_1L	99	72	967	967	64
CM_f_move_zero_1L	59	43	483	483	64
CM_s_move_zero_1L	60	45	500	499	64
CM_u_move_zero_1L	57	36	483	483	64
CM_f_move_zero_always_1L	55	54	322	322	64
CM_s_move_zero_always_1L	25	21	322	322	64
CM_u_move_zero_always_1L	49	29	322	322	64
CM_f_mult_add_1L	156	122	1370	1370	64
CM_f_mult_add_const_1L	167	150	1193	1193	64
CM_f_mult_const_add_1L	163	162	1112	1112	64

Instruction	Vpr 1 real time	Vpr 1 CM time	Vpr16 real time	Vpr16 CM time	field width
CM_f_mult_const_add_const_1L	134	132	919	919	64
CM_f_mult_const_sub_1L	165	143	1096	1096	64
CM_f_mult_const_sub_const_1L	127	127	919	919	64
CM_f_mult_const_subf_const_1L	73	72	919	919	64
CM_f_mult_sub_1L	176	145	1354	1354	64
CM_f_mult_sub_const_1L	167	163	1112	1112	64
CM_f_mult_subf_1L	188	146	1354	1354	64
CM_f_mult_subf_const_1L	171	156	1096	1096	64
CM_f_multiply_2_1L	143	104	1056	1056	64
CM_f_multiply_3_1L	157	137	1048	1048	64
CM_s_multiply_2_1L	3657	3657	58451	58451	64
CM_s_multiply_3_1L	3697	3697	59112	59112	64
CM_s_multiply_3_3L	1974	1974	31532	31532	64
CM_u_multiply_2_1L	3400	3400	54322	54322	64
CM_u_multiply_3_1L	3400	3400	54354	54354	64
CM_u_multiply_3_3L	1747	1747	27886	27886	64
CM_f_multiply_always_2_1L	68	67	935	935	64
CM_f_multiply_always_3_1L	83	79	935	935	64
CM_f_multiply_const_always_2_1L	102	93	596	596	64
CM_f_multiply_const_always_3_1L	112	70	596	596	64
CM_f_multiply_constant_2_1L	118	114	790	790	64
CM_f_multiply_constant_3_1L	123	102	790	790	64
CM_s_multiply_constant_2_1L	3701	3701	58984	58984	64
CM_s_multiply_constant_3_1L	3740	3740	59661	59661	64
CM_u_multiply_constant_2_1L	3439	3439	54871	54871	64
CM_u_multiply_constant_3_1L	3483	3483	55677	55677	64
CM_multispread_f_add_1L	588	258	3844	2671	64
CM_multispread_f_max_1L	764	408	5919	4628	64
CM_multispread_f_min_1L	805	444	6653	5832	64
CM_multispread_logand_1L	704	349	4973	3784	64
CM_multispread_logior_1L	635	287	4920	3225	64
CM_multispread_logxor_1L	636	272	4746	3246	64
CM_multispread_s_add_1L	649	261	4613	3333	64
CM_multispread_s_max_1L	680	304	6080	4534	64
CM_multispread_s_min_1L	757	438	6186	4912	64
CM_multispread_u_add_1L	651	261	4600	3423	64
CM_multispread_u_max_1L	636	320	6440	5062	64
CM_multispread_u_min_1L	756	414	6279	4889	64
CM_my_news_coordinate_1L	254	222	1092	1092	64
CM_my_send_address_1L	121	43	820	696	64
CM_f_ne_1L	62	48	580	580	64
CM_s_ne_1L	87	73	806	806	64
CM_s_ne_2L	73	64	806	806	64
CM_u_ne_1L	91	45	806	806	64
CM_u_ne_2L	70	66	806	806	64
CM_f_ne_constant_1L	57	47	322	322	64
CM_s_ne_constant_1L	136	107	1080	1080	64
CM_u_ne_constant_1L	138	133	1080	1080	64
CM_f_ne_zero_1L	46	44	322	322	64
CM_s_ne_zero_1L	134	125	1064	1064	64
CM_u_ne_zero_1L	140	104	1064	1064	64
CM_f_negate_1_1L	23	9	32	32	64
CM_f_negate_2_1L	77	64	790	790	64
CM_s_negate_1_1L	76	76	677	677	64
CM_s_negate_2_1L	91	58	822	822	64

Instruction	Vpr 1 real time	Vpr 1 CM time	Vpr16 real time	Vpr16 CM time	field width
CM_s_negate_2_2L	163	156	1661	1661	64
CM_u_negate_1_1L	89	65	677	677	64
CM_u_negate_2_1L	74	45	822	822	64
CM_u_negate_2_2L	172	160	1693	1693	64
CM_f_news_add_2_1L	423	277	1408	1187	64
CM_f_news_add_always_2_1L	333	247	1343	1056	64
CM_f_news_add_always_3_1L	380	247	1472	1229	64
CM_f_news_add_const_3_1L	402	284	2015	1424	64
CM_f_news_add_const_a_3_1L	365	274	1424	994	64
CM_f_news_add_mult_4_1L	445	328	2512	1862	64
CM_f_news_mult_2_1L	396	273	2159	1700	64
CM_f_news_mult_3_1L	410	300	2240	1375	64
CM_f_news_mult_add_4_1L	461	371	2895	2167	64
CM_f_news_mult_always_2_1L	380	295	1983	1629	64
CM_f_news_mult_always_3_1L	400	262	2000	1570	64
CM_f_news_mult_const_3_1L	394	271	1903	1268	64
CM_f_news_mult_const_a_3_1L	348	278	1392	894	64
CM_f_news_mult_sub_4_1L	468	321	2767	1758	64
CM_f_news_sub_2_1L	407	260	2416	1675	64
CM_f_news_sub_3_1L	442	322	2223	1728	64
CM_f_news_sub_always_2_1L	387	229	2240	1505	64
CM_f_news_sub_always_3_1L	390	267	2240	1451	64
CM_f_news_sub_const_3_1L	387	280	1711	1052	64
CM_f_news_sub_const_a_3_1L	387	298	1744	920	64
CM_f_c_phase_2_1L	3812	2546	29999	28140	64
CM_physical_vp_set	3	0	3	0	64
CM_c_c_power_2_1L	10750	6804	75999	75809	64
CM_c_c_power_3_1L	10224	7684	75999	75998	64
CM_f_f_power_2_1L	2774	1973	20500	20498	64
CM_f_f_power_3_1L	2374	1505	20500	20498	64
CM_f_s_power_2_2L	25399	15829	163000	163000	64
CM_f_s_power_3_2L	25875	15514	165500	163029	64
CM_f_u_power_2_2L	25474	16398	158500	158500	64
CM_f_u_power_3_2L	24350	16210	158999	158559	64
CM_s_s_power_2_1L	463939	463669	7384500	7384500	64
CM_s_s_power_3_1L	463880	463699	7385000	7385000	64
CM_s_s_power_3_3L	230220	230159	3665499	3665499	64
CM_s_u_power_3_3L	226050	225889	3598000	3598000	64
CM_u_s_power_3_3L	216649	216450	3444999	3444999	64
CM_u_u_power_2_1L	432080	431840	6876999	6876999	64
CM_u_u_power_3_1L	432440	431870	6877999	6877999	64
CM_u_u_power_3_3L	212569	212430	3381999	3381999	64
CM_f_f_power_constant_2_1L	1374	1374	19999	19993	64
CM_f_f_power_constant_3_1L	1599	1374	19500	19497	64
CM_f_s_power_constant_2_1L	274	274	2499	2497	64
CM_f_s_power_constant_3_1L	150	149	2000	1997	64
CM_f_u_power_constant_2_1L	99	44	600	599	64
CM_f_u_power_constant_3_1L	49	49	799	799	64
CM_s_s_power_constant_2_1L	7300	7299	115999	115999	64
CM_s_s_power_constant_3_1L	7325	7324	116999	116999	64
CM_s_s_power_constant_3_2L	7350	7349	116999	116999	64
CM_s_u_power_constant_2_1L	7300	7299	116499	116499	64
CM_s_u_power_constant_3_1L	7350	7349	116999	116999	64
CM_s_u_power_constant_3_2L	7325	7324	117499	117490	64
CM_u_s_power_constant_2_1L	6825	6824	108999	108989	64

Instruction	Vpr 1 real time	Vpr 1 CM time	Vpr16 real time	Vpr16 CM time	field width
CM_u_s_power_constant_3_1L	6899	6899	108999	108999	64
CM_u_s_power_constant_3_2L	6825	6824	108999	108999	64
CM_u_u_power_constant_2_1L	6825	6824	108499	108499	64
CM_u_u_power_constant_3_1L	6825	6824	108999	108989	64
CM_f_random_1L	2206	2190	34306	34306	64
CM_u_random_1L	6885	6885	110050	110050	64
CM_s_read_from_news_array_1L	31099	28728	600000	570930	64
CM_u_read_from_news_array_1L	43499	39889	855000	849579	64
CM_f_read_from_processor_1L	381	139	419	118	64
CM_s_read_from_processor_1L	306	119	322	106	64
CM_u_read_from_processor_1L	292	112	451	116	64
CM_reduce_with_f_add_1L	5033	4039	7893	5942	64
CM_reduce_with_f_max_1L	5532	4413	9266	7843	64
CM_reduce_with_f_min_1L	5492	4395	9186	7751	64
CM_reduce_with_logand_1L	4439	3645	7299	6066	64
CM_reduce_with_logior_1L	4251	3290	6681	5370	64
CM_reduce_with_logxor_1L	4251	3347	6910	5707	64
CM_reduce_with_s_add_1L	4325	3303	6636	5427	64
CM_reduce_with_s_max_1L	5226	4284	8933	7654	64
CM_reduce_with_s_min_1L	5454	4294	8873	7521	64
CM_reduce_with_u_add_1L	4309	3389	6719	5061	64
CM_reduce_with_u_max_1L	5214	4260	8426	7155	64
CM_reduce_with_u_min_1L	5466	4401	9093	7571	64
CM_f_rem_2_1L	7482	6279	88257	88257	64
CM_f_rem_3_1L	7652	6221	87886	87886	64
CM_s_rem_2_1L	3839	3839	61999	61999	64
CM_s_rem_3_1L	3898	3898	62774	62726	64
CM_u_rem_2_1L	3721	3721	60097	60022	64
CM_u_rem_3_1L	3766	3766	60984	60880	64
CM_f_rem_constant_2_1L	7873	6482	88613	88613	64
CM_f_rem_constant_3_1L	7654	6440	89468	89468	64
CM_s_rem_constant_2_1L	3892	3885	62515	62474	64
CM_s_rem_constant_3_1L	3930	3925	63386	63386	64
CM_u_rem_constant_2_1L	3766	3763	60589	60589	64
CM_u_rem_constant_3_1L	3813	3810	61354	61354	64
CM_f_f_round_1_1L	1978	1477	24483	21609	64
CM_f_f_round_2_1L	2165	1496	26289	22137	64
CM_scan_with_copy_1L	4538	3702	11144	11141	64
CM_scan_with_f_add_1L	6604	5809	10595	9104	64
CM_scan_with_f_max_1L	3777	2814	10389	10022	64
CM_scan_with_f_min_1L	3661	2876	10641	10257	64
CM_scan_with_f_multiply_1L	6975	5986	12311	11149	64
CM_scan_with_logand_1L	5244	4289	9931	7993	64
CM_scan_with_logior_1L	1342	1323	4347	4332	64
CM_scan_with_logxor_1L	5298	4324	8306	7119	64
CM_scan_with_s_add_1L	1332	1325	4347	4347	64
CM_scan_with_s_max_1L	1885	1877	6453	6452	64
CM_scan_with_s_min_1L	1923	1910	6498	6498	64
CM_scan_with_u_add_1L	1354	1344	4347	4347	64
CM_scan_with_u_max_1L	1872	1872	6292	6292	64
CM_send_1L	2084	2084	63753	63020	64
CM_send_aset32_logior_1L	10420	4988	67941	66862	64
CM_send_aset32_overwrite_1L	6766	3255	44050	43202	64
CM_send_aset32_u_add_1L	10567	5027	69199	68140	64
CM_send_to_news_1L	389	276	1532	1257	64

	Vpr 1		Vpr16		field
Instruction	real time	CM time	real time	CM time	width
CM_send_to_news_always_1L	254	209	787	773	64
CM_send_with_f_add_1L	11629	5698	189940	188130	64
CM_send_with_f_max_1L	6112	5054	124739	124739	64
CM_send_with_f_min_1L	6412	4987	123300	123290	64
CM_send_with_logand_1L	2334	2334	86223	86223	64
CM_send_with_logior_1L	2099	2099	83752	83751	64
CM_send_with_overwrite_1L	1535	1530	53249	53249	64
CM_send_with_s_add_1L	2099	2099	107160	107160	64
CM_send_with_s_max_1L	3028	3028	118629	118629	64
CM_send_with_s_min_1L	3271	3271	122060	122060	64
CM_send_with_u_add_1L	2099	2099	106159	106150	64
CM_send_with_u_max_1L	3025	3025	120090	120090	64
CM_send_with_u_min_1L	3257	3257	123680	123680	64
CM_set_bit	11	9	48	48	64
CM_set_bit_always	12	7	32	32	64
CM_set_context	3	2	32	32	64
CM_set_overflow	11	9	48	48	64
CM_set_test	11	8	48	48	64
CM_set_vp_set	3	0	3	0	64
CM_set_vp_set_geometry	3	0	3	0	64
CM_c_c_signum_1_1L	1643	1244	12338	12338	64
CM_c_c_signum_2_1L	1823	1276	14193	14193	64
CM_f_f_signum_1_1L	70	53	612	612	64
CM_f_f_signum_2_1L	80	70	612	612	64
CM_s_s_signum_2_1L	134	123	1128	1128	64
CM_s_s_signum_2_2L	117	88	1112	1112	64
CM_f_sin_1_1L	712	712	8600	8598	64
CM_f_sin_2_1L	787	787	8799	8799	64
CM_f_sinh_1_1L	2924	1951	20800	20798	64
CM_f_sinh_2_1L	3375	2600	22399	22399	64
CM_spread_with_copy_1L	3744	2925	5354	3279	64
CM_spread_with_f_add_1L	4732	3892	8558	6865	64
CM_spread_with_f_max_1L	5217	4105	10777	9007	64
CM_spread_with_f_min_1L	5327	4447	10389	8428	64
CM_spread_with_logand_1L	4188	3470	8054	6984	64
CM_spread_with_logior_1L	4009	3144	7734	7141	64
CM_spread_with_logxor_1L	4018	3218	8901	6478	64
CM_spread_with_s_add_1L	3971	3245	7551	6124	64
CM_spread_with_s_max_1L	5041	4175	10435	8620	64
CM_spread_with_s_min_1L	5201	4139	10366	8601	64
CM_spread_with_u_add_1L	3974	3191	7414	5807	64
CM_spread_with_u_max_1L	4980	3854	10228	7730	64
CM_spread_with_u_min_1L	5255	4338	10525	8507	64
CM_f_sqrt_2_1L	224	224	3599	3599	64
CM_store_context	5	2	22	22	64
CM_store_overflow	9	5	51	51	64
CM_store_overflow_always	10	3	19	19	64
CM_store_test	12	5	51	51	64
CM_store_test_always	9	3	19	19	64
CM_f_sub_const_mult_1L	157	133	1112	1112	64
CM_f_sub_const_mult_const_1L	148	136	919	919	64
CM_f_sub_mult_1L	163	110	1370	1370	64
CM_f_sub_mult_const_1L	169	145	1112	1112	64
CM_f_subf_const_mult_1L	178	167	1096	1096	64
CM_f_subf_const_mult_const_1L	144	128	919	919	64

Instruction	Vpr 1 real time	Vpr 1 CM time	Vpr16 real time	Vpr16 CM time	field width
CM_f_subfrom_2_1L	151	136	1064	1064	64
CM_s_subfrom_2_1L	241	202	2467	2467	64
CM_u_subfrom_2_1L	246	160	2451	2451	64
CM_f_subfrom_always_2_1L	121	114	935	935	64
CM_f_subfrom_const_always_2_1L	112	110	596	596	64
CM_f_subfrom_const_always_3_1L	103	86	596	596	64
CM_f_subfrom_constant_2_1L	121	90	790	790	64
CM_f_subfrom_constant_3_1L	118	80	806	806	64
CM_s_subfrom_constant_2_1L	245	199	2209	2209	64
CM_s_subfrom_constant_3_1L	254	184	2193	2193	64
CM_u_subfrom_constant_2_1L	251	199	2193	2193	64
CM_u_subfrom_constant_3_1L	251	179	2193	2193	64
CM_f_subtract_2_1L	134	107	1064	1064	64
CM_f_subtract_3_1L	150	118	1064	1064	64
CM_s_subtract_2_1L	71	47	854	854	64
CM_s_subtract_3_1L	239	181	2274	2274	64
CM_s_subtract_3_3L	128	127	1387	1387	64
CM_u_subtract_2_1L	68	46	822	822	64
CM_u_subtract_3_1L	239	187	2258	2257	64
CM_u_subtract_3_3L	134	87	1161	1161	64
CM_f_subtract_always_2_1L	118	86	935	935	64
CM_f_subtract_always_3_1L	127	126	951	951	64
CM_f_subtract_const_always_2_1L	106	74	596	596	64
CM_f_subtract_const_always_3_1L	96	65	596	596	64
CM_f_subtract_constant_2_1L	138	122	790	790	64
CM_f_subtract_constant_3_1L	125	107	790	790	64
CM_s_subtract_constant_2_1L	140	110	1048	1048	64
CM_s_subtract_constant_3_1L	241	213	1854	1854	64
CM_u_subtract_constant_2_1L	128	112	1048	1048	64
CM_u_subtract_constant_3_1L	211	195	1854	1854	64
CM_swap_2_1L	261	197	2419	2419	64
CM_f_tan_1_1L	2025	1361	19799	19797	64
CM_f_tan_2_1L	2524	1856	19799	19798	64
CM_f_tanh_1_1L	2325	1801	18999	18998	64
CM_f_tanh_2_1L	2749	2125	18999	18998	64
CM_u_to_gray_code_1_1L	224	161	2322	2322	64
CM_u_to_gray_code_2_1L	148	121	1709	1709	64
CM_f_f_truncate_1_1L	1032	746	9451	9451	64
CM_f_f_truncate_2_1L	1077	694	8806	8806	64
CM_s_truncate_2_1L	4066	4065	65516	65516	64
CM_s_truncate_3_1L	4067	4067	65660	65660	64
CM_s_truncate_3_3L	4052	4048	65209	65118	64
CM_s_f_truncate_2_2L	889	889	14774	14774	64
CM_u_truncate_2_1L	3804	3799	61184	61174	64
CM_u_truncate_3_1L	3795	3795	61193	61193	64
CM_s_truncate_constant_2_1L	4110	4110	66031	66031	64
CM_s_truncate_constant_3_1L	4112	4112	66096	66096	64
CM_u_truncate_constant_2_1L	3845	3843	61806	61778	64
CM_u_truncate_constant_3_1L	3837	3837	61726	61726	64
CM_vp_set_geometry	5	0	5	0	64
CM_s_write_to_news_array_1L	35599	30181	586669	490619	64
CM_u_write_to_news_array_1L	35500	32737	538330	527589	64

Instruction	Vpr 1		Vpr16		field
	real time	CM time	real time	CM time	width
CM_f_write_to_processor_1L	422	88	404	81	64
CM_s_write_to_processor_1L	140	103	193	136	64
CM_u_write_to_processor_1L	140	114	64	64	64
CM_u_write_to_processor_1L	140	114	64	64	64