Appendix

Man Pages

This appendix contains the text of man pages for `cs` and for certain C* header files. These man pages are also available on-line.

cs

C* compiler for the CM-5 Connection Machine system

SYNOPSIS

cs [option] ... *file* ...

DESCRIPTION

cs invokes the CM-5 C* compiler. File names ending in **.cs** are treated as C* source files; file names ending in **.c** are treated as C source files and are passed directly to the *cc* compiler; file names ending in **.o** are treated as object files and are passed directly to the linker; and file names ending in **.a** are treated as object libraries and are passed directly to the linker.

The resulting executable program can be run on a CM-5 Connection Machine system. An option is also available to run the program on a Sun-4.

cs accepts a number of the options and filename endings that *cc* accepts, plus many specific to *cs*.

OPTIONS

Options specific to *cs*

–cc*cmdname* Use *cmdname*, rather than *cc*, as the compiler to perform C compilations.

–cm*X* Compile and link for CM model *X*. Accepted values of *X* are 2, 200, and 5. The values 2 and 200 invoke the CM-200 C* compiler (if it's installed); see the CM-200 *cs* man page for information about this compiler. There is a site-specific default, which would typically be 5 for a CM-5 site.

–cmdebug Compile for debugging. The program will run faster when compiled with **–cmdebug** than when compiled with **–g**, but debugging will be somewhat less precise.

–cmmd_root *dir* Use *dir* as the CMMD root directory, when compiling as a node-level program. The environment variable **CMD_ROOT** can also be used to specify this directory.

–cmprofile Produce performance analysis code. Performance data can then be generated and viewed using *prism*.

–cmsim Compile and link to run on a Sun-4.

–dirs Print a message showing where the compiler searches for binaries, libraries, include files, and temporary files.

–dryrun Show, but do not execute, all commands to be executed in compiling and linking.

–force Force input files with the **.c** suffix to be passed through the C* compiler rather than the *cc* compiler.

–help Print a summary of available command line switches without compiling.

–h Synonym for **–help**.

–keep *ext* Keep the intermediate file with the extension *ext*. Specify **s** to keep the assembly-language source code. Specify **o** to keep the object file. Specify **dp** to keep a DPEAC assembly source file (ending in **.pe.dp**) for code that runs on vector units. Using this option does not inhibit assembly or linking.

–node Compile and link to run as a node-level program.

–overload For each call to an overloaded function, print the actual symbol name of the function called. This is useful for debugging.

–q Suppress progress reports normally produced by the compiler.

–sparc Compile and link to run on a CM-5 without vector units. This option implies **–cm5**.

–temp *dir* Change the location in which C* temporary files are created from the standard temp directory to *dir*. See tmpnam(3).

–v Show the commands executed as compilation and linking proceeds.

–vecunit Compile and link to run on a CM-5 with vector units. This option implies **–cm5**.

–verbose	Synonym for **-v**.
–version	Print the C* compiler version number before compiling.
–vu	Synonym for **–vecunit**.
–warn	Suppress warnings from the C* compilation phase.
–w	Synonym for **–warn**.
–Wimplicit	Print warnings when you call a function that has not previously been declared or defined.
–Z*comp switch*	Pass option *switch* to *comp*, where *comp* is **cc**, **cmld**, **ld** (if the **–cmsim** option is specified), **dpas**, or **as**. For example, **–Zcc –O** turns on the C compiler's optimizer.

Options in common with *cc*

–c	Suppress the linking phase of the compilation and force an object file to be produced even if only one program is compiled.
–D*name*[=*def*]	Define the symbol *name* to the preprocessor. If =*def* is not supplied then *name* is defined with a value of 1.
–g	Have the compiler produce additional symbol table information for use specifically with *prism*. This slows execution considerably.
–I*dir*	Seek **#include** files whose names do not begin with "/" in this directory. The compiler looks for include files first in directories named in **–I** options, then in the include directory listed by the **–dirs** option, then in **/usr/include**.
–L*dir*	Add directory *dir* to the list of directories in the object library search path.
–l*x*	Look for library **lib***x***.a** in the library search path. The libraries specified via the **–L option** are searched first, followed by the library directory listed by the **–dirs** option, then libraries specified by the environment variable **LD_LIBRARY_PATH**, if set. Finally, *ld* tries the directories **/lib**, **/usr/lib**, and **/usr/local/lib**. Libraries are searched in the order in which their names are encountered, so the placement of a **–l** is significant.

–o *output*	Name the final output file *output*. This option can also be used to rename the output from the **–c** and **–S** options. It applies to parallel **.pe** files as well as scalar files; see **LINKING**, below, for more information on these files. If this option is used, the file **a.out** will be left undisturbed. If this option is not used, the output file is called **a.out**, unless the **–c** or **–S** option is also specified.
–S	Create assembler source files (ending in **.S**) as output.
–U*name*	Undefine the preprocessor symbol *name*.

PRE-DEFINED PREPROCESSOR SYMBOLS

The C* compiler provides the following default preprocessor symbols, each defined as **1**.

unix	Any UNIX system
sun	Sun only
sparc	Sun-4 only
__CM5__	CM-5 only
__CM5_SPARC__	CM-5 with SPARC processors only
__CM5_VECUNIT__	CM-5 with vector units
__CSTAR__	This is a C* compiler
__STDC__	This is an ANSI compiler

LINKING

The C* compiler and the CM-5 linker, *cmld* (or *ld* if you specify the **–cmsim** option), generate a single output file that combines a scalar executable program for the partition manager and a parallel executable program for the nodes. As intermediate output, however, the compiler generates separate files for the partition manager and the nodes. For example:

With the **–S** option, the compiler generates both parallel and scalar assembly files: for example, **myprogram.s** and **myprogram.pe.s**.

With the **–c** option, the compiler generates two object files: for example, **myprogram.o** and **myprogram.pe.o**.

However, the linker generates only one executable file: for example, **a.out**. There is no file **a.out.pe** corresponding to the parallel intermediate files.

If you work with intermediate files — explicitly linking object files, for instance — you need only specify the scalar file (for example, **myprogram.o**). The corresponding parallel file is linked automatically.

In any case, recall that the separate intermediate files exist. If you copy or move intermediate files to another directory, be sure to move both the scalar and the parallel files.

If you put object files in a library, you must remember to put the **.pe** object files in a library too. The library's name must begin with the same name as the library containing the corresponding scalar object files, and it must end in **_pe.a**. It must be in the same directory as the one that contains the corresponding scalar object files.

When linking, you then need only specify the library containing the scalar object files.

FILES

*file***.cs**	input C* code
*file***.o**, *file***.pe.o**	relocatable object files
*file***.s**, *file***.pe.s**	assembler source files
*file***.a**, *file***.pe.a**	object libraries
a.out	linked executable output
include_dir	directory of C* include files
*bin_dir***/cstar-compiler**	C* compiler executable
/bin/cc	C compiler
*lib_dir***/libcs_cm5_sparc_sp.a**,	
*lib_dir***/libcs_cmf_sparc_pn.a**,	
*lib_dir***/libcs_cm5_vu_sp.a**,	
*lib_dir***/libcs_cm5_vu_pn.a**,	
*lib_dir***/libcs_cm5_cmsim.a**	C* libraries linked by default

where *include_dir*, *bin_dir*, and *lib_dir* are directories listed by the **–dirs** option.

SEE ALSO

cc(1), cmld(1), prism(1)

C Programming Guide, C* Release Notes, CM-5 C* User's Guide, CM-5 C* Performance Guide*, and *Getting Started in C*.*

RESTRICTIONS

Bugs and restrictions are listed in the C* bug-update file, by default in **/usr/doc/cstar-***release***.bugupdate**.

Error messages are reported in a non-standard order. Rather than the messages appearing in the order in which the error was encountered, they are grouped together by file, and appear in the order in which the file was first seen. Thus, errors for included files can show up after errors in the source file.

cscomm.h

C* communication functions

SYNTAX

#include <cscomm.h>

SYNOPSIS

```
overload get, send, scan, global;

overload spread, copy_spread, multispread, copy_multispread,
    reduce, copy_reduce;

overload rank, read_from_position, write_to_position,
    make_multi_coord, make_send_address;

overload from_grid, from_grid_dim, to_grid, to_grid_dim;

overload from_torus, from_torus_dim, to_torus, to_torus_dim;

overload read_from_pvar, write_to_pvar;

type:current get(CMC_sendaddr_t:current send_address,
     type :void *sourcep, CMC_collision_mode_t collision_mode);

void get(void:current *destp, CMC_sendaddr_t:current
    *send_addressp, void:void *sourcep, CMC_collision_mode_t
     collision_mode,int length);

type:current send(type:void *destp, CMC_sendaddr_t:current
    send_address, type:current source, CMC_combiner_t combiner,
    bool:void *notifyp);

void:current *send (void:void *destp, CMC_sendaddr_t:current
    *send_addressp, void:current *sourcep, int length,
    bool:void *notifyp);

type:current scan(type:current source, int axis, CMC_combiner_t
    combiner, CMC_communication_direction_t direction,
    CMC_segment_mode_t smode, bool:current *sbitp,
    CMC_scan_inclusion_t inclusion);
```

type `global(`*type*`:current source, CMC_combiner_t combiner);`

type`:current spread(`*type*`:current source, int axis, CMC_combiner_t combiner);`

type`:current copy_spread(`*type*`:current *sourcep, int axis, int coordinate);`

type`:current multispread(`*type*`:current source, unsigned int axis_mask,CMC_combiner_t combiner);`

type`:current copy_multispread(`*type*`:current *sourcep, unsigned int axis_mask,CMC_multicoord_t multi_coord);`

`void reduce(`*type*`:current *destp,` *type*`:current source, int axis, CMC_combiner_t combiner, int to_coord);`

`void  copy_reduce(`*type*`:current *destp,` *type*`:current source, int axis,int to_coord, int from_coord);`

`unsigned int:current rank(`*type*`:current source, int axis, CMC_communication_direction_t direction, CMC_segment_mode_t smode, bool:current *sbitp);`

type `read_from_position(CMC_sendaddr_t send_address,` *type* `:void *sourcep);`

type `write_to_position(CMC_sendaddr_t send_address,` *type*`:void *destp, bool source);`

```
CMC_multicoord_t make_multi_coord(shape s, unsigned int
     axis_mask, CMC_sendaddr_t send_address);

CMC_multicoord_t make_multi_coord(shape s, unsigned int
    axis_mask,int axes[ ]);

CMC_multicoord_t make_multi_coord(shape s, unsigned int
    axis_mask,int axis, ...);

CMC_sendaddr_t:current make_send_address(shape s,
    int:current axis, ...);

CMC_sendaddr_t:current make_send_address(shape s,
    int:current axes[ ]);

CMC_sendaddr_t make_send_address(shape s, int axis, ...);

CMC_sendaddr_t make_send_address(shape s, int axes[ ]);
```

```
type:current from_grid(type:current *sourcep, type:current value,
    int distance, ...);

void from_grid(void:current *destp, void:current *sourcep,
    void:current *valuep, int length, int distance, ...);

type:current from_grid_dim(type:current *sourcep, type:current value,
    int axis, int distance);

void from_grid_dim(void:current *destp, void:current *sourcep,
    void:current *valuep, int length, int axis, int distance);

type to_grid(type:current *destp, type:current source, type:current
    *valuep, int distance, ...);

void to_grid(void:current *destp, void:current *sourcep,
    void:current *valuep, int length, int distance, ...);

void to_grid_dim(type:current *destp, type:current source,
    type:current *valuep, int axis, int distance);

void to_grid_dim(void:current *destp, void:current *sourcep,
    void:current *valuep, int length, int axis, int distance);

type:current from_torus(type:current *sourcep, int distance, ...);

void from_torus(void:current *destp, void:current *sourcep,
    int length, int distance,  ...);

type:current from_torus_dim(type:current *sourcep, int axis,
    int distance);

void from_torus_dim(void:current *destp, void:current *sourcep,
    int length, int axis, int distance);

void to_torus(type:current *destp, type:current source,
    int distance, ...);

void to_torus(void:current *destp, void:current *sourcep,
    int length,int distance, ...);

void to_torus_dim(type:current *destp, type:current source,
    int axis, int distance);

void to_torus_dim(void:current *destp, void:current *sourcep,
    int length, int axis, int distance);

void read_from_pvar(type *destp, type:current source);
```

```
type:current write_to_pvar(type *sourcep);

unsigned int:current enumerate(int axis,
    CMC_communication_direction_t direction,
    CMC_scan_inclusion_t inclusion, CMC_segment_mode_t smode,
    bool:current *sbitp);
```

DESCRIPTION

The C* communication functions, which duplicate and supplement communication features of the language, support grid communication, communication with computation, and general communication. Communication functions are overloaded to support arithmetic, aggregate, and void types.

In the function prototypes listed above, there exists a function definition for the following values of *type*: **bool, signed char**, **signed short int**, **unsigned short int**, **signed int**, **unsigned int**, **signed long int**, **unsigned long int**, **float, double, long double**, and **void**.

SEE ALSO

cs, *CM-5 C* Users' Guide*, *C* Programming Guide*

---

math.h

C* mathematical library

SYNTAX

#include <math.h>

SYNOPSIS

```
overload acos, asin, atan;

overload atan2;

overload cos, sin, tan;

overload cosh, sinh, tanh;

overload asinh, acosh, atanh;

overload exp, log, log10, logb;

overload pow, ceil, sqrt, fabs, floor;

overload copysign, drem, finite, scalb; float:current
     acos(float:current);

double:current acos(double:current);

float:current asin(float:current);

double:current asin(double:current);

float:current atan(float:current);

double:current atan(double:current);

float:current atan2(float:current f, float:current f2);

double:current atan2(double:current d, double:current d2);

float:current cos(float:current);

double:current cos(double:current);
```

```
float:current sin(float:current);

double:current sin(double:current);

float:current tan(float:current);

double:current tan(double:current);

float:current cosh(float:current);

double:current cosh(double:current);

float:current sinh(float:current);

double:current sinh(double:current);

float:current tanh(float:current);

double:current tanh(double:current);

float:current acosh(float:current);

double:current acosh(double:current);

float:current asinh(float:current);

double:current asinh(double:current);

float:current atanh(float:current);

double:current atanh(double:current);

float:current exp(float:current);

double:current exp(double:current);

float:current log(float:current);

double:current log(double:current);

float:current log10(float:current);

double:current log10(double:current);

float:current logb(float:current f);

double:current logb(double:current d);

float:current pow(float:current,float:current);

double:current pow(double:current,double:current);
```

```
float:current ceil(float:current);

double:current ceil(double:current);

float:current sqrt(float:current f);

double:current sqrt(double:current d);

float:current fabs(float:current);

double:current fabs(double:current);

float:current floor(float:current);

double:current floor(double:current);

float:current copysign(float:current f, float:current f2);

double:current copysign(double:current d, double:current d2);

float:current drem(float:current f, float:current f2);

double:current drem(double:current d, double:current d2);

int:current finite(float:current f);

int:current finite(double:current d);

float:current scalb(float:current f, int:current i);

double:current scalb(double:current d, int:current i);
```

DESCRIPTION

The mathematical library under C* contains the entire serial C mathematical library, along with parallel overloadings of many of the functions. In addition, only parallel versions of the following functions, which have *no* scalar overloadings, are provided: **acosh, asinh,** and **atanh.**

SEE ALSO

cs, *CM-5 C* User's Guide*, *C* Programming Guide*

RESTRICTIONS

Because the scalar and parallel versions of some routines are implemented using different algorithms, results of routines given the same numerical input may be slightly different in a serial context than in a parallel context.

stdarg.h

C* variable arguments

SYNTAX

#include <stdarg.h>

SYNOPSIS

```
void va_start(va_list ap, parmN) ;

type = va_arg(va_alist ap, type) ;

void va_end(va_list ap) ;
```

DESCRIPTION

The macros **va_start, va_arg,** and **va_end** can be used to write functions that can operate a variable number of arguments.

The **va_start** macro must be called to initialize **ap** before use by **va_arg** and **va_end.**

The **va_arg** macro expands to an expression that has the type and value of the next argument in the call. The value of **ap** is modified so that successive calls to **va_arg** will continue to read arguments in the call.

The **va_end** macro facilitates a normal return from a function that calls the macros **va_start** and **va_arg** to read a variable argument list.

EXAMPLE

```
#include <stdarg.h>
#define MAXARGS    32

void f(int n_params, ...)
{
   int i, array[32] ;
```

```
   va_list ap ;
   va_start(ap, n_params) ;
   for ( i = 0 ; i < MAXARGS; i++)
      array[i] = va_arg(ap, int) ;
   va_end(ap) ;
}
```

SEE ALSO

ANSI C Programming Language Standard, *C* Programming Guide*

stdlib.h

C* generic utilities

SYNTAX

#include <stdlib.h>

SYNOPSIS

```
int abs(int i);
int rand(void);
void srand(unsigned seed);
overload abs;
int:current abs(int:current i);
void psrand(unsigned seed);
int:current prand(void);
void deallocate_shape(shape *s);
void:void *palloc(shape s, int bsize);
void pfree(void:void *pvar);
```

DESCRIPTION

The C* generic utilities contain the parallel and scalar overloading of **abs.** The serial function is documented on the *abs* man page; the parallel function behaves exactly like the scalar function. Which **abs** function is called depends on whether a scalar or parallel integer is passed as the argument.

The function **psrand** reseeds the random number generator in all processors, even those that are not selected when the call occurs. Even though a scalar integer is passed to **psrand,** every processor will be seeded for a different sequence of random numbers. (Actually, it may be possible for two processors to have the same sequence, given a Connection Machine configuration with many virtual processors.)

The function **prand** is the parallel version of the **rand** function.

SEE ALSO

cs, abs(3), rand(3)

C Programming Guide*

LIMITATIONS

Seed values of **0** and **–1** are not accepted by **psrand.**

---

string.h

C* string handling functions

SYNTAX

#include <string.h>

SYNOPSIS

```
bool:current *boolcpy (bool:current *s1,
    bool:current *s2, size_t n);

bool:current *boolmove (bool:current *s1,bool:current *s2,
    size_t n);

int:current boolcmp (const bool:current *s1, bool:current *s,
    size_t n);

bool:current *boolset (bool:current *s, bool:current c,
    size_t n);

void:current *memcpy (void:current *s1, void:current *s2,
    size_t n);

void:current *memmove (void:current *s1, void:current *s2,
    size_t n);

int:current memcmp (const void:current *s1, void:current *s,
    size_t n);

void:current *memset (void:current *s, int:current c, size_t n);
```

DESCRIPTION

The string handling functions under C* contain the serial C string handling functions along with parallel overloadings of the functions.

SEE ALSO

ANSI C Programming Language Standard