Doc. no: N1862
Date: 2014-09-18
Reply to: Clark Nelson

Programming languages — C — Extensions for parallel programming

This is the latest working draft of the CPLEX study group, for the consideration of WG14.

Contents

Tables

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of technical committees is to prepare International Standards. Draft International Standards adopted by the technical committees are circulated to the member bodies for voting. Publication as an International Standard requires approval by at least 75% of the member bodies casting a vote.

In other circumstances, particularly when there is an urgent market requirement for such documents, a technical committee may decide to publish other types of normative document:

— an ISO Publicly Available Specification (ISO/PAS) represents an agreement between technical experts in an ISO working group and is accepted for publication if it is approved by more than 50% of the members of the parent committee casting a vote;

— an ISO Technical Specification (ISO/TS) represents an agreement between the members of a technical committee and is accepted for publication if it is approved by 2/3 of the members of the committee casting a vote.

An ISO/PAS or ISO/TS is reviewed every three years with a view to deciding whether it can be transformed into an International Standard.

ISO/TS *CPLEXTS* was prepared by Technical Committee ISO/IEC JTC1/SC22/WG14.[1)]

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights.

[1)] EN: This is the only paragraph in the Foreword that has anything in it that's not just boilerplate.

Introduction

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights other than those mentioned above. ISO [and/or] IEC shall not be held responsible for identifying any or all such patent rights.

Programming languages — C — Extensions for parallel programming

1 Scope

The following are within the scope of this technical specification:

— Extensions to the C language to simplify writing a parallel program.

The following are outside the scope of this technical specification:

— Support for writing a concurrent program.

2 Normative references

The following normative documents contain provisions which, through reference in this text, constitute provisions of this technical specification. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. However, parties to agreements based on this technical specification are encouraged to investigate the possibility of applying the most recent editions of the normative documents indicated below. For undated references, the latest edition of the normative document referred to applies. Members of ISO and IEC maintain registers of currently valid International Standards.

ISO/IEC 9899:2011(E), *Programming languages — C*

ISO/IEC 14882:2014(E)[2)], *Programming languages — C++*

[2)] To be published.

3 Terms and definitions

For the purposes of this document, the following terms and definitions apply.

3.1
thread of execution
flow of control within a program, including a top-level statement or expression, and recursively including every function invocation it executes[3)]

3.2
OS thread
service provided by an operating system for executing multiple threads of execution concurrently

NOTE 1 There is typically significant overhead involved in creating a new OS thread.

3.3
thread
thread of execution, or OS thread

NOTE 1 This word, when used without qualification, is ambiguous.

3.4
execution agent
entity, such as an OS thread, that may execute a thread of execution in parallel with other execution agents[4)]

3.5
task
thread of execution within a program that can be correctly executed asynchronously with respect to (certain) other parts of the program

3.6
concurrent program
program that uses multiple concurrent interacting threads of execution, each with its own progress requirements

EXAMPLE 1 A program that has separate server and client threads is a concurrent program.

3.7
parallel program
program whose computation is divided into tasks, which may be distributed across multiple computational units to be executed simultaneously

NOTE 1 If sufficient computational resources are available, a parallel program may execute significantly faster than an otherwise equivalent serial program.

[3)] EN: Adapted from the C++ standard.

[4)] EN: Adapted from the C++ standard.

4 Document conventions

1 *[C++: Text that is specific to C++ is enclosed in square brackets and presented in oblique sans-serif type.]*

2 Definitions of terms and grammar non-terminals defined in the C *[C++: or C++]* standard are not duplicated in this document. Terms and grammar non-terminals defined in this document are referenced in the index. The "cplex_" prefix of library identifiers is omitted from the index entry.

3 According to the ISO editing directives, the use of footnotes "shall be kept to a minimum." Almost all of the footnotes in this document are not intended to survive to final publication. Most footnotes are classified by an abbreviation:

EN: Editor's note. These mostly call attention to an area that needs more work.

DFEP: Departure from existing practice.

5 Counted loops

5.1 Introduction

1 A *counted loop* is a `for` statement *[C++: or range-based `for` statement]* that is required to satisfy additional constraints. The purpose of these constraints is to ensure that the loop's iteration count can be computed before the loop body is executed.

2 There shall be no `return`, `break`, `goto` or `switch` statement that might transfer control into or out of a counted loop.

5.2 Constraints on a counted `for` statement

5.2.1 Introduction

1 The syntax of a `for` statement includes three *control clauses* between parentheses, separated by semicolons. The first of these is called the initialization clause; the second is called the condition clause or controlling expression; the third is called the *loop-increment*.

2 When a constraint limits the form of an expression, parentheses are allowed around the expression or any required subexpression.

5.2.2 Constraints on the form of the control clauses

1 *[C++: The condition shall be an expression.*

NOTE 1 A condition with declaration form is useful in a context where a value carries more information than just whether it is zero or nonzero. This is not believed to be useful in a counted loop.

]

2 The controlling expression shall be a comparison expression with one of the following forms:[5)]

relational-expression `<` *shift-expression*
relational-expression `>` *shift-expression*
relational-expression `<=` *shift-expression*
relational-expression `>=` *shift-expression*
equality-expression `!=` *relational-expression*

3 Exactly one of the operands of the comparison operator shall be an identifier designating an induction variable, as described below. This induction variable is known as the *control variable*. The operand that is not the control variable is called the *limit expression*. *[C++: Any implicit conversion applied to that operand is not considered part of the limit expression.]*

4 The loop-increment shall be an expression with the following form:[6)]

loop-increment:
single-increment
loop-increment `,` *single-increment*

[5)] DFEP: OpenMP does not (yet) allow comparison with `!=`.

[6)] DFEP: OpenMP and "classic" Cilk allow only a single induction variable: the loop control variable. Allowing multiple induction variables is implemented in Intel's compiler.

single-increment:
identifier `++`
identifier `--`
`++` *identifier*
`--` *identifier*
identifier `+=` *initializer-clause*
identifier `-=` *initializer-clause*
identifier `=` *identifier* `+` *multiplicative-expression*
identifier `=` *identifier* `-` *multiplicative-expression*
identifier `=` *additive-expression* `+` *identifier*

5 *[C++: Each comma in the grammar of loop-increment shall represent a use of the built-in comma operator.]* The identifier in each grammatical alternative for single-increment names an *induction variable*. If *identifier* occurs twice in a grammatical alternative for *single-increment*, the same variable shall be named by both occurrences. If a grammatical alternative for *single-increment* contains a subexpression that is not an identifier for the induction variable, that is called the *stride expression* for that induction variable.

6 An induction variable shall not be designated by more than one *single-increment*.

NOTE 2 The control variable is identified by considering the loop's condition and loop-increment together. If exactly one operand of the condition comparison is a variable, it is the control variable, and must be incremented. If both operands of the condition comparison are variables, only one is allowed to be incremented; that one is the control variable. It is an error if neither operand of the condition comparison is a variable.

NOTE 3 There is no additional constraint on the form of the initialization clause of a counted `for` loop.[7)]

5.2.3 Other statically checkable constraints

1 Each induction variable shall have unqualified integer, *[C++: enumeration, copy-constructible class,]* or pointer type, and shall have automatic storage duration.

2 Each stride expression shall have integer *[C++: or enumeration]* type.

3 The *iteration count* is computed according to Table 1.[8)] The iteration count is computed after the loop initialization is performed, and before the control variable is modified by the loop. *[C++: The iteration count expression shall be well-formed.]*

4 The type of the difference between the limit expression and the control variable is the *subtraction type*, *[C++: which shall be integral. When the condition operation is !=, (limit)-(var) and (var)-(limit) shall have the same type.]* Each stride expression shall be convertible to the subtraction type. *[C++: The loop odr-uses whatever operator-functions are selected to compute these differences.]*

[C++:

5 *For each induction variable V, one of the expressions from Table 2 shall be well-formed, depending on the operator used in its single-increment. In the table, X stands for some expression with the same*

[7)] DFEP: OpenMP and "classic" Cilk require that the control variable be initialized. This relaxation is implemented in Intel's compiler.

[8)] EN: We need to say something about error cases.

Table 1 – Method of computing the iteration count

Form of condition	**Form of single-increment**			
	id* ++** **++ *id	***id* --** **-- *id***	***id* += *stride*** ***id* = *id* + *stride*** ***id* = *stride* + *id***	***id* -= *stride*** ***id* = *id* - *stride***
id* < *lim ***lim* > *id***	$((lim) - (id))$	ERROR	$((lim) - (id) - 1)/$ $(stride) + 1$	$((lim) - (id) - 1)/$ $(stride) + 1$
id* > *lim ***lim* < *id***	ERROR	$((id) - (lim))$	$((id) - (lim) - 1)/$ $-(stride) + 1$	$((id) - (lim) - 1)/$ $-(stride) + 1$
id* <= *lim ***lim* >= *id***	$((lim) - (id))$ $+1$	ERROR	$((lim) - (id))/$ $(stride) + 1$	$((lim) - (id))/$ $(stride) + 1$
id* >= *lim ***lim* <= *id***	ERROR	$((id) - (lim))$ $+1$	$((id) - (lim))/$ $-(stride) + 1$	$((id) - (lim))/$ $-(stride) + 1$
id* != *lim ***lim* != *id***	$((lim) - (id)$	$((id) - (lim))$	$((stride) < 0)$? $((id) - (lim) - 1)/$ $-(stride) + 1$: $((lim) - (id) - 1)/$ $(stride) + 1$	$((stride) < 0)$? $((lim) - (id) - 1)/$ $-(stride) + 1$: $((id) - (lim) - 1)/$ $(stride) + 1$

Legend:

Name	**In the form of an expression**	**In the iteration count expression**
id	The name of the control variable.	An expression with the type and value of the control variable.
lim	The limit expression.	An expression with the type and value of the limit expression.
stride	The stride expression.	An expression with the type and value of the stride expression for the control variable.

Table 2 – Method of advancing an induction variable

Single-increment operator	**Expression**
`++ += +`	V `+=` X
`-- -= -`	V `-=` X

type as the subtraction type. The loop odr-uses whatever `operator+=` and `operator-=` functions are selected by these expressions.]

5.2.4 Dynamic constraints

1 If an induction variable is modified within the loop other than as the side effect of its single-increment operation, the behavior of the program is undefined.

[C++: If evaluation of the iteration count, or a call to a required `operator+=` or `operator-=` function, terminates with an exception, the behavior of the program is undefined.]

2 If X and Y are values of the control variable that occur in consecutive evaluations of the loop condition in the serialization, then the behavior is undefined if $((limit) - X) - ((limit) - Y)$, evaluated in infinite integer precision, does not equal the stride.

NOTE 1 In other words, the control variable must obey the rules of normal arithmetic. Unsigned wraparound is not allowed.

3 If the condition expression is true on entry to the loop, then the behavior is undefined if the computed iteration count is not greater than zero. If the computed iteration count is not representable as a value of type `unsigned long long`, the behavior is undefined.

5.2.5 Evaluation relaxations

1 The stride expressions shall not be evaluated if the iteration count is zero; otherwise, the stride and limit expressions are evaluated exactly once.[9)]

2 Within each iteration of the loop body, the name of each induction variable refers to a local object, as if the name were declared as an object within the body of the loop, with automatic storage duration and with the type of the original object. *[C++: If the loop body throws an exception that is not caught within the same iteration of the loop, the behavior is undefined, unless otherwise specified.]*

[C++:

5.3 Constraints on a counted range-based `for` statement

1 *In a counted range-based `for` statement ([stmt.ranged] 6.5.4), the type of the `__begin` variable, as determined from the begin-expr, shall satisfy the requirements of a random access iterator.*

NOTE 1 Intel has not yet implemented support for a parallel range-based `for` statement.

]

[9)] DFEP: Neither OpenMP nor Cilk specifies how many times these expressions must be evaluated.

6 Parallel loops

1 A *parallel loop* is a `for` statement with loop qualifiers. The grammar of the iteration statement (6.8.5, paragraph 1) is modified to read:

iteration-statement:
`while (` *expression* `)` *statement*
`do` *statement* `while (` *expression* `) ;`
loop-qualifiers$_{opt}$ `for (` *expression*$_{opt}$ `;` *expression*$_{opt}$ `;` *expression*$_{opt}$ `)` *statement*
loop-qualifiers$_{opt}$ `for (` *declaration expression*$_{opt}$ `;` *expression*$_{opt}$ `)` *statement*

[C++: The grammar of iteration-statement (6.5 [stmt.iter], paragraph 1) is modified to read:

iteration-statement:
`while (` *expression* `)` *statement*
`do` *statement* `while (` *expression* `) ;`
loop-qualifiers$_{opt}$ `for (` *for-init-statement condition*$_{opt}$ `;` *expression*$_{opt}$ `)` *statement*
loop-qualifiers$_{opt}$ `for (` *for-range-declaration* `:` *for-range-initializer* `)` *statement*

]

2 The following rules are added to the grammar:

loop-qualifiers:
`_Task_parallel` *loop-parameters*$_{opt}$

loop-parameters:
`[` *expression* `]`

3 A parallel loop is a counted loop, and shall satisfy all the constraints of a counted loop.

4 In a parallel loop with the `_Task_parallel` loop qualifier, each iteration is a separate task, which is unsequenced with respect to all other iterations of that execution of the loop.

5 If loop parameters are specified as part of the loop qualifiers, the contained expression shall have type `cplex_loop_params_t`,[10)] as defined in header `<cplex.h>`.[11)]

6 The *serialization* of a parallel loop is obtained by deleting the loop qualifiers from the loop.

[10)] EN: This should probably be "pointer to `cplex_loop_params_t`", for consistency with the first argument of the set/get macros.

[11)] DFEP: This syntax for specifying tuning parameters for a loop is a CPLEX invention.

7 Spawning tasks

7.1 Task statements

1 The grammar of a statement (6.8, paragraph 1) *[C++: (clause 6, paragraph 1)]* is modified to add task-statement as a new alternative.

Syntax

task-statement:
task-block-statement
task-spawn-statement
task-sync-statement
task-call-statement

7.2 The task block statement

Syntax

task-block-statement:
`_Task_parallel _Block` *compound-statement*

Semantics

1 Defines a task block, within which tasks can be spawned. At the end of the contained compound statement, all child tasks spawned directly or indirectly within the compound statement are *joined*: the statement following the task block is not executed until all of the child tasks complete. [12) 13)]

2 For a given statement, the *associated task block* is defined as follows. For a statement within a task spawn statement, there is no associated task block, except within a nested task block statement or parallel loop. For a statement within a task block statement or parallel loop, the associated task block is the smallest enclosing task block statement or parallel loop. Otherwise, for a statement within the body of a function declared with the spawning function specifier, the associated task block is the same as it was at the point of the task spawning call statement that invoked the spawning function. For a statement in any other context, there is no associated task block.

NOTE 1 Task blocks can be nested lexically and/or dynamically. Determination of the associated task block is a hybrid process: lexically within a function, and dynamically across calls to spawning functions.[14)] Code designated for execution in another thread by means other than a task statement (e.g. using `thrd_create`) is not part of any task block.

[12)] EN: Is a branch allowed into or out of a task block?

[13)] EN: What about C++ EH?

[14)] DFEP: In Cilk, this determination can be done entirely lexically. In OpenMP, this determination can be done entirely dynamically.

7.3 The task spawn statement

Syntax

task-spawn-statement:
`_Task_parallel _Spawn` *compound-statement*

Constraints

1 A task spawn statement shall have an associated task block. [15)]

Semantics

2 Creates a child task of the associated task block of the task spawn statement, in which the contained compound statement is executed. The execution of the task is unsequenced with respect to the statements of the associated task block following the spawn statement. The completion of the task synchronizes with the completion of the associated task block, or with the next execution of a sync statement within the associated task block.

7.4 The task sync statement

Syntax

task-sync-statement:
`_Task_parallel _Sync ;`

Constraints

1 A task sync statement shall have an associated task block.

Semantics

2 Joins all child tasks of the associated task block of the task sync statement.

7.5 The task spawning call statement

Syntax

task-call-statement:
`_Task_parallel _Call` *expression-statement* [a]

[a] EN: Using the same keyword pair for the statement prefix and the function specifier introduces a case where three tokens of lookahead (and hopefully no more) are sometimes needed to disambiguate a declaration from a statement. Is this what we want?

Constraints

1 A task spawning call statement shall have an associated task block.

[15)] EN: Is a branch allowed into or out of a spawned statement?

Semantics

2 The contained expression statement is executed normally. Any called spawning function is allowed to spawn tasks; any such tasks are associated with the associated task block of the task spawning call statement, and are unsequenced with respect to the statements of the task block following the task spawning call statement.

7.6 The spawning function specifier

Syntax

1 A new alternative is added to the grammar of function specifier (6.7.4 paragraph 1):

function-specifier:
`_Task_parallel _Call`

Constraints

2 If a spawning function specifier appears on any declaration of a function, it shall appear on every declaration of that function. A function declared with a spawning function specifier shall be called only from a task spawning call statement. [16)]

[16)] EN: Is a spawning function specifier part of the function's type? If so, are function-pointer conversions allowed?

8 Parallel loop hint parameters <cplex.h>

8.1 Introduction

1 The header <cplex.h> defines several types and several macros.

2 The cplex_loop_params_t type is a structure type with an unspecified number of members for specifying parameters for tuning hints for a parallel loop. A program whose output depends on the value specified for any tuning hint parameter is not considered a correct program.

NOTE 1 There is no guarantee that setting any tuning hint parameter will improve the performance of the program.

3 The cplex_sched_kind_t type is an enumerated type with at least the following enumeration constants, each with nonzero value:

```
cplex_sched_static
cplex_sched_dynamic
cplex_sched_guided
```

4 The cplex_workload_t type is an enumerated type with at least the following enumeration constants, each with nonzero value:

```
cplex_workload_balanced
cplex_workload_unbalanced
```

5 The cplex_affinity_t type is an enumerated type with at least the following enumeration constants, each with nonzero value:

```
cplex_affinity_close
cplex_affinity_spread
```

6 When an object of type cplex_loop_params_t is used as the loop parameter of a parallel loop, the loop is described as being associated with the object.[17)] When executing a parallel loop associated with an object of type cplex_loop_params_t, for any parameter for which the corresponding member has the value zero, an unspecified default value is used.

7 Each parameter is represented by a pair of macros: one to set the value of the parameter in the parameter block, and one to get the value of the parameter from the parameter block.

NOTE 2 Because these methods are specified as macros, not functions, taking the address of any of them need not be supported. However, an implementation is also free to provide functions with these names.

EXAMPLE 1 Hint parameters for a parallel loop can be specified as follows:

```
#include <cplex.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
    cplex_loop_params_t hints = { 0 };
    if (argc > 1) {
```

[17)] EN: What if the object is modified during the execution of the loop?

```
        cplex_set_num_threads(&hints, atoi(argv[1]));
    }
    cplex_set_chunk_size(&hints, 1000);
    for _Task_parallel[hints] (long i = 0; i < 1000000; i++) {
        do_something_with(i);
    }
}
```

8.2 The `num_threads` parameter

Synopsis

```
#include <cplex.h>
void cplex_set_num_threads(cplex_loop_params_t *hints, int num_threads);
int cplex_get_num_threads(cplex_loop_params_t *hints);
```

Description

1 The `cplex_set_num_threads` macro sets to `num_threads` the recommended number of execution agents to be used to execute the iterations of a parallel loop associated with the object pointed to by `hints`.

8.3 The `chunk_size` parameter

Synopsis

```
#include <cplex.h>
void cplex_set_chunk_size(cplex_loop_params_t *hints, int chunk_size);
int cplex_get_chunk_size(cplex_loop_params_t *hints);
```

Description

1 The `cplex_set_chunk_size` macro sets to `chunk_size` the recommended maximum number of iterations of a parallel loop associated with the object pointed to by `hints` to be grouped together to be executed sequentially in a single thread of execution.

8.4 The `schedule_kind` parameter

Synopsis

```
#include <cplex.h>
void cplex_set_schedule_kind(cplex_loop_params_t *hints, cplex_sched_kind_t kind);
cplex_sched_kind_t cplex_get_schedule_kind(cplex_loop_params_t *hints);
```

Description

1 The `cplex_set_schedule_kind` macro sets to `kind` the recommended scheduling algorithm for a parallel loop associated with the object pointed to by `hints`.

NOTE 1 Setting the `schedule_kind` parameter to a particular value may (but need not) select the corresponding OpenMP loop-scheduling algorithm.

8.5 The `workload_balance` parameter

Synopsis

```
#include <cplex.h>
void cplex_set_workload_balance(cplex_loop_params_t *hints, cplex_workload_t kind);
cplex_workload_t cplex_get_workload_balance(cplex_loop_params_t *hints);
```

Description

1 The `cplex_set_workload_balance` macro sets to `kind` the workload-balancing characteristic for a parallel loop associated with the object pointed to by `hints`.

2 For a loop with a balanced workload, each iteration should be assumed to execute in approximately the same amount of time. A loop with an unbalanced workload should be assumed to have iterations taking widely varying amounts of time.

NOTE 1 This parameter is semantically a statement about the associated loop, whereas the `schedule_kind` parameter is semantically a request to the implementation. Setting this parameter to `cplex_workload_balanced` may have an effect similar to setting the schedule to `cplex_schedule_static`. Setting this parameter to `cplex_workload_unbalanced` may have an effect similar to setting the schedule to `cplex_schedule_dynamic` or `cplex_schedule_guided`.

8.6 The `affinity` parameter

Synopsis

```
#include <cplex.h>
void cplex_set_affinity(cplex_loop_params_t *hints, cplex_affinity_t kind);
cplex_affinity_t cplex_get_affinity(cplex_loop_params_t *hints);
```

Description

1 The `cplex_set_affinity` macro sets to `kind` the recommended affinity for a parallel loop associated with the object pointed to by `hints`.

2 The affinity of a loop indicates whether the loop benefits from being executed by co-located hardware threads, or whether performance is likely to improve if the software threads are spread over multiple cores.

Bibliography

[1] *Intel© Cilk™ Plus Language Extension Specification,* Intel Corporation: <https://www.cilkplus.org/sites/default/files/open_specifications/Intel_Cilk_plus_lang_spec_1.2.htm>

[2] *OpenMP Application Program Interface,* OpenMP Architecture Review Board: <http://www.openmp.org/mp-documents/OpenMP4.0.0.pdf>

Index