On Parallel Invocations of Functions in Parallelism TS | N4063

Artur Laksberg

2014-16-20

Abstract

This document proposes a change to N3989 (Technical Specification for C++ Extensions for Parallelism).

Introduction

At the Issaquah meeting in February 2014, Hans Boehm raised the following issue regarding N3850 (Working Draft, Technical Specification for C++ Extensions for Parallelism):

> It seems to me that the execution policies need to be a bit more precise about which calls to parameter functions they can make. Something needs to specify that sort may make parallel invocations to swap, but only if the arguments for concurrent calls don't overlap. I didn't quickly find such text. In general, "non-racing" calls can be made concurrently, but "racing" ones cannot.

Upon further discussion, it was decided that the resolution of this issue warrants a separate paper that can be presented for discussion in SG1 during the Rapperswil meeting.

Revision History

This paper supersedes N3993. The discussion in Rapperswil illuminated the need for additional changes to describe the difference between read-only and mutating functons. For example, we need to specify that given the comparison predicate `comp`, `std::sort` with `par` execution policy may invoke `comp(a,b)` and `comp(b,c)` concurrently, but not `swap(a,b)` and `swap(b,c)`. The changes from N3993 are in magenta and mulberry.

Proposed Resolution

~~First, we introduce changes to 17.6.5.9 [res.on.data.races]/8 that state that an implementation cannot introduce a data race on objects accessed during the execution of the algorithms.~~

First, instead of talking about "applying user-defined function objects", we define an umbrella term "element access functions", which includes the functions required by the specification, as well as the user-provided function objects. We then define the behavior in terms of "invoking the element access functions".

Additionally, we state that `BinaryPredicate`, `Compare` and `BinaryOperation` are special kinds of operations that are not allowed to modify its arguments, and are therefore safe to call on the same objects concurrently.

Specification Changes

~~The wording changes proposed in this section are relative to the contents of N3936.~~

~~In 17.6.5.9, change paragraph 8 as follows:~~

~~A C++ standard library function with execution policy `vec` or `par` shall not introduce a data race on any object accessible by the means described in paragraphs 2-5. Otherwise, specified, C++ standard library functions shall perform all operations solely within the current thread if those operations have effects that are visible (1.10) to users.~~

Wording Changes Relative to N3989

The wording changes proposed in this section are relative to the contents of N3989.

Add paragraph 3 to 1.3.1:

Parallel algorithms access objects indirectly accessible via their arguments by invoking the following functions:

- All operations of the categories of the iterators that the algorithm is instantiated with.
- Functions on those sequence elements that are required by its specification.
- User-provided function objects to be applied during the execution of the algorithm, if required by the specification.

These functions are herein called the *element access functions*.

[*Example:* The `sort` function may invoke the following element access functions:

- Methods of the random-access iterator of the actual template argument, as per 24.2.7, as implied by the name of the template parameters `RandomAccessIterator`.
- The `swap` function on the elements of the sequence (as per 25.4.1.1 [sort]/2)
- The user-provided `Compare` function object.
  – end example]

Change 2.1 as follows:

~~This subclause describes classes that represent *execution policies*. An *execution policy* is an object that expresses the requirements on the ordering of functions invoked as a consequence of the invocation of a standard algorithm. Execution policies afford standard algorithms the discretion to execute in parallel.~~

This clause describes classes that are *execution policy* types. An object of an execution policy type indicates to an algorithm whether it is allowed to execute in parallel and expresses the requirements on the element access functions.

Change 3.1 paragraph 2:

During the execution of a standard parallel algorithm, if the ~~application of a function object~~ invocation of an element access function terminates with an uncaught exception, the behavior of the program is determined by the type of execution policy used to invoke the algorithm:

...

All uncaught exceptions thrown during the ~~application of user-provided function objects~~ invocations of element access functions shall be contained in the `exception_list`.

Change 4.1.1 as follows, starting from paragraph 1:

Parallel algorithms have template parameters named `ExecutionPolicy` which describe the manner in which the execution of these algorithms may be parallelized and the manner in which they apply ~~user-provided function objects~~ the element access functions.

The ~~applications of function objects~~ invocations of element access functions in parallel algorithms invoked with an execution policy object of type `sequential_execution_policy` execute in sequential order in the calling thread.

The ~~applications of function objects~~ invocations of element access functions in parallel algorithms invoked with an execution policy object of type `parallel_execution_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

[*Note:* It is the caller's responsibility to ensure correctness, for example that the invocation does not introduce data races or deadlocks. — *end note*]

Change paragraph 4 as follows:

The ~~applications of function objects~~ invocations of element access functions in parallel algorithms invoked with an execution policy of type `vector_execution_policy` are permitted to execute in an unordered fashion in unspecified threads, and unsequenced within each thread. [*Note:* as a consequence, function objects governed by the `vector_execution_policy` policy must not synchronize with each other. Specifically, they must not acquire locks. — *end note*]

Add paragraph 4.1.1, as follows:

4.1.1 Requirements on user-provided function objects

Function objects passed into parallel algorithms as objects of type `BinaryPredicate`, `Compare` and `BinaryOperation` shall not directly or indirectly modify objects via their arguments.