**Doc No:** P0335R0

**Date:** 2016-05-28

**Audience:** SG1 (parallelism & concurrency)

**Authors:** Pablo Halpern, Intel Corp.  
 [phalpern@halpernwightsoftware.com](mailto:phalpern@halpernwightsoftware.com)

Context Tokens for Parallel Algorithms

**Contents**

1 Abstract 1

2 Motivation 1

3 Proposal overview 2

3.1 Variations 2

4 Alternatives considered 3

5 Future directions 3

6 Proposed Wording 3

6.1 Document Conventions 3

6.2 Define context\_token nested type in each execution policie 3

6.3 Specify optional context token to element access functions 3

6.4 Move vec\_off and ordered\_update into vector context token 4

7 References 5

8 Acknowledgements 5

# Abstract

This paper proposes a mechanism whereby the *element-access function* of a parallel algorithm (i.e., the function or lambda that is invoked in parallel) can invoke specific operations provided by the execution policy. The proposed mechanism takes the form of a *cookie* or *token* passed from the algorithm to the function. The type of this token can be different for each execution policy. Acceptance and use of this token is entirely optional, allowing simple use cases to remain simple and backwards-compatible with the existing parallel algorithms library. This model of using tokens is similar to the *execution agents* in the [Agency](https://github.com/jaredhoberock/agency/wiki/Quick-Start-Guide) library.

# Motivation

Consider the following use of the parallel for\_loop algorithm, as described in [P0075](http://www.open-std.org/JTC1/SC22/WG21/docs/papers/2016/p0075r2.pdf) with the vector\_execution\_policy described in [P0076](http://www.open-std.org/JTC1/SC22/WG21/docs/papers/2016/p0076r2.pdf):

parallel::for\_loop(parallel::vec, 0, N, [&](int i){

++parallel::ordered\_update(histagram[A[i]]);

});

The ordered\_update function is specifically tied to the vec execution policy, yet there is no syntactic connection between them. Replacing vec with par would render this code incorrect – undefined behavior. It would be better that such a substitution render the code ill-formed, but the library syntax does not give us a good way to express such a *syntactic* restriction.

Although the vector\_execution\_policy is the first demonstrated example of this kind of problem, the problem will not remain limited to the for\_loop, nor to the vector\_execution\_policy. For example, a future enhancement of the parallel execution policy might provide support for critical sections or thread-local storage in a way that is safer and/or more efficient than the direct use of mutex and thread\_local. As we move towards combining execution policies with executors, it might also be desirable to query the executor.

# Proposal overview

This proposal is targeted for version 2 of the parallelism TS.

What is proposed is that each parallel algorithm may pass a special token that communicates execution context to the element-access functions passed to the algorithm by the user. Policy-specific operations and queries, rather than being free functions, would be member functions of the token object. The above example would be rewritten as follows:

parallel::for\_loop(parallel::vec, 0, N, [&](auto context, int i){

++context.ordered\_update(histagram[A[i]]);

});

The token can carry information about the execution policy, the iteration of the loop, etc.. The type of this context token is defined as a nested type within the execution policy:

struct vector\_execution\_policy {

struct context\_token { ... };

...

};

If the token is not needed, it can simply be omitted from the argument list for the element-access function:

parallel::for\_each(parallel::vec, 0, N, [&](int i){

A[i] = A[i + 1] + 10;

});

Thus, existing uses of the parallel algorithms are unaffected by the context token. This flexibility is enabled through the use of metaprogramming to invoke the element access function either with or without extra initial argument. If the invocable object has overloads both with or without the extra argument, the overload with the extra argument is preferred.

## Variations

The formal wording, below, makes the context token available in any parallel algorithm that has ExecutionPolicy and Function template arguments. This includes for\_each and for\_each\_n in the C++17 WD and for\_loop, proposed in [P0075](http://www.open-std.org/JTC1/SC22/WG21/docs/papers/2016/p0073r1.pdf) for the next parallelism TS. There is a question as to whether this is the correct set of algorithms. Michael Garland suggests that only for\_loop, being a low-level function, should provide the context token. Conversely, the context token could be potentially useful in other algorithms such as transform and reduce. Making it available there would require expanding the criteria for what kind of element-access functions can receive a context token. The current proposal works with for\_each, which is similar to a range-based for\_loop. It might make sense, however, to leave for\_each alone and create a new, low-level variant of for\_loop that works with iterators.

# Alternatives considered

I briefly considered having the parallel context token be the same type as the execution policy. However, the token can encapsulate more than the execution policy; it can contains enough context about the specific iteration to, for example, provide a type of thread-local storage, or a worker index, etc.. It may not be possible to provide functionality such as ordered\_update from the execution policy alone. It might make sense, however, for all context tokens to provide a method that returns a reference to the execution policy. Such a method is not being proposed here because there is no identified use case.

# Future directions

The token proposed herein is similar to the task\_block argument passed to the invocable object in define\_task\_block. We should consider whether the concepts can and should be unified in some way.

# Proposed Wording

## Document Conventions

All section names and numbers are relative to [N4578](http://www.open-std.org/JTC1/SC22/WG21/docs/papers/2016/n4578.html) Working Draft of the Parallelism TS, Version 2, as modified by [P0076r2](http://www.open-std.org/JTC1/SC22/WG21/docs/papers/2016/p0076r2.pdf). Note that much of the Parallelism TS has been voted into the C++17 working draft and will be removed from the TS. These changes would need to be reflected in the new Parallelism TS, possibly as deltas from the C++17 working draft.

Existing working paper text is indented and shown in dark blue. Edits to the working paper are shown with red strikeouts for deleted text and green underlining for inserted text within the indented blue original text.

Comments and rationale mixed in with the proposed wording appears as shaded text.

Requests for LWG opinions and guidance appear with light (yellow) shading. It is expected that changes resulting from such guidance will be minor and will not delay acceptance of this proposal in the same meeting at which it is presented.

## Define context\_token nested type in each execution policie

Modify the first paragraph of section 2.1 [parallel.execpol.general], as follows:

Every execution policy, shall have a nested type named context\_token, which shall meet the requirements of CopyConstructible. Objects of context\_token type are created by the implementation and need not provide any other public constructors.

## Specify optional context token to element access functions

Modify the first paragraph of section 4.1.2 [parallel.alg.general.exec] as follows:

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 the element access functions. If a parallel algorithm’s template parameter is named Function, the actual template argument is an element access function that is invoked zero or more times with an argument list specified by the algorithm, preceded by an argument of type ExecutionPolicy::context\_token. If the Function cannot be invoked with an initial context token argument, then the context token is omitted from the argument list. The context token, if present, provides operations that can be accessed by the element access function. Those operations differ among execution policies. [*Note:* if a function can be invoked either with or without a context token, the overload with the context token is preferred. – *end note*]

As specified above, only a handful of algorithms, such as for\_each and for\_each\_n can take advantage of the context token. Other element access functions, such as predicate functors and calls to swap are not invoked with a context token. Is this the right place to draw the line? See section 3.1 for more discussion.

## Move vec\_off and ordered\_update into vector context token

Modify section [parallel.execpol.vec] from P0076 as follows:

class vector\_execution\_policy{ *unspecified*

public:

class context\_token {

public:

template<typename F>

auto vec\_off(F&& f) -> decltype(f());

template<class T>

class ordered\_update\_t;

template <class T>

ordered\_update\_t<T> ordered\_update(T& ref);

};

};

And remove the namespace-scope versions of vec\_off, ordered\_update\_t, and ordered\_update from section 4.3 [parallel.alg.ops].

Move the contents of sections [parallel.alg.vecoff], [parallel.alg.ordupdate.class] and [parallel.alg.ordupdate.func] from P0076 into a new section entitled “vector\_execution\_policy::context\_token members”, making the following small changes:

template<typename F>

auto vec\_off(F&& f) -> decltype(f());

*Effects*: Evaluates std::forward<F>(f)(). If For two calls to vec\_off that are horizontally matched within a wavefront application of an element access function over input sequence S, then the evaluation of f() in the application for one element in S is sequenced before the evaluation f() in the application for a subsequent element in S; otherwise (for other execution policies) there is no effect on sequencing.

Editorial note: If P0076 is applied to the TS as currently written, the above description will precede the definition of *wavefront application*. Either some reordering or a forward reference is needed.

# References

[Agency](https://github.com/jaredhoberock/agency/wiki/Quick-Start-Guide) *Quick Start Guide*, a low-level library for abstracting parallel execution, Jared Hoberock, 2015-12-03

[P0076r2](http://www.open-std.org/JTC1/SC22/WG21/docs/papers/2016/p0076r2.pdf) *Vector and Wavefront Execution Policies*, Arch Robison, Pablo Halpern, Robert Geva, Clark Nelson, Jens Maurer, 2016-05

[P0075r1](http://www.open-std.org/JTC1/SC22/WG21/docs/papers/2016/p0073r1.pdf) *Template Library for Index-Based Loops*, Arch Robison, Pablo Halpern, Robert Geva, Clark Nelson, 2016-02-12

# Acknowledgements

Thanks to Hans Boehm, Michael Garland, and Lawrence Crowl for their helpful review comments.