AI22-0069-1

!standard 4.5.10(21/5) 23-08-25 AI22-0069-1/04
!standard 4.5.10(27/5)
!standard 4.5.10(28/5)
!class binding interpretation 24-06-13 (WG 9 Resolution 87-5)
!status Corrigendum 1-2022 23-06-27
!status WG9 Approved 23-10-12
!status ARG Approved 6-0-0 23-06-11
!status work item 23-03-23
!status received 23-03-23
!priority Low
!difficulty Easy
!subject Empty subsequences in parallel reduction expressions

!summary

Empty chunks (subsequences) do not participate in parallel reduction expressions.

!issue

4.5.10(21/5) says, in the parallel case, that "the sequence of values is generated by a parallel iteration (as defined in 5.5, 5.5.1, and 5.5.2), as a set of non-empty, non-overlapping contiguous chunks (subsequences)".

However, nothing in the referenced subclauses prevents empty chunks. Specifically, 5.5.1(22/5) lets the First operation of a chunk return a cursor for which Has_Element is False, giving an empty subsequence. The wording in 5.5.2(10.3/5) expects empty subsequences.

In addition, empty subsequences can be produced in the presence of an iterator_filter, presuming that the overall sequence might be split up into subsequences *before* applying the filter, so that the filter computation can be performed in parallel.

But 4.5.10 does not say how to reduce an empty subsequence; 4.5.10(27/5) assumes that each subsequence has a "first value". We need to explain how empty subsequences work.

!recommendation

(See Summary.)

!wording

Modify 4.5.10(21/5):

If the value_sequence does not have the reserved word **parallel**, it is produced as a single sequence of values by a single logical thread of control. If the reserved word **parallel** is present in the value_sequence, the enclosing reduction_attribute_reference is a parallel construct, and the sequence of values is generated by a parallel iteration (as defined in 5.5, 5.5.1, and 5.5.2), as a set of [non-empty,] non-overlapping contiguous chunks (*subsequences*) with one logical thread of control (see Clause 9) associated with each subsequence. If there is a chunk_specification, it determines the maximum number of chunks, as defined in 5.5; otherwise the maximum number of chunks is implementation defined.

Modify 4.5.10(27/5):

Each logical thread of control creates a local accumulator for processing its subsequence. The accumulator for a subsequence is initialized to the first value {conditionally produced for} [of] the subsequence{, if any}, and calls on Reducer start with the second value of the subsequence (if any). The result for the subsequence{, if non-empty,} is the final value of its local accumulator.

Modify 4.5.10(28/5):

After all logical threads of control of a parallel reduction expression have completed, Reducer is called for each {non-empty} subsequence{, if any}, in the original sequence order, passing the local accumulator for that subsequence as the second (Value) parameter, and the overall accumulator [Redundant: (initialized above to the initial value)] as the first (Accumulator) parameter, with the result saved back in the overall accumulator. The parallel reduction expression yields the final value of the overall accumulator.

!discussion

It should be clear that it would be impractical to require user-written code to never return any empty subsequences. Moreover, if the entire set of values is empty, it's hard to see any alternative to returning an empty subsequence. So we have to assume that they could happen.

Filters, in particular, can result in empty subsequences, because we presumably don't actually evaluate a filter until *after* splitting a parallel iteration into subsequences, and then it is quite possible that no values pass the condition of the filter. The semantics of parallel reduction are defined as they are to allow the "initial value" to not be an identity of the reduction operator, and that seems important to preserve. So we don't want to fall back to the initial value for the subsequences.

So for efficiency in the presence of a filter, you might want to have two loops, the first would be to find the first element that passes the filter, to use to initialize the accumulator, and then the second loop would combine any further elements that pass the filter into that accumulator,

calling the Reduce operation with each one. If the first loop ends without finding a value that satisfies the filter condition, then that chunk needs to be omitted from the final reduction pass.

The author of the Github issue suggested that the fix would be for 4.5.10(27/5) to say that if a subsequence is empty, the corresponding logical thread completes without further actions, and 4.5.10(28/5) should apply Reducer only to the non-empty subsequences. We have followed that suggestion above.

!corrigendum 4.5.10(21/5)

@drepl
If the @fa{value_sequence} does not have the reserved word @b{parallel}, it is produced as a single sequence of values by a single logical thread of control. If the reserved word @b{parallel} is present in the @fa{value_sequence}, the enclosing @fa{reduction_attribute_reference} is a parallel construct, and the sequence of values is generated by a parallel iteration (as defined in 5.5, 5.5.1, and 5.5.2), as a set of non-empty, non-overlapping contiguous chunks (@i{subsequences}) with one logical thread of control (see Clause 9) associated with each subsequence. If there is a @fa{chunk_specification}, it determines the maximum number of chunks, as defined in 5.5; otherwise the maximum number of chunks is implementation defined.
@dby
If the @fa{value_sequence} does not have the reserved word @b{parallel}, it is produced as a single sequence of values by a single logical thread of control. If the reserved word @b{parallel} is present in the @fa{value_sequence}, the enclosing @fa{reduction_attribute_reference} is a parallel construct, and the sequence of values is generated by a parallel iteration (as defined in 5.5, 5.5.1, and 5.5.2), as a set of non-overlapping contiguous chunks (@i{subsequences}) with one logical thread of control (see Clause 9) associated with each subsequence. If there is a @fa{chunk_specification}, it determines the maximum number of chunks, as defined in 5.5; otherwise the maximum number of chunks is implementation defined.

!corrigendum 4.5.10(27/5)

@drepl
@xindent{Each logical thread of control creates a local accumulator for processing its subsequence. The accumulator for a subsequence is initialized to the first value of the subsequence, and calls on Reducer start with the second value of the subsequence (if any). The result for the subsequence is the final value of its local accumulator.}
@dby
@xindent{Each logical thread of control creates a local accumulator for processing its subsequence. The accumulator for a subsequence is initialized to the first value conditionally produced for the subsequence, if any, and calls on Reducer start with the second value of the subsequence (if any). The result for the subsequence, if non-empty, is the final value of its local accumulator.}

!corrigendum 4.5.10(28/5)

@drepl
@xindent{After all logical threads of control of a parallel reduction expression have completed, Reducer is called for each subsequence, in the original sequence order, passing the local accumulator for that subsequence as the second (Value) parameter, and the overall accumulator [(initialized above to the initial value)] as the first (Accumulator) parameter, with the result saved back in the overall accumulator. The parallel reduction expression yields the final value of the overall accumulator.}
@dby
@xindent{After all logical threads of control of a parallel reduction expression have completed, Reducer is called for each non-empty subsequence, if any, in the original sequence order, passing the local accumulator for that subsequence as the second (Value) parameter, and the overall accumulator (initialized above to the initial value) as the first (Accumulator) parameter, with the result saved back in the overall accumulator. The parallel reduction expression yields the final value of the overall accumulator.}

!ACATS test

An ACATS C-Test should be constructed to check that empty chunks do not participate in reductions. That could be accomplished by defining a filter that excludes most of the elements, such that one or more of the subsequences end up empty, after applying the filter.

!appendix

This AI was created from GitHub issue #36 (https://github.com/Ada-Rapporteur-Group/User-Community-Input/issues/36).