Parallel STL backend API reference.

Overview

Privided by the standard a parallel execution policy (*std::execution::par, std::execution::par\_unseq*) permits parallel execution of a STL algorithm. To achive it the parallel version of the algorithm, implemented by Parallel STL, uses one of general parallel basic patterns provided by a *Parallel backend*. In other words, a parallel backend specifies a set of function with certain signatures to express the all parallel patterns of Parallel STL.

Parallel backend function name

*pstl::par\_backend::parallel\_for*

Summary

Performs parallel execution of a functor over a range of data.

Syntax

template<class ExecutionPolicy, class Index, class F>
void parallel\_for(ExecutionPolicy&& exec, Index first, Index last, F f);

Notes

exec – an execution policy, may be usefull for some special back-ends (will be avaliable with the GitHub update).

Evaluation of a functor f[i,j) for each subrange [i,j) of [first,last).

f – a function object. The signature of it should be equivalent to the following:

 void f(Index begin, Index end);

Parallel backend function name

*pstl::par\_backend::parallel\_reduce*

Summary

Performs parallel execution of a functor over subranges and reduction the results.

Syntax

template<class ExecutionPolicy, class T, class  Index, class  Body, class  Reduction>
T parallel\_reduce(ExecutionPolicy&& exec, Index first, Index last, const T& identity, const Body& body, const Reduction& reduction);

Notes

exec – an execution policy, may be usefull for some special back-ends.

[first,last) - a range of data to be processed.

identity – an identical element.

body – a function object. The signature of it should be equivalent to the following:

 T body(Index begin, Index end, const T& value);

reduction – a function object. The signature of it should be equivalent to the following:

 T reduction (const T& a , const T& b);

Parallel backend function name

*pstl::par\_backend::parallel\_transform\_reduce*

Summary

Performs parallel execution of am unary operation and reduction over subranges and combines the results.

Syntax

template<class ExecutionPolicy, class  Index, class  U, class  T, class  C, class  R>
T parallel\_transform\_reduce(ExecutionPolicy&& exec, Index first, Index last, U unary, T initial, C combine, R brick\_reduce);

Notes

exec – an execution policy, may be usefull for some special back-ends.

[first,last) - a range of data.

unary(i) – unary operation.

initial – initial value.

combine(x,y) combines values x and y that were the result of brick\_reduce or unary.

brick\_reduce (i,j,init) returns reduction of init with reduction over [i,j).

Parallel backend function name

*pstl::par\_backend::* *parallel\_transform\_scan*

Summary

Performs parallel execution of am unary operation and scan over subranges and combines the results.

Syntax

template<class ExecutionPolicy, class  Index, class  U,  class T, class C, class R, class S>
T parallel\_transform\_scan(ExecutionPolicy&& exec, Index n, U unary, T initial, C combine, R brick\_reduce, S brick\_scan);

Notes

exec – an execution policy, may be usefull for some special back-ends.

n – number of elements to be processed.

unary – unary operation.

initial – initial value.

combine(x,y) combines values x and y that were the result of brick\_reduce or unary.

brick\_reduce (i,j,init) returns reduction of init with reduction over a subrange [i,j).

brick\_scan – perfoms a scan operation over a subrange [i,j).

Parallel backend function name

*pstl::par\_backend::parallel\_strict\_scan*

Summary

Performs parallel execution of a strict scan operation over a range of data.

Syntax

template<class ExecutionPolicy, class  Index, class T, class R, class C, class S, class A>
T parallel\_strict\_scan(ExecutionPolicy&& exec, Index n, T initial, R reduce, C combine, S scan, A apex);

Notes

exec – an execution policy, may be usefull for some special back-ends.

n – number of elements to be processed.

initial - initial value.

reduce(i,len) -> s - returns reduction value of i:len.

combine(s1,s2) -> s - returns merged sum

scan(i,len,initial) - performs scan over i:len starting with initial.

apex(s) - does any processing necessary between reduce and scan.

Parallel backend function name

*pstl::par\_backend::parallel\_stable\_sort*

Summary

Performs parallel execution of a stable/unstable sort algorithm.

Syntax

template<class ExecutionPolicy, class  RandomIt, class Compare, class LeafSort>
void parallel\_stable\_sort(ExecutionPolicy&& exec, RandomIt xs, RandomIt xe, Compare comp, LeafSort leaf\_sort, std::size\_t nsort);

Notes

exec – an execution policy, may be usefull for some special back-ends.

[xs,xe) - a range of data to be sorted.

сomp - comparison function object (i.e. an object that satisfies the requirements of Compare) which returns ​true if the first argument is less than (i.e. is ordered before) the second. The signature of the comparison function should be equivalent to the following:

 void comp(constType1 &a, const Type2 &b);

leaf\_sort - performs serial execution of a stable/unstable sort algorithm.

nsort – number of the first elements to be sorted. In case of zero the whole range will be sorted.

Parallel backend function name

*pstl::par\_backend::parallel\_merge*

Summary

Performs parallel execution of merging two sorted ranges.

Syntax

template<class ExecutionPolicy, class  RandomIt1, class  RandomIt2, class  RandomIt3, class Compare, class LeafSort>
void parallel\_merge(ExecutionPolicy&& exec, RandomIt1 xs, RandomIt1 xe, RandomIt2 ys, RandomIt2ys ye, RandomIt3 zs, Compare comp, LeafMerge leaf\_merge);

Notes

exec – an execution policy, may be usefull for some special back-ends.

[xs,xe), [ys,ye) - sorted ranges of data to be merged.

zs – output the merged ranges to be stored.

сomp - comparison function object (i.e. an object that satisfies the requirements of Compare) which returns ​true if the first argument is less than (i.e. is ordered before) the second. The signature of the comparison function should be equivalent to the following:

 void comp(constType1 &a, const Type2 &b);

leaf\_sort - performs serial merging of two sorted ranges of data.