



# Writing C++ Standard Conforming Parallel Algorithms in HPX

Grant Mercer, Hartmut Kaiser

Louisiana State University, Center for Computation and Technology



## Abstract

This project focuses on implementing standard conforming parallel algorithms in HPX. The requirements and specification are defined by open-std standards proposal N4071, a technical specification for C++ extensions for parallelism. HPX is a general purpose C++ runtime system for parallel and distributed applications, and this proposal can benefit HPX greatly with asynchronous algorithms. The new algorithms will introduce a new argument, known as an *execution policy*. This execution policy object will specify whether the algorithm will run asynchronously or synchronously. All other algorithm requirements are defined by its predecessor.

## A New Approach

According to standard proposal N4071, an execution policy object can be either sequential, parallel, or vectorized; represented in C++ as *seq*, *par* and *vec*. Our HPX implementation however offers one additional policy, *task*. A task execution policy will return a *hpx::future* of the result, giving the programmer the ability to choose when and where to synchronize with their main thread. A *task* parallel execution can be written as:

```
using hpx::parallel;
//
std::for_each(v.begin(), v.end(), func);

//execute for_each in parallel
hpx::future<void> r = for_each(task,
v.begin(), v.end(), func);

//do work

//now synchronize
r.wait();
```

Figure 1: Scaling of for\_each Overlapping Task Execution  
16 Cores

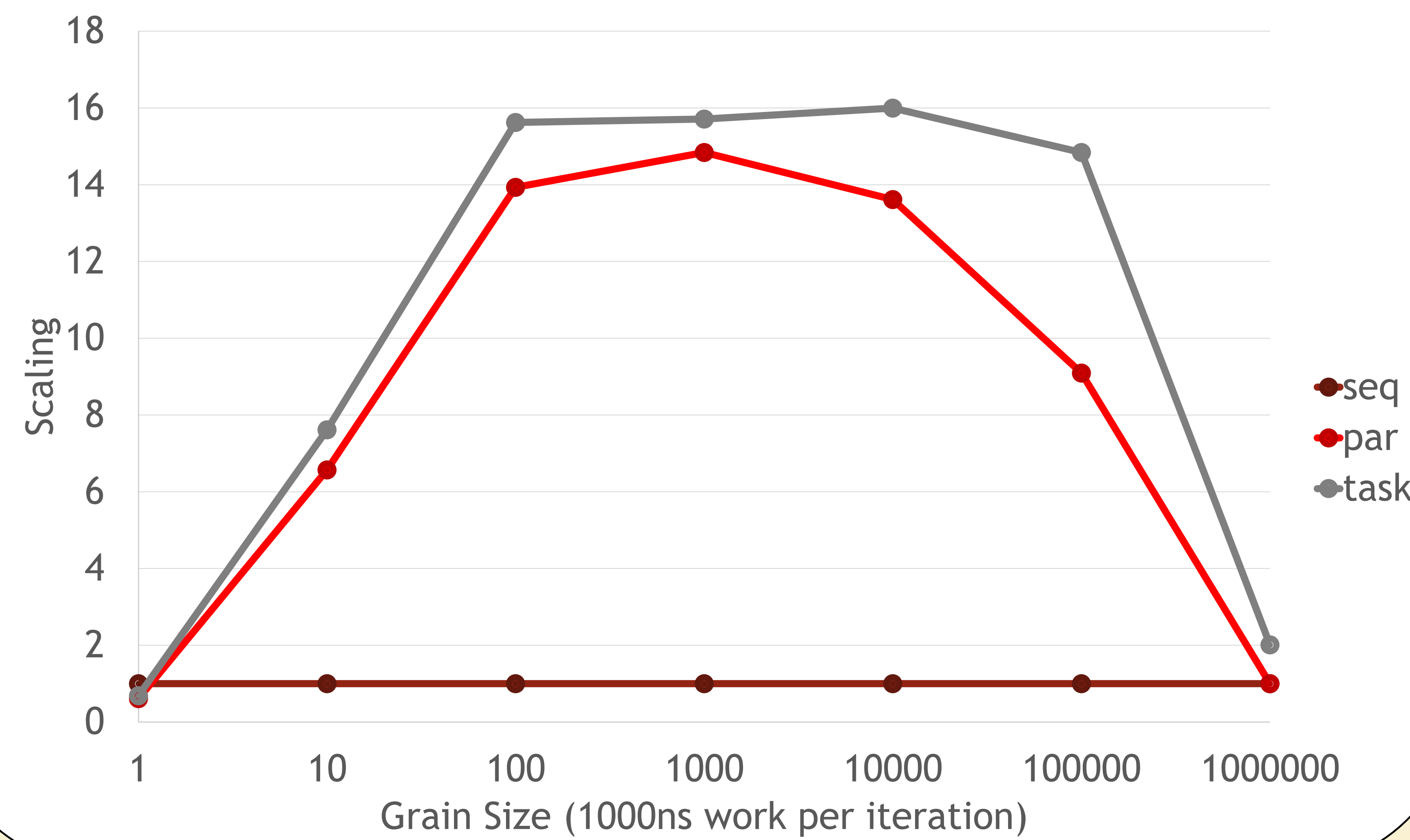


Figure 2: Execution Time of for\_each Overlapping task  
16 Cores

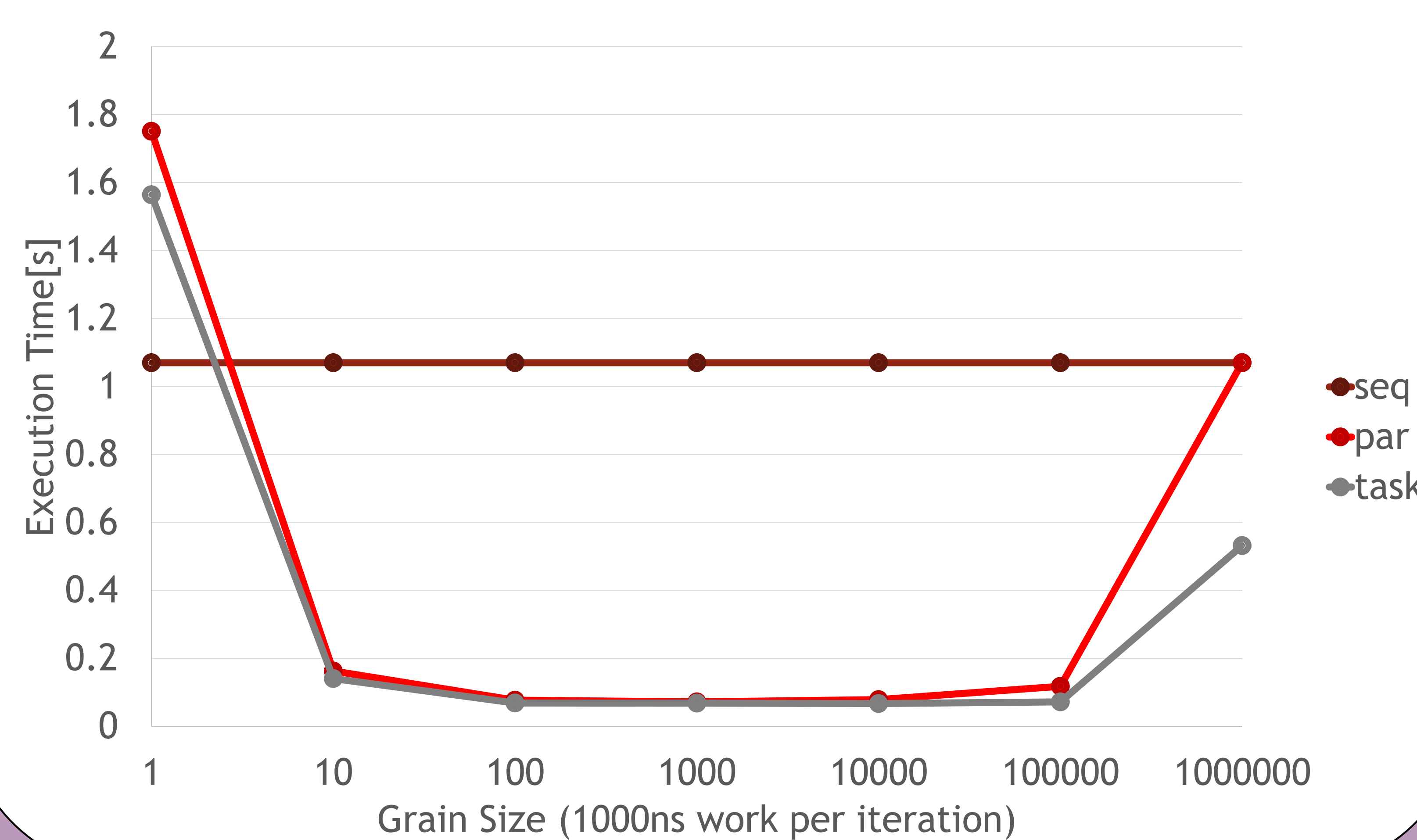
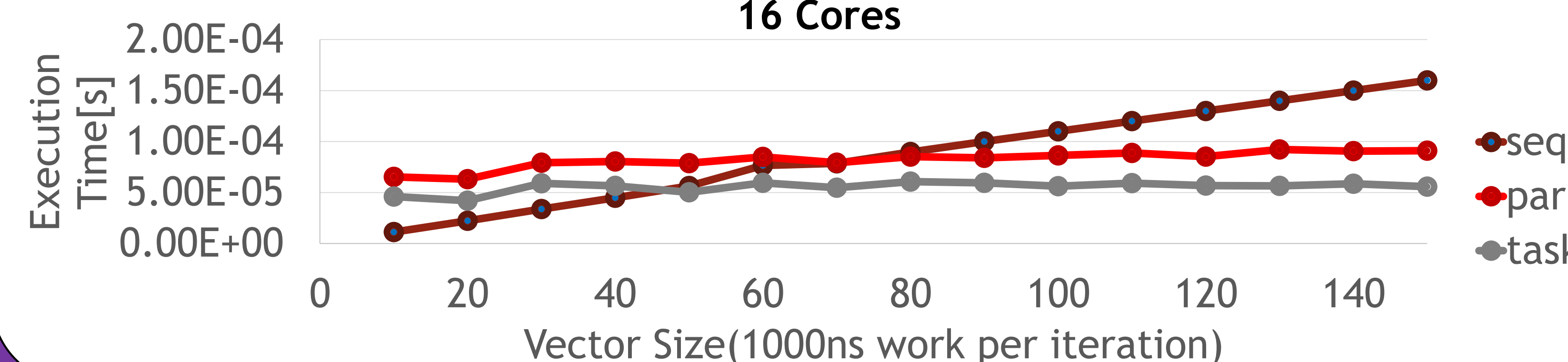


Figure 3: Execution of for\_each Overlapping Task  
16 Cores



## Task Execution

The introduction of a *task* execution policy creates the possibility to **overlap** loops, as shown in Fig. 2, the scaling of *task* when overlapping by one loop offers a large improvement over parallel execution. Fig. 1 and 3 verify that in the case overlapping is possible, *task* should always be used.

## Flexibility

Execution Policies can be determined during runtime. An algorithm can transition between policies seamlessly. Fig. 4 Demonstrates where a threshold may be placed on a certain amount of work, in this case anything greater than 70 elements can be parallelized

Figure 4: for\_each parallel overhead  
16 Cores

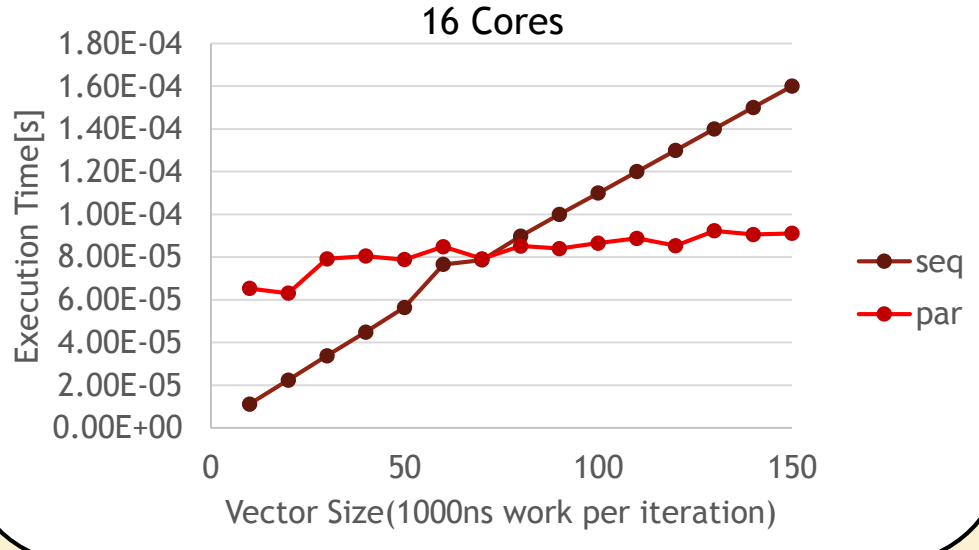
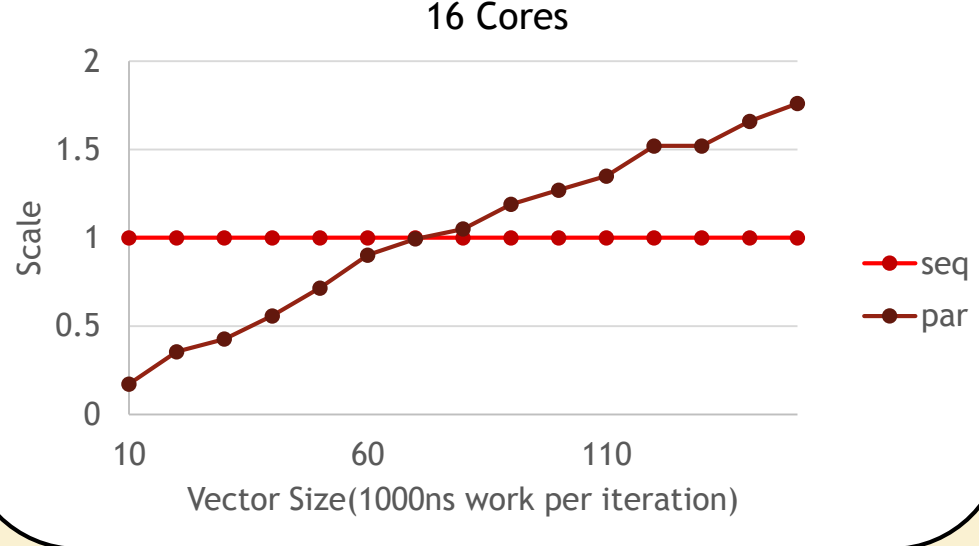


Figure 5: parallel for\_each scaling  
16 Cores



## Conclusion

HPX has a lot to gain from standard conforming parallel algorithms. The task execution policy is very important to the algorithms success in HPX, and will continue to play a large role in improving performance. Future plans with this project are to continually improve the performance of the underlying partitioners and add additional algorithms to HPX.

## Acknowledgements

- CCT
- Agustín Bergé