STE||AR GROUP

HPX

A GENERAL PURPOSE C++ RUNTIME SYSTEM FOR PARALLEL AND DISTRIBUTED APPLICATIONS OF ANY SCALE

Hartmut Kaiser (hkaiser@cct.lsu.edu)

Why HPX?

Tianhe-2's projected theoretical peak performance: 54.9 PetaFLOPs

16,000 nodes, ~3,200,000 computing cores (32,000 Intel Ivy Bridge Xeons, 48,000 Xeon Phi Accelerators)

STE||AR GROUP

Runtime Systems

FUTURE FRONTIERS FOR TASK BASED PARALLELISM

STE||AR GROUP

HPX – A General Purpose Runtime System

HPX is a parallel runtime system which builds upon the C++11/14 standard
- Facilitates distributed operations
- Enables fine-grained constraint based parallelism
- Supports runtime adaptive resource management

Solidly based on a theoretical foundation – the ParalleX execution model
- A general purpose runtime system for applications of any scale
  - http://stellar-group.org/libraries/hpx/, https://github.com/STEllAR-GROUP/hpx/

Exposes an uniform, standards-oriented API for ease of programming parallel and distributed applications.
- Enables to write fully asynchronous code using hundreds of millions of threads.
- Provides unified syntax and semantics for local and remote operations.
- Enables composable parallelism

Enables writing applications which outperform and out-scale existing ones

Highly portable, runs on Linux, Android, Windows, MacOS, Xeon/Phi, BlueGene/Q, Cray, …

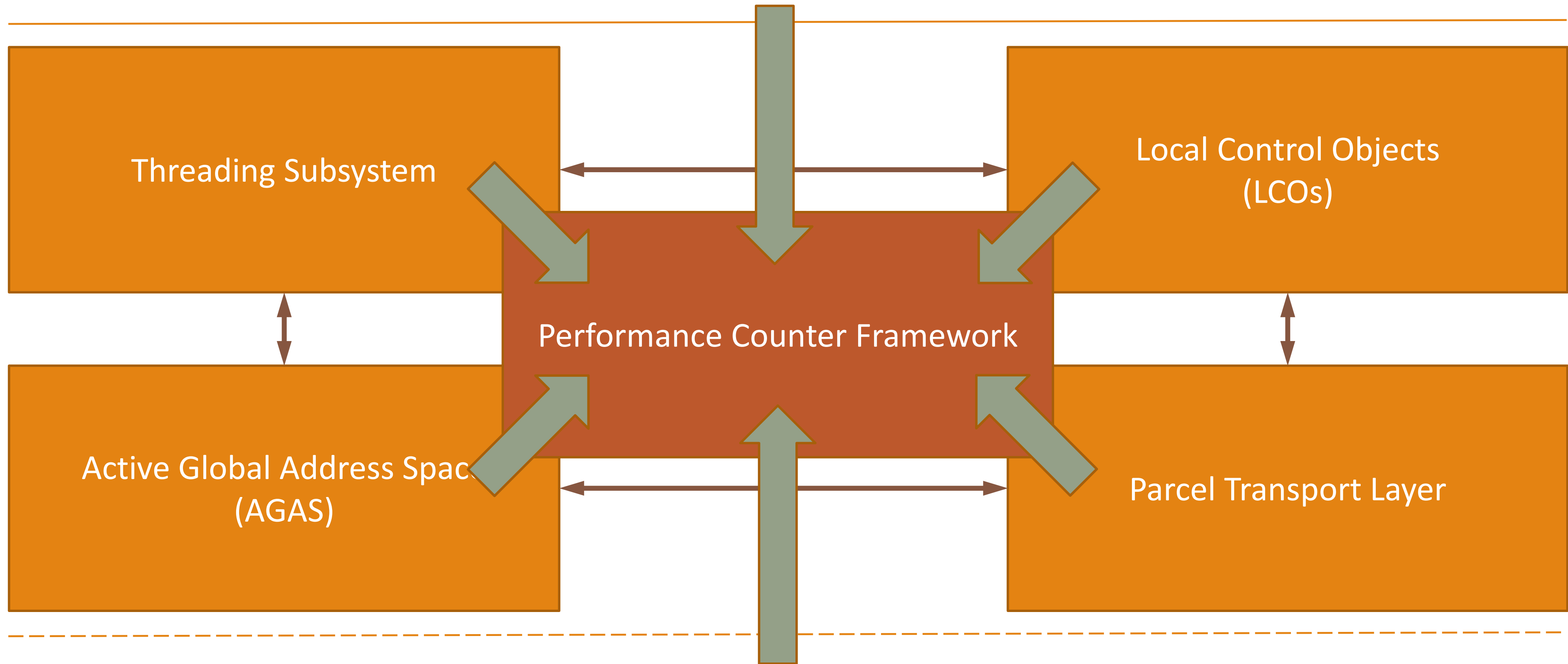
Is published under Boost license and has an open, active, and thriving developer community.

HPX – A General Purpose Runtime System

Governing principles

- Active global address space (AGAS) instead of PGAS
- Message driven computation instead of message passing
- Lightweight control objects instead of global barriers
- Latency hiding instead of latency avoidance
- Adaptive locality control instead of static data distribution
- Moving work to data instead of moving data to work
- Fine grained parallelism of lightweight threads instead of Communicating Sequential Processes (CSP/MPI)

HPX – A General Purpose Runtime System



STE||AR GROUP

HPX – The API

CONFORMING TO EXISTING STANDARDS

HPX – The API

Centered around the concept of data dependencies, not tasks

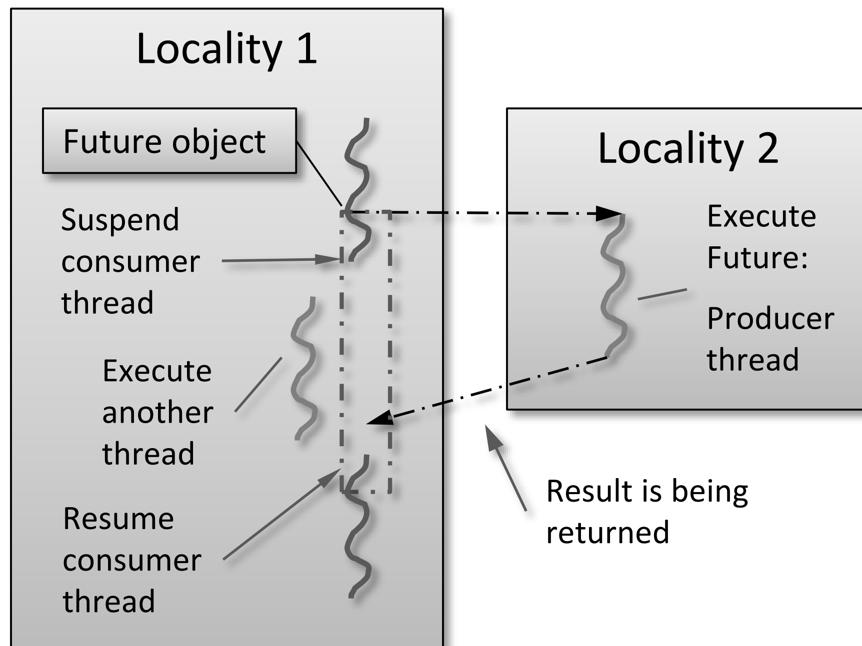
- Uses hpx::future to represent the result of an operation

Fully asynchronous

- All possibly remote operations are asynchronous by default
  - 'Fire & forget' semantics (result is not available)
  - 'Pure' asynchronous semantics (result is available via hpx::future)
- Composition of asynchronous operations (N3634)
  - hpx::when_all, hpx::when_any, hpx::when_n
  - hpx::future::then(f)
- Can be used 'synchronously', but does not block
  - Thread is suspended while waiting for result
  - Other useful work is performed transparently

What is a (the) future

A future is an object representing a result which has not been calculated yet



- Enables transparent synchronization with producer
- Hides notion of dealing with threads
- Makes asynchrony manageable
- Allows for composition of several asynchronous operations
- Turns concurrency into parallelism

What is a (the) Future?

Many ways to get hold of a future, simplest way is to use (std) async:

```
int universal_answer() { return 42; }

void deep_thought()
{
    future<int> promised_answer = async(&universal_answer);

    // do other things for 7.5 million years

    cout << promised_answer.get() << endl;   // prints 42
}
```

HPX – The API

R f(p...)	Synchronous (return R)	Asynchronous (return future<R>)	Fire & Forget (return void)
Functions (direct)	f(p…) C++	async(f, p…)	apply(f, p…)
Functions (lazy)	bind(f, p…)(…)	async(bind(f, p…), …) C++ Library	apply(bind(f, p…), …)
Actions (direct)	HPX_ACTION(f, a) a(id, p…)	HPX_ACTION(f, a) async(a, id, p…)	HPX_ACTION(f, a) apply(a, id, p…)
Actions (lazy)	HPX_ACTION(f, a) bind(a, id, p…)(…)	HPX_ACTION(f, a) async(bind(a, id, p…), …)	HPX_ACTION(f, a) apply(bind(a, id, p…), …) HPX

STE||AR GROUP

HPX – The API

As close as possible to C++11 standard library, where appropriate, for instance

- std::thread → hpx::thread
- std::mutex → hpx::mutex
- std::future → hpx::future (including N3857)
- std::async → hpx::async
- std::bind → hpx::bind
- std::function → hpx::function
- std::tuple → hpx::tuple
- std::any → hpx::any (N3508)
- std::cout → hpx::cout
- std::parallel::for_each, etc.
  → hpx::parallel::for_each (N4071, N4088)

STE||AR GROUP

HPX – The API

Fully move enabled (using C++11 move semantics)

- hpx::bind, hpx::function, hpx::tuple, hpx::any

Fully type safe remote operation

- Extends the notion of a ‘callable’ to remote case (actions)
- Everything you can do with functions is possible with actions as well

Data types are usable in remote contexts

- Can be sent over the wire (hpx::bind, hpx::function, hpx::any)
- Can be used with actions (hpx::async, hpx::bind, hpx::function)

Unifies local and remote operation for the application programmer

- Object migration to other localities

STE||AR GROUP

HPX – The API

Additional constructs for composing futures (N3857)

- Sequential composition (attach continuation):

```
future<decltype(F(future<T>))>
    future<T>::then(F f);
```

- Parallel composition:

```
future<tuple<future<T>, ...>>
    when_all(future<T> f, ...);

future<tuple<future<T>, ...>>
    when_any(future<T> f, ...);
```

- Dataflow:

```
future<decltype(F(future<T> f1, ...))>
    dataflow(F f, future<T> f1, ...);
```

Parallel Algorithms

Parallel algorithms (N4071)

- Mostly, same semantics as sequential algorithms
- Additional, first argument: execution_policy (seq, par, etc.)

Extension

- task_execution_policy
- Algorithm returns future<>

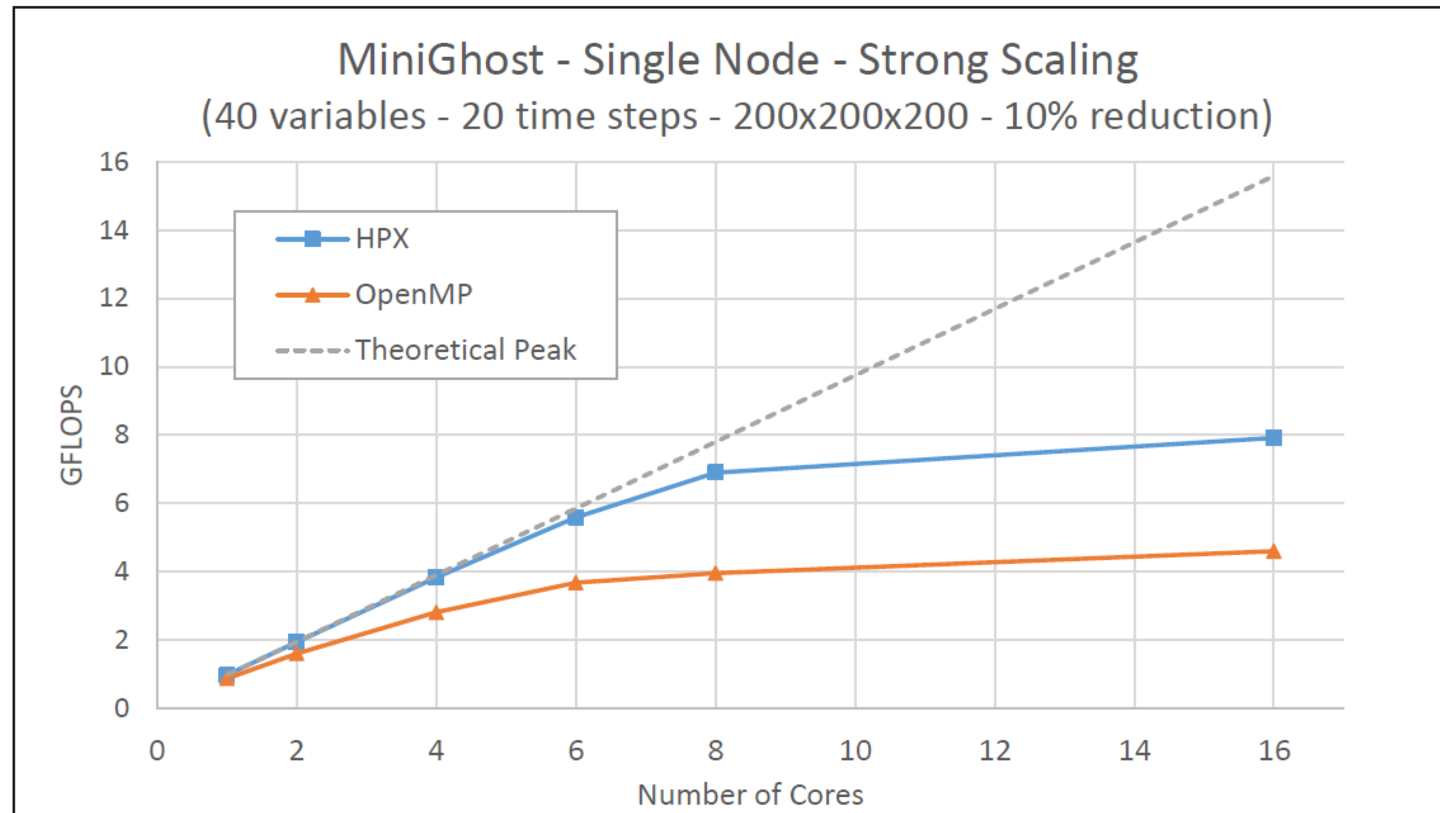
adjacent_difference	adjacent_find	all_of	any_of
copy	copy_if	copy_n	count
count_if	equal	exclusive_scan	fill
fill_n	find	find_end	find_first_of
find_if	find_if_not	for_each	for_each_n
generate	generate_n	includes	inclusive_scan
inner_product	inplace_merge	is_heap	is_heap_until
is_partitioned	is_sorted	is_sorted_until	lexicographical_compare
max_element	merge	min_element	minmax_element
mismatch	move	none_of	nth_element
partial_sort	partial_sort_copy	partition	partition_copy
reduce	remove	remove_copy	remove_copy_if
remove_if	replace	replace_copy	replace_copy_if
replace_if	reverse	reverse_copy	rotate
rotate_copy	search	search_n	set_difference
set_intersection	set_symmetric_difference	set_union	sort
stable_partition	stable_sort	swap_ranges	transform
uninitialized_copy	uninitialized_copy_n	uninitialized_fill	uninitialized_fill_n
unique	unique_copy		

STE||AR GROUP

Recent Results

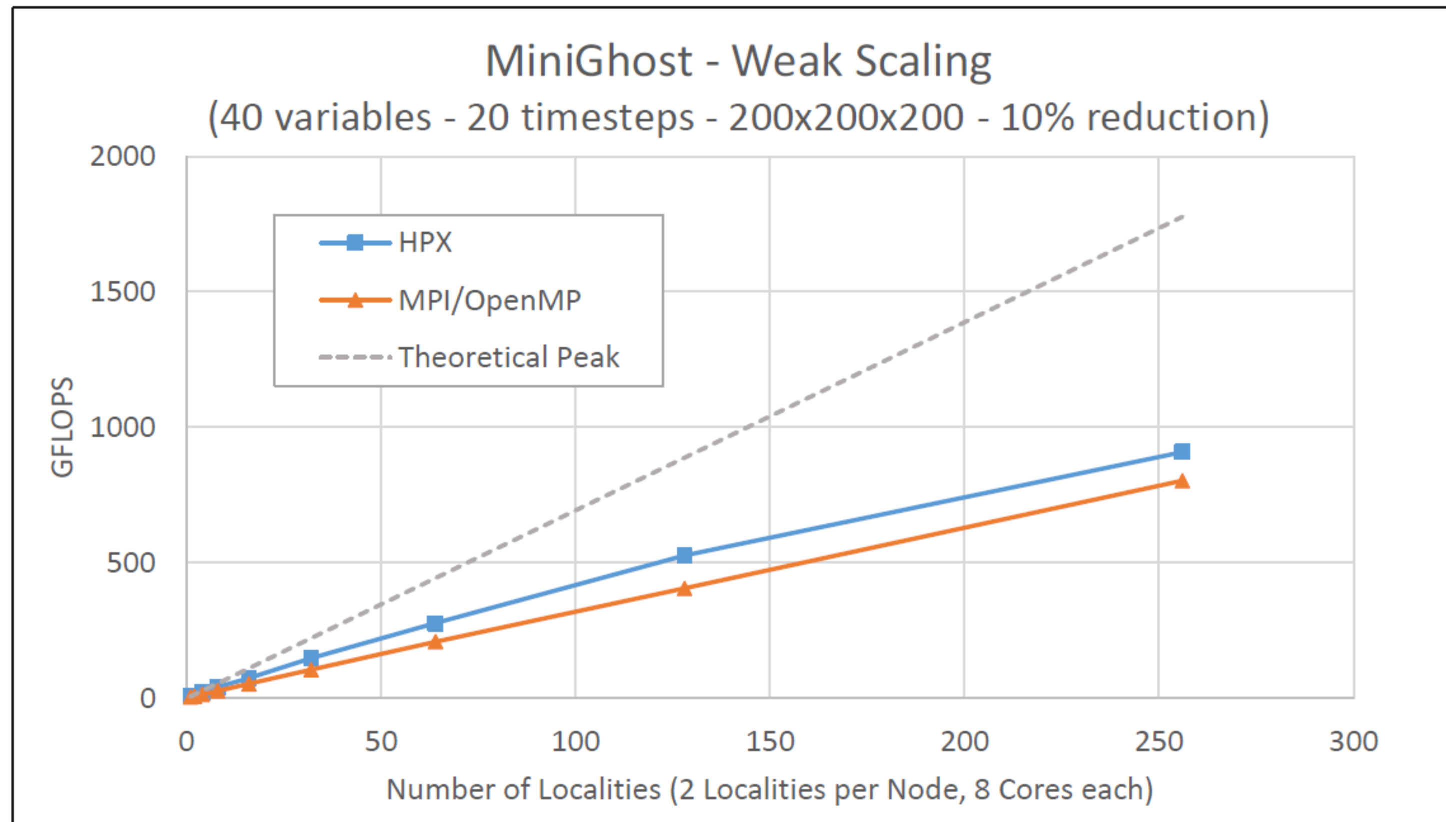
STE||AR GROUP

Mini-Ghost (SMP)



STE||AR GROUP

Mini-Ghost (distributed runs)



STE||AR GROUP

Future Work

STE||AR GROUP

STORM

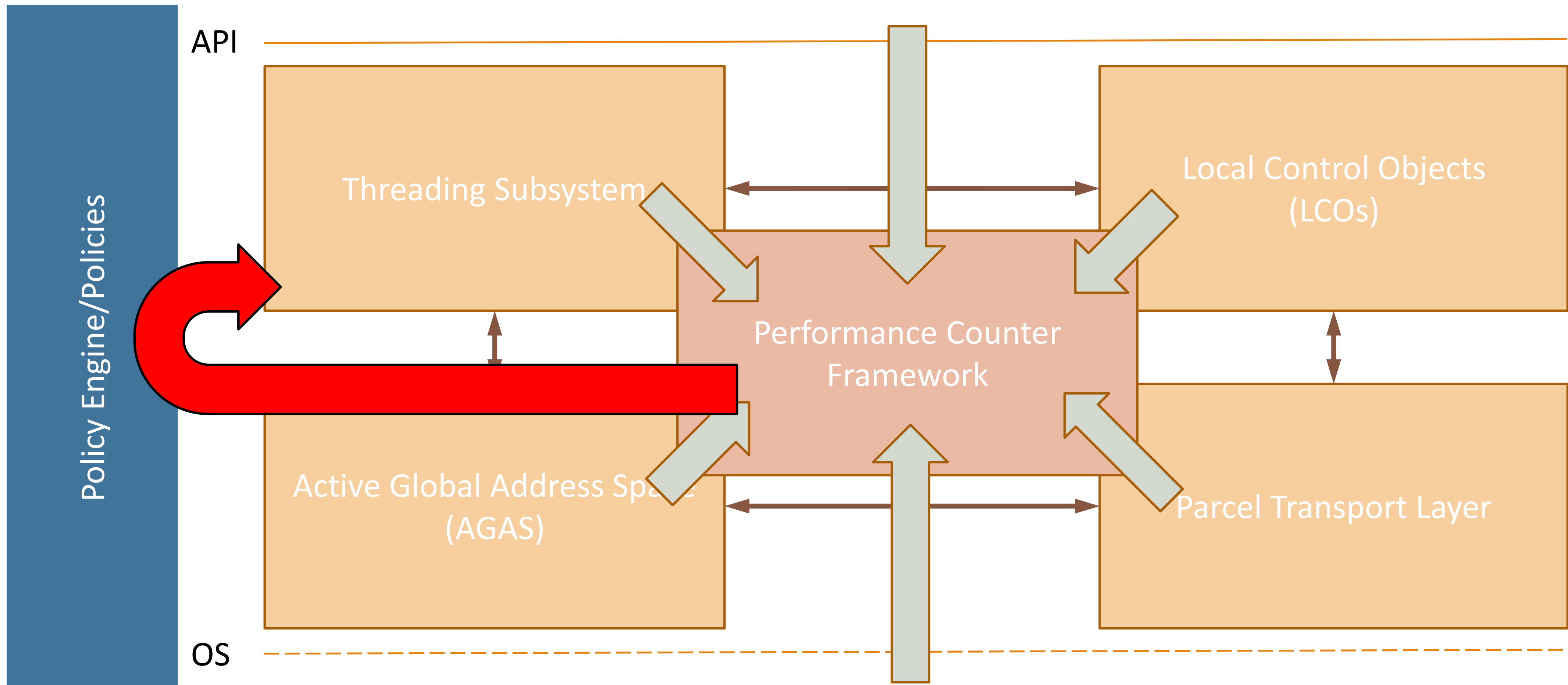
Utilize HPX’ fine grain parallelism and distributed management capabilities

Implement runtime adaptive features to improve load balancing and parallel efficiency

Enable running ADCIRC on future architectures

STE||AR GROUP

Runtime Adaptive Resource Management



LSU
CENTER FOR COMPUTATION
& TECHNOLOGY

STE||AR GROUP