How runtime systems can support resource awareness in HPC: the HPX case

Tommaso Bianucci

Technische Universität München

22 June 2018

Exascale will be hard

- 1 ExaFLOPS $= 10^{18}$ FLOPS
- Billions of cores?
- Etherogeneous hardware
  - Manycore CPUs
  - GPUs
  - FPGAs

$\longrightarrow$ These machines expose an extreme degree of parallelism.

Exascale will be hard

- 1 ExaFLOPS = 10^{18} FLOPS
- Billions of cores?
- Etherogeneous hardware
  - Manycore CPUs
  - GPUs
  - FPGAs

$\longrightarrow$ These machines expose an extreme degree of parallelism.

Applications are hard

- *Scaling-impaired* applications
- Unbalanced execution tree

This causes:

- Poor parallel performance
- Suboptimal resource usage

$\longrightarrow$ Some applications do not scale well.

Applications are hard

- *Scaling-impaired* applications
- Unbalanced execution tree

This causes:

- Poor parallel performance
- Suboptimal resource usage

$\longrightarrow$ Some applications do not scale well.

Applications are hard

- *Scaling-impaired* applications
- Unbalanced execution tree

This causes:

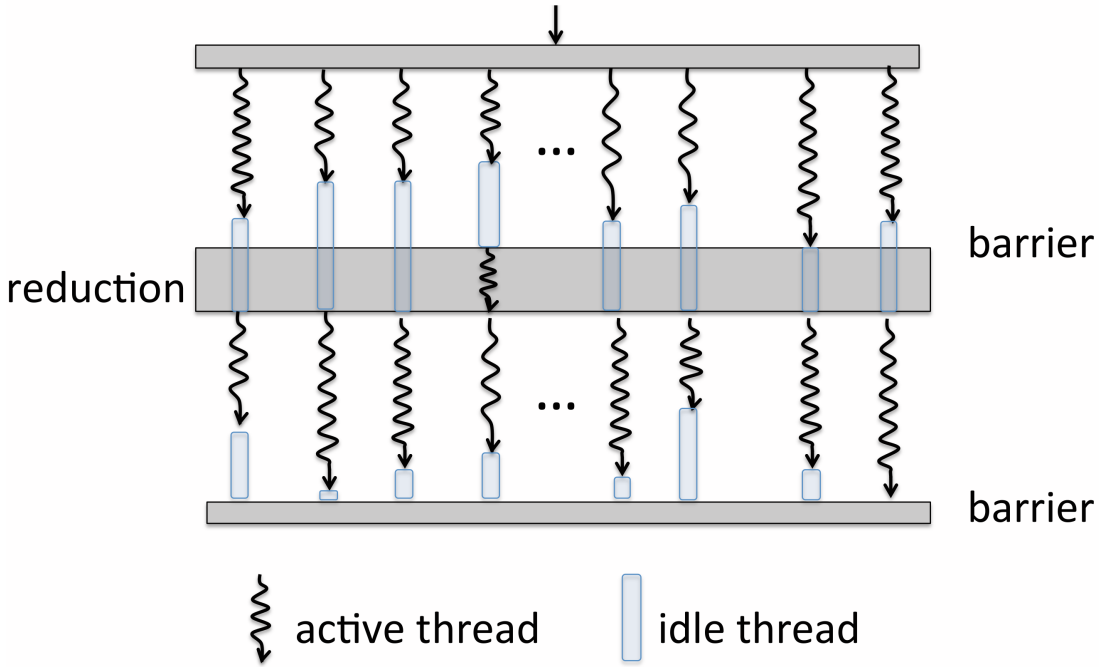
- Poor parallel performance
- Suboptimal resource usage

$\longrightarrow$ Some applications do not scale well.

Programming model matters

Current predominant model in HPC:

- *Fork-join* for shared memory (OpenMP)
- *Communicating Sequential Processes* for distributed memory (MPI)



Problems:

- Global barriers
- Load imbalance

P.A.Grubel:"Dynamic adaptation in hpx, a task-based parallel runtime system" 2016.

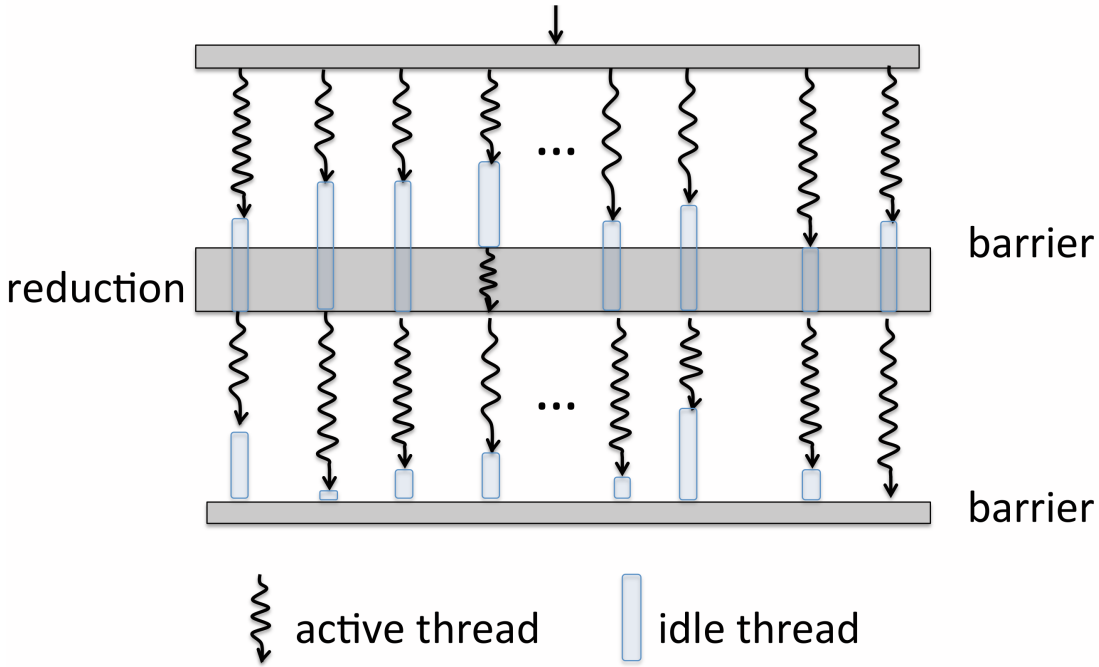
Programming model matters

Current predominant model in HPC:

- *Fork-join* for shared memory (OpenMP)
- *Communicating Sequential Processes* for distributed memory (MPI)

Problems:

- Global barriers
- Load imbalance



P.A.Grubel:"Dynamic adaptation in hpx, a task-based parallel runtime system" 2016.

Resource awareness: clever software for complex hardware

Resource awareness

- Adaptive allocation and usage of resources
- The system is aware of its own resources
- At runtime vs. before execution

Resource awareness: clever software for complex hardware

Resource awareness

- Adaptive allocation and usage of resources
- The system is aware of its own resources
- At runtime vs. before execution

What are resources?

1. Hardware entities
   - Computational units
   - Memory
   - Bus/network bandwidth
   - I/O devices
   - Power
   - Thermal
2. Software entities
   - Buffers
   - Queues

What are resources?

1. Hardware entities
   - Computational units
   - Memory
   - Bus/network bandwidth
   - I/O devices
   - Power
   - Thermal
2. Software entities
   - Buffers
   - Queues

Different levels of awareness

1. Embedded computing
   E.g.: Invasive computing on MPSoC

2. Application/runtime system level
   E.g.: load balance, task scheduling

3. Supercomputing facility
   E.g.: Invasive MPI + job scheduler integration

Different levels of awareness

1. Embedded computing
   E.g.: Invasive computing on MPSoC

2. Application/runtime system level
   E.g.: load balance, task scheduling

3. Supercomputing facility
   E.g.: Invasive MPI + job scheduler integration

Different levels of awareness

1. Embedded computing
   E.g.: Invasive computing on MPSoC

2. Application/runtime system level
   E.g.: load balance, task scheduling

3. Supercomputing facility
   E.g.: Invasive MPI + job scheduler integration

High Performance paralleX

C++ runtime system for

- *Task-based* parallelism
- *Shared memory* + *Distributed memory* parallelization
- Fine-grained parallelism

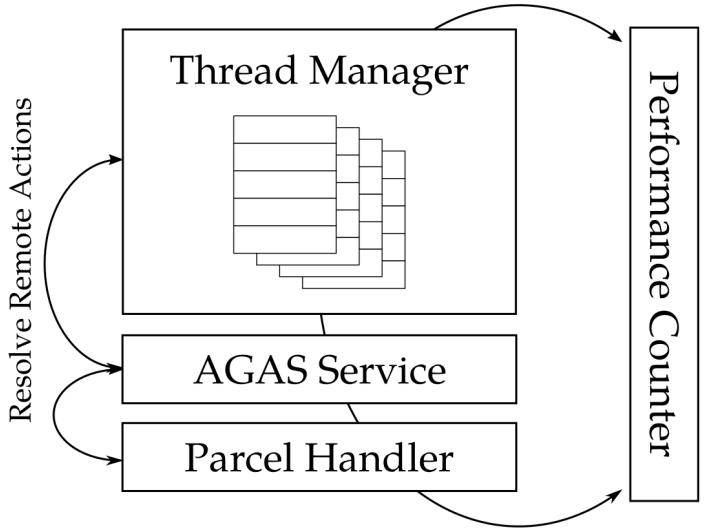
High Performance paralleX

C++ runtime system for

- *Task-based* parallelism
- *Shared memory* + *Distributed memory* parallelization
- Fine-grained parallelism

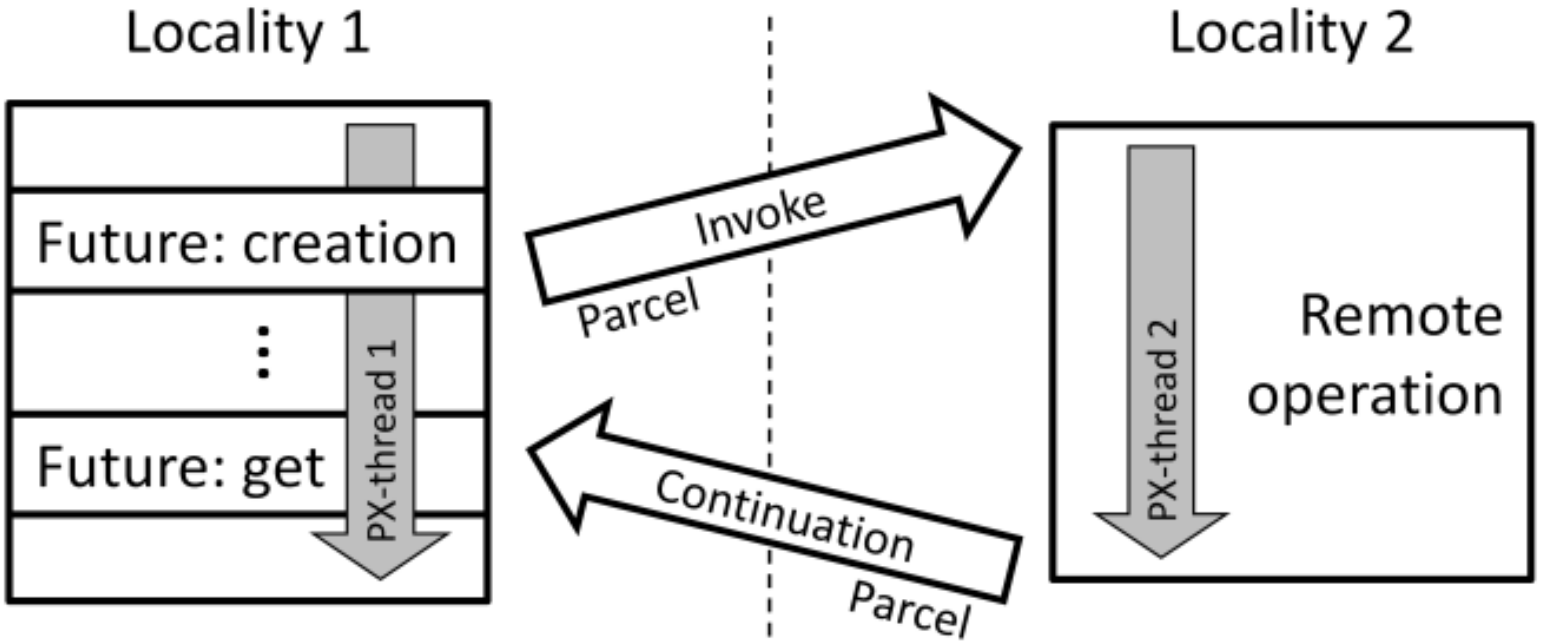
HPX foundations

- Asynchronous scheduling and execution
- Lightweight synchronization
- Active Global Address Space (AGAS)
- Performance monitoring framework



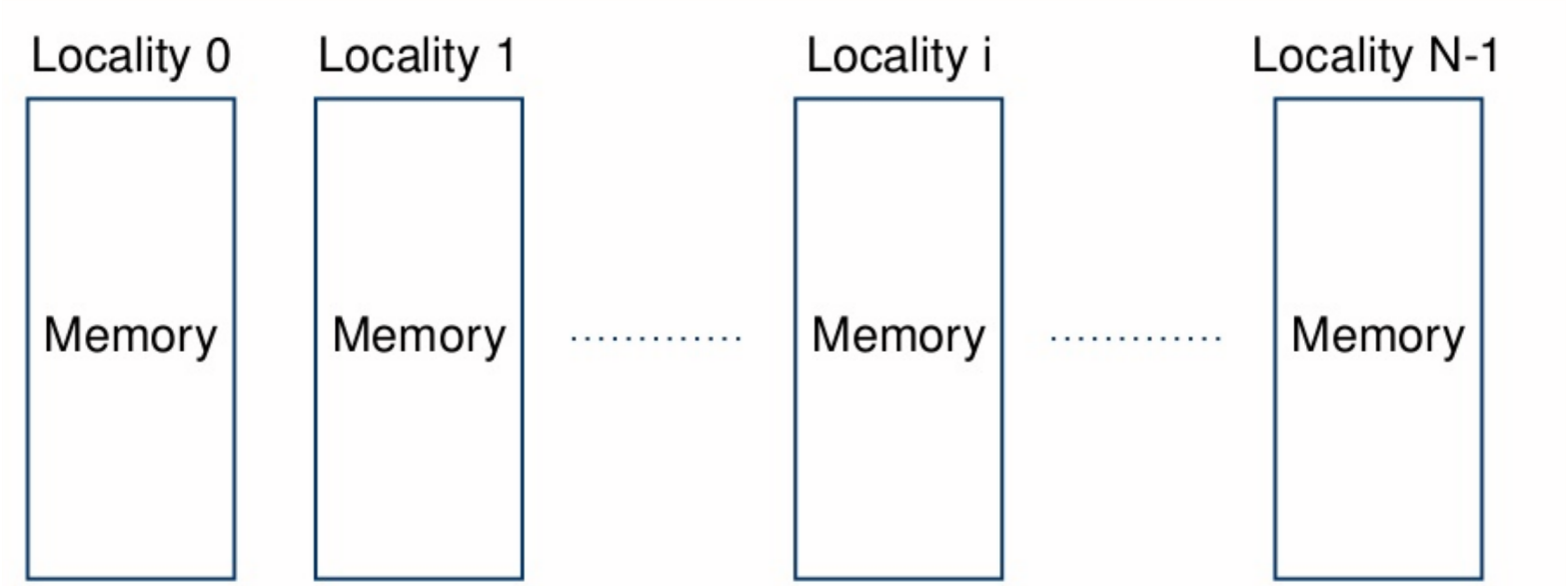
T. Heller et al.: "Hpx – an open source c++ standard library for parallelism and concurrency" 2017.

Futures



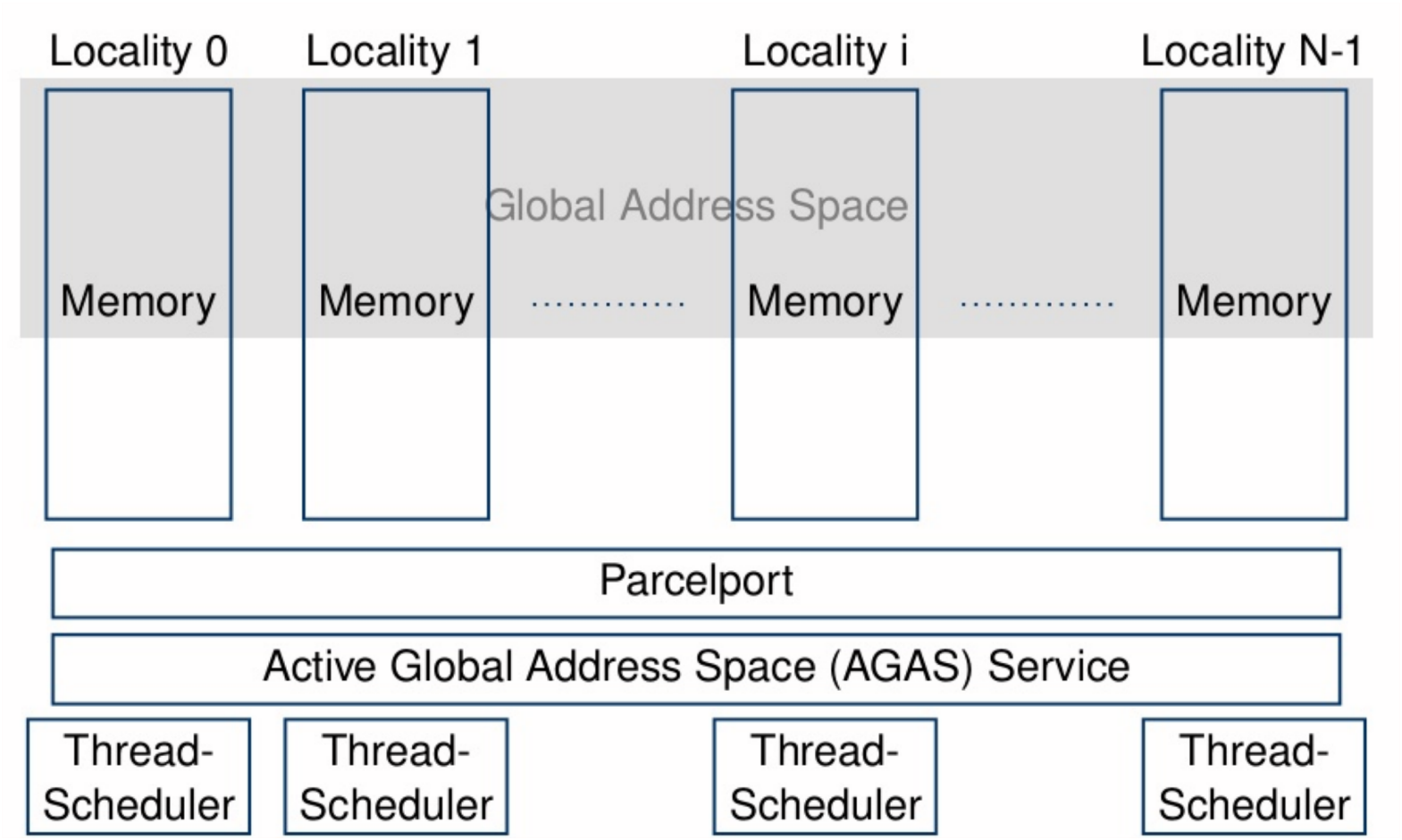
H. Kaiser et al.: "Parallex an advanced parallel execution model for scaling-impaired applications" 2009.

Programming model



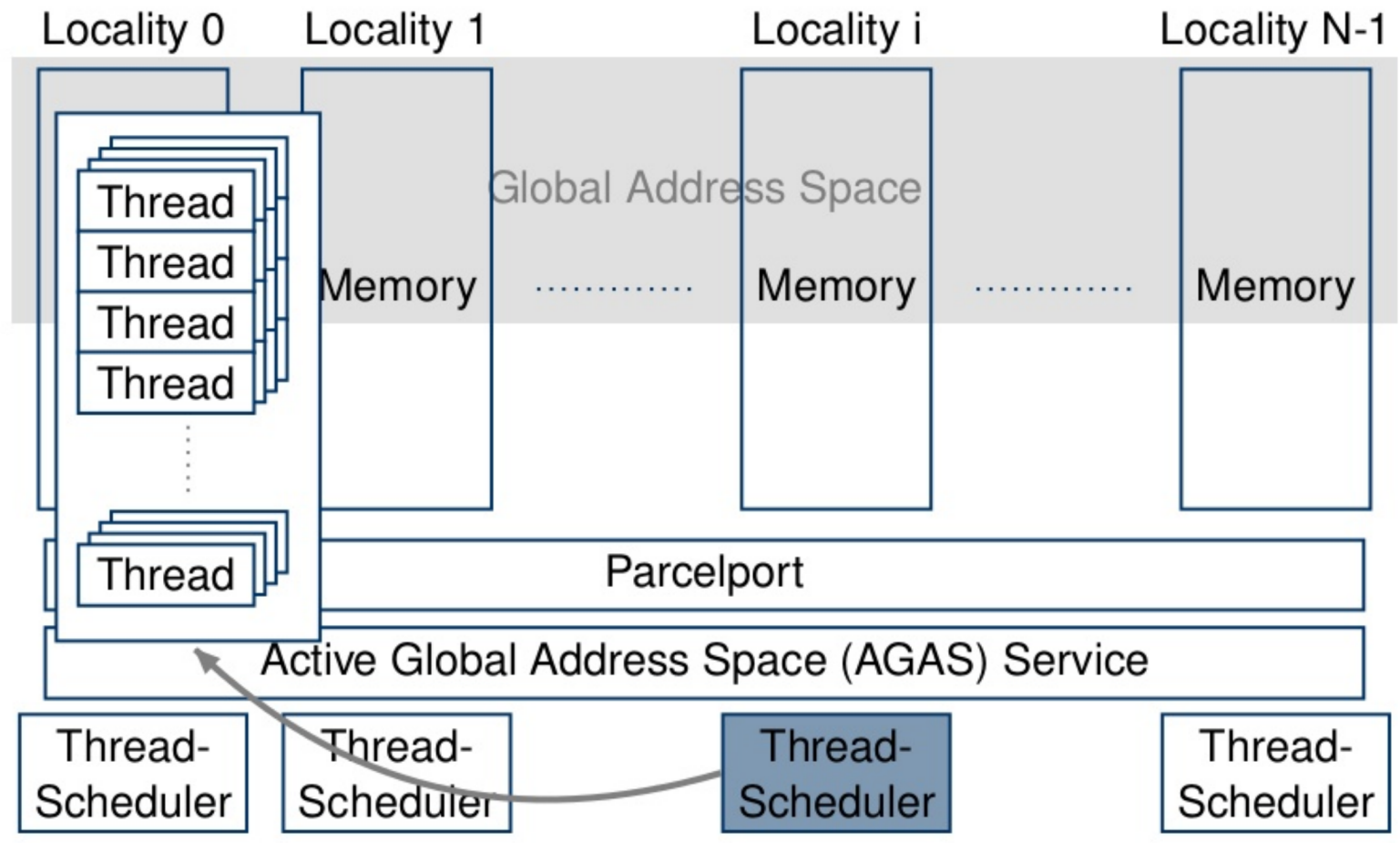
T. Heller: "C++ on its way to exascale and beyond - The HPX Parallel Runtime System" 2016.

Programming model



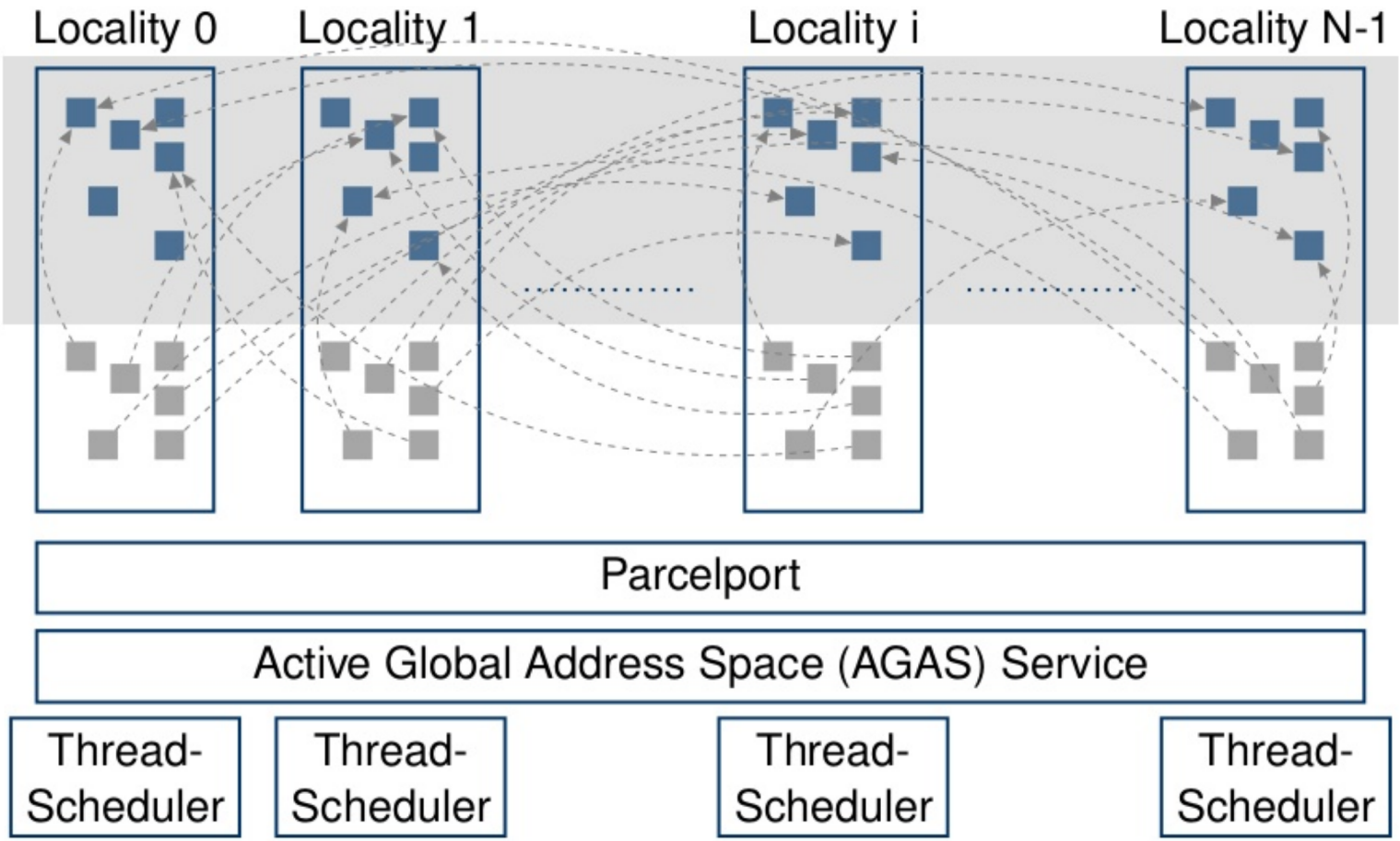
T. Heller: "C++ on its way to exascale and beyond - The HPX Parallel Runtime System" 2016.

Programming model



T. Heller: "C++ on its way to exascale and beyond - The HPX Parallel Runtime System" 2016.

Programming model



T. Heller: "C++ on its way to exascale and beyond - The HPX Parallel Runtime System" 2016.

HPX and Resource Awareness

Capabilities

1. Task scheduling
   $\longrightarrow$ Work stealing + NUMA-awareness
2. AGAS
   $\longrightarrow$ Dynamic relocation of objects
3. Percolation
   $\longrightarrow$ Directly addressing HW accelerators
4. Performance counters
   $\longrightarrow$ Easier integration into applications

HPX and Resource Awareness

Capabilities

1. Task scheduling
   $\longrightarrow$ Work stealing + NUMA-awareness
2. AGAS
   $\longrightarrow$ Dynamic relocation of objects
3. Percolation
   $\longrightarrow$ Directly addressing HW accelerators
4. Performance counters
   $\longrightarrow$ Easier integration into applications

HPX and Resource Awareness

Capabilities

1. Task scheduling
   $\longrightarrow$ Work stealing + NUMA-awareness
2. AGAS
   $\longrightarrow$ Dynamic relocation of objects
3. Percolation
   $\longrightarrow$ Directly addressing HW accelerators
4. Performance counters
   $\longrightarrow$ Easier integration into applications

HPX and Resource Awareness

Capabilities

1. Task scheduling
   $\longrightarrow$ Work stealing + NUMA-awareness
2. AGAS
   $\longrightarrow$ Dynamic relocation of objects
3. Percolation
   $\longrightarrow$ Directly addressing HW accelerators
4. Performance counters
   $\longrightarrow$ Easier integration into applications

HPX and Resource Awareness

Capabilities

1. Task scheduling
   $\longrightarrow$ Work stealing + NUMA-awareness
2. AGAS
   $\longrightarrow$ Dynamic relocation of objects
3. Percolation
   $\longrightarrow$ Directly addressing HW accelerators
4. Performance counters
   $\longrightarrow$ Easier integration into applications

HPX and Resource Awareness (2)

Limitations

1. (An)elasticity of HPX
   $\longrightarrow$ Worker threads and localities cannot be changed at runtime
2. Energy unawareness
   $\longrightarrow$ E.g. no DVFS support
3. Fault tolerance
   $\longrightarrow$ No built-in facility

HPX and Resource Awareness (2)

Limitations

1. (An)elasticity of HPX
   $\longrightarrow$ Worker threads and localities cannot be changed at runtime
2. Energy unawareness
   $\longrightarrow$ E.g. no DVFS support
3. Fault tolerance
   $\longrightarrow$ No built-in facility

HPX and Resource Awareness (2)

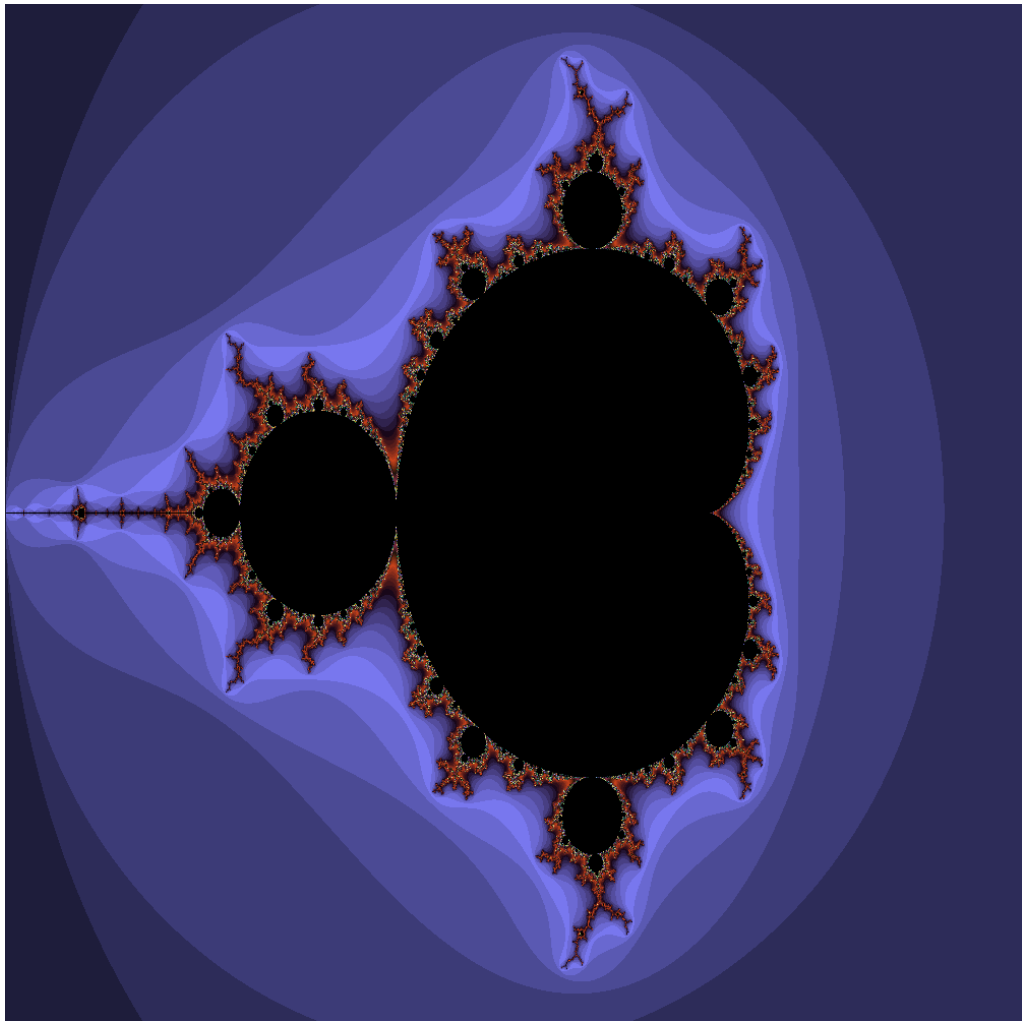
Limitations

1. (An)elasticity of HPX
   $\longrightarrow$ Worker threads and localities cannot be changed at runtime
2. Energy unawareness
   $\longrightarrow$ E.g. no DVFS support
3. Fault tolerance
   $\longrightarrow$ No built-in facility

HPX coding example: the Mandelbrot set

$$\begin{cases} z_{c,0} & = 0 \\ z_{c,n+1} & = z_{c,n}^2 + c \end{cases}$$

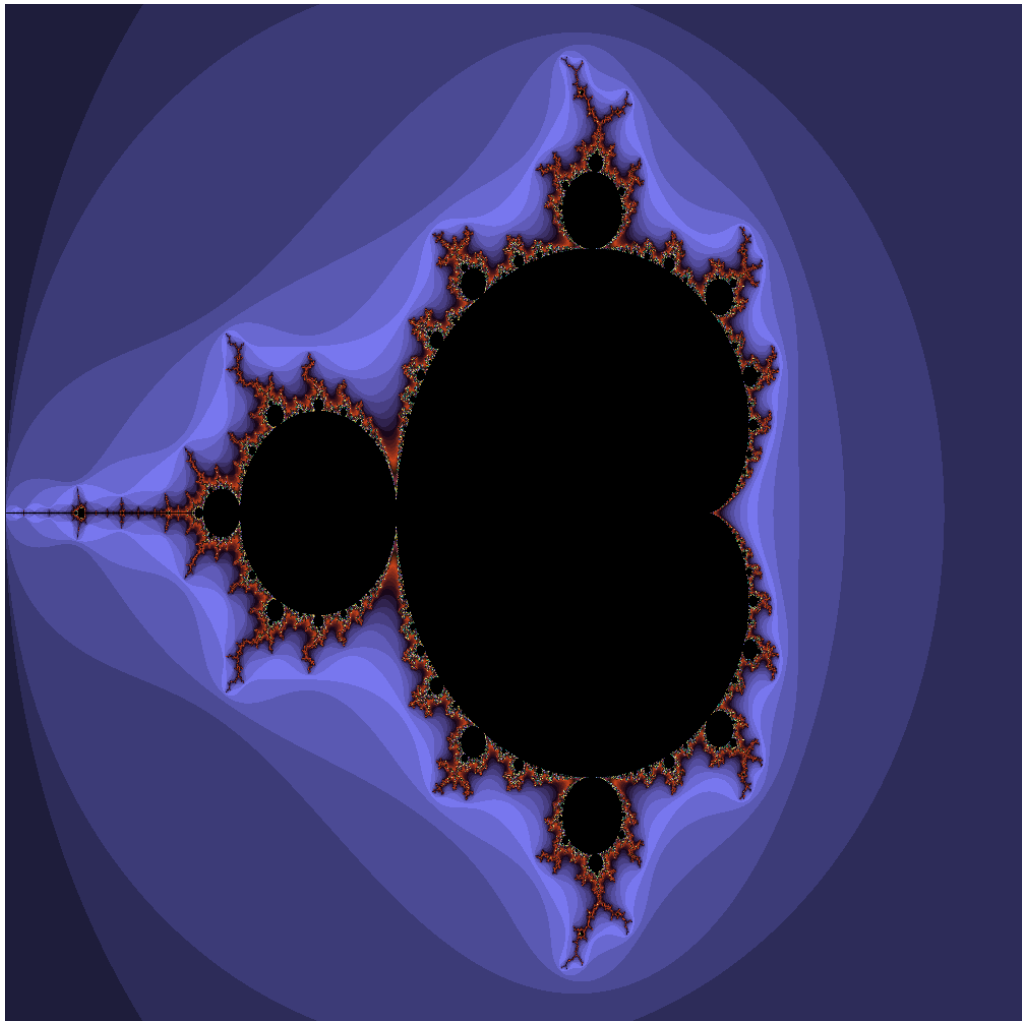
$$\mathcal{M} = \{c \in \mathbb{C} : \lim_{n \to \infty} |z_{c,n}| < +\infty\}$$



HPX coding example: the Mandelbrot set

$$\begin{cases} z_{c,0} & = 0 \\ z_{c,n+1} & = z_{c,n}^2 + c \end{cases}$$

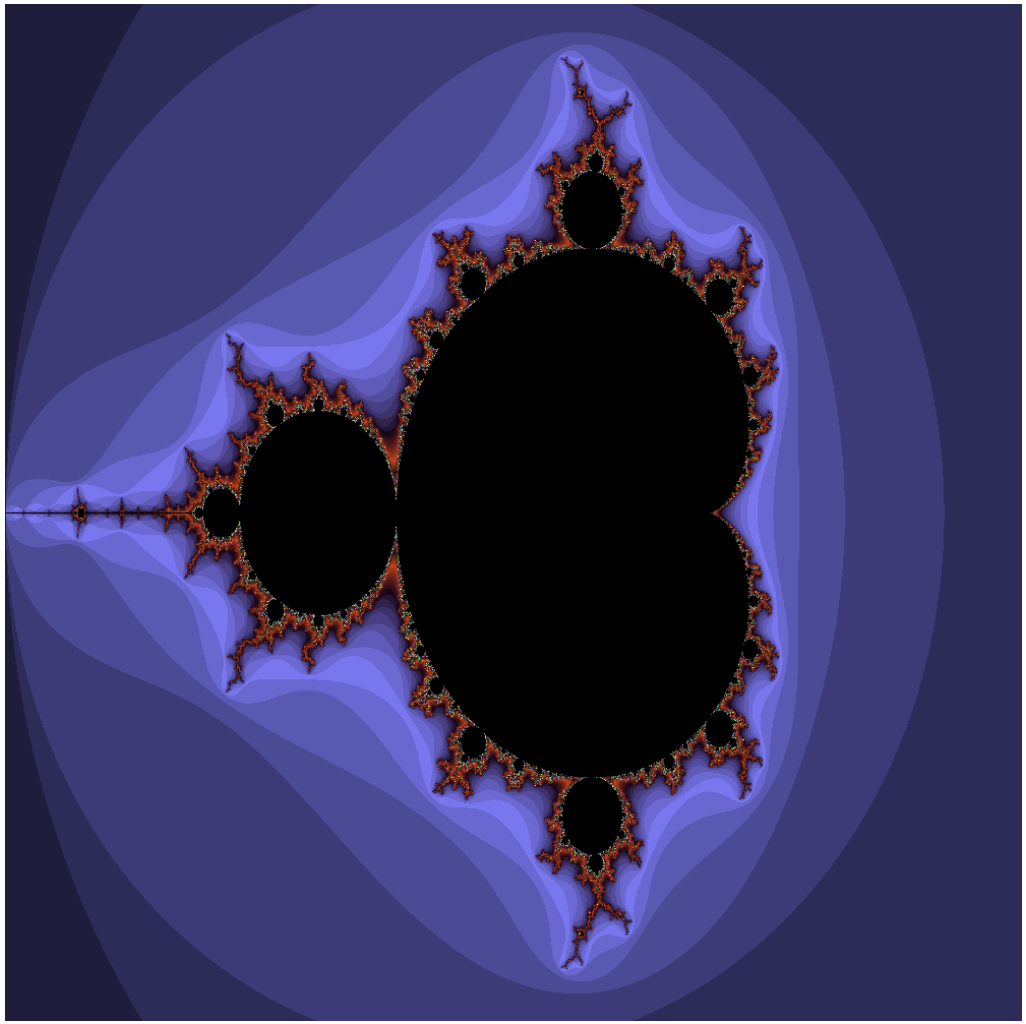
$$\mathcal{M} = \{c \in \mathbb{C} : \lim_{n \to \infty} |z_{c,n}| < +\infty\}$$



HPX coding example: the Mandelbrot set

$$\begin{cases} z_{c,0} & = 0 \\ z_{c,n+1} & = z_{c,n}^2 + c \end{cases}$$

$$\mathcal{M} = \{c \in \mathbb{C} : \lim_{n \to \infty} |z_{c,n}| < +\infty\}$$



Mandelbrot: kernel

```
1  void mandelbrot_kernel(int taskNo, staticData *sd) {
2    // Computes one row of the image
3    int i = taskNo;
4    for (int j=0; j < sd->xRes; ++j) {
5      int x = getX(j, sd);
6      int y = getY(i, sd);
7      complex double Z = 0 + 0*I;
8      complex double C = x + y*I;
9
10     int k = 0;
11     do {  // Check the convergence of the sequence
12       Z = Z * Z + C;
13       k++;
14     } while (cabs(Z) < 2 && k < max_iter);
15
16     if (k == max_iter) {  // In case it did not diverge...
17       memcpy(img[i][j], black, 3); //...we set a black pixel
18     }
19     else {  // If it diverged...
20       //...we set the color according to k (#iterations)
21       memcpy(img[i][j], getColor(k), 3);
22     }
23   }
24 }
```

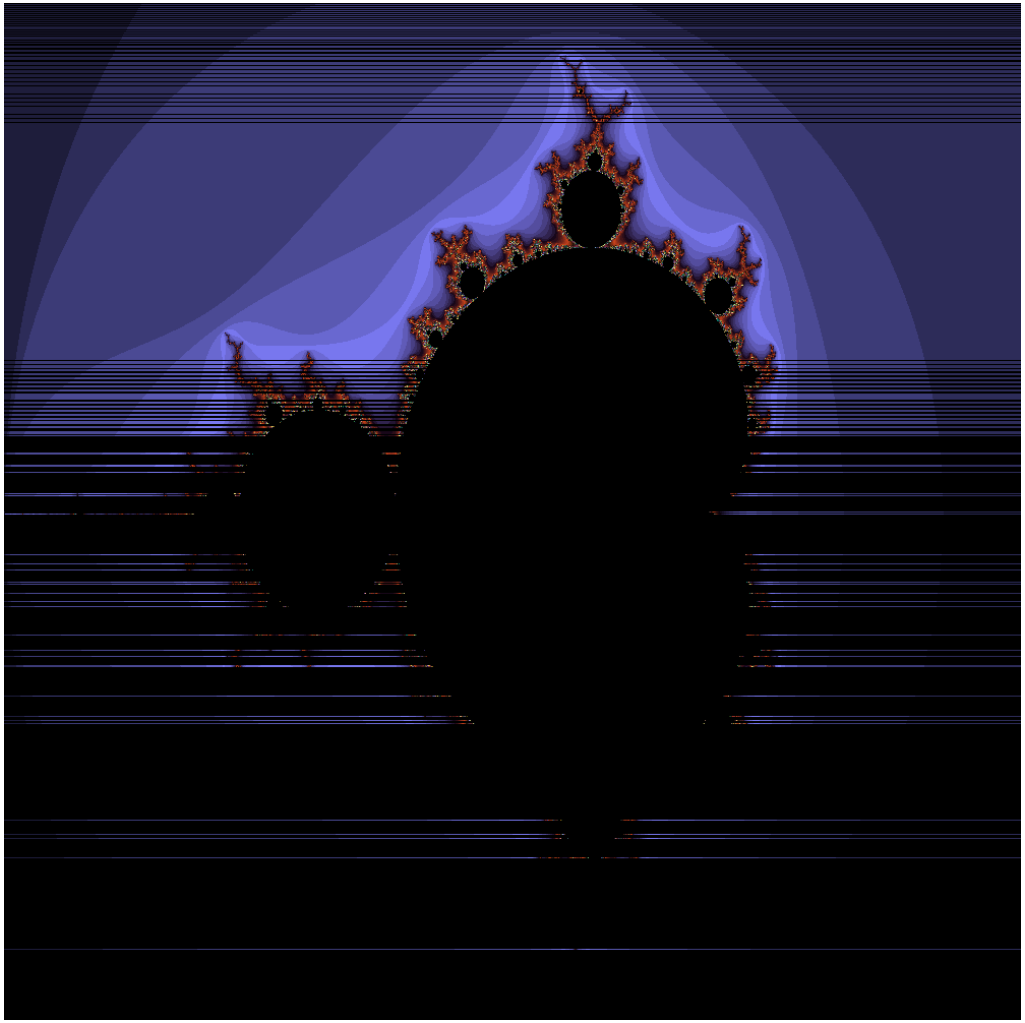
Mandelbrot: sequential code

```
1  void mandelbrot_seq(...) {
2
3    staticData *sd = assembleStaticData(...);
4
5    // Iterate on rows, sequentially
6    for (int i=0; i < yRes; ++i)
7    {
8      mandelbrot_kernel(i, sd);
9    }
10 }
```

Mandelbrot: futurized code

```
1  void mandelbrot_hpx(...) {
2
3    staticData *sd = assembleStaticData(...);
4
5    std::vector<hpx::future<void>> futures;
6    for (int i=0; i < yRes; ++i)
7    {
8      hpx::future<void> f = hpx::async(&mandelbrot_kernel, i, sd);
9      futures.push_back(f);
10   }
11
12   hpx::wait_all(futures);
13 }
```

Mandelbrot step by step



Summary

- Future exascale computing requires smart code.
- Resource awareness can be a way to achieve better performance.
- HPX has the potential to become a major runtime system for HPC, thanks to both its performance and programmability.

Thanks!

Summary

- Future exascale computing requires smart code.
- Resource awareness can be a way to achieve better performance.
- HPX has the potential to become a major runtime system for HPC, thanks to both its performance and programmability.

Thanks!