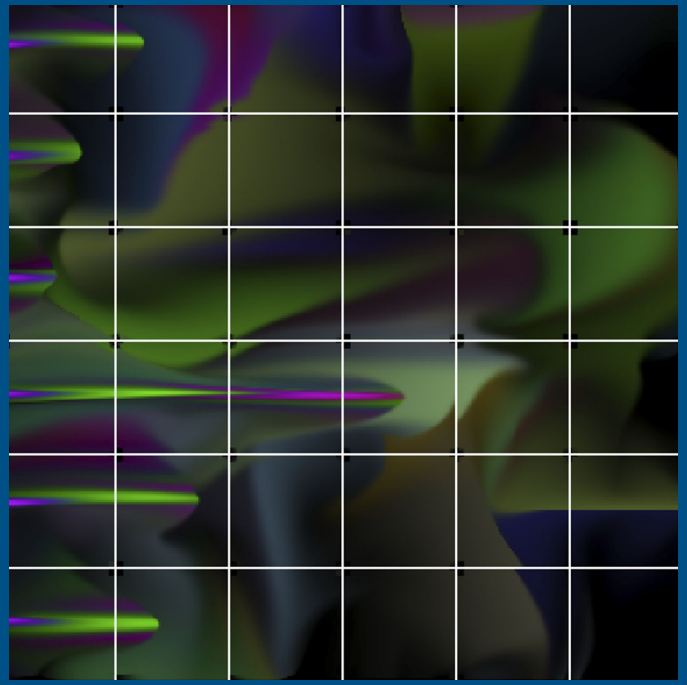
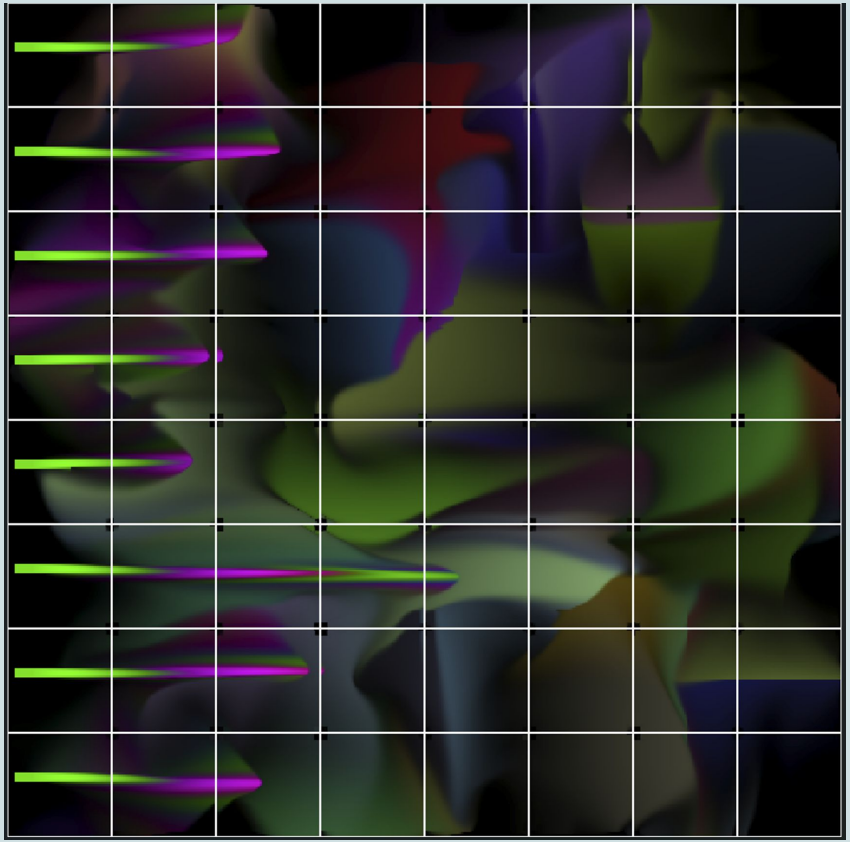
Distributed 2D Fluid Simulation

Panagiotis (Panos) Syskakis
Graduate Student – LSU



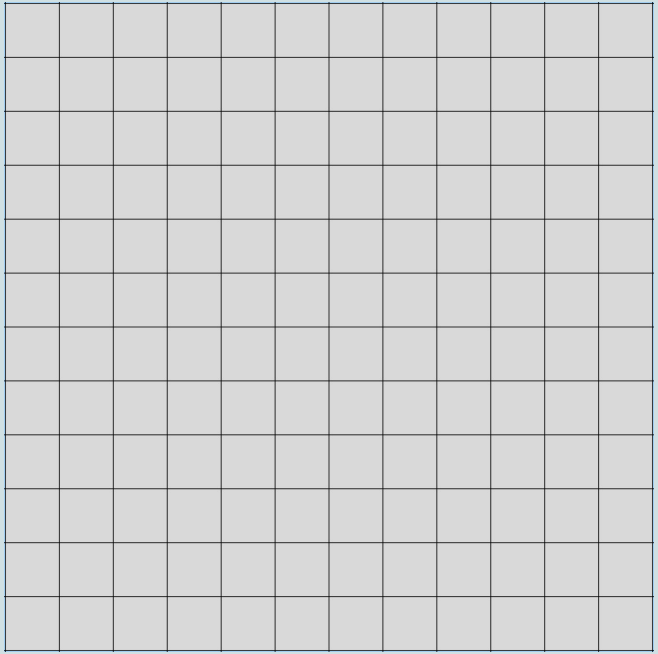
Course project for CSC 4700: Parallel C++ for Scientific Applications

Tease



Tile2D

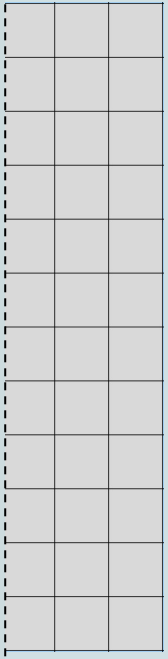
```
template <typename T>
class Tile2D{
    T& get(int x, int y);

private:
    std::vector<T> data_;
};
```



Iterators

```cpp
// Motivating example
#include "Tile2D.hpp"

Tile2D<int> t(12, 12);

std::generate(t.begin(), t.end(),
  []() { return get_rand_int(0,9)
);

sum = std::reduce(t.begin(), t.en
```

```cpp
template <typename T>
class Tile2D{
    T& get(int x, int y);
    Iterator2D<Tile2D> begin();
    Iterator2D<Tile2D> end();
};

template <typename Tile2D_t>
class Iterator2D{
    T& operator*();
    Iterator2D& operator++();
    bool operator!=(Iterator2D& other);
    /* ... */
};
```



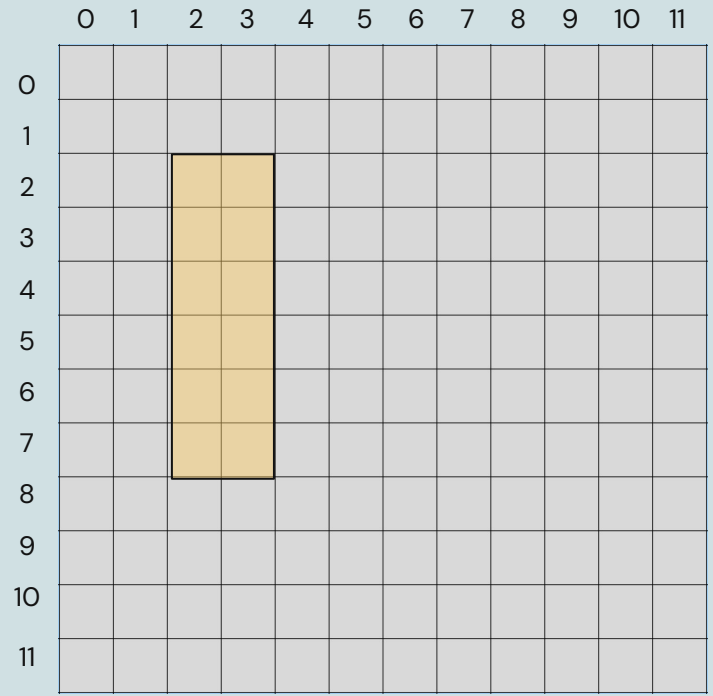
View

```
template <typename T>
class Tile2D{
    T& get(int x, int y);
    Iterator2D<Tile2D> begin();
    Iterator2D<Tile2D> end();
    View2D<Tile2D> view(int x0, int x1, int y0 , int y1);
};
```

```
template <typename Tile2D_t>
class View2D{
    T& get(int x, int y);
    Iterator2D<View2D> begin();
    Iterator2D<View2D> end();
};
```

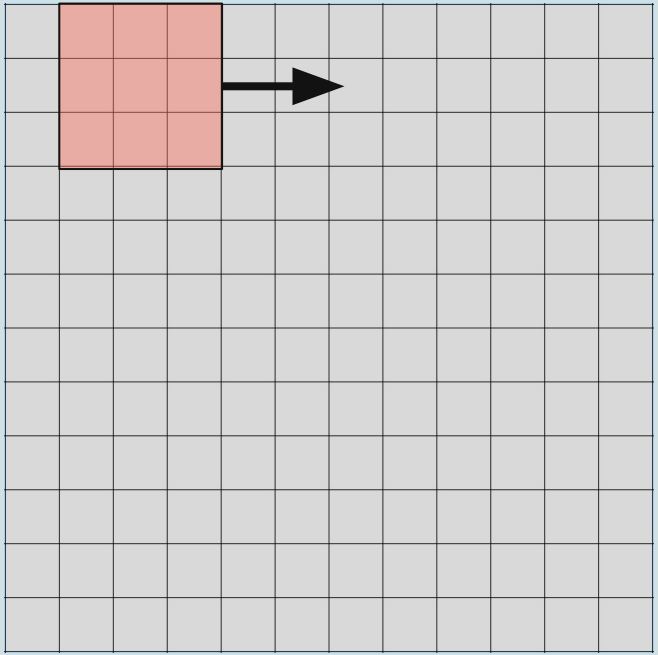
Views

```
#include "Tile2D.hpp"

Tile2D<int> t(12, 12);

/*Fill t with values*/

View2D v = t.view(2, 4, 2, 8);

std::for_each(v.begin(), v.end(),
  [](int& x) { x = x*x; }
);
```



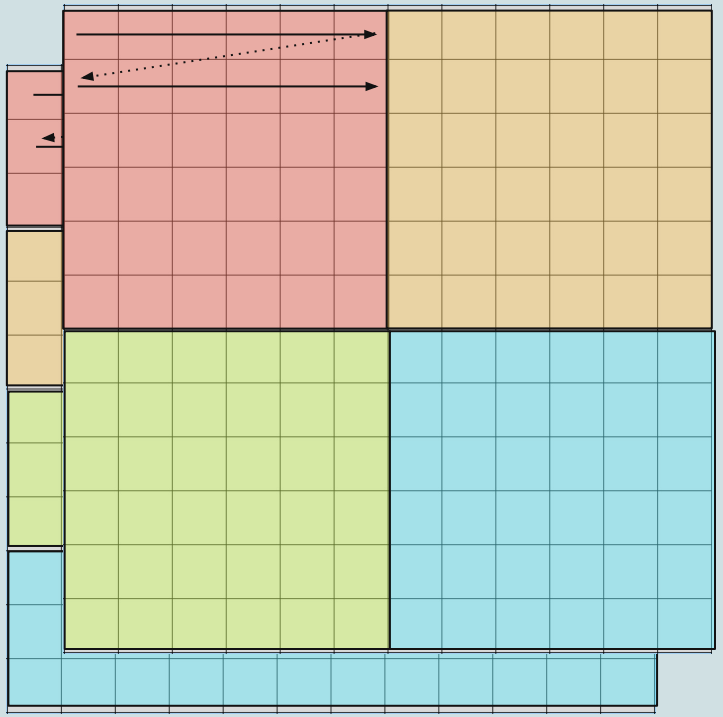
Iterator Offset & Kernels

```
template <typename Tile2D_t>
auto blur_kernel(Iterator2D<Tile2D_t>& iter){

    auto sum = *iter //center
    + iter.get(-1, 0) //left
    + iter.get(1, 0) //right
    + iter.get(0, -1) //top
    + iter.get(0, 1) //bottom

    return sum / 5.0;
}
```



Going Parallel

```
// Parallel example
#include "Tile2D.hpp"

Tile2D<int> t(12, 12);

std::generate(std::execution:par,
  t.begin(), t.end(),
  []() { return get_rand_int(0,9); }
);

sum = std::reduce(std::execution:par,
  t.begin(), t.end());
```

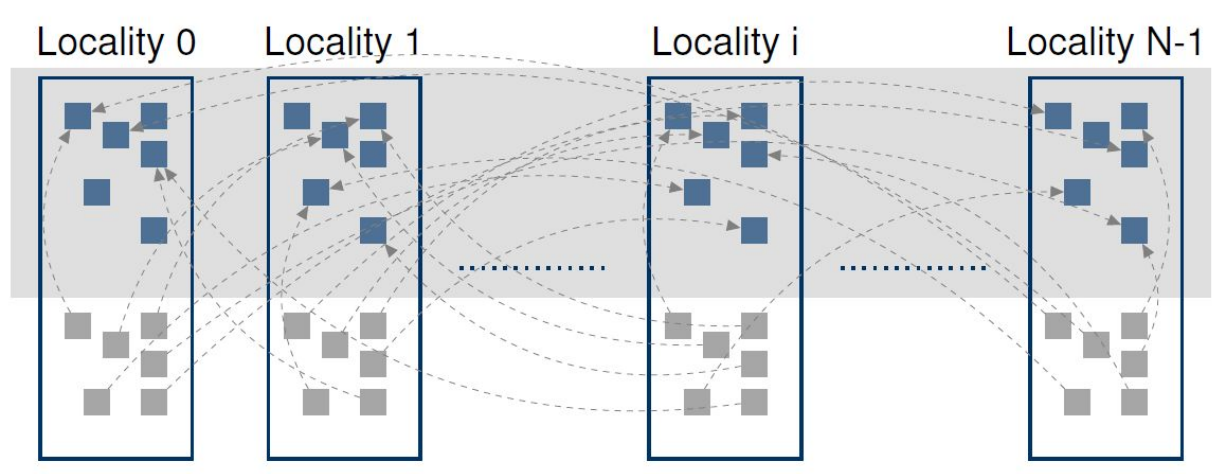


Going Distributed

HPX Components:

- Globally addressable object
- `Tile2D` component → inherits from *`hpx::components::component_base`*

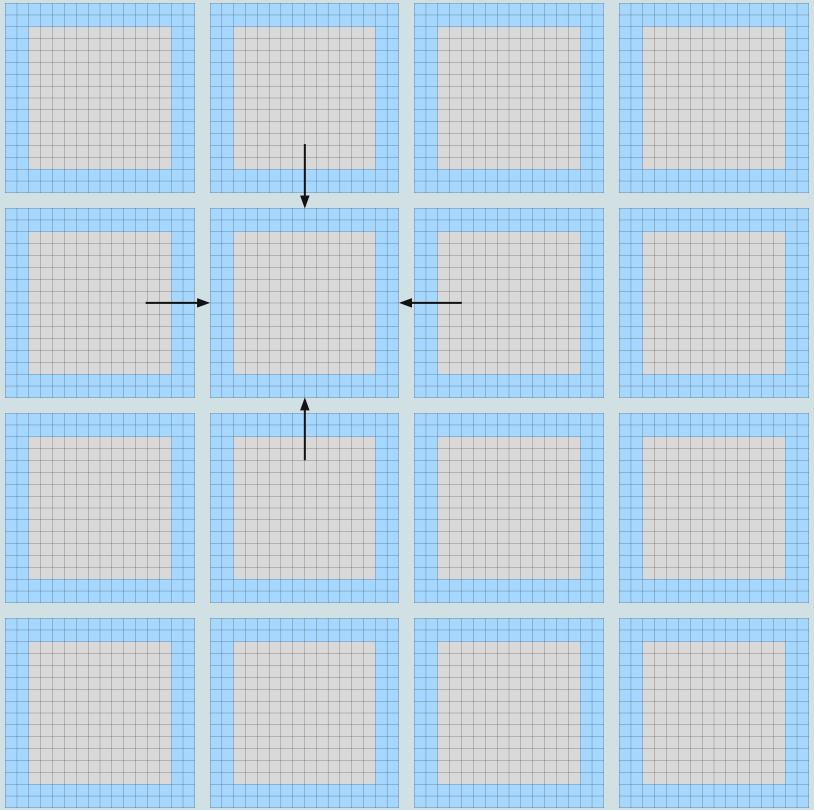
```
// Create new Tile2D at “locality”
hpx::id_type gid = hpx::new_<Tile2D<int>>(locality, 12, 12).get();
```



Distributed Space Partitioning

```
Tile2D<Tile2D<...>> world(4, 4);
```

- Partitions can be on arbitrary localities
- Ghost region → Neighbor data exchange



Fluid Simulation

Finite Element Method

Data type:

```
struct FluidElement{
    float vx, vy;          // Velocity
    float p;               // Pressure
    float m_R, m_G, m_B;   // Material (RGB)
};
```

Several passes (kernels) per time-step:

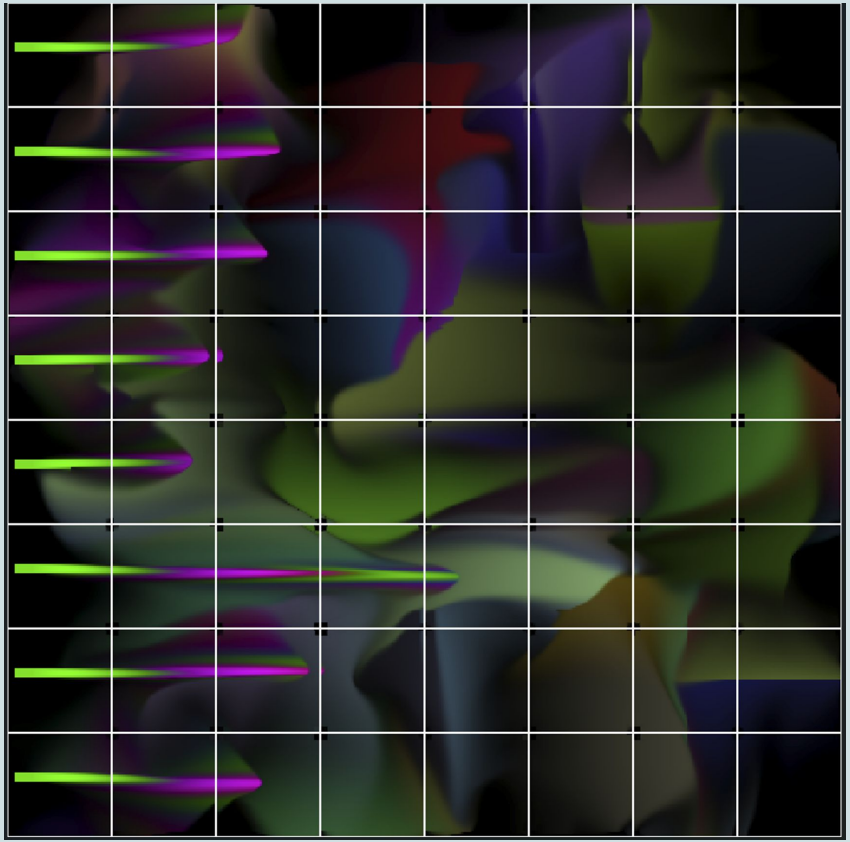
1. Advection (move based on velocity)
2. Pressure solver (incompressibility)
3. Apply pressure forces

Simulation Loop

Every time step:

- Exchange ghost cells
- Run simulation passes
- Send low-res sampling to main locality
- Output to .bmp

Python script: .bmp → Video



Thank You!