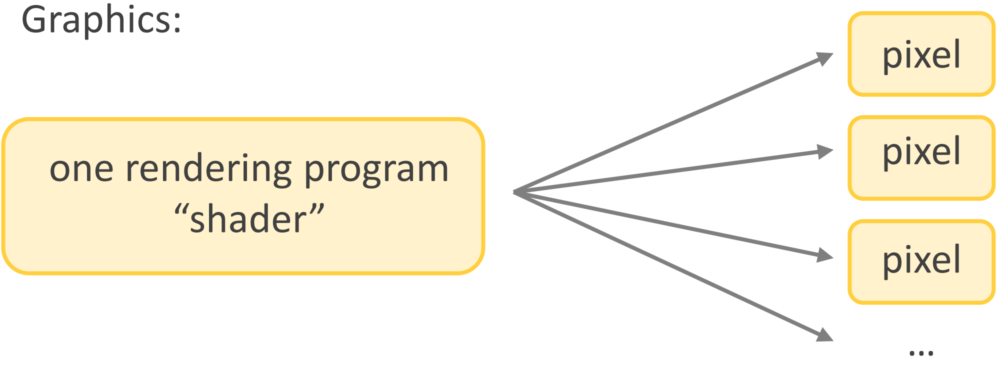
Matthias Müller, Ten Minute Physics

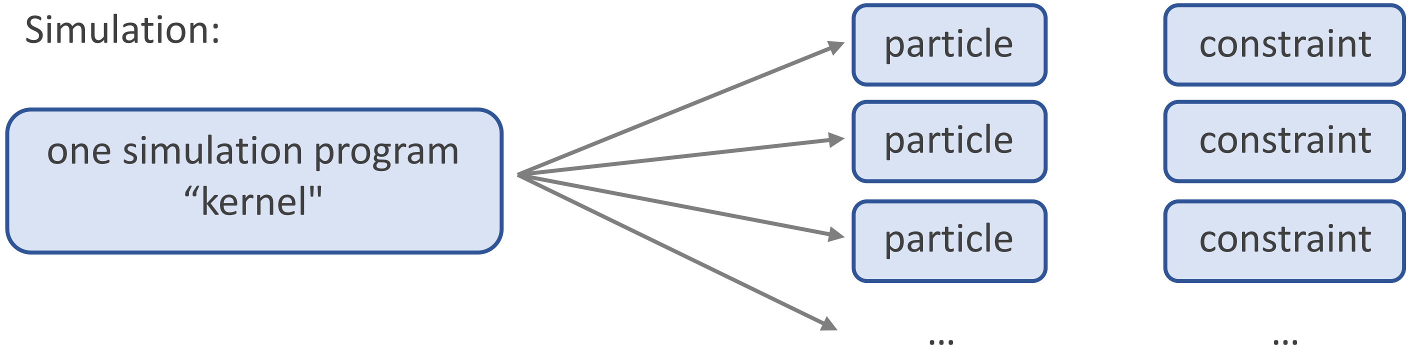
www.matthiasmueller.info/tenMinutePhysics

GPUs are Perfect for Simulations

- Designed to run one program for multiple objects
- Graphics:



- Simulation:

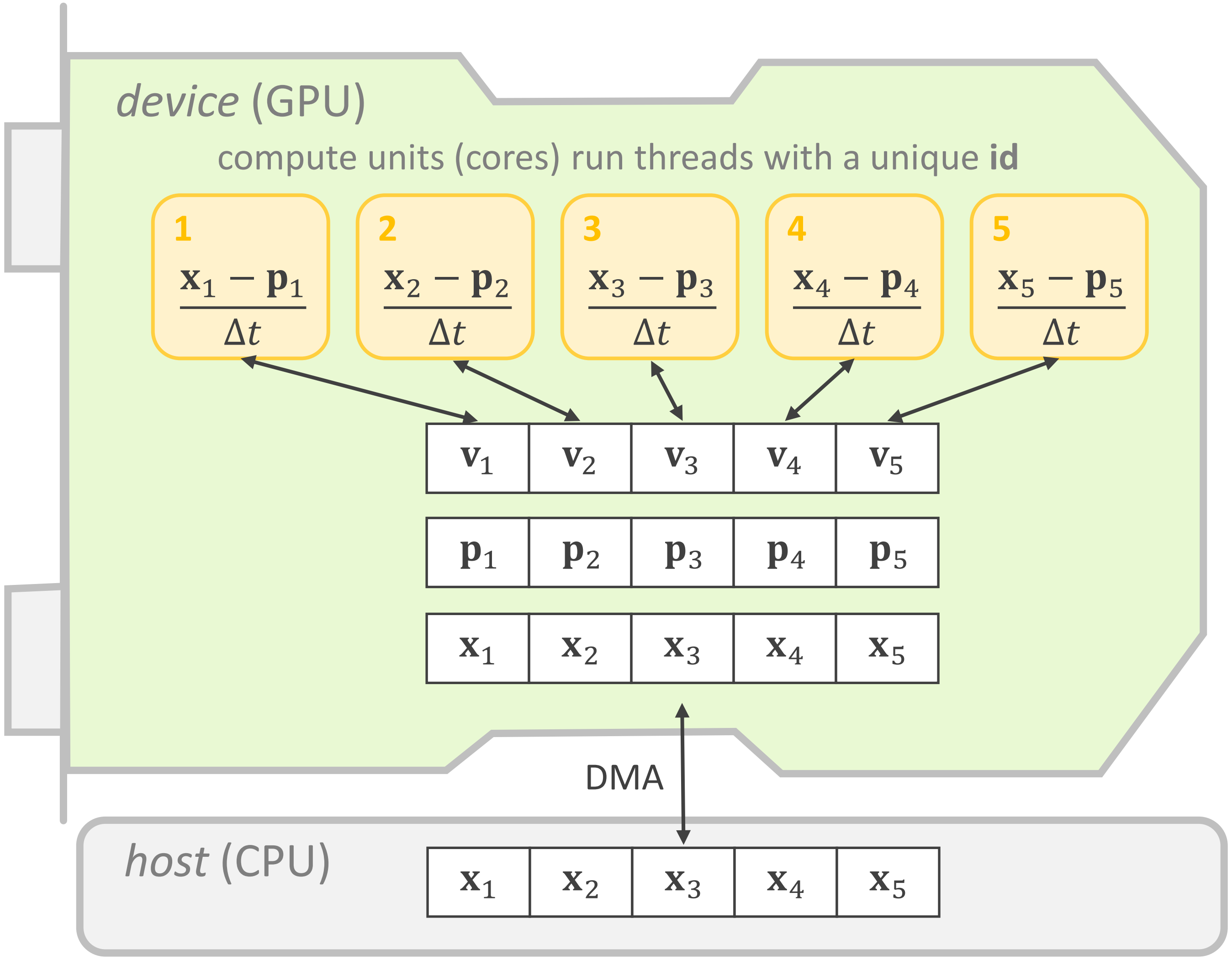


Example: PBD Velocity Update

while simulating
  for all particles i
    $\mathbf{v}_i \leftarrow \mathbf{v}_i + \Delta t\, \mathbf{g}$
    $\mathbf{p}_i \leftarrow \mathbf{x}_i$
    $\mathbf{x}_i \leftarrow \mathbf{x}_i + \Delta t\, \mathbf{v}_i$

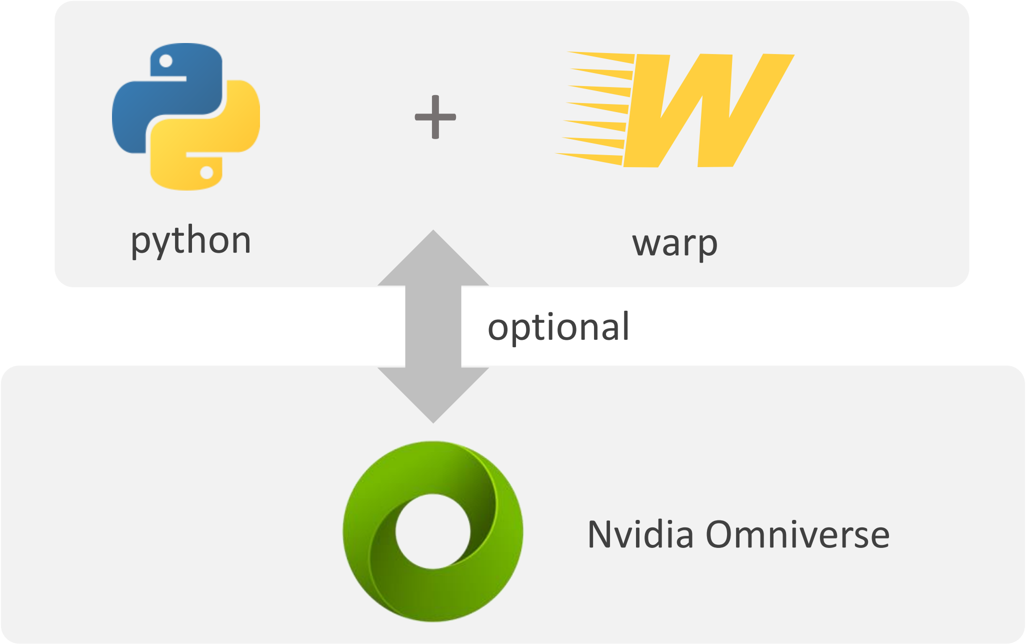
  for all constraints C
    solve$(C, \Delta t)$

  for all particles i
    $\mathbf{v}_i \leftarrow (\mathbf{x}_i - \mathbf{p}_i)/\Delta t$



Implementation

- Has never been easier: use the new nvidia *warp* python extension!



Simulation

Visualization, interaction

- developer.nvidia.com/warp-python
- github.com/NVIDIA/warp

Example: PBD Velocity Update

```
import warp as wp

self.pos =          wp.array(pos, dtype = wp.vec3, device = "cuda")
self.prevPos =      wp.array(pos, dtype = wp.vec3, device = "cuda")
self.vel =          wp.array(vel, dtype = wp.vec3, device = "cuda")
self.hostPos =      wp.array(pos, dtype = wp.vec3, device = "cpu")
```

```
@wp.kernel
def updateVel(dt:       float,
              prevPos:  wp.array(dtype = wp.vec3),
              pos:      wp.array(dtype = wp.vec3),
              vel:      wp.array(dtype = wp.vec3)):

    pNr = wp.tid()
    vel[pNr] = (pos[pNr] - prevPos[pNr]) / dt
```

```
wp.launch(kernel = updateVel,
                   inputs = [dt, self.prevPos, self.pos, self.vel], dim = self.numParticles,
                   device = "cuda")

wp.copy(self.hostPos, self.pos)
```

One-time Setup

- Setup Python and Visual Studio for editing and debugging

  code.visualstudio.com/docs/python/python-tutorial

- Install NumPy

  pip install numpy

- Install Warp

  pip install warp-lang

- Install PyOpenGl

  www.lfd.uci.edu/~gohlke/pythonlibs/

  Pyhton version → 39; OS version → 64

  download PyOpenGL_accelerate-3.1.6-cp39-cp39-win_amd64.whl

  PyOpenGL-3.1.6-cp39-cp39-win_amd64.whl

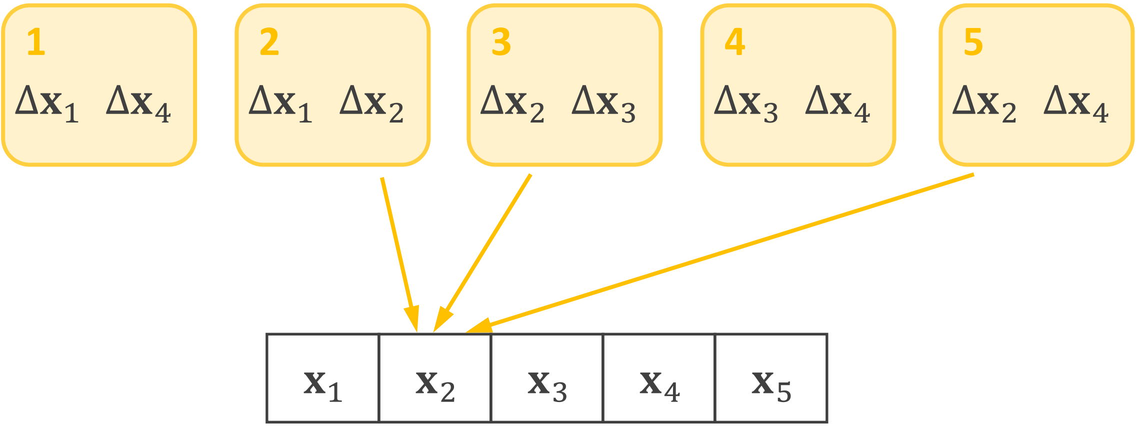
  pip install [name].whl

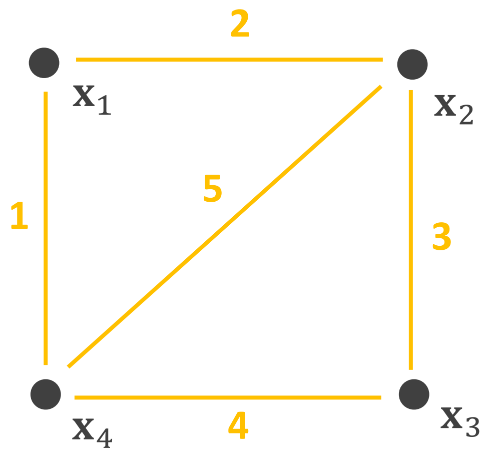
- Demo at www.matthiasmueller.info/tenMinutePhysics

- Updates in the video description below ↓

Challenge 1: Simultaneous Adds

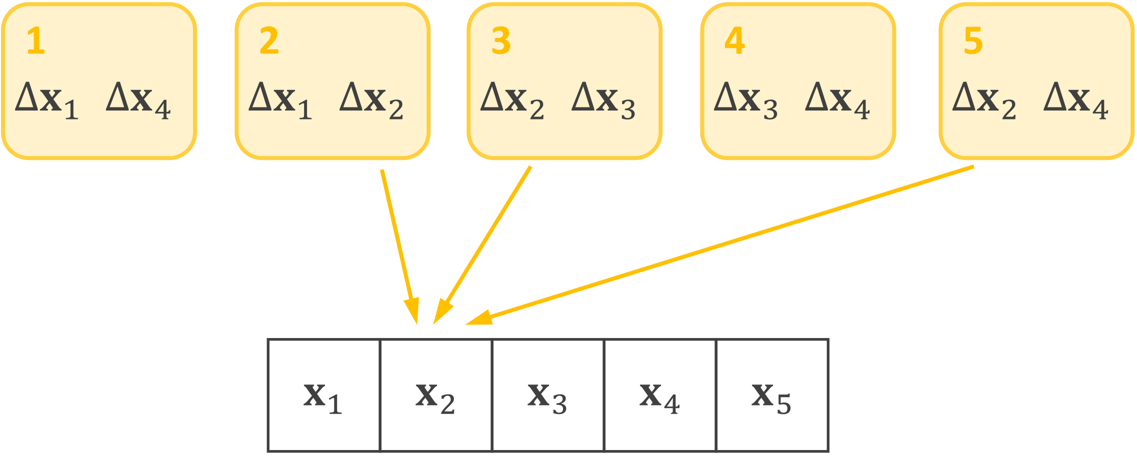
- Per particle loops: each thread writes to a separate array entry
- Constraints: multiple threads write to the same array entry!

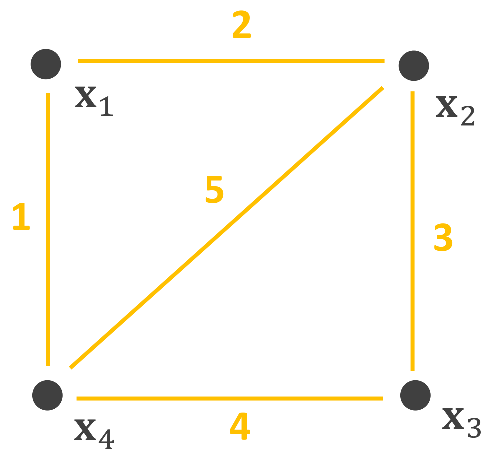




- Problem: If a thread starts adding before the addition operation of another thread is finished, the previous addition is lost!
- Use *atomic* operations! `wp.atomic_add(pos, pNr, deltaPos)`

Challenge 2: Simultaneous Read and Add





- XPBD corrections of constraint 3 depends on $\mathbf{x}_2$ and $\mathbf{x}_3$
- Different result before and after threads 2 and 5 have added their corrections
- Result is non–deterministic (depends on random thread order, jittering)
- Two solutions: Jacobi solver or graph coloring

Jacobi Solve (vs. Gauss Seidel)

for all particles i

  $\mathbf{d}_i \leftarrow \mathbf{0}$

for all constraints C

  solve$(C, \Delta t)$

for all particles i

  $\mathbf{x}_i \leftarrow \mathbf{x}_i + \mathrm{s}\,\mathbf{d}_i$

replace all

  $\mathbf{x}_i \leftarrow \mathbf{x}_i + \Delta\mathbf{x}_i$

by

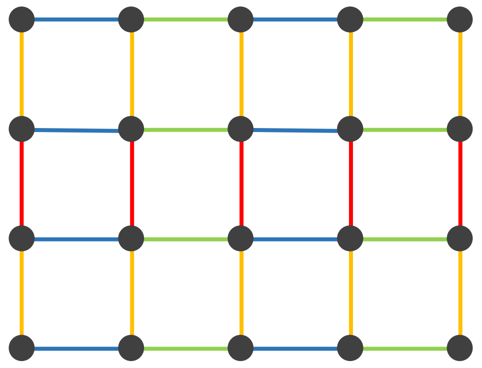
  $\mathbf{d}_i \leftarrow \mathbf{d}_i + \Delta\mathbf{x}_i$

- Pros
  - Positions and $\mathbf{x}_i$ are not changed by the threads → all threads work with the same $\mathbf{x}_i$
  - Easy to implement
- Cons
  - Slower convergence (error propagation)
  - Possible overshooting, multiply by a scalar "s"
  - Average → momentum conservation violated, strength depends on number of adjacent constraints
  - Use global magic value, e. g. $s = \frac{1}{4}$

Graph Coloring

- Idea:
  - Use multiple passes
  - Each pass processes a subset of independent constraints
  - Stable, no magic s to choose

- Regular cloth mesh (no shear resistance)



- General case?

Graph Coloring

- Mathematical problem:

  Given a graph, color all edges with as few colors as possible
  such that no pair of edges with the same color touches the same node

- Finding the optimal solution is NP-hard:

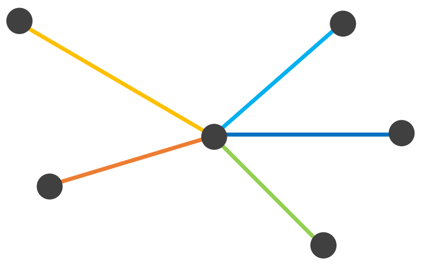
  → it is very likely that there is no better way than to test all possible colorings!

- Greedy algorithm does not find the optimal but a typically good solution

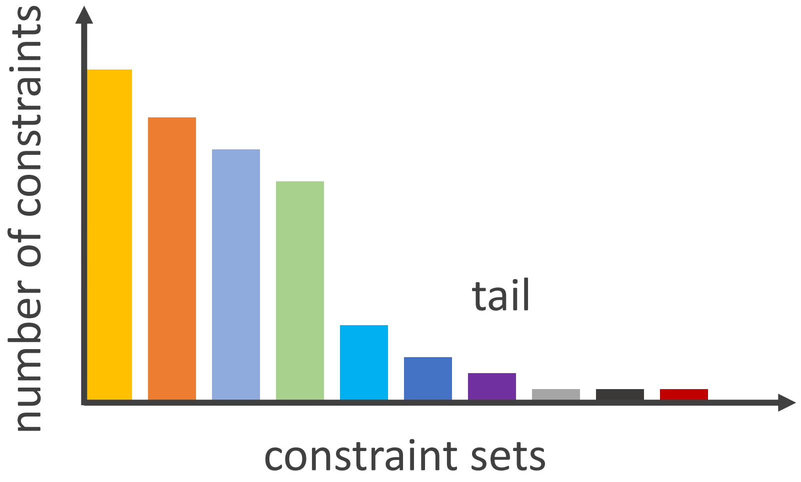
```
while there exist unmarked constraints
    create new set S
    clear all particle marks
    for all unmarked constraints C
        if no adjacent particle is marked
            add C to S
            mark C
            mark all adjacent particles
```

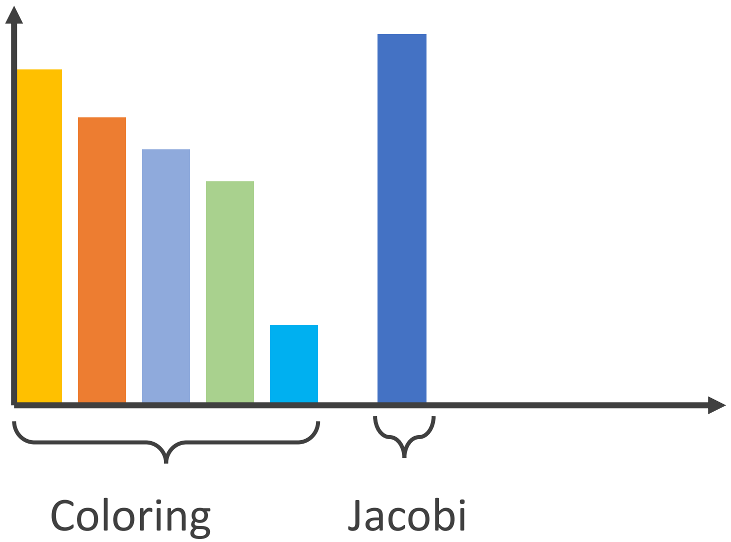
Hybrid Solution

- Often many passes are required
- At least as many sets as the maximum valence in the system!



- Typical constraint set sizes





Demo & Implementation