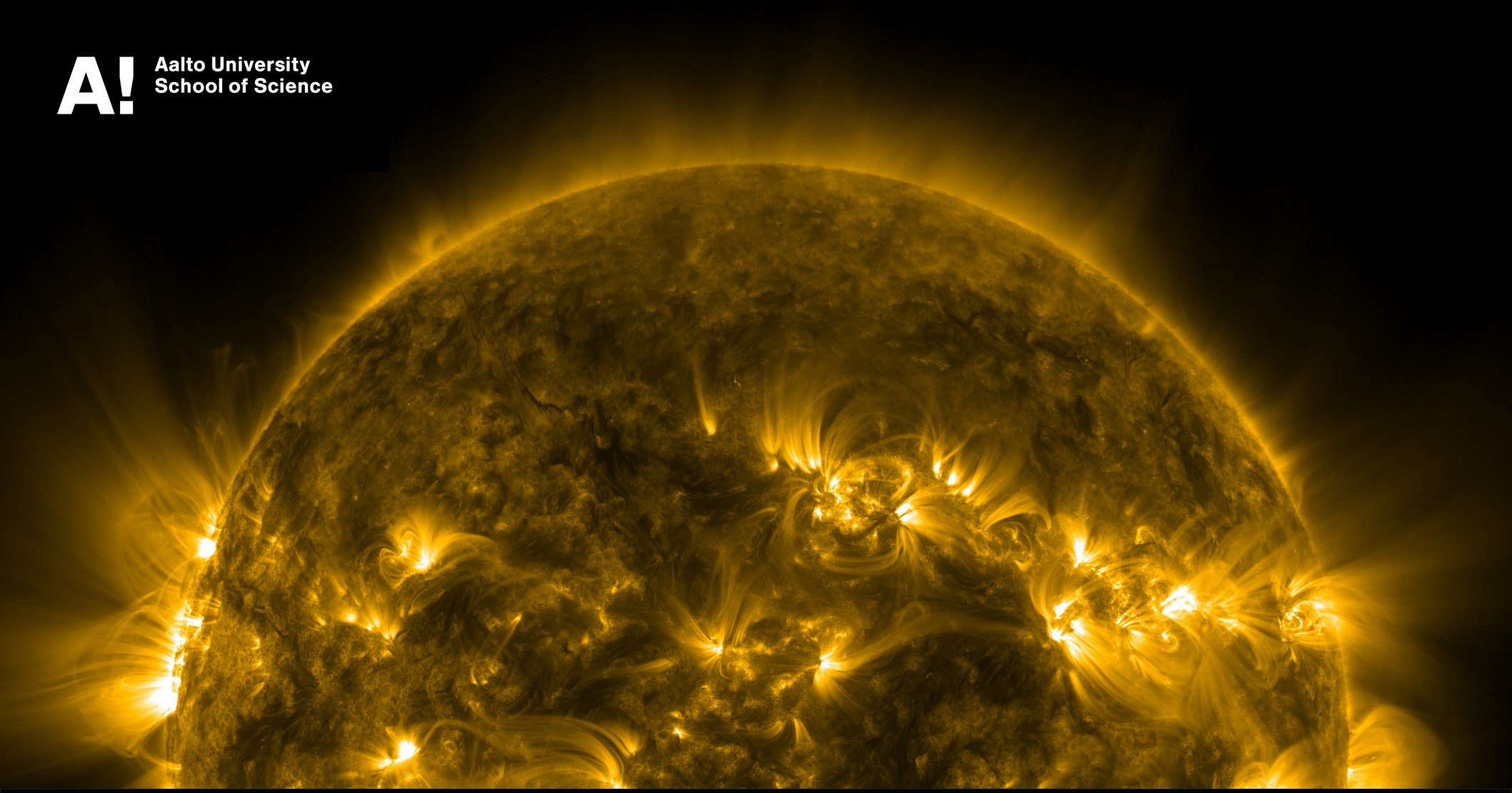
Aalto University
School of Science



Astaroth

Johannes Pekkila, PC user meeting 2020

Image credit: NASA/SDO

Astaroth

Accelerated stencil computations on GPUs

- Speaker: Johannes Pekkilä
  - Doctoral Student at Aalto University, Finland
  - Field: Computer Science (Big Data and Large-Scale Computing)
  - Worked on accelerating physical simulations on GPUs since 2014

Astaroth

Accelerated stencil computations on GPUs

- Astaroth is a multi-GPU library for high-performance stencil computations
- The API consists of an host-level interface (C, C++, Fortran) and a domain-specific language (DSL)
- Specialized in high-order stencils and computations with coupled fields → very high cache-efficiency in comparison to competitors

Astaroth

Recap

- Usable for production now
- Full MHD (hydro, magnetic, entropy) + forcing + upwinding
- All physics are implemented in the DSL
- Efficient multi-GPU MPI implementation (GPUDirect RDMA)

This talk

1) Performance

2) Astaroth DSL

3) Astaroth API

Single-node performance

Full MHD, double precision

Pencil Code on 2x Intel Xeon Gold 6230 CPUs, 40 cores:

- 23 ns / grid point / step

35x speedup

Astaroth on 4x Tesla V100 GPUs

- 0.65 ns / grid point / step

Strong scaling

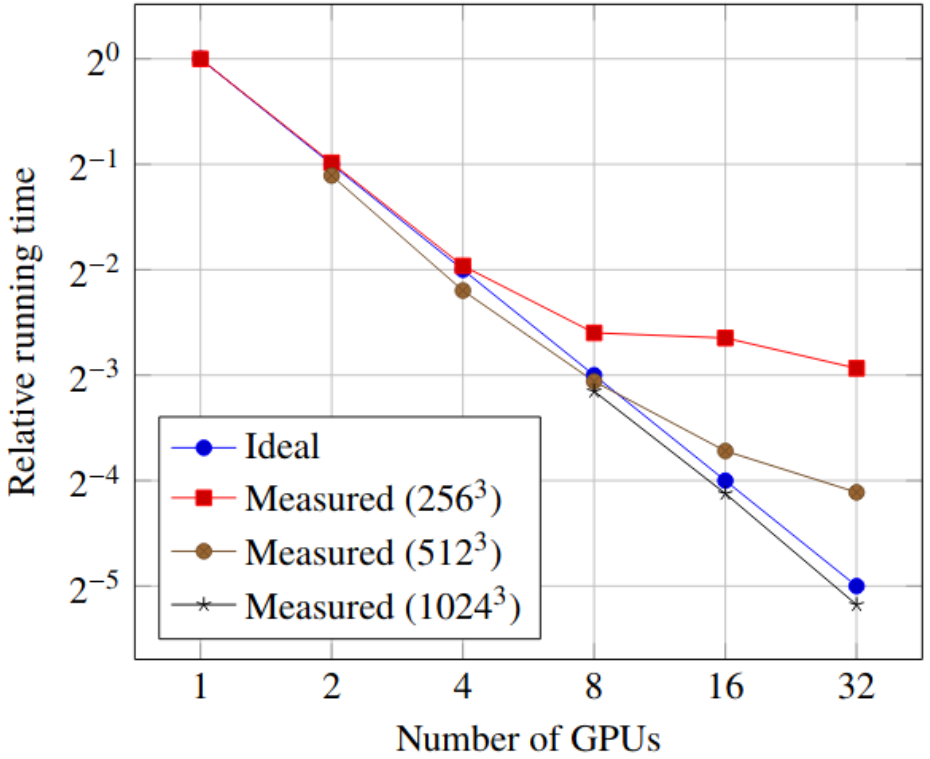


Figure 15: Strong scaling for 6th-order stencils. The largest submesh that could be held in the memory of a single GPU was approximately 448^3.

Table 8: Strong scaling for 6th-order stencils when varying mesh size. Times are in milliseconds per integration step.

nprocs	256^3	512^3	1024^3
1	41.74		
2	21.08	155.03	
4	10.70	72.80	
8	6.89	40.07	300.40
16	6.66	25.36	153.33
32	5.46	19.33	73.82

This talk

1) Performance

2) Astaroth DSL

3) Astaroth API

Astaroth DSL

Disclaimer

Astaroth DSL (AKA Astaroth Code, AC) is a procedural stream programming language

Why?

1) Industry standard
2) Easy to translate to CUDA (less bugs)
3) Stream programming maps well to GPU hardware

Astaroth DSL is NOT

1) a functional programming language.
2) the simplest possible representation for physical equations.

All major design decisions are based on obtaining maximum performance with the simplest possible language, but no simpler.

Domain-specific languages

- DSL below, optimized CUDA to the right
- CUDA code presented here is overly complex for just the heat equation, but demonstrates the optimizations required for more complex tasks

```
uniform Scalar alpha;

Vector
laplacian(in Vector T)
{
    return (Vector){laplacian(T.x), laplacian(T.y), laplacian(T.z)};
}

Vector
heat_equation(in Vector T)
{
    return alpha * laplacian(T);
}

in  VectorField T_in(X, Y, Z);
out VectorField T_out(X, Y, Z);

Kernel
solve(Scalar dt)
{
    T_out = T_in + heat_equation(T_in) * dt;
}
```

```
__constant__ float alpha;

static __device__ __forceinline__ float3
laplacian(const PreprocessedVertex3& vtx)
{
    return (float3){vtx.x.laplacian, vtx.y.laplacian, vtx.z.laplacian};
}

static __device__ __forceinline__ float3
heat_equation(const PreprocessedVertex3& T)
{
    return alpha * laplacian(T);
}

static __device__ __forceinline__ float3
stencil_process(const PreprocessedVertex3& T, const float dt)
{
    return T.value + heat_equation(T) * dt;
}


template <int step_number>
static __global__ void
__launch_bounds__(RK_THREADBLOCK_SIZE, RK_LAUNCH_BOUND_MIN_BLOCKS)
solve(const int3 start, const int3 end, const float dt,
      VertexBufferArray buffer)
{
    const int3 vertexIdx = (int3) {
        threadIdx.x + blockIdx.x * blockDim.x + start.x,
        threadIdx.y + blockIdx.y * blockDim.y + start.y,
        threadIdx.z + blockIdx.z * blockDim.z + start.z
    };

    if (vertexIdx.x >= end.x ||
        vertexIdx.y >= end.y ||
        vertexIdx.z >= end.z) {
            return;
    }

    const PreprocessedVertex3 T = stencil_assembly(vertexIdx,
                                                   buffer.in[0],
                                                   buffer.in[1],
                                                   buffer.in[2]);
    const float3 result = stencil_process(T, dt);

    buffer.out[0][IDX(vertexIdx)] = result[0];
    buffer.out[1][IDX(vertexIdx)] = result[1];
    buffer.out[2][IDX(vertexIdx)] = result[2];
}
```

Astaroth Code

- Syntax of AC is C-like and similar to shading languages
- We provide a standard DSL library for computing derivatives and commonly-used math operations (curl, laplacian, etc.)
- The standard library can be extended to support non-equidistant grids, and spherical and cylindrical coordinate systems in the future

Astaroth Code

Function type qualifiers

- **Kernel.** The main function, analogous to __global__
- **Device.** Analogous to __device__
- **Preprocessed.** Evaluated at the preprocessing stage and cached for use in *Kernel* and *Device* functions
- **<Empty>.** Helper functions for *Preprocessed* functions.

Astaroth Code

- **Function type qualifiers:** *Preprocessed*, *Kernel*, and *Device* functions
- **Data types**: *Scalar, Vector, Complex, Matrix, ScalarField, VectorField.* Precision of real numbers is determined at compile-time. Device constants are declared with *uniform* type qualifier
- **Input and output arrays** are declared with *in* and *out* qualifiers
- **Built-in variables:**

  - vertexIdx – local index of the current vertex
  - globalVertexIdx – global index of the current vertex, offset based on the multi-GPU decomposition
  - globalGridN – dimensions of the computational domain

Astaroth Code

```
#include <stdderiv.h>

uniform ScalarField FIELD_HANDLE;
uniform Scalar dt = 1.0;
uniform int flag = 0;

in  ScalarField field_in(FIELD_HANDLE);
out ScalarField field_out(FIELD_HANDLE);

Preprocessed
blur(in ScalarField field)
{
    return (field[vertexIdx] +
            field[vertexIdx.x + 1, vertexIdx.y,     vertexIdx.z] +
            field[vertexIdx.x - 1, vertexIdx.y,     vertexIdx.z] +
            field[vertexIdx.x,     vertexIdx.y + 1, vertexIdx.z] +
            field[vertexIdx.x,     vertexIdx.y - 1, vertexIdx.z] +
            field[vertexIdx.x,     vertexIdx.y,     vertexIdx.z + 1] +
            field[vertexIdx.x,     vertexIdx.y,     vertexIdx.z - 1]) / 7.0;
}

Kernel
solve()
{
    if (flag) {
        field_out = value(field_in) + derx(field_in) * dt;
    } else {
        field_out = blur(field_in);
    }
}
```

This talk

1) Performance

2) Astaroth DSL

3) Astaroth API

Astaroth API

- Astaroth library handles host-side functionality
  - Memory operations (initialization, transfers)
  - Queuing kernels
- Essentially all API functions are asynchronous by default
  - Required for efficient multi-GPU communication

Astaroth API

The API is divided into two layers: Device and Node

- **Device layer**
  - For controlling a single GPU
  - Functions start with acDevice
- **Node layer**
  - Abstracts the GPUs available on a node behind a single interface
  - Functions start with acNode

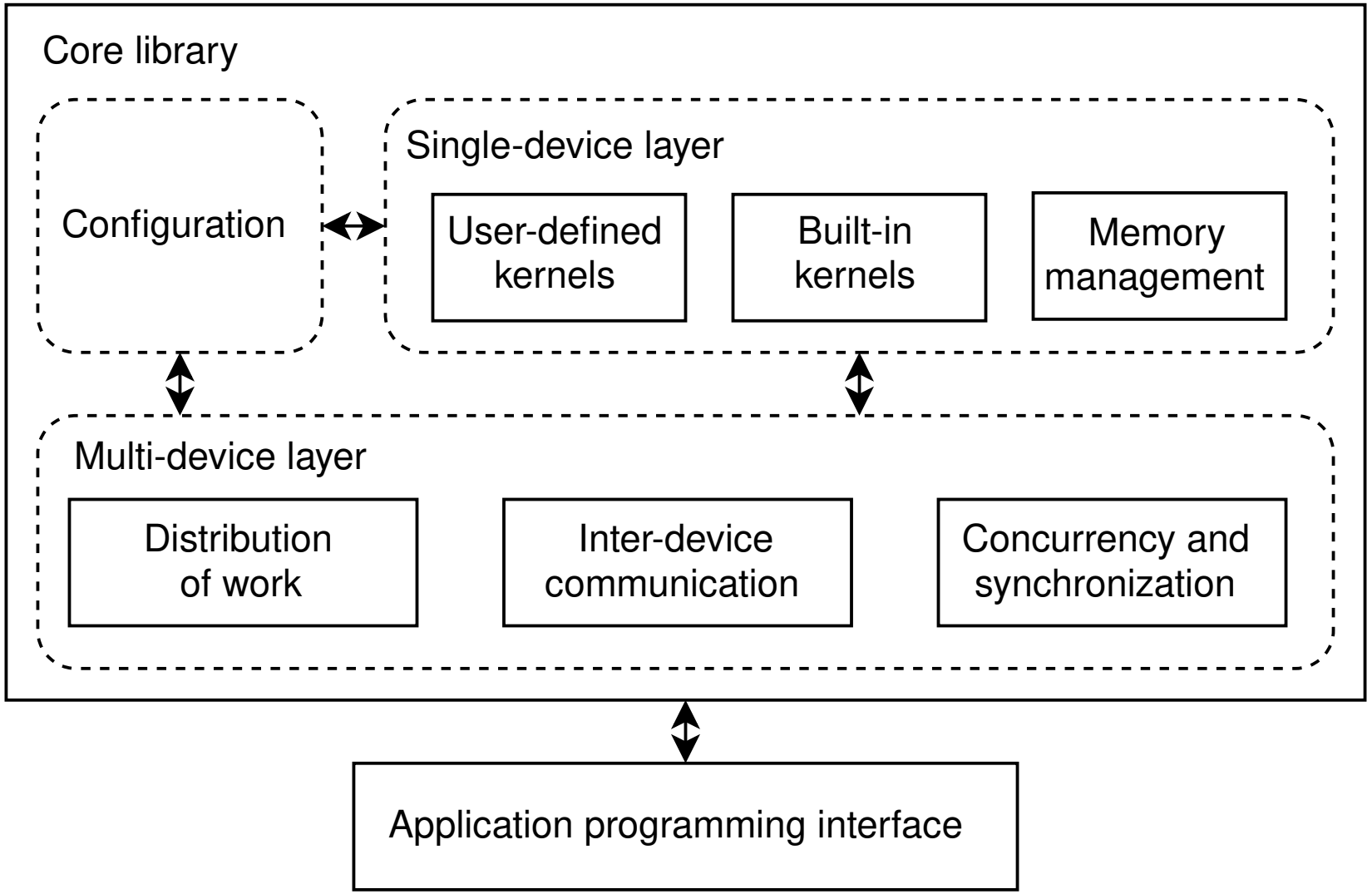
Astaroth API

- The library is configured by passing a config struct when initializing devices and nodes
- Devices and nodes are managed with handles

```
Device device;
acDeviceInit(info, &device);
acDeviceLoad(mesh, device);
acDeviceIntegrate(device);
```

```
Node node;
acNodeInit(info, &node);
acNodeLoad(mesh, node);
acIntegrate(node);
```

Astaroth API



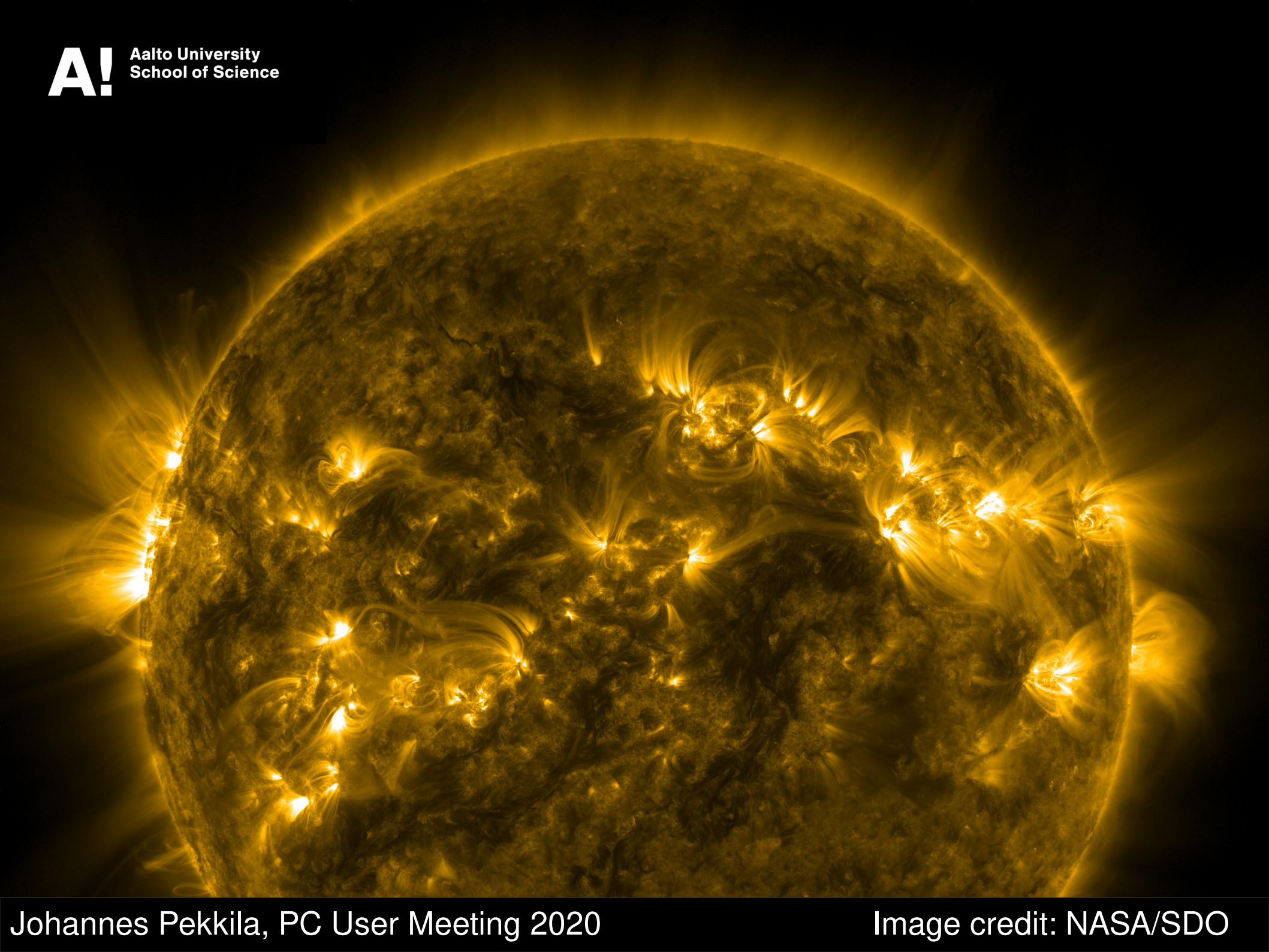
Astaroth

Accelerated stencil computations on GPUs

Code
http://bitbucket.org/jpekkila/astaroth

Documentation
http://libastaroth.bitbucket.io/

Thesis
http://urn.fi/URN:NBN:fi:aalto-201906233993


Aalto University
School of Science
Johannes Pekkila, PC User Meeting 2020
Image credit: NASA/SDO

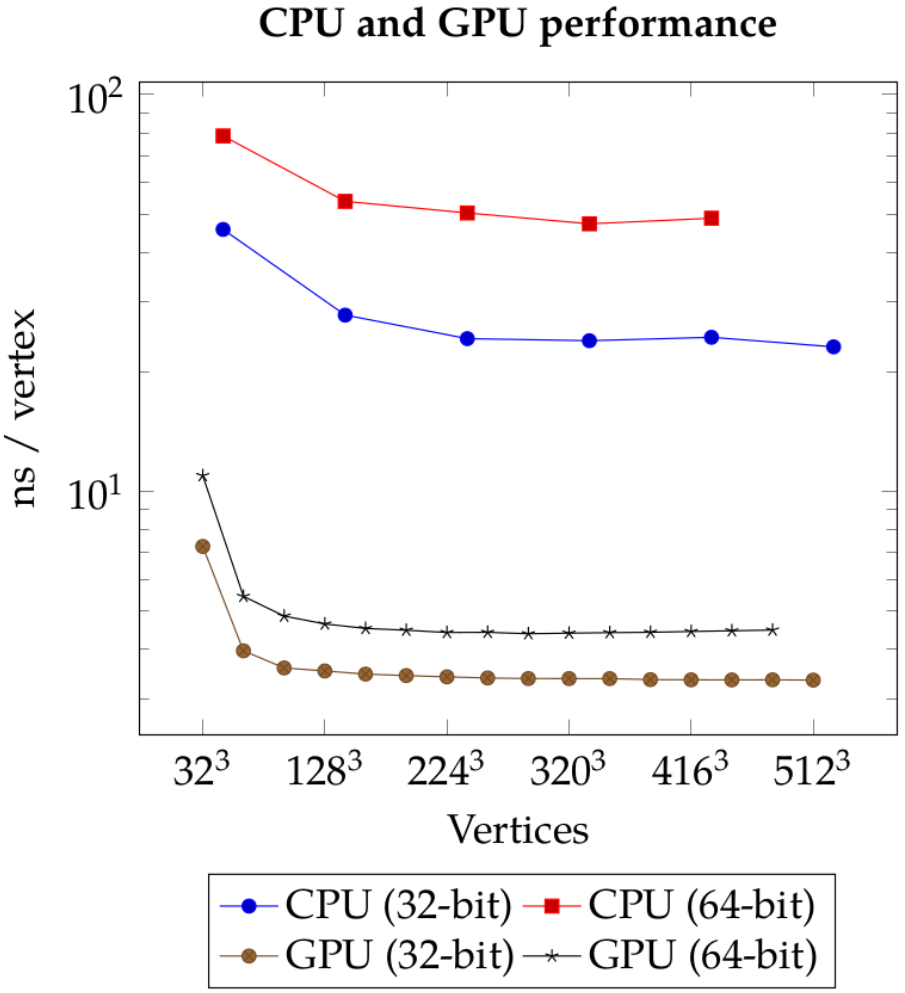
Backup slides

Single-GPU performance

Test case:
- Full MHD
- 6th order finite-differences
- 3rd order Runge-Kutta

CPU solver:
- Pencil Code
- 2x Intel Xeon E5-2690 v3
- Benchmarked on a total of 24 cores
- 47 ns / grid point / step (dbl)

GPU solver:
- Astaroth
- Tesla P100 PCIe
- 4.6 ns / grid point / step (dbl)

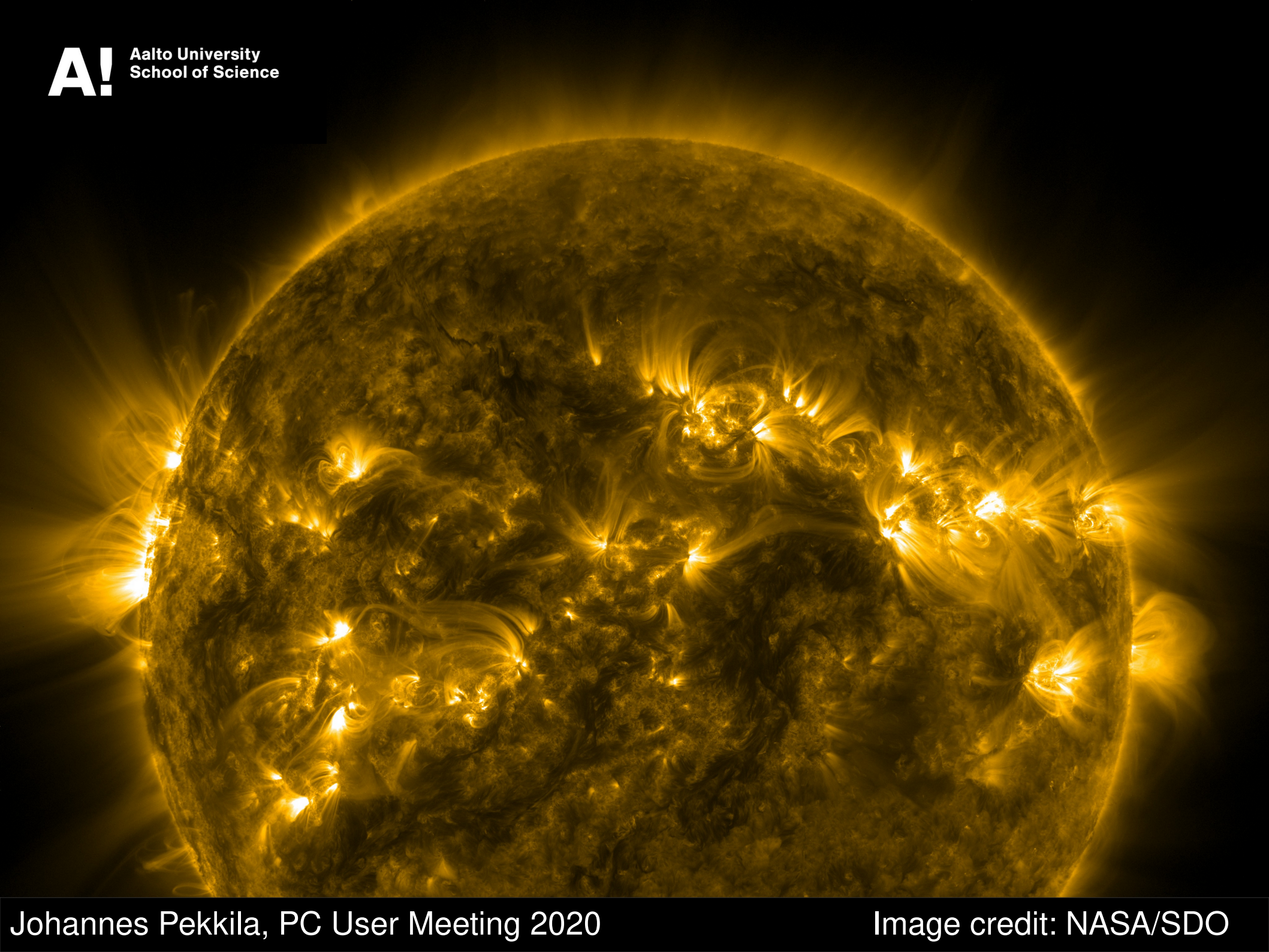


Domain-specific languages

Performance and productivity

Takeaways:

- Future is parallel
- All problems will be eventually bound by memory bandwidth (for the foreseeable future)
- The only way forward is to increase arithmetic intensity (primarily by caching)
- Domain-specific languages can offer high productivity and performance


Aalto University
School of Science
Johannes Pekkila, PC User Meeting 2020
Image credit: NASA/SDO