



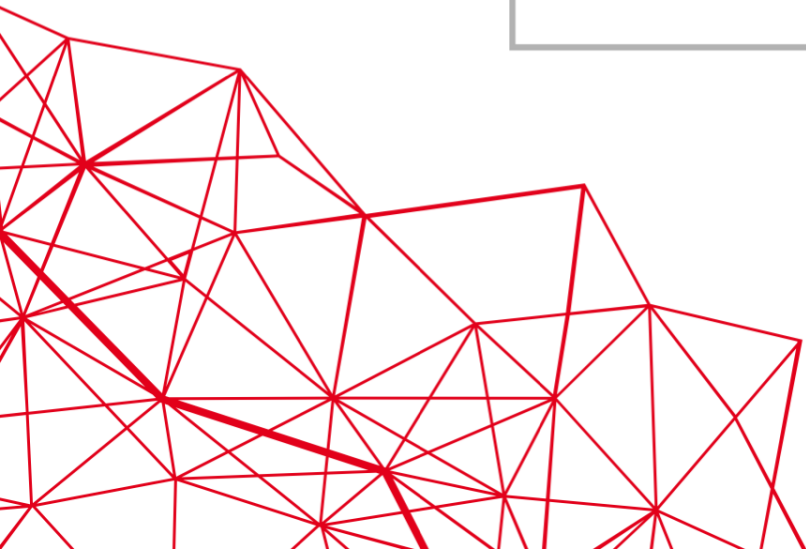
**ISC**

High Performance

**IMAGINE**

**TOMORROW**

**MAY 21 – 25, 2023 | HAMBURG, GERMANY**



# Efficient GPU Offloading with OpenMP for a Hyperbolic Finite Volume Solver on Dynamically Adaptive Meshes

Mario Wille<sup>1</sup>, Tobias Weinzierl<sup>2</sup>, Gonzalo Brito Gadeshi<sup>3</sup>, Michael Bader<sup>1</sup>

<sup>1</sup>TUM School of Computation, Information and Technology  
Technical University of Munich, Garching, Germany

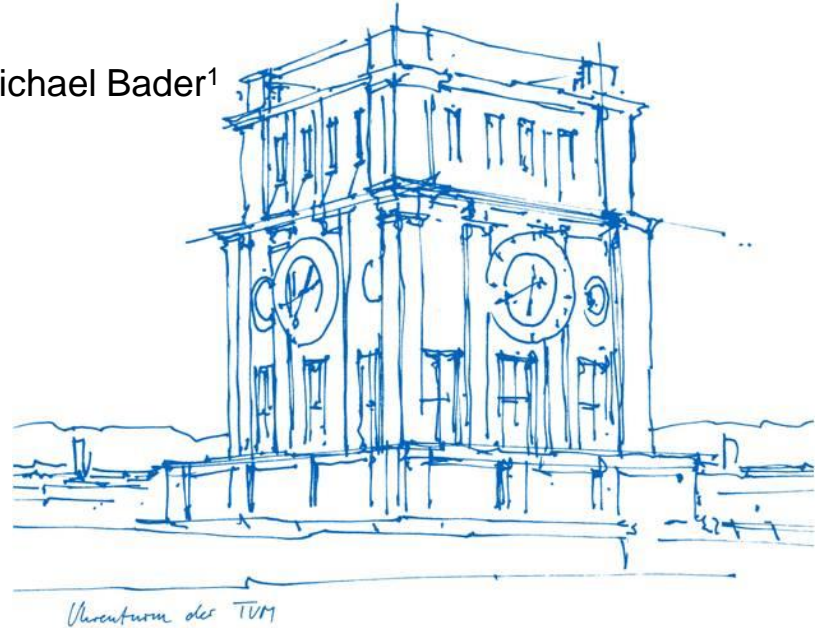
<sup>2</sup>Department of Computer Science  
Institute for Data Science – Large-scale Computing  
Durham University, Durham, UK

<sup>3</sup>NVIDIA, Munich, Germany

**ISC High Performance 2023**

May 21-25, 2023

Hamburg, Germany

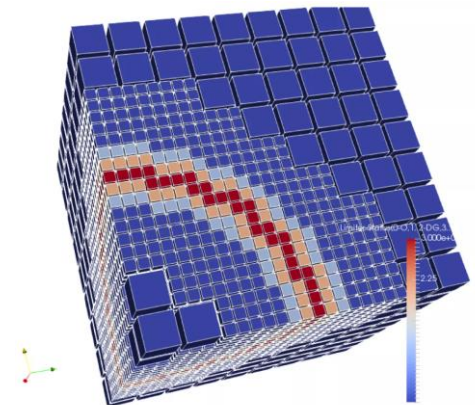
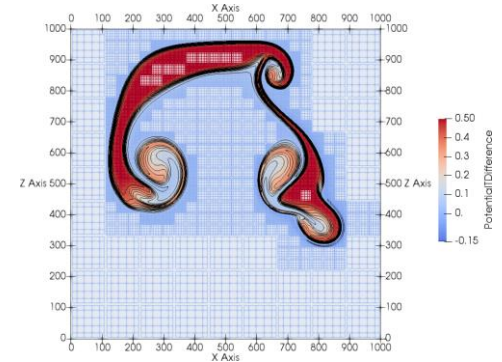


# Motivation and Project Overview

- ExaHyPE: A wave equation solver with explicit time stepping

$$\frac{\partial Q}{\partial t} + \nabla \cdot F(Q) + \sum_{i=1}^d B_i(Q) \frac{\partial Q}{\partial x_i} = S(Q)$$

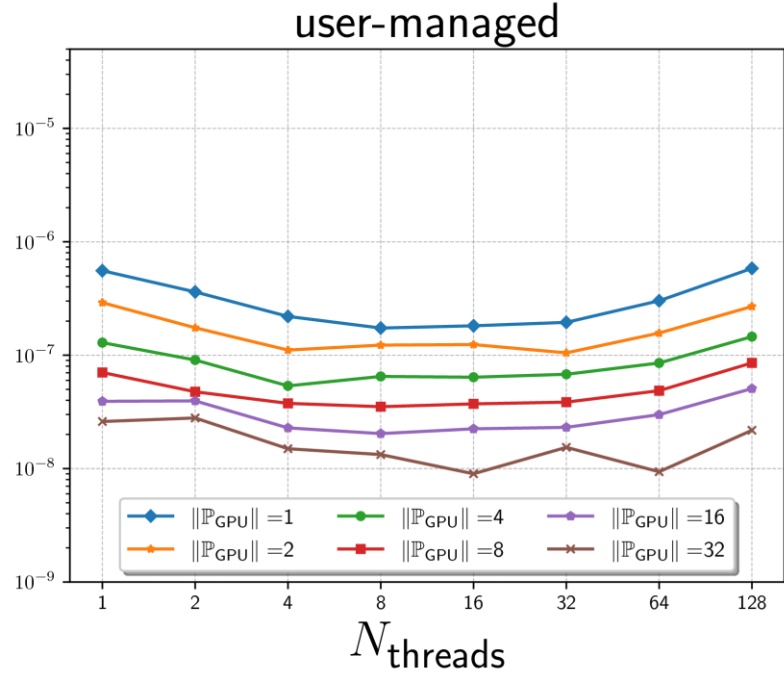
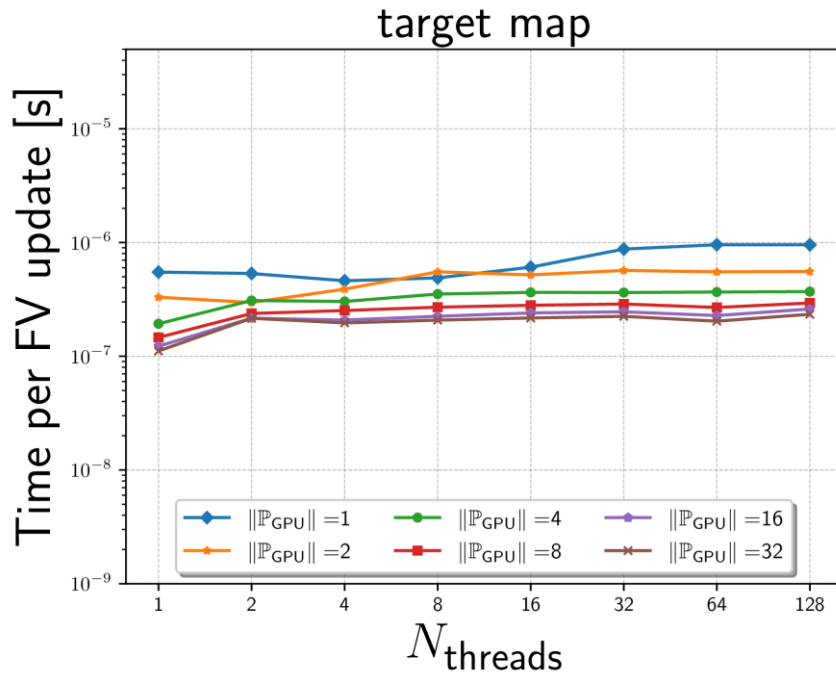
- Peano: Dynamically adaptive mesh refinement
- Patches of cells constructed through octree-type AMR
- MPI+X paradigm: Many ranks, each hosting several threads
- Each thread offloads Finite Volume patches to the GPU
- Gather multiple patches into batch of update tasks
- Handle tasks in one rush on the GPU via one kernel call
- Realisation by OpenMP's *target* constructs



# OpenMP Runtime Overhead with *target map*

- We do not constrain which core can access a GPU → multiple cores may hit the GPU simultaneously
- OpenMP runtime **locks** access to the GPU to prevent race conditions
- GPU can only read from page-locked host-pinned memory → requires **staging**
- **We identify** significant **overhead** and **synchronisation**
- Challenges:
  - Overlap computation and data transfer
  - **Avoid data allocations/frees during computations**

# Overcoming the OpenMP Runtime Overhead



## Realisation with *target map*

```
1 Procedure offload_map( $\|P_{GPU}\|$ , host_patch_data): // Data per patch are stored in one large array of structures (AoS)
2 mapped_pointers  $\leftarrow$  allocate_host( $\|P_{GPU}\|$ )
3 for i  $\leftarrow$  0 to  $\|P_{GPU}\|$  do
4     patch_data  $\leftarrow$  host_patch_data[i]
5     #pragma omp target enter data map(to:patch_data) // Map each patch's data onto the device
6     mapped_pointers[i]  $\leftarrow$  omp_get_mapped_ptr(patch_data) // Construct the list of pointers on the GPU
7 end

8 #pragma omp target teams distribute map(to:mapped_pointers)
9 for i  $\leftarrow$  0 to  $\|P_{GPU}\|$  do
10     // Do computations on Finite Volumes
11 end

12 for i  $\leftarrow$  0 to  $\|P_{GPU}\|$  do
13     patch_data  $\leftarrow$  host_patch_data[i]
14     #pragma omp target exit data map(from:patch_data) // Copy back the GPU outcomes to the host
15 end
16 mapped_pointers  $\leftarrow$  free_host()
```

# Our Proposal: A GPU Memory Manager

- Each rank holds one instance of a thread-safe GPU memory manager
- **Reserves** memory on the GPU on demand
- Hands out **pre-reserved** memory to threads and reuses the memory
- Memory ownership resides on **host**
- Synchronisation and coordination of memory can be handled on the **host**
- **Avoid locking** by the OpenMP runtime
- **Allocation** routine returns a device pointer
- **Free** routine **releases** the device memory
- **Share** pre-allocated data between threads

# Realisation with our GPU Memory Manager

```
1 Procedure offload_managed( $\|P_{GPU}\|$ , host_patch_data) :
2 patch_data  $\leftarrow$  GPUManager $\rightarrow$ allocate_device( $\|P_{GPU}\|$ ) // Allocate GPU data and store the
   memory address
3 patch_data  $\leftarrow$  omp_target_memcpy(host_patch_data,  $\|P_{GPU}\|$ )

4 #pragma omp target teams distribute is_device_ptr(patch_data)
5 for i  $\leftarrow$  0 to  $\|P_{GPU}\|$  do
6     // Do computations on Finite Volumes
7 end

8 host_patch_data  $\leftarrow$  omp_target_memcpy(patch_data,  $\|P_{GPU}\|$ )
9 patch_data  $\leftarrow$  GPUManager $\rightarrow$ free() // Release memory handle for possible reuse
```



# Loop Collapsing and Reordering

```
1 Procedure collapse(offset, N):
2 #pragma omp target teams distribute parallel for simd collapse(3)
5 for i ← 0 to N do
6   for j ← 0 to N do
7     for k ← -offset to N+offset do
8       volume_index ← get_volume_index(i, j, k)
9     end
10  end
11 end
```

# Loop Collapsing and Reordering

```
1 Procedure collapse(offset, N):  
2 #pragma omp target teams distribute parallel for simd collapse(3)  
5 for i ← 0 to N do  
6   for j ← 0 to N do  
7     for k ← 0 to N+2·offset do  
8       volume_index ← get_volume_index(i, j, k-offset)  
9     end  
10  end  
11 end
```

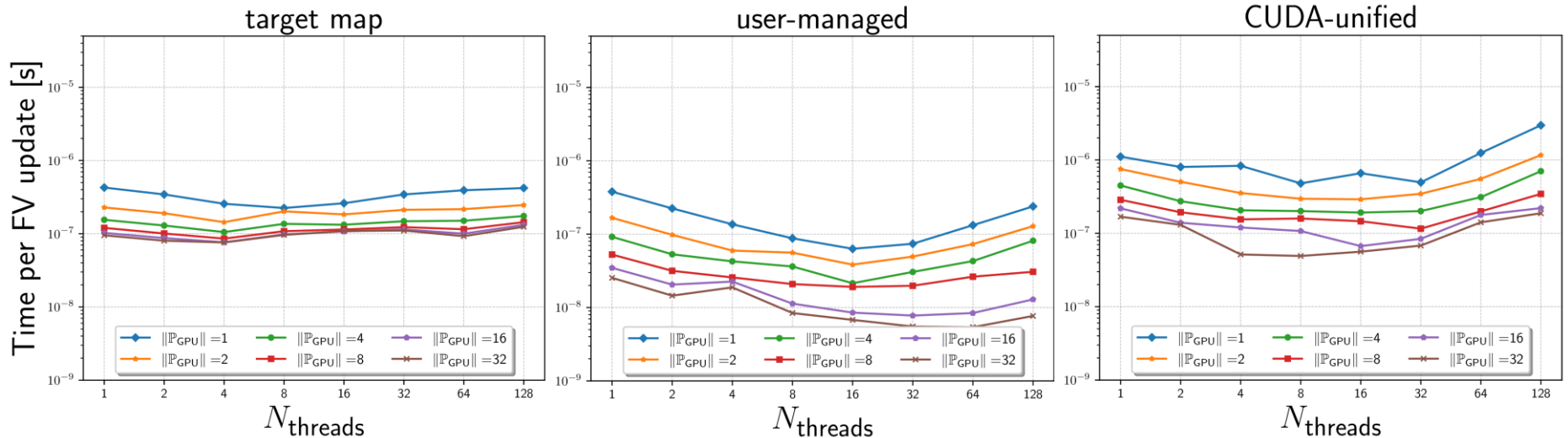
# Two Flavours of Equations

Euler Equations	Conformal and Covariant Z4 (CCZ4) Equations
<ul style="list-style-type: none"> <li>• Describes the evolution of density, energy and velocity</li> <li>• System of <math>N = d + 2</math> non-linear partial differential equations</li> <li>• <math>d = 2</math> or <math>3</math></li> <li>• Low arithmetic intensity</li> </ul>	<ul style="list-style-type: none"> <li>• Describes the evolution of space-time curvature</li> <li>• Models gravitational waves</li> <li>• System of <math>N = 59</math> equations</li> <li>• <math>d = 3</math> only</li> <li>• High arithmetic intensity</li> </ul>
<p style="text-align: center;">Solution via Finite Volumes with a generic Rusanov Riemann solver</p>	

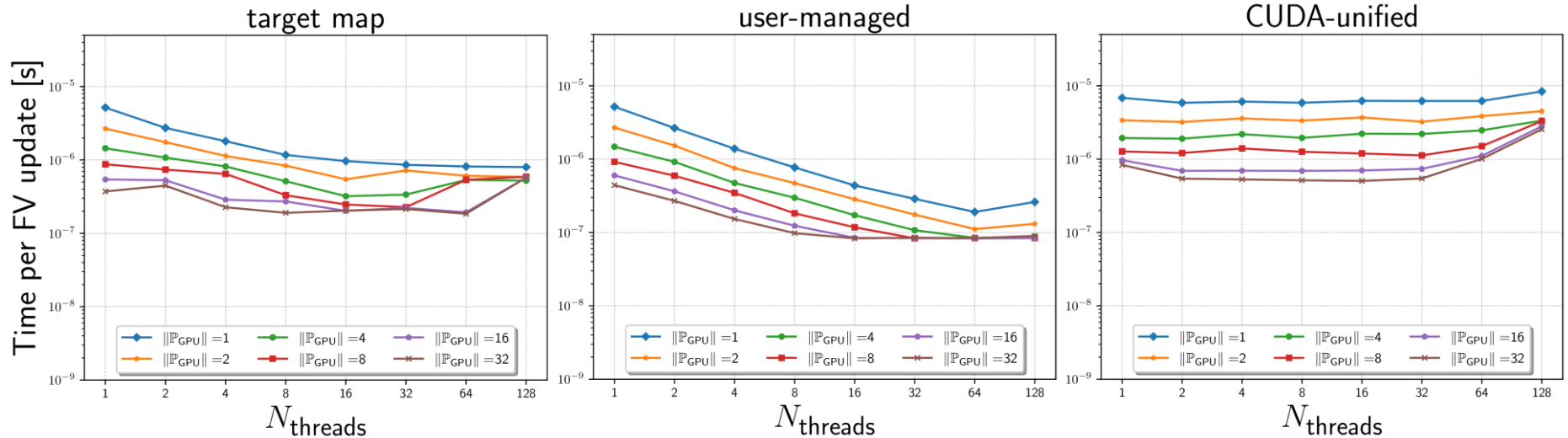
# Benchmark Systems

<b>Alex cluster: Erlangen National High Performance Computing Center (NHR@FAU)</b>	<b>JURECA DC Evaluation Platform: Jülich Supercomputing Centre (JSC)</b>
<ul style="list-style-type: none"> <li>• AlmaLinux 8.7 (Stone Smilodon)</li> <li>• 2 x AMD EPYC 7713 Milan CPUs (64 cores per chip)</li> <li>• 8 x NVIDIA A100 GPUs (80 GB memory)</li> <li>• NVIDIA HPC SDK (v23.1)</li> <li>• CUDA (v12.0)</li> </ul>	<ul style="list-style-type: none"> <li>• Rocky Linux 8.7 (GreenObsidian)</li> <li>• 2 x AMD EPYC 7443 Milan CPUs (24 cores per chip and SMT-2)</li> <li>• 4 x AMD Instinct MI250 GPUs (128 GB memory, MCM)</li> <li>• ROCm AOMP (v17.0-0)</li> </ul>

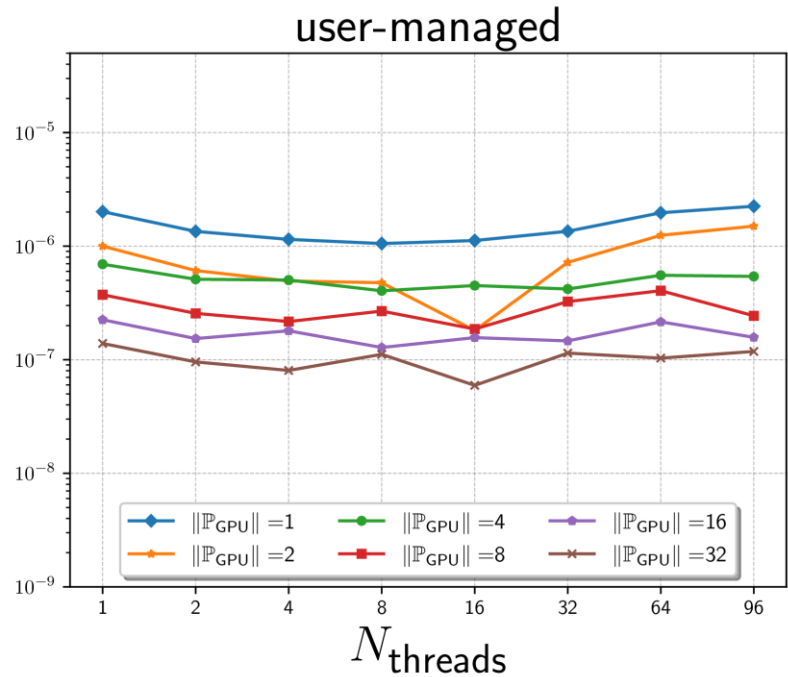
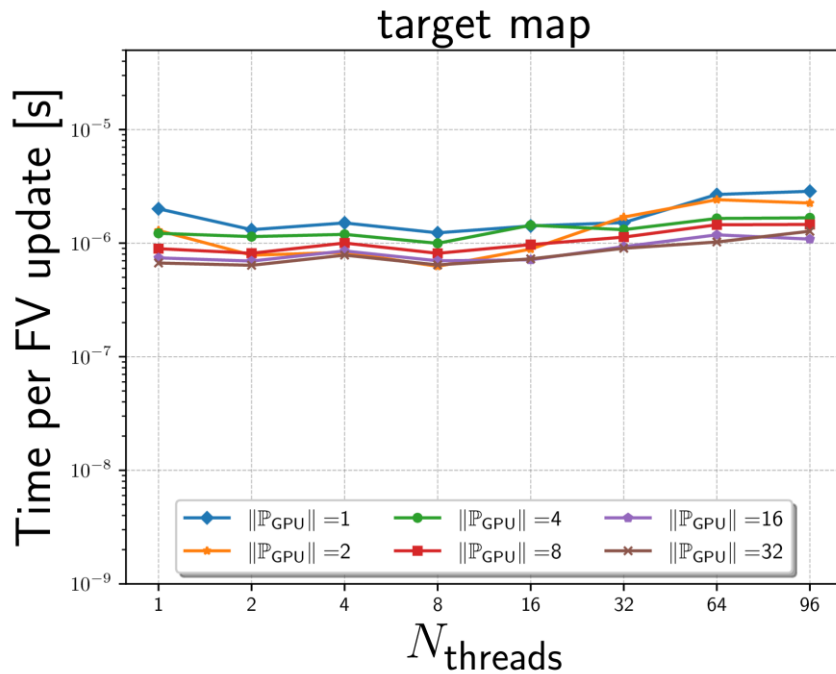
# Performance Results – Euler Equations (NVIDIA)



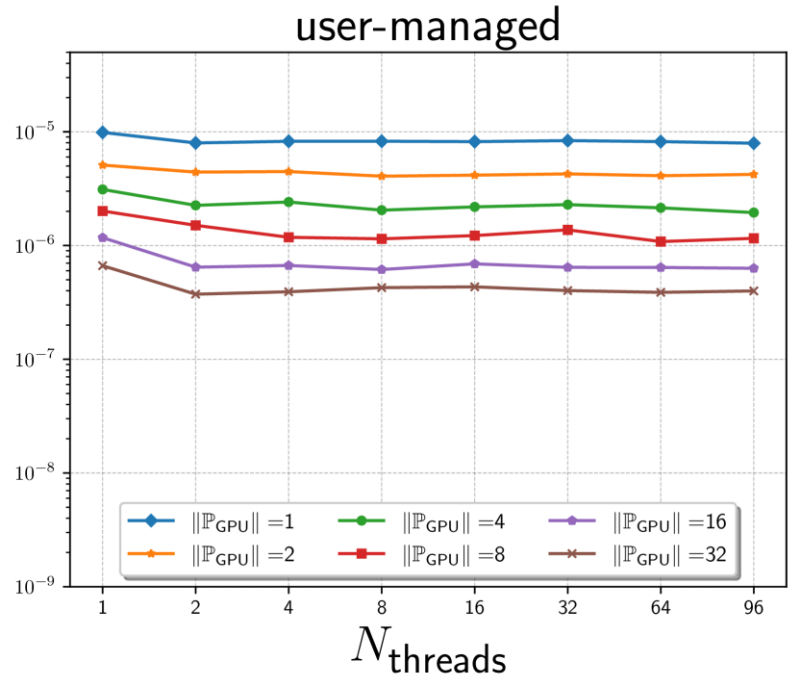
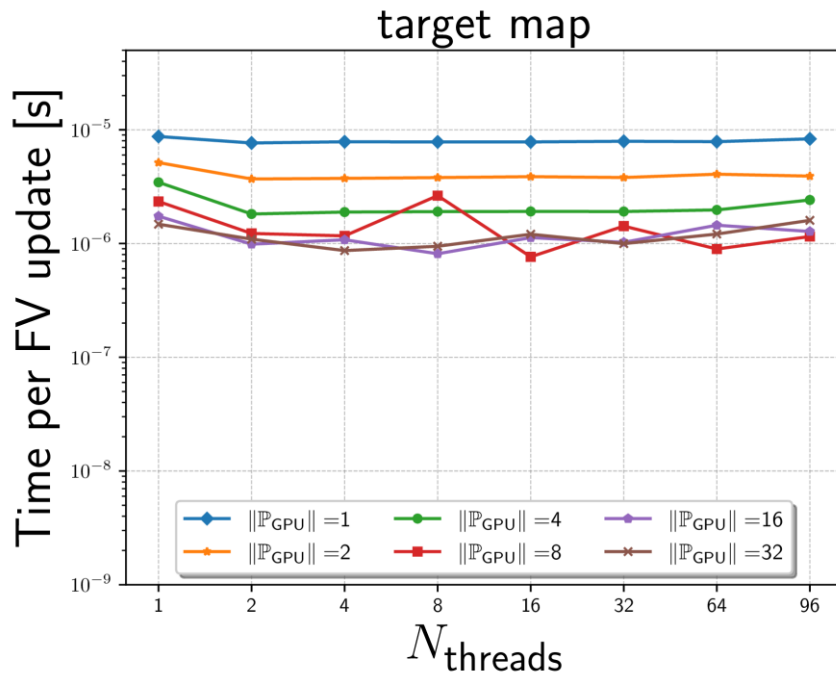
# Performance Results – CCZ4 Equations (NVIDIA)



# Performance Results – Euler Equations (AMD)



# Performance Results – CCZ4 Equations (AMD)





# Conclusion

- GPU Memory Manager withdraws memory management from the accelerator and assigns it to the host
- Host-centric realisation leads to a speedup of some calculations by an order of magnitude
- Increases robustness of concurrent offloading of patches
- No need for a static offloading pattern of patches or huge patches
- Presented a worst-case scenario motivated by a real-world science case

# Acknowledgements

- This research has been supported by EPSRC's ExCALIBUR programme (projects **EX20-9**, **PAX-HPC** and **MGHyPE**), by the German Ministry of Education and Research (**BMBF**, project **targetDART**) and by **Intel's Academic Centre of Excellence**.
- Supercomputing resources and support was provided by the ARCHER2 UK National Supercomputing Service, the Erlangen National High Performance Computing Center, Jülich Supercomputing Center and CINECA.
- Special thanks goes to Han Zhang und Baojiu Li (<https://www.icc.dur.ac.uk/>) for providing us with the CCZ4 science case.

SPONSORED BY THE







Federal Ministry  
of Education  
and Research



Durham  
University



## Efficient GPU Offloading with OpenMP for a Hyperbolic Finite Volume Solver on Dynamically Adaptive Meshes

Mario Wille<sup>1</sup>, Tobias Weinzierl<sup>2</sup>, Gonzalo Brito Gadeschi<sup>3</sup>,  
and Michael Bader<sup>1</sup>

<sup>1</sup> TUM School of Computation, Information and Technology,  
Technical University of Munich, Garching, Germany  
([mario.wille,michael.bader@tum.de](mailto:mario.wille,michael.bader@tum.de))

<sup>2</sup> Department of Computer Science, Institute for Data Science—Large-scale  
Computing, Durham University, Durham, UK  
([tobias.weinzierl@durham.ac.uk](mailto:tobias.weinzierl@durham.ac.uk))

<sup>3</sup> NVIDIA, Munich, Germany  
([gonzalob@nvidia.com](mailto:gonzalob@nvidia.com))

**Abstract.** We identify and show how to overcome an OpenMP bottleneck in the administration of GPU memory. It arises for a wave equation solver on dynamically adaptive block-structured Cartesian meshes, which keeps all CPU threads busy and allows all of them to offload sets of patches to the GPU. Our studies show that multithreaded, concurrent, non-deterministic access to the GPU leads to performance breakdowns, since the GPU memory bookkeeping as offered through OpenMP's `map` clause, i.e., the allocation and freeing, becomes another runtime challenge besides expensive data transfer and actual computation. We, therefore, propose to retain the memory management responsibility on the host: A caching mechanism acquires memory on the accelerator for all CPU threads, keeps hold of this memory and hands it out to the offloading threads upon demand. We show that this user-managed, CPU-based memory administration helps us to overcome the GPU memory bookkeeping bottleneck and speeds up the time-to-solution of Finite Volume kernels by more than an order of magnitude.

**Keywords:** GPU offloading · Multithreading · OpenMP · Dynamically adaptive mesh refinement

This research has been supported by EPSRC's ExCALIBUR programme (projects EX20-9, PAX-HPC and MGHYPE), by the German Ministry of Education and Research (BMBWF, project targetDART) and by Intel's Academic Centre of Excellence at Durham University. Supercomputing resources and support was provided by the ARCHER2 UK National Supercomputing Service, the Erlangen National High Performance Computing Center, Jülich Supercomputing Center and CINECA. See the Acknowledgements section for details.

© The Author(s) 2023  
A. Bhatnagar et al. (Eds.): ISC High Performance 2023, LNCS 13948, pp. 65–85, 2023.  
[https://doi.org/10.1007/978-3-031-32941-5\\_4](https://doi.org/10.1007/978-3-031-32941-5_4)

# Contact

- Feel free to contact me via Mail: [mario.wille@tum.de](mailto:mario.wille@tum.de)
- Event platform Swapcard: [Mario Wille](#)
- Source code webpage: [www.peano-framework.org](http://www.peano-framework.org)
- Source code repository: <https://gitlab.lrz.de/hpcsoftware/peano>
- Artifact description: <https://doi.org/10.5281/zenodo.7741217>