Durham University
Department of Computer Science

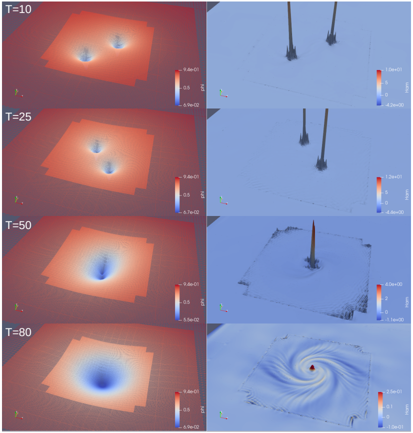
# Performance and performance-portability in ExaGRyPE

## Numerical astrophysics with tasks on heterogeneous systems

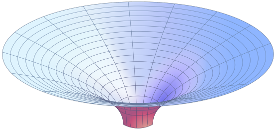
Tobias Weinzierl

January 13, 2026

# CoSeC—Numerical Relativity



# A simplified setup



(C) Wikipedia

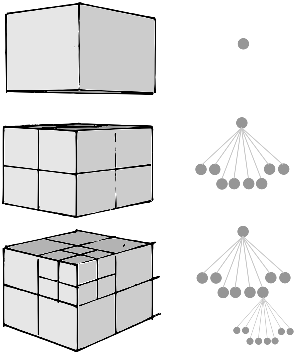
Simplification:

- No matter
- One singularity
- Smooth (multiscale) behaviour

www.peano-framework.org

# Software Stack

Durham University
Department of Computer Science



Peano 4
- Spacetree formalism
- Data management
- Domain decomposition (SFCs)

ExaHyPE 2

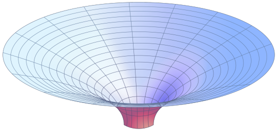
$$\frac{\partial}{\partial t} Q + \nabla F(Q) + \sum_i \mathcal{B}_i \frac{\partial Q}{\partial x_i} = \mathbf{S}(Q)$$

- Patch-based formalism: FV and FD4
- Hyperbolic infrastructure
- Solver coupling

ExaGRyPE
- CCZ4 and FO-CCZ4
- Boundary conditions
- Postprocessing
- ...

# Simplified setup revisited

Durham University
Department of Computer Science



(C) Wikipedia

Simplification:

- No matter, i.e. only 58 nonlinear PDEs
  (FO-CCZ4, but also vanilla CCZ4 (2nd order))
- One singularity
  (two solvers: FV within BH)
- Smooth (multiscale) behaviour
  (static AMR plus higher-order FD scheme outside)

Challenges:

- Fine-granular parallelism
  (no over-regularised AMR)
- Heterogeneous workload
  (two solvers)
- Vendor agnostic

# The White Knight

(C) Wikipedia

Task

- Fine-granular parallelism
  (no over-regularised AMR)
- Heterogeneous workload
  (two solvers)
- Vendor agnostic

# The White Knight

Durham University
Department of Computer Science

(C) Wikipedia

Task

- Fine-granular parallelism
  (no over-regularised AMR)
- Heterogeneous workload
  (two solvers)
- Vendor agnostic

The big task disappointments

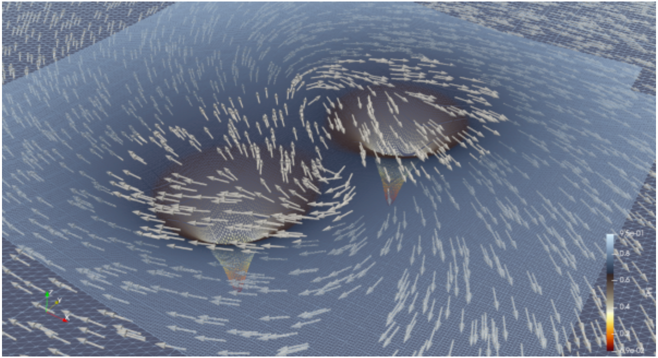
- Task dependencies tricky and assembly expensive
- Tasks runtimes tricky
- Tasks too small for GPUs

⇒ Not performing* or performance-portable

* Scalability is not performance.

# Task graph over binary black hole simulation

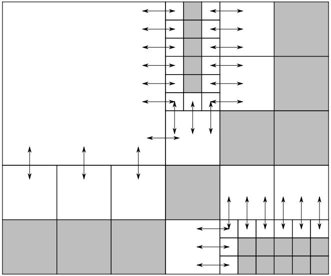
Durham University
Department of Computer Science



- Arbitrary complicated (unless 2:1 balancing = regularisation)
- Permanently changing (unless AMR criterion kicks in only every N steps)
- Interplay with MPI non-trivial

# Enclave tasking in ExaHyPE

Durham University
Department of Computer Science



D.E. Charrier, B. Hazelwood, T. Weinzierl: Enclave Tasking for DG Methods on Dynamically Adaptive Meshes. SISC 42(3) (2020)

- Distinguish critical and not-that-urgent tasks in AMR code:
  - MPI boundary
  - AMR boundary
  - Coupling cells (multiple solvers)
- Two sweep paradigm:
  - Pimary sweep: spawn enclaves and handle skeleton (urgent) cells
  - Secondary sweep: collect enclave outcomes
  - Identify simple (regular) data dependencies

$\Rightarrow$ (almost) assembly-free tasking

# (Minor) Showstoppers

Durham University
Department of Computer Science

OneAPI / OneTBB

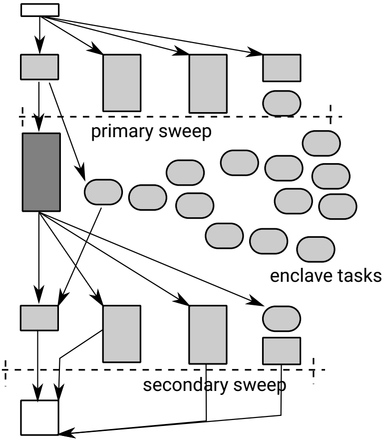
- Clean-up API with transition from TBB to OneTBB
- Remove dynamic task graphs (all preassembled)
- Mirror CUDA graphs

⇒ Patch it and release it

OpenMP

- Transparent tasks required
- Strict tree topology, not producer-consumer

⇒ Wait or write (manual) workaround



# Three regimes of results

Durham University
Department of Computer Science

1. Very coarse meshes vs. many cores
   $\Rightarrow$ Strong scaling issues
2. Medium size, pretty regular meshes; “nice” core count
   $\Rightarrow$ Brilliant performance
3. Medium size, strongly adaptive meshes; “nice” core count
   $\Rightarrow$ Good performance
4. Very fine, pretty regular meshes; “any” core count
   $\Rightarrow$ Brilliant performance

D.E. Charrier, B. Hazelwood, T. Weinzierl: Enclave Tasking for DG Methods on Dynamically Adaptive Meshes. SISC 42(3) (2020)

# Three regimes of results

Durham University
Department of Computer Science

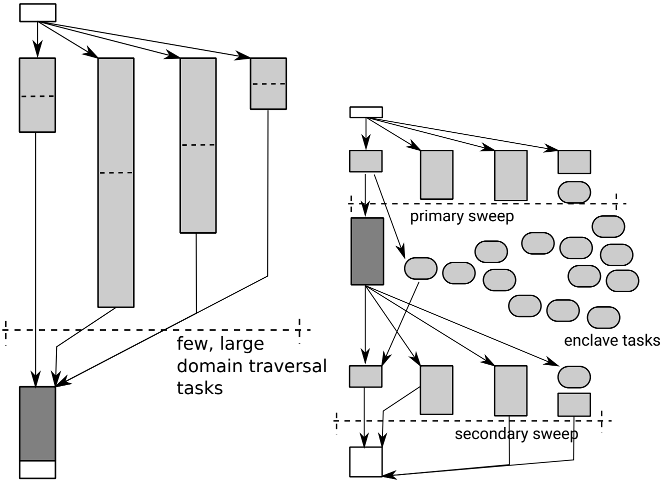
1. Very coarse meshes vs. many cores
   $\Rightarrow$ Strong scaling issues
2. Medium size, pretty regular meshes; “nice” core count
   $\Rightarrow$ Brilliant performance
3. Medium size, strongly adaptive meshes; “nice” core count
   $\Rightarrow$ Good performance
4. Very fine, pretty regular meshes; “any” core count
   $\Rightarrow$ Brilliant performance

   D.E. Charrier, B. Hazelwood, T. Weinzierl: Enclave Tasking for DG Methods on Dynamically Adaptive Meshes. SISC 42(3) (2020)

5. Very fine, strongly adaptive meshes; any core count
   $\Rightarrow$ Breaks down

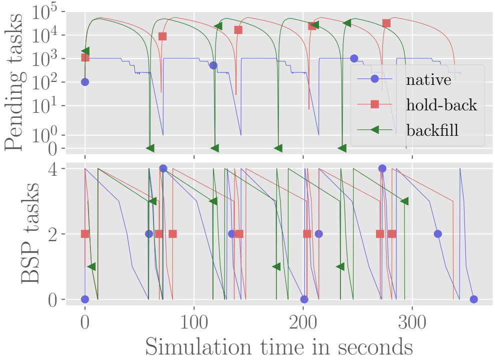
# Automatic load balancing through task stealing

Durham University
Department of Computer Science



# Too much tasking can kill you

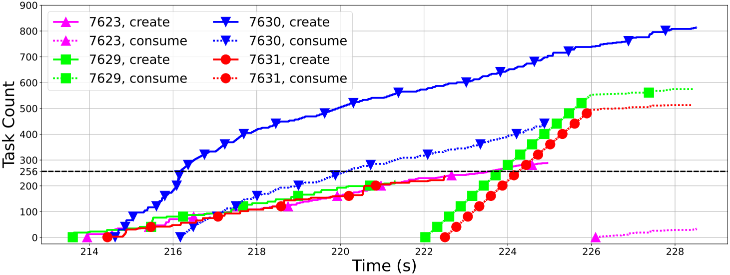
Durham University
Department of Computer Science




H. Schulz et al: *Task inefficiency patterns for a wave equation solver* (arXiv:2105.12739)

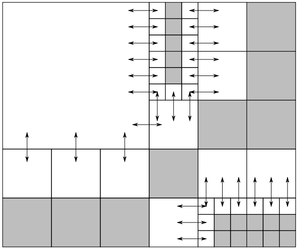
- OpenMP switches to immediate processing
- Critical path (cmp. previous slide) not always active
- \- Flaw report to OpenMP community
- \+ Introduce user-defined queue
- \- LLVM runtime slightly better
- \- Priorities are still not supported

# Constructive way forward




- Publish insight on IWOMP together with simulator: quantify gain
- Propose `taskwait` annotation
- Write prototype implementation
  (work by M. Klemm)
- Ship workaround for own code for the time being

# Enclave tasks



**Skeleton cells**

⇒ Process in-situ

**Enclave cells**

⇒ Huge producer-consumer pattern

**Motivation**

- Compensate fork-join imbalances
  (“historic motivation”)
- Deploy to GPUs
- All hidden from user

# Almost there . . .

Durham University
Department of Computer Science

. . . we only need to throw the tasks onto GPUs.

# OpenMP starting point

Durham University
Department of Computer Science

```
[...]
#pragma omp parallel for simd collapse(3)
for (int z = 0; z < numberOfVolumesPerAxisInPatch; z++)
for (int y = 0; y < numberOfVolumesPerAxisInPatch; y++)
for (int x = 0; x < numberOfVolumesPerAxisInPatch; x++) {
  internal::copySolutionAndAddSourceTerm_LoopBody<Solver>(
    [...]
  );
}
```

- Theory: Just replace

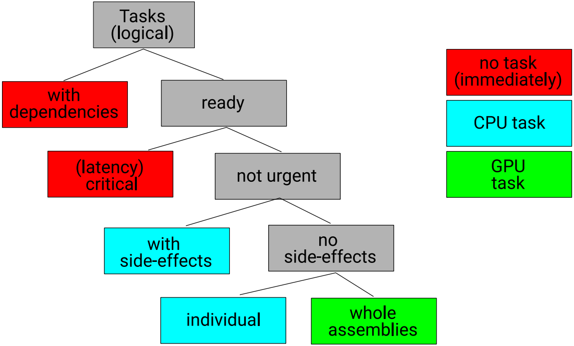
```
#pragma omp parallel for simd collapse(3)
```

with

```
#pragma omp target teams distribute parallel for simd collapse(3) device(targetDevice)
```

- Praxis
  - Code crashes ⇒ User tasks alter state
  - Code very slow ⇒ Tasks too cheap

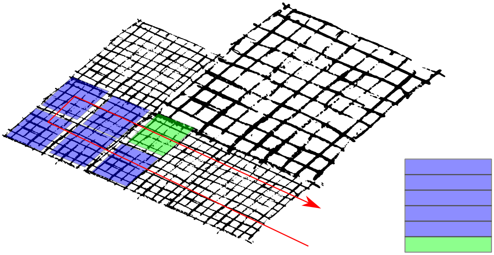
# Masking: Pick GPU tasks carefully




- ready: No need to maintain dependencies over devices
- not urgent: Hide latency induced by a remote accelerator or NUMA
  (enclave idea)
- no side-effects: Embarrassingly parallel; do not maintain global state
  (can be found out/modelled through C++'s `static`)
- individual: Individual tasks might be too lightweight

# Hold back tasks

Durham University
Department of Computer Science



- Spawn tasks into intermediate layer $\Rightarrow$ buffer
- If $k \geq K$ tasks of same type w/o side-effects: deploy in one rush/burst to GPU
- If task of different type than previous ones: release to CPU

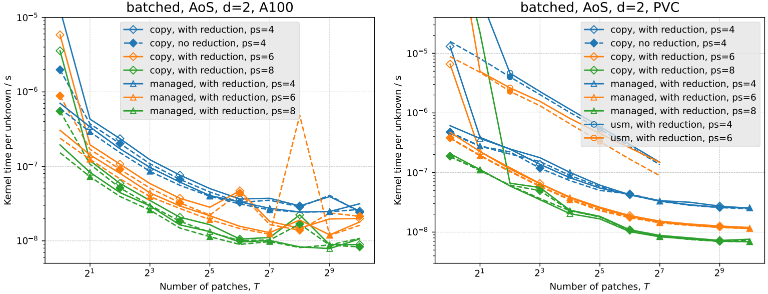
$\Rightarrow$ Arrangement/assignments of tasks to GPU is not fixed

- Changes with AMR
- Changes with physics (side effects)
- Changes with GPU occupation
- ...

B. Li, H. Schulz, T. Weinzierl, H. Zhang: *Dynamic task fusion for a block-structured finite volume solver over a dynamically adaptive mesh with local time stepping*. ISC High Performance 2022, LNCS 13289, pp. 153-173 (2022)

# Task fusion in action

Durham University
Department of Computer Science




Left: A100, right: PVC (one stack); single task producer fires onto GPU

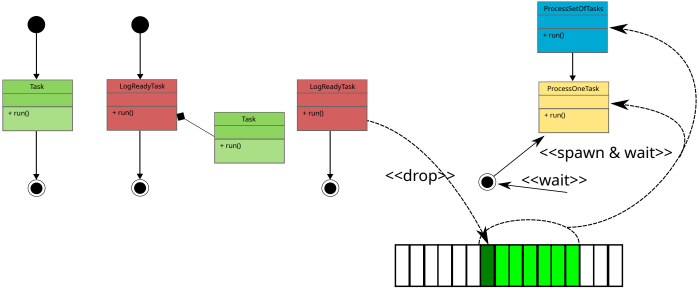
- Different realisation variants (loop ordering; not yet xDSL)
- OpenMP and SYCL back-end
- Clear dependency on task fusion count

⇒ Fusing too few tasks bad idea (throughput)

⇒ Fusing tasks too late bad idea (algorithmic latency)

Durham University
Department of Computer Science

# WiP: Dynamic task fusion




Concept:

- Collect tasks as we run through mesh
- Bundle (fuse) them on-the-fly
- Offload full bundle to GPU

Work:

- Three different realisation variants of on-the-fly fusion
- Integrate into both OpenMP and TBB
- Embed dynamic task fusion into dynamically assembled graphs with in-/out-dependencies

⇒ Hide completely in threading runtime

# Summary

Durham University
Department of Computer Science

(C) Wikipedia

Task disappointments

- Task dependencies tricky and assembly expensive
- Tasks runtimes tricky
- Tasks too small for GPUs

Lessons learned: Good modelling tool

- Distinguish modelling from implementation
- Performance not free
  (tweak runtimes)
- Performance-portability myth
  (tweak algorithmics)
- Even best abstraction layers require in-depth understanding
  (OpenMP, TBB)
- Performs requires compromises!?
  (2:1 balancing, excessive patches, temporarily stationary task graphs, . . . )