A Massively-Parallel, Fault-Tolerant Solver for Time-Dependent PDEs in High Dimensions

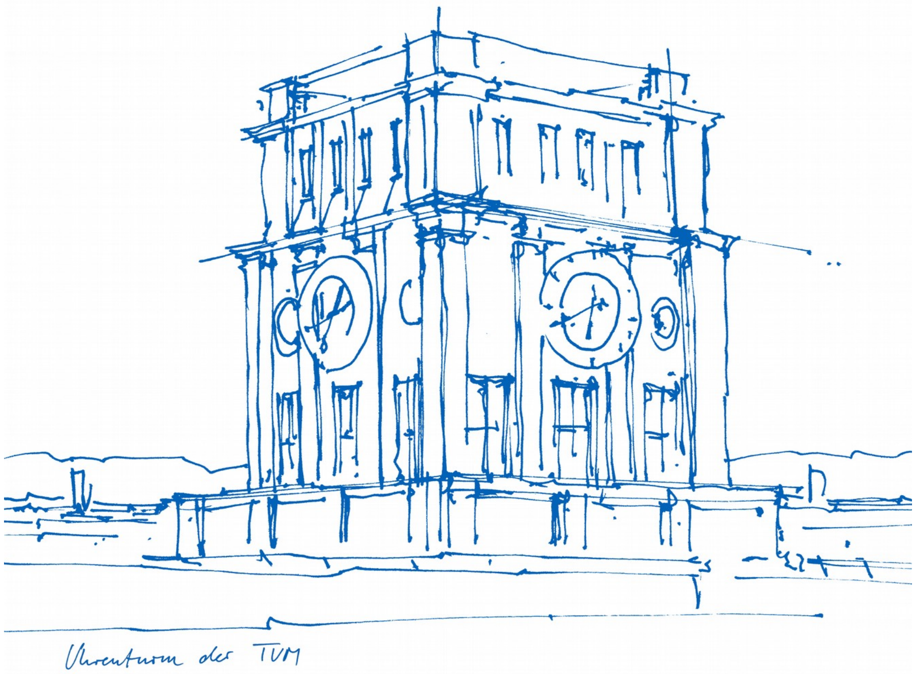
Euro-Par 2016: Resilience

Mario Heene[2], Alfredo Parra[1], Hans-Joachim Bungartz[1], Dirk Pflüger[2]

[1]Technical University of Munich
Chair of Cientific Computing

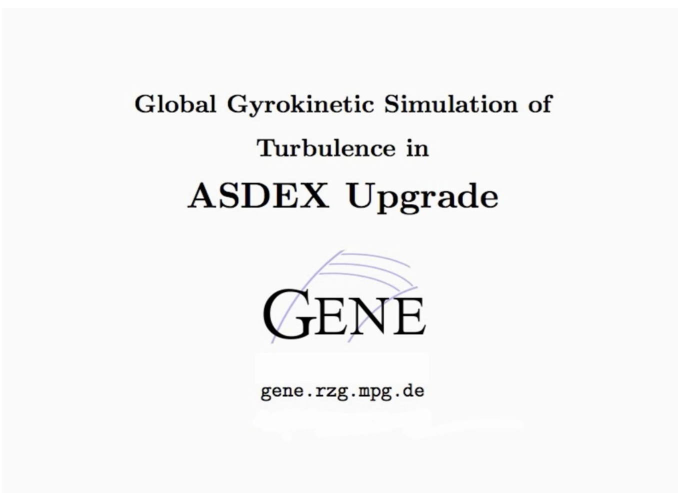
[2]University of Stuttgart
Institute for Parallel and Distributed Systems

August 23, 2016



EXAHD Project

- Goal: (exa)scalable solution of high-dimensional PDEs
- Main challenge: *curse of dimensionality*
  - 2^n discretization points per dimension → $(2^n)^d$ total points



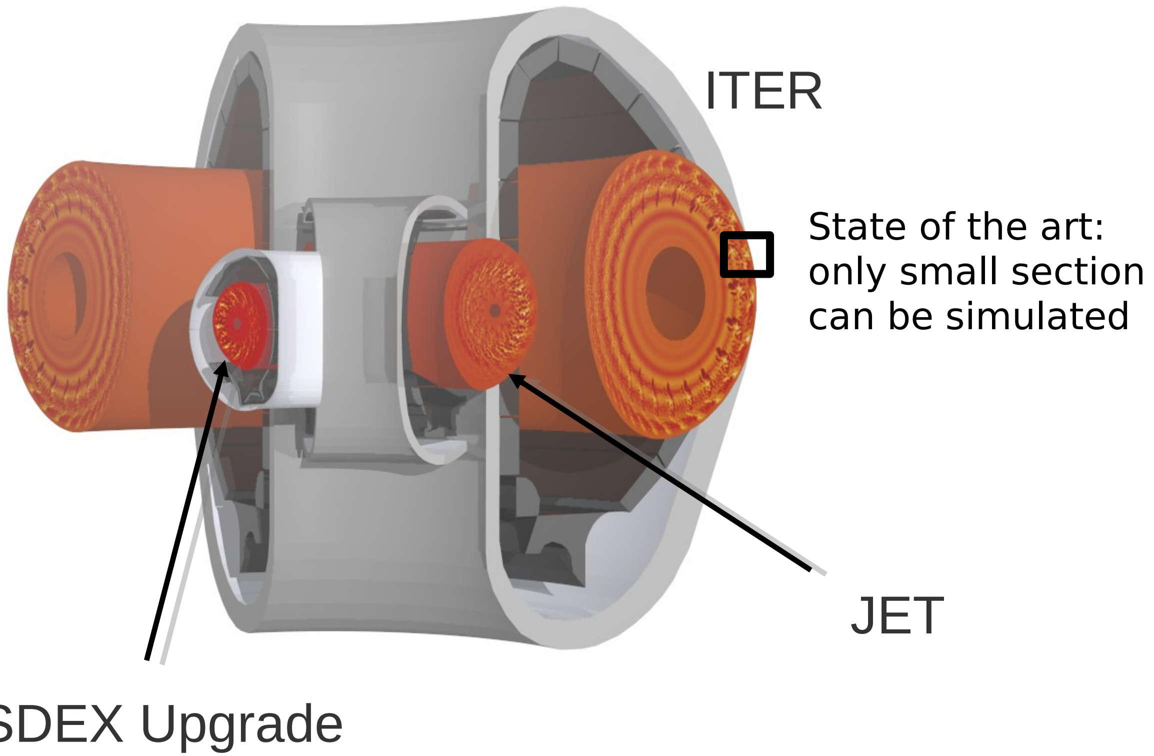
Example: *GENE* Code [1]

- Plasma simulations
- 5D + time nonlinear PDE (Vlasov equations):

$$u(t; x, y, z, \mu, \nu) \rightarrow 2^{n_x} \times 2^{n_y} \times 2^{n_z} \times 2^{n_\mu} \times 2^{n_\nu}$$

- Production runs: billions of grid points!
- ~10^7 core-hours; several TB of short-term storage
- Future of clean energy?

Problem 1: Computational resolution limit reached



Gyrokinetic Electromagnetic Numerical Experiment

Problem 2: *GENE* is not fault tolerant

- 100,000's lines of Fortran code!
- Replace MPI with ULFM?
- Implement efficient checkpointing
- Redefine communicators
- Implement restart / recovery routines
- 1.5 Post Doc years later + 1,000's lines of code changed + 10's of E-Mails with ULFM devs
  → Little progress...

Goal 1: Increase computational resolution of high-dimensional PDE solvers

Goal 2: Resilience

Our approach: New *algorithmic* approaches

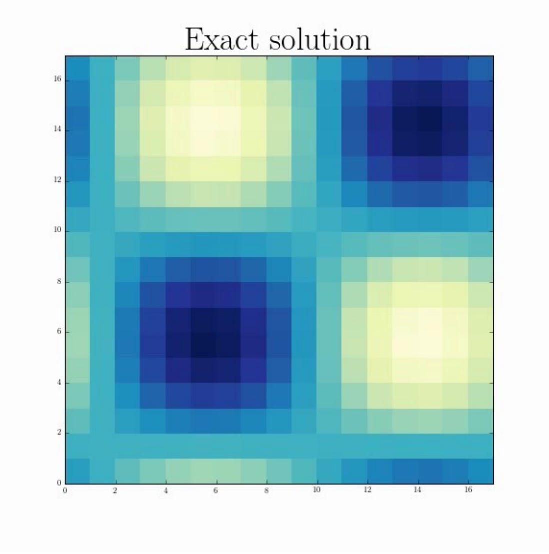
→ The Sparse Grid Combination Technique

The Sparse Grid Combination Technique

- Extrapolation method to solve high-dimensional problems
- Simple example in 2D

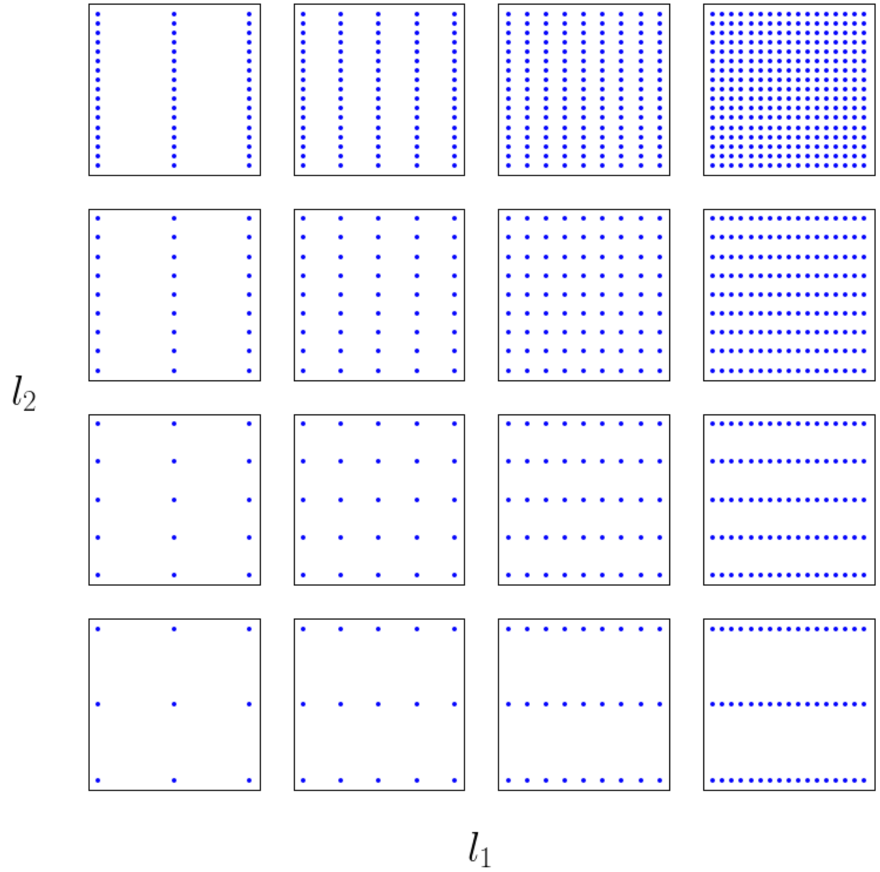
$$\frac{\partial u}{\partial t} + \frac{\partial u}{\partial x} + \frac{\partial u}{\partial y} = 0$$

with $u(x, y, t = 0) = \sin(2\pi x)\sin(2\pi y)$



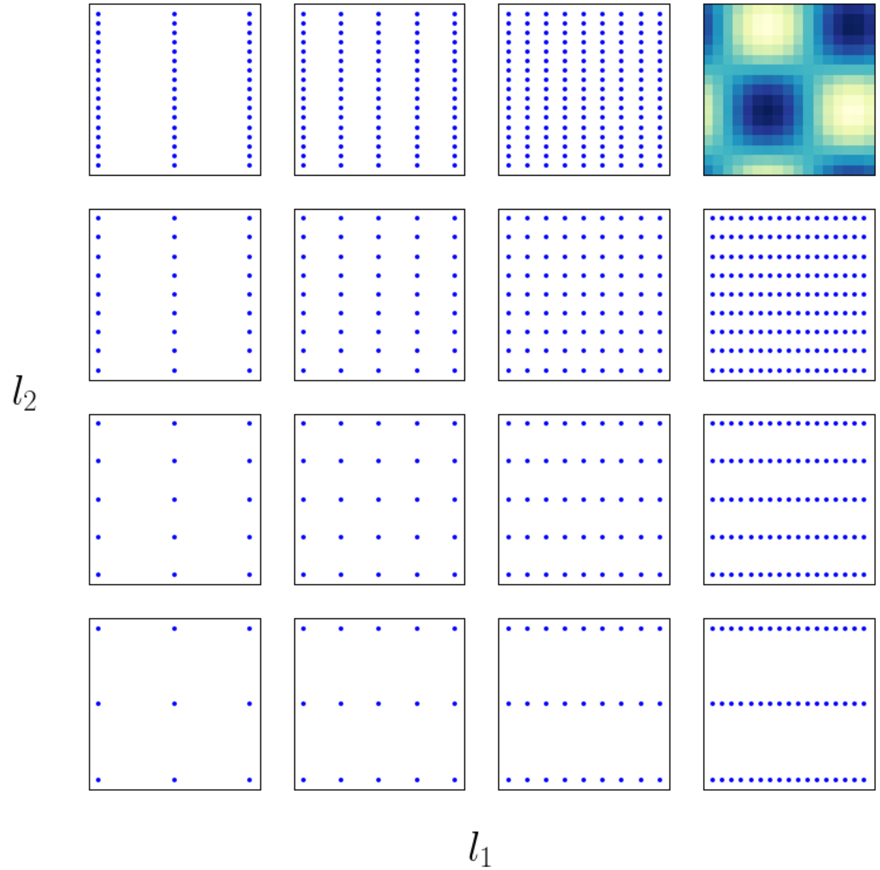
The Sparse Grid Combination Technique

- Each grid has $(2^{l_1}+1)\times(2^{l_2}+1)$ grid points



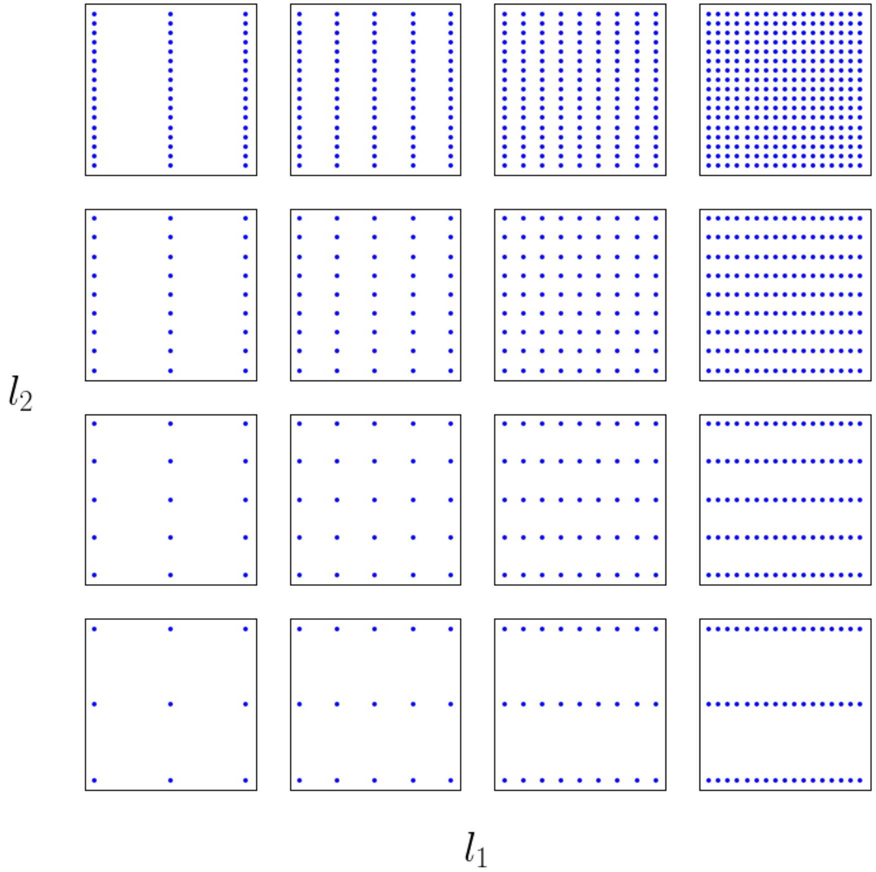
The Sparse Grid Combination Technique

- Each grid has $(2^{l_1}+1)\times(2^{l_2}+1)$ grid points



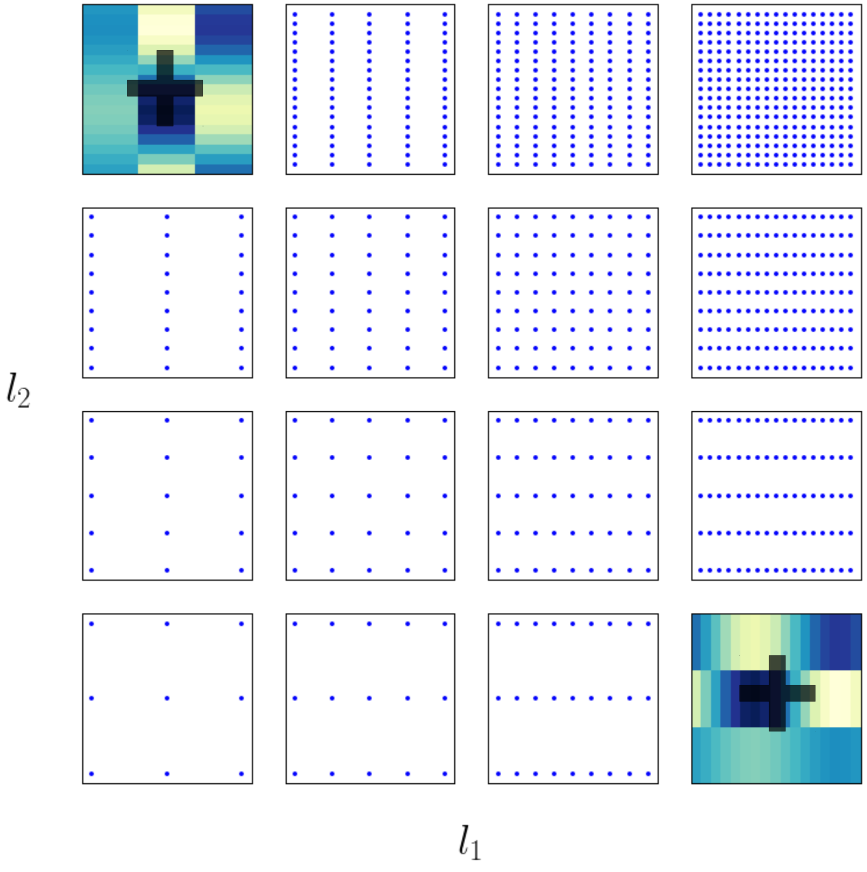
The Sparse Grid Combination Technique

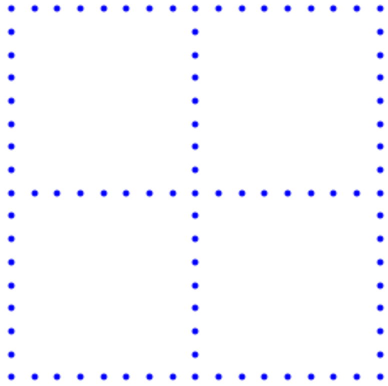
- A very simple extrapolation scheme



The Sparse Grid Combination Technique

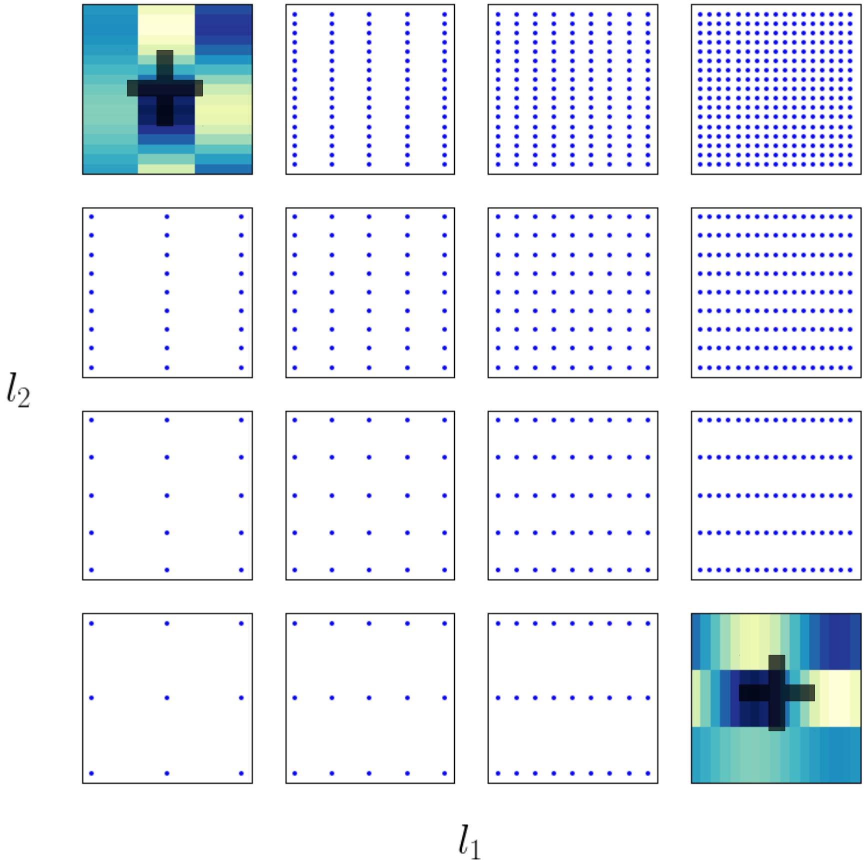
- A very simple extrapolation scheme

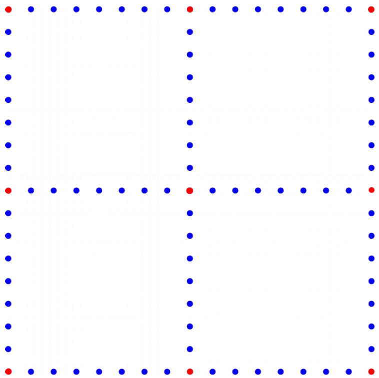




The Sparse Grid Combination Technique

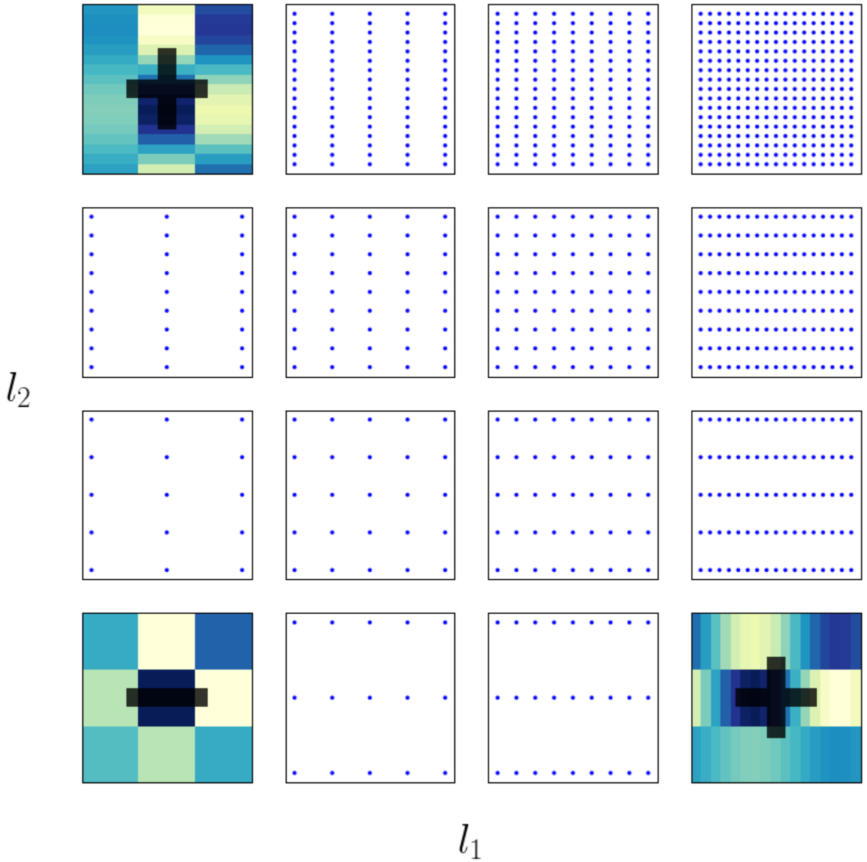
- A very simple extrapolation scheme

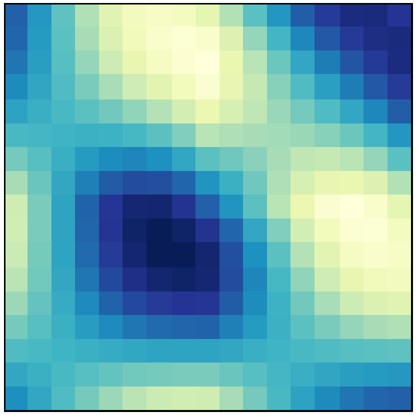




The Sparse Grid Combination Technique

- A very simple extrapolation scheme

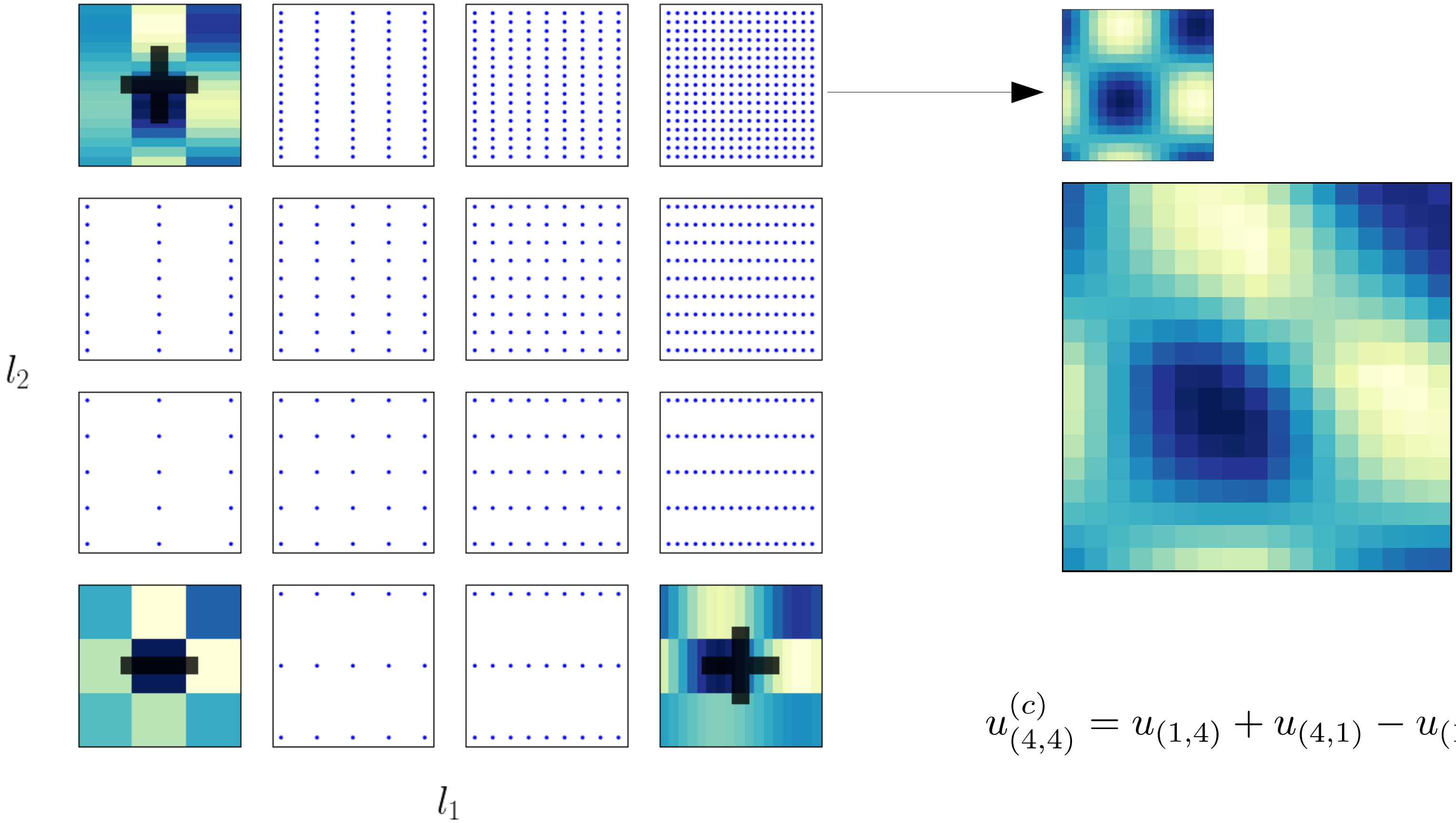




$$u^{(c)}_{(4,4)} = u_{(1,4)} + u_{(4,1)} - u_{(1,1)}$$

The Sparse Grid Combination Technique

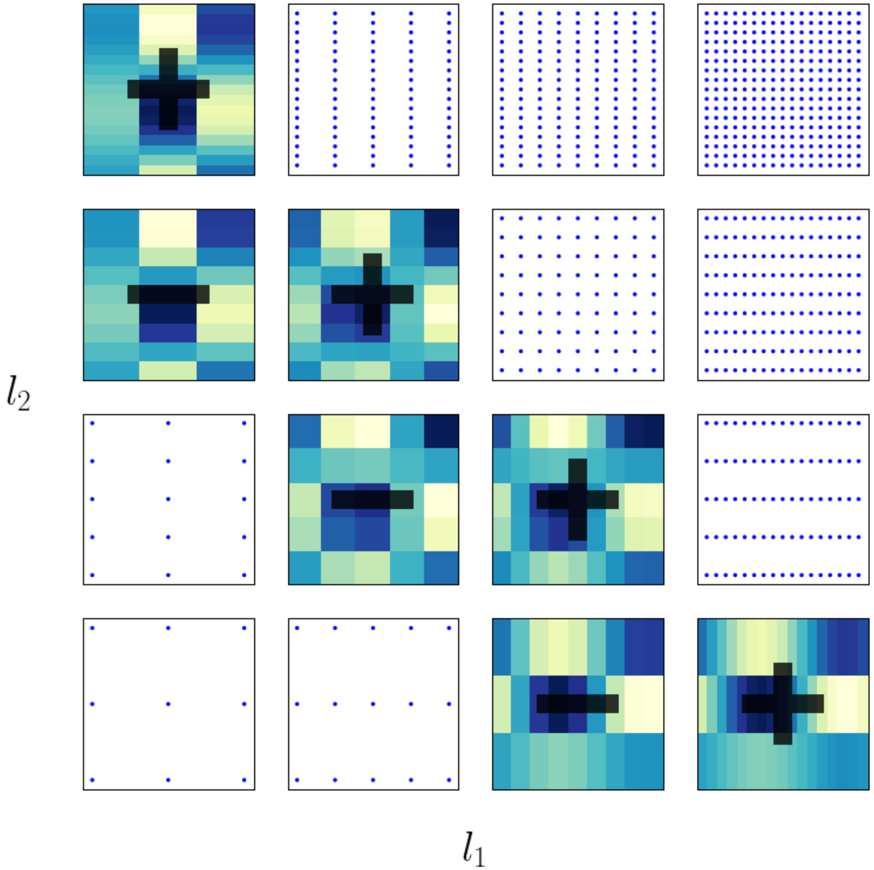
- A very simple extrapolation scheme

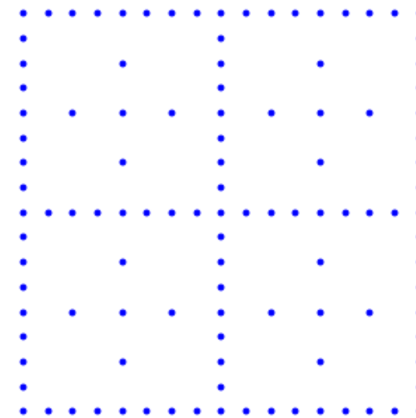


$$u^{(c)}_{(4,4)} = u_{(1,4)} + u_{(4,1)} - u_{(1,1)}$$

The Sparse Grid Combination Technique

- The *Classical Combination Technique*

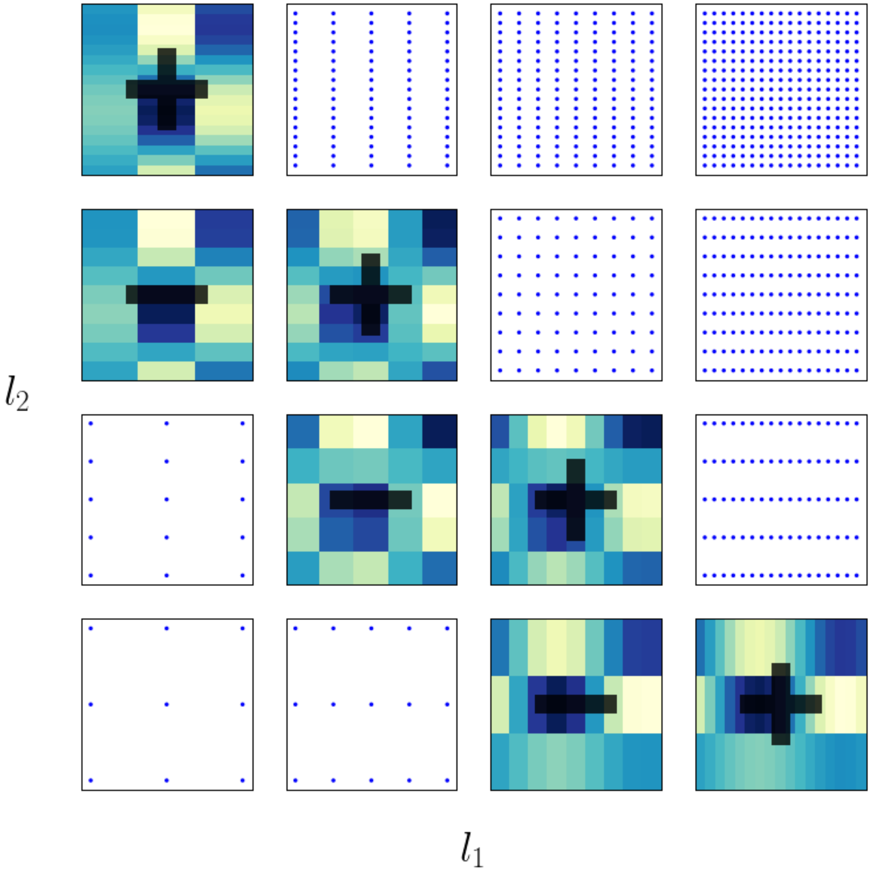


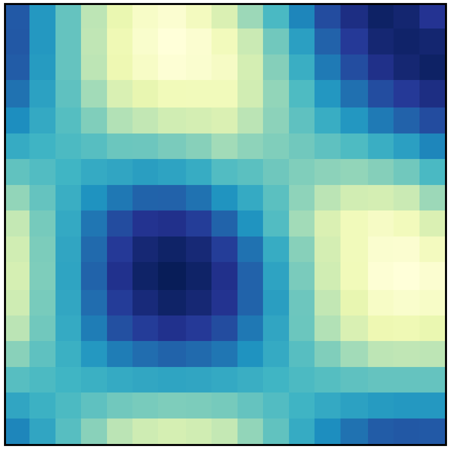


$$u^{(c)}_{(4,4)} = u_{(1,4)} + u_{(2,3)} + u_{(3,2)} + u_{(4,1)} - u_{(1,3)} - u_{(2,2)} - u_{(3,1)}$$

The Sparse Grid Combination Technique

- The *Classical Combination Technique*

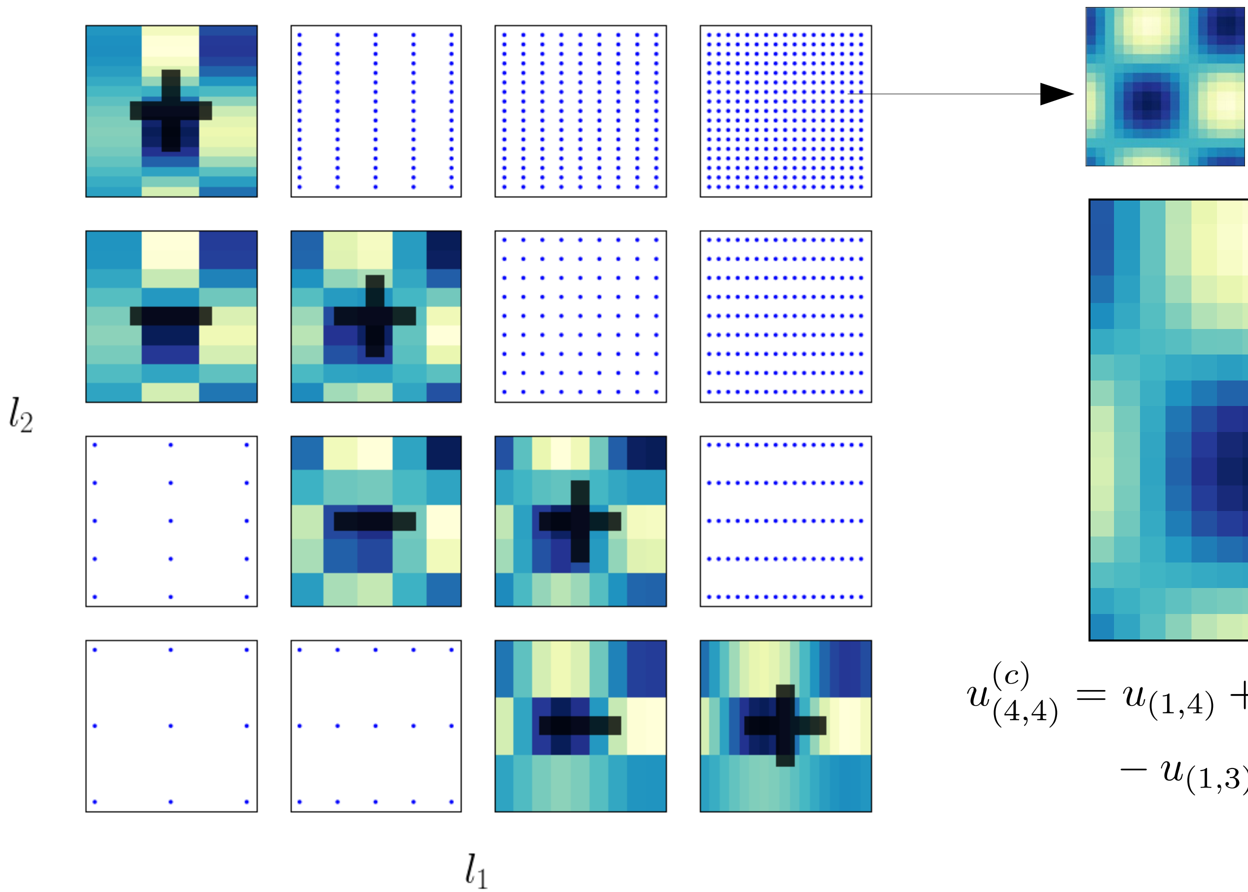


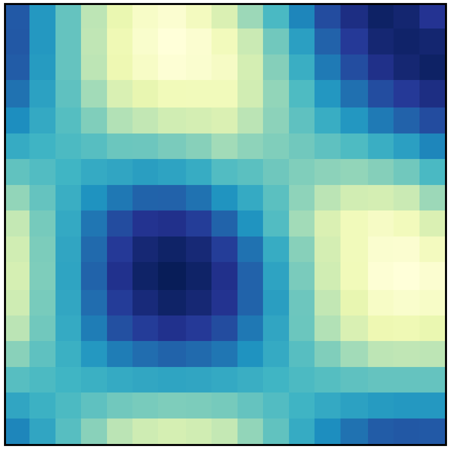


$$u^{(c)}_{(4,4)} = u_{(1,4)} + u_{(2,3)} + u_{(3,2)} + u_{(4,1)} - u_{(1,3)} - u_{(2,2)} - u_{(3,1)}$$

The Sparse Grid Combination Technique

- The *Classical Combination Technique*

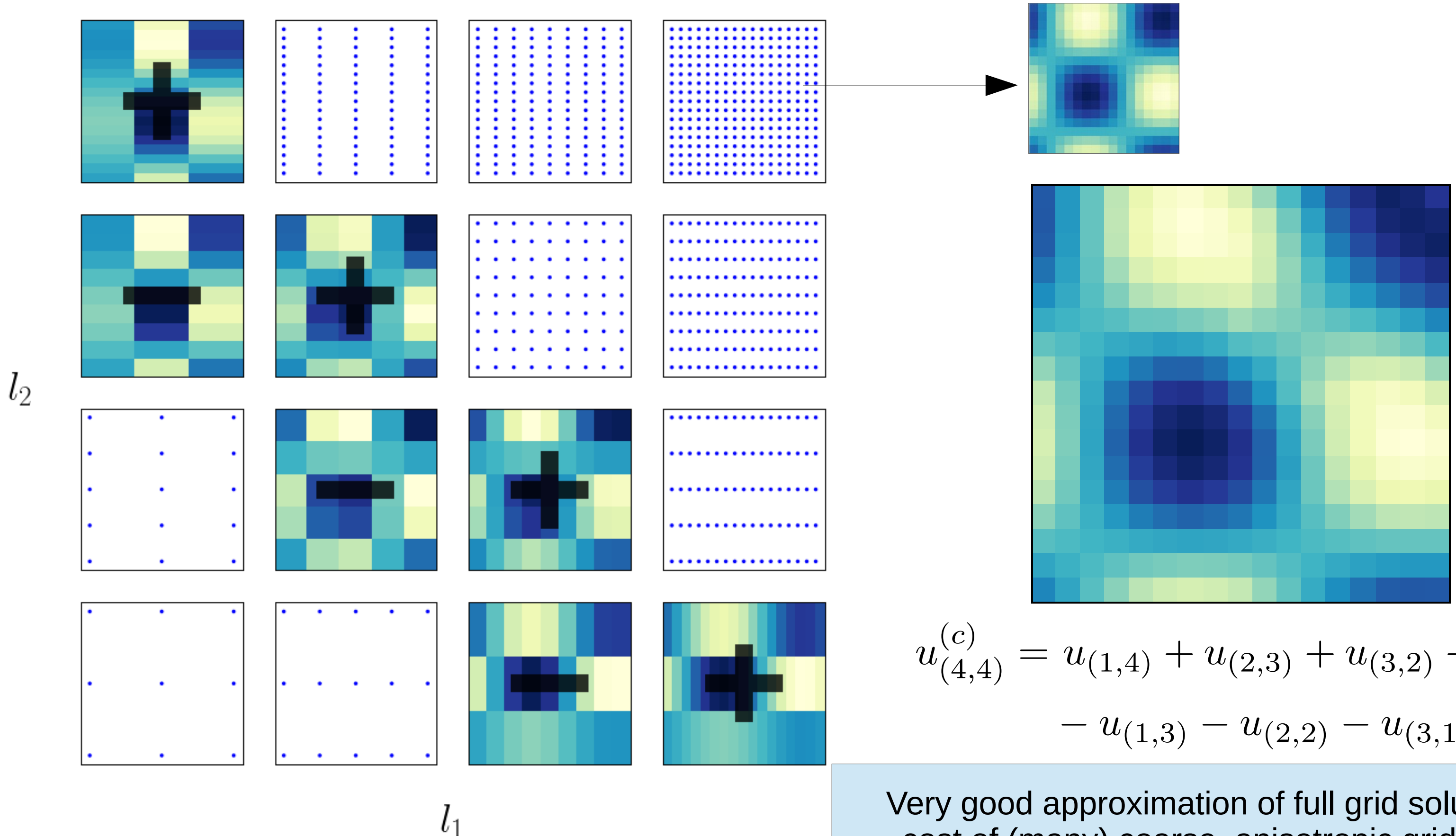




$$u^{(c)}_{(4,4)} = u_{(1,4)} + u_{(2,3)} + u_{(3,2)} + u_{(4,1)}$$
$$- u_{(1,3)} - u_{(2,2)} - u_{(3,1)}$$

The Sparse Grid Combination Technique

- The *Classical Combination Technique*

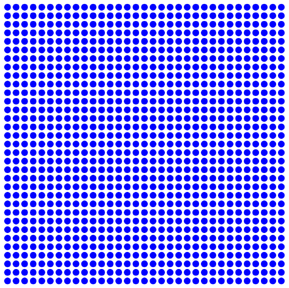


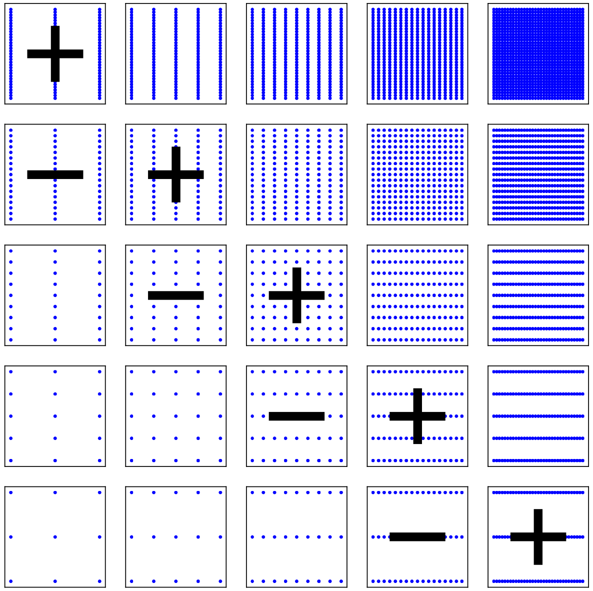
$$u^{(c)}_{(4,4)} = u_{(1,4)} + u_{(2,3)} + u_{(3,2)} + u_{(4,1)} - u_{(1,3)} - u_{(2,2)} - u_{(3,1)}$$

Very good approximation of full grid solution at the cost of (many) coarse, anisotropic grids (cheap!)

The Sparse Grid Combination Technique

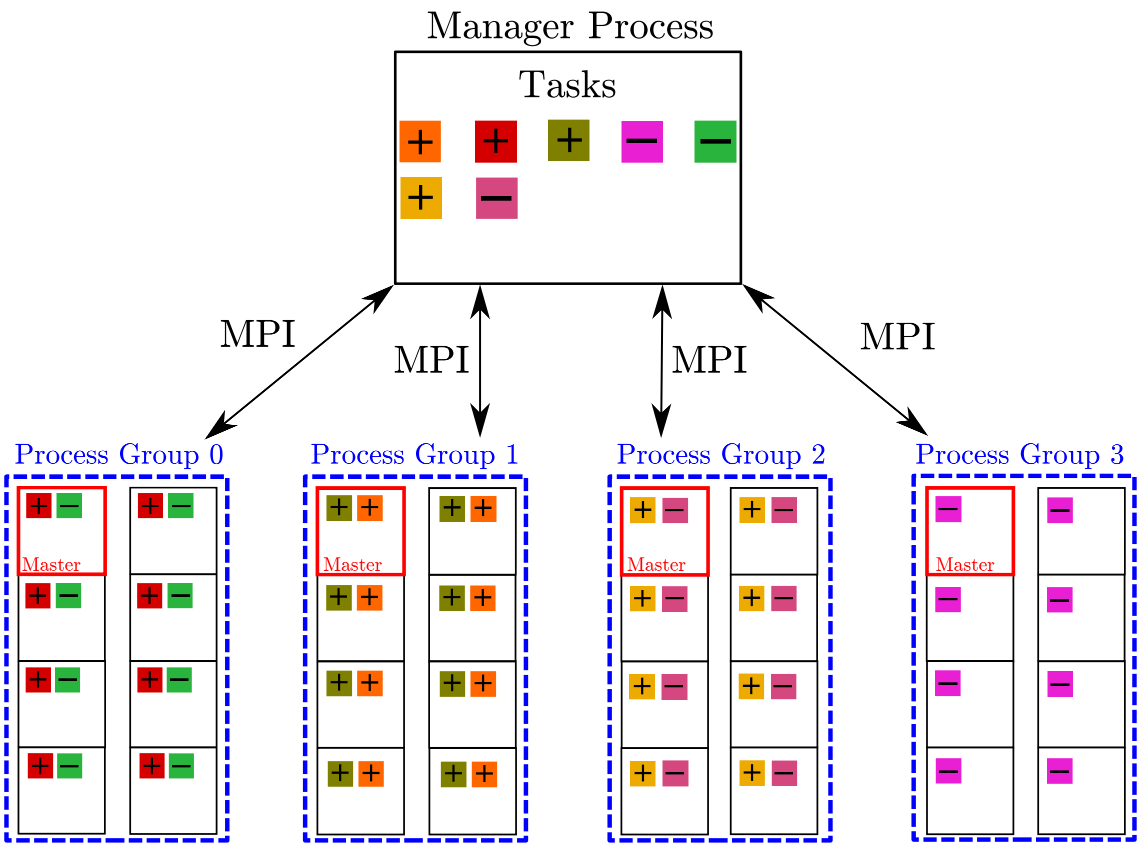
Dimension	# points per dimension	# points full grid	# points Combination Technique
6	2^{10}	$> 10^{18}$	4,096 × 249,000
10	2^{12}	$> 10^{37}$	352,705 × 80,641,000





Parallelizing the Combination Technique

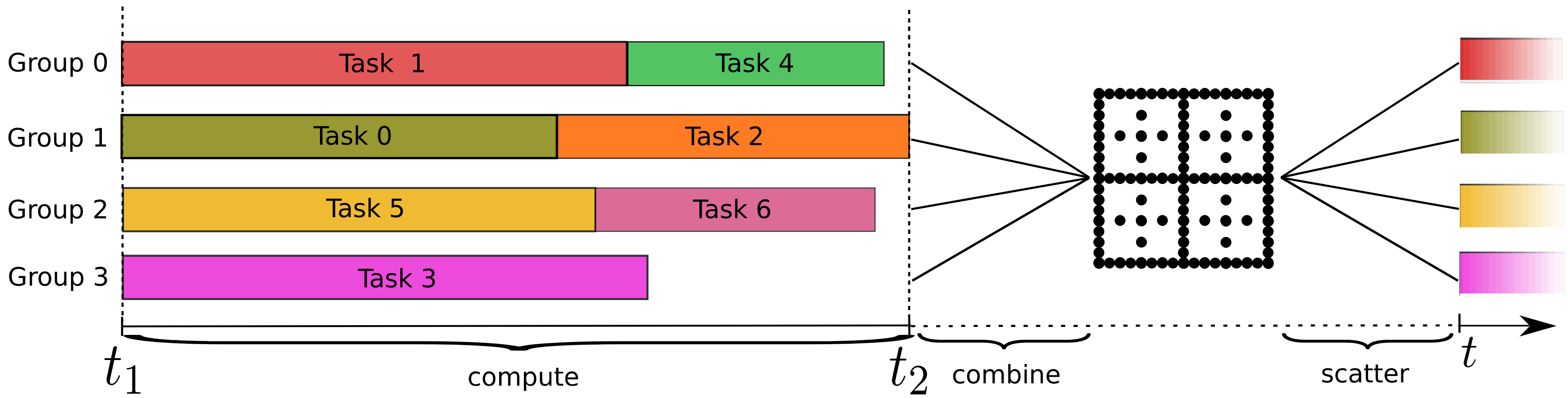
Manager-Worker Model



Parallelizing the Combination Technique

Basic algorithm:

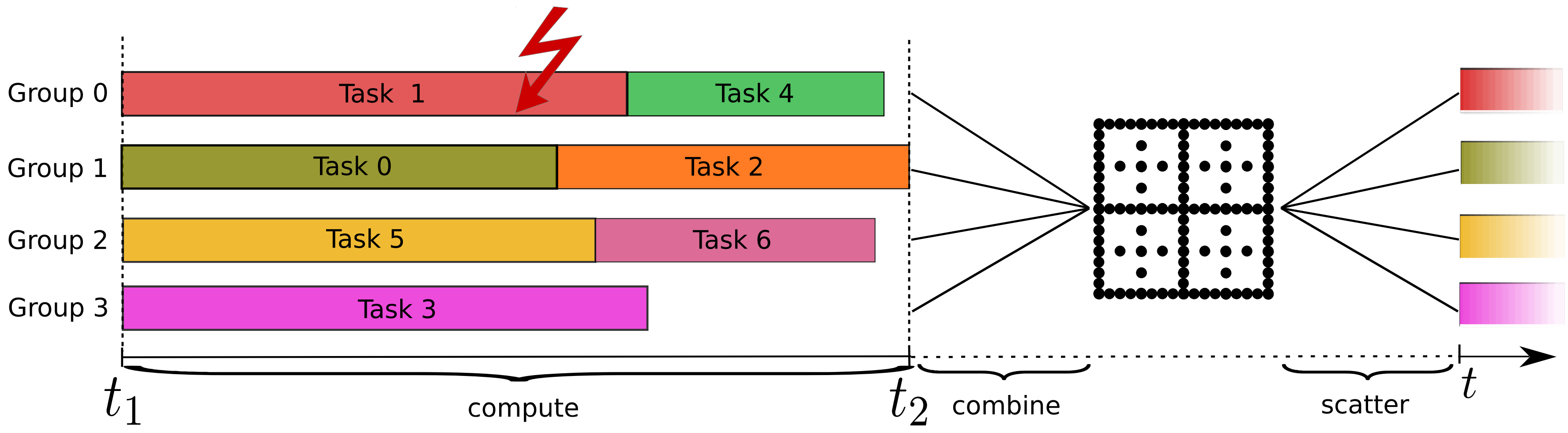
1. Distribute tasks among groups and set initial conditions
2. Each group solves N timesteps of each task
3. Combine tasks to obtain sparse grid
4. Use combined sparse grid solution as initial condition for next N timesteps



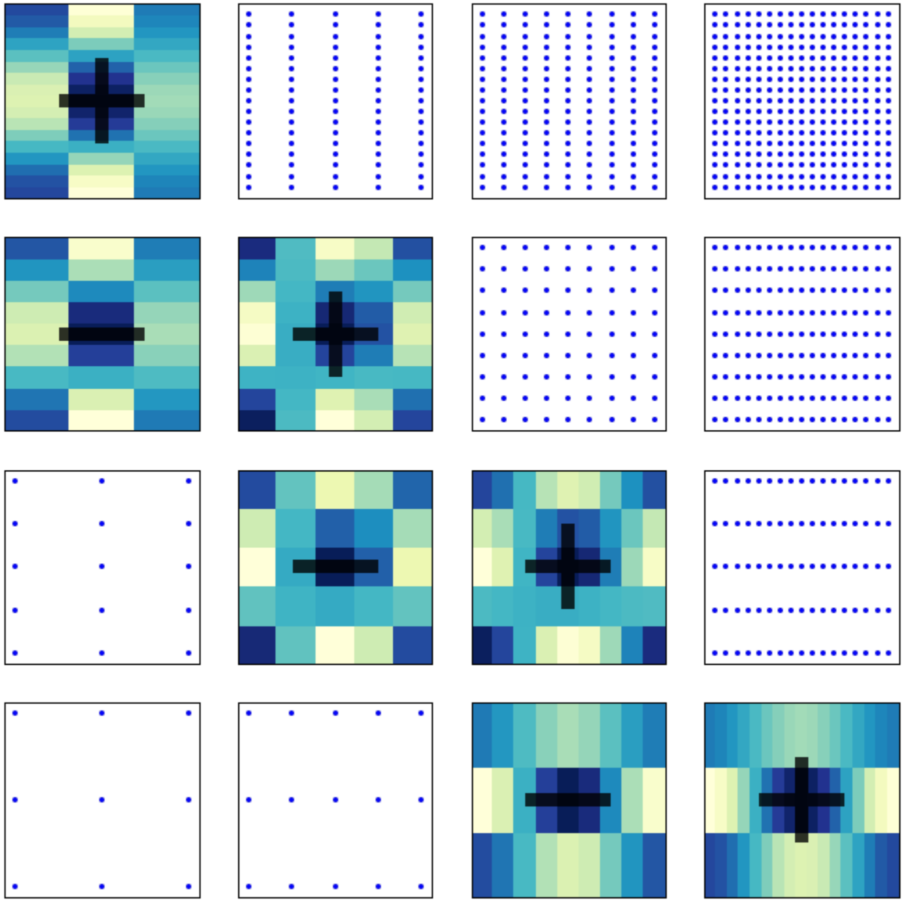
Parallelizing the Combination Technique

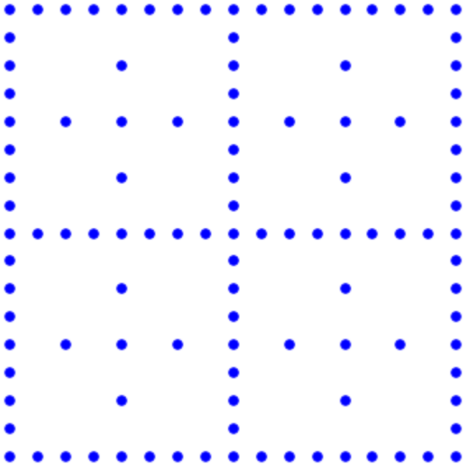
Basic algorithm:

1. Distribute tasks among groups and set initial conditions
2. Each group solves N timesteps of each task
3. Combine tasks to obtain sparse grid
4. Use combined sparse grid solution as initial condition for next N timesteps

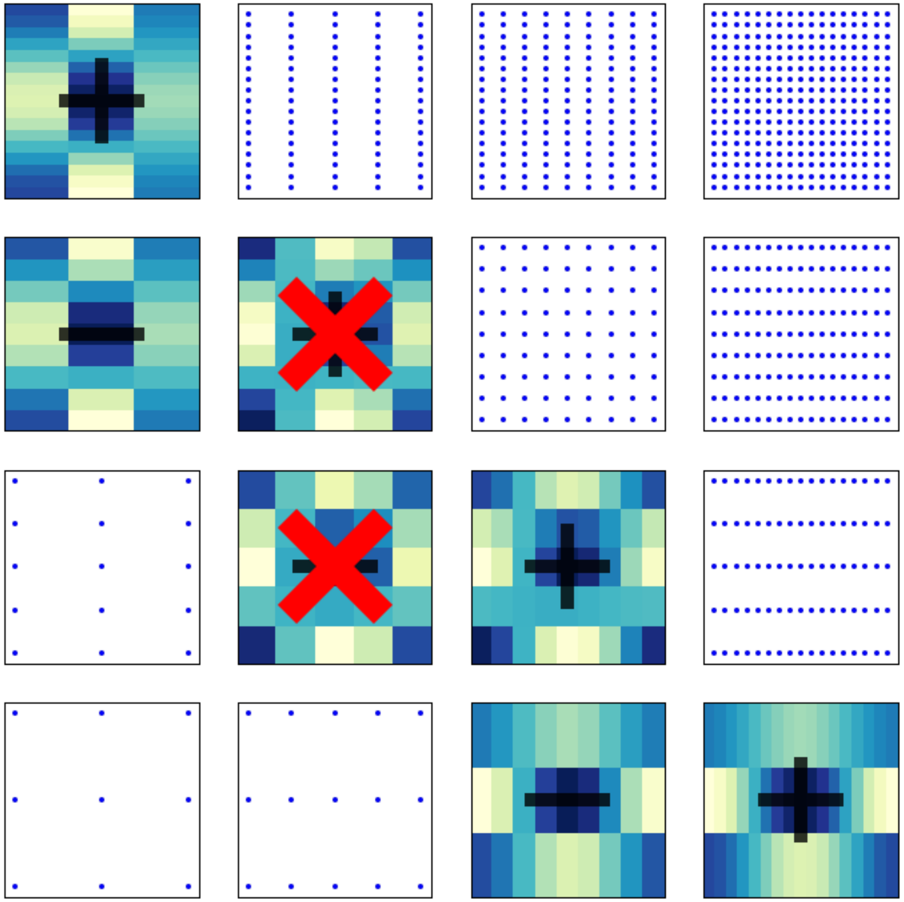


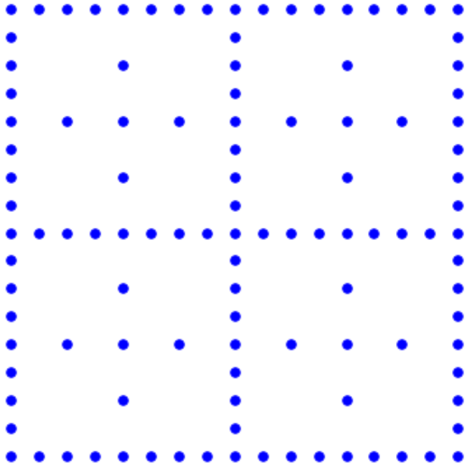
Algorithmic approach to fail-stop failures [2]



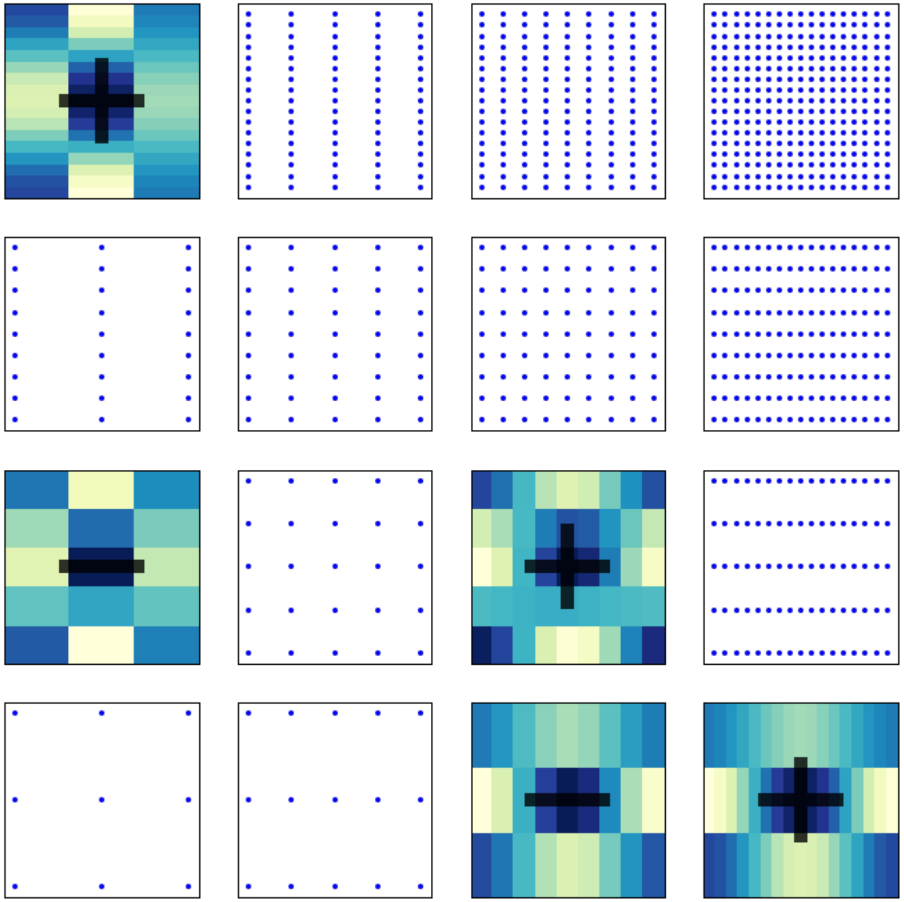


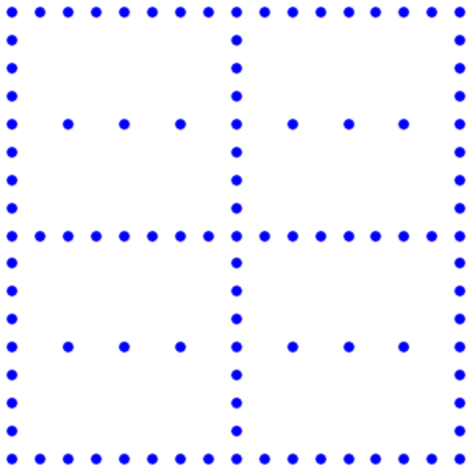
Algorithmic approach to fail-stop failures [2]





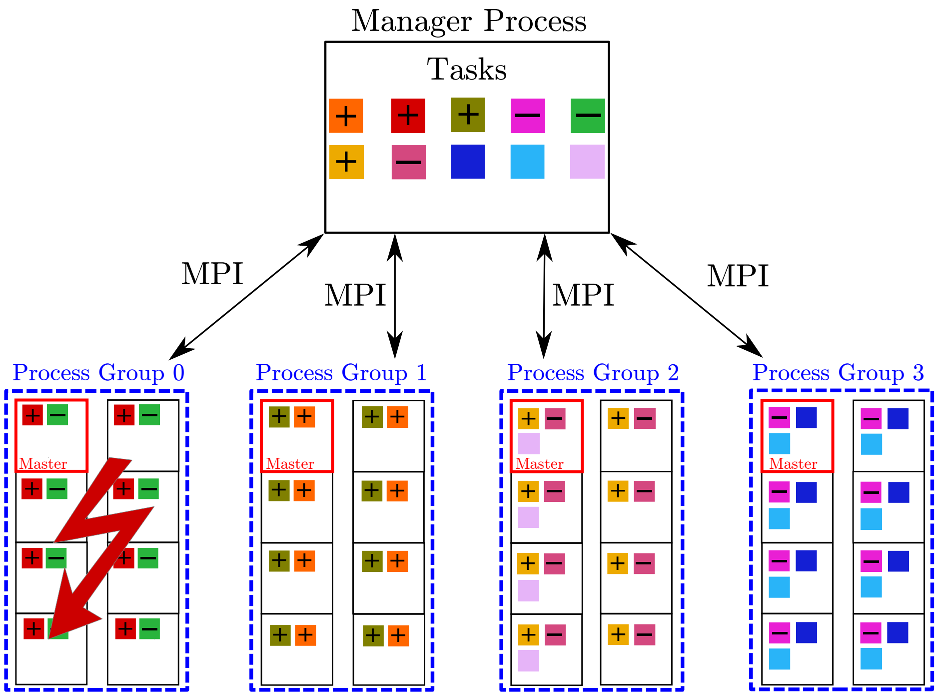
Algorithmic approach to fail-stop failures [2]





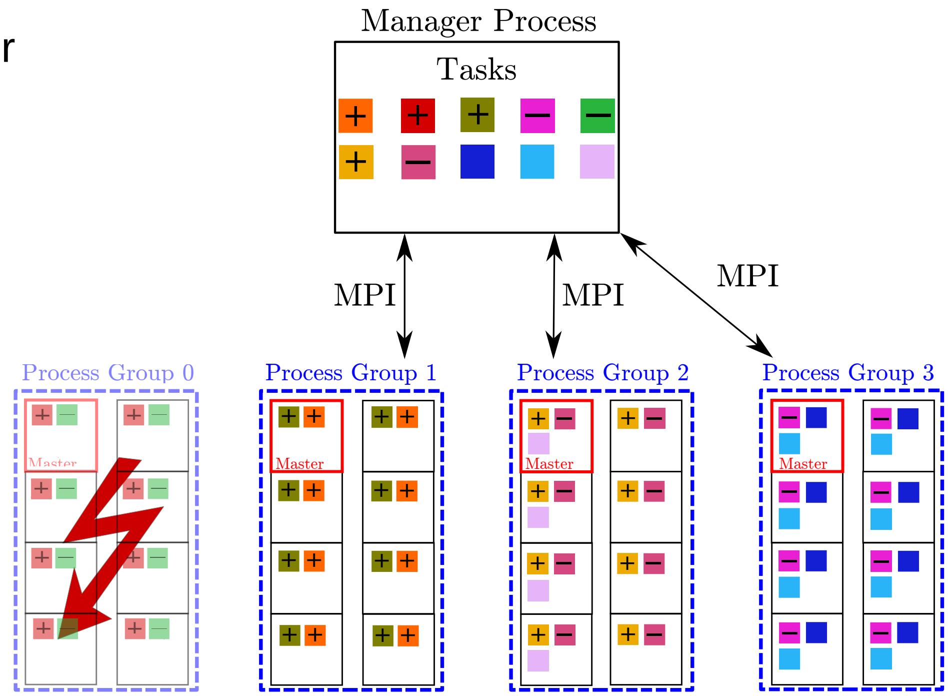
Recovery strategy (ULFM-like)

- Detect failed group during collective operation (combine)



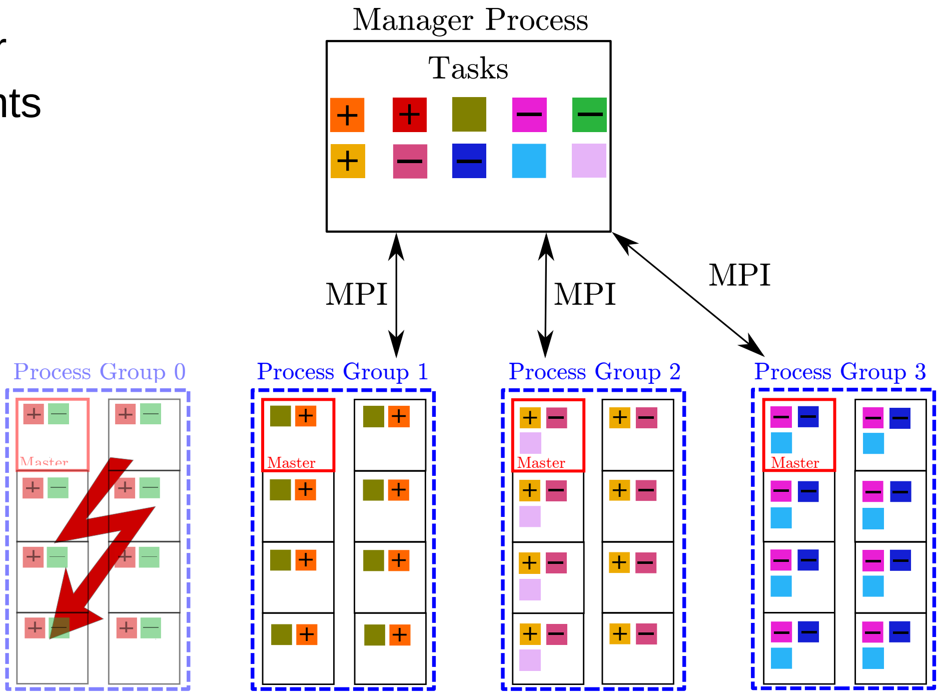
Recovery strategy (ULFM-like)

- Detect failed group during collective operation (combine)
- Exclude failed group(s) from communicator



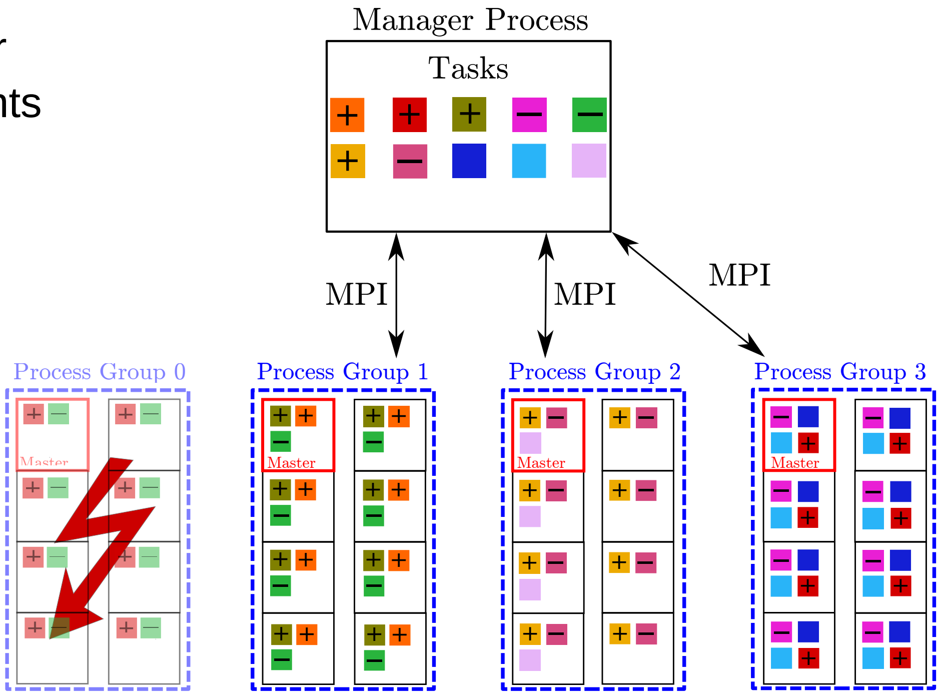
Recovery strategy (ULFM-like)

- Detect failed group during collective operation (combine)
- Exclude failed group(s) from communicator
- Compute alternative combination coefficients and perform combination*



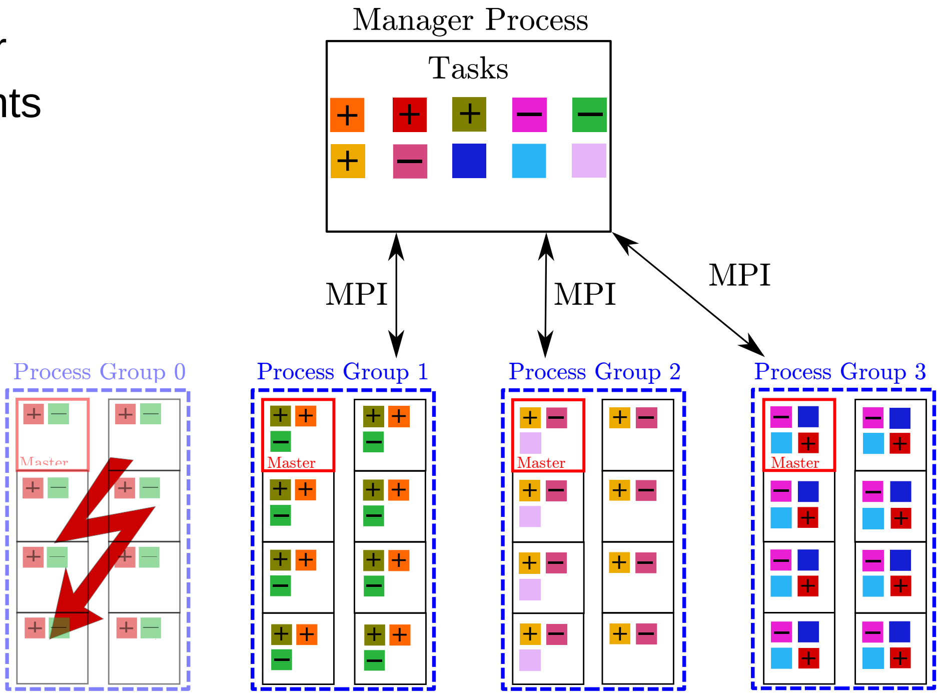
Recovery strategy (ULFM-like)

- Detect failed group during collective operation (combine)
- Exclude failed group(s) from communicator
- Compute alternative combination coefficients and perform combination*
- Redistribute lost tasks to living groups for next set of timesteps



Recovery strategy (ULFM-like)

- Detect failed group during collective operation (combine)
- Exclude failed group(s) from communicator
- Compute alternative combination coefficients and perform combination*
- Redistribute lost tasks to living groups for next set of timesteps



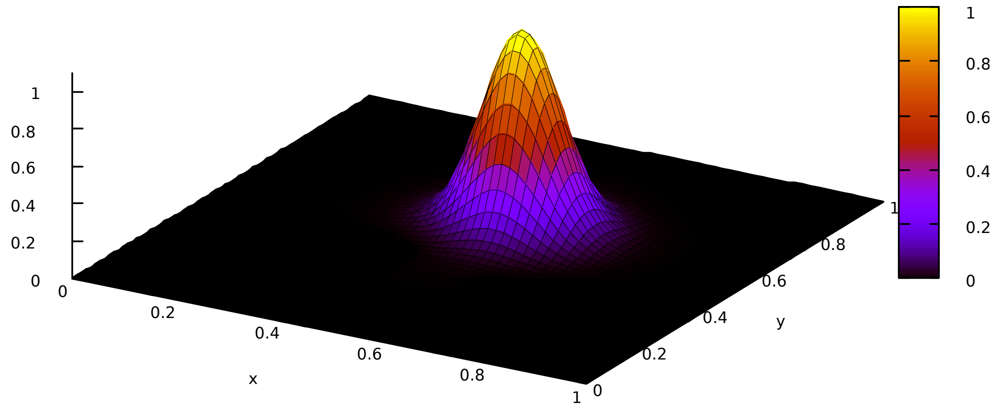
- Key questions:
  1) How good is the solution after faults compared to the solution without faults?
  2) What is the overhead of the fault tolerance functions? Does it scale?

Simulation scenario

- Solve *d*-dimensional advection-diffusion equation using *DUNE*[5]

$$\begin{aligned} \partial_t u - \Delta u + \vec{a} \cdot \nabla u &= f && \text{in } \Omega \times [0,T) \\ u(\cdot,t) &= 0 && \text{in } \partial\Omega \end{aligned}$$

with $\Omega = [0,1]^d$, $\vec{a} = (1,1,...,1)^T$ and $u(\cdot,0) = e^{-100\sum_{i=1}^{d}(x_i-0.5)^2}$



Simulation scenario

- Supercomputer *Hazel Hen (CRAY XC40)* at the *High-Performance Computing Center,* Stuttgart (#9 in Top 500, 2016)

Convergence experiments

Convergence experiments

- 2D and 5D: increase resolution of the Combination Technique and compare to reference full grid solution

Convergence experiments

- 2D and 5D: increase resolution of the Combination Technique and compare to reference full grid solution
- Compute 1,000 timesteps and combine after every timestep

Convergence experiments

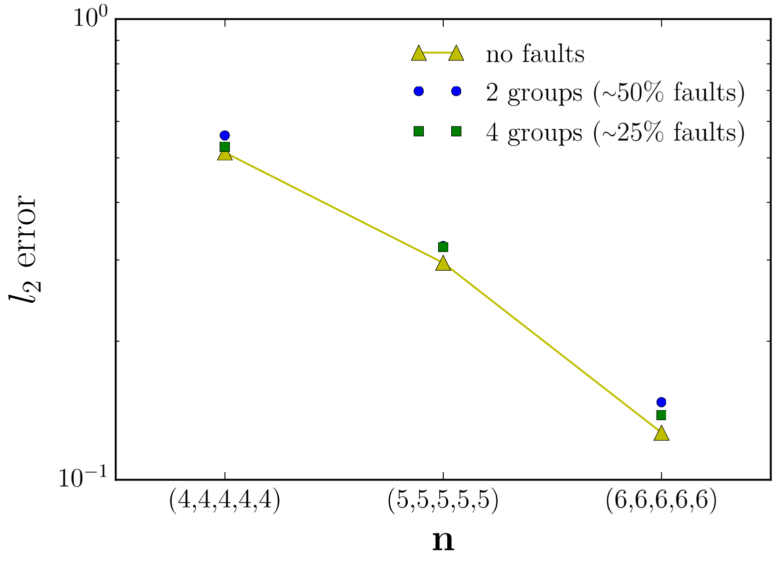
- 2D and 5D: increase resolution of the Combination Technique and compare to reference full grid solution
- Compute 1,000 timesteps and combine after every timestep
- Choose randomly which group fails (makes little difference)

Convergence experiments

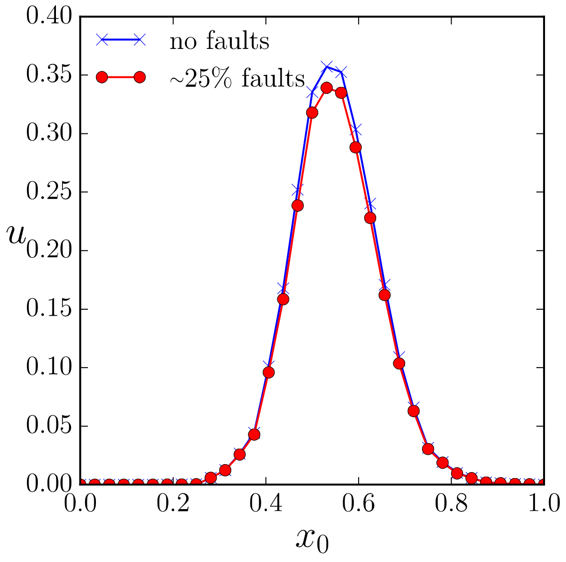
- 2D and 5D: increase resolution of the Combination Technique and compare to reference full grid solution
- Compute 1,000 timesteps and combine after every timestep
- Choose randomly which group fails (makes little difference)
- Inject fault at first iteration (worst case scenario)

Convergence experiments

- 2D and 5D: increase resolution of the Combination Technique and compare to reference full grid solution
- Compute 1,000 timesteps and combine after every timestep
- Choose randomly which group fails (makes little difference)
- Inject fault at first iteration (worst case scenario)



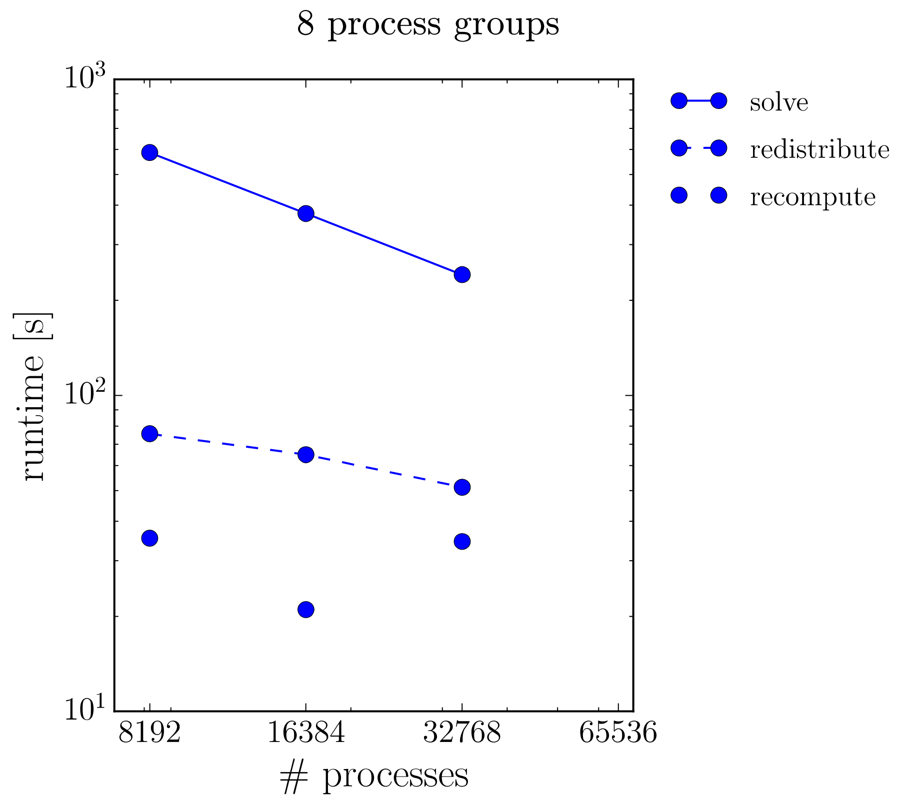
5D convergence



Cross section of 5D solution with and without faults

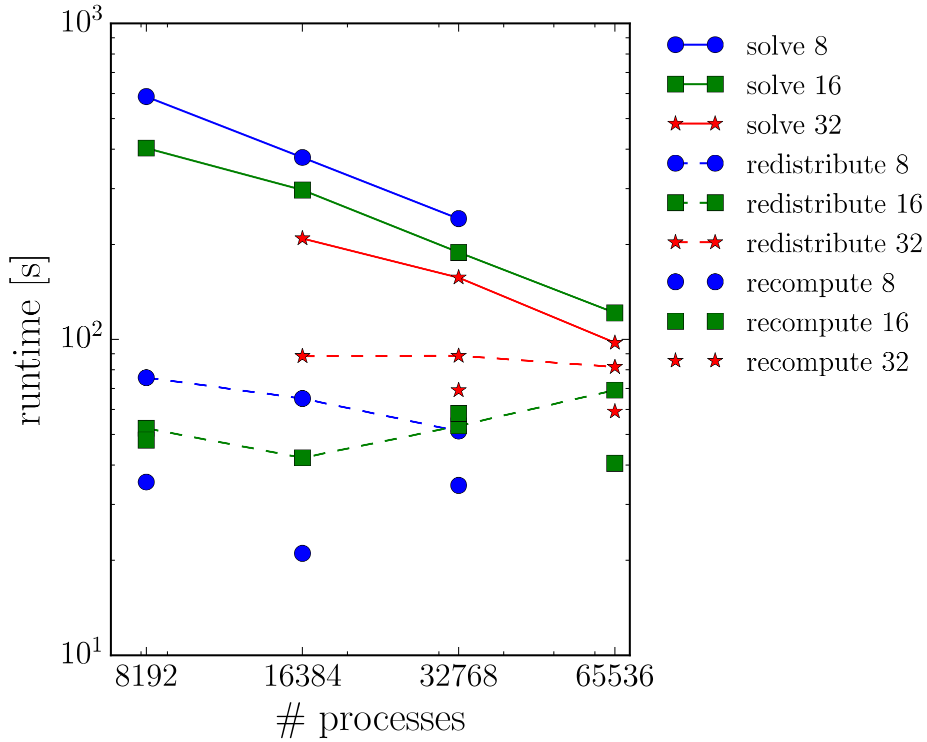
Strong scaling experiments

- Compare time to ***solve*** one timestep vs time to ***redistribute*** (and reinitialize) tasks vs time to ***recompute*** some tasks



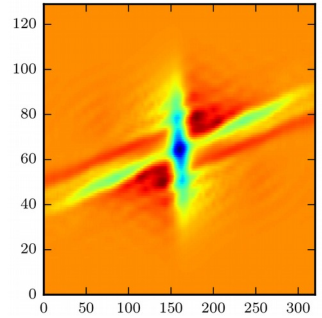
Strong scaling experiments

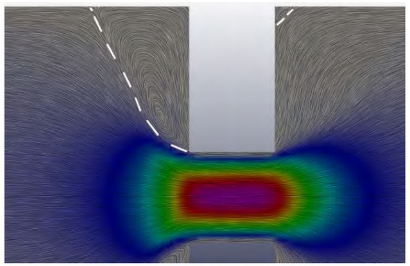
- Compare time to ***solve*** one timestep vs time to ***redistribute*** (and reinitialize) tasks vs time to ***recompute*** some tasks

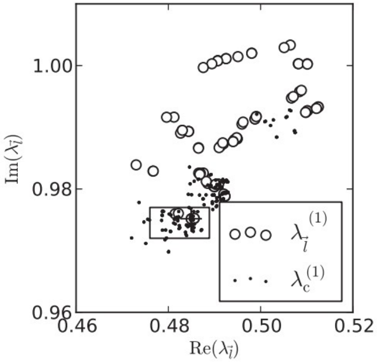


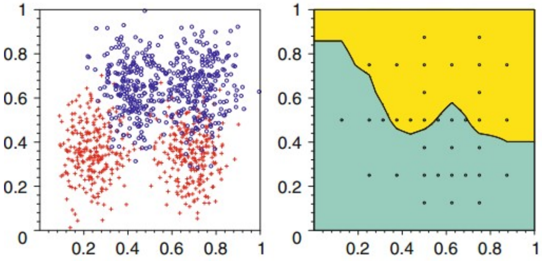
Takeaways

- Combination Technique is attractive for peta-/exascale:
  1. Solve PDEs (approximately) with high resolution (impossible with full grids) at the cost of many cheap solves (in parallel!)
  2. Offers algorithmic fault tolerance: no checkpoint/restarting, duplication, etc.



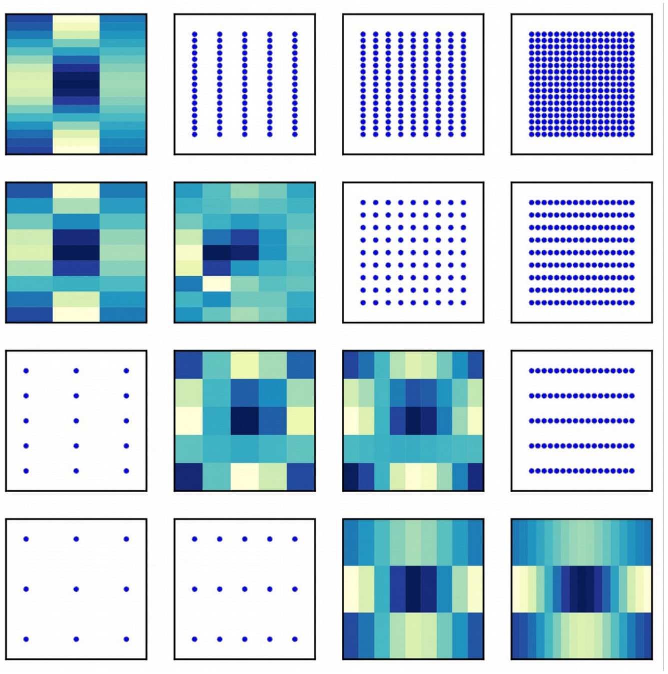


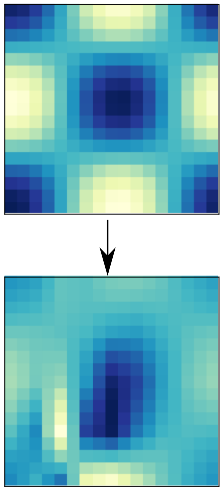




Ongoing: detection of silent data corruption [6]

Ongoing: detection of silent data corruption [6]





References

1. Jenko, F., et al.: *Electron temperature gradient driven turbulence*. Physics of Plasmas (1994-present) 7(5), 1904–1910 (2000), http://www.genecode.org/

2. Harding, B., et al.: *Fault tolerant computation with the sparse grid combination technique*. SIAM Journal on Scient. Comp. 37(3), C331–C353 (2015)

3. Parra Hinojosa, A., et al.: *Towards a fault-tolerant, scalable implementation of GENE*. In: Proceedings of ICCE 2014. LNCSE, Springer-Verlag (2015)

4. Strazdins, P.E., Ali, M.M., Harding, B.: *Highly scalable algorithms for the sparse grid combination technique*. In: Parallel and Distributed Processing Symposium Workshop (IPDPSW), 2015 IEEE International. pp. 941–950 (May 2015)

5. Bastian, P., et al.: *A generic grid interface for parallel and adaptive scientific computing*. Part I: Abstract framework. Computing 82(2-3), 103–119 (2008)

6. A. Parra Hinojosa, B. Harding, H. Markus and H.-J. Bungartz: *Handling Silent Data Corruption with the Sparse Grid Combination Technique*. Proceedings of the SPPEXA Symposium, LLNCSE. Springer-Verlag (February 2016).