The Sparse Grid Combination Technique for Resilient Extreme-scale Simulations

Peter Strazdins
School of Computing,
The Australian National University
(joint work with Mohsin Ali (National Computational Infrastructure),
Brendan Harding (University of Adelaide)
and Markus Hegland (The Australian National University))

(slides available from http://cs.anu.edu.au/~Peter.Strazdins/seminars)

10th International Conference on Advances in Computing and Communications, Kochi, India, 22 October 2021

1 Sparse Grid Combination as a Innovative Computational Method

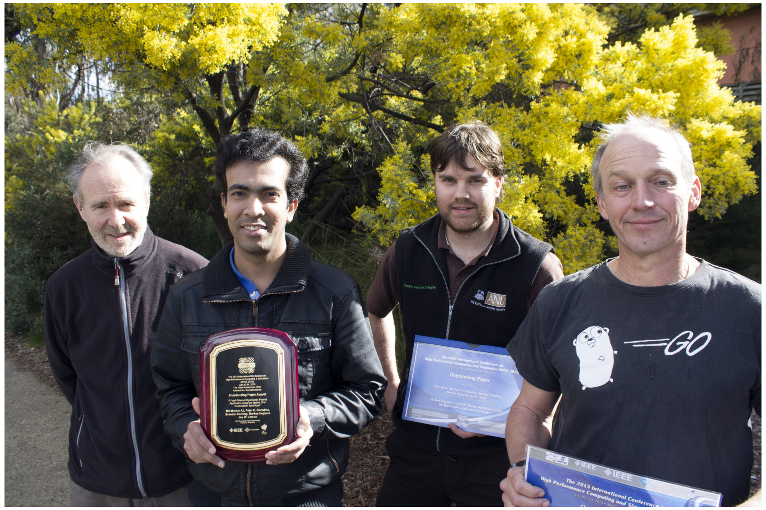
- a set of techniques from applied mathematics (the robust sparse grid combination technique)

  is combined with various aspects of parallel computing (d-dimensional parallel algorithms, resilient middleware)

  to make extreme-scale physical simulations:

  - faster (for a given accuracy)
  - and resilient (or fault-tolerant)

  The scalability of the simulation will also be enhanced



HPCS 2015 Outstanding Paper: the GENE gyrokinetic plasma application was made fault-tolerant using the SGCT

2 Talk Overview

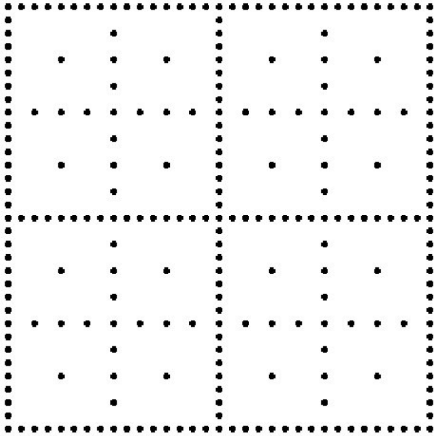
- background: why FT, solving PDEs via sparse grids with the combination technique, hierarchical surplus representation
- parallel sparse grid combination technique (SGCT) algorithms
  - mappings for the block distribution in d-dimensional space
  - direct SGCT algorithm: idea, components, overall
  - hierarchical surplus algorithm: forming surpluses, coalescing surpluses, direct SGCT extensions
  - limitations and extensions
- analysis & experimental results (on Raijin cluster, NCI National Facility)
- making real-world applications fault tolerant using the SGCT
  - general methodology
  - process recovery using ULFM MPI
  - GENE gyrokinetic plasma, Taxila Lattice Boltzmann method, Solid Fuel Ignition
- conclusions and future work

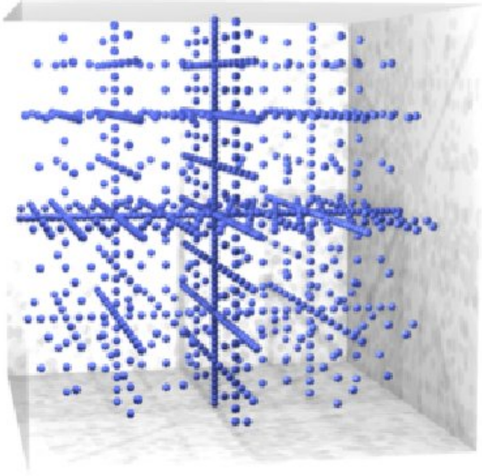
3 Background: Why Fault-Tolerance is Becoming Important

- exascale computing: for a system with n components, the mean time before failure is proportional to $1/n$
  - a sufficiently long-running application will *never* finish!
  - by 'failure' we usually mean a transient or permanent failure of a component (e.g. node) – this is called a hard fault
- cloud computing: resources (e.g. compute nodes) may have periods of scarcity / high costs
  - for a long-running application, may wish to shrink and grow the nodes it is running on accordingly – this scenario is also known as elasticity
- low power or adverse operating condition scenarios may cause failures even with moderate number of components
  - of typical interest are 'bit-flips' in memory or logic circuitry
  - these are termed as soft faults

4 Background: Sparse Grids

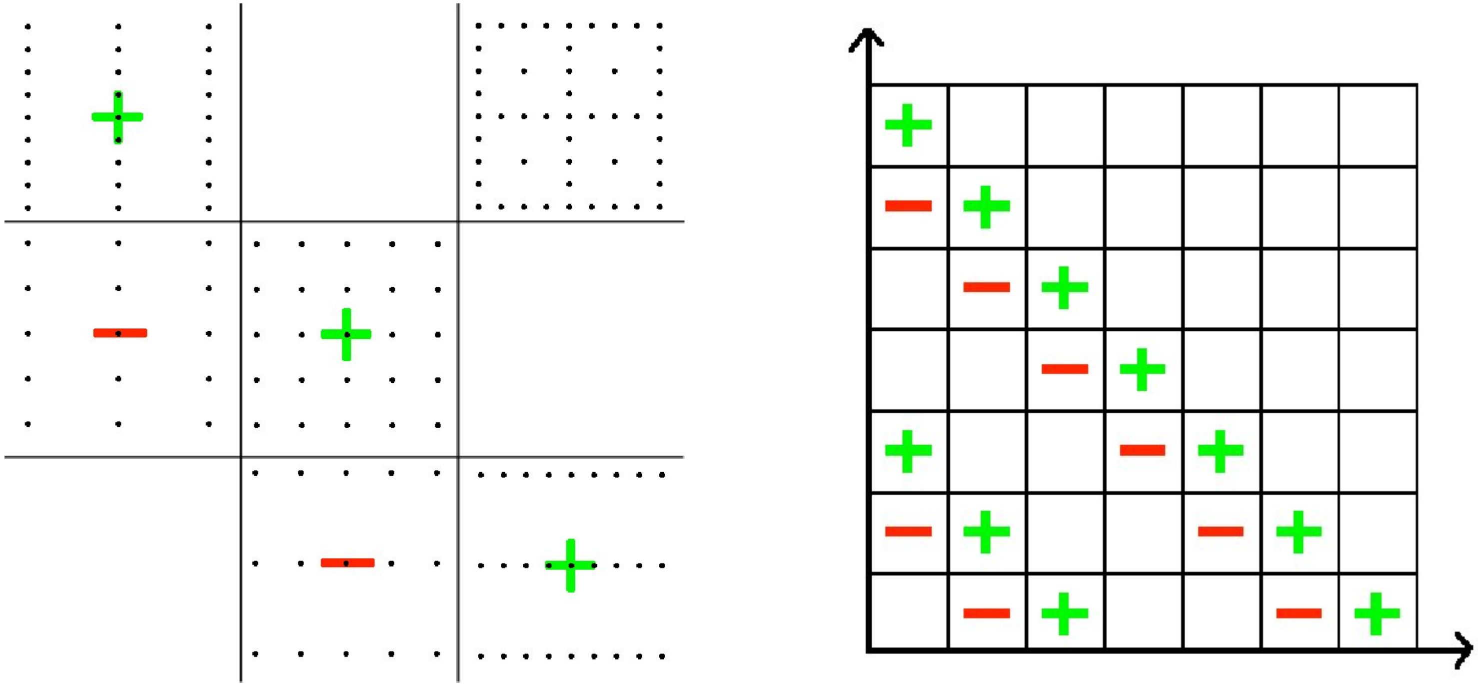
- introduced by Zenger (1991)
- for (regular) grids of dimension d having uniform resolution $n = 2^l + 1$ in all dimensions, the number of grid points is n^d
  - known as the *curse of dimensionality*
- a **sparse grid** provides fine-scale resolution
- can be constructed from regular sub-grids that are fine-scale in some dimensions and coarse in others
- has been proven successful for a variety of different problems:
  - good accuracy for given effort ($O(n\lg(n)^{d-1})$ points)
  - various options for fault-tolerance!





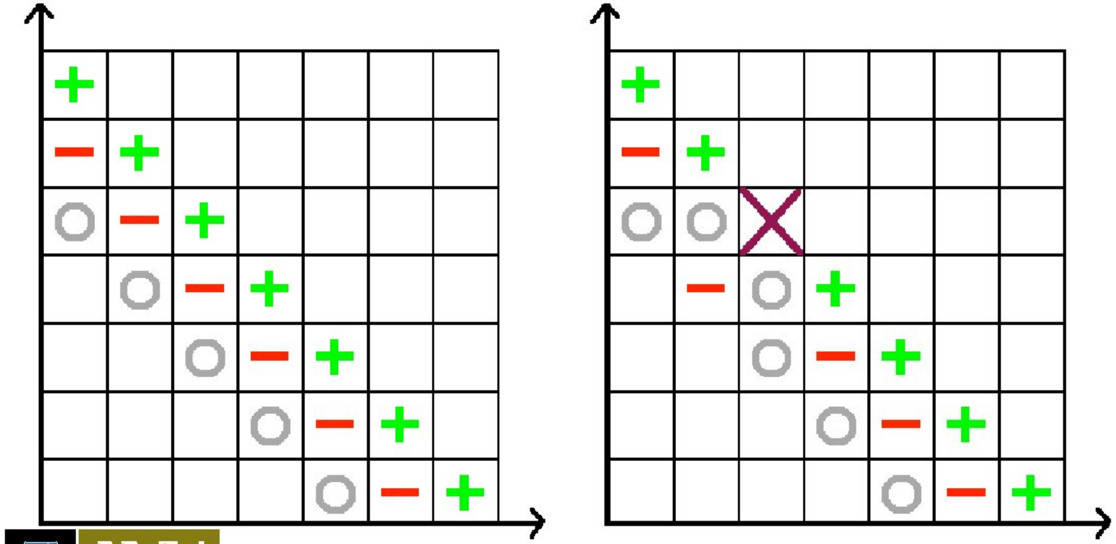
5 Background: Combination Technique for Sparse Grids

- computations over sparse grids may be approximated by being solved over the corresponding set of regular sub-grids
  - overall solution is from 'combining' sub-solutions via an inclusion-exclusion principle (complexity is still $O(n \lg(n)^{d-1})$ where $n = 2^l + 1$)
- for 2D at 'level' $l = 3$, combine grids $(3,1)$, $(2,2)$ $(1,3)$ minus $(2,1)$, $(1,2)$ onto (sparse) grid $(3,3)$ (interpolation is required)



6 Robust Combination Techniques

- uses extra set of smaller sub-grids
  - the redundancy from this is $< 1/(2(2^d - 1))$
- for a single failure on a sub-grid, can find a new combination formula with an inclusion/exclusion principle avoiding the failed sub-grid
- works for many cases of multiple failures (using a 4th set covers all)
- a failed sub-grid can be recovered from its projection on the combined sparse grid

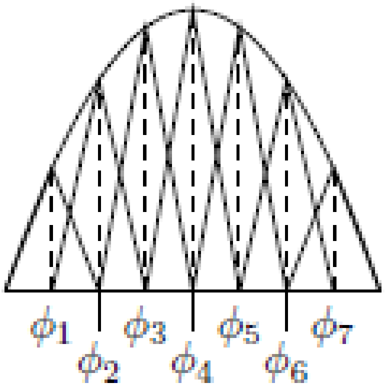


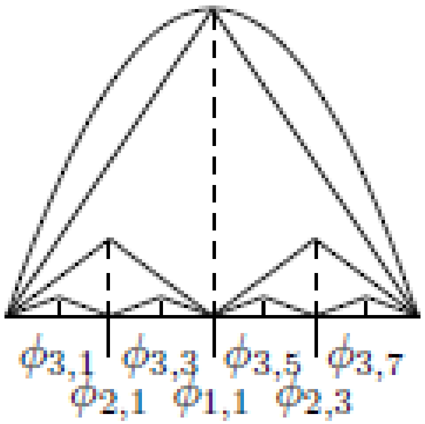
7 Background: Hierarchical Surplus Representation of Grids

- normally use a nodal representation: the value at point x_k is $v_k = f(x_k)$
- we can also use a hierarchical representation: the value at $x_{i,k} = x_{2^{l-i}.k}$ is the difference between $x_{i,k}$ and the average of its hierarchical neighbours

$$v_{i,k} = \begin{cases} f(x_{i,k}) - \frac{1}{2}\left(\begin{array}{l} f(x_{i-1,(k-1)/2}) \\ +f(x_{i-1,(k+1)/2}) \end{array}\right) & \text{for } i > 0 \\ f(x_{i,k}) & \text{for } i = 0 \end{cases}$$

- we can perform the combination algorithm on each of the component grid's common hierarchical surpluses (a grid of index (i, j) has $(i+1)(j+1)$ surpluses)
  - √ this reduces communication (surpluses corresp. to the upper diagonal are unique) and avoids interpolation





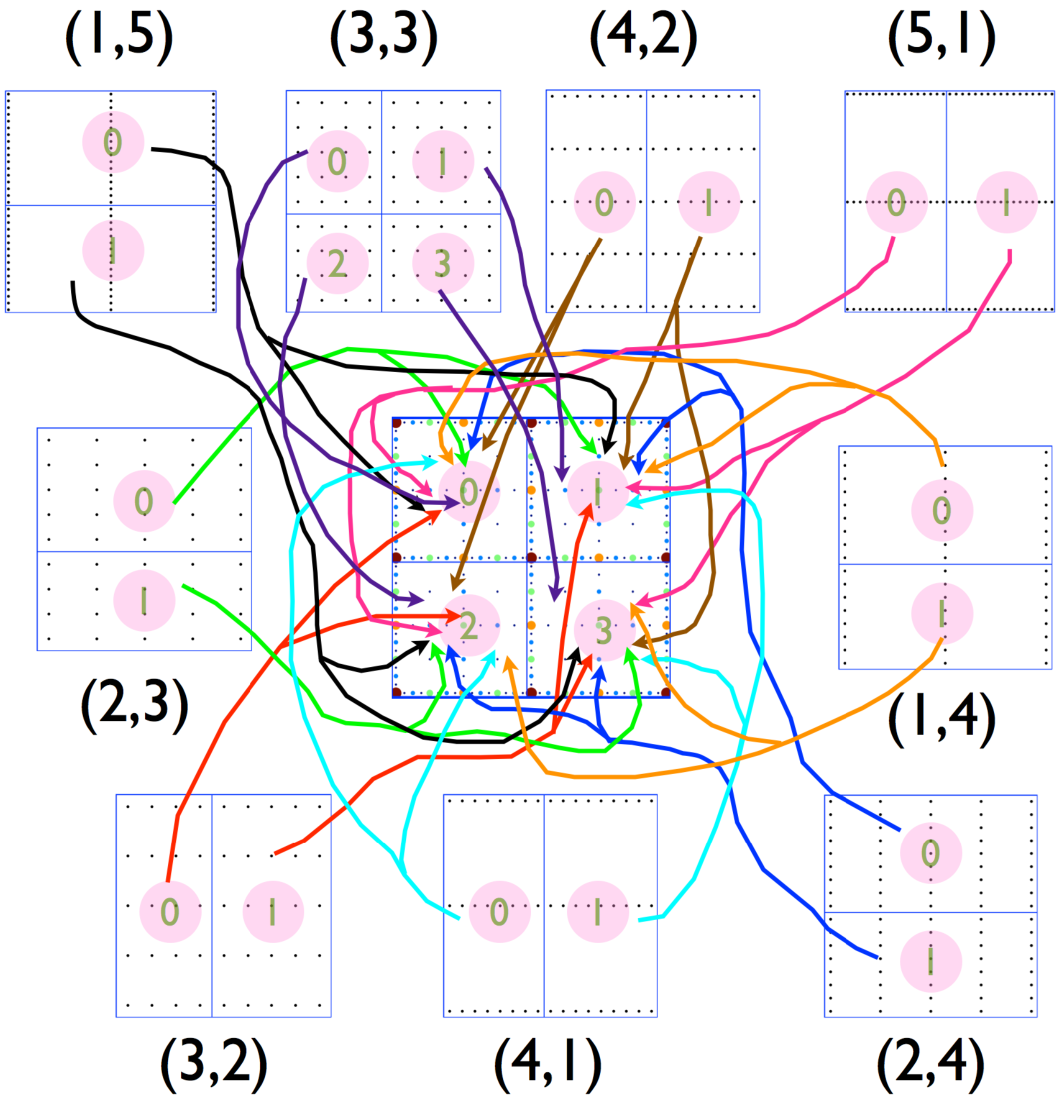
8 Background: Hierarchical Surplus Formation

- on a component grid (G_i), each element will correspond to a hierarchical surplus of index j, where $j \leq i$
- e.g. for $i = (3, 3)$

00	30	20	30	10	30	20	30	00
03	33	23	33	13	33	23	33	03
02	32	22	32	12	32	22	32	02
03	33	23	33	13	33	23	33	03
01	31	21	31	11	31	21	31	01
03	33	23	33	13	33	23	33	03
02	32	22	32	12	32	22	32	02
03	33	23	33	13	33	23	33	03
00	30	20	30	10	30	20	30	00

- the hierarchization process occurs in-place, with the surpluses computed from the initial grid values
- note that the size of surplus of index j is 2^j – independent of G_i
- hierarchical surpluses contain common information across different component grids

9 Direct SGCT Algorithm: the Gather-Scatter Idea



- evolve independent simulations over time T on a set of component grids, solution is a d-dimensional field (here $d{=}2$)
- each grid is distributed over a process grid (here these are 2×2, 2×1 or 1×2)
- gather: combine fields on a sparse grid (index $(5,5)$), here on a 2×2 process grid
- scatter: sample (the more accurate) combined field and redistribute back to the component grids

10 Mappings for the d-dimensional Block Distribution

- can be succinctly expressed in terms of d-dimensional vector arithmetic
  - for $d = 2, M = (M_x, M_y), N = (N_x, N_y) \in \mathbb{N}^2$, and $a \in \mathbb{N}$,

$M \le N \equiv (M_x \le N_x) \wedge (M_y \le N_y)$; $a \le N \equiv (a \le N_x) \wedge (a \le N_y)$

$M * N \equiv (M_x * N_x, M_y * N_y)$; $a * N \equiv (a * N_x, a * N_y)$

$\Pi(N) = N_x * N_y$

- we have the following mappings for the block-distribution of a global length $N \in \mathbb{N}^d$ over a process grid $P \in \mathbb{N}^d$,

for a process of id $p \in \mathbb{N}^d$, $0 \le p < P$,

and for a global index $\hat{N} \in \mathbb{N}^d$, $0 \le \hat{N} < N$:

$l(N, p, P) = n + (p == P - 1) * (N\%P)$: local length of N at process p

$g_0(N, p, P) = p * n$: global index of local index 0 at p

$p(\hat{N}, N, P) = \min(\hat{N}/n, P - 1)$: id of process holding global index $\hat{N}$

$o(\hat{N}, N, P) = \hat{N}\,\%\,n$: local offset within this process corresponding to $\hat{N}$

where $n = N/P$

THE AUSTRALIAN NATIONAL UNIVERSITY

◀◀ ◀ ● ▶ ▶▶

11 Direct SGCT Algorithm: Gather Stage

- for component grid of size N on process grid P; sparse grid is of size N' on process grid P' ($r = (N'-1)/(N-1)$): sending part is:

```
$\hat{N}' = r * g_0(N,p,P)$;                                   // scaled global starting index on $P$
$p' = p(\hat{N}', N', P')$;  $\hat{o}' = o(\hat{N}, N', P')$;  // process id & local offset on $P'$ . . .
                                                               // . . . for 1st message

$i$=0;  $n = l(N,p,P)$;
while $i_x < n_x$
    while $i_y < n_y$
        $o' = \hat{o}' * (i{=}{=}0)$;                          // local offset @ $p'$
        $n' = l(N',p',P') - o'$;                               // local size @ $p'$
        $dn = \min(n'/r, n-i)$;                                // local size here
        send local points $i : i + dn$ of $u$ to $p'$;         // extra points for interpolation
        $i_y$+=$dn_y$;  $p'_y$++;
    $i_x$+=$dn_x$;  $p'_x$++;
```

- receiving part is similar, except each component grids’ message is performed serially & received points are interpolated into the sparse grid

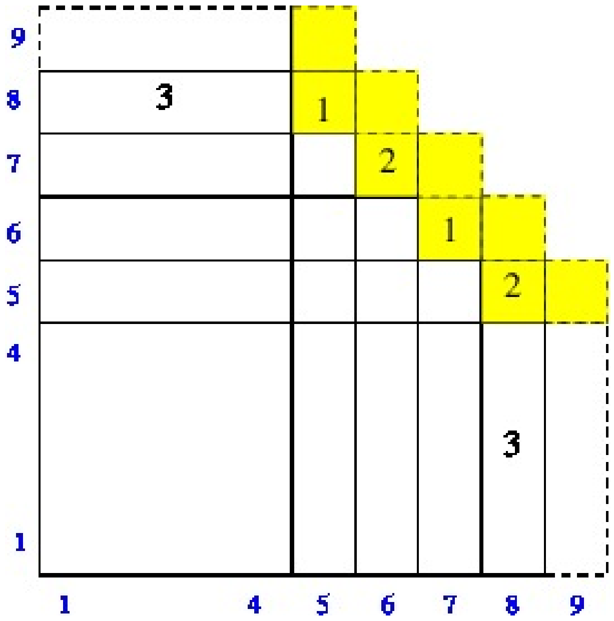
12 Direct SGCT Algorithm

- scatter stage, similar to gather (in reverse)
  - send stage on sparse grids' process grid *down-samples* respective points for each component grid
- for fault tolerance, a 3rd (smaller) diagonal of component grids is utilized
  - if a process on a component grid fails, a revised set of combination coefficients are supplied to the SGCT (with 0 for the failed grid)
  - the algorithm (and implementation) are otherwise unaffected
- only limitation in terms of process grid size of algorithm is that P' must be a power of 2
  - can be overcome if we send extra points to left for interpolation
- current implementation supports $d \leq 3$
  - main complexity for extending to larger d is in enumerating the component grids and the interpolation function
  - can deal with $d' > 3$ dim. fields if only d dims. are used for the SGCT
  - the gather is performed on a (partial) sparse grid data structure

13 Hierarchical Surplus-Based SGCT Algorithm

Overall algorithm:

1. hierarchize each component grid, in-place (independently)
   - involves $\Pi(\lg_2 N)$ send-receive stages
2. apply the (direct) SGCT over each hierarchical sub-space common to > 1 process grids
   - in each, only the process grids involved need participate
   - note that interpolation is *not* required as each surplus is the same size on each grid
3. un-hierarchize the surpluses to recover the original grids

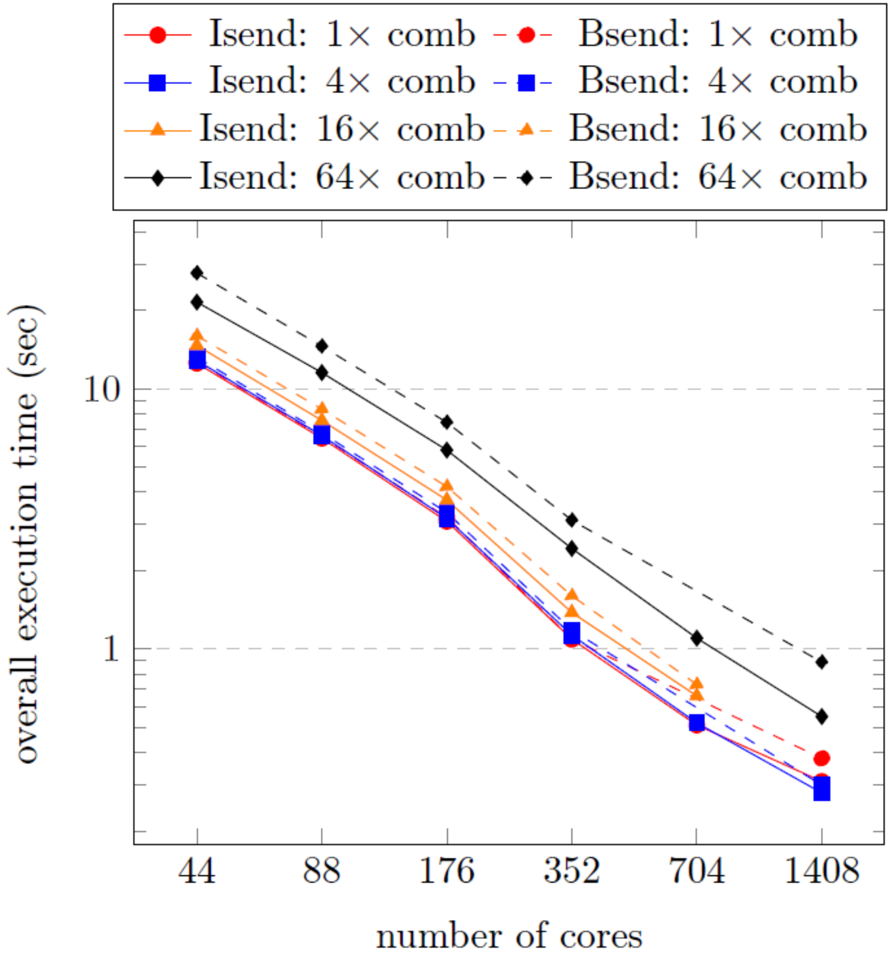


A 2D $l = 5$ SGCT on a sparse grid of index $(9, 9)$.
Indices of component grids are in yellow.
Can coalesce SGCT over sub-spaces to reduce overheads.

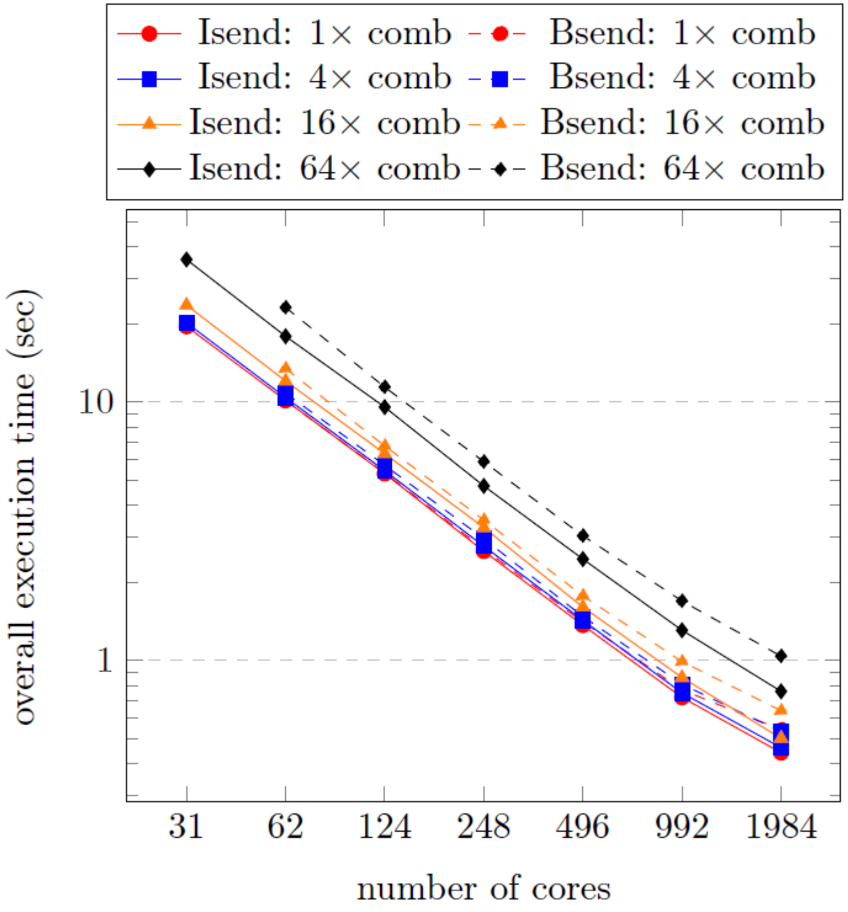
14 Analysis

- typical operating conditions of the SGCT:
  - the sparse grid's process grid P' comprises of a subset of processes from the process grids of the components (P_i)
  - assume P_i, P' are powers of 2 (required for hierarchical algorithm)
  - each sub-grid on a lower diagonal has half the processes as that above
- let $g = g(d, l) = O(l^{d-1})$ be the number of sub-grids involved, m denote the number of data points per process
- direct SGCT, each process in P' will receive $< 2m$ points, each process in each P_i sends and receives $\Pi(P'/P_i) \leq g$ messages
  - total cost is then $t \leq 2g\alpha + 3m\beta$
  - should be efficient for large m, but not for large g
- hierarchical SGCT avoids communication of $\frac{1}{2^d}$ of the surpluses
  - will have more startups even if coalesced, partially offset by a ≈30% lower average effective value of g
  - average degree of $||$ization $\approx 2/3$, but a load imbalance factor of $\approx 2^d$

15 Results: SGCT Advection Performance



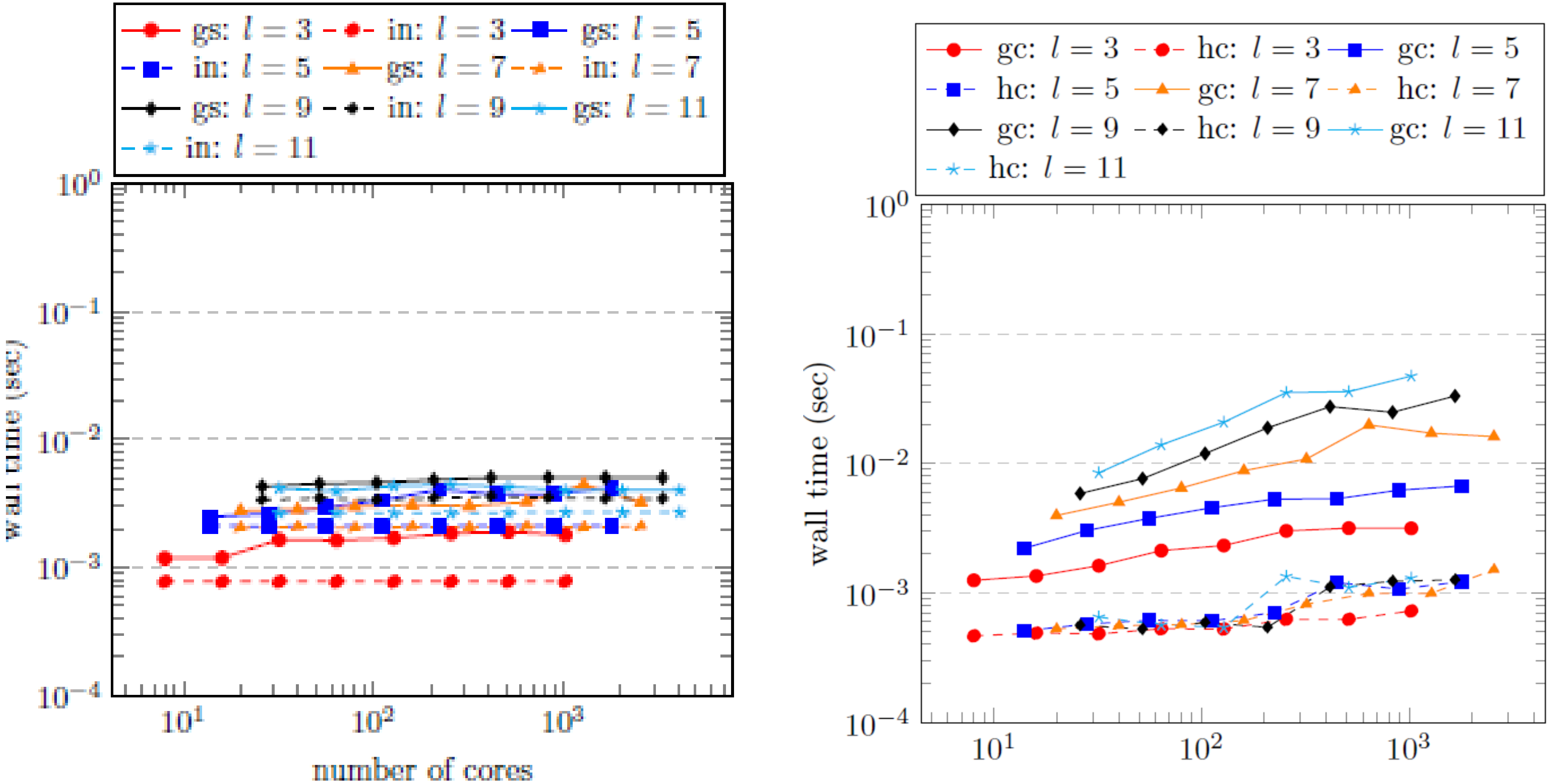
(a) 2D problem with $l = 4$ and a $2^{13} \times 2^{13}$ (sparse) grid, 1024 timesteps.



(b) 3D problem with $l = 3$ and $2^9 \times 2^9 \times 2^8$ grid, 1024 timesteps.

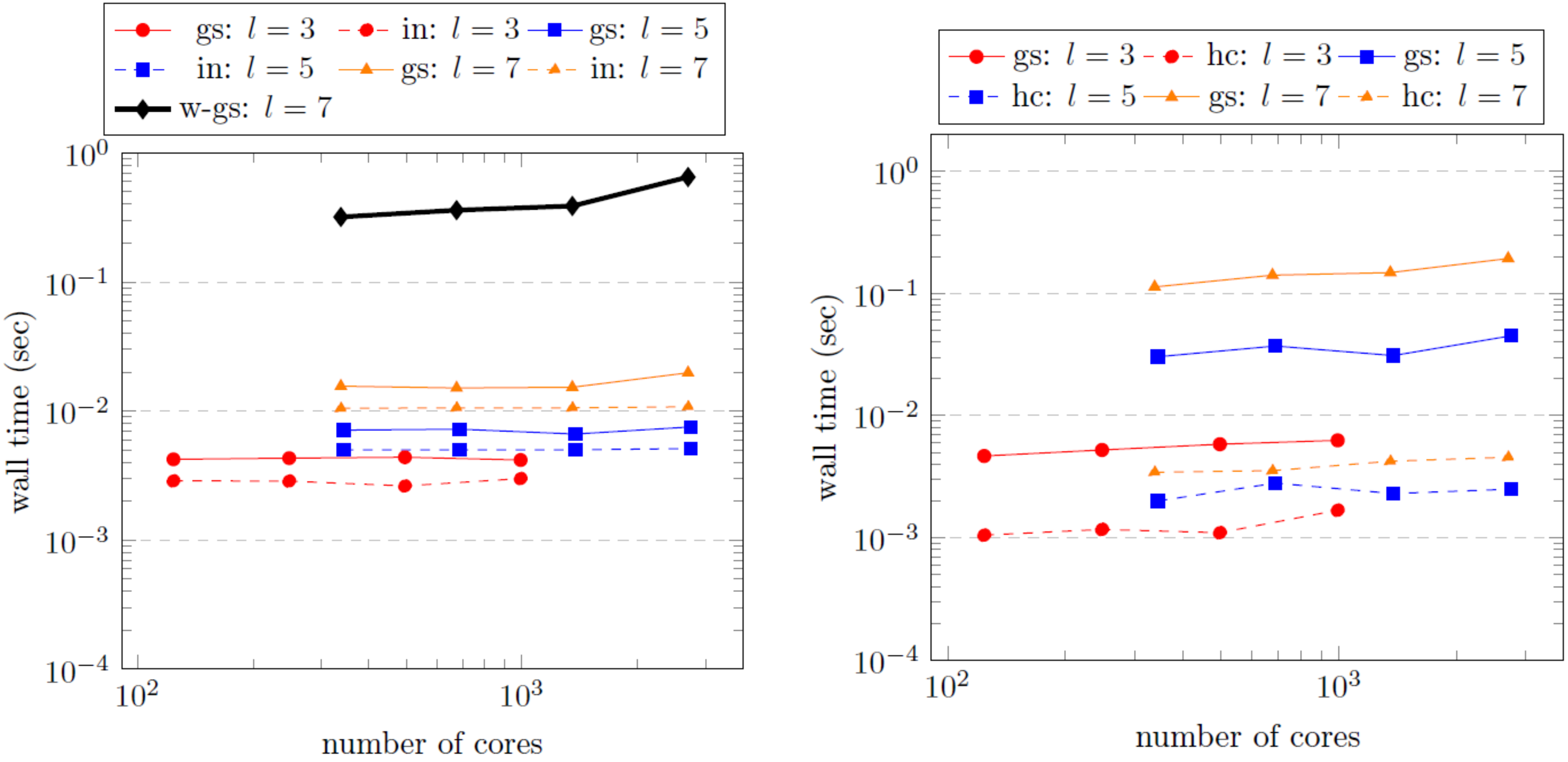
16 Results: 2D SGCT Algorithm Performance

Weak Scaling with $m = 2^{14}$ points per process for 2D SGCT performance (*after* a warmup run) with SGCT level l: direct (left) vs hierarchical



17 Results: 3D SGCT Algorithm Performance

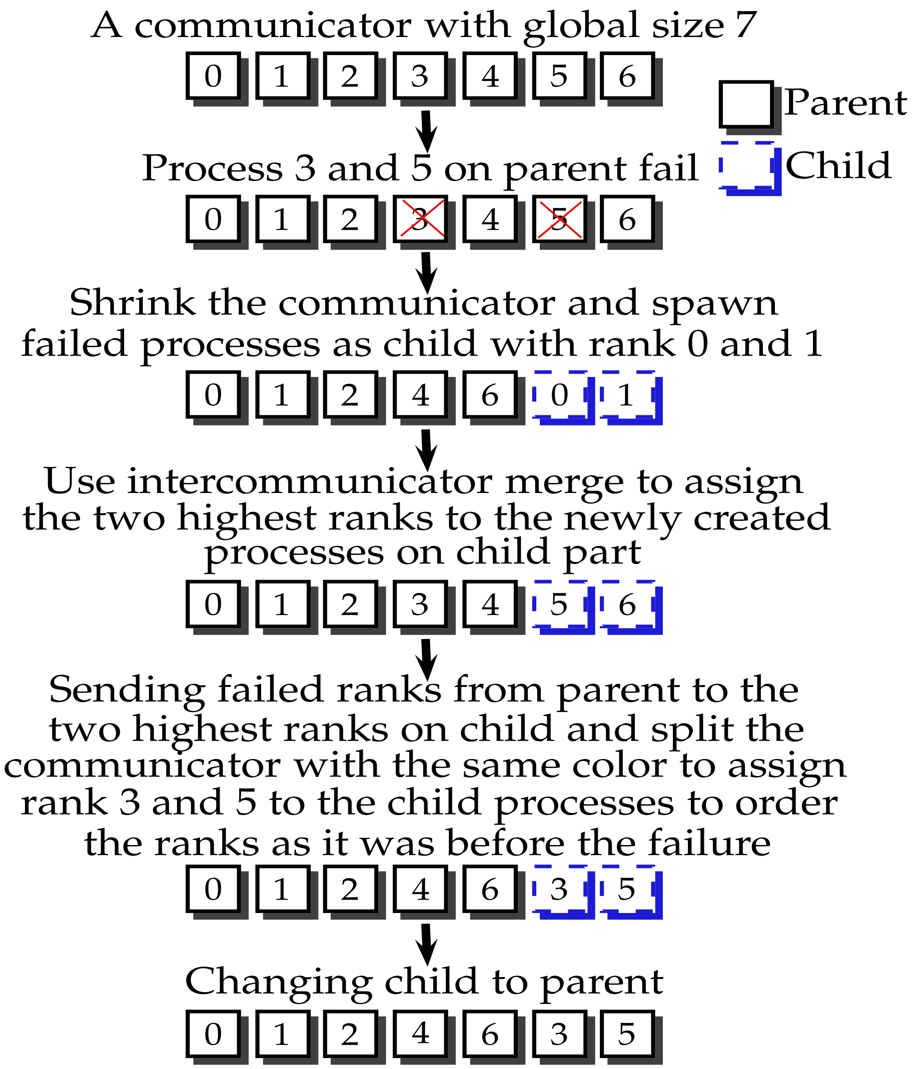
Weak scaling with $m = 2^{14}$ points per process for 3D SGCT performance (*after* a warmup run) with SGCT level l:



(a) direct algorithm

(b) hierarchical algorithm

18 Fault Recovery Procedure: Detect Failed Processes



A communicator with global size 7

0 1 2 3 4 5 6

Parent

Child

Process 3 and 5 on parent fail

0 1 2 ~~3~~ 4 ~~5~~ 6

Shrink the communicator and spawn failed processes as child with rank 0 and 1

0 1 2 4 6 0 1

Use intercommunicator merge to assign the two highest ranks to the newly created processes on child part

0 1 2 3 4 5 6

Sending failed ranks from parent to the two highest ranks on child and split the communicator with the same color to assign rank 3 and 5 to the child processes to order the ranks as it was before the failure

0 1 2 4 6 3 5

Changing child to parent

0 1 2 4 6 3 5

- can detect failed processes in ULFM MPI as follows:
  - attach an error handler ensuring failures get acknowledged on (original) communicator `comm`
  - call `MPI_Barrier(comm)`; if fails:
  - revoke it via `MPI_Comm_revoke(comm)` and create shrunken communicator via `OMPI_Comm_shrink(comm, &scomm)`
  - use `MPI_Group_difference(..., &fg)` to make a globally consistent list of failed processes

19 Fault Recovery Procedure: Process and Data

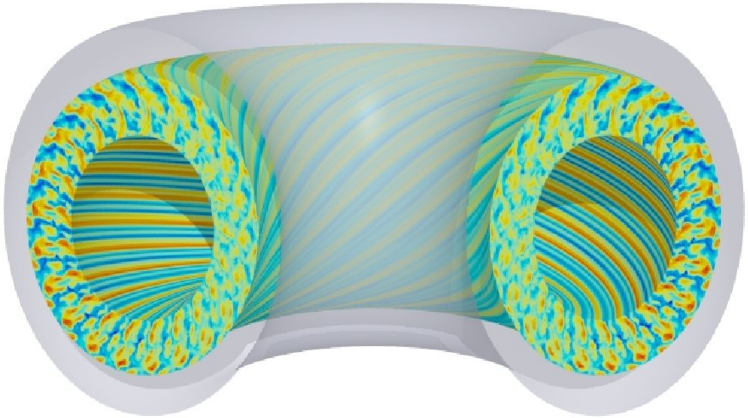
- process recovery in ULFM MPI:
  - use `MPI_Group_translate_ranks(fg, ..., comm, ...)` to re-rank remaining processes
  - spawn required number of failed processes via `MPI_Comm_spawn_multiple()`
    - these are called *child processes* and have own communicator
  - use `MPI_Intercomm_merge()` to merge child's comm. with parent's with `MPI_Comm_split()` to order the ranks
  - finally, `OMP_Comm_agree()` used to synchronize child and parent processes
- data recovery using the SGCT:

  must be done on whole of grid where a process has failed (data on non-failed process will be out-of-date)
  - identify lost grids; assign combination coefficient of 0 (do not participate in gather stage of SGCT)
  - receive down-sample of combined grid on the scatter stage

20 Methodology for Integrating the SGCT into an Application

```
$G = \{G_i\}$: set of sub-grids;
$C = \{C_i\}$: set of sub-grid communicators created from $W$;
$g = \{g_i\}$: set of fields returned from the application computed on $G$;
$u = \{u_i\}$: corresponding set of sub-grid solutions;
$u_I^c$: combined solution of the SGCT;
for each $C_i \in C$ do in parallel
    $u_i \leftarrow$ null; //will make runApplication() initialize $g_i$
for each required combination do
    for each $C_i \in C$ do in parallel
        $g_i \leftarrow$ runApplication$(u_i, G_i, C_i)$;
        $u_i \leftarrow g_i$; //on their common points
        updateBoundary$(u_i, C_i)$;
    reconstructFaultyCommunicator$(W)$; //using ULFM MPI
    $u_I^c \leftarrow$ gather$(u, W)$; //reconstructed grids don't participate
    $u \leftarrow$ scatter$(u_I^c, W)$;
```

21 The GENE Application



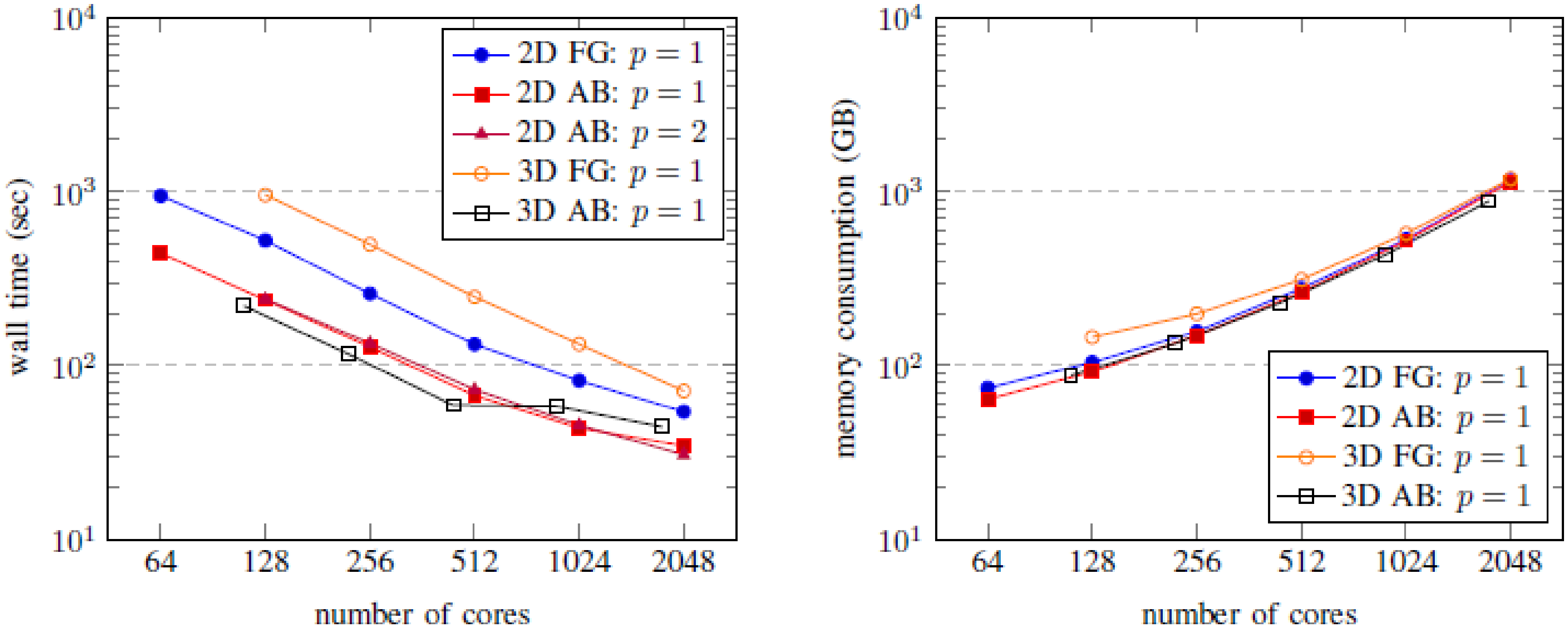
- GENE: Gyrokinetic Electromagnetic Numerical Experiment
  - plasma micro-turbulence code
  - multidimensional solver of Vlasov equation
  - fixed grid in five-dimensional phase space $(x_r, x_\perp, x_{||}, v_\perp, v_{||})$
- computes gyroradius-scale fluctuations and transport coefficients
  - these fields are the main output of GENE
- hybrid MPI/OpenMP parallelization – high scalability to 2K cores
- dimensions are limited to powers of two
- sparse grid combination technique has yielded good results!
  - physical system is relatively homogeneous

22 Incorporating the SGCT into GENE

- computes a density field `g_1`, stored in a double-precision array of dimensionality $(2, N_x, N_y, N_z, N_v, N_u, s)$, s is the number of 'species'
- the SGCT can be applied in any 2 or 3 contiguous dimensions

  e.g. for a 2D SGCT on N_v and N_u dimensions, we pass a block factor of $B = 2N_xN_yN_z$ to the SGCT algorithm, and iterate over s
- must pad dimensions of size 2^N to $2^N + 1$ for the SGCT: zero for v, u; for z, a 'shift' is required (using GENE routines)
- a parallelization of p over the non-SGCT dimensions is possible: perform p SGCT calculations in parallel
- a script creates different directories for each component grid to run in, and places an appropriately modified `parameters` file there
- `ISO_C_BINDING` & C wrappers to interface Fortran to C++ SGCT code
- small modifications to `rungene()` to pass down MPI communicator created by the SGCT constructor
- in `initial_value()`, code is added to pass `g_1` to the SGCT code

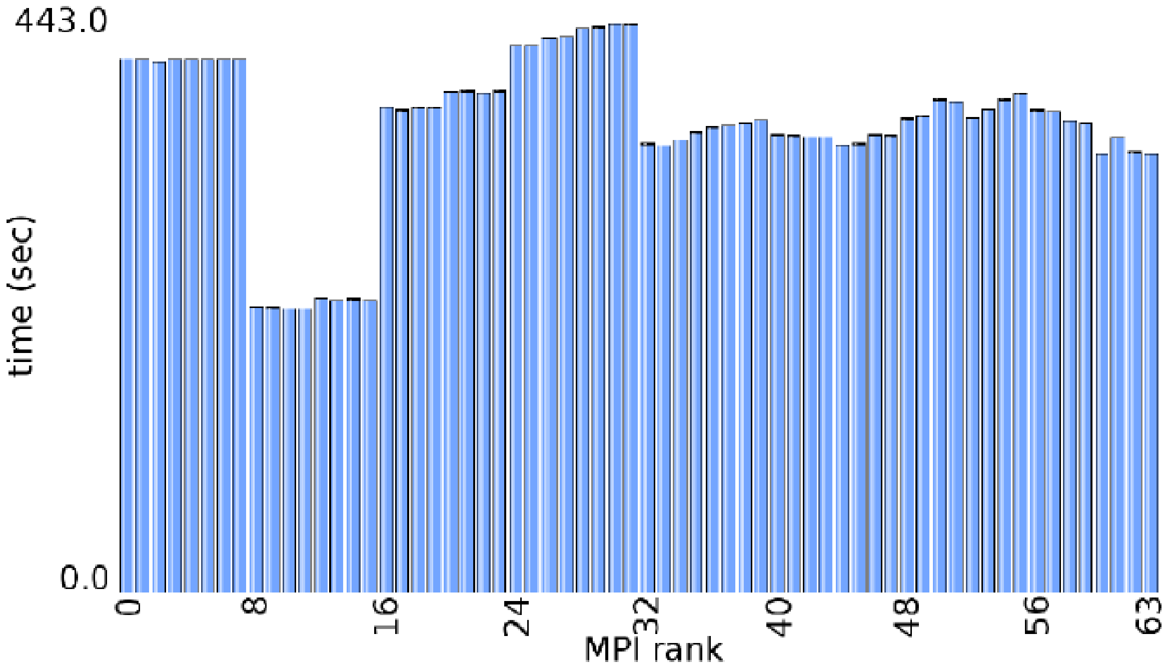
23 SGCT GENE Performance



- used `2d_big_6` with an $l = 5$ 2D SGCT over $(N_v, N_u) = (2^8, 2^8)$ and $N_x = 64, N_y = 4, N_z = 16, s = 1$, and `3d_big_6` with an $l = 4$ 3D SGCT over $(N_z, N_v, N_u) = (2^6, 2^8, 2^8)$ and $N_x = 32, N_y = 4, s = 1$. Run for 100 timesteps.
- SGCT (AB) has less work & storage than the corresp. full grid (FG)

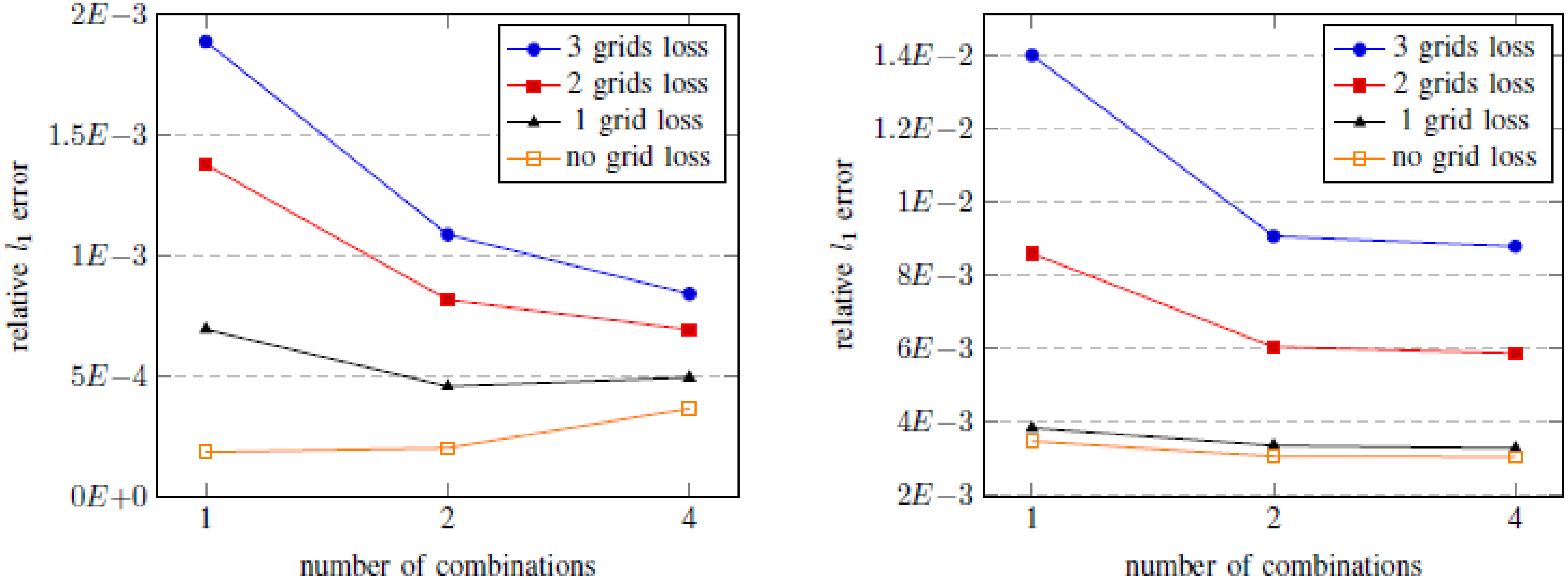
24 Load Balance for SGCT GENE

- general SGCT strategy to load balance across component grids
  - allocate p processes to uppermost diagonal grids, $\lceil \frac{p}{2} \rceil$ to next diag.
  - thus, no. of data points (hence work) per process should be equal
- however, data and process grid shape may affect computation and communication performance



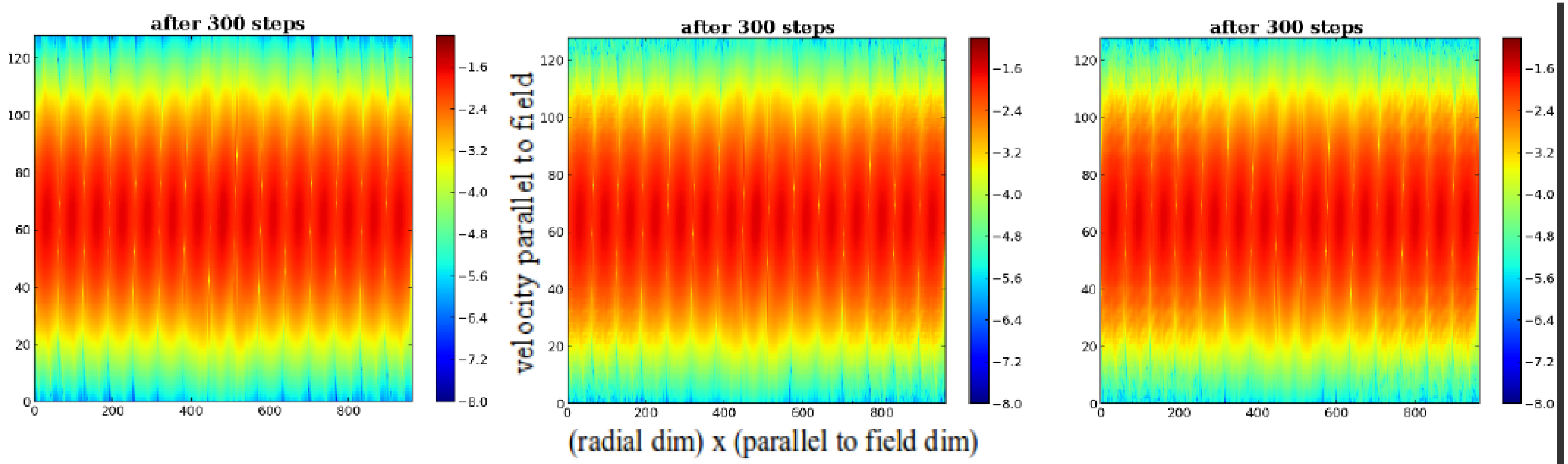
- TAU profile for 2D problem with $p = 8$
- 3D problem & other apps were similar

25 SGCT GENE Accuracy



- relative 1-norm error over full grid solution for 2D (left) and 3D (right)
- deemed ‘acceptable’
- multiple applications of the SGCT can reduce the error

26 SGCT GENE Accuracy - Visualization



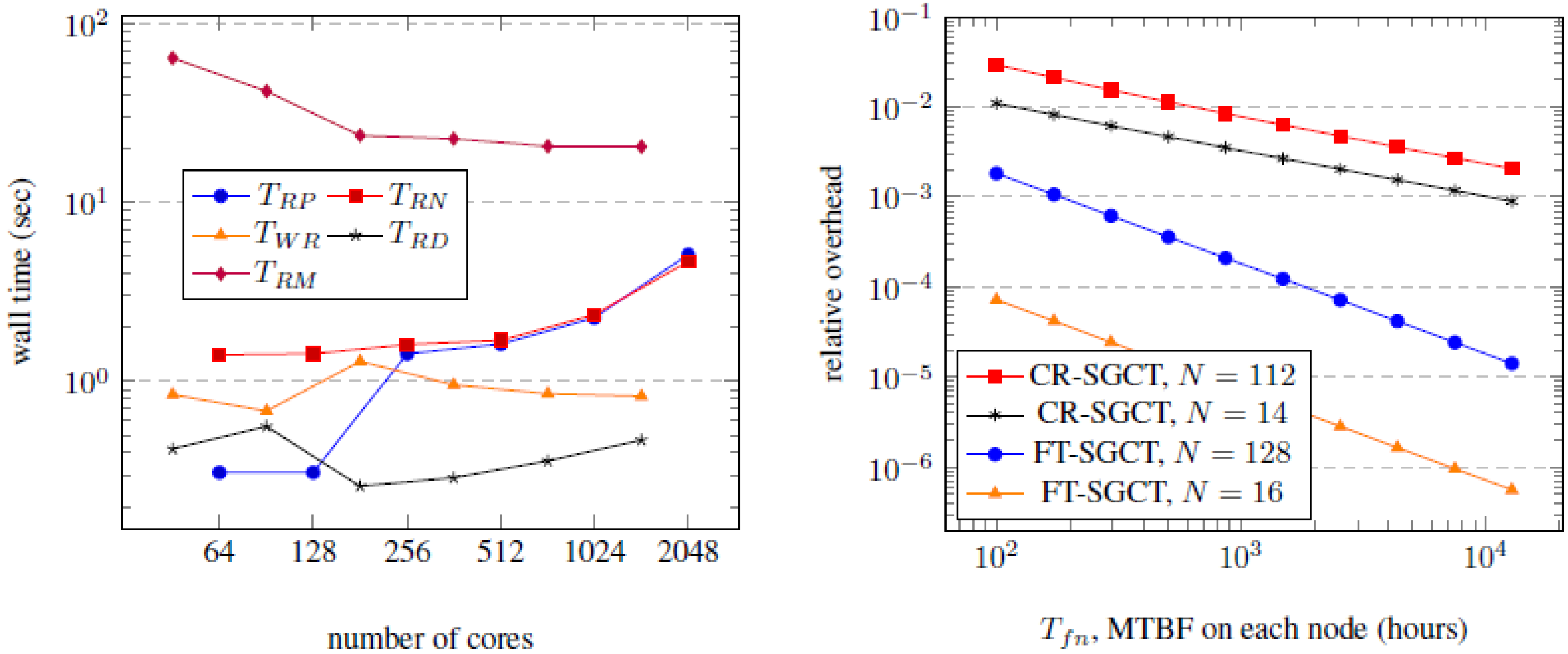
full grid field no failure

combined grid field
no failure
(relative l_1 error 5.0%)

combined grid field
3 grids lost
(relative l_1 error 6.22%)

- little discernible difference with or without faults

27 SGCT GENE Fault Recovery



(a) recovery overhead of a single occurrence of failure for shorter computation

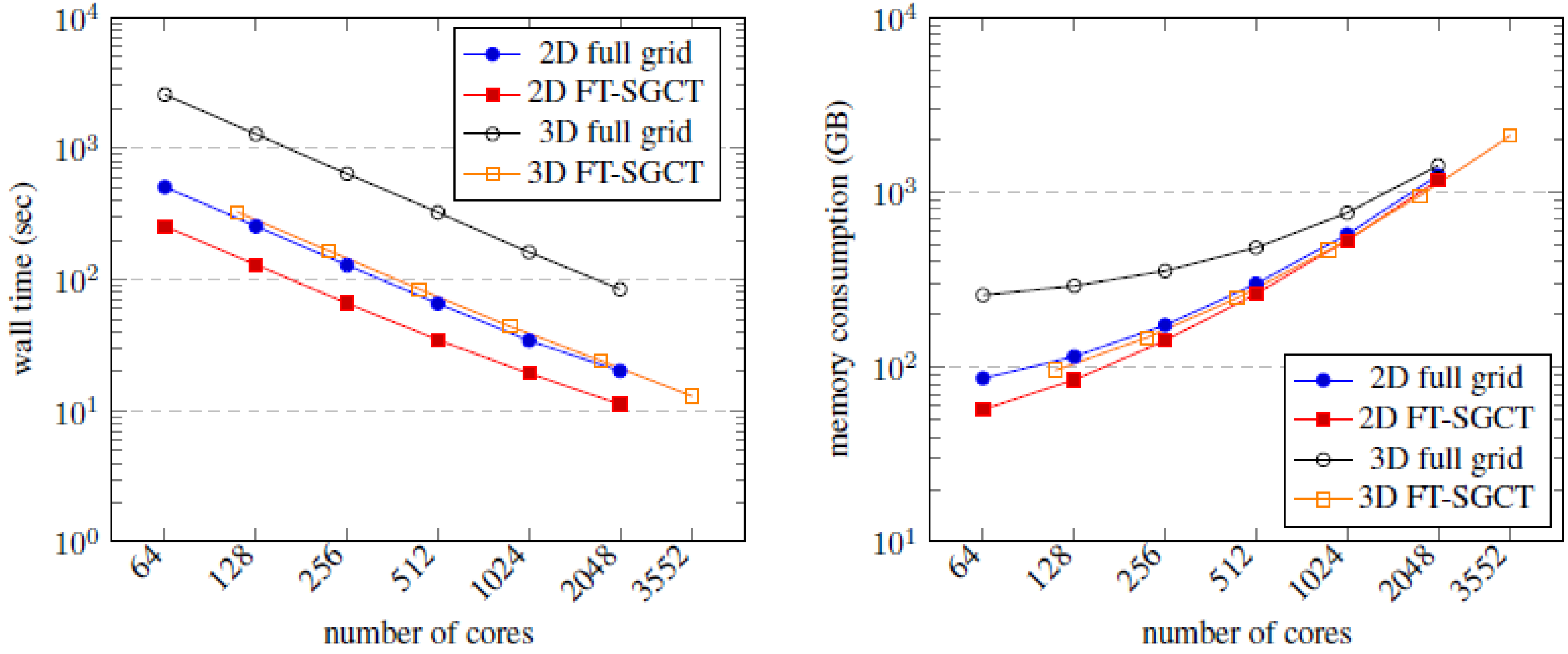
(b) expected relative recovery overhead for longer computation

- GENE has in-built checkpointing of `g_1` (note: very fast file system here!)
- WR/RD: read/write checkpoint, RM: relaunch MPI application
- RP/RN: recover process on same/different node
    - we should have $T_{RN} << T_{RM}$ (may improve in future ULFM MPI)

28 The Taxilla Lattice Boltzmann Method Application

- Taxila LBM is open source software for the LBM simulation of flow in porous and geometrically complex media
- highly scalable Fortran 90-based PetSc modular implementation
- chose a *bubble test*, in which one partially miscible fluid forms a bubble inside the other
- the density field is chosen for the output and used for the SGCT
- incorporating the SGCT similar to GENE, with $\{u_i\}$ corresponding to the `rho` array
    - default global communicators in `LBMCreate()` are replaced with C_i
    - process and data grid sizes are also passed in as parameters
    - local `rho` field extracted for SGCT after running `LBMRun()` using a shared pointer
    - periodic boundary conditions are used

29 SGCT Taxilla LBM Performance and Accuracy

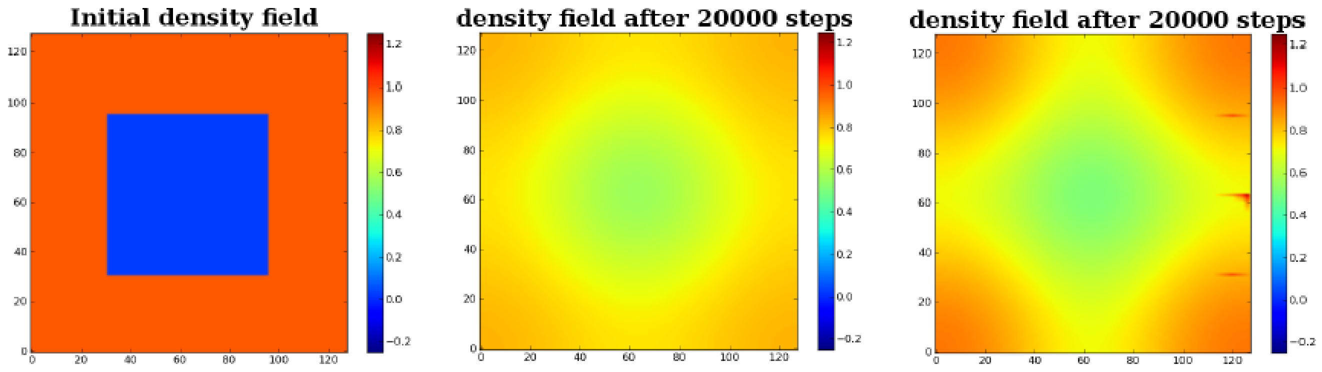


(a) overall execution time

(b) overall memory usage

- 2D problem has $2^{13} \times 2^{13}$ full grid size with $l = 5$; 3D has $2^9 \times 2^9 \times 2^9$ and $l = 4$. 200 timesteps.
- accuracy (relative 1-norm difference to full grid) is $1.13E^{-2}$ and $3.98E^{-2}$, respectively

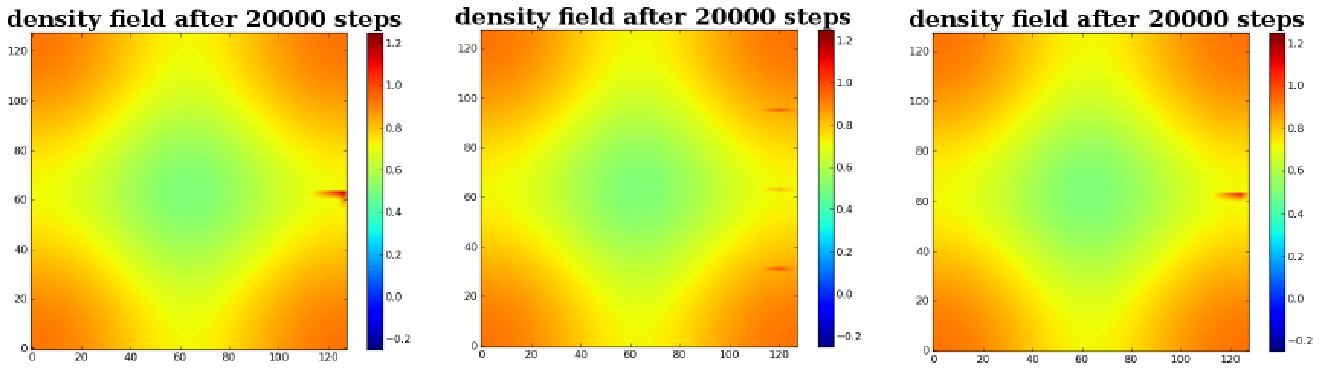
30 Taxilla Accuracy - Visualization



(a) initial full grid with no failure

(b) full grid with no failure

(c) combined grid with no failure (relative l_1 error is 5.0%)



(d) combined grid with processes on grid $G_{\underline{i}}$ with $\underline{i} = (1,3)$ fail (relative l_1 error is 5.07%)

(e) combined grid with processes on grid $G_{\underline{i}}$ with $\underline{i} = (3,1)$ and $(1,2)$ fail (relative l_1 error is 5.01%)

(f) combined grid with processes on grid $G_{\underline{i}}$ with $\underline{i} = (3,1), (1,3)$, and $(2,1)$ fail (relative l_1 error is 5.13%)

- comparison of density field for a $2^7 \times 2^7$ grid for an $l = 5$ SGCT
- smaller grid is used due to expense of computation

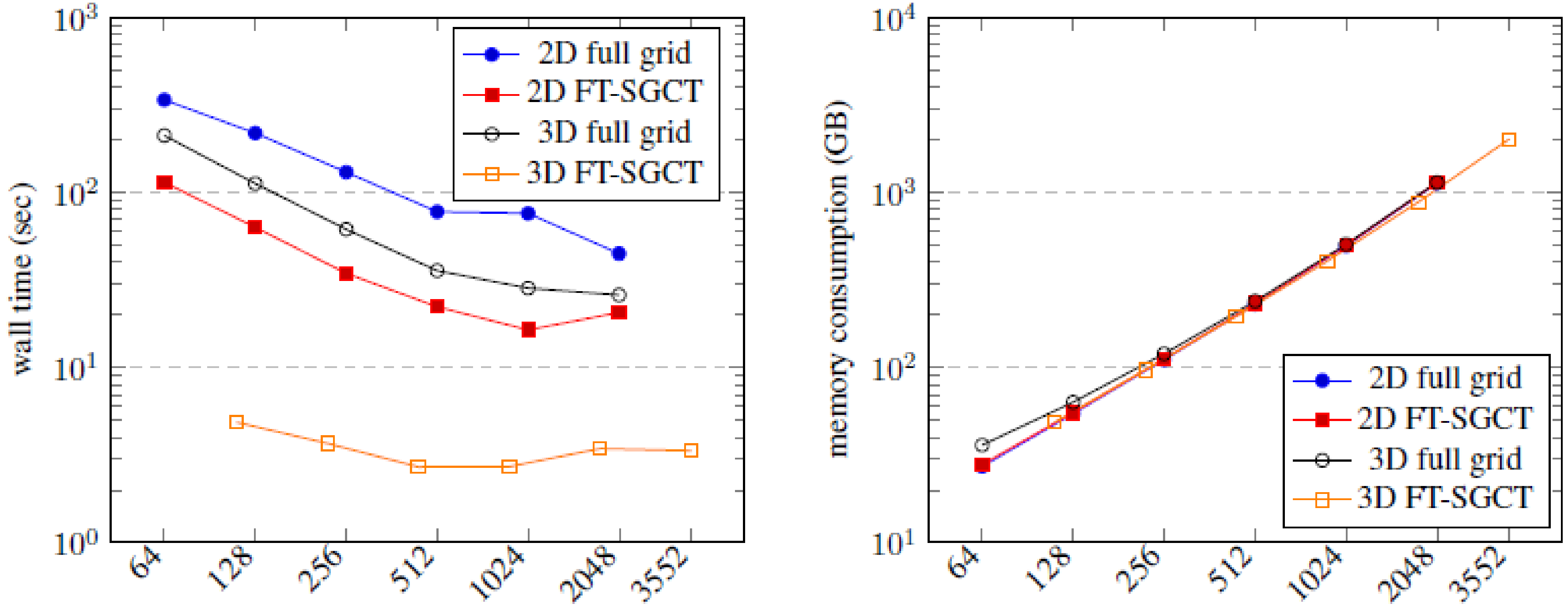
31 The Solid Fuel Ignition Application

- involves solving the Bratu problem

  $$-\Delta u(x,y,z) - \lambda \exp^{u(x,y,z)} = 0, 0 < x,y,z < 1$$

  where Δ is the Laplace operator and λ defines the degree of non-linearity
- a simpler application; also Fortran-90 PetSc code base
- incorporating the SGCT similar to Taxilla LBM, with $\{u_i\}$ corresponding to the `x` array in `SNESSolve()`
  - default global communicators in `SNESCreate()` and `DMDACreate2d()` are replaced with C_i
  - process and data grid sizes are also passed in as parameters to `DMDACreate2d()`
  - `c_get_sfi_field()` is called to pass the field to the SGCT codes
  - zero boundary conditions are used
- experiments used $\lambda = 6$ and Jacobian finite difference approximations

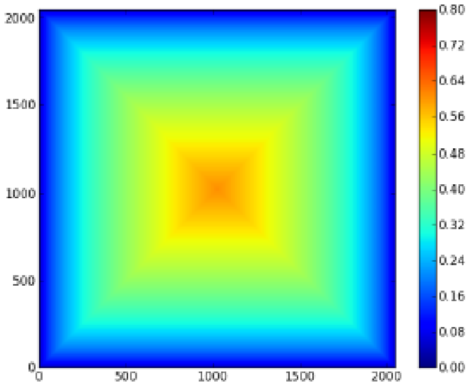
32 Solid Fuel Ignition: Performance and Accuracy



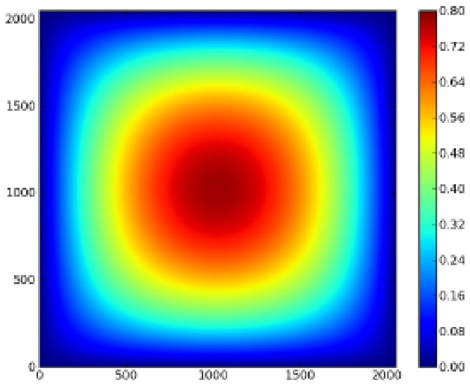
(a) overall execution time (b) overall memory usage

- 2D problem has $2^{11} \times 2^{11}$ full grid size with $l = 5$; 3D has $2^8 \times 2^8 \times 2^8$ and $l = 4$. 200 timesteps.
- 2D SGCT is $\approx 3\times$ faster, 3D $\approx 9\times$; accuracy is $1.27E^{-3}$ and $1.28E^{-3}$, respectively

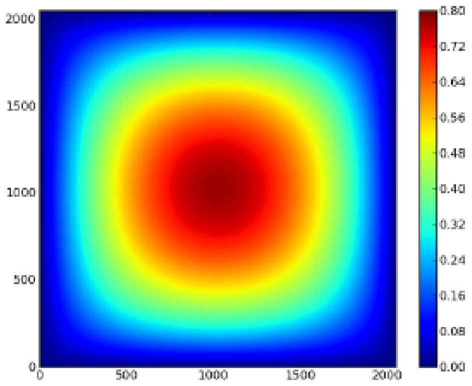
33 Solid Fuel Ignition: Accuracy - Visualization



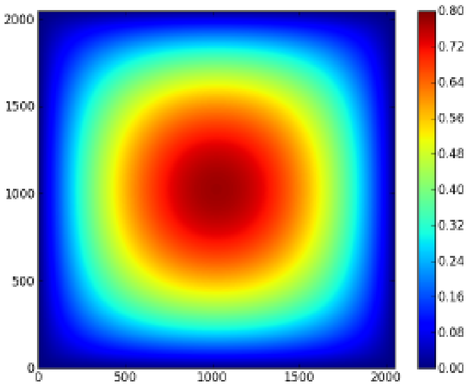
(a) initial full grid field with no failure



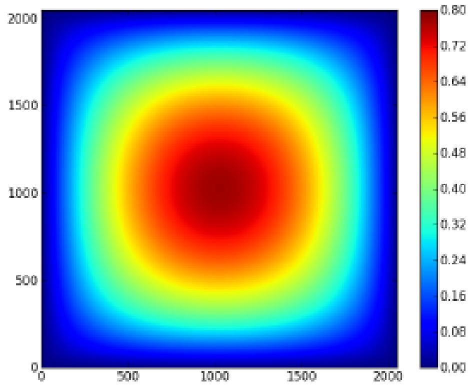
(b) full grid field with no failure



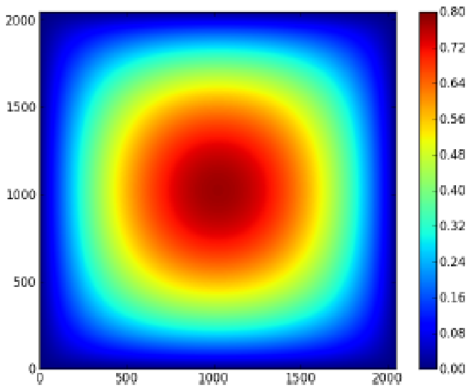
(c) combined grid field with no failure (relative l_1 error is 0.127%)



(d) combined grid field with processes on grid $G_{\underline{i}}$ with $\underline{i} = (1,3)$ fail (relative l_1 error is 0.127%)



(e) combined grid field with processes on grid $G_{\underline{i}}$ with $\underline{i} = (3,1)$ and $(1,2)$ fail (relative l_1 error is 0.111%)



(f) combined grid field with processes on grid $G_{\underline{i}}$ with $\underline{i} = (2,2), (0,4)$, and $(2,1)$ fail (relative l_1 error is 0.123%)

- comparison of field for a $2^{11} \times 2^{11}$ grid for an $l = 5$ SGCT

34 Conclusions (I)

- the SGCT can give good accuracy-performance tradeoffs on a range of PDE simulations
  - with little extra computational cost, it can also be made fault-tolerant!
  - current ULFM MPI infrastructure is sufficient to support this
- the first fully parallel SGCT algorithms have been developed for 2D&3D
  - complexity managed by vector arithmetic description
  - sparse grid data structured needed for direct algorithm, coalescing of surpluses needed for the hierarchical
  - the direct algorithm is faster and is very scalable with core courts; also more scalable with level l and dimensionality d
    - if fields are already hierarchized, recommend de-hierarchizing and using the direct algorithm
  - algorithms designed for high resolution grids on smaller l and d
  - codes are available from http://users.cecs.anu.edu.au/∼peter/projects/sgct

35 Conclusions (II)

- a methodology to incorporate the SGCT has been proven on 3 complex pre-existing applications
  - relatively modest source code modifications required
  - for GENE, a level of $l = 5$ ($l = 4$) for 2D (3D) gave $2\times$ (5–$9\times$) speed benefit for an ‘acceptable’ loss of accuracy
  - multiple SGCT can reduce error loss, especially for multiple failures
  - SGCT recovery time compares favourably to checkpointing
  - system is robust to multiple failures and combinations
  - GENE, Taxilla LBM and SFI are successful case studies!
- the SGCT is ready to support exascale computing!

36 Future Work

- currently, we *restart* failed processes (on same node or spare nodes). An alternate approach is to ‘shrink’ the process grids on failure: the SGCT can perform the necessary data distribution for free!
- test the methodology on other applications
  - solution must be ‘smooth’ for the SGCT to be effective
- SGCT algorithm can be extended to higher d (4D version by Si Xu, 2020)
- apply the SGCT to handle soft faults
  - detection may be challenging: ‘smearing’, application dependence
  - combine point-wise, in blocks or whole grids?
  - the hierarchical algorithm has a major advantage: common information in the component grids can be directly compared
  - more challenging time and memory requirements are likely

Thank You!! …Questions??? Comments???

Acknowledgements:

- NCI National Facility, for access to the Raijin cluster
- Australian Research Council for funding under Linkage Project LP110200410
- Fujitsu Laboratories Europe, for funding as a collaborative partner
- colleagues Jay Larson and Chris Kowitz for advice

Publications:

- Md Mohsin Ali, Peter E. Strazdins, Brendan Harding, Markus Hegland, J. Walter Larson, *A Fault-Tolerant Gyrokinetic Plasma Application using the Sparse Grid Combination Technique*, Proceedings of the 2015 International Conference on High Performance Computing & Simulation (HPCS 2015), pp499-507, Amsterdam, July 2015. **(Outstanding Paper Award)**.
- Peter E. Strazdins, Md Mohsin Ali and Brendan Harding, *Design and Analysis of Two Highly Scalable Sparse Grid Combination Algorithms*. Journal of Computational Science, 17(3):547-561, Nov 2016.
- M.M. Ali, P.E. Strazdins, B. Harding, and M. Hegland, *Complex scientific applications made fault-tolerant with the sparse grid combination technique*, International Journal of High Performance Computing Applications, 30(3):335–359, Aug 2016.