Efficient Local Resorting Techniques with Space Filling Curves

Applied to a Parallel Tsunami Simulation Model

Natalja Rakowsky and Annika Fuchs
AWI, Tsunami-Modelling-Group

The 10th International Workshop on Multiscale (Un-)structured Mesh Numerical Modelling for coastal, shelf and global ocean dynamics

Alfred Wegener Institute for Polar and Marine Research

Bremerhaven, 22 - 25 August 2011

Outline

- introducing TsunAWI
- motivation for resorting
- construction of Hilbert space filling curve (SFC) ordering
- comparison to other sortings
- conclusions

The AWI Tsunami Modell TsunAWI

TsunAWI in a nutshell

- shallow water equations with inundation
- unstructured $P_1 - P_1^{\mathrm{NC}}$ finite element grid
- explicit time stepping scheme
- OpenMP parallel Fortran90 code

The AWI Tsunami Modell TsunAWI

TsunAWI in a nutshell

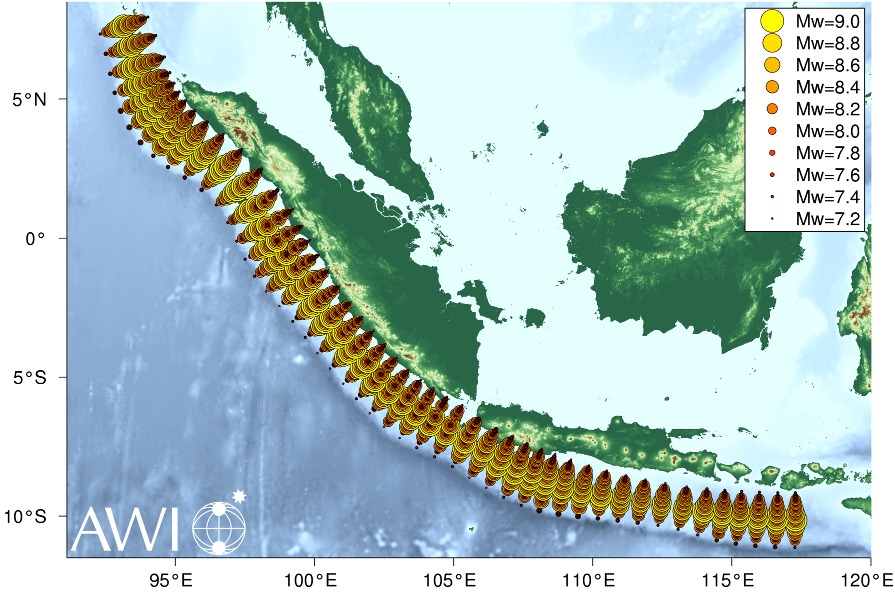
- shallow water equations with inundation
- unstructured $P_1 - P_1^{NC}$ finite element grid
- explicit time stepping scheme
- OpenMP parallel Fortran90 code

Most important application:

German-Indonesian Tsunami Early Warning System GITEWS

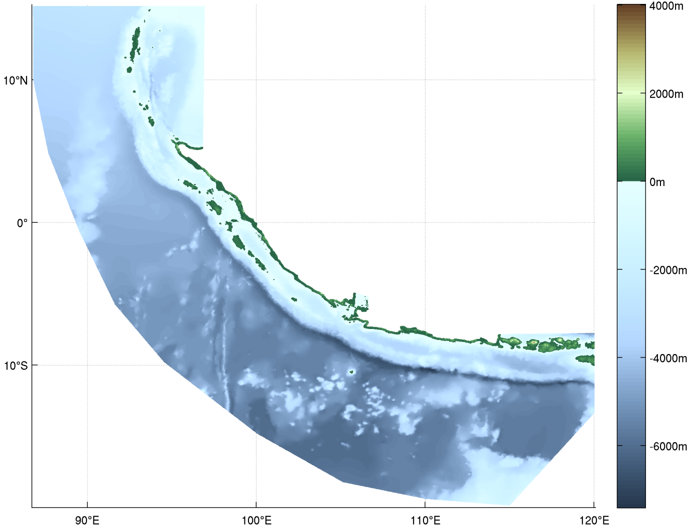
- 3470 scenarios for different prototypic ruptures
- 3h modeltime (10.800 timesteps of 1s)

TsunAWI Scenario Repository for GITEWS, March 2011
scenarios for 3470 prototypic ruptures (RuptGen2.1)



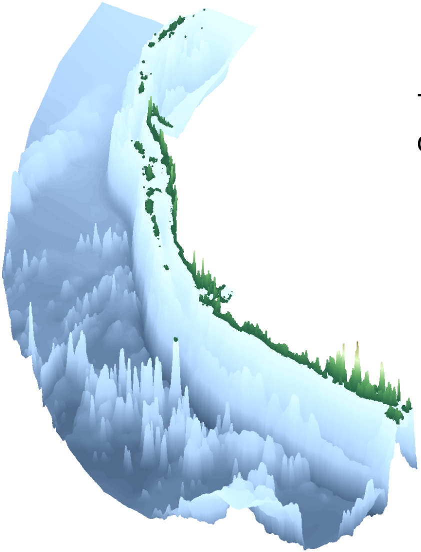
TsunAWI: example for a computational domain

regional grid for the Sunda Arc



TsunAWI: example for a computational domain

regional grid for the Sunda Arc

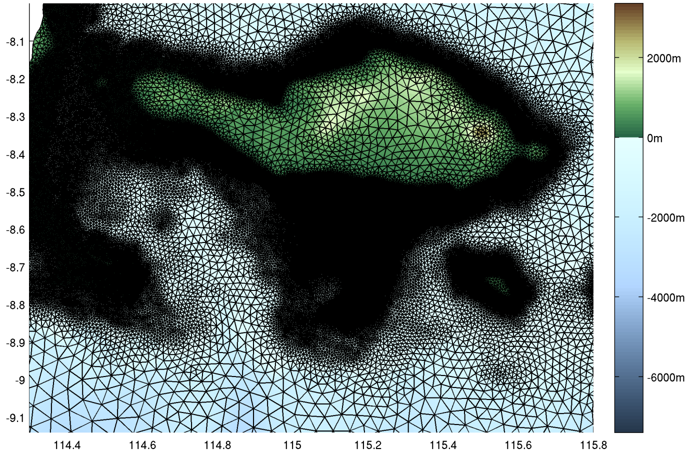


The computational grid discretizes the domain with

- varying resolution
  - 50m areas of interest
  - 500m all other coastal areas
  - 15km deep ocean
- 2.366.319 nodes
- 4.721.884 elements

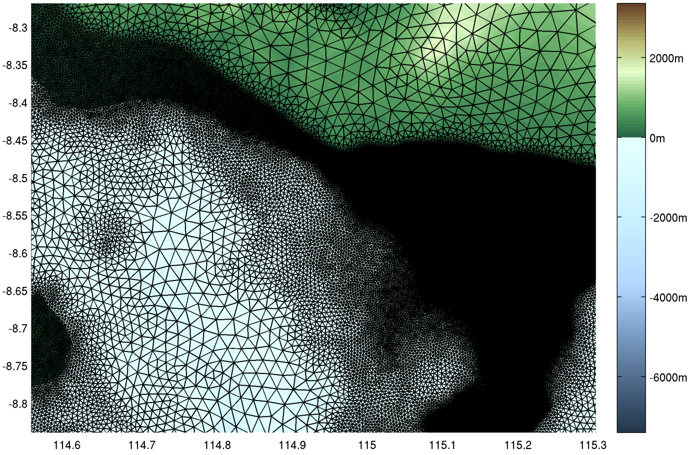
TsunAWI: example for a computational domain

regional grid for the Sunda Arc, focus on Bali



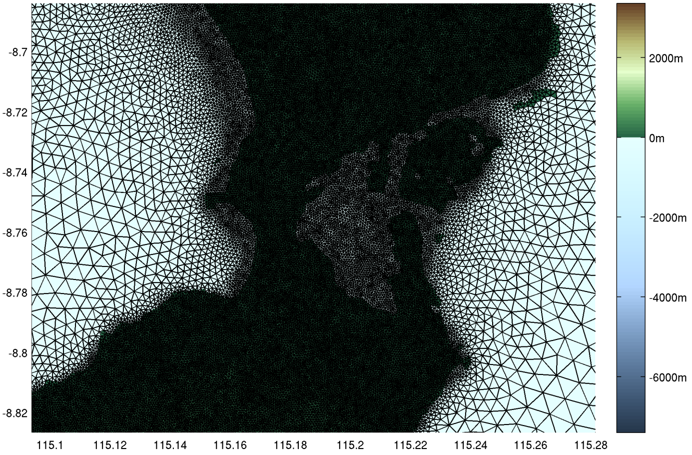
TsunAWI: example for a computational domain

regional grid for the Sunda Arc, focus on Bali



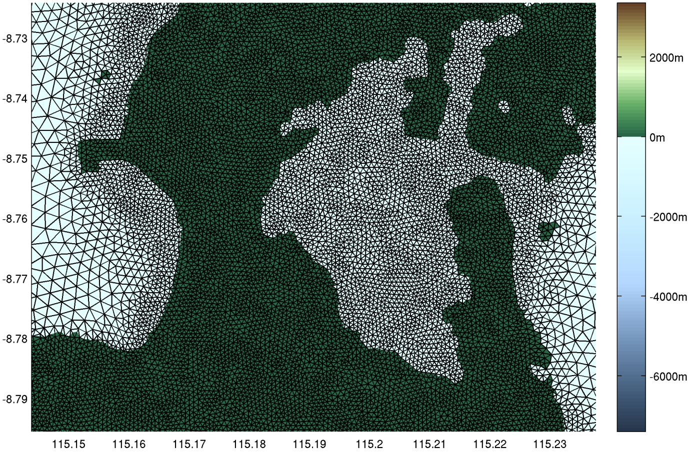
TsunAWI: example for a computational domain

regional grid for the Sunda Arc, focus on Bali



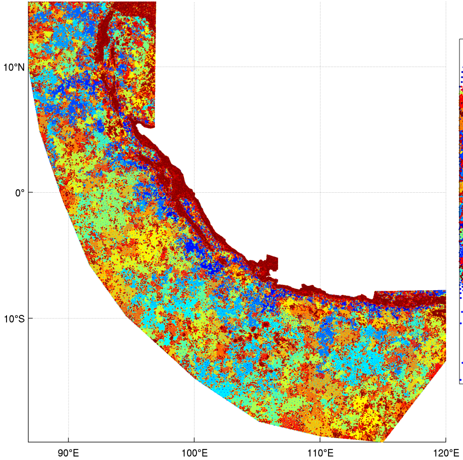
TsunAWI: example for a computational domain

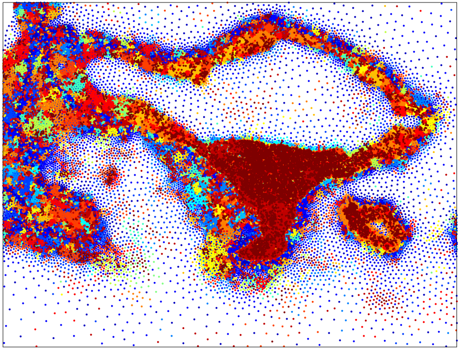
regional grid for the Sunda Arc, focus on Bali



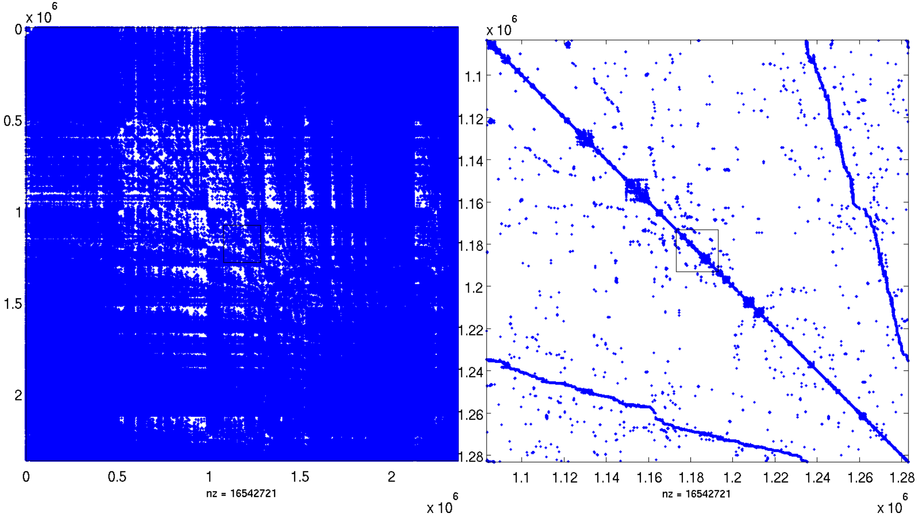
TsunAWI: example for a computational domain

Original numbering of nodes as provided by the grid generator

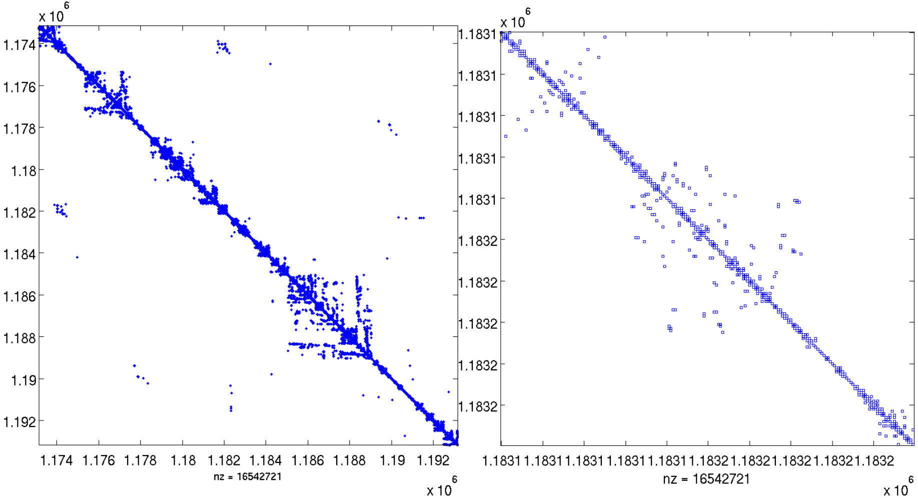




adjacency matrix, original grid



adjacency matrix, original grid



Motivation for resorting

Data locality on the original grid is very, very bad.

E.g., each computation on all nodes of one element results in at least one cache miss.

Motivation for resorting

Data locality on the original grid is very, very bad.

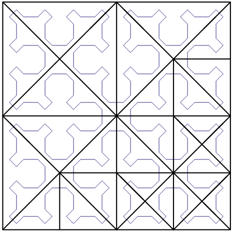
E.g., each computation on all nodes of one element results in at least one cache miss.

Most time consuming routines in every timestep:

`compute_velocity_at_nodes`	v(node) = F(adjacent edges, elems)
`compute_velocity`	v(edge) = F(adjacent elems, nodes)
`compute_ssh`	ssh(node) = F(adjacent elems, nodes)
`compute_gradient`	$\text{grad}_{x,y}$(elem) = F(adjacent nodes)

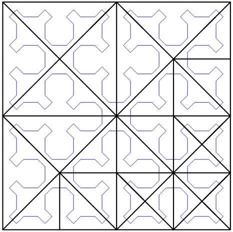
Ideas for resorting

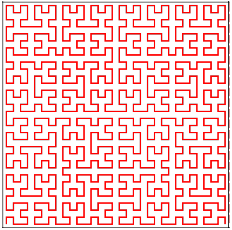
- SFC like Sierpinski curve in adaptive grid (J. Behrens et al., KlimaCampus Uni Hamburg) could help.
  But how to derive SFC for highly unstructured grid?



Ideas for resorting

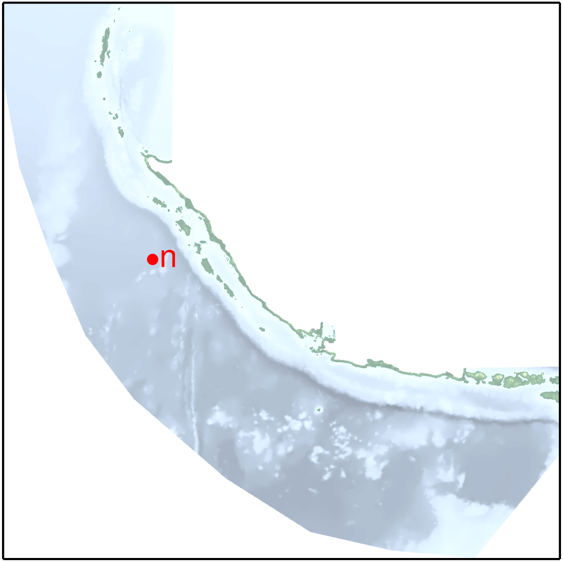
- SFC like Sierpinski curve in adaptive grid (J. Behrens et al., KlimaCampus Uni Hamburg) could help.
  But how to derive SFC for highly unstructured grid?





- Construct SFC like 3D Hilbert curve in particle code Gadget-2 (communication with T. Rung, TU Hamburg-Harburg)

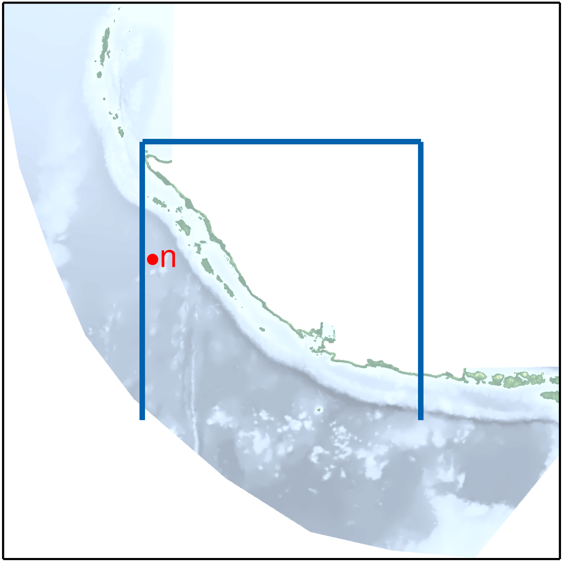
SFC construction



For all nodes n calculate the index in the Hilbert curve as a quad number:

SFC_index(n) =

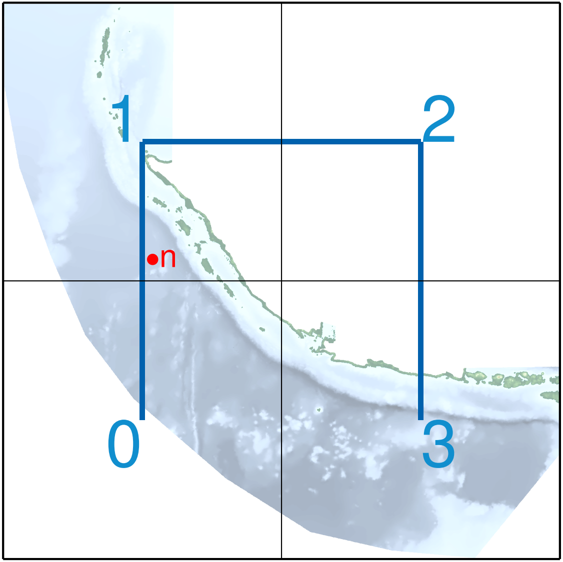
SFC construction



For all nodes *n* calculate the index in the Hilbert curve as a quad number:

SFC_index(n) =

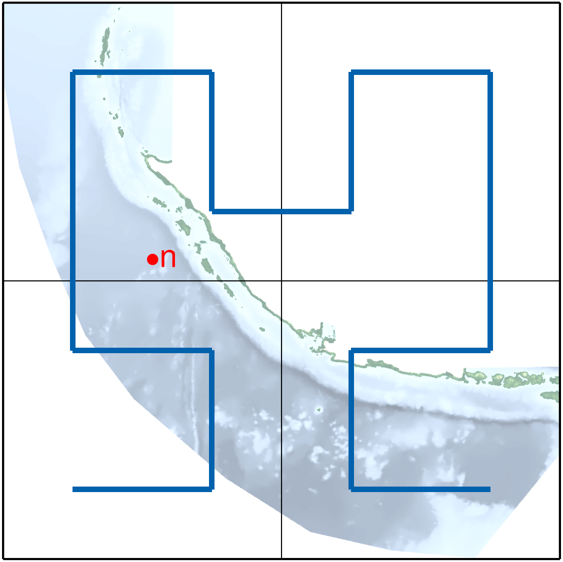
SFC construction



For all nodes *n* calculate the index in the Hilbert curve as a quad number:

SFC_index(*n*) =
1

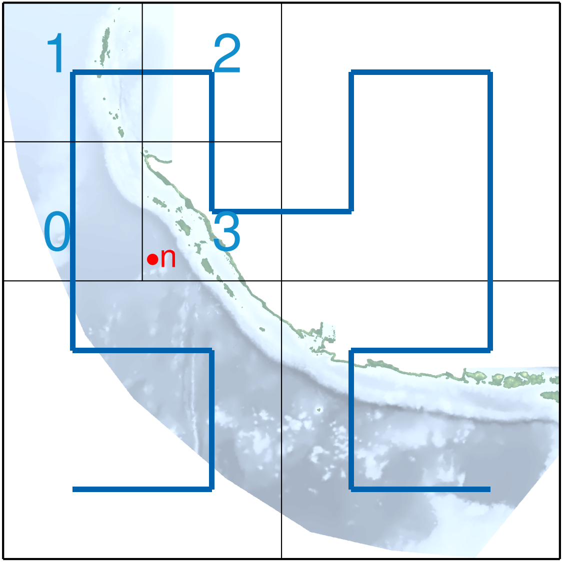
SFC construction



For all nodes *n* calculate the index in the Hilbert curve as a quad number:

SFC_index(n) =
1

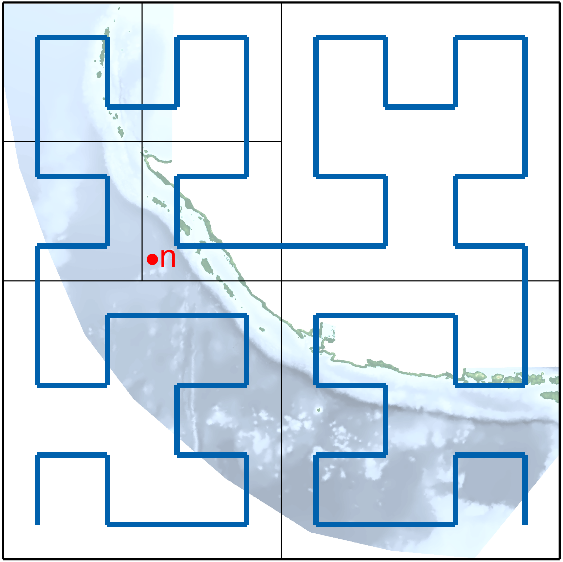
SFC construction



For all nodes *n* calculate the index in the Hilbert curve as a quad number:

SFC_index(n) = 13

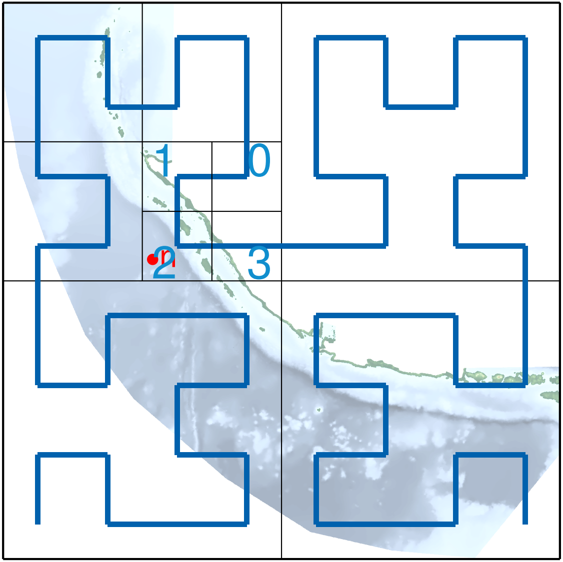
SFC construction



For all nodes *n* calculate the index in the Hilbert curve as a quad number:

SFC_index(*n*) = 13

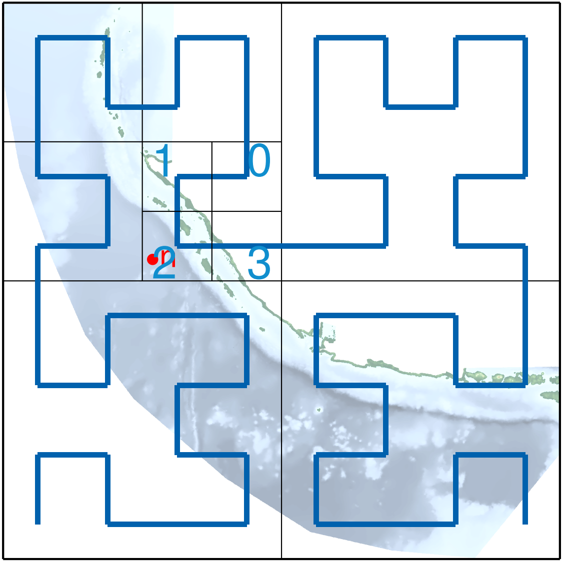
SFC construction



For all nodes *n* calculate the index in the Hilbert curve as a quad number:

SFC_index(*n*) = 132

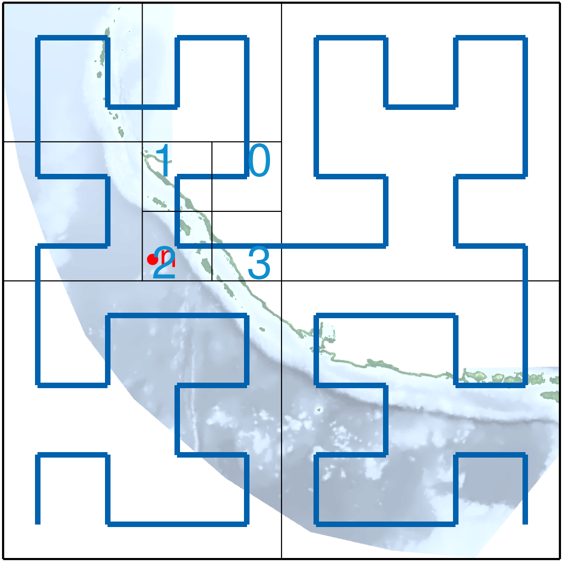
SFC construction



For all nodes n calculate the index in the Hilbert curve as a quad number:

SFC_index(n) =
132. . .

SFC construction



For all nodes *n* calculate the index in the Hilbert curve as a quad number:

SFC_index(n) = 132. . .

e.g. for 8 levels:

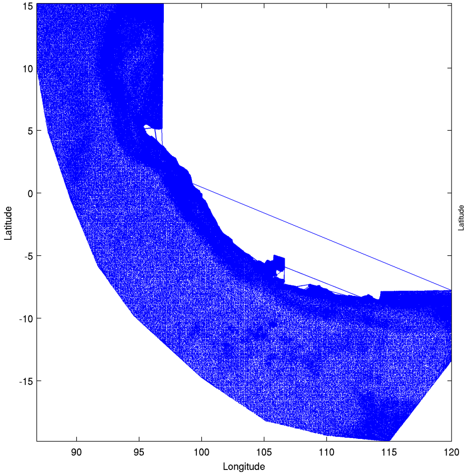
SFC_index(n) = $1 \cdot 4^8 + 3 \cdot 4^7 + 2 \cdot 4^6 + \ldots$

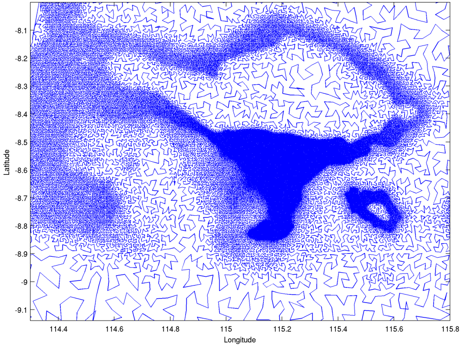
SFC reordering

- Reorder the nodes according to SFC_index.
- Reorder the elements
  - by an SFC separatly, or
  - numerically by node indicees
    (more efficient for TsunAWI)
- Edges are constructed in TsunAWI (sorted along the nodes)

SFC ordering of the nodes

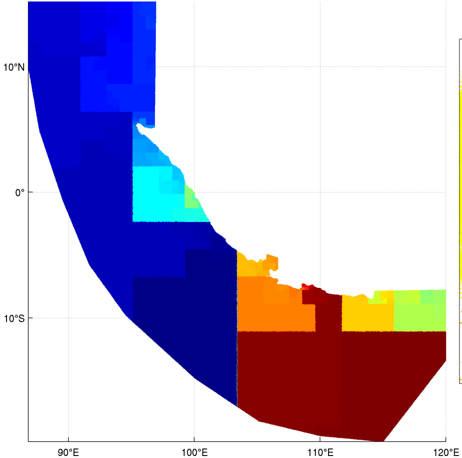
for TsunAWI regional indonesian grid



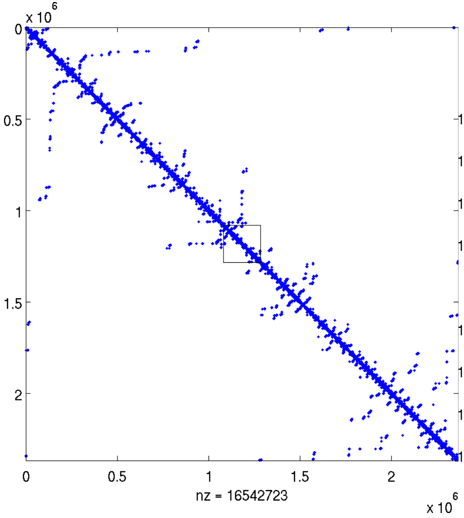


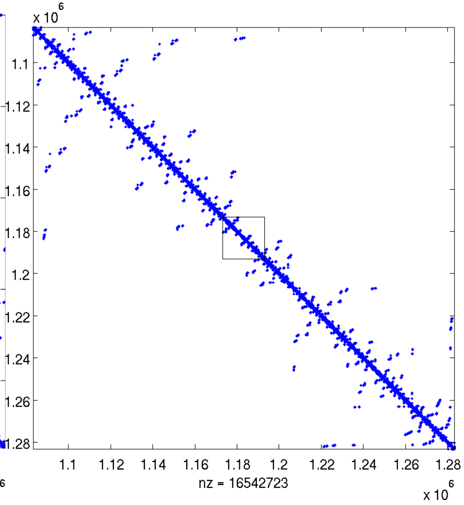
SFC ordering of the nodes

for TsunAWI regional indonesian grid

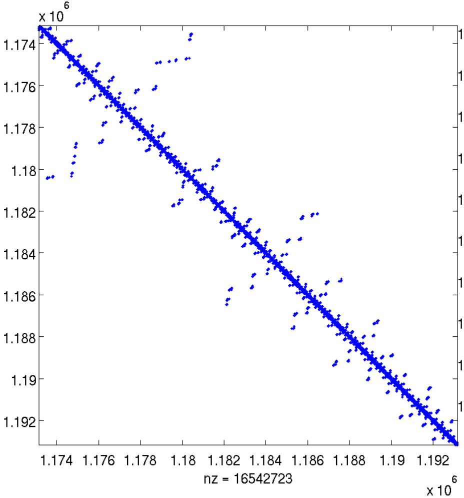


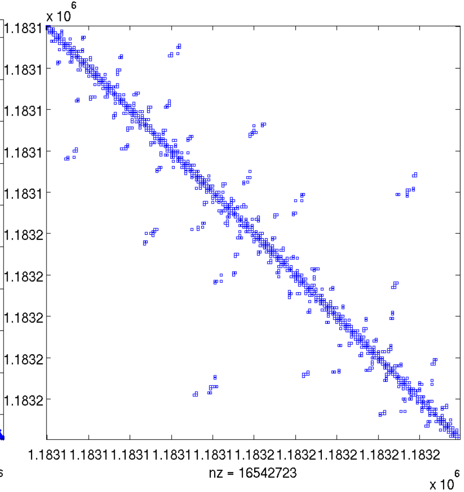
adjacency matrix for SFC sorted grid





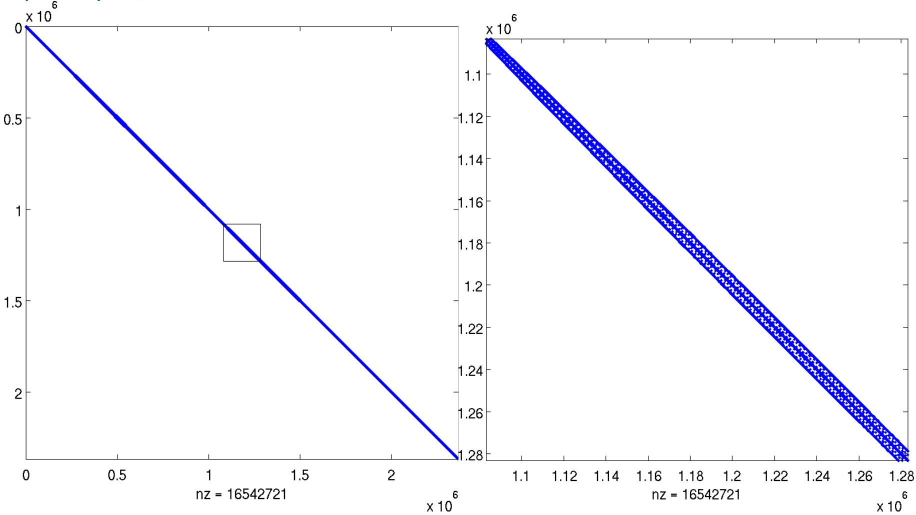
adjacency matrix for SFC sorted grid





Comparison: RCM ordering

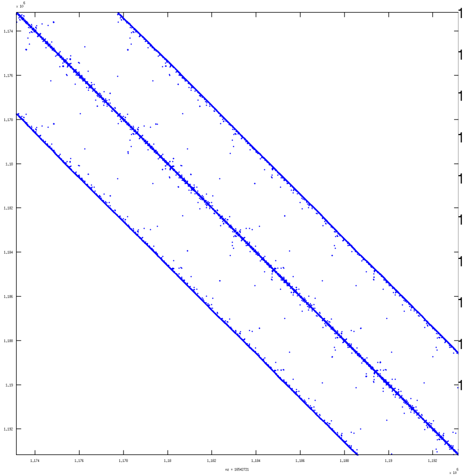
adjacency matrix

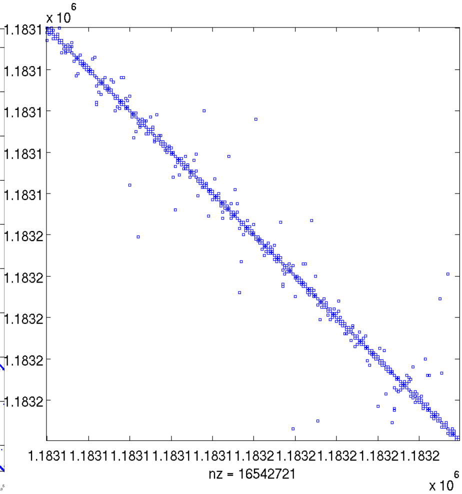


RCM (reverse Cuthill McKee) ordering obtained via adjacency matrix and Matlab symrcm for sparse matrices.

Comparison: RCM ordering

adjacency matrix

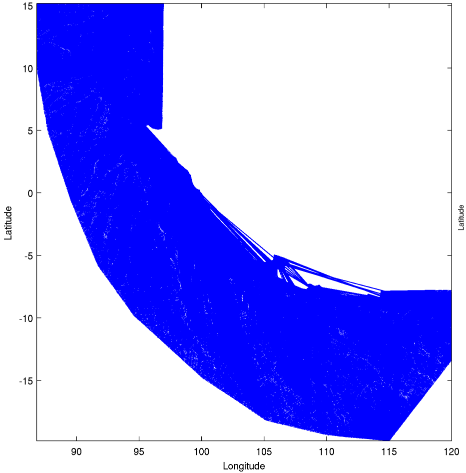


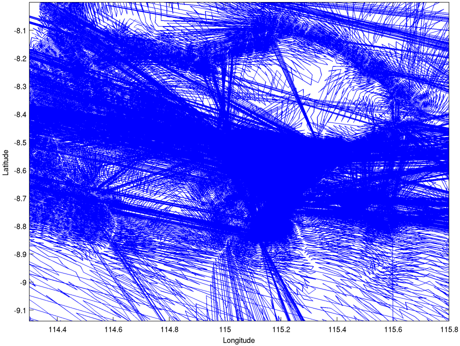


RCM (reverse Cuthill McKee) ordering obtained via adjacency matrix and Matlab symrcm for sparse matrices.

Comparison: RCM ordering

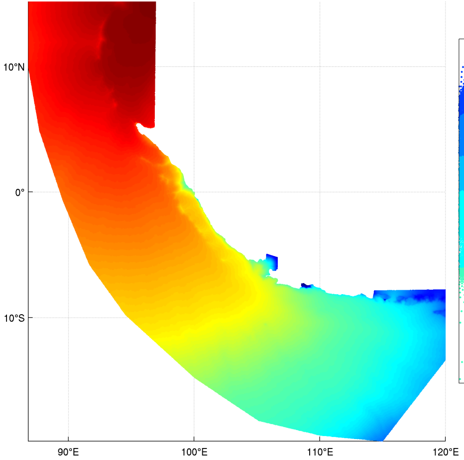
for TsunAWI regional indonesian grid

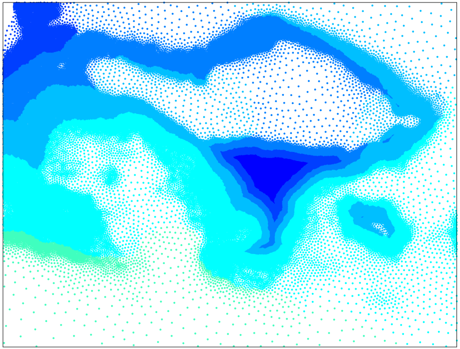




Comparison: RCM ordering

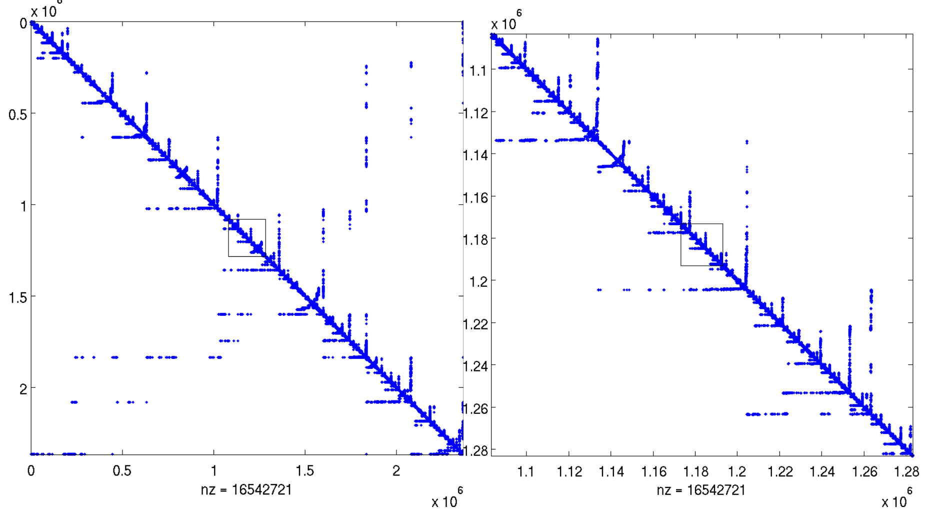
for TsunAWI regional indonesian grid





Comparison: AMD ordering

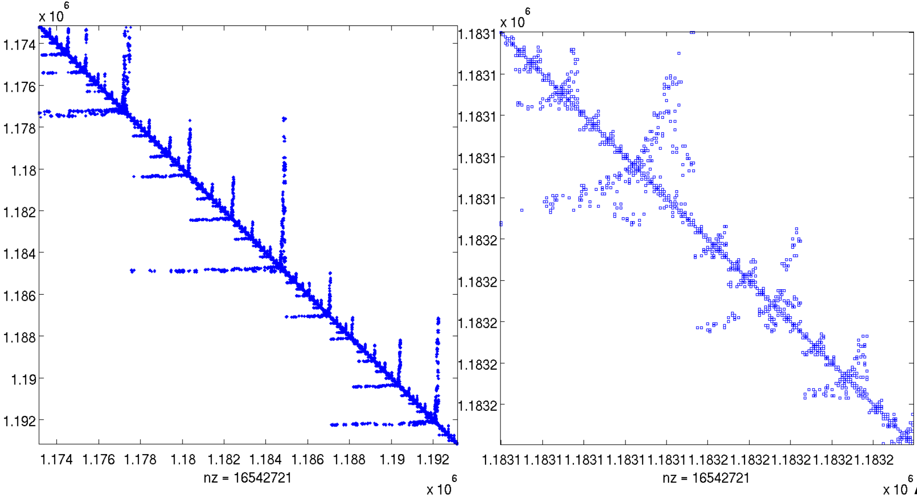
adjacency matrix



AMD (approximate minimum degree) ordering obtained via adjacency matrix and Matlab symamd for sparse matrices.

Comparison: AMD ordering

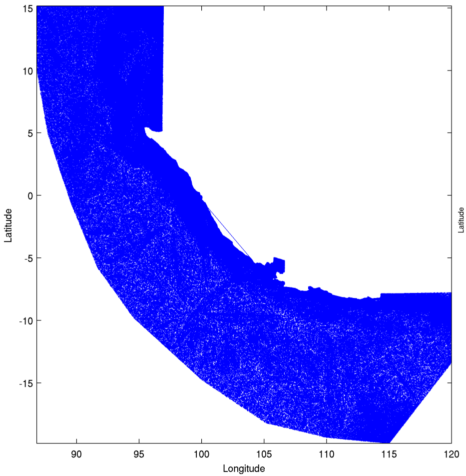
adjacency matrix

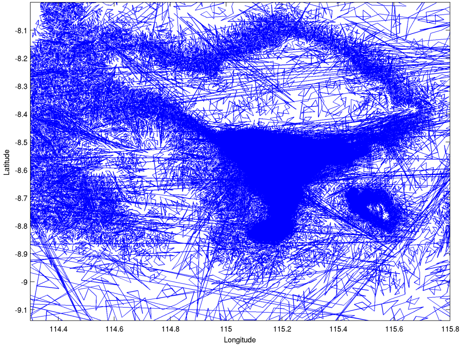


AMD (approximate minimum degree) ordering obtained via adjacency matrix and Matlab symamd for sparse matrices.

Comparison: AMD ordering

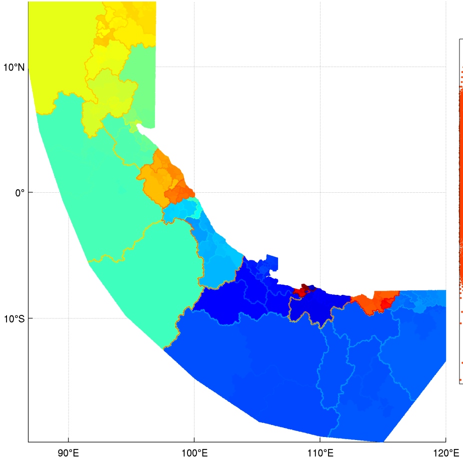
for TsunAWI regional indonesian grid

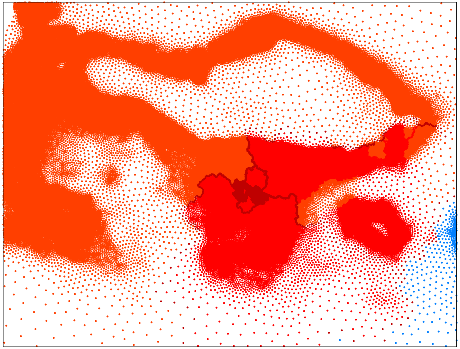




Comparison: AMD ordering

for TsunAWI regional indonesian grid





SFC compared to unsorted, RCM, SymAMD

computation time: IBM Power6

Computational time [seconds] for timestep on a cluster node

$1\times$ IBM Power6 (4 Cores, $2\times$ hyperthreading)

	OMP_NUM_THREADS			
	1	2	4	8
orig.	9.77	4.08	2.91	1.57
RCM	2.78	1.77	0.97	0.69
AMD	2.76	1.42	0.95	0.66
SFC	2.69	1.58	0.92	0.60

SFC compared to unsorted, RCM, SymAMD

Hardware counters: IBM Power6

IBM Hardware counter hpmcount for 1000 timesteps on $1\times$ IBM Power6 (4 Cores, $2\times$ hyperthreading, OMP_NUM_THREADS=8)

	hpmcount event	
	L2 cache misses	Number of loads per load miss
orig.	274,478,564,540	17.8
RCM	57,244,100,260	64.0
AMD	54,709,662,295	65.6
SFC	49,980,798,689	88.5

SFC compared to unsorted, RCM, SymAMD

computation time: Intel Xeon Nehalem-EX

Computational time [seconds] for one timestep on
one blade SGI Altix UV (HLRN, ZIB Berlin and RRZN Hannover)

2× Intel Xeon 5570 (8 Cores, 2× hyperthreading)

	OMP_NUM_THREADS					
	1	2	4	8	16	32
orig.	3.84	2.16	1.48	0.89	0.52	0.40
RCM	1.64	1.12	0.59	0.35	0.20	0.19
AMD	1.47	0.77	0.50	0.30	0.18	0.16
SFC	1.47	0.90	0.51	0.31	0.17	0.14

SFC compared to unsorted, RCM, SymAMD

computation time: Intel Xeon Nehalem-EX

Computational time [seconds] for one timestep on
one blade SGI Altix UV (HLRN, ZIB Berlin and RRZN Hannover)

2× Intel Xeon 5570 (8 Cores, 2× hyperthreading)

	OMP_NUM_THREADS						
	1	2	4	8	16	32	64
orig.	3.84	2.16	1.48	0.89	0.52	0.40	1.63
RCM	1.64	1.12	0.59	0.35	0.20	0.19	0.37
AMD	1.47	0.77	0.50	0.30	0.18	0.16	0.32
SFC	1.47	0.90	0.51	0.31	0.17	0.14	0.30

SFC compared to unsorted, RCM, SymAMD

computation time: Intel Xeon Nehalem-EX

Computational time [seconds] for one timestep on
one blade SGI Altix UV (HLRN, ZIB Berlin and RRZN Hannover)

2× Intel Xeon 5570 (8 Cores, 2× hyperthreading)

	OMP_NUM_THREADS							32, No
	1	2	4	8	16	32	64	First Touch
orig.	3.84	2.16	1.48	0.89	0.52	0.40	1.63	0.51
RCM	1.64	1.12	0.59	0.35	0.20	0.19	0.37	0.32
AMD	1.47	0.77	0.50	0.30	0.18	0.16	0.32	0.19
SFC	1.47	0.90	0.51	0.31	0.17	0.14	0.30	0.18

Remark on OpenMP

importance of first touch for data locality

```
allocate(array(dim))

array(:)  = 0.
```

```
!$OMP PARALLEL DO
do n=1,dim
array(n) = ...
end do
!$OMP END PARALLEL DO
```

Remark on OpenMP

importance of first touch for data locality

```
allocate(array(dim))



!$OMP PARALLEL DO
do n=1,dim
array(n) = 0.
end do
!$OMP END PARALLEL DO

!$OMP PARALLEL DO
do n=1,dim
array(n) = ...
end do
!$OMP END PARALLEL DO
```

properties of resorting by a SFC

SFC is a very valuable method, because

- it is cheap to compute
- provides good data locality
  on all levels of the memory hierarchy
- as domain decomposition, it keeps interfaces small
  (though not optimal)

work to do

- Influence of SFC ordering on
  - ILU based preconditioners
    - fill-in
    - computational load
    - convergence rate
  - sparse matrix computations in general
- SFC compared to generic partitioning algorithms (MeTiS, scotch,. . .)
- TsunAWI
  - further optimize OpenMP parallelization
  - MPI parallelization