HBM code modernization

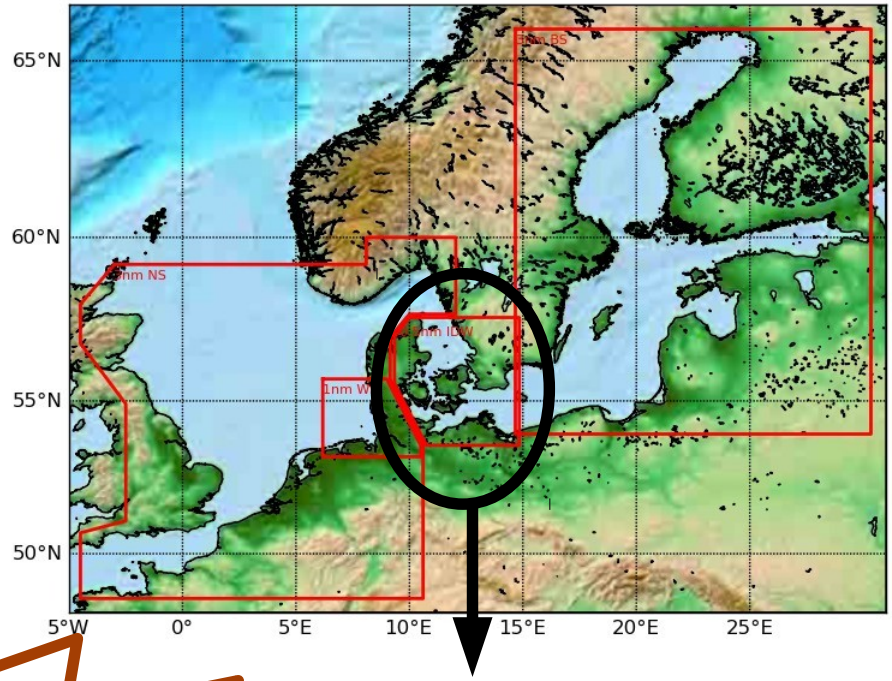
Jacob Weismann Poulsen, DMI, Denmark
Per Berg, DMI, Denmark
Karthik Raman, Intel, USA

Outline

- HBM
  - Introduction
  - Data structures and parallelization
  - Performance
- ESCAPE
  - Initial study of a radiation kernel (if time permits)

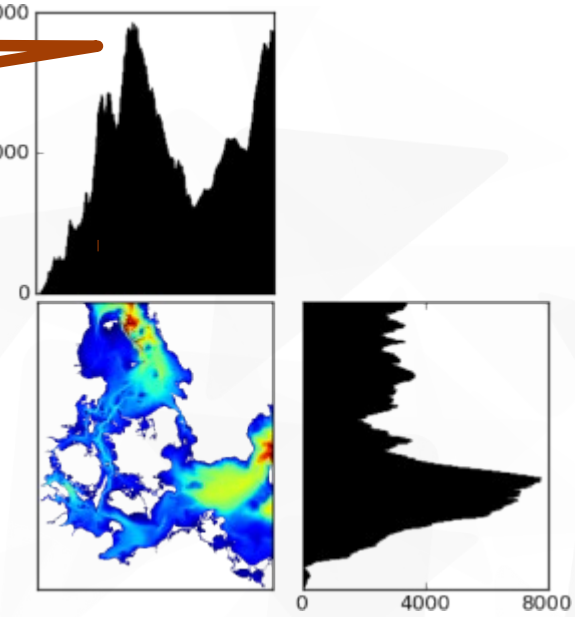
Introduction

- A 3D ocean circulation model (solves IBVP)
- Basis of DMI's regional production forecasts:
  - storm surge warning in DK
  - MyOcean & Copernicus Baltic MFC
- 2-way dynamical nesting
  - horizontal, vertical, time
  - any number of nesting levels
  - high resolution in regional seas
  - very high resolution in straits
- HBM community: DMI, BSH, MSI, FMI + third parties (MOU)



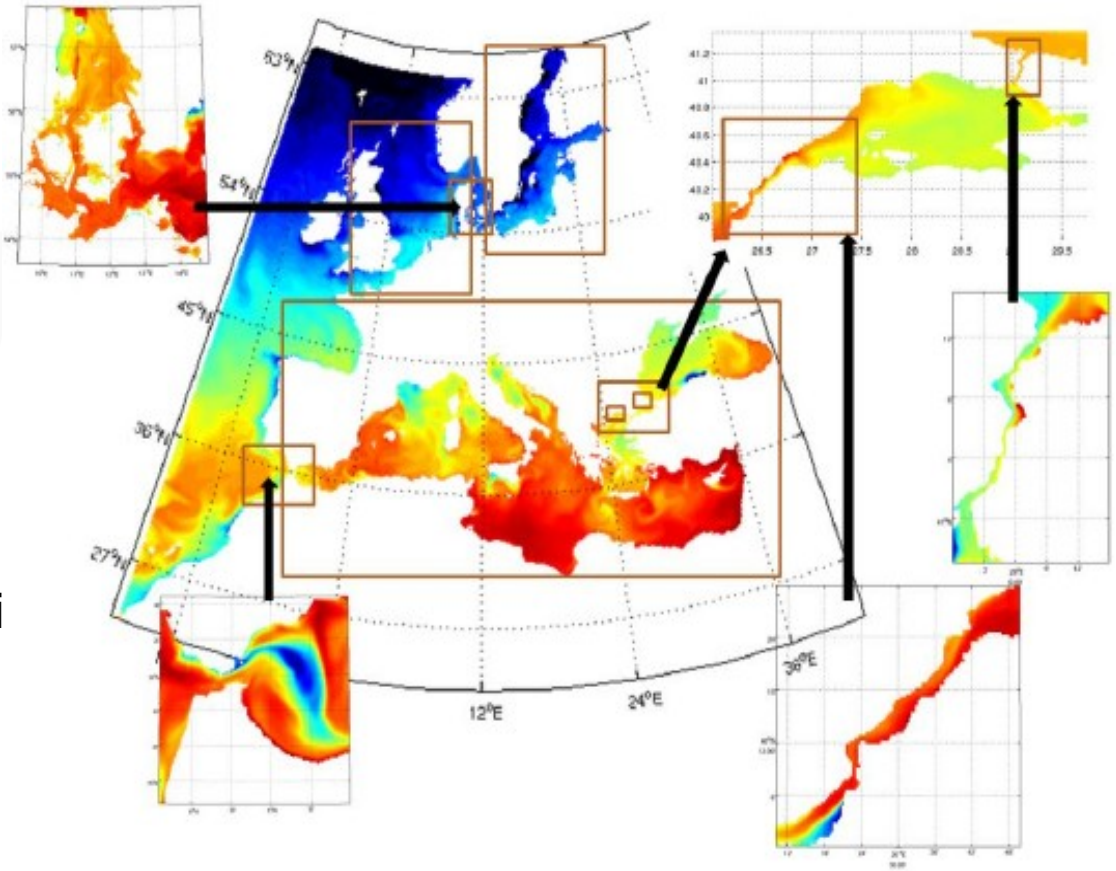
Points [mill]: #3d:68, iw3:8
Memory [Gb]: 3.2

	mmx [N/S]	nmx [E/W]	kmx [layers]	#3d	dt	iw3/#3d [%]	iw2/#2d [%]
baltic	720	567	122	49805280	12.5	12.3	29.2
idw	482	396	77	14697144	12.5	10.8	42.4
ns	348	194	50	3375600	25	14.2	28.0
ws	149	154	24	550704	25	18.8	50.5



Introduction

- Homogenised production *forecast* on pan-European scale
- Regional climate setups covering a small domain is less interesting but with this they can do pan-European climate runs at rather high resolution
- Reference: 1nm~1.85 km~1.15 mi



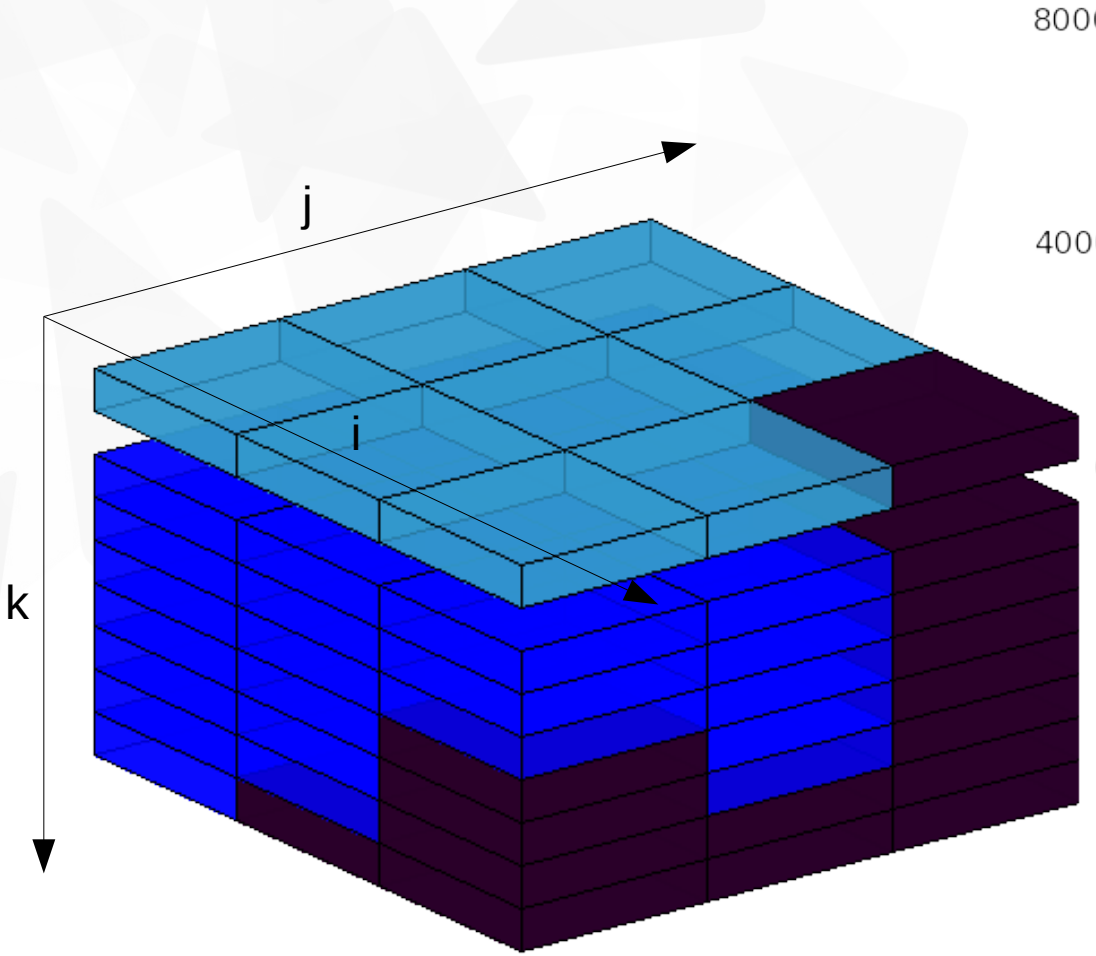
- Resolution in this setup:
  - Vertical resolution from 1 meter
  - Horizontal resolution from ~200 meter to 5.5km

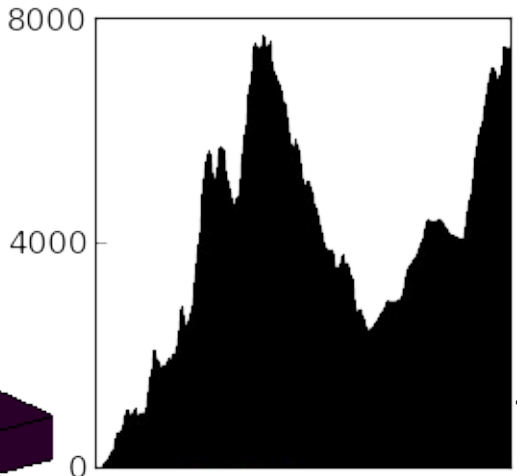
Points [mill]: #3d:110, iw3:19
Memory [Gb]: 7.2

10 two-way nested domains:

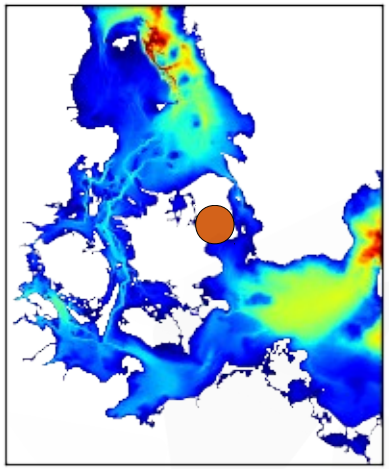
Bosphorus / Dardanelles Straits:	~0.1 n.m.
Inner Danish waters:	~0.5 n.m.
Marmara Sea / Gibraltar / Baltic Sea:	~1 n.m.
North Sea / Shelf / Med.Sea / Black Sea:	~3 n.m.

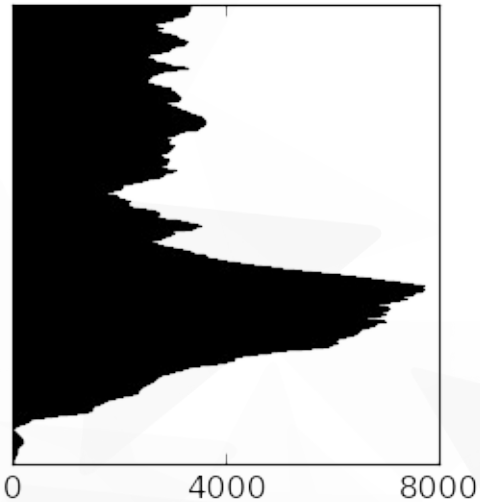
The data is sparse and highly irregular





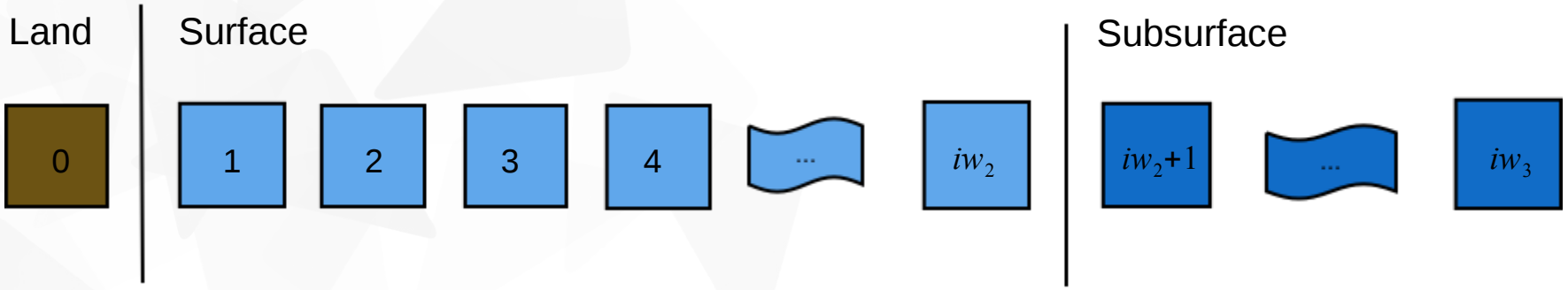
10.8%





Data structures and parallelization

SIMD (no interactions between columns)




```
do iw = 1,iw2
  i = ind(1,iw)
  j = ind(2,iw)
  ! all surface wet-points (i,j) reached with stride-1
      ... u(iw) …
enddo
do iw = 1,iw2
  kb = kh(iw)
  if (kb < 2) cycle
  i = ind(1,iw)
  j = ind(2,iw)
  mi0 = mcol(iw)-2
  do k = 2, kb
    ! all subsurface wet-points (k,i,j) are reached with stride-1
    mi = mi0 + k
    ... u(mi) ...
  enddo
enddo
```

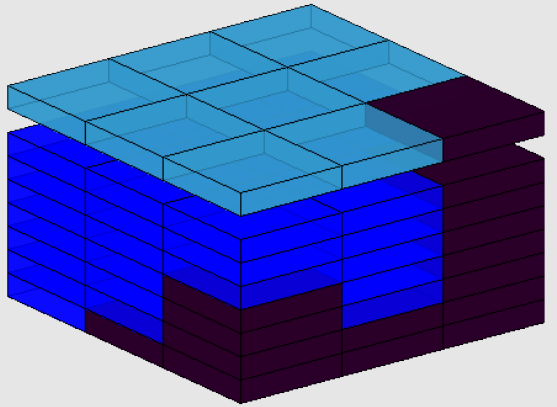
This shows only, the most simple case where there is no interactions between columns.

SIMD (interactions between columns)

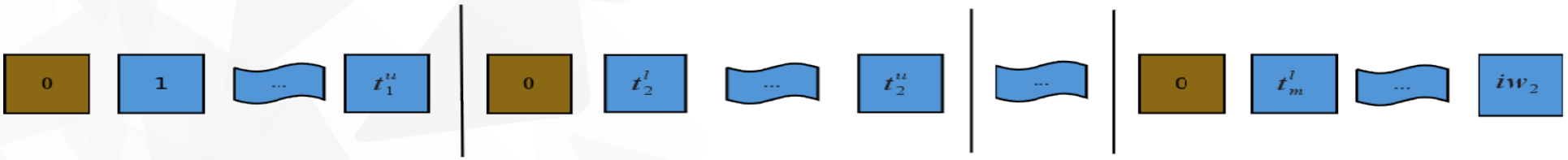
```
do iw = 1,iw2
  kb = kh(iw)
  if (kb < 2) cycle
  i = ind(1,iw)
  j = ind(2,iw)
  mi0 = mcol(iw) - 2
  me0 = mcol(msrf(i,  j+1)) - 2
  mw0 = mcol(msrf(i,  j-1)) - 2
  mn0 = mcol(msrf(i-1,j  )) - 2
  ms0 = mcol(msrf(i+1,j  )) - 2

  kmx = min(kb,kh(msrf(i,j+1)),kh(msrf(i,j-1)),kh(msrf(i-1,j)),kh(msrf(i+1,j)))
  ! The FAT loop
  do k = 2, kmx
    mi = mi0 + k
    me = me0 + k
    mw = mw0 + k
    mn = mn0 + k
    ms = ms0 + k
       ... t(mi) ... t(me) ...
  enddo

  ! and a bunch of SKINNY remainder loops
  do k=max(2,kmx+1),min(kb,kh(msrf(i,j+1))) ! only mi, me
    mi = mi0 + k
    me = me0 + k
    ...
  enddo
  ...
enddo
```



Outer parallelization (birds eye)




```
! initialization
...
call hbm_mpi_init() ! decompose the problem (1,...,iw2 Cs) such that each MPI task
                    ! deals with a subset of C locally enumerated by index: 1...iw2_l
call hbm_omp_init() ! decompose the task-local problem (1,...,iw2_l Cs) into threadlocal
                    ! sub-chunks and NUMA first-touch according to this decomposition.
...
! Timeloop (set of calls followed by halo-swaps, followed by set of calls, followed...)
!$OMP PARALLEL DEFAULT(SHARED)
call foo( ... );call bar(...); ...
call halo_update(...) ! deals with MPI and openMP
call baz( ... );call quux(...); ...
...
!$OMP END PARALLEL

subroutine foo(...)
  ...
  call domp_get_domain(kh, 1, iw2_l, nl, nu, idx)
  do iw=nl,nu
    i = ind(1,iw)
    j = ind(2,iw)
    ! all task and threadlocal wet-points (:,:,:) are reached here
    ...
  enddo
end subroutine foo
```

No more synchronization for threads than for MPI tasks

Thread load balancing

- Each thread will handle a sub***interval*** of columns! Another set of sub***sets*** will impose another split of the threads.
- Formal definition:

> Let $I = \{1, \ldots, m\}$ be the column index set and let $\{w_1, \ldots, w_m\}$ be the weights associated with the individual columns. Let n denote the number of threads/tasks. A disjoint subinterval $I_i = \{[l_i; u_i]\}_{i=1,\ldots,n}$ covering of I induces a cost vector $(c_1, \ldots, c_n)$ with $c_i = \sum_{j=l_i}^{u_i} w_j$. The cost c of the covering is defined as $\max_i c_i$. The balance problem is to find a covering that minimizes c.

- The NP-hard problem is reduced to the integer partition problem which provides an exact solution within time complexity: $O(m^2n)$. The weights can be a sum of sub weights while retaining problem complexity!
- Heuristics: Greedy approach with runtime complexity: O(n).

Thread parallelism - summary

- SPMD based (like MPI) and *not* loop based in order to minimize synchronization. A single OpenMP block with orphaned barriers surrounding synchronization points such as MPI haloswaps will do

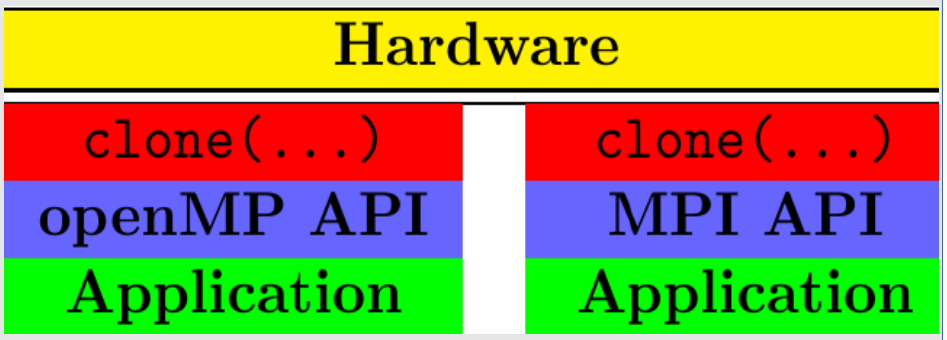
  (nice side-effect: No explicit scoping).
- Consistent loop structures and consistent data layout and usage throughout the whole code implying that it is easy to ensure a proper NUMA first-touch.
- The OpenMP standard does not provide us with a clause that allows for advanced balance control so we have to wrap our own. It can be done either offline (exact) or online (heuristic).

Halo swaps (birds eye)

```
!Timeloop (set of calls followed by halo-swaps, followed by set of calls, followed...)
!$OMP PARALLEL DEFAULT(SHARED)
call foo( ... );call bar(...); ...
call halo_update(...) ! deals with MPI and openMP
call baz( ... );call quux(...); ...
!$OMP END PARALLEL

subroutine halo_update(ia,a,...)
  ...
  if (sloppy_halo_omp) then
    !$OMP BARRIER
  else
    ! more involved tracking and swap via !$OMP FLUSH
  endif
  if (mpi_swap2d) then
    call hbm_mpi_halo_update_2d(ia,a)
  else
    call hbm_mpi_halo_update_3d(ia,a)
  endif
end subroutine halo_update

subroutine hbm_mpi_halo_update_2d(ia,a)
  integer(4), intent(in) :: ia
  integer(4), intent(in) :: a(:)
  call MPI_neighbor_alltoallv(a,sendc2d(ia)%p,sdispls2d(ia)%p,desttype2d(ia)%p,&
                              a,recvc2d(ia)%p,rdispls2d(ia)%p,srctype2d(ia)%p, &
                              halo_comm(ia),ierr)
end hbm_mpi_halo_update_2d
```



Numa initialization

```
...
!$OMP PARALLEL DEFAULT(SHARED)
...
call numa_init( ... )
...
!$OMP END PARALLEL

subroutine numa_init(...)
  ...
  call domp_get_domain(kh, 1, iw2, nl, nu, idx)
  ! surface init
  a(nl:nu)  = 0.0_8
  ...
  ! depth init
  do iw=nl,nu
    kb = kh(iw)
    if (kb < 2) cycle
    ml = mcol(iw)
    mu = ml + kb – 2
    a(ml:mu)   = 0.0_8
      ...
  enddo
end subroutine numa_init
```

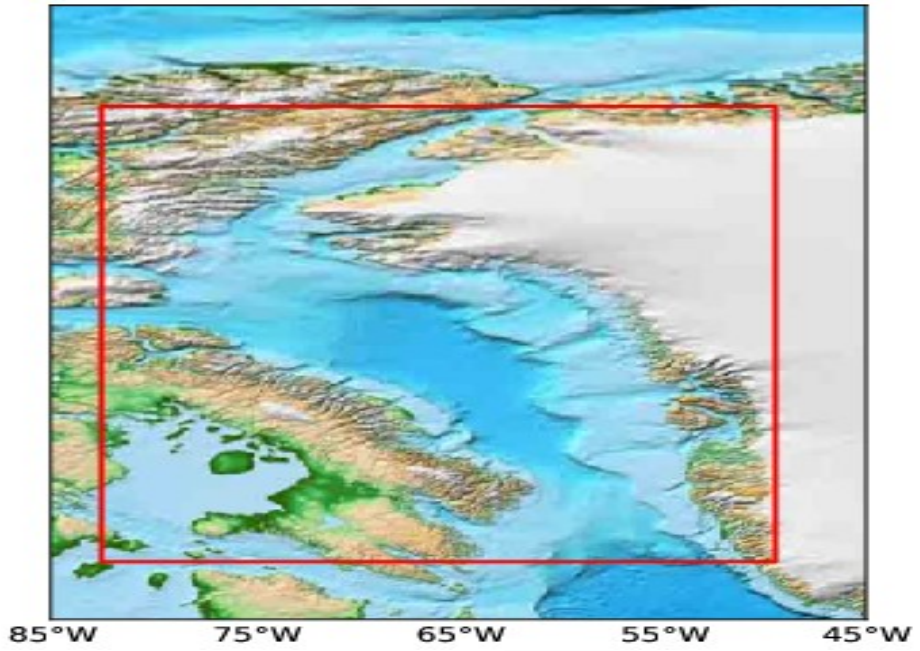
Thread parallelism – premature optimization

- Could improve the BW usage by splitting the surface and depth handling. This will improve the temporal locality but may require additional barriers.
- OMP BARRIERS may be done with p2p via OMP FLUSH. Why don't openMP have an OMP UPDATE similar to openACC to reduce cache coherency overhead ?
- Manual padding to deal with missing system support at the cacheline level (false sharing) and at the page level (perfect NUMA locality).
- Several heuristics to improve the balancing as mentioned earlier. HOWEVER, we need more performance insights before considering pursuing these ideas further.

Performance

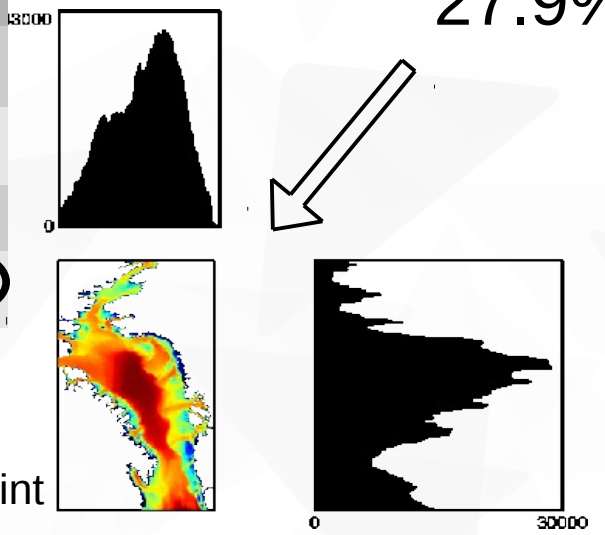
Testcase (timeloop focus)

- Input files provided by:
  Lars Johanson & Jens Murawsky
- No nesting, no IO, no met forcing
- No restrictions on data, cf.
  http://lotsofcores.com
- Summary:



res [m]	[N/S]	[E/W]	[layers]	*3d [mill]*	*iw3 [mill]*	dt [sec]	*Mem [mb]*
3600	565	331	137	25.6	7.1	10	3355
1800	1130	661	137	102.3	27.8	5	13068
900	2260	1322	137	409.3	111.4	2.5	52225
450	4520	2644	137	1637.3	445.4	1.25	206030

27.9%



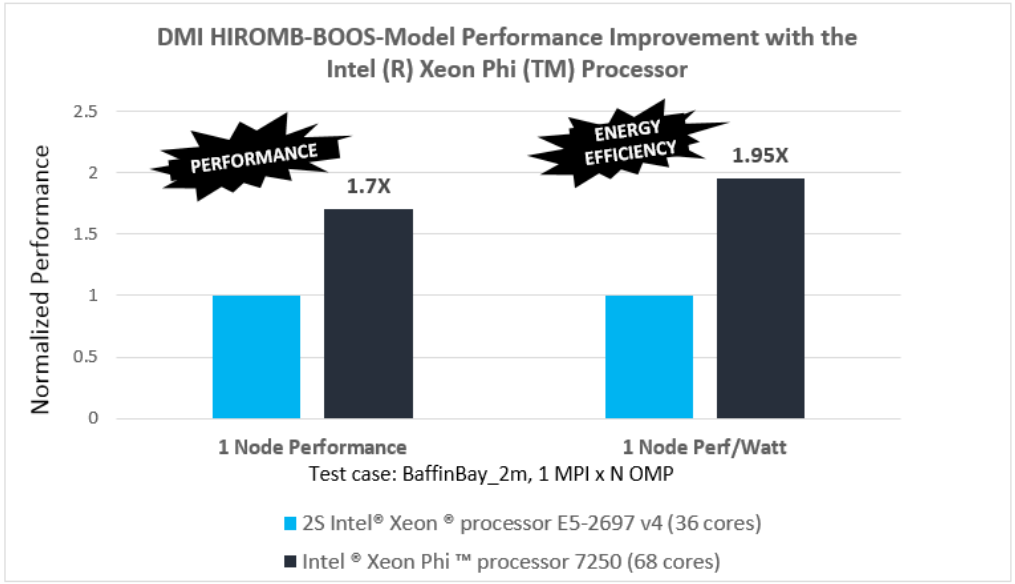
#gridpoints is 1.5xN640, but the model still have a “small” memory footprint
https://software.ecmwf.int/wiki/display/FCST/Increase+in+data+volume

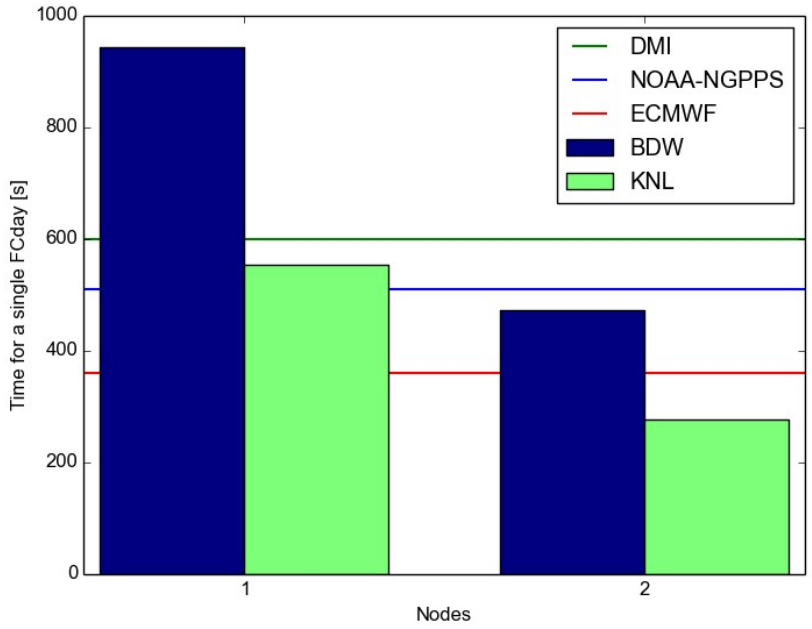
Intermezzo: Musings on performance

- Performance = FCT(method , implementation)
- Evaluation of performance: Use generic measures like FCdays/hour and Energy2solution at different node counts and under the use of different nodes.
- Given good scientific performance - all “we” really care about is that time2solution (T2S) and energy2solution (E2S) comply with our requirements

  E2S ~ (Power Draw/Node) x (Node Hours) x PUE

Performance results (1S KNL vs 2S BDW)





- Are these universal results ?
  - 1nm (1800m) setup: T2S is not 1.7x but 1.9x faster
  - Nested production setup (p. 3): T2S is not 1.7x but 1.4x
- A single KNL node is sufficient to complete 1 forecast day within 10 minutes for both the 2nm BaffinBay setup (p. 16) and our current nested production setup (p. 3).

Performance summary (1nm setup, single node)

	Fraction of time		Normalized time	
	E5-2697v4@2.3	**Xeon-Phi 7250@1.40**	**E5-2697v4@2.3**	**Xeon-Phi 7250@1.40**
advection	43%	41%	100%	50%
deform	3%	2%	100%	34%
uvterm	3%	2%	100%	41%
smag	3%	3%	100%	48%
momeqs	7%	11%	100%	82%
turbulence	6%	9%	100%	76%
vdiff	1%	3%	100%	109%
diffusion	4%	3%	100%	43%
density	2%	3%	100%	64%
sumuvwi	6%	4%	100%	32%
bcldens	2%	3%	100%	65%
masseqs	2%	2%	100%	47%
tflow_up	8%	5%	100%	37%
timeloop	**100%**	**100%**	**100%**	**53%**

Hardware

Micro-architecture	IvyBridge	Haswell	Broadwell
Model	E5-2697v2	E5-2697v3	E5-2697v4
Released	Q3, 2013	Q3, 2014	Q1, 2016
Cores/node	24	28	36
Frequency [GHz]	2.7	2.6	2.3
#cores [%]	100	117	150
#cores time [%]	100	85.7	66.7
HPL [GF/s]	492	949	1236
HPL efficiency [%]	95	81	93
HPL time [%]	100	52	40
HPL power [W]	700	750	545
HPL [GF/s/W]	0.7	1.26	2.26
Triad [GB/s]	86	107	129
Triad efficiency [%]	84	78	84
Triad time [%]	100	80	67
Triad power [W]	620	380	425
Triad [GB/s/W]	0.135	0.287	0.303

Acknowledgement

- Michael Greenfield, Intel
- Larry Meadows, Intel
- John Levesque, Cray
- Bill Long, Cray
- Kevin Thomas, Cray
- Brent Leback, Nvidia

ESCAPE – The acraneb* dwarf

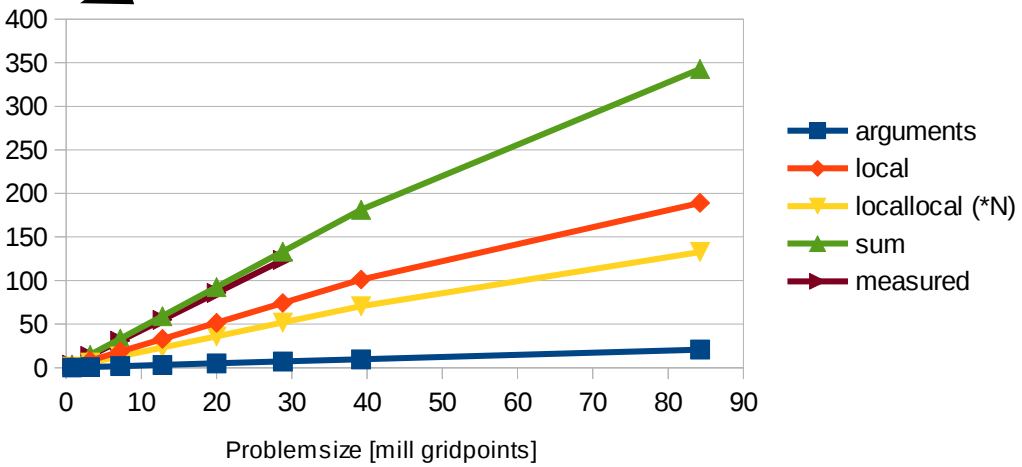
*1) Single interval shortwave radiation scheme with parameterized optical saturation and spectral overlaps by J. Masek et al, Q. J. R. Meteorol. Soc. (2015) DOI:10.1002/qj.2653
*2) Single interval longwave radiation scheme based on the net exchanged rate decomposition with bracketing by J.F. Geleyn et al, preprint (2016)

Initial study

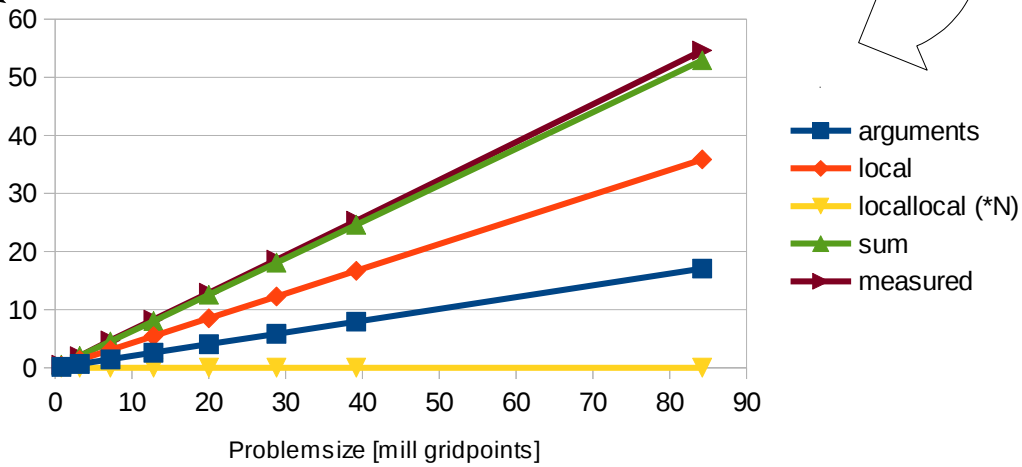
Baseline	case [s]
SLOC [lines]	5687
Language state	F77 fixed-size, cast
Technical state	OK
Numerical state	4 digits
Largest psize on 16Gb	200x200x80
Largest psize on 64Gb	400x400x80 1653
Largest psize on 128Gb	600x600x80
Target psize	1200x1080x65

Now, < 64Gb

Memory footprint [Gb] in baseline



Memory footprint [Gb] after refactoring

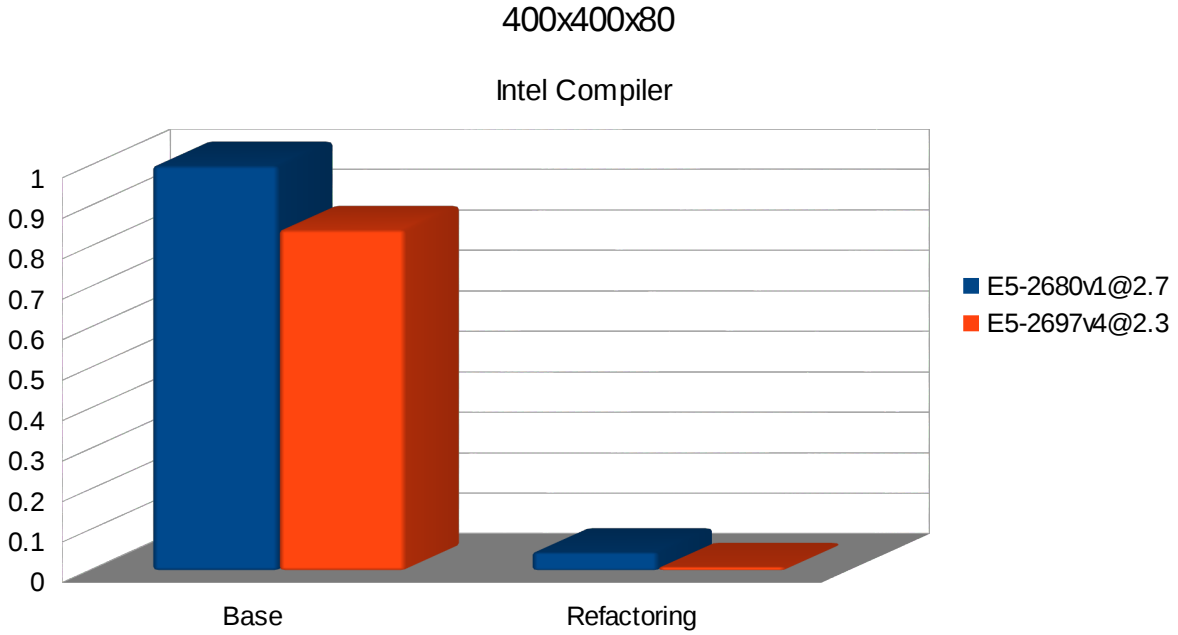


Performance improvement

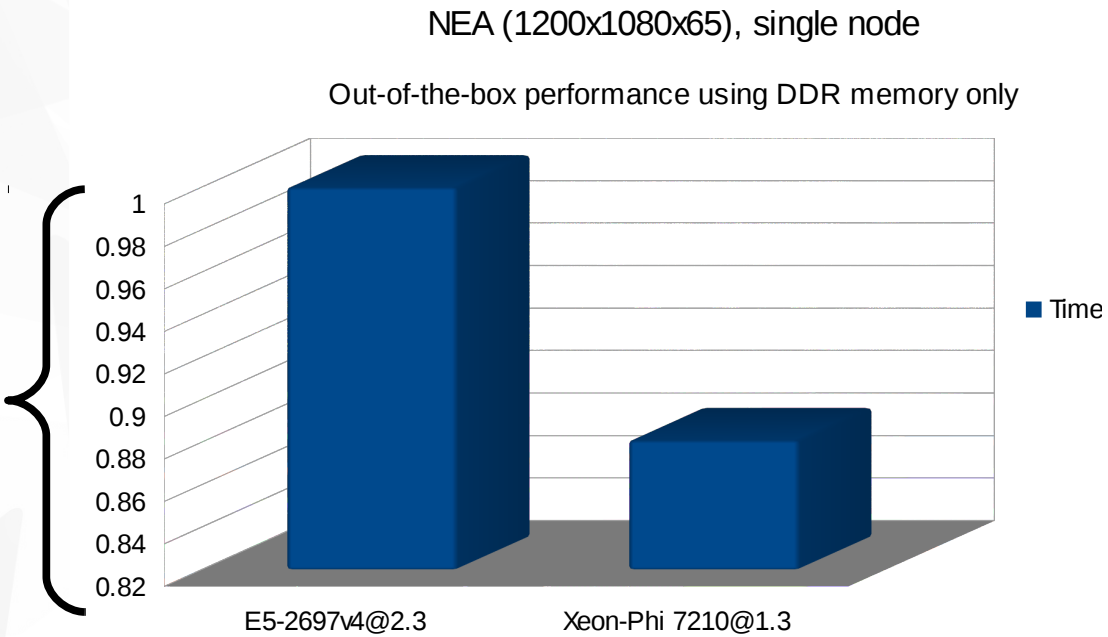
- Current refactorization improvement:
  - Memory footprint reduction: 6.6x
  - Time (E5-2680v1@2.7 reference timing provided: 1.600)

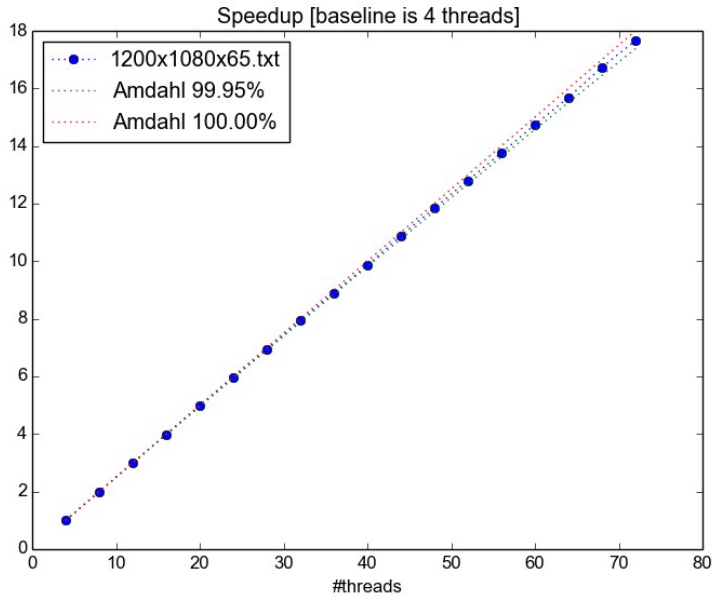
	E5-2680v1@2.7	E5-2697v4@2.3	E5-2680v1@2.7	E5-2697v4@2.3
Base	1.000	1.000	1.000	0.841
Refactoring	0.045	0.010	0.045	0.008

  - Investment in hardware vs software (matters more today!)



Target problemsize (NEA: 1200x1080x65)





- The sustained performance is **not** impressive neither on BDW nor on KNL but promising that KNL already outperforms BDW after our initial refactorization steps (memory trimming, SPMD threading and SIMD vectorization of the most expensive loop)