4D-VAR Optimization Efficiency Tuning

Tomas Wilhelmsson

Swedish Meteorological and Hydrological Institute

(now ECMWF)

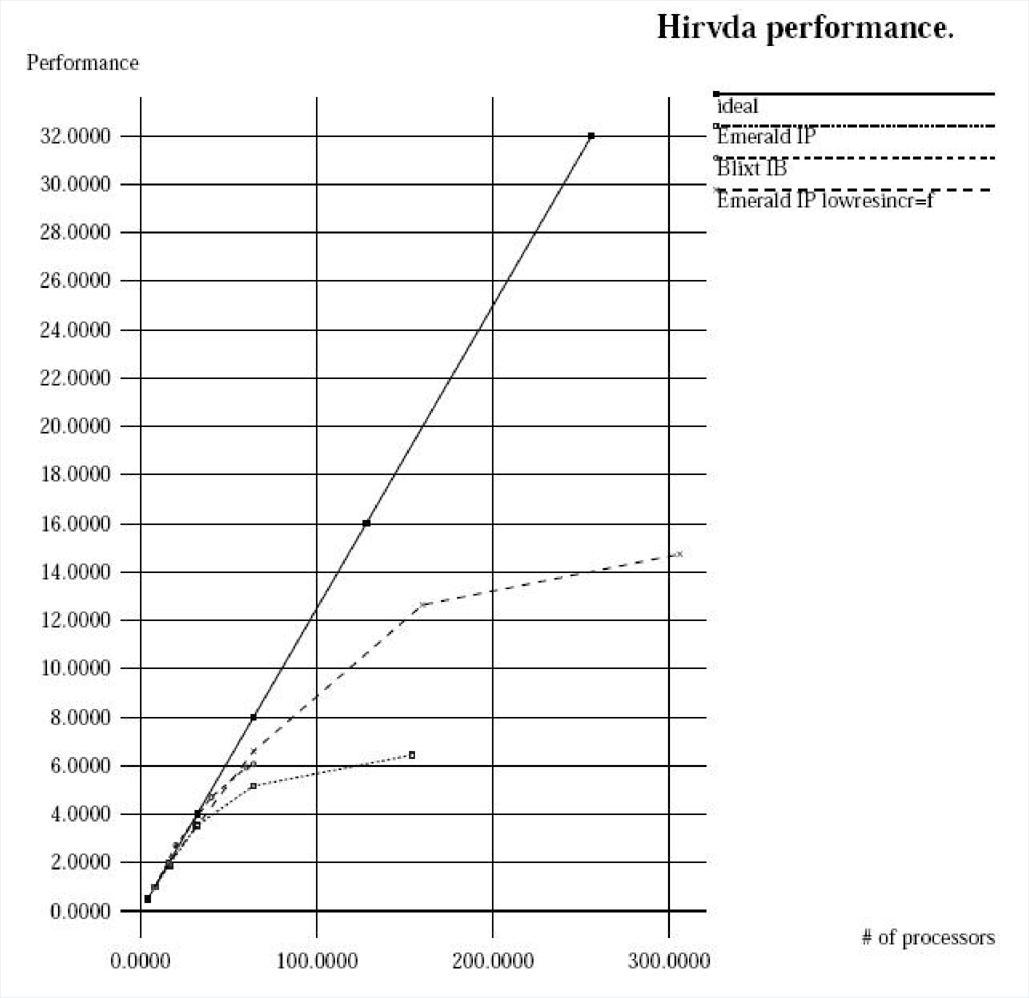
2009-05-14

Previous HIRLAM 4D-VAR performance scaling

Niko Sokka (FMI):

"In the end of day, 4DVAR scales up to 84 processors in our system and then stalls."

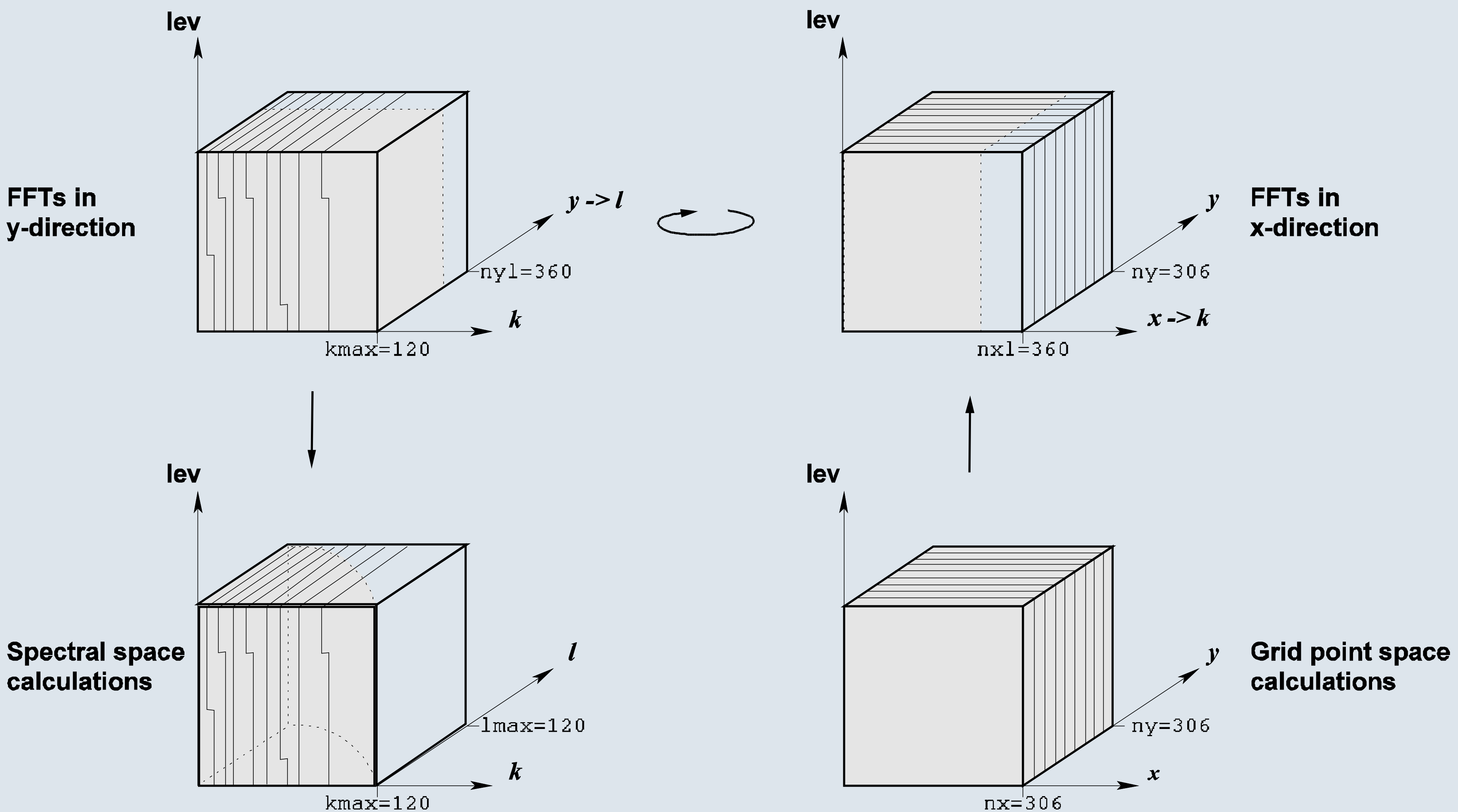
Torgny Faxén (NSC):



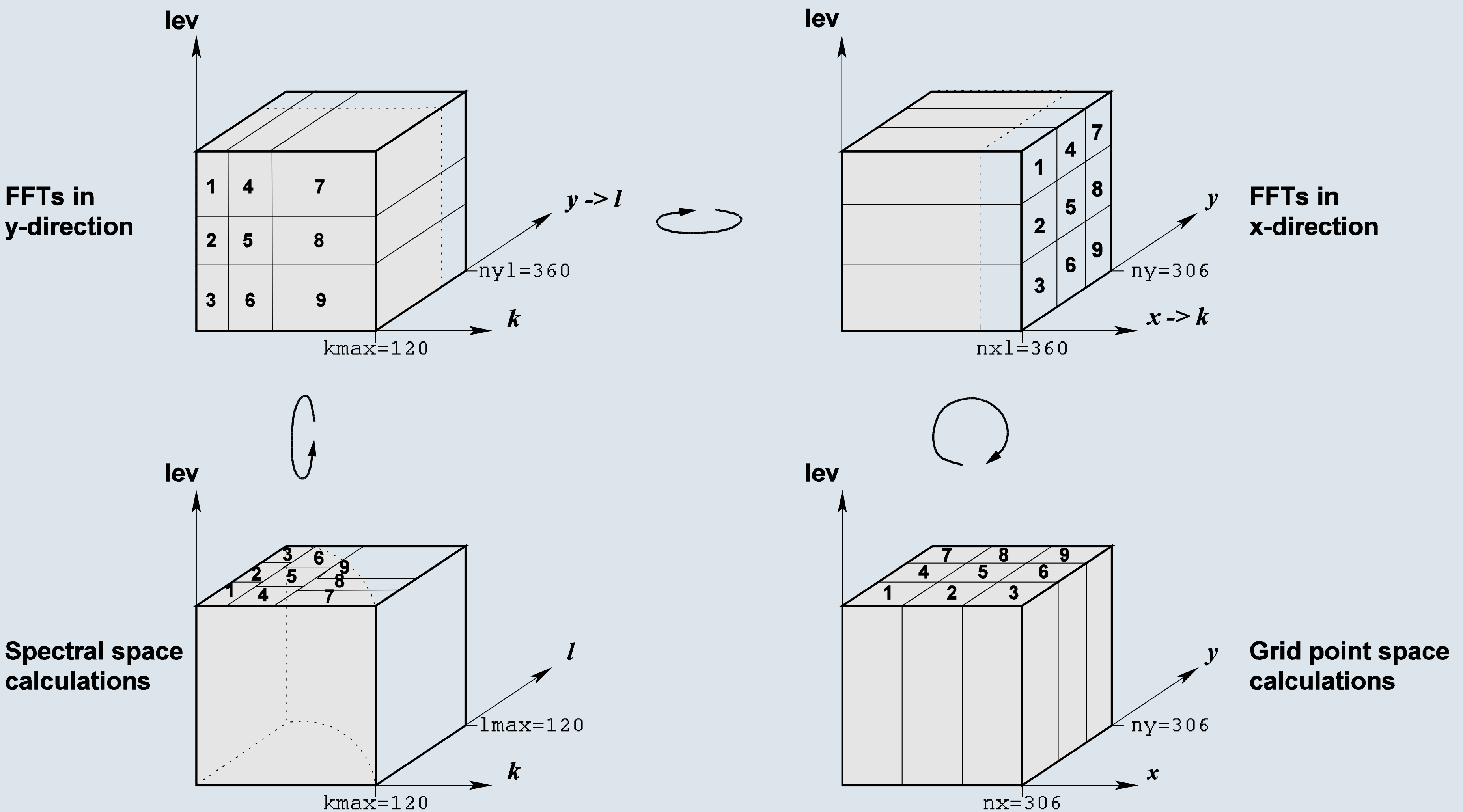
Efforts to improve HIRLAM 4D-VAR scaling

- **Switch from 1D to 2D partitioning?**

Transposes with 1D decomposition



Transposes with 2D partitioning



Efforts to improve HIRLAM 4D-VAR scaling

- **Switch from 1D to 2D partitioning?**
  - **Evaluated in a toy model 2005, no clear benefit**

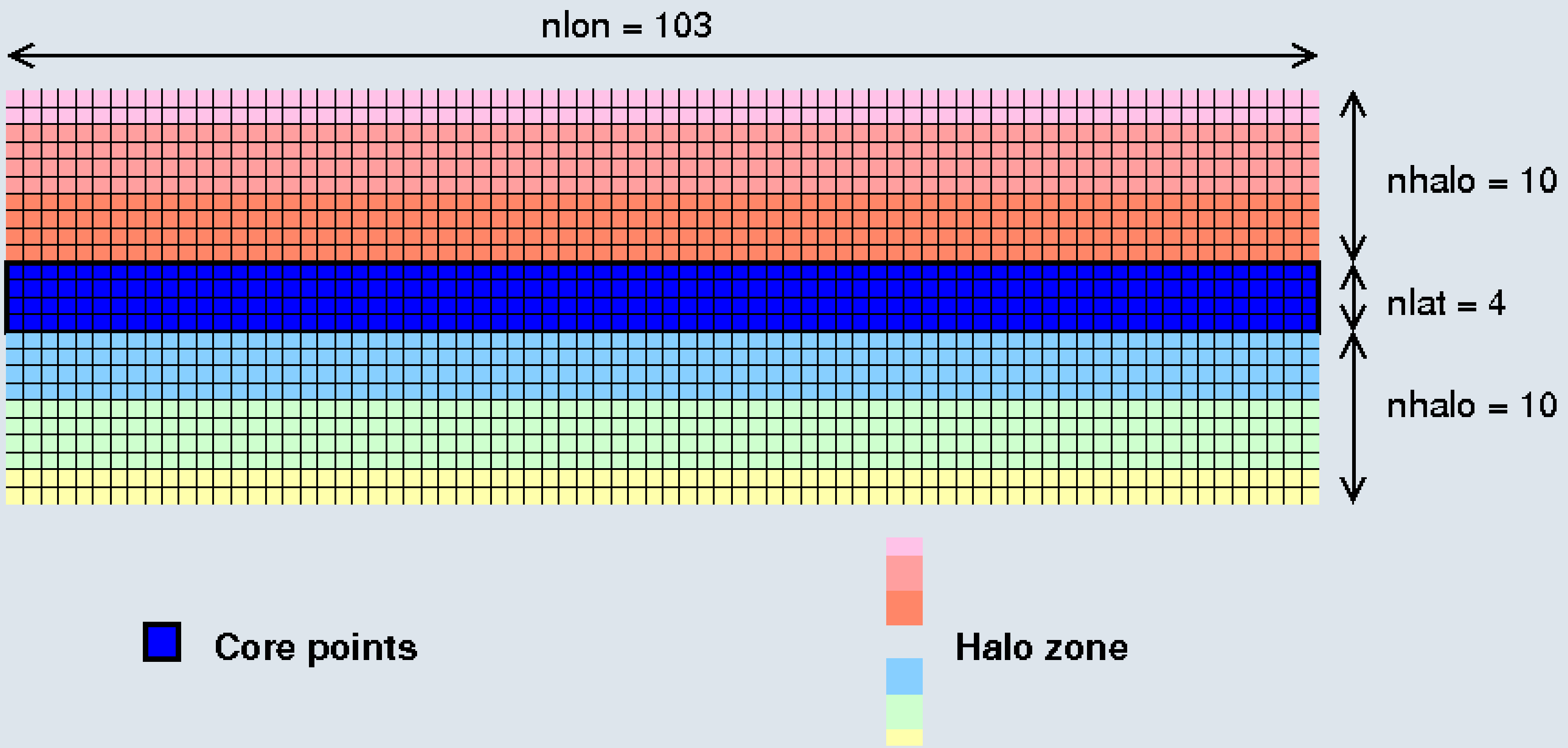
Efforts to improve HIRLAM 4D-VAR scaling

- **Switch from 1D to 2D partitioning?**
  - **Evaluated in a toy model 2005, no clear benefit**

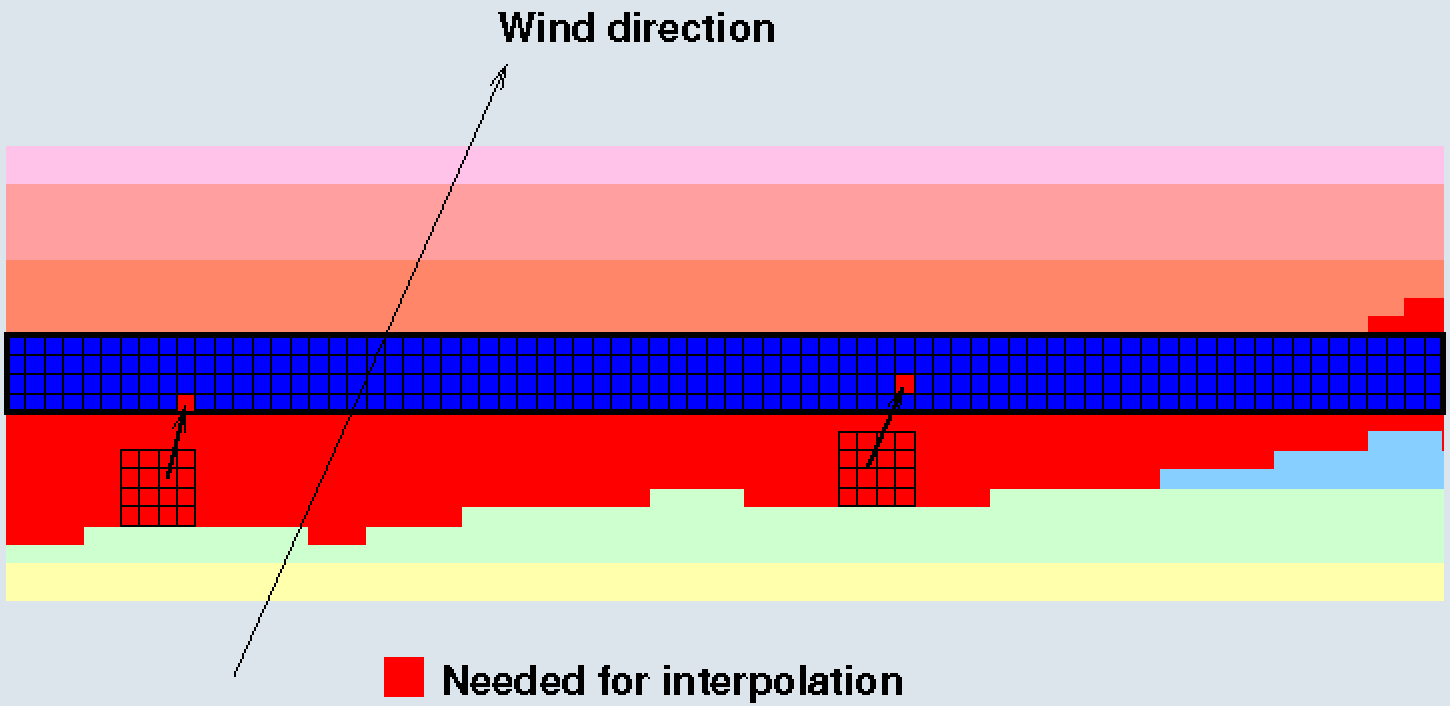
- **Reduce interprocessor communication**

HIRLAM 4D-VAR 1D partitioning

- **SMHI C22 area inner loop at 1/3 resolution, with 30 minute time step**
  - **103 x 103 grid distributed over 26 processors, each get a 103x4 slice**



Accumulated stencils in halo zone



Efforts to improve HIRLAM 4D-VAR scaling

- **Switch from 1D to 2D partitioning?**
  - **Evaluated in a toy model 2005, no clear benefit**

- **Reduce interprocessor communication**
  - **Slswap on demand, implemented 2008**
  - **Investigate "HALO-lite" (Mozdzynski, 2008) approach?**

Efforts to improve HIRLAM 4D-VAR scaling

- **Switch from 1D to 2D partitioning?**
  - **Evaluated in a toy model 2005, no clear benefit**

- **Reduce interprocessor communication**
  - **Slswap on demand, implemented 2008**
  - **Investigate “HALO-lite” (Mozdzynski, 2008) approach?**

- **Reduce work**
  - **Fewer FFTs, which also reduces communication (Gustafsson, 2008)**

Efforts to improve HIRLAM 4D-VAR scaling

- Switch from 1D to 2D partitioning?
  - Evaluated in a toy model 2005, no clear benefit

- Reduce interprocessor communication
  - Slswap on demand, implemented 2008
  - Investigate “HALO-lite” (Mozdzynski, 2008) approach?

- Reduce work
  - Fewer FFTs, which also reduces communication (Gustafsson, 2008)

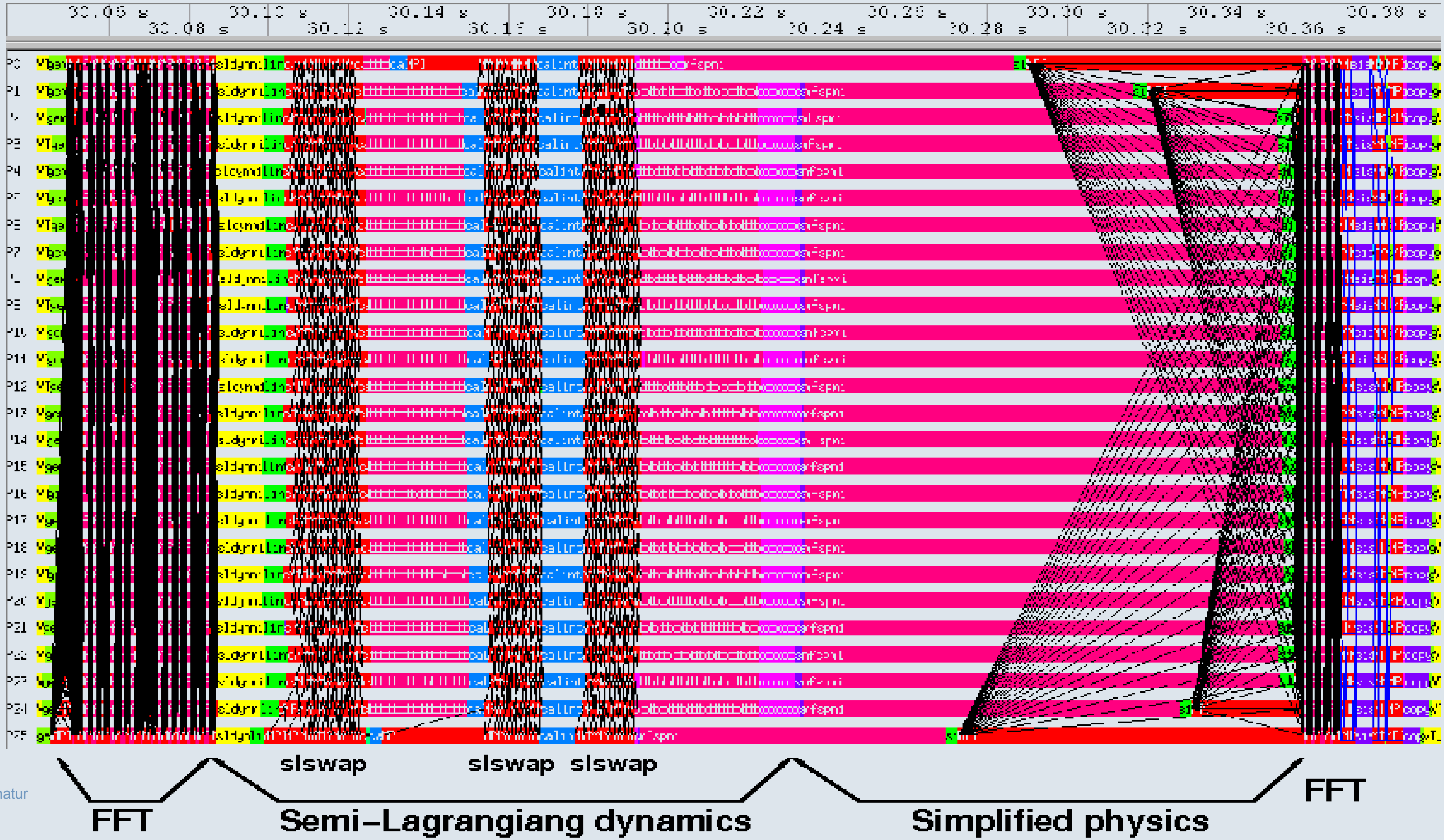
- Additional sources of parallelism
  - OpenMP

1D decomposition => a strict upper bound on number of MPI tasks

- SMHI C22 area has 306x306 grid points:
  - 4D-VAR inner loop at one third resolution with 103x103 grid points.
  - At least 2 rows per task (for efficiency) gives at most 52 MPI-tasks!
  - That is only 12% (7 out of 56 nodes) of SMHI's operational cluster.
- C11 area could use 24% of the machine, but has 4 times the work
- Even worse on future computers (Moore's law)
- OpenMP can enable better use of multi-core processors

SMHI

TL time step on 26 processors

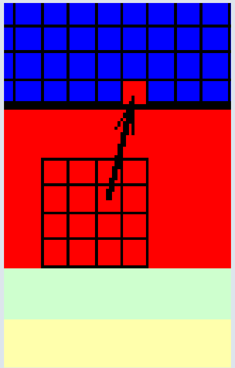


SMHI

OpenMP implementation

- **Physics was already "done" (through `phcall` and `phtask`)**
  - **But check `len_loops` in namelist (thread granularity)**
- **Semi-Langrangian dynamics by parallelizing outer "vertical" loop**
  - **Vertical loop over 40 to 60 levels**
  - **572 `!$omp` directives added in `spdy/*.F`**
  - **Large parallel sections, using orphaned directives**
  - **Some `nowait` directives**
  - **MPI calls within `!$omp master / !$omp end master`**

VERINT_AD: adjoint of interpolation stencil



- Departure points are interpolated with a 4*4*4 stencil
  - can be computed one level at a time
- In its adjoint, contributions are summed up over the stencil
  - writing to 4 levels => write races with parallelized vertical loop
- First implementation: use `omp_set_lock / omp_unset_lock` to limit concurrent writes to each vertical level
  - Slower than serial version!
- Second implementation: reuse the `#ifdef VECTOR` code ☺
  - Put contributions in temporary vector
  - Sum within `!$omp critical / !$omp end critical`
  - Scales beautifully

SMHI

HOWTO
combine Intel compiler OpenMP with Scali MPI

- **Set number of OpenMP threads**

```
setenv OMP_NUM_THREADS  4
```

- **Intel compiler runtime environment**

```
setenv KMP_LIBRARY turnaround
setenv KMP_STACKSIZE 128m
setenv KMP_AFFINITY "none,granularity=core"
setenv KMP_VERBOSE TRUE
```

- **ScaMPI version 3.13.8-5915 bug workaround**

```
setenv SCAFUN_CACHING_MODE 0
```

- **Launch (with socket affinity for up to 4 threads)**

```
mpirun -affinity_mode automatic:bandwidth:socket

mpirun -affinity_mode none
```

HIRLAM 7.2 4D-VAR OpenMP performance on gimle.nsc.liu.se (8 cores per node)

- SMHI C22 area, 306x306 grid, 40 “minimize” iterations
  - Inner TL/AD loop at 1/3 resolution, 103x103 grid
- Times for “minimize” (no I/O):
  - 13 nodes, best configuration for pure MPI or MPI/OpenMP hybrid:
    - 126 seconds with 52 tasks
    - 91 seconds with 26 tasks and 4 threads/task
  - 26 nodes:
    - 92 seconds with 52 tasks
    - 65 seconds with 52 tasks and 4 threads/task
- Best results with OpenMP within sockets, and MPI between

SMHI

Is 4D-VAR operationally feasible for SMHI's C11 area?

- **C11 area, 606x606 grid, 120 "minimize" iterations, total runtime**
- **Inner TL/AD loop at 1/3 resolution, 203x203 grid**

Time (seconds)	Model version	Configuration	Comment
ca 3600	7.1.2	24 nodes, 192 tasks, len_loops=2047	Original setup based on current operational C22
1752	7.1.2	26 nodes, 51 tasks, len_loops=16	Improved configuration
1486	7.2	as above	Fewer FFTs, slswap on demand
1387	7.2	as above, and 102 tasks	2*tasks
1286	7.3	as above, and 2 threads => 204 cores	2 OpenMP threads
1086	7.3	26 nodes, 51 tasks, 4 threads => 204 cores	Best setup!

Signatur

Conclusion

- **OpenMP in HIRLAM 4D-VAR necessary for today's clusters**

- **Slswap on demand works, and expected to be more important for high numbers of processor**

- **Nothing beats avoiding work completely**

- **No silver bullet**
  - **Performance from small incremental improvements**

Technology will allow larger areas "for free", but increasing the number of time steps and maintaining runtime will be harder

ITC - Intel Thread Checker

- Finds OpenMP parallelization bugs, like possible race conditions
- Very easy to use
- Compile
  - `ifort -openmp -tcheck`
  - No optimization (any –O is ignored)
  - No specific thread knowledge allowed, like `omp_get_thread_num()`
- Instrument and run
  - `tcheck_cl a.out`
  - Takes an order of magnitude more memory and runtime

SMHI

Intel Thread Checker – Output

```
------------------------------------------------------------------------------
|Write -> |Err|omp p|Memory write of w0 at                |"horin|"horin|
|Write    |or |arall|"horint_ad_local.F":113 conflicts     |t_ad_l|t_ad_l|
|data-race|     el re|with a prior memory write of w0 at|ocal.F|ocal.F|
|         |   |gion |"horint_ad_local.F":113 (output      |":113 |":113 |
|         |   |     |dependence)                          |      |      |

------------------------------------------------------------------------------

|Write -> |Err|omp p|Memory read of ddiv_ad at            |"calin|"calin|
|Read     |or |arall|"calintf.F":412 conflicts with a     |tf.F":|tf.F":|
|data-race|   |el re|prior memory write of                |412   |412   |
|         |   |gion |ddiv_ad at "calintf.F":412           |      |      |
|         |   |     |(flow dependence)                    |      |      |

------------------------------------------------------------------------------
```

1. **Forgot to declare w0 as private in horint_ad_local.F**

2. **Wrongly placed nowait in calintf.F**