Towards Exascale Computing with the Atmospheric Model NUMA

Andreas Müller,
Daniel S. Abdi,
Michal Kopera,
Lucas Wilcox,
Francis X. Giraldo
Department of Applied Mathematics
Naval Postgraduate School, Monterey (California), USA

Tim Warburton
Virginia Tech
Blacksburg (Virginia), USA

- **NUMA** = **N**on-hydrostatic **U**nified **M**odel of the **A**tmosphere
- dynamical core inside the Navy's next generation weather prediction system NEPTUNE (**N**avy's **E**nvironment **P**rediction sys**T**em **U**sing the **N**uma **E**ngine)
- developed by **Prof. Francis X. Giraldo** and generations of postdocs

Goal

- **NOAA: HIWPP project plan: Goal for 2020:**
  ~ 3km – 3.5km global resolution within operational requirements

Goal

- **NOAA: HIWPP project plan: Goal for 2020:**

  ~ 3km – 3.5km global resolution within operational requirements

- **Achieved with NUMA:** baroclinic wave test case at 3.0km within 4.15 minutes per one day forecast on supercomputer Mira

  double precision, no shallow atmosphere approx., arbitrary terrain, IMEX in the vertical

Goal

- **NOAA: HIWPP project plan: Goal for 2020:**

  ~ 3km – 3.5km global resolution within operational requirements

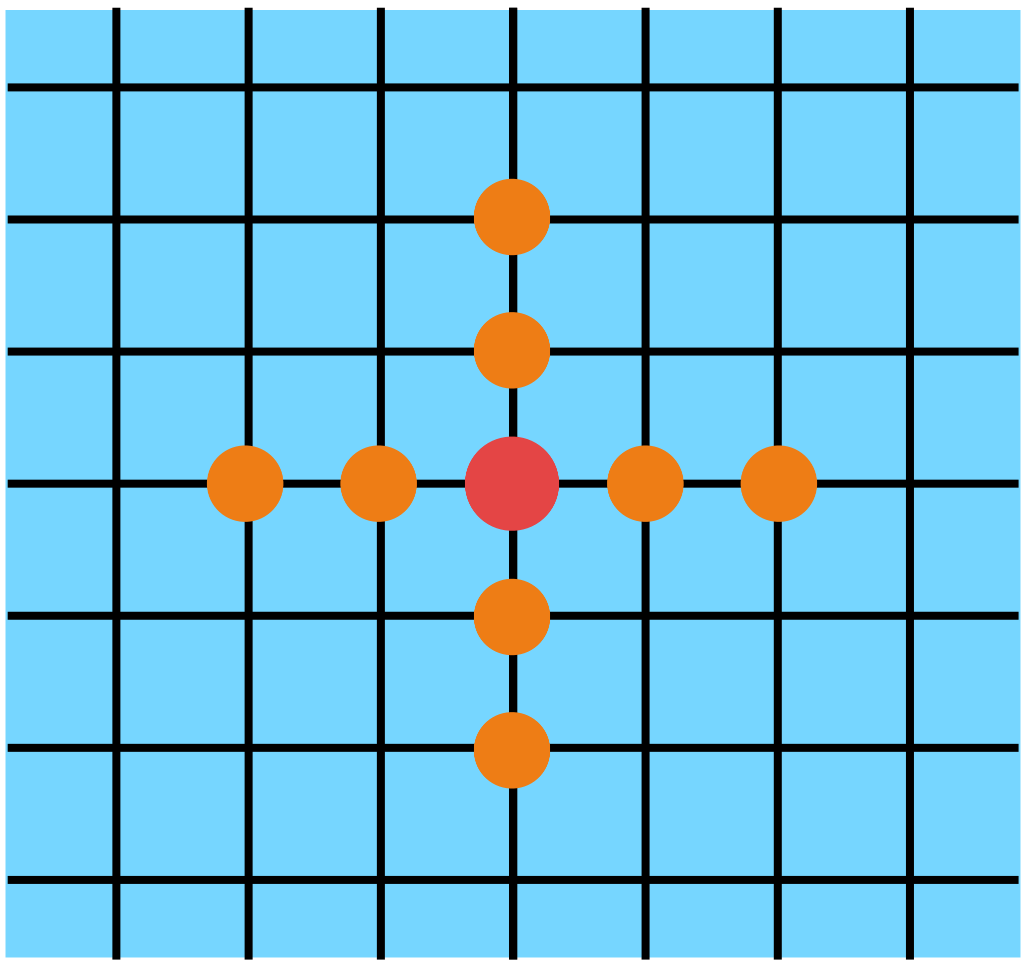
- **Achieved with NUMA:** baroclinic wave test case at 3.0km within 4.15 minutes per one day forecast on supercomputer Mira

  double precision, no shallow atmosphere approx., arbitrary terrain, IMEX in the vertical

- **Expect:** 2km by doing more optimizations

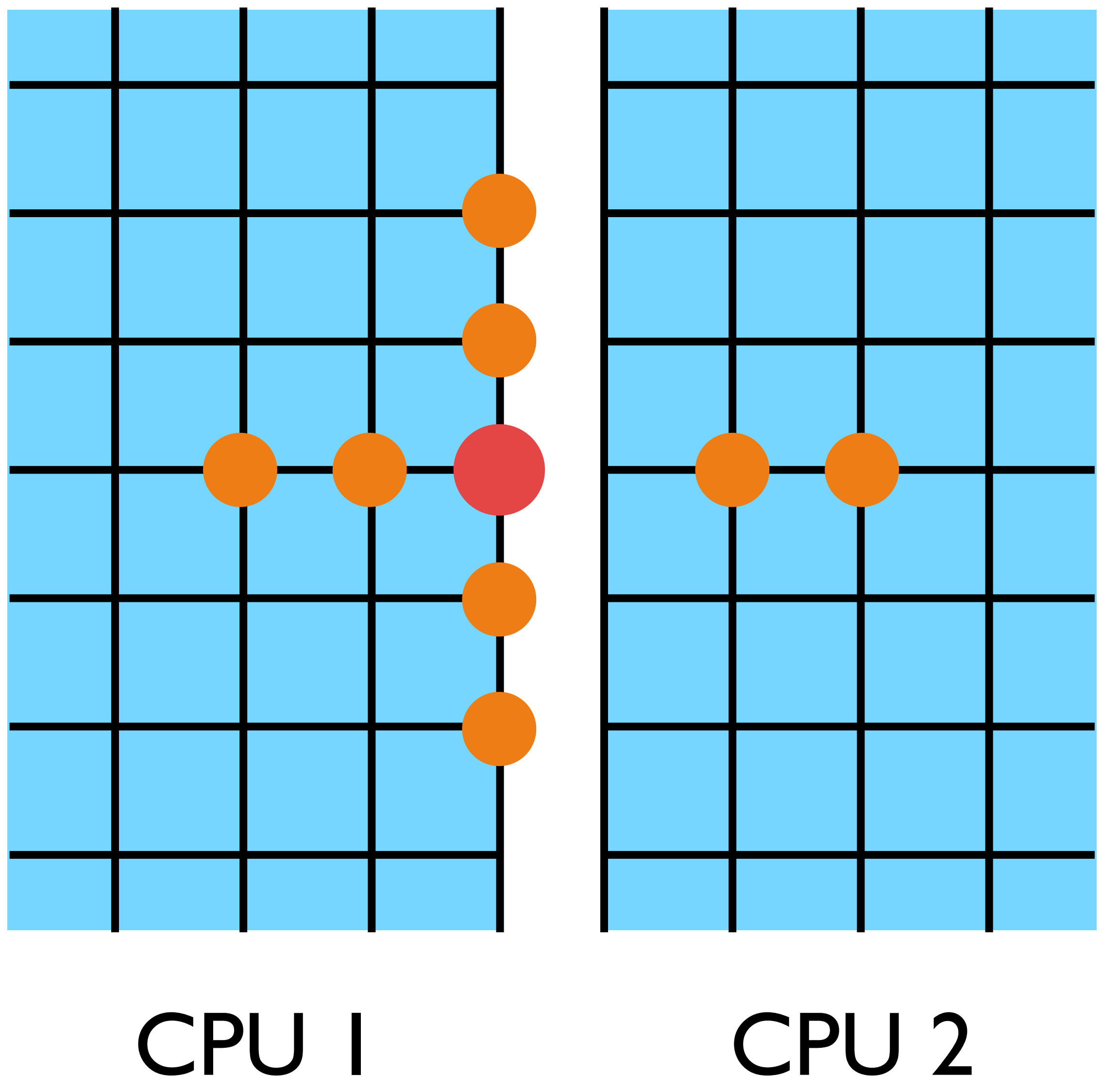
Communication between processors

4th order Finite Difference



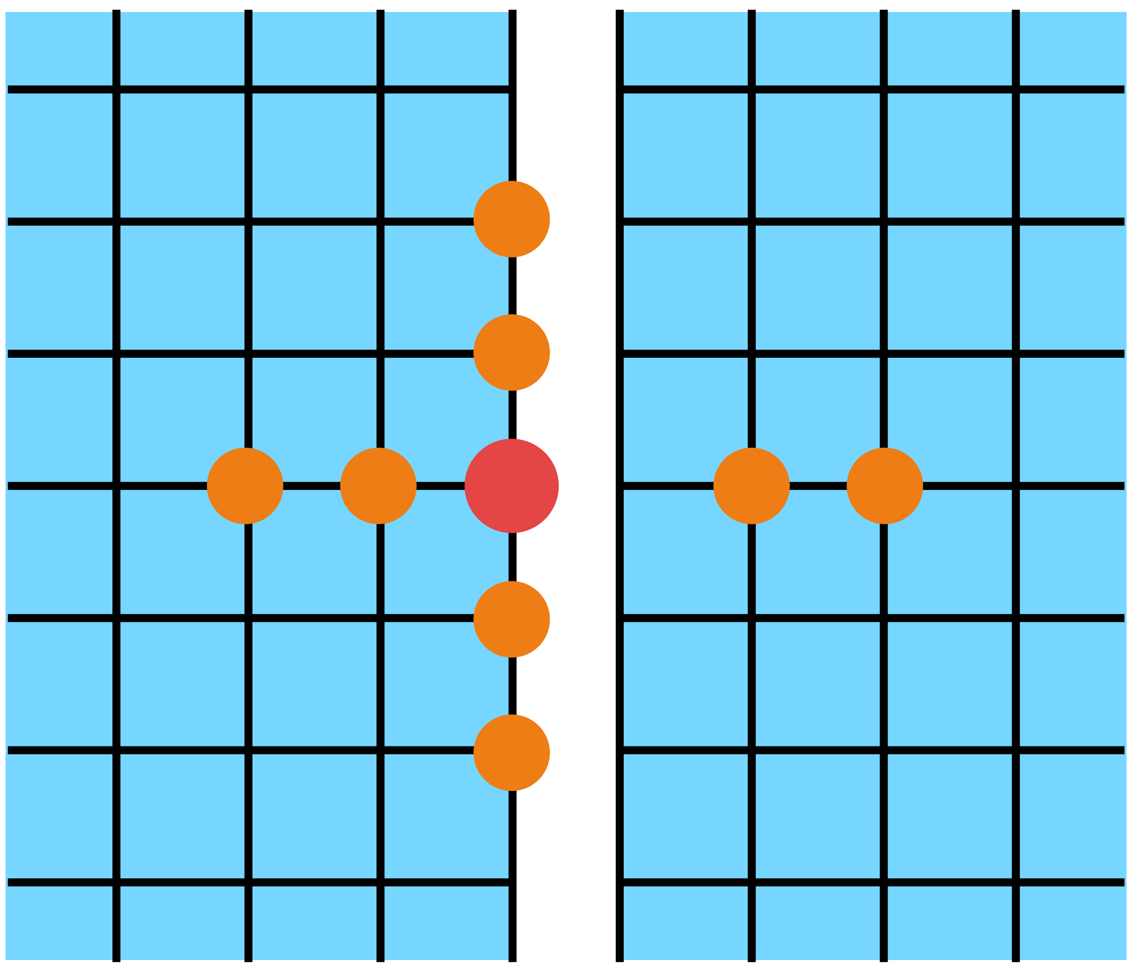
Communication between processors

4th order Finite Difference



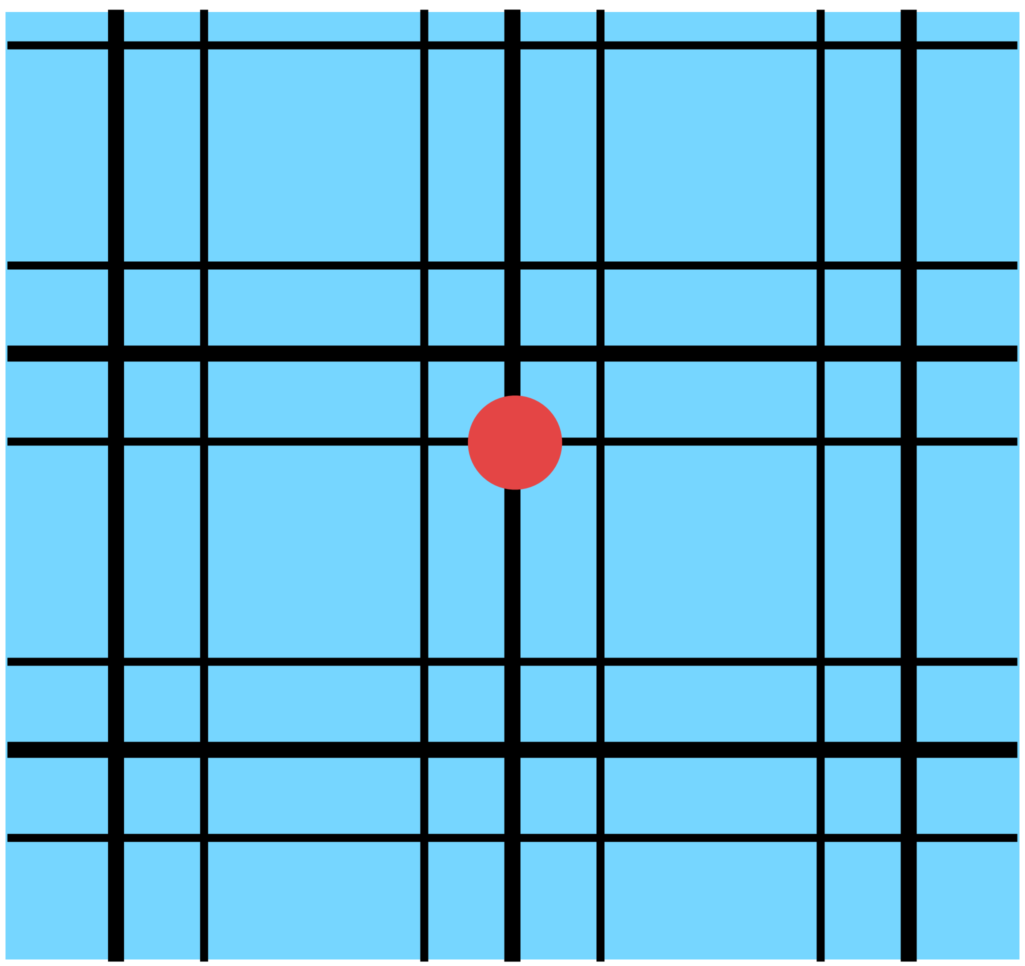
Communication between processors

4th order Finite Difference



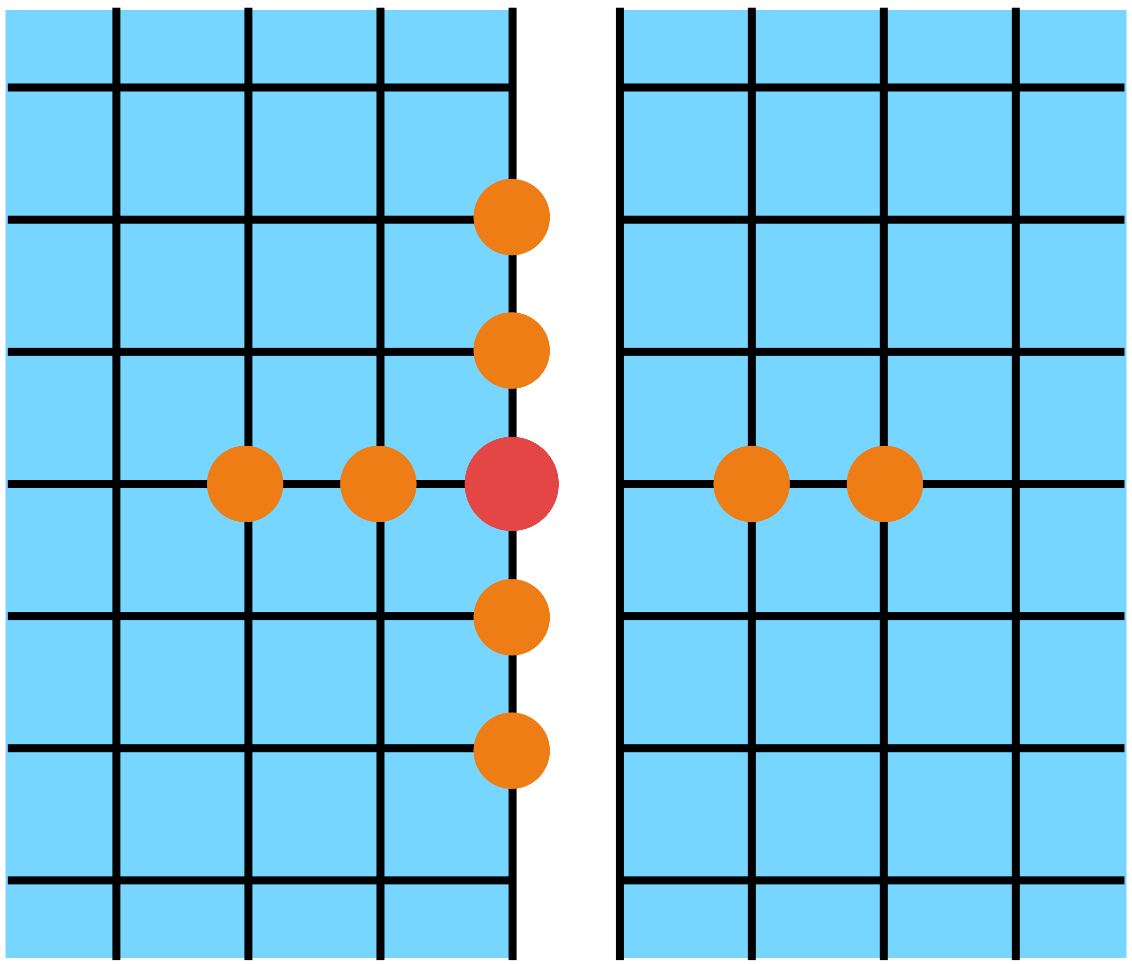
CPU 1 CPU 2

compact stencil methods (like in NUMA)



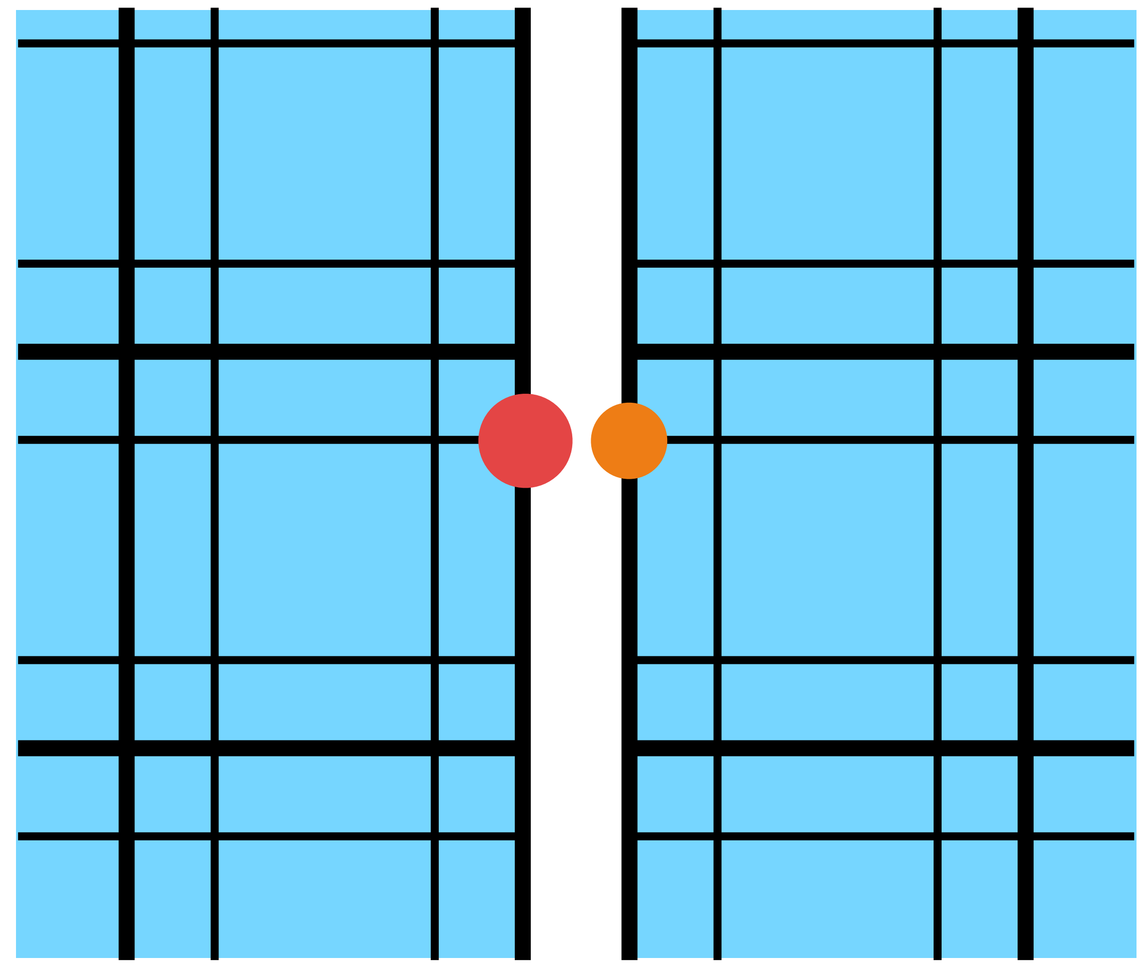
Communication between processors

4th order Finite Difference



CPU 1 CPU 2

compact stencil methods (like in NUMA)

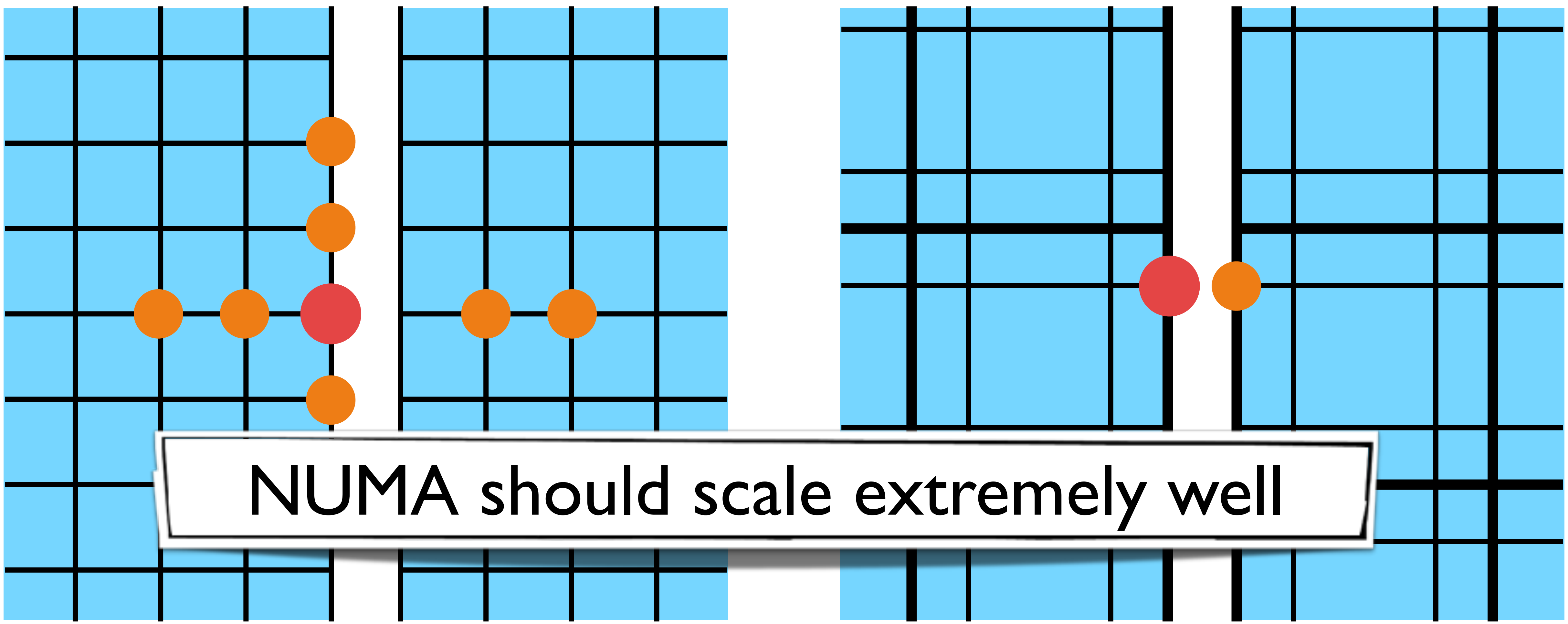


CPU 1 CPU 2

Communication between processors

4th order Finite Difference

compact stencil methods (like in NUMA)



CPU 1 CPU 2 CPU 1 CPU 2

Fastest Supercomputers of the World
according to top500.org

#	NAME	COUNTRY	TYPE	PEAK (PFLOPS)
1	TaihuLight	China	Sunway	125.4
2	Tianhe-2	China	Intel Xeon Phi	54.9
3	Titan	USA	NVIDIA GPU	27.1
4	Sequoia	USA	IBM BlueGene	20.1
5	K computer	Japan	Fujitsu CPU	11.2
6	Mira	USA	IBM BlueGene	10.0

1 PFlops = 10^{15} floating point ops. per sec.

Fastest Supercomputers of the World

according to top500.org

#	NAME	COUNTRY	TYPE	PEAK (PFLOPS)
1	TaihuLight	China	Sunway	125.4
2	Tianhe-2	China	Intel Xeon Phi	54.9
3	**Titan**	**USA**	**NVIDIA GPU**	**27.1**
4	Sequoia	USA	IBM BlueGene	20.1
5	K computer	Japan	Fujitsu CPU	11.2
6	**Mira**	**USA**	**IBM BlueGene**	**10.0**

1 PFlops = 10^{15} floating point ops. per sec.

overview

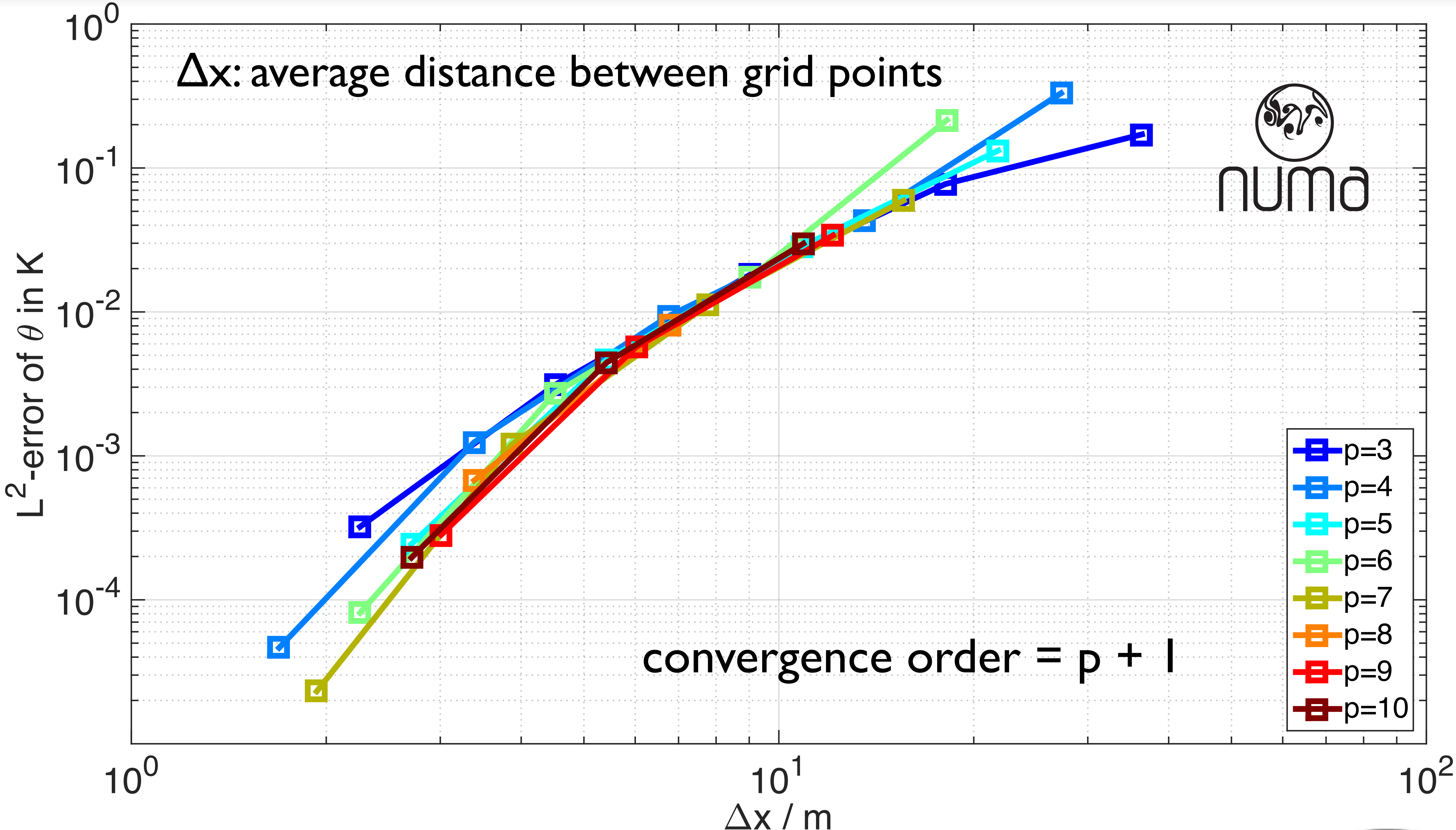
- Mira: Blue Gene
  strategy: optimize one setup for one specific computer
- Titan: GPUs
  strategy: keep all options, portability on CPUs and GPUs
- Intel Knights Landing

Mira: Blue Gene

optimize one setup for this specific computer

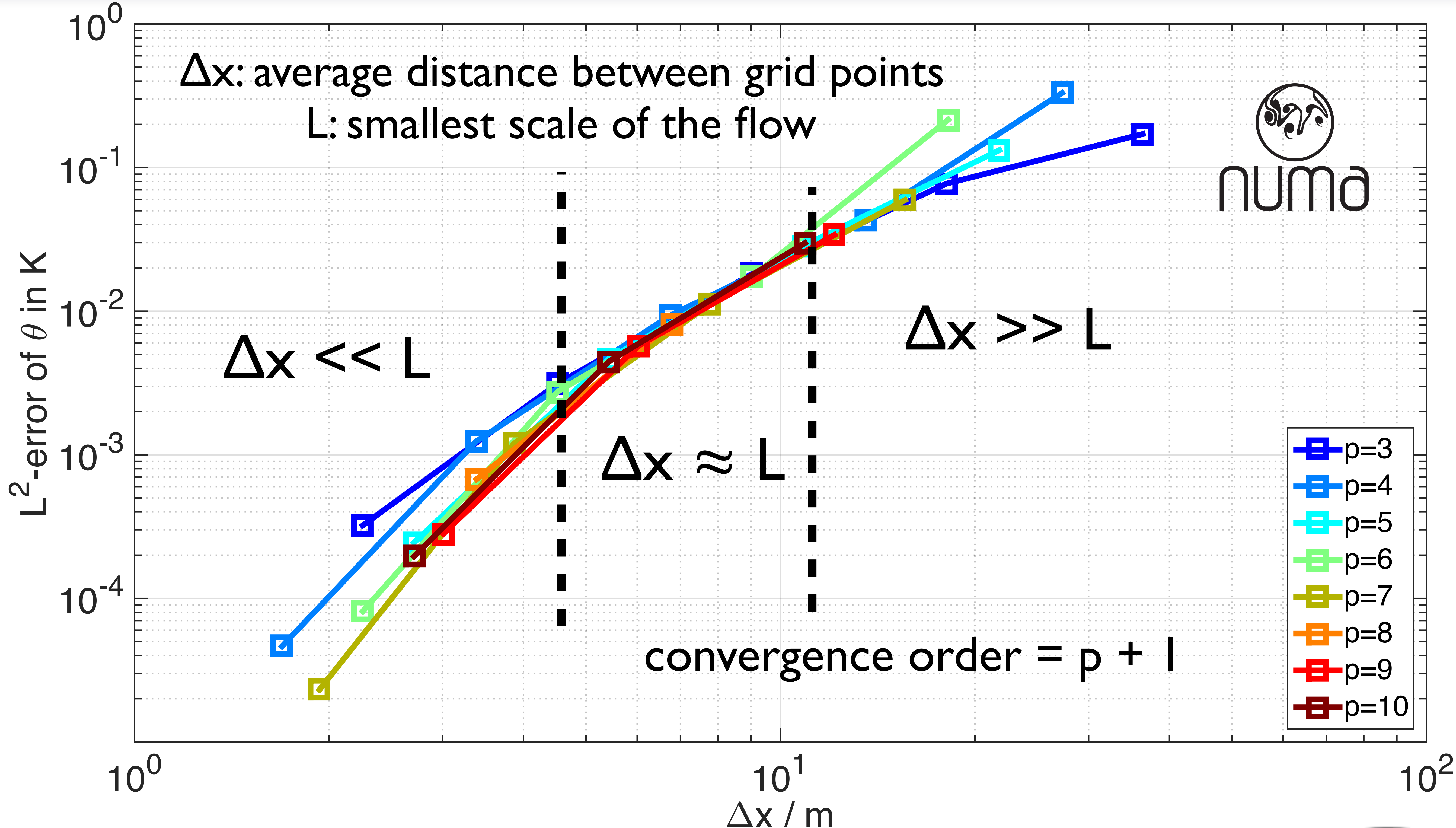
Which convergence order to use?

L2-error of pot. temperature for 2D rising bubble with viscosity μ=0.1m^2/s



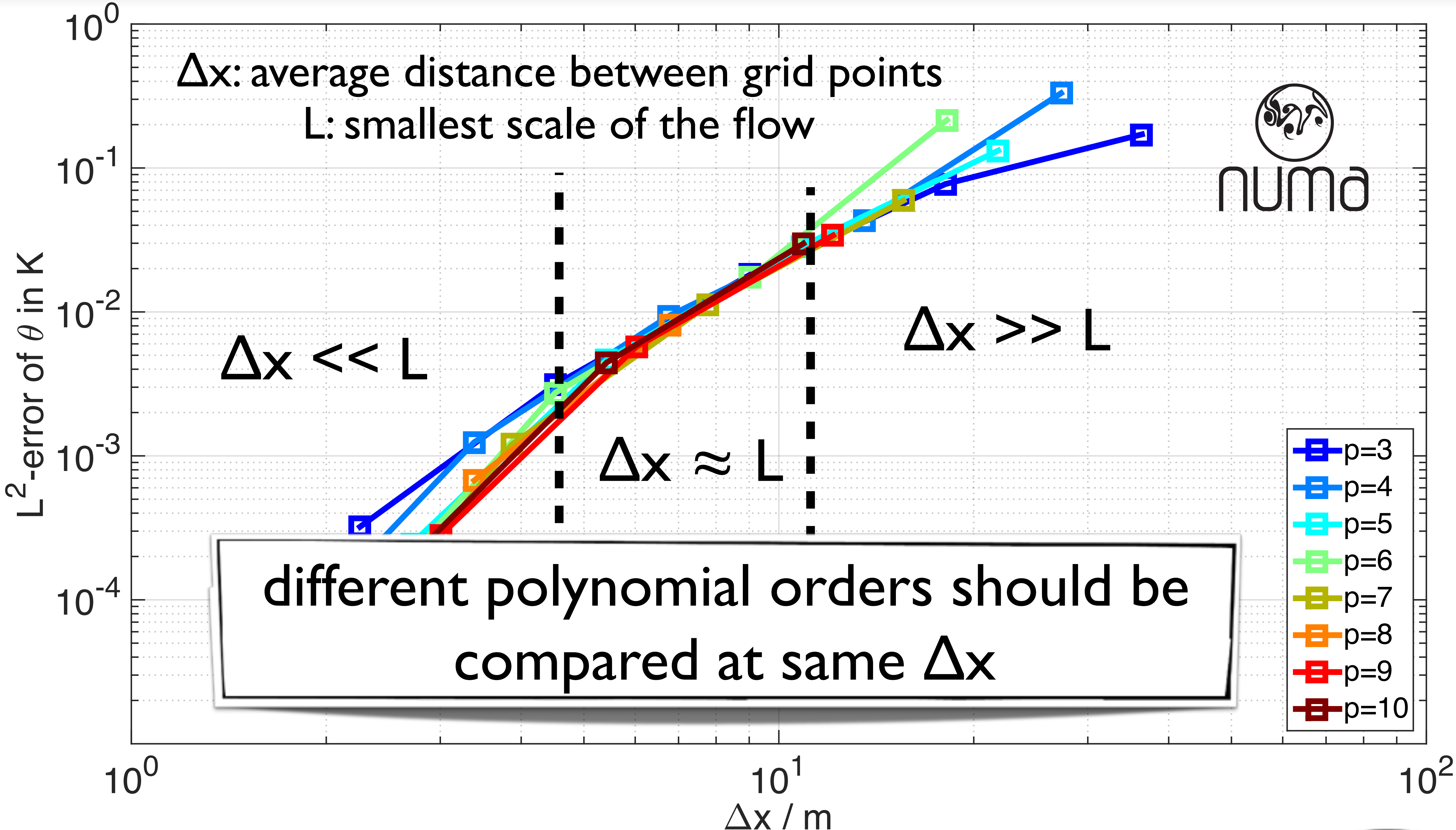
Which convergence order to use?

L2-error of pot. temperature for 2D rising bubble with viscosity μ=0.1m²/s



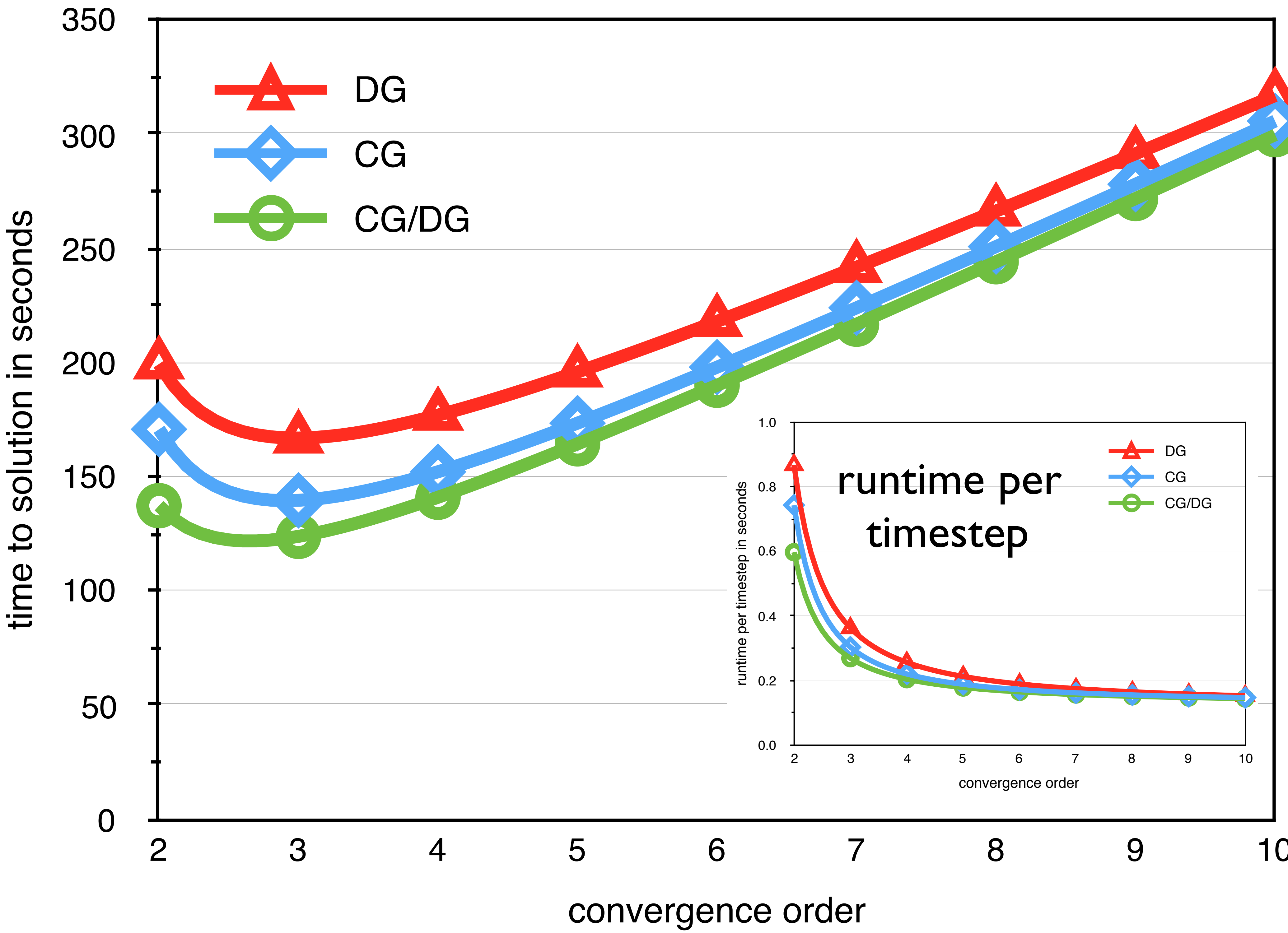
Which convergence order to use?

L2-error of pot. temperature for 2D rising bubble with viscosity μ=0.1m²/s



Which convergence order to use?

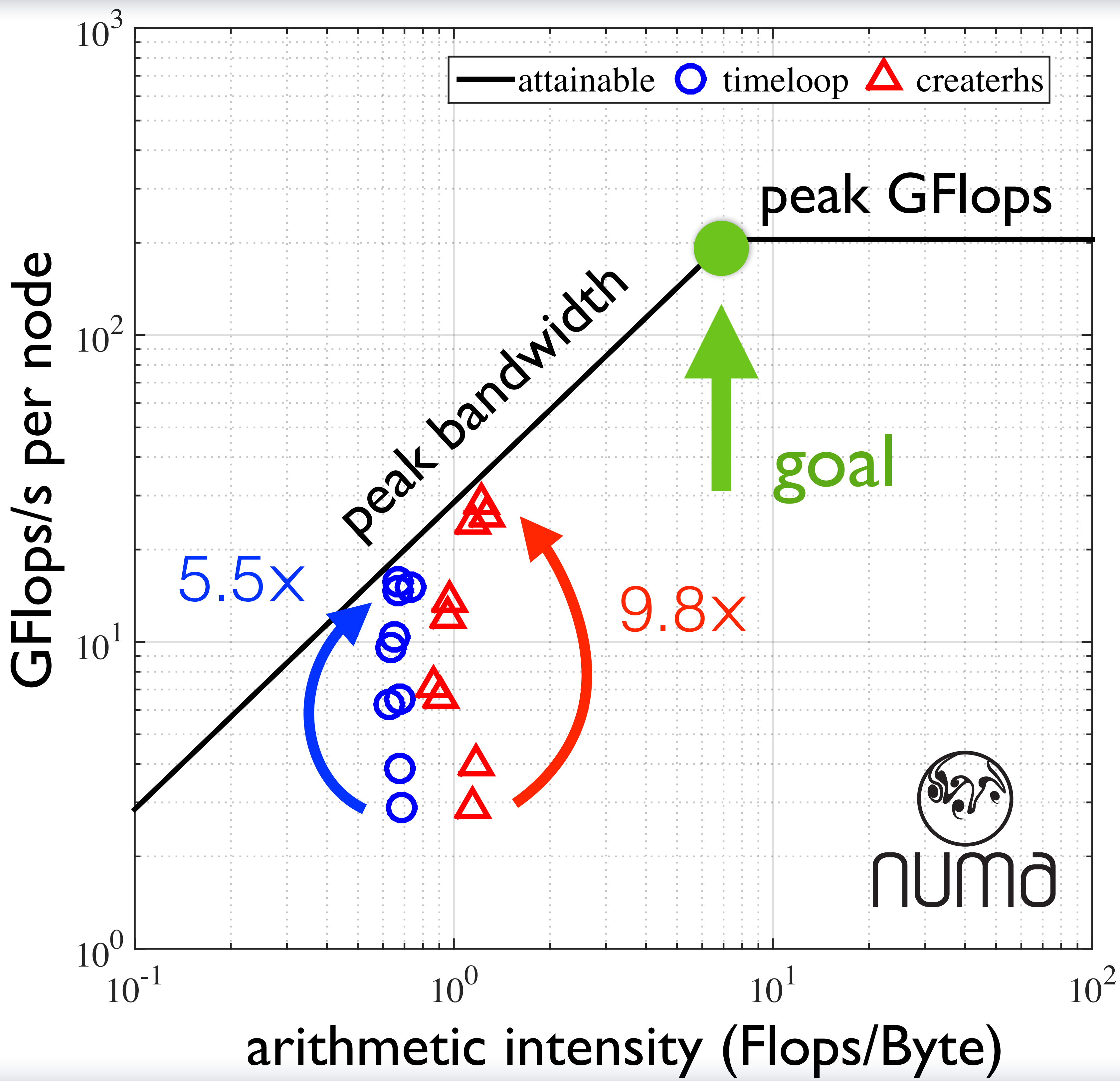
theoretical performance model for fixed Δx



Mira Titan KNL

Measurements: roofline plot

rising bubble test case on Mira



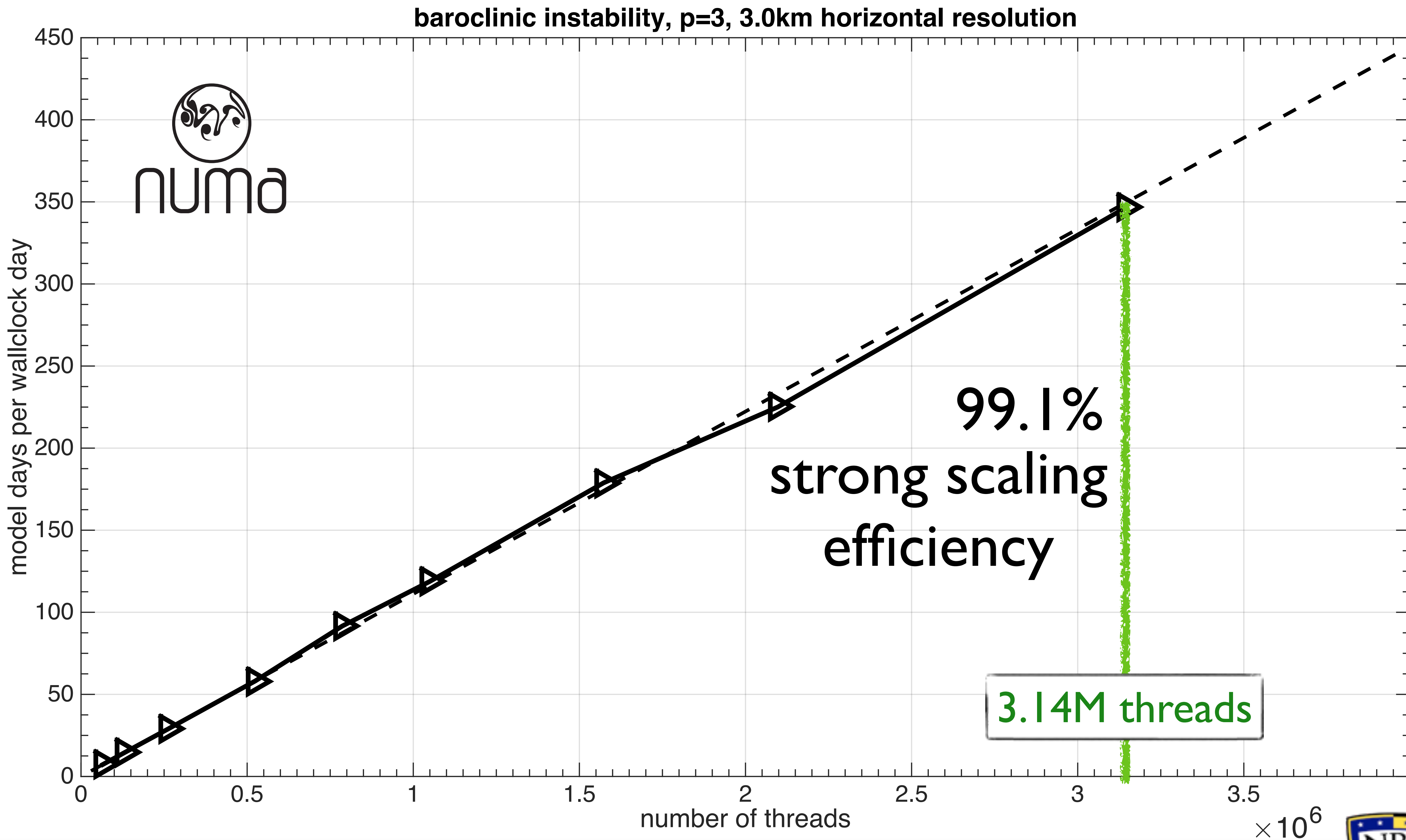
blue: entire timeloop

red: main computational kernel

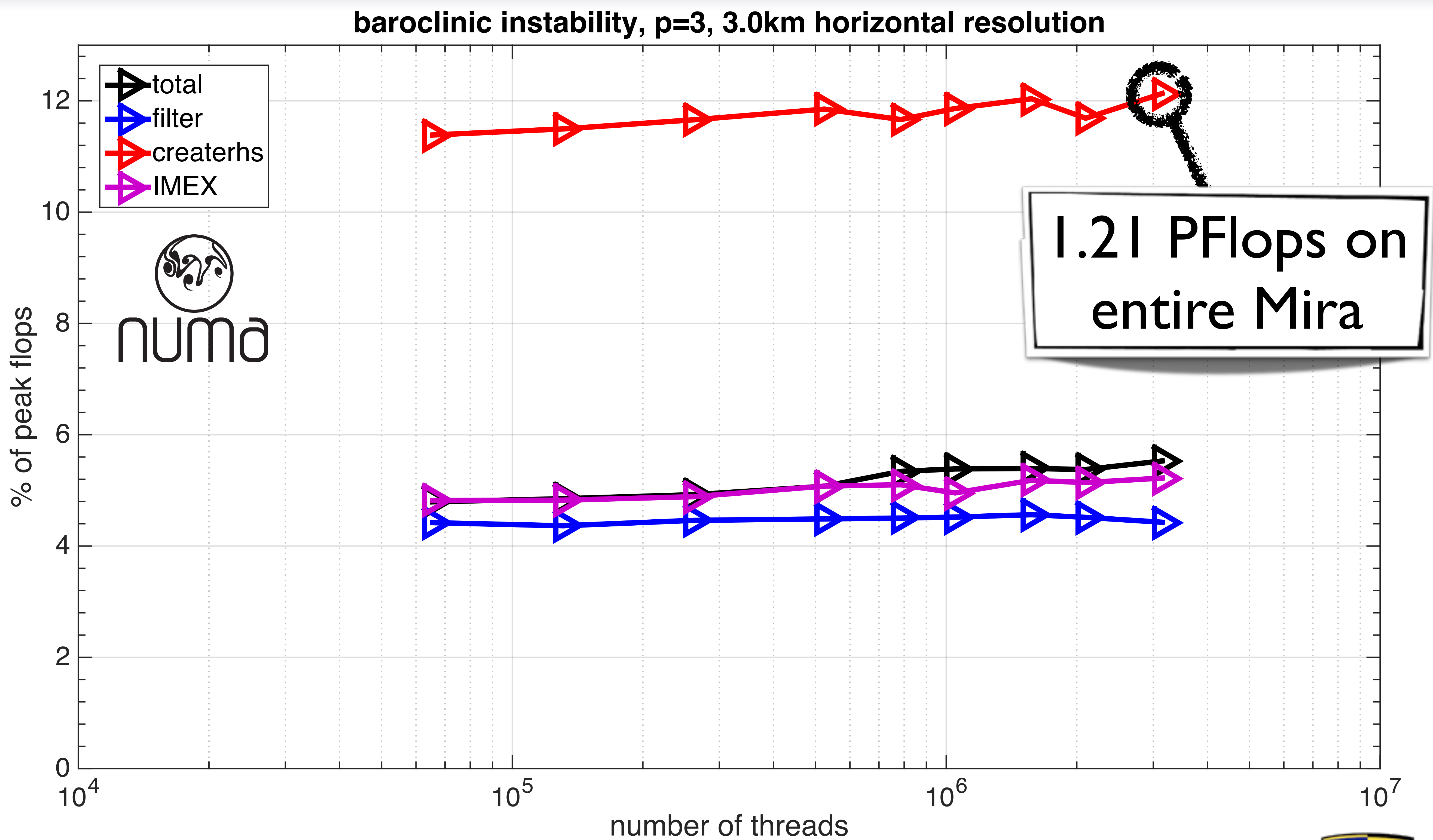
data points: different optimization stages

Strong scaling with NUMA

1.8 billion grid points (3.0km horizontal, 31 levels vertical)

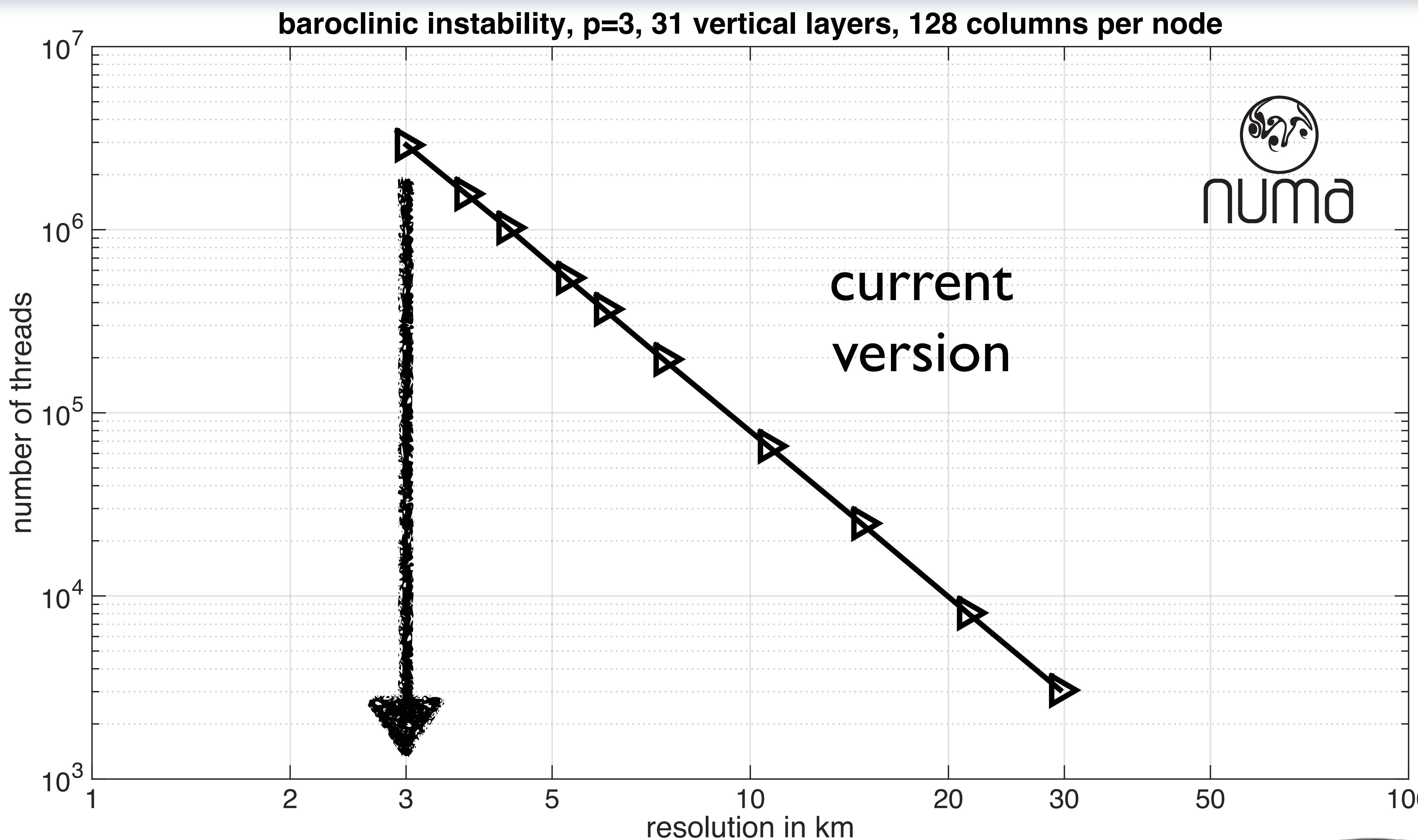


12.1% of theoretical peak (flops)



Where are we heading?

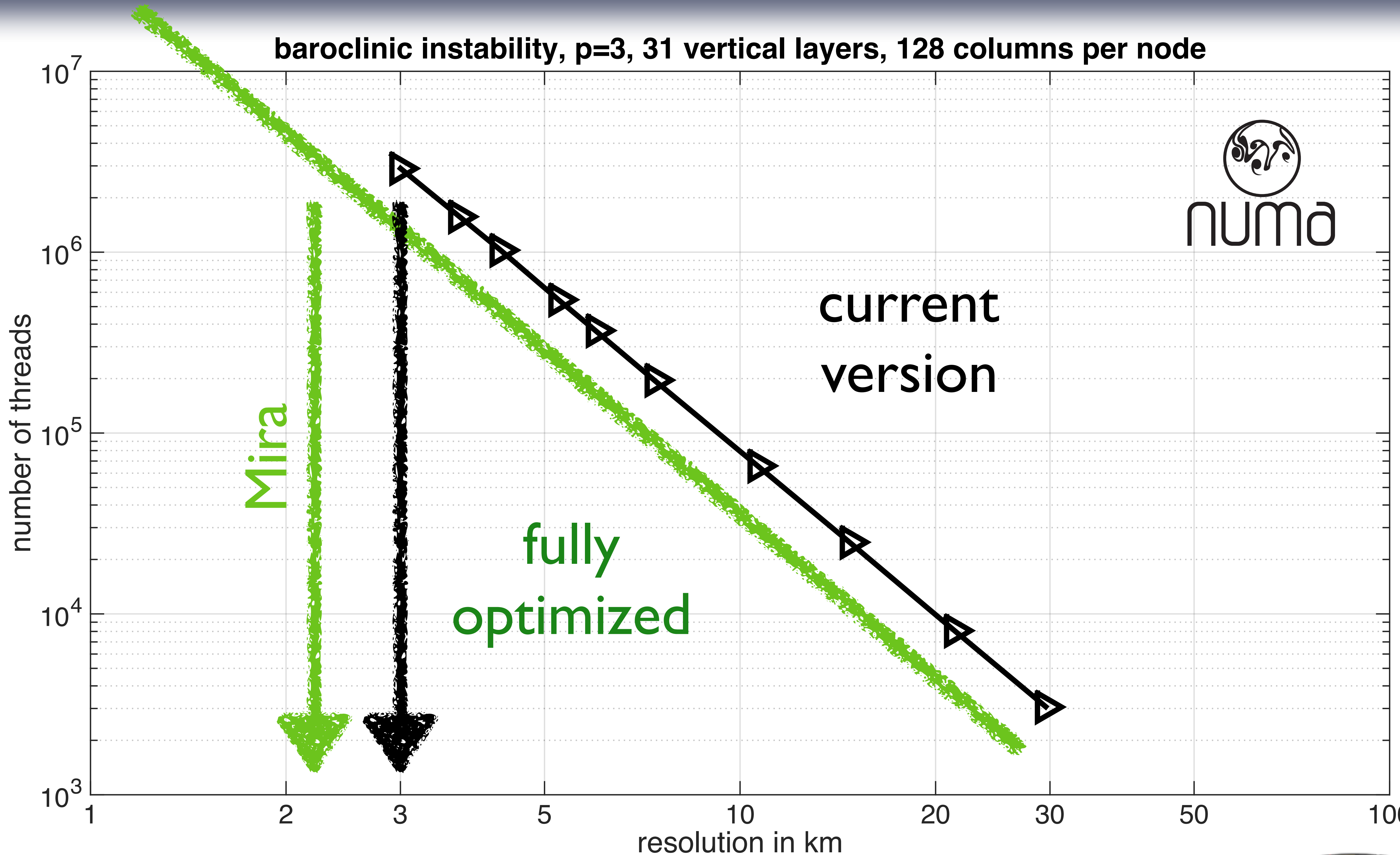
dynamics within 4.5 minutes runtime per one day forecast



Mira Titan KNL

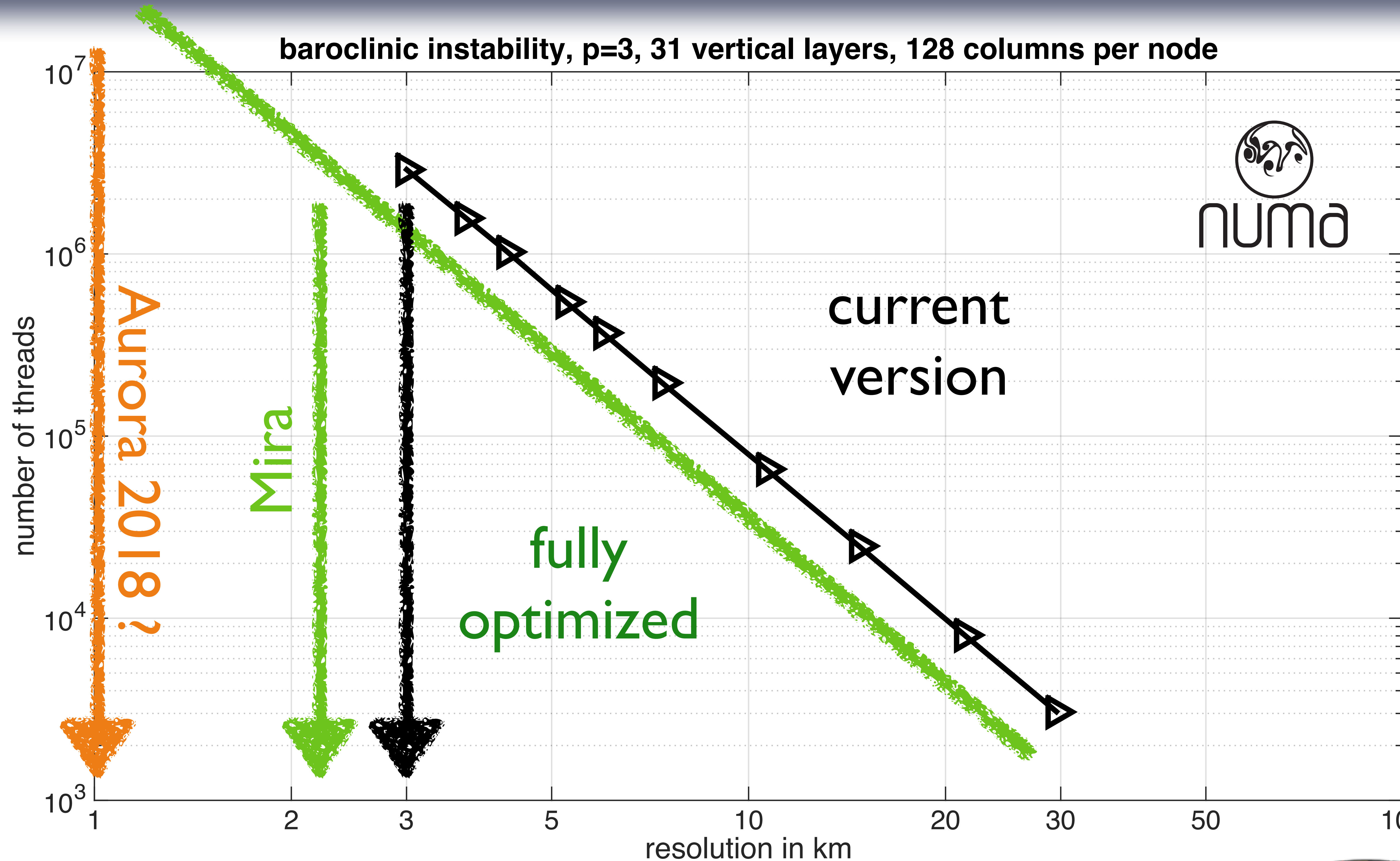
Where are we heading?

dynamics within 4.5 minutes runtime per one day forecast



Where are we heading?

dynamics within 4.5 minutes runtime per one day forecast



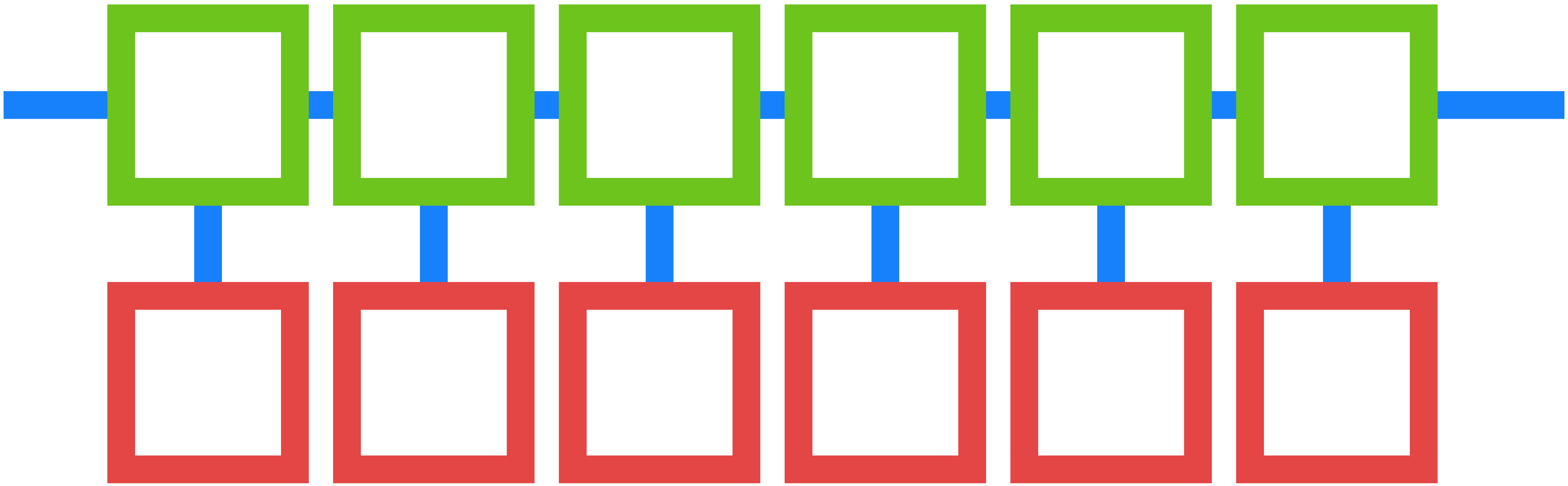
Titan: GPUs

keep all options, portability on CPUs and GPUs

Titan

18,688 CPU nodes
(16 cores each)

network

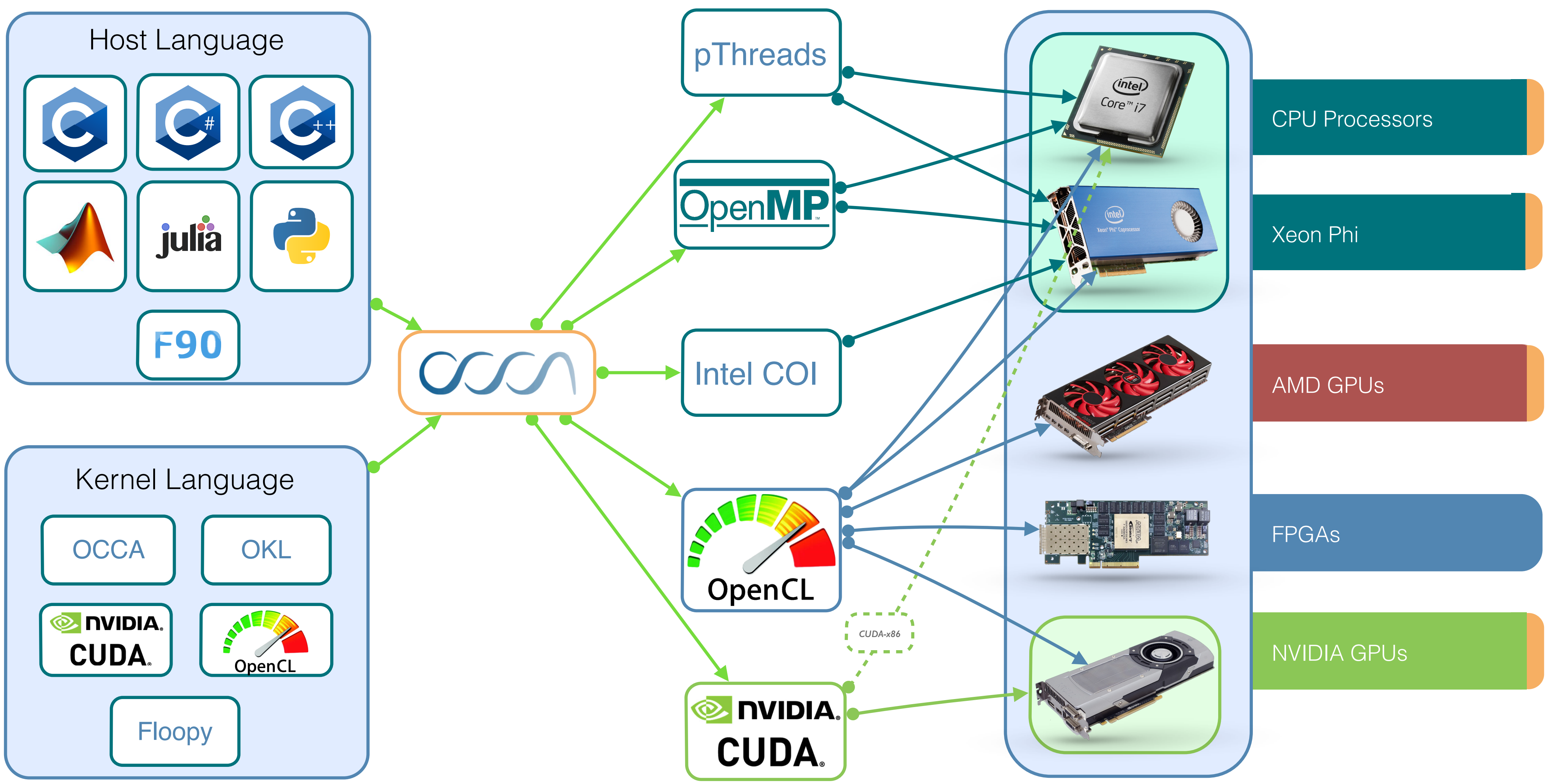


18,688 NVIDIA GPUs
(2,688 CUDA cores each)

OCCA2: unified threading model

(slide: courtesy of Lucas Wilcox and Tim Warburton)

Portability & extensibility: device independent kernel language (or OpenCL / CUDA) and native host APIs.



Available at: `https://github.com/tcew/OCCA2`

Floopy/OCCA2 code

(slide: courtesy of Lucas Wilcox and Tim Warburton)

```
allocate(a(1:entries), b(1:entries), ab(1:entries), stat = alloc_err)
if (alloc_err /= 0) stop "*** Not enough memory ***"

do i=1,entries
  a(i) = i-1
  b(i) = 2-i
  ab(i) = 0
end do

device = occaGetDevice(mode, platformID, deviceID)

o_a  = occaDeviceMalloc(device, int(entries,8)*4_8)
o_b  = occaDeviceMalloc(device, int(entries,8)*4_8)
o_ab = occaDeviceMalloc(device, int(entries,8)*4_8)

addVectors = occaBuildKernelFromFloopy(device, "addVectors.floopy", "addVectors", "")

dims     = 1
innerDim = 16

call occaKernelSetAllWorkingDims(addVectors, dims, &
                                 int(innerDim,8), 1_8, 1_8, &
                                 int((entries + innerDim - 1)/innerDim,8), 1_8, 1_8)

call occaCopyPtrToMem(o_a, a(1), int(entries,8)*4_8, 0_8);
call occaCopyPtrToMem(o_b, b(1));

call occaKernelRun(addVectors, occaTypeMem_t(entries), o_a, o_b, o_ab)

call occaCopyMemToPtr(ab(1), o_ab);

print *,"a = ", a(:)
print *,"b = ", b(:)
print *,"ab = ", ab(:)

deallocate(a, b, ab, stat = alloc_err)
if (alloc_err /= 0) stop "*** deallocation not successful ***"
```

Floopy/OCCA2 code

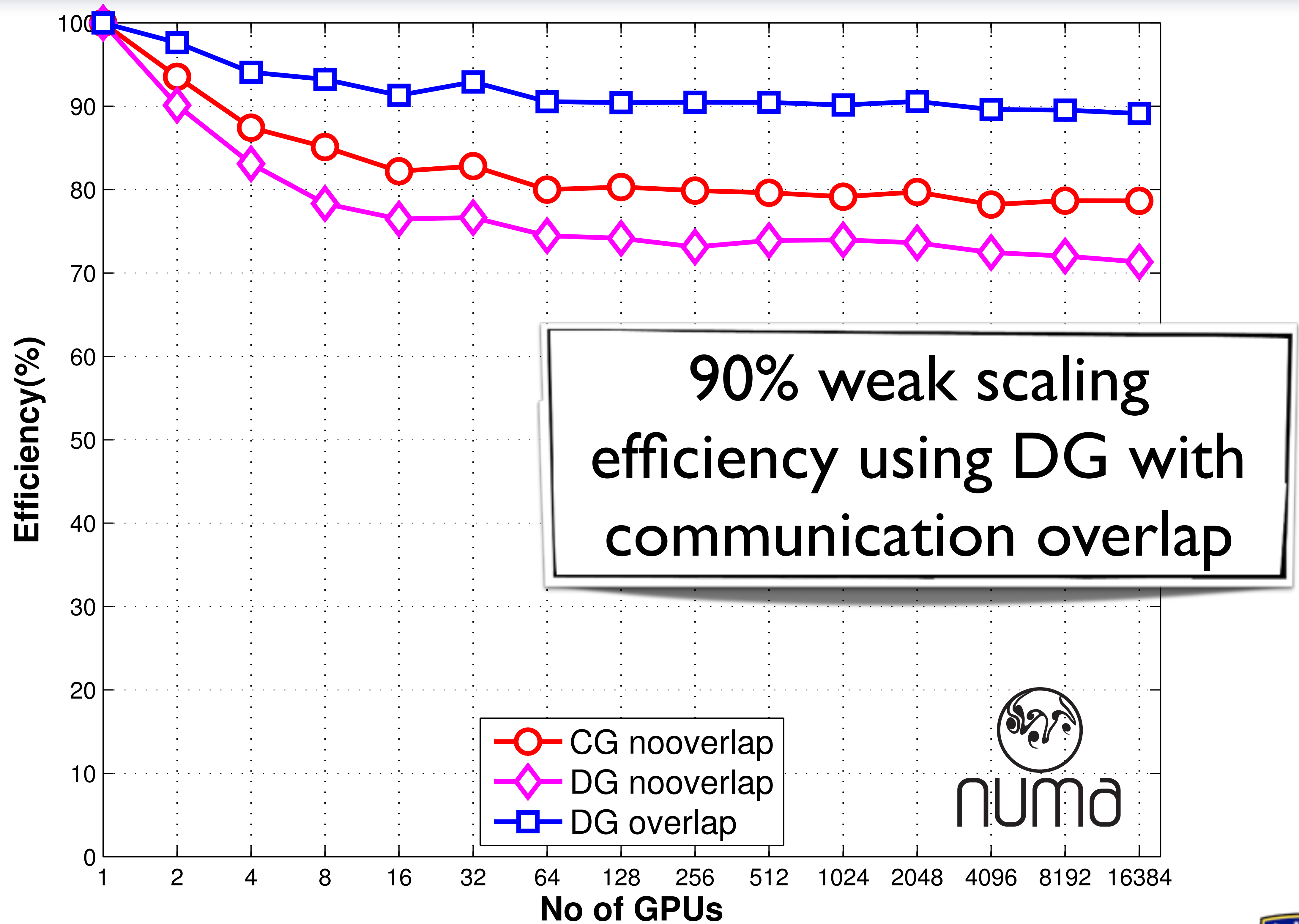
(slide: courtesy of Lucas Wilcox and Tim Warburton)

```
allocate(a(1:entries), b(1:entries), ab(1:entries), stat = alloc_err)
if (alloc_err /= 0) stop "*** Not enough memory ***"

do i=1,entries
  a(i) = i-1
  b(i) = 2-i
  ab(i) = 0
end do

device = occaGetDevice(mode, platformID, deviceID

o_a  = occaDeviceMalloc(device, int(entries,8)*4_
o_b  = occaDeviceMalloc(device, int(entries,8)*4_
o_ab = occaDeviceMalloc(device, int(entries,8)*4_

addVectors = occaBuildKernelFromFloopy(device, "a

dims     = 1
innerDim = 16

call occaKernelSetAllWorkingDims(addVectors, dims
                                 int(innerDim,8),
                                 int((entries + i

call occaCopyPtrToMem(o_a, a(1), int(entries,8)*4
call occaCopyPtrToMem(o_b, b(1));

call occaKernelRun(addVectors, occaTypeMem_t(entr

call occaCopyMemToPtr(ab(1), o_ab);

print *,"a = ", a(:)
print *,"b = ", b(:)
print *,"ab = ", ab(:)

deallocate(a, b, ab, stat = alloc_err)
if (alloc_err /= 0) stop "*** deallocation not successful ***"
```

```
subroutine addVectors(entries, a, b, ab)
  implicit none

  integer*4 entries
  real*4 a(entries), b(entries), ab(entries)

  do i = 1, entries
    ab(i) = a(i) + b(i)
  end do
end

!$loopy begin transform
!
! addVectors = lp.split_iname(addVectors, "i", 16,
!     outer_tag="g.0", inner_tag="l.0")
!
!$loopy end transform
```

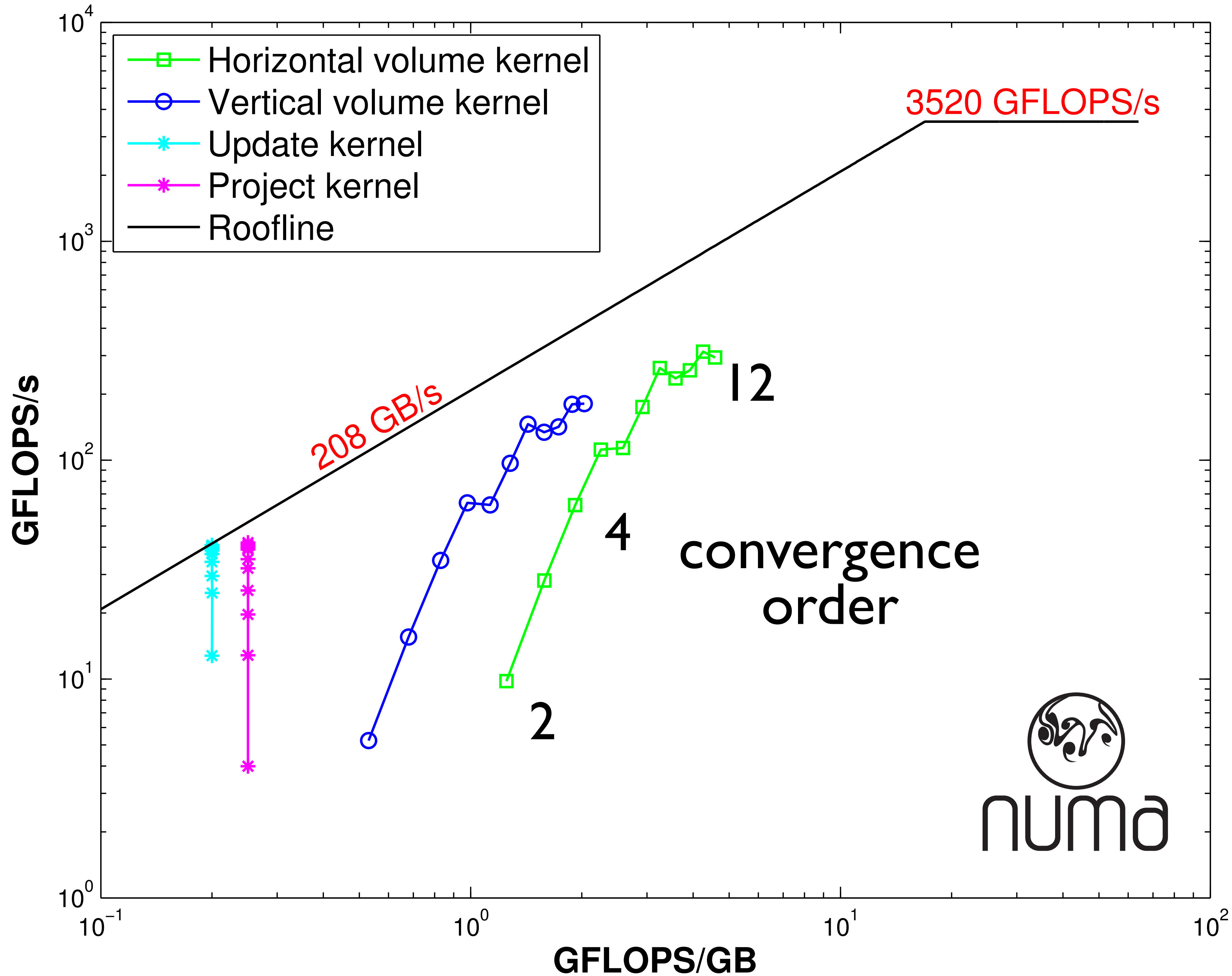
Weak scaling up to 16,384 GPUs (88% of Titan)

using 900 elements per GPU, polynomial order 8 (up to 10.7 billion grid points)



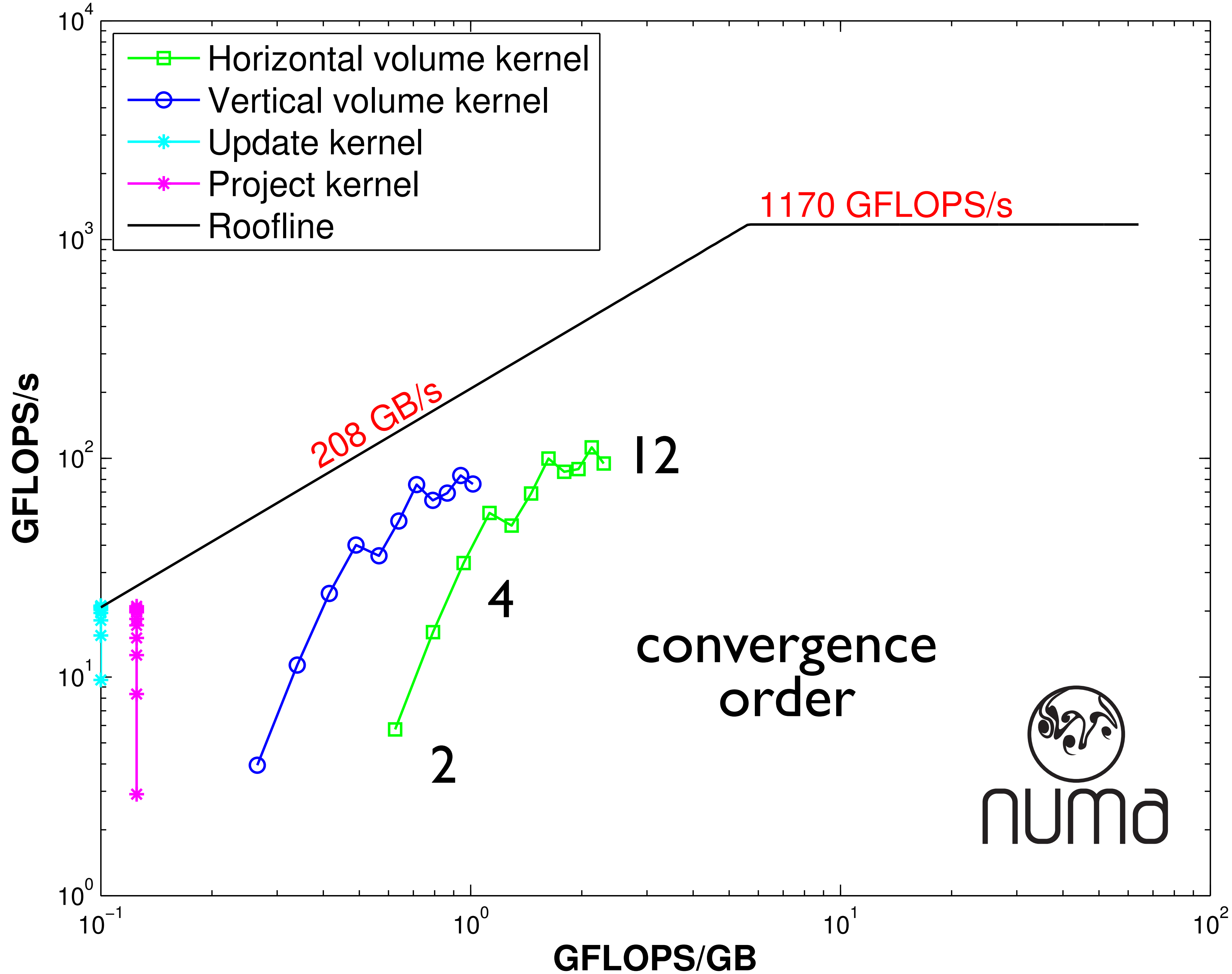
roofline plot: single precision

on the NVIDIA K20X GPU on Titan



roofline plot: double precision

on the NVIDIA K20X GPU on Titan



some more results from Titan

- GPU: up to 15 times faster than original version on one CPU node for single precision (CAM-SE and other models: less than 5x speed-up)

- CPU: single precision about 30% faster than double

- GPU: single about 2x faster than double

	CPU node	GPU
	16-core AMD Opteron 6274	NVIDIA K20X
peak performance	141 GFlops/s	1.31 TFlops/s
peak bandwidth	32 GB/s	208 GB/s
peak energy	115 W	235 W

Intel Knights Landing

Knights Landing

Why is this so exciting?

- 64 cores (Mira: 64 hardware threads)
- each Knights Landing chip has 16GB MCDRAM with 440GB/s memory bandwidth

$\Rightarrow$ more than 15 times faster than Mira

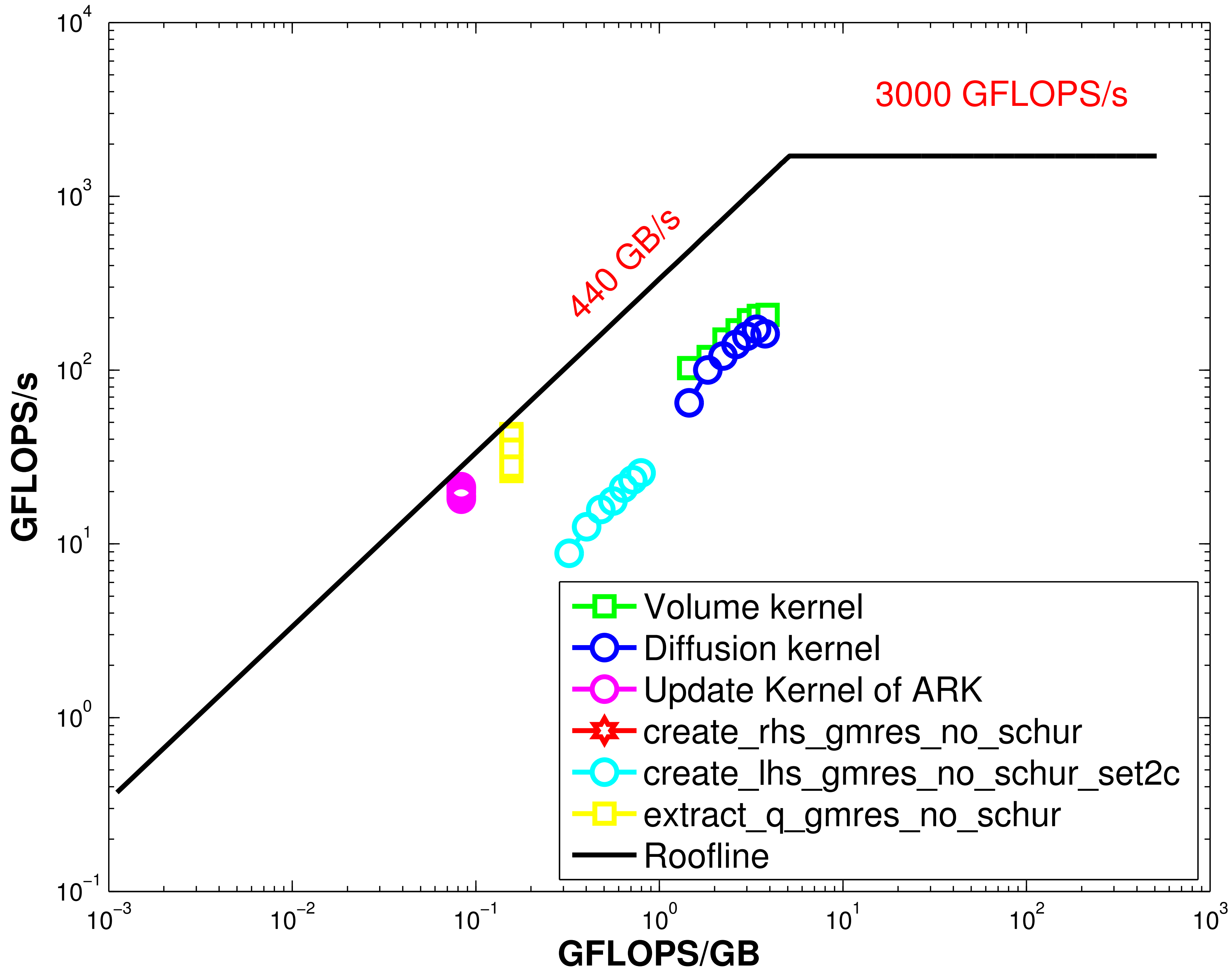
- Aurora (successor of Mira, expected in 2018): more than 50k Knights Landing nodes

$\Rightarrow$ more nodes than Mira

- overall: Aurora should allow us to run the same simulations like on Mira but 15 times faster

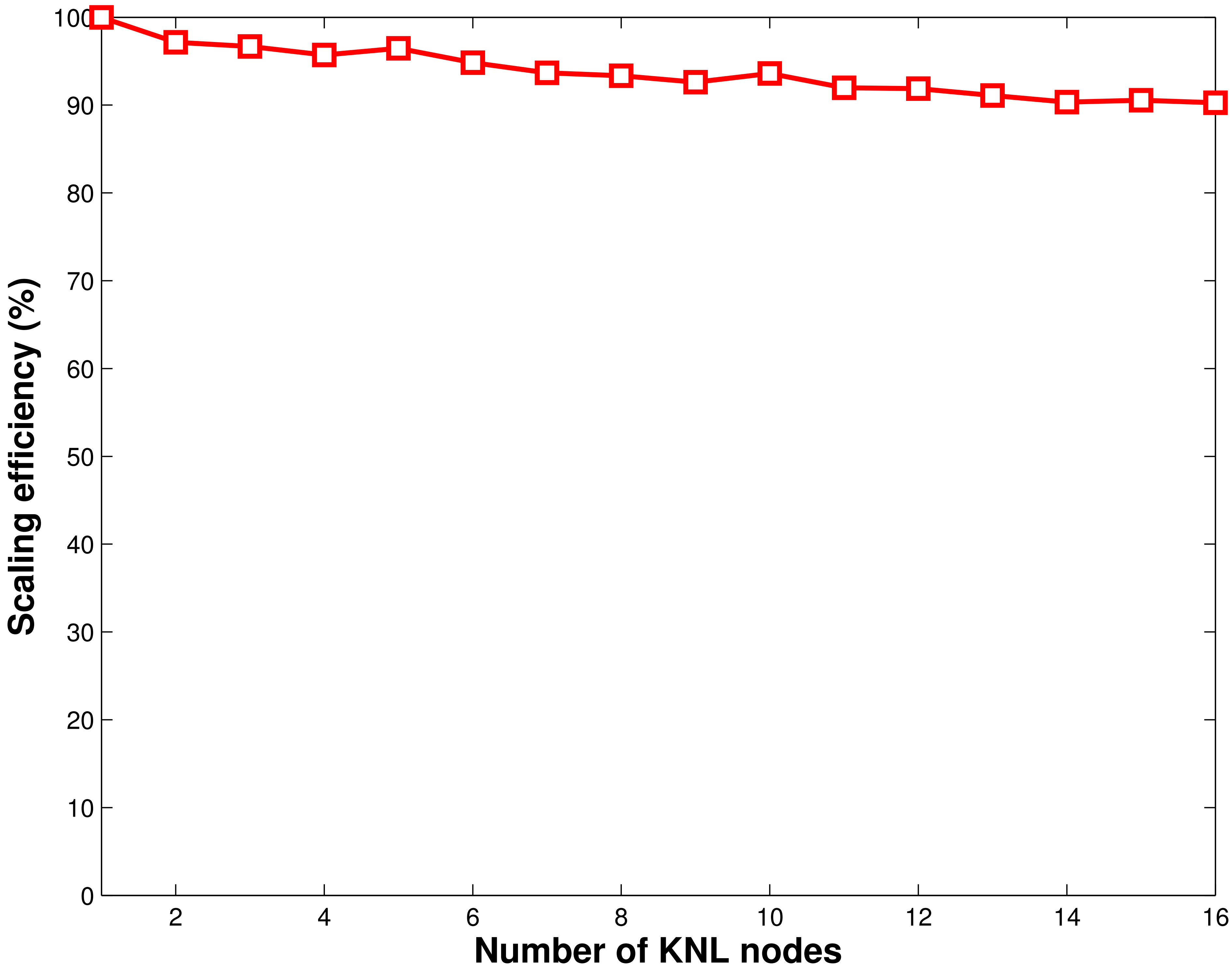
Knights Landing: roofline plot

OCCA version of NUMA



Knights Landing: weak scaling

work in progress



Summary

- Mira:
  - 3.0km baroclinic instability within 4.15 minutes runtime per one day forecast
  - 99.1% strong scaling efficiency on Mira, 1.21 PFlops
- Titan:
  - 90% weak scaling efficiency on 16,384 GPUs
  - GPU: runs up to 15x faster than on one CPU node
- Intel Knights Landing:
  - looks good but still room for improvement

Thank you for your attention!