Leibniz Supercomputing Centre
of the Bavarian Academy of Sciences and Humanities

Optimizing Astrophysical Simulations and Data Analysis codes on Intel Architectures

Salvatore Cielo, LRZ
Luigi Iapichino, LRZ
Fabio Baruffa, Intel

LRZ machines

▲ CoolMUC-3 – KNL

64 cores/node, 16GB MCDRAM
Used in quadrant/cache mode

SKX – SuperMUC-NG ►

48 cores/node, 2GB/core RAM

6336 thin, 144 fat (768 GB/node)

I. Simulation codes (CFD + astro)

Optimizing code for Astrophysical Simulations

The FLASH[1] code

(Adaptive) Mesh code

CFD / MHD + Physics (+ Astro)

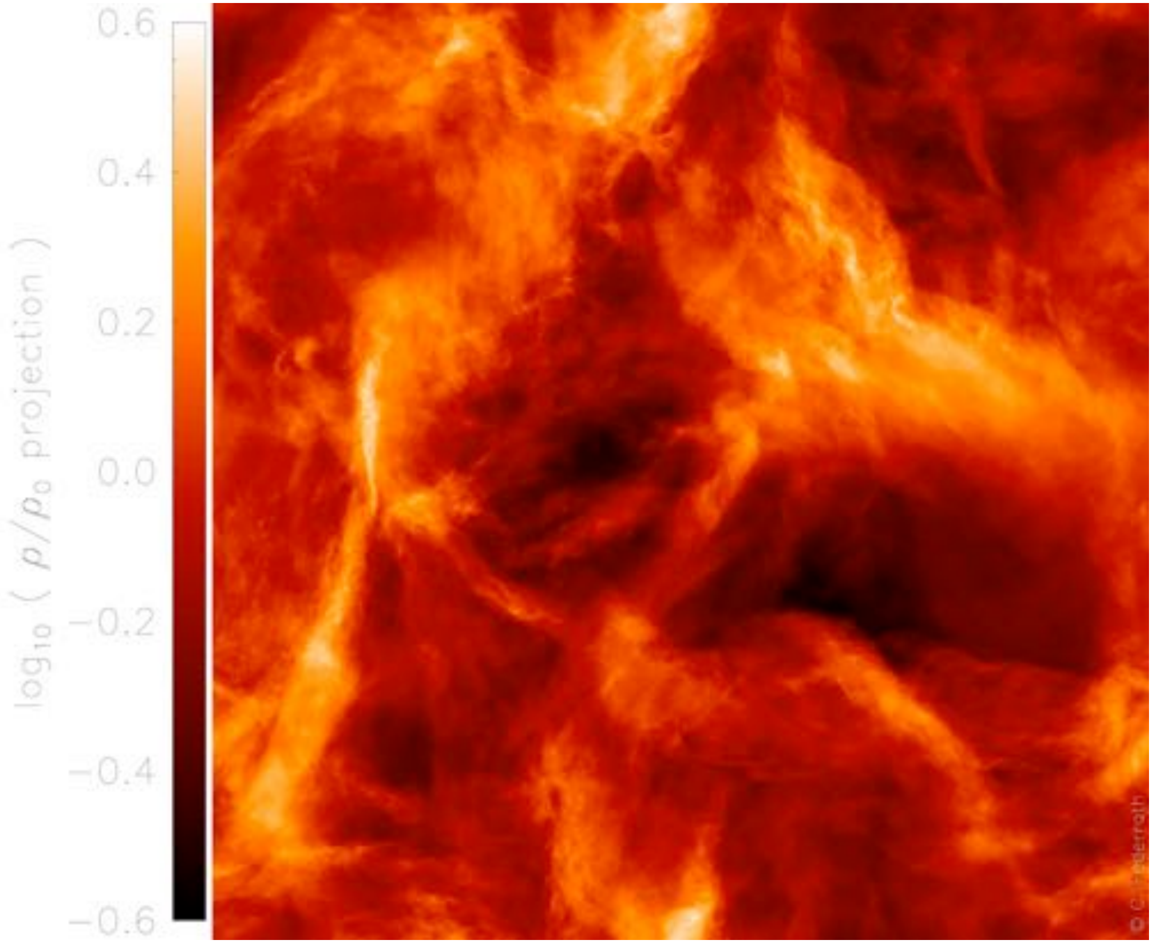
- nuclear + radiation (+ stars) +…
- here turbulence, shocks …

Optimizations by C. Federrath[2] with LRZ

Hybrid single/double-precision

Even in a difficult turbulent setup:

- improves MPI and memory usage
- further advantage with vectorization
- now extended to MHD

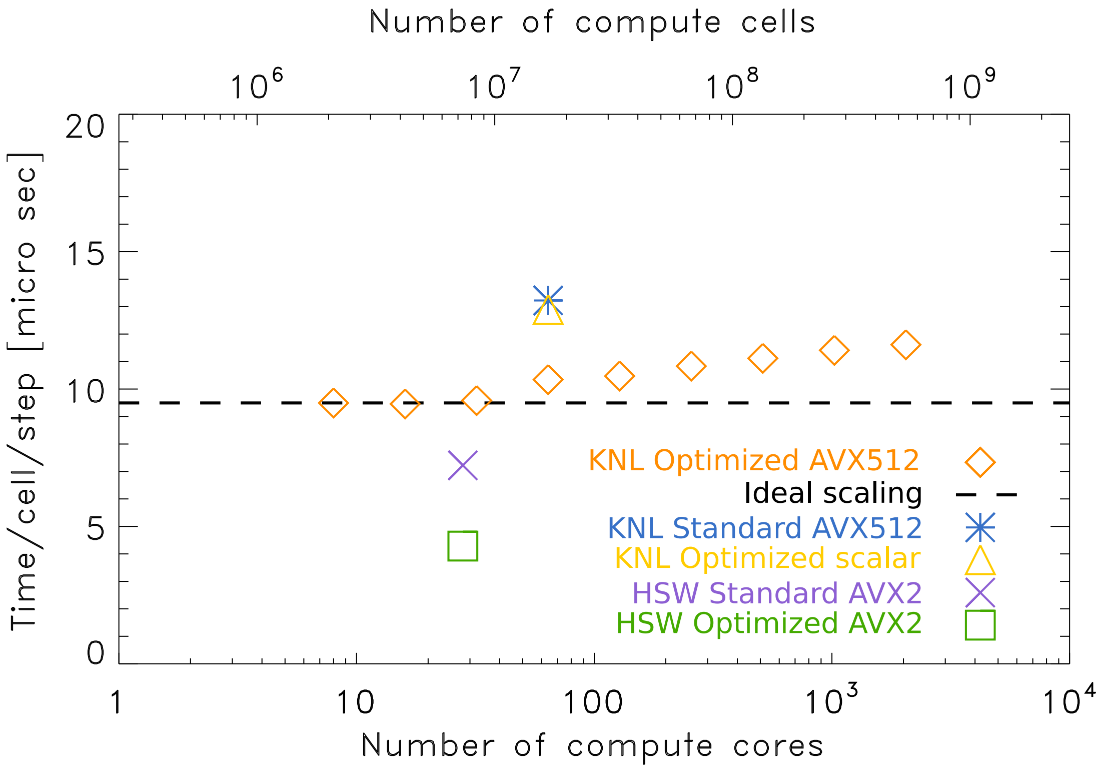


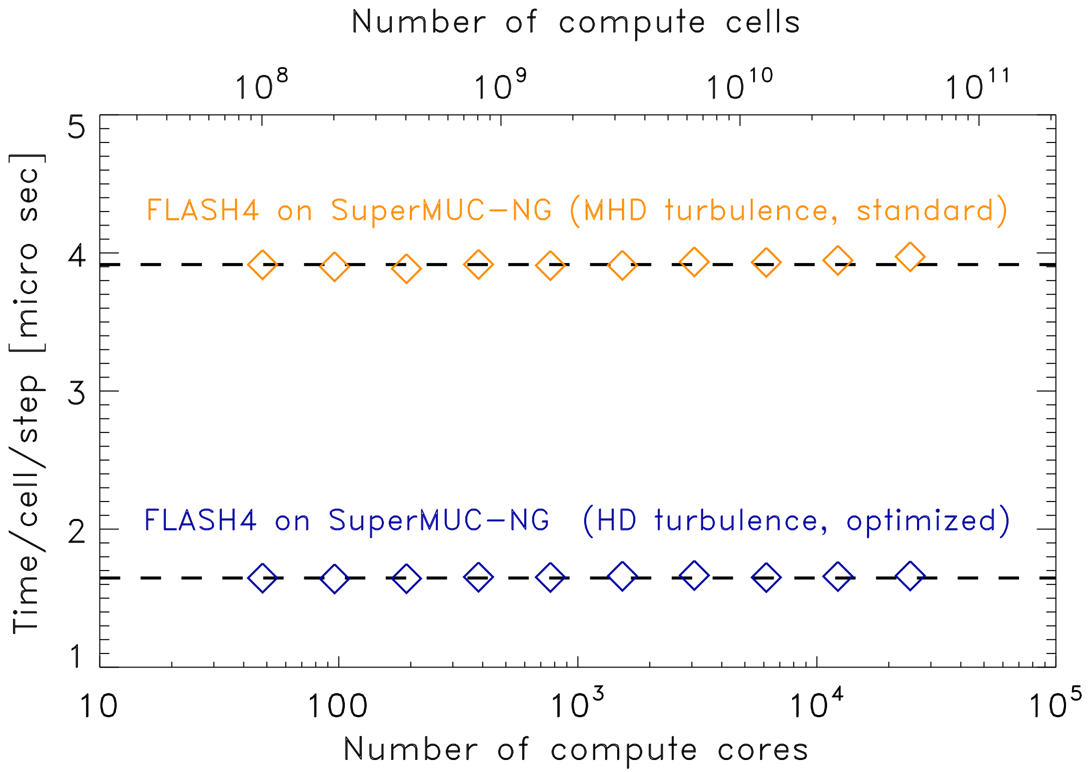
Credit: C. Federrath. Density and sonic-scale projections of a 10008^3 grids simulation, no MHD.

[1] Fryxell et al. 2000

[2] Federrath et al. 2016 „The world's largest turbulence simulations" + Federrath et al. 20XX – submitted to Nature Astronomy

The FLASH code: weak scaling tests





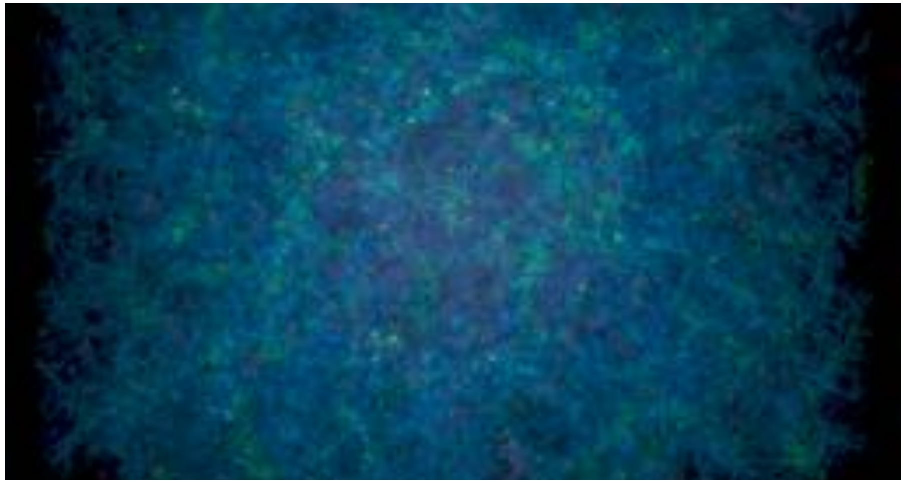
Optimization work on previous architectures shines on both KNL and SKX!

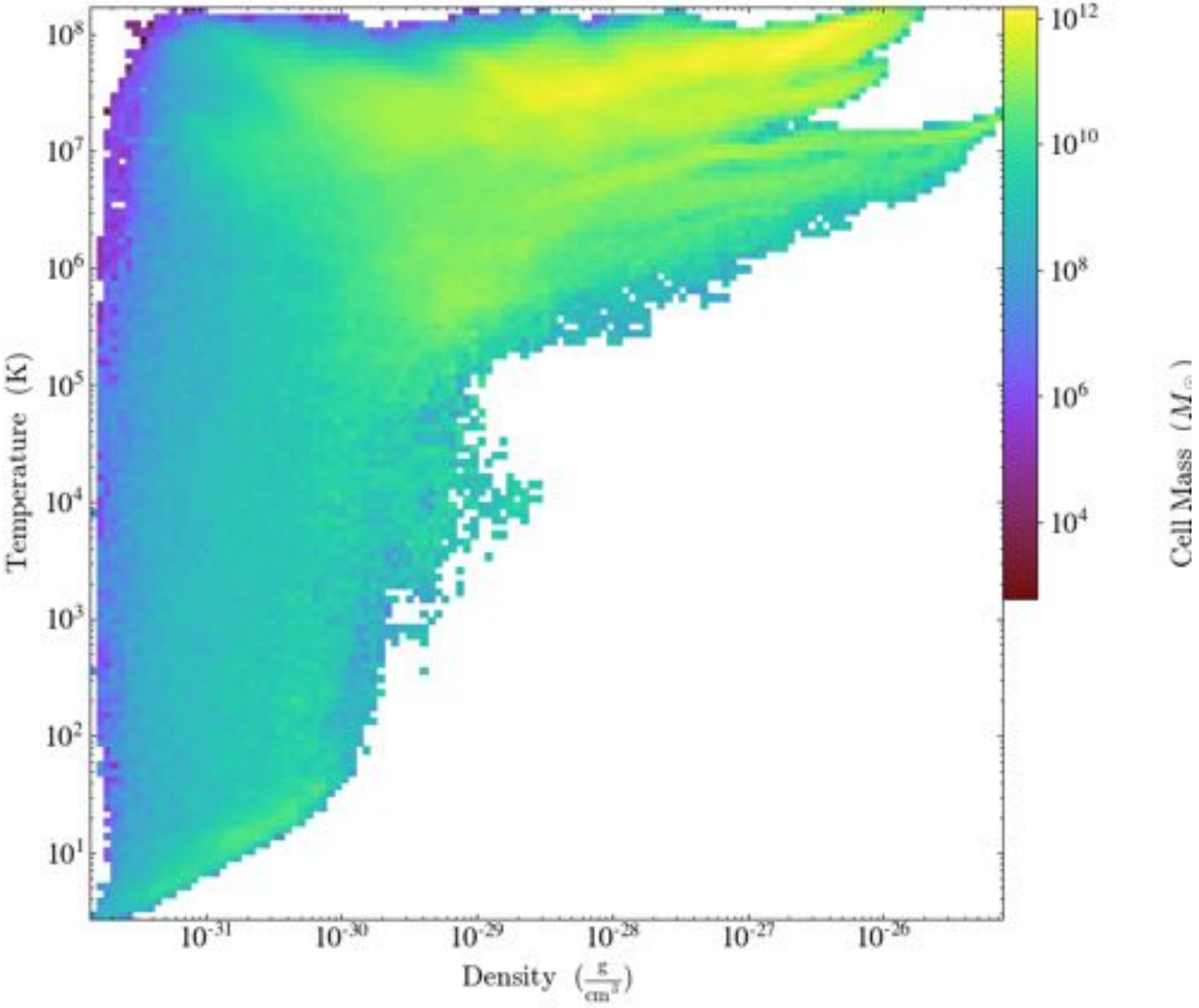
II. yt: Intel VS Anaconda python on SKX

yt: data analysis python package

submbitted to Intel PUM, Issue #38

- Python-based: numpy, scipy, mpi4py, …
- Integration with professional tools:
  RT, X-ray, mock observations, …
- Cross-code, general-purpose
- Sample tasks: 2D phase plot ►
  ▼ Volume rendering





Scripted, and parallel via mpi4py

yt: tasks with synthax

submitted to Intel PUM, Issue #38

Preamble

```
import yt, mpi4py
yt.enable_parallelism()
ds = yt.load("RD0028/RedshiftOutput0028") # Opening file header only
sp = ds.sphere("c", (10.,"Mpc"))          # Central 10 Megaparsec sphere
```

Derived

```
j  = sp.quantities.angular_momentum_vector(use_gas=True, use_particles=True)
```

Phase

```
pp = yt.PhasePlot(sp, "density", "temperature", ["cell_mass"], weight_field=None)
pp.save()
```

Volume

```
im, sc = yt.volume_render(ds, ('gas', 'density'), fname='volume.png')
```

X-ray: Preamble

```
import soxs, pyxsim
soxs.soxs_cfg.set("soxs", "response_path", "./soxs_responses" )
redshift  = 0.05
src_model = pyxsim.ThermalSourceModel("apec", 0.05, 11.0, 10000, Zmet=0.3)
exp_time  = (100., "ks")
area      = (2000.0, "cm**2")
sp        = ds.sphere("c", (50.,"Mpc")) # Enlarge to 50 Megaparsec
```

X-ray: Photons

```
# Computation 1/2: Montecarlo radiative transfer
photons  = pyxsim.PhotonList.from_data_source(sp, redshift, area, exp_time, src_model)
```

X-ray: Events

```
# Computation 2/2: Photons produce events into simulated detector
events_z = photons.project_photons("z", (45.,30.), absorb_model="tbabs", nH=0.04)
```

X-ray: Printout

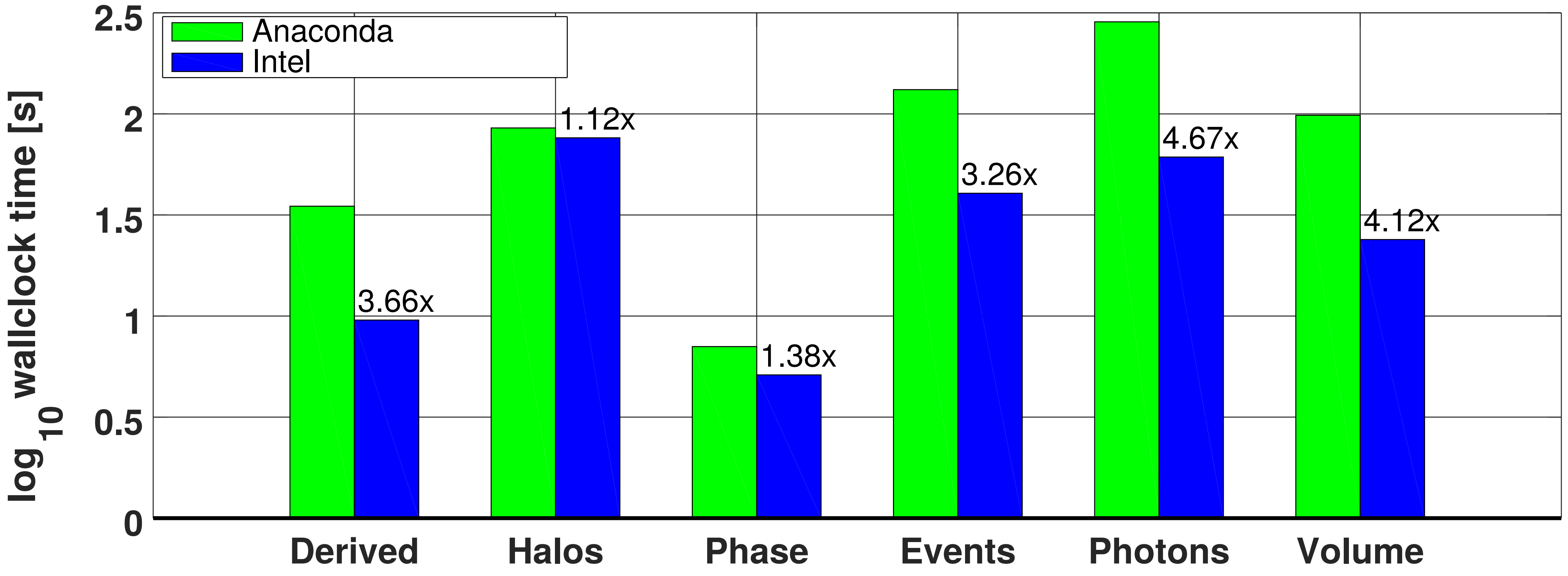
```
events_z.write_simput_file("RD0028", overwrite=True) # Warning: very large fits file!
soxs.instrument_simulator("RD0028_simput.fits", "evt.fits", (100.0, "ks"), "acisi_cy0", [45.,30.],overwrite=True)
soxs.write_image("evt.fits", "img.fits", emin=0.5, emax=11.0, overwrite=True)
exit()
```

Intel vs Anaconda python

yt: performance

(lower is better)

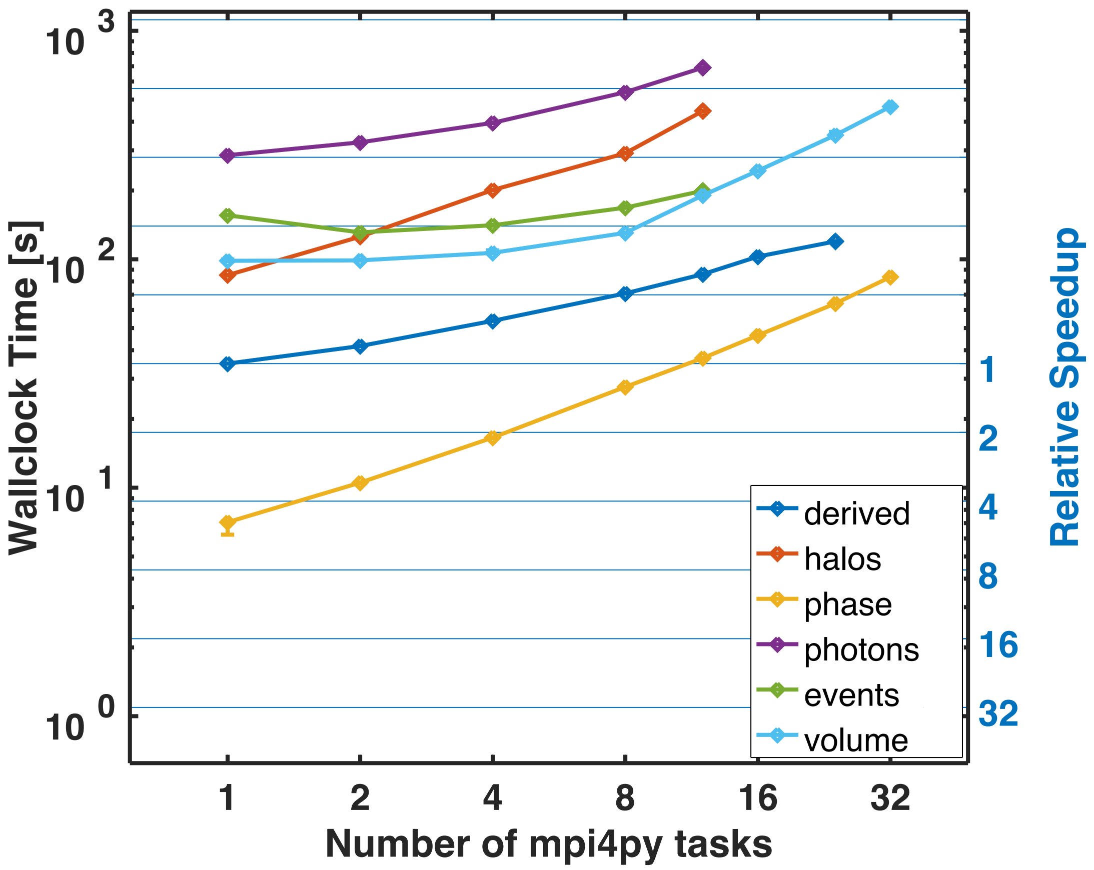
submitted to Intel PUM, Issue #38

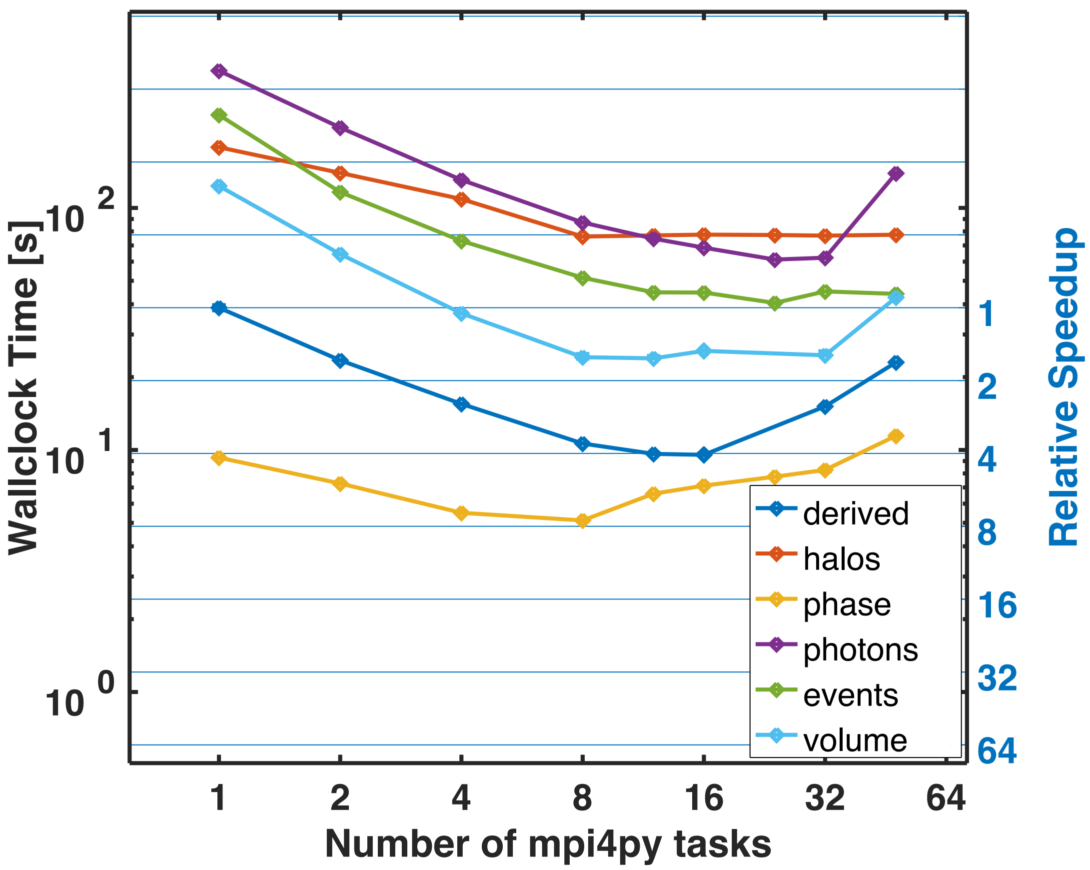


Intel vs Anaconda python

yt: detailed scaling

submitted to Intel PUM, Issue #38





III. Deeper into yt: parallelism beyond mpi4py on KNL

yt and HPC: beyond mpi4py

OpenMP Enabling HPC since 1997

- native support through mpi4py, just set OMP_NUM_THREADS
- pure or hybrid

Cython

- optimising static compiler for both Python and itself.
- Get efficient C code from python…
- ... but one must go that extra mile to rewrite and compile your code!
- No tutorials from yt! Users must learn from Cython tutorials and apply to yt functions.

yt scripted synthax

```
$ mpiexec -np 8 python volume.py
```

volume.py

```
import yt, mpi4py
yt.enable_parallelism()
ds = yt.load("RD0028/RedshiftOutput0028")
im, sc = yt.volume_render(ds,  ('gas', 'density'), fname='volume.png')
exit()
```

Cython compiled synthax

```
$ python setup.py build_ext --inplace
$ mpiexec -np 8 python launch.py
```

setup.py

```
from setuptools import setup
from Cython.Build import cythonize
setup(name = 'Volume_app', ext_modules = cythonize("*.pyx"))
```

launch.py

```
import myvolume, yt
ds = yt.load('RD0028/RedshiftOutput0028')
myvolume.do_volume(ds)
exit()
```

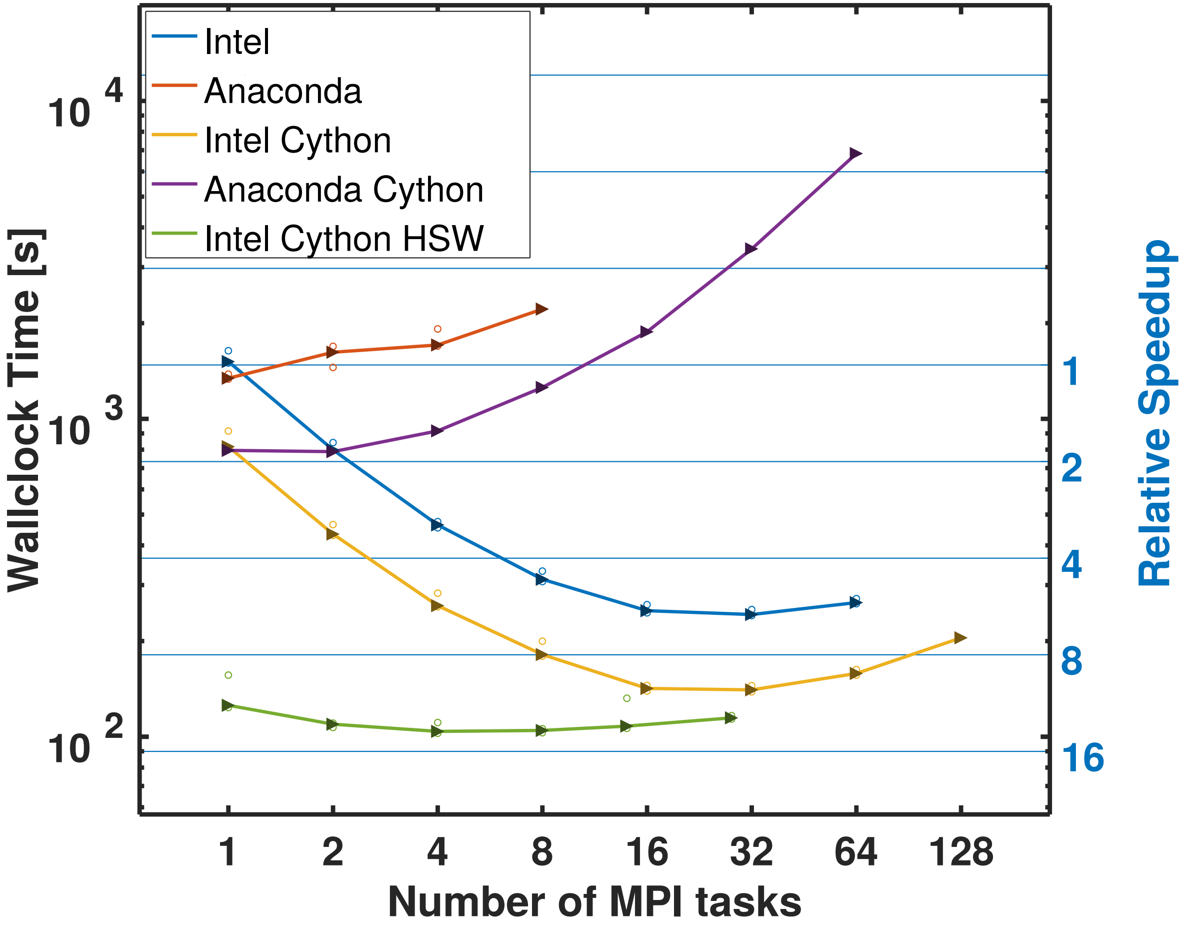
myvolume.pyx

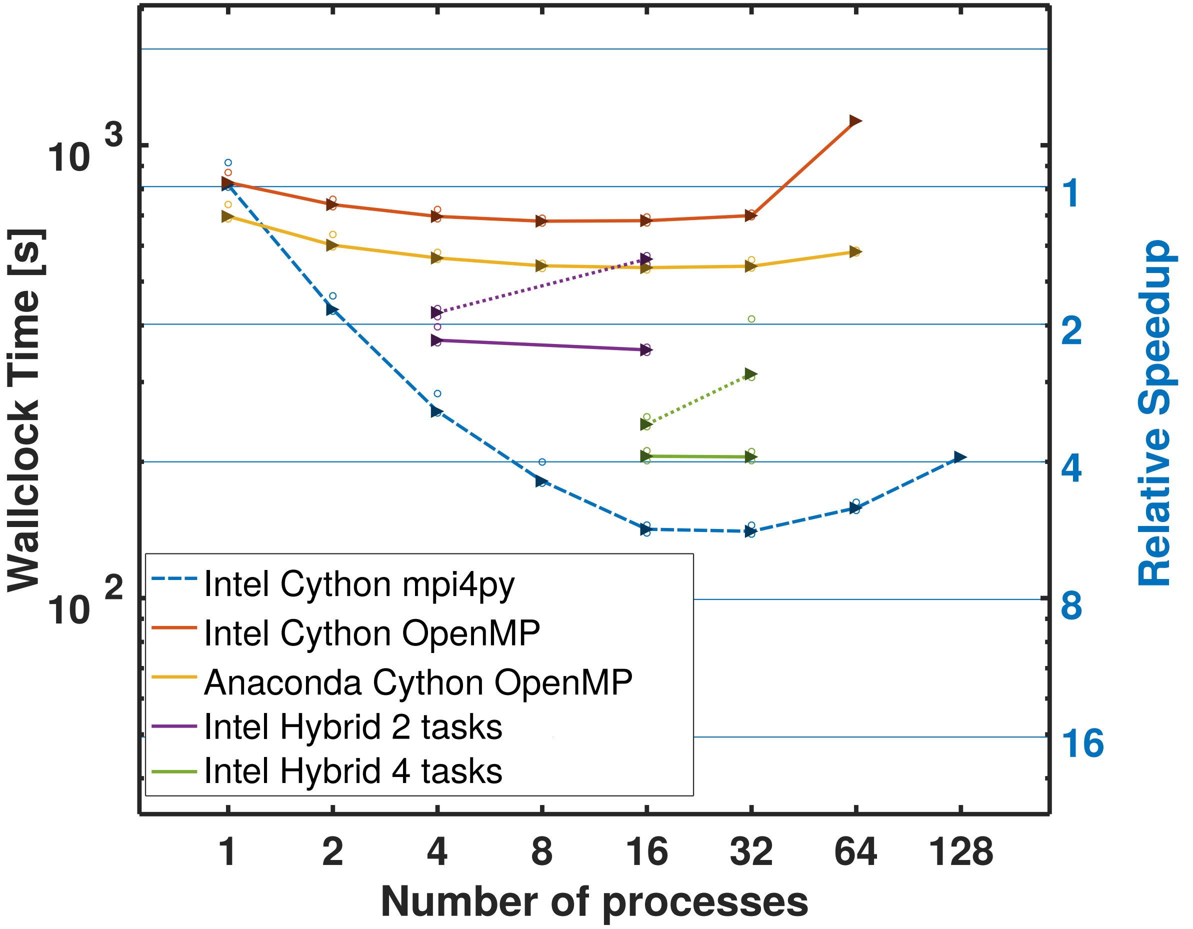
```
def do_volume(ds):
    import yt
    im, sc = yt.volume_render(ds, 'density', fname='rendering.png')
```

Intel → scalability out-of-the-box Cyhton→ work harder for performance OpenMP → ???

Scaling behaviour with Cython and OpenMP

submitted to FGCS, in review





Also: automatic process pinning best option (dotted lines). Hard to get anything from hyperthreading.

Summary

Simulation Codes

- Code optimization on previous Intel architecture shines on KNL and SKX
- Treasure the lesson of the KNL!

yt and data analysis

- post-processing with HPC techniques is possible (and fun!)
- using Intel python is a must, and works out-of-the-box
- further optimization requires changing workflow

For the near future…

- Involve yt developers in Intel python discussion
- Investigate where and why yt scaling breaks
  - Characterization with Intel Advisor