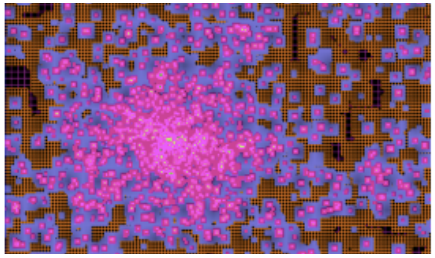

# Xeon-Phi days @ StarPlan

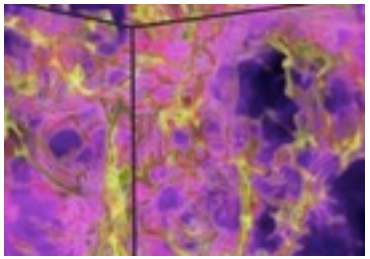



# Computational Astrophysics @ NBI on Xeon-Phi

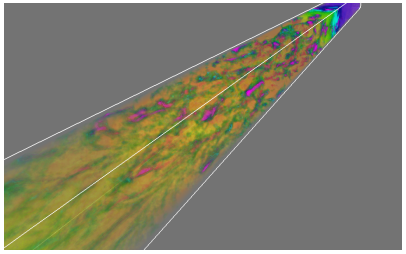


RAMSES

PENCIL-CODE



NIRVANA



PP-CODE

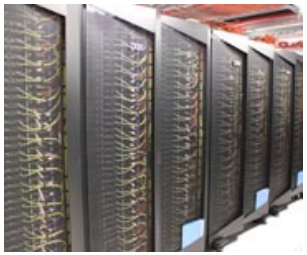
Troels Haugbølle – haugboel@nbi.ku.dk

# A brief history of *astro-HPC* in CPH

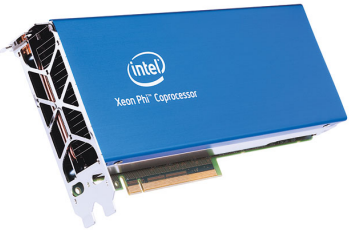
INCREASING COMPLEXITY

**Shared memory era**
2002: DCSC established – SGI Origin in CPH
2004: SGI Altix with 64 CPUs

**Infiniband clusters**
2005: Steno is born: Opteron + infiniband
2007: Expansion of infiniband cluster (astro)
2008: First Nehalem based cluster (astro2)



**Arrival of accelerators**
2009: First GPU cluster. 20 nodes, C1060 Tesla (astro_gpu)
2010: Expansion with 30 Fermi C2050 nodes (astro_gpu2)
2012: More memory to GPUs – 72 GB per node

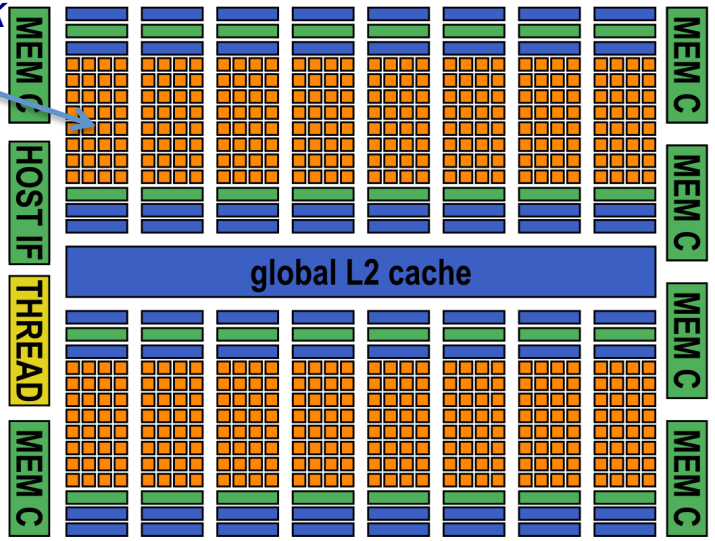



**Our installation is reborn**
2014: Ivy nodes, Xeon-Phi, Analysis frontends
1 PB storage, 3000+ cores

# What makes an accelerator fast ?

*GPU's have many cores optimized for simple parallel work*

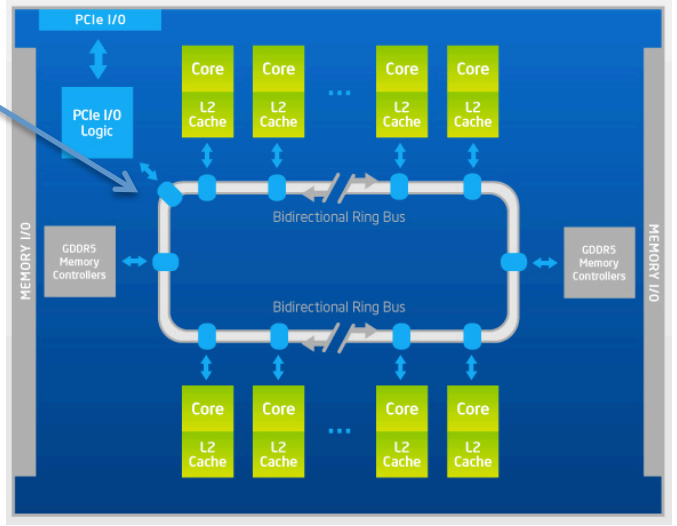
**1000s of slim cores**




- Execute simple parallel code
  - GPU's do the same operation on many data
  - Nvidia use groups of 32 cores all doing the same
- High throughput of data
  - Very high memory bandwidth, but little cache
  - Use multi-threading instead: Oversubscribe the GPU and only work on data when it has arrived

*Xeon Phi is designed like a GPU, but more versatile*

**60 simple cores**




- Execute code in-order using simple cores
  - Get performance from 512-bit vectors units
  - Memory is cache coherent. Looks like a x86 CPU
- Always keep the PHI busy by over-subscription
  - Use 4-way hyper-threading to try to do useful work

# Why shift from GPUs to Xeon-Phi?

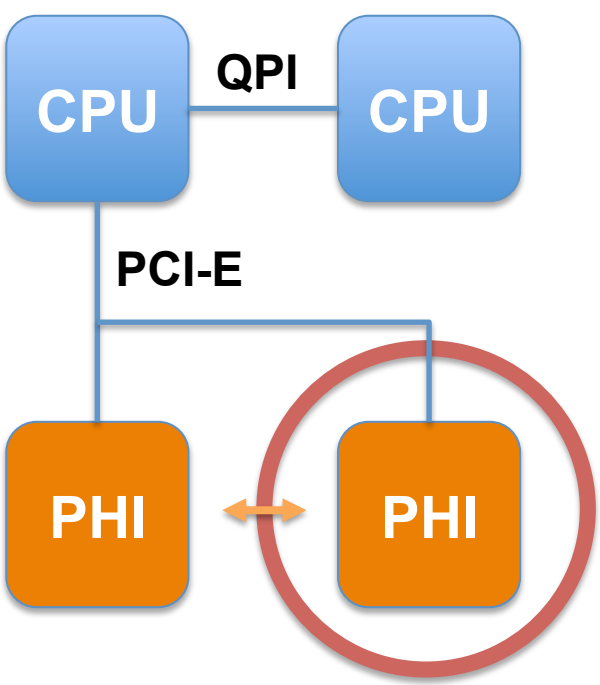
- GPUs can deliver very high performance, but code has to be rewritten specifically to exploit the architecture.
- In practice people too busy doing science to care. Only local code that ever made it to the GPUs: PP-code
- After much effort, PP-code speedup is still only 5x compared to CPUs (8 cores to 8 cores + 4 GPUs)
- Xeon-Phi promises to execute any fairly well behaved code after a simple recompile
- With 240 threads but only 8 GB memory pure MPI is almost impossible. MPI + OpenMP works fine.

# First Benchmarks

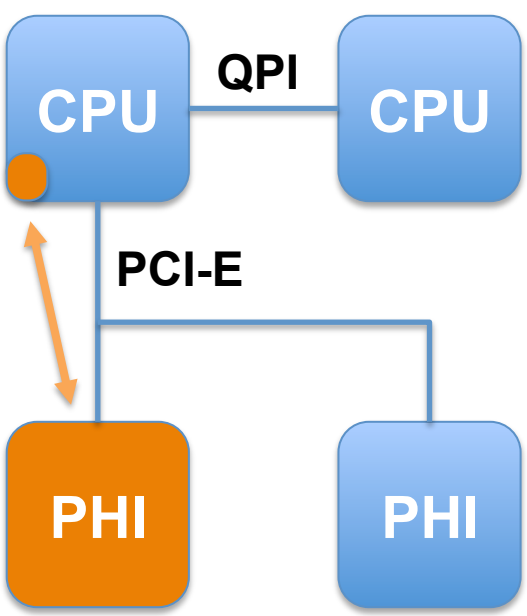
# Can Xeon-Phi deliver?

- Benchmark a number of codes in active use by the group
  - **PP-code:** Particle-in-cell code for plasmas
  - **RAMSES:** Finite volume MHD on oct-tree AMR
  - **Pencil-Code:** Finite difference MHD on unigrid
  - **Nirvana:** Finite volume MHD with block AMR

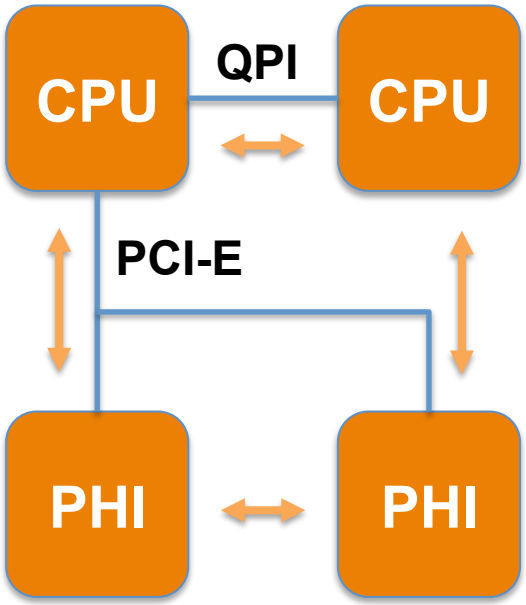
# Remember: Programming models




native




offload

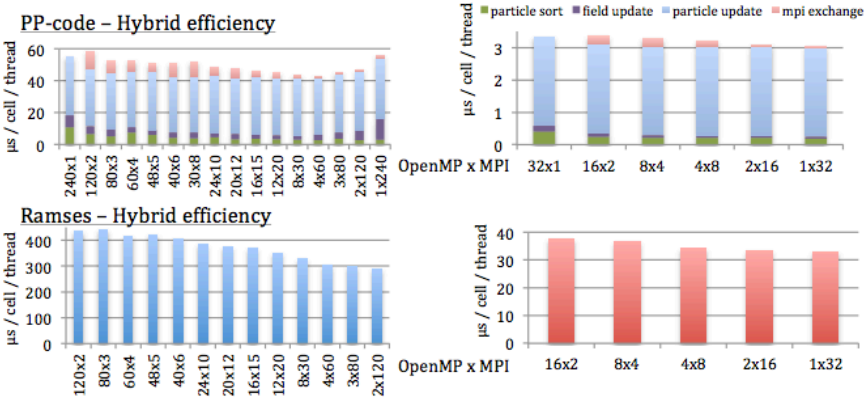


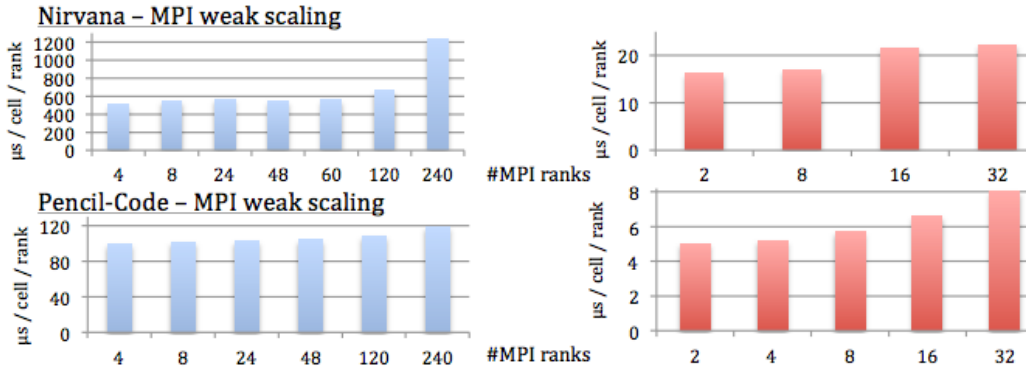

symmetric

# Can Xeon-Phi deliver?

- Benchmark a number of codes in active use by the group
  - **PP-code:** Particle-in-cell code for plasmas
  - **RAMSES:** Finite volume MHD on oct-tree AMR
  - **Pencil-Code:** Finite difference MHD on unigrid
  - **Nirvana:** Finite volume MHD with block AMR
- Compile and execute codes natively on a Xeon-Phi
- Done in late 2013 on test equipment from Dell

# Can Xeon-Phi deliver?

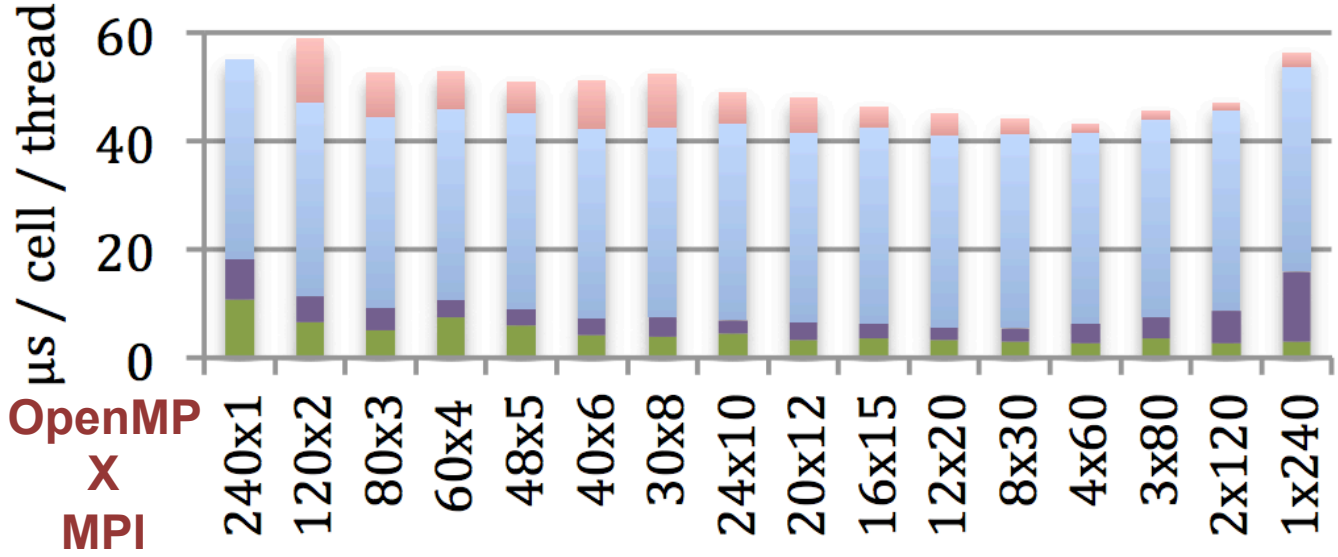


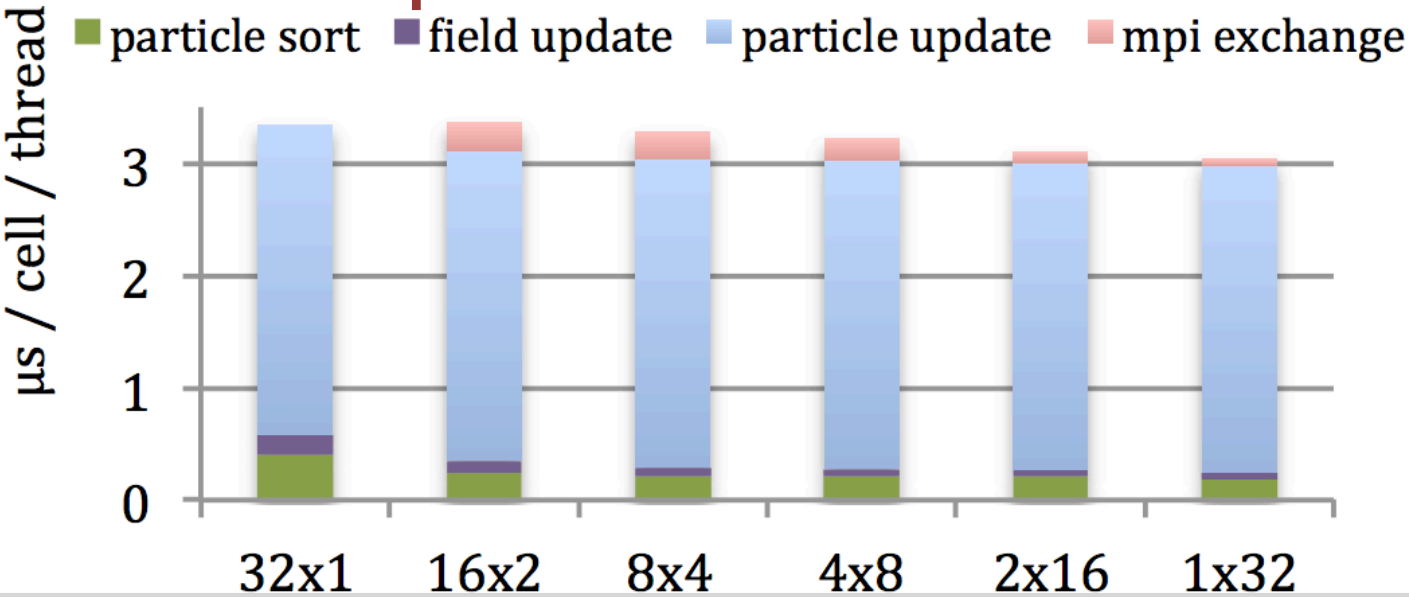

Single card results for the four codes, without any source code changes. To the left (right) is shown results on a 5120D Xeon-Phi card (Dual 8C Xeon E5-2650 Host). The two first codes are hybrid, and the scaling is using different full card / host configurations with 240 / 32 threads, while for the two other codes MPI-only weak scaling results are shown. The workloads are in all cases scaled to be the same for a single Xeon-Phi card (240 threads) and a single CPU socket (16 threads). For all codes the total raw performance, measured as the time it takes to do a cell update in the model, is comparable between a single 8-core CPU socket and a Xeon-Phi card.

[From IPCC application]

## Xeon-Phi performance




## Host performance – SB 2 GHz




# PP-code

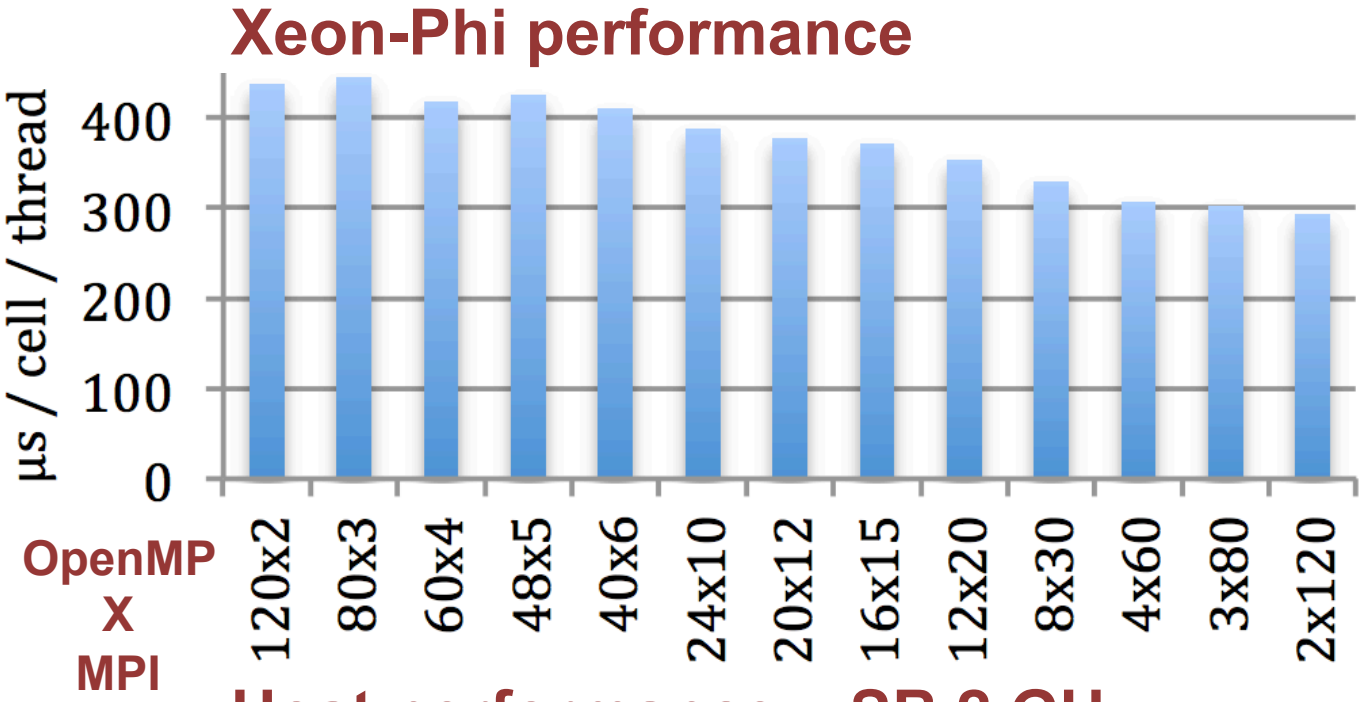
+ Already hybrid OpenMP + MPI

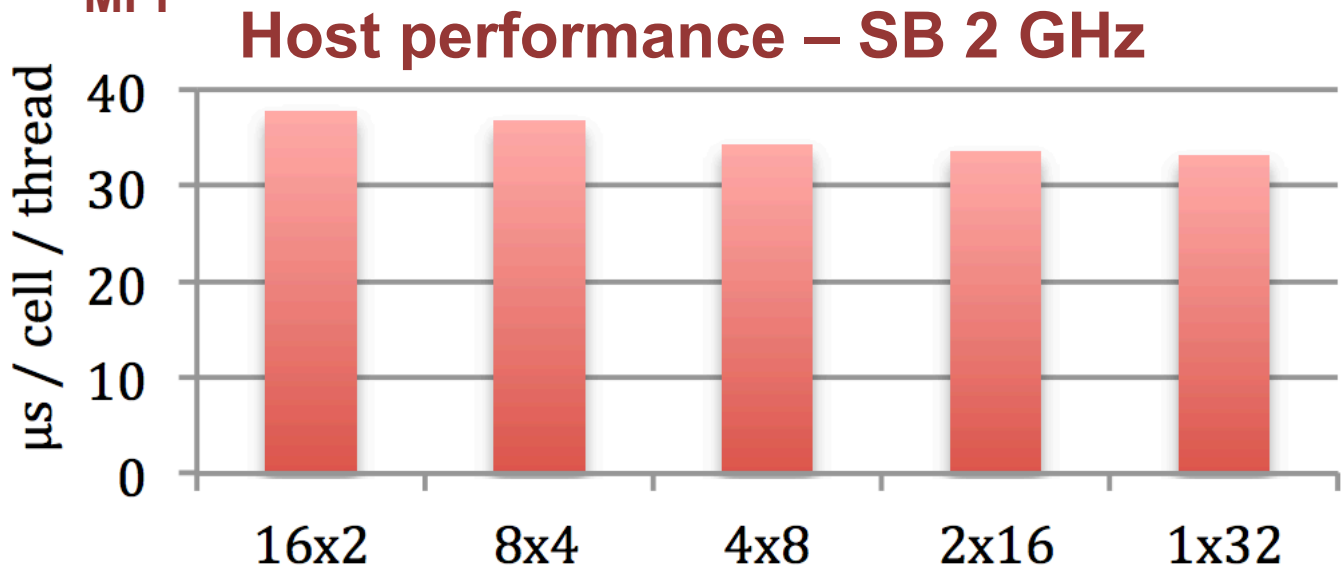
- Vectorization of key kernels

- More OpenMP

- Consider Offload

# RAMSES




+ Already hybrid OpenMP + MPI

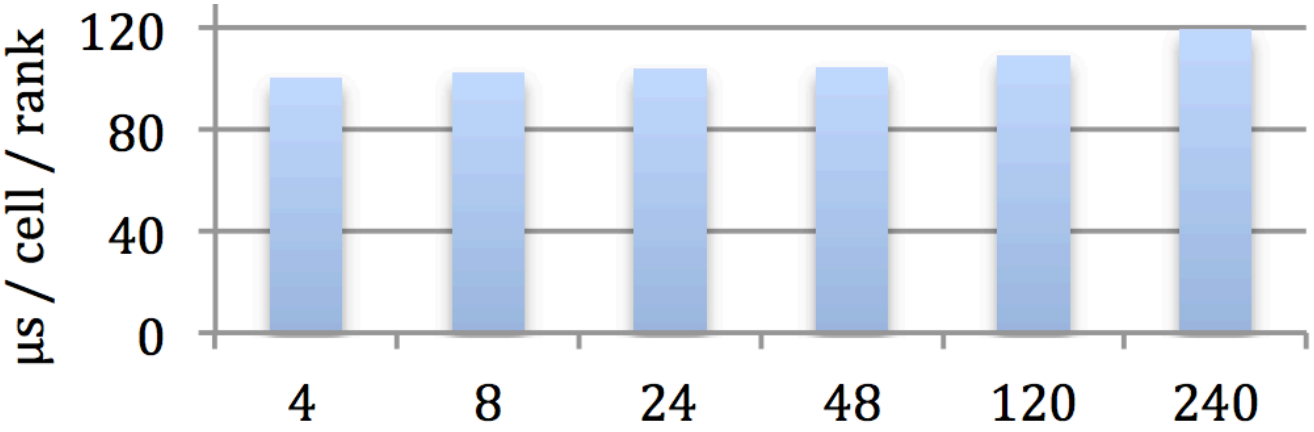
- Loadbalancing tricky with adaptive mesh refinement

- Vectorization not pervasive

- Reorder data for linear memory access

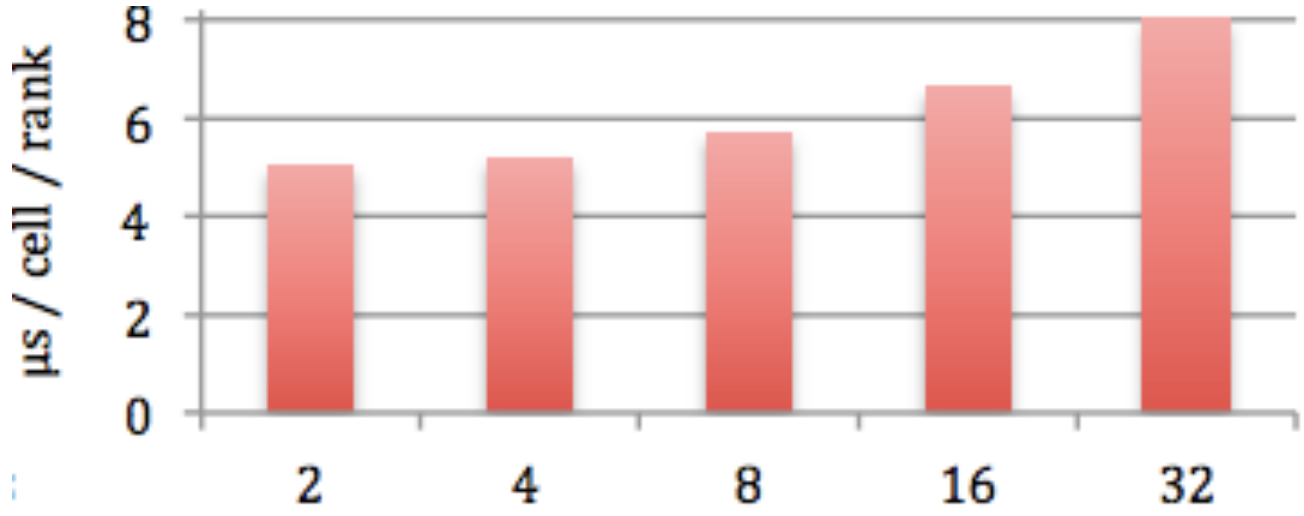
# Pencil-Code

## Xeon-Phi performance




#MPI ranks – weak scaling

## Host performance – SB 2 GHz




+ well vectorized(?)

+ OK with small domain per rank

+ unigrid: balanced

+ low memory bw

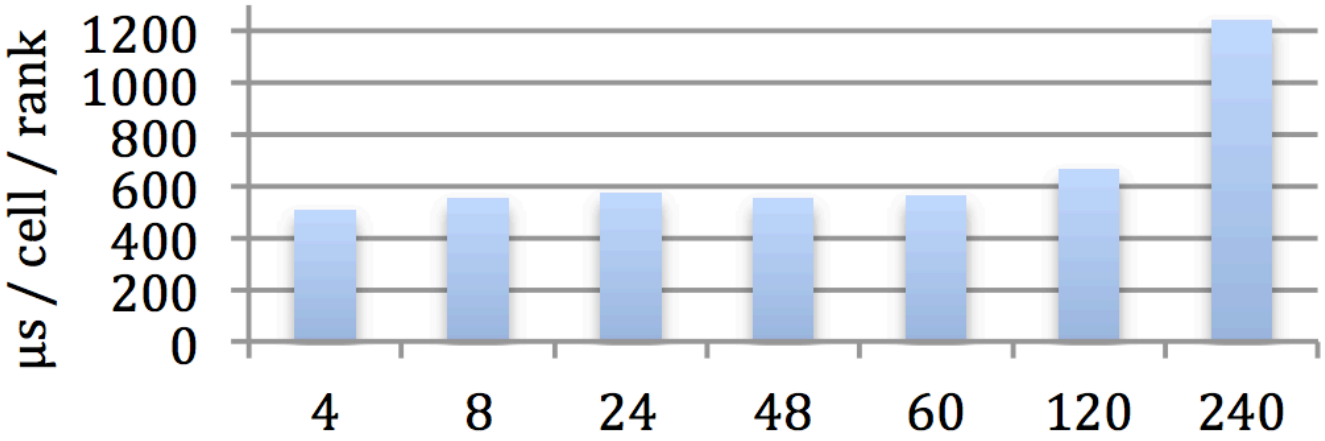
- pure MPI only

- X files per MPI rank; too many with Xeon-Phi

Troels Haugbølle – haugboel@nbi.ku.dk

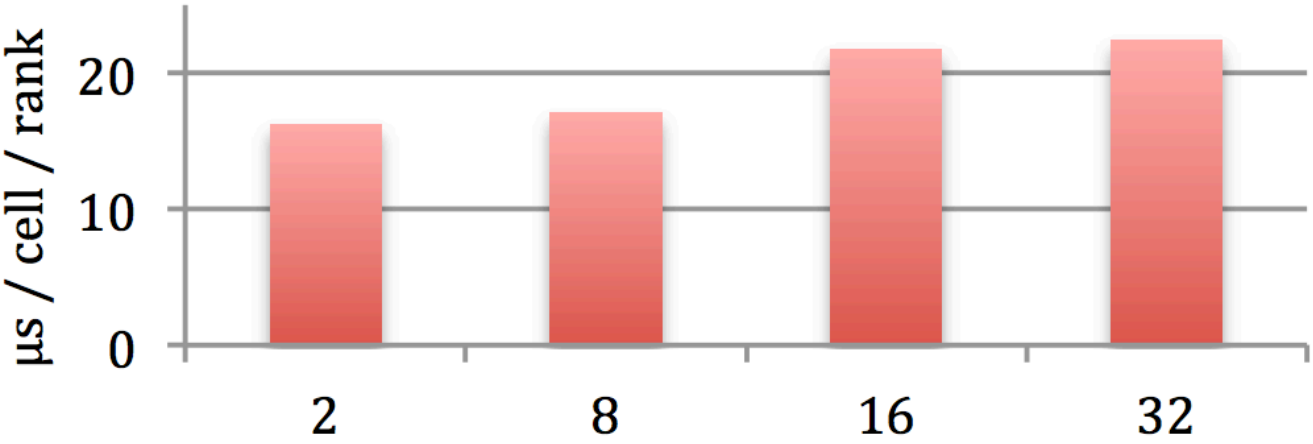
# Nirvana

## Xeon-Phi performance




#MPI ranks – weak scaling

## Host performance – SB 2 GHz




+ Block AMR easier to vectorise

- Pure MPI only

- Vectorization of kernels

- Memory bandwidth starved?

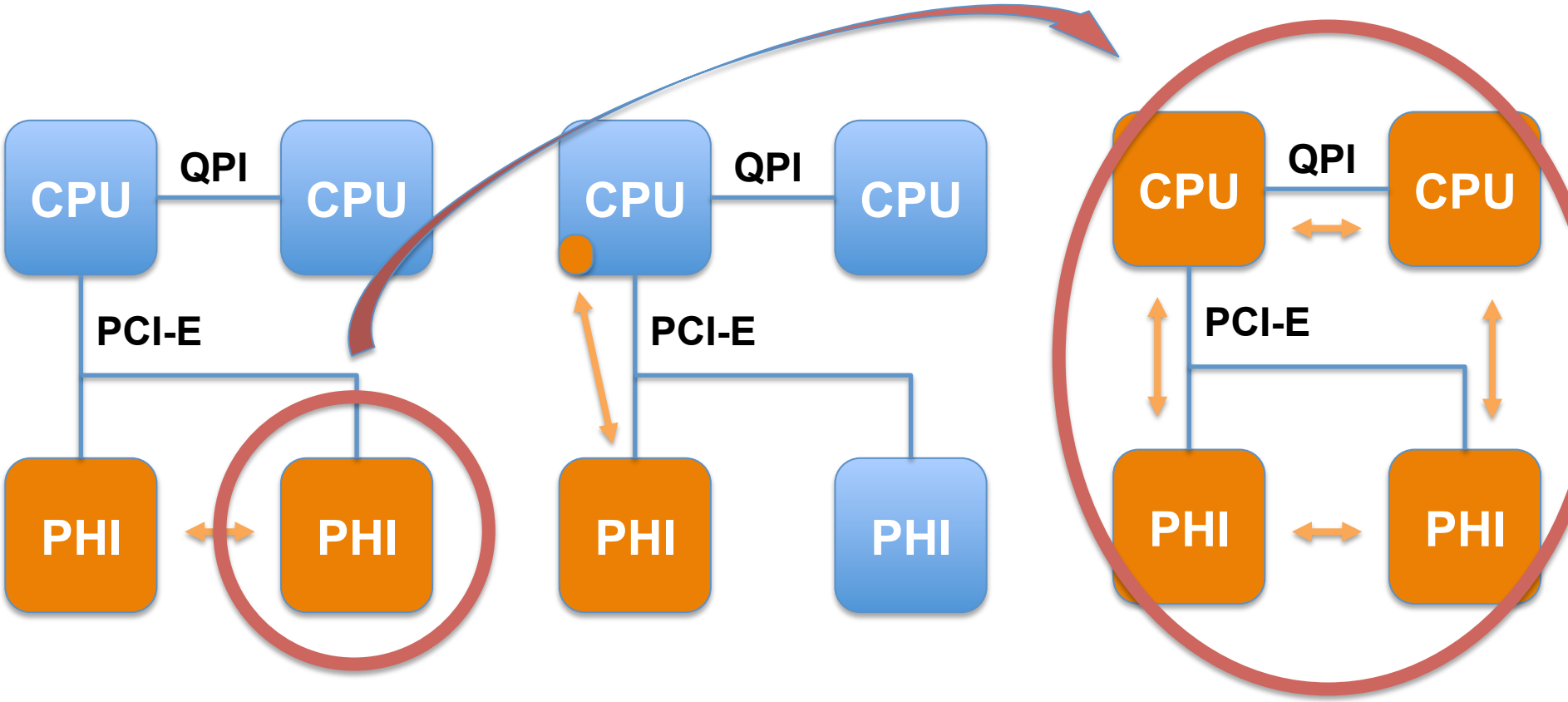
# Can Xeon-Phi deliver?

- Different codes have different bottlenecks and problems. They all perform equally on 2xPhi's and on 16 SB cores
- Work needed for PP-code and RAMSES to get them to perform well on Xeon-Phi, in particular vectorisation
- Significant work needed in the case of Pencil-Code and Nirvana: they have to become OpenMP+MPI to scale
- All codes are >>20.000 lines with lots of kernels and physics. They have to run natively. Only exception is PP-code, where off-load could be based on GPU code
- ***Profiling tools (vtune) and timers key to pin-point hot-spots***
- ***Correctness tools (Intel Inspector) essential for OpenMP***
- ***OMP SIMD directives needed to implement vectorisation***

# Efficient use of resources in symmetric mode
# &
# OpenMP + MPI considerations

Troels Haugbølle – haugboel@nbi.ku.dk

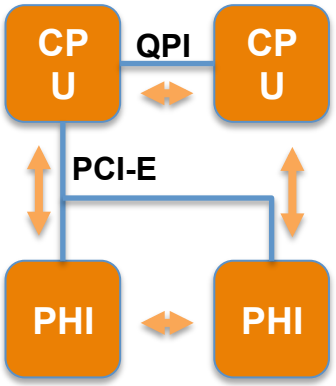
# Remember: Programming models




native offload symmetric

# Beyond pure MPI

- Xeon-Phi cards have 60 cores x 4 HT = 240 threads
- CPUs have (in our system) 10 cores x 2 HT = 20 threads
- How to load balance code across system?




(1) Make load balancing aware that different nodes have different speed

(2) Make each MPI rank roughly the same speed

# A solution: OpenMP + MPI

- Xeon-Phi cards have 60 cores x 4 HT = 240 threads

- CPUs have (in our system) 10 cores x 2 HT = 20 threads

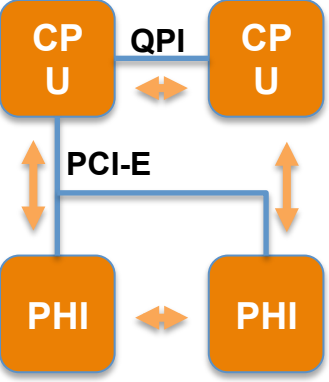
- If a code has a robust two-layer parallelization, it can be future proofed:
    - The number of cores per node (and per socket) is increasing, but network speed is not going up proportionally. Example
        - First Steno cluster had dual single-core opteron, and 10 gbit/s infiniband
        - Now 20 much faster cores, but only 56 gbit/s IB

    - Many cores per node demand a “shared memory layer”

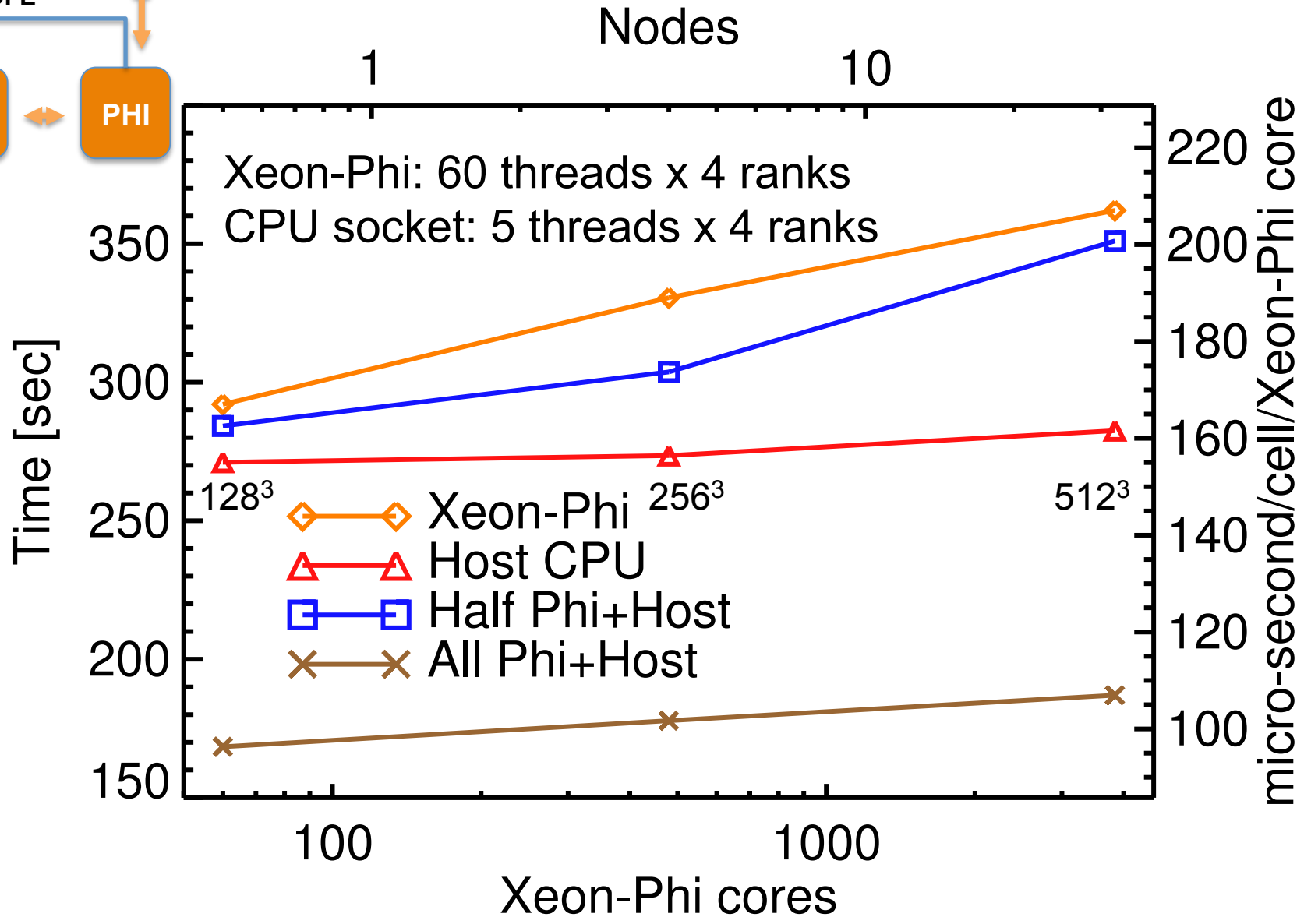
| Xeon-Phi 2x2x2x2x3x5=240 threads | | Ivy-Bridge Host 2x2x2x5=40 threads | |
|---|---|---|---|
| MPI ranks | OpenMP threads | MPI ranks | OpenMP threads |
| 1 | 240 | 1 | 40 |
| 2 | 120 | 2 | 20 |
| 3 | 80 | 4 | 10 |
| 4 | 60 | 5 | 8 |
| 5 | 48 | 8 | 5 |
| 6 | 40 | 10 | 4 |
| 8 | 30 | 20 | 2 |
| 10 | 24 | 40 | 1 |
| 12 | 20 | | |
| 15 | 16 | | |
| 16 | 15 | | |
| 20 | 12 | | |
| 24 | 10 | | |
| 30 | 8 | | |
| 40 | 6 | | |
| 48 | 5 | | |
| 60 | 4 | | |
| 80 | 3 | | |
| 120 | 2 | | |
| 240 | 1 | | |

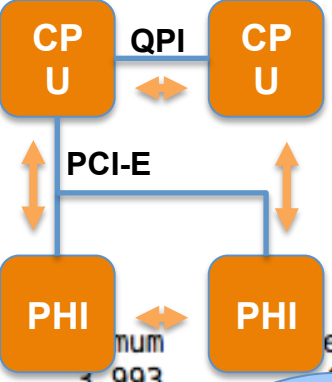
# OpenMP + MPI

- Different number of threads per MPI rank on Xeon-Phi and on host CPUs
- Even performance per MPI rank
- Xeon-Phi has many choices
- Ex 1 card=1 CPU:
  60 Phi threads=
  5 CPU threads




# RAMSES WEAK SCALING






# RAMSES - DETAILS

***Host executing benchmark on 32 nodes / 640 cores / 1280 threads***

| minimum | average | maximum | std dev | std/av | % | rmn | rmx | TIMER |
|---|---|---|---|---|---|---|---|---|
| 3.993 | | | | 0.002 | 1.4 | 5 | 254 | ref fine - kill grid |
| 10.211 | | | | 001 | 3.6 | 255 | 1 | update random forcing |
| 1.125 | | | | 47 | | 200 | 1 | courant |
| 1.642 | | | | 0 | 0.8 | 1 | 192 | hydro - new vars |
| 243.9 | | | | 00 | 85.2 | 252 | 47 | hydro - godunov |
| 10 | | | | | 3.8 | 1 | 47 | hydro - ghostzones |
| | | | | | 0.4 | 48 | 1 | hydro - old vars |
| 9 | | | | | 3.4 | 227 | 142 | do forcing |
| 1.8 | | | | 19 | 0.8 | 47 | 147 | hydro - boundaries |
| 286 | | | | | | | | |

Fortunately, RAMSES has been running on Blue-Gene/Q on up to 16 cores / 64 threads. Still, going to 240 threads, Amdahl hits back.

***Xeon-Phi ... rds / 3840 cores / 15360 threads***

| minimum | average | maximum | std dev | std/av | % | rmn | rmx | TIMER |
|---|---|---|---|---|---|---|---|---|
| 3.374 | 3.614 | | 0.096 | 0.027 | | 161 | 18 | ref fine - authorize_fin |
| 2.322 | 2.713 | | 0.156 | 0.057 | 0.7 | 240 | 41 | ref fine - make grid |
| 44.543 | | | 146 | 0.003 | 11.0 | 37 | 240 | ref fine - kill grid |
| 7.548 | | | | 0.008 | 1.9 | 255 | 2 | update random forcing |
| 12.267 | | | | 0.005 | | 192 | 140 | courant |
| 8.088 | | | | 0.008 | 2.0 | 140 | 192 | hydro - new vars |
| 244.68 | | | | 0.000 | | 1 | 18 | hydro - godunov |
| 52.27 | | | | 0.0 | 12.8 | 1 | 231 | hydro - ghostzones |
| 1.101 | | | | 0.00 | 0.3 | 117 | 156 | hydro - old vars |
| 9.212 | | | 72 | 0.049 | 2.4 | | | do forcing |
| 17.282 | | | 493 | 0.069 | 5.3 | 253 | 129 | hydro - boundaries |
| 412.005 | | | | | | | | |

Ghostzone update (MPI) is much more expensive

# Looking forward

- ***RAMSES is ready for production on Xeon-Phi + CPUs***
- PP-code will be ready very soon
- KROME is trivially OpenMP'ed, and ready too
- We have ***no code with stellar performance*** on Xeon-Phi

- Non-trivial to get good performance on Xeon-Phi.

Three layers of parallelism:

(1) Vectorisation for good serial performance ***[hard; if implicit]***
(2) Threading to parallelize inside card ***[60+ threads? some work]***
(3) Tolerant MPI between cards and host CPUs ***["easy"]***

- Platform is memory constrained and MPI communication latency / bandwidth not good. Weak cores bad for communication (?)

- **Good news:** Enhancements will improve performance on any architecture, and future proof codes ***(better than offload!)***

- ***Vectorisation the biggest problem (Knights Landing solves rest)***