Porting, optimization and bottleneck of OpenFOAM in KNL environment

I. Spisso[a], G. Amati[b], V. Ruggiero[b], C. Fiorina[c]

a) SuperCompunting Application and Innovation Department, Cineca, Via Magnanelli 6/3, 40133, Casalecchio di Reno, Bologna, Italy.

b) SuperCompunting Application and Innovation Department, Cineca, Via dei Tizii, 6 40133, Roma, Italy.

c) Milano Multiphysics S.R.L.S, Via Giorgio Washington 96, 20146 Milan, Italy

Intel eXtreme Performance Users Group (IXPUG) Europe Meeting, CINECA, Casalecchio di Reno, 5-7 March 2018

Outline of the presentation

- HPC Hardware technological trend: towards the exascale (Ivan)
- OpenFOAM & HPC bottlenecks (Ivan)
- Up to date performance using Marconi (Giorgio)
- KNL & vectorization: preliminary results (Carlo)
- Further suggested work (All)

- Target: share our experience with other KNL center/Intel staff

HPC Hardware technological trend: towards the exascale

HPC & CPU Intel evolution @ CINECA

HPC & CPU

Intel evolution: 2010-2018

Westmere (a.k.a. plx.cineca.it)

–Intel(R) Xeon(R) CPU E5645 @**2.40GHz**, 6 Core per CPU

Sandy Bridge (a.k.a. eurora.cineca.it)

–Intel(R) Xeon(R) CPU E5-2687W 0 @**3.10GHz**, 8 core per CPU

Ivy Bridge (a.k.a pico.cineca.it)

–Intel(R) Xeon(R) CPU E5-2670 v2 @2.**50GHz,** 10 core per CPU

–Infiniband FDR

Hashwell (a.k.a. galileo.cineca.it)

–Intel(R) Xeon(R) CPU E5-2630 v3 @**2.40GHz**, 8 core per CPU

–Infiniband QDR/True Scale (x 2)

Broadwell A1 (a.k.a marconi.cineca.it)

–Intel(R) Xeon(R) CPU E5-2697 v4 @ **2.30GHz**, 18 core per CPU (x2)

–OmniPath

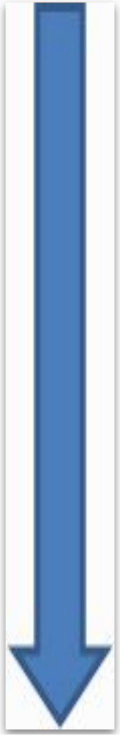
KNL A2 (a.k.a marconi.cineca.it)

– Intel(R) Knights Landing @ **1.40GHz**, 68 cores per CPU

–OmniPath

SKL A3 (a.k.a marconi.cineca.it)

–Intel Xeon 8160 CPU @ **2.10GHz,** 24 core per CPU (x2)



Increasing # of cores,
Same clock

exascale: computing system capable of al least one exaFLOPs calculation per second.

exaFLOPs = 10^18 FLOPS or a billion of billion calculations per seconds

As clock speeds may for reasons of power efficiency be as low as 1 Ghz

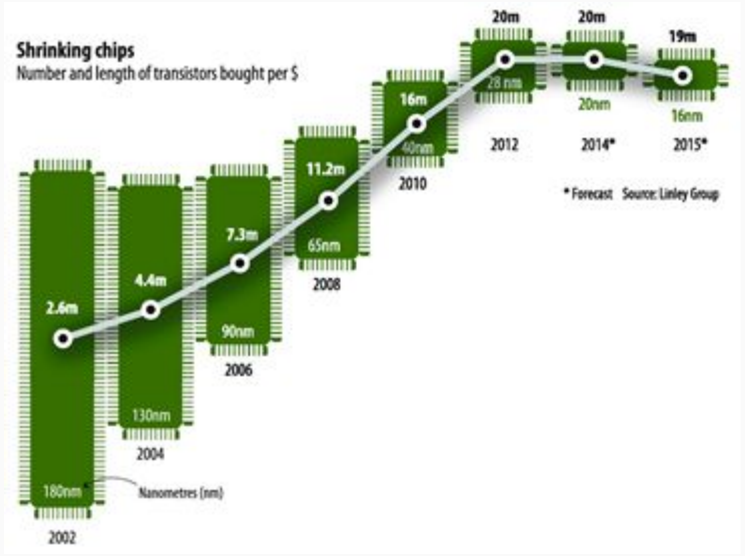
To Performe 1 Eflop/s peak performance Need to execute 1 billion floating-point operations concurrently (Total Concurrency)

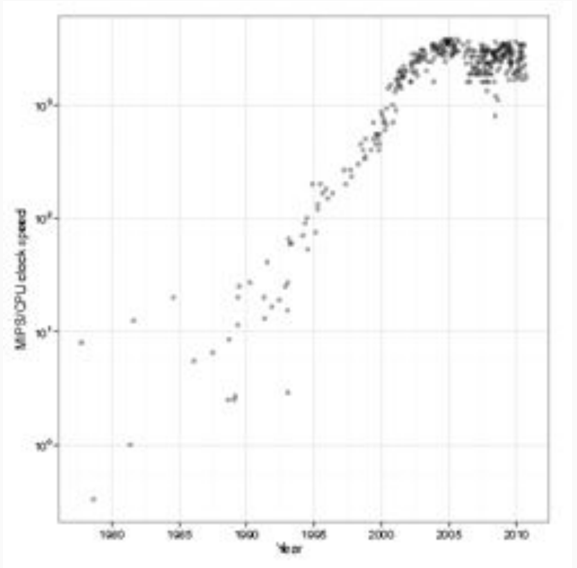
MTTI = Mean Time to interrupt, order of day(s)

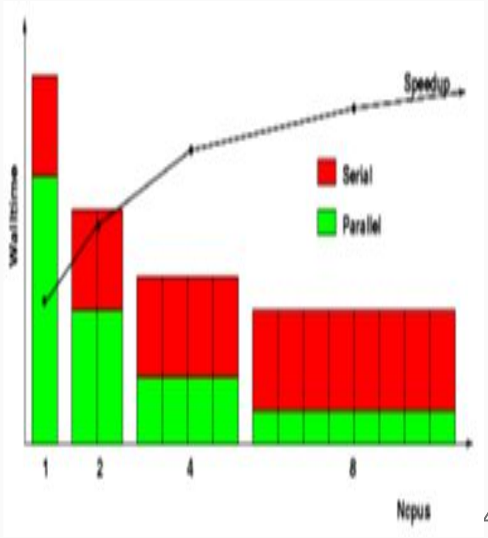
HPC Hardware technological trend: towards the exascale

Road Map to exascale: Governing Laws

Moore's law: is the observation that the number of transistors in a dense integrated circuit doubles approximately every two years (18 months, Intel executive David House), hold untill 2006.
Dennard scaling law: also known as MOSFET scaling states that as transistors get smaller their power density (P) stays constant, so that the power (D) use stays in proportion with area: both voltage (V) and current scale (downward) with length.
Amdahl's law is a formula which gives the theoretical speedup in latency of the execution of a task at fixed workload that can be expected of a system whose resources are improved. maximum speedup tends to $1 / (1 - P)$ P= parallel fraction. This menas that 1,000,000 core, P = 0.999999, Serial Fraction 0,000001







HPC Hardware technological trend: towards the exascale

Road Map to exascale

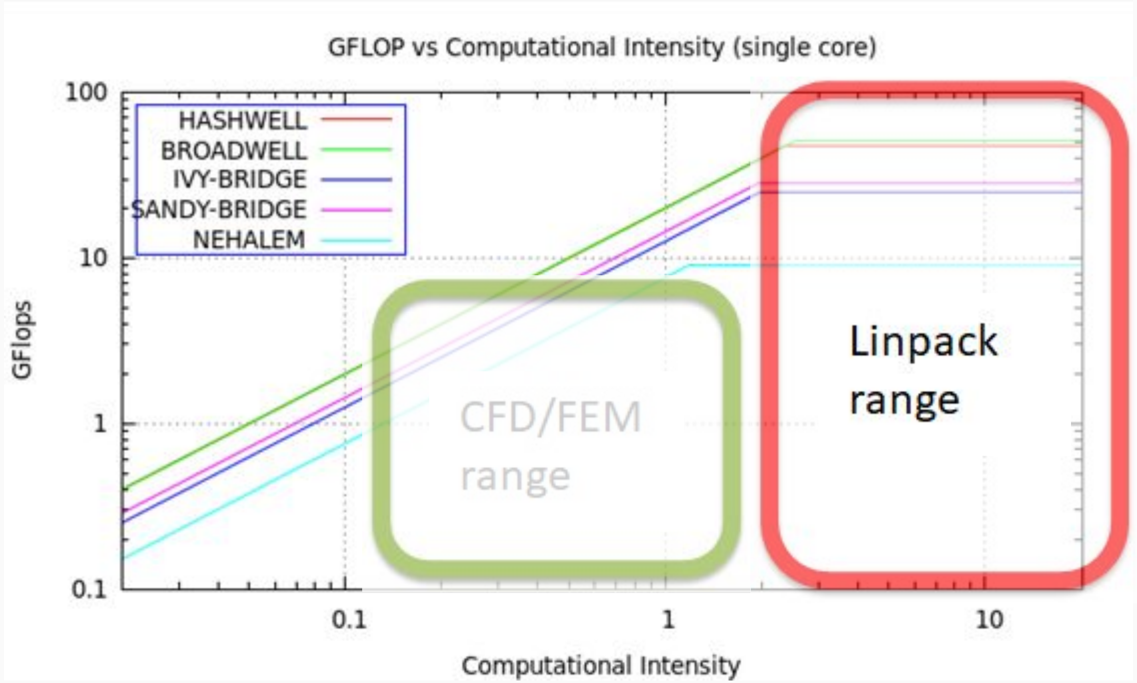
- Now power and heat exchange are the limiting factor of the down-scaling **Dennard scaling law does not hold no more.**
- The core frequency and performance do not grow following the Moore’s law any longer
- Increase the number of cores to maintain the architectures evolution on the Moore’s law
- Change of Paradigm: Energy Efficiency.
- New chips designed for maximum performance in a small set of workloads, Simple functional units, poor single thread performance, but maximum throughput.
- New architectural solution towards the exascale: heterogeneous systems examples CPU + ACC (GPU/MIC/FPGA) or Intel KNC and evolution KNL.

HPC Hardware technological trend: towards the exascale

Roofline model and Computational Intensity

- The roofline model: Performance bound (y-axis) ordered according to arithmetic intensity (x-axis) (i.e. GFLOPs/Byte)
- Arithmetic Intensity: is the ratio of total floating-point operations to total data movement (bytes): i.e. flops per byte

Which is the OpenFoam (CFD/FEM) arithmetic intensity? About 0.1, may be less….

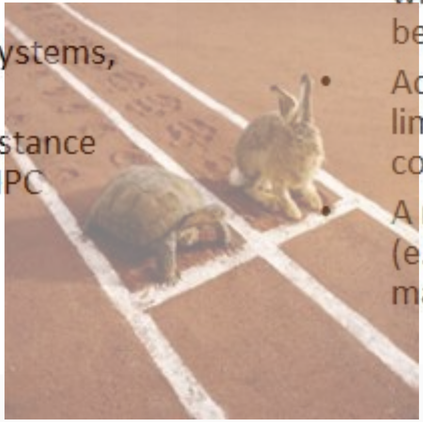


HPC Hardware technological trend

Towards the exascale: co-design is required!

Software (turtle)

- As usual software lags behind hardware but must learn to exploit accelerators and other innovative technologies such as FGPAs, PGAS
- Reluctance by some software devs to learn new languages such as CUDA, OpenCL is driving interest in compiler-directive languages such as OpenAcc and OpenMP (4.x)
- Continued investment in efficient filesystems, checkpointing, resilience, parallel I/O
- **co-design** is the way the reduce the distance between hardware and software for HPC

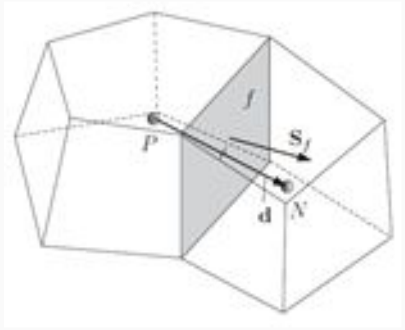


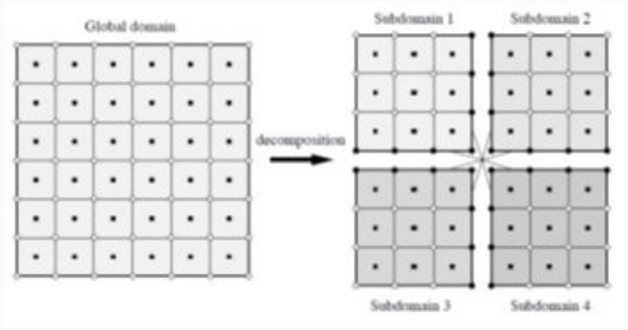
Hardware (hare)

- Reaching physical limits of transistor densities and increasing clock frequencies further is too expensive and difficult (energy consumption, heat dissipation)
- Parallelism only solution in HPC but the Blue Gene road is no longer being persued. Hybrid with accelerators such as GPUs or Xeon Phi become the norm
- Accelerator technologies advancing to remove limits associated with CPU/ACC communication(Intel KNL or Nvidia NVLINK)
- A range of novel architectures being explored (e.g. Mont Blanc, DEEP) and technologies in many areas

OpenFOAM: algorithmic overview

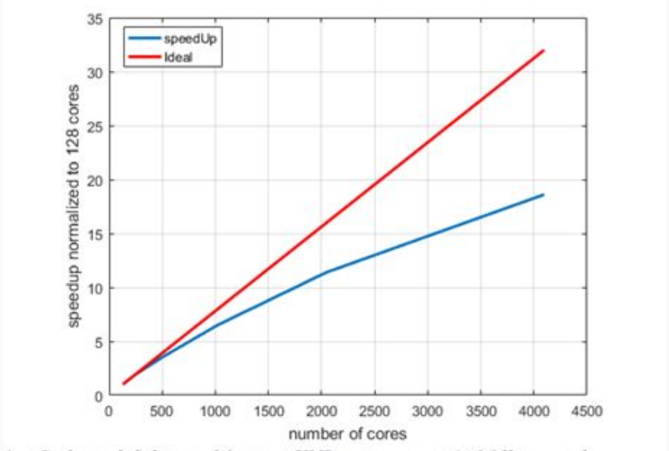
- OpenFOAM is first and foremost a C++ library used to solve in discretized form systems of Partial Differential Equations (PDE). OpenFOAM stands for **Open F**ield **O**peration **a**nd **M**anipulation
- The Engine of OpenFOAM is the Numerical Method. To solve equations for a continuum, OpenFOAM uses a numerical approach with the following features:
  - segregated, iterative solution (PCG/GAMG), unstructured finite volume method, co-located variables, equation coupling.
- The method of parallel computing used by OpenFOAM is based on the standard Message Passing Interface (MPI) using the strategy of domain decomposition.
  - The geometry and the associated elds are broken into pieces and allocated to separate processors for solution. (zero layer domain decomposition)
  - A convenient interface, **Pstream**, is used to plug any Message Passing Interface (MPI) library into OpenFOAM. It is a light wrapper around the selected MPI Interface
- OpenFOAM scales reasonably well up to thousands of cores



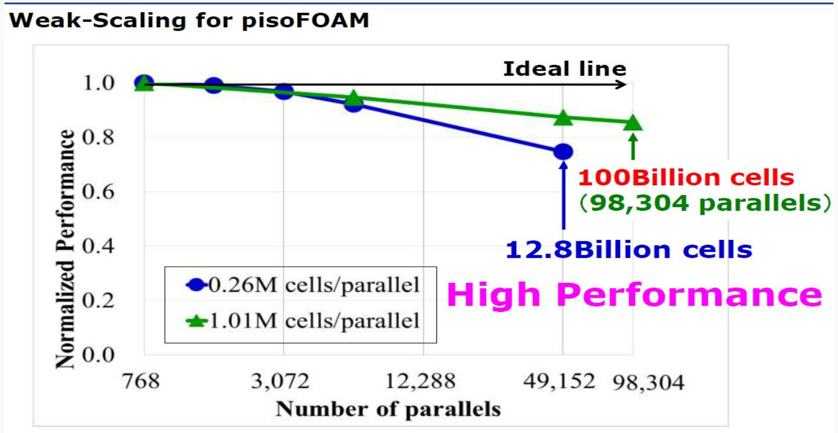


OpenFOAM: HPC Performances

- OpenFOAM scales reasonably well up to thousands of cores, upper limit orders of thousands of cores. Where we are looking at is radical scalability =) The real issues are in the scaling of cases of 100's of millions of cell on 10K+ cores.
- Custom version by Shimuzu Corp., Fujitsu Limited and RIKEN on K Computer (K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect) was able to achieve a large scale transient CFD simulation up to 100 billion cell meshes and achieved a high performance for 100 thousand MPI parallels.



1 – Code scalability on Marconi KNL system on a 64 Million mesh



[1] **Pham Van Phuc et al., Shimizu Corporation, Fujitsu Limited, Riken: Evaluation of MPI Optimization of C++ CFD Code on the K Computer, SIG Technical Reports Vol. 2015-HPC-151 No. 19 2015/10/01. (in Japanese)**

OpenFOAM HPC bottlenecks

Up to date, the well known bottlenecks for a full enabling of OpenFOAM for massively parallel clusters are

1. Scalability of the linear solvers and limit in the parallelism paradigm. In most cases the **memory bandwidth** is a limiting factor for the solver. Additionally, global reductions are frequently required in the solvers.

   The linear algebra core libraries are the main communication bottlenecks for the scalability
   Whole bunch of MPI Allreduce stems from an algorithmic constraint and is unavoidable, increasing with the number of cores, . . . unless an algorithmic rewrite is proposed.
   Generally speaking, the fundamental difficulty is the inability to keep all the processors busy when operating on very coarse grids. Need for communication-friendly agglomeration (geometric) linear multigrid solver.

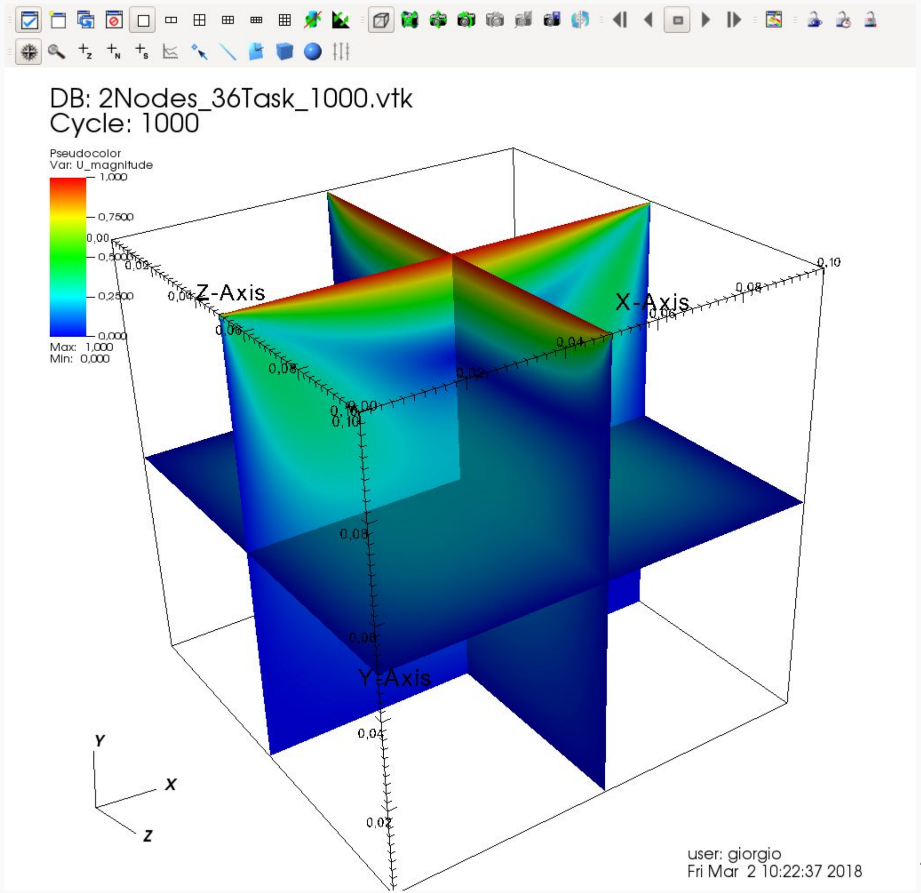
2. Sparse Matrix storage format: The **LDU** sparse matrix storage format used internally does not enable any cache-blocking mechanism (SIMD, vectorization).
3. the **I/O** data storage system: when running in parallel, the data for decomposed fields and mesh(es) has historically been stored in multiple files within separate directories for each processor, which is a bottleneck for big simulation. For example LES/DNS with hundreds of cores requires very often saving on disk. Work on going:
   a. Implemention of ADIOS Parallel I/O library by Esi-OpenCFD and Oak Ridge National Laboratory https://openfoam.com/documentation/files/adios-201610.pdf
   b. Collocated file format by OpenFOAM Foundation, https://openfoam.org/news/parallel-io/

Up to date performance using Marconi

- Both 3 partition are tested for reference
  - BDW: 36 core for node
  - KNL: 272 core for node (HT on)
  - SKL: 48 core for node
- Node level comparison (serial performance is meaningless for HPC)
- Two test case
  - Driven Cavity
    - From 100^3 to 400^3 (1 M, 8M, 27M and 64 M of cells)
    - **`icofoam,PCG, scotch`**
  - Flow around a cylinder
    - About 2M elements
    - **`pimpleFoam, GAMG, scotch`**
- Target: Define a reference value for performance

Driven Cavity

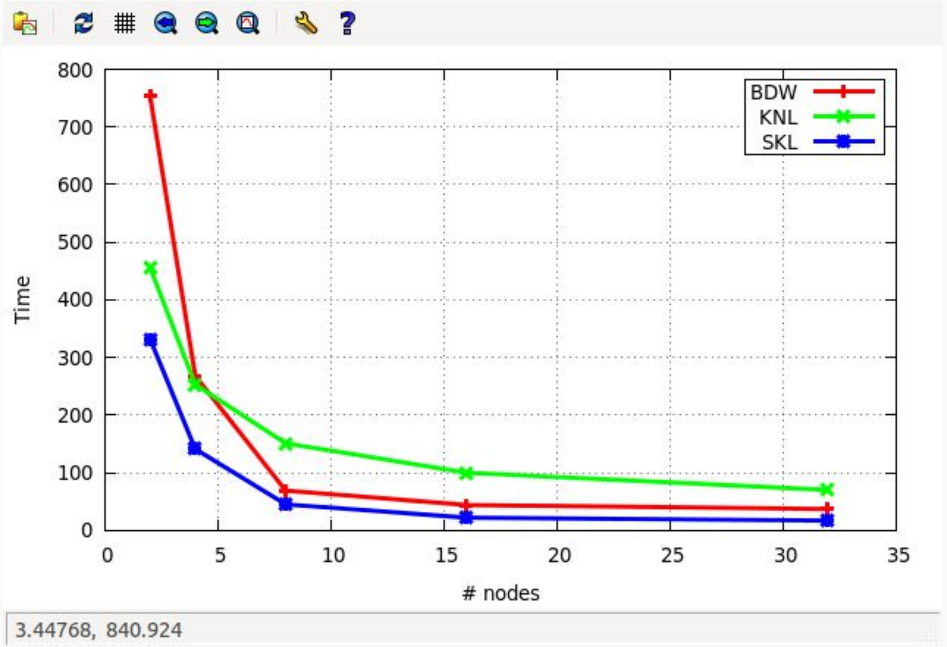
- `100^3, T=0.5, deltaT=0.0005`
- `200^3, T=0.2, deltaT=0.005`
- `300^3, T=0.2, deltaT=0.005`
- `400^3, T=0.1, deltaT=0.0025`
- Target:
  - Looking for performance
  - Looking for saturation
  - Looking for bottleneck



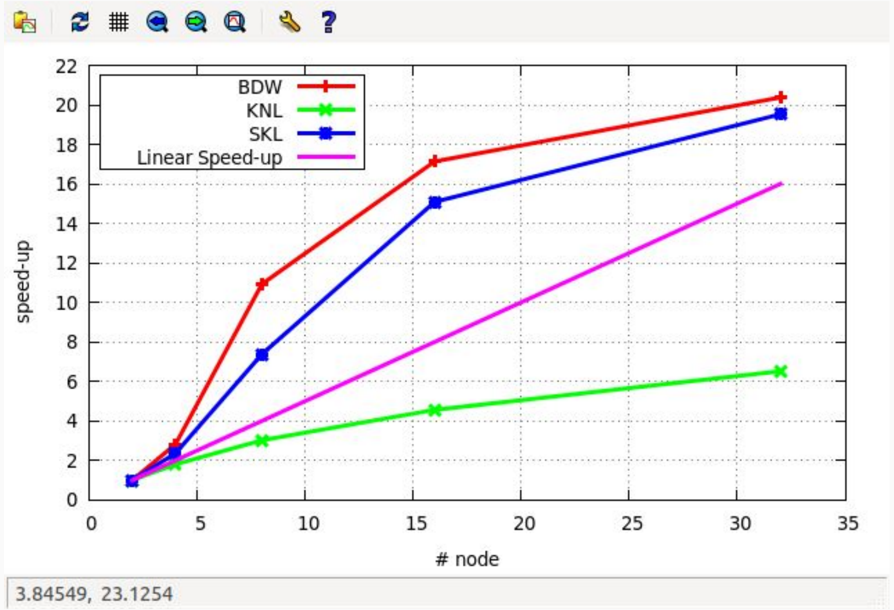
Driven Cavity: some info

- **`openFoam+`** release used (from ESI)
    - Compiled using intel compiler + intelmpi
- SKL: Release **`v1712`** slightly faster then **`v1606,v1612,v1712`**
- BDW: Release **`v1712`** slightly faster then **`v1606,v1612,v1712`**
- KNL: knl flag (**`-avx512`**) seems ineffective (see vectorization section)
- 100^3 only to validate the flow, too small for a an HPC testcase
- “strange/slow” performance are tested many times to avoid “dirty” figures
- Validation done comparing residual
- For first level MPI profiling
    - **`I_MPI_STATS=ipm`**

Driven Cavity: 200^3

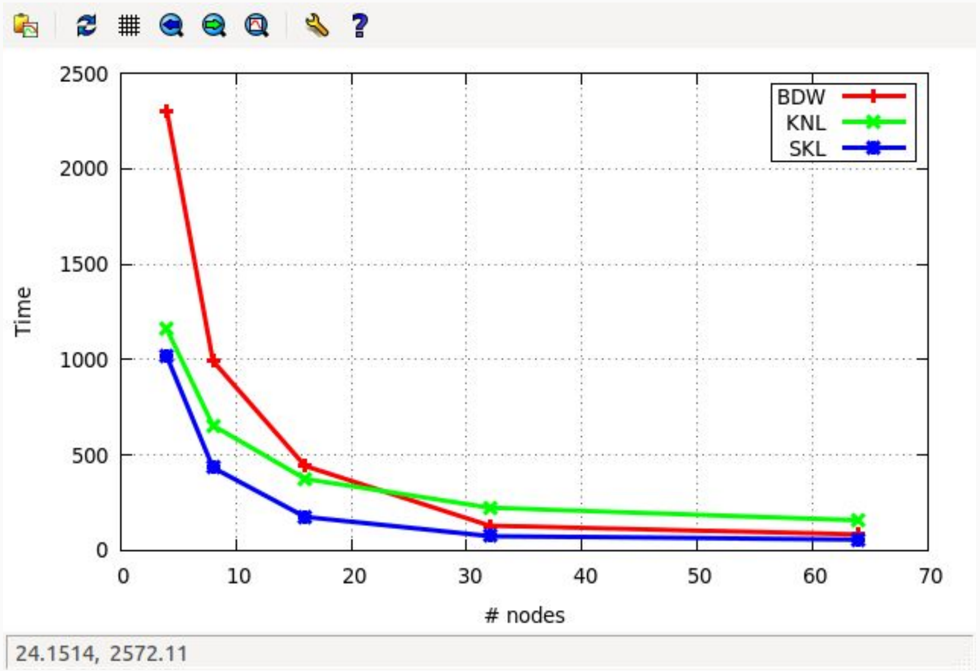


3.44768, 840.924

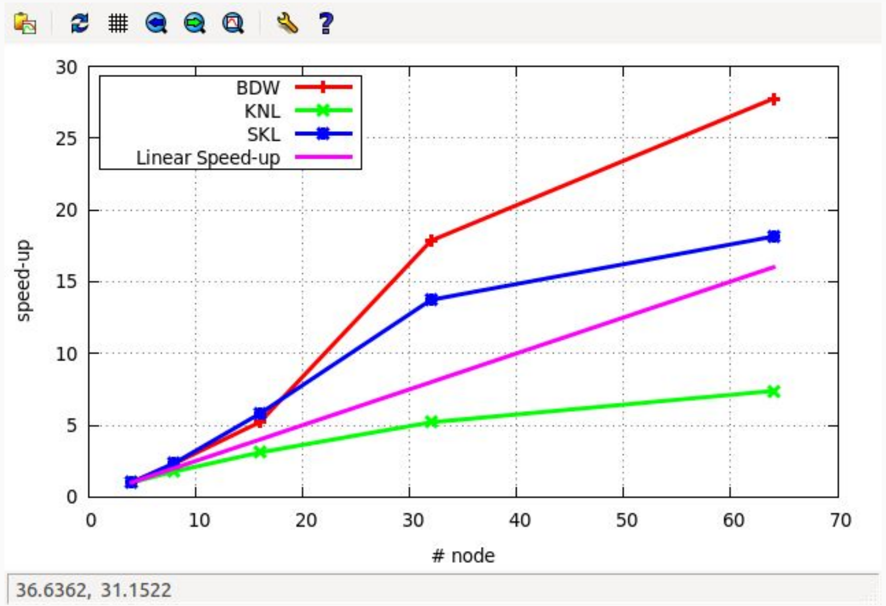


3.84549, 23.1254

Driven Cavity: 300^3

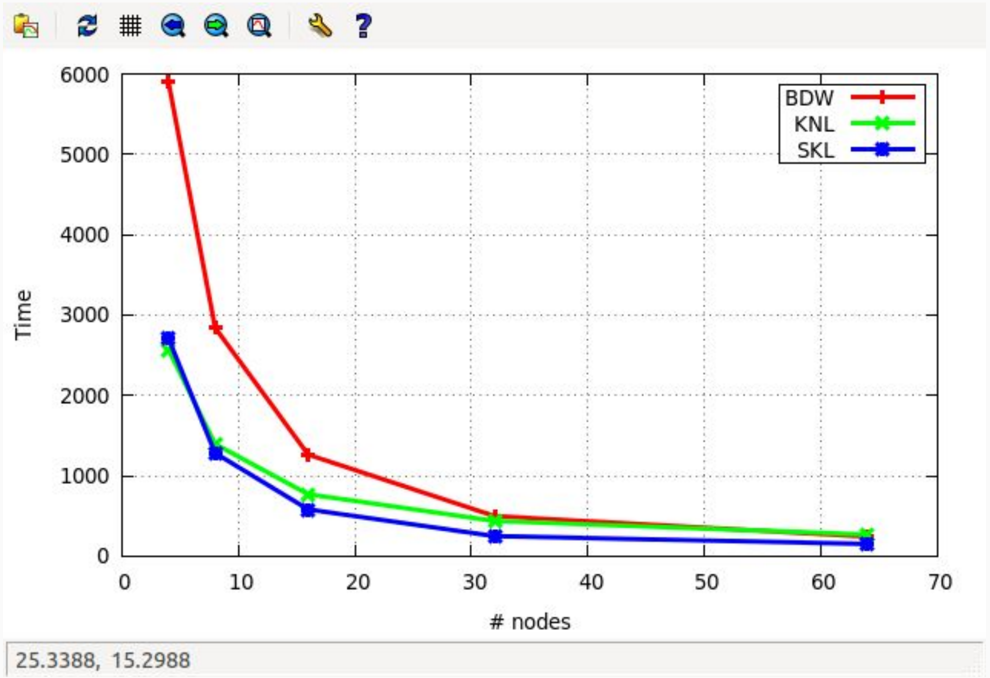


24.1514, 2572.11

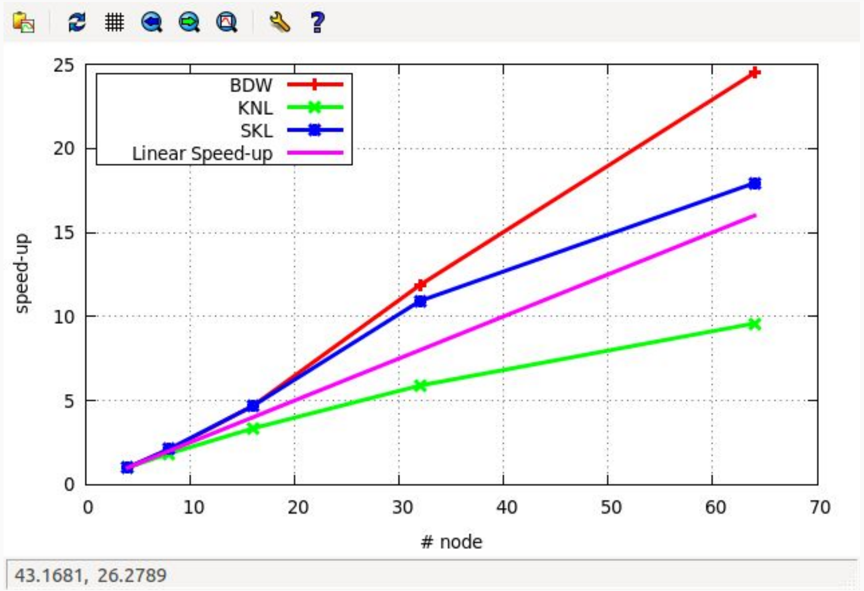


36.6362, 31.1522

Driven Cavity: 400^3



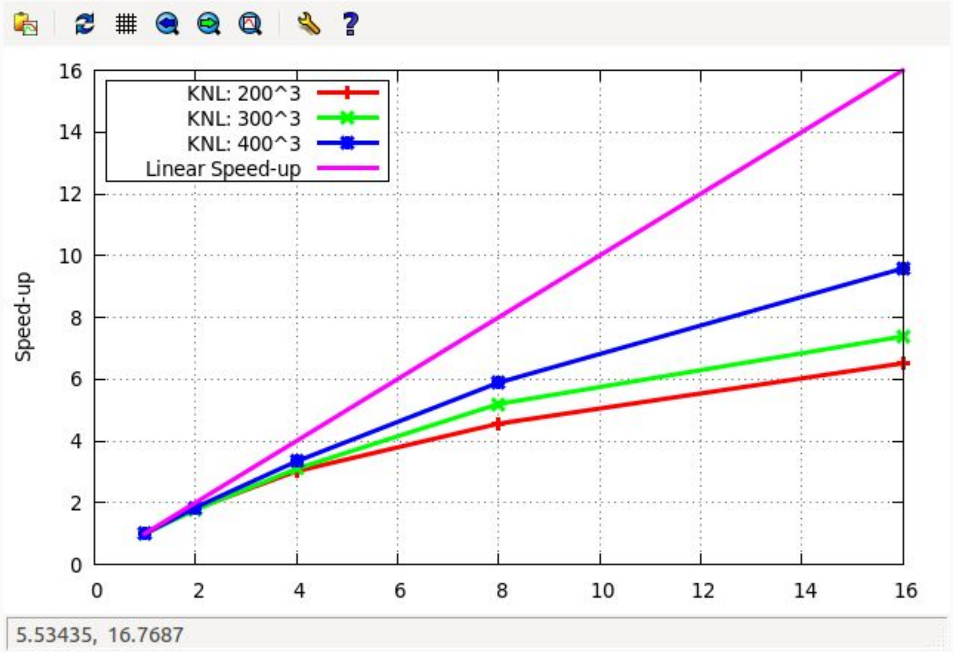
25.3388, 15.2988



43.1681, 26.2789

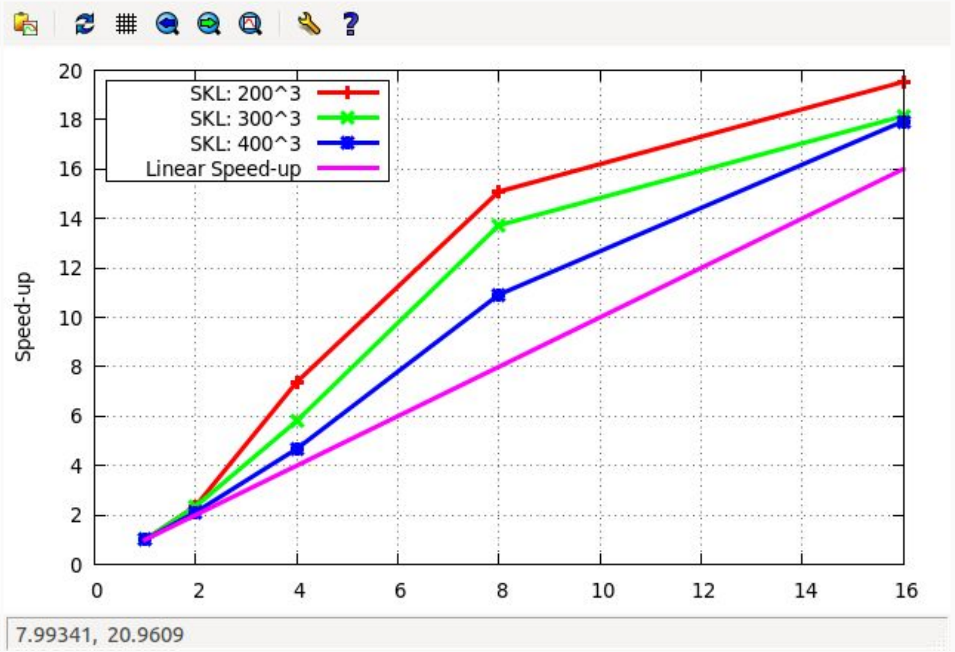
Driven Cavity: KNL & SKL speed-up

KNL



5.53435, 16.7687

SKL

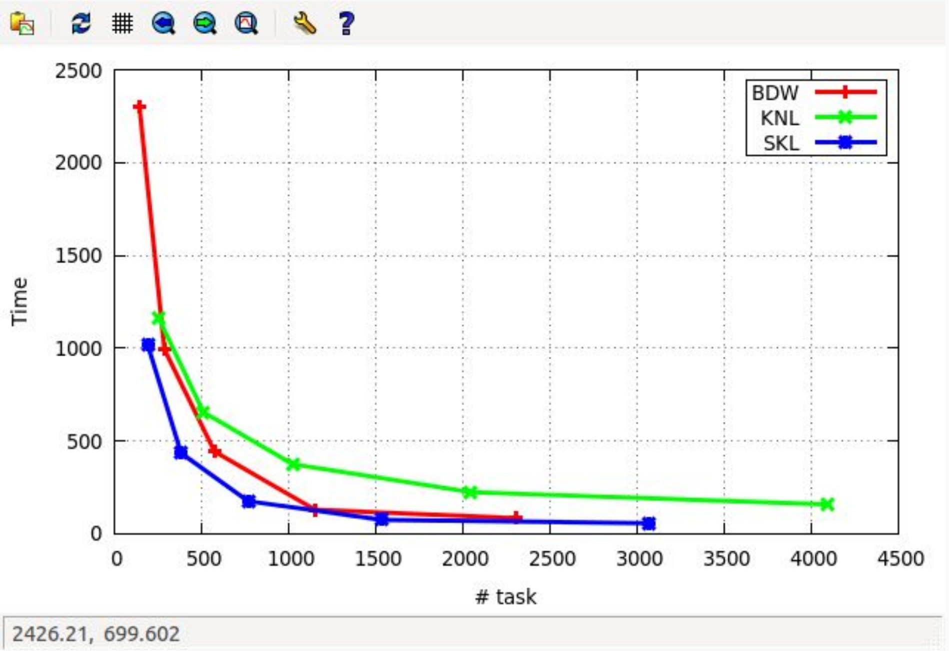


7.99341, 20.9609

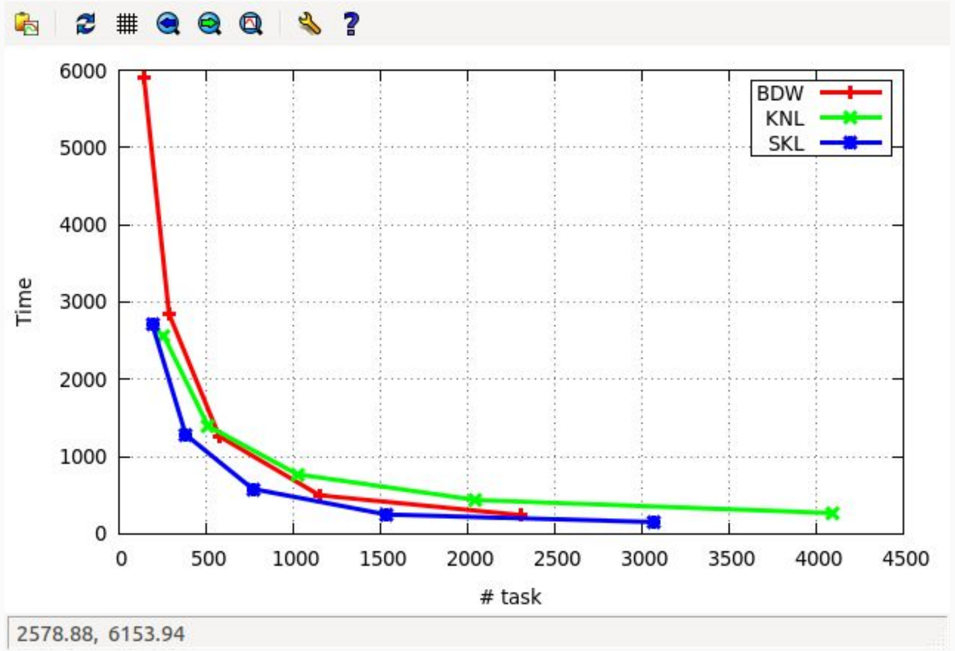
Driven Cavity: time vs. task (300 & 400)

300^3

400^3



2426.21, 699.602



2578.88, 6153.94

Driven Cavity:KNL MPI overhead

- 200^3, **`I_MPI_STAT=ipm`**
- BDW not reported, overhead > 50% in time
- **`MPI_INIT`** removed

Task	nodes	time	% mpi	% allreduce	% waitall	% recv
128	2	466	**17**	**14**	1.4	0.2
256	4	255	**22**	**17**	2.5	0.3
512	8	158	**35**	**25**	4.6	1.1
1024	16	103	**48**	**33**	6.8	1.5
2048	32	77	**62**	**44**	8.0	1.7

Driven Cavity: SKL MPI overhead

- 200^3, **`I_MPI_STAT=ipm`**
- BDW not reported, overhead > 50% in time
- **`MPI_INIT`** removed

Task	nodes	time	% mpi	% allreduce	% waitall	% recv
96	2	331	**8.6**	**6.0**	1.4	0.1
192	4	133	**18**	**13**	2.9	0.5
384	8	43	**29**	**20**	4.5	0.6
768	16	22	**43**	**29**	7.0	1.4
1536	32	22	**69**	**49**	6.0	6.0

Driven Cavity:KNL MPI overhead

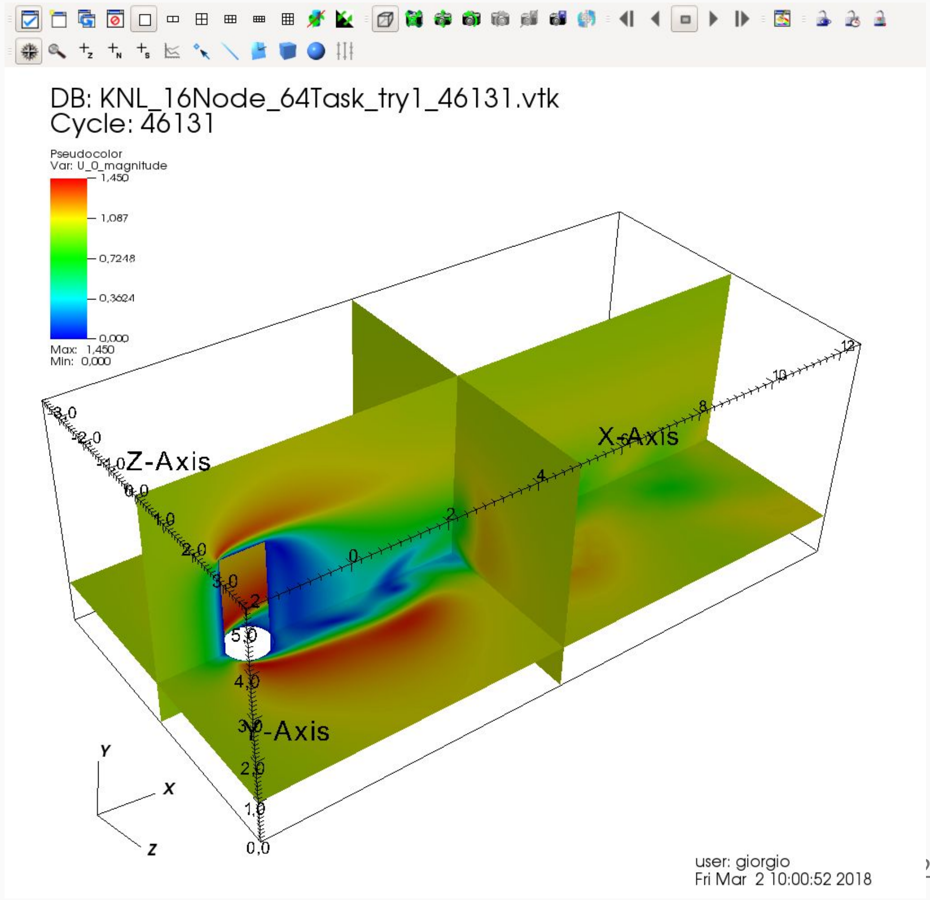
- From OSU microbenchmark **`MPI_allreduce`** benchmark
- Almost all with message **size 8 byte**
- Time in microseconds

HW	Task	nodes	Mean time	OSU time
KNL	1024	16	**208**	54
KNL	2048	32	**210**	62
SKL	768	16	**48**	13
SKL	1536	32	**79**	17

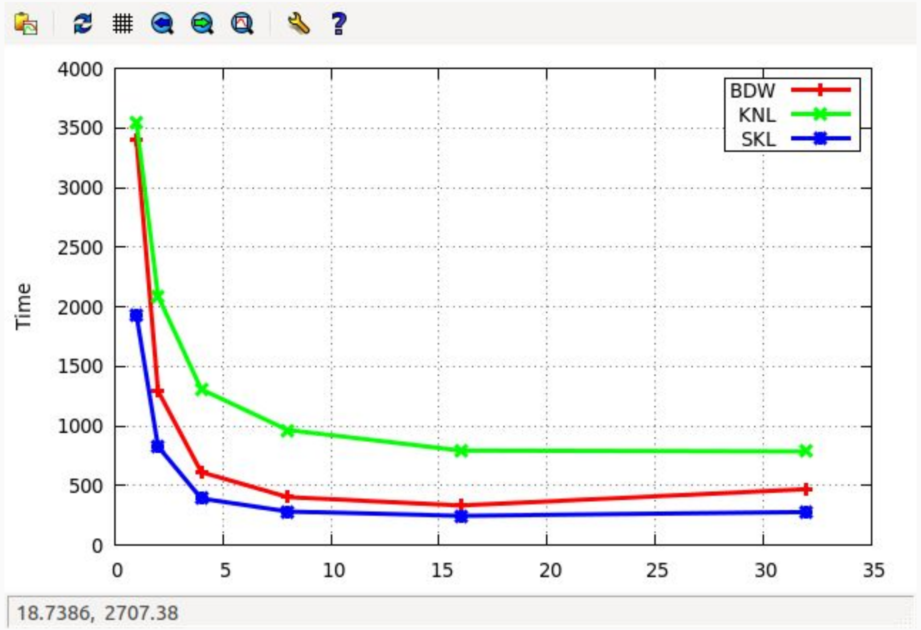
- 5 time slower….why?

Flow around a cylinder

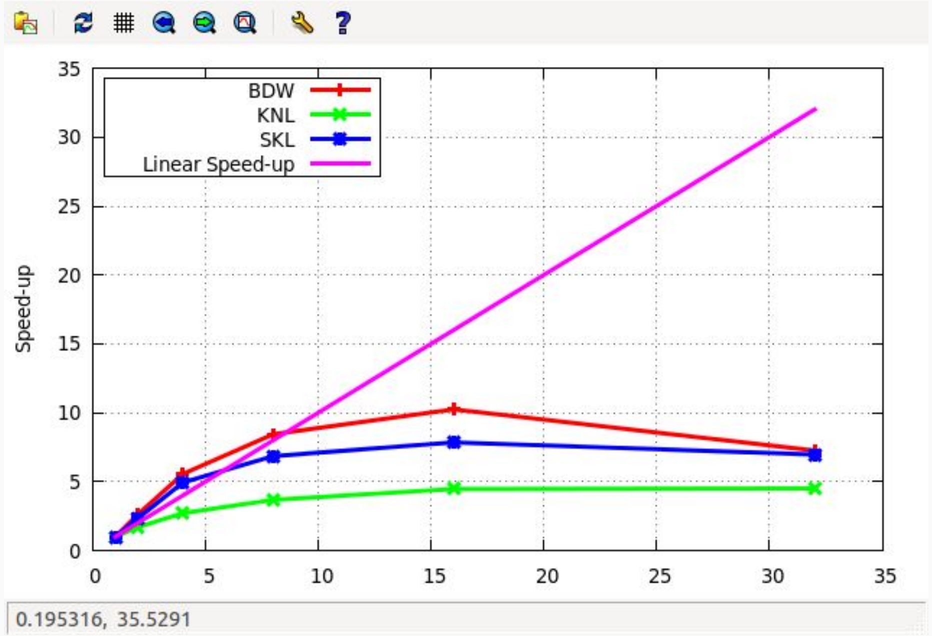
- **`T=10, deltaT=1e-3`**
- **`adjustTimeStep yes`**
- **`2 M of cells`**
- More complex test (academic)
- Target:
  - Looking for performance
  - Looking for saturation
  - Looking for bottleneck


DB: KNL_16Node_64Task_try1_46131.vtk
Cycle: 46131

Flow around a Cylinder

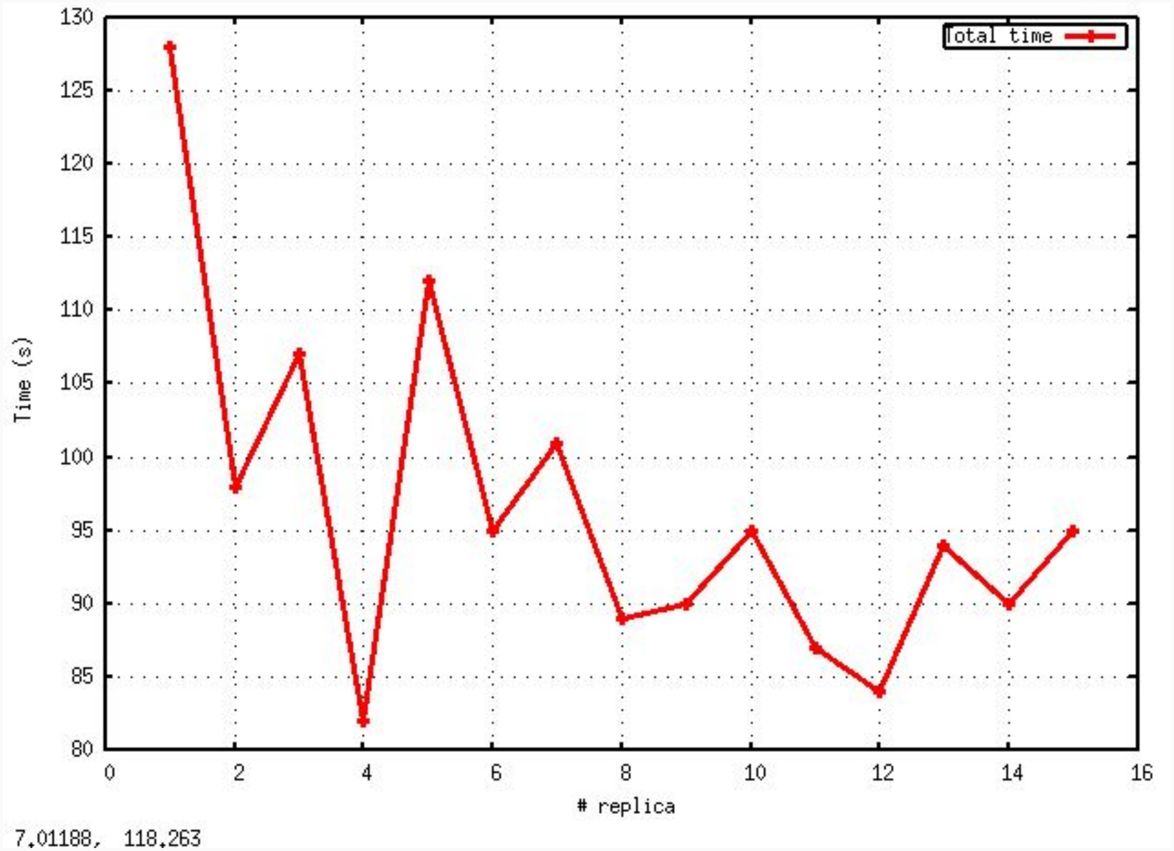


18.7386, 2707.38



0.195316, 35.5291

Performance issue (Marconi?)



- SKL
- 400^3
- 128 Nodes
- 48 Task per node

More than 50% variation!!!

Flow around a cylinder: MPI overhead

- 16 node testcase, **`I_MPI_STAT=ipm`**
  - BDW not reported, overhead > 50% in time
  - SKL: from 246” → 274”
  - KNL: from 800” → 890
- **`MPI_INIT`** removed

HW	time	% mpi	% allreduce	% waitall	% recv
SKL	274	**77**	**44**	**22**	5
KNL	849	**65**	**35**	**18**	4

HW	Task	nodes	Mean time (micros)	OSU time
KNL	1024	16	**150**	54
SKL	768	16	**60**	13

Effect of vectorization: DIC-PCG

- 100^3 lid-driven cavity as test case - 8 cores
- preconditioned conjugate gradient algorithm (PCG) with a diagonal-based incomplete cholesky (DIC) preconditioner
- 353.3 to 330.7 seconds, less than 10% speed-up

Effect of vectorization: DIC-PCG

File location	Line	Code snippet	Vectorized	Time w/o vectorization	Time with vectorization
src/OpenFOAM/matrices/lduMatrix/preconditioners/DICPreconditioner/DICPreconditioner.C	109	`for (label cell=0; cell<nCells; cell++) { wAPtr[cell] = rDPtr[cell]*rAPtr[cell]; }`	Yes	9.2 (2.6%)	5.3 (1.6%)
	114	`for (label face=0; face<nFaces; face++) { wAPtr[uPtr[face]] -= rDPtr[uPtr[face]] *upperPtr[face]*wAPtr[lPtr[face]]; }`	No	59.9 (17.0%)	59.5 (18.0%)
	119	`for (label face=nFacesM1; face>=0; face--) { wAPtr[lPtr[face]] -= rDPtr[lPtr[face]]* upperPtr[face]*wAPtr[uPtr[face]]; }`	No	66.6 (18.9%)	67.1 (20.3%)

Effect of vectorization: DIC-PCG

File location	Line	Code snippet	Vectorized	Time w/o vectorization	Time with vectorization

src/OpenFOAM/matrices/lduMatrix/preconditioners/DICPreconditioner/DICPreconditioner.C	109	`for (label cell=0; cell<nCells; cell++)`
		`{`
		`    wAPtr[cell] = rDPtr[cell]*rAPtr[cell];`
		`}`
			Yes	9.2 (2.6%)	5.3 (1.6%)

	114	`for (label face=0; face<nFaces; face++)`
		`{`
		[illegible]
		`}`
			No	59.9 (17.0%)	59.5 (18.0%)

	119	`for (label face=nFacesM1; face>=0; face--)`
		`{`
		`    wAPtr[lPtr[face]] -= rDPtr[lPtr[face]]*upperPtr[face]*wAPtr[uPtr[face]];`
		`}`
			No	66.6 (18.9%)	67.1 (20.3%)

Limited speed-up

Possible bottlenecks from fetching data from memory (large sparse matrix)

Effect of vectorization: DIC-PCG

File location	**Line**	**Code snippet**	**Vectorized**	**Time w/o vectorization**	**Time with vectorization**

src/OpenFOAM/matrices/lduMatrix/preconditioners/DICPreconditioner/DICPreconditioner.C	109	for (label cell=0; cell<nCells; cell++)
		{
		wAPtr[cell] = rDPtr[cell]*rAPtr[cell];
		}
			Internal dependencies		5.3 (1.6%)

	114	for (label face=0; face<nFaces; face++)
		{
		wAPtr[uPtr[face]] -= rDPtr[uPtr[face]] *upperPtr[face]***wAPtr**[lPtr[face]];
		}
			No	59.9 (17.0%)	59.5 (18.0%)

	119	for (label face=nFacesM1; face>=0; face--)
		{
		wAPtr[lPtr[face]] -= rDPtr[lPtr[face]]* upperPtr[face]***wAPtr**[uPtr[face]];
		}
			No	66.6 (18.9%)	67.1 (20.3%)

Effect of vectorization: DIC-PCG

File location	Line	Code snippet	Vectorized	Time w/o vectorization	Time with vectorization

src/OpenFOAM/matrices/lduMatrix/preconditioners/DICPreconditioner/DICPreconditioner.C	109	for (label cell=0; cell<nCells; cell++)
		{
		wAPtr[cell] = rDPtr[cell]*rAPtr[cell];
		}
					5.3
					(1.6%)

	114	**#pragma simd**
		for (label face=0; face<nFaces; face++)
		{
		wAPtr[uPtr[face]] -= rDPtr[uPtr[face]]
		*upperPtr[face]*wAPtr[lPtr[face]];
		}			

	119	**#pragma simd**
		for (label face=nFacesM1; face>=0; face--)
		{
		wAPtr[lPtr[face]] -= rDPtr[lPtr[face]]*
		upperPtr[face]*wAPtr[uPtr[face]];
		}
					(20.3%)

Forced vectorization

One loop speeds up by 2 times, the other slows down by 2.5 times (data retrival from RAM, double indexing, ?)

Effect of vectorization: GAMG

- 100^3 lid-driven cavity as test case
- Geometric agglomerated algebraic multigrid solver (GAMG) with Gauss-Seidel smoother
- **No speed up**

Effect of vectorization: take home messages

OpenFOAM makes very little use of vectorization:

- Non-vector-friendly algorithms
- Non-vector friendly implementation of these algorithm
    - Double indexing frequently used
    - Inefficient retrieval of data from memory (unstructured meshes, large sparse matrices)

Further on-going work

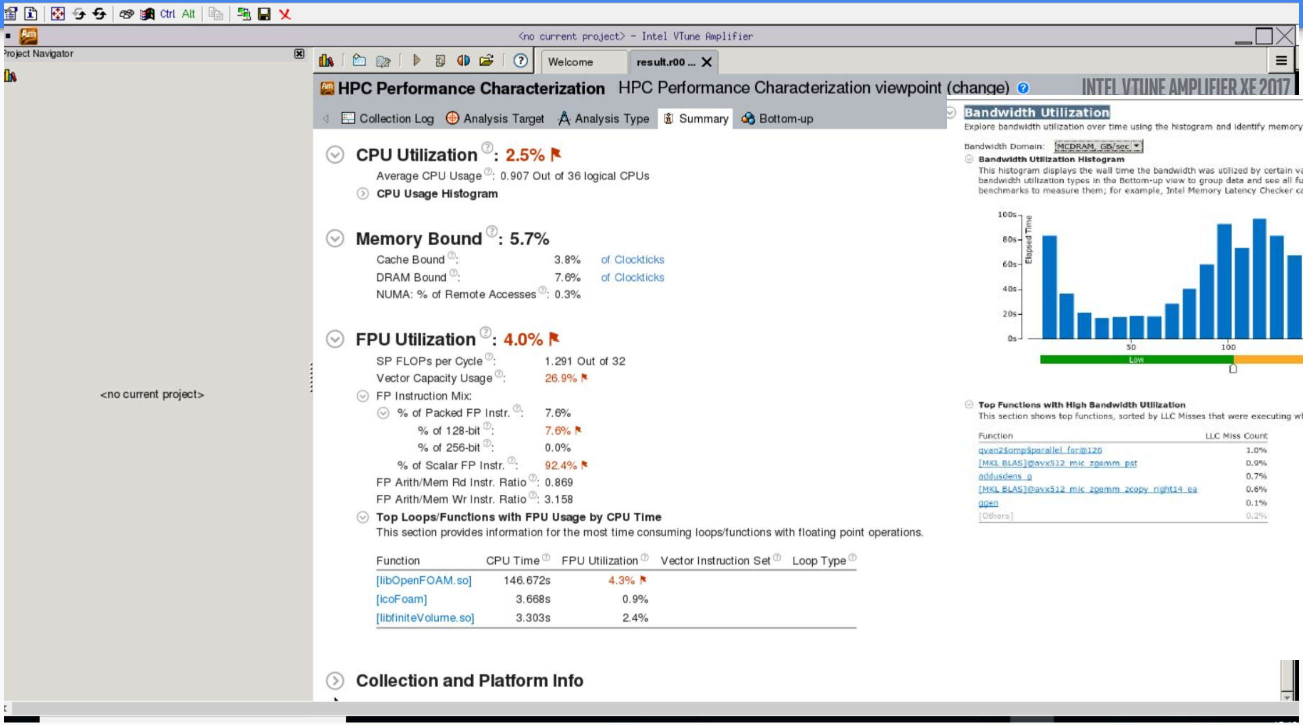
HPC Profiling and testing

- An in depth profiling analysis is aiming to identify a set of useful tools and corresponding metrics to spot the HPC bottlenecks
- Tools used
  - Intel Amplifier
  - Intel Advisor
  - Intel Monitor Performance Snapshot
- Metrics to be explored/evaluated
  - Memory bound
  - Memory bandwidth

- Test and Installation on Intel Patch on KNL architecture

Further on-going work

HPC Profiling: VTune bandwidth utilization



HPC Performance Characterization HPC Performance Characterization viewpoint (change)

CPU Utilization: 2.5%

Average CPU Usage: 0.907 Out of 36 logical CPUs

CPU Usage Histogram

Memory Bound: 5.7%

Cache Bound:	3.8%	of Clockticks
DRAM Bound:	7.6%	of Clockticks
NUMA: % of Remote Accesses:	0.3%	

FPU Utilization: 4.0%

SP FLOPs per Cycle:	1.291 Out of 32
Vector Capacity Usage:	26.9%
FP Instruction Mix:	
% of Packed FP Instr.:	7.6%
% of 128-bit:	7.6%
% of 256-bit:	0.0%
% of Scalar FP Instr.:	92.4%
FP Arith/Mem Rd Instr. Ratio:	0.869
FP Arith/Mem Wr Instr. Ratio:	3.158

Top Loops/Functions with FPU Usage by CPU Time

This section provides information for the most time consuming loops/functions with floating point operations.

Function	CPU Time	FPU Utilization	Vector Instruction Set	Loop Type
[libOpenFOAM.so]	146.672s	4.3%		
[icoFoam]	3.668s	0.9%		
[libfiniteVolume.so]	3.303s	2.4%		

Collection and Platform Info

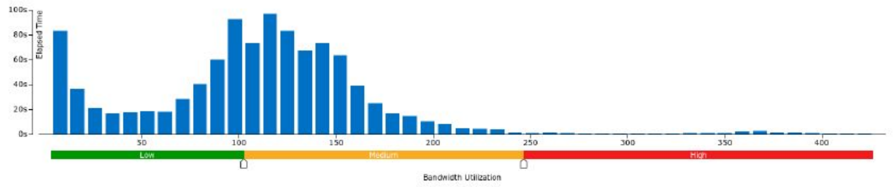
Bandwidth Utilization

Explore bandwidth utilization over time using the histogram and identify memory objects or functions with maximum contribution to the high bandwidth utilization.

Bandwidth Domain: MCDRAM, GB/sec

Bandwidth Utilization Histogram

This histogram displays the wall time the bandwidth was utilized by certain value. Use sliders at the bottom of the histogram to define thresholds for Low, Medium and High utilization levels. You can use these bandwidth utilization types in the Bottom-up view to group data and see all functions executed during a particular utilization type. To learn bandwidth capabilities, refer to your system specifications or run appropriate benchmarks to measure them; for example, Intel Memory Latency Checker can provide maximum achievable DRAM and QPI bandwidth.



Top Functions with High Bandwidth Utilization

This section shows top functions, sorted by LLC Misses that were executing when bandwidth utilization was high for the domain selected in the histogram area.

Function	LLC Miss Count
qvan2$omp$parallel_for@126	1.0%
[MKL BLAS]@avx512_mic_zgemm_pst	0.9%
addusdens_g	0.7%
[MKL BLAS]@avx512_mic_zgemm_zcopy_right14_ea	0.6%
ggen	0.1%
[Others]	0.2%

10 / 14

Further on-going work
CSR format for sparse matrix, external linear solver algebra

Reminder two (of three) bottlenecks (1) limit in the parallelism paradigm. In most cases the memory bandwidth is a limiting factor for the solver. Additionally, global reductions are frequently required in the solvers.(2) The LDU sparse matrix storage format used internally does not enable any cache-blocking mechanism (SIMD, vectorization).

(1) and (2) are both related to the sparse matrix solvers and linear algebra.

Collaboration with ESI-OpenCFD

Prototyping stage would be to add in an interface to transcribe the LDU format into another sparse matrix format CSR (Compress Row Storage) that is supported by an external solver package such as hypre or petsc.

Use a data structure that will use efficiently the data locality of the new chip-set, and will enable efficiently simd/vectorizaion

This temporarily interface could be used to benchmark improvements possible by use of one of these external solver packages. On the assumption that there are substantial benefits

- PETSC: https://www.mcs.anl.gov/petsc/
- HYPRE: https://computation.llnl.gov/projects/hypre-scalable-linear-solvers-multigrid-methods

Bonus Slide

In Situ Visualization with OpenFOAM

Simone, mi fai un riassunto di una slide?

Grazie

Ok, per quando?

Martedì mattina grazie