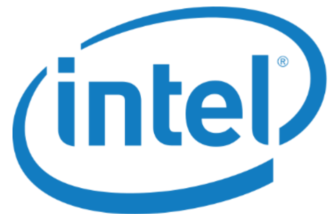
YALES2 porting on the Xeon- Phi
Early results

Othman Bouizi

Ghislain Lartigue



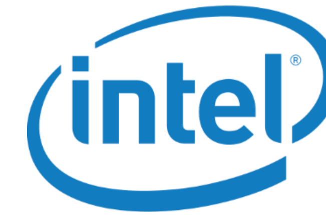
Innovation and Pathfinding Architecture
Group in Europe, Exascale Lab. Paris

CRIHAN - Demi-journée calcul intensif, 16 juin 2015

Outline

- Presentation of the Exascale Lab Paris and related work with partners
- Presentation of YALES2
- Status of the code on the Intel Architectures
- Motivations of the study
- Presentation of the results on Sandy Bridge & KNC
- Future work

The Exascale Lab Paris



IPAG Europe Locations

Research areas:

- Scientific HPC applications:
  - Molecular dynamics
  - CFD
  - Geophysics
  - Climate Science
  - Fusion
- Performance Analysis Tools
  - Static analysis
  - Dynamic analysis
  - Performance prediction
  - Optimization projection

Few words on Xeon & Xeon Phi

	XEON - HSW	XEON PHI - KNC	XEON PHI - KNL
Cores	18+	-61	60+
Freq.	2-4GHz	-1.1GHz	TBA
Mem.	-6TB	-16GB	-384GB
Mem. Bandwidth	55GB/s	180GB/s	400+GB/s
Cache L2	256kB	512kB	512kB
Cache L3	30MB		-16GB integrated on package memory
FLOPs (double)	662GFlops	1TFlops	3+TFlops
Instructions set	AVX2	KNC instructions	MIC-AVX512
Flop/Byte (double)	12	5.5	(7.5)

The YALES2 solver

YALES2

www.coria-cfd.fr

- YALES2 is an unstructured low-Mach number code for the DNS and LES of reacting two-phase flows in complex geometries.

- It solves the unsteady 3D Navier-Stokes equations on massively parallel machines

- It is used by more than 160 people in labs and in the industry
  - SUCCESS scientific group (http://success.coria-cfd.fr):
    - CORIA, I3M, LEGI, EM2C, IMFT, CERFACS, IFP-EN, LMA
  - Other labs: ULB, MONS, LOMC, …
  - Collaboration with INTEL/CEA/GENCI/UVSQ Exascale Lab
  - Industry: SAFRAN, RHODIA (SOLVAY), AREVA, …

- Awards
  - 2011 IBM faculty award
  - 3rd of the Bull-Joseph Fourier prize in 2009
  - Principal investigator of 2 PRACE proposals

YALES2

www.coria-cfd.fr

Mesh Management
- 1D, 2D, 3D
- Partitioning
- Load balancing
- Refinement

IOs
- Gambit/Fluent
- Partitioned HDF5
- Ensight
- AVBP

Linear Solvers
- Deflated PCG
- BICGSTAB2
- Residual recycling

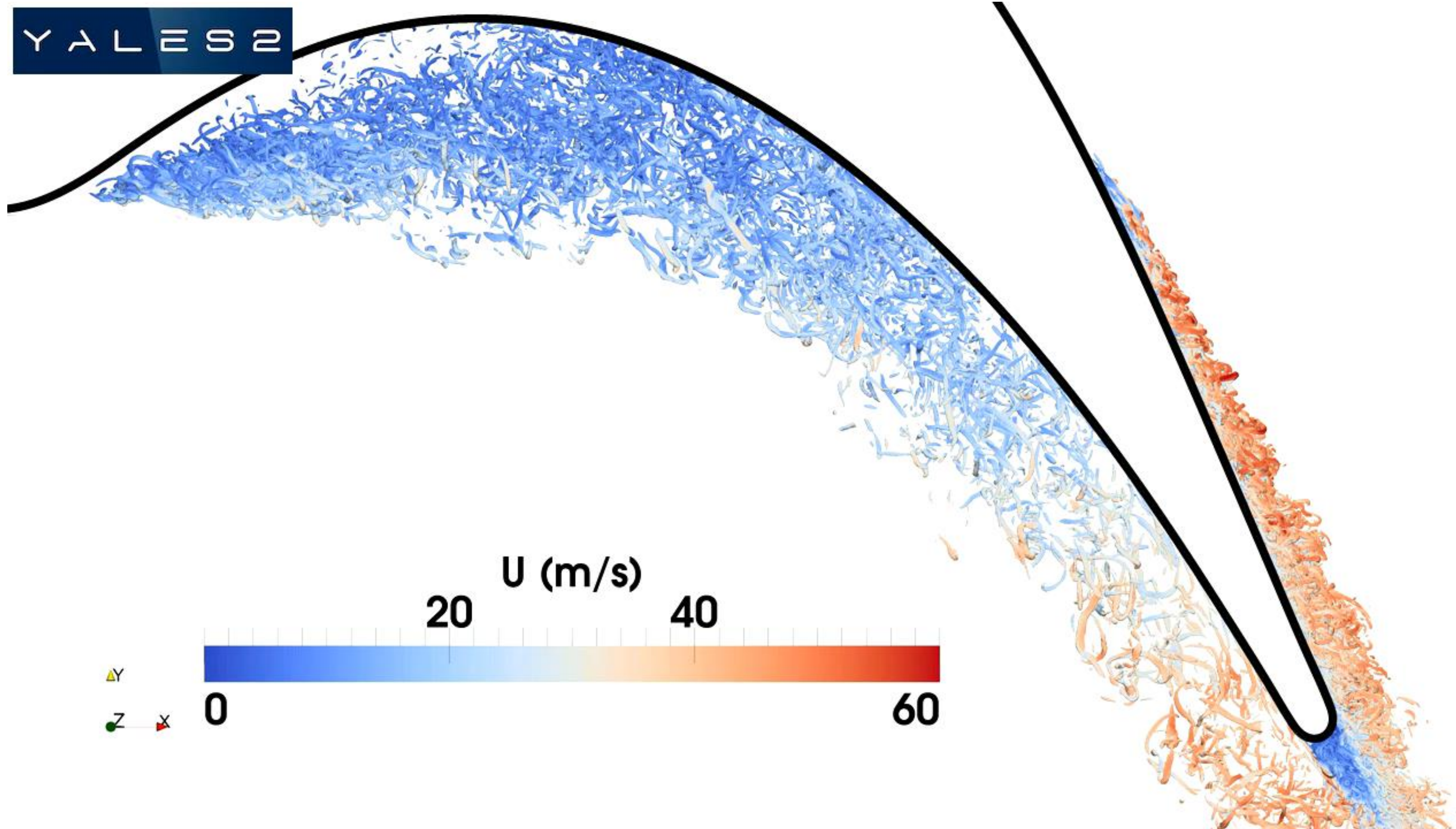
YALES2 Solvers

Complex Chemistry
- Tabulated chemistry
- Complex chemistry
- Stiff integrators
- Dynamic load balancing

Data analysis
- Probes
- Postproc. variables
- High-order filters
- FFT, POD, DMD
- Statistics

Numerics
- Particles
- Level sets
- 4-th order FV schemes

- 2 main maintainers
  - V. Moureau
  - G. Lartigue
- 300 000 lines of object-oriented F90
- Git version management
- www.coria-cfd.fr
- Portable on all the major platforms

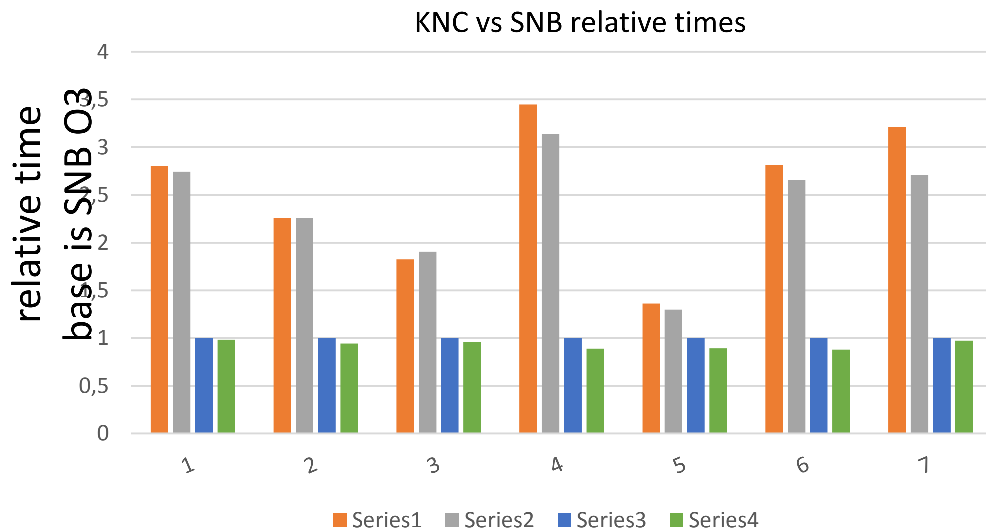


Turbine blade – Re=150’000 – 18 billions cells – 8’192 cores

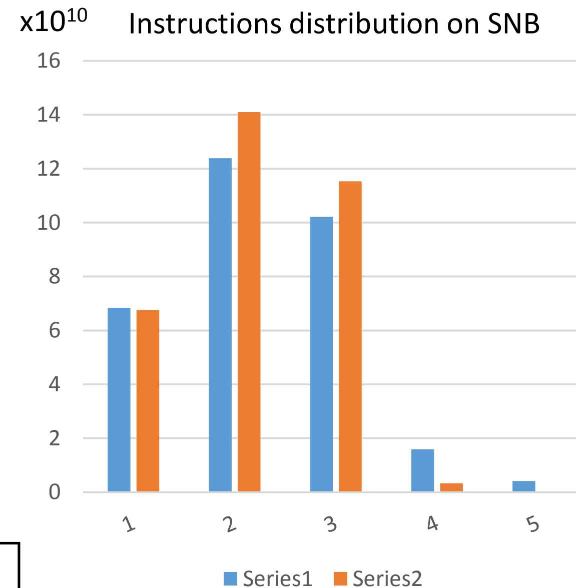
Status of the code on the Intel Architectures

Performance within one node of SNB (2x8 cores @2.6GHz) & KNC (61 cores @ 1.1GHz)

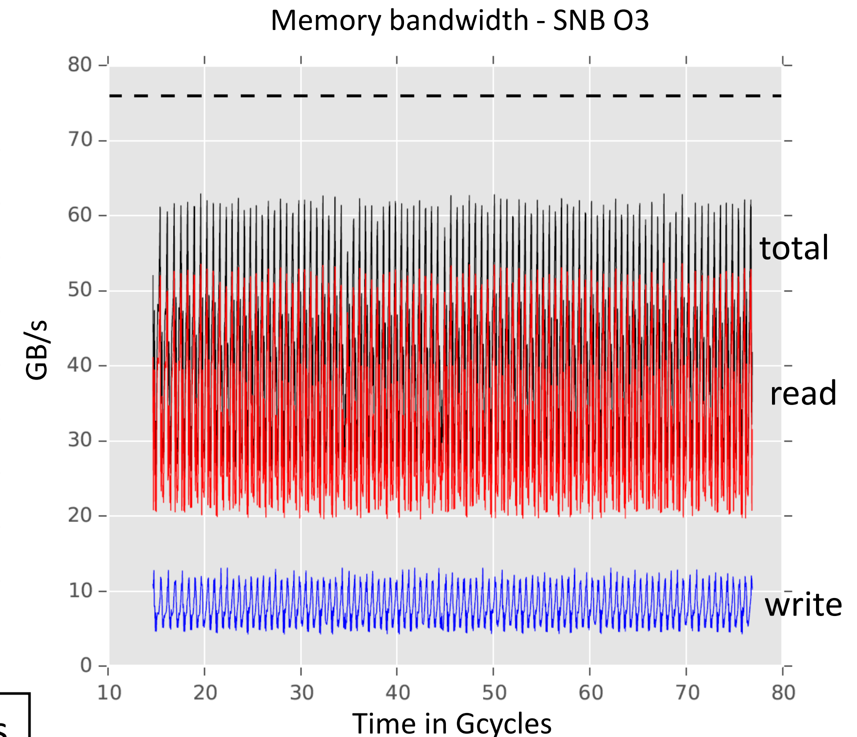
Dataset preccinsta 1.7M – 2GB mem. footprint – 100 iterations



No gain with vectorization (limited by the bandwidth, lot of indirections)
KNC is ~2.5x slower than SNB – ! But amount of work on each core of the KNC is lower than on SNB (relevance of the dataset?). On KNC, Front End decodes 1 instruction every 2 cycles.



a lot of arithmetic instructions but very few are vectorized. GPR instructions are ~50% of “operations”. Max. gain of vectorization is ~50%



Peak bandwidth is high. The code is bandwidth sensitive.

Wrap up

- Both -O3 vectorized and not vectorized have the same performance. Autovectorization generates few vector instructions.
- In the Temporal Loop, the most of the time is spent in the Poisson solver

How to execute faster the Poisson solver ?

- Vectorizing the code can be a solution
- Changing the algorithm
- Reorganizing the data

Poisson solver overview

- Low-Mach formulation of NS ➔ Poisson equation on pressure

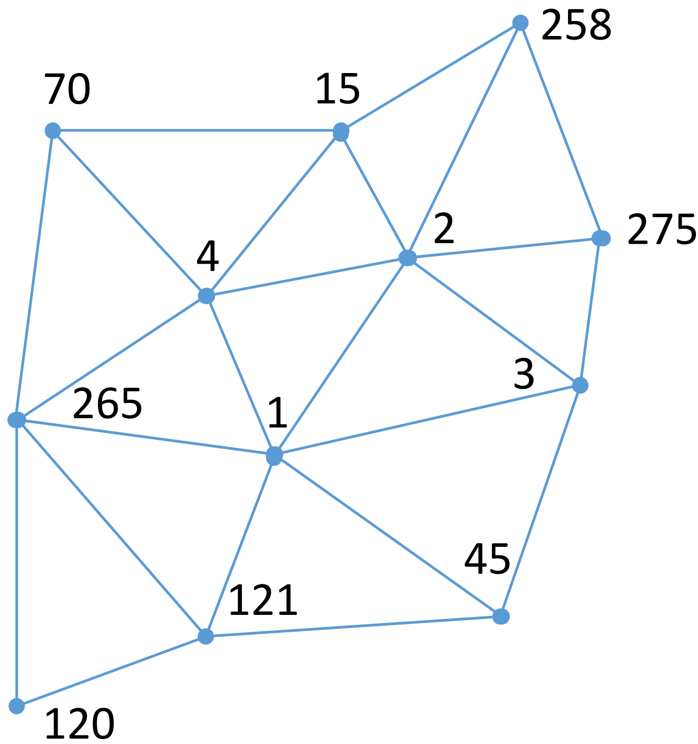
$$\Delta P = \boldsymbol{\nabla} \cdot \mathbf{u}^*$$

- Finite volume centered scheme transforms the Laplacian differential operator in a symmetric matrix and the diff. eqn. in linear algebra

$$Ax = b$$

- Each element of A represents the contribution of pairs of nodes of the mesh
- Connectivity of the mesh is such that each nodes has ~20 neighbours
- This sparse system is solved with CG based iterative methods

Focus on the Poisson solver loop



Mesh sample

Pairs	
Node 1	Node 2
1	2
1	3
1	4
1	45
1	121
1	265
2	3
2	4
2	15
2	258
2	275
3	23
3	45
3	59
3	70
3	71
3	275
…	

Connectivity description

```
! for each cell group
do i = 1, size_outer
   elt => elts(i)
   pair2node1 => elt%ind1%val
   pair2node2 => elt%ind2%val
   sym_op     => elt%sym_op%val
   data       => elt%data%val
   laplacian  => elt%laplacian%val
   laplacian(1:el_grp%nnode) = 0.0
   ! for each pair
   do ip=1,el_grp%npair
      ino1 = pair2node1(ip)
      ino2 = pair2node2(ip)
      coeff = sym_op(ip)*(data(ino2)-data(ino1))
      laplacian(ino1) = laplacian(ino1) + coeff
      laplacian(ino2) = laplacian(ino2) - coeff
   end do
end do
```

Non vectorizable due to dependencies

Improvements of the loop – focus on data

Pairs	
Node 1	Node 2
1	2
1	3
1	4
1	45
1	121
1	265
2	3
2	4
2	15
2	258
2	275
3	23
3	45
3	59
3	70
3	71
3	275
…	

v.1
not vectorizable

Remove the data dependencies

Pairs	
Node 1	Node 2
1	2
	3
	4
	45
	121
	265
2	3
	4
	15
	258
	275
3	23
	45
	59
	70
	71
	275
…	

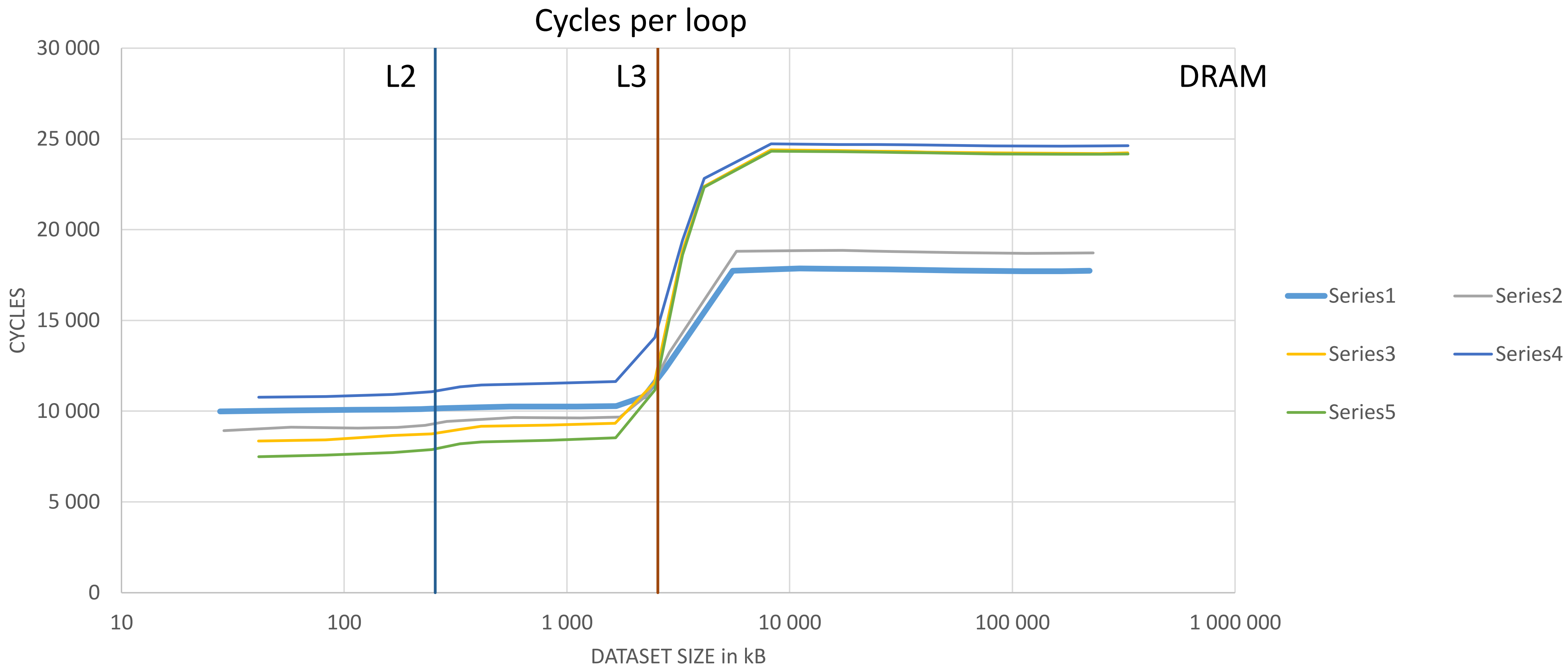
v.2
not vectorizable (indirections)

Gather the groups of the same size to change the direction of the vectorization and reduce the effect of the tail loop.

Node 1	Node 2					
228	139	140	154	182	205	243
242	166	167	170	181	195	231
245	157	188	189	246	247	248
249	187	190	247	248	255	256
273	132	193	207	225	241	272
8	5	6	7	161	257	278
21	19	20	22	24	25	278
111	9	10	96	110	171	278
161	7	8	103	160	257	
200	40	150	169	175	233	
28	1	4	27	29	278	
97	10	12	85	96	278	
131	6	17	99	130		
254	156	157	227	246		
261	62	86	259	260		
264	46	60	258	263		

v.3
vectorizable
2x operations than v.1

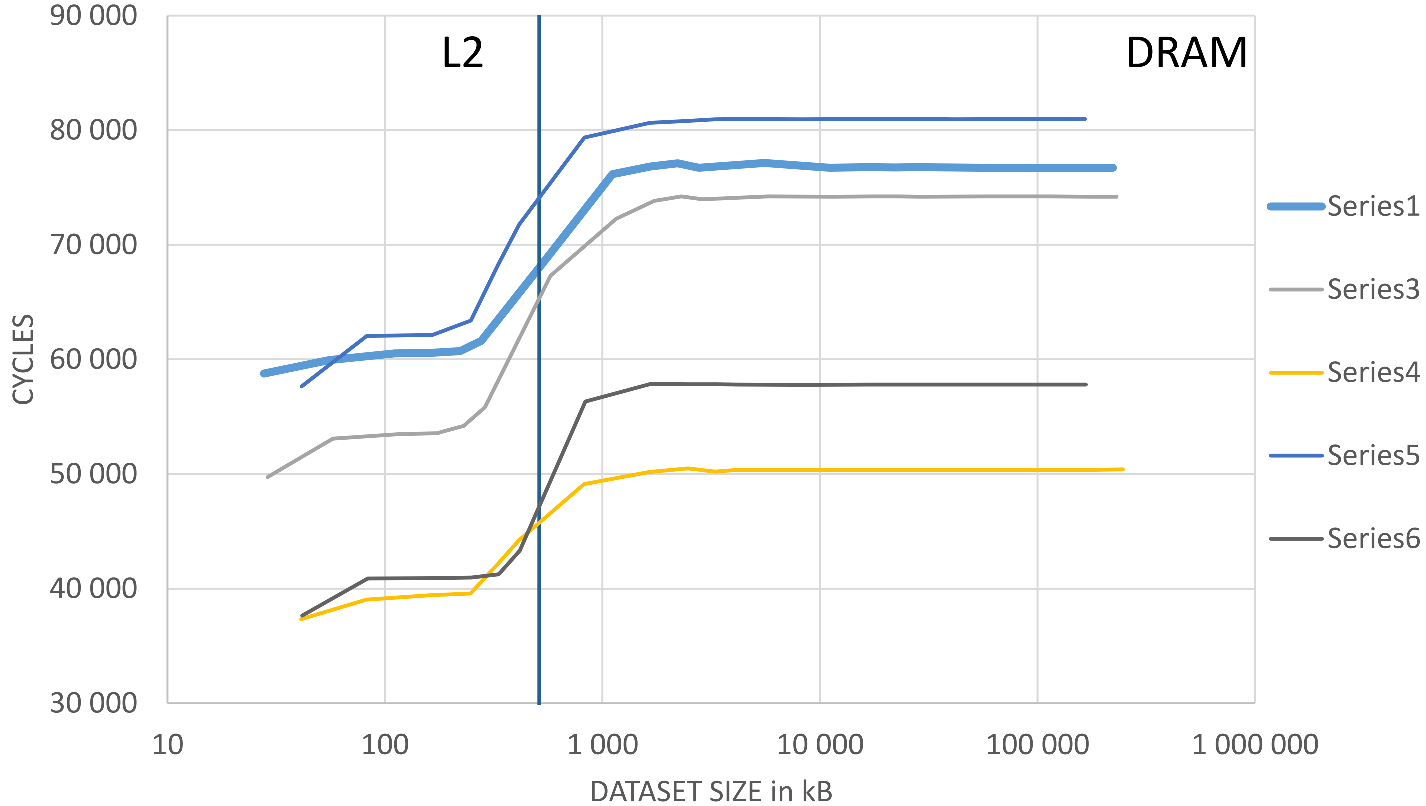
Behavior of the loops on Xeon SNB

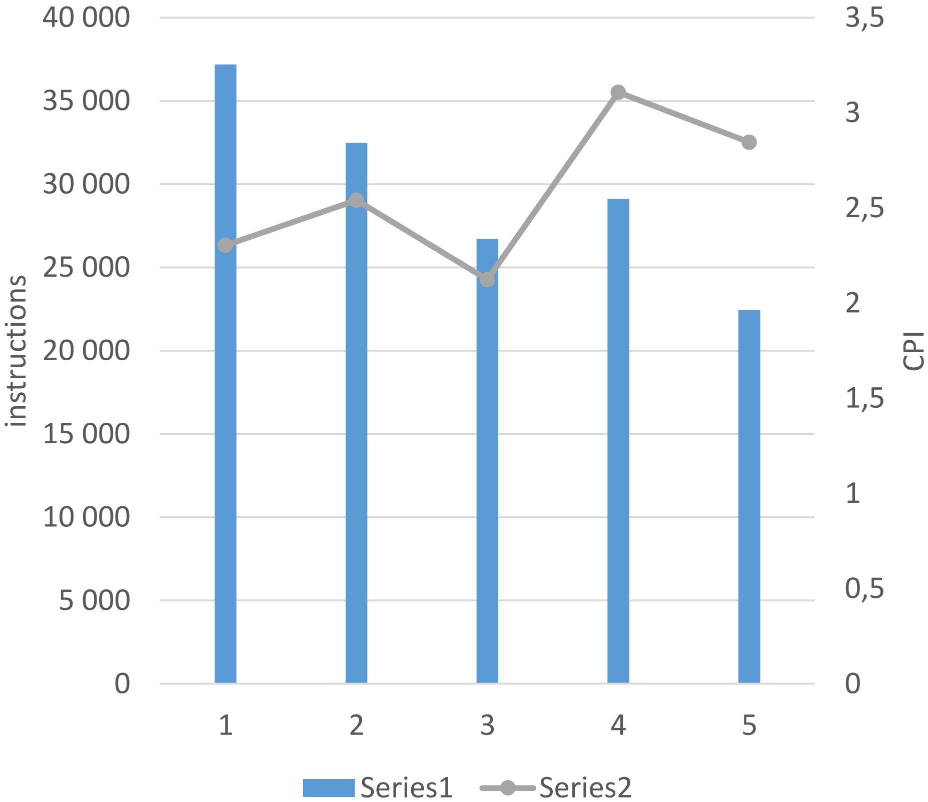


v.2 & v.3 in F90 & C are not vectorized by the compiler

Original version is faster when data comes from the DRAM because it has the minimal memory accesses

Behavior of the loops on Xeon Phi KNC

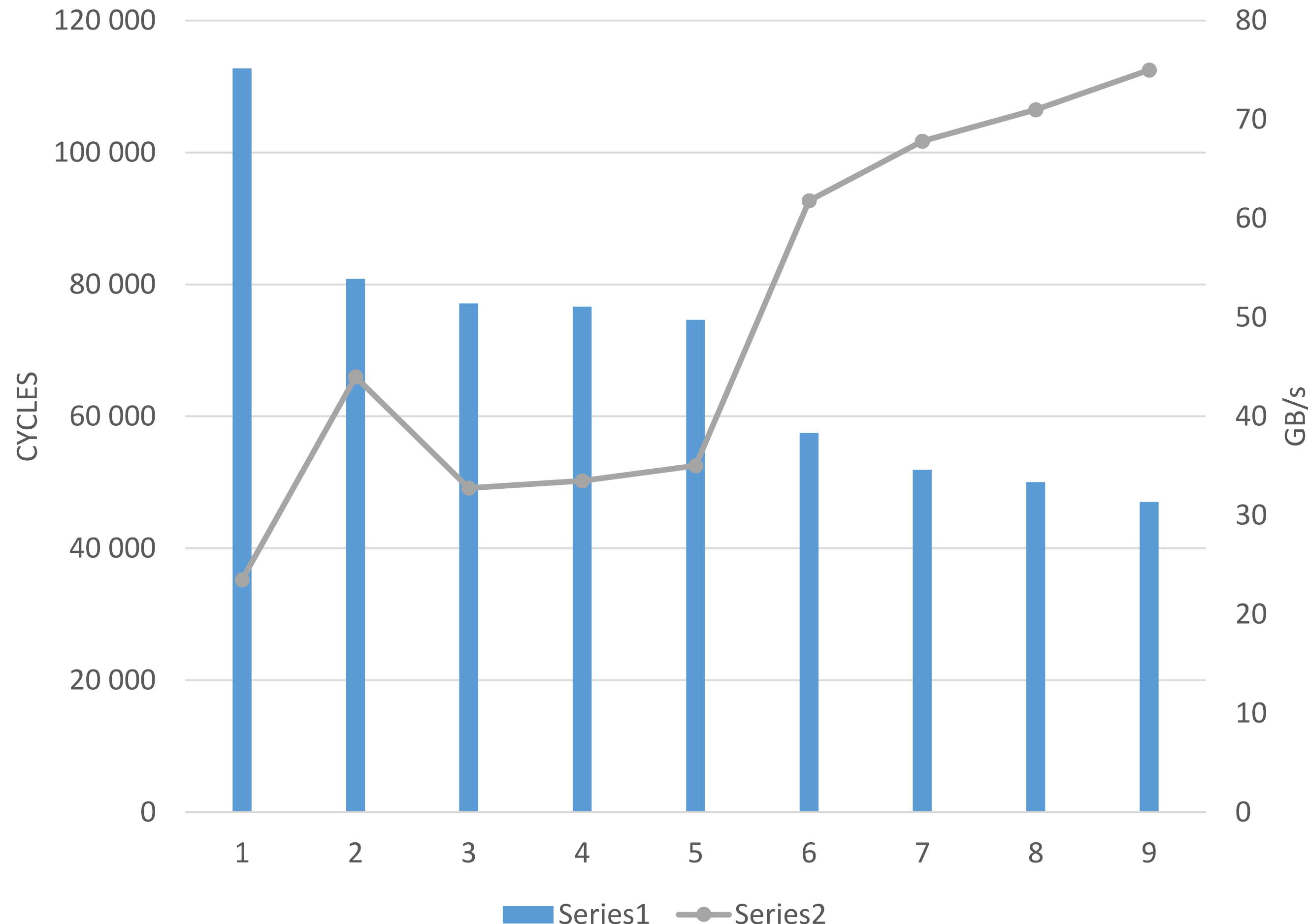




v.2 & v.3 in F90 are not vectorized by the compiler, v.3 C has #pragma simd
v.3 in F90 version is faster, perhaps because of the overlap of the computation and the software prefetcher

Cycles per instructions is between 2 & 3. Even if the fully vectorized loop v.3 C executes less instructions, they take more time to be executed

Behavior of the loops on Xeon Phi KNC – bandwidth & cycles



Cycles per loop for dataset in DRAM

- When the measured bandwidth increases, the cycles per loop decreases.
- Non vectorized versions are faster than vectorized version, due to latency vs prefetching ?
- ! KNC is in order and decodes 1 instruction every 2 cycles for 1 thread.

Future work

Improve the memory access with:

- prefetching (only if not at the maximum bandwidth)
- reducing the data dispersion in the cache lines by testing 2 strategies:
  - Touching the least cache lines such that the prefetcher is not stressed
  - Touching as much as possible cache lines at the beginning of the loop such that the arrays come in the L2 cache
- reorganize the code to compute everything on one group of nodes (very hard)
- Focus on the vectorization of loop v.2

Thank You