



Cluster performance, how to get the most out of Abel

Ole W. Saastad, Dr.Scient

USIT / UAV / FI / FT

April 6th 2016

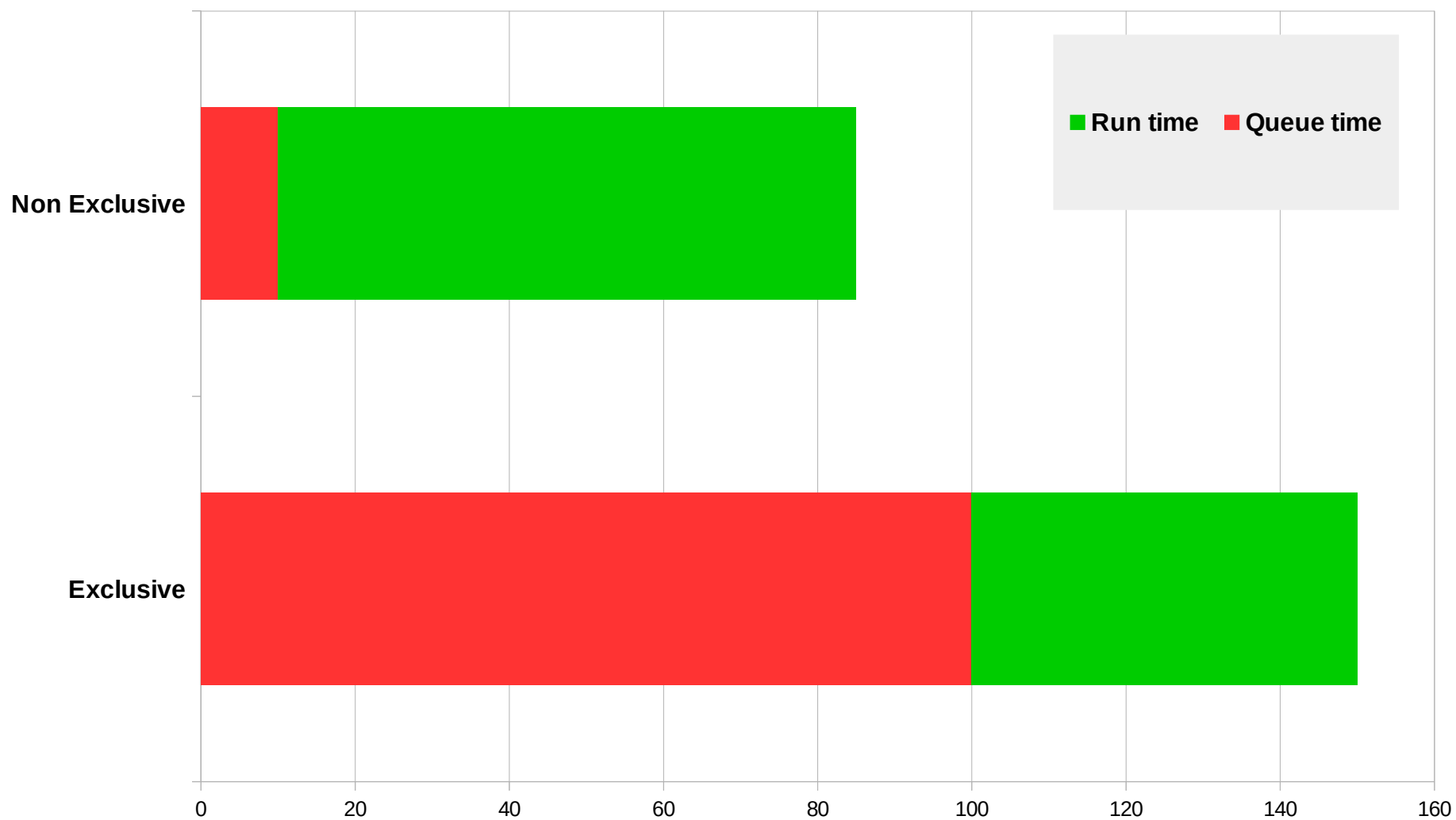


Introduction

- Smarte queueing
- Smart job setup
- Scaling
- Effective IO
- Big data handling
- Compiler usage
- Programming for the future



Smart queueing



Smart queueing

- Exclusive or none-exclusive ?
- Exclusive
 - `--ntasks-per-node=16` (or 20 for rack 31)
 - `--nodes=n`
- None-exclusive
 - `--ntasks=m`



Smart job setup

- Ask only for what you really need
- Stage data smart
- Carefully consider the IO load
- Have you control over scaling and input ?
- What happen if your job crashes or is killed ?
- What about log files ?



Ask only for what you need

- Queue system allocate and prioritise based on your footprint
- Footprint is based on
 - number of cores/nodes
 - Memory
 - Wall time
- Large footprint means longer wait time in queue



Stage data smart

- /work is faster and larger than /cluster
- There is NO backup on /work
- \$SCRATCH is on /work
- \$SCRATCH is deleted when job is finished
- /work/users/<your user name> is available
 - Files are kept for 45 days, no backup
- /project is slow, stage files on /work



Carefully consider the IO load

- /project is slow and not part of Abel
- /cluster is backed up and high performance
- /work is not backed up and has even higher performance
- Backup cannot tackle millions of files
- Small log files are ok on /cluster/home
- Large files should be on /work
 - \$SCRATCH for temporary scratch files
 - /work/user/<username> for log files
- Files on /work/users are **deleted** after 45 days
 - Touching the files does not prevent this !

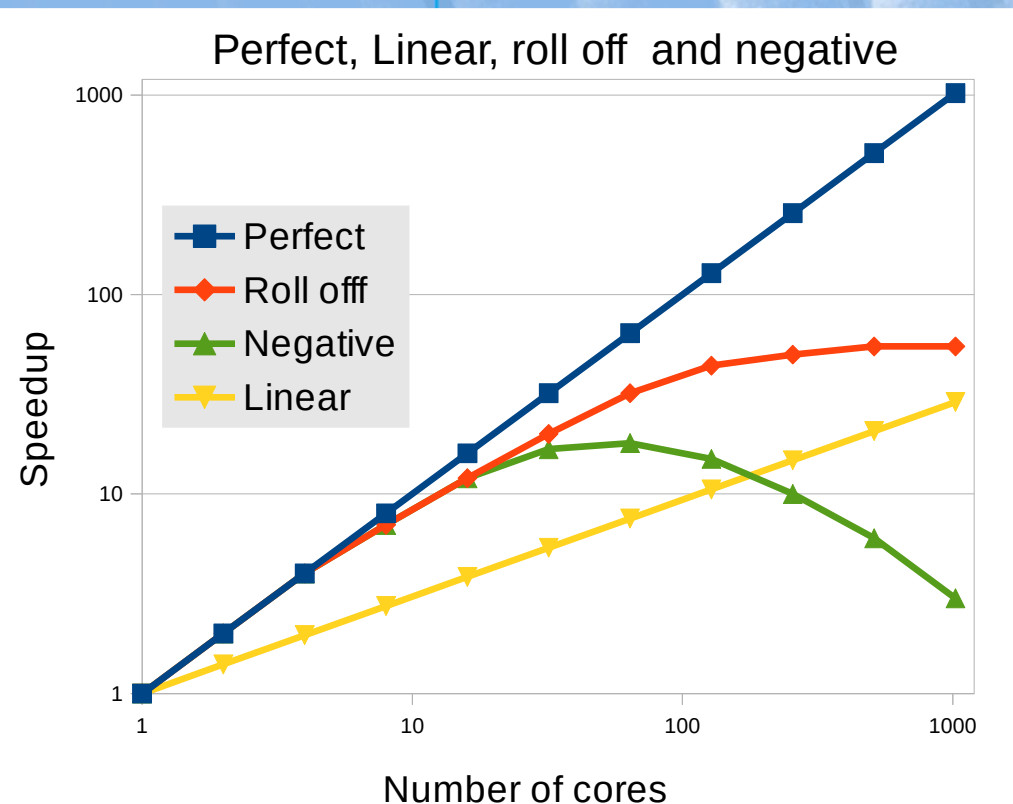


Carefully consider the IO load

- /cluster and /work file system are unhappy with millions of small files
- If you use millions of temporary files ask for local scratch named \$LOCALTMP
 - --gres=localtmp:x (x is in GiB)
- Backup system are unhappy with millions of small files
 - Use a directory called nobackup or /work

Have you control over scaling and input ?

- Do your application scale ?
- Does it scale to the core count you request?
- Does it scale with the given input ?
- OpenMP threads
- MPI ranks
- Hybrid (multi threaded MPI executables)





What happen if your job crashes or is killed ?

- If you application crashes everything works as if it terminated normally
- If your job is killed (memory, time limit etc)
 - `chkfile <file>` work as expected
 - Directory `$SCRATCH` is removed
 - Files on `$SCRATCH` are lost



What about log files ?

- Log files can reside on /cluster/home or /work
 - No need to use \$SCRATCH for them
 - Logs of reasonable size on \$SUBMITDIR
- If log files are written to \$SCRATCH remember to use chkfile
- Log files are required when reporting problems (RT)



Effective IO

- How much IO is your application doing ?
- Few large files or millions of small files ?
- Where are the files placed ?
- Copying of files
- Archiving many files to one large
- Do the files compress ?
 - Not all files compress well



Effective IO

- /work and /cluster are global parallel file systems, BeeGFS.
- In order to make it coherent locks and tokens are needed.
- Any operation must be locked to prevent corruption if more than one writer want to update a file
- All parallel file systems struggle to cope with huge amount of metadata updates.
- They love few large files !



Effective IO

- Use \$SCRATCH or /work/users/ during runs
- Use \$LOCALTMP for millions of files
 - Make a tar archive of these before moving them to /work or /cluster
- Stage files from /projects to /work/users
 - /project has limited bandwidth



Big data handling

- Few large files or millions of small files ?
- BeeGFS can handle large files easy
 - Bandwidth like 8-10 GiB/s
 - /work have higher bandwidth than /cluster
- Use infiniband if possible
 - `scp file.ext <host ib ip number>:`
 - Doubles the transfer speed



Big data handling

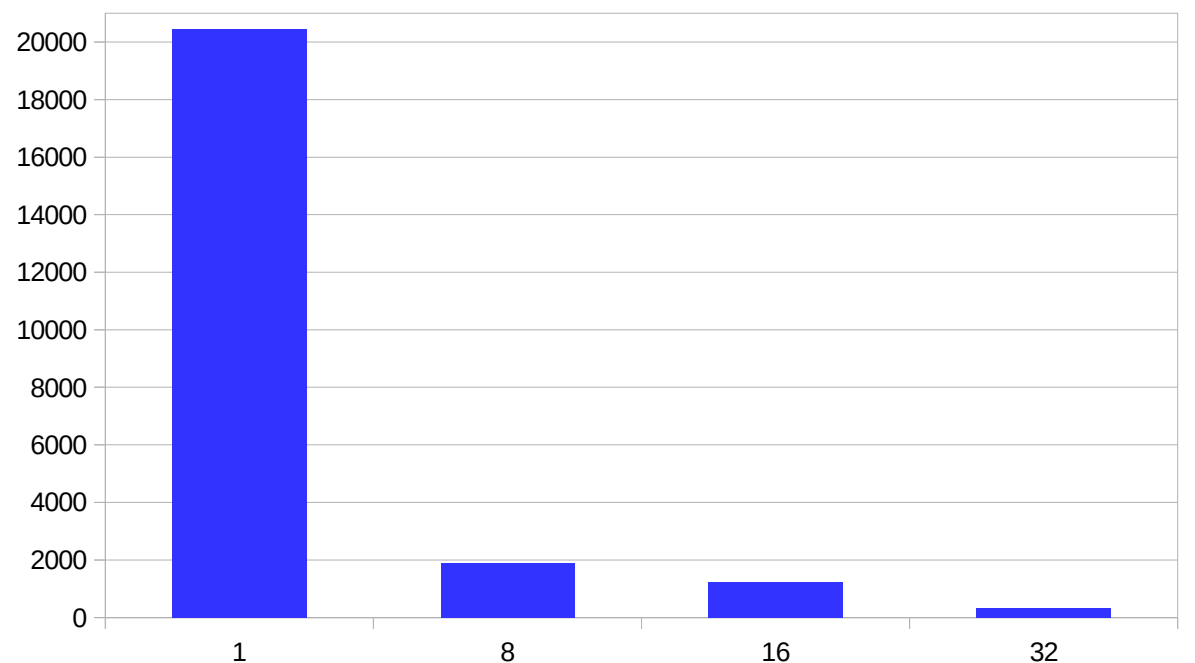
- Many small files are problematic
- Backup system struggle with millions of files
- BeeGFS struggle with metadata of millions of entries
- Make archives of directories with large number of files
- Use local scratch or temporal storage if possible
 - These sizes are relatively small
 - Tar directory to a single large file
- Mount an image on /work as a file system
 - Require help from support team

Big data handling

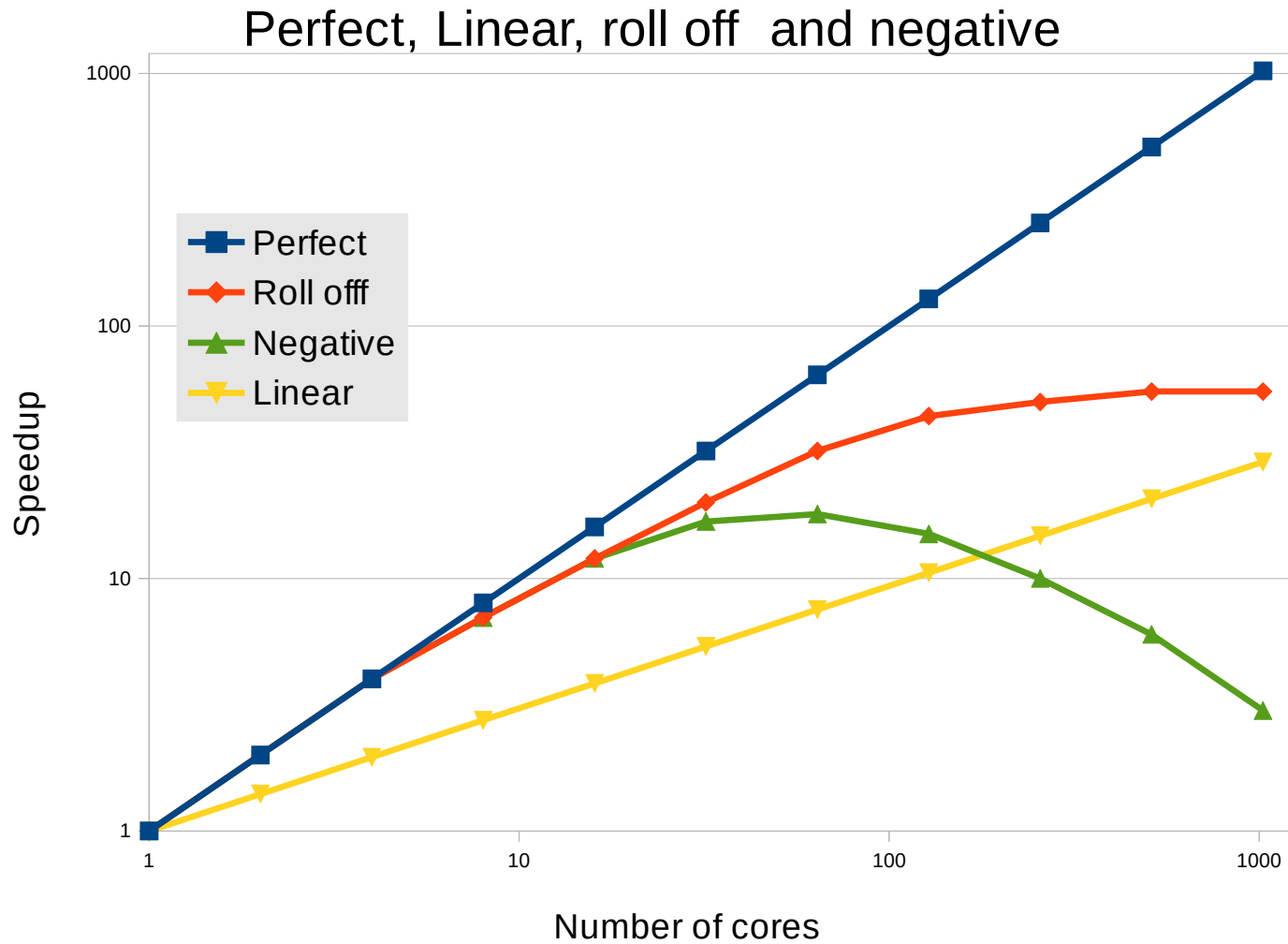
- Copying of millions of files are slow
- Rsync may take hours
- Option to use parallel rsync

1: rsync

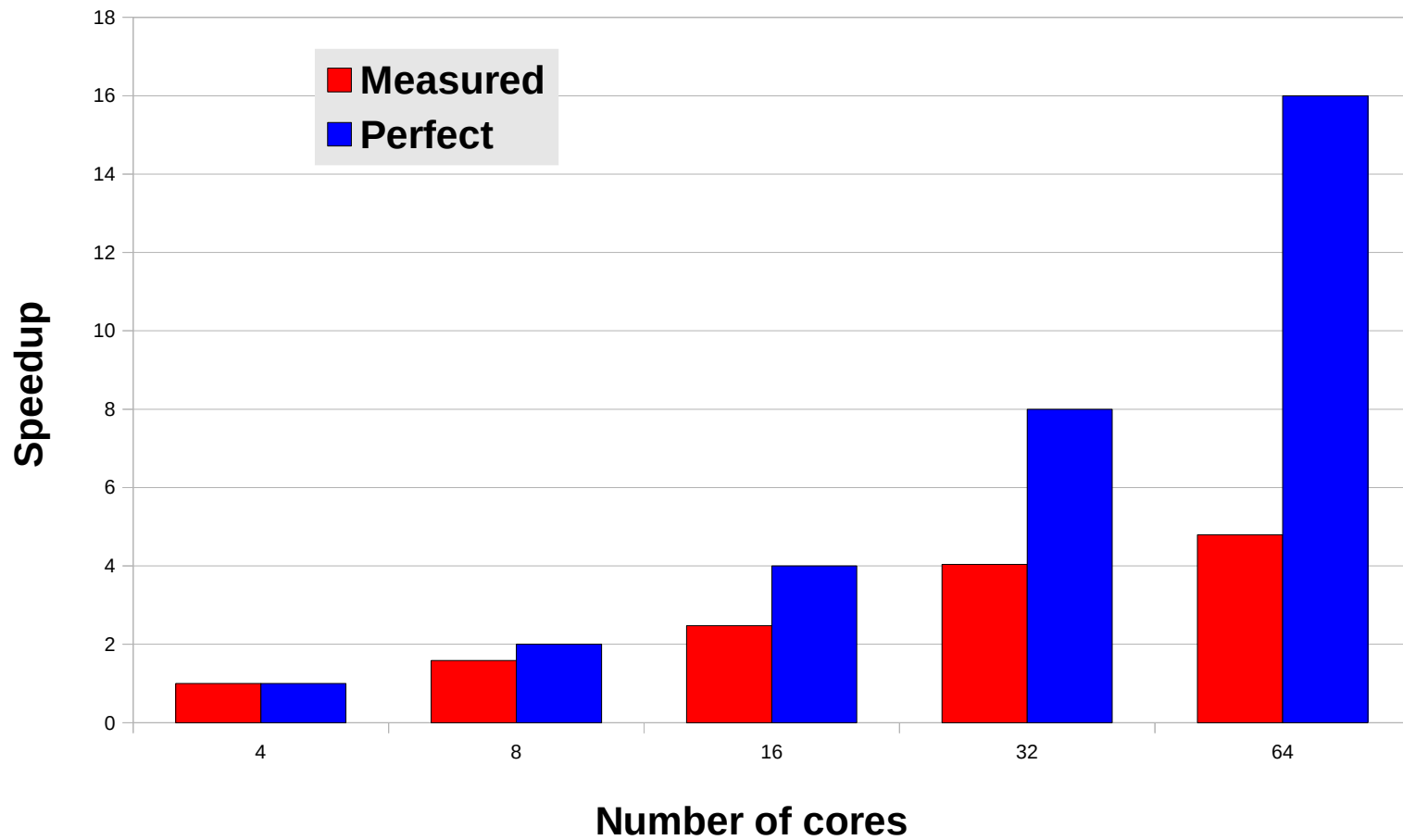
2-32 : parsync



Scaling



Scaling



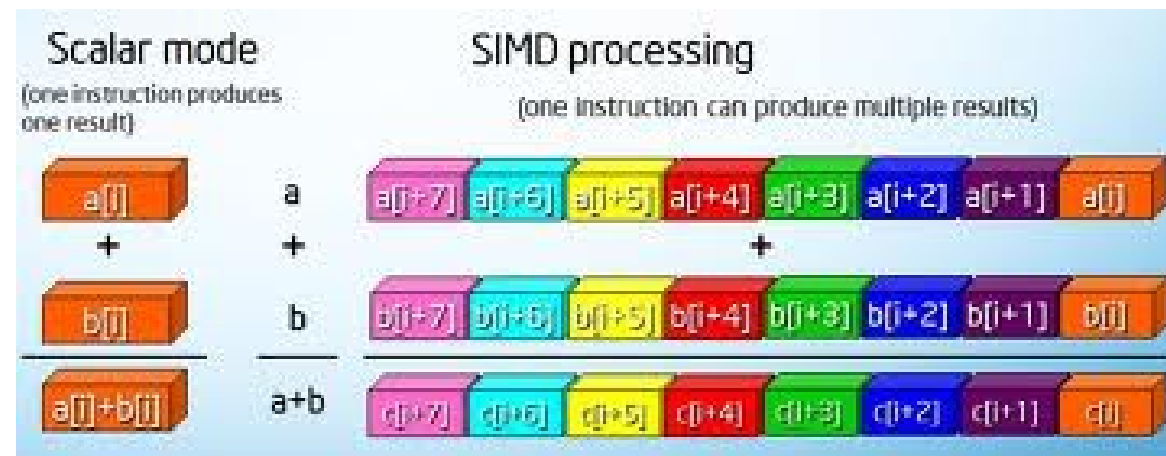


Scaling

- How does your application scale ?
- Is it dependent on input ?
- Are you running at the sweet spot ?
- Threads ?
- MPI ?
- Hybrid ?
- Will your application scale to more than 120 threads to use the new Intel Knights Landing processor ?

Compiler usage

- Parallel – 16 to 20 cores in a node
 - Vectorisation
 - OpenMP
 - MPI
- More and more focus on vectorisation



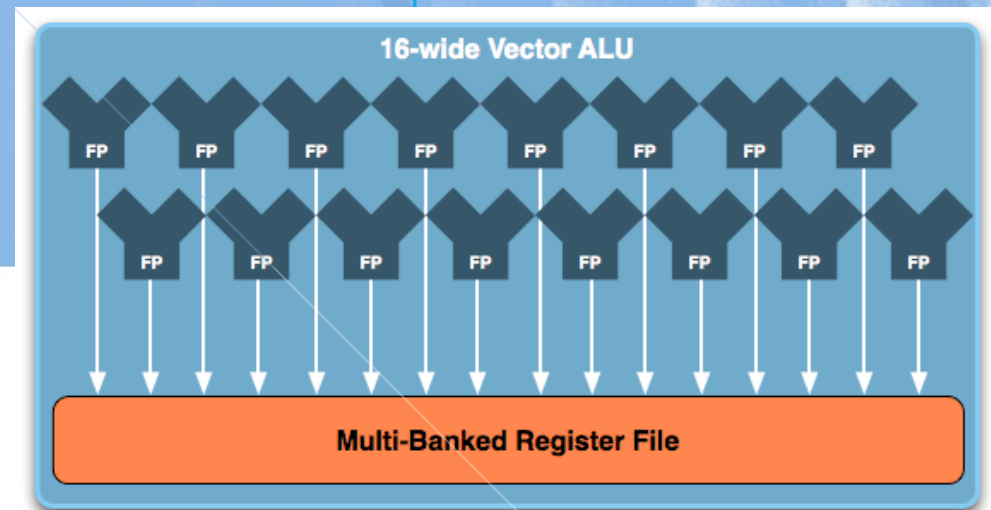


Programming for the future

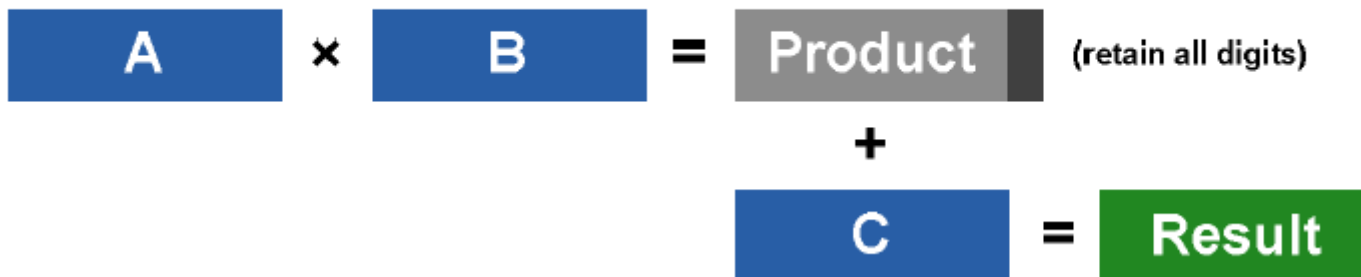
- Intel Many Integrated Cores, MIC
 - Knight Corner (KC) installed
 - Knights Landing (KL) later 2016
 - KC : 1 Tflops/s
 - KL : 3 Tflops/s
 - Same architecture x86-64 (Silvermont core)
 - Same compilers
 - Same libraries

Programming for the future

- Intel Knight Landing 72 cores and 288 threads
 - Need at least 144 threads → OpenMP+MPI
 - 512 bits vector unit
 - Fused Multiply add



Fused Multiply-Add (FMA)





Programming for the future

- Pure MPI may yield too many ranks
 - Large memory demand
- Pure threads may scale weakly
 - OpenMP scaling is often limited
- Hybrid multi threaded MPI

- One MPI rank per core, 2-4 threads per core



Programming for the future

- Vectorization vital for performance
 - Scalar code perform at 1/16 performance
- OpenMP 4.0 introduces SIMD directives
- All OpenMP programming facilities now apply to SIMD
- Vectorization is made easier with OpenMP, just like threading was made simpler.



Programming for the future

- OpenMP SIMD - vectorization
- Single program, multiple data
- Vector units are becoming increasingly more important
- AVX / AVX2 256 bits = 4 dp and 8 sp floats
- MIC has 512 bits vector = 8 dp and 16 sp floats
- Vector fused multiply add double vector performance

Programming – OpenMP 4.0

- OpenMP 4.0 SIMD

```
!$omp simd reduction(+:sum)
do i = 1,n
  x = h*(i-0.5)
  sum = sum + f(x)
enddo
pi = h*sum
```

Programming - OpenMP 4.0

- OpenMP 4.0 SIMD

```
!$omp simd reduction(+:sum)
do i = 1,n
  x = h*(i-0.5)
  sum = sum + f(x)
enddo
pi = h*sum
```

Programming – OpenMP 4.0

- OpenMP 4.0 SIMD

```
module func_f
  contains
!$omp declare simd f
  function f(x)
    real(8) :: f
    real(8) :: x
    f = (4/(1+x*x))
  end function f
end module func_f
```



Programming - Intel tuning tools

- Tuning tools
- Compiler – *only time the source code is read*
- MPI library – *can provide vital statistics*
- Vector Advisor - *vectorization*
- Thread Inspector - *Threads*
- Vtune Amplifier – *CPU / memory performance*
- Trace Analyser – *MPI calls*



Programming - Intel tuning tools

- Compiler – *only time the source code is read*
- `-qopt-report5 -qopt-report-phase=vec`
- Read the `*.optrpt` files !



Programming - Intel tuning tools

- Vector Advisor
- Checks the program for vectorization
- Checks performance scalar vs. vector
- Checks loop performance



Intel XE-Advisor

Where should I add vectorization and/or threading parallelism? Intel Advisor XE 2016

Elapsed time: 56.36s Vectorized Not Vectorized FILTER: All Modules All Sources

Summary Survey Report Refinement Reports Annotation Report

Loops	Vector Issues	Self Time	Total Time	Loop Type	Why No Vectorization?	Vectorized Loops				Instru
						Vector...	Efficiency	Gain E...	VL (V...	
[loop in intcubelayer_2d at radiation_GS_mpi.f90:...	2 Possible i...	3.159s	11.126s	Vectorized (Body; ...		AVX2	~13%	1.02x	8	Blend
[loop in intlayerlayer_2dlin at radiation_GS_mpi.f9...		0.770s	0.770s	Scalar	vectorization possible but ...				4	Divisio
[loop in intcubelayer_2d at radiation_GS_mpi.f90:...	2 Possible i...	0.400s	1.370s	Vectorized (Body; ...		AVX2	~13%	1.02x	8	Blend
[loop in intcubelayer_2d at radiation_GS_mpi.f90:...	2 Possible i...	0.360s	1.000s	Vectorized (Body; ...		AVX2	~14%	1.08x	8	Blend
[loop in compute_si_layer at radiation_GS_m...	1 Ineffectiv...	0.330s	0.330s	Vectorized (Body;...		AVX2	~70%	5.57x	4; 8	FMA
[loop in intlayerlayer_2dlin at radiation_GS_mpi.f9...		0.330s	0.330s	Scalar	vectorization possible but ...				4	Divisio
[loop in lambda_diagonal_layer at radiation_GS_...	2 Possible i...	0.330s	0.910s	Vectorized (Body; ...		AVX2	~13%	1.04x	8	Blend
[loop in quenchy at quench3_mpi.f90:159]		0.300s	0.300s	Vectorized (Body)		AVX2	~61%	4.86x	8	Divisio
[loop in quenchx at quench3_mpi.f90:113]		0.220s	0.220s	Vectorized (Body; ...		AVX2	~93%	7.45x	8	Divisio
[loop in inteartion_constants at radiation_GS_m...		0.200s	1.510s	Scalar	vectorization possible but ...				4	

Source Top Down Loop Assembly Recommendations Compiler Diagnostic Details

File: radiation_GS_mpi.f90:3653 compute_si_layer

Line	Source	Total Time	%	Loop Time	%	Traits
3646	! Higher-order interpolation constants					
3647	real :: u, v, alz, a2z, a3z, dz1, dz2, alpha1, alpha2, a1, a2, a3, a4, d1, d2, d1md2					
3648	real :: df, df1, df2					
3649	! Helper array					
3650	real, dimension(xsbr:xebr, ysbr:yebr) :: scr					
3651						
3652	! Compute the source function integral					
3653	SIlocal=psilayer(xs:xe,ys:ye,1)*Slayerup+ &	0.360s		0.710s		FMA
3654	& psilayer(xs:xe,ys:ye,2)*S(xs:xe,ys:ye,k,bin)+ &	0.020s				
3655	& psilayer(xs:xe,ys:ye,3)*Slayerdn					
3656						
3657	! Fill intensities from neighbor subdomains into the ghost zones, except					
3658	! for 2D case					
3659	if(mx.ne.1)then					

Selected (Total Time): 0.360s



Programming - Intel tuning tools

- XE thread Inspector
- Checks the theading in the program
- Checks the loops and threads



Intel XE-Inspector

The screenshot displays the Intel XE-Inspector interface for a project named 'HYDRO'. The main window title is '/usit/abel/u1/olews/intel/inspxe/projects/HYDRO - Intel Inspector (on compute-31-19.local)'. The interface is divided into several panes:

- Project Navigator:** Shows a tree view of the project structure under 'HYDRO', listing sub-projects r000mi3 through r005ti3.
- Locate Deadlocks and Data Races:** The main analysis pane, currently showing 'Problems'. It contains a table of detected data races:

ID	Type	Sources	Modules	State
P1	Data race	main.f90	hydro	New
P2	Data race	module_hydro_principal.f90	hydro	New
P3	Data race	module_hydro_principal.f90	hydro	New

Below the table, there are filter sections for Severity, Type, Source, Module, State, Suppressed, and Investigated, each showing the count of items matching the filter.

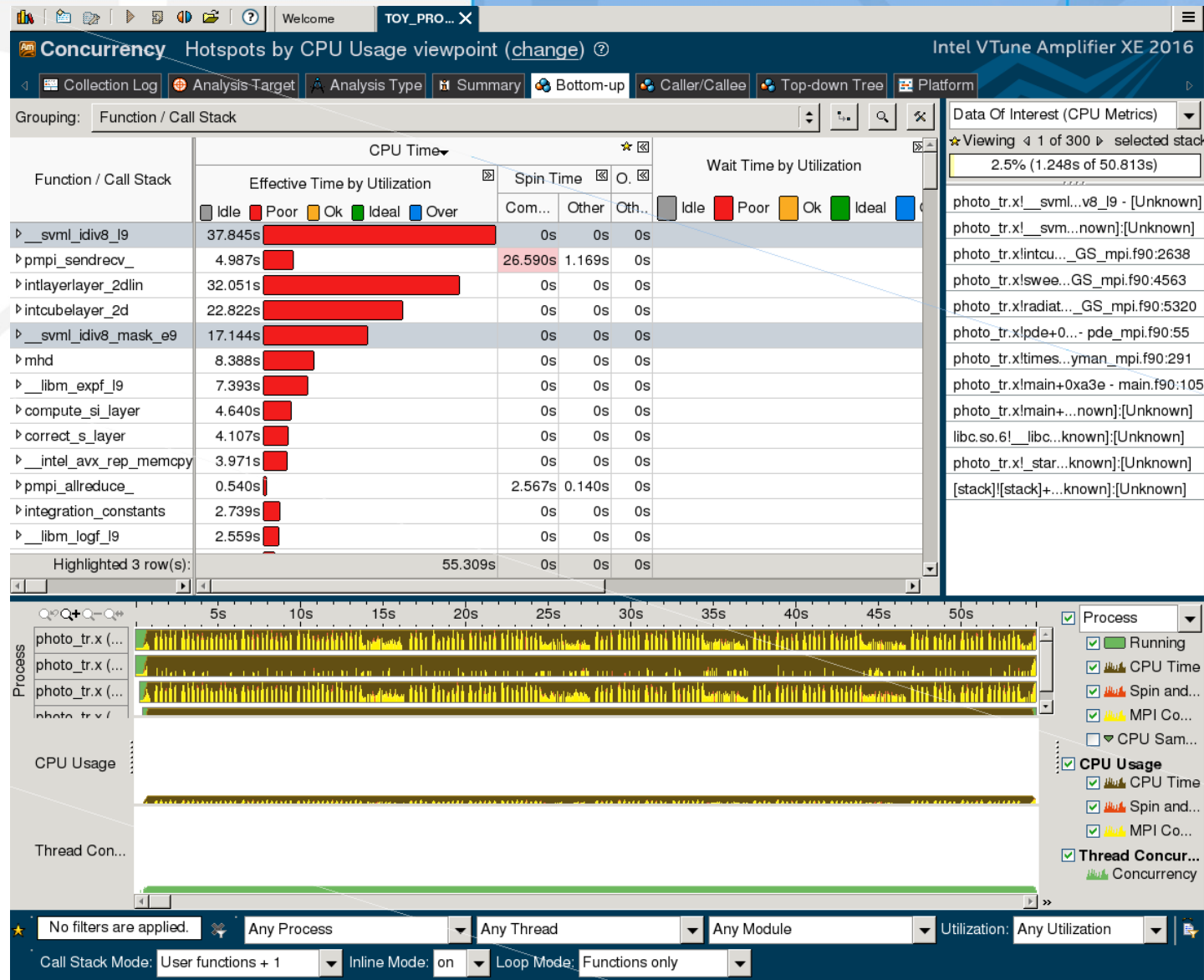
- Code Locations: Data race:** Shows the source code for the detected data race. It displays two identical code snippets for 'main.f90:29 MAIN_\$omp\$parallel@28 hydro'. The code includes OMP directives and a print statement.
- Timeline:** Shows a visual representation of the execution timeline, with two threads highlighted: 'OMP Worker Thread #10 (15249)' and 'OMP Worker Thread #11 (15250)'.



Programming - Intel tuning tools

- XE Vtune Amplifier
- Checks the CPU performance
- Originally a tool for CPU design
- Provide a large range of metrics
- Instructions, cache, memory usage

Intel Vtune-Amplifier

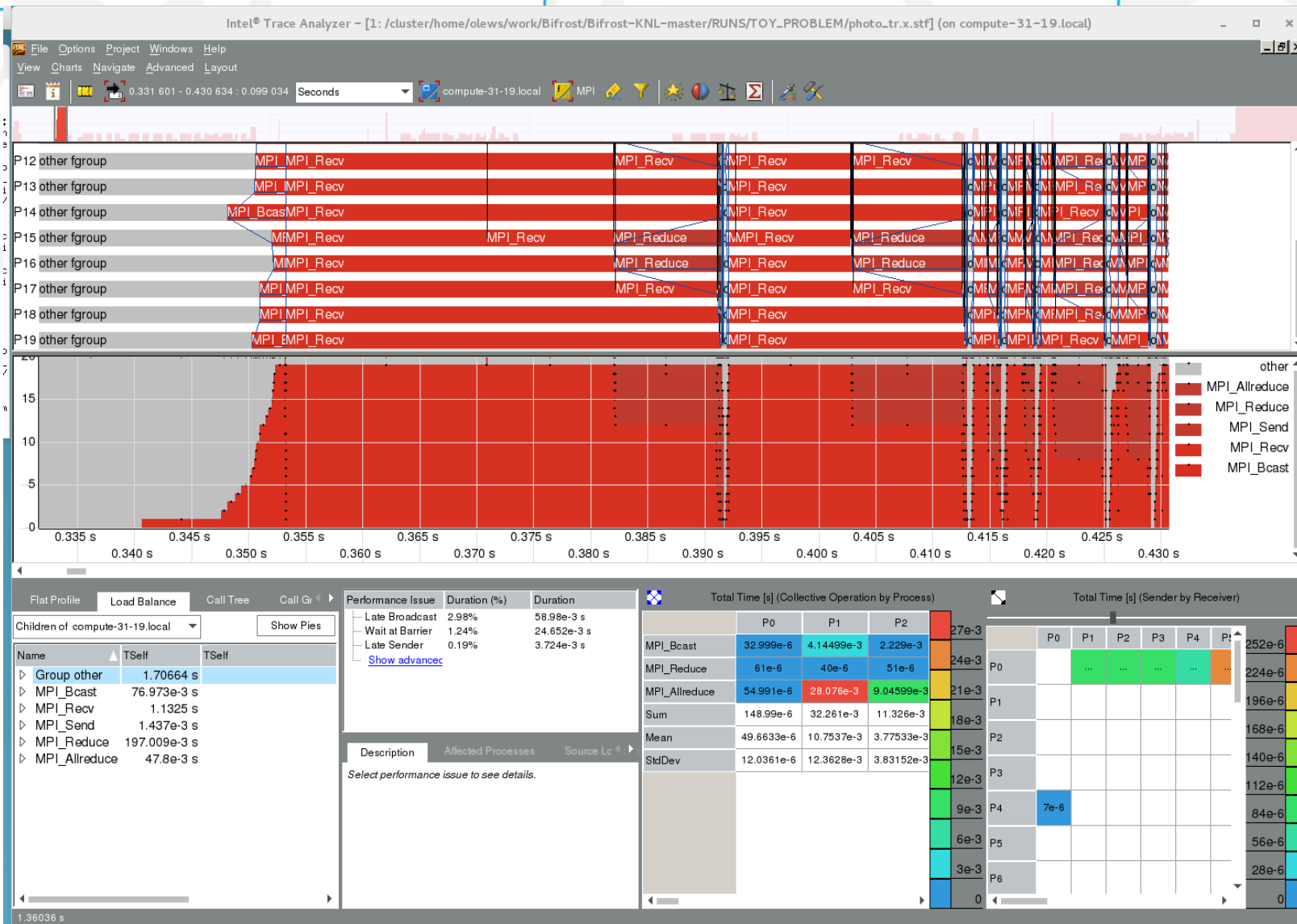




Programming - Intel tuning tools

- Trace Analyser
- Checks the MPI communication
- Trace all MPI calls

Intel Trace-Analyser



Intel tuning tools

