HPCG: shared memory implementation and evaluation on ThunderX2

Dani Ruiz
Daniel.Ruizmunoz@arm.com
Filippo Mantovani*
filippo.mantovani@bsc.es

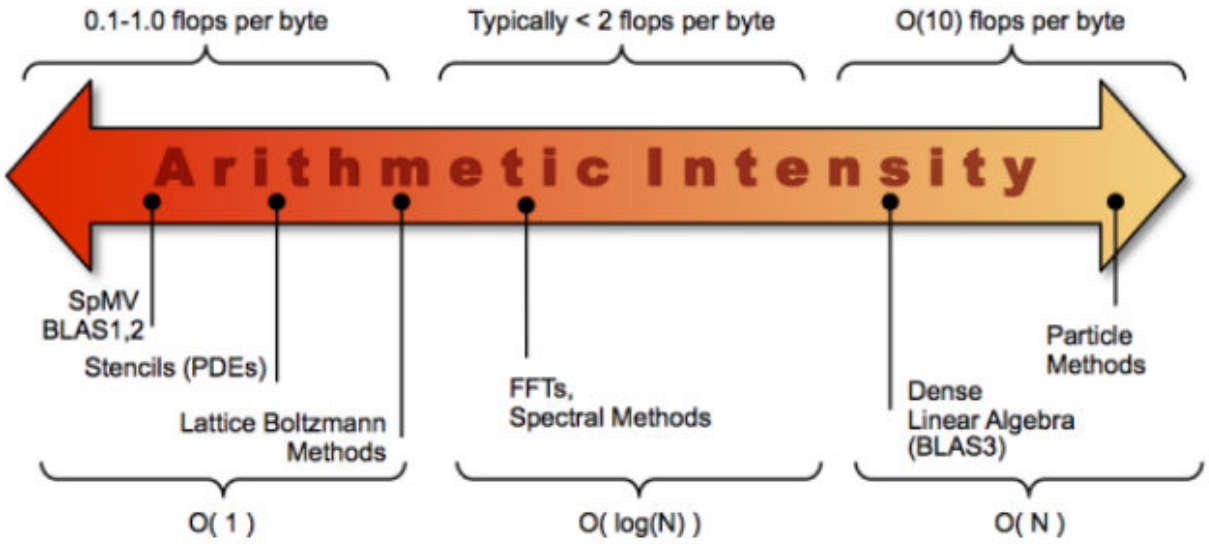
GoingArm workshop @ ISC18

Frankfurt – 2018, Jun 28th

Mont-Blanc Project
EU-H2020 GA–671697

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

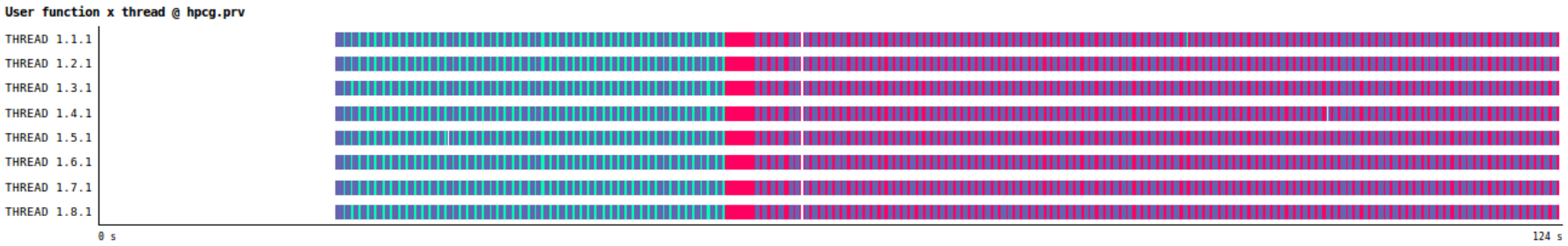
Why HPCG?



- Relevant / "fancy" for the HPC community
- Not fully explored on Arm architecture
- Part of the Student Cluster Competition 2017 (and 2018)

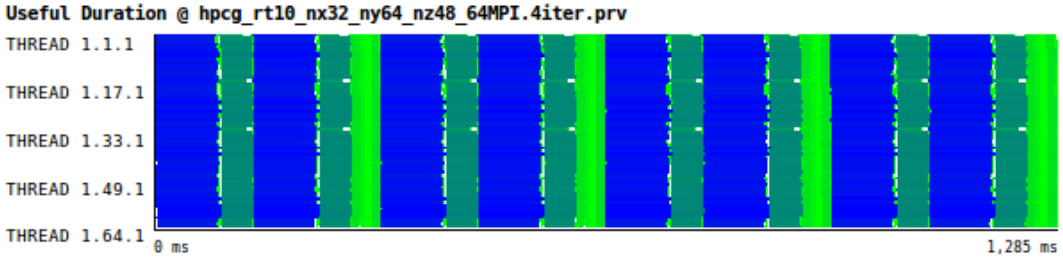
HPCG structure

Generate problem + Time of reference code + Time of optimized code

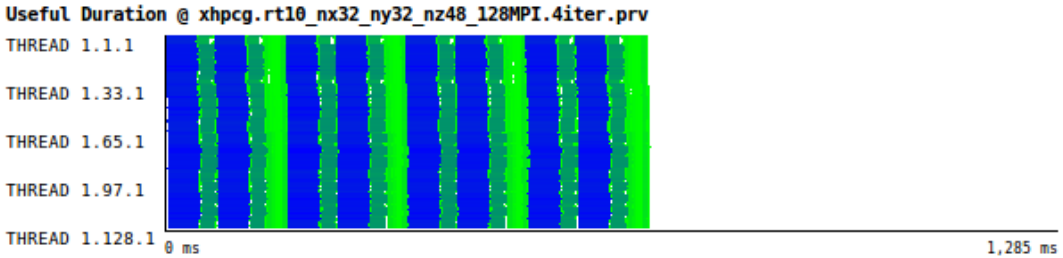


The benchmark is not based on a fixed number of operations. It relies on the "residual", computed after 50 iterations of the reference code.

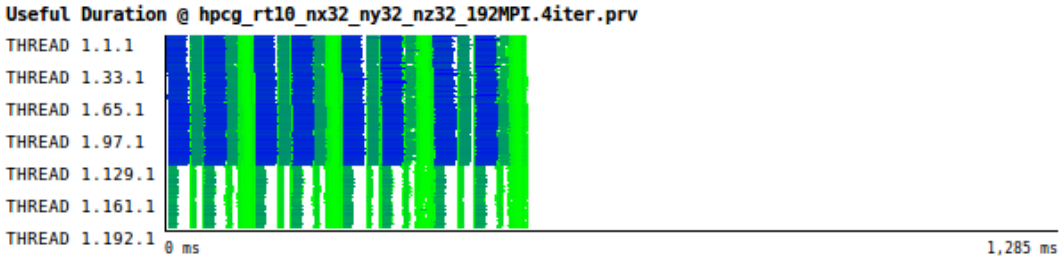
MPI-only strong scaling (4 iterations) on KNL



10% in MPI calls



20% in MPI calls



40% in MPI calls

MPI-only weak scaling (4 iterations) on KNL

	8	16	32	64	128	192	256
Global efficiency	89.13	81.97	58.22	31.79	31.53	30.51	30.87
Parallel efficiency	89.13	90.72	90.66	92.79	90.56	86.66	87.63
Load balance	90.67	91.87	92.27	96.03	95.45	92.25	94.51
Communication efficiency	98.30	98.76	98.25	96.62	94.88	93.94	92.72
Serialization efficiency	98.87	99.41	98.96	97.42	95.86	95.60	93.62
Transfer efficiency	99.42	99.34	99.29	99.19	98.98	98.26	99.04
Computation scalability	100.00	90.35	64.21	34.27	34.81	35.20	35.23
IPC scalability	100.00	90.54	64.38	34.85	35.32	35.86	35.86
Instruction scalability	100.00	99.91	99.82	99.72	99.67	99.64	99.62
Frequency scalability	100.00	99.88	99.93	98.60	98.89	98.53	98.60
Speedup	1.00	1.84	2.61	2.85	5.66	8.21	11.08
Average IPC	0.46	0.41	0.29	0.16	0.16	0.16	0.16
Average frequency (GHz)	1.49	1.49	1.49	1.47	1.47	1.47	1.47

- Computational scalability is the biggest problem
  - The memory bandwidth is the liming resource (as expected)
- Load balance introduced not by instructions, but by IPC
- Communication / serialization

Explore OpenMP parallelization

- For addressing scalability limitations mitigating the impact of MPI communications at high number of compute nodes
- To increase data reuse by threads working on the same local domain, thus improving effective bandwidth

Reference OpenMP version offers very limited parallelization:

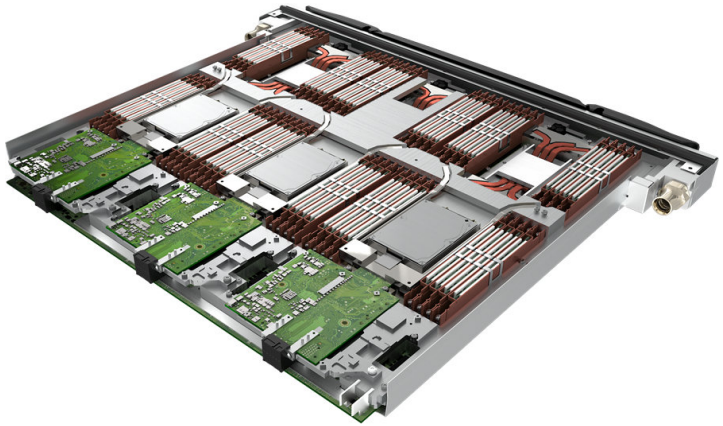
Disclaimer – Several scientists studied this problem before us.
Iwashita et al. `https://doi.org/10.1109/IPDPS.2012.51`
Iwashita et al. `https://doi.org/10.1109/20.996114`
Marjanovic et al. `https://doi.org/10.1007/978-3-319-17248-4_9`
Park et al. `https://doi.org/10.1109/SC.2014.82`
Zhang et al. `https://doi.org/10.1007/978-3-319-11197-1_3`

The evaluation platform: Dibona

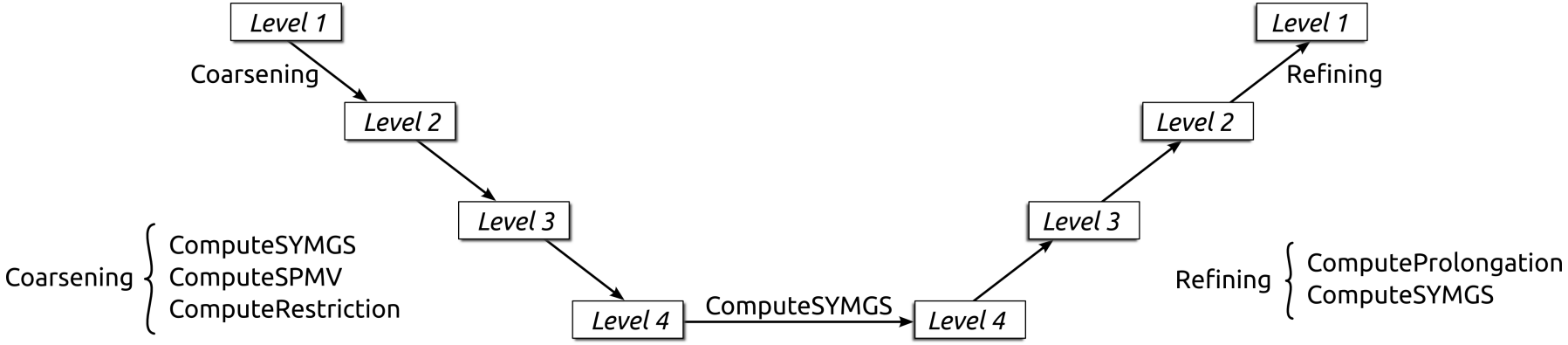
- 48 compute nodes
- 96 CTX2 SoCs
- $\sim$ 3000 cores @ 2.2GHz
- Infiniband EDR
- Direct liquid cooling
- RHEL

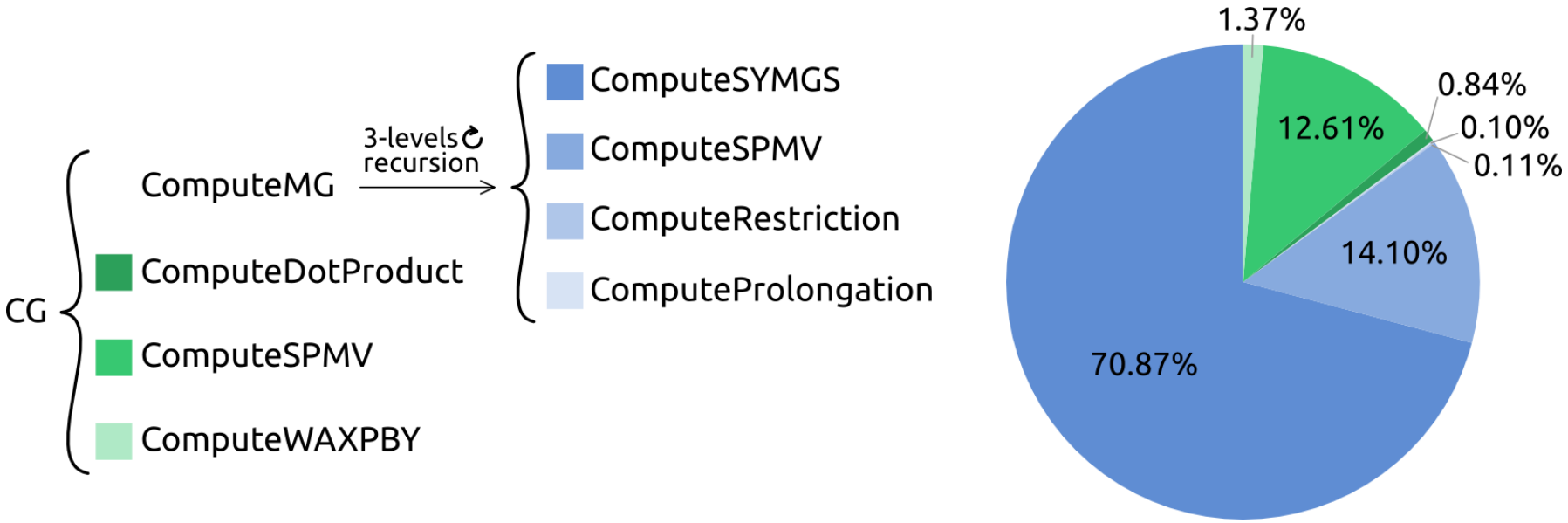
Platform availability

- Now $\rightarrow$ 3 nodes
- Mid of July $\rightarrow$ 21 nodes
- End of July $\rightarrow$ 48 nodes

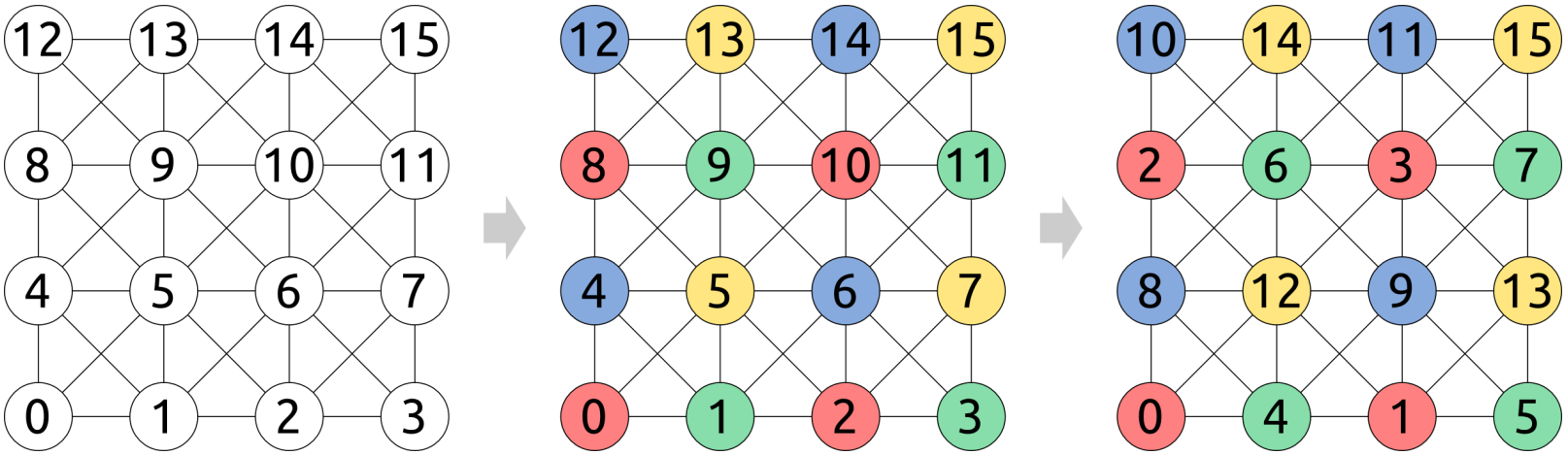


Profiling execution time



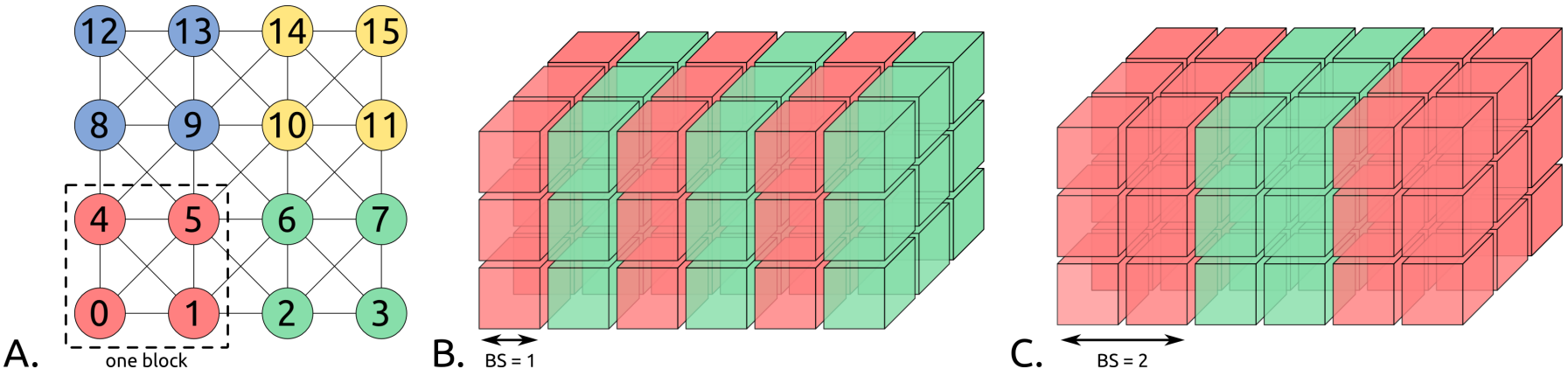


Optimization step 1: coloring



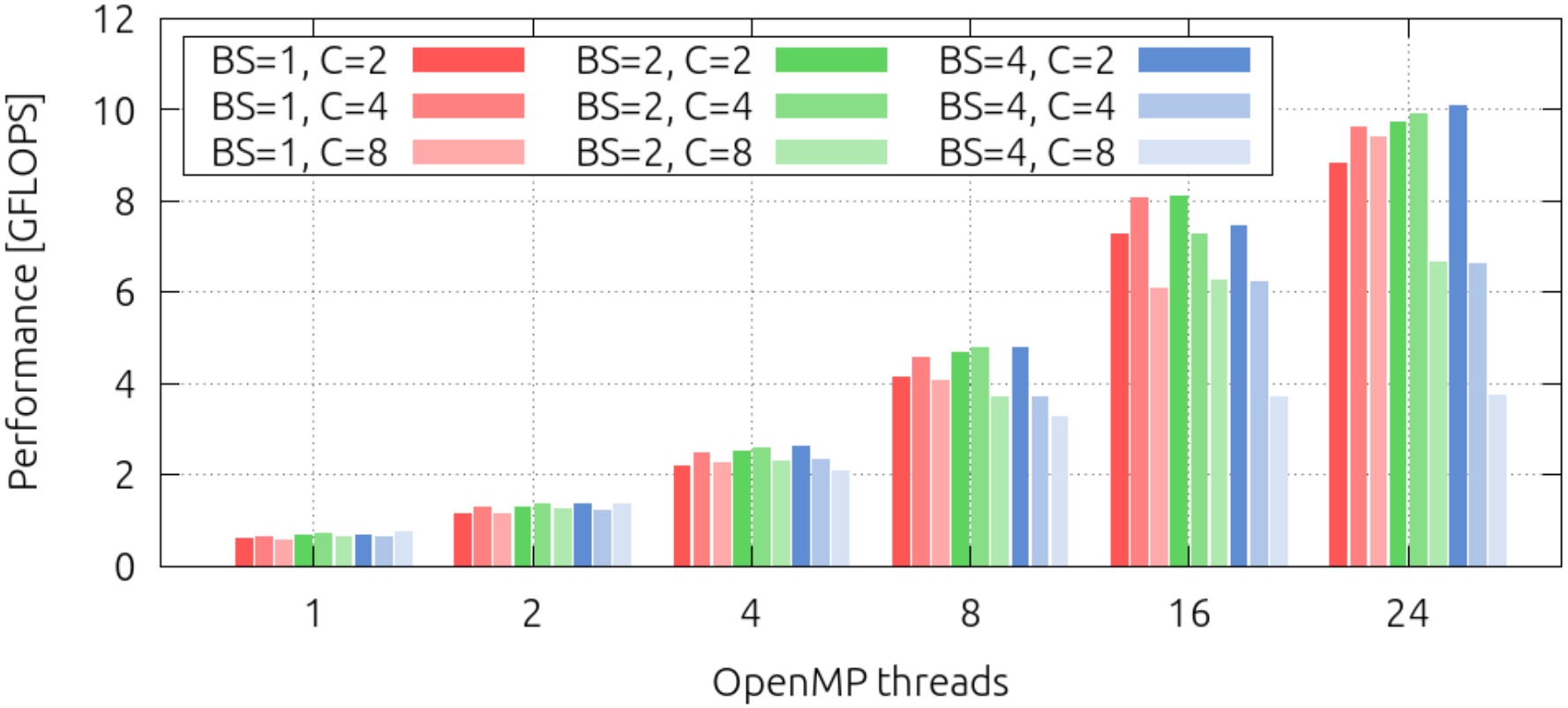
- Increased parallelism
- Not cache friendly
- It relaxes the Gauss-Seidel algorithm
  $\rightarrow$ it requires more iterations to converge

Optimization step 2: block coloring –1–



- Performance depends on 2 parameters: `BS` and `C`
- Cache friendly
- Within slices it takes advantage of single core speed
- Small impact ($< 10\%$) on iterations to converge

Optimization step 2: block coloring –2–



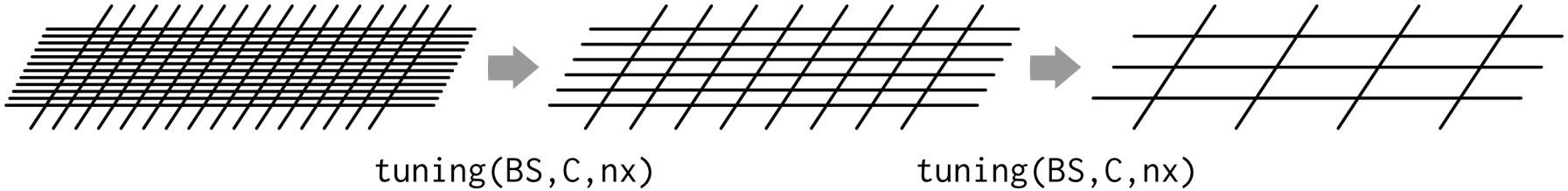
$$B_{\|} = \frac{nx}{(BS * C)}$$

Depending on values of `nx`, `BS` and `C`, it can be $B_{\|} < T$

$B_{\|} < T \rightarrow$ idle threads $\rightarrow$ suboptimal performance

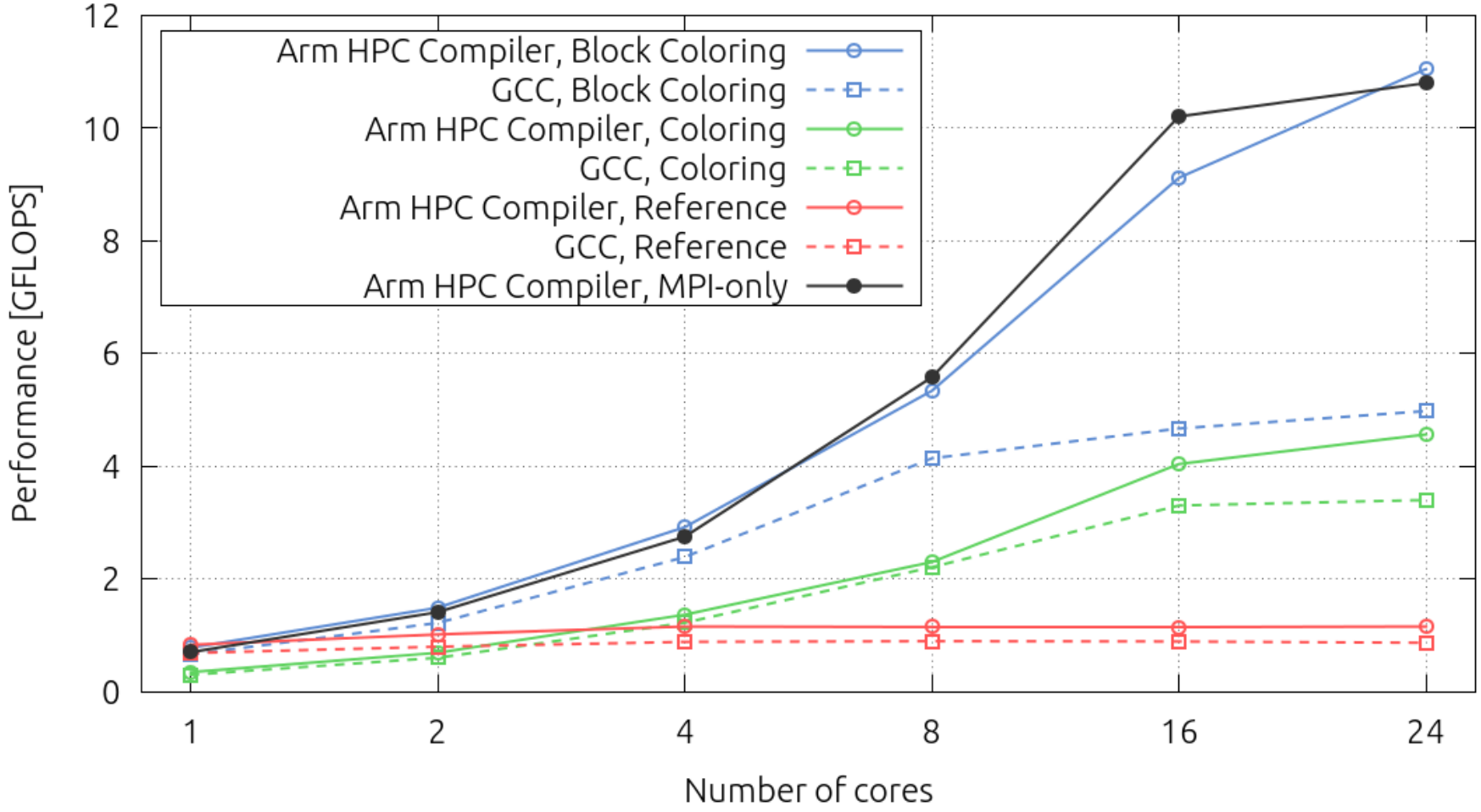
Optimization step 3: dynamic block coloring

In order to avoid thread starvation, we dynamically tune `BS` and `C` each time we change recursion step (i.e. each time the algorithm requires to change `nx`).



As positive side effect, the convergence is not badly affected: this method requires in fact only 1 extra iteration to converge.

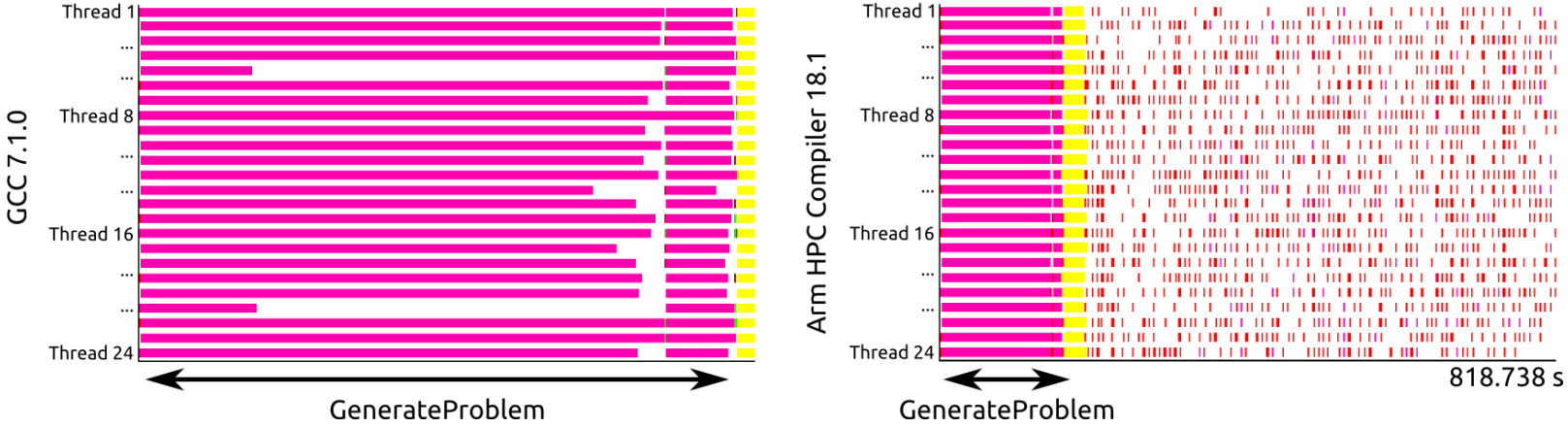
Single node evaluation



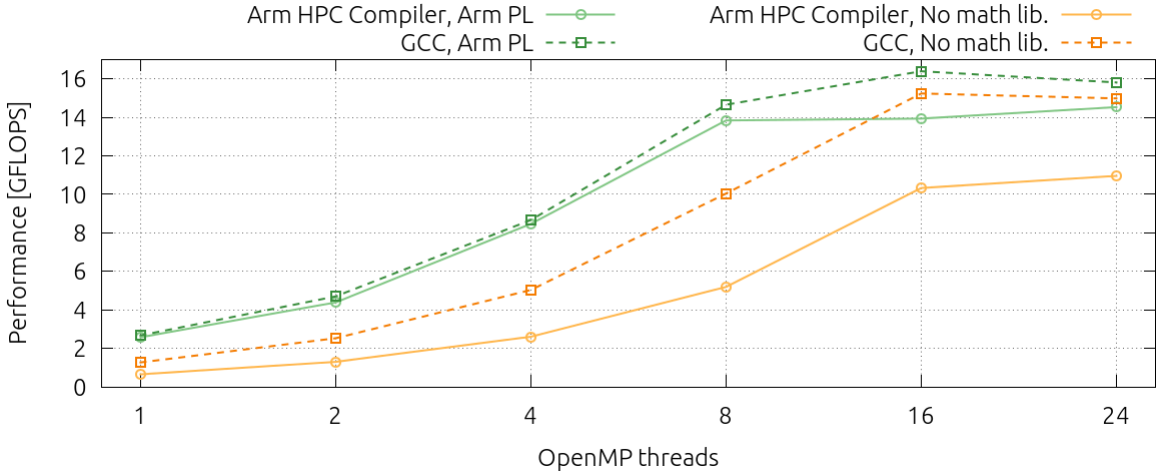
B0 silicon, OpenMPI 3.0, GCC 7.1.0, Arm HPC Compiler 18.1

Arm Compiler and performance libraries

- Arm compiler delivers better performance compared to GCC



- Very limited auto-vectorization by the Arm compiler
- Linear algebra kernels $< 5\%$ of the execution time



Conclusions

- Three shared memory implementation of the HPCG benchmark
- Single node evaluation in Dibona (CTX2 by Bull)
- Source available: https://gitlab.com/arm-hpc/benchmarks/hpcg

☺ Arm tools + our implementation outperforms MPI-only version

☹ There are still corners to explore (e.g. the 'GenerateProblem')

- Next research steps enabled by this contribution:
  - Optimized implementation of SpMV kernel in Arm PL
  - Exploration of vectorization with SVE
  - Memory access optimizations for improving cache reuse

More details can be found in:
https://upcommons.upc.edu/handle/2117/116642

Acknowledgment: We warmly thank the Mont-Blanc consortium and Bull for providing the evalutation platform, Arm for the support, Michael Heroux for advising us.