PSyGS Gen: A Generator of DSAs for Sparse Linear System Resolution

30/06/2023

Francesco Negri *francescorenato.negri@mail.polimi.it*
Francesco Pesce *francesco1.pesce@mail.polimi.it*
Niccolò Nicolosi *niccolo.nicolosi@mail.polimi.it*

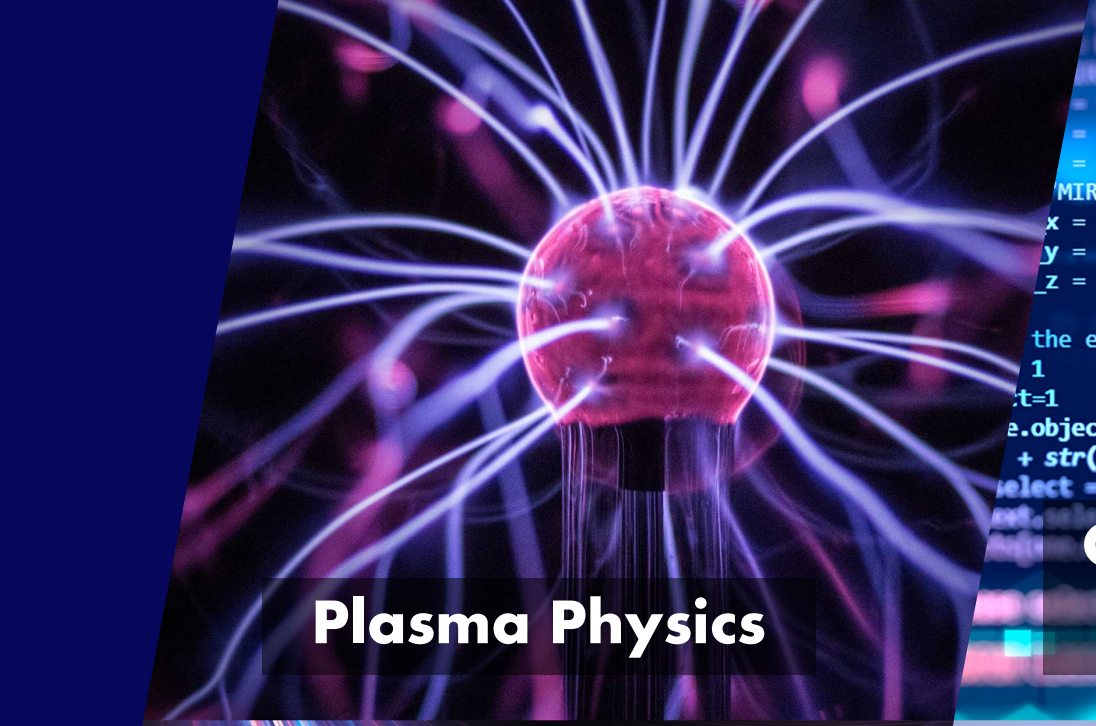
POLITECNICO
MILANO 1863

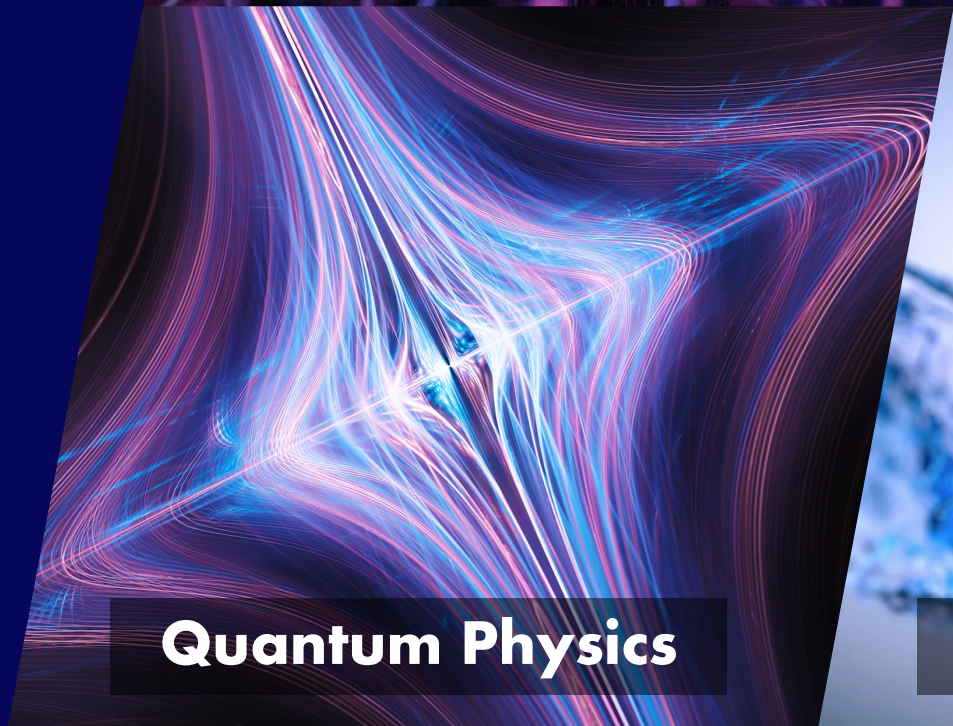
POLITECNICO MILANO 1863
NECST laboratory

The problem:

accelerate sparse linear system resolution

Some applicative domains





Where we start: the Gauss-Seidel algorithm

- An iterative algorithm to solve sparse linear systems in the form:

$$A\bar{x} = \bar{b}$$

- At each iteration the following update formula is used to sequentially update each component of $\bar{x}$:

$$x_i := \frac{1}{a_{ii}} \left(b_i - \sum_{j \neq i}^{n} a_{ij} x_j \right)$$

- After an arbitrary number of iterations the convergence is checked using the following formula:

$$\|A\bar{x} - \bar{b}\| < T$$

State of the art: GS parallelization

Coloring techniques can be used to highlight dependencies between updates: independent updates can be done in parallel [1]

Coloring technique	Highlighted dependencies
Graph coloring	Few
Block coloring	Many
Dependency graph	All

Less dependencies highlighted → **Faster iterations**
More iterations to converge

More dependencies highlighted → **Slower iterations**
Fewer iterations to converge

[1] D. Ruiz, F. Spiga, M. Casas, M. Garcia-Gasulla and F. Mantovani, "Open-Source Shared Memory implementation of the HPCG benchmark: analysis, improvements and evaluation on Cavium ThunderX2," *2019 International Conference on High Performance Computing & Simulation (HPCS)*, Dublin, Ireland, 2019, pp. 225-232, doi: 10.1109/HPCS48598.2019.9188103.

Related work: DSAs embedding GS

- **Different DSAs that embed GS (serial implementation) in their computational flow [2], [3], [4].**

- **No DSAs to accelerate GS specifically:**

No directly comparable performance

- **Existing DSAs would benefit from a better GS hardware accelerator**

[2] J. -H. Byun, A. Ravindran, A. Mukherjee, B. Joshi and D. Chassin, "Accelerating the Gauss-Seidel Power Flow Solver on a High Performance Reconfigurable Computer," 2009 17th IEEE Symposium on Field Programmable Custom Computing Machines, Napa, CA, USA, 2009, pp. 227-230, doi: 10.1109/FCCM.2009.23.

[3] D. P. Chassin, P. R. Armstrong, D. G. Chavarria-Miranda and R. T. Guttromson, "Gauss-Seidel accelerated: implementing flow solvers on field programmable gate arrays," 2006 IEEE Power Engineering Society General Meeting, Montreal, QC, Canada, 2006, pp. 5 pp.-, doi: 10.1109/PES.2006.1709227.

[4] J. Zeng, J. Lin and Z. Wang, "An Improved Gauss-Seidel Algorithm and Its Efficient Architecture for Massive MIMO Systems," in IEEE Transactions on Circuits and Systems II: Express Briefs, vol. 65, no. 9, pp. 1194-1198, Sept. 2018, doi: 10.1109/TCSII.2018.2801867.

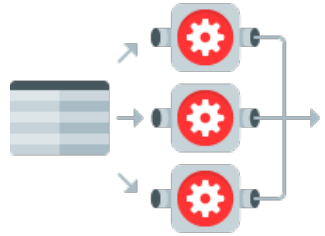
Our solution: PSyGS Gen

We propose PSyGS Gen: a generator of DSAs to accelerate GS parallelizing it by exploiting data preprocessed with coloring techniques

Main features:

- **A single generator, many DSAs:**
  highly parametrized DSA template to match different use cases

- **Implements parallel GS:**
  to maximize the speed of the computation



- **Prototyping ready:**
  fully integrated with the Chipyard environment

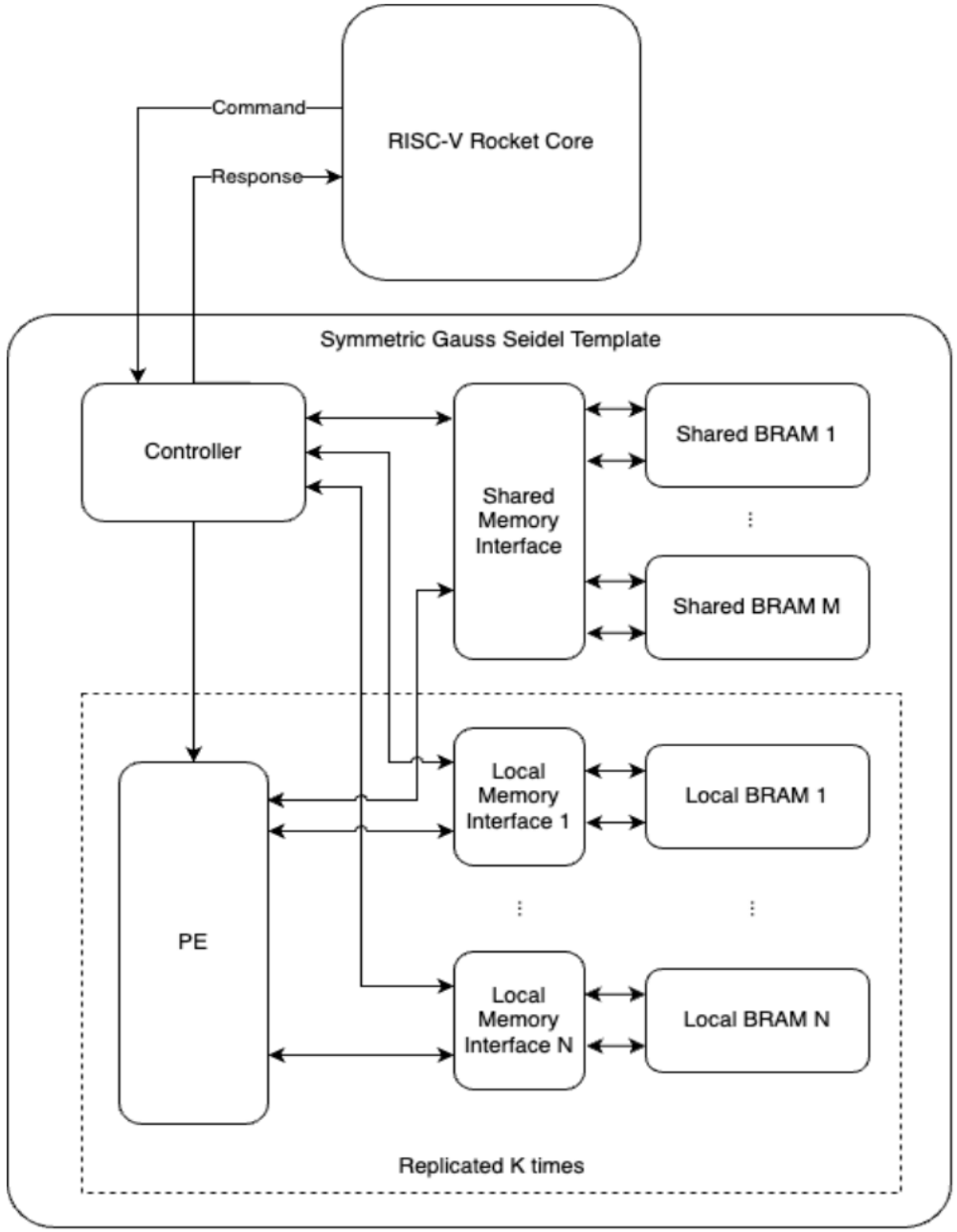
Architecture overview

The Rocket Core executes the host code that leads the algorithm and sends commands to the controller of our accelerator

The controller orchestrates the Processing Elements (PEs) to execute the commands

Vector $\bar{x}$ is stored in the shared memory, split among the different banks

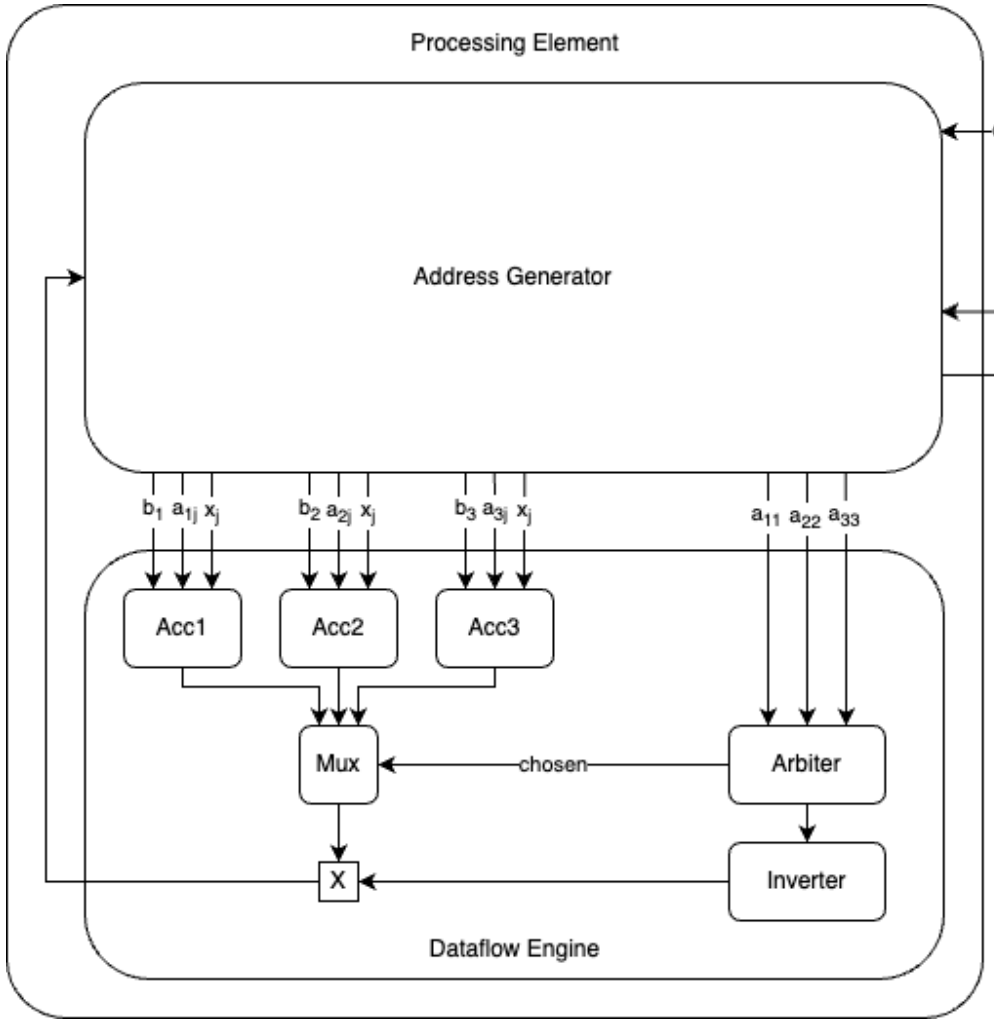
Each PE has dedicated local memory in which the data needed for its computations are stored



Processing element

The PE is composed by:

- **The Address Generator, which reads and writes memory**
- **The Data Flow Engine, which does the actual computation**



The DFE computes the update formula:

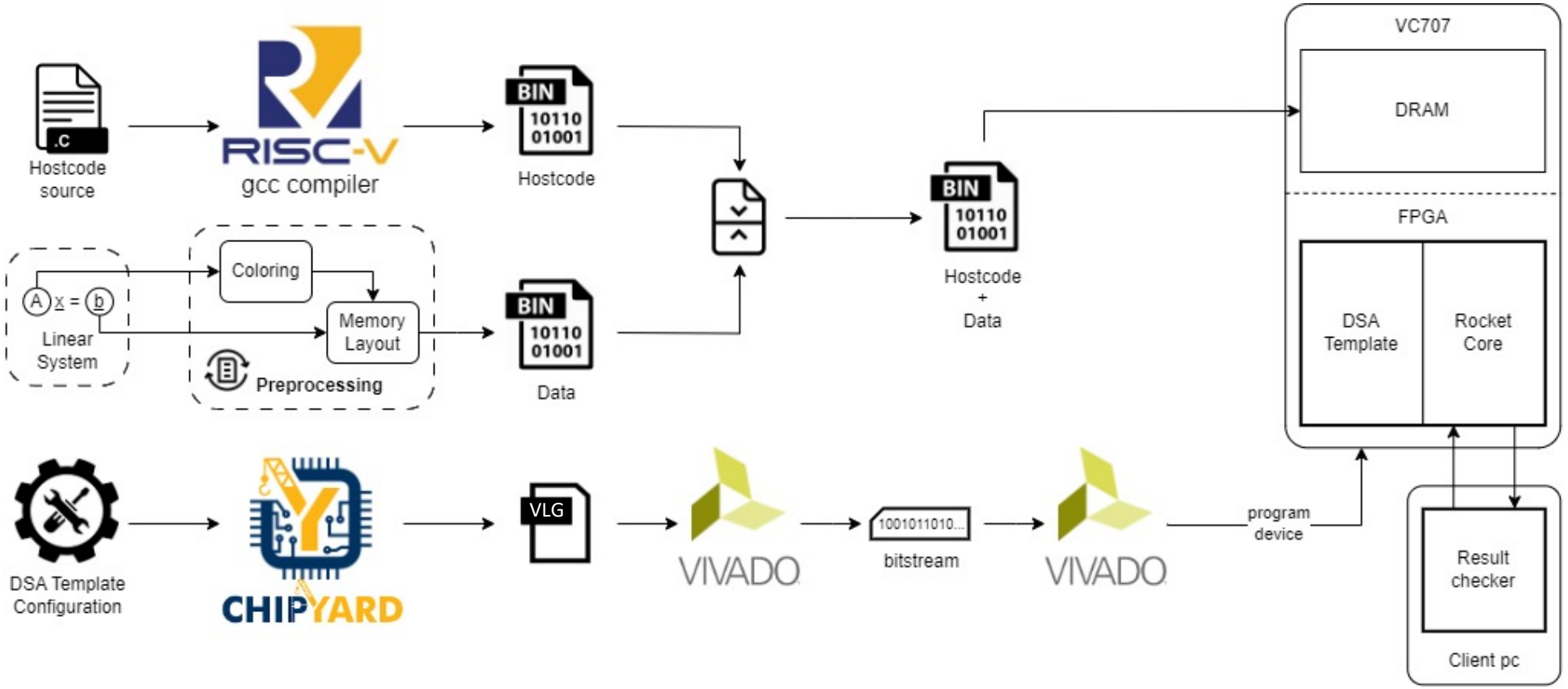
$$x_i := \frac{1}{a_{ii}}\left(b_i - \sum_{j \neq i}^{n} a_{ij}x_j\right)$$

Example PE parametrized to have three accumulation units. In this example the updates of x_1, x_2 and x_3 are computed in parallel within the single PE.

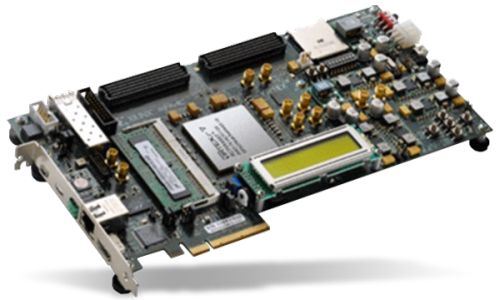
The PE can be replicated.

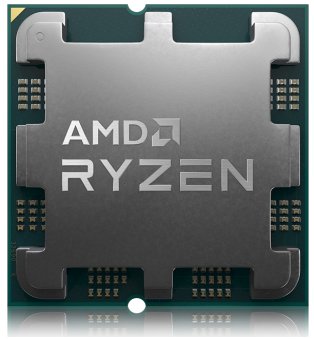
Prototyping Flow

We built an automated flow for prototyping different configuration of our template



Performance comparison with CPU

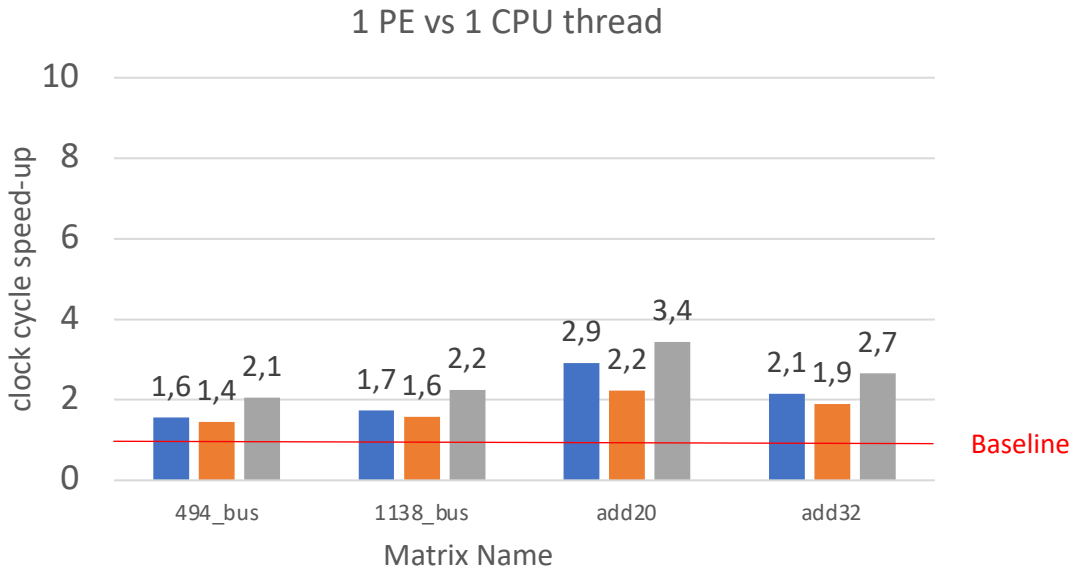




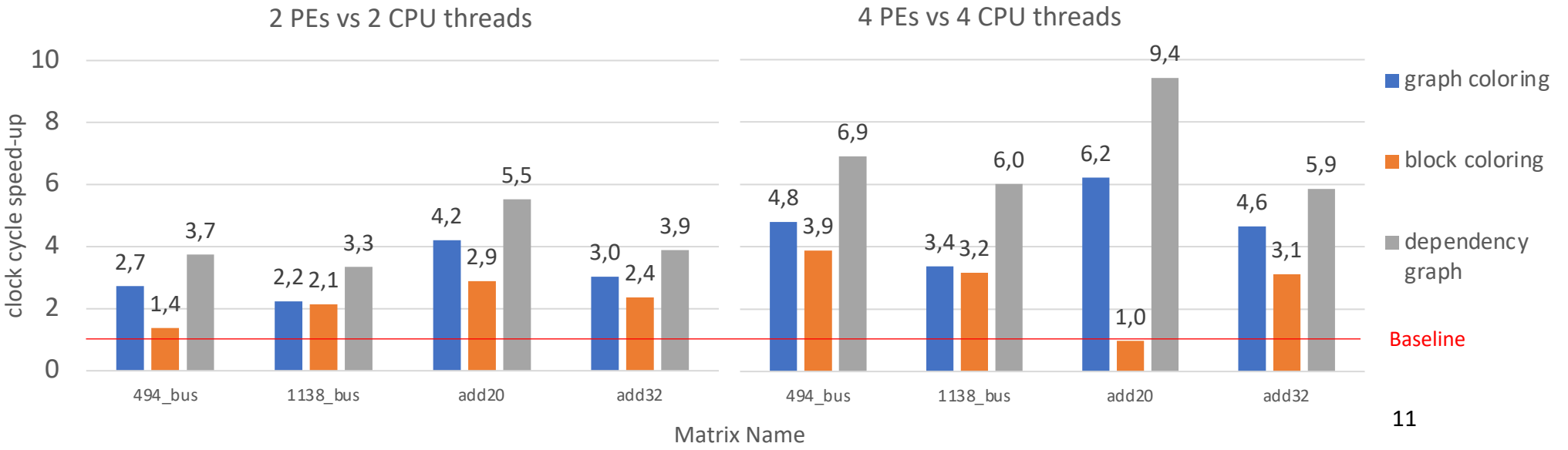
Virtex 7 FPGA	
Released in 2010	
28nm transistors	

AMD Ryzen 7 3750H	
Released in 2019	
12nm transistors	
Number of CPU Cores	4
Number of Threads	8
Base Clock	2.3GHz
Max. Boost Clock	4.0GHz
L1 Cache	384KB
L2 Cache	2MB
L3 Cache	4MB

Performance comparison with CPU

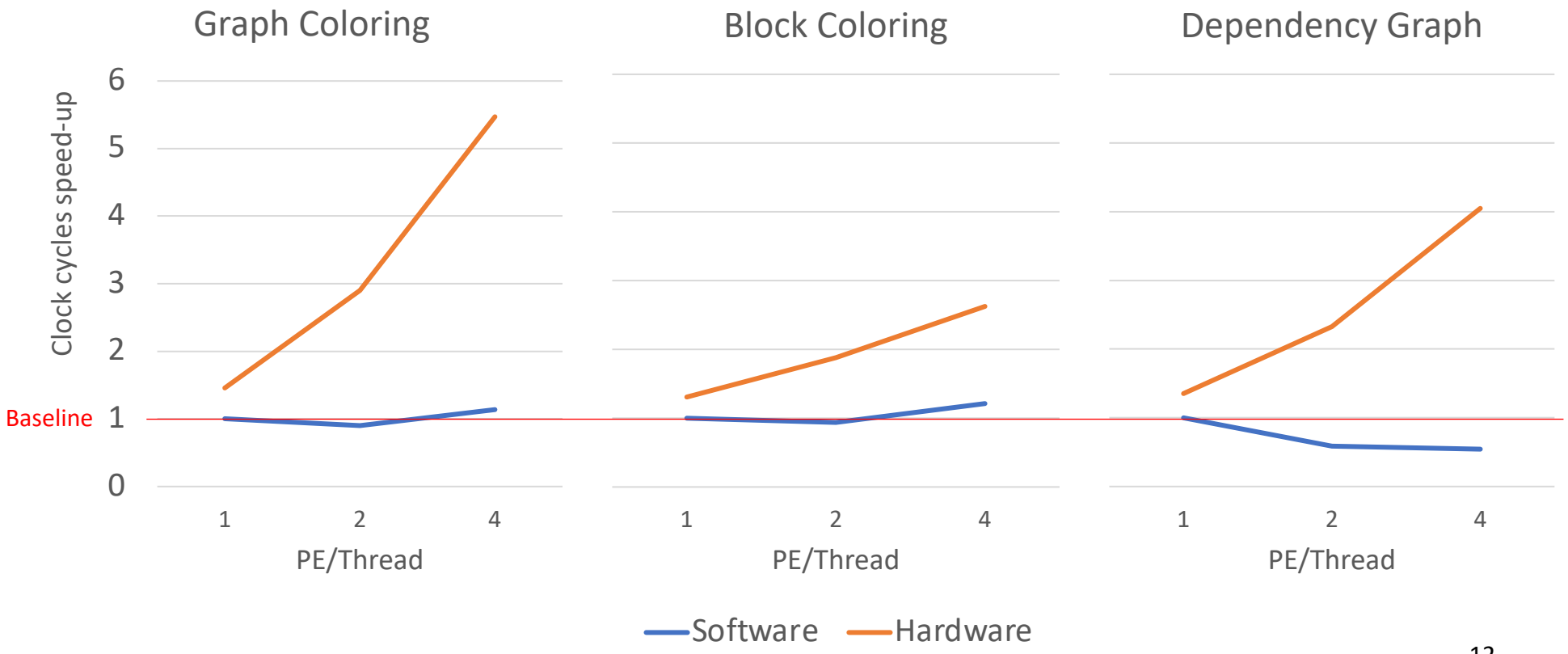


The generated DSAs outperform the software counterparts with the same level of parallelism in terms of clock cycles.

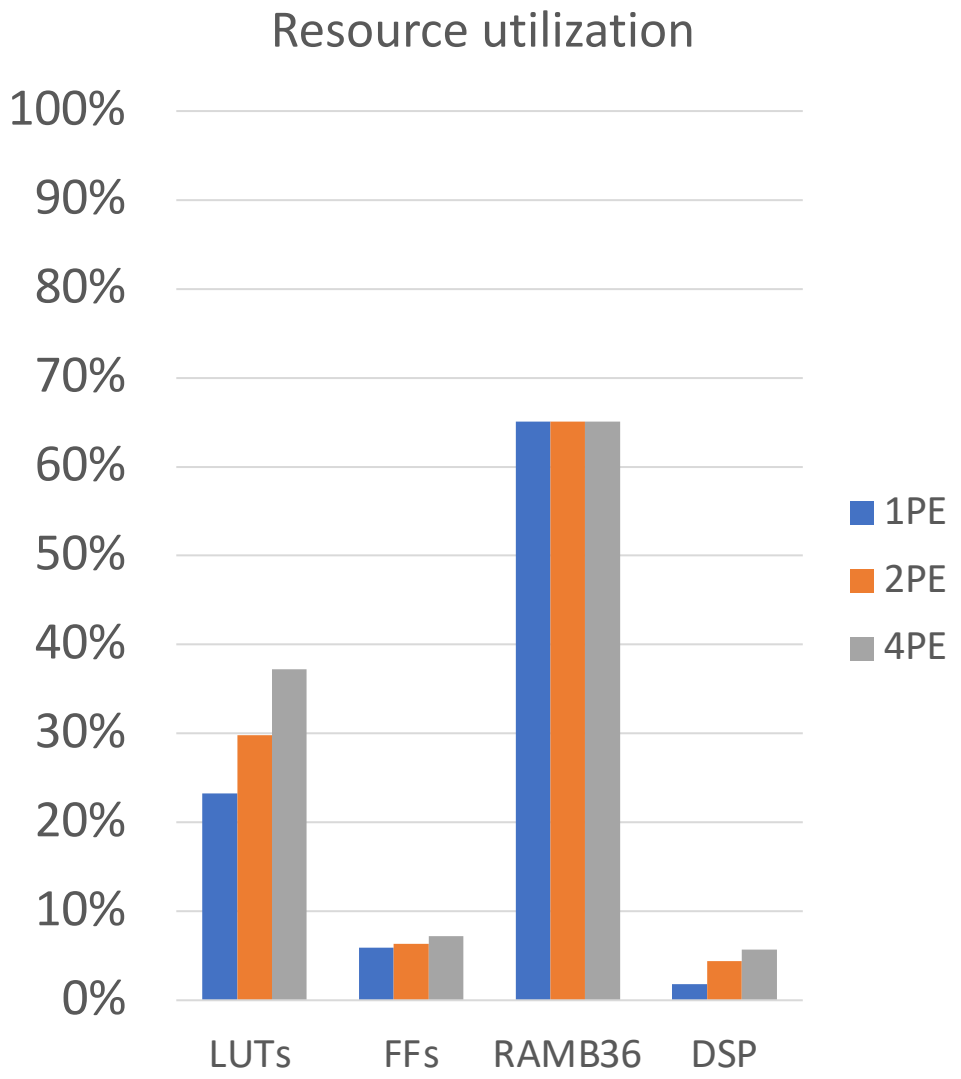


Scale-up comparison with CPU

The speed-up trends with respect to the number of PEs/Threads show that while the software implementation of GS improves very little or even gets worse when using dependency graph, our DSAs scale progressively better.



FPGA resource utilization scaling



The usage of FPGA resources by the generated DSAs scales promisingly with the number of PEs.

This implies that DSAs with even more PEs can be generated to reach better performance, without requiring excessive resources.

Total available resources	
LUTs	303600
FFs	607200
RAMB36	1030
DSP	2800

Conclusions and future work

The results obtained confirm that PSyGS Gen is capable of generating DSAs that outperform the software counterparts in terms of number of execution clock cycles.

Even if the execution times are still worse than the software implementation due the relevant difference in clock frequency, we demonstrated that by properly scaling the DSA and increasing its clock frequency enough, we can surpass the performance of the software implementation.

Future work

- **Optimize the data flow engine with piplined float modules to increase throughput and allow higher clock frequency**
- **Switch from the current soft-core host processor (Rocket Core) to a hard-core to increase the overall performance by decoupling the accelerator from the host, allowing higher clock frequency and reducing FPGA resources utilization**

References

1. D. Ruiz, F. Spiga, M. Casas, M. Garcia-Gasulla and F. Mantovani, "Open-Source Shared Memory implementation of the HPCG benchmark: analysis, improvements and evaluation on Cavium ThunderX2," *2019 International Conference on High Performance Computing & Simulation (HPCS)*, Dublin, Ireland, 2019, pp. 225-232, doi: 10.1109/HPCS48598.2019.9188103.

2. J. -H. Byun, A. Ravindran, A. Mukherjee, B. Joshi and D. Chassin, "Accelerating the Gauss-Seidel Power Flow Solver on a High Performance Reconfigurable Computer," 2009 17th IEEE Symposium on Field Programmable Custom Computing Machines, Napa, CA, USA, 2009, pp. 227-230, doi: 10.1109/FCCM.2009.23.

3. D. P. Chassin, P. R. Armstrong, D. G. Chavarria-Miranda and R. T. Guttromson, "Gauss-Seidel accelerated: implementing flow solvers on field programmable gate arrays," 2006 IEEE Power Engineering Society General Meeting, Montreal, QC, Canada, 2006, pp. 5 pp.-, doi: 10.1109/PES.2006.1709227.

4. J. Zeng, J. Lin and Z. Wang, "An Improved Gauss-Seidel Algorithm and Its Efficient Architecture for Massive MIMO Systems," in IEEE Transactions on Circuits and Systems II: Express Briefs, vol. 65, no. 9, pp. 1194-1198, Sept. 2018, doi: 10.1109/TCSII.2018.2801867.

Thank you for your attention

Francesco Negri

francescorenato.negri@mail.polimi.it

Francesco Pesce

francesco1.pesce@mail.polimi.it

Niccolò Nicolosi

niccolo.nicolosi@mail.polimi.it