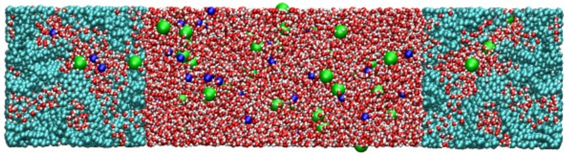
Matrix free conjugate gradient with Maxeler Data Flow Engine technology

Charles Prouveur

Table of content

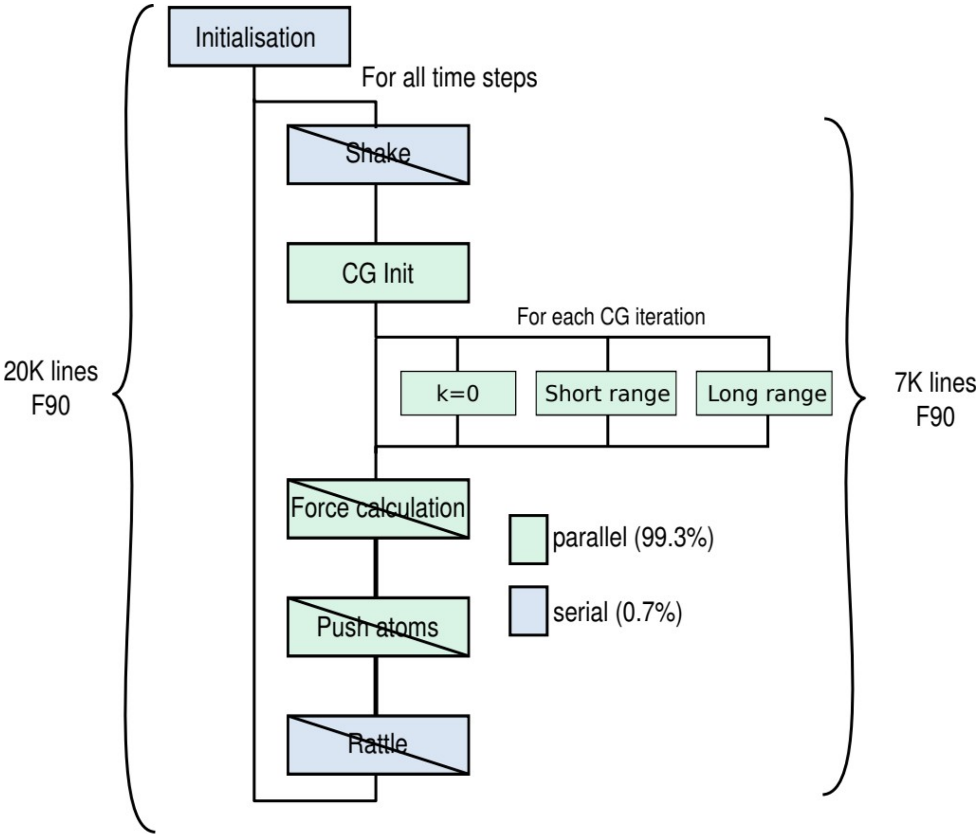
- Introduction
- Metalwalls
- Target device
- FPGA design
  - Numerical analysis
  - Kernel balancing in a « all in one » design
  - Design using Multiple FPGAs
- Results
- Conclusion

MetalWalls



- Molecular dynamic production code used by Sorbonne university researchers to simulate electrochemical systems such as «supercapacitors»
- Fortran 90 base code, parallelised with MPI
- Most of the computing time = electrostatic potential computing
- Computing efficiency published : model running during several weeks on 512 cores while maintaining a parallel efficiciency above 75 %
- Currently using CPU and GPU implementations (OpenAcc)

Miniapp extracted from the production code



Ideal code for testing different frameworks and architectures

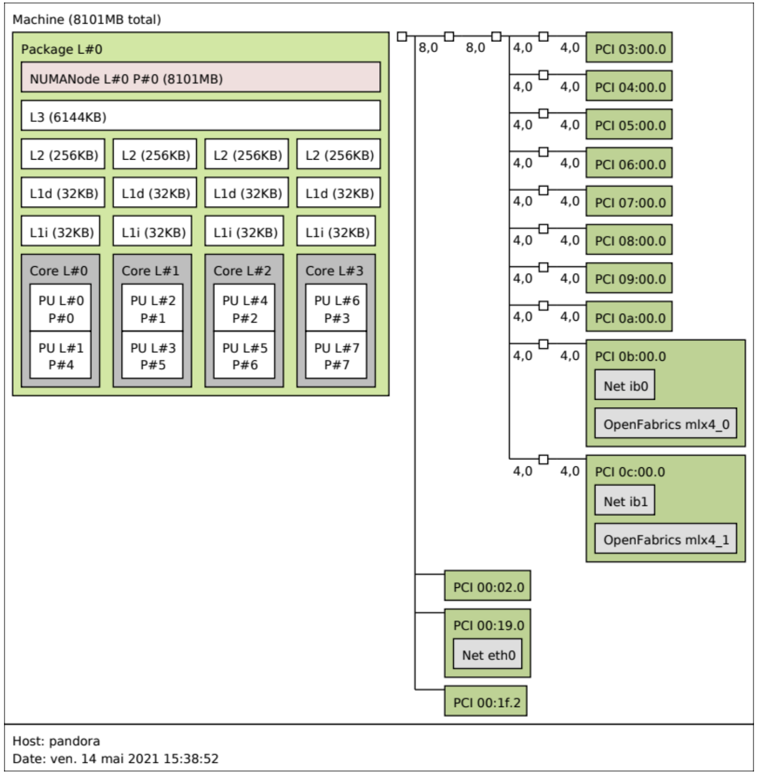
- Framework SkePU (skeletons)
- Framework StarPU (tasks using codelets)
- Maxcompiler (FPGA acceleration)

The FPGA used here

- Max5C is a U200 with one less DIMM
- 3 SLR : the chip is divided in 3 Super Logic Region connected by solders (very low bandwidth)
- SLR communication should be avoided as much as possible

	MAX5C	Alveo U200
FPGA	VU9P	VU9P
SLRs	3	3
LUTs (k)	1,182	1,182
FFs (k)	2,364	2,364
DSPs	6,840	6,840
BRAM18s	4,320	4,320
URAMs	960	960
On Chip Memory Capacity	43.2 MByte	43.2 MByte
DDR DIMMs	3	4
DDR Capacity per DIMM	16 GB	16 GB
Supported DDR Frequencies (MHz)	933, 1066, 1200	1200
DIMM to SLR Mapping	DIMM_0 ->SLR0 DIMM_1 ->SLR1 DIMM_2 ->SLR2	DIMM_0 ->SLR0 DIMM_1 ->SLR1 DIMM_2 ->SLR1 DIMM_3 ->SLR2
PCIe Placement	SLR1	SLR0
PCIe	PCIe Gen 2 x8	PCIe Gen 2 x8
Networking	1 x 100 GBit/s	2 x 100GBit/s
Networking Placement	SLR2	SLR2

Target device on Jumax at Juelich computing center



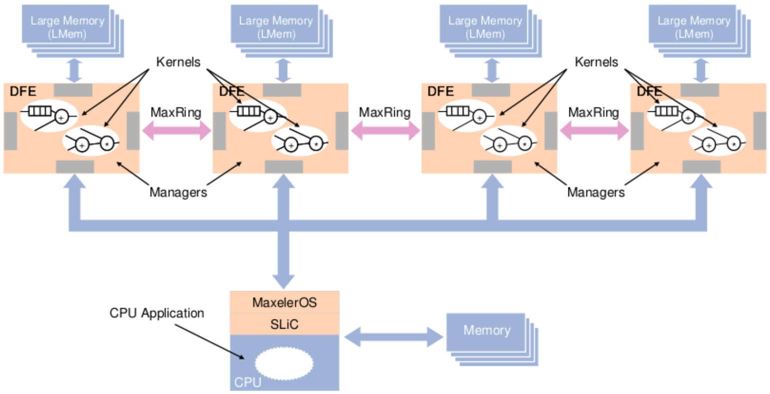


Figure 1: Overview of a Maxeler acceleration system

- CPU host : AMD EPYC 7601
- 8 nodes Xilinx VU9P on one blade as target devices
- 8x PCIe 2.0 lanes → 4.0 GB/s for all nodes
- Each node has three 16-GB DDR4 Dual In-Line Memory Modules (DIMMs), which provide a theoretical peak bandwidth of 15 GB/s each

Designing all kernels on one FPGA

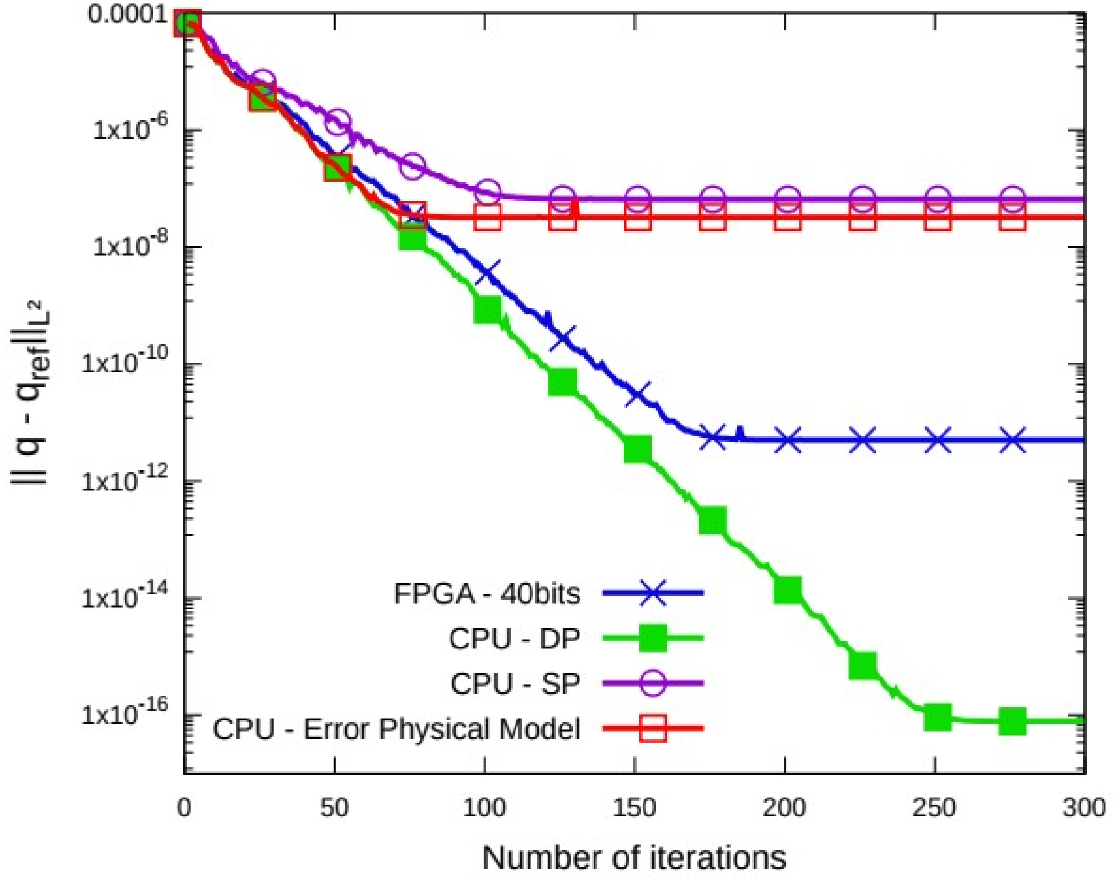
- Due to hardware contraints, each kernel is put into one SLR (super logic region)
- For simplicity, the conjugate gradient is computed on the host
- We want the highest frequency possible
- We also want the biggest number of separate pipelines
- All kernels work synchronously
- Use as much as possible the device’s DRAM to reduce communications

Challenges

- Limited ressources
- More logic available→ higher design frequency likely
- More pipelines → less logic available
- Using DRAM makes meeting timings harder

  → harder compilation, ie we are less likely to achieve high frequency with as many pipelines

Numerical accuracy analysis



We can save ressources but there is a catch : the number of iterations to converge increases

Balancing the kernels

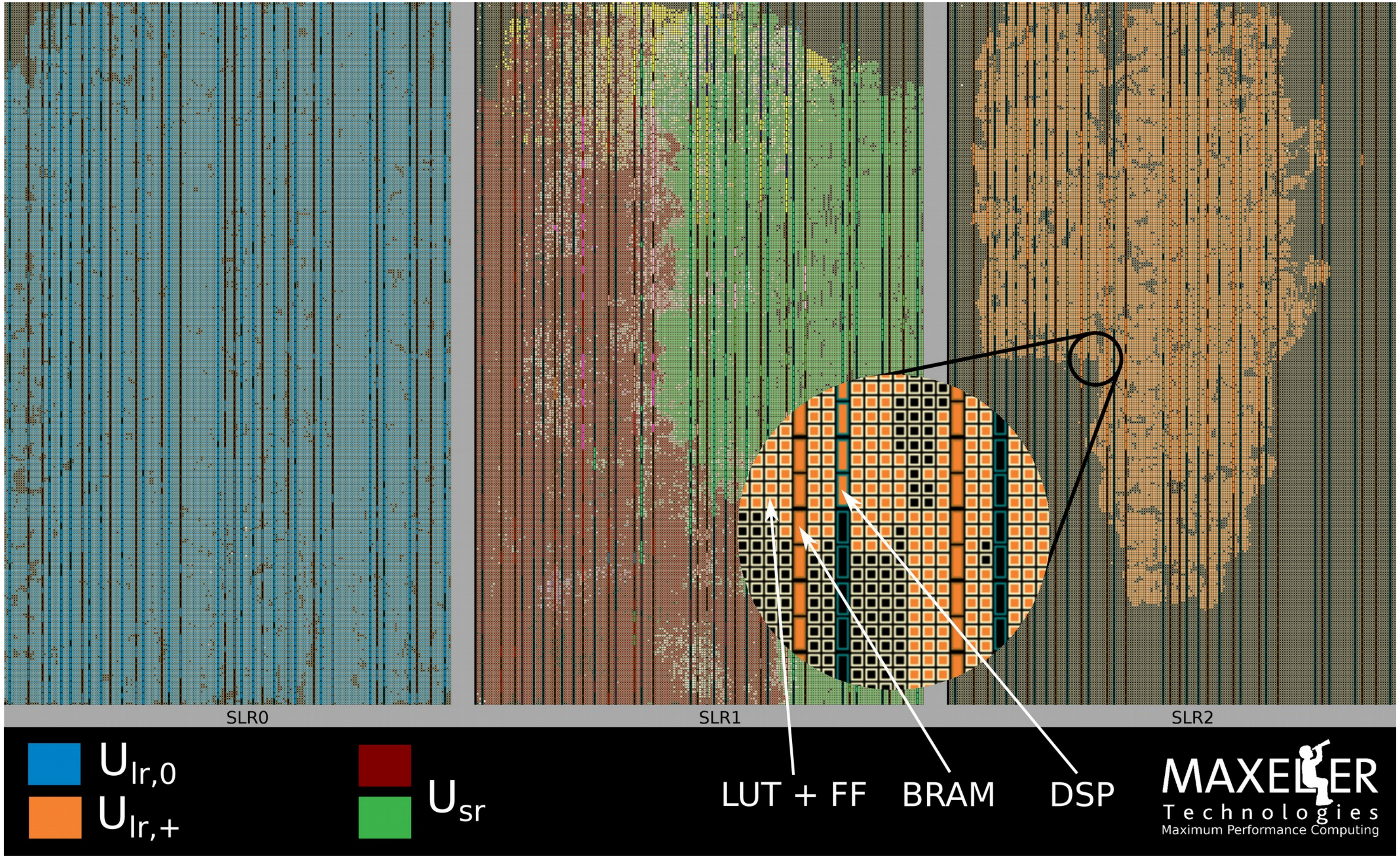
- Need for synchronicity → balance needed
- Theoretical time for each sequential kernel is known
- Balance is case dependent
- For the production test case considered here, the balance is (8,4,1)

Ressources usage of the multiple kernel designs

Design Name	64 bits Design	40 bits Design	Final design
Design frequency (MHz)	200	200	300
Pipes ($U_{\mathrm{lr},0}, U_{\mathrm{sr}}, U_{\mathrm{lr},+}$)	(8,4,1)	(16,8,2)	(32,16,4)
Logic (LUTs & FFs)	27.7%	33.4%	44.6 %
DSPs	33.42%	29.52%	53.3%
On-chip Mem	22.7%	20.3%	28.8%

DSP limited in the U_lr,0 kernel (87 % DSPs used in its SLR)

Airview of the compiled final design

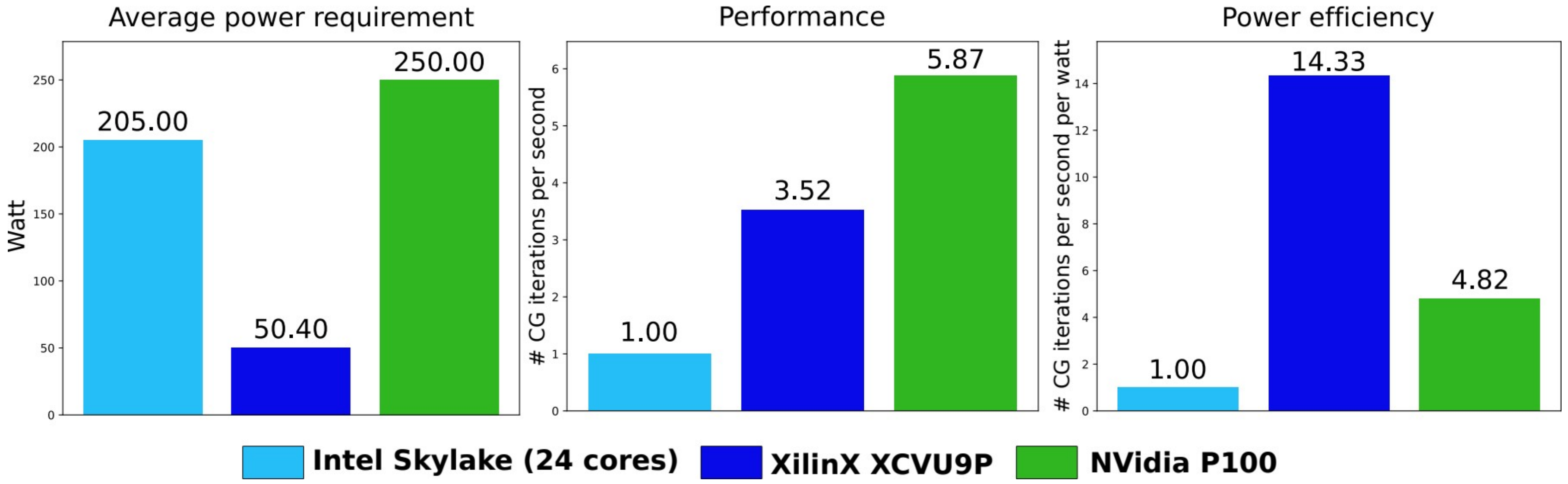


Design using the device’s DRAM

- The previous design does not use the DRAM
- Q has to be sent every iteration from and to the FPGA
- (x,y,z) they can be stored
- Using LMEM makes the design harder to compile

  → have to make concessions
- Best design is (24,12,3) with a frequency of 260 MHz

Results comparison



Scaling to multiple FPGAs

- We expect to be communications bound.
- Our « all in one » designs can be immediately tested with multiple FPGAs, but they are not adapted to all test cases.
- To better use each FPGA we can make a design for each kernels.

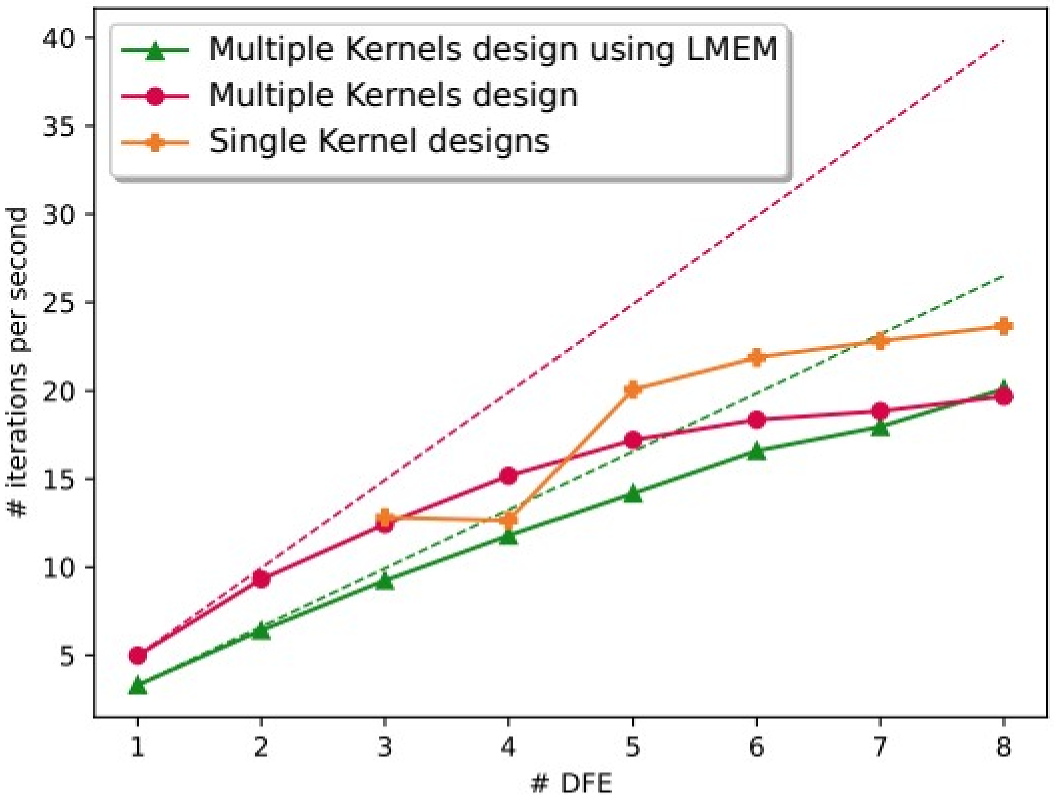
→ this also allows for load balancing for any test case

Ressources usage of the single kernel designs

Design Name	Design $U_{\mathrm{lr},0}$	Design U_{sr}	Design $U_{\mathrm{lr},+}$
Design frequency (MHz)	300	300	300
Total number of pipes	96	48	42
Logic (LUTs & FFs)	51.9%	63.3%	62.8%
DSPs	87.2%	55.4%	83.5%
On-chip Mem	27.2%	25.8%	38.4%

- DSP limited in two kernels and Logic limited in one kernel
- Adapting the (8,4,1) ratio

Speedup using multiple FPGAs



Conclusion

- Knowledge of the target device is mandatory for an efficient design
- Great importance of variables size for more ressources → better performances
- An efficient design is about balance
- Our FPGA implementation showed better performance per watt than a GPU of similar transistor size and even better against skylake CPU
- Multiple FPGAs performance bottlenecked by old interconnect technology but achieved nonetheless

Thank you for your attention