XPU General- Purpose Accelerator (XPU-GPA)

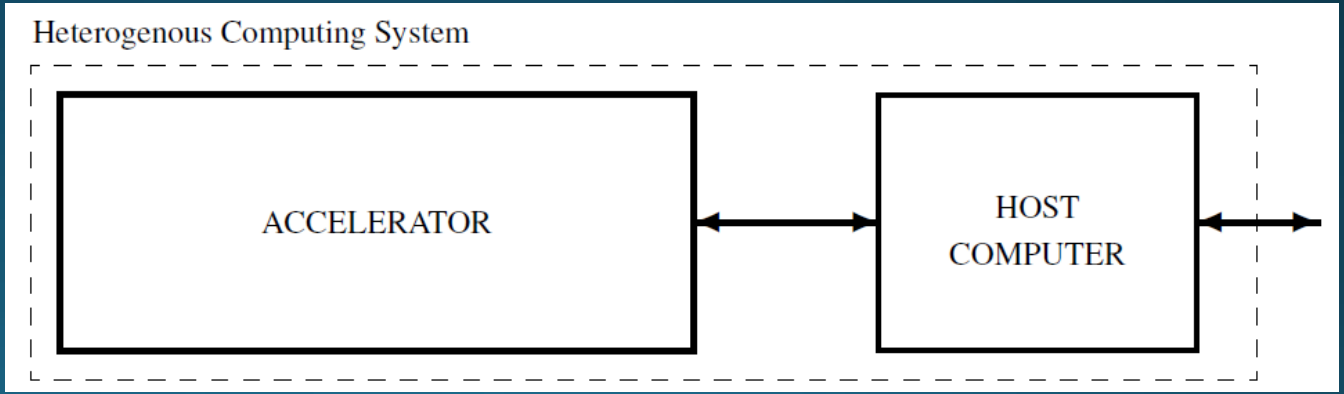
UNSTPB - Politehnica Bucharest / Faculty of Electronics
ETTI-DCAE-ARH group

Speaker: Calin Bira (calin.bira@upb.ro / calin.bira@cern.ch)

Jan.2024

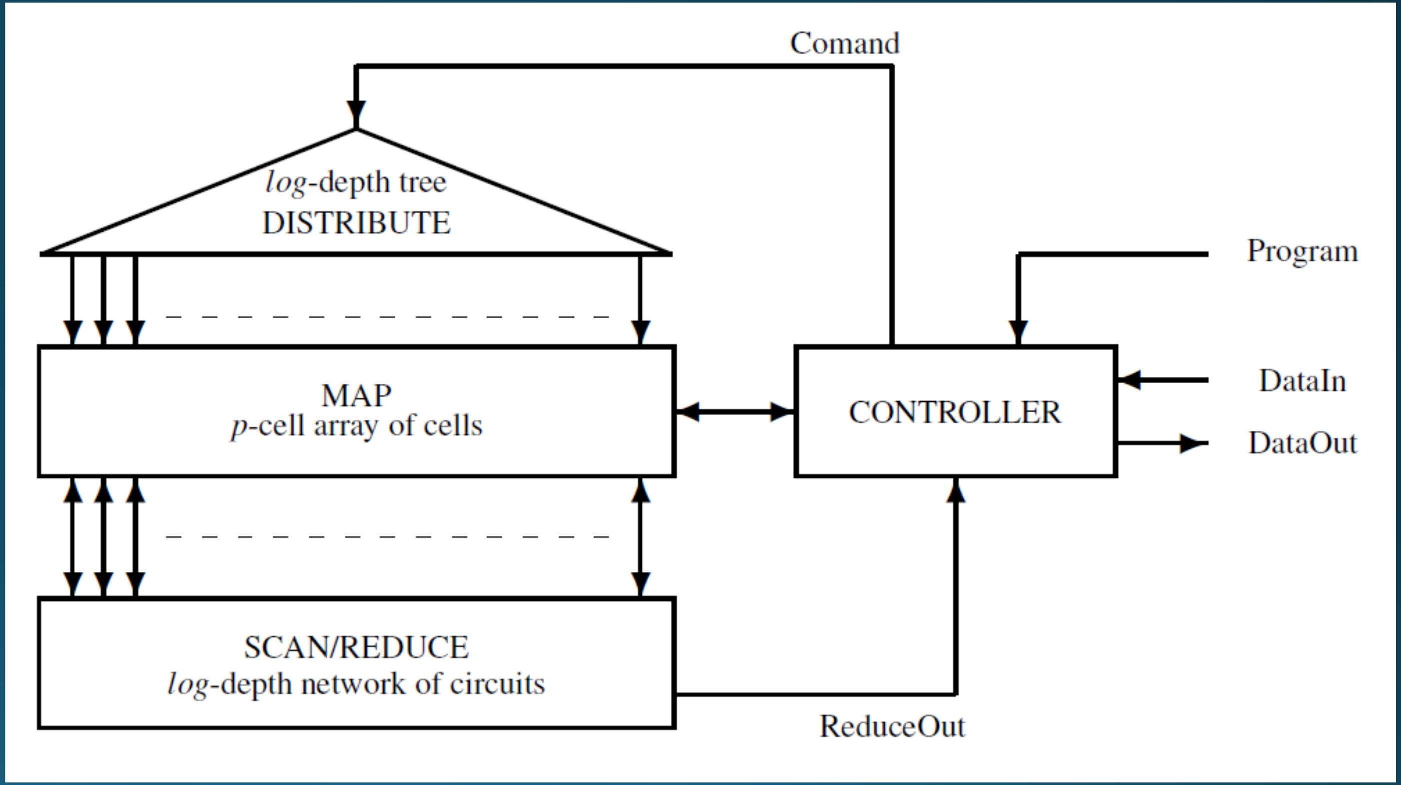
Heterogeneous Computing System

- Complex computation runs on HOST: a mono- or multi-core computation structure (ARM, RISC V, …)
- Intense computation runs on ACCELERATOR: a many-core computation structure
- **ACCELERATOR is seen by the HOST as a hardware library of functions**



XPU GENERAL PURPOSE ACCELERATOR (XPU-GPA)

- MAP: linear array of execution cells with big register files
- CONTROLLER: custom micro-computer used to issue in each cycle a command for MAP
- DISTRIBUTE: pipelined *log*-depth distribution network
- SCAN/REDUCE: pipelined *log*-depth circuit performing reduce functions (add, min, max, …) and scan functions (add prefixes, permute, …)



Architectural acceleration

- Test configuration for GEMM of $N \times N$ matrices :
  - HOST: ARM single-core
  - ACCELERATOR: our MapScanReduce accelerator with $\boldsymbol{p = N}$ cells
- Architectural acceleration (A): acceleration with HOST and ACCELERATOR running at the same frequency with a x86 mono-core engine
  - $T_{Multiply} + T_{Transfer} = ((2p^2 + p\log_2 p + 9p + 5) + (1.5p^2 + 17p + 6)$ *clockCycles*

    (validated by measurement on GPA simulator's clock counter)
  - $T_{multiply+Transfer} = 22 \times N^3$ *clockCycles* (measured running on x86 mono-core)

A => 6.28×p

1024×1024 Matrix Multiplication for ML at 7 nm

- On GPU Nvidia's A100: execution time 0.4ms,
  - on 846 mm^2, with 6912 cells, in 7nm, 1.275 GHz, ~150W, Memory Bus: 5120 bits
- On our GPA: execution time 2.9ms,
  - on 40 mm^2 , with 1024 cells, in 7nm, 1.275 GHz, 5.12W, Memory Bus: 128 bits
- #cells(GPU)/#cells(GPA) = 6.75 ≈ 7 ≈ time(GPA)/time(GPU) = 7.25
- Power(GPU)/Power(GPA) = 78 ➔ 11x computation for the same energy
- Area(GPU)/Area(GPA) = 21 ➔ 3x computation for the same area

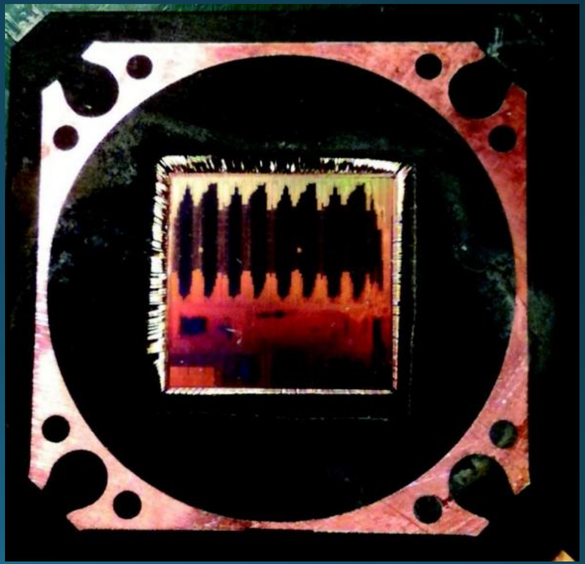
The evaluation is based on simulation and synthesis made in a master thesis using Cadence environment for GPA,

and on https://www.techpowerup.com/gpu-specs/a100-sxm4-80-gb.c3746

https://docs.nvidia.com/deeplearning/performance/dl-performance-matrix-multiplication/index.html for GPU.

Current stage

- Three silicon versions of the accelerator produced in a Silicon Valley start-up (more at: http://users.dcae.pub.ro/~gstefan/2ndLevel/connex.html)
- Working prototype, on FPGA development board, for $p = 128$
- The accelerator is programmed in assembly
- The performance was investigated for a large number of application domains (dense & sparse linear algebra, FFT, molecular dynamic, automotive, …)

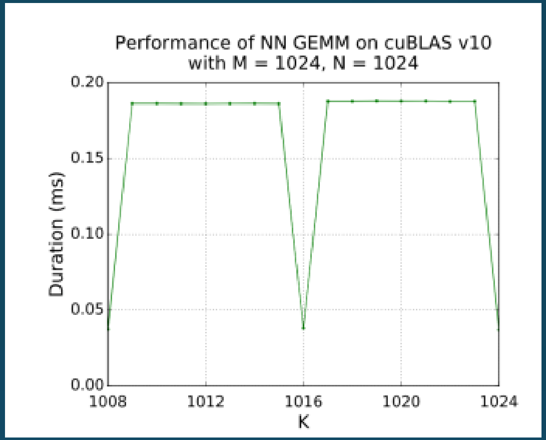


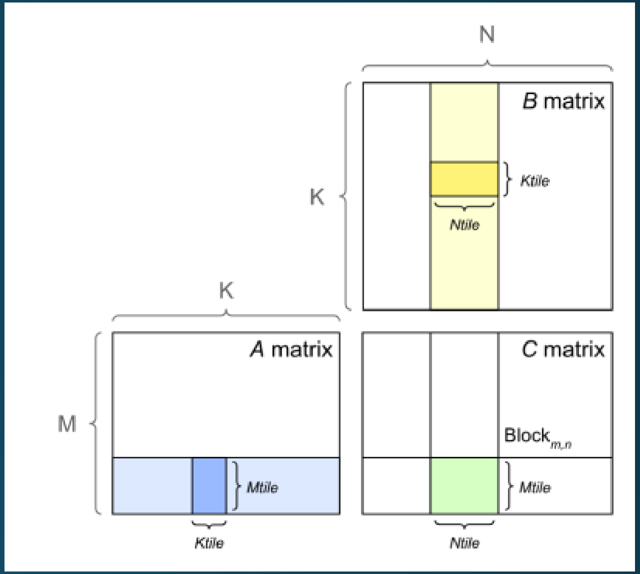
Application Domains

- Artificial Intelligence
- Automotive
- Robotics
- Bio-Informatics
- Security
- Digital Signal Processing

Evaluations against NVIDIA GA100

The evaluation for GPA is based on simulation and synthesis using Cadence environment:
• technology node: 7 nm
• area: 40 mm 2
• number of cells: 1024
• clock frequency: 1.275 GHz
• power: 5.12 W
• memory bus: 128 bis





while for GA100 GPU we used the spec [6]:
• technology node: 7 nm
• area: 846 mm 2
• number of cells: 6912
• clock frequency: 1.275 GHz
• power: 400 W
• memory bus: 5120 bits
For dense matrix multiplication on GA100 the information is provided by [5]

According to figures, the matrix multiplication time for M=K=N on GA100 GPU is t_GPU (1024) = 0.4ms
According to sim on a 1024 GPA the execution time for multiplying 1024 × 1024 matrices is t_GPA (1024) = 2.9ms.

Recap for: 1024×1024 Matrix Multiplication for ML at 7 nm

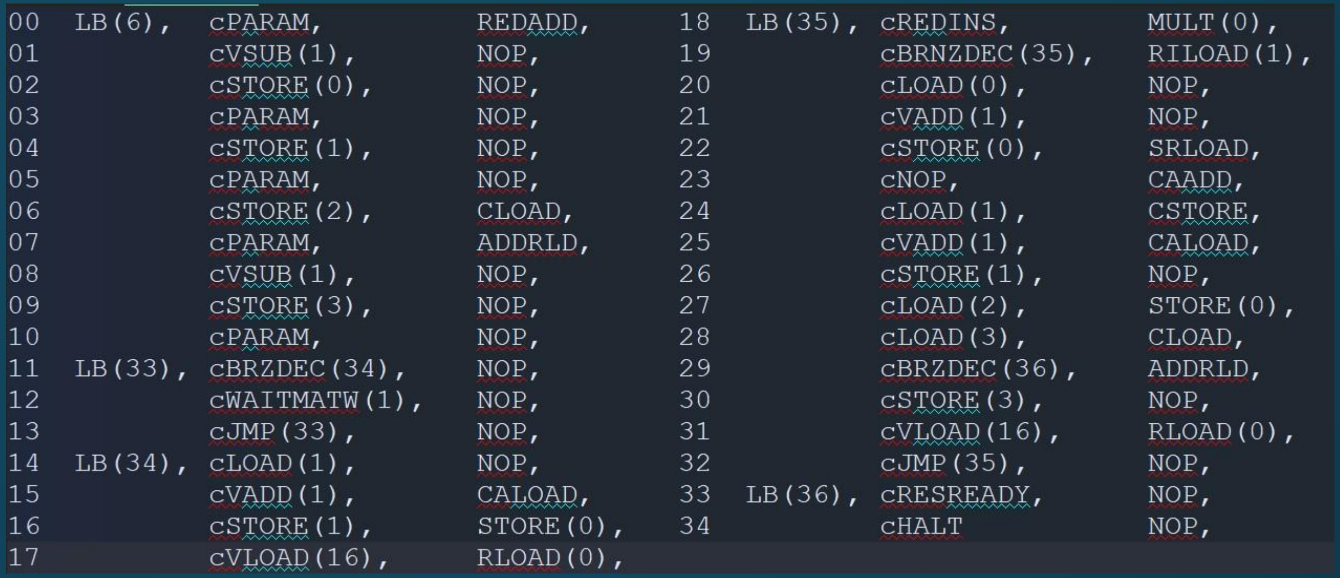
GPU Nvidia’s A100 vs our GPA:

◈ #cells(GPU)/#cells(GPA) = 6.75 ≈ 7 ≈ time(GPA)/time(GPU) = 7.25

◈ Power(GPU)/Power(GPA) = 78 => 11x computation for the same energy

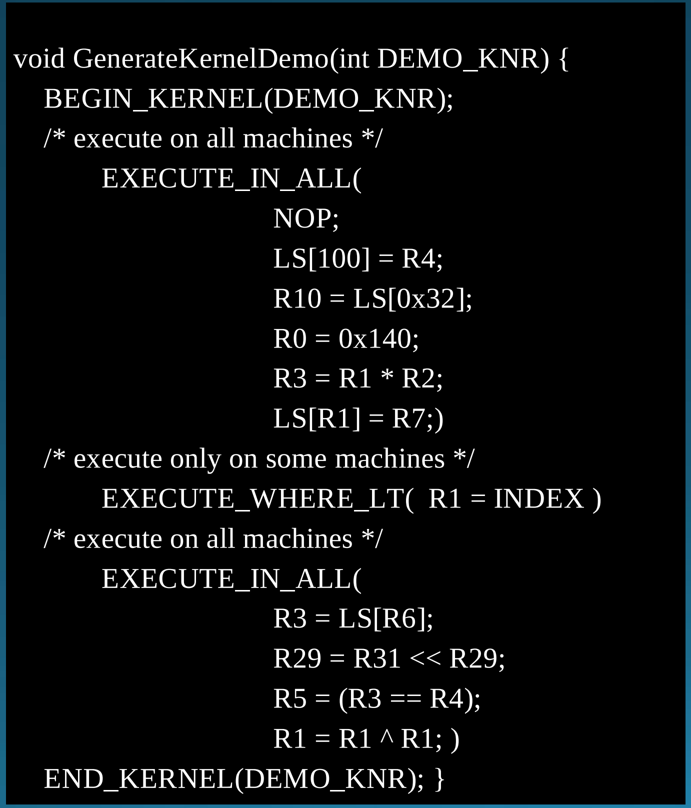
◈ Area(GPU)/Area(GPA) = 21 => 3x computation for the same area

Coding the GPA:
- by coding explicitly the controller and the array, with library of functions:



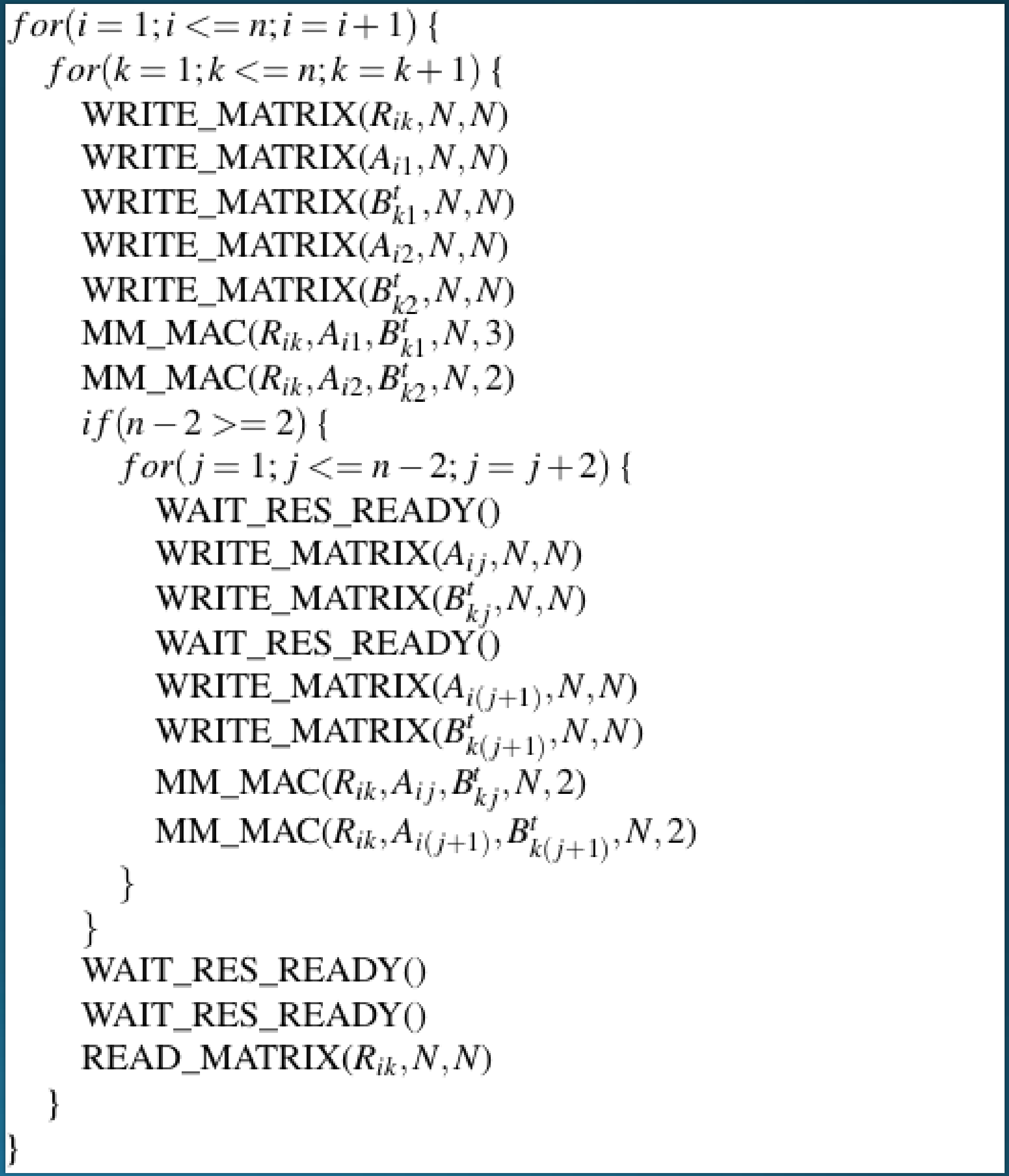
```
00  LB(6),   cPARAM,           REDADD,
01           cVSUB(1),         NOP,
02           cSTORE(0),        NOP,
03           cPARAM,           NOP,
04           cSTORE(1),        NOP,
05           cPARAM,           NOP,
06           cSTORE(2),        CLOAD,
07           cPARAM,           ADDRLD,
08           cVSUB(1),         NOP,
09           cSTORE(3),        NOP,
10           cPARAM,           NOP,
11  LB(33),  cBRZDEC(34),      NOP,
12           cWAITMATW(1),     NOP,
13           cJMP(33),         NOP,
14  LB(34),  cLOAD(1),         NOP,
15           cVADD(1),         CALOAD,
16           cSTORE(1),        STORE(0),
17           cVLOAD(16),       RLOAD(0),
18  LB(35),  cREDINS,          MULT(0),
19           cBRNZDEC(35),     RILOAD(1),
20           cLOAD(0),         NOP,
21           cVADD(1),         NOP,
22           cSTORE(0),        SRLOAD,
23           cNOP,             CAADD,
24           cLOAD(1),         CSTORE,
25           cVADD(1),         CALOAD,
26           cSTORE(1),        NOP,
27           cLOAD(2),         STORE(0),
28           cLOAD(3),         CLOAD,
29           cBRZDEC(36),      ADDRLD,
30           cSTORE(3),        NOP,
31           cVLOAD(16),       RLOAD(0),
32           cJMP(35),         NOP,
33  LB(36),  cRESREADY,        NOP,
34           cHALT             NOP,
```

- by coding directly using source-code based C++ classes that implements a 2-pass assembler



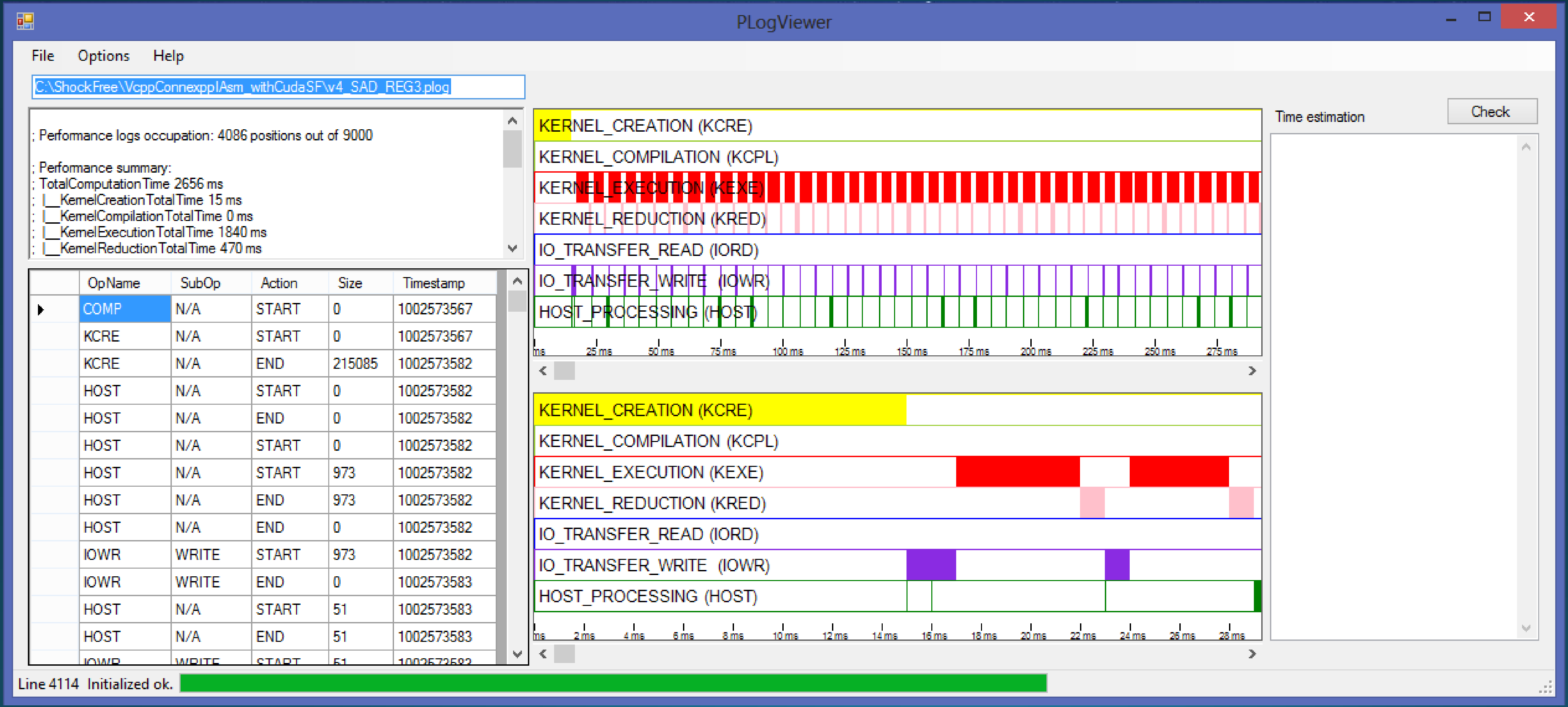
```
void GenerateKernelDemo(int DEMO_KNR) {
   BEGIN_KERNEL(DEMO_KNR);
   /* execute on all machines */
         EXECUTE_IN_ALL(
                          NOP;
                          LS[100] = R4;
                          R10 = LS[0x32];
                          R0 = 0x140;
                          R3 = R1 * R2;
                          LS[R1] = R7;)
   /* execute only on some machines */
         EXECUTE_WHERE_LT(  R1 = INDEX )
   /* execute on all machines */
         EXECUTE_IN_ALL(
                          R3 = LS[R6];
                          R29 = R31 << R29;
                          R5 = (R3 == R4);
                          R1 = R1 ^ R1; )
   END_KERNEL(DEMO_KNR); }
```

For GEMM, when using large matrices, one will have to use an intermediate layer of software to split the task and merge the results of GEMM-ing smaller matrices:



```
$for(i = 1; i <= n; i = i + 1)$ {
  $for(k = 1; k <= n; k = k + 1)$ {
    WRITE_MATRIX($R_{ik}, N, N$)
    WRITE_MATRIX($A_{i1}, N, N$)
    WRITE_MATRIX($B^t_{k1}, N, N$)
    WRITE_MATRIX($A_{i2}, N, N$)
    WRITE_MATRIX($B^t_{k2}, N, N$)
    MM_MAC($R_{ik}, A_{i1}, B^t_{k1}, N, 3$)
    MM_MAC($R_{ik}, A_{i2}, B^t_{k2}, N, 2$)
    $if(n - 2 >= 2)$ {
      $for(j = 1; j <= n - 2; j = j + 2)$ {
        WAIT_RES_READY()
        WRITE_MATRIX($A_{ij}, N, N$)
        WRITE_MATRIX($B^t_{kj}, N, N$)
        WAIT_RES_READY()
        WRITE_MATRIX($A_{i(j+1)}, N, N$)
        WRITE_MATRIX($B^t_{k(j+1)}, N, N$)
        MM_MAC($R_{ik}, A_{ij}, B^t_{kj}, N, 2$)
        MM_MAC($R_{ik}, A_{i(j+1)}, B^t_{k(j+1)}, N, 2$)
      }
    }
    WAIT_RES_READY()
    WAIT_RES_READY()
    READ_MATRIX($R_{ik}, N, N$)
  }
}
```

Tools for profiling the code/app


PLogViewer
File
Options
Help
Time estimation
Check
KERNEL_CREATION (KCRE)
KERNEL_COMPILATION (KCPL)
KERNEL_EXECUTION (KEXE)
KERNEL_REDUCTION (KRED)
IO_TRANSFER_READ (IORD)
IO_TRANSFER_WRITE (IOWR)
HOST_PROCESSING (HOST)
ms 25 ms 50 ms 75 ms 100 ms 125 ms 150 ms 175 ms 200 ms 225 ms 250 ms 275 ms
ms 2 ms 4 ms 6 ms 8 ms 10 ms 12 ms 14 ms 16 ms 18 ms 20 ms 22 ms 24 ms 26 ms 28 ms

C:\ShockFree\VcppConnexpplAsm_withCudaSF\v4_SAD_REG3.plog

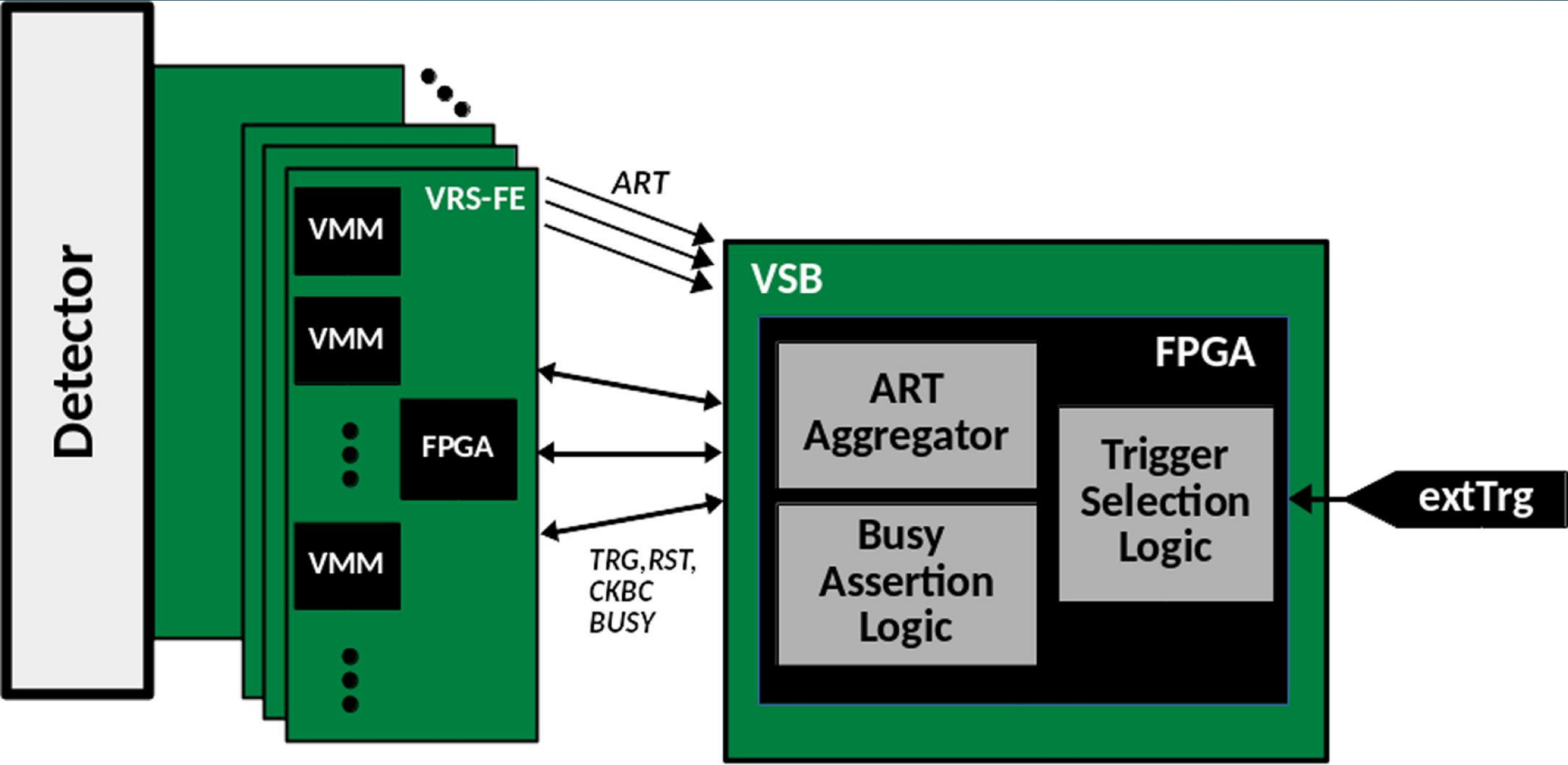
```
; Performance logs occupation: 4086 positions out of 9000

; Performance summary:
; TotalComputationTime 2656 ms
; |__KernelCreationTotalTime 15 ms
; |__KernelCompilationTotalTime 0 ms
; |__KernelExecutionTotalTime 1840 ms
; |__KernelReductionTotalTime 470 ms
```

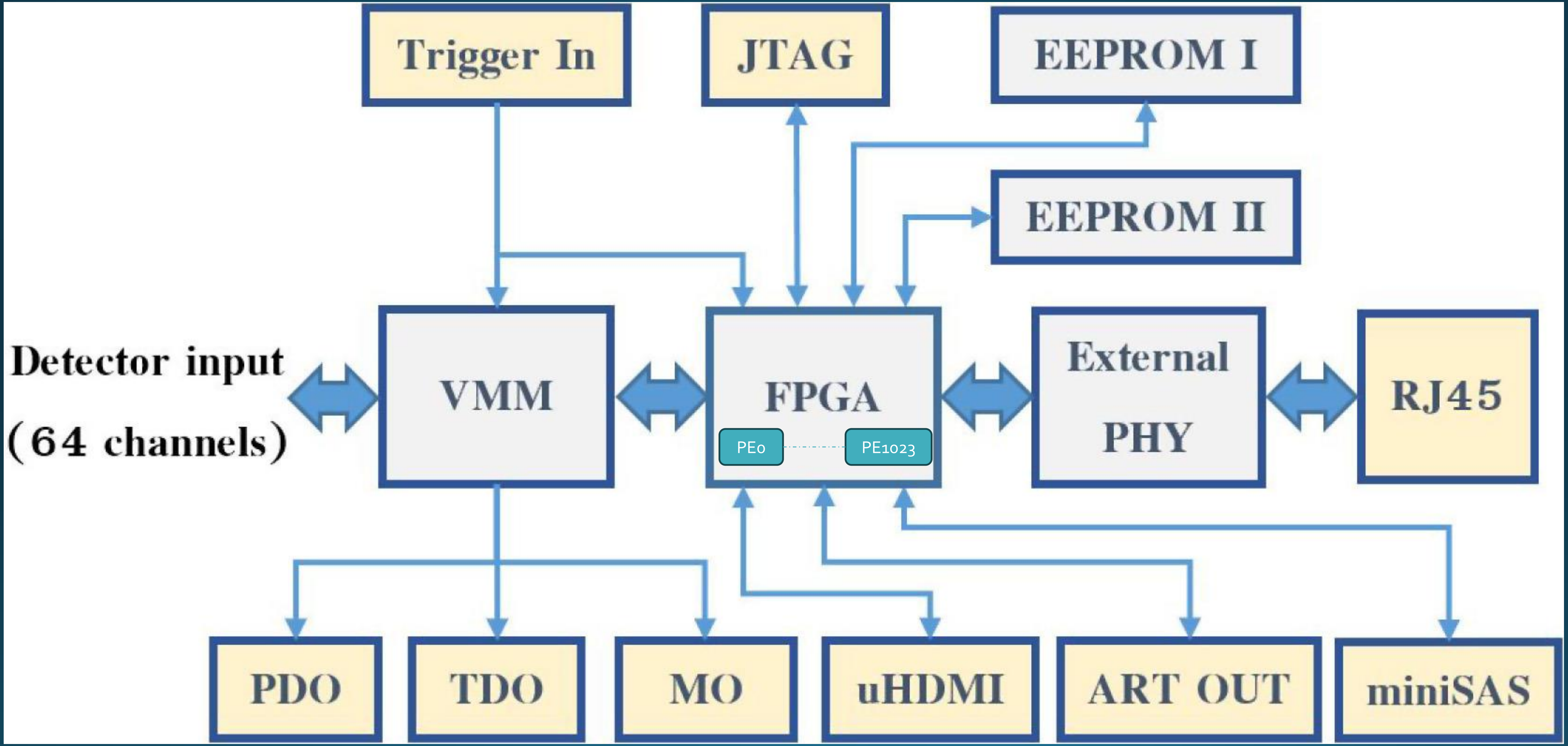
	OpName	SubOp	Action	Size	Timestamp
▶	COMP	N/A	START	0	1002573567
	KCRE	N/A	START	0	1002573567
	KCRE	N/A	END	215085	1002573582
	HOST	N/A	START	0	1002573582
	HOST	N/A	END	0	1002573582
	HOST	N/A	START	0	1002573582
	HOST	N/A	START	973	1002573582
	HOST	N/A	END	973	1002573582
	HOST	N/A	END	0	1002573582
	IOWR	WRITE	START	973	1002573582
	IOWR	WRITE	END	0	1002573583
	HOST	N/A	START	51	1002573583
	HOST	N/A	END	51	1002573583
	IOWR	WRITE	START	51	1002573583

Line 4114 Initialized ok.

Trigger processor chain



Expected usage in ATLAS: processing 1000+ VMMs output



Advantage: software-programming of PEs, without resynthesis

The ARH research team

Prof. Gheorghe Ștefan

Asoc.Prof.
Călin Bîră

Assoc. Prof.
Radu Hobincu

Asst.Prof.
George-Vlăduț
Popescu

SR-III Marius
Stoian

PhD.stud. Costin-
Emanuel Vasile

++
PhD.stud. Mihai Antonescu
PhD.stud. Andrei Simion

Stud. Vlad-Gabriel Serbu
Stud. Andrei Haiducescu
++

Msc. Andrei
Alexandru
Ulmămei

Q & A ?

Backup Slides

References

[1] Mihaela Malit , a, Gheorghe Stefan, Dominique Thiébaut (2007) Not Multi-, but Many-Core: Designing Integral Parallel Architectures for Embedded Computation, ACM SIGARCH Computer Architecture News, 35(5)32:38, Special issue: ALPS ’07 - Advanced low power systems; communication at International Workshop on Advanced Low Power Systems held in conjunction with 21st International Conference on Supercomputing June 17, 2007 Seattle, WA, USA.

[2] Gheorghe Ştefan (2006) The CA1024: A Massively Parallel Processor for Cost-Effective HDTV, SPRING PROCESSOR FORUM JAPAN, June 8-9, 2006, Tokyo.

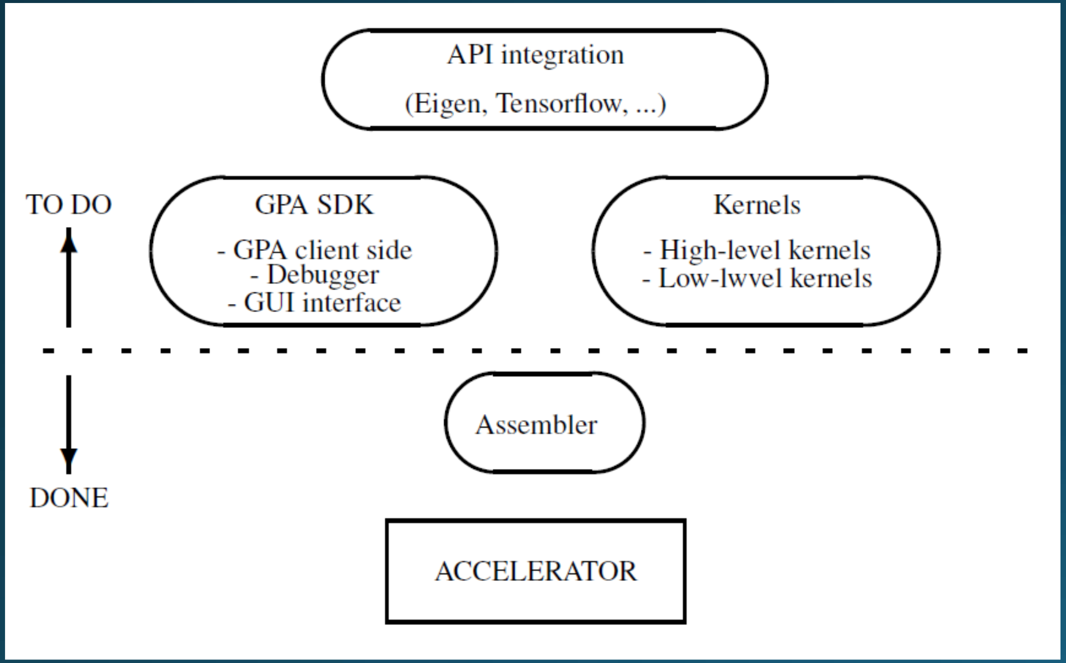
[3] Gheorghe Ştefan, Anand Sheel, Bogdan Mîţu, Tom Thomson, Dan Tomescu (2006) The CA1024: A Fully Programmable System-On-Chip for Cost-Effective HDTV Media Processing, Hot Chips: A Symposium on High Performance Chips, Memorial Auditorium, Stanford University, August 20 to 22, 2006.

[4] Gheorghe M. S , tefan, Mihaela Malit , a (2014) Can One-Chip Parallel Computing Be Liberated From Ad HocSolutions? A Computation Model Based Approach and Its Implementation, 18th International Conference on Circuits, Systems, Communications and Computers , Santorini Island, Greece, pp 582-597. http://www. inase.org/library/2014/santorini/bypaper/COMPUTERS/COMPUTERS2-42.pdf

[5] User’s Guide — NVIDIA Docs (2023) Matrix Multiplication Background, https://docs.nvidia. com/deeplearning/performance/pdf/Matrix-Multiplication-Background-User-Guide.pdf

[6] NVIDIA GA100 (2023) NVIDIA GA100, https://www.techpowerup.com/gpu-specs/nvidia-ga100.g931

The project



- Stage 0: we already have:
  - ACCELERATOR in FPGA
  - Assembler language
- **Stage 1: with 10 people in 12 months :**
  - **GPA SDK**
  - **the frame for API integration**
  - **partially Kernels up to the level at which system performance can be demonstrated (ONNX)**
- Stage 2: fully developed Kernels

Subset of kernel library of functions used by the host computer

- START_CC : start cycles counter
- STOP_CC : stop cycles counter
- SEND_INT : send interrupt and cycles counter
- MM_EWO : element-wise operation on matrix
- SM_MULT : scalar-matrix multiplication
- MM_MULT : matrix-matrix multiplication
- MM_MAC : matrix-matrix multiplication & accumulate
- WRITE_MATRIX : write matrix
- READ_MATRIX : read matrix
- WAIT_RES_READY : wait for result ready

Evaluations on FPGA

- Matrix-Matrix multiplication on GPA with 16 cells
  - 16x16 matrices: 1489 clock cycles
  - 32x32 matrices: 9826 clock cycles
  - 64x64 matrices: 70190 clock cycles
  - 128x128 matrices: 527806 clock cycles
- 128x128 Matrix-Matrix MULT on GPA of various size
  - 16 number of cells: 527806 clock cycles
  - 32 number of cells: 242253 clock cycles
  - 64 number of cells: 127082 clock cycles
  - 128 number of cells: 77204 clock cycles