Prospects for Low-power Acceleration of HPC Workloads in EuroExa: FPGA Acceleration of a Numerical Weather Forecast Code

Mike Ashworth, Graham Riley, Andrew Attwood and John Mawer
Advanced Processor Technologies Group
School of Computer Science,
University of Manchester, United Kingdom
mike.ashworth.compsci@manchester.ac.uk

Project ID: 754337

Project outline

Horizon 2020 FETHPC-01-2016:

Co-design of HPC systems and applications

EuroExa started 1st Sep 2017, runs for 3½ years

16 Partners, 8 countries, €20M

Builds on previous projects, esp. ExaNoDe, ExaNeSt, EcoScale

Aim: design, build, test and evaluate an Exascale prototype

Architecture based on ARM CPUs with FPGA accelerators

Three testbed systems: #3 will deliver 2-3 Pflop/s peak

Scalable to 400 Pflop/s at high Gflop/s/W

Low-power design goal to target realistic Exascale system

Architecture evolves in response to application requirements
= co-design

@euroexa

euroexa.eu

Kick-off meeting 4th-5th Sep 2017, Barcelona

Wide range of apps, incl. weather forecasting, lattice Boltzmann, multiphysics, astrophysics, astronomy data processing, quantum chemistry, life sciences and bioinformatics

Motivation

- FPGAs offer large (OsOM) gains in performance/W
- Also gains in performance/{$£€฿}
- Major corporations are using FPGAs in datacentres for cloud services, analytics, communication, etc.
- H/W traditionally led by Xilinx (ARM CPU + FPGA single chip)
- Intel's acquisition of Altera led to Heterogeneous Architecture Research Platform (HARP) (also single chip)
- Predictions: up to 30% of datacenter servers will have FPGAs by 2020

LFRic Weather and Climate Model

Brand new weather and climate model: LFRic

named after Lewis Fry Richardson (1881-1953)

- Dynamics from the GungHo project 2011-2015
- Scalability – globally uniform grid (no poles)
- Speed – maintain performance at high & low resolution and for high & low core counts
- Accuracy – need to maintain standing of the model
- Separation of Concerns – PSyClone generated layer for automated targeting of architectures
- Operational weather forecasts around 2022 – anniversary of Richardson (1922)

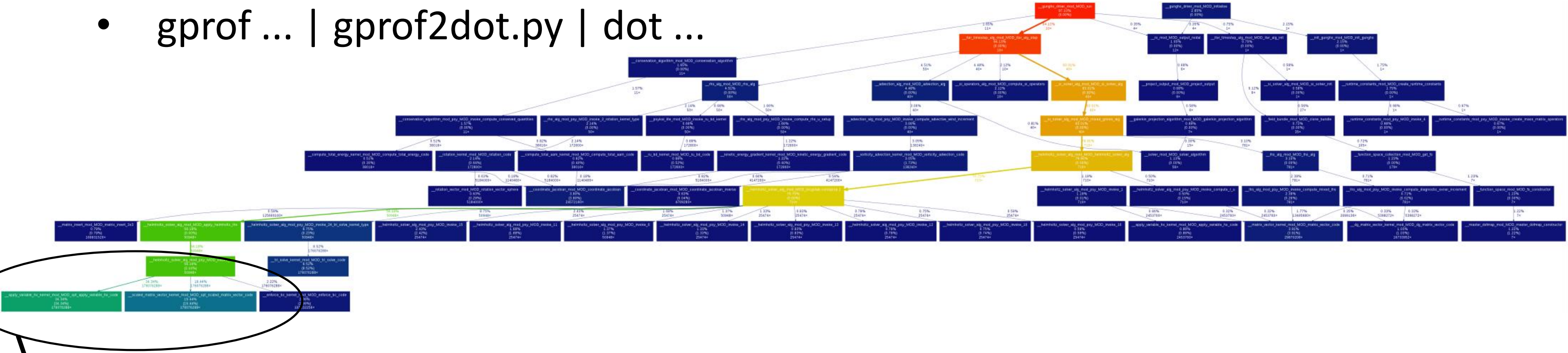
Globally
Uniform
Next
Generation
Highly
Optimized

工合

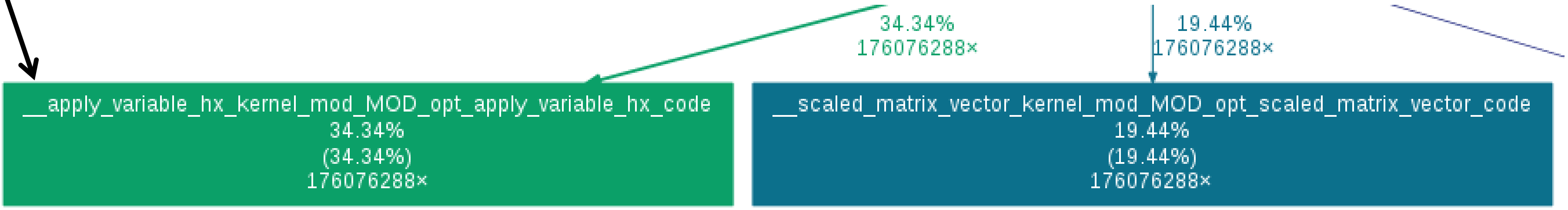
"Working together harmoniously"

LFRic profile & call graph

- Baroclinic performance benchmark case
- gprof ... | gprof2dot.py | dot ...

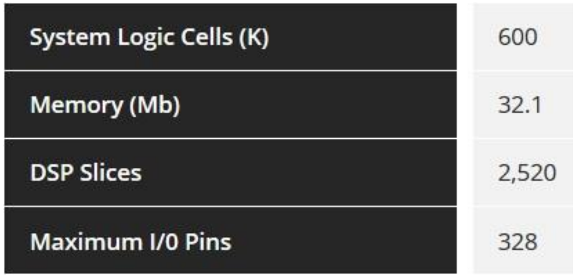


- Two subroutines in the Helmholtz solver use 54% of runtime
- Most is in matrix-vector products within a loop over vertical levels

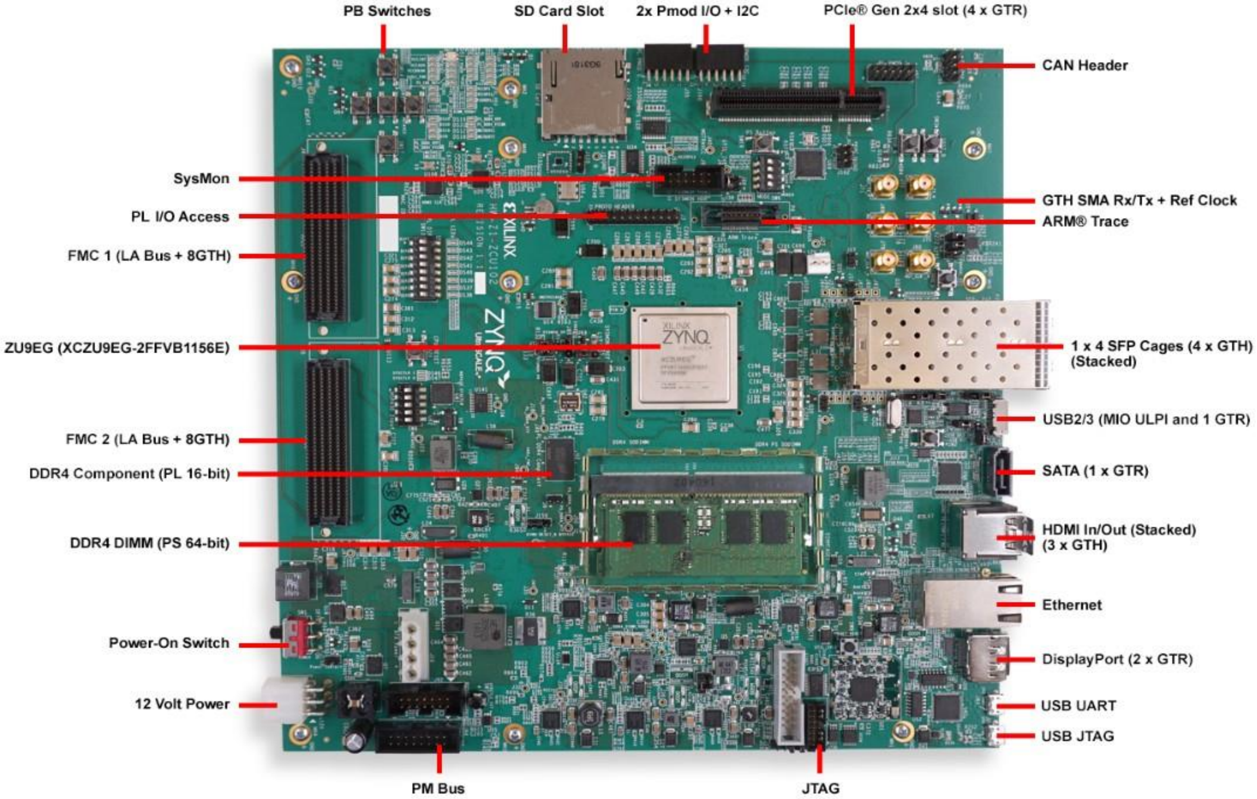

34.34%
176076288×
19.44%
176076288×
__apply_variable_hx_kernel_mod_MOD_opt_apply_variable_hx_code
34.34%
(34.34%)
176076288×
__scaled_matrix_vector_kernel_mod_MOD_opt_scaled_matrix_vector_code
19.44%
(19.44%)
176076288×

Zynq UltraScale+ ZCU102 Evaluation Platform

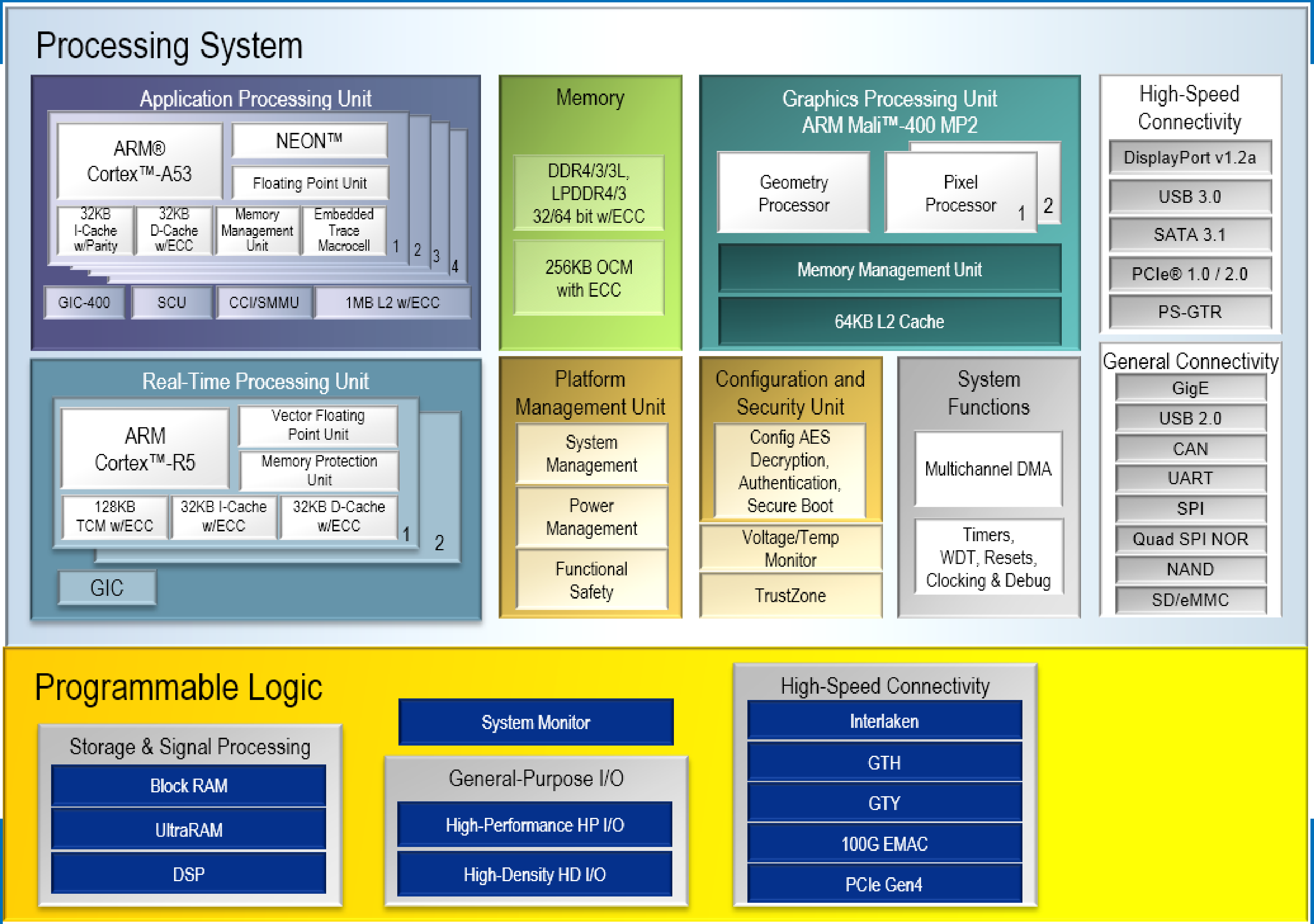
- ARM Cortex A53 quad-core CPU 1.2 GHz
- Dual-core Cortex-R5 real-time processor
- Mali-400 MP2 GPU
- Zynq UltraScale XCZU9EG-2FFVB1156 FPGA



System Logic Cells (K)	600
Memory (Mb)	32.1
DSP Slices	2,520
Maximum I/O Pins	328



Zynq UltraScale+ MPSoC EG



Processing System

Application Processing Unit

ARM® Cortex™-A53

NEON™

Floating Point Unit

32KB I-Cache w/Parity

32KB D-Cache w/ECC

Memory Management Unit

Embedded Trace Macrocell

1 2 3 4

GIC-400

SCU

CCI/SMMU

1MB L2 w/ECC

Memory

DDR4/3/3L, LPDDR4/3 32/64 bit w/ECC

256KB OCM with ECC

Graphics Processing Unit ARM Mali™-400 MP2

Geometry Processor

Pixel Processor 1 2

Memory Management Unit

64KB L2 Cache

High-Speed Connectivity

DisplayPort v1.2a

USB 3.0

SATA 3.1

PCIe® 1.0 / 2.0

PS-GTR

Real-Time Processing Unit

ARM Cortex™-R5

Vector Floating Point Unit

Memory Protection Unit

128KB TCM w/ECC

32KB I-Cache w/ECC

32KB D-Cache w/ECC

1 2

GIC

Platform Management Unit

System Management

Power Management

Functional Safety

Configuration and Security Unit

Config AES Decryption, Authentication, Secure Boot

Voltage/Temp Monitor

TrustZone

System Functions

Multichannel DMA

Timers, WDT, Resets, Clocking & Debug

General Connectivity

GigE

USB 2.0

CAN

UART

SPI

Quad SPI NOR

NAND

SD/eMMC

Programmable Logic

Storage & Signal Processing

Block RAM

UltraRAM

DSP

System Monitor

General-Purpose I/O

High-Performance HP I/O

High-Density HD I/O

High-Speed Connectivity

Interlaken

GTH

GTY

100G EMAC

PCIe Gen4

Range of Programming Models

1. C code with Xilinx Vivado HLS and Vivado Design Suite
2. OmpSs@FPGA directive-based (BSC)
3. MaxJ compiler for Maxeler systems
4. OpenCL code with Xilinx SDAccel
5. OpenStream (Uni Man)

- Options 2-5 being investigated by other members of the project

Starting code for Vivado HLS

```
#define NDF1 8
#define NDF2 6
#define NK 40
#define MVTYPE double
int matvec_8x6x40_vanilla (MVTYPE matrix[NK][NDF2][NDF1],
              MVTYPE x[NDF2][NK], MVTYPE lhs[NDF1][NK]) {
  int df,j,k;
  for (k=0;k<NK;k++) {
    for (df=0;df<NDF1;df++) {
      lhs[df][k] = 0.0;
      for (j=0;j<NDF2;j++) {
        lhs[df][k] = lhs[df][k]
          + x[j][k]*matrix[k][j][df];
      }
    }
  }
  return 0;
}
```

Notes:

- Data sizes hard-wired for HLS
- Vertical loop k is outer
- Vectors x and lhs are sequential in k (k-last in C)
- Matrix is not (k-first)
- Read-then-write dependence on lhs
- Flops = 2*NK*NDF1*NDF2 = 3840
- Mem refs = 2*flops = 7680 doubles

Optimizations in Vivado HLS

- Make k the inner loop (loop length 40, independent, sequential access)
- Transpose matrix to k-last to ensure sequential memory access
- HLS pragma to unroll inner loops on k (no benefit from hand unrolling)
- HLS pragma to pipeline outer loop on df
- HLS pragma for input and output arguments including
  - num_read_outstanding=8
  - max_read_burst_length=64
- Access input and output arguments by memcpy to local arrays to ensure streaming of loads/stores to/from BRAM (see later)

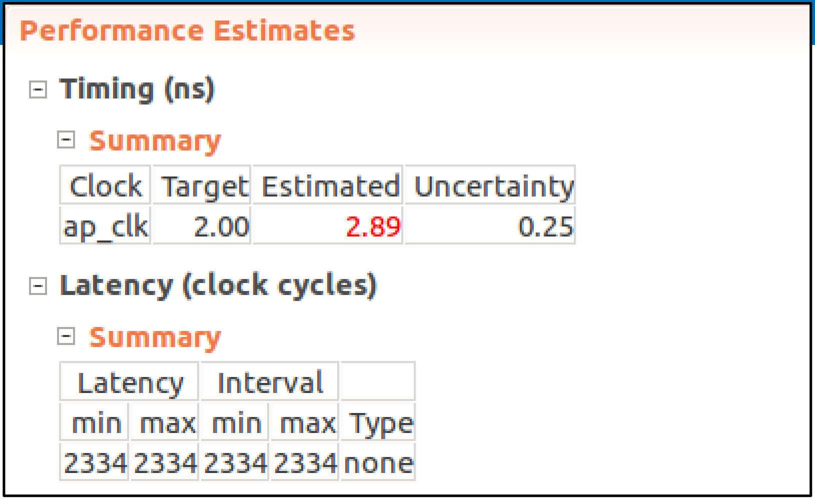
Optimized code in Vivado HLS

```
#pragma HLS INTERFACE m_axi depth=128
  port=matrix offset=slave bundle=bram /
        num_read_outstanding=8 /
        num_write_outstanding=8 /
        max_read_burst_length=64 /
        max_write_burst_length=64
< pragmas for m_axi interfaces for x, lhs
  and s_axilite interface for return>


  int df,j,k;
  MVTYPE ml[NDF2][NK], xl[NDF2][NK],
   ll[NDF1][NK];
  memcpy (xl, x, NDF2*NK*sizeof(MVTYPE));
  for (df=0;df<NDF1;df++) {
#pragma HLS PIPELINE
```

```
    for (k=0;k<NK;k++) {
#pragma HLS UNROLL
       ll[df][k] = 0.0;
     }
     memcpy (ml, matrix+df*NDF2*NK, /
       NDF2*NK*sizeof(MVTYPE));
     for (j=0;j<NDF2;j++) {
       for (k=0;k<NK;k++) {
#pragma HLS UNROLL
            ll[df][k] = ll[df][k]+
   xl[j][k]*ml[j][k];
       }
     }
   }
   memcpy (lhs, ll,
   NDF1*NK*sizeof(MVTYPE));
```

Vivado HLS Performance Estimate



Performance Estimates

⊟ Timing (ns)

⊟ Summary

Clock	Target	Estimated	Uncertainty
ap_clk	2.00	2.89	0.25

⊟ Latency (clock cycles)

⊟ Summary

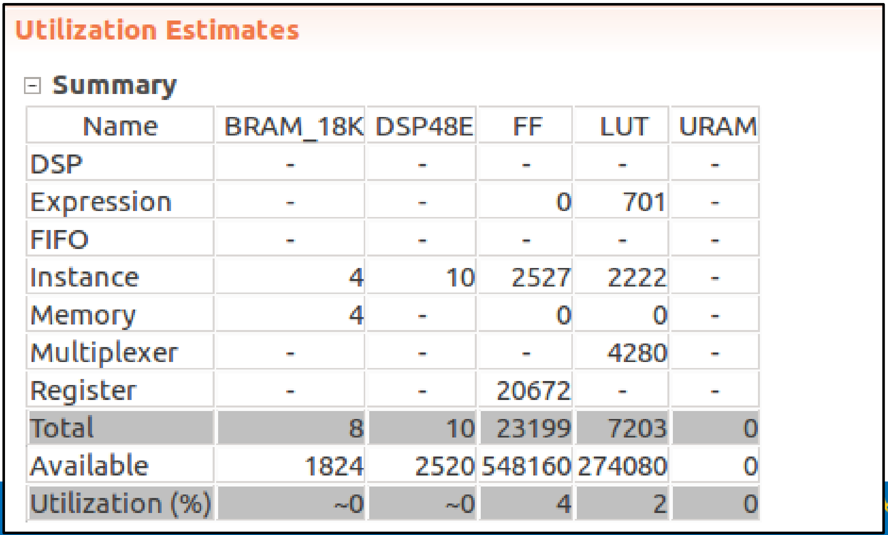
Latency		Interval		
min	max	min	max	Type
2334	2334	2334	2334	none

Performance Estimate:

- Target 2ns clock: design validated at 2.89ns = 346 MHz
- 2334 cycles for 3840 flops = 1.65 flops/cycle
- Overlapped dmul with dadd
- Starting code was 69841 cycles

Utilization Estimate:

- Try to maximize performance while minimizing utilization
- Shows percentage of chip ‘real-estate being utilized

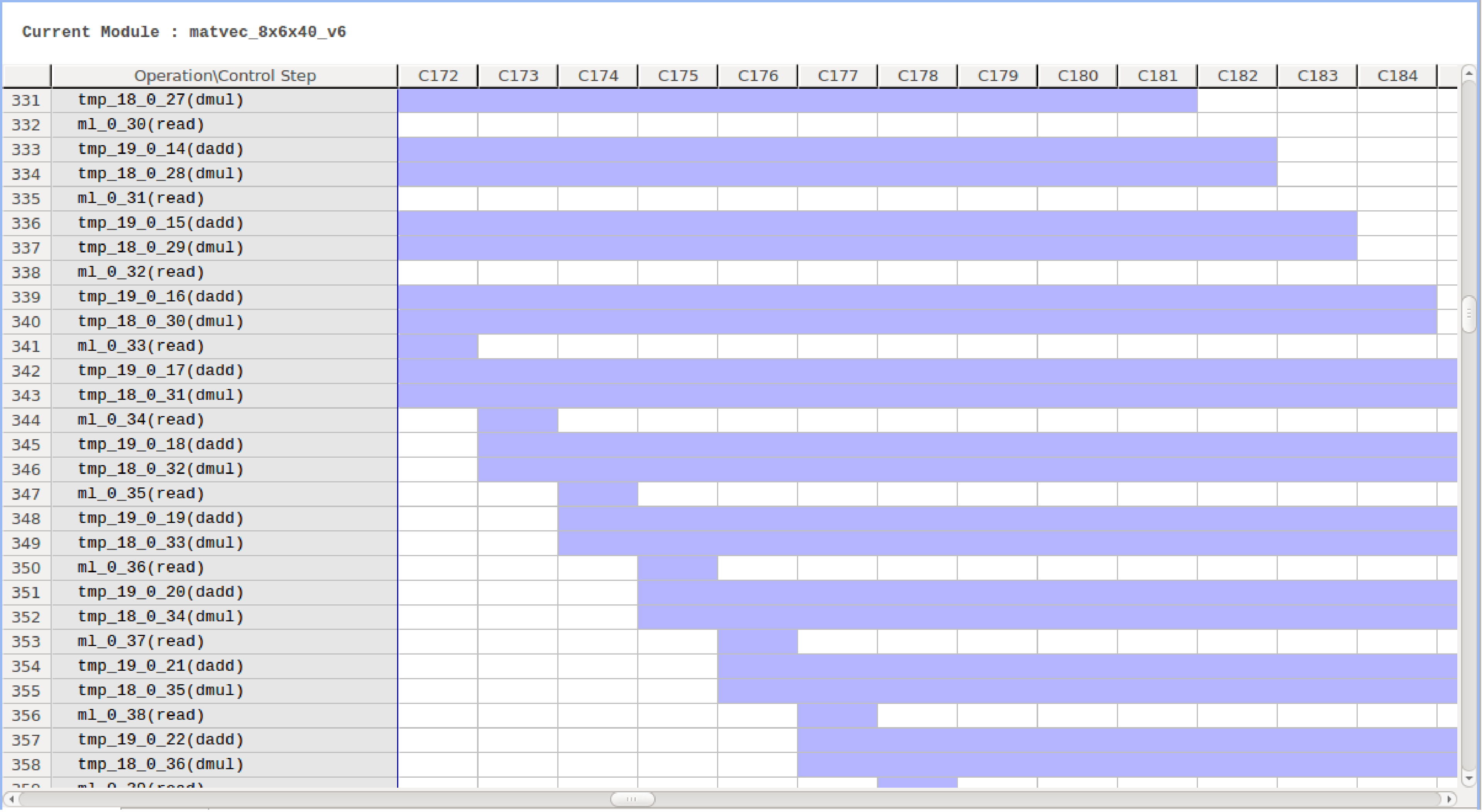


Utilization Estimates

⊟ Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	701	-
FIFO	-	-	-	-	-
Instance	4	10	2527	2222	-
Memory	4	-	0	0	-
Multiplexer	-	-	-	4280	-
Register	-	-	20672	-	-
Total	8	10	23199	7203	0
Available	1824	2520	548160	274080	0
Utilization (%)	~0	~0	4	2	0

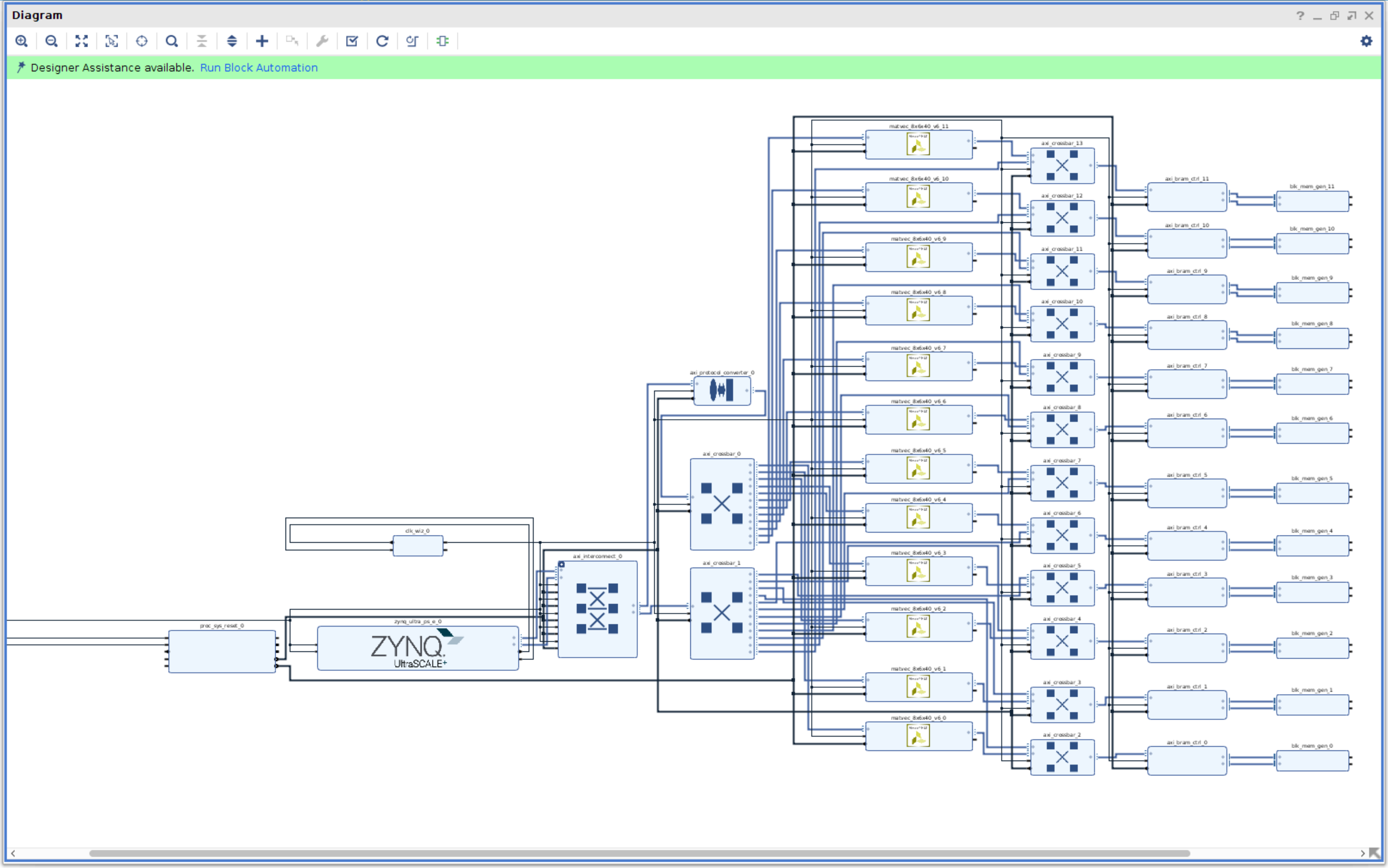
Vivado HLS Performance Timeline



Current Module : matvec_8x6x40_v6

	Operation\Control Step	C172	C173	C174	C175	C176	C177	C178	C179	C180	C181	C182	C183	C184
331	tmp_18_0_27(dmul)	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨			
332	ml_0_30(read)													
333	tmp_19_0_14(dadd)	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨		
334	tmp_18_0_28(dmul)	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨		
335	ml_0_31(read)													
336	tmp_19_0_15(dadd)	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	
337	tmp_18_0_29(dmul)	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	
338	ml_0_32(read)													
339	tmp_19_0_16(dadd)	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
340	tmp_18_0_30(dmul)	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
341	ml_0_33(read)	▨												
342	tmp_19_0_17(dadd)	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
343	tmp_18_0_31(dmul)	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
344	ml_0_34(read)		▨											
345	tmp_19_0_18(dadd)		▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
346	tmp_18_0_32(dmul)		▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
347	ml_0_35(read)			▨										
348	tmp_19_0_19(dadd)			▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
349	tmp_18_0_33(dmul)			▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
350	ml_0_36(read)				▨									
351	tmp_19_0_20(dadd)				▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
352	tmp_18_0_34(dmul)				▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
353	ml_0_37(read)					▨								
354	tmp_19_0_21(dadd)					▨	▨	▨	▨	▨	▨	▨	▨	▨
355	tmp_18_0_35(dmul)					▨	▨	▨	▨	▨	▨	▨	▨	▨
356	ml_0_38(read)						▨							
357	tmp_19_0_22(dadd)						▨	▨	▨	▨	▨	▨	▨	▨
358	tmp_18_0_36(dmul)						▨	▨	▨	▨	▨	▨	▨	▨
359	ml_0_39(read)							▨						

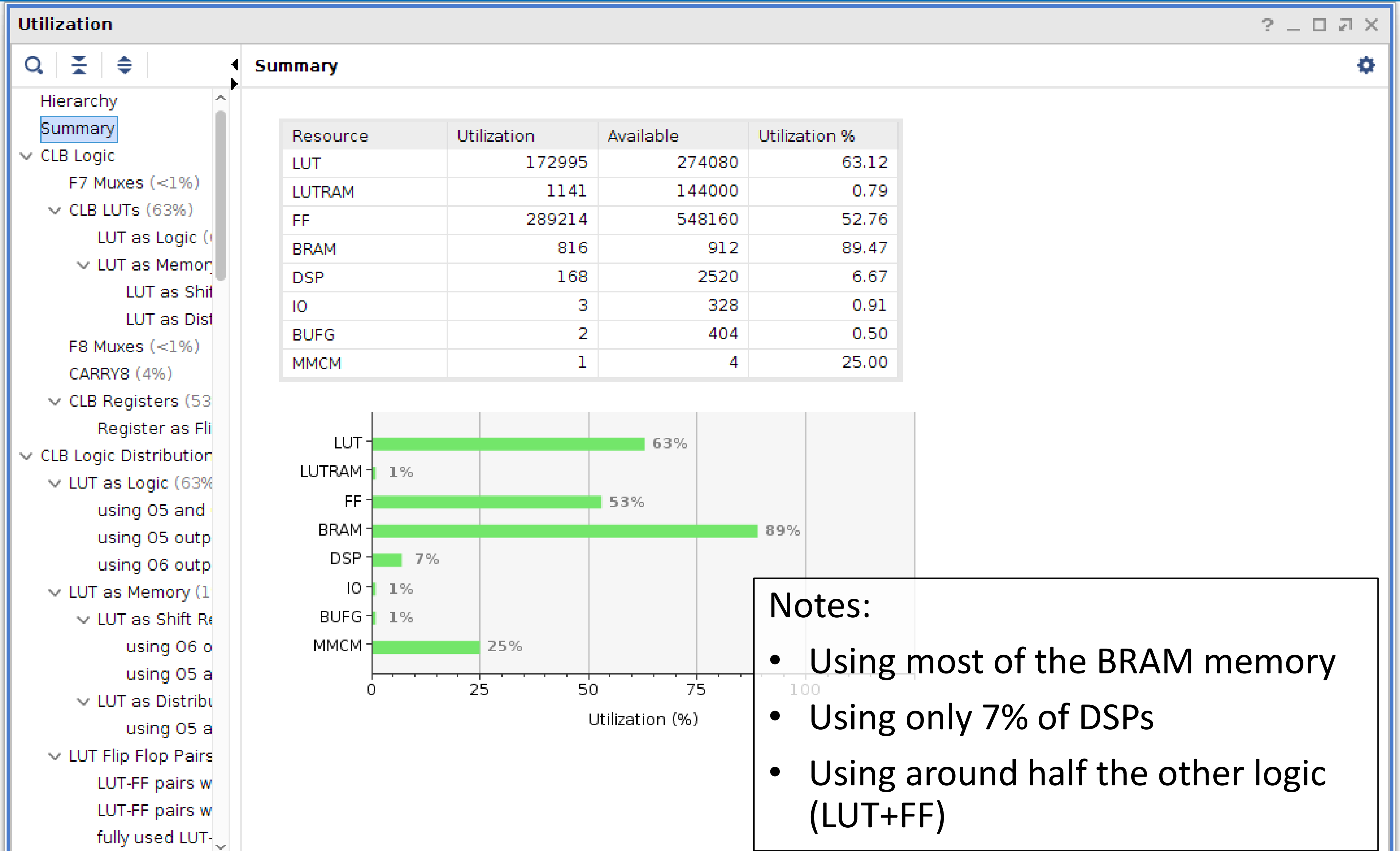
Design with 12 Matrix-Vector Blocks



Designer Assistance available. Run Block Automation

EUROEXA
FUNDED BY THE EUROPEAN UNION

Vivado DS Resource Utilization



Utilization

Hierarchy
Summary
CLB Logic
F7 Muxes (<1%)
CLB LUTs (63%)
LUT as Logic (
LUT as Memor
LUT as Shif
LUT as Dist
F8 Muxes (<1%)
CARRY8 (4%)
CLB Registers (53
Register as Fli
CLB Logic Distribution
LUT as Logic (63%
using O5 and
using O5 outp
using O6 outp
LUT as Memory (1
LUT as Shift Re
using O6 o
using O5 a
LUT as Distribu
using O5 a
LUT Flip Flop Pairs
LUT-FF pairs w
LUT-FF pairs w
fully used LUT-

Summary

Resource	Utilization	Available	Utilization %
LUT	172995	274080	63.12
LUTRAM	1141	144000	0.79
FF	289214	548160	52.76
BRAM	816	912	89.47
DSP	168	2520	6.67
IO	3	328	0.91
BUFG	2	404	0.50
MMCM	1	4	25.00

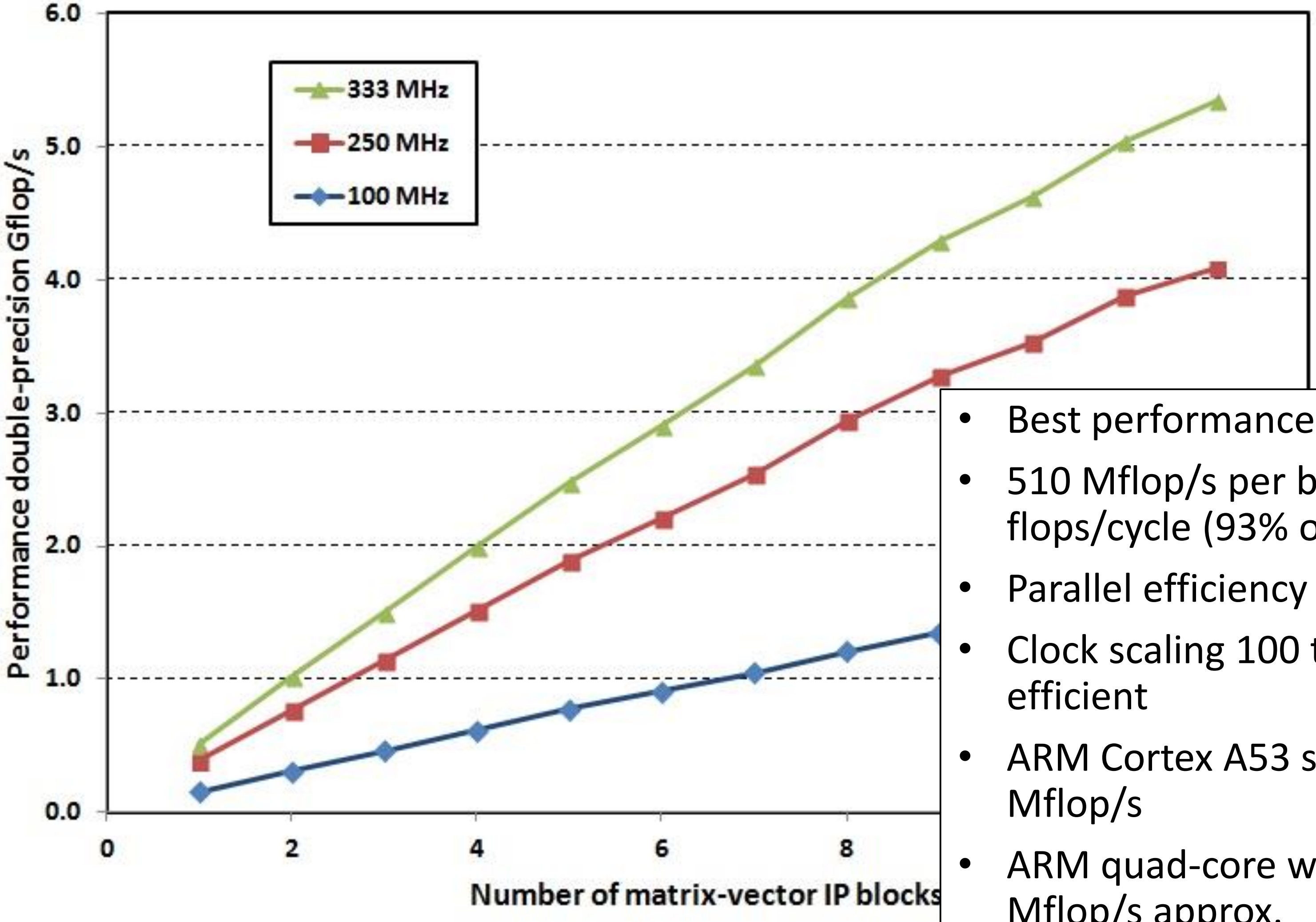
Notes:

- Using most of the BRAM memory
- Using only 7% of DSPs
- Using around half the other logic (LUT+FF)

ARM driver code

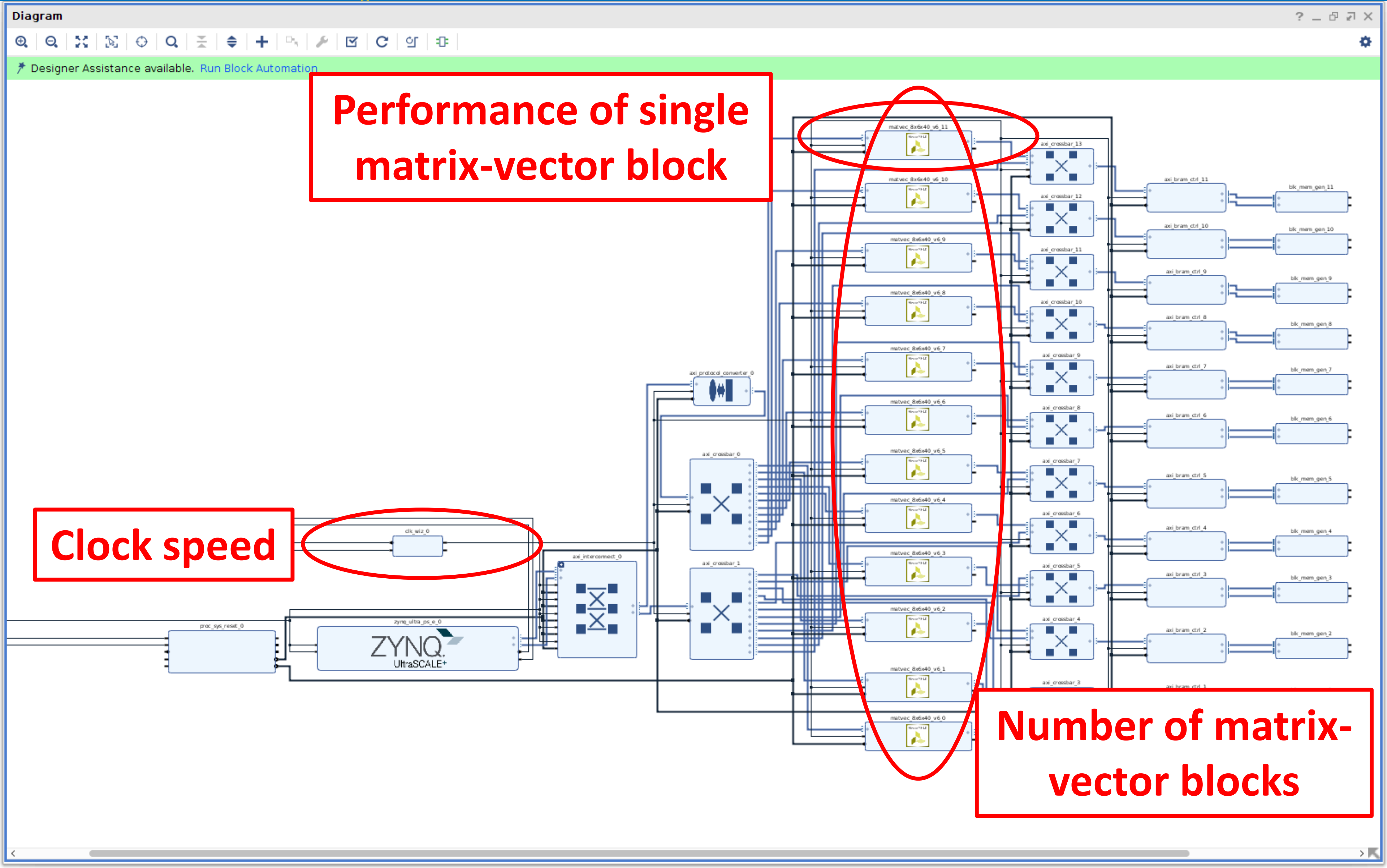
- Setup a two devices /dev/uio0 and /dev/uio1 – two ports on the ZynQ block
- Use mmap to map the FPGA memory into user space
- Assign pointers for each data array to location in user space
- Control loop to divide up the work into 12 “chunks” which will fit into the FPGA BRAM memory (maximum 12 x 256kB = 3MB) (13 columns in this LFRic model)
- For each chunk:
  - Assign work to one of the matrix-vector blocks
  - Copy input data into BRAM
  - Set the control word “registers” for the block
  - Start the block by setting AP_START
  - Wait for block to finish by watching AP_IDLE (opportunity for overlap)
  - Copy output data from BRAM
- In practice we fill 3MB BRAM, then run all 12 matrix-vector blocks, then copy output data back and repeat
- Check correctness and time the code

Results for 12 blocks



- Best performance 5.3 Gflop/s
- 510 Mflop/s per block => 1.53 flops/cycle (93% of HLS estimate)
- Parallel efficiency at 12 IP blocks 87%
- Clock scaling 100 to 333 MHz is 94% efficient
- ARM Cortex A53 single core 177 Mflop/s
- ARM quad-core with OpenMP 615 Mflop/s approx.
- FPGA:ARM quad-core speed-up: 8.6x

Critical Performance Factors



Performance of single matrix-vector block

Clock speed

Number of matrix-vector blocks

LFRic matrix-vector performance comparison

Hardware	Matrix-vector performance (Gflop/s	Peak performance (Gflop/s)	Percentage peak	Price	Power
ZCU102 FPGA	5.3	600	0.9%	$	W
Intel Broadwell E5-2650 v2 2.60GHz 8 cores	9.86	332.8	3.0%	$$$	WWW

- FPGA performance is 54% of Broadwell single socket
- Should be scaled by price & power

Final thoughts

- Matrix-vector (MVM) vs. matrix multiply (MXM)
  - For large N, MVM asymptotically approaches computational intensity (CI) of 0.25 flops/byte
  - MXM has a computational intensity of N/12, so even for small matrices (12x12) CI is one flop/byte
  - **Matrix-vector is much harder than matrix-multiply**
- Performance/price and performance/power
  - “GPU vs FPGA Performance Comparison”, Berton White Paper
  - GPU: 0.07-0.12 vs. FPGA: 0.23 €/Gflop/s/W
  - GPU: 20 vs. FPGA: 70 Gflops/W
  - **FPGAs have a large benefit in power efficiency**

Summary

We have

- Used Vivado HLS to develop a matrix-vector kernel which runs on the UltraScale+ FPGA at 5.3 double precision Gflop/s (single precision: similar performance, 63% resources)

Issues

- Timing constraints in the Vivado design prevent larger numbers of blocks and higher clock speeds
- However, performance against Xeon is compelling

Future work

- Generate an IP block and driver for the LFRic code: apply_variable_hx_kernel_code (done; HLS 1.75 flops/cycle)
- Exploit MPI within LFRic to run across multiple nodes and multiple FPGAs (done trivially with the matrix-vector kernel)
- How many other kernels can we port to the FPGAs?
- Can we link kernels to avoid data transfer?
- When do we need to reconfigure? At what cost?
- Future hardware: now ZU9, VU9 (early 2019) and HBM

(Xilinx white paper)

Many thanks
Please connect at
@euroexa or euroexa.eu

Mike Ashworth, Graham Riley, Andrew Attwood and John Mawer
Advanced Processor Technologies Group
School of Computer Science,
University of Manchester, United Kingdom
mike.ashworth.compsci@manchester.ac.uk

Project ID: 754337