Why does scientific High Performance Computing (HPC) need FPGAs?

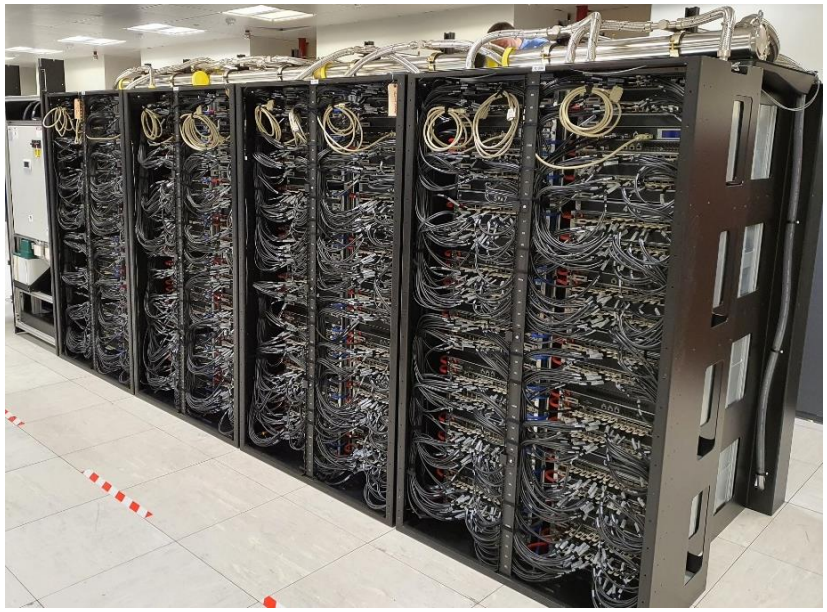
Dr Nick Brown, EPCC

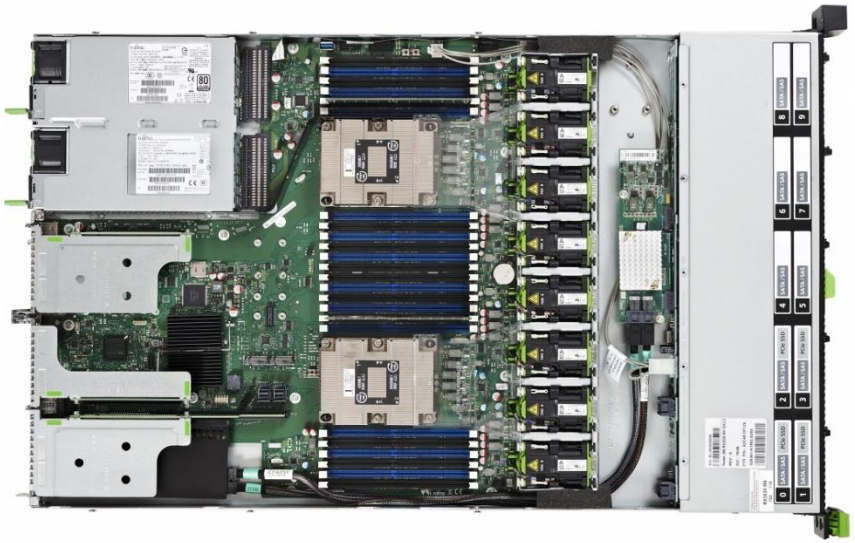
n.brown@epcc.ed.ac.uk

epcc

THE UNIVERSITY OF EDINBURGH

An embarrassment of hardware riches for computation





And an FPGA testbed….

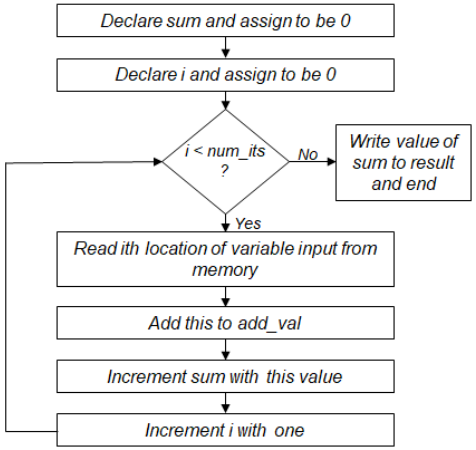
- Over 10 years ago had FPGA cluster in EPCC
  - But immaturity of the hardware (struggling to match CPU performance) and software ecosystem (difficult to program and lack of tooling) ultimately meant that this was not adopted

- But a decade is a long time, and things change!
  - Much more capable hardware
  - Significantly enhanced software ecosystem
  - Testbed funded by ExCALIBUR, details at fpga.epcc.ed.ac.uk

Moving from Von Neumann to dataflow

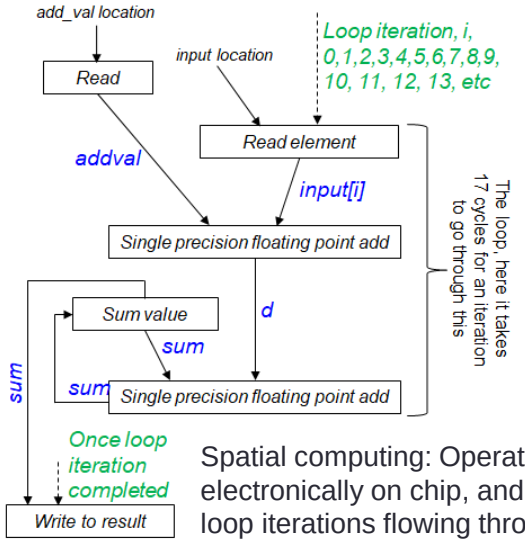
```
float sum=0;
for (unsigned int i=0;i<num_its;i++) {
  float d=input[i] + add_val;
  sum+=d;
}
*result=sum;
```

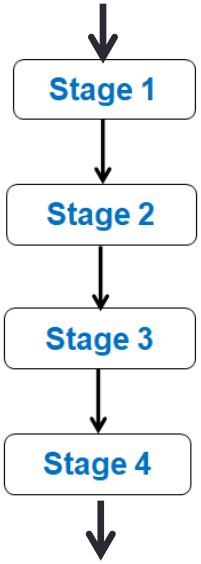
Temporal computing (CPU or GPU)



Temporal computing: Can be thought of like a flowchart, with the PE (e.g. CPU or GPU) executing one stage after another

Reconfigurable architecture (dataflow)





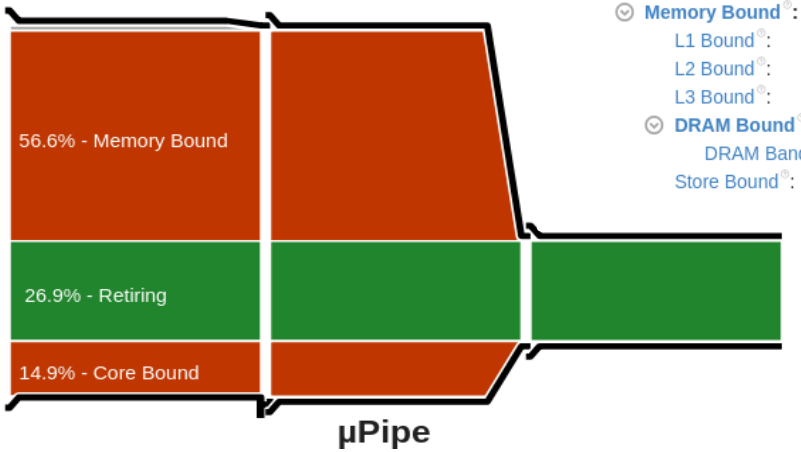
Spatial computing: Operations implemented electronically on chip, and acts as a pipeline, loop iterations flowing through

epcc

THE UNIVERSITY OF EDINBURGH

So where can FPGAs be helpful?

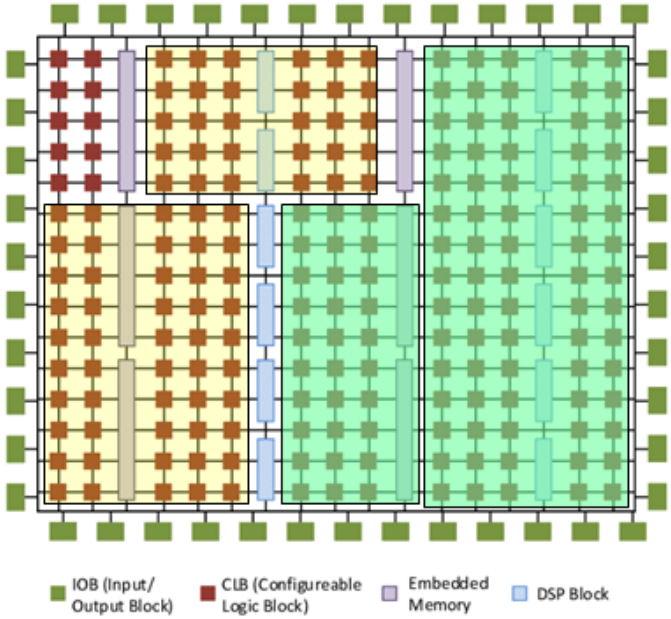
1. When the code is not compute bound



Memory Bound:	57.4%	of Pipeline Slots
L1 Bound:	3.6%	of Clockticks
L2 Bound:	13.3%	of Clockticks
L3 Bound:	1.6%	of Clockticks
DRAM Bound:	34.3%	of Clockticks
DRAM Bandwidth Bound:	58.2%	of Elapsed Time
Store Bound:	0.2%	of Clockticks

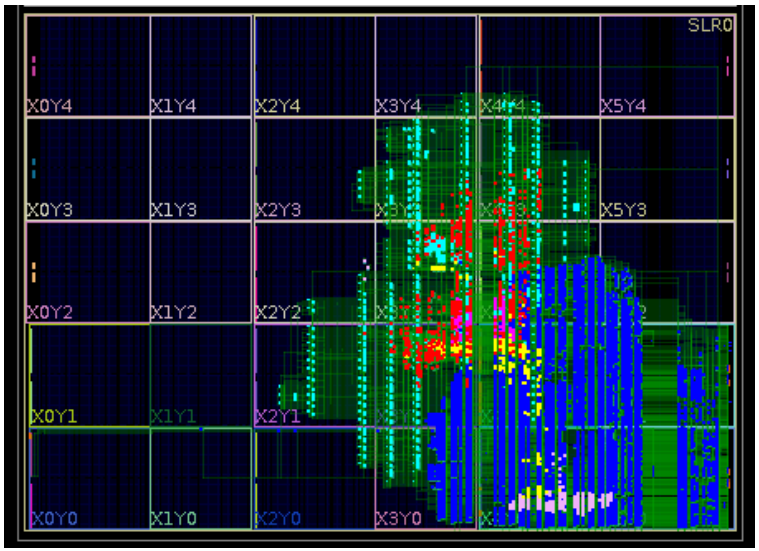
2. When we wish to exploit the high bandwidth connections of the FPGA
   - Between FPGA and outside (e.g. HBM2 or potentially QSFP28 networking ports)

3. For performance predictability

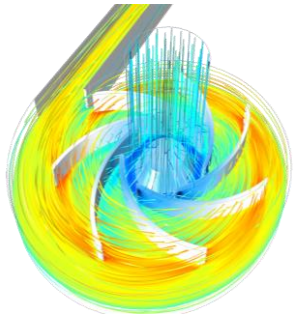


But the devil is in the detail….

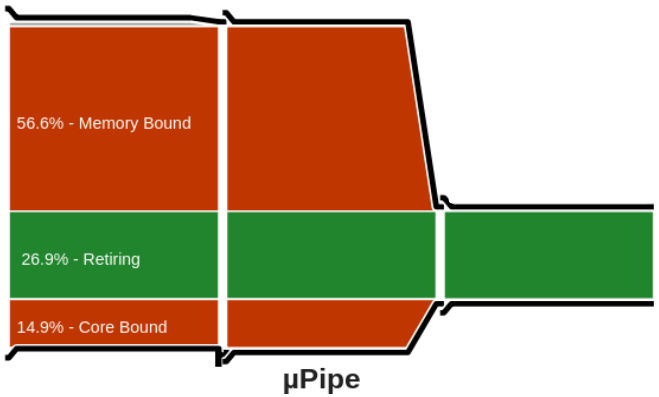
```
subroutine ax(n, nelt, w, u, g, dxm1, dxtm1)
  integer, intent(in) :: n, nelt
  real(n,n,n,nelt), intent(in) :: u, g, dxm1, dxtm1
  real(n,n,n,nelt), intent(out) :: w

  do e=1, nelt
    ax_e(n, nelt, w(:,:,:,e), u(:,:,:,e), ...)
  enddo
end subroutine ax

subroutine ax_e(n, w, u, g, dxm1, dxtm1)
  integer, intent(in) :: n
  real(n,n,n), intent(in) :: u, g, dxm1, dxtm1
  real(n,n,n), intent(out) :: w

  real(n*n*n) :: ur, us, ut
  real :: wr, ws, wt

  call local_grad3(ur, us, ut, u, n, dxm1, dxtm1)

  do i=1,n*n*n
    wr = g(1,i)*ur(i) + g(2,i)*us(i) + g(3,i)*ut(i)
    ws = g(2,i)*ur(i) + g(4,i)*us(i) + g(5,i)*ut(i)
    wt = g(3,i)*ur(i) + g(5,i)*us(i) + g(6,i)*ut(i)
    ur(i) = wr
    us(i) = ws
    ut(i) = wt
  enddo

  call local_grad3_t(w, ur, us, ut, n, dxm1, dxtm1)
end subroutine ax e
```



Example: AX kernel of Nekbone



- Nek5000 is used for high fidelity simulation of rotating components, such as turbines
  - Nekbone is a proxy-app capturing basic structure of Nek5000



On 24 CPU cores: 65.74 GFLOPs

- ➢ Only 11.7 times faster than one CPU core

On a Xeon Platinum Cascade Lake with N=16, 800 elements

```
subroutine ax(n, nelt, w, u, g, dxm1, dxtm1)
  integer, intent(in) :: n, nelt
  real(n,n,n,nelt), intent(in) :: u, g, dxm1, dxtm1
  real(n,n,n,nelt), intent(out) :: w

  do e=1, nelt
    ax_e(n, nelt, w(:,:,:,e), u(:,:,:,e), ...)
  enddo
end subroutine ax

subroutine ax_e(n, w, u, g, dxm1, dxtm1)
  integer, intent(in) :: n
  real(n,n,n), intent(in) :: u, g, dxm1, dxtm1
  real(n,n,n), intent(out) :: w

  real(n*n*n) :: ur, us, ut
  real :: wr, ws, wt

  call local_grad3(ur, us, ut, u, n, dxm1, dxtm1)

  do i=1,n*n*n
    wr = g(1,i)*ur(i) + g(2,i)*us(i) + g(3,i)*ut(i)
    ws = g(2,i)*ur(i) + g(4,i)*us(i) + g(5,i)*ut(i)
    wt = g(3,i)*ur(i) + g(5,i)*us(i) + g(6,i)*ut(i)
    ur(i) = wr
    us(i) = ws
    ut(i) = wt
  enddo

  call local_grad3_t(w, ur, us, ut, n, dxm1, dxtm1)
end subroutine ax_e
```

Iterate over element

```
subroutine local_grad3(ur, us, ut, u, n, dxm1, dxm2)
  integer, intent(in) :: n
  real(n,n,n), intent(in) :: u, dxm1, dxm2
  real(n,n,n), intent(out) ::ur, us, ut

  call mxm(dxm1, n, u, n, ur, n*n)
  do k=0,n
    call mxm(u(:,:,k), n, dxtm1, n, us(:,:,k), n)
  enddo
  call mxm(u, n*n, dxtm1, n, ut, n)
end subroutine local_grad3
```

Matrix multiplications

Multiply and add values calculated in local_grad3

- All double precision floating point
- AX kernel applies the Poisson operator of the CG solver accounts for approx. 75% of overall runtime of Nekbone 800 elements, and a size of N, with the number of grid points equal to N^3
  - For instance with N=16 then there are 831488 double precision floating point operations per element

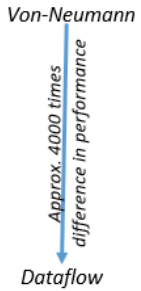
Use FPGA to ameliorate overhead of memory access and keep compute fed with data

epcc | THE UNIVERSITY OF EDINBURGH

Overview of single kernel performance

```
void ax_kernel(double * w, double * u, double * gxyz, double * dxm1, double * dxtm1, double * ur, double * us,
    double * ut, double * wk, int nx1, int ny1, int nz1, int nelt, int ldim) {
  #pragma HLS INTERFACE m_axi port=w offset=slave
  #pragma HLS INTERFACE m_axi port=u offset=slave
  #pragma HLS INTERFACE m_axi port=gxyz offset=slave
  #pragma HLS INTERFACE m_axi port=dxm1 offset=slave
  #pragma HLS INTERFACE m_axi port=dxtm1 offset=slave
  #pragma HLS INTERFACE m_axi port=ur offset=slave
  #pragma HLS INTERFACE m_axi port=us offset=slave
  #pragma HLS INTERFACE m_axi port=ut offset=slave
  #pragma HLS INTERFACE m_axi port=wk offset=slave
  #pragma HLS INTERFACE s_axilite port=nx1 bundle=control
  #pragma HLS INTERFACE s_axilite port=ny1 bundle=control
  #pragma HLS INTERFACE s_axilite port=nz1 bundle=control
  #pragma HLS INTERFACE s_axilite port=nelt bundle=control
  #pragma HLS INTERFACE s_axilite port=ldim bundle=control
  #pragma HLS INTERFACE s_axilite port=return bundle=control

  for (int e=0;e<nelt;e++) {
    ax_e(&w[nx1*ny1*nz1*e], &u[nx1*ny1*nz1*e], &gxyz[nx1*ny1*nz1*2*ldim*e], dxm1, dxtm1, ur, us, ut, wk, nx1, ny1, nz1);
  }
}
```

Description	Performance GFLOPs	% CPU performance
24 cores of Xeon (Cascade Lake) CPU	65.74	-
Initial FPGA port	0.020	0.03%
Optimised top down for dataflow	0.28	0.43%
Optimise bottom up	27.78	42.26%
Ping-pong buffering	59.14	89.96%
Increase clock frequency to 400 Mhz	77.73	118%



For N=16, Runs performed on a Xilinx Alveo U280

	Negative Slack	BRAM	DSP	FF	LUT
ax_kernel	0.39	4	182	65210	42524
ax_e	0.39	0	173	62095	39963
local_grad3_t	0.39	0	78	28760	18725
mxm16_1	0.39	0	24	9080	5964
mxm16_4	0.39	0	24	9046	5194
mxm16_3	0.39	0	21	8645	5406
add2	0.01	0	3	1282	1069
local_grad3	0.39	0	72	28311	18239
mxm16	0.39	0	24	9495	5723
mxm16_2	0.39	0	21	9250	5657
mxm16_1	0.39	0	24	9080	5964

	Pipelined	Latency	Iteration Latency	Initiation Interval	Trip
mxm16_1	-	-	-	-	-
Loop 1	yes	-	108	102	-

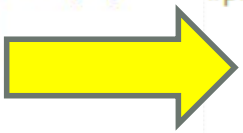
Detailed information available at https://arxiv.org/pdf/2011.04981.pdf

epcc

THE UNIVERSITY OF EDINBURGH

Bottom up optimisations

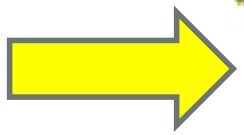
```
for (int i=0; i<N; i++) {
    double d=x*y;
    double j=d*z;
    double p=d*j
    result=p;
}
```



```
for (int i=0; i<N; i++) {
#pragma HLS pipeline II=1
    double d=x*y;
    double j=d*z;
    double p=d*j
    result=p;
}
```

Loop pipelining

```
for (int i=0; i<N; i++) {
#pragma HLS pipeline II=1
    double d=x*y;
    double j=d*z;
    double p=d*j
    result=p;
}
```



```
for (int i=0; i<N; i++) {
#pragma HLS pipeline II=1
#pragma UNROLL FACTOR=4
    double d=x*y;
    double j=d*z;
    double p=d*j
    result=p;
}
```

Loop unrolling

```
double val=0;
for (int i=0; i<N; i++) {
#pragma HLS pipeline II=1
    val=val+external[i];
}
```

Spatial dependency

```
for (int i=0; i<N; i++) {
#pragma HLS pipeline II=1
    double d=external_data[i];
    double j=external_data[i+1];
    ...
}
```

Conflict on external port to HBM2/DDR memory

```
double local_data[M];
for (int i=0; i<N; i++) {
#pragma HLS pipeline II=1
    local_data[i-2]=a;
    local_data[i-1]=b;
    double v=local_data[i];
}
```

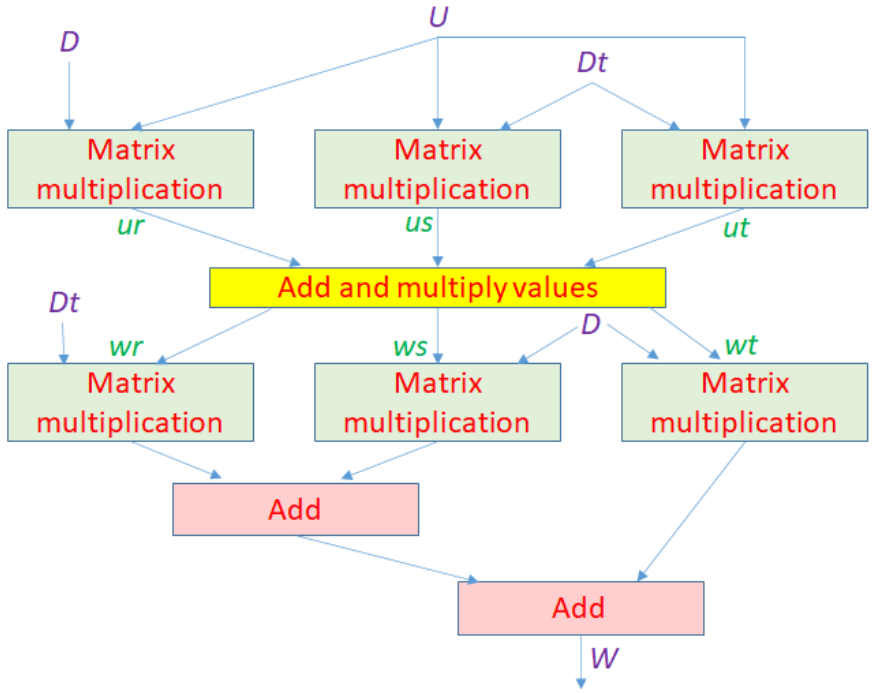
Conflict on (dual-ported) on-chip BRAM memory

epcc

THE UNIVERSITY OF EDINBURGH

Working top down: Adopting a dataflow design

- Each stage is an independent function running concurrently and connected via streams
  - Idea is to keep each part continually fed with data and processing
- Golden rule: Keep the data flowing, each cycle generating a result



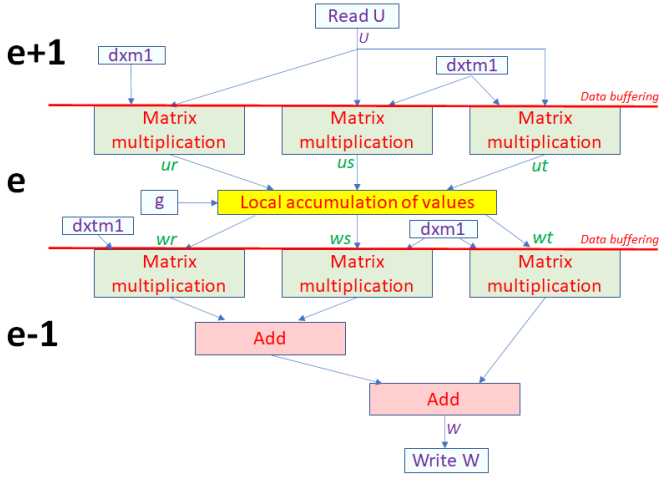
Description	Performance GFLOPs	% CPU performance
24 cores of Xeon (Cascade Lake) CPU	65.74	-
Initial FPGA port	0.020	0.03%
Optimised top down for dataflow	0.28	0.43%
Optimise bottom up	27.78	42.26%
Ping-pong buffering	59.14	89.96%
Increase clock frequency to 400 Mhz	77.73	118%

epcc

THE UNIVERSITY OF EDINBURGH

Buffering data between stages

- Adopt ping-pong (double) buffering
  - Works in three phases, loading data for the next element, processing the first three MM for the current element, and the last three MM for the previous element
  - Keeps all parts running concurrently



- Is also possible to accurately predict the realistic theoretical best performance our algorithm can deliver
  - Each MM is 31 FLOP/cycle and accumulation is 17/cycle, equals 203 FLOP/cycle. Multiply this by clock frequency for theoretical FLOPS
  - If not achieving this then not keeping data flowing!

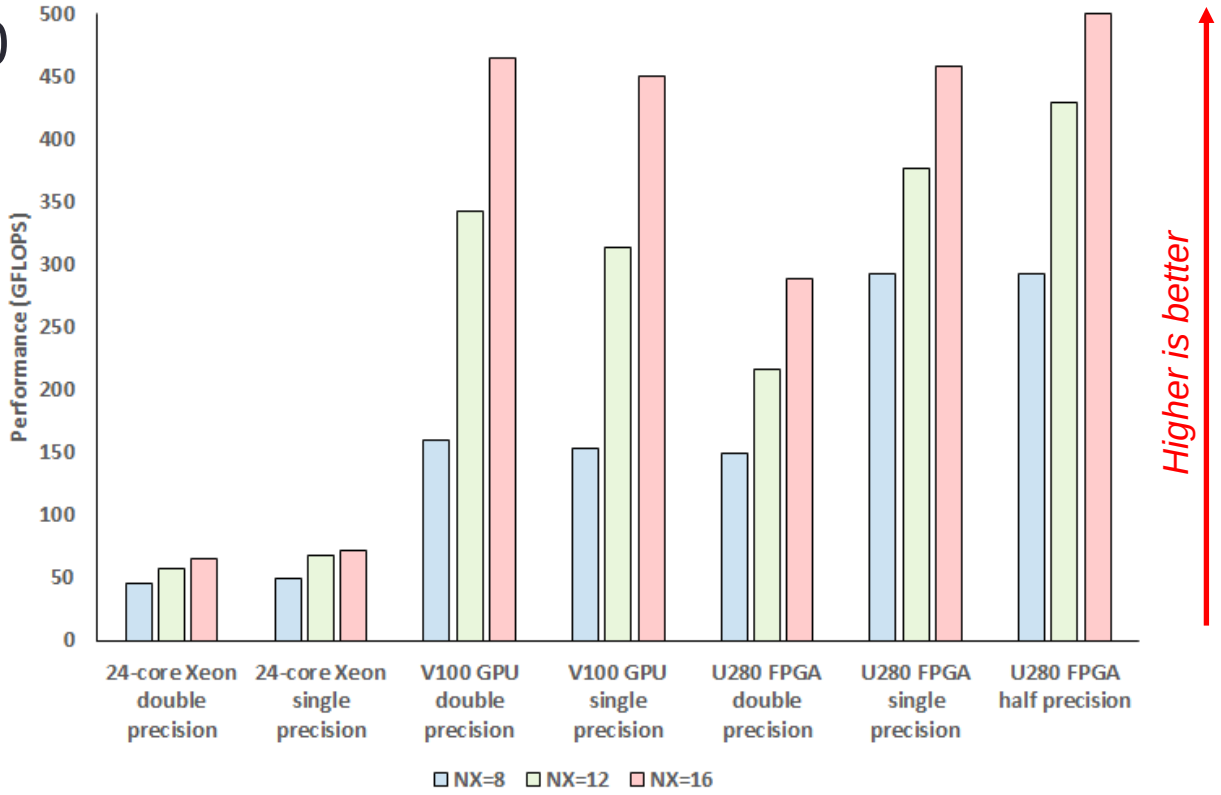
Description	Performance GFLOPs	% CPU performance	Theoretical performance	% Theoretical performance
24 cores of Xeon (Cascade Lake) CPU	65.74	-	-	-
Initial FPGA port	0.020	0.03%	6.9 GFLOPs	0.29%
Optimised top down for dataflow	0.28	0.43%	6.9 GFLOPs	4.06%
Optimise bottom up	27.78	42.26%	61 GFLOPs	45.54%
Ping-pong buffering	59.14	89.96%	61 GFLOPs	96.95%
Increase clock frequency to 400 Mhz	77.73	118%	81.2 GFLOPs	95.73%

Detailed information available at https://arxiv.org/pdf/2011.04981.pdf

epcc | THE UNIVERSITY OF EDINBURGH

Performance against CPU and GPU

- FPGA an Alveo U280
  - 4 FPGA kernels for double precision
  - 7 FPGA kernels for single & half precision
- CPU a 24-core Cascade Lake Xeon Platinum (8260M)
- GPU a NVIDIA V100

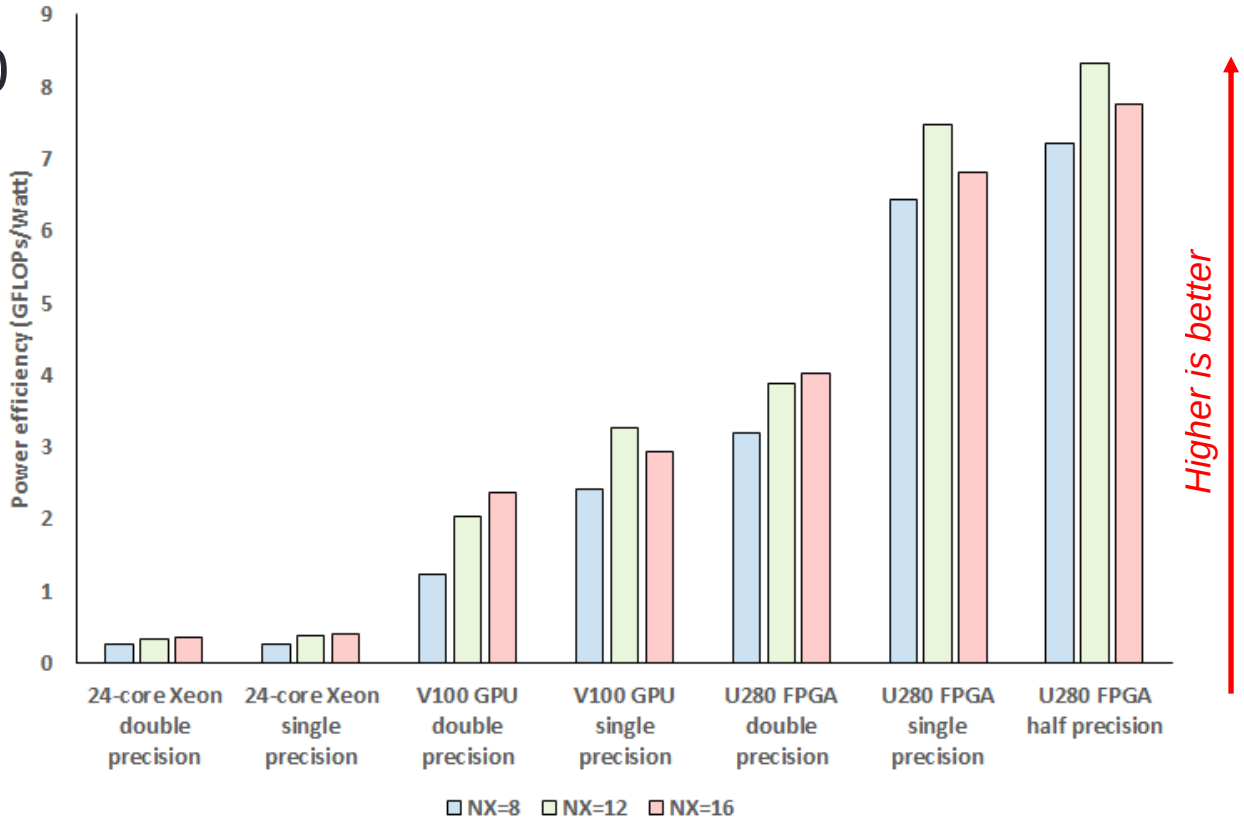


epcc

THE UNIVERSITY OF EDINBURGH

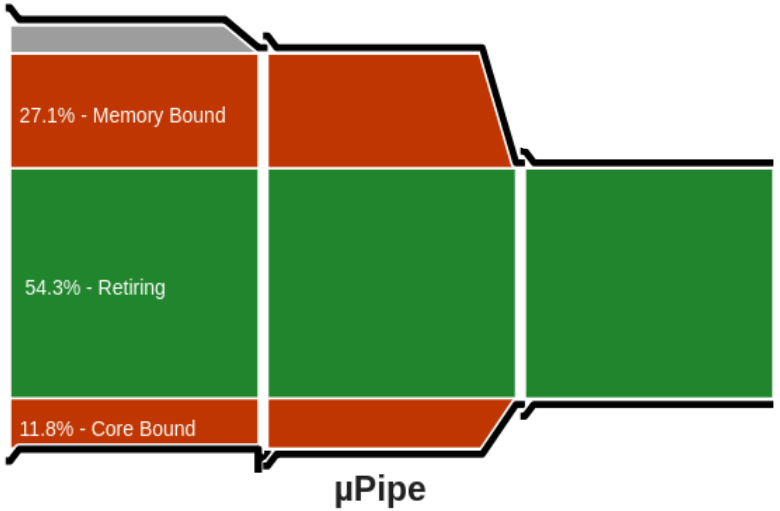
Power efficiency against CPU and GPU

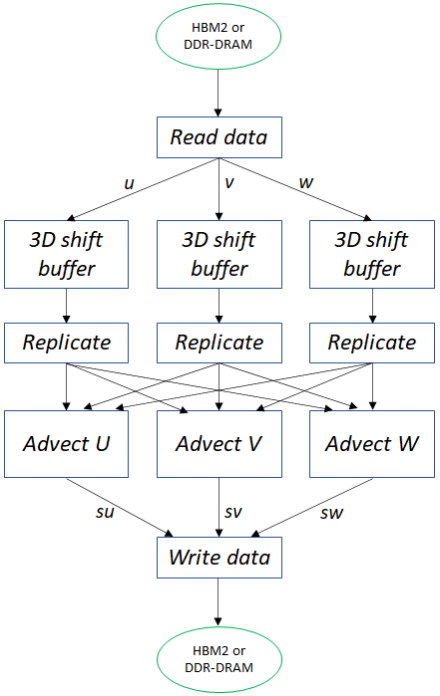
- FPGA an Alveo U280
  - 4 FPGA kernels for double precision
  - 7 FPGA kernels for single & half precision
- CPU a 24-core Cascade Lake Xeon Platinum (8260M)
- GPU a NVIDIA V100



Another example: Atmospheric advection

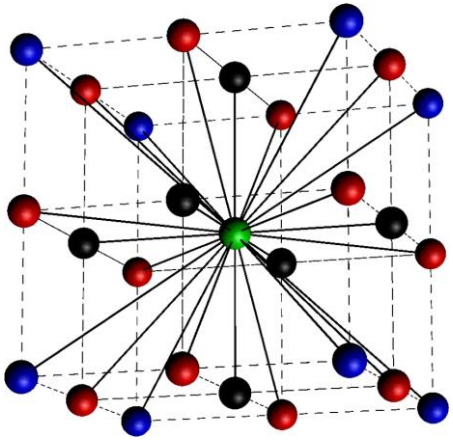
- Part of Met Office NERC Cloud (MONC) model
  - Accounts for around 40% of the model runtime
  - Stencil code working on three fields (U, V, W which is wind in x, y, z dimensions)

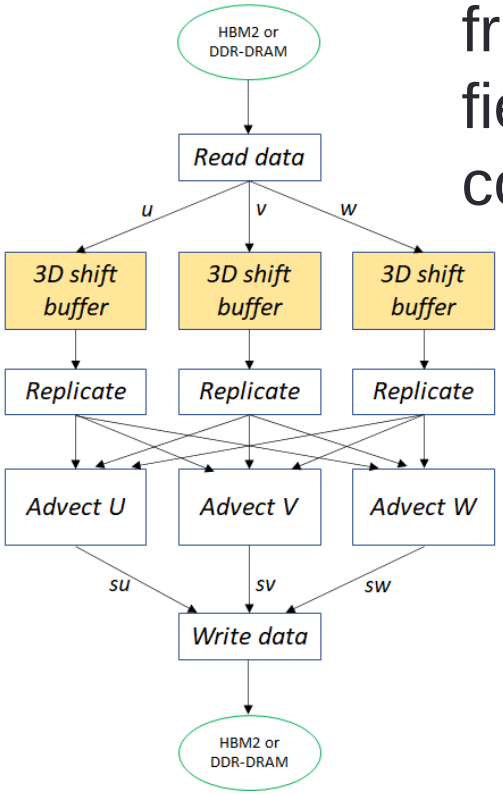




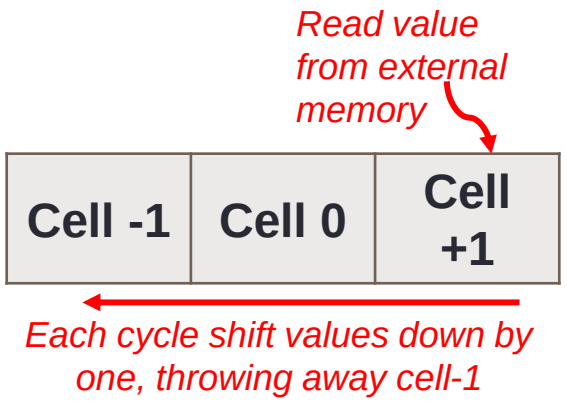
Tailoring caching of data via shift buffer

- 3D domain, where stencil computations require up to 27-points to calculate value for each grid cell
  - Want to read only one new value from external memory for each field per cycle as otherwise get conflicts on the memory port





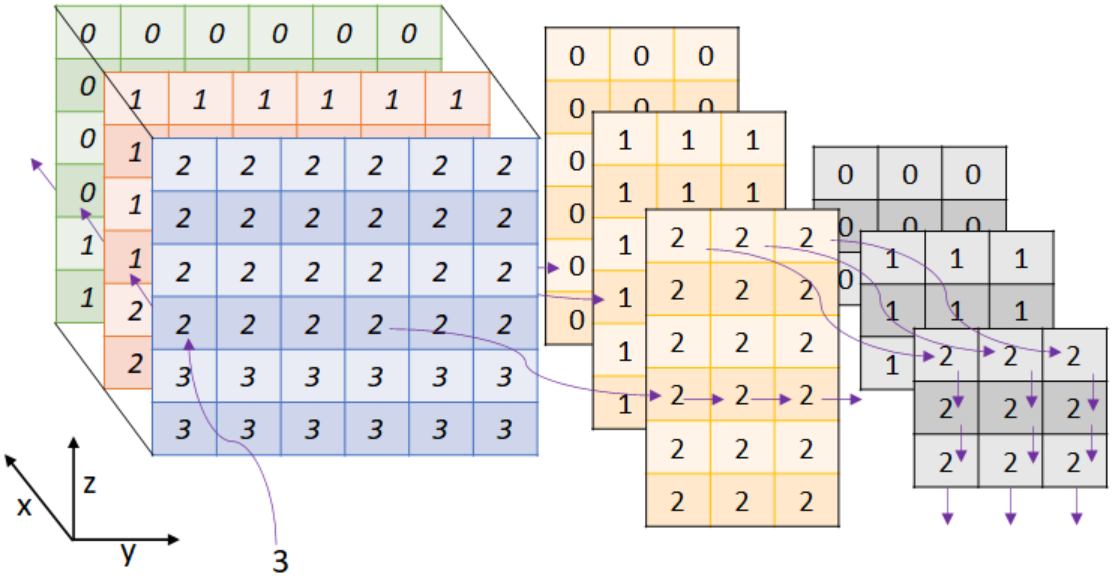
- But need to provide 27 values per cycle to the advect routine in order to achieve a result for each field for each clock cycle
- Use a shift buffer - 1D example



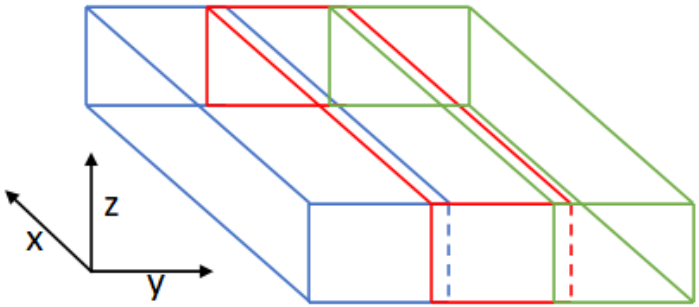
epcc

THE UNIVERSITY OF EDINBURGH

Tailoring caching of data via shift buffer



- Have these windows running across the 3D domain
- Generates 27-point stencil each cycle
- Memory on FPGA limits size in Y dimension so work in chunks



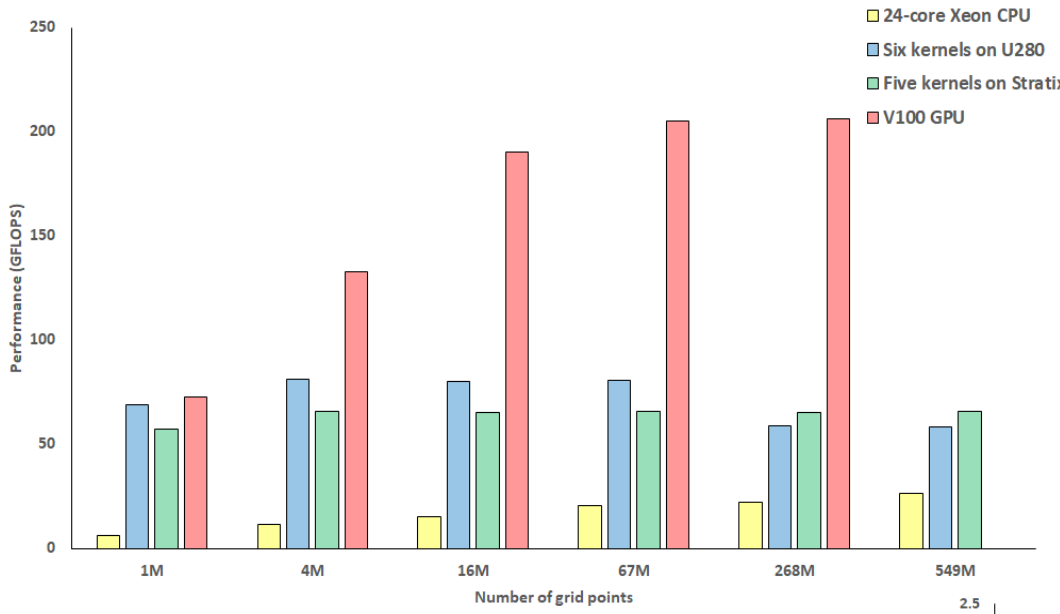
- The golden rule of keeping the data flowing and generating a result per cycle necessitated this shift buffer, which then impacted how the code is running and having to chunk in Y

Detailed information available at https://arxiv.org/pdf/2107.13500.pdf

epcc

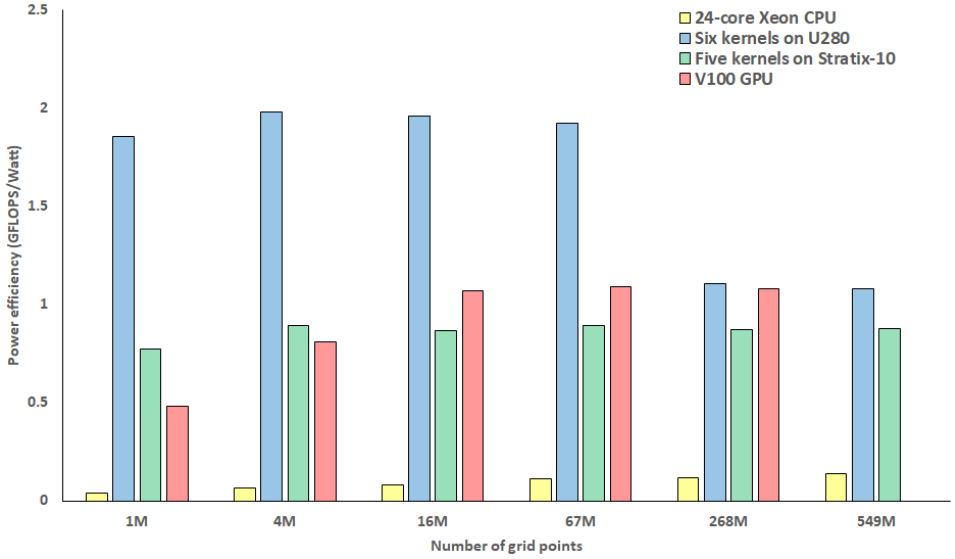
THE UNIVERSITY OF EDINBURGH

Performance comparison



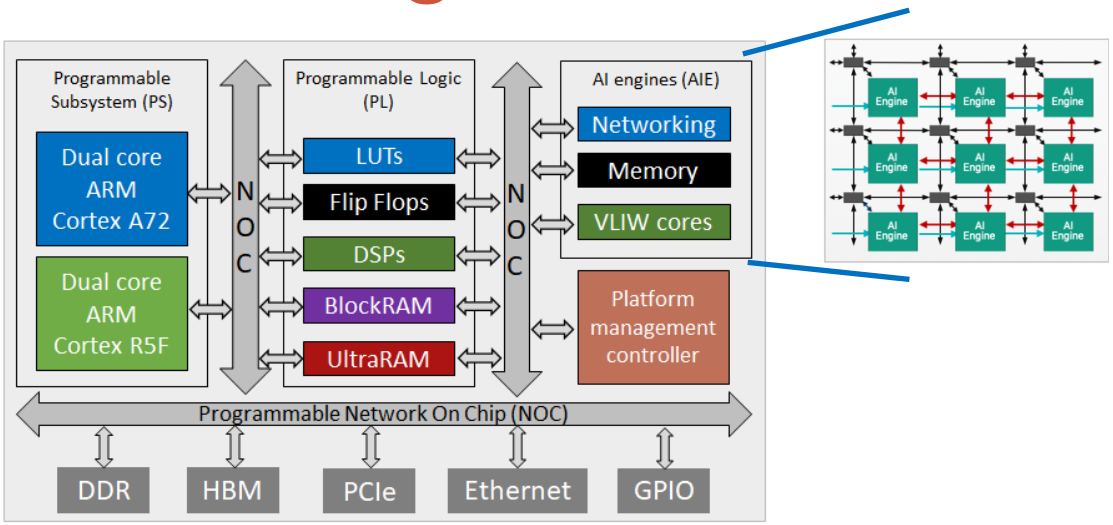
- FPGAs outperform the CPU by a long way, but GPU is a tough test!
- The Xilinx Alveo U280 tends to outperform the Intel Stratix 10
- At largest problem sizes Alveo must use DDR-DRAM rather than 8GB HBM2 resulting in performance decrease

- Alveo U280 has excellent power efficiency until switch from HBM2 to DDR-DRAM
- Higher power draw of Stratix 10 means it is worse, but still competitive against the GPU, especially for smaller problem sizes



Leveraging Versal AI engines

- 400 AIEs which run at 1.2GHz and each can handle 8 (single precision) FP arithmetic operations per cycle.



- Therefore can we use the PL for data loading/reordering and the AI engines for compute?

Description	Performance (GFLOPS)
VCK5000 AIEs *(4 CUs)*	68.73
VCK5000 PL-only *(8 CUs)*	101.78
Alveo U280 *(6 CUs)*	72.32
24-core Xeon Platinum CPU	23.52
V100 GPU	227.89

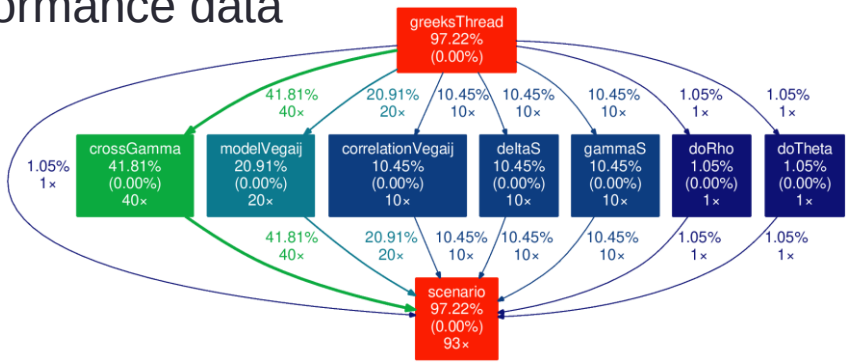
- Shows promise, but currently limited by number of connections between PL and AIEs

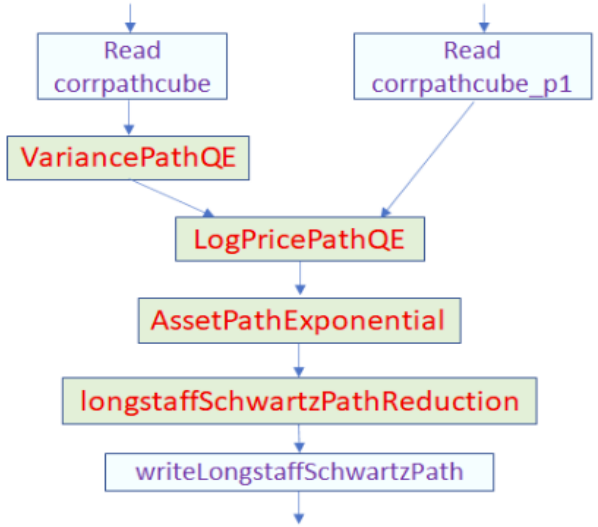
epcc

THE UNIVERSITY OF EDINBURGH

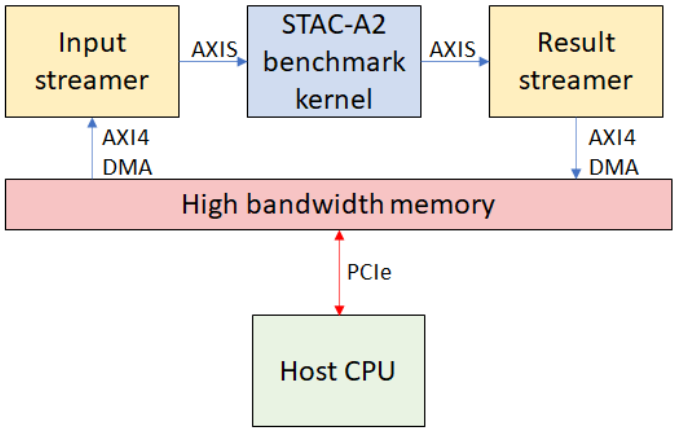
STAC-A2 financial benchmark

- STAC research are the industry standard body for financial benchmarks
  - Many companies such as technology vendors and fintech are members and use the benchmarks as a basis for performance data
- STAC-A2 focuses on the computation of market risk sensitivities
  - This describes price movements in a market and their impact on the value of an investor's position in holding a financial instrument.
  - Involves simulation using the Heston stochastic volatility and the Longstaff and Schwartz models.
- In the reference code around 50% of cycles are stalled due to memory access or other issues.





STAC-A2 financial benchmark



- Whilst Alveo shells are DMA only, we can provide a streaming-like approach by decomposing the data into batches
  - Keeps the different parts running concurrently
  - Especially useful here where we need to undertake data reordering of input data and results

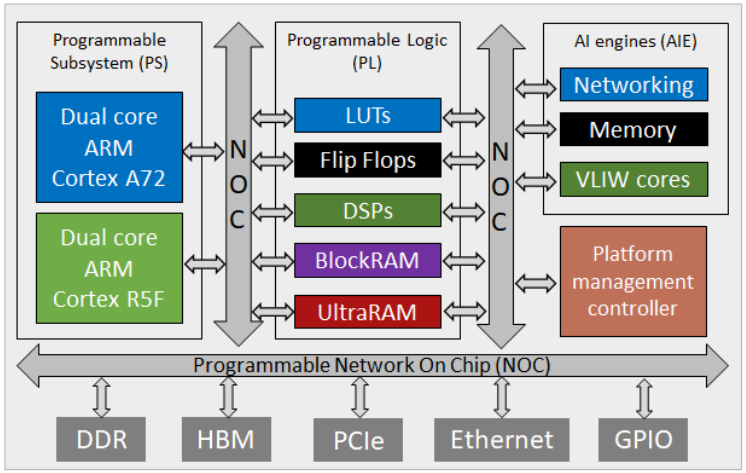
Problem size	Data type	Runtime (ms) – Lower is better				Total Energy Usage (J) – Lower is better			
		CPU	GPU	Alveo U280	Stratix-10	CPU	GPU	Alveo U280	Stratix-10
Tiny	float	372.15	1119.88	77.81	93.65	92.72	62.43	2.55	3.71
	double	369.07	1148.14	67.33	79.13	94.35	64.12	2.51	4.25
Small	float	629.18	1195.73	98.46	102.99	151.50	67.12	3.20	5.21
	double	638.67	1258.58	97.64	110.55	153.85	70.57	3.76	6.35
Medium	float	1545.94	1656.99	221.71	254.30	397.48	100.71	7.40	14.56
	double	1551.90	1932.30	286.90	301.52	393.34	123.88	10.92	18.73
Large	float	4574.41	2850.95	544.48	618.79	1277.14	187.99	18.46	42.58
	double	4558.11	3718.64	714.96	772.63	1266.59	246.14	27.15	61.03
Huge	float	15825.42	-	1928.11	-	5011.16	-	54.68	-
	double	15561.09	-	2301.29	-	4900.16	-	92.89	-

"CPU" is running on two, 24-core Cascade Lake Xeon Platinum CPUs, "GPU" is running on Nvidia V100

The experiments conducted have not been designed to comply with official STAC benchmarking rules and regulations. Therefore, the experimental results that we present are of a research nature and are not representative of official STAC audits.

Conclusions & keen to collaborate!

- Really exciting time for FPGAs based on the advances being made by vendors around hardware and software ecosystems
  - Many of our HPC codes are not compute bound, so worth exploring whether the flexibility offered by FPGAs can be beneficial
- FPGA tooling is mature enough to consider these from a dataflow algorithmic perspective
  - Golden rule of keeping the data flowing!
  - Can get good performance on the FPGA, but expertise is still needed
    - Same as GPUs and CPUs, more challenges to overcome.



Keen to collaborate exploring computational workloads on FPGAs

epcc

THE UNIVERSITY OF EDINBURGH