Implementation of the Matrix Vector Multiplication on FPGA through QuickPlay HLS

C. Rouaud on behalf of P. Palazzari

Real-Time Control for Adaptive Optics (5th edition)
Introducing QuickPlay

Graphical/C/C++ modeling → FPGA board target → Deployed application

IP: HDL, C/C++, built-in, 3rd-party

FPGA Boards: Development Kits, COTS, Full Custom

Tools: drivers, compilers, HLS, debuggers, ...

Acceleize
QuickPlay Modeling

Streamed I/Os

Untyped Streaming Channels (AXI4-ST)

Access to memory storage (AXI4-MM)

C/C++, HDL or IP

Streaming I/Os

Untyped Streaming Channels (AXI4-ST)
Objectives

- Objectives: Perform Matrix Vector Multiplication using HLS while
  - Saturating memory BW
  - Keeping lowest FPGA footprint

- Hardware: Microgate μXComp board
  - Intel Arria10 GX-1150 FPGA (427K ALM + 54K M20K)
  - PCIe Gen3 x8, 40 Gb Ethernet QSFP
  - 2 GB HMC with 4 independent links @ 10 Gbps
  - Theoretical max BW per link ≈ 16 lanes x 10 Gbps x 90% ≈ 18 GB/s
Application limits

- Best computation/memory efficiency conditions are given by

\[ T_{\text{comp}} = T_{\text{mem}} \]

Which for a scalar product (n members) can be written:

\[ T_{\text{comp}} = 2 \times n \times T_{fp} \quad (n \text{ MULT } + n \text{ ADD}) \]

\[ T_{\text{mem}} = 4 \times \frac{n}{BW} \quad (1 \text{ FP32 } = 4 \text{ Bytes}) \]

also expressed as \( Sc = \frac{BW}{2} \quad (Sc = \frac{1}{T_{fp}} \text{ is « computational speed »}) \)

1 Gflop/s requires 2 GB/s of memory BW
Coarse Grain Parallelism

read(a, m)
memcpy(memport, b, m)
s=0;
for (i=0; i<m; i++)
    s += a[i]*b[i]
b.write(s, m)
Coarse Grain Parallelism

- Limitations:
  - Number of Scalar Product kernels limited to 4 (# of HMC links)
  - Low computation bandwidth due to recursive Scalar Product operation
  - Leads to poor & inefficient memory BW utilization

- Optimization:
  - Improve Scalar Product kernel efficiency
Scalar Product: basic pipeline

- Scalar Product:

\[ S_{i+1} = a_i \times b_i + S_i \]

Commutativity

\[ s = \sum_{i=0}^{m-1}(a_i \cdot b_i) \]

- Split operation into smaller sub-vectors:

\[
\begin{align*}
S_0 &= a_0 \cdot b_0 \\
S_1 &= a_1 \cdot b_1 \\
S_2 &= a_2 \cdot b_2 \\
S_3 &= a_3 \cdot b_3
\end{align*}
\]
Scalar Product: basic pipeline

Scalar Product: $s_i = a_i b_i$

1 box = 1 clock cycle

Computing now 16 SP in 16 clock cycles instead of 16*4 clock cycles

$SP = S0 + S1 + S2 + S3$
Scalar Product: basic pipeline

- Limitations:
  - Each MADD kernel has a low bandwidth usage (1 FP32 required at each clock cycle ie 600 MB @150 MHz),
  - Requires large number of physical ports accessing to same memory bank to saturate memory BW (around 30 in our case) ➔ not feasible

- Optimization:
  - Find a new kernel architecture that maximizes memory BW usage
Scalar Product : Spatial Parallelism

- Each sub-vector now performed in parallel
MVM : Architecture

- Matrix Vector Multiplication of $M_{[n,m]} \times a_{[m]} : n$ independent SP
- Combining SP pipelining (L) and parallelism (P) into a single kernel
MVM : Operations

Matrix size = 8K x 8K
Pipeline latency L = 8
Parallelization factor P = 32

HMC Link → Local Memory

MADD32 produces P*L = 256 partial scalar products

m / (P*L) = 32 iterations to compute a single row*vector computation
MVM : Coarse grained pipeline

- Can be decomposed in 3 steps:
  - Load row M[n] from memory
  - Compute Scalar Product partial results
  - Sum partial results

\[
\begin{align*}
\text{Load } m_i & \quad \text{compute } s_j \quad \text{compute } b_i = \text{sum}(s_j) \\
\begin{array}{c|c|c|c|c}
  m_1 & m_2 & m_3 & m_4 & m_n \\
  s_{ij}(m_1) & s_{ij}(m_2) & s_{ij}(m_3) & s_{ij}(m_4) & s_{ij}(m_n) \\
  b_1 = s_j(m_1) & b_2 = s_j(m_2) & b_3 = s_j(m_3) & b_4 = s_j(m_4) & b_n = s_j(m_n)
\end{array}
\end{align*}
\]
QuickPlay Design

- 4 independent Scalar Product (SP) kernels
- Matrix (pseudo-constant) loaded in HMC memory
- Vector provided and spread to each kernel via PCIe
- Each kernels reads n/4 lines of the matrix from its dedicated HMC link
QuickPlay HLS implementation

- SP split into L*P sub-scalar products

\[ s_{ij} = m_{l:i} \cdot a_{ij} = \sum_{k=0}^{m \frac{L}{P} - 1} (m_{l:iP+j+kL} \cdot a_{iP+j+kL}) \quad i = 0, 1, \ldots, L - 1 \]
\[ j = 0, 1, \ldots, P - 1; \]

- MADD\textsubscript{32} implementation:

```c
/*#qp pipeline */
void MADD(float a1, ..., float a32,
           float b1, ..., float b32,
           float &c1, ..., float &c32)
{
    c1 += a1 * b1;
    ...
    c32 += a32 * b32;
}
```

32 independent (parallel) MADD
Latency: 7 clock cycles
QuickPlay HLS implementation

- $S_{ij}$ partial sums

```c
/*#qp unroll 32*/
for (i=0; i<(NbElem)/ComputedValues; i++) /*#qp ALOE */
{
    a1...a32 <= new values from memory
    MADD(a1,...,a32,b1,...,b32,cr0_0,...,cr0_31);

    a1...a32 <= new values from memory
    MADD(a1,...,a32,b1,...,b32,cr1_0,...,cr1_31);

    ...

    a1...a32 <= new values from memory
    MADD(a1,...,a32,b1,...,b32,cr7_0,...,cr7_31);
}

Sequential execution, 8 x 32 partial results
```
QuickPlay HLS: performances

- MVM (8 K x 8K float32)

Linear scaling in computational performance
QuickPlay HLS : FPGA usage

ALM

M20K

1 Kernel

4 Kernels
QuickPlay HLS : limitations

- Reminder: Theoretical maximal performance for 1 kernel = HMC \( \text{BW/2} = 9 \text{ GFlop/s} \)

- Performance drop due to non-ideal memory usage, leading in excessive memory read time wrt processing time

\[
\begin{align*}
\text{Load } m_i & \quad \text{compute } s_j \\
\text{Load } m_i & \quad m_1 \quad m_2 \quad m_n \\
\text{compute } s_j & \quad s_i(m_1) \quad s_i(m_2) \quad s_i(m_n) \\
b_i = \text{sum}(s_j) & \quad b_1 = s_i(m_1) \quad b_2 = s_i(m_2) \quad b_3 = s_i(m_n) \\
\end{align*}
\]

- \( m_i \approx 450 \text{ cycles} / s_i \approx 250 \text{ cycles} / b_i \approx 250 \text{ cycles} \)
QuickPlay HLS : improvements

- Memory loading is high due to high HMC latency (∼1 µs)
- Requires HLS logic to fully support pipeline read from HMC (AXI4 RD outstanding request)

Expected BW = 5.3 * \(\frac{450}{250}\) ≈ 9.3 GFlop/s per kernel
Improvement Perspectives

- QuickPlay HLS level:
  - Improve HMC memory bandwidth usage
  - Lower precision input (FP16, INTx) support to allow larger matrix support

- Hardware and Architecture
  - Improve HMC transceivers link speed from 10 to 12.5 Gbps \( \Rightarrow +25\% \)
  - FPGA boards cluster (spatial parallelism)
  - Usage of other boards and FPGAs with higher memory bandwidth: HBM/HBM2
    - Virtex Ultra Scale +
    - Intel Stratix10
Conclusion

- QuickPlay allowed:
  - Taking advantage of FPGA real-time capabilities (low latency/jitter)
  - Zero hardware/FPGA knowledge to target computing application on the board
  - Focusing on application and avoid dealing with interfacing (PCIe, memories, …)
  - Easy interaction with targeted hardware through its SDK
  - Easy migration to any other HW platform supported by the tool (on premise Xilinx / Intel boards, cloud deployment)

- Architecture exploration allowed to build kernels through HLS using the whole memory BW (allowed by QuickPlay framework)