Scalable performance by scaling out FPGA systems via inexpensive interconnects and the open-source AXIOM software stack

Roberto Giorgi
University of Siena, Italy

http://www.dii.unisi.it/~giorgi
Outline

• The bricks: AXIOM-board and AXIOM-sw_stack
• Scaling out FPGAs: how about this?
• OmpSs @ FPGA
• The Runtime system
• Optimizing data exchange via DF-Threads
• Next: the GLUON-board!
• UNIQUE FEATURES
  ➢ SCALABILITY via USB-C cable: building clusters up to 255 boards
  ➢ 4 channels on USB-C cable @ 18Gbps (custom protocol)
  ➢ PROGRAMMABILITY via OpenMP: TRANSPARENT FPGA acceleration + CLUSTER distribution
  ➢ ARDUINO-UNO socket on board for easy interfacing with the physical world
  ➢ 250MHz Trace port – Lauterbach compatible
  ➢ Open-source software stack + BSP: https://git.axiom-project.eu/ (10^6+ Lines of Code!)

http://www.axiom-project.eu/AXIOM_BOARD_GUIDE.pdf

✔ DELIVERED March 2018
Scalable Embedded Systems – A new concept?

- 1 board → ~10s, 2 board (USB-C connected) → ~7s (no optimizations)
- SAME board type, SAME software, NO programming efforts, still EMBEDDED


Roberto Giorgi -- http://www.dii.unisi.it/~giorgi
Scalable Embedded Systems – A new concept?

- 1 board $\rightarrow$ ~10s, 2 board (USB-C connected) $\rightarrow$ ~7s (no optimizations)
- SAME board type, SAME software, NO programming efforts, still EMBEDDED
Benchmark

- **MatrixMultiply** with **OmpSS@cluster** over **AXIOM conduit**
  - [https://git.axiom-project.eu/axiom-evi/blob/master/tests/ompss/src/ompss_evimm.c](https://git.axiom-project.eu/axiom-evi/blob/master/tests/ompss/src/ompss_evimm.c)
  - Block Size = 4 - Matrix Size = 800 - Element size = 8 bytes
  - Standard OmpSs@cluster
    - 2 boards (3 working threads + 1 I/O thread per board)
      - Execution time: **6889 msec**
    - 1 board (4 working threads)
      - Execution time: **10245 msec**
  - SCHED DEADLINE patch
    - 2 boards (4 working threads + 1 I/O thread per board)
      - Execution time: **6308 msec**
FPGA+SMP load balancing

• Originally, helper threads were exclusively dedicated to manage with the FPGA
  – Loose of performance

• Hybrid workers
  – Manage the FPGA
  – Execute SMP tasks
OmpSs2 compilation infrastructure

• New Mercurium compiler autoVivado plug-in

OmpSs Application

Mercurium
OmpSs phase
Host code + Nanos calls

FPGA phase
Code generation

GCC
Nanos
FPGA xtaks
FPGA dmalib
Extrae OMPT

Petalinux
AXIOM BSP +
OS (Linux) +
Platform Device Tree +
FPGA DMA driver
(BOOT.bin, image.ub)

autoVivado Tool
Vivado HLS
Task Manager
Instrumentation
Interconnect

Hardware generation

Vivado
DMA engines

Netlist

SMP
FPGA
bitstream

AXIOM id. 645496
http://www.axiom-project.eu

Roberto Giorgi -- http://www.dii.unisi.it/~giorgi
OmpSs2 autoVivado

- Directives guide the compilation & bitstream generation

```c
#pragma omp target device(fpga) copy_deps num_instances(2)
#pragma omp task in(a,b) inout(c)
void matrix_multiply(float a[BS][BS], float b[BS][BS], float c[BS][BS]);
```

```c
for (i_b=0; i_b<NB_I; i_b++)
  for (j_b=0; j_b<NB_J; j_b++)
    for (k_b=0; k_b<NB_K; k_b++)
      matrix_multiply(AA[i_b][k_b], BB[k_b][j_b], CC[i_b][j_b]);
```

```c
#pragma omp taskwait
// Or
// other tasks depending on input output c of matrix multiply task
```

Roberto Giorgi -- http://www.dii.unisi.it/~giorgi
OmpSs2 autoVivado

• Coverage

```c
#define BS 128

void matrix_multiply(float a[BS][BS], float b[BS][BS], float c[BS][BS])
{
    #pragma HLS inline
    int const FACTOR = BS/2;
    #pragma HLS array_partition variable=a block factor=FACTOR dim=2
    #pragma HLS array_partition variable=b block factor=FACTOR dim=1

    // matrix multiplication of a A*B matrix
    for (int ia = 0; ia < BS; ++ia)
        for (int ib = 0; ib < BS; ++ib) {
            #pragma HLS PIPELINE II=1
            float sum = 0;
            for (int id = 0; id < BS; ++id)
                sum += a[ia][id] * b[id][ib];
            c[ia][ib] += sum;
        }
}
```

Towards Discrete FPGAs

Xilinx ZCU102 (work in progress)

Trenz Electronics Zynq U+
TE0808 XCZU9EG-ES1

2x Cortex-A9 cores + FPGA
(32-bit platforms)

SECO AXIOM Board

Zynq-7000 Family

Zynq U+ XCZU9EG-ES2

4x Cortex-A53 cores + FPGA (64-bit platforms)
OmpSs/OmpSs2 availability

• pm.bsc.es & public github

The OmpSs Programming Model

OmpSs is an effort to integrate features from the StarSs programming model developed at BSC into a single programming model. In particular, our objective is to extend OpenMP with new directives to support asynchronous parallelism and heterogeneity (devices like GPUs). However, it can also be understood as new directives extending other accelerator-based APIs like CUDA or OpenCL. Our OmpSs environment is built on top of our Mercury compiler and Nanos++ runtime system.

Asynchronous parallelism is enabled in OmpSs by the use of data-dependencies between different tasks of the program. To support heterogeneity a new construct is introduced: the target construct.

In OmpSs the task construct also allows the annotation of function declarations or definitions in addition to structured-blocks. When a function is annotated with the task construct each invocation of that function becomes a task creation point. Note that only the execution of the function itself is part of the task not the evaluation of the task arguments (which are not privatized). Another restriction is that the task is not allowed to have any return value, that is, the return must be void.

DATA DEPENDENCES

The task construct allows to express data-dependencies among tasks using the in, out and inout clauses (standing for input, output and input/output respectively). They allow to specify for each task in the program what data a task is waiting for and signaling is readiness. Note, that whether the task easily uses that data in the specified way (i.e., the programmer responsibility. Each time a new task is created its in and out dependencies are matched against those of existing tasks. If a dependency, either full, half or none, is found the task becomes a successor of the corresponding tasks. This process creates a task dependency graph at runtime. Tasks are scheduled for execution as soon as all their predecessors in the graph have finished (which does not mean they are executed immediately) or at.

ompss-ee

OmpSs Examples and Exercises

- C  5  Updated an hour ago

nanox

Nanos++ is a runtime designed to serve as runtime support in parallel environments. It is mainly used to support OmpSs, a extension to OpenMP developed at BSC.

- C++  23  Updated 20 hours ago

dlb

DLB (Dynamic Load Balancing) library is a tool, transparent to the user, that will dynamically react to the application imbalance modifying the number of resources at any given time.

- library  openmp  mpi  load-balancer  ompss

- C  4  Updated a day ago
GitHub OmpSs/OmpSs2

OmpSs GitHub: https://github.com/bsc-pm

BSC - Programming Models
Barcelona Supercomputing Center
https://pm.bsc.es/

Repositories

- mcxx
  Mercurium is a C/C++/Fortran source-to-source compilation infrastructure aimed at fast prototyping developed by the Programming Models group at the Barcelona Supercomputing Center.

- nanox
  Nanos++ is a runtime designed to serve as runtime support in parallel environments. It is mainly used to support OmpSs, a extension to OpenMP developed at BSC.

- dlh
  DLB (Dynamic Load Balancing) library is a tool, transparent to the user, that will dynamically react to the application imbalance modifying the number of resources at any given time.
Boot Image Generation: XGenImage

Tool to automate generating the boot image

**Workflow**

- XGENIMAGE
- User-defined tool's environment
- Configurable environment
- Create Image

**Folder tree structure**

- **Main script**
- **Config**
- **Image**
- **Logs**
- **Roofs**
- **Xconfig**
- **Xconfig BSP**
- **Xconfig Output**

**Main Tasks**

<table>
<thead>
<tr>
<th>Task</th>
<th>Function Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialization</td>
<td>(REQUIRED) Set the tool with the default configuration</td>
</tr>
<tr>
<td>Configuration</td>
<td>(OPTIONAL) User configuration manipulation</td>
</tr>
<tr>
<td>Image creation</td>
<td>(FINAL TASK) creation of bootable SD image</td>
</tr>
</tbody>
</table>
The AXIOM software stack

- Running on the AXIOM board
- AXIOM-Link software specs available
- Device drivers
- Memory allocator
- User space app management
- Utility apps
- Ethernet over AXIOM
- GASNet Spawner
- OmpSs@Cluster
Ethernet over the AXIOM Link

• **axiom-ethtap (user space application)**
  – creates a TAP interface (ax0)
    • TAP (namely network tap) simulates a link layer device and it operates with “layer 2 packets” like Ethernet frames
  – **Set last byte of MAC address with AXIOM NodeID**
  – Forwards ethernet frames through AXIOM-nic
    • Destination NodeID extracted from the Destination MAC address

• **init script**
  – Launched from axiom-init after the end of the discovery algorithm
  – Starts axiom-ethtap and set IP on ax0
    (192.168.17.X -> X = NodeID)

• **iperf throughput**
  – 1 thread - 1.9 Gbps / 6 threads - 2.9 Gbps
Lauterbach and AXIOM board

- Debug and Trace
  - U-boot
  - Linux kernel
  - Linux kernel modules
  - User-space app/libraries

- Energy profiling using the Analog Probe
  - AXIOM board modified with cables attached on shunt resistors
  - Possibility to correlate the source code with the energy consumption

- Power profiling of Vimar application (see D7.3)
Distribution across nodes/boards

Diagram showing the distribution of an application (APP) across different nodes/boards using Nanos++ and DF-Threads.
Fundamental approach: DATAFLOW

A Scheme of Computation in which an activity is initiated by presence of the data it needs to perform its function

(Jack Dennis)
What is a DF-Thread

Program:

DF-thread

DF-thread1

DF-thread2

DF-thread3

preload

execute

poststore
DF-Threads Memory Model

- DF-thread
  - A function that expects no parameters and returns no parameters.
    - The body of this function can refer to any memory location for which it has got the pointer through DFT function calls (e.g., xpreload, xpoststor, xsubscribe, ...). A DF-thread is identified by an object of type dftid_t (df-thread identifier). In other words:
    ```
    typedef void (*dft_t)(void)
    ```

- INPUT_FRAME, OUTPUT_FRAME
  - INPUT_FRAME: A buffer which is allocated in the local memory and contains the input values for the current DF-thread.
  - OUTPUT_FRAME: A buffer which is allocated in the local memory and contains values to be used by other DF-threads (consumer DF-threads)

- SYNCHRONIZATION_COUNT
  - A number which is initially set to the number of input values (or events) needed by an DF-thread. The SYNCHRONIZATION_COUNT has to be decremented each time the expected data is written in an OUTPUT_FRAME.
void fibo(void) { // DF-Thread Fibonacci
    uint64_t* myfp = (uint64_t*) df_ldframe();
    int n = myfp[1];
    if (n <= 1) {
        df_write(myfp[0], n);
    } else {
        uint64_t* tfib1 = df_schedule(&fibo, 2);
        uint64_t* tfib2 = df_schedule(&fibo, 2);
        df_write(&tfib1[1], n - 1);
        df_write(&tfib2[1], n - 2);
        uint64_t* tadd = df_schedule(&adder, 3);
        df_write(&tadd[0], myfp[0]);
        df_write(&tfib1[0], tadd+1);
        df_write(&tfib2[0], tadd+2);
    }
    df_destroy();
}

void adder(void) {
    uint64_t* myfp = (uint64_t*) df_ldframe();
    uint64_t f1 = myfp[1];
    uint64_t f2 = myfp[2];
    df_write(myfp[0], f1+f2);
    df_destroy();
}

int fibo (int n) {
    if (n <= 1) return n;
    return fibo(n-1)+fibo(n-2);
}
...in other words...

```c
void fibo(void) { // DF-Thread Fibonacci
    uint64_t* myfp = (uint64_t*) df_ldframe();
    int n = myfp[1];
    if (n <= 1) {
        df_write(myfp[0], n);
    } else {
        uint64_t* tfib1 = df_schedule(&fibo, 2);
        uint64_t* tfib2 = df_schedule(&fibo, 2);
        df_write(&tfib1[1], n-1);
        df_write(&tfib2[1], n-2);
        uint64_t* tadd = df_schedule(&adder, 3);
        df_write(&tadd[0], myfp[0]);
        df_write(&tfib1[0], tadd+1);
        df_write(&tfib2[0], tadd+2);
    }
    df_destroy();
}

void adder(void) {
    uint64_t* myfp = (uint64_t*) df_ldframe();
    uint64_t f1 = myfp[1];
    uint64_t f2 = myfp[2];
    df_write(myfp[0], f1+f2);
    df_destroy();
}
```

- A simple dataflow runtime to test DF-Threads program: [http://drt.sf.net](http://drt.sf.net)
Running fibonacci(35) (zoom)
Benchmark: MATRIX MULTIPLICATION (MATRIX_SIZE+BLOCK_SIZE)

Execution Time (s)
[Input Size 432+8]
Benchmark: MATRIX MULTIPLICATION (MATRIX_SIZE+BLOCK_SIZE)

Execution Time (s)
[Input Size 832+8]
Benchmark: MATRIX MULTIPLICATION (MATRIX_SIZE+BLOCK_SIZE)

SpeedUp vs. 1node/1core OpenMPI

[Input Size 832+8]

Nodes

1 2 4 8

DF-THREADS/1  DF-THREADS/4  CILK/1  CILK/4  OPENMPI/1  OPENMPI/4  JUMP/1  JUMP/4
Implementation on the AXIOM platform

H.264 Video Decoding → Gaussian Pyramid → LBP Cascade Classifier → CNN Inference → UI

[PS] ARM Cortex A53 CPUs
[PL] Reconfigurable Logic
[PS] ARM Mali 400 GPU

CONV + PReLU
Pool STD
Local Norm
CONV + PReLU
FC
MaxOut
FC
Softmax
LBP Kernel Offloading to the PL

- The lack of a cache memory hierarchy in the PL domain forced us to split the **image pyramid** into chunks, and later fit them in the on-die **BlockRAM** [4 MB].

![Diagram showing the offloading process from DDR4 RAM to BRAM and LBP Model]
A single LBP accelerator consumed 75% of the ZU9EG BlockRAM, and exploited only 8 FPGA ALUs.
- Therefore, it was not feasible to synthesize several accelerators in the PL.

The achieved clock frequency of the generated bitstream was **200 MHz**

Conclusions:
- The LBP kernel was challenging for the Vivado HLS compiler. The evaluation of the classifiers was input dependent → Very hard to optimize at compile time for HLS.
- Very low FPGA DSP/ALU utilization.
- Latency-oriented kernel constrained by the bandwidth of the memory hierarchy.

<table>
<thead>
<tr>
<th></th>
<th>BlockRAM</th>
<th>DSP48E</th>
<th>FF</th>
<th>LUT</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Used</strong></td>
<td>1380</td>
<td>8</td>
<td>11599</td>
<td>113376</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>1824</td>
<td>2520</td>
<td>548160</td>
<td>274080</td>
</tr>
<tr>
<td><strong>(%)</strong></td>
<td>75</td>
<td>0.3</td>
<td>2.1</td>
<td>41.3</td>
</tr>
</tbody>
</table>
Demographics CNN Inference

• Herta’s proprietary CNN inference engine ported to the AXIOM board architecture (originally implemented in CUDA).
  – SGEMM backend replaceable by BSC’s OmpSs@FPGA matrix multiply kernel.

• The CNN models developed during Year 2 were updated with latest ML advances (batch normalization layers).

Roberto Giorgi -- http://www.dii.unisi.it/~giorgi
AXIOM Face Dataset

• All face pictures collected by UNISI [14 GB] were released to the public, and are now stored and indexed at Zenodo for enriching existing deep learning facial datasets:

https://doi.org/10.5281/zenodo.1187127
System Integration

VIMAR-ELVOX Video Door Entry System

VIMAR User Interface

AXIOM Board

Customization of the video door entry system and the AXIOM_Bio_ID Application
- HW: Define the correct Lens
- HW: Illumination/LEDs
- SW: Input video resolution (720x578 px)
- SW: image rectification
- SW: Graphical User Interface

Edge Computing prototype for Smart Home solution

User Interface to test user interaction

Roberto Giorgi -- http://www.dii.unisi.it/~giorgi
# The GLUON-board (preview)

## Bottom view

## Top view

<table>
<thead>
<tr>
<th>Specifications</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL GT Transceivers</td>
<td>8 Differential Pair in total, 16.3 Gb/s</td>
</tr>
<tr>
<td>SD Card</td>
<td>Boot the Operating System</td>
</tr>
<tr>
<td>JTAG/UART header</td>
<td>To configure the Si5345 Clock Generator</td>
</tr>
<tr>
<td>10 Pin I2C header for “Silabs” Clock Builder Field Programmer</td>
<td></td>
</tr>
<tr>
<td>Done, Error/Status LEDs</td>
<td></td>
</tr>
<tr>
<td>PS I/O loopbacks</td>
<td></td>
</tr>
<tr>
<td>PL I/O loopbacks</td>
<td></td>
</tr>
</tbody>
</table>

Roberto Giorgi -- http://www.dii.unisi.it/~giorgi
The GLUON-board (preview)
GLUONs connect QUARKs
(i.e., TE0808 Trenz-modules based on Zynq Ultrascale+)

The module will go on top of the GLUON-board
CONCLUSIONS

• This presentation introduced the "AXIOM software stack", which provides the possibility to distribute computations across a cluster of FPGAs by using the OmpSs programming model.

• OmpSs allows the programmer to increase the performance by either parallelizing the application on cores and nodes/boards of the cluster or by offloading a computation on an FPGA accelerator.

• The AXIOM project explored both the options and the combination of them.

• This open-source software, along the full software stack based on a standard Linux/Ubuntu 16.04 LTS distribution has been demonstrated on the AXIOM-board, which is based on the Zynq ZU9EG Ultrascale+ Soc.

• To further simplify the deployment and provide an inexpensive means for connecting up to 255 boards, the AXIOM-board provides four USB-C connectors, each one capable of up to 18Gbps data transfer rate.

• This model permits to address the need of a scalable performance depending on the size of the problem, e.g., in the Smart Home in order to serve a building or an apartment, by using the same hardware and software platform, or in the Smart Videosurveillance.

• The new GLUON board will provide a more cost effective solution for implementing AXIOM project concepts.
Thank you.

Questions?

Facebook: https://www.facebook.com/theaxiomproject?ref=hl
Twitter: https://twitter.com/axiom_project @axiom_project
Google+: google.com/+Axiom-projectEu
LinkedIn: https://www.linkedin.com/grp/home?gid=8294592
Yotube Channel: https://www.youtube.com/channel/UCsQmVv98aPV0vzSjn7y1bMw