Transprecision Computing

Dionysios Diamantopoulos
IBM Research - Zurich
A look into the next 15 years

-8x

Source: The International Roadmap for Devices and Systems: 2017
Approximate computing so far:

- Targets only very specific problems / applications
- Approximation introduced only in selected routines (lack of automation)
- Typically leads to a quality loss in the final result
- Typically no error bound / error control

“applications don’t all require that computing be done at the same level of accuracy. For some, you could use lower-precision floating-point arithmetic instead of the commonly used IEEE 754 standard”
—David Patterson, 2018
From approximate computing to transprecision

Transprecision computing features:

- Use **adaptive precision** during computation (e.g., 8-12 bits exotic floating-point variables)
- Offload large part of computation to **low-power HW** units (e.g., Accelerators)
- Use **adaptive stopping criteria** (e.g., stops when needed precision is reached)
- **Recover accuracy** by end of calculation (up to required level)
OPRECOMP’s scope

Transprecision computing
- Flexible precision
- Adequate accuracy
- Algorithmic innovation: scalable, low-power, reliable, accurate

Realistic technology
- Silicon technology platform
- Non-CMOS enhancements explored

Scalable architecture demonstrated
- mW full demonstrator
- MW node demonstrator: sub1KW node

Open Innovation
- IBM for the MW space
- SME for the MW space
“OPRECOMP” aims to build an innovative and reliable foundation for computing.

This is accomplished by demolishing the ultra-conservative precise computing abstraction and replacing it with a more flexible and efficient one, namely “Transprecision Computing”.

“OPRECOMP” will demonstrate this idea in the domains of:
- Big Data Analytics,
- Deep Learning,
- High Performance Computing
and over two platforms:
- mW,
- kW
Disruptive Technology Modeling and Exploration

Memory contribute a major part to system power
Reduce refresh cycle to save power
Smart memory controllers, use optimal control to resolve bank/channel contention

Modeling & Exploration Tools
Heterogeneous 3D stacks
New technologies (e.g. ReRAM, NEMS devices)
Interfaces (wide I/O, serial, optical)
QoS-aware adaptive memory controllers

• Approximate storage concepts to increase energy efficiency.
• Models on different abstraction-levels will be developed to permit design exploration for transprecision memory architectures.
Architecture and Circuits Design: mW- up to kW-platforms

A hardware architecture that allows fine-grained control of both temporal and spatial precision, in order to achieve energy-precision tradeoffs over a wide range.

- Develop hardware architecture: Processing, memory, communication
- Operating over a wide range: From IoT (mW) to HPC (kW) applications
- Fine-grained control of precision: Add the capability to adjust precision
- Exploiting energy-precision trade-offs: Allow the hardware to be tuned over a wide range

kw Anchor: IBM POWER8/9 CAPI+FPGA/GPU
FPGA-based PULP platform emulator with transprecision support

mw Anchor: ASIC implementations of PULP with transprecision support
- Enable silicon measurements
- Derive model for large-scale implementation (kW anchor)
Software Environment and Tools

Software stack for developing, configuring and executing transprecise programs.

- Programming interface (API) based on transprecision algorithms
- Compiler support for automatic/semi-automatic static/dynamic precision tuning
- Input to hardware platforms in the mW and kW range

```
#pragma omp task transprecise(var1,prec1, cond1, var1, prec2, cond2, ....)
{ ... task body ... }
```
- Defining quality metrics for measuring the balance between quality loss against energy gains
- Identifying error-resilient kernels in terms of variables and operations
- Evaluating the effects of multiple approximate computing techniques
- Elaborating new algorithms and produce multiple approximation variants of existing SW
- **FloatX**: C++ template library for experimenting with flexible floating-point formats even if hardware is not yet available (BlasX, FloatXR under dev.)

Algorithms

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1. <code>float</code> <code>a = 1 / 3.0f;</code></td>
<td>1. <code>#include &lt;floatx.hpp&gt;</code></td>
<td></td>
</tr>
<tr>
<td>2. <code>float</code> <code>b = 45.3;</code></td>
<td>2. <code>flx::floatx&lt;6,8&gt; a = 1/ 3.0f;</code></td>
<td></td>
</tr>
<tr>
<td>3. <code>float</code> <code>c;</code></td>
<td>3. <code>flx::floatx&lt;6,8&gt; b = 45.3;</code></td>
<td></td>
</tr>
<tr>
<td>4. <code>[...]</code></td>
<td>4. <code>flx::floatx&lt;6,8&gt; c;</code></td>
<td></td>
</tr>
<tr>
<td>5. <code>c = a*b - 33;</code></td>
<td>5. <code>[...]</code></td>
<td></td>
</tr>
<tr>
<td>6. <code>[...]</code></td>
<td>6. <code>c = a*b - 33;</code></td>
<td></td>
</tr>
<tr>
<td>7. <code>[...]</code></td>
<td>7. <code>[...]</code></td>
<td></td>
</tr>
</tbody>
</table>

The diagram illustrates floating-point formats with different bit widths:
- IEEE 754 double (64bit)
- IEEE 754 float (32bit)
- FloatX<`w`, `t`> (1+`w`+`t` bits)
Goal: Develop a computation node of a HPC system based on a high-performance processing node (IBM POWER 8/9) and an array of interconnected controllable precision processors.

- CAPI connection between the Power8 platform and the PULP accelerator
- Support also for the Virtual platform

Extend POWER™ with transprecision co-processors
PULP team has designed over 20 PULP based ASICs already

www.pulp-platform.org

http://asic.ethz.ch
Transprecision options for PULP

- **Auxiliary Processing Units (APU)** extensions into the core and ISA

- **Shared Auxiliary Processing Units** with direct connection to the pipeline of processors in the cluster

- **Hardware accelerators** with direct access to shared memory

- **Accelerators over standard interfaces** (AXI/APB)
• Implemented a small float unit
  • Additional small float unit smaller than a typical RISC-V core
  • Able to run at 350MHz, in UMC65nm

• Unit Supports:
  • 2 alternative formats (8 and 16 bit)
  • Allows vector operations
    • 1x 32bit, 2x16 bit, 4x8 bit

- PULP system interconnect is AXI4-based
- **AXI-to-CAPI adapter** gives PULP access into POWER8 memory space
- Jobs from POWER8 are added to a **WED FIFO** where PULP can fetch them (woken up by interrupt)
• POWER8 uses only lower 48 bits of address; upper bits sign-extension
• PULP itself is 32 bit (processors, clusters, peripherals)
• Selectively extend system-level interconnects to 49 bit
• MSB decides whether to access POWER8 or PULP memory
• Gained space in PULP memory can be used for on-board DRAM/NVM
1. All cores start execution at the same internal address.
2. Cores cannot directly execute code from host memory; kernel needs to be in PULP memory.
3. Don’t want to embed kernels into FPGA bitstream; would need to regenerate bitstream for every kernel change.
4. Embed a bootloader program into a ROM in the bitstream.
5. Send the PULP binary to execute with every WED.
6. Bootloader copies binary from POWER8 memory into PULP memory.
We want to be able to load an ELF binary to PULP.

1. Load the binary into memory.
2. Parse the ELF header (ehdr) and program headers (phdr); these contain all sections that need to be loaded.
3. Copy section offsets and sizes into a *section table*, and create a new WED:

```c
struct wed {
    struct sec *sec_ptr;
    size_t sec_num;
    void *wed; // WED to be passed to loaded binary
};

struct sec {
    void *src; // host memory
    uint32_t dst; // PULP memory
    uint32_t src_sz;
    uint32_t dst_sz;
};
```

4. Send to PULP as WED; bootloader then copies sections.
We bundled the binary loading, parsing, and offloading code as a C library:

```c
// liboprecomp

/* Binary loading and parsing */
opc_kernel_new // create new kernel
opc_kernel_load_file // parse binary from file
opc_kernel_load_buffer // parse binary from memory
opc_kernel_free // destroy kernel

/* Offloading onto PULP */
opc_dev_new // create new device (= accelerator)
opc_dev_open_any // open any device on the system
opc_dev_open_path // open device by ‘/dev/cxl/...’ path
opc_dev_launch // offload kernel onto device
opc_dev_wait // wait for completion of one kernel
opc_dev_wait_all // wait for completion of all kernels
opc_dev_free // destroy device
```

Wraps around `libcxl`, so this should be the only thing you need to interface with PULP.
// Error handling omitted for brevity.
// (Shame on me!)
#include <liboprecomp.h>

// Load the kernel.
const char *elf_path = "hello_world";
opc_kernel_t knl = opc_kernel_new();
opc_kernel_load_file(knl, elf_path);

// Open any accelerator on the system.
opc_dev_t dev = opc_dev_new();
opc_dev_open_any(dev);

// Offload a job and wait for completion.
uint64_t wed = 0xdeadbeeffacefeed;
opc_dev_launch(dev, knl, &wed, NULL);
opc_dev_wait_all(dev);

// Clean up.
opc_dev_free(dev);
opc_kernel_free(knl);
Building *kwPilot*: leveraging TP for HPC through OpenPOWER ecosystem

Vibrant ecosystem through open collaborative & development

**kwCDK**: Cloud Development Kit
- Automated VM/Docker deployment on Supervessel

**kwSDK**: Software Development Kit
- POWER/PULP SW co-design + PULP VP

**kwHDK**: Hardware Development Kit
- PULP RTL + CAPI to FPGA bitstream

Energy-efficient computing through precision relaxation

Transprecision Computing Architecture
Functional Units, Processing Elements, Communication Infrastructure, Memory Hierarchies, Runtime Management, Programming Model

- Automated VM/Docker images generation with precompiled PRECOMP SW
- POWER8/9 worker node UBUNTU docker
- PULP VP

SuperVessel Shared Services

Standard Hardware

PULP RTL

kwHDK
PULP RTL + CAPI to FPGA bitstream
**kwPilot v1.0 setup**

**Accelerator Cards**

<table>
<thead>
<tr>
<th></th>
<th>8KU</th>
<th>KU3</th>
</tr>
</thead>
<tbody>
<tr>
<td>FPGA:</td>
<td>XCKU115-2-FLVA1517E</td>
<td>XCKU060-FFVA1156</td>
</tr>
<tr>
<td>CLBs</td>
<td>1451 k</td>
<td>726 k</td>
</tr>
<tr>
<td>DSP Slices</td>
<td>5520</td>
<td>2760</td>
</tr>
<tr>
<td>Block RAM</td>
<td>75.9 Mbit</td>
<td>38.0 Mbit</td>
</tr>
<tr>
<td>DRAM:</td>
<td>16 GiB DDR4-2400</td>
<td>8 GiB DDR3-1600</td>
</tr>
<tr>
<td>PCIe:</td>
<td>Gen3 x8</td>
<td>Gen3 x8</td>
</tr>
<tr>
<td>PULP Clusters:</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>Speed:</td>
<td>50 MHz</td>
<td>50 MHz</td>
</tr>
<tr>
<td>Where:</td>
<td>ETH/QUB</td>
<td>IBM Cloud</td>
</tr>
</tbody>
</table>

Server
IBM POWER8 Minsky:
Server
IBM POWER9 Witherspoon:

Accelerator Card

kwPilot v2.0 setup

CAPI 2.0
OpenCAPI 3.0

9V3

FPGA: VU3P-2 - FFVC1517
CLBs 1182
DSP Slices 2280
Block RAM 25.3Mbit + 90 Mbit UltraRAM

DRAM: 16GiB DDR4-2400

PCle: Gen3 x 16, Gen4 x8

PULP Clusters: 4
Speed: 50MHz
Where: IBM Cloud
CAPI/SNAP extensions for transprecision accelerators
Evaluation results on BLSTM

**Input:** Image of one text-line (3600 images)

**Algorithm:**
- FW and BW pass to updated LSTM cells
- Merge and produce final prediction

**Output:** Detected text sequence

\[
I^t = l(W_I x^t + R_I y^{t-1} + b_I) \\
i^t = \sigma(W_i x^t + R_i y^{t-1} + p_i \odot c^{t-1} + b_i) \\
f^t = \sigma(W_f x^t + R_f y^{t-1} + p_f \odot c^{t-1} + b_f) \\
c^t = i^t \odot I^t + f^t \odot c^{t-1} \\
o^t = \sigma(W_o x^t + R_o y^{t-1} + p_o \odot c^t + b_o) \\
y^t = o^t \odot h(c^t)
\]

Baseline performance: **98.23%** accuracy (float)

---

Rybakin, V., et al., Hardware Architecture of Bidirectional Long Short-Term Memory Neural Network for Optical Character Recognition, 2017.
Evaluation results on BLSTM

- The BLSTM AXI4 transprecision accelerator
  - Not a typical systolic array but a heterogenous dataflow engine
  - Multiple streaming engines in dataflow architecture, each with different precision (in fixed-point).
  - Weight & activation precision calibration
  - Quantization of trained FP32 models to custom fixed-point, while minimizing accuracy loss.
  - High-throughput & resource-efficient accelerator for FPGA.

8-bits less affect final precision with only 0.01%
Evaluation results on BLSTM

- Instantiated up to 4-Accs on the FPGA (KU3)
  - comparing with up to 8-POWER8 cores (affinity OpenMP threads / cores = 1:1).
  - saturating the available FPGA on-chip memory with 96%.
  - constant speedup of 4.8-5.6x using the same acceleration scalability step, i.e. OpenMP threads for software and FPGA accelerators for HW.
  - Minimal CPU usage of “HW solution” (interrupt-based CAPI API).
## Evaluation results on BLSTM

<table>
<thead>
<tr>
<th>Accelerators</th>
<th>FPGA (Reference, float32, math.h functions for activations)</th>
<th>FPGA (OPRECOMP) (Transprecision datapath, near-data processing, interleaved on-chip memories, approximation of activation functions)</th>
<th>GPU (OPRECOMP)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Configuration</strong></td>
<td>1-accelerator</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td><strong>Power [Watt]</strong></td>
<td>13.2</td>
<td>11.8</td>
<td>12.1</td>
</tr>
<tr>
<td><strong>Time [s]</strong></td>
<td>2274</td>
<td>75.2</td>
<td>44.6</td>
</tr>
<tr>
<td><strong>Energy per Image [J/Image]</strong></td>
<td>8.8</td>
<td>0.26</td>
<td>0.16</td>
</tr>
</tbody>
</table>

- **22x energy efficiency in kPixels/J** compared to CPU (POWER8 4.2GHz).
- **2.4x energy efficiency in kPixels/J** compared to GPU (high-end P100 in float16)
- Negligible accuracy loss compared to software (<0.6% for 3401 images).
Floorplans for 1-4 BLSTM transprecision accelerators

1-Accelerator
16% Logic, 47% BRAMs

2-Accelerators
19% Logic, 63% BRAMs

3-Accelerators
21% Logic, 78% BRAMs

4-Accelerators
24% Logic, 96% BRAMs

22x energy efficiency in kPixels/J compared to CPU (POWER8 4.2GHz).
2.4x energy efficiency in kPixels/J compared to GPU (high-end P100 in float16)
Floorplans for an array of 4-PULP interconnected controllable precision processors.

- AlphaData 8K5 XCKU115
- 16 cores
  - 4 clusters, 4 cores
Take away

• For the Summit:
  ▪ Establish the first **full transprecision framework** for future technologies
  ▪ Establish the first international **open transprecision computing community**

• For us:
  ▪ Lot of tasks in the pipeline, both SW & HW
    ○ Compiler support with transprecision pragmas
    ○ kwPilot v2.0 leveraging OpenCAPI3.0
    ○ Application porting to kwPilot, 12 ongoing from BigData, HPC, DL
    ○ Full support for float8, float16, float16alt in VP
    ○ Transprecision FPU in FPGA bitstream
    ○ ... TP DRAM controllers, TP Noise Tolerance Models, ...

• For you:
  ▪ There is plenty to be done
  ▪ Let us know if you want/can to contribute
Thank you!

http://oprecomp.eu/

https://github.com/oprecomp

IBM Research GmbH (IBM)
ETH Zürich (ETH)
Commissariat à l’énergie atomique et aux énergies alternatives (CEA)
Università degli Studi di Perugia (UNIPG)
Università di Bologna (UNIBO)
CINECA
Universitat Jaume I de Castellon (UJI)
The Queen's University of Belfast (QUB)
Greenwaves Technologies (GWT)
Technische Universität Kaiserslautern (TUKL)
Backup
Architecture for near-data transprecision accelerators