The 11th International workshop on OpenCL and SYCL

IWOCL & SYCLCON 2023

Towards a SYCL API for Approximate Computing

Lorenzo Carpentieri, University of Salerno (UNISA)

Biagio Cosenza, University of Salerno (UNISA)

Overview

- Approximate computing introduction
- Software techniques
- SYprox: a SYCL API for approximate computing
  - Perforation Schema
  - Reconstruction schema
  - Host and device perforation
- Experimental evaluation

Introduction to Approximate Computing

- Data/computation can be inaccurate and still produce acceptable results
- Trade accuracy for higher speedup or smaller energy consumption
- Many applications:
  - machine learning, neural networks
  - computer vision, image processing
  - signal processing
- Many techniques:
  - Hardware: approximate and faulty hardware, memoization etc.;
  - Software: perforation, mixed precision, synchronization elision;

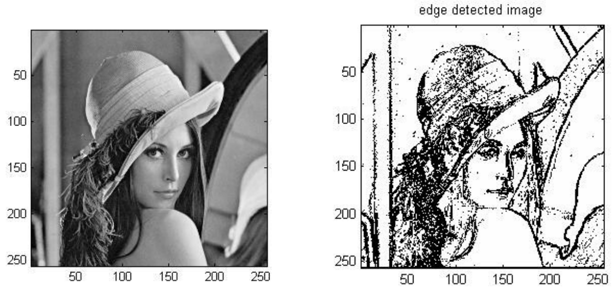
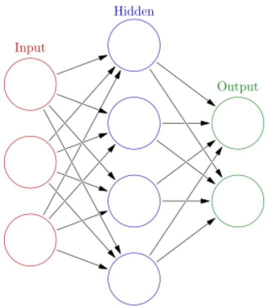


Image Processing



Neural Networks

Mixed precision

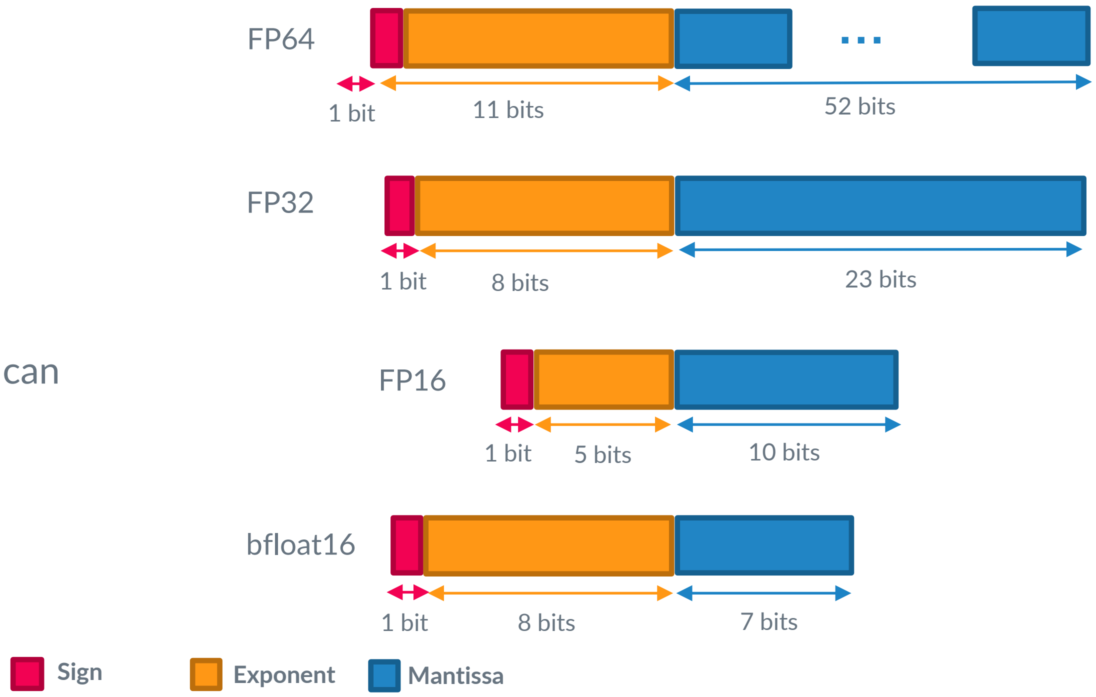
Mixed precision methods combine the use of different numerical formats in one computational workload.

Lower-precision pros:

- Faster computation and less memory footprint
- Transmit more numbers
- Use less energy

Cons:

- Limits the range of values we can represent
- Introduce quantization error



Loop and code perforation

```
// original loop
for (i = 0; i < N; i++){
  Instruction_1
  Instruction_2
  Instruction_3
}
```

```
// loop_perforation[1]
for (i = 0; i < N; i += k){
  Instruction_1
  Instruction_2
  Instruction_3
}
```

```
// code_perforation[2]
for (i = 0; i < N; i++){
  Instruction_1
  // skipped instruction
  Instruction_3
}
```

- Loop perforation (coarse grained approach):
  - skips loop iterations to reduce computation;
  - k is the skip factor used for tuning accuracy vs. performance
- Code perforation (fine grained approach):
  - skips loop instructions to reduce computation

[1] Sidiroglou-Douskos, Stelios, et al. "Managing performance vs. accuracy trade-offs with loop perforation.“ *Proceedings of the 19th ACM SIGSOFT symposium*
[2] Li, Shikai, Sunghyun Park, and Scott Mahlke. "Sculptor: Flexible approximation with selective dynamic loop perforation." (SC 2018)

From loop to data perforation

- Many applications are memory-bound
  - computation is cheap, memory access is expensive
- Data often contains redundancy
  - many applications can deal with some amount of error
- Data perforation:
  - skip the loading of redundant parts in input data
  - exploit data locality to reconstruct perforated data, reducing the final error

a) Original b) Perforated c) Reconstructed

Input and output reconstruction

A reconstruction phase is needed in order to reduce the error introduced by the data perforation:

- **output reconstruction**[1] (high error);
- **input reconstruction**[2] (less error).

```
// original loop
for (i = 0; i < n; i ++) {
  output[i] = calc(input[i]);
}
```

```
for (i = 1; i < n; i += 3) {
  output[i] = calc(input[i]);
  // output_reconstruction1
  output[i+1] = output[i];
  output[i+2] = output[i];
}
```

```
for (i = 0; i < n; i += 3) {
  x0 = input[i];
  x2 = input[i+2];
  // input_reconstruction2
  x1 = (x0+x2)/2;
  output[i] = calc(x0);
  output[i+1] = calc(x1);
  output[i+2] = calc(x2);
}
```

[1] *Samadi, Mehrzad, et al. "Paraprox: Pattern-based approximation for data parallel applications." Proceedings of the 19th international conference on Architectural support for programming languages and operating systems. 2014.*

[2] Maier, Daniel, Biagio Cosenza, and Ben Juurlink. "Local memory-aware kernel perforation." *Proceedings of the 2018 International Symposium on Code Generation and Optimization*.

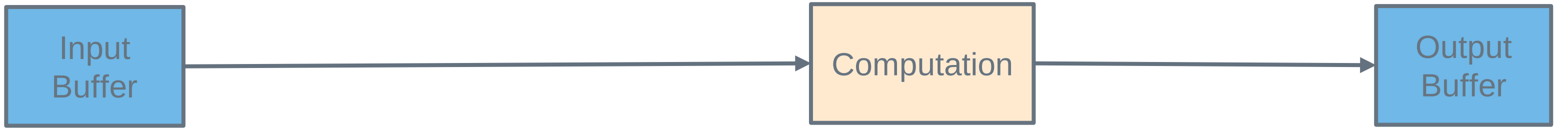
SYprox: a SYCL API for Approximate Computing

SYprox a portable SYCL API for developing approximate computing techniques:

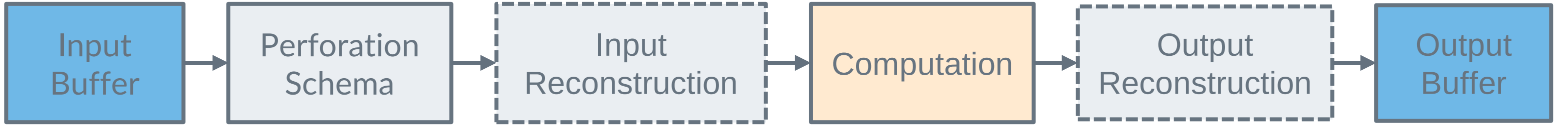
- New perforation approach:
  - Host perforation
  - Device perforation
- Different perforation schemes
- Input and output reconstruction
- Data perforation + mixed precision

```
pbuffer<half, 2, pcol<half> buf_a(a,range<2>{N,N});
// output reconstruction with lerp
pbuffer <half,2,pcol::lerp>out_buf(out,range<2>{N,N});
// global size and work group size
range<2> gl{N,N/2}, ws{32, 32};
q.submit([&](handler &h){

paccessor<float,2,prow<float> > perf_acc{buf_a, h, read};

h.parallel_for(nd_range<2>{gl,ws},
[&](nd_item<2> it){
  id<2> id = it.get_global_id();
// acc_a data are perforated host side
  out_acc[id*2] = acc_a[id] * 2;
// perf_acc data are perforated device side
  out_acc[id*2] = perf_acc[id] * 2;
 });
}
```

SYprox approach



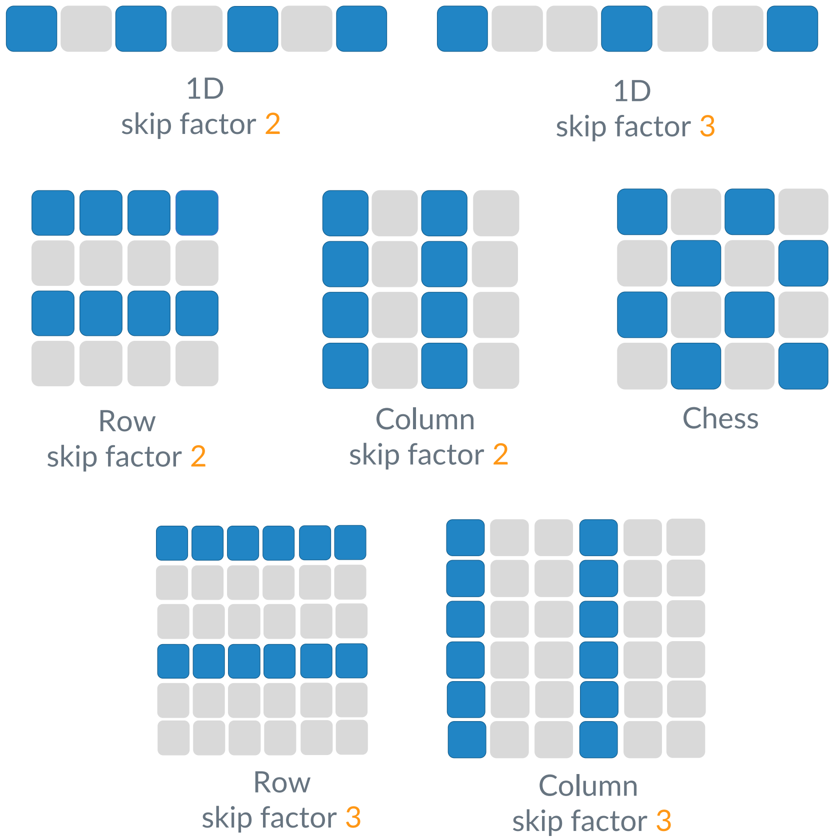
a) Accurate execution



b) SYprox approach

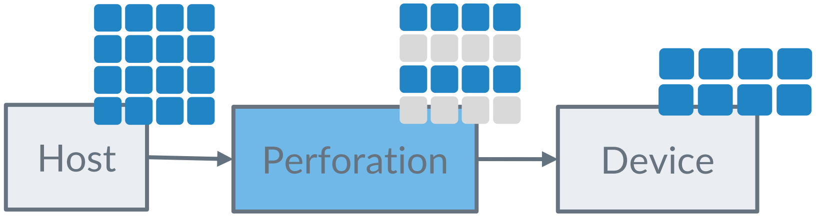
Perforation Schemes

- Perforation schemes define which data should be not computed.
- SYprox provides 4 built-in perforation schemes.
- The skip factor for the 1D/Row/Column schemes defines the number of elements/rows/columns to skip.



Host perforation (pbuffer)

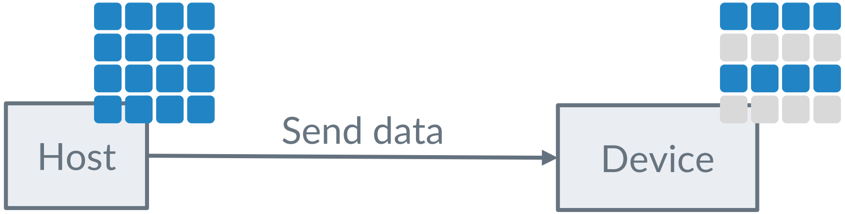
- Data perforation happens before sending data on the device;
- Less data transfer between host and device;
- All the perforation schemes adopted are aligned with memory architecture




```
pbuffer<half, 2, prow<half>
            buf_a(a, range<2>{N,N});
buffer<half, 2>
            out_buf(out, range<2>{N,N});
range<2> gl{N/2,N}, ws {32, 32};
q.submit([&](handler &h){
 h.parallel_for(nd_range<2>{gl,ws},
 [&](nd_item<2> it){
  id<2> id = it.get_global_id();
  out_acc[id*2] = acc_a[id] * 2;
 });
}
```

Device perforation (paccessor)

- Data perforation happens on the device;
- Send all data from host to device;
- Perforation schemes can be affected by the array memory layout (e.g. row-major, column major)




```
buffer<half, 2> buf_a(a, range<2>{N,N});
buffer<half, 2>
            out_buf(out, range<2>{N,N});
range<2> gl{N/2,N}, ws {32, 32};
q.submit([&](handler &h){
 paccessor<half, 2, prow<float>>
            perf_acc{buf_a, h, read};
 h.parallel_for(nd_range<2>{gl,ws},
 [&](nd_item<2> it){
  id<2> id = it.get_global_id();
  out_acc[id*2] = perf_acc[id] * 2;
 });
}
```

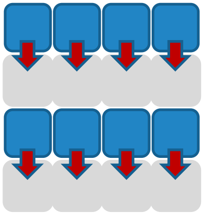
Reconstruction strategies

SYprox provides two reconstruction strategies:

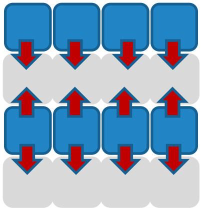
- **Output reconstruction** approximates perforated data after computation;
- **Input reconstruction:** perforated elements are reconstructed in local memory before computation.

SYprox provides 3 built-in way to reconstruct data:

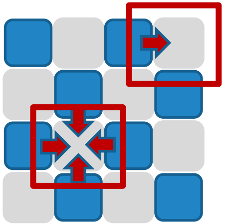
- Nearest neighbour;
- Basic linear interpolation;
- Stencil interpolation;



Nearest neighbour



Linear



Stencil

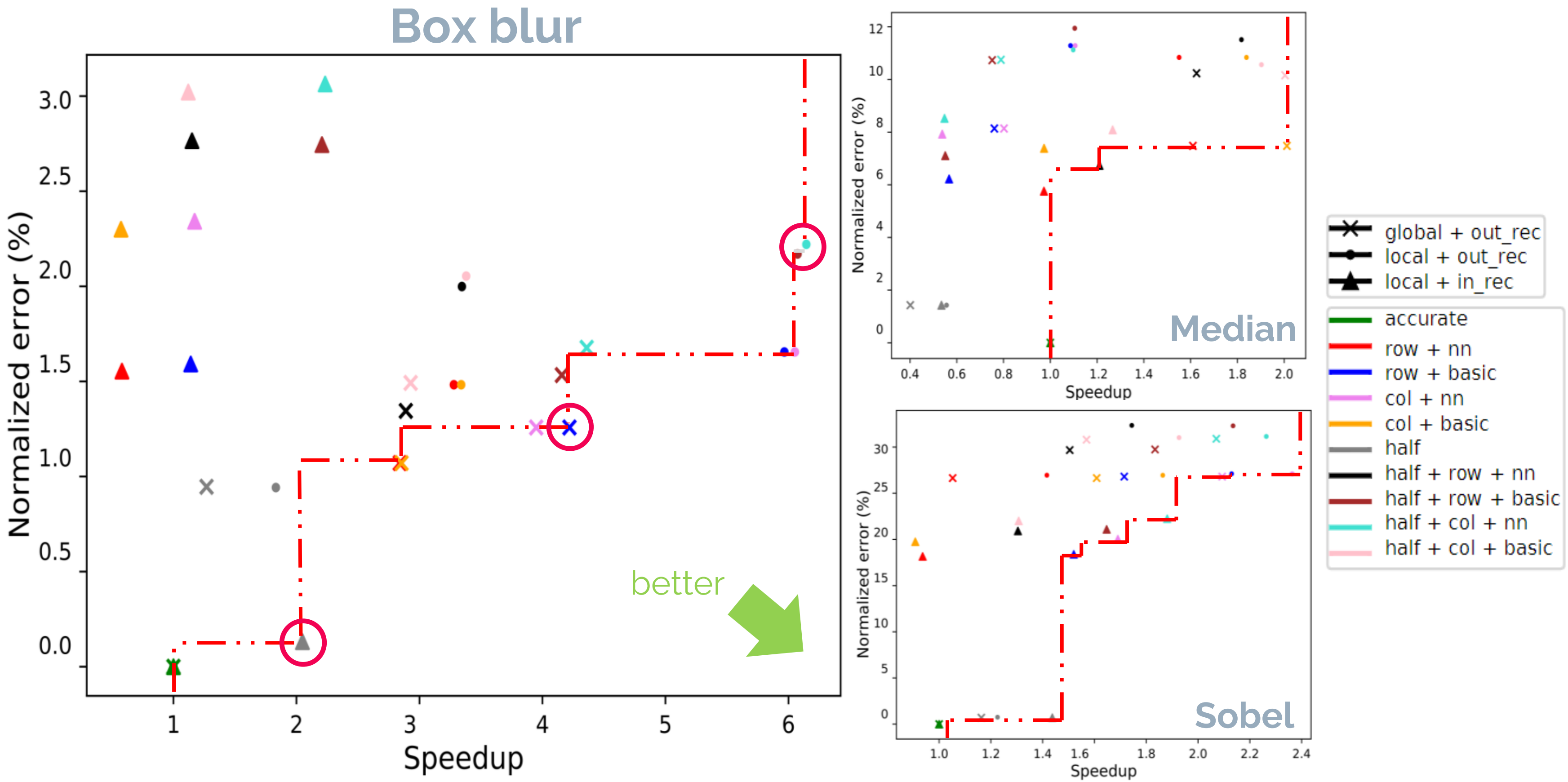
Evaluation

- Implemented 3 image processing applications using SYprox;
- Combined different approximate computing techniques:
  - Data perforation;
  - Input and output reconstruction;
  - Mixed precision using half precision floating point.
- Run applications on NVIDIA V100 GPU;
- Measure run time and error:
  - Speedup 1.2x to 6x;
  - Average error less than 10%.

Applications

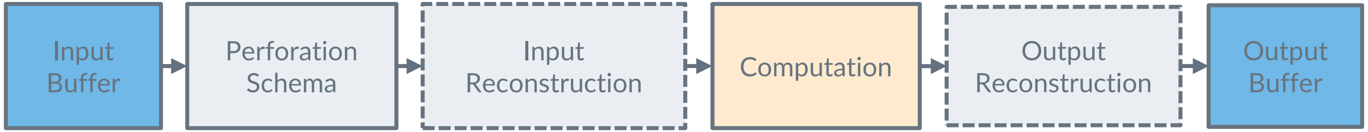
- `Box blur`
- `Median filter`
- `Sobel filter`

Results



Box blur
Normalized error (%)
Speedup
better
Median
Normalized error (%)
Speedup
Sobel
Normalized error (%)
Speedup
global + out_rec
local + out_rec
local + in_rec
accurate
row + nn
row + basic
col + nn
col + basic
half
half + row + nn
half + row + basic
half + col + nn
half + col + basic

Summary



SYprox: a SYCL API for approximate computing

- Combined different approximate computing techniques:
  - Data perforation;
  - Input and output reconstruction;
  - Mixed precision using half precision floating point.
- Speedup 1.2x to 6x;
- Average error less than 10%.

Contacts:
Lorenzo Carpentieri: lcarpentieri@unisa.it
Biagio Cosenza: bcosenza@unisa.it