SYprox: Combining Host and Device Perforation with Mixed Precision Approximation on Heterogeneous Architectures

Lorenzo Carpentieri
Department of Computer Science
University of Salerno
Fisciano, Salerno, Italy
lcarpentieri@unisa.it

Biagio Cosenza
Department of Computer Science
University of Salerno
Fisciano, Salerno, Italy
bcosenza@unisa.it

Abstract

Approximate computing is an emerging paradigm that aims to exploit the inherent error tolerance of many applications, particularly in domains such as image processing and machine learning. Taking advantage of this property, applications can trade off accuracy for significant gains in performance and power consumption. Existing approximation techniques for GPUs are limited to very specific approaches, do not fully exploit the host-device execution model, and are often restricted in terms of programming models and supported target hardware. This paper introduces SYprox, a new approximate computing framework based on SYCL that allows programmers to easily implement heterogeneous approximated applications. SYprox supports multiple techniques, including data perforation, signal reconstruction, and mixed precision, and allows them to be combined to support a wide range of approximations. In particular, SYprox extends existing perforation approaches to allow both host and device data perforation. Experimental results show that SYprox's approximations are Pareto dominant with respect to state-of-the-art approaches and are portable to AMD, Intel and NVIDIA GPUs.

CCS Concepts

• **Software and its engineering** → **Software development techniques**; **Parallel programming languages**; **Software performance**;

Keywords

Approximate Computing, Programming Models, SYCL, Data Perforation, Reconstruction, Mixed Precision.

ACM Reference Format:
Lorenzo Carpentieri and Biagio Cosenza. 2025. SYprox: Combining Host and Device Perforation with Mixed Precision Approximation on Heterogeneous Architectures. In *2025 International Conference on Supercomputing (ICS '25), June 08–11, 2025, Salt Lake City, UT, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3721145.3725741

1 Introduction

In many real-world applications, it is recognized that absolute precision is not always necessary, especially when weighed against the potential benefits of consistently improved performance, increased energy efficiency, and better resource utilization. This has led to the development of approximate computing techniques based on the observation that many computational tasks can produce acceptable results even when subjected to controlled inaccuracies.

However, the use of approximate computation techniques presents several challenges [2, 24]. By trying to minimize error and maximize other metrics, typically performance, we are actually formulating a multi-objective problem. Since approximation accuracy and performance are not correlated, this leads to a multi-objective problem without a single optimal solution, but rather a set of Pareto optimal dominant solutions [7]. Dealing with multi-objective problems and Pareto sets makes the optimization process more complicated for users, who need to understand the different trade-offs at stake.

The second challenge is presented by the wide range of techniques that span multiple levels of the computing stack, from hardware design to software implementations. From a software perspective, some of the most promising approaches are the use of lower-precision arithmetic units in mixed precision methods [4], the perforation of data, either input or output, and the use of signal reconstruction techniques to mitigate error [19, 20]. Those promising techniques are typically used standalone.

However, the potential combined application of such techniques can unleash unprecedented improvement, as it results in more overall approximations in the multi-objective accuracy/performance space, ultimately leading to better Pareto optimal solutions. In fact, this is not easy because existing techniques are difficult to apply and often rely on very specific approaches [17, 36], compilers [14, 32, 35], or programming languages [9, 25, 38]. While compiler approaches are desirable, they are also complex and often limited to very specific architectures.

This paper proposes a novel approximation framework based on the SYCL programming model, which aims to easily implement approximated applications on a wide range of GPUs from different vendors, as well as providing a way to combine state-of-the-art approximation techniques in a composable way, so that they can be easily combined to deliver unprecedented gains in accuracy and performance.

This paper makes the following contributions:

- *SYprox*, a new approximate computing interface based on SYCL that allows programmers to easily implement heterogeneous approximated applications with state-of-the-art approximation techniques such as perforation, mixed precision, and signal reconstruction;

ICS '25, Salt Lake City, UT, USA

ACM ISBN 979-8-4007-1537-2/25/06
https://doi.org/10.1145/3721145.3725741

- a new perforation approach called *host perforation*, which applies perforation on the host data and only sends/receives perforated data to/from the device;
- for the first time, the ability to *combine* data perforation and reconstruction techniques (on the host, device, or both) with mixed precision to support an unprecedented number of approximations;
- experimental results of SYprox approximation techniques (individually and in combination) and a comparison with state-of-the-art frameworks (HPAC [9, 25] and Maier et al. [20]) for eight benchmarks on 100 input datasets evaluated on GPUs from different vendors, including AMD, Intel, and NVIDIA.

The rest of the paper is organized as follows. Section 2 discusses approximate computing techniques and related works. Section 3 provides an overview of the SYprox framework. Section 4 introduces the SYprox interface. Section 5 presents the host perforation technique, while Section 6 describes our combined approximation strategy. Section 7 shows the experimental setup and evaluation, and Section 8 concludes the paper.

2 Background and Related Work

Over recent years, several software techniques have emerged to enable approximate computation, helping applications enhance performance while retaining acceptable accuracy levels. This section provides an overview of the key approaches and how they are implemented in existing programming models [24].

2.1 Approximation Techniques

Mixed Precision [16, 28] involves the use of different levels of precision within the same program. Rather than uniformly adopting high precision data types (e.g., double precision), mixed precision methods selectively apply lower precision (e.g. half precision) in suitable situations, consequently lowering computational workload, memory usage, and energy demands. By carefully balancing precision levels, mixed precision can significantly enhance performance with minimal accuracy loss.

Loop Perforation [27, 36] is a technique that reduces the number of iterations in a loop. Instead of executing every iteration, the loop is "perforated" by skipping some iterations based on a predefined pattern, thereby reducing overall execution time and energy consumption.

Data Perforation [8, 19] is similar to loop perforation, but operates at the level of data rather than the control flow. In this technique, a subset of data points is skipped during computation, effectively reducing the total execution times.

Reconstruction techniques try to fill the accuracy gaps left by skipped computations or data points to maintain the quality of the output. Missing values are approximated on the basis of available data. There exist two types of reconstruction: output reconstruction approximates the data that were perforated after the computation using interpolation between the output elements; input reconstruction involves a set of local reconstruction techniques that work with local memory and efficiently combine the sparse data fetched by global perforation schemes while consistently improving the accuracy of the approximation [21, 22]. Reconstruction allows perforation techniques to achieve significant computational savings while maintaining an acceptable level of accuracy in the final output.

With respect to the state-of-the-art techniques, we implemented a new data perforation approach called host perforation, which applies perforation on the host data and only sends/receives perforated data to/from the device

2.2 Programming Models Approaches

With the increasing demand for higher performance in computing systems, approximate computing methods have become essential to improve computational efficiency. To facilitate the adoption of these techniques, various frameworks have been implemented.

Manual Approach. Approximate computing techniques can be applied directly by the programmer without any assistance from compilers or specialized programming models. Maier et al. [21, 22] define a local memory-aware approximation approach based on loop perforation that approximates the input data of GPU applications by reducing the amount of data loaded from global memory and reconstructing a high-accuracy approximation with local reconstruction techniques. *With respect to manual approaches, our work proposes a library-based framework that automatically applies approximations, eliminating the need for manual implementation by programmers.*

Compilers. Although manual approaches method can yield significant performance gains, it places a heavy burden on developers and is error-prone due to the lack of automated tools. Several works have focused on integrating approximate computing techniques directly into compilers. These compilers can automatically identify opportunities for approximation and apply transformations that improve performance, such as reducing computational precision [5, 6, 15, 16, 28, 33], loop perforation [1, 10, 18, 19, 23, 29, 31], or relaxing memory consistency [3, 17, 26]. Paraprox [29] is a software solution applicable to commodity hardware that automatically discovers and approximates common patterns such as map, scatter/-Gather, reduction, and others. For each pattern, Paraprox provides an approximation strategy with one or more knobs that can be used to navigate the performance-accuracy trade-off. Lou et al. [19] presented a new prototype language and compiler that applies loop perforation and output reconstruction only to the image processing pipeline. Laguna et al. [15] propose GPUMixer a tool to tune the data precision of applications on GPUs. Although common mixed-precision approaches change the precision of variable declarations, a fine-grained approach is to express the precision of each floating-point operation in the program. GPUMixer provides a practical method to select the computations to be performed on FP32 or FP64 precision so that user-defined accuracy constraints are maintained and performance is significantly improved. By automating the application of approximations, compiler-based approaches reduce the complexity of adopting approximate computing techniques. However, with fully automatic approaches, programmers are required to trust the system, with no way to intervene when opportunities are overlooked or invariants are compromised. *In contrast, our work abstracts compiler-level approximations to the library level, without requiring novel programming model prototypes and specialized*

SYprox API

```
range<2> r = range<2>{N,N};
pbuffer<half,2,prow<half,2>> inBuf(a,r);
pbuffer<float,2,prow<float,1,nn_out>> outBuf(b,r);
prange<> gl{N,N}, ws{32, 32};
q.submit([&](handler &h){
 paccessor<float,2,prow<float>> in_acc{in,h,read};
 paccessor<float,2> out_acc{out,h};
 h.parallel_for(gl,[&](id<2> id){
     out_acc[id] = perf_acc[id] * 2;
  });
}
```

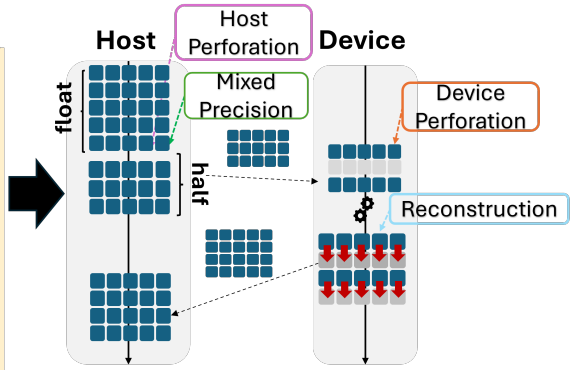


Figure 1: Overview of SYprox approximation

compilers. Furthermore, our approach applies to general-purpose applications, enabling approximation beyond a specific domain.

Language Extensions. Programming models have also been extended to support approximate computing by providing abstractions and frameworks that allow developers to specify approximate behaviors more easily [9, 37, 38] . These models often provide APIs, annotations, or directives that enable programmers to define where and how the approximation should occur. This simplifies the development process by integrating approximate computing into higher-level programming constructs, offering a more structured and less error-prone approach than manual or compiler methods. Parasyris et al. [25] propose HPAC, a framework that extends the OpenMP programming model in order to provide approximate computing techniques. This approach incorporates pragma-based annotations composable with standard OpenMP annotation, code transformations, and, analysis to enable developers to identify approximation opportunities in their applications. HPAC facilitates the integration of loop perforation and memoization, offering a mechanism to investigate the accuracy-performance trade-offs for a given application. This system suggests potential approximations that can enhance performance while maintaining the accuracy levels defined by the user. The framework has been extended to enable approximate computing techniques also on GPUs [9] by extending the OpenMP annotation for GPU offloading. *With respect to the state of the art, we propose the first header-only library in SYCL that implements data perforation, reconstruction, and mixed precision allowing programmers to define configurable and heterogeneous approximated application.*

Table 1 compares the capability of several related works and SYprox. By comparison, our proposed solution introduces an innovative library-based method for approximate computing in heterogeneous architectures, implementing host/device perforation, reconstruction, and mixed precision. To the best of our knowledge, SYprox is the first library-based framework that allows programmers to write heterogeneous applications that combine host/device perforation with input/output reconstruction and mixed precision.

Table 1: Comparison against state of the art

	Approach	*Perforation*	*Reconstruction*	*Mixed Precision*
Maier et al. [20]	manual	device	input	✗
Paraprox [30]	compiler	loop	output	✗
HPAC [9]	compiler	loop	✗	✗
GPUMixer [15]	compiler	✗	✗	✓
Lou et al. [19]	compiler	image*	output	✗
SYprox	library	host, device	input, output	✓

*Lou et al. apply device perforation only to images.

3 Overview

SYprox is a header-only library designed to extend the SYCL programming model with advanced approximate computing techniques, including perforation, reconstruction, and mixed precision. This extension enables developers to integrate approximations into their applications with minimal code modifications. Figure 1 illustrates a SYCL code enriched with approximate computing features and shows the combination of applied approximations. SYprox provides an easy-to-use and highly customizable interface. Programmers can implement approximations using built-in parameterizable schemes (prow, pcol) for perforation (pbuffer, paccessor) and reconstruction (input and output), develop custom schemes tailored to specific applications, or mix different data types for mixed precision computing. SYprox introduces a novel technique called *host perforation*, which selectively perforates data on the host side before transferring them to the device, thus optimizing both computation and communication. In Figure 1 the host data are perforated and reduced to the `half` data type before data transfer. The data on the device are processed in a perforated shape using *device perforation* (`paccessor`). The perforated elements are then reconstructed according to the selected schema (`nn_out`) and sent back to the host. The ability of SYprox to combine different approximations increases the range of feasible configurations by generating new trade-offs between performance and accuracy (Section 6). The flexibility of the SYprox interface not only facilitates the adoption of

approximate computing techniques across heterogeneous architectures but also expands the approximation domain by enabling new Pareto-optimal configurations.

4 SYprox Programming Interface

4.1 SYCL Programming Model

```
1 queue q(gpu_selector_v);
2 buffer<int, 2> outBuf(out,range<2>(N,N));
3 q.submit([&](handler& h) {
4  accessor outAc(outBuf,h);
5  h.parallel_for(r,[=](id<2> i) {
6   outAc[i] += outAc[i] * 2;
7  });
8 });
```

Listing 1: SYCL accurate code

The SYprox library is based on SYCL a single-source C++ programming model designed to improve the performance and portability of applications running on heterogeneous architectures, such as CPUs, GPUs, and other accelerators.
Listing 1 shows a simple SYCL code. The `parallel_for` (line 5) define a kernel code to be executed on a device such as a GPU. The device on which the kernel code is executed is represented by a queue (line 1). SYCL offers two methods for handling data transfers between the host and the device: a pointer-based strategy called Unified Shared Memory (USM) and the `buffer`/`accessor`. Our work leverages `buffer` and `accessor` but can be easily extended to USM. Buffers (lines 2) abstract memory management and represent a range of memory that can be used on either the host or the device. Accessors (lines 5) are used to specify how to access the data within a buffer (e.g read or write). Differently from USM, with `buffer` and `accessor` the SYCL runtime automatically manages data movements between the host and the device. The next sections describe in detail how SYprox extends the `buffer` and `accessor` of SYCL to enable approximate computing features.

4.2 Data Perforation

The SYprox library implements two types of data perforation: the *device perforation* defined by Maier et al. [20] and a novel approach called *host perforation*.

```
1 // buffers ...
2 q.submit([&](handler &h){
3  paccessor<float,2,prow<float,2>> inAc{inBuf,h};
4  paccessor<float,2,prow<float,2>> outAc{outBuf,h};
5  h.parallel_for(prange<>{N,N},[&](id<2> id){
6     outAc[id[0]][id[1]] = inAc[id[0]][id[1]]*2;
7  });
8 }
```

Listing 2: SYprox code with device perforation

4.2.1 Device Perforation. The *device perforation* is implemented as an extension of the SYCL `accessor` class. The `paccessor` adds a new template parameters to the SYCL `accessor` which specify how the data are accessed on the device. The class implements an on-line perforated access to the data by overloading the subscript operator ([]) so that, while all original data remain in memory, only specific parts are accessed according to the selected perforation schema (Section 4.2.3). Using a row schema and skip factor of x the access to `a[i][j]` is translated into `a[i*x][j]`. Listing 2 shows a SYprox code with device perforation. Lines 3-4 define two bidimensional perforated accessors using a row schema with a skip factor of two. In line 6 according to the row schema, the data will be accessed in a perforated shape. The access to the element at the index `{id[0],id[1]}` translates in `{id[0]*2, id[1]}`. The `prange` semantic (line 5) defines a set of deduction rules to automatically infer the `range` of the kernel according to the perforation schema used.

```
1 range<2> r{N,N}
2 pbuffer<float,2,prow<float,2>> inBuf(in,r);
3 buffer<float,2> outBuf(out,r);
4 q.submit([&](handler &h){
5  // accessors...
6  h.parallel_for(prange<>{N, N},[&](id<2> id){
7   outAc[{id[0]*2,id[1]}]=inAc[id]*2;
8  });
9 }
```

Listing 3: SYprox code with host perforation

4.2.2 Host Perforation. SYprox implements *host perforation* by defining a `pbuffer`, which extends the SYCL buffer. The `pbuffer` introduces new template parameters into the SYCL buffer to specify the type of perforation scheme to be used (4.2.3). The `pbuffer` intercepts the SYCL `buffer` constructor, applying the perforation to the data before the invocation of the `buffer` constructor. This process involves iterating over the elements passed to the `pbuffer`, perforating data based on the approximation strategy defined by the `ApproxSchema` class (e.g. `prow`, `pcol`). Listing 3 shows a SYprox code with *host perforation* and a row scheme (line 2).

4.2.3 Perforation and Reconstruction Schemes.

SYprox `pbuffer` and `paccessor` class perforate and reconstruct data according to a perforation and reconstruction schema defined by a specialization of the SYprox `ApproxSchema` class. SYprox defines for the `pbuffer` and the `paccessor` three built-in schemes configurable by the type of data used and the number of data to skip (`skip_factor`).

Strided scheme skips a fixed number of data points in a consistent stride. For example, with a skip factor of 2, every other data point is skipped.

Row scheme applies to bi-dimensional data. Skip entire rows of data according to the defined skip factor.

Col is similar to the row scheme but operates column-wise.
For each schema, SYprox also provides input and output reconstruction strategies of two types: *nearest neighbor* where perforated elements are reconstructed using the neighbor element; *lerp* where perforated elements are reconstructed with a linear interpolation of two or more elements. Both reconstructions are implemented in an optimized way leveraging the SYCL sub-group and group algorithms. The SYprox `ApproxSchema` class is designed with flexibility in mind to serve as a base class for customizing data perforation methods according to the specific use case. Programmers can define any kind of static approximation schema by implementing the method defined by the `ApproxSchema` class.

```
range<2> r = range<2>{N,N};
pbuffer<float,2,prow<float,2,nn_out>> inBuf(in,r);
q.submit([&](handler &h){
 paccessor<float,2> inAc{inBuf,h,read_write};
 h.parallel_for(prange<>{N,N},[&](id<2> id){
  outAc[id] = inAc[id] * 2;
 });
}
```

Listing 4: SYprox code with host perforation and output reconstruction

Listing 4 shows a SYprox code that combines host perforation with a row perforation schema and output reconstruction with the nearest-neighbor reconstruction schema (line 2).

4.3 Mixed Precision

```
std::vector<float> in;
pbuffer<half,2> inBuf(in,range<2>{N,N});
buffer<float,2> outBuf(out,range<2>{N,N});
q.submit([&](handler &h){
 // accessors...
 h.parallel_for(prange<>{N, N},[&](id<2> id){
  outAc[id] = inAc[id]*2;
});
}
```

Listing 5: SYprox code with mixed precision

Listing 5 shows a mixed precision computation with `float` and `half` data type with SYprox. The `float` data in the `in` vector are converted in `half` precision during buffer construction thorugh the `sycl::reinterpret` function . Then the kernel performs a mixed-precision computation on `float` and `half` data. The library supports all the lower precision formats defined in the SYCL standard and also other formats such as `bfloat16`, which are implemented as experimental extensions in DPC++ [12].

5 Host Perforation

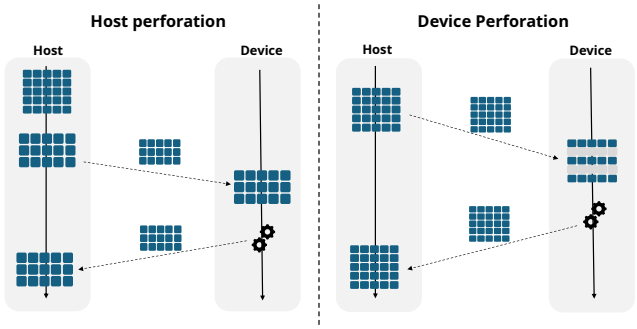


Figure 2: Host and device perforation approach.

This section presents *host perforation* a novel data perforation technique implemented in SYprox. Figure 2 shows a comparison between the traditional *device perforation* and the *host perforation* approach. *Host perforation* performs data perforation on the host before sending the data to the device. In contrast, device perforation requires transferring all data from the host to the device, where they are then accessed in a perforated shape according to a pattern defined by the SYprox approximation schema (Section 4.2.3). The *host perforation* offers two distinct advantages over the device perforation. Firstly, it significantly reduces the amount of data transfer needed between the host and the device. This reduction in data transfer can lead to improved performance in applications where data movement is a bottleneck. Secondly, host perforation provides a better cache utilization, since eliminates data access issues due to the data layout. By perforating the data on the host, accesses to the device data can be performed continuously, maximizing cache utilization (Section 7.3). However, in *host perforation*, once the data have been perforated in the host, they cannot be used on the device. This limitation may affect scenarios where devices require direct access to perforated data for further processing or computations. Moreover, host perforation is only beneficial when the time required to perforate the data on the host is less than the time needed to transfer the full dataset to the device. In our experiments (Section 7), we tested data sizes up to the maximum available and did not observe cases where device perforation was more efficient. However, this may not hold for systems with higher host-device memory bandwidth or hosts with lower compute capabilities.

6 Combined Approximation

This section demonstrates how the integration of various approximation techniques can lead to improvements in both the accuracy and efficiency of applications. Figure 3 illustrates the speedup and error for the individual approximations and how they can be combined to generate new performance and accuracy trade-offs using a blur filter application as an example. To perforate the data, we applied a skip factor of two, reducing the number of rows and columns by half for row and column schema, respectively. The red line represents the Pareto frontier: a set of optimal solutions where no solution can be improved without degrading another objective. Here, the Pareto front helps to identify trade-offs between performance and error.

In the following sections, we provide insight into the behavior of individual approximation techniques and their combination.

6.1 Individual Approximation

SYprox implements 5 approximation techniques: Mixed Precision, Device and Host Perforation, and Reconstruction. Each approximation can be represented as a set of different configurations.

Mixed Precision. For mixed precision, we used floating point data as a baseline and half precision as a lower precision data type ($Mp = \{ floating, half \}$). With the *half* configuration we mix floating-point and half data types. Unlike data perforation, which skips data processing, mixed precision results in only a 1% error, as it only processes the entire dataset with reduced precision.

Device Perforation. The possible configurations with device perforation depend on the number of schemes implemented. In our experiment, we used the row and column schemes with a skip factor of two, resulting in two configurations ($Dp = \{\text{row, col}\}$). Data perforation without reconstruction leads to an error of approximately 50% since for skip factor of two half of the data are perforated. Furthermore, the column schema exhibits a significant performance slowdown due to increased cache misses caused by the data layout.

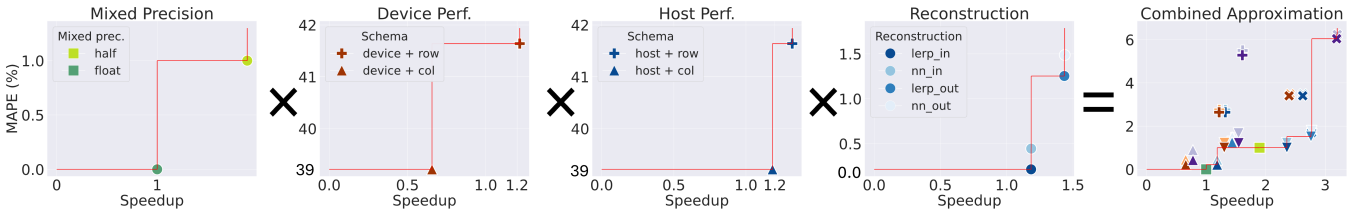


Figure 3: Individual and combined approximation

Host Perforation. The host approach shares the same number of configurations as the device perforation. Both approaches result in the same error, since they operate on the same data. However, the host approach achieves a higher speedup due to reduced data transfer and optimized data layout for the column schema.

Reconstruction. SYprox provides two types of reconstruction schemes nearest-neighbor (`nn`) and linear interpolation (`lerp`) both implemented with input and output reconstructions, resulting in four different configurations (R = {`nn_in`, `nn_out`, `lerp_in`, `lerp_out`}). Nearest-neighbor reconstruction is faster but less accurate since the reconstruction only uses one of the computed elements to approximate the perforated data , while linear interpolation provides higher accuracy with only a minor impact on speedup due to the time spent in the interpolation. Input reconstruction involves reconstructing data before computation, resulting in less error but lower performance. Output reconstruction, on the other hand, reconstructs data after computation, which usually leads to higher error but also higher speedup.

6.2 Approximation by Combining Techniques

The SYprox interface provides a way to combine individual approximations. When combining these techniques, the number of available configurations is equal to the Cartesian product of each individual approximation: $|Mp| \times |Dp| \times |Hp| \times |R|$, resulting in an approximation space composed of 32 points. Combining different approximation techniques allows us to expand the approximation domain and explore new performance-accuracy trade-offs. In fact, Figure 3 (Combined Approximation) shows that we can achieve a 3x speedup with a maximum error of 6%. However, not all combinations yield efficient results. For example, any combination that applies *device perforation* with a column schema typically shows lower performance due to cache misses related to the type of data layout. Composing different approximations can also increase the error. However, signal reconstruction techniques can help mitigate the error by generating new Pareto-optimal solutions. Notice that the number of available configuration can also be higher since we can have more data types (e.g. `bfloat16`) or for perforation schemes `skip_factors` higher then two.

7 Experimental Evaluation

In this section, we present an analysis to assess the efficacy of approximate computing techniques comparing SYprox with state-of-the-art approaches [9, 20, 25].

7.1 Experimental Setup

7.1.1 Benchmark Description. We conducted the experimental evaluation on eight benchmarks described in Table 2.

Table 2: Applications used for experimental evaluation

Benchmark	Domain	Size	Kernels' LoC
median	Medical imaging	3072^2	45
sobel	Edge detection	3072^2	40
blur	Image blurring	3072^2	21
tv	Edge detection	3072^2	18
gaussian	Image blurring	3072^2	34
hotspot	Phisical simulation	4096^2	150
lavaMD	Molecular dynamics	128^3	219
leukocytes*	Medical imaging	219x640	281

*leukocytes consists of 3 kernels that process several frames.

The benchmarks were selected to have a direct comparison of the proposed host perforation approach with the device perforation method described by Maier et al. [20]. We implemented the applications in SYCL using the SYprox library to apply our approximation. To ensure a direct comparison with the HPAC framework [9, 25], all benchmarks were ported from SYCL to OpenMP with GPU offloading and then integrated into the framework using specific HPAC directives for approximation.

The image processing benchmarks are executed on a dataset composed of 100 images of 3072^2 size to analyze the error variation of each approximation on different inputs. The input data sets are taken from the USC-SIPI Image Database [39]. For `lavaMD` and `hotspot` we randomly generated 100 input files. The speedup is calculated using the accurate application as a baseline, while the error uses the mean absolute percentage error, defined as MAPE = $\frac{100\%}{n} \sum_{i=1}^{n} \left| \frac{I_i - \hat{I}_i}{I_i} \right|$, where I_i and $\hat{I}_i$ are the accurate and approximated data, respectively.

7.1.2 Parameter Description. In all experiments, we executed host and device perforations with row and column schemes using a skip factor of 2, while for mixed precision, we adopted floating and half-data types. The half-configuration mix floating point and half-data types. For the reconstruction step, we applied the input (`in`) or output (`out`) reconstruction with the nearest neighbor (`nn`) or linear interpolation (`lerp`). Benchmarks implemented with local

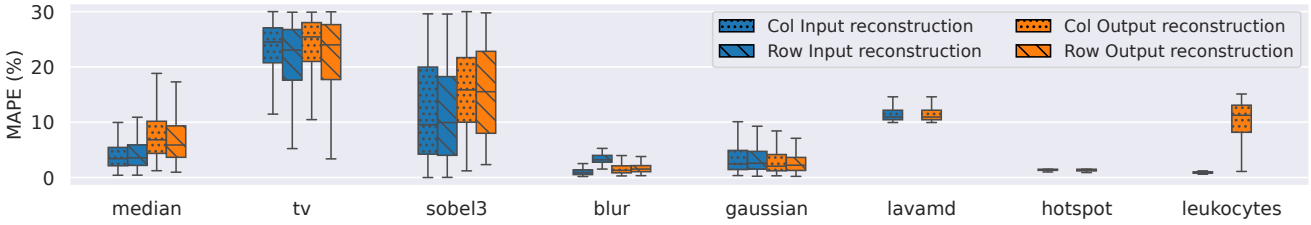
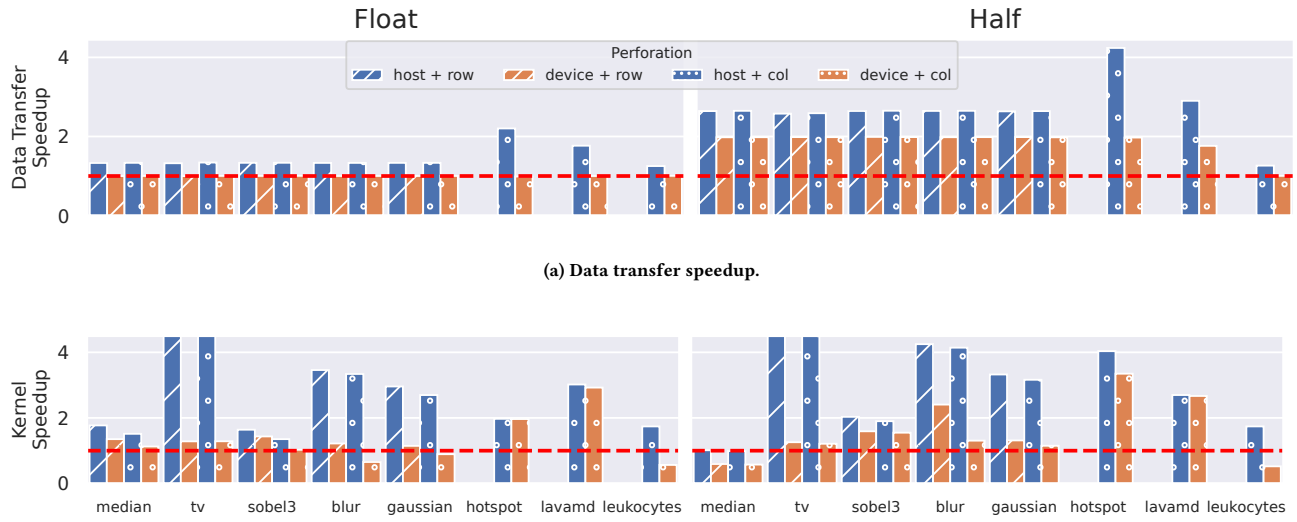


Figure 4: Error of different approximate strategies.



(a) Data transfer speedup.

(b) Kernel speedup.

Figure 5: Data transfer 5a and kernel 5b speedup of all applications for host/device perforation and float/half precision. The red line represents the baseline defined as the accurate execution.

memory use a size defined by the block size, which in our case is fixed to 32x32.

7.1.3 Software and Hardware Configuration. For the experimental evaluation of our approach, we rely on Intel DPC++ [11] SYCL compiler and the one provided by HPAC [9] for OpenMP. All SYCL and OpenMP codes have been compiled using the *-O3* flag. We performed our experiments on three separate nodes, each equipped with: an AMD EPYC 7313 CPU and AMD MI100 GPU; an Intel Xeon Platinum 8480 CPU and an Intel Max 1100 GPU; an Intel Xeon Gold 5218 and NVIDIA V100S.

7.2 Error Analysis

Figure 4 shows the MAPE for each benchmark executed with host perforation and input/output reconstruction on 100 different data sets. For `lavamd`, `hotspot`, and `leukocytes`, we only show the results for the column schema as they have been implemented using one-dimensional buffers and a one-dimensional perforation scheme that skips every other element, effectively corresponding to a column schema in two dimensions. The error analysis is not affected by the type of perforation applied (host or device), as both techniques perforate the same elements, leading to identical errors. For this reason, our analysis focuses only on *host perforation*. The results highlight that the amount of error depends on type of computation performed by the application, input data, perforation schema applied (row, col) and the type of reconstruction (input, output).

Computation. The `tv` and `sobel` benchmarks exhibit an error of up to 30% compared to the 1-15% error range of the other applications. This variation is due to the type of computation performed by each application. Applications based on average and median calculations (`gaussian`, `blur`, and `median`) are less affected by data perforation compared to the `sobel` and `tv` filters.

Input Data. The error introduced by data perforation is also correlated with the type of input. Applying perforation and reconstruction on inputs with higher data similarity produces a lower error. Looking at the column schema results for `leukocytes` and `guassian` the MAPE is in the range 1-15% while for `blur` 3-5%. The

variation in error is much clearer on the `sobel` and `tv` benchmark, where MAPE varies in the range 5-30%.

Perforation Schemes. The error is also affected by the type of schema used. As an example, the blur application shows an error between 3-5% for the row schema and 1-2% for the column schema.

Reconstruction. The input reconstruction usually achieves a lower error compared to the output reconstruction. With the input approach, the perforated data are reconstructed prior to computation. Consequently, the computation uses the same number of data as the accurate application, resulting in a lower error. For all applications, the input reconstruction has a lower or similar error compared to the output reconstruction. For instance, in the `leukocytes` and `sobel` application, output reconstruction leads to an error of 15% and 30%, while the input reconstruction keeps it below 5% and 20%.

To reduce the number of configuration in each plot, we only show results using a `skip_factor` of two. However, we conducted additional experiments to explore how the variation of the `skip_factor` affects both performance and error. As the `skip_factor` increases, performance improves approximately linearly-typically, a skip factor of X yields an X-fold speedup. In contrast, the error does not increase linearly, as it is influenced by multiple factors discussed above.

7.3 Host vs Device Perforation

Here, we focus on a performance comparison of the *host perforation* implemented in SYprox with the *device perforation* defined by Maier et al. [20]. Figure 4 illustrates the performance improvement in data transfer and kernel computation for each application compared to accurate execution (red dashed line) for both host (blue) and device (orange) perforation. The dashed and dotted hatch represent the row and column perforation schemes, respectively.

Host perforation consistently outperforms device perforation in terms of data transfer speedup, as shown in Figure 5a. This is because host perforation reduces data transferred to the device, achieving a speedup of 1.3x to 2x for floating-point data and 2.5x to 4.2x for half-precision data. In contrast, device perforation offers no such improvement, as all data are sent to the device.

Looking at the kernel speedup results in Figure 5b, host perforation outperforms device perforation in all applications, with the exception of `lavamd` and `hotspot`, where both methods achieve the same speedup. Host perforation speedups range from 1.7x to 4.2x, while device perforation range from 0.5x to 3.5x. The primary reason for the lower performance of device perforation is related to increased cache misses, which significantly impact its efficiency, in particular for the column schema. Profiling the applications with *NVIDIA Nsight Compute* we can notice an L2 cache hit rate of 76% for *host perforation* against the 49% of the device approach. In device perforation, the perforated data are still in memory, while accesses are performed in a perforated shape. Therefore, with a column schema, more than half of the data loaded into the cache are never used during computation, increasing the number of cache misses. In contrast, with *host perforation*, we achieve comparable performance for both schemes, since the data are reorganized in memory to avoid the load of unused data in the cache. For the column schema, this issue is highlighted in `blur`, `leukocytes`, and `gaussian`, which achieve speedups of 0.5x, 0.6x, and 0.75x, respectively, resulting in a slowdown compared to the accurate version, while host perforation reaches a speedup of up to 2x. For the row schema, `gaussian`, `tv`, and `blur` show a similar behaviour since each kernel thread processes a 6x6 filter, leading to the same access problem of the column schema. In contrast, for `sobel` and `median`, which use a smaller 3x3 filter, device perforation achieves similar performance compared to host perforation due to the lower number of cache misses. The slowdown related to cache misses is mitigated using the `half` data types, since with reduced precision, we can store more data in the cache, resulting in a lower number of cache misses.

7.4 SYprox vs HPAC

In this section, we conducted a multi-objective evaluation to analyze the trade-offs between speedup and error in the SYprox and HPAC frameworks. All SYprox benchmarks were ported from SYCL to OpenMP with GPU offloading and subsequently integrated into the HPAC framework. We tested HPAC applications with two loop perforation approaches: *small* skips one iteration for every n iterations; *large* executes one iteration for every n iterations. The `hotspot` results with HPAC are unavailable because using the perforation pragma defined by HPAC causes the application to run indefinitely. Figure 6 illustrates multi-objective plots for the 8 applications comparing the SYprox and HPAC frameworks. The x-axis represents the speedup, while the y-axis the Mean Absolute Percentage Error (MAPE). Points located in the bottom right corner of the plot are preferable as they indicate better performance with a lower error. For SYprox, the color of markers with different shades of red and blue represent the device and host perforation approaches, respectively, while the violet point represents the combination of host and device perforation. The different shades of a color indicate different reconstruction methods: nearest-neighbor or linear interpolation with input or output reconstruction (`nn_in`, `nn_out`, `lerp_in`, `lerp_out`). Markers are utilized to differentiate between combinations of perforation schemes (row, column) and data types (float, half). Within the HPAC setup, the blue markers denote configurations that employ loop perforation of type *'large'*, while the orange ones employ loop perforation of type *'small'*. The different markers in the HPAC framework correspond to different skip factors. The red and green lines represent the Pareto frontier of SYprox (P_S) and HPAC (P_H), respectively.

Multi-objective Analyses. For all benchmarks, the HPAC configurations show a higher error range compared to SYprox, due to the lack of reconstruction techniques. For the `gaussian` benchmark, it achieves up to 3x speedup at the cost of the 80% error, while for the other benchmarks, the speedup is between 1.5x and 2x with an error in the range of 10-60%. On the other hand, SYprox has multiple configurations involving mixed precision and perforation/reconstruction with different schemes. This diversity allows for a wide range of trade-offs, potentially making it more flexible in tuning performance versus accuracy. Most the SYprox configurations achieve an error less than 20% while achieving a nearly 4x speedup for all benchmarks. Furthermore, except for the `tv` benchmark, the Pareto frontier of SYprox always dominates the one of HPAC. This implies that for any given range of error or performance, SYprox offers

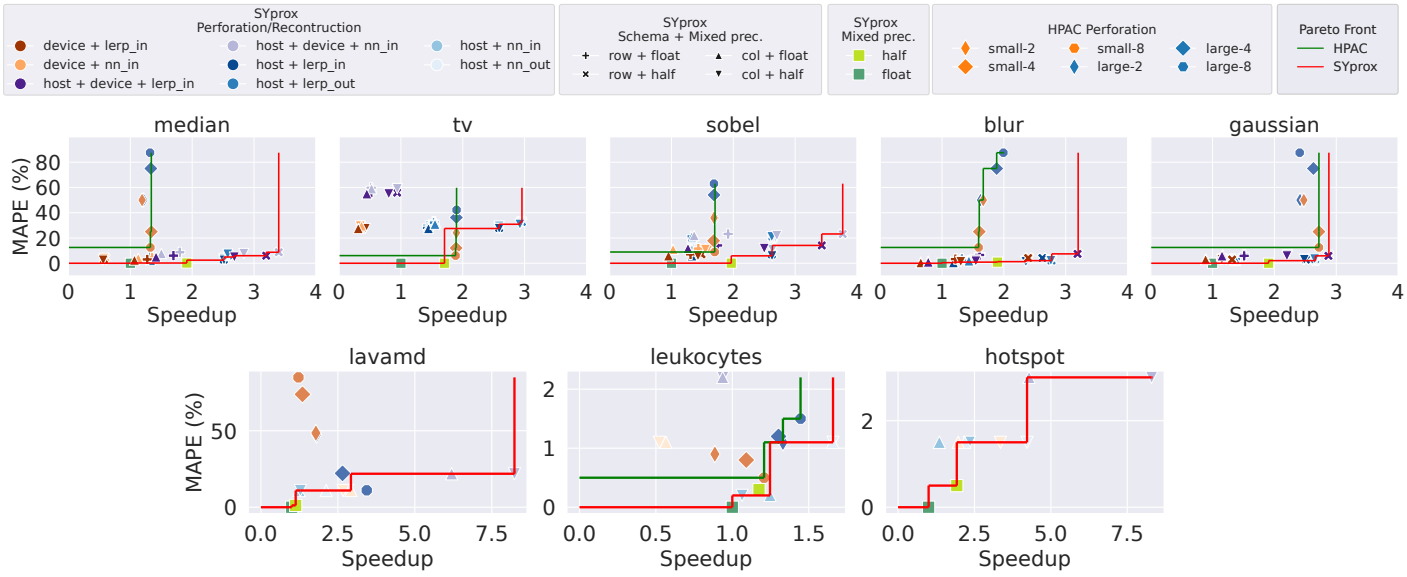


Figure 6: SYprox vs HPAC. The colors represent perforation and reconstruction techniques. Markers define the combination of schemes and data types. The green and red lines represent the HPAC and SYprox Pareto frontier.

configurations that are at least as good as, and often better than, those offered by HPAC.

Table 3: Evaluation of HPAC and SYprox Pareto fronts

Benchmark	$\|P_H\|$	$\|P_S\|$	HV_H	HV_S	$D(P, P_H)$
median	3	6	0.1	5.9	0
sobel	2	5	0	18.7	0
blur	3	7	3.3	5.7	0
tv	3	5	1.7	14.6	1.2
gaussian	3	5	2.2	3.8	0
leukocytes	3	3	1.4	1.8	0
lavaMD	1	5	3.05	6.8	0.07
hotspot	-	4	-	8.1	-

Hypervolume Analyses. Table 3 shows different metrics to compare the configuration of HPAC and SYprox. $|P_H|$ and $|P_S|$ represent the number of points in the Pareto set of HPAC and SYprox, respectively. For all benchmarks, we can notice $|P_S| >= |P_H|$ meaning that the SYprox framework generates more Pareto optimal solution compared to HPAC. In multi-objective optimization, the hypervolume metric [40] (HV) is used to evaluate the performance of a Pareto front by calculating the volume of the space in the objective domain that is dominated by the Pareto front, up to a reference point. A larger hypervolume indicates better performance, as it means that the Pareto front spans a larger region of the objective space, representing more optimal trade-offs between the objectives. In our case, we are interested in the coverage difference between two sets: the Pareto set P calculated considering the configuration of both SYprox and HPAC; and the Pareto set calculated only considering the SYprox configurations P_S. Therefore, we use the binary hypervolume metric [34], which is defined as $D(P, P_S) = HV(P) - HV(P_S)$, where $HV(P)$ and $HV(P_S)$ represents

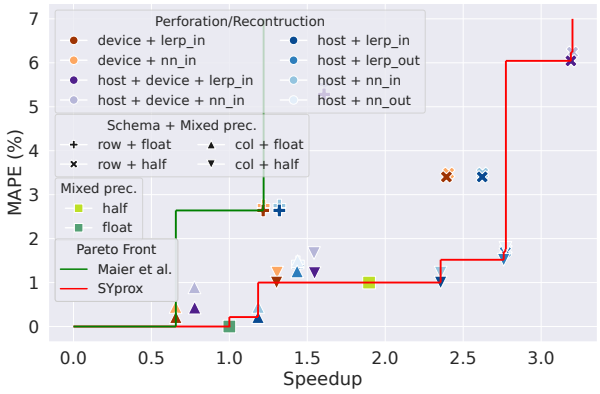


Figure 7: Domain space of the approximate computing techniques for Maier et al. and SYprox approach. Different colors represent combination of perforation and reconstruction. Markers distinguish the combination of perforation schemes and data types.

the hypervolume of the P and P_S Pareto frontier. In our experiment, the reference point for each benchmark has been selected according to the analysis provided by Ishibuchi et al.[13] in order to have accurate hypervolume results. Looking at the hypervolume values the SYprox Pareto front always covers a larger region of the objective space compared to HPAC. This observation indicates that the SYprox approximations are distributed along the speedup and error axes in a way that encompasses wider levels of performance and accuracy. The coverage difference ($D(P, P_H)$) shows that for all the applications the Pareto frontier of SYprox dominates the one

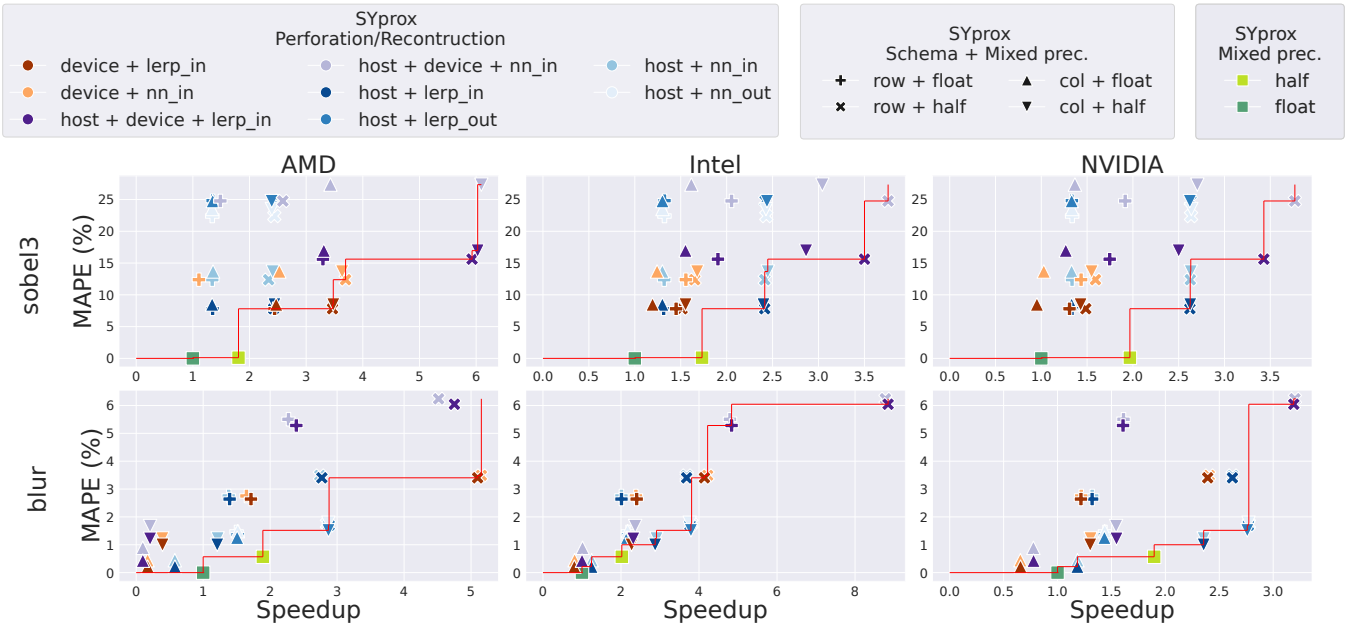


Figure 8: Performance evaluation of SYprox on AMD, Intel and NVIDIA hardware. The color represents different perforation and reconstruction techniques. Markers are used to distinguish the combination of perforation schemes and data types.

of HPAC. The only exception is the `tv` and `lavamd` benchmarks, where the coverage difference is non-zero because there are two and one points, respectively, in P_H that have no counterparts in the SYprox Pareto front that outperform them in both dimensions.

7.5 Approximation Space Evaluation

Figure 7 shows the speedup and error of SYprox in comparison to the Maier et al. approach, highlighting several key advantages of our method. *Enhanced Coverage of the Objective Space.* One of the primary benefits of SYprox lies in its ability to generate a substantially larger number of configurations compared to the approach proposed by Maier et al. This advantage is illustrated in Figure 7, where the SYprox configurations span a wider section of the objective space. This expanded coverage allows an exploration of a wider spectrum of speedup and error trade-offs, thereby facilitating more precise optimization tailored to diverse application requirements. The SYprox configurations extend across both axes, indicating a versatile approach capable of addressing various performance and error needs. Additionally, the approximation domain can be further broadened by adjusting the skip factor parameter, thereby generating new potential Pareto-optimal solutions.

Discovery of New Pareto Optimal Solutions. By combining different approximation techniques, SYprox not only expands the configuration space, but also identifies new Pareto optimal solutions that were previously unattainable with existing methods. SYprox's advantages are evident when considering device perforation with the column schema. In this case, the column schema can cause cache misses due to the data layout, leading to a performance slowdown of up to two times. By applying host perforation instead of device perforation, we mitigate the cache misses generated by the column data layout. This adjustment allows the configuration with host perforation and the column schema to perform comparably to the row schema and even become a Pareto optimal solution. This demonstrates how SYprox can overcome specific performance bottlenecks and optimize configurations that were previously suboptimal. This capability is reflected in the red Pareto front associated with SYprox, which dominates the green Pareto front of Maier et al., showing that our approach consistently outperforms the prior state-of-the-art.

Combining Approximation. SYprox demonstrates significant performance improvements by combining various approximation techniques. Combining mixed precision with the other approximation often generates new configurations that are Pareto optimal, since reducing precision in most cases introduces a small error with up to 2x speedup. Furthermore, by combining host and device perforation with mixed precision, we achieve up to 3.5x speedup with only a 7% error. This combination is particularly effective because it takes advantage of the strengths of all the approximations.

7.6 Performance and Accuracy Portability

Figure 8 demonstrates the portability of our approach across different hardware architectures, including AMD MI100, Intel Max 1100 and NVIDIA V100S GPUs. A key observation is that the error remains consistent across all three hardware platforms. This consistency highlights that the approximate computing techniques implemented in SYprox are predominantly data-dependent rather than hardware-dependent. The only notable exception is half-precision, which introduces slight variations due to differences in hardware precision handling. However, these variations are minimal and do not significantly affect the overall error profile. When analyzing performance, we notice some variability between hardware

platforms. All platforms achieve speedups of more than 3x, with some hardware yielding even better performance using the same approximation.

8 Conclusion

This paper presents SYprox, a novel approach for heterogeneous approximate computing. SYprox makes three major breakthroughs: a SYCL-based interface that extends SYCL buffer and accessor with approximate computing capabilites; a new data perforation approach that allows to fully exploit the host-device execution model; a way to combine data perforation, reconstruction, and mixed process expanding the approximate space with new Pareto-optimal configurations. We have experimentally assessed the SYprox methodology on AMD MI100, Intel Max 1100, and NVIDIA V100, comparing it with state-of-the-art frameworks. The results highlighted the advantages of host perforation, which consistently outperforms device perforation by reducing both kernel computation and data transfer times. Moreover, the ability of SYprox to combine multiple approximation techniques provides a rich set of Pareto optimal solutions that outperform prior methods, such as those proposed by Maier et al. and the HPAC framework. SYprox not only discovers more Pareto optimal solutions, but also expands the coverage of the objective space, offering greater flexibility for tuning trade-offs between speedup and error. Finally, our approach demonstrated robust performance and accuracy across different GPUs, validating the SYprox performance and accuracy portability.

Acknowledgments

We acknowledge financial support under the National Recovery and Resilience Plan (NRPP), call for tender No. 104 published on 02/02/2022 by the Italian Ministry of University and Research (MUR), funded by the European Union - Next Generation EU, Mission 4, Component 1, CUP D53D23008590001, project title LibreRT.

We thank CINECA for providing access to the Intel Max 1100 GPU.

References

[1] Woongki Baek and Trishul M Chilimbi. 2010. Green: A framework for supporting energy-conscious programming using controlled approximation. In *Proceedings of the 31st ACM SIGPLAN Conference on Programming Language Design and Implementation*.

[2] Hrishav Bakul Barua and Kartick Chandra Mondal. 2019. Approximate computing: A survey of recent trends—bringing greenness to computing and communication. *Journal of The Institution of Engineers (India): Series B* (2019).

[3] Simone Campanoni, Glenn Holloway, Gu-Yeon Wei, and David Brooks. 2015. HELIX-UP: Relaxing program semantics to unleash parallelization. In *2015 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*.

[4] Stefano Cherubin and Giovanni Agosta. 2020. Tools for reduced precision computation: a survey. *ACM Computing Surveys (CSUR)* (2020).

[5] Stefano Cherubin, Daniele Cattaneo, Michele Chiari, Antonio Di Bello, and Giovanni Agosta. 2020. TAFFO: Tuning Assistant for Floating to Fixed Point Optimization. *IEEE Embedded Systems Letters* (2020).

[6] Eva Darulova and Viktor Kuncak. 2017. Towards a compiler for reals. *ACM Transactions on Programming Languages and Systems (TOPLAS)* (2017).

[7] Kalyanmoy Deb, Karthik Sindhya, and Jussi Hakanen. 2016. Multi-objective optimization. In *Decision sciences*.

[8] Mikhail Figurnov, Aizhan Ibraimova, Dmitry P Vetrov, and Pushmeet Kohli. 2016. Perforatedcnns: Acceleration through elimination of redundant convolutions. *Advances in neural information processing systems* (2016).

[9] Zane Fink, Konstantinos Parasyris, Giorgis Georgakoudis, and Harshitha Menon. 2023. HPAC-Offload: Accelerating HPC Applications with Portable Approximate Computing on the GPU. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*.

[10] Henry Hoffmann, Sasa Misailovic, Stelios Sidiroglou, Anant Agarwal, and Martin Rinard. 2009. Using code perforation to improve performance, reduce energy consumption, and respond to failures. (2009).

[11] Intel. 2022. oneAPI Data Parallel C++ compiler. https://github.com/intel/llvm/releases/tag/2022-09

[12] Intel Corporation. 2024. *SYCL EXT ONEAPI Bfloat16 Math Functions*. https://github.com/intel/llvm/blob/sycl/sycl/doc/extensions/experimental/sycl_ext_oneapi_bfloat16_math_functions.asciidoc Accessed: 2024-09-12.

[13] Hisao Ishibuchi, Ryo Imada, Yu Setoguchi, and Yusuke Nojima. 2017. Reference point specification in hypervolume calculation for fair comparison and efficient search. In *Proceedings of the genetic and evolutionary computation conference*.

[14] Maria Kotsifakou, Prakalp Srivastava, Matthew D Sinclair, Rakesh Komuravelli, Vikram Adve, and Sarita Adve. 2018. Hpvm: Heterogeneous parallel virtual machine. In *Proceedings of the 23rd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*.

[15] Ignacio Laguna, Paul C Wood, Ranvijay Singh, and Saurabh Bagchi. 2019. Gpumixer: Performance-driven floating-point tuning for gpu scientific applications. In *High Performance Computing: 34th International Conference, ISC High Performance 2019, Frankfurt/Main, Germany, June 16–20, 2019, Proceedings 34*.

[16] Michael O. Lam, Jeffrey K. Hollingsworth, Bronis R. de Supinski, and Matthew P. Legendre. 2013. Automatically Adapting Programs for Mixed-Precision Floating-Point Computation. In *Proceedings of the 27th International ACM Conference on International Conference on Supercomputing*.

[17] Kooktae Lee and Raktim Bhattacharya. 2016. On the relaxed synchronization for massively parallel numerical algorithms. In *2016 American Control Conference (ACC)*.

[18] Shikai Li, Sunghyun Park, and Scott Mahlke. 2018. Sculptor: Flexible approximation with selective dynamic loop perforation. In *Proceedings of the 2018 International Conference on Supercomputing*.

[19] Liming Lou, Paul Nguyen, Jason Lawrence, and Connelly Barnes. 2016. Image perforation: Automatically accelerating image pipelines by intelligently skipping samples. *ACM Transactions on Graphics (TOG)* (2016).

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

[21] Daniel Maier and Ben Juurlink. 2021. Model-Based Loop Perforation. In *European Conference on Parallel Processing*.

[22] Daniel Maier, Nadjib Mammeri, Biagio Cosenza, and Ben Juurlink. 2019. Approximating memory-bound applications on mobile GPUs. In *2019 International Conference on High Performance Computing & Simulation (HPCS)*.

[23] Subrata Mitra, Manish K Gupta, Sasa Misailovic, and Saurabh Bagchi. 2017. Phase-aware optimization in approximate computing. In *2017 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*.

[24] Sparsh Mittal. 2016. A survey of techniques for approximate computing. *ACM Computing Surveys (CSUR)* (2016).

[25] Konstantinos Parasyris, Giorgis Georgakoudis, Harshitha Menon, James Diffenderfer, Ignacio Laguna, Daniel Osei-Kuffuor, and Markus Schordan. 2021. HPAC: evaluating approximate computing techniques on HPC OpenMP applications. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*.

[26] Lakshminarayanan Renganarayana, Vijayalakshmi Srinivasan, Ravi Nair, and Daniel Prener. 2012. Programming with relaxed synchronization. In *Proceedings of the 2012 ACM workshop on Relaxing synchronization for multicore and manycore scalability*.

[27] Martin Rinard, Henry Hoffmann, Sasa Misailovic, and Stelios Sidiroglou. 2010. Patterns and Statistical Analysis for Understanding Reduced Resource Computing. *ACM Sigplan Notices* (2010).

[28] Cindy Rubio-González, Cuong Nguyen, Hong Diep Nguyen, James Demmel, William Kahan, Koushik Sen, David H. Bailey, Costin Iancu, and David Hough. 2013. Precimonious: Tuning Assistant for Floating-Point Precision. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*.

[29] Mehrzad Samadi, Davoud Anoushe Jamshidi, Janghaeng Lee, and Scott Mahlke. 2014. Paraprox: Pattern-based approximation for data parallel applications. In *Proceedings of the 19th international conference on Architectural support for programming languages and operating systems*.

[30] Mehrzad Samadi, Davoud Anoushe Jamshidi, Janghaeng Lee, and Scott Mahlke. 2014. Paraprox: Pattern-based approximation for data parallel applications. In *Proceedings of the 19th international conference on Architectural support for programming languages and operating systems*.

[31] Mehrzad Samadi, Janghaeng Lee, D Anoushe Jamshidi, Amir Hormati, and Scott Mahlke. 2013. Sage: Self-tuning approximation for graphics engines. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*.

[32] Adrian Sampson, André Baixo, Benjamin Ransford, Thierry Moreau, Joshua Yip, Luis Ceze, and Mark Oskin. 2015. Accept: A programmer-guided compiler framework for practical approximate computing. *University of Washington Technical Report UW-CSE-15-01* (2015).

[33] Adrian Sampson, Werner Dietl, Emily Fortuna, Danushen Gnanapragasam, Luis Ceze, and Dan Grossman. 2011. EnerJ: Approximate Data Types for Safe and General Low-Power Computation. In *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation*.

[34] Ke Shang, Hisao Ishibuchi, Linjun He, and Lie Meng Pang. 2020. A survey on the hypervolume indicator in evolutionary multiobjective optimization. *IEEE Transactions on Evolutionary Computation* (2020).

[35] Hashim Sharif, Yifan Zhao, Maria Kotsifakou, Akash Kothari, Ben Schreiber, Elizabeth Wang, Yasmin Sarita, Nathan Zhao, Keyur Joshi, Vikram S Adve, et al. 2021. ApproxTuner: a compiler and runtime system for adaptive approximations. In *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*.

[36] Stelios Sidiroglou-Douskos, Sasa Misailovic, Henry Hoffmann, and Martin Rinard. 2011. Managing performance vs. accuracy trade-offs with loop perforation. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*.

[37] Vassilis Vassiliadis, Charalampos Chalios, Konstantinos Parasyris, Christos D Antonopoulos, Spyros Lalis, Nikolaos Bellas, Hans Vandierendonck, and Dimitrios S Nikolopoulos. 2016. Exploiting significance of computations for energy-constrained approximate computing. *International Journal of Parallel Programming* (2016).

[38] Vassilis Vassiliadis, Konstantinos Parasyris, Charalambos Chalios, Christos D Antonopoulos, Spyros Lalis, Nikolaos Bellas, Hans Vandierendonck, and Dimitrios S Nikolopoulos. 2015. A programming model and runtime system for significance-aware energy-efficient computing. *ACM SIGPLAN Notices* (2015).

[39] Allan G. Weber. 2006. The USC-SIPI Image Database. http://sipi.usc.edu/database/database.php. Accessed: August 2018.

[40] Eckart Zitzler, Lothar Thiele, Marco Laumanns, Carlos M Fonseca, and Viviane Grunert Da Fonseca. 2003. Performance assessment of multiobjective optimizers: An analysis and review. *IEEE Transactions on evolutionary computation* (2003).