Integrating FPGA Acceleration to OpenMP Distributed Computing

Pedro Henrique Di Francia Rosso <pedro.rosso@ic.unicamp.br>

Guido Araújo <guido@unicamp.br>

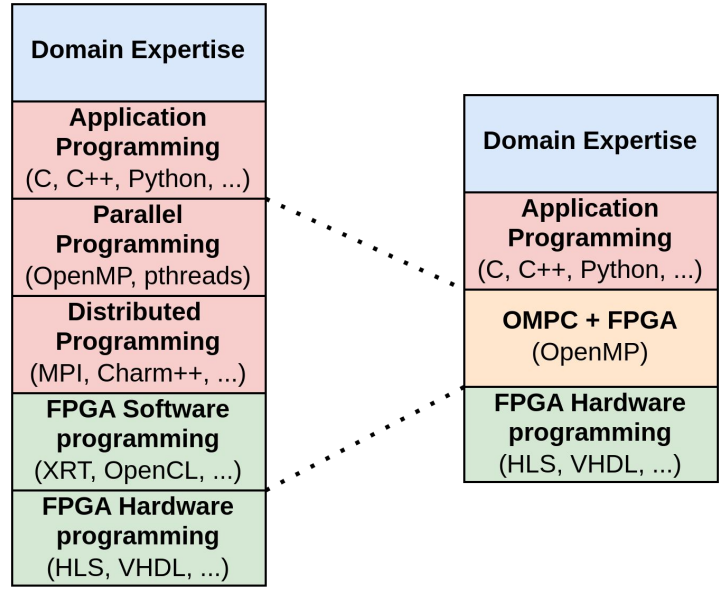
Universidade Estadual de Campinas (UNICAMP) - Brazil

March 08th 2024

Table of Contents

Introduction

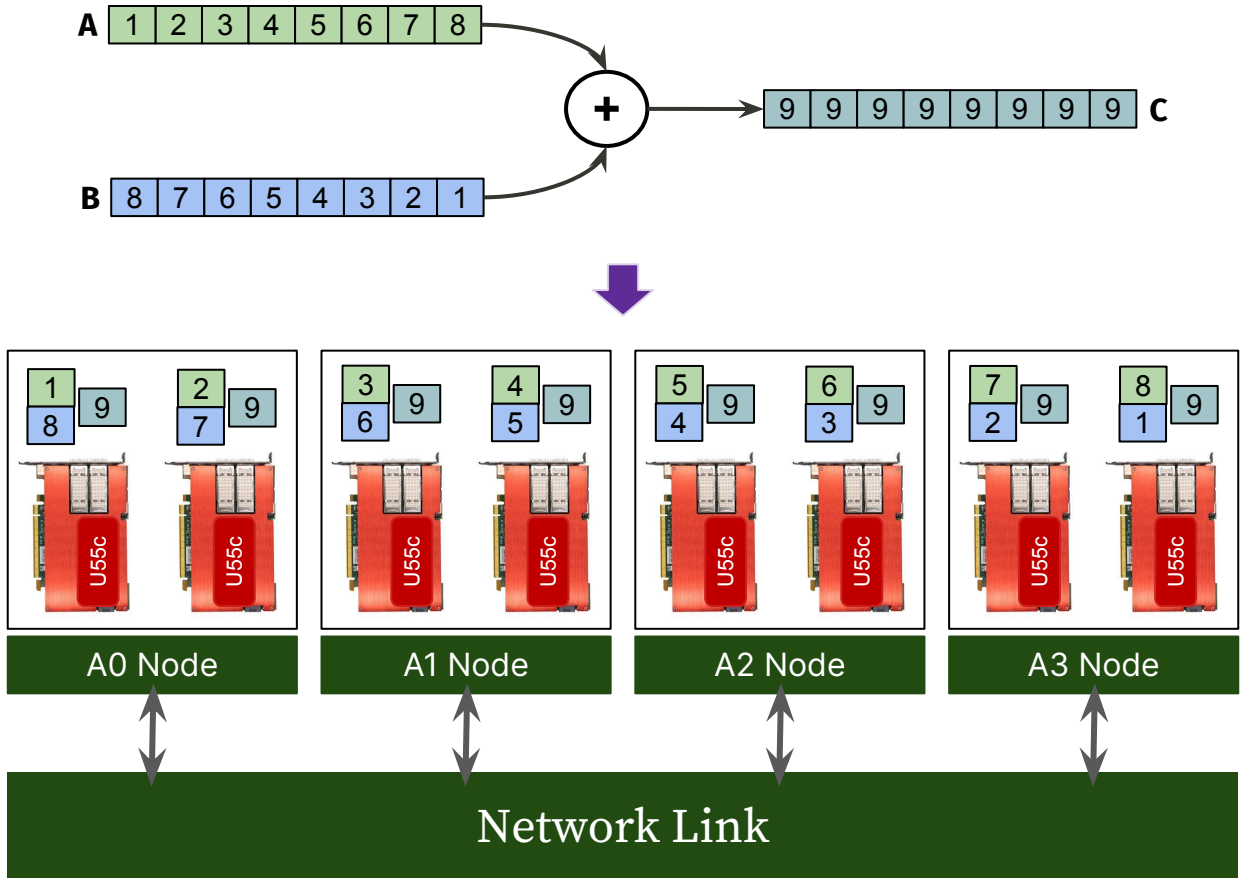
- Accelerators become a paramount for HPC nowadays
- FPGA programming can require a lot of knowledge in different aspects, which is not practical
- Many aspects can be abstracted when we extend OMPC for FPGA offloading
- The resulting stack is a flexible version of the regular stack that separates hardware and software development (being more flexible)
- Capable of handling FPGA execution and communication between FPGAs transparently



(a) FPGA Regular Stack

(b) Reduced Stack

Motivation


A 1 2 3 4 5 6 7 8
+
9 9 9 9 9 9 9 9 C
B 8 7 6 5 4 3 2 1
1 8 9
2 7 9
3 6 9
4 5 9
5 4 9
6 3 9
7 2 9
8 1 9
U55c
A0 Node
A1 Node
A2 Node
A3 Node
Network Link

Motivation

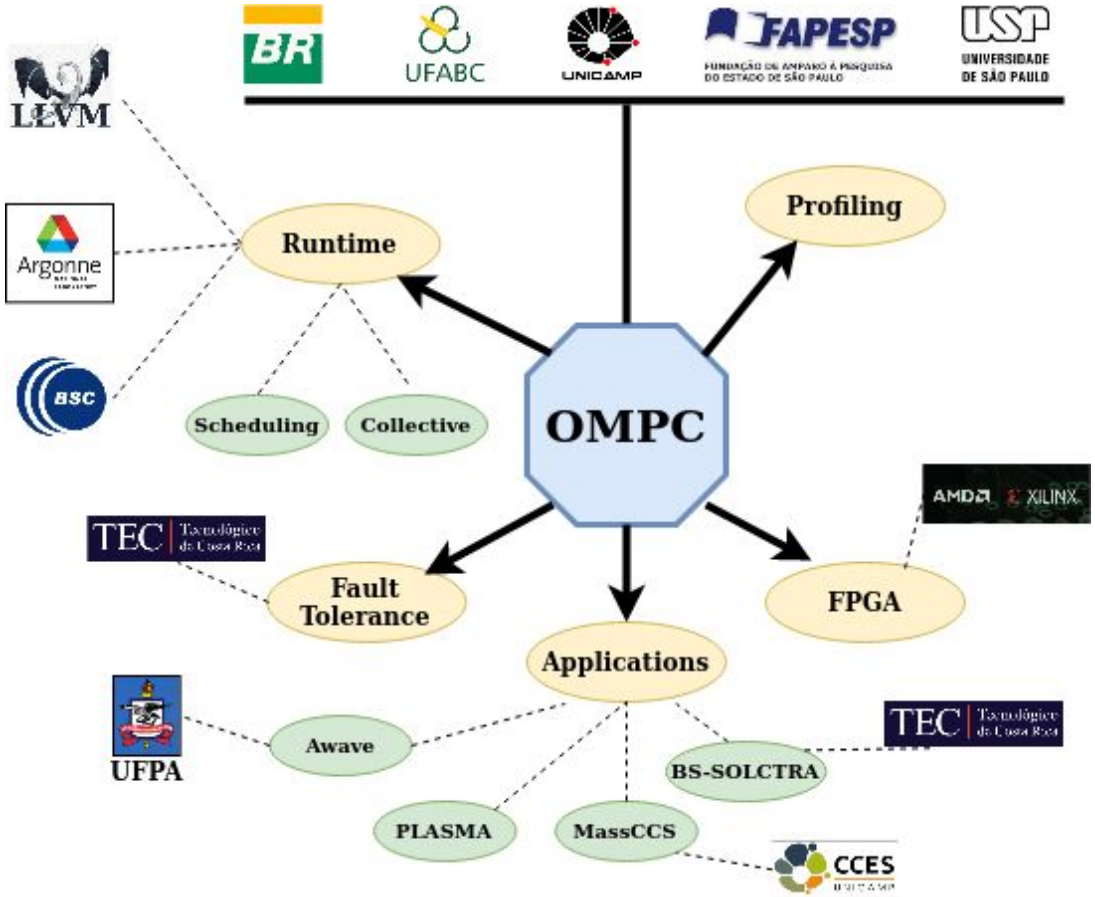
Halstead Metrics

Metric	Baseline	OMPC	OpenMP MPI XRT
Program Length (N)	405	507	911
Vocabulary Size (n)	40	66	93
Program Volume (V)	2155,38	3064,51	5957,17
Difficulty level (D)	84,14	66,06	93,27
Program Level (L)	0,01	0,02	0,01
Effort to Implement (E)	181359,91	202435,69	555641,84
Time to Implement (T)	10075,55	11246,43	30868,99
Delivered Bugs (B)	1,07	1,15	2,25

HALSTEAD, Maurice H. **Elements of Software Science (Operating and programming systems series)**. Elsevier Science Inc., 1977.

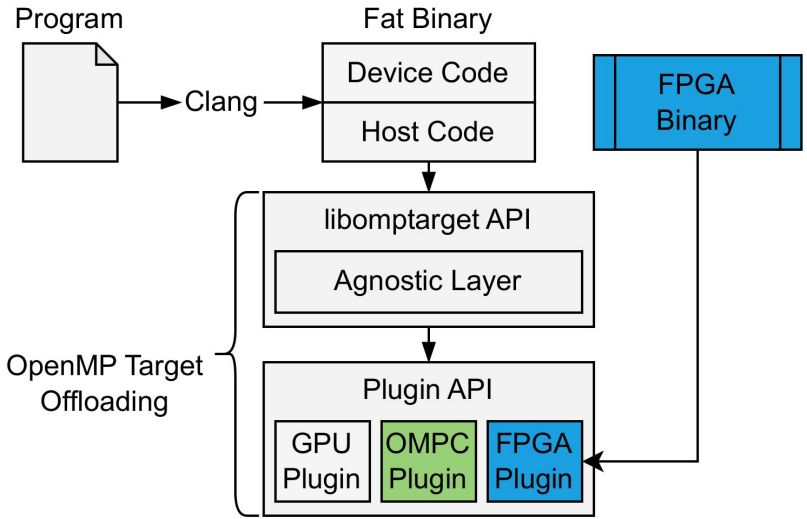
OpenMP Cluster (OMPC)

The OpenMP Cluster (OMPC)

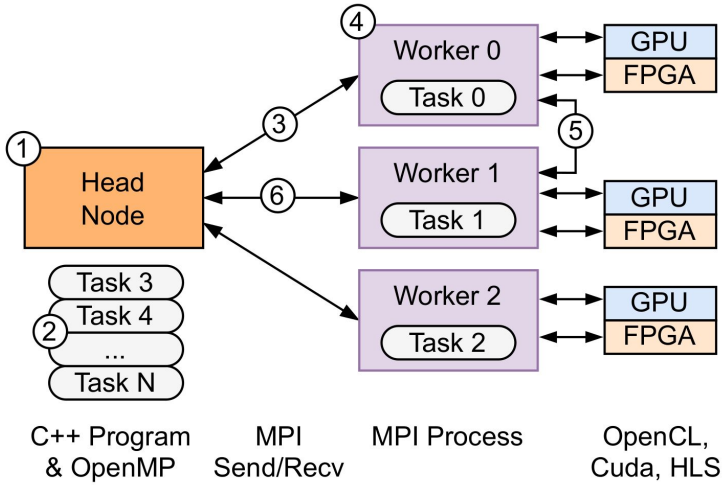


The OpenMP Cluster (OMPC)

Architecture



Execution Flow



Yviquel, Hervé, et al. **"The OpenMP Cluster Programming Model."** *Workshop Proceedings of the 51st International Conference on Parallel Processing*. 2022.

```
void array_sum(int *A, int *B, int *C, int N) {
 for(int i = 0; i < N; i++)
   C[i] = A[i] + B[i];
}

void blocked_sum(int *A, int *B, int *C, int N, int BS) {
  for(int i = 0; i < N/BS; i++) {
    int *BA = &A[i*BS];
    int *BB = &B[i*BS];
    int *BC = &C[i*BS];
    #pragma omp target map(to: BA[:N], BB[:N])      \
                       map(from: BC[:N])            \
                       depend(in: BA[0], BB[0])     \
                       depend(out: BC[0]) nowait
    array_sum(BA, BB, BC, N);
  }
  #pragma omp taskwait
}
```

OpenMP Target Example

Target Directives

- target
- target data
- target enter data
- target exit data
- declare target

Target directives are executed on the available devices (compiled for)

FPGA Extension

Enabling FPGA kernels on OpenMP

- Two steps to enable target directives to offload computation to FPGAs

a. Inclusion of ***declare variant*** directive in the application

```
void array_sum_hw(int *A, nt *B, nt *C, size_t size); // FPGA Prototype
#pragma omp declare variant(array_sum_hw) match(device = {arch(alveo)})
void array_sum(int *A, nt *B, nt *C, size_t size); // CPU Prototype
```

b. Compile using the ***alveo*** target

```
$ clang++ -fopenmp -fopenmp-targets=alveo vadd.cpp -o vadd
```

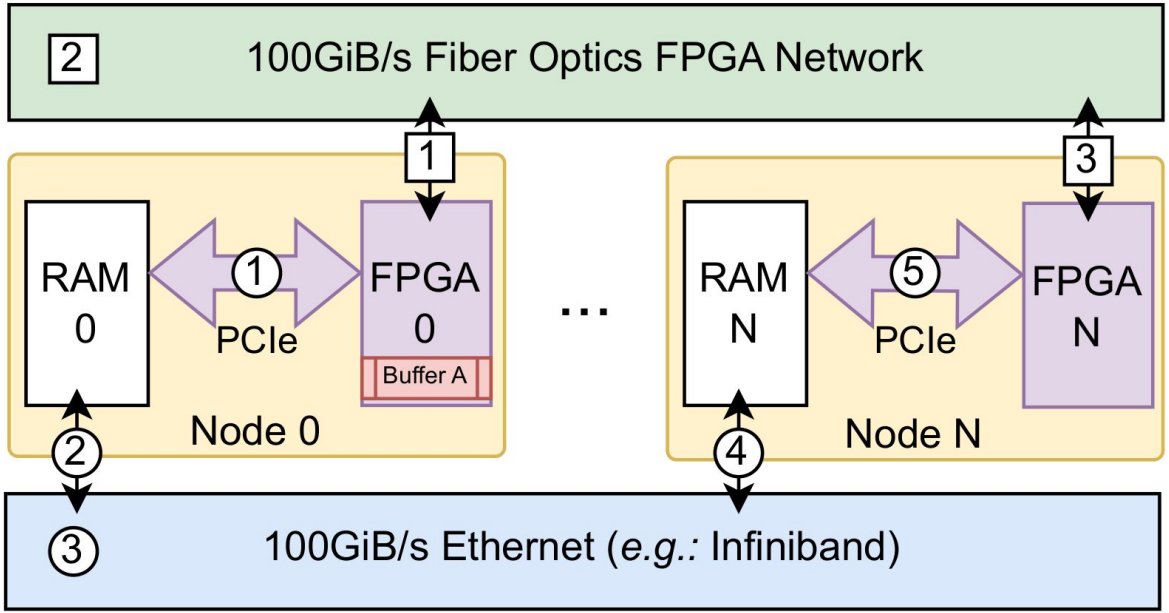
- For every call of ***array_sum*** in the application, ***array_sum_hw*** on FPGA will be used instead

```
void array_sum_hw(int *A, int *B, int *C, int N);
#pragma omp declare variant(array_sum_hw) match(device = {arch(alveo)})
void array_sum(int *A, int *B, int *C, int N) {
 for(int i = 0; i < N; i++)
   C[i] = A[i] + B[i];
}

void blocked_sum(int *A, int *B, int *C, int N, int BS) {
  for(int i = 0; i < N/BS; i++) {
    int *BA = &A[i*BS];
    int *BB = &B[i*BS];
    int *BC = &C[i*BS];
    #pragma omp target map(to: BA[:N], BB[:N])      \
                       map(from: BC[:N])            \
                       depend(in: BA[0], BB[0])     \
                       depend(out: BC[0]) nowait
    array_sum(BA, BB, BC, N);
  }
  #pragma omp taskwait
}
```

The FPGA Extension

- OpenMP ***declare variant*** directive
- Offloads computation to FPGA kernels
- Leverages the Alveo Collective Communication Library (ACCL) for communication directly between FPGAs.
- Heterogeneous execution

Accelerating Communication

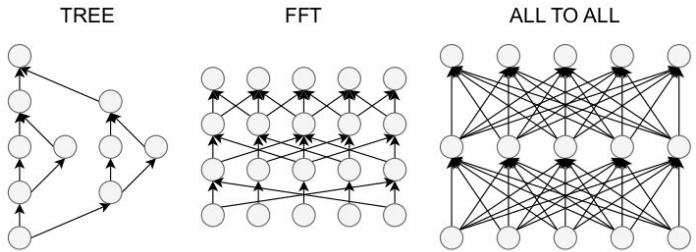


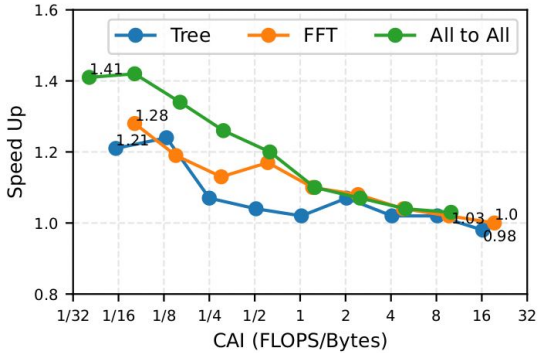
He, Zhenhao, et al. **"ACCL: FPGA-accelerated collectives over 100 Gbps TCP-IP."** *2021 IEEE/ACM International Workshop on Heterogeneous High-performance Reconfigurable Computing (H2RC)*. IEEE, 2021.

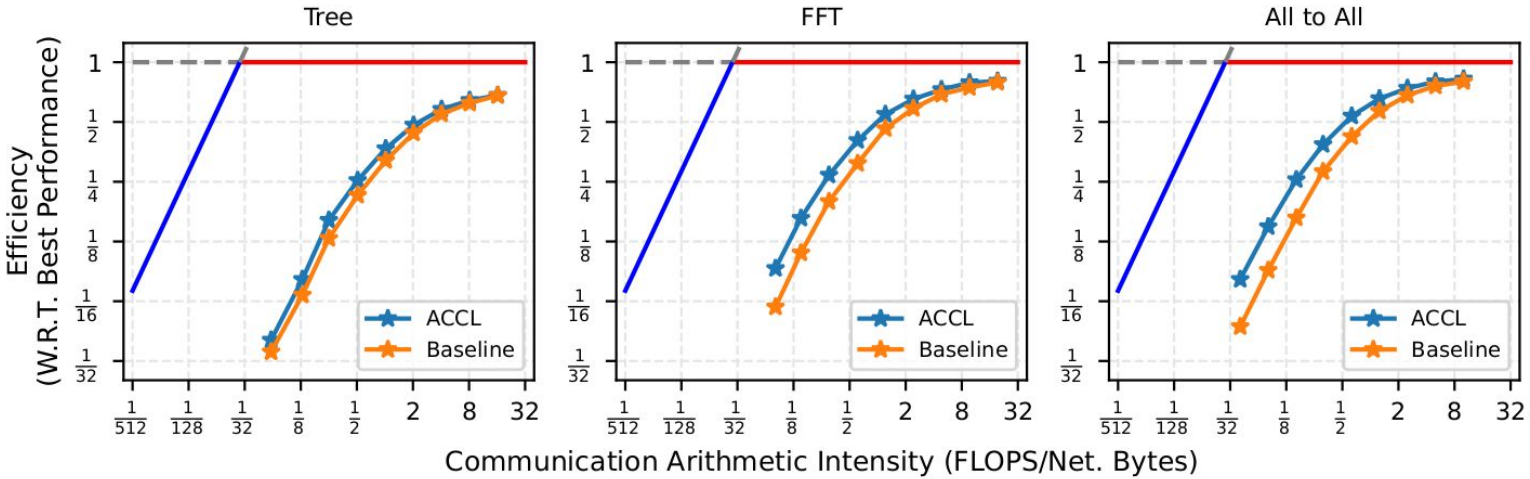
Accelerating Communication Results

- ETH Zurich Cluster - (Part of AMD HACC Clusters)
- 9x Nodes with FPGA - (AMD u55c)
- 100 Gbps Ethernet / 100 Gbps FPGA Network
- 4 MB Buffers

Baseline: Comm. Through MPI

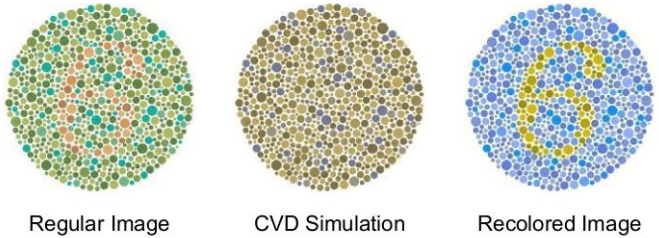


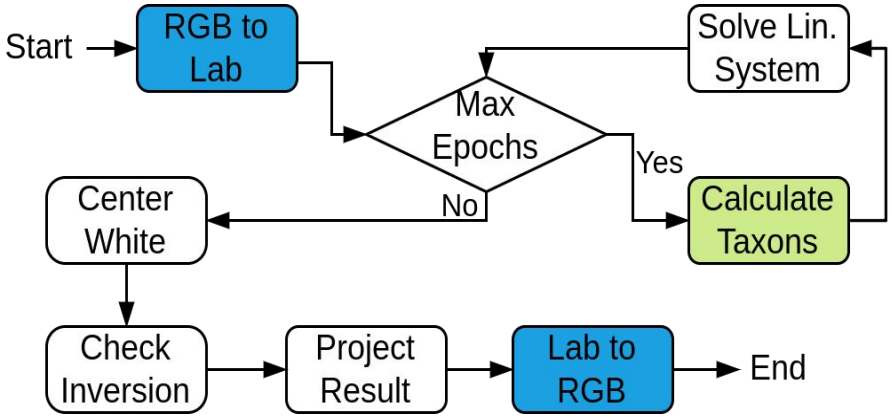




Exploring Heterogeneity

Color Vision Deficiency (CVD) Image Recoloring Algorithm





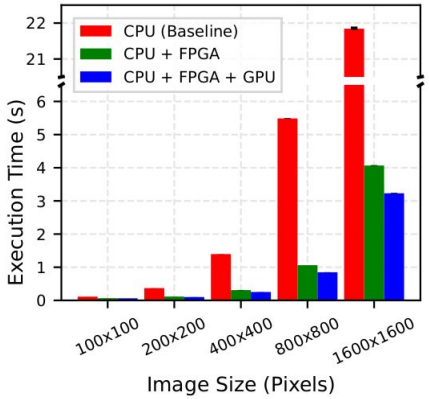
Using ***declare variant*** to set GPU and FPGA Kernels

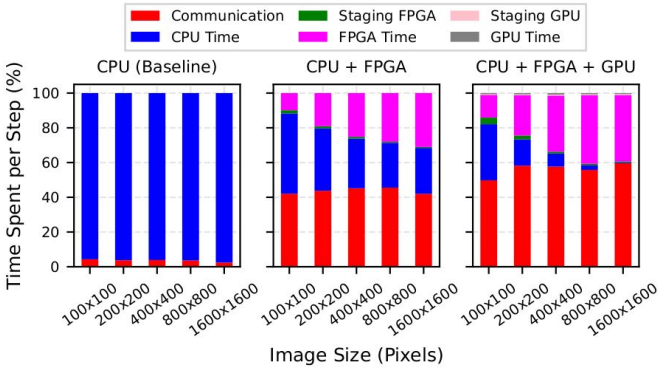
```
#pragma omp declare variant (calctaxons_hw) match (device = {arch(alveo)})
```

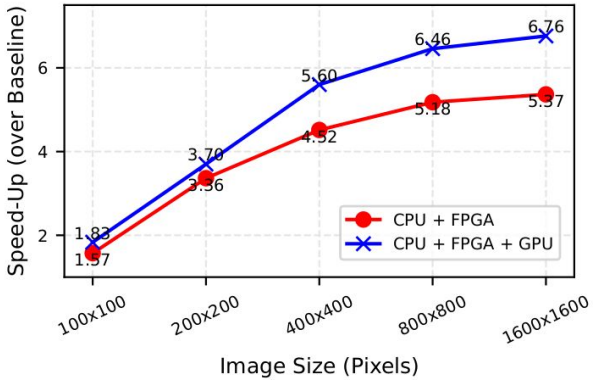
```
#pragma omp declare variant (rgb_lab_gpu) match (user = {condition(ENABLE_CUDA)})
```

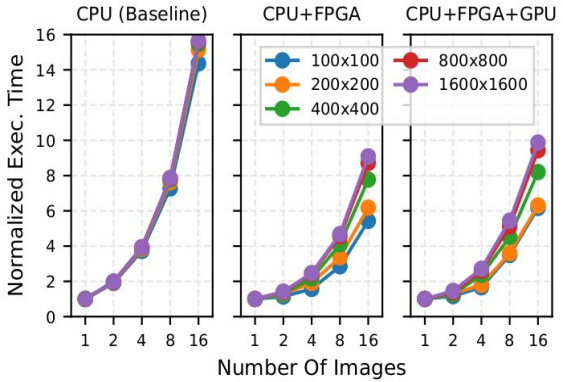
Exploring Heterogeneity Results

- 1x Node with FPGA - (AMD u55c)
- 1x Node with GPU - (NVidia Tesla P100)
- 1x Node with CPU - (Intel Xeon Silver 4108)
- 1 Gbps Ethernet Connection









Conclusions

Contributions

- Reduced amount of time spent to integrate FPGA kernels into application.
- Fully abstracted use of FPGAs in distributed environments.
- Automatically communicating directly through the FPGAs rather than relying on the default mechanism.
- Easy validation mechanism by switching between CPU and FPGA kernels during the application compilation.

Limitations

- Lack of support for FPGA data streaming
- Scheduling (currently uses a slightly modified Round-Robin)
- Blocking Communication between FPGAs

Future Work

- Optimization path: scheduling, FPGA capabilities (e.g.: data streaming), further acceleration in communication (e.g.: non-blocking)
- Exploring domain applications (e.g.: Physics, Machine Learning, etc.)
- Documentation, containerization

Thank You!

Pedro Henrique Rosso <pedro.rosso@ic.unicamp.br>

Guido Araújo <guido@unicamp.br>

This research is funded by FAPESP grant 2021/09355-2

OpenMP Blocked Vadd


```
#include <iostream>

void vadd(unsigned int *A, unsigned int *B, unsigned int *C, size_t size) {
#pragma omp parallel for
 for (int i = 0; i < size; i++)
   C[i] = A[i] + B[i];
}

void init_array(unsigned int *V, size_t size, unsigned int value) {
#pragma omp parallel for
 for (int i = 0; i < size; i++)
   V[i] = value;
}
```

```
int main(int argc, char *argv[]) {
 size_t size = static_cast<size_t>(std::stoi(argv[1]));
 size_t n_blocks = static_cast<size_t>(std::stoi(argv[2]));
 size_t b_size = size / n_blocks;

 unsigned int *A = new unsigned int[size];
 unsigned int *B = new unsigned int[size];
 unsigned int *C = new unsigned int[size];

 init_array(A, size, 1);
 init_array(B, size, 2);
 init_array(C, size, 0);

 #pragma omp parallel for
 for (int i = 0; i < n_blocks; i++)
   vadd(&A[i * b_size], &B[i * b_size], &C[i * b_size], b_size);

 delete[] A;
 delete[] B;
 delete[] C;

 return 0;
}
```

OpenMP + MPI + XRT Vadd


```cpp
#include "xrt/xrt_bo.h"
#include "xrt/xrt_device.h"
#include "xrt/xrt_kernel.h"
#include <cstdlib>
#include <experimental/xrt_xclbin.h>
#include <iostream>
#include <mpi.h>

void init_array(unsigned int *V, size_t size, unsigned int value) {
#pragma omp parallel for
 for (int i = 0; i < size; i++)
   V[i] = value;
}

int main(int argc, char *argv[]) {
 MPI_Init(NULL, NULL);

 int mpi_rank, mpi_size;
 MPI_Comm_size(MPI_COMM_WORLD, &mpi_size);
 MPI_Comm_rank(MPI_COMM_WORLD, &mpi_rank);
 MPI_Barrier(MPI_COMM_WORLD);

 int device_id = mpi_rank / 3;
 xrt::device dev(device_id);
 auto file_info = xrt::xclbin(argv[3]);
 auto uuid = dev.load_xclbin(file_info);
 xrt::kernel vadd_krnl(dev, uuid.get(), "vadd_hw");
 xrt::run vadd_run(vadd_krnl);

 size_t size = static_cast<size_t>(std::stoi(argv[1]));
 size_t n_blocks = static_cast<size_t>(std::stoi(argv[2]));
 size_t b_size = size / n_blocks;

 unsigned int *A = (unsigned int *)aligned_alloc(4096, (mpi_rank == 0 ? size : b_size) * sizeof(unsigned int));
 unsigned int *B = (unsigned int *)aligned_alloc(4096, (mpi_rank == 0 ? size : b_size) * sizeof(unsigned int));
 unsigned int *C = (unsigned int *)aligned_alloc(4096, (mpi_rank == 0 ? size : b_size) * sizeof(unsigned int));

 if (mpi_rank == 0) {
   init_array(A, size, 1);
   init_array(B, size, 2);
   init_array(C, size, 0);
 }

 MPI_Scatter(A, b_size, MPI_UNSIGNED, mpi_rank == 0 ? MPI_IN_PLACE : A, b_size, MPI_UNSIGNED, 0, MPI_COMM_WORLD);
 MPI_Scatter(B, b_size, MPI_UNSIGNED, mpi_rank == 0 ? MPI_IN_PLACE : B, b_size, MPI_UNSIGNED, 0, MPI_COMM_WORLD);

 xrt::bo A_bo = xrt::bo(dev, A, b_size * sizeof(unsigned int), xrt::bo::flags::normal, vadd_krnl.group_id(0));
 xrt::bo B_bo = xrt::bo(dev, B, b_size * sizeof(unsigned int), xrt::bo::flags::normal, vadd_krnl.group_id(1));
 xrt::bo C_bo = xrt::bo(dev, C, b_size * sizeof(unsigned int), xrt::bo::flags::normal, vadd_krnl.group_id(2));

 A_bo.sync(XCL_BO_SYNC_BO_TO_DEVICE);
 B_bo.sync(XCL_BO_SYNC_BO_TO_DEVICE);

 vadd_run.set_arg(0, A_bo);
 vadd_run.set_arg(1, B_bo);
 vadd_run.set_arg(2, C_bo);
 vadd_run.set_arg(3, b_size);
 vadd_run.start();
 vadd_run.wait();

 C_bo.sync(XCL_BO_SYNC_BO_FROM_DEVICE);

 MPI_Gather(mpi_rank == 0 ? MPI_IN_PLACE : C, b_size, MPI_UNSIGNED, C, b_size, MPI_UNSIGNED, 0, MPI_COMM_WORLD);

 free(A);
 free(B);
 free(C);

 MPI_Finalize();
 return 0;
}
```

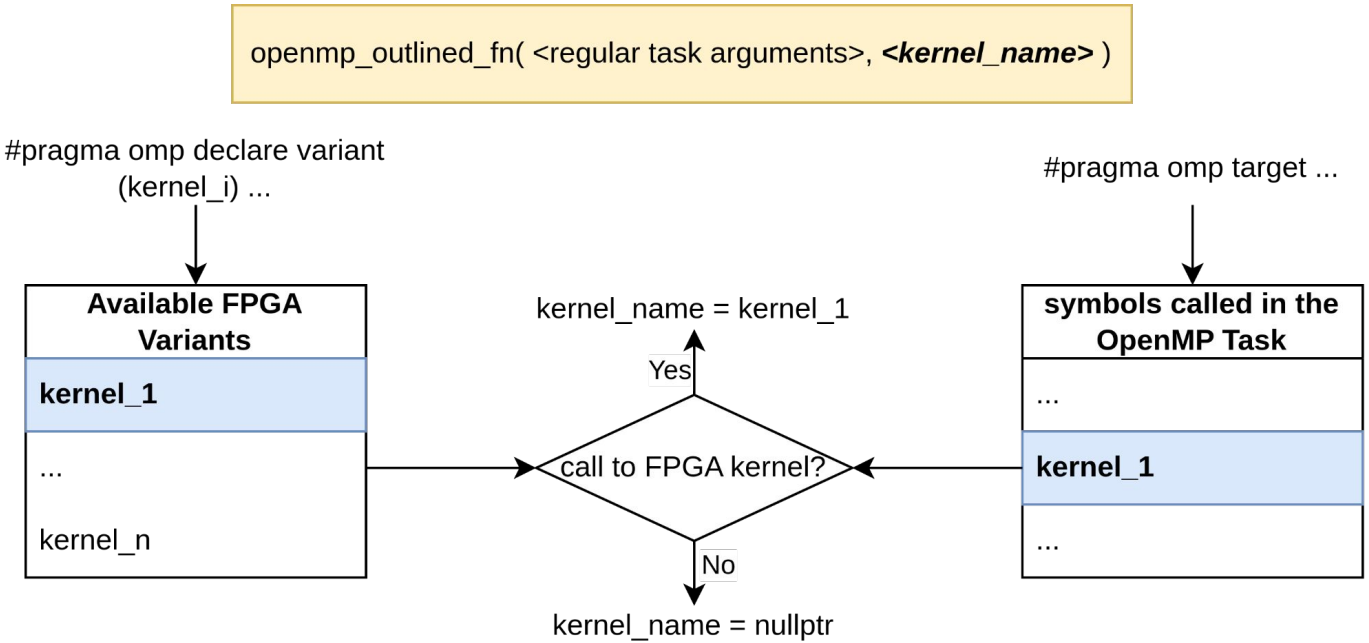
OMPC + FPGA Vadd


```cpp
#include <iostream>

void vadd_hw(unsigned int *A, unsigned int *B, unsigned int *C, size_t size);
#pragma omp declare variant(vadd_hw) match(device = {arch(alveo)})
void vadd(unsigned int *A, unsigned int *B, unsigned int *C, size_t size) {
#pragma omp parallel for
 for (int i = 0; i < size; i++)
   C[i] = A[i] + B[i];
}

void init_array(unsigned int *V, size_t size, unsigned int value) {
#pragma omp parallel for
 for (int i = 0; i < size; i++)
   V[i] = value;
}
```

```cpp
int main(int argc, char *argv[]) {
 size_t size = static_cast<size_t>(std::stoi(argv[1]));
 size_t n_blocks = static_cast<size_t>(std::stoi(argv[2]));
 size_t b_size = size / n_blocks;

 unsigned int *A = new unsigned int[size];
 unsigned int *B = new unsigned int[size];
 unsigned int *C = new unsigned int[size];

 init_array(A, size, 1);
 init_array(B, size, 2);
 init_array(C, size, 0);

 for (int i = 0; i < n_blocks; i++) {
    unsigned int *lA = &A[i * b_size];
    unsigned int *lB = &B[i * b_size];
    unsigned int *lC = &C[i * b_size];

 #pragma omp target map(to: lA[:b_size], lB[:b_size]) map(from: lC[:b_size]) \
                  depend(in: *lA, *lB) depend(inout: *lC) nowait
    vadd(lA, lB, lC, b_size);
  }
 #pragma omp taskwait

 delete[] A;
 delete[] B;
 delete[] C;

 return 0;
}
```

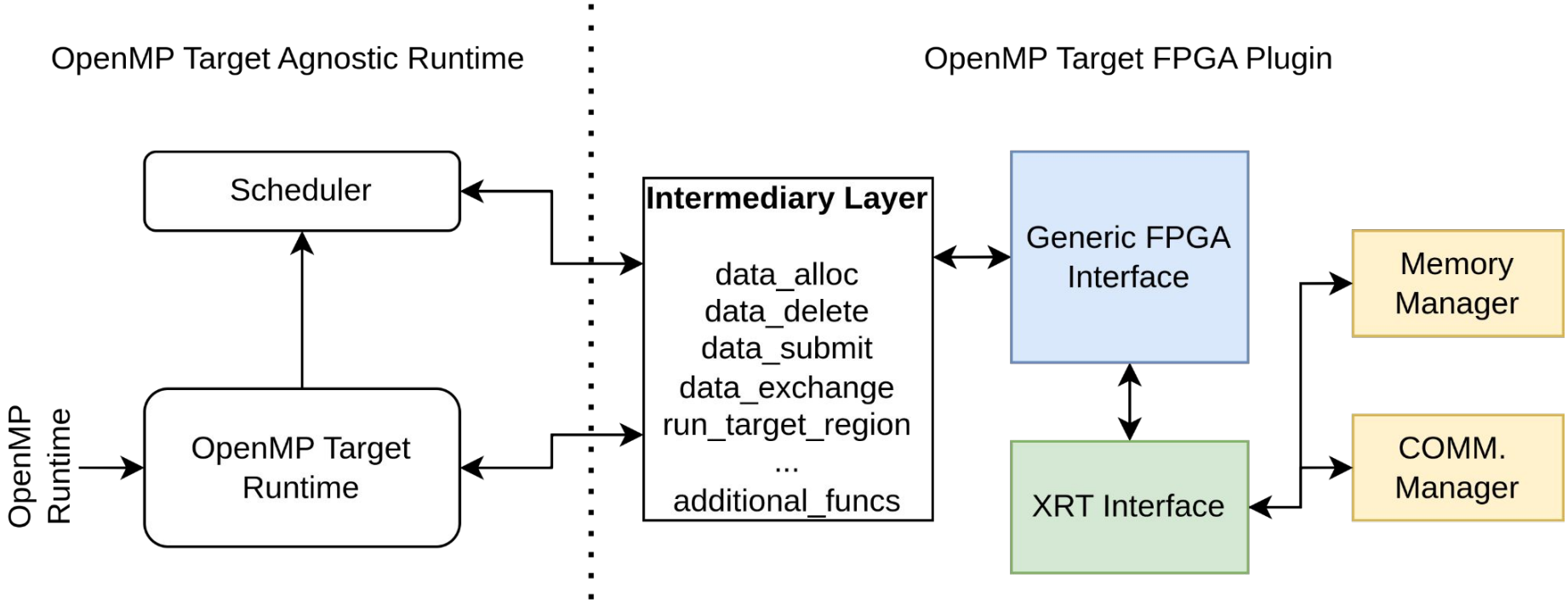
Clang extensions

- Improving previous architecture construction in LLVM's Clang [2].



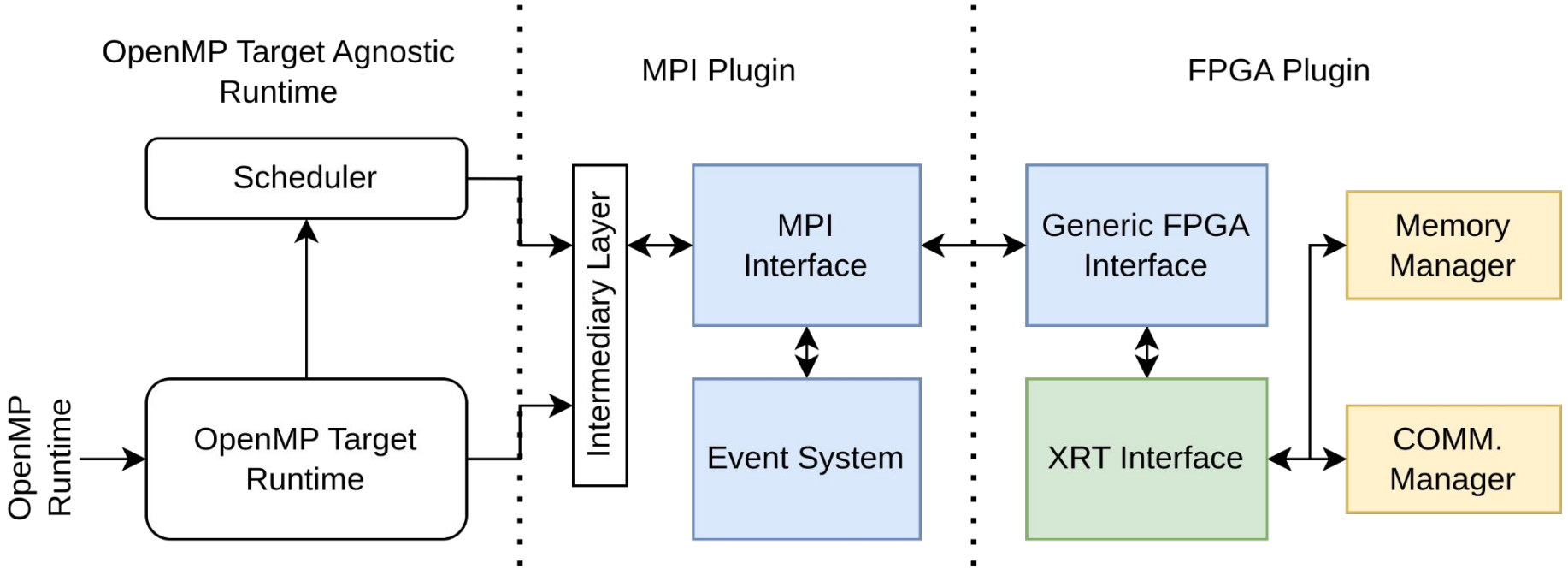
Source: Author

FPGA Plugin



Source: Author

Integration With OMPC



Source: Author

Literature Review

Reference[1]	Year	Distr.	OpenMP	HW/SW Flow	Toolkit	Streaming	FPGA Space	Compatibility	System Target	Bitstream Flexibility
[-] Rosso	2024	Yes	Standard	Separated	LLVM	Yes*	No*	XRT Compatbile	Heterogenous	Yes
[1] de Haro	2022	Yes	OmpSs	Together	Mercurium/Nanos	No	Yes	Vitis Compatible	Heterogenous	No
[2] Nepomuceno	2021	Yes	Standard	Separated	LLVM	No	Yes	Xilinx VC709	Heterogenous	No
[3] Mayer	2021	No	Standard	Together	ORKA Compiler	No	No	Intel/Xilinx	CPU+FPGA	No
[4] Huthmann	2020	No	Standard	Together	LLVM/Nymble	No	Yes	Intel FPGAs	CPU+FPGA	No
[5] Knaust	2019	No	Standard	Together	LLVM	No	No	Intel Arria 10	CPU+FPGA	No
[6] Àlvarez	2019	No	Extended	Together	LLVM	No	No	Unclear	Heterogenous	No
[7] Bosch	2018	No	OmpSs	Together	Mercurium/Nanos	No	Yes	Xilinx Zynq Ultrascale	Heterogenous	No
[8] Ceissler	2018	No	Extended	Separated	LLVM	No	Yes	Intel Arria 10	Heterogenous	No
[9] Sommer	2017	No	Standard	Together	LLVM/Tapasco	No	Yes	TPC Compatible	CPU+FPGA	No
[10] Podobas	2016	No	Standard	Together	Custom C89	No	Yes	Altera Stratix V	FPGA	No
[11] Podobas	2014	No	Standard	Together	Custom C89	No	Yes	Altera Stratix V	FPGA	No
[12] Filgueras	2014	No	OmpSs	Together	Mercurium/Nanos	No	Yes	Xilinx Zynq SoC	Heterogenous	No
[13] Choi	2013	No	Standard	Together	LLVM	No	Yes	Altera Stratix IV	FPGA	No
[14] Cilardo	2013	No	Standard	Together	Custom OpenMP	No	No	Microblaze Compatible	FPGA	No
[15] Cabrera	2009	No	Extended	Separated	Mercurium/GCC	No	No	SGI RASC	CPU+FGPA	No
[16] Leow	2006	No	Standard	Together	C-Breeze	No	No	Xilinx Spartan	FPGA	No

1. Refers to this presentation references