An efficient FPGA overlay for MPI-2 RMA parallel applications

Mathieu Leonel Mba, Roland Christian Gamom Ngounou Ewo, Julien Denoulet, Paulin Melatagia Yonta, Bertrand Granado

An efficient FPGA overlay for MPI-2 RMA parallel applications

Mathieu Leonel Mba*†‡, R. C. Gamom Ngounou Ewo§, Julien Denoulet*, Paulin Melatagia Yonta†‡, Bertrand Granado*
*LIP6, CNRS UMR 7606, Sorbonne University, Paris, France
†Department of Computer Sciences, University of Yaounde I, Yaounde, Cameroon
‡IRD, UMMISCO, F-93143, Bondy, France
§ENSET, University of Douala, Douala, Cameroon
mathieu-leonel.mba@lip6.fr, bertrand.granado@sorbonne-universite.fr, paulin.melatagia@facsciences-uy1.cm,
julien.denoulet@lip6.fr, christian.gamom@univ-douala.com

***Abstract*—Design productivity issues, including difficult hardware design and long compile times, are major barriers to the widespread adoption of FPGA-based accelerations in mainstream computing. Enabling virtualized execution of software and hardware tasks on FPGA platforms would make them more accessible to application developers accustomed to software API abstractions such as MPI and fast development cycles. In this work, we show that the MATIP platform provides a viable and efficient FPGA overlay architecture for the design of MPI parallel applications. We support this with a parallel model implementation of a feature extraction algorithm for tone language recognition, which is shown to be at least 7 times more efficient than a C++ MPI-2 RMA implementation of the same parallel model on a CPU and almost 3 times more efficient than a naive FPGA IP implementation.**

***Index Terms*—FPGA Overlay, Time-multiplexed, Spatially configured, HLS, Parallel Applications, MPI.**

I. Introduction

FPGAs provide a viable platform for accelerating computationally complex applications in areas as diverse as computer vision, digital signal processing, and networking. Progress on design tools with High-Level Synthesis (HLS) *"promise to increase the accessibility of designing on FPGAs to a broader number of designers "* according to Matai [1]. HLS tools allow designers to use high-level languages such as C/C++/Scala/Java. Several HLS tools have emerged such as Bambu[2], Intel® FPGA SDK for OpenCL [3], Xilinx Vivado HLS[4], etc. But these tools do not always take into account the different parallel programming models such as OpenMP and MPI (Message Passing Interface) commonly used in parallel software development and which eases the realization of parallel applications.

Some authors have gone in this direction by proposing tools that allow the synthesis of MPSoCs using the OpenMP and MPI standards. Such as LegUp[5] which supports OpenMP and Pthreads and synthesizes them into parallel hardware structures. The other tool in the same category is SDMPSoC (Software-defined MPSoCs)[6] which is an environment that consists of an automatic flow that analyzes an MPI-based program to build an appropriate MPSoC. The resulting architecture is an FPGA-based MPSoC capable of efficiently executing the MPI program with MicroBlaze processors or hardware modules as PEs. The system can be optimized by specialized hardware modules that are connected to the processors acting as hardware accelerators.

In parallel, researchers have investigated other techniques for improving design productivity known as FPGA overlays. According to [7], *"an FPGA overlay is a virtual reconfigurable architecture that overlays on top of the physical FPGA configurable fabric "*. In other words, an FPGA overlay is a virtual layer of architecture that conceptually locates between the user application and the underlying physical FPGA. This category of tools has many advantages; in terms of virtualization, reduced compilation time improved debugging capabilities, etc[7].

An FPGA overlay can be either spatially configured (SC) or time-multiplexed (TM), depending on its run-time reconfigurability. If an FPGA overlay has functional units (FUs) with fixed assigned tasks, it is referred to as an SC overlay. If an FPGA overlay can change the operation of its functional units on a cycle-by-cycle basis, the overlay is referred to as a TM overlay. The interconnection between functional units in SC and TM overlays can be fixed or reconfigurable at run time. The survey done in [8] highlights the fact that many overlay tools have been developed in both classes; we can mention DeCO[9] for SC overlays; GRVI Phalanx [10], and reMORPH [11] for TM overlays. The work done in[12] highlights a list of some previous parallel processing overlays.

All of the tools presented in[12] integrate parallel computing models; however, we have not encountered an FPGA overlay tool that addresses the design of MPI parallel applications without the intervention of CPUs. MPI parallelization compared to OpenMP has advantages of no parallelization overhead, except for the explicit communications that have been added to the program once the MPI parallel program has been configured; moreover, all aspects of MPI programs are generally executed in parallel, unlike OpenMP[13]. In addition, the MPI-2 RMA standard implements one-sided communication, which, unlike traditional two-sided and collective communication models, eliminates overhead from unneeded synchronization, allows greater concurrency, and leads to a

significant reduction in communication costs[14].

In this work, we demonstrate that MATIP[1][15], which can be used both as SC overlay and TM overlay, constitutes an efficient overlay for designing hardware parallel applications following the MPI-2 RMA standard.

The main contributions of this work can be summarized as follows: (1) Through the design and the implementation of a real-life MPI application, we demonstrate that MATIP is an efficient overlay; (2)We analyze the highlights and characteristics of the MATIP platform on a real MPI application.

In section II, we present MATIP from an FPGA overlay perspective. Then section III highlights the parallel model of a feature extraction algorithm for tonal language recognition. Its implementations on CPU, on FPGA through the MATIP overlay, and as hardware IP, is described in section IV. In section V we present the results of the experiment of the three implementations. In paragraph VII we conclude this work by highlighting some perspectives.

II. MATIP ON AN FPGA OVERLAY PERSPECTIVE

A. MATIP architecture

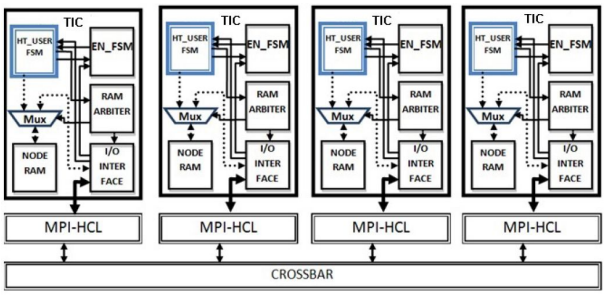

Fig. 1. MATIP platform architecture.

MATIP is an environment designed in [15] to deploy hardware tasks with a distributed memory configuration using the MPI paradigm. MATIP is structured according to a layered architecture inspired by the work described by Pavel Zaykov in [16]. This architecture is made of three layers: the interconnection layer which realizes the physical links allowing the transit of the information between the tasks; the communication layer which offers communication services between the tasks and the application layer which gathers the various tasks which constitute the application.

In MATIP, it was chosen as interconnection, a dynamic network simply made up of a crossbar that does not require complex routing and which makes it possible to carry out a dynamic network compatible with the dynamic reconfiguration.

The communication layer consists of a component that recreates a hardware version of the MPI-2 communication environment and operates the network. This component is called MPI-HCL (Message Passing Interface Hardware Communication Layer). The MPI-HCL communication component is a communication processor that executes the MPI primitives provided as micro-instructions. This processor is composed of two parts: one to execute the primitives requested by the local hardware task, and the other to process the primitives initiated by the remote hardware tasks through the interconnection network.

The purpose of the application layer is to facilitate the integration of a hardware task with the MATIP platform. It is on this layer that the designer interacts with the MATIP platform to deploy his application. To deploy an application on MATIP, the designer will have to decompose his application in the form of modules, each module carrying out a functionality, or required processing that is called in the MATIP environment a *hardware task*. Thus, the designer has a template called TIC (Task Integration Component) which is the environment that allows this hardware task to communicate with the outside. The MATIP platform defines a model for the realization of communicating hardware tasks, associating to a user task, a communication memory, and primitives, to send and receive messages based on MPI. Figure 1 shows a summary of the architecture of the MATIP platform.

B. Characteristics of MATIP seen as an overlay architecture

TABLE I
THE CHARACTERISTICS OF MATIP AS AN FPGA OVERLAY ARCHITECTURE.

Caracteristic	Value
Type of applications	*Parallel MPI*
Parallel Model	Distributed Memory
Type of FU	*SC + TM*
Level	*RTL*
FPGA compatibility	*any*
Interconnection	*NoC*
Max Number of FUs	*16*
Memory word length	*8 bits*

MATIP is fully implemented in VHDL, the tasks deployed in MATIP can be designed in VHDL or, as we propose in this article, from any tool exploiting an HLS methodology. Table I summarizes the characteristics of MATIP as an FPGA overlay.

Compared to the state of the art on parallel processing overlays[17], MATIP presents advantages as much on the variety of PE types as on its flexibility (SC/TM). Table II[2] shows the positioning of MATIP with other FPGA overlays that deal with parallel applications in state of art.

TABLE II
THE SITUATION OF MATIP IN COMPARISON WITH THE STATE OF THE ART

Overlay	Type of PE	FUs SC/TM	Topology	Communication
DRAGON[17]	Processor	SC	Mesh/Torus	Shared Memory
GRVI	Processor RV64I	TM	2D Torus soft NoC	Shared memory + MP
GRVI Phalanx[10]	Processor RV32I	TM	Hopelite NoC	Shared memory + MP
SSA[18]	FMAC	SC	Torus+Mesh	Stream
SIMD-Octavo[19]	soft-processor	TM	Mesh	Shared Memory
reMORPH[11]	CGRM	TM	2D Mesh	Shared Memory
SCMA[20]	SCM	SC	2D Mesh	Shared Memory
MATIP[15]	**HW Module**	**SC/TM**	**NoC**	**Distributed Memory**

In the next section, we present one of the most dominant and accurate[21] Pitch Determination Algorithms (PDA), used

[1]MPI Application Task Integration Platform

[2]MP(Message Passing), FMAC(Floating-point Multiply Accumulate Unit), CGRM (Coarse GRained Modules), SCM (Systolic Computational Memory)

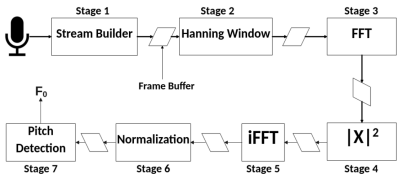

Fig. 2. Praat auto-correlation function pipeline model

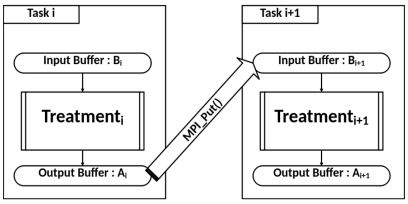

Fig. 3. Pipelined MPI-2 RMA communication scenario for Praat auto-correlation function

as a principal feature for tonal language recognition that has been the focus of our work as a parallel application.

III. Parallel model of a feature extraction algorithm for tonal language recognition

A. Praat Auto-correlation Function (Praat ACF)

Praat ACF presented in[22] is according to the classification of Hess & al[23], a *short-term analysis PDA*. According to the work of Boersma & al[22], the autocorrelation $r_x(\tau)$ of the original signal segment is obtained, by dividing the autocorrelation $r_a(\tau)$ of the windowed signal by the autocorrelation $r_w(\tau)$ of the window as in Equation (1).

$$r_x(\tau) = \frac{r_a(\tau)}{r_w(\tau)} \tag{1}$$

B. Parallel algorithm Design

To parallelize the Praat algorithm, we use a pipeline of 7-stages inspired by its different steps (see *Figure 2*). The choice of the pipeline model here is explained by the gradual arrival of the data. Indeed, the samples of the vocal signal are sent in a stream by a microphone. It is, therefore, more efficient to process one window at a time; hence the choice of the pipeline model.

C. MPI communication scenario design

To leverage the performance offered by the Remote Memory Access mechanism, we have designed a communication scenario that exploits the MPI-2 RMA communication mechanism while building the pipeline parallelization model. *Figure 3* illustrates this communication scenario. Each task processes a data frame that comes from the previous task (except the first task that operates on the data coming from the source as a microphone) through the *MPI_PUT* primitive [24]. Then it treats the information according to the computation function allocated to it and writes the result to the local memory of the next task. From one to the next, the various tasks thus form a pipeline.

IV. Parallel model implementation

We have implemented this algorithm in three versions: a parallel MPI-2 RMA CPU version, implemented in C++, a parallel MPI-2 RMA FPGA version developed with the MATIP overlay, and an FPGA version implemented as a hardware IP.

The MPI-2 RMA CPU parallel version was debugged and compiled using the Open MPI implementation [25]. The hardware testing environment was a 2.2 GHz, 12 nodes *Intel(R) Core(TM) i7-8750H CPU* computer with 8 GB of RAM, and the software testing environment is based on Ubuntu 20.04 as Operating System and OpenMPI 4.0.5 used as Message Passing Interface (MPI) library exploiting C++ as the programming language.

For better productivity, we implemented both FPGA versions using the HLS approach. For the MATIP version, we implemented the different modules with *Vivado HLS* and exported them as VHDL IPs. Then we integrated the generated IPs in MATIP through the *HDL Wrapper* and implemented the communication scenario through the *Vivado* tool. The FPGA version as hardware IP is a naive implementation of the Praat algorithm obtained by successive calls of the different modules. We have optimized all HLS implementations using the *HLS PIPELINE* and *HLS DATAFLOW* pragmas. The resulting implementations are clocked at a frequency of 100MHz. Table III presents the resource utilization of the hardware IP version, the application modules, and that of the MATIP version on *Xilinx xc7a100tcsg324-1* FPGA.

TABLE III
Resource utilization of sequential HLS version, application modules, and the HLS+MATIP architecture on Xilinx xc7a100tcsg324-1 FPGA.

	LUT	LUTRAM	FF	BRAM	DSP
Hanning	426 (0.67%)	0 (0.0%)	242 (0.19%)	2.5 (1.85%)	2 (0.83%)
FFT	13575 (21.41%)	247 (1.30%)	8071 (6.37%)	8.5 (6.30%)	101 (42.08%)
$\mid X \mid^2$	210 (0.33%)	0 (0.0%)	265 (0.21%)	0.5 (0.37%)	8 (3.33%)
iFFT	14047 (22.16%)	309 (1.63%)	9180 (7.24%)	8.5 (6.30%)	93 (38.75%)
Normalization	1444 (2.28%)	0 (0.0%)	697 (0.55%)	0.5 (0.37%)	2 (0.83%)
Pitch detection	17302 (27.29%)	110 (0.58%)	12008 (9.47%)	1.5 (1.11%)	174 (72.50%)
Hardware IP	38626 (60.92%)	96 (0.51%)	16229 (12.80 %)	15 (11.11%)	234 (97.50%)
MATIP Version	51384 (81.05%)	376 (1.98%)	79381 (62.60%)	19.5 (14.44%)	0 (00.0.0%)

V. Experiments and results

A. Dataset

Our experiment was carried out on a corpus of 8 minimal pairs of words of the *Kóló language*[3] recorded with *Audacity*[4] at a frequency of 8000 Hz.

B. Results

To validate our implementation, we evaluated the percentage of well-estimated F_0 (F_0 for which the percentage of error between the result of our implementation and that produced by the Praat software is less than 10%). The accuracy obtained for the various records is given in Table IV.

[3]Commonly known as *ewondo*, *kóló* is a language spoken in the central and southern parts of Cameroon by more than 2,500,000 people.

[4]https://www.audacityteam.org/

TABLE IV
THE ACCURACY OF OUR IMPLEMENTED PDA RUN ON THE MINIMAL PAIRS OF WORDS RECORDS

Word 1		Word 2	
Word - meaning	**Accuracy in %**	**Word - meaning**	**Accuracy in %**
$bàm$ - scold	96.29	$bám$ - worry	100.00
$bòg$ - extract	95.69	$bóg$ - pile up	99.00
$kòb$ - graze	100.00	$kób$ - join	100.00
$màan$ - reward	96.74	$máan$ - crossroads	98.86
$minkùd$ - bag	95.19	$minkúd$ - clouds	98.96
$sēg$ - decrease	91.46	$ség$ - cut out	95.06
$yēm$ - know	94.37	$yém$ - tighten	98.02
$zàm$ - good taste	96.33	$zám$ - raffia	93.26

To evaluate the performance of our model and to validate the relevance of MATIP overlay, we have executed the computation of 221 Frames on the CPU version and the two FPGA implementations (Hardware IP and MATIP version). The latency measurements in the number of cycles reveal that the MATIP implementation is at least 7 times more efficient than the one implemented on the CPU, while the hardware IP version is at least 2,7 times more efficient. The figure 4 shows the evolution curve of the latency in the number of cycles of the three implementations according to the number of executed frames. And the figure 5 shows the evolution curves of the speedup rate obtained from the ratio between the latency of the CPU implementation and that of the Hardware IP and MATIP versions. The two curves have horizontal asymptotes $y = 7$ for MATIP version and $y = 2,7$ for Hardware IP. These asymptotes give information about the speedup values when the number of frames tends to infinity.

To understand these results, we have performed measurements (cumulative average latency of the 7 stages in cycles) on the main internal steps of the tasks (see figure 6) namely: *Wincreate*, *Winpost*, *Winwait*, *Compute*[5], *Put data*, and *Wincomplete*. The *Wincreate* part (removed from the graph[6]) is very time-consuming on CPU but is executed only at the launch of the tasks, hence the high speedup rate at the first frame which drops quickly. As we can see, in these different processing segments, we almost always have better performance on the MATIP version compared to the CPU version.

VI. DISCUSSION

The above results show that MATIP is an efficient FPGA overlay for the design of MPI applications on FPGA in terms of speedup compared to a CPU version. This overlay allows new users to use FPGA in giving them the same approach as a pure software MPI program. The compatibility of MATIP with the HLS approach is a great asset for more productivity in application design. Compared to SDMPSoC[6](see Table V), MATIP implements the one-sided communication known to be more efficient than the two-sided one. Furthermore, made in VHDL, MATIP is compatible with all FPGAs and ASICs. It admits reconfigurable PEs, hence its flexibility. Its scalability depends on the network of the interconnection layer.

[5]Internal processing function (Hanning, FFT, IFFT, etc.) for each task called *Treatment* in Figure 3

[6]Values, CPU version: 481122826 cycles; MATIP version: 264858 cycles

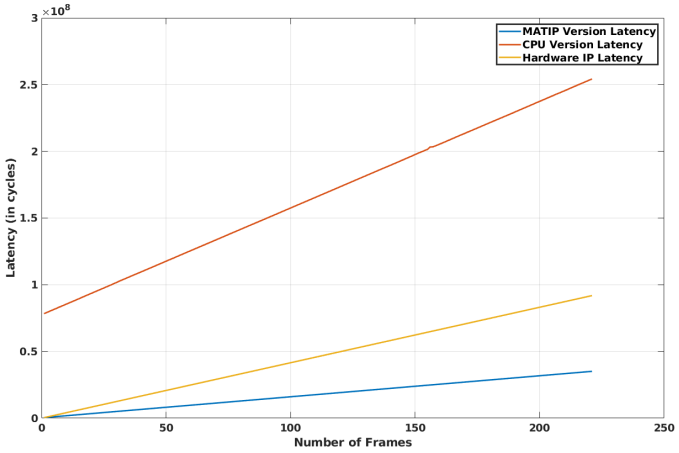

Fig. 4. Evolution curve of the latency in the number of cycles of CPU implementation, Hardware IP version and MATIP version.

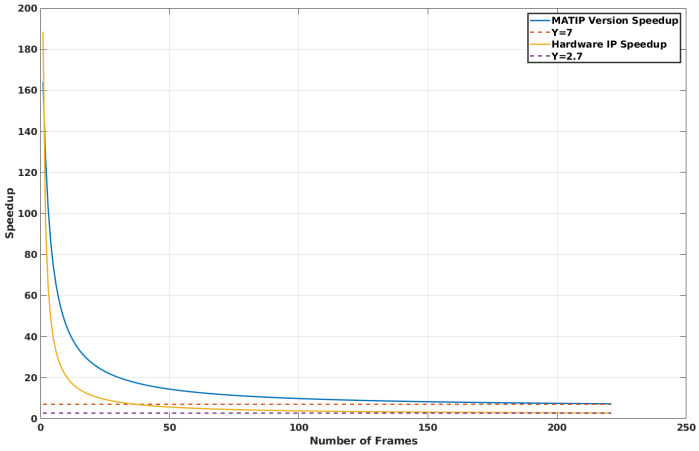

Fig. 5. MATIP version speedup versus Hardware IP version speedup computed in comparison with the CPU implementation

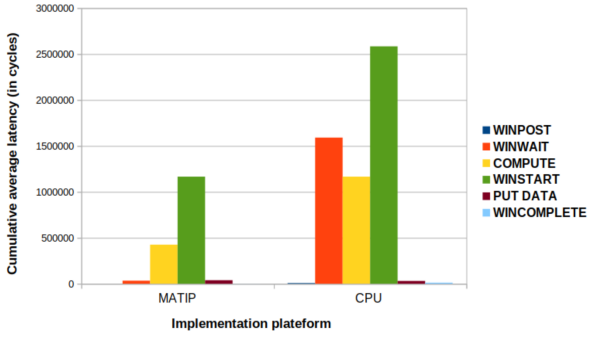

Fig. 6. Histograms of comparative analysis of the internal execution parts of the tasks on the MATIP and CPU implementations

TABLE V
SDMPSOC VS. MATIP QUALITATIVE COMPAREASON

	MPI Com.	**FPGA**	**PEs types**	**Design Flow**	**Reconf. PEs**	**Scalability**
SDMPSoC	two-sided	Xilinx	MB & HW Module	Automated	No	FPGA Limited
MATIP	one-sided	Any	HW Module	Semi-Automated	Yes	NoC Limited

VII. Conclusion

The HLS approach offers productivity but does not support parallel MPI applications. FPGA overlays encapsulate many of the problems in FPGA applications design. In the state of the art, we have not encountered FPGA Overlay tools that address MPI parallel applications design without the intervention of CPUs that can constitute a halo on the performance of the global system. In this work, we have described and analyzed MATIP, highlighting its characteristics when used as an FPGA overlay. Through the implementation of a parallel model of a real-life application, MATIP proved to be an efficient overlay architecture for the design of parallel applications based on the MPI-2 RMA standard; having obtained an implementation at least 7 times more efficient on FPGA than on a Core i7 CPU (see Figure 5). As a perspective, we consider the automation of the design flow of the HLS approach combined with MATIP, as this would provide an overlay that combines efficiency and productivity in parallel applications design on FPGA.

References

[1] Janarbek Matai et al. "Enabling fpgas for the masses". In: *arXiv preprint arXiv:1408.5870* (2014).

[2] Christian Pilato and Fabrizio Ferrandi. "Bambu: A modular framework for the high level synthesis of memory-intensive applications". In: *2013 23rd International Conference on Field programmable Logic and Applications*. IEEE. 2013, pp. 1–4.

[3] Tomasz S Czajkowski et al. "From OpenCL to high-performance hardware on FPGAs". In: *22nd international conference on field programmable logic and applications (FPL)*. IEEE. 2012, pp. 531–534.

[4] Tom Feist. "Vivado design suite". In: *White Paper* 5 (2012), p. 30.

[5] Andrew Canis et al. "Legup high-level synthesis". In: *FPGAs for Software Programmers*. Springer, 2016, pp. 175–190.

[6] Jens Rettkowski and Diana Göhringer. "Sdmpsoc: Software-defined mpsoc for fpgas". In: *Journal of Signal Processing Systems* 92.10 (2020), pp. 1187–1196.

[7] Hayden Kwok-Hay So and Cheng Liu. "FPGA overlays". In: *FPGAs for Software Programmers*. Springer, 2016, pp. 285–305.

[8] Xiangwei Li and Douglas L Maskell. "Time-multiplexed FPGA overlay architectures: A survey". In: *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 24.5 (2019), pp. 1–19.

[9] Abhishek Kumar Jain et al. "DeCO: A DSP block based FPGA accelerator overlay with low overhead interconnect". In: *2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE. 2016, pp. 1–8.

[10] Jan Gray. "Grvi phalanx: A massively parallel risc-v fpga accelerator accelerator". In: *2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE. 2016, pp. 17–20.

[11] Kolin Paul, Chinmaya Dash, and Mansureh Shahraki Moghaddam. "reMORPH: a runtime reconfigurable architecture". In: *2012 15th Euromicro Conference on Digital System Design*. IEEE. 2012, pp. 26–33.

[12] Riadh Ben Abdelhamid, Yoshiki Yamaguchi, and Taisuke Boku. "A Highly-Efficient and Tightly-Connected Many-Core Overlay Architecture". In: *IEEE Access* 9 (2021), pp. 65277–65292.

[13] Nor Asilah Wati Abdul Hamid and Paul Coddington. "Comparison of MPI Benchmark Programs on Shared Memory and Distributed Memory Machines (Point-to-Point Communication)". In: *The International Journal of High Performance Computing Applications* 24.4 (2010), pp. 469–483. DOI: 10.1177/1094342010371106.

[14] James Dinan et al. "An implementation and evaluation of the MPI 3.0 one-sided communication interface". In: *Concurrency and Computation: Practice and Experience* 28.17 (2016), pp. 4385–4404.

[15] Gamom Ngounou Ewo and Roland Christian. "Déploiement d'applications parallèles sur une architecture distribuée matériellement reconfigurable". PhD thesis. Cergy-Pontoise, 2015.

[16] Pavel Zaykov. "MIMD implementation with PicoBlaze microprocessor using MPI functions". In: *Proceedings of the 2007 international conference on Computer systems and technologies*. 2007, pp. 1–7.

[17] Riadh Ben Abdelhamid, Yoshiki Yamguchi, and Taisuke Boku. "Condensing an overload of parallel computing ingredients into a single architecture recipe". In: *2020 IEEE 31st International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE. 2020, pp. 25–28.

[18] Kentaro Sano, Yoshiaki Hatsuda, and Satoru Yamamoto. "Multi-FPGA accelerator for scalable stencil computation with constant memory bandwidth". In: *IEEE Transactions on Parallel and Distributed Systems* 25.3 (2013), pp. 695–705.

[19] Charles Eric Laforest and Jason H Anderson. "Microarchitectural comparison of the MXP and Octavo soft-processor FPGA overlays". In: *ACM Transactions on Reconfigurable Technology and Systems (TRETS)* 10.3 (2017), pp. 1–25.

[20] Kentaro Sano et al. "FPGA-array with bandwidth-reduction mechanism for scalable and power-efficient numerical simulations based on finite difference methods". In: *ACM Transactions on Reconfigurable Technology and Systems (TRETS)* 3.4 (2010), pp. 1–35.

[21] Sofia Strömbergsson. "Today's Most Frequently Used F0 Estimation Methods, and Their Accuracy in Estimating Male and Female Pitch in Clean Speech." In: *Interspeech*. Dresden. 2016, pp. 525–529.

[22] Paul Boersma et al. "Accurate short-term analysis of the fundamental frequency and the harmonics-to-noise ratio of a sampled sound". In: *Proceedings of the institute of phonetic sciences*. Vol. 17. Citeseer. 1993, pp. 97–110.

[23] Wolfgang J Hess. "Pitch determination of speech signals—a survey". In: *Spoken Language Generation and Understanding*. Springer, 1980, pp. 263–278.

[24] Marc Snir et al. *MPI–the Complete Reference: the MPI core*. Vol. 1. MIT press, 1998.

[25] Edgar Gabriel et al. "Open MPI: Goals, concept, and design of a next generation MPI implementation". In: *European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting*. Springer. 2004, pp. 97–104.