Investigation and Communication between Simulators for Molecular Dynamics with Reconfigurable Hardware

Maicon Aparecido Sartin, Ney Laert Vilar Calazans, Osmar Norberto de Souza

Abstract— Molecular dynamics simulation presents a several of important information about molecular system. However, to execute this simulations is necessary a support with high performance of computational resources for computing calculus math with real numbers. Reconfigurable devices are widely used how accelerators of dedicated hardware for the handling of the more intensive kernel computational in the simulations. The main contribution os this paper is a proposed of communication method between a host machine and a reconfigurable hardware platform based in FPGAs suggesting a software architecture used to accelerate applications of Molecular dynamics simulation, which more intensive kernel computational is investigated.

Index Terms— Application Programming Interface, Bioinformatics, Communication, FPGA, Molecular Dynamics, Parallel Processing, Profile Trace, Reconfigurable Device.

1 Introduction

BIOINFORMATICS and computational molecular biophysics are areas that require many processing resources and benefiting from availability of High Performance Computing (HPC). Specifics applications for this areas can be highlighted the simulation of macromolecules by molecular dynamics (MD). MD is a technique to simulate the movement and interactions between atoms in a molecule or between molecules using the classical equations of newton movement [1]. For example, the prediction of protein structures to require few simulation nanoseconds needs days of processing in a conventional computer or grid computing [2]. Depending on the size of them protein which structure if desire to predict can be necessary hundred of simulated nanoseconds for macromolecule unfeasible its prediction by means of computational resources [3].

The combination of conventional computers and FPGAs (Field Programmable Gate Arrays) with distinct granularity levels produce an approach technological called of High Performance Reconfigurable Computing (HPRC). Some good results of this approach to solve the problem of this work have been reported in [4] and [5]. Applications requires HPC determine a new target of technological development that involve HPRC platforms. However, there are limitations that are identified in [6]: FPGA resources and availability of hardware designers.

FPGA platforms are often used to acceleration by hardware in applications of bioinformatics and computational molecular biophysics [1] [7] [8] [9]. This hardware is used by facility relative in software migrate for hardware through of dedicated compiler. However, with the direct implementation in VHDL can to reach best performance [10] [11], although this increases too the complexity and the run time of the project.

In a HPRC platform, perform a direct communication between host computer and hardware with FPGAs. The application is divided between these components. The reconfigurable hardware implies in the employe of software and tools of the hardware manufacturing for communication between host and FPGA. Given the diversity of manufacturers is necessary the used of Application Programming Interfaces (API) to connect these equipments in several platforms bringing portability. There are already APIs open and specific proprietary for determined hardware platforms, but this have low portability.

This work describes a software architecture for the communication of the host with a hardware based in FPGA. In the host software is coupled with application of simulation by DM (more intensive kernel computational) aiming the data transfer of this application for the hardware platform via standard communication interface.

The rest of the paper is organized of the following form. The section II describe a review of the state of art in APIs guided for the target of this work. The section III lays out the bases concepts as materials and tools used in the experiments, in addition of the software investigation of simulation by MD. Section IV presents software proposed for communication API. Section V brings results about the use and validation of the software architecture and section VI discusses final considerations about work.

2 Procedure for Paper Submission

2.1 Review Stage

There are many research groups working to accelerate applications by MD, by improving the performance in modern computer architecture based in reconfigurable hardware as proposes in [7], [8] [9] e [12].

The development of APIs is not a trivial task [13] and different development environments provide by FPGAs manufacturers only relieve part of that complexity. In [13] the authors propose a generic API (OpenFPGA GENAPI) for supporting the programs integration in the calculus of classical force fields by MD with hardware accelerators based in FPGAs. Among the various features shown as needs, one can mention some: resource allocation, FPGA initialization, management algorithms in FPGA (file stream), mapping to the FPGA, and memory allocation for host data transfer to the FPGA, vice-verse, and explicit interface to block data transfer functions. The authors in [14] argued that in HPC applications there are two important aspects. First, how to use of the API hiding the parallel system and the related questions to system architecture. This work demonstrate that a big bandwidth and low latency connectivity may be important, but the correct way of using the API can also be relevant. The authors have implemented two types of applications a FFT (Fast Fourier Transform) and DGEMM (Dense Matrix Multiply Operation) operations with several hardware architectures and different sizes of floating point numbers. The authors maintained that the wrong use of the API could have a major impact in the application performance. In [10] the authors show significant reduction in development time of applications in reconfigurable computing by use of portable libraries with optimize hardware modules. Literature presents the exchange and the challenges found in the project of such libraries and provided one guidelines set in the development of portable libraries and the validation of this support through a case study with RCLib library. The library of hardware RCLib has been demonstrated (in use and performance) by two application examples, simple and multiple nodes of processing and two parallelism levels (chip and system) with different algorithms in the implementation.

3 System Features and Software Investigation of Simulation by Molecular Dynamic

3.1 FPGA

The implementation on FPGAs circuits is performed by hardware designers and its main feature is to enable a process of software development, which circuits can be rapidly designed, configured and tested in a short time and without the manufacturing costs of a dedicated circuit.

Reconfigurable hardware (FPGA) design produced locally for communication with PCI interface by means Main Bus, this interface was used to read and write tests (section XX) [17]. Fig. 1 illustrate the hardware with main module (MB_Target), four slaves modules (Slave_Read, Slave_Write, Slv_Handler_R e Slv_Handler_W), two memory modules (Mem_Out e Mem_In), module for implementations (Top) and signals in the hardware implementation. Main module have function to manage the first two modules slaves for read and write transactions in the Main Bus. Slaves modules responsible to access into memory for write and read are Handlers. Lastly, TOP module make data manipulation, in this work the module have function of data transfer of the input memory for output memory (it does not make any calculus).

Hardware is defined to perform data transfer (send and receive) of 64 bits of data for call and memory is formed for Block RAMs of the FPGA, full storage is 81.600 words of 32 bits. Data flow in the input and output of this hardware design by write and read call in Main Bus is determined by FIFO (First In, First Out) sort technique.

This hardware is viable to make tests and validation with API in the data transfer between host and FPGA platform. The board intended for realization and evidence communication in the initial tests was DN8000K10PCI with only Xilinx FPGA (XCV4FX100).

• *Maicon A. Sartin is currently adjunct professor at UNEMAT, Colider, Brazil. E-mail: mapsartin@unemat.br*
• *Ney L. V. Calazans is currently associate professor at PUCRS, Porto Algre, Brazil. E-mail: ney.calazans@pucrs.br*
• *Osmar N. de Souza is currently associate professor at PUCRS, Porto Algre, Brazil. E-mail: osmar.norberto@pucrs.br*

3.2 Molecular Dynamic

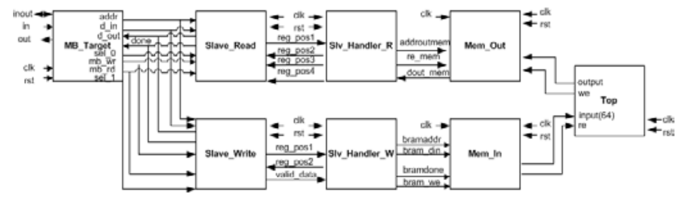

Fig. 1. Hardware design to perform data transfer by Main Bus.

The molecular mechanics uses simple functions of potential energy (such as harmonic oscillator or the potential of Coulomb) for model molecular systems. This mechanics is often applied in the refinement of molecular structures and simulations by MD, Monte Carlo (MC) and molecular docking [11]. This refinement is first step in preparation phase of a molecular system, subsequently submitted to simulations by MD.

The method of simulation by MD is widely used to obtain information about the kinetics and thermodynamics of proteins or other molecular systems over the simulation period. In the simulation can to obtain details of the movement of individual particles as a function of time of a molecular system planned to build. The information are important for the synthesis of drugs or study of the systems properties in the search for cures of diseases. However, the running such simulations is necessary high-performance computing resources. A regular molecular simulation is necessary tens or hundreds nanoseconds for researcher to analyze a molecular system. In the case study of the section III-D was performed with only 10 picoseconds, the researchers needed of a longer simulation time, causing a increase fr various days or to about one month of wait of the simulation. There are others factors that can increase the time simulation as the size of the molecular system, the reduction of the interaction time (timesteps), and others.

MD and several other techniques are related in a methodology of computational simulation integrating the equation of Newton movement. Thus, model of the temporal evolution of the interactions between atoms or molecules determine the force on each atom repeating these interactions among all and generating the evolution of the molecular system. MD is based in classical mechanics or second law of Newton. Function of potential energy is divided in two parts: bonded and non-bonded atoms, such Eq. (1). In the first part the bonded atoms contains calculus of the number of covalent bond, angles and dihedral. Nonbonded atoms have the calculus of Van Der Walls and the electrostatics interactions [7].

$$E(\text{potential}) = \sum_{\text{bonded}} E_b + \sum_{\text{angles}} E_a + \sum_{\text{diedral}} E_d + \sum_{j=1}^{N} \sum_{i=1}^{N} (E_{\text{vdw}}) + \sum_{j=1}^{N} \sum_{i=1}^{N} (E_{\text{el}}) \qquad (1)$$

In the case studies performed of this paper employ the package of simulations by MD the AMBER [18] with version 9. Two modules were used of this package (SANDER and PMEMD). The software was chosen as popular and respected between the researchers in bioinformatics area.

Initial parameters for simulation by MD and molecular system are defined in input files of the experiments, of this files were obtained locally [19]. These features are equals in all experiments presented subsequently. The MD used in the simulation is isobaric-isothermal (NPT) type. Molecular system is compound by a protein molecule, a coenzyme and 6 counterions, water molecules 10,532, containing a total of 35,681 atoms as Fig. 2.

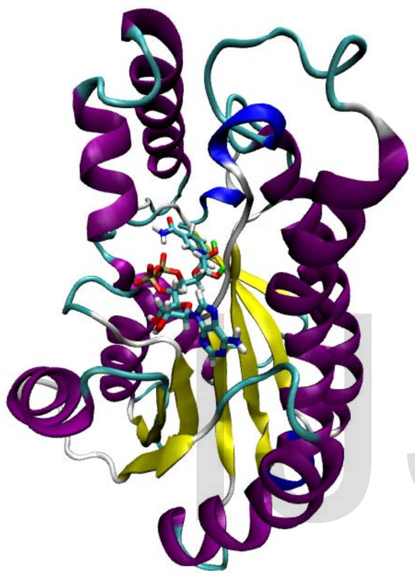
Fig. 2. Tertiary structure of the molecule without the orthorhombic box and featuring NaH waste by drawing in the form of sticks in the center of molecule structure. This molecule is immersed in an orthorhombic box size 72.372 Å x 68.278 Å x 72.019 Å with PDB ID 1ENY.

3.3 Application Profile

GPROF (GNU PROFiler) tool [15] of the GNU was used to analyze the more intensive computational kernel and to generate profile trace of the applications running in the host with profile plan and call graph. In profile plan can be visualized with details the spend time in each module and application routine. The call graph obtained hierarchy and sequence of the calls performed in all application, included quantity of calls in each module or routine. These results are exposed in Fig. 3. The quantity of calls in each routine is in brackets (left side) with execution sequence of the PMEMD software routines. In subroutine short_ene can to observe basic structure of the software together with quantity of repetitions. The fundamental loop PCH (Part Converted Hardware) is presented in Fig. 3. In application of simulation by MD of the PMEMD (target software) are run 5 modules as sequence of the Fig. 3 (left side), at least until the routine short_ene (target routine).

The values shown in Fig. 3 were acquired of a simulation using PMEMD with 200 interactions and a timestep of 2 femtoseconds (fs) for each iteration. The values was calculated based on the data amount generated in the short_ene subroutine monitoring. This number was opted to show clearly the functions more visited together the hierarchy and spending time equal to a larger simulation. Thus, the ideal piece of code to perform parallelism and consequently the transport for hardware platform in the acceleration. In [16] shows the same subroutine (short_ene) such as greater computational cost for simulations based in SANDER. Total time of the subroutine (short_ene) is 73.14% and force calculus for nonbonded atoms arrive 70.82% of the application runtime. In SANDER were defined the same input parameters presented by MD, but with 5.000 timesteps or 10 picoseconds of simulation time in serial mode. In this test short_ene subroutine spend approximately 68.77% of all time runtime simulation or part with greater computational cost. Percentage of each calculation was obtained with JAC (Joint Amber-Charmm) benchmark of the AMBER package.

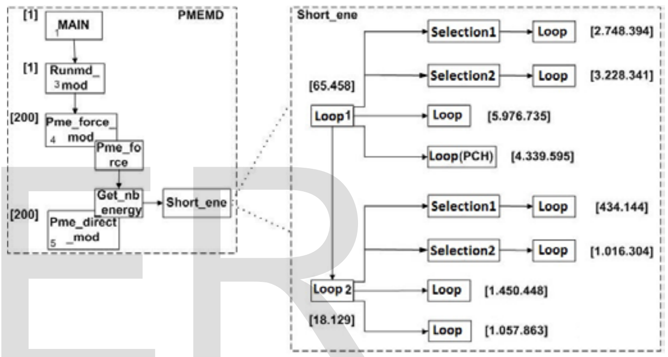

Fig. 3. Call graph of a simulation by MD with 400 fs using PMEMD..

3.4 Simulations by MD with AMBER Package

The simulation performed with PMEMD contain same features presented in section III-A and simulation time was defined as 10 picoseconds or 5.000 timesteps. Results of this case study in the two applications by MD (PMEMD and SANDER) in different environments were suitable for comparative effect between the same. Two computers contained a four cores processor and RAM memory 4 GB and other a two cores processor an RAM memory 2 GB. Simulation was running in each one computer distributing process between the multi-core processors. Other environment employed two computers to form a litle cluster distributing process in the two nodes with MPI communication between them. Simulations used to 8 different process quantity: 1, 2, 3, 4, 5, 6, 8 and 12.

In results for two independent computers with simulations using two applications by MD, there was clear difference between the two computers in the two applications. In the simulations with cluster obtained worst results than the only computer with four cores (Fig. 4). Other important point as can be observed and main stimulus for achievement of this simulation is a comparative between two applications of the AMBER package (PMEMD and SANDER). In all environments shown, the PMEMD application was higher than SANDER, even with architectures and quantities process distinct. For this reason and simplicity and code optimization, it was opted that PMEMD as a good basis for this work.

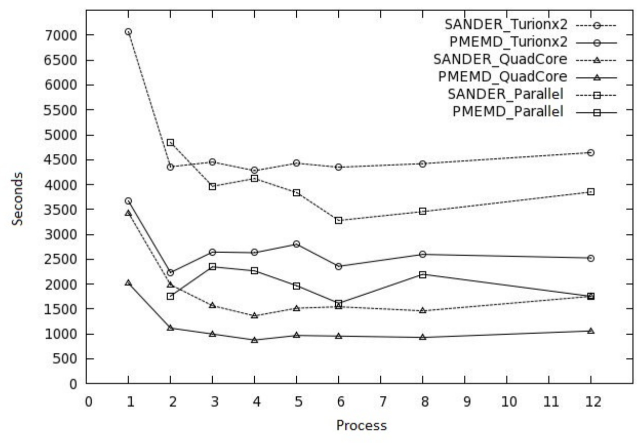

Fig. 4. Simulation by MD with PMEMD and SANDER: a comparative between a parallel environment (two computers) and one computer with four cores processor (Intel) and one computer with two cores processor (AMD TurionX2).

4 Proposal for Software Architecture

In the software architecture proposed in this work is very important the definition of the form and structure in API call for communication between hardware and software. It's important to emphasize the organization of this software architecture for approaching the communication of hardware and software and the different abstraction levels.

Three abstraction levels were employed to provide reasonable abstraction to the software and with a view to facilitating the use of the API. The first level understands of the higher layer of software where the programmer makes API call. In the second, communication interface is defined that will be used by the hardware platform, for example, PCI, Ethernet, USB, etc. The last level makes access to hardware platforms to be used depending on the interface defined previously. This division of the layers of the API aspires to adapt functionalities and hardware platforms, as Fig. 5. The API organization and its abstraction levels are presented in Fig. 6.

API proposed facilitates to aggregate the functions for this application and providing the API portability. API offerings support a FPGA platform through of a PCI interface communication. The adapted modules to PMEMD software were organized of form to present a structure as the abstraction layer that adopting initially as guide. The project is modified only in the level desired, for example, the FPGA platform or communication interface in a new project, thus reducing the complexity of this integration with the application of simulation by MD. Nowadays, API is used with the PMEMD application. The API portability with other software different of the AMBER package can be performed easily, provided that the software has support to C language.

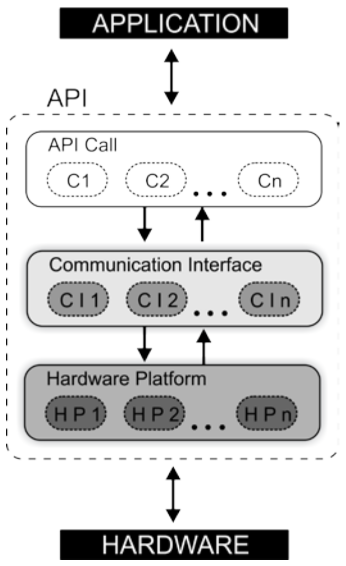

Fig. 5. API abstraction layers.

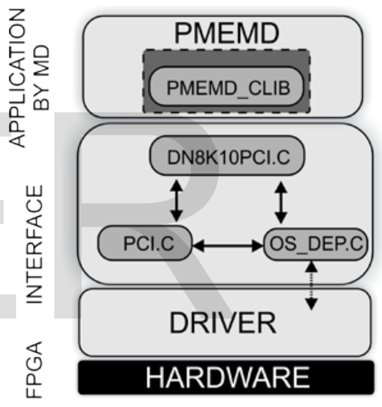

Fig. 6. Modules organization involved in the API and its abstraction layers.

Software/Hardware communication in this routines are initialized in the loading of the driver module of the hardware platform (third level), selecting the platform model used by means of requisition to driver. After to load driver of the platform can be executed the routines which function is write and read in communication bus (Main Bus) according to primitive type (second level). At first, the API was defined with some standard primitives in the high abstraction level as Table 1. The part of start and stop in the API is determined in the calls sent and received of data respectively with a block call between the modes.

5 Results

The API validation in a simulation by MD works together with PMEMD software. Data sent were performed in simulation time in the MD application with hardware platform. Data return were received in the MD application with API, which function is to make interface between host (PMEMD) and hardware platform (FPGA). To demonstrate these validation

TABLE 1
DESCRIPTION OF PRIMITIVES HIGHER LEVEL OF ABSTRACTION OF THE API PROPOSED.

ROUTINE	DESCRIPTION
API_DATA (control, data1,..., data15)	Routine that enables data send for running of the more costly loop of the short_ene subroutine. Primitive contains data of several types and sets, usually in floating point and first field is used for running control.
API_DATA_I (control, data, parameter, size)	Routine to send integer data to FPGA. Third field defines the type of data sets (variable, vector or matrix) and fourth field defines the size of the data type.
API_DATA_D (control, data, parameter, size)	Routine to send floating point data (double precision) to FPGA. The fields follow the same definition and constraints found in API_DATA_I.
API_REC_DATA(control, data1, ..., data13)	Once the data has been sent as the functions, it is necessary to wait for FPGA that will perform the calculus and returning only the variables which were necessary for the simulation running in the PMEMD software. MD application in the host performs this function and then wait to receive the data. This primitive contains data of several types and sets.
API_REC_I(control, data, parameter, size)	This routine receives just one type of integer data at time different from API_REC_DATA() routine. Second field determines the variable that will receive data of the hardware platform respecting the data type and following the same definition and constraints found in API_DATA_I().
API_REC_D(control, data, parameter, size)	Routine receives a type of floating point data and double precision as defined in API_DATA_D() and in according with the API_REC_I().

were carried out two preliminar experiments with equal parameters of the simulation by MD: hardware project in the FPGA and modes of transfer in the API.

Data transfer (receiving or sending) is composed by a call to PCI bus with one word of 32 bits in hexadecimal. In the API determined the variables of the integer type using just one word and the floating point two words. The start and stop definition is made in the coding and decoding of all data to correspond the hexadecimal number employed by bus, holding a owner protocol for data identification. Thus, the informations transferred between platforms are proof, following to bus constraint and making API validation.

5.1 First Experiment

In a first instant, the API had objective to make communication between host and hardware platform without worrying about performance.

In the code of the PMEMD software has to include the API primitives in Fortran language with due to its characteristics. Thus, operations of writing and reading are triggered by first level of the API passing by intermediate level with the basic definitions of operating system and bus addressing. In the third level makes the driver initialization and mapping of the communication in buses as Fig. 7.

For each sending and receiving of two words of 32 bits were defined that the two and three levels will be initialized and disabled after termination of the communication aiming to ensure the facility of API validation. Then, each two words have an interruption to driver of the DN8000K10PCI board for sending and receiving of these words. In this interruption changes access mode of user to kernel that allowing access to hardware device as Fig. 7.

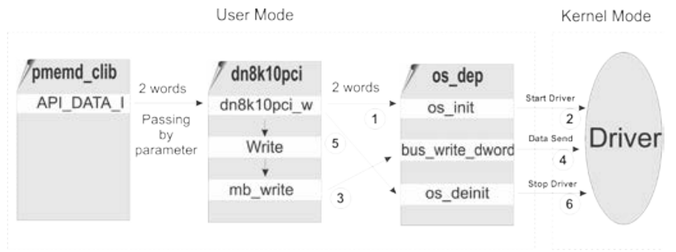

Fig. 7. API call of data sendo of the integer type with two words sent by interruption.

Experiment results are showed by Fig. 8 and last character in each operation corresponds to data type (I – integer and D - double precision floating point).

In the experiments were demarcated different sizes, variables types and transfer types. At first, two variable types were included (integer and floating point) in the PCH of the PMEMD software. Sizes are only a variable, three vector dimensions (8, 32, 128) and matrices (3x8, 3x32, 3x128). In order to make things easier to see put follow values 1, 8, 24, 32, 96, 128 e 384 (vectors - 8, 32 e 128; matrices - 24, 96, 384). Zero value in the Fig. 8 corresponds to running of the simulation in the PMEMD software without API call. Three types of transfers were performed sending (SEND), receiving (REC) and sending and receiving (SEND_REC) for each value of variable amount. By Fig. 8 can observe similarity in the operation times between SEND_I and REC_I, and also in SEND_D, REC_D and SEND_REC_I. In all cases, which quantity is greater than 1 the transfer times are unacceptable in the bus use.

5.1 Second Experiment

As in first experiment observed a high runtime of the transfer operations of API data. In the second experiment sought to generate a performance improvement. Thus, in each transfer of two words verified that there was no need of the interruption for enable and disable the hardware platform driver. This cause a big increase of time in the transfer with a greater number of variables.

The interruption was established for each call done to API and to ensure the right time of transfer. Hence, the number of variables to transfer, will not make any difference and the interruption time will be the same in all cases, using dynamic allocation of memory and passing just only pointer for access the data to be transfer.

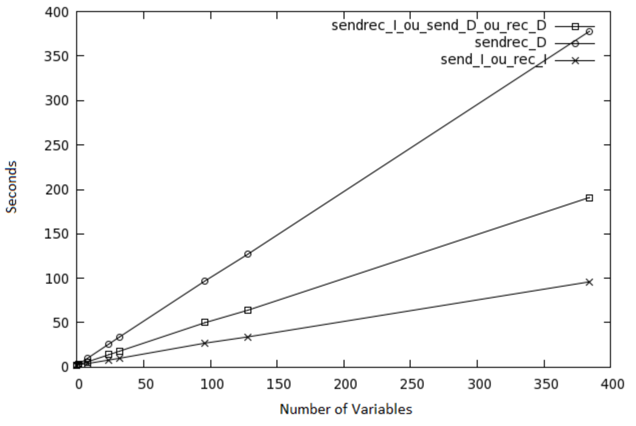

Fig. 8. Results of the first experiment with API.

In the validation of this improvement can to observe just a little variation in the runtime in each operation. This is because of the own simulation that has different processing time, influencing the results. In some cases, the variations occur in descending order of quantity of variables. In this way, API running is not influencing in the simulation runtime and these differences are determined by own simulation.

In this experiment was possible to observe the API validation with a several parameters and runtimes. Results present that there was no influence the simulation time of the molecular system in the PMEMD software. The data were tested in the sending and receving for the software in the host without any problem.

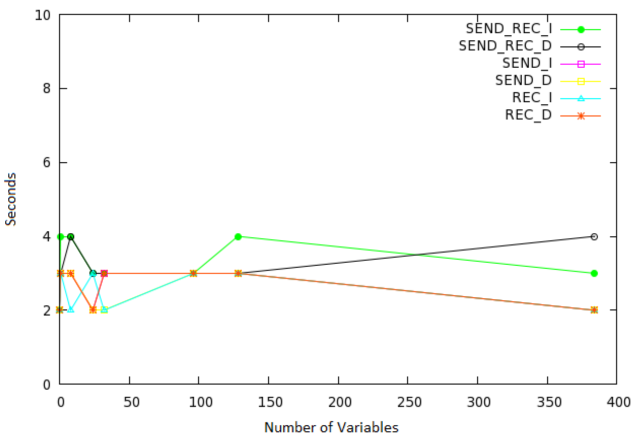

Fig. 2. Results of the second experiment with API.

6 Conclusion

This work presents the development of a software architecture for communication of a host with a hardware platform based in FPGA to given support in the organization model of a high performance structure with dedicated hardware.

Conducting the necessary case studies were very important by acquisition of determinant informations to project. API was validates with demonstration of two experiments performed with several characteristics as different parameters, number of variables and communication modes, providing a comparation between them. The same molecular system was used in all cases.

Acknowledgment

The authors wish to thank UNEMAT, PUCRS, FAPEMAT and FACIN.

References

[1] X. Yang, S. Mou, and Y. Dou, “FPGA-Accelerated Molecular Dynamics Simulations: An Overview”, In: Int. Work. on App. Reconfig. Comp. (ARC), 2007, pp. 293-301.
[2] C. Bystroff, S. Garde, “Helix propensities of short peptides: Molecular dynamics versus Bioinformatics”, Proteins: Structure, Function and Bioinformatics J., 50, 2001, pp. 552-562.
[3] B. Batson et al., “Anton, a Special-Purpose Machine for Molecular Dynamics Simulation”, In: The 34th Annu. Int. Symp. on Comp. Architecture (ISCA), 2007, pp.1-12.
[4] M. Beauchamp et al, “Architectural modifications to improve floating-point unit efficiency in FPGAs”, In: Proc. Field Prog. Logic and Applications, 2006, pp. 515–520.
[5] D. Strenski, “FPGA Floating Point Performance – a pencil and paper evaluation”. HPC Wire, 2007.
[6] Y. Gu, M.C. Herbordt, “High Performance Molecular Dynamics Simulations with FPGA Coprocessors”. In: Reconfig. Systems Summer Institute (RSSI), 2007.
[7] P.K. Agarwal et al, “Using FPGA Devices to Accelerate Biomolecular Simulations”. IEEE Computer, 40(3), 2007, pp. 66-73.
[8] Y. Gu, T. Vancourt, amd M.C. Herbordt, “Explicit design of FPGA-based coprocessors for short-range force computations in molecular dynamics simulations”, Parallel Comp., 34(4-5), 2008, pp. 261-277.
[9] R. Scrofano et al, “Accelerating Molecular Dynamics Simulations with Reconfigurable Computers”, In: IEEE Trans. on Par. and Distr. Systems, 19(6), 2008, pp. 764-778.
[10] P. Saha et al, “Portable library development for reconfigurable computing systems: A case study”, Parallel Comp., 34, 2008, pp. 245-260.
[11] S.A. Adcock, J.A. Mccammon, “Molecular Dynamics: Survey of Methods for Simulating the Activity of Proteins”, J. Chem. Rev., 106(5), 2006, pp. 1589-1615.
[12] A. Patel et al, “A Scalable FPGA-based Multiprocessor”. In: 14th Annu. IEEE Symposium on Field-Prog. Custom Comp. Machines (FCCM), 2006, pp. 111-120.
[13] E. Stahlberg et al, “Molecular Simulations with Hardware Accelerators: A Portable Interface Definition for FPGA Supported Acceleration”. In: Reconfig. Systems Summer Institute (RSSI’07), 2007.
[14] K.D. Underwood, K.S. Hemmert, C. Ulmer, “Architectures and APIs: Assessing Requirements for Delivering FPGA Performance to Applications”. In: Int. Conf. for High Performance Comp., Networking, Storage and Analysis (SC), 2006.
[15] S. L. Graham, P.B. Kessler, and M.K. Mckusick, “Gprof: A call graph execution profiler”. SIGPLAN J., 17(6), 1982, pp. 120-126.
[16] J. Kuehn, “A Brief Overview of Activities in the Future Technologies Group at Oak Ridge National Laboratory”. In: Workshop on Research Alliance in Math and Science (RAMS), 2005.
[17] Hardware Design Support Group, “Grupo de Apoio ao Projeto de Hardware (GAPH),” https://corfu.pucrs.br/. 2015.
[18] AMBER, “Amber Home Page,” http://ambermd.org/. 2015.
[19] LABIO, “Laboratório de Bioinformática, Modelagem e Simulação de Biossistemas (LABIO),” http://www.labio.org/. 2015.