A High Performance FPGA-Based Accelerator for BLAS Library Implementation

Sébastien Rousseaux
CETIC[1], Belgium
sr@cetic.be

Damien Hubaux
CETIC[1], Belgium
dh@etic.be

Pierre Guisset
CETIC[1], Belgium
pg@cetic.be

Jean-Didier Legat
UCL[2], Belgium
doyen@fsa.ucl.be

[1]Center of Excellence in Information and Communication Technologies
[2]Université Catholique de Louvain

Abstract

This paper describes the implementation and the performance analysis of a hardware accelerator for the BLAS library matrix multiplication operation. This accelerator is based on a dual-FPGA board and on an implementation BLAS software library making use of the FPGA-based hardware. In order to evaluate the performance of such a system, we implemented the matrix multiplication operation (BLAS "dgemm" function) using an optimized matrix multiplication FPGA design and we implemented the software "dgemm()" function to make use of the FPGA-based board in a completely transparent way for the user. In contrast with others works [2,5,6,10], the measured performance is based on the global runtime of the FPGA-accelerated "dgemm" function at software level, taking into account the data transfers between the host computer and the FPGA board, and the software pre- and post-processing. We show that using the developed FPGA-based BLAS accelerator it is possible to achieve 60% higher performance than a fully software implementation running on a high-end computer. Through a detailed analysis, this paper also shows that the most limiting factors are data transfers between the host computer memory and the FPGA board memory, and the data transfers between this memory and the FPGA itself.

1. Introduction

Over the last few years, the quality and the precision of models produced by virtual prototyping and numerical simulation software increased significantly. Keeping up with such high standards of quality and precision requires significant and constant increase of computation power. The current solution consists in using machines with more and more processors. Unfortunately, this solution is suboptimal and very expensive.

Next to classic processors, the computation power of another sort of device is growing every day: That of programmable logic components like FPGA. One of their advantages compared to the classic processors is their capacity to perform a large number of operations in parallel. Several supercomputers manufacturers are interested in the use of the performance obtained with FPGA to accelerate the execution of software [12].

The prototyping and numerical simulation software massively use algebraic methods of system resolution. Most of these methods are based on operations such as matrix multiplication, matrix factorisation, etc. According to their importance these operations are grouped together in libraries. Among the most used, we find the BLAS library (Basic Linear Algebraic Subprograms). Most of the BLAS operations requires a lot of computing power, so it could be useful to decentralise their computation on a specialised board to accelerate their execution.

This paper studies the conception of a FPGA based Hardware accelerator for the BLAS library and analyses the constraints impacting the global performances of the system. The purpose of this accelerator is to decentralize and to speed up the execution of the BLAS library.

This article is organized as follows: Section 2 presents the theoretical basis; Section 3 describes the preliminary evaluation of theoretical performance; Section 4 describes the implemented algorithms and the hardware used to build the prototype; Section 5 introduces the measured results; Section 6 presents a detailed performance analysis; based on this analysis, section 7 introduces the requirements of an adapted FPGA board and section 8 draws the conclusions of the work.

2. Background

This section presents the theoretical base for the implementation of a BLAS operation and introduces the architecture of the implemented system.

2.1 The BLAS Library

Algebraic operations such as matrix multiplication or factorisation are key operations in numerous software algorithms of systems resolution. For this reason, these operations are often grouped together in standardised libraries. One of the most used library is the BLAS library (Basic Linear Algebraic Subprogram) [11]. It defines a standard interface and a standard implementation for basic vector and matrix operations. They are grouped in 3 levels following the type of the operands: level 1 for operands of type vector-vector, level 2 for operands of type vector-matrix and level 3 for operands of type matrix-matrix. These operations can be reimplemented by the developers to optimize performance.

2.2 Preliminary System Description

The main purpose of the FPGA Hardware accelerator is to speed up BLAS library operations. Its main requirements are:

- The use of the FPGA hardware accelerator has to be transparent for the user
- It has to install easily in a common computer to increase its computing performance while using BLAS library operations. The board will be then composed of a standard communication interface (PCI/PCI-X/PCIe).

The architecture of the hardware is composed of a host computer running the BLAS based application

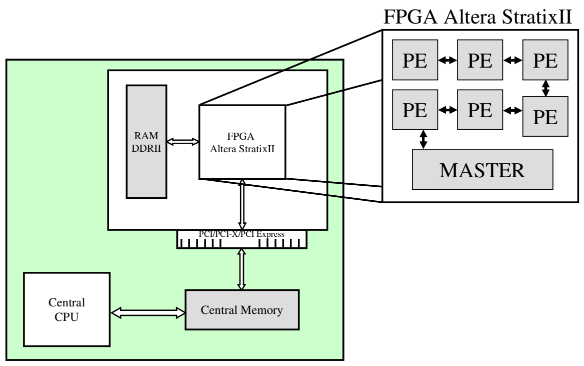

Figure 1 : General System Architecture

and a FPGA board plugged in the mainboard of the computer.

As shown at Figure 1 the global system is composed of 5 main components:

1. The CPU of the host computer, which performs the software data processing of the BLAS library and manages the data transfers between the Central Memory and the FPGA board
2. The Central Memory, which contains the global data operands to be transferred to the FPGA board
3. The FPGA board communication interface (PCI/PCI-X/PCIe): This interface has an important impact on the global performances of the Hardware accelerator.
4. The FPGA, which performs the calculations and controls the RAM memory of the FPGA board.
5. The local RAM memory of the board, which contains the operands of the operations.

An advantage of the FPGA for High Performance Computing applications is their ability to perform a huge number of operations in parallel. Taking this specificity into account, the structure of the FPGA design will be composed of many Processing Elements (PE) running in parallel and of a module managing computations and data transfers (see Figure 1).

3. Theoretical Analysis

3.1 Performance

Based on the FPGA design architecture (see Figure 1), the FPGA theoretical raw computation performance (FTRCP) can be evaluated using the following expression [2]:

$$FTRCP = Op \times PE \times Fmax \quad [\text{Gflops/s}] \qquad (1)$$

Where:

- Op = The number of floating point operations performed by each Processing Element per clock cycle
- PE = The number of Processing elements in the FPGA design
- Fmax = The maximal frequency of the FPGA design

It clearly appears that the raw computation performance strongly depends on the number of PE in the FPGA and thus on the number of operations performed in parallel. The FPGA features impacting the number of PE are their spatial density (number of Logical Elements) and their available resources (embedded multipliers, internal SRAM, etc). The number of PE can be increased by selecting a FPGA containing more logical elements.

The maximal frequency mainly depends on complexity and optimisation of the FPGA design. It is difficult to increase this value significantly.

This expression (1) doesn't take into account the data transfer performance. The impact of data transfer on global performance is discussed in sections 3.2 and 6.1.

3.2 Preliminary Limitation Analysis

As described in section 2.2, the global system is composed of 5 main components. To analyse the performance limitations we consider that the FPGA has an important power of raw computation and isn't limiting the global performance of the system. In such circumstances the high computation performance of the FPGA are mainly limited by external factors. There are two main factors that can reduce the performances:

1. The software processes running on the host computer and limiting the availability of computation data for the FPGA board
2. The data transfers

Data transfers between the host computer and the FPGA board

Actual FPGA boards are equipped with SDRAM DDR or SDRAM DDR2 memory. This type of memory isn't "Dual Port": it is not possible to perform write and read operations at the same time. During the computation phase, the FPGA reads continuously its operands in the RAM memory of the board. As it isn't possible to write data in this FPGA board RAM memory during this phase without an important loss of performances, the host computer has to wait for the end of the FPGA computations to download the results from the board memory to the central memory of the system and to upload new data for the next computation. During these transfer phases, the FPGA isn't able to perform any computation which significantly reduces the global performance.

Consequently, global performance will closely depend on the maximum bandwidth of this communication interface. The most widespread board interfaces and their maximal bandwidth are presented in Figure 2. We can see that currently, the PCI Express 8x bus has the largest bandwidth.

Bus Type	Bandwidth
PCI 66Mhz 64bits	533 Mbytes/s
PCI-X 133Mhz	1 Gbyte/s
PCI Express 1x	500 Mbytes/s
PCI Express 4x	2 Gbytes/s
PCI Express 8x	4 Gbytes/s

Figure 2 : Communication Interfaces

Data transfers between the board memory and the FPGA

In order to run at full speed, the FPGA needs to access as fast as possible the operands contained on the board memory. The performance of memory accesses is linked to the performance of the memory controller implemented in FPGA logic. As the maximal running frequency (Fmax) of the FPGA design is limited due to its complexity, this will impact the performance of the memory controller. Ideally, if the FPGA design and thus the memory controller ran at 200 Mhz, the FPGA would receive new valid data from the board memory at every clock cycle at 200 Mhz. As the SDRAM memories have a yield of around 75%, the FPGA design only receives new valid data every clock cycle at approximately 150 Mhz. So to avoid breaks in the data stream, the FPGA design will run at a maximal speed of 150 Mhz. This will reduce the raw computation performance of the FPGA and thus the global performance of the system.

Global Performance Maximisation

To maximize the performance we have to:

- Use a FPGA board with the fastest interface. E.g. PCI Express 8x
- Optimise the FPGA design to maximise the running frequency of the FPGA
- On the algorithmic side: Minimize the amount of data accesses and maximize the number of operations performed on one transferred data.

By analogy to the theory introduced in [9], we define the factor Va representing the number of operations that can be performed for every Word transferred from the board memory to the FPGA.

$$Va = O / Ma$$

Where:

- O = the total number of performed operations
- Ma = the total number of memory accesses

3.3 BLAS Operation Selection

According to the previous section analysis, the most interesting operation to study maximise the factor Va. As described in section 2.1 the BLAS library is composed of 3 levels according to operand type.

- BLAS level 1: scalar multiplication of 2 vectors of size n:
  O = 2 x n, Ma = 2 x n → Va = 1
- BLAS level 2: vector-matrix multiplication:
  O = 2 x n2, Ma = n x (n+1) → Va = (2xn)/(n+1)
- BLAS level 3: matrix multiplication of square matrixes of size nxn:
  O = 2 x n3, Ma = 2 x n2 → **Va = n**

As shown in these three examples the BLAS operations with the highest factor Va is a level 3 operation. The matrix multiplication is an interesting operation to implement in the prototype due to its parallelisation capabilities and its importance in many system resolution algorithms. The operations based on 64-bit floating point operands require a lot of processing power. We thus implement the 64-bit floating point matrix multiplication operation in the FPGA based hardware accelerator prototype. This operation corresponds to the BLAS operation: “dgemm()”.

4. FPGA based Hardware Accelerator Prototype

This chapter describes the implementation of the FPGA based BLAS “dgemm()” accelerator.

4.1 FPGA Design Description

This section describes the FPGA design implementing Matrix multiplication operation and optimisations applied to maximise its performance.

Consider 3 matrixes A, B and C of sizes respectively MxN, NxK and MxK. Matrix C is the result of the matrix product of A and B. The elements of C are calculated as follows:

$$C_{ij} = \sum_{k=0}^{N} A_{ik} \cdot B_{kj}$$

With i = 1,…,M and j = 1,…,K

We notice that the matrix multiplication is exclusively composed of sums of products (“Multiply and Accumulate” operations (MAC)). The processing core of each PE is thus a MAC unit. According to the number of PE and the size of matrix A and B each PE computes (MxK) \ PE elements of C. The total number of performed operations is: 2 x N x M x K.

Following the analysis presented in [2], two main algorithmic optimisations have been applied to the basic matrix multiplication algorithm to take advantage of the FPGA features:

1. <u>Block product:</u> This optimization is important for the implementation of matrix multiplication operation on FPGA in order to reduce the FPGA on-chip memory requirement. The matrix C is divided into a certain number of blocks of size (SixSj). Each of these blocks is the result of the matrix product of a block (SixN) of A and of a block (NxSj) of B. The computation of the various blocks of C are independent some of the others.

2. <u>Re-use of elements of matrix A:</u> If we analyze the classic algorithm of the matrix multiplication, we notice that every element of the matrix A is reused K time. For every element of A transferred, we can execute K calculation using K elements of B. Considering this feature, by storing in FPGA internal memory a small number of elements of matrix A we can perform a large number of operations. We reduce significantly the requirements of bandwidth between the RAM memory on the FPGA board and the FPGA itself.

On the FPGA design architecture side, the organisation of the PE in the FPGA plays an important role in the performance of the matrix multiplier. Indeed, every clock cycle, every PE performs a multiplication followed by an addition in 64 bits floating point. Every clock cycle, each PE thus requires two 64-bits data operands. If all the PE of Figure 1 were connected in parallel to module « Master » and if the FPGA contained, for example, 16 PE, every clock cycle, 32 words of 64-bits would be downloaded from the SDRAM. At 200 Mhz it represents a need in bandwidth of more than 50 Gbytes/s which cannot be reached with current SDRAM memories.

An approach to tackle this problem, presented in [2], is a pipelined structure for the PE (see Figure 1). Except for the first PE which is directly connected to the "master", every PE is connected only to the previous PE and to the next PE in the pipeline. All the elements of matrixes A and B are thus going to circulate through every PE of the design. Each PE selects the elements they have to store according to the values of the matrix C they have to calculate and according to its position in the pipeline. Every clock cycle, the module « master » receives and transmits one element of A and one element of B to the first PE. These elements are then going to progress from PE to PE every clock cycle. The required bandwidth is now of 2 x 64-bits x 200 Mhz = 3,2 Gbytes/s what can be reached with two 64 bits SDRAM DDRII 333Mhz memories for example.

We implemented the MAC units using the optimised algorithms presented in [2,7,8] and making an intensive use of architectural features of modern FPGA like fast embedded multipliers and shift registers [7,8].

4.2 Hardware Requirements

According to the analysis presented in previous sections, to perform matrix products with good a performance, the Hardware has to hit some criteria:

- FPGA: High density FPGA containing a maximum of embedded multipliers are required to achieve good performance. High-end FPGA such as StratixII from Altera or VirtexII Pro and Virtex4 from Xilinx hit these criteria. It is also interesting to study the parallelisation of the matrix multiplication operation at a higher level by using a multi-FPGA board.
- On-board Memory: to perform multiplications of matrixes 1000x1000, the board must be able to store in its RAM 3x10^6 words of 64 bits (= 24 Mbytes). The ideal case would be to have « dual port » memories which allow to write and to read data simultaneously. Using this sort of memory, while the FPGA computes the data and reads the data in RAM, we could write the data of the operands of the next matrix product, and thus pipeline the matrix multiplications and minimize the length of the periods during which the FPGA waits for the availability of new data operands. Unfortunately, these memories are expensive and of low capacity.
- Interface: the FPGA board must have the fastest interface towards the host computer. Currently the fastest interface is the PCI Express 8x (see Figure 2).

4.3 Selected Prototyping Board Description

Currently, only few boards hitting the requirements described in the previous section are available. The FPGA boards with PCI Express 8x interface contain very often only low RAM capacity or low density FPGAs. It is then necessary to find a compromise between capacity of storage, speed of data transfers and raw computation power.

The selected board is a bi-FPGA board. It is composed of 2 FPGA StratixII 60 from Altera. The board has a PCI 64-bits 66 Mhz interface. It provides a theoretical bandwidth of 533 Mbytes/s. Each FPGA is connected to two 64 Mbytes SDRAM DDRII 333Mhz onboard memory banks with 32 bits data busses. The second FPGA is connected to a SODIMM interface which allows adding up to 1 Gbyte of SDRAM DDRII memory. The board SODIMM interface has a 64-bits data bus.

4.4 Dual FPGA Computation

The FPGA prototyping board is composed of 2 FPGA (see Figure 3). As each block of matrix C can be computed independently, matrix C is divided in 2 blocks (see Figure 4) which are computed independently by the two FPGA of the board. As shown in Figure 3 and Figure 4, the onboard SDRAM memory banks connected to FPGA 1 receives the

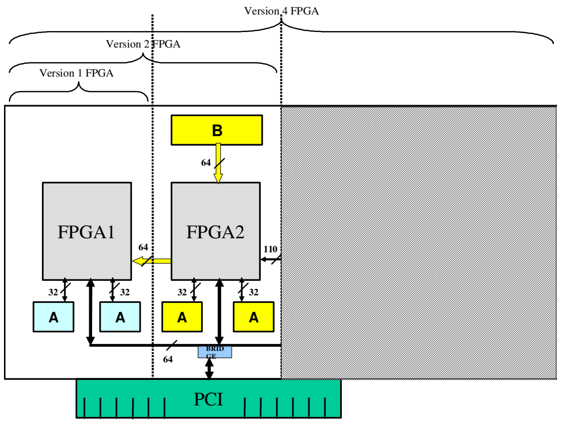

Figure 4 : FPGA board description

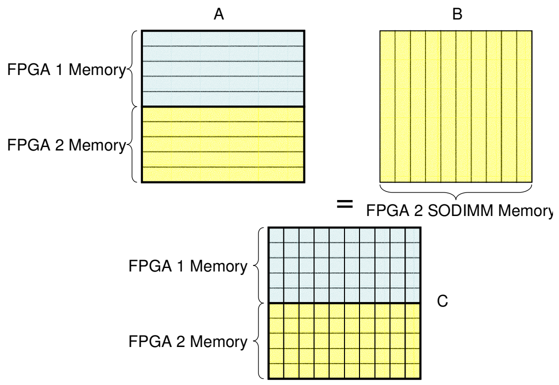

Figure 4 : FPGA Computation Repartition

upper half of matrix A and the onboard SDRAM memory banks connected to FPGA 2 receives the lower half of matrix A. As the 2 FPGA requires the complete matrix B to compute their block of matrix C, matrix B is stored in SODIMM memory bank and the data are transferred directly to FPGA 2 and are transferred to FPGA 1 through a direct FPGA 1 – FPGA 2 interconnection which implies the use of a synchronisation mechanism.

4.5 DGEMM Implementation

This section introduces the software part of the implementation of the hardware accelerated dgemm() function. As explained in the introduction of this article, the matrix multiplication on FPGA has to be performed in a transparent way for software using the BLAS library to perform this operation. The software interface of the hardware accelerated dgemm() is identical to the interface of the original dgemm() function of the BLAS library. Its functionality is thus reimplemented to compute the matrix multiplication on the FPGA board instead of the host computer processor.
The software part has several roles:

- Perform the conversions and reorganize the data operands and results to send them to the FPGA in the most appropriate order and format to maximize the "burst mode" data transfers between the SDRAM memory board and the FPGA. Indeed as explained in section 3.3, the onboard memory banks connected to the two FPGA provide 32-bits data busses. As we are working with 64-bits floating point data, in order to maximise the burst mode transfers at FPGA board level we have to split these 64-bits words in two independent 32-bits words. These words will be sent to the physically separated onboard memory banks of the FPGA board. This splitting operation is time consuming for the host processor and could be overcome by using a board providing onboard memory banks with 64-bits data busses.
- Manage data transfers between the host computer and the FPGA board (PCI DMA Transfers),
- Manage the execution of the operations performed by the FPGA. This operation is performed using registers available in the FPGA board memory and accessible by both the Software and the FPGA.

5. Results

This section introduces the measured results for the global performance of the implemented prototype. This section also compares the measured performance of the hardware accelerated implementation of BLAS dgemm() operation to the measured performance of its ATLAS [12] software implementation. We performed the tests for matrix multiplication of 2 square matrix of size 1000x1000. The FPGA board is installed in a computer based on an Intel P4 dualcore 3Ghz processor with 1 Gbyte of SDRAM DDR.

Related works [2,5,6,10] evaluate the performance of the matrix multiplication FPGA implementation only at FPGA local level by applying the theoretical relation (1) to FPGA design synthesis results. These results don't take into account external factors like data transfers or software pre- and post-processing (see section 4.5) having an important impact on the global performances of the system.

The results presented in this section are based on the global execution time of the hardware accelerated dgemm() function measured at host computer level. The measured results include:

- Software data pre- and post-processing (host computer)
- Data transfers between the host computer and the prototyping FPGA board
- FPGA data processing (matrix multiplication operation)

The measurement method we used is based on Windows system routines that count the number of host computer CPU clock cycles spent during the execution of the hardware accelerated dgemm() function. Using this method, we measured for this function an average execution time of 374,06 ms which corresponds to a computing power of **5,35 Gflops/s**.

In order to compare this result with a fully software implementation of the same operation, we implemented the ATLAS optimised software version of the "dgemm" function [12]. On our Intel P4 dualcore with 1 GByte of RAM we obtained the following total execution time for 100 matrix multiplications: 59,97 s. This result corresponds to a computing performance of **3,34 Gflops/s**.

6. Measured Performance Analysis

This chapter analyses the measured performance presented in previous section. Section 6.1 estimates the theoretical raw computation performance of FPGA devices and section 6.2 introduces the factors

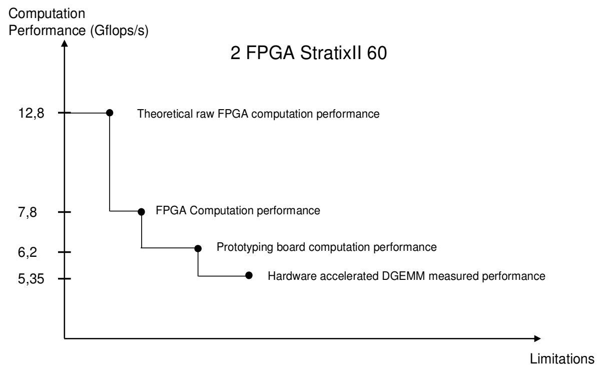

Figure 5 : Limitations impact on computation performances

impacting this raw performance result and introduces adapted solutions to minimize their impact on the global performance of the system.

6.1 Theoretical Raw FPGA Computation Performance

The theoretical raw FPGA computation performance can be calculated using expression (1). It depends mainly on total number of PE that can be placed in FPGA logic and their maximal running frequency. We estimated those two factors using a functional VHDL model of a PE we implemented. Moreover the most important component of each PE is the MAC unit. Consequently, according to routing and timing results of the VHDL implementation of a 64 bit floating point MAC unit (see Table 1) we can extrapolate theoretical raw computation performance of the implementation of matrix multiplication on FPGA.

We can see at Table 1 that the FPGA resource limiting the total number of PE is the number of available embedded multipliers. A StratixII 60 contains 144 embedded 18x18 bits multipliers and as presented in [2] one PE requires 9 embedded multipliers. It is thus possible to place 16 PE in a StratixII 60. According to the expression (1) and assuming the FPGA design will run at 200 Mhz, it is possible to evaluate the FPGA theoretical raw computation performance for one StratixII 60:
FTRCP = 2 x 16 x 2OO Mhz = 6,4 Gflops/s

As the prototyping board is composed of 2 FPGA StratixII 60 the total theoretical prototyping board raw computation performance is 12,8 Gflops/s.

6.2 Performance Limitations

Result of previous section doesn't take into account the limiting factors introduced at section 2.4. Current section analyses and quantifies their impact on global performance of the system in order to estimate this value (see Figure 5).

FPGA Design Complexity

Based on the extrapolation of routing and timing results of the implementation of one PE, we estimated in section 4.1 that 2 StratixII 60 could contain up to 32 PE and run at 200 Mhz delivering a total theoretical FPGA raw computation performance of 12,8 Gflops/s. Current section takes into account the FPGA internal hardware limitations.

To evaluate this limitation, we implemented the complete FPGA design which is composed of 16 PE, the Master module (see Figure 1) and the SDRAM DDR2 memory controllers. Due to routing complexity of such a design the limiting factor isn't the embedded multipliers anymore but now these are the available FPGA logical elements which are the limiting factor. Consequently, due to its limitation, it is not possible to place in the logic of a FPGA StratixII 60 a design composed of more than 14 PE.

According to the timing analysis of the complete FPGA design, this one will not be able to run at more than 172 Mhz. As explained in section 3.2, if the SDRAM DDR2 memory controller runs at 172 Mhz the Master and the PE will receive new valid data from the SDRAM at a frequency of approximately 140 Mhz (measured result).

Based on these elements, we can evaluate the computation performance for 2 FPGA (CPF):

CPF = 2 (FPGA) x 2 x 14 x 140 Mhz = 7,8 Gflops/s

Table 1 : 64-bits MAC Performance

	ALTERA MAC IP EP2S60-3	OUR MAC IP EP2S60-3
ALUT	2451 / 48352	1539 / 48352
MULT 18x18	9 / 144	9 / 144
Fmax	143,74 Mhz	235,32 Mhz
Pipeline Stages	14	14

As explained in section 3.1 the performance reduction caused by this limitation is hard to control and to minimize. It mainly depends on the internal structure of the FPGA. The only valid solution to significantly increase FPGA computation performance (see Figure 5) is to implement the design in a FPGA containing more logical elements and more embedded multipliers.

Data Transfers

As the prototyping FPGA board does not include « dual port » memories, the data transfers between the host computer and the FPGA board memory cannot be performed at the same time as the computation phase during which the FPGA performs continuous readings in local memory (see section 2.4). Separate transfer phases are thus added to the computation phases what will significantly reduce the global performance of the system.

To evaluate the influence of these transfer phases on the computation performance we can calculate the computation time of the matrix multiplication on FPGA and add to this value the data transfer time. For operand square matrixes 1000x1000 the computation time of the matrix multiplication at 7,8 Gflops/s is 256 ms. For this operation three 1000x1000 matrixes have to be transferred. As these matrixes are composed of 64 bits elements the total amount of data to be transferred is 24 MBytes. According to the measured performances of the PCI 64-bits 66 Mhz (370 Mbytes/s) the data transfer time is around 64,8 ms. The total latency including the computation and the data transfers is: 320,8 ms. According to this latency we are able to calculate the computation performance of the FPGA board (CPFB) [2]:

$$\text{CPFB} = \frac{2 \cdot 10^9}{0{,}3208} \text{ flops/s} = 6{,}234 \text{ Gflops/s}$$

This performance limitation could be overcome by adding more physically separated memory banks. Simply switching between the available memory banks, we are able to perform the memory transfers and the computations at the same time. The computation performance of the FPGA board (CPFB) could then be close to the computation performance of the FPGA (CPF): 7,8 Gflops/s.

Background Software Processes

As described in section 4.5 the host computer processor has to perform specific operations like data rearrangement, data conversion and data transfer management. These data rearrangement operations are time consuming for the host computer. As the FPGA design requires rearranged data in order to maximise its own "burst mode" memory readings, it has to wait that the host computer finished the rearrangement operations (see "Conv" steps at Figure).

The running steps shown at Figure :

1. The host computer ("µP" line) performs the rearrangement and the conversion of the data operands : "Conv A1", "Conv A2" and "Conv B" steps. During this period the FPGA board doesn't perform any computation.
2. "Wr 1 & 2": data transfers between the host computer and the FPGA board. As shown at Figure 6, this operation involves the central processor and the FPGA.
3. "Matrix Mult FPGA 1" and "Matrix Mult FPGA 2": the FPGA performs the matrix multiplication operation.
4. "Rd 1" and "Rd 2": The result of the matrix multiplication is transferred from the FPGA board to the Host computer.
5. "Conv C1" and "Conv C2": rearrangement and the conversion of the resulting matrix C.

By implementing this sequence, we were able to measure the performance of the global system by measuring the execution time of 10 runs of the hardware accelerated function BLAS "dgemm". According to the number of operations to be performed to multiply 2 matrixes 1000x1000 (2 Gflops), the global performances in Gflops/s can be easily calculated for one execution of the matrix multiplication. If we consider the timing representation of Figure , we notice that the complete operation sequence takes 487,2 ms. This latency corresponds to the execution of 2 Gflops. The Hardware Accelerated DGEMM Measured Performances (HADMP) is:

$$\text{HADMP} = \frac{2}{487{,}2} \cdot 1000 = 4{,}1 \text{ Gflops/s}$$

We notice at Figure that during the FPGA computation phase "Matrix Mult FPGA 1" the processor of the host computer doesn't perform any operation related to the "dgemm" function and during the conversion phases "Conv A1", "Conv A2", "Conv B" and "Conv C2" the two FPGA don't perform any computation. If the data of the next computation (iteration J+1) are available at iteration J, it is possible to pipeline the conversion phases and the FPGA computation in order to achieve better performances. We can see on Figure that the pipelined version of

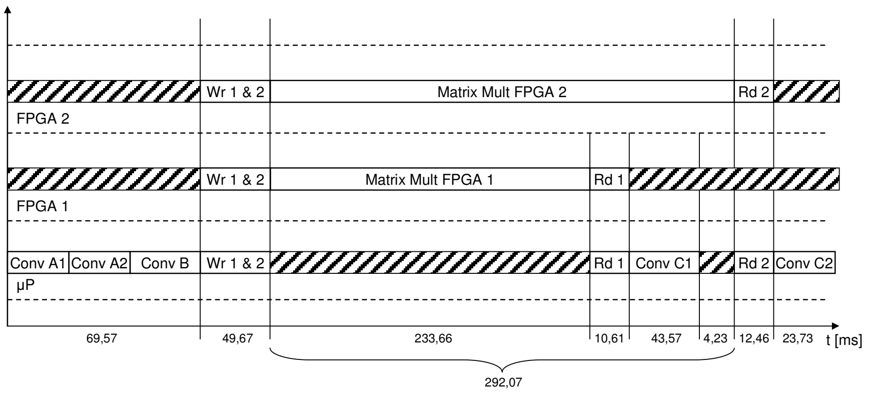

Figure 6 : Running steps of the Hardware Accelerated DGEMM

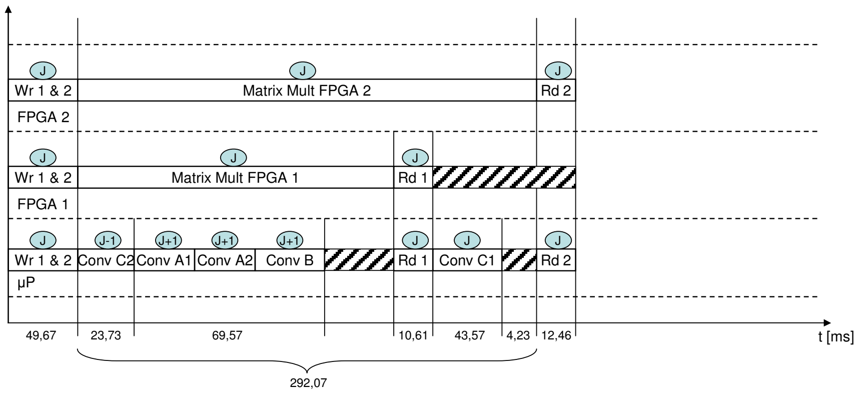

Figure 7 : Pipelined Hardware Accelerated DGEMM

the hardware accelerated “dgemm” function has a total latency of:

487,2 ms – 69,57 ms – 43,57 ms = 374,06 ms

which corresponds to a computing power of **5,35 Gflops/s**.

Analysing the results of pipelined version of the accelerated “dgemm” function presented at Figure 8, we notice that using a board with more FPGA or a board with higher density FPGA will lead to a pipeline stall. Indeed, using higher density FPGA will allow to place more PE in the FPGA and will thus reduce “Matrix Mult FPGA 1” and “Matrix Mult FPGA 2” phases. If these 2 phases are too short, the processor of the host computer cannot sustain the data operand need of the FPGA board. The FPGA will have to wait that the host processor finished the data rearrangement and conversion. To calculate the maximal achievable performance without pipeline stall, we have to calculate the processing time of the host computer during “Matrix Mult FPGA 2” phase: 147,14 ms. This value corresponds to the lower limit of “Matrix Mult FPGA 2” duration. Knowing this lower limit we can evaluate the minimal global latency by adding to 147,14 ms the latency of “Wr 1 & 2” and “Rd 2” phases. This minimal global latency corresponds to a maximal achievable performance of:

$$\text{Max Performance} = \frac{2}{209{,}27} \cdot 1000 = \mathbf{9{,}55\ Gflops/s}$$

7. Adapted FPGA Board Requirements

Considering the analysis of the previous sections, we can build the requirements of an ideal FPGA board for high performance computing applications. This board has to respect the following criteria:

- It has to contain one or several high density FPGA containing a large number of embedded multipliers.
- It has to provide at least 4 independent memory banks with 64-bits data busses for each FPGA in order to parallelize the computation phases and the transfer phases.
- It has to provide a communication interface to the host computer with the largest bandwidth possible. E.g. PCI Express 8x.

8. Conclusion

In this paper, we introduce the implementation of a FPGA-based hardware accelerator for BLAS library. We first describe a theoretical analysis and introduce the main possible limitations. We show that the limiting factors come mainly from sources external to the FPGA. Based on this analysis we choose to implement BLAS matrix multiplication operation "dgemm", using optimised algorithms and optimised FPGA architecture. Using a dual-FPGA board, selected according to preliminary requirements described in the theoretical analysis, we have developed a fully functional prototype. Based on the theoretical analysis of performance and limitations, and on the prototype measured performance, we highlight the three main limitation sources. We analyse their impact on the system performance and introduce possible solutions to minimise it. We show that our prototype is able to achieve a computation performance of 5,35 Gflops/s which is 60% higher than the measured performance of a high-end processor running the ATLAS optimised version of the BLAS library. Based on all these results we finally introduce the requirements of an adapted FPGA board allowing the highest performance possible.

9. Acknowledgments

The work described in this paper is based on the results of the CANAPE project ("Calcul Numérique sur Architecture Programmable") partially funded by Belgian Walloon region.

10. References

[1] Dr. Olaf Storaasli, "Reconfigurable Scalable Computing (RSC)", *Computational Structures and Materials & Electronics Systems*, NASA Langley Research Center

[2] Yong Dou, S. Vassiliadis, G. K. Kuzmanov and G. N. Gaydadjiev, "64-bit Floating-Point FPGA Matrix Multiplication", *FPGA'05,* Monterey, California, USA, February 20–22, 2005

[3] Keith Underwood, "FPGAs vs. CPUs: Trends in Peak FloatingPoint Performance", *FPGA'04,* Monterey, California, USA, February 22-24, 2004

[4] M. deLorimier, "FloatingPointSparse MatrixVector Multiply for FPGAs", *FPGA'05,* Monterey, California, USA, February 20–22, 2005

[5] K. D. Underwood and K. S. Hemmert, "Closing the gap: CPU and FPGA Trends in sustainable floating-point BLAS performance", In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing machines (FCCM 2004)*, April 2004.

[6] L. Zhuo and V. K. Prasanna, "Scalable and Modular Algorithms for Floating-Point Matrix Multiplication on FPGAs" In *Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS'04)*, April 2004.

[7] E. Roesler, B. Nelson, "Novel Optimizations for Hardware Floating-Point Units in a Modern FPGA Architecture", *FPL'2002*, Aug/Sep 2002.

[8] J-L. Beuchat, A. Tisserand, "Small Multiplier-based Multiplication and Division Operators for Virtex-II Devices", research report of the Laboratoire de l'Informatique du Parallélisme, July 2002

[9] Ralf Gruber, Pieter Volgers, Alessandro De Vita, Massimiliano Stengel and Trach-Minh Tran, "Parameterisation to tailor commodity clusters to applications", *Future Generation Computer Systems, Volume 19, Issue 1,* January 2003, Pages 111-120

[10] G. Govindu, L. Zhuo, S. Choi, and V. Prasanna, "Analysis of High-performance Floating-point Arithmetic on FPGAs", In Proceedings of the 18th *International Parallel and Distributed Processing Symposium (IPDPS'04)*, pages 149-156, April 2004.

[11] http://www.netlib.org/blas

[12] http://math-atlas.sourceforge.net/

[13] http://www.openfpga.org