UCLouvain

epl

École polytechnique de Louvain

Vectorisation of Robotran Equations through FPGA

Author: **Maxence van Lede**
Supervisors: **Paul Fisette, Jean-Didier Legat**
Readers: **Nicolas Docquier, David Bol**
Academic year 2021–2022
Master [120] in Electro-mechanical Engineering

Abstract

This thesis aims at characterising the potential of parallel computation for multibody systems and the consequential reduction of simulation time, understanding the limitations, boundary conditions which we are currently facing and the future improvements that could take place in order to push further away these boundaries.

Parallel computing in multibody dynamics is pushed by multiple causes: the increasing complexity of multibody systems, the need for real-time applications, the use of parallel computing in other research areas, the democratisation of multi-core CPU.

During this thesis we develop a dedicated parallel architecture on a FPGA in order to implement and test a multibody system. This implementation allows to characterise the limitations, challenges and benefits of such an architecture.

Given the architecture we develop and in parallel to this, we create an algorithm that is able to handle the multibody systems equations to extract the fine-grain parallelism of these equations.

As multibody dynamic systems vary both in size and in morphological characteristics, we make a study case of multibody systems with the variation of these characteristics. These are on size, system with 23-, 50-, 70-, 100- and 130-dof and on topology, with either a serial, a semi-serial or a parallel multibody system. We will use these in order to compare their potential differences and similarities.

To further increase the extraction of the fine-grain parallelism, we propose three methods in order to modify the equations, exploiting the following two concepts: the atomisation into a binary expression tree and the critical path of the equations.

Acknowledgements

I would like to start by thanking my two supervisors, Jean-Didier Legat and Paul Fisette, for their continued support during my thesis. The advice and guidance they provided, in addition to the interest for the topic they brought, which was key to this work.
Specifically to Jean-Didier Legat for the support and advice on the design and prototyping of the architecture on the FPGA and to Paul Fisette for providing the systems we have been analysing. Their respective expertise was valuable to shape to my research and to send me in the right direction.

I also would like to thank Jean for the daily discussions we had on the issues I faced during my work and my parents Christelle and Tanguy for the remarks on this text and the support throughout the difficulties.

Contents

Abstract **ii**

Acknowledgment **iv**

Introduction **xii**

State of the Art **xiv**

1 FPGA Model **1**
1.1 Proposed Architecture . . . 3
1.1.1 Technological Advances . . . 3
1.1.2 Constrains & Solutions . . . 4
1.1.3 Interconnections . . . 7
1.2 On Pipelining . . . 10
1.2.1 Pipeline Principle . . . 10
1.2.2 Altera IP blocks . . . 12
1.2.3 Pipeline Study . . . 14
1.3 Implementation . . . 16
1.3.1 Resources usage . . . 16
1.3.2 Limitations & Future Perspectives . . . 18

2 Vectorisation of Equations **21**
2.1 Atomisation of Equations . . . 23
2.2 Indexing of Equations . . . 24
2.2.1 ALAP and ASAP Attributes . . . 25
2.2.2 ASAPCycle and ALAPCycle Attributes . . . 26
2.3 Placement of Equations . . . 27
2.3.1 Classical vs Mobility List Scheduling . . . 30
2.4 Leads on improving cycles . . . 33
2.4.1 Zero Removal . . . 34
2.4.2 Tree Swapping . . . 35

2.4.3 Tree Balancing 36
2.4.4 Effectiveness 37

3 Robotran Equations 39
3.1 Topological Parallelism 41
3.2 Study Cases 43
3.3 Minimal and effective Cycle Time 45
3.4 Parallelisation Rate 50
3.5 Projection 54
3.6 Occupation Rate of PE 55
3.6.1 Serial Topological Systems 57
3.6.2 Semi-Serie Topological Systems 60
3.6.3 Parallel Topological Systems 63
3.6.4 Critical Path Reduction 66

Conclusion 71
Synthesis 71
Limitations & Perspectives 72

A FPGA 81
A.1 ADD block 82
A.2 MULTI block 83
A.3 SignalTap 84

List of Figures

1.1 Cyclone V Architecture . . . 2
1.2 Read and Write Cycles . . . 6
1.3 Dataflow Block Architecture . . . 6
1.4 Static Interconnections : A) Bus, B) Hypercube, (C) 2D-mesh, (D) 2D-torus . . . 7
1.5 Dynamic Interconnections: A) Crossbar Switch, B) Omega Switch . . . 8
1.6 5-step classical ARM CPU[37] . . . 10
1.7 5-Stage Pipelined ARM CPU[37] . . . 11
1.8 5-Stage Pipelined ARM CPU - Hazard[37] . . . 11
1.9 Maximal frequency by latency for ADD and MULTI PE . . . 13
1.10 Number of Cycles depending on the number of pipeline stages . . . 14
1.11 Computing time depending on the number of pipeline stages . . . 14
1.12 Parallelisation rate depending on the number of pipeline stages . . . 15

2.1 Treelike structure of an equation . . . 24
2.2 ASAP and ALAP indices for dependency tree . . . 25
2.3 ASAPCYCLE and ALAPCYCLE indices for dependency tree . . . 26
2.4 Memory associated with PE for the two placement methods - 23-dof - railway-bogie . . . 29
2.5 Memory associated with Buffer for the two placement methods - 23-dof railway-bogie . . . 29
2.6 Memory associated with PE for the two placement methods - 70-dof system . . . 30
2.7 Memory associated with Buffer for the two placement methods - 70-dof system . . . 30
2.8 Dependency Tree - Zero Removal . . . 34
2.9 Dependency Tree - Tree Swapping . . . 35
2.10 Binary Expression Tree[3] . . . 36
2.11 Dependency Tree - Tree balancing . . . 37

3.1 Multibody Applications . . . 40

3.2 System topology . . . 41
3.3 Example of closed-loop multibody systems . . . 42
3.4 Cycletime for the 23 dof . . . 47
3.5 Cycletime for the 50 dof . . . 47
3.6 Cycletime for the 70 dof . . . 48
3.7 Cycletime for the 100 dof . . . 48
3.8 Cycletime for the 130 dof . . . 49
3.9 Parallelisation Rate 23-dof . . . 51
3.10 Parallelisation Rate 50-dof . . . 51
3.11 Parallelisation Rate 70-dof . . . 52
3.12 Parallelisation Rate 100-dof . . . 52
3.13 Parallelisation Rate 130-dof . . . 53
3.14 Number Y given topology and size . . . 54
3.15 Occupation 23-dof railway-bogie . . . 56
3.16 Occupation 70-dof . . . 56
3.17 Occupation 23-dof serie . . . 57
3.18 Occupation 50-dof serie . . . 58
3.19 Occupation 70-dof serie . . . 58
3.20 Occupation 100-dof serie . . . 59
3.21 Occupation 130-dof serie . . . 59
3.22 Occupation 23-dof semi-serie . . . 60
3.23 Occupation 50-dof semi-serie . . . 61
3.24 Occupation 70-dof semi-serie . . . 61
3.25 Occupation 100-dof semi-serie . . . 62
3.26 Occupation 130-dof semi-serie . . . 62
3.27 Occupation 23-dof parallel . . . 63
3.28 Occupation 50-dof parallel . . . 64
3.29 Occupation 70-dof parallel . . . 64
3.30 Occupation 100-dof parallel . . . 65
3.31 Occupation 130-dof parallel . . . 65
3.32 Occupation 23-dof - railway-bogie - lowered critical path . . . 66
3.33 Occupation 23-dof - serie - lowered critical path . . . 67
3.34 Occupation 23-dof - semi-serie - lowered critical path . . . 67
3.35 Occupation 23-dof - parallel - lowered critical path . . . 68
3.36 Occupation 50-dof - serie - lowered critical path . . . 68
3.37 Occupation 50-dof - semi-serie - lowered critical path . . . 69
3.38 Occupation 50-dof - parallel - lowered critical path . . . 69

A.1 Detailed ADD Block Architecture . . . 82
A.2 Detailed MULTI Block Architecture . . . 83
A.3 SignalTap Screenshot :expected vs results . . . 84

List of Tables

1.1 Properties of Addition-Subtraction IP blocks 12
1.2 Properties of Multiplication IP blocks 12
1.3 Coupling latency of subtraction/addition and multiplication 12
1.4 Resources Usage - DE10-Nano : 8 PE (4 ADD & 4 MULTI) 18

2.1 Preferential Placement 28
2.2 Number of Cycles for classical Alternative Classification 31

3.1 Study Cases Characteristics 44

Introduction

Multibody dynamics is a scientific area that is concerned with the kinematic and dynamic study of multibody systems (MBS). These systems are characterized as a set of bodies linked together by joints which are present in our everyday lives: road- and railway vehicles, biomechanics, robots, etc. Research in this field focuses both in the analysis of the multibody systems themselves and the production of formalisms that are accurately able to simulate multibody systems.
In recent years, research has extended to the development of physical coupling between the multibody systems and other physical domains such as flexible-joints, granular mechanics, fluid-solid interactions, flexible bodies, etc.

As the complexity of the multibody systems grows, the need for performing software increases. On software, a distinction is first made between the numerical approach (e.g. Adams-MSC or Samcef-Mecano software) and the symbolic approach (e.g. Neweul-M2, Maplesoft, ROBOTRAN software) [1].
This second approach uses a symbolic generator that is able to apply arithmetic and trigonometric simplifications such that the number of equations produced to describe the multibody system is very small. It allows the software to produce systems of higher complexity while keeping the number of equations and therefore the computing time low.
Most of the research that has been done recently is focused on the production and improvement of formalisms and software that is able to exploit the topological parallelism of multibody systems[2] and the implementation of these for parallel computations. These can be made on homogeneous platform such as CPU or GPU, or on heterogeneous platform as CPU/GPU, or on platform such as FPGA with a dedicated architecture.

At the multibody team of UCLouvain, we are still looking for ways to decrease the computing time of the multibody simulations. During the earlier days of Robotran, Tony Postiau, a doctorate at the team studied the possibilities given at that time for parallel computations [3], both on FPGA and on OpenMP.
Since then the technology has improved and will continue his work, specifically on

the vectorization with the fine-grain parallelism.

The structure of this thesis is organized as follow: after a state of the art on the parallel computation taking place in multibody dynamics and closely related fields, we will propose a FPGA architecture that will allow to test the following steps and give real-world constrains regarding a performing computation on current-day FPGA technology in Chapter 1. Chapter 2 will explain how we are able to extract the fine-grain parallelism of the Robotran equations given the constrains of the FPGA, thanks to a placement algorithm, additionally, we will present improvement to increase the potential of parallelisation. Chapter 3 will take a set of multibody systems which will be going through the placement algorithm, we will analyse the different systems given their size, including some systems we have improved manually using one of the method we presented. The concluding section will point out the findings and the potential perspectives of this research.

State of the Art

Multibody dynamics is a scientific area that did originate from the work of Newton and Euler which did respectively describe the free particle in 1686 and the rigid body in 1776 [4]. This heritage is characterized by the formulation of the Newton-Euler equations used in multibody dynamics.
For a number of years, the study of multibody dynamics was limited to the study of mechanical systems for the development of new devices or the improvement of existing ones. The limited access to performing computers meant that the simulations couldn't reach real-time computations, but were limited to computer simulations. The development of more performing hardware means, both in processors and in memory allowed to improve the computing time of the multibody simulations. Combined to this improvement in computational tools, researchers did look into the production of compact formalisms, such as the symbolic generation [5], recursive methods or order-N methods, the use of formalisms to be able to produce accurately parallel formalism to exploit the most advanced computer architecture.

The research of today continues to study the optimisation of design and control devices, applications in biomechanics, robotics and vehicle dynamics, real-time simulation including haptic applications [1] to only name a few [6].
Multibody dynamics are also being linked together to have more real-world applications with other physical fields [7], driven by the need of the industry and the possibilities offered by multibody simulation concepts[8]. In some of these coupling applications, researchers have been able to produce parallel computation methods, mostly in particles dynamics such with granular contact, fluid-solid contact or terrain modules, where we have a spacial parallelism leverage [9, 10].
Regarding the multibody dynamics of rigid multibody systems, research on parallel computation is focused on the development of formalisms able to extract the parallel topology of the multibody system [11, 2, 12]. Effectively reducing the computation time. This has been done a number of times on various practical cases such as a the study on a N-bar linkage [13], a 4-bar linkage and 3-D steering system [14] or the computation of 1/4 car and a full car [15].

Methods to exploit the possibilities of parallel computing have been developed over the years, these include specifically for rigid multibody systems: Method of flexible joints combined with Heterogeneous Multiscale Methods (HMM) [16], Modified State Space Methods (MSS) for flexible bodies [17], divide and conquer methods [2, 12, 11, 18], and various matrix decomposition methods [19, 20]. The method of flexible joints combined with HMM is used to decouple the equations while the HMM is correcting the high frequency caused by this method. MSSs for flexible natural coordinates leads to constant mass matrix and linear Jacobian matrix, where these matrices are sparse and well structured. As a result when MSSs are used in parallel form, the complexity of computation is linearly dependent on the size of the system despite the occurrence of kinematic loops. Divide and conquer methods divide the whole multibody system into subsystems, which are integrated on several processors [2, 12, 11, 18]. For serial multibody systems, they present the same parallelisation potential, because they will still be constrained by the kinematic chains. However for parallel multibody systems, the computation time follows a $\Omega(log(N))$, where N is the number of degrees of freedom.

Regarding the implementation of those parallel formalisms, we have seen a number of studies on both homogeneous hardware such as CPU or GPU. On CPU, these are made through well established parallel standards such as OpenMP and MPI and take into account multithreading. On the GPU, most of these implementations take advantage of a spacial divide and conquer algorithm to compute particles dynamics such as granular contact, fluid-solid contact and interaction or terrain effect on a multibody systems [10, 9].
Some research has also been done on other hardware such as FPGA and embedded processors. For example, the computation of a quarter of a car followed the full car on a heterogeneous platform containing both a FPGA and an ARM processor [15], the use of a ARM processor for predictive controllers and state observer of a 4-bar linkage and a 3D-steering[14]. In a closely linked field, there has also been the usage of FPGA for molecule dynamics [21].
As we have multiple kind of architectures at our disposal and still evolving, the debate arose regarding the architecture the most adapted to each application, the challenges and perspectives [22].

FPGA have been used in recent years for numerous applications as they provide a compromise between the dedicated architecture of ASIC and the polyvalence of the CPU/GPU. They are now used to implement dedicated parallel architecture such as in the encryption and decryption [23], for data analysis [24], soft-error and fault-tolerant design for aerospace applications [25], artificial neural networks for deep learning purposes [26], genetic algorithms [27], particle swarm [28].

Chapter 1

FPGA Model

Field programmable gate arrays (FPGAs) are a type of reconfigurable integrated circuit which are made out of basic building blocks linked via configurable routing interconnects. This allows to implement a particular hardware system, in the same way an application specific integrated circuit (ASIC) could.
FPGAs are gaining a larger place alongside the classical general-purpose CPU/GPU fixed hardware system and the purpose-specific ASIC.
While ASIC generally outperforms FPGA on power consumption, performance and size, they need high amounts of time and money to develop. Compared to CPU/GPU, FPGA can be used on several very different applications and can be used on prototyping as they are re-programmable. FPGA are a trade-off between the advantages and disadvantages of ASIC and CPU/GPU.

FPGA can be constituted of different kinds of blocks from a family to another in the same way any other electronic hardware could. But is generally made out of the following three elements [29];

- Adaptive Logic Module (ALM) : this is a volatile module that can be used for different functions being made out of a combination of look-up tables, multiplexer and registers.
  The design of the ALM varies lightly between families. (Adaptive) Look-up tables (LUT/ALUT) are representing a N input boolean function to implement the functions. Registers are dedicated for improving time-closure. The multiplexer can be used to choose between those two functions. As the ALM can vary from generation to generation and from manufacturer to manufacturer, we use a standardised unit called a logic element constituted of a 4-input LUT, a flip-flop and a register.

- Digital Signal Processing (DSP) : Used for floating-point and fixed-point calculations : additions, multiplications or both. They can also be used

for multiply and accumulate purposes (MAC). These blocks are made to be used in specific applications such as video signal processing, Finite Impulse Reponse (FIR) Filter and pipelined/high-frequency computations.

- Embedded Memory Blocks : These are specialized blocks of a specific memory size, M10K and M20K are respectively two blocs of 10 and 20 Mb of memory which are used specifically as ROM or RAM in the design.

Alongside the FPGA, some boards contain a Hard processor system (HPS), including an embedded processor such as a ARM Cortex, to produce a System On Chip (SOC). This allows the user to have at the same time the hardware possibilities from the FPGA and software capacities to execute applications through an OS (Linux) or Bare Metal.
Effectively extending the possibilities of applications while replacing the resource usage of a potential embedded microprocessor, which will also be faster than using the FPGA blocks.
The Cyclone V architecture with the different building blocks is illustrated in figure 1.1:

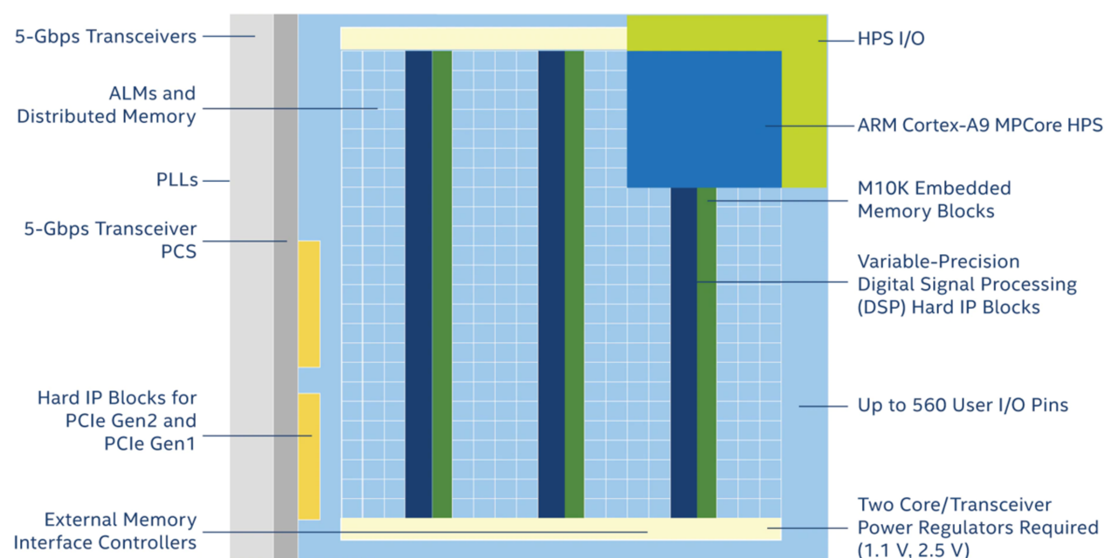


Figure 1.1: Cyclone V Architecture

1.1 Proposed Architecture

In order to prove the feasibility and the interest of a vectorisation of Robotran equations, we need to implement a dedicated architecture which can fully exploit the fine-grain parallelism that the equations will contain. A FPGA allows to develop such a specific architecture. For our implementation, we have at our disposal a DE10-Nano containing a Cyclone V FPGA and an ARM processor, we will here only rely on the FPGA.
When looking very closely at the Robotran equations, we notice that these equations use three basic arithmetic operations (+, - , *) and two trigonometric functions (sinus and cosinus). For the purpose of this thesis, we decided to express the trigonometric functions with their Taylor series of order 1 equivalent.
This leaves us with the three basic arithmetic operations. Another possibility could be to compute the sinus before computing the direct dynamics and add it to the initial variables.
We will group the operations of addition and subtraction together, as the two operations are sensitive to the sign of the terms.
Therefore, we can make two specialised processing elements (PE), which are either making the operations of addition and subtraction, and the multiplication, which we will simply call *ADD* and *MULTI*.

1.1.1 Technological Advances

Previously, the architecture used by Postiau in [3] suffered from FPGA size limitations. The system was made out of 4 FPGA FLEX 10K100 that were mounted on a PCI, implementation which is further explained in [30] and in [31]. This implementation contained the three operations using a single PE on each FPGA, because of the size of the FPGA. The PE were connected through a mesh-like structure between the 4 FPGA's.

However, almost 20 years later, the technology has radically changed. At the time Tony Postiau did his PhD, the FPGA used for implementing the architecture, the FPGA FLEX 10K100, contained 4,992 LE[3].
The FPGA we will use, the DE10-Nano of the Cycle V family, contains 110 000 LE and 112 DSP from 224 18 x 19 multipliers[32].
Another way of looking to this technological advance is by looking at the most high-end FPGA proposed back then and today. In 2004, the Stratix II family could offer up to 180 000 LE, 96 DSP or 392 18 x 18 multipliers compared to the succession of the Stratix family as Stratix 10, which can offer up to 2 Million LE, 3,456 DSP or 6,912 18 x 19 multipliers. These increases allow to effectively

implement the dedicated architecture to test our small implementation, validate our design and eventually in the future implement architecture with a higher amount of PE [3, 33].

1.1.2 Constrains & Solutions

The architecture will separate the two operations to be made on two different blocks as each of these blocks will contain a single type of PE. These two blocks will contain either a PE that will be able to do 32-bits floating-point addition-subtraction with the help of a bit selector for the operation or a PE that will be able to do 32-bits floating-point multiplication.
Additionally, we will need one or more memories, both for storing the initial and intermediate values but also to drive the memory and the PE's.
We will not impose any restriction on the design of the overall but we will consider that the final block or blocks containing both the PE and the memories will be likewise with a difference due to the wire that is needed for selecting the operation in the case of the ADD PE.

In an ideal world, we would have a shared memory scheme where all the results of each cycle could be written in the memory and the values that are needed for the PE could be read from that single shared memory. However this is not possible in our case as we are limited on the DE10-Nano to two read or write, or a read and a write per cycle [34, 35].
This is specifically the case for our DE10-Nano which belongs to the Cyclone V family, but more recent families could offer a higher number of ports to write and/or read per cycle, such as the Stratix 10 family that allows a 4-port memory in the M20K memory block[33].
We haven't talked yet about the frequency, but we consider at this moment that we have a single frequency over the whole design.
We could implement a shared memory scheme using two frequencies on the design, one for the shared memory and one for the PE's. We could indeed decide to write down all the results before having the start of a new cycle in the PE's. As a consequence, we would have multiple read and write cycles in the memory while the PE's have a single cycle.
It might present itself as a valid solution for a low number of PE, but as the number of PE grows the scalability of this solution is less valid.
Indeed, the frequency of the memory is limited to a certain ceiling. The frequency of the PE would need to be several times lower than the maximal frequency of the memory. This frequency would be equal to the frequency of the memory divided by the number of read and write cycles we would need.

Therefore we looked into a different implementation for the memory.

Another solution is to use a distributed memory scheme where each PE has his own memory. This memory is used for storing for a certain amount of time intermediate results we will need in future computations for each PE. These intermediate values will either be obtained directly after they have been computed or transmitted later by the PE. The memory will also be used for reading the variables we need for the next cycle.
This solution however still presents a downside due to the amount of results we could have to write down at a particular cycle. Indeed, at a cycle, we could need most if not all of the results produced. Therefore, we return to the issue we faced with the shared memory scheme. Quite intuitively, we know that at worst we would have to read the two variables at each cycle and keep up writing at least two variables per cycle in order to be able to keep feeding the PE with operations. The extra variables that we would need in the future can be placed on hold until we find a cycle where we have an empty writing slot.

As we choose to implement the second solution, we need a memory that is associated to the PE that will contain the initial data and the intermediate values that will be needed later in the processing. This memory, called *PE Memory* is driven by instructions contained in another memory, called *Instructions Memory*, that contains the instructions for the management of the memory and the selection bit in the case of a PE for addition-subtraction.
We need a method to eventually send the needed variables to the PE at a certain cycle after they are computed. For that purpose we use a buffer structure that write and read the results. This Buffer structure is made out of two elements; first a memory, called *Buffer Memory*, to save the results and be able to read them at the right time and a memory, called *Buffer Instructions*, which contains the instructions in order to drive the Buffer Memory.

We still face one issue regarding our PE memory. As we said earlier, the memory is only able to do for each cycle either two read or two write, or one of each. We need however to make, in the worst-case scenario, two reads and two writes for each cycle, as we need to read the two values for the computation and to write the two values to be used in future computations.
In the shared memory scheme, we proposed the two frequencies in the design, one for the PE and one for the PE Memory. We use the same principle to solve this issue. We divide the cycle of the PE into two cycles for the memory: one cycle to read and one to write. Therefore here we will have a frequency on the PE Memory that will be twice the frequency of the PE. The clock on the PE will be defined

using a register and changing at each rising edge of the clock on the PE Memory.

This workaround is illustrated in figure 1.2, using a write-enable, we will be able to read a first address and write on a second using the same port in the memory.

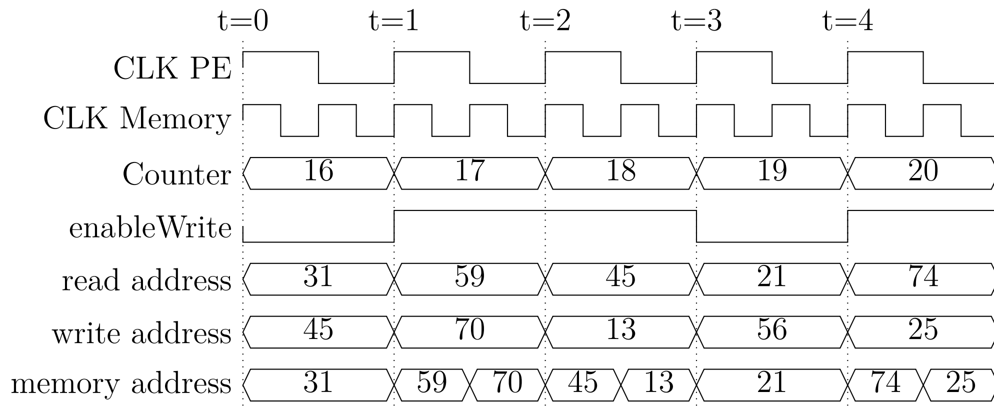


Figure 1.2: Read and Write Cycles

This structure, which we will call block is illustrated in figure 1.3 with a dataflow model. For more details, we added a detailed schematic in the appendix, with the different building parts and the connections, respectively with the ADD PE and the MULTI PE in figure A.1 and A.2.

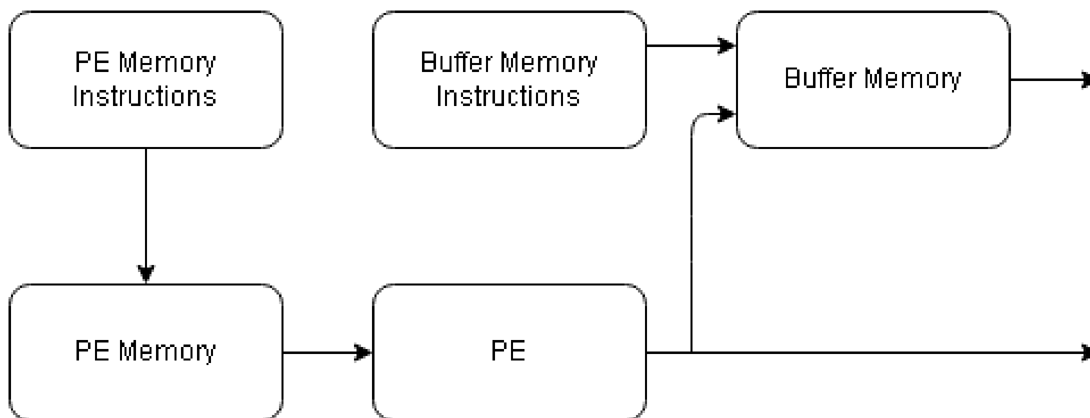


Figure 1.3: Dataflow Block Architecture

1.1.3 Interconnections

Once we created the two blocks needed for computing our equations, we have to look into the interconnections to link our blocks. Some of them are described in [36], where interconnections are mainly proposed for parallel computations where PE need variables that are being computed by other PE's.
These can be either static interconnections such as shown on figure 1.4, the the latency is the maximal number of cycles we have between two nodes. They can be made out of dynamic interconnections, as shown on figure 1.5.

For static intersections, a decrease of the latency between two nodes will be at the cost of the amount of connections/links of the interconnection. This will also increase the amount of links each node will have to handle.
On figure 1.4 A, the bus interconnection is a simple and cost-effective design

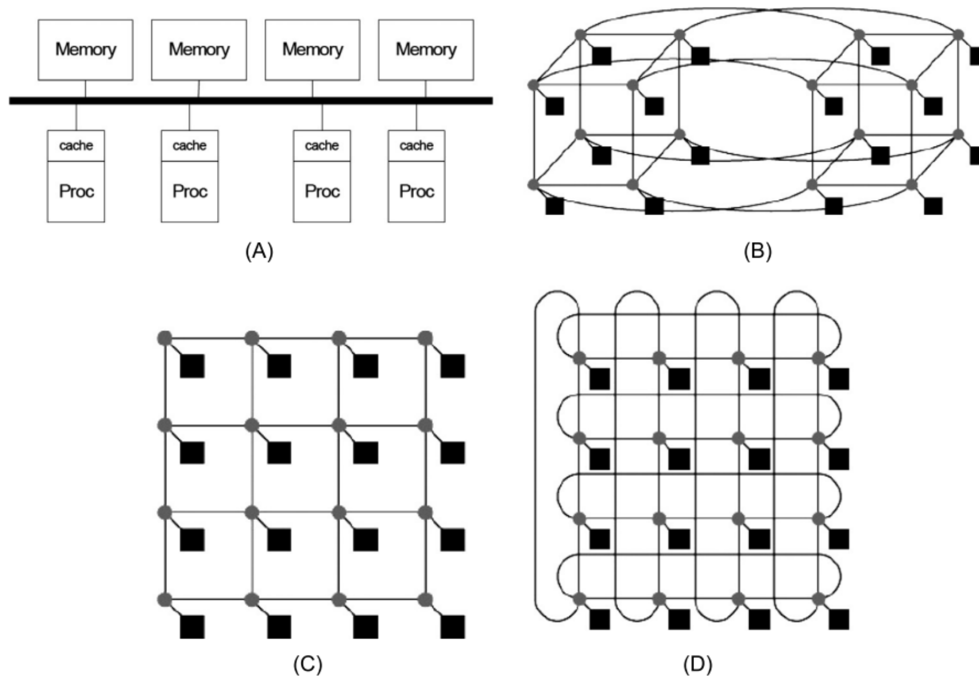


Figure 1.4: Static Interconnections : A) Bus, B) Hypercube, (C) 2D-mesh, (D) 2D-torus

but it is not possible to scale it to a high number of PE as only one PE can send through the bus at each cycle while the others are writing down the data that is being transmitted. For a number N of PE the cost is $\Omega(1)$ and the latency is $\Omega(N)$.

On figure 1.4 B, the hypercube of order n contains 2^n PE, where each node is connected to n other nodes. This interconnection reduces even more the latency as the nodes increases. For a number N of PE, the cost is $\Omega(2^n \cdot n)$ and the latency is $\Omega(n)$.
On figure 1.4 C, the 2D-Mesh is a mesh with n^2 PE which connects the adjacent nodes, where n is the size of each row and column. This interconnection allows for a better communication between the PE to the cost of more resources usage but shows difference between the middle of the mesh and the sides of it regarding both number of connections and maximal latency. For a number N of PE the cost is $\Omega(N)$ and the maximal latency is $\Omega(\sqrt{N})$.
On figure 1.4 D, the 2D-Torus is an improved mesh with n^2 PE which connects the adjacent nodes and the opposing nodes. This interconnection corrects the increased latency for the nodes on the sides for a 2D-Mesh. For a number N of PE, the cost is $\Omega(N^2)$ and the maximal latency is $\Omega(N)$.

For dynamic interconnections, we also have an optimum between the latency and the number of connections for different architectures. As these interconnections use switches, we need to precise which interconnections are being closed and opened at each clock-cycle.
On figure 1.5 A, the Crossbar Switch is an architecture with an assembly of

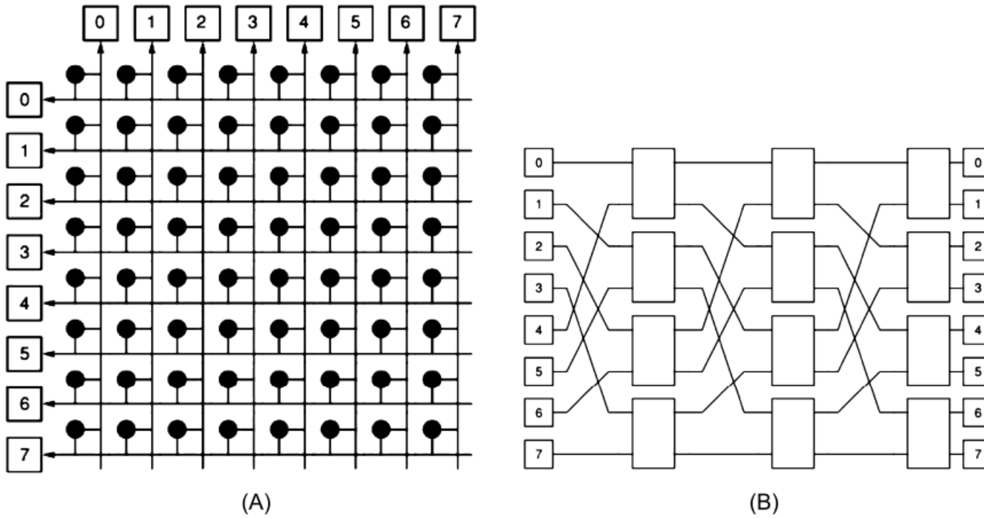


Figure 1.5: Dynamic Interconnections: A) Crossbar Switch, B) Omega Switch

individual switches organized in a matrix between a set of N inputs and a set of M outputs. We will therefore call it a M x N Crossbar Switch. This allows the

interconnections to have multiple times a single input being connected to several outputs, but the user needs to have a single connection for each output, otherwise the output will be undetermined. The cost is $\Omega(m \cdot n)$ and the number of steps is $\Omega(1)$. We need to precise that as the number of inputs and outputs increases, the frequency at which this interconnection works will reduce as the number of elements the signal needs to go through will increase.
On figure 1.5 B, the Omega Switch is an evolution of the Crossbar Switch. In fact we use multiple smaller Crossbars, which are then used in the same pattern than previously. Here for example, we create our interconnection between our 8x8 matrix for a 3-stage Omega Switch with 2x2 Crossbars. We have a cost of $\Omega(N \cdot log(N))$ and a latency of $\Omega(log(N))$.

1.2 On Pipelining

1.2.1 Pipeline Principle

The design of modern CPU and other architectures use multi-stage pipelines. A classical RISC ARM CPU, for example, uses five consecutive steps : Instruction Fetch, Instruction Decode, Execute, Memory Access, Register Write Back. These can be pipelined into a 5-stage pipeline, where each stage represents a step of the ARM CPU, where at each clock, each stage is supposed to pass on the following stage the instruction it has been working on.
This means that the 5-stage pipelined ARM CPU can execute instructions faster than the equivalent non-pipelined ARM. This can be seen as a first level of parallel computation, specifically on instruction-level parallelism, as we will see later in this section.
Figure 1.6 represents the ARM CPU without pipelining which executes 3 instructions by inserting the next instruction when the previous is finished. Therefore, the total number of cycles taken for the 3 instructions corresponds to 15 clock-cycles. If we wanted to generalize this, the total number of cycles is equal to the number of instructions multiplied by the number of steps.

Figure 1.6: 5-step classical ARM CPU[37]

Figure 1.7 represents an ARM CPU with pipelining executing 5 equations. Hypothetically, if all equations are independent from each other, we could insert an instruction at each clock-cycle and wait 4 clock-cycles more for the result of the last instruction. Therefore, the total number of cycles taken for 5 instructions raises to 9. If we generalize this, the total number of cycles is predictable by the following formula :

$$(\text{Number of Cycles}) = 5\cdot(\text{Number of Instructions}>1)+(\text{Number of Instructions-1})$$

However, this is only true if at each cycle, the instruction that we insert in the ARM CPU is not dependent on the results of the instructions still in the pipeline. For example, if the fifth instruction depends on the result of the first instruction,

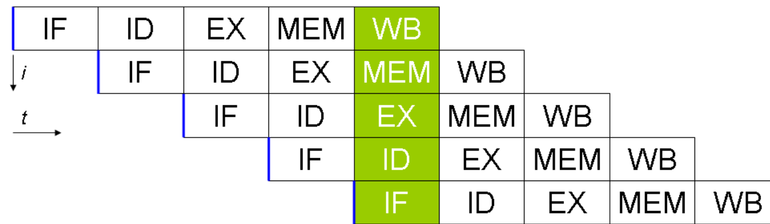


Figure 1.7: 5-Stage Pipelined ARM CPU[37]

we will not be able to insert this fifth instruction until the first instruction is finished. Therefore we will wait for a cycle until the first instruction is finished and only afterwards insert the fifth instruction, as can be seen on figure 1.8. This will decrease the effectiveness of the pipelined processor, which effectively shows the importance of having independent instructions and being able to select them accordingly to the possibility of inserting them in the pipeline.

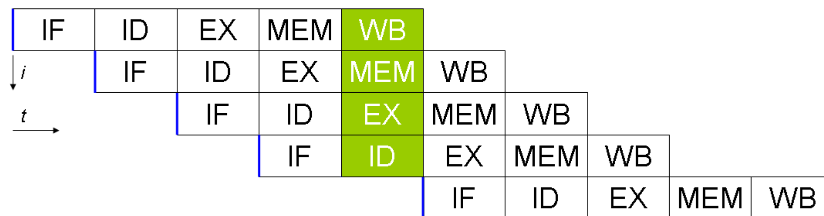


Figure 1.8: 5-Stage Pipelined ARM CPU - Hazard[37]

Taking these elements into account, we see that pipelining is useful in the case of instruction-level parallelism at the condition that we are able to effectively insert independent instructions. This compensates a parallelisation we would have to provide with multiple PE with the benefit of higher frequencies and resources optimisation.
With regards to the frequency, the division of a sequence into successive smaller ones decreases the total length that the signal must make in a single clock cycle. Therefore increasing the potential frequency at which each of the stages of the sequence will be ran. This effect decreases with the amount of stages as it becomes systematically more difficult to divide the circuit into smaller coherent parts.

1.2.2 Altera IP blocks

As we showed in section 1.1, we have mainly memories and Processing Elements (PE) which are either 32-bits floating point multiplications or 32-bits floating point addition-subtraction. We are not able to apply this pipelining on the memories but it is possible from the IP library proposed by Altera to find pipelined PE of a specific number of stages or frequency for the operations we want to implement.
We will first focus on the possibilities offered for each type of operations, then compared both types of operations and finally take into account the memory and the dependency it causes. We will consider these elements with the DE10-Nano, which is a FPGA board of the Cyclone V family.
For the Cyclone V family we have the following stages, the resulting maximal frequency and the use in resources as predicted by the internal compiler:

Latency (Stages)	1	2	3	4	5	6	7	9	10	14
Frequency	37	55	72	89	103	120	144	159	178	233
LUT	538	569	587	630	652	656	695	762	796	905

Table 1.1: Properties of Addition-Subtraction IP blocks

Latency (stages)	2	3	4	5	8	9
Frequency	63	109	153	199	218	230
LUT	170	193	207	231	362	391
Multipliers	2	2	2	2	4	4

Table 1.2: Properties of Multiplication IP blocks

Taking in account the frequency of both types of operations and starting from the hypothesis that we will use a single clock for both types of operations, we can search for valid couples of pipelined architecture for each frequency and the related pipelined stage number of both types of operations.

ADD	1	2	3	3	4	5	6	6	7	9	9	10	14	14	14
MULTI	2	2	2	3	3	3	3	4	4	4	5	5	5	8	9
Frequency	37	55	63	72	89	103	109	120	144	153	159	178	199	218	230

Table 1.3: Coupling latency of subtraction/addition and multiplication

We could also illustrate this with the following figure 1.9:

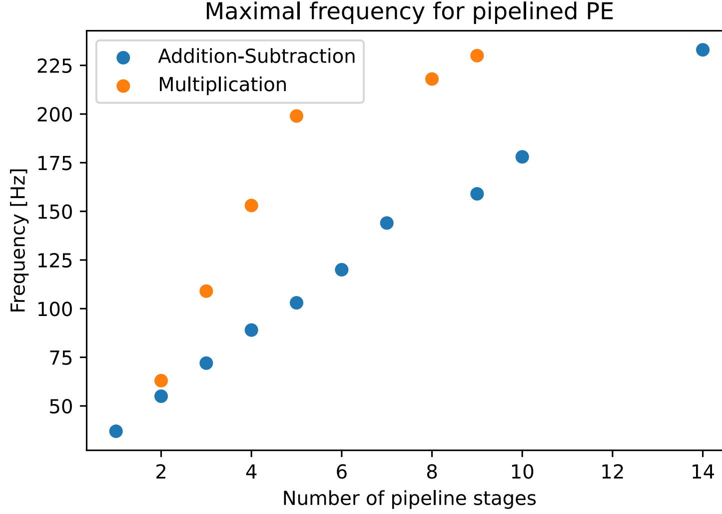


Figure 1.9: Maximal frequency by latency for ADD and MULTI PE

We can choose a combination of pipelined PE which will constrain the architecture to a specific maximal frequency. In order to concretely find out which kind of pipelining scheme is the most efficient, we need to test out a few of these combinations and compare them between each other. On top of this, it will be a first characterisation of a multibody system. For this example, we use a simple multibody system of a 23-dof railway-bogie, apply the equation placement algorithm for 3 pipelining schemes (ADD, MULTI) = [(3+2,2+2), (6+2,4+2), (10+2,5+2)] which constrain the frequency to respectively 72, 120 and 178 Hz. This gives an approximately equal time of computation for the three cases for a single instructions and will show or not the benefits for our application.

Remark - As mentioned previously, we implemented this study knowing the number of bits, here 32-bits floating-points following the IEEE 754 standard. But there are also 64-bits floating-points IP blocks for higher precision, these will require for the same frequency, a higher amount of resources combined with a higher latency, approximately the double of that of the 32-bits floating-points. Finally, the IP block can be customised for a specific number of bits between 32 and 64, where the breakdown of bits through the mantissa and the exponent is determined by the user following its own necessities.

1.2.3 Pipeline Study

First we will show the number of cycles being produced for each case. Secondly, we will show the amount of time needed to complete the equations for each case. And finally, we will show the parallelisation rate for each case. Each of theses cases can be found respectively in figure 1.10, 1.11 and 1.12.

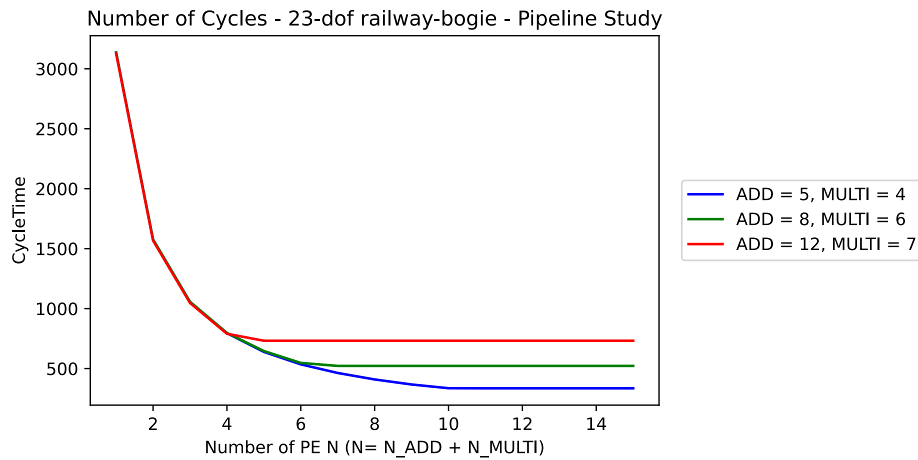


Figure 1.10: Number of Cycles depending on the number of pipeline stages

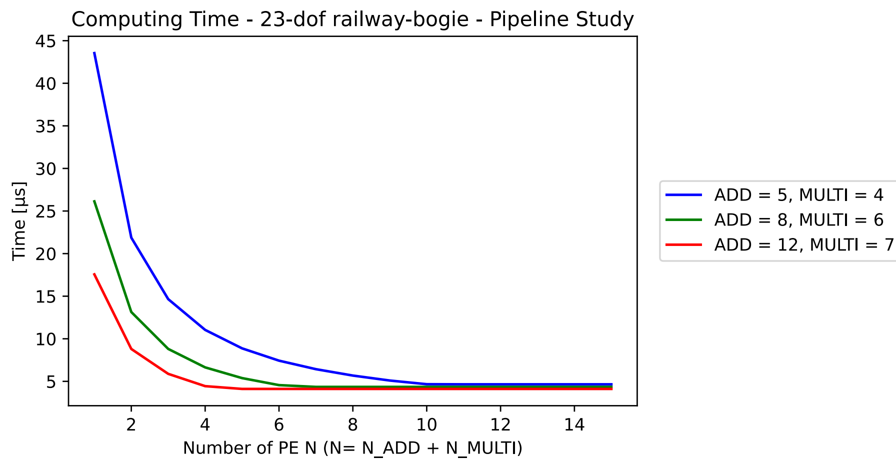


Figure 1.11: Computing time depending on the number of pipeline stages

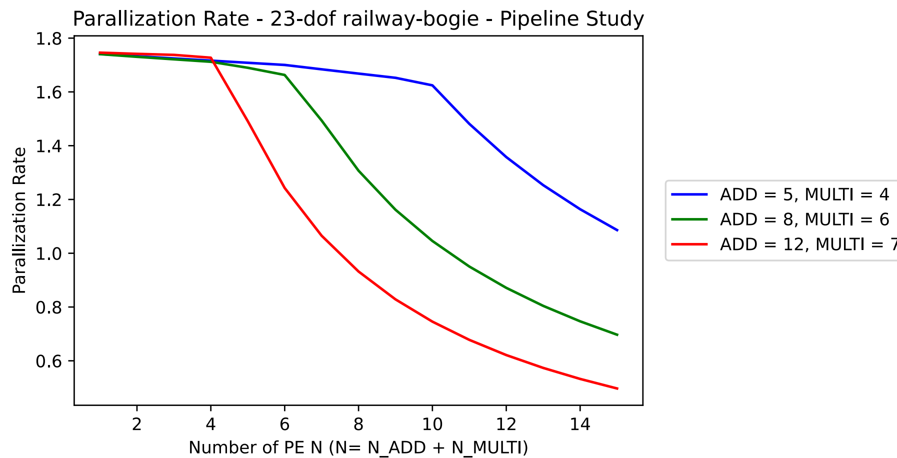


Figure 1.12: Parallelisation rate depending on the number of pipeline stages

From the figure 1.10, we notice that the lowest amount of cycles is obtained for a configuration where not only the number of PE is high, but also the number of stages of pipeline is low. Before reaching that low amount of cycles, we see that we have the same number of cycles for the same number of PE while varying the number of stages. This is due to the fact that we are simply dividing the number of operations by the number of PE.
For the figure 1.11, we notice that the computation time is starting lower for the configurations that have a higher number of stages in their PE. This is kept while increasing the number of PE until all the computations times reach a minimum low which is common. Therefore, we will have for the same amount of resources, in other words, the same amount of PE's, a lower computation time when we increase the number of stages in the pipeline. In our case, this is exceptionally true because there is a proportional increase of the frequency with the number of stages. This cause the final computation to tend to be the same.
For the figure 1.12, the inflexion of the curves corresponds to the moment we can't reduce any more the number of cycles. We should also note that the parallelisation rate at this inflexion, is higher for configurations with a higher number of stages. We conclude that we should take PE with a higher number of stages as long as the frequency increases in the same proportion.

1.3 Implementation

In this section, we will describe the practical implementation on our DE10, we will characterize the amount of resources needed to compute the forward dynamics of the 23-dof railway-bogie for 8 PE, equally divided to have 4 ADD PE and 4 MULTI PE. Additionally, we will talk about the current limitations and the future perspectives for a FPGA implementation.

Regarding the frequency of both clocks, we know that the frequency for M10K is limited to 240 in the case of the DE10-Nano. As the clock of the PE Memory should run at twice the frequency of the PE, we know that the frequency of the rest of the circuit is limited to 240. The other elements that could cause a limitation on our frequency is the PE and the interconnections. In this case, we decide to aim to a limitation only due by the PE Memory and have a maximal frequency for the PE that would be higher to eventually compensate the delay in the interconnection. Therefore choose the last couple of PE we had in 1.2.2, the ADD PE will have 10 steps for a maximal frequency of 178 MHz and the MULTI PE will have 5 steps for a maximal frequency of 199 MHz. We need 2 additional steps for the passing through the interconnections and the writing down of the intermediate values.

1.3.1 Resources usage

The final implementation will contain 8 PE, equally divided to have 4 ADD PE and 4 MULTI PE. The choice of the number of PE will also determine the size of the other parts inside each block: Instructions Memory, PE Memory, Buffer Instructions and Buffer Memory, and the Interconnections.
The memory sizes will not only vary with the number of PE, but they will also vary depending on the equations we are going to insert.
Regarding the amount of resources taken for the PE themselves, we already estimated them previously, but we will correct them with the final numbers from the software report.
Given the configuration and considering we want to process the direct dynamics of the 23-dof railway-bogie, the number of cycles will be 987.

About the Instructions Memory, we need to drive 4 'virtual' ports on the PE Memory, because we need to write two addresses and to read two addresses. Divided over two cycles because we are limited to 2 ports on the memory.
Additionally, we need to have an enable-write bit for each port during the write cycle. In the case of the Instructions Memory linked with the ADD PE, we need to drive the selection of the operation with an extra bit. For the size of the ports,

we consider that our PE memory contains 2048 addresses, which is depth of 11 bits. We decide to represent the addresses using 3 hexadecimal numbers. Therefore we have a total width of 52 bits because we cannot implement a lower amount in width for the M10K. The Instructions Memory will contain 48128 bits for the ADD and 47104 bits for the MULTI PE. The memory is made out of 3 to 4 M10K.
The PE Memory will not only contain the intermediary variables but also the initial data we needed at the start and throughout the computations. The memory will contain 32-bits floating-points and as said previously, it would need to contain 2048 addresses. We will therefore use 65536 bits inside 7 M10K.
About the Buffer Instructions, we need to insert two addresses to read and write the saved results, we include a bit for the enable-write. This leaves us with a final 8 bits width memory. Therefore the total number of bits is 48128 divided over 2 to 3 M10K.
About the Buffer Memory, we use 16 addresses to save the results, which is more than needed to save the 32-bits floating-point results. Therefore, we need 512 bits using a single M10K.

About the ADD PE, we have for each one around 360 ALM (565 ALUT, 600 Logic Registers) and 2 DSP. This is more than expected, and additionally, we have DSP. The addition of DSP is due to the combination of the frequency with the interconnections, which causes the design to be a loop.
About the MULTI PE, we have for each one around 100 ALM (169 ALUT, 170 Logic Registers) and 3 DSP.
In general we see that there is a higher need than what was written in the Altera IP blocks. This is due to the practical implementation that is using more resources than the ideal implementation. We could reduce the usage by changing the compilation parameters of the software. But the final use is representative of what we could expect if we wanted to add more elements.

About the interconnections, to connect the 8 blocks, we need to have 16 inputs and 16 outputs for the interconnections. Respectively to connect the 8 results, one from each PE and 8 data read from the Buffer, one for each PE as well. We use a multiplexer(MUX) for each PE such that each uses 160 ALU (110 to 150 ALM). Therefore we use 2560 ALU (2080 ALM).
About the Interconnections Memory, the memory size will be equal to the number of cycle times multiplied by $2 \cdot N \cdot sqrt(2 * N)$, where N is the number of PE. We will therefore use 65536 bits, for which the software would use 20 M10K.

We made a summary out of all this resources use which is available at table 1.4

	ALM	ALUT	Logic Registeries	M10K	DSP
PE ADD	360	565	600	0	2
PE MULTI	100	169	170	0	3
Instructions Memory	0	0	0	3-4	0
PE Memory	3	4	0	7	0
Buffer Instructions	0	0	0	2-3	0
Buffer Memory	0	0	0	1	0
Interconnections	2080	2560	0	0	0
Interconnections Memory	0	0	0	20	0
Total (%)	4413/41910	-	-	120/553	20/112
Total (%)	11 %	-	-	22%	18%

Table 1.4: Resources Usage - DE10-Nano : 8 PE (4 ADD & 4 MULTI)

1.3.2 Limitations & Future Perspectives

We made the design expecting to have two final clocks. The first clock should be running to incrementally increase a counter giving the current cycle we are in, called CLK120. A second clock is used specifically for the PE memory to be able to have the two cycles during a single cycle of CLK120, we call that clock CLK240. These clocks should respectively run at a frequency of 120 and 240 Mhz, as they are linked. As said in section 1.1.2 Constrains & Solutions, we implement the CLK120 using the rising edge of the CLK240.

Unfortunately, the claimed maximal frequency of the design we were able to implement for CLK240 is about 50 Mhz, which is only a fifth of what we expected from our initial discussion.
The only element we did not determine originally at which frequency it should work was the interconnections. As a consequence, we tried out the design by ignoring the interconnections. In that case, the maximal frequency of our CLK240 would be around 50 MHz which is exactly the same as previously.
A second element could be simply the CLK120 we use for selecting the write or read operation. As this clock is both used for that purpose and is being derived from CLK240, which is also used as the assigned clock of the memory. We removed in that purpose the CLK120 and tried to find the new maximal frequency. In this case CLK240 is limited to 240 MHz.
Therefore we know that although the solution we implemented successfully handled the issue regarding the number of ports. This causes the frequency to decrease significantly. In the end, the main objective was to both test out the architecture and the placement algorithm we will later explain. This objective was fully obtained as we successfully computed our test case on ModelSim and on the FPGA with

SignalTap but further development on the architecture will be needed.

Regarding the use of the resources, we see that for the DE10, we can have two limitations, Memory and DSP. First on Memory, we have both an increase of memory need in case of bigger system which needs more memory addresses and has a higher number of cycles. Second on DSP, if we are willing to increase the number of PE, we should be able to increase it until we have a number of PE equal to 40; potentially even more as ADD PE are not supposed to contain any DSP and MULTI PE are supposed to contain only 2.
In the same aspect of the maximal frequency, this is highly dependent of the design we are able to implement. If we change some of the compilations parameters we could be able to decrease that amount to contain more PE.

As said before, the use of a higher end FPGA could solve both of our issues. For example the Statix 10 propose a 4-port memory block which allows to solve our frequency issue. Additionally, the maximal frequency of the block on the Statix 10 is higher than what we are able to implement on our Cyclone V series, about twice the frequency. Finally, the resources we have at our disposal would be higher eventhough we have also a workaround that is possible with the embedded ARM [33].

Chapter 2

Vectorisation of Equations

The final objective of this work is to show the potential vectorisation of the equations that are produced by Robotran, as this work is written in the continuation of the work accomplished by Tony Postiau during his PhD Thesis. We therefore took back some of the advances he made and got further into the subject. What is meant by *Vectorisation* is a fine-grain parallelisation where instruction-level parallelism is exploited from the equations produced by the Robotran symbolic generator.
The advantage of this method is the applicability to any multibody system, regarding both size and topology. Unfortunately, the dependencies between the successive equations raise two issues. The first is the production of a parallel architecture or interface. This architecture should be able to efficiently exploit the fine grain parallelism of the instructions. We proposed such an architecture in section 1. The second is a performing algorithm that is able to accurately place the equations such that the maximum parallelism is exploited given the specific architecture.
The vectorisation of equations was born from an simple observation, as explained in [38, 39], and exploited in [3], which we will further develop regarding the amount of operations being made for each line of the vector step. As stated in [39], the equations produced using a recursive algorithm might not each and every time be dependent on the previous equation. The result of this can be illustrated in 2.1, where we take every equation and establish successive steps to compute them in parallel.

$$\begin{array}{rc}
AF24 = & -g(3)\cdot S4;\\
AF34 = & -g(3)\cdot C4;\\
OM15 = & qp(4)\cdot C5;\\
OM35 = & qp(4)\cdot S5;\\
AF15 = & -AF34\cdot S5;\\
AF35 = & AF34\cdot C5;\\
OM16 = & qp(5)\cdot S6 + OM15\cdot C6;\\
OM26 = & qp(5)\cdot C6 - OM15\cdot S6;
\end{array}
\longrightarrow
\begin{array}{rc}
z1 = & \begin{bmatrix} -g(3)\cdot S4;\\ -g(3)\cdot C4;\\ qp(4)\cdot C5;\\ qp(4)\cdot S5; \end{bmatrix}\\
z2 = & \begin{bmatrix} -z1[2]\cdot S5;\\ z1[2]\cdot C5;\\ qp(5)\cdot S6 + z1[3]\cdot C6;\\ qp(5)\cdot C6 - z1[3]\cdot S6; \end{bmatrix}
\end{array}
\tag{2.1}$$

The observation is exploited by Robotran to make computations schemes where the independent equations are detected and grouped together in successive steps or vectors taking into account the dependencies towards the parents. This produces a number of equation vectors which show the parallelisation potential of multibody systems.
For example, a 23-dof railway bogie is constituted by 1812 equations which can be grouped together in 18 successive vectors, which represent a reduction of 100 of the number of computation steps. A bigger multibody system such as a 70-dof multibody system will produced 18536 equations which can be grouped into 80 successive vectors, which represents a reduction of 20 computation steps.
However, this vectorisation scheme presents a number of disadvantages:

- First, it is inefficient regarding the amount of computing units we should put in parallel to compute the equation at each step. We could have for some steps an under-use of the computing units as they have no equation to process.

- Second, in the same way, the equation themselves are presenting a difference in their computing cost as they can contain more or less operations. This will cause during the step to have computing units which will have completed their equation early and will have to wait for the most costly operations to be computed before being able to compute the next step.

In the following sections, we will explain how we solve these issues by using the individual successive operations and how by changing the expressions themselves we are able to improve even more the vectorisation potential of these equations.

2.1 Atomisation of Equations

Although our parallelisation potential was put in evidence thanks to the transformation of sequential equations into sequential vectors. The number of operations in each equation and the number of equations in each step would prevent us to fully exploit the parallelisation potential.
Therefore, we need to further manipulate the equations in order to fully exploit the instruction-level parallelism of the equations.

In order to solve the issue regarding the difference in cost of the equations. We exploit the same principle as in the PhD thesis of Postiau [3]. This consists into dividing the equations into smaller parts, into unitary operations.
Taking the previous series of equations, we can atomise the complex expression of the original equations using intermediate values such that will have the following series of equations:

$$\begin{array}{ll} AF24 & = -g(3) \cdot S4; \\ AF34 & = -g(3) \cdot C4; \\ OM15 & = qp(4) \cdot C5; \\ OM35 & = qp(4) \cdot S5; \\ AF15 & = -AF34 \cdot S5; \\ AF35 & = AF34 \cdot C5; \\ OM16 & = qp(5) \cdot S6 + OM15 \cdot C6; \\ OM26 & = qp(5) \cdot C6 - OM15 \cdot S6; \end{array} \longrightarrow \begin{array}{ll} AF24d & = g[3] \cdot S4; \\ AF24 & = 0.000 - AF24d; \\ AF34d & = g[3] \cdot C4; \\ AF34 & = 0.000 - AF34d; \\ OM15 & = qp[4] \cdot C5; \\ OM35 & = qp[4] \cdot S5; \\ AF15d & = AF34 \cdot S5; \\ AF15 & = 0.000 - AF15d; \\ AF35 & = AF34 \cdot C5; \\ OM16d & = OM15 \cdot C6; \\ OM16g & = qp[5] \cdot S6; \\ OM16 & = OM16g + OM16d; \\ OM26d & = OM15 \cdot S6; \\ OM26g & = qp[5] \cdot C6; \\ OM26 & = OM26g - OM26d; \end{array} \quad (2.2)$$

With the atomisation we have to insert a zero in front of some equations in order to effectively have an operation. This increases lightly the number of operations.

2.2 Indexing of Equations

Once we atomised the equations into individual operations, we need to decide which operations are prioritized in order to decrease the computation time and increase the parallelisation rate. For this purpose, we will be using a list scheduling mechanism, which will be able to separate the operations between each other based on a range of factors.
We will first revisit and underline a few elements of the list scheduling as presented in the PhD of Postiau [3], and afterwards explain how we implement this part in the placement algorithm.
From the following equation, we represent a binary expression tree (BET), illustrated in figure 2.1.

$$\begin{aligned} CF323 =& 0.000 - FA123 * L[2,23] + FA223 * L[1,23] - I[1,23] * OM123 * OM223 \\ &+ I[5,23] * OM123 * OM223 + I[9,23] * OA323; \end{aligned}$$

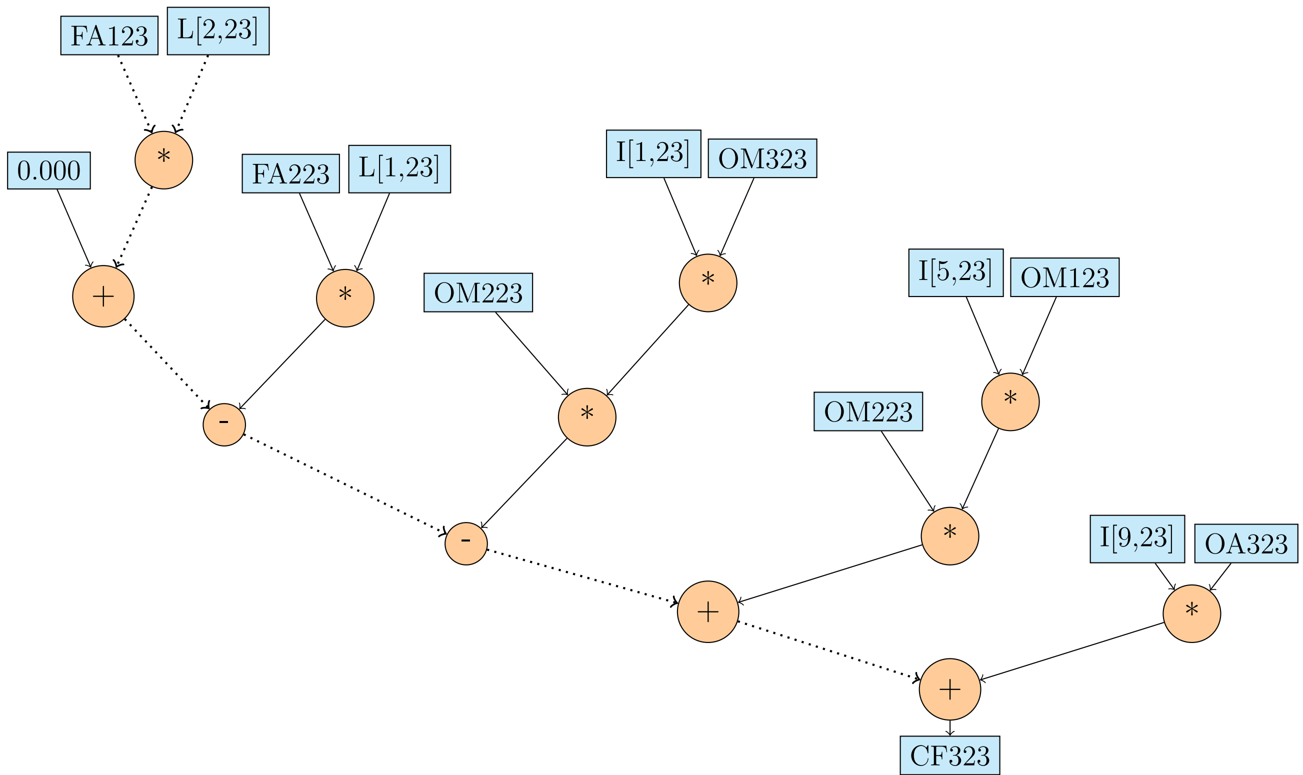


Figure 2.1: Treelike structure of an equation

2.2.1 ALAP and ASAP Attributes

In [3], we took the classical attribution of two indices which are showing both the interdependence of equations and the successive operations. These two indices are ASAP (As Soon As Possible) and ALAP (As Late As Possible). We will use them for our individual operations.
The first attribute, ASAP, is representing the cycle in which the operation should be computed considering that it must be done the earliest knowing the ASAP of the two parents. To process this, we are starting at the initial variable and successively increment on the children considering the maximal value of both parents.
The second attribute, ALAP, is representing the cycle in which the operation should be computed considering that it must be done as late as the parents of the operation allows. To process this, we start at the end of the list of equations. We give the operations which do not have any children the maximal ASAP attribute from our list of equations. And we will gradually get to the beginning of the list of equations by decrement the minimal ALAP attribute of the children.
We took binary expression tree, gave ASAP attributes to the initial variables and deduced the successive attribute which can be seen in figure 2.2.

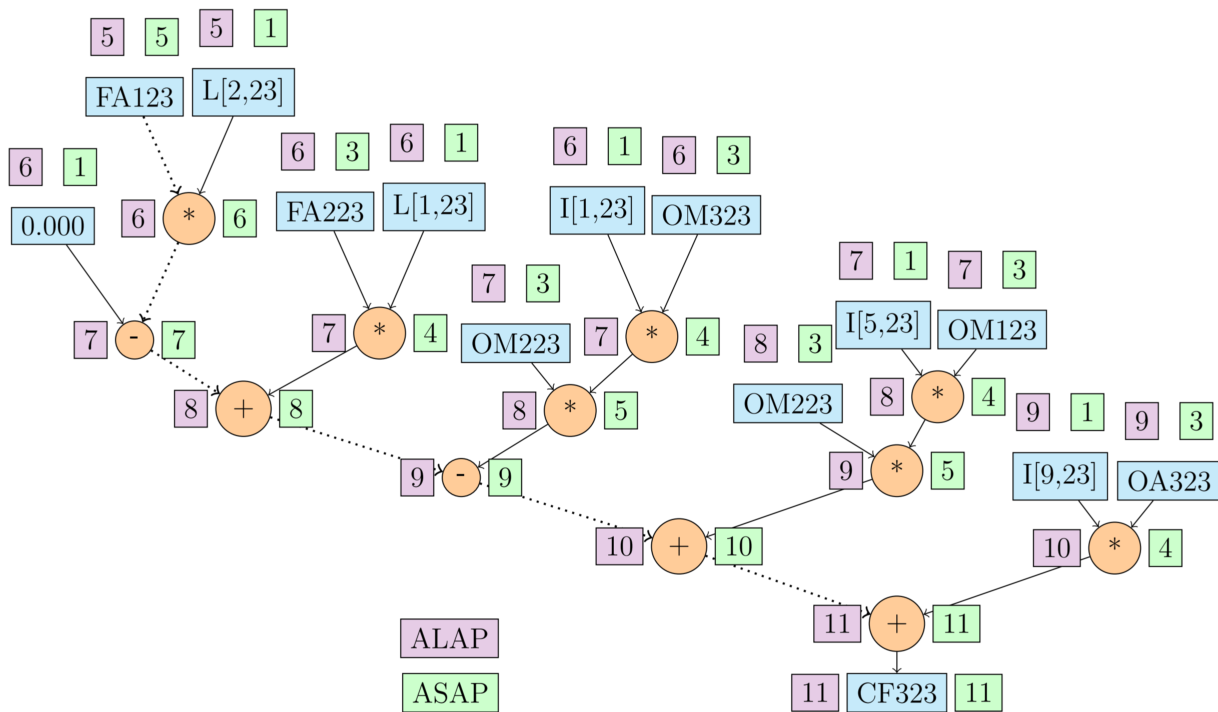


Figure 2.2: ASAP and ALAP indices for dependency tree

2.2.2 ASAPCycle and ALAPCycle Attributes

Previously, we explained a list scheduling based on the simple indices ASAP and ALAP, where the weight of the operations is of a single unit. An improvement over this will be presented in this section. We will take into account the number of cycles each operation by introducing them in the BET. We call these new attributes ASAPCycle and ALAPCycle.
We will use for example the maximum of these attributes in the chapter 3 Robotran Equations.
We take the latency due to our current architecture, therefore we associate each operation with a certain weight which represents the number of cycles taken for each type of operations. We consider that multiplication has a weight of 7 cycles due to the 5-stage pipelined multiplication PE and two cycles for the memory. While the addition/subtraction operation has a weight of 12 cycles due to the 10-stage pipelined addition/subtraction PE and two cycles for the memory.
The resulting dependency tree with the associated attributes can be seen at figure 2.3.

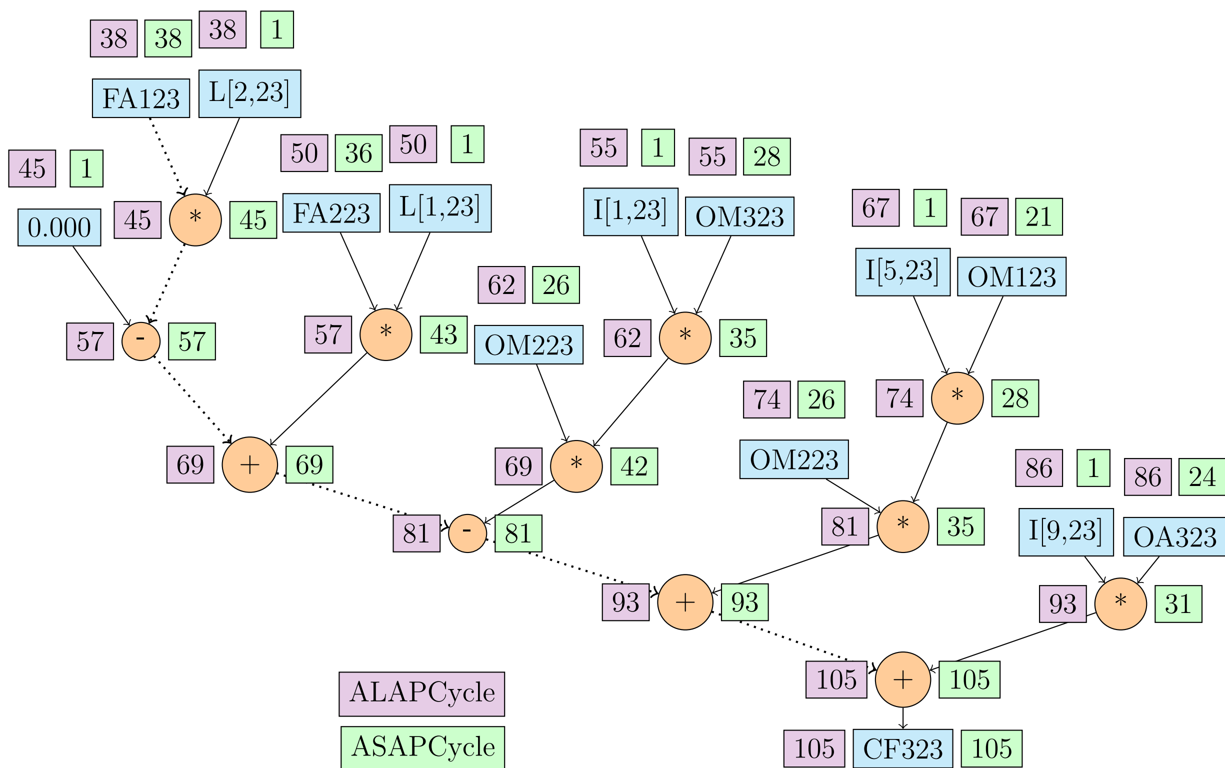


Figure 2.3: ASAPCYCLE and ALAPCYCLE indices for dependency tree

2.3 Placement of Equations

Once the equations have been atomised into single operations and these have been classified according to a classical ALAP-ASAP scheduling list, we are able to describe the final part of the algorithm we developed.
For the correctness of the algorithm, it shall verify at each cycle a few elements when proposing to insert an operation:

1. What is the status of the two parents of the operation. For each parent, we could either not have started the computation; or if the computation has started, it could still be in the pipeline of the PE. We are only selecting the operation if the two parents have been processed. Otherwise, we would need to restart back for the next operation.

2. When this verification is done, we still need to check for each parent whether it is already in the PE Memory or if eventually we could write it. If we need to write down the variable, two possibilities are offered:

   (a) First possibility called *Direct* is when the variable is being written down directly in the PE memory after being computed, without passing through a write and read in the buffer.

   (b) The second possibility called *Indirect*, writes down the variable after it has been computed, by writing it down in the Buffer memory and reading it later in order to write it in the PE memory.

A naive selection of the PE will insert the equation in the first PE that is available, independently of any other consideration such as memory.
We propose an alternative where we make a difference based on a memory status, which is aimed at reducing PE memory writing rates, the size of the PE memory, the Buffer memory read and write rates and the size of the Buffer memory. We expect the load of the computation to be more equally divided between the PE and as a consequence also reach the goals we mentioned regarding memory size and usage.

We will consider a status for each parent, if the parent is in the PE memory, if the parent can be written directly or indirectly in the PE memory.
We respectively call these as PE (Left and Right), Direct (Left and Right) and Indirect (Left and Right).
These three cases present different memory usages. We will always prefer to have less memory usage and therefore we decide to use the following preference, ranged from 1 to 9. We show this preferential selection in table 2.1.

	PE Right	Direct Right	Indirect Right
PE Left	1	2	5
Direct Left	3	4	7
Indirect Left	6	8	9

Table 2.1: Preferential Placement

We tested both placement techniques to prove that indeed the second technique would give significant advantage over the previous, more naive. The memory usage of both techniques is illustrated in figure 2.4 and 2.5 for the 23-dof railway-bogie and in figure 2.6 and 2.7 for the 70-dof system, respectively for the memory of the PE and the Buffer at different numbers of PE. Where the bar represents the minimal and maximal size and the points represent the mean usage.

In the case of the 23-dof railway-bogie there isn't a very large difference between both techniques. We can explain this by the low number of PE we need to extract the maximal parallelisation for that particular multibody system and by the small number of equations, around 5500 operations. We tested it with a bigger system such as the 70-dof system, which has 100000 operations and where the size would allow to show the utility of our preferential treatment.

The reader should note that when N_ADD=N_MULTI=1, the size is the same as the placement algorithm doesn't have a choice over the PE it could use, as there are only one PE for each type of operation.
We notice the following elements:

1. A difference on the size of the PE Memory and of the Buffer Memory between the ADD and MULTI blocks.

2. Classical Scheme has a slight increase in the size of the PE Memory for the MULTI block. Opposed to that, the Preferential Scheme has a constant decrease of size.

3. Classical Scheme has a size increase on the Buffer Memory MULTI, while the Preferential Scheme has a constant decrease.

4. In general, the Preferential Scheme has a lower size regarding both memories compared to the Classical Scheme. The difference between the maximal and minimal size of the memories for the different PE is smaller for the Preferential Scheme.

We will simply conclude that our preferential scheme shows a constant decrease throughout the increase of PE numbers both for the PE Memory and the Buffer

Memory.

This issue of having an equal memory size (or nearly equal) is dominated by good practices in FPGA design, we desire to have regular design element. Additionally, decreasing the size of it lowers the usage of resources on the FPGA and potentially avoid frequency limitations due to increased interconnections.

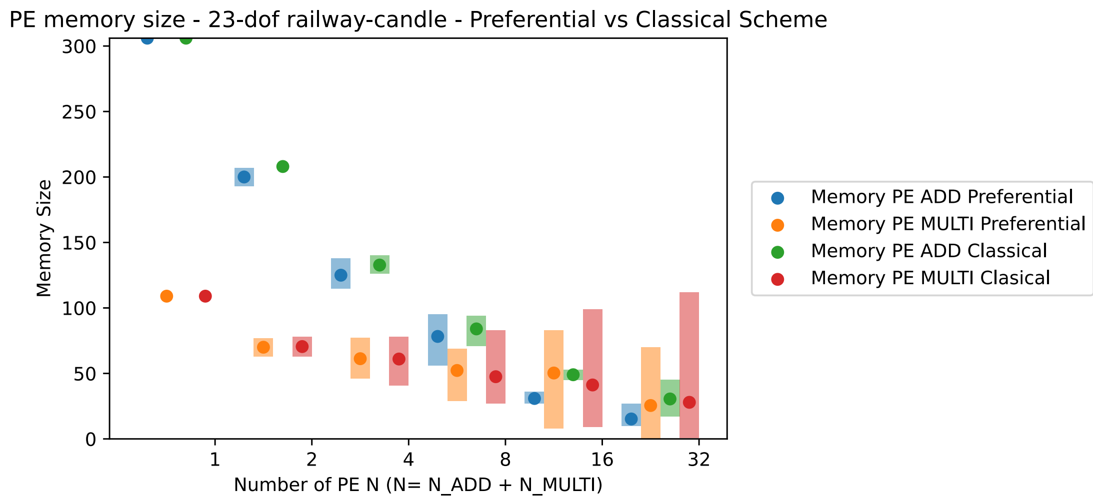


Figure 2.4: Memory associated with PE for the two placement methods - 23-dof - railway-bogie

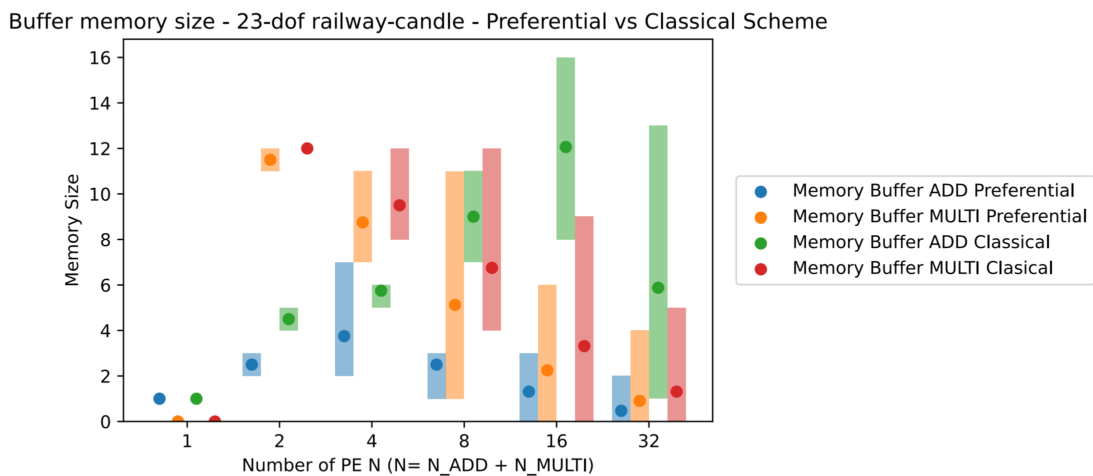


Figure 2.5: Memory associated with Buffer for the two placement methods - 23-dof railway-bogie

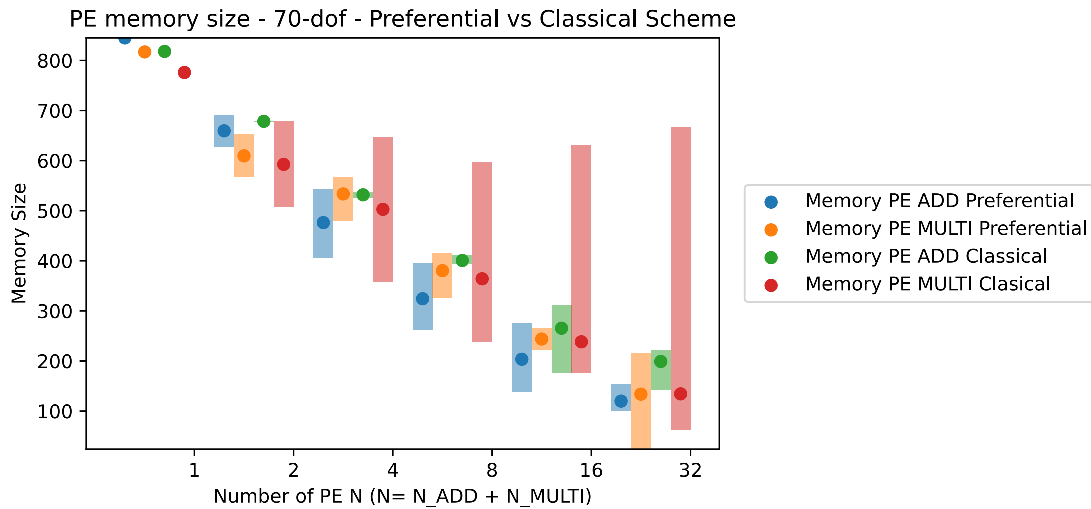


Figure 2.6: Memory associated with PE for the two placement methods - 70-dof system

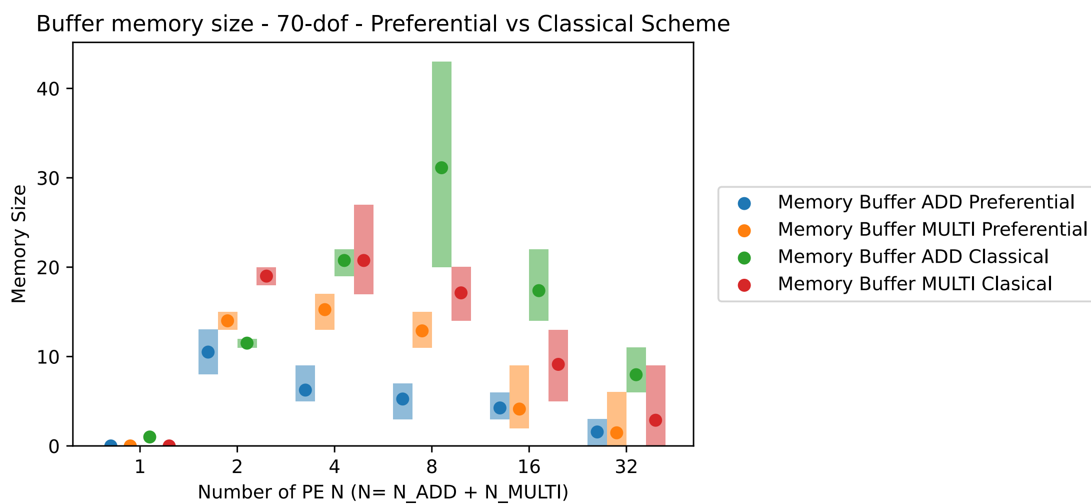


Figure 2.7: Memory associated with Buffer for the two placement methods - 70-dof system

2.3.1 Classical vs Mobility List Scheduling

The classical classification consists in sorting the operations first based on their ALAP attribute, then secondly for the operations with equal ALAP attribute, in sorting operations by their ASAP attribute. This allows to group together the operations which should in last instance be computed at the same time. The second

sorting argument does not have a significant influence on the result.
We could talk about a third attribute, derived from the previous two, which we call mobility. We compute it as the difference between the ASAP and ALAP attributes. This attribute represents as the name suggest the number of cycles we can pass before we insert the operation without causing an increase on the critical path. If an operation has a mobility of 0, this means that the equation is on the critical path. Higher number of mobility means that the operations is further away from the critical path allowing us to compute more critical operations. For paths that have a low mobility and therefore are close, failing to insert the equation for more than the mobility will cause the children of this equation to become the new critical path.

Therefore, we could use this third attribute as an alternative to the classical classification.

This alternative classification is to first sort the operations based on the mobility and in second instance to sort the operations having the same mobility based on the ALAP. This classification will rather give a priority on the critical path and successively on the paths that have a higher mobility, where we are inside each path prioritising the elements that are at the start of the path.
The main difference between these two elements is that the classical classification will consider the operations based on the stage in which they need to computed at the latest, while the alternative we have with the mobility attribute, the operations will be seen with regards to the path they are in.
We looked into the practical differences between both classifications and if either is better than the other. We can characterise this based on two elements, the first is the number of cycles from the placement algorithm and the second is regarding the efficiency of the placement algorithm.
Efficiency of the placement algorithm, there is no doubt that the classical algorithm is better than our alternative. as our alternative does iterate on all the critical path from the beginning until the end, before getting on paths which are less critical but yet are locking the critical path. For the first element, we did pass the 23-dof railway-bogie through both cases and the results can be seen in table 2.2.

Number of PE	1	2	4	8	16	32
Classical ALAP - ASAP	3137	1578	806	743	743	743
Mobility Based	3187	1606	854	743	743	743

Table 2.2: Number of Cycles for classical Alternative Classification

The conclusion is that currently, the classical classification works better compared

to the alternative classification, at least in the way we implemented it. It iterates more efficiently over the dependency tree, and also more effectively vectoring the equations. This is because our alternative classification computes first the critical path before being constrained at an operation where one of the two parents is out of a path which has not yet been started. Therefore the critical path is delayed because we need to compute this less critical first before being able to continue computing the critical path.
The classical classification is able to compute all the paths so that when they meet they do not represent a bottleneck for each other.

2.4 Leads on improving cycles

The existence of the critical path is limiting the parallelisation of the equations because the number of cycles cannot be decreased below the length of the critical path. This constrain is permanent but if the length of the critical path could be lowered, this would enable us to increase the parallelisation of the equations.

Currently, we have taken directly the equations produced by Robotran without applying any kind of transformation except the atomisation of these into individual operations. We notice that although we have extracted the fine-grain parallelism of those equations, there can be improvements if we change the expression itself. This is due to the binary expression tree as the atomisation of the equations is processing the equation from the left to the right. In the case of the placement algorithm, we took into account the attributes to place the operations to improve the parallelisation. In the atomisation, we did not exploit the attributes to manipulate the operations to rewrite the equations such that the operations can be processed in parallel or at a more opportune timing.

In that purpose, we take back the binary expression tree (BET) and attributes from 2.2.2ASAPCycle and ALAPCycle Attributes, but we could also make this using the the BET and attributes we saw in 2.2.1ALAP and ASAP Attributes. We make this choice here to show the decrease in the number of cycles we can bring to the original Robotran equations.
In this section we present three improvements that could be implemented to decrease the critical path. These three improvements are: the zero removal, the tree balancing and the tree swapping.

2.4.1 Zero Removal

Some equations present either a minus before the equal sign or before a parenthesis, in order to manage this case we simply added a zero in front causing the minus to be handled thanks to the transformation in an operation. This operation can be avoided in most cases with two techniques, the first is to find a variable in the expression that we are able to move and put forward the minus sign, the second is to simply report the minus sign to the children of the variable.

When taking the BET, we have a zero along the path the most on the left. It turns out that this path is also the critical path we are willing to shorten. We therefore exchange the terms $FA123 \cdot L[2,23]$ and $FA223 \cdot L[1,23]$ such that we can remove the zero and decrease the critical path. We see the resulting tree in 2.8. With the removal of one addition/subtraction operation, we notice that we could end up with another critical path which is passing through other variables. In this case, this situation does occur, as the critical path goes again through FA123 but we see a decrease of the ALAPCycle Attributes on other variables showing that as we decrease the critical path length, we also decrease the mobility on other paths.

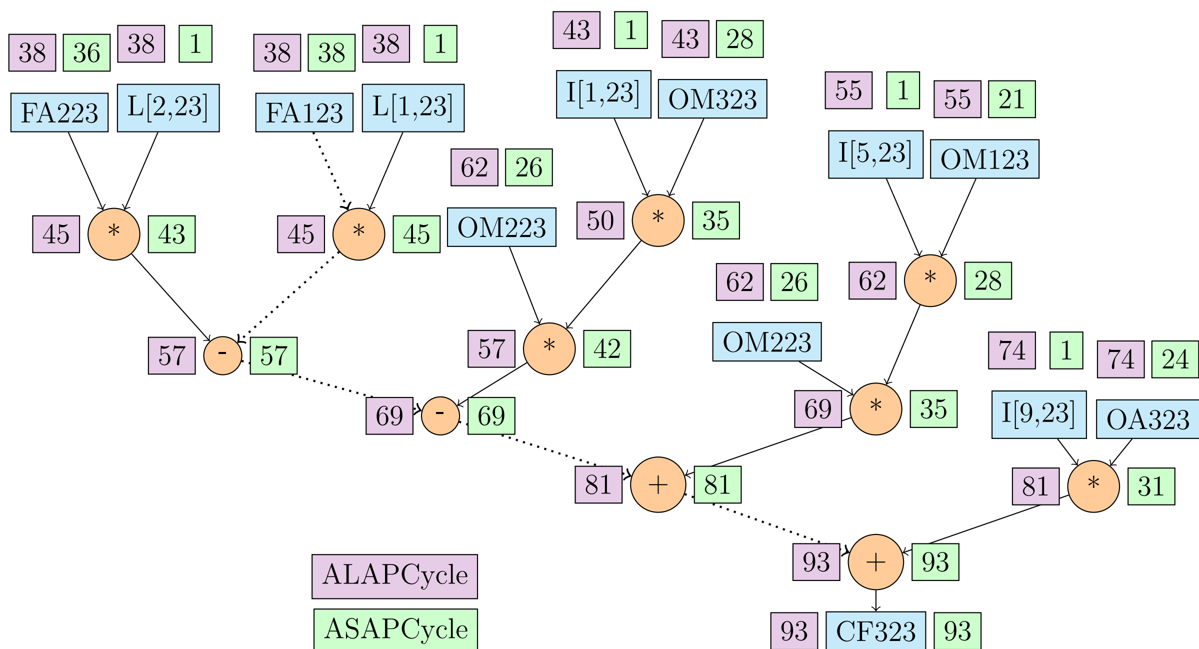


Figure 2.8: Dependency Tree - Zero Removal

2.4.2 Tree Swapping

The zero removal is a simple example of what we want to implement on a larger scale on a tree. In a more general context, we are willing to exchange the place of some paths or sub-trees to take advantage of the difference of their respective attributes. When looking at our BET, we notice a succession of additions/subtractions which are currently the most left path and also the critical path.
On top of this we see that the mobility of nodes connected to that succession of additions/subtractions has a different ASAPCycle attribute. We will therefore cut the link between the nodes and rearrange these such that the sub-trees that have the head with the biggest ASAP attribute are closer to the head of our BET.
The final result can be seen in figure 2.9.

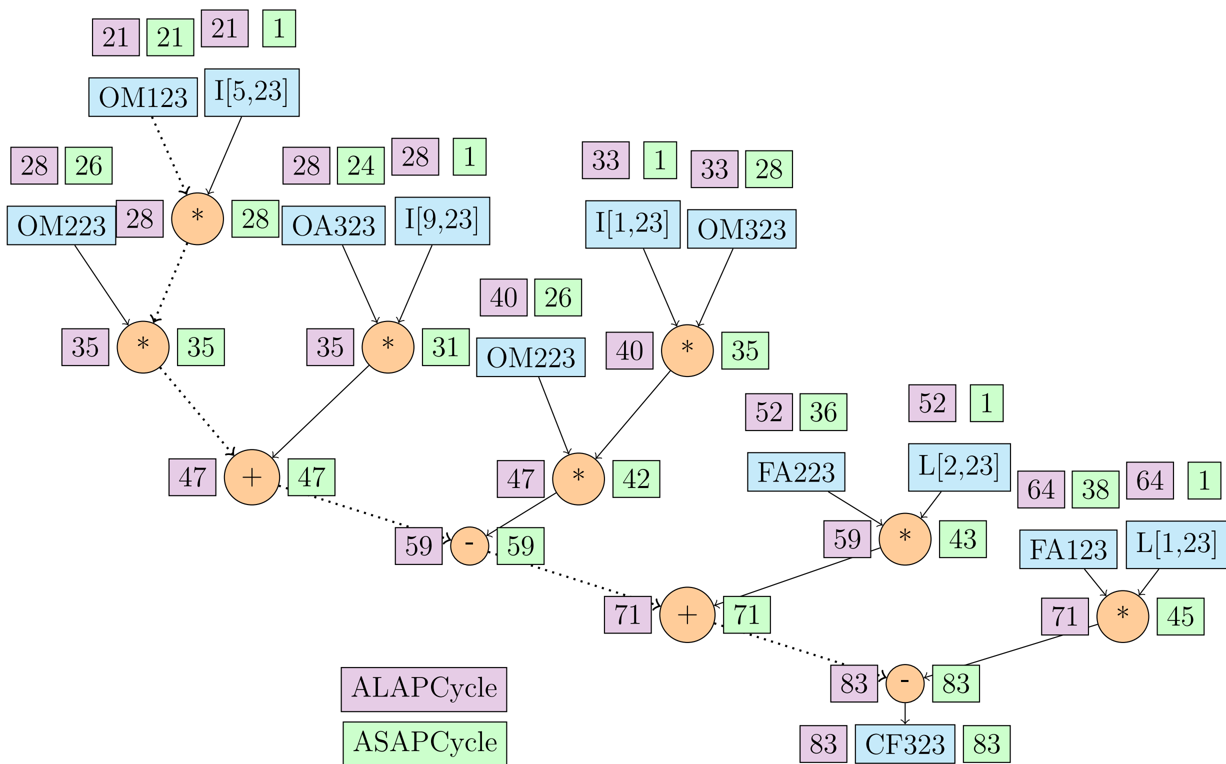
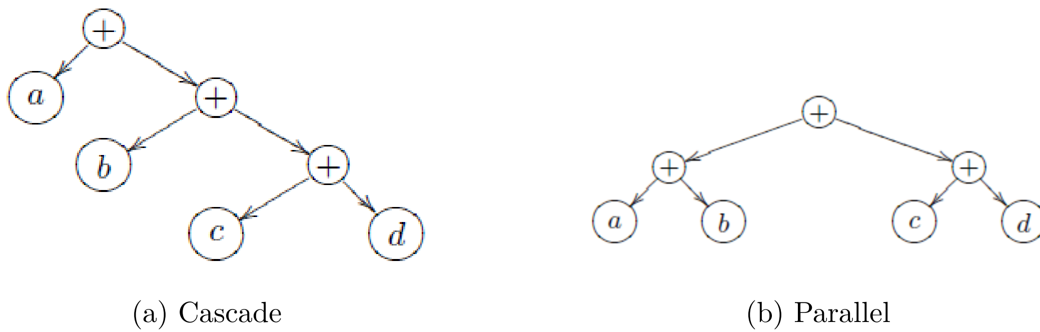


Figure 2.9: Dependency Tree - Tree Swapping

2.4.3 Tree Balancing

In his PhD. Postiau [3] explained that there were two different ways to make our tree. Either in a cascade scheme or a parallel scheme, illustrated in figure 2.10. For an equation such as $d + c + b + a$, the cascade scheme is represented by $((d + c) + b) + a)$ and the parallel scheme is represented as $(d + c) + (b + a)$. The expression in parallel represents a shorter critical path and a higher parallelisation rate but the Robotran Equation represents itself in a cascade scheme. The multitude of those cascade expressions together however still presents an interesting parallelisation rate.



(a) Cascade (b) Parallel

Figure 2.10: Binary Expression Tree[3]

As we are trying to decrease the length of the critical path, we are attracted by the idea of a parallel tree. In our case, we need to take into account the ASAPCycle Attribute to determine if this is useful. We saw from the previous BET that we have 3 sub-trees that have a very low mobility and that the critical path is due to OM123. The objective is therefore to change the mean head to the first left node. This decreases the critical path of 12 cycles unless the path changes of origin. Taking the BET modified in the previous section, we apply these changes and obtain the BET visible in figure 2.11.

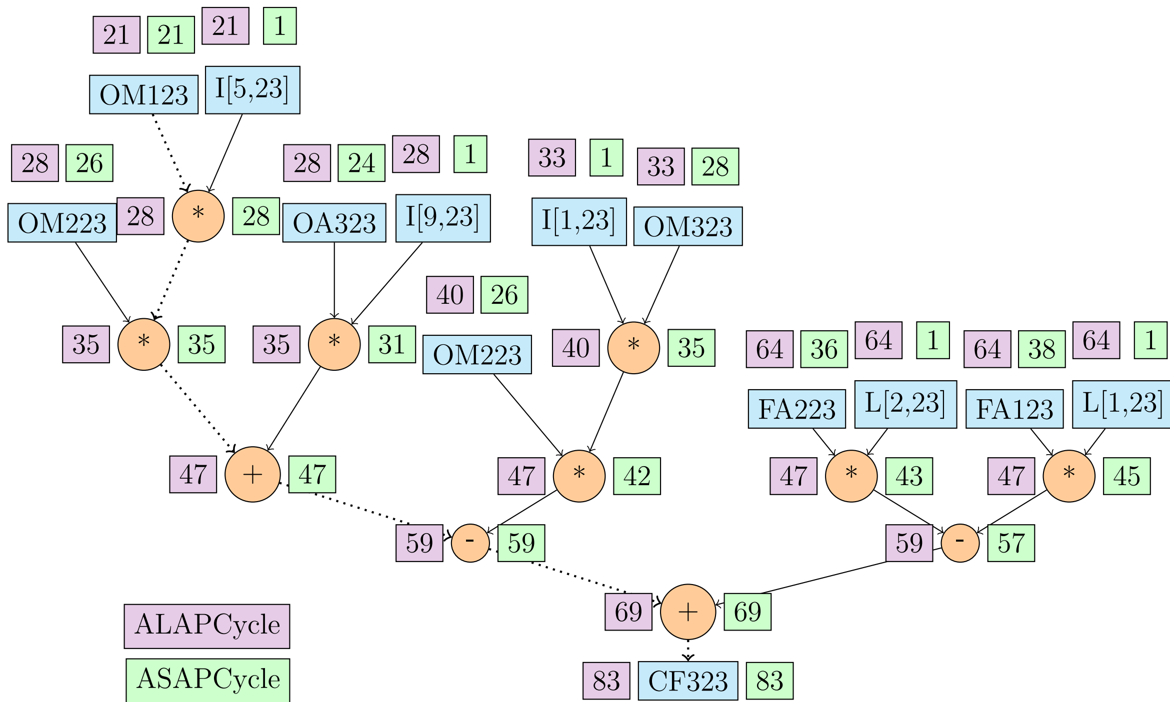


Figure 2.11: Dependency Tree - Tree balancing

2.4.4 Effectiveness

Regarding the effectiveness, we manually exploited the tree swapping technique. Because it is the method that present the highest gain while being easy to apply to our cases. For small systems of any topologies, we have been able to decrease the length of the critical path, to about one third in some cases. More data on this is presented in 3.2 Study Cases.

Regarding the tree swapping we have two situations:

- Serie of additions/subtractions : swap out the sub-trees to take advantage of their attributes. We need to take their associated sign and prevent the need for the insertion of an unnecessary zero. We used specifically this in our example.
- Serie of multiplications : swap out the sub-trees to take advantage of their attributes. As multiplications are associative, we are able to switch all the terms of the series of multiplications independently.

The tree balancing is the extra step we can implement after the tree swapping. Because the sub-trees are ordered such that we exploit the most out of our cascade

type structure. The tree balancing will change the structure of the tree to a more parallel type structure.
The principle of the tree balancing is like-wise the divide-and-conquer paradigm we use for sorting. We divide the operations that are on the critical path or very close over multiple parallel operations instead of successive ones. This allow us to reduce further the time when we have a high amount of elements that have the same or close attributes.
One of the issues this could present is that all the different sub-trees results need to be communicated between the PE. In architectures where communication between PE takes a high amount of cycles, this could in fact worsen the number of cycles we are taking to compute the equations. Indeed, instead of having all successive calculations, we need to communicate between the PE the successive results. Taking into account a large communication cost we have for OpenMP and MPI, we could have a higher number of cycles.

Regarding the zero removal, we are mainly aiming to show that for the additions/subtractions, we need to take into account the sign of the operation. For most equations, we should implement this improvement. The exceptions are present when the displacement of the sub-tree in front of the subtractions would increase the length of the critical path.

Chapter 3

Robotran Equations

A multibody system (MBS) is a mechanical system composed of bodies connected by joints. Two studies can be made on a MBS, direct dynamics and inverse dynamics. The direct dynamics consists of predicting the motion of the MBS given the known forces and torques. The inverse dynamics consists of producing the joint forces and torques to follow a specific trajectory.
The direct dynamics for open-loop systems is characterised by the following equation, which can be obtained by the Euler-Newton equations recursively computed on this topology [1]:

$$M(q)\ddot{q} + c(q, \dot{q}, frc, trq) = Q(q, \dot{q})$$

Where q and $\dot{q}$ are respectively the generalized relative velocity and acceleration, $M(q)$ is the system mass matrix, $c(q, \dot{q}$ represent the Coriolis, centripedal, gyroscopic forces as well as the externally forces $f(q, \dot{q})$ and the torques $trq(q, \dot{q})$ and $Q(q, \dot{q})$. Theses forces and torques depend from a user-input perspective through the implementation of the active and passive elements, such as a motor, a contact, etc.

For the inverse dynamics in open-loop, the same equations are used to find the forces and torques therefore [1]:

$$Q(q, \dot{q}) = \phi(q, \dot{q}, \ddot{q} frc, trq)$$

with

$$\phi \triangleq M(q)\ddot{q} + c(q, \dot{q}, frc, trq)$$

In the case of closed-loop systems, we apply the Lagrange multipliers such that the equations of direct dynamics are defined as:[1]

$$M(q)\ddot{q} + c(q, \dot{q}, frc, trq) = Q(q, \dot{q}) + J^t(q)\lambda$$

$$h(q) = 0 \; ; \; \dot{h}(q, \dot{q}) = J(q)\dot{q} = 0 \; ; \; \ddot{h}(q, \dot{q}, \ddot{q}) = 0$$

While for the inverse dynamics, the equations are defined as :[1]

$$Q(q, \dot{q}) = \phi(q, \dot{q}, \ddot{q} frc, trq) - J^t(q)\lambda$$

$$h(q) = 0 \; ; \; \dot{h}(q, \dot{q}) = J(q)\dot{q} = 0 \; ; \; \ddot{h}(q, \dot{q}, \ddot{q}) = 0$$

Robotran is a program generating symbolic multibody equations (kinematic, dynamic) of rigid multibody systems for applications in dynamic behavior of railway and road systems, biomechanics of the human body, satellites and much more(Figure 3.1).

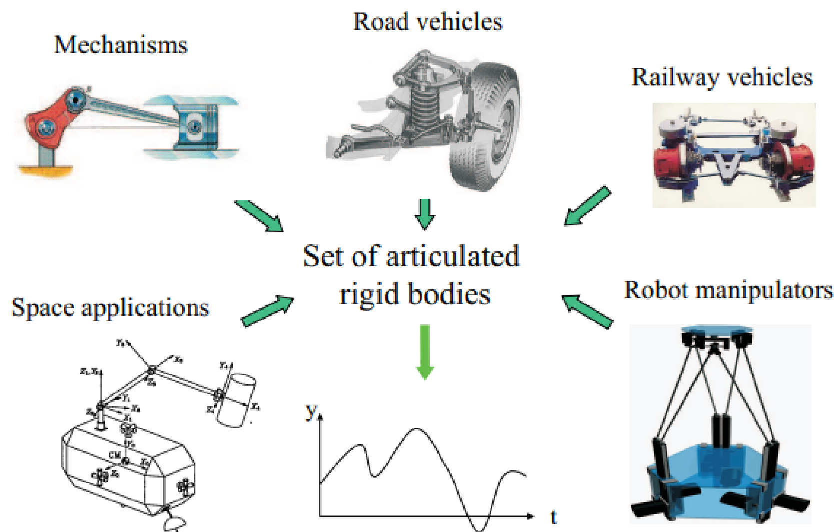


Figure 3.1: Multibody Applications

Using the symbolic approach allows Robotran to simplify the mathematical expressions both on arithmetic and trigonometric functions. Therefore reducing the number of operations and increasing the computation speed.
Moreover, Robotran is portable, meaning that once the equations related to the mechanical system are generated, the systems does not need any further programming for the characterisation of the system. Implementations are possible in Matlab, C and python, allowing the user to implement the desired functionalities. Multiple

extensions can be provided such as in fluid mechanics, granular material & contact and robotics (ROS). Simulations can be done in real-time and can be rendered in a 3D animation through a graphical interface.
Robotran has been developed for more than 30 years by the Multibody research group of the Center for Research in Mechatronics (CEREM) which is part of Institute of Mechanical, Material and Civil Engineering (iMMC) of the Université Catholique de Louvain (UCLouvain).[40]

3.1 Topological Parallelism

The topology of a system, meaning the way the bodies are connected, vary from system to system. We can't say that each system has a specific type of topology but we can explain why they lean towards one of the two absolute topologies which are serial and parallel topologies. Both are illustrated in figure 3.2 a) and b) respectively.
The serial topology is characterized by a tree-like structure which has in the most extreme cases only a single branch. The parallel topology is characterized by the existence of a closed loop between bodies. These closed-loops need to be opened in order to find back the tree-like structure of serial systems. For that purpose, we will cut either a body or a joint in two.

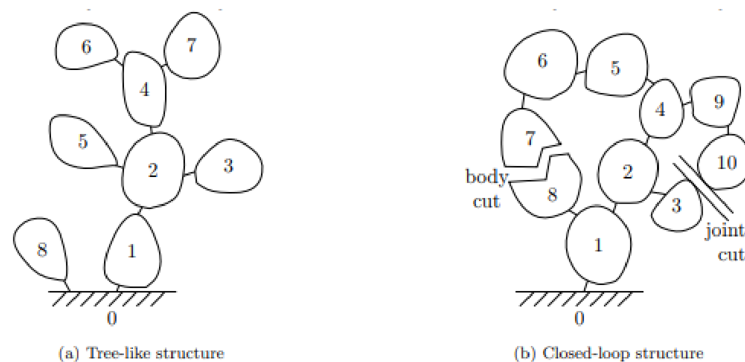


Figure 3.2: System topology

Parallel systems have therefore multiple kinematic chains, succession of bodies connected with joints. This allows some formalism to differentiate the successive chains and enable parallel computation. Such parallel systems are used in our everyday life such as illustrated in figure 3.3, like a 4-bar system, a railway-bogie, a parallel robot or the suspension system in a vehicle.

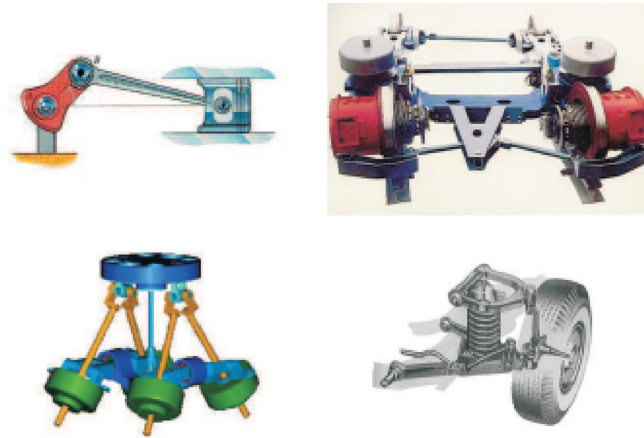


Figure 3.3: Example of closed-loop multibody systems

3.2 Study Cases

As the amount of multibody systems is very large we have to prove that our findings are valid and representative for most of the MBS. This will highlight similarities and differences across the different categories we could divide a range of MBS into. To do this, we took the two systems we initially used to develop the algorithm, a 23-dof railway-bogie and a 70-dof system, and we added on top of these a set of representative MBS. These systems vary both in their size and their topological/morphological characteristics. The variations are first on the number of dof; we have chosen to take 23-,50-,70-,100- and 130-dof systems, and for each we will choose 3 systems with different topological characteristics as explained in section 3.1 Topological Parallelism, these are : a multibody system of serial topology, a multibody system of parallel topology and a multibody system of a intermediate parallel topology. This gives a total of 17 cases that can be compared with each other to draw strong conclusions.
As we wanted to show the effectiveness of the leads we presented in section 2.4Leads on improving cycles, we applied the tree swapping technique manually on the systems of 23-dof and 50-dof, mainly to show his potential. We will compare these against their initial counterparts. It should be noted that the potential isn't limited to our manual work as this would be easier with the appropriate software tooling. Unfortunately, we weren't able to develop an implementation of the leads in the algorithm in the frame of this thesis, therefore we satisfy ourselves with the manual changes we applied which will raise the interest in this approach.
The cases and their characteristics are presented in the following table 3.1. We included the number of operations we are computing for each system and the length of the critical path both in the simple unitary weight and in our specific latency. Additionally we computed a fourth characteristic which is a number Y which corresponds to the number of operations divided by the ASAPCycle, we will explain this further in section 3.3.

Characteristics comparison - From the table 3.1, we see that the amount of operations is both correlated to the topology of the systems and the amount of dof. First with respect to the topology, we see that the number of operations can vary from 4 to 10 times between a parallel topology and a serial topology. Second with respect to the number of dof, there is an increase of the number of operations that is linked to the increase of dof but that increase does not follow a linear or quadratic evolution. The evolution on the dof seems to be mostly linked to the specific complexity of simplicity of the MBS. While the reason of the difference on the topology is due to the length of the kinematic chain, which is shorter in a parallel MBS as we have multiple kinematic chains compared to a a single or two in the case of a serial MBS.

	Number of Operations	ASAP-ASAPCYCLE	Number Y
23-dof railway-bogie	5469	69 - 737	7.42
23-dof serial	19970	342 - 3968	5.03
23-dof semi-serial	9150	155 - 1784	5.12
23-dof parallel	6410	136 - 1576	4.06
23-dof railway-bogie - Modified	5469	47 - 471	11.61
23-dof serial - Modified	19970	115 - 1239	12.25
23-dof semi-serial - Modified	9150	53 - 560	15.35
23-dof parallel - Modified	6410	43 - 460	13.38
50-dof serial	93525	757 - 8793	10.64
50-dof semi-serial	40348	349 - 4042	9.98
50-dof parallel	15187	206 - 2416	6.28
50-dof serial - Modified	93525	266 - 2939	31.82
50-dof semi-serial - Modified	40348	107 - 1228	32.85
50-dof parallel - Modified	15187	56 - 626	24.26
70-dof	100143	662 - 7703	13
70-dof serial	194946	1077 - 12488	15.61
70-dof semi-serial	37847	260 - 3024	12.51
70-dof parallel	19444	107 - 1228	15.83
100-dof serial	367118	1517 - 12488	29.39
100-dof semi-serial	66960	253 - 2935	22.81
100-dof parallel	42676	343 - 4050	10.53
130-dof serial	620257	1965 - 22874	27.12
130-dof semi-serial	140783	427 - 4963	28.36
130-dof parallel	68844	462 - 5473	12.58

Table 3.1: Study Cases Characteristics

On the critical path, we see that most of the operations are additions or subtractions, because the ratio between ASAP and ASAPCycle is in almost all cases higher than 10 and closes to 12 in most MBS. Therefore an improvement on the addition-subtraction block could be much more beneficial to the computation time of our simulation, even if it would be detrimental to the multiplication block.
Regarding the ASAP and ASAPCycle attribute, we can highlight the same tendencies as for the number of operations. These will be lower in the case of a parallel MBS compared to a serial MBS of the same dof and will increase with respect to their dof without respecting a linear or quadratic evolution.

3.3 Minimal and effective Cycle Time

In the PhD Thesis of Postiau [3], he proposed methods to find the number of units of each type needed in order to have a certain execution time, or in another way to have the maximal parallelisation rate with still a correct execution time.

We propose a method, which we call lower bound guidance, which uses the maximal path with the maximal asap attribute and two assumptions:

1. The first assumption is to consider that the number of additions/subtractions is equal and therefore that the number of PE is equally divided between both types of operations.

2. Neither type of operations will be delayed because of a bottleneck in the computation of the other type of operations.

The method's objective is to guide the number of cycles and provide a reliable prediction for the number of cycles we will obtain, given a certain number of PE and the minimal number of PE to reach the minimum number of cycles.

The method uses two distinct cases. In the first case, the dependence of the equations does not impact the reduction in time. In the second case, the cycle time is constrained by the length of the critical path. Therefore, the lower bound guidance is made out of two curves corresponding to theses cases:

1. First curve corresponds to the number of cycles we would have considering a perfect parallelisation rate of 100%.

2. Second curve corresponds to the critical path defined by the ALAPCycle, as we are not able to compute faster than the critical path.

Therefore, as we mentioned in section 2.4Leads on improving cycles, we can reduce the critical path length. This will result in a lower second curve.
At the intersection of both curves, we have a point which we called previously the number Y. This number corresponds to the point where we transition from a parallelisation to a saturation of this parallelisation. In others words, we cannot to reduce the number of cycles. This is possible considering our hypothesis, if we indeed have the same amount of both operations and that none of the type of operations will restrict the placement algorithm.
We are able to see it in two ways:

1. The first is that we have enough PE to insert an operation on the critical path when their parents are computed. This is possible if the number of

PE is at least greater than the number of operations divided by ASAPcycle. Because the rest of the PE could handle the rest of the operations which are not on the critical path.

2. The second is that we need to have a PE, or in this instance a fraction of the PE, free for the critical path, in order to have that free fraction, we need to have a number of PE that is at least equal to the number of operations divided by ASAP.

We will therefore describe three different zones in the following figures, which are called parallelisation zones, transition zone and saturation zone:

1. Parallelisation zone before the number Y, where the addition of PE allows a significant reduction in the cycle time while holding a significant parallelisation rate;

2. Transition zone around the number Y, where the addition of PE still brings a reduction of the cycle time but at the cost of the parallelisation rate

3. Saturation zone after the number Y, where the addition of the PE does not reduce the cycle time but still reduces the parallelisation rate.

This gives a clearer view of the resources that will be needed. On top of this, it allows in case we want to reduce the cycletime to the maximum to effectively find a number of PE and then by iterating increase this number.
We therefore studied the systems and verified this for all systems. We can respectively see the 23-dof, 50-dof, 70-dof, 100-dof and 130-dof systems in figure 3.4, 3.5, 3.6, 3.7 and 3.8. We include for the 23-dof and 50-dof systems the cases where we decreased the critical path. The figures 3.4 and 3.5, which include the systems with a reduced critical path, show that the decrease of the critical path causes the second curve to lower and the number Y to shift towards the right, effectively showing that these method could further improve the decrease in computing time we are looking for.

More specifically, we see that for MBS that contain a low number of dof, we have numbers Y which is quite low, around 4 to 7, while for MBS with higher numbers Y we will increase until almost 30 for serial systems.
We see that for the systems where we decreased the critical path, we can see an increase by 3 of the number Y, both for the 23-dof systems and for the 50-dof systems. We need to remind that the number Y takes into account the latency of

the two types of operations. Therefore in fact, each incremental increase of the number Y represents the equivalent of several non-pipelined computing units.

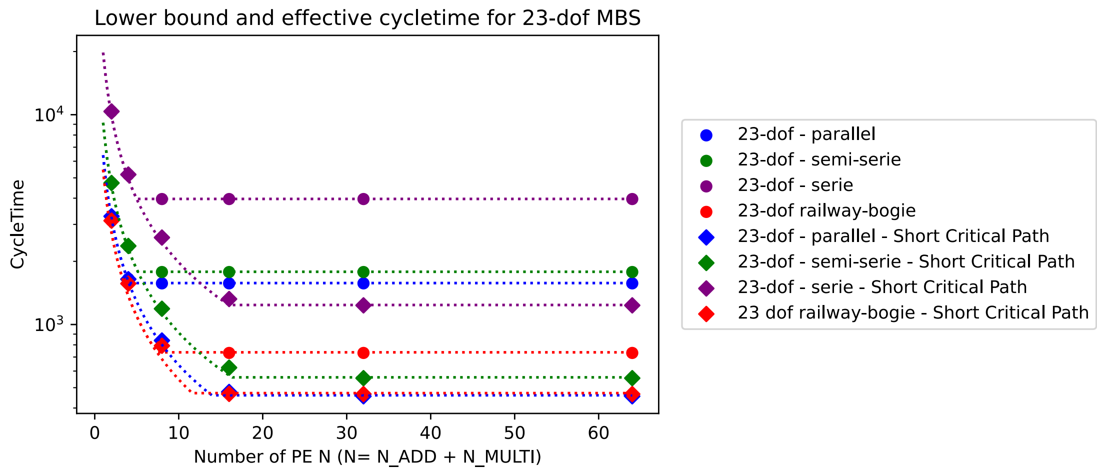


Figure 3.4: Cycletime for the 23 dof

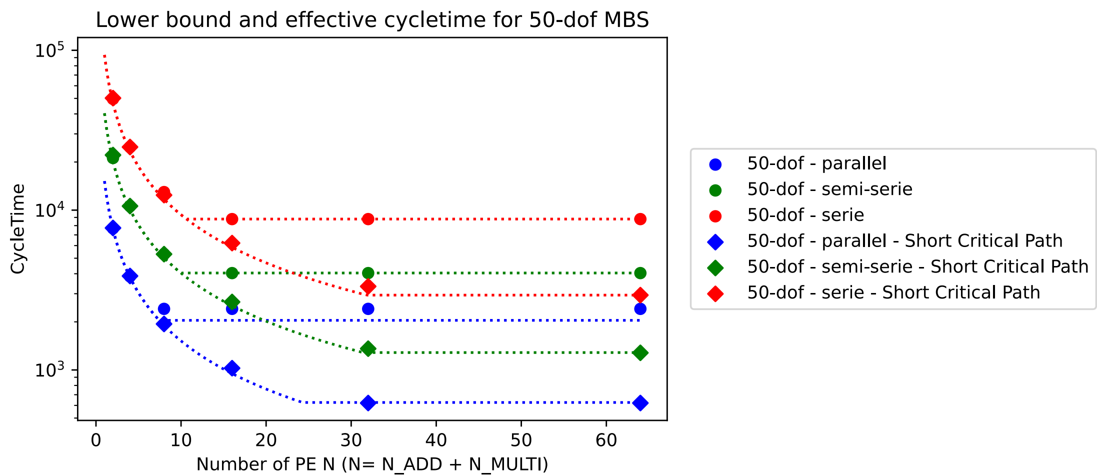


Figure 3.5: Cycletime for the 50 dof

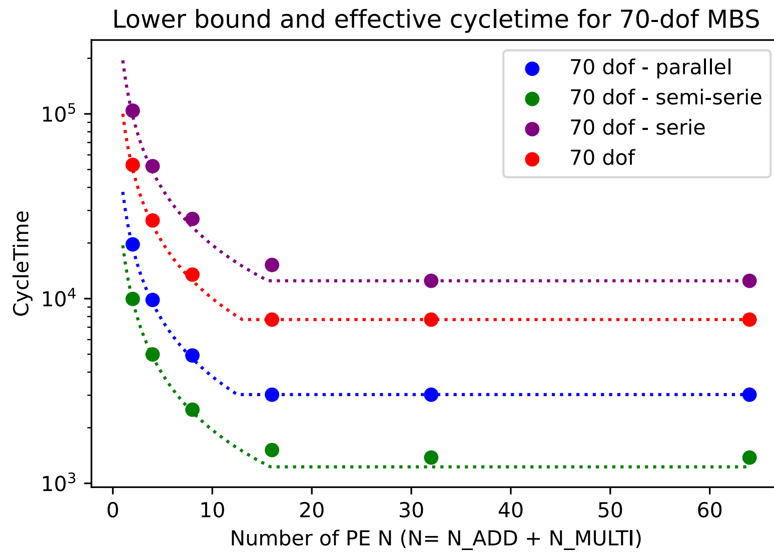


Figure 3.6: Cycletime for the 70 dof

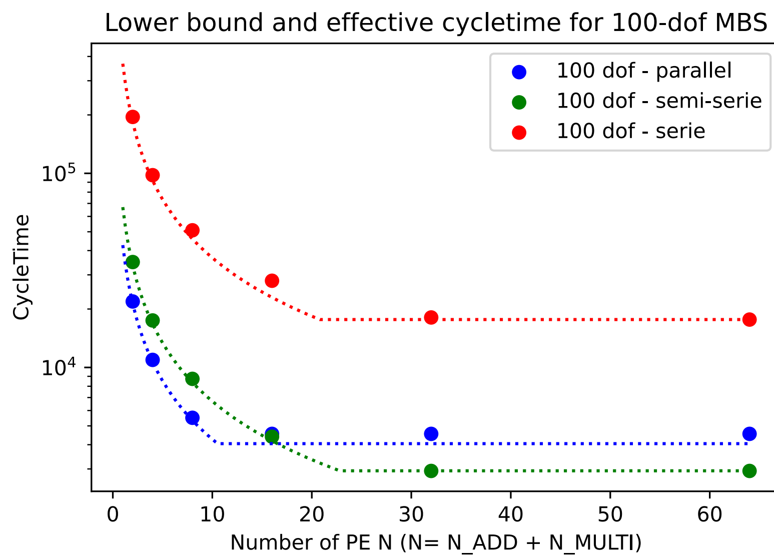


Figure 3.7: Cycletime for the 100 dof

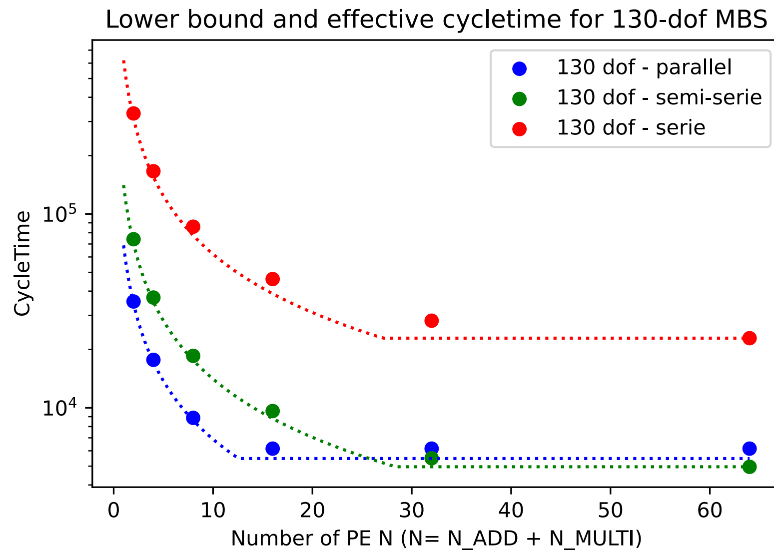


Figure 3.8: Cycletime for the 130 dof

3.4 Parallelisation Rate

A second way of looking at the vectorisation, is to look at the parallelisation rate of the systems at the different configurations with a varying number of PE.
The parallelisation rate will be computed by comparing the amount of operations the system contains divided by the maximal amount of operations the configuration could compute given the configuration and the cycletime taken to make the computations.

We can see the parallelisation rate in the following figures 3.9, 3.10, 3.11, 3.12 and 3.13, which represent respectively the parallelisation rate of the 23-dof, 50-dof, 70-dof, 100-dof and 130-dof. We included the 23-dof and 50-dof with the reduction of the critical path.

Regarding the differences between different topology, we will again divide the study into three cases or zones, these are the parallelisation zone, the transitional zone and the saturation zone as described in 3.3.

In the parallelisation zone, we notice that more parallel systems tend to have a better parallelisation rate compared to the more serial systems. This is mainly due to the existence in those systems of several kinematic chains which allows the equations to be less dependent from each other therefore achieving a better parallelisation rate. However some exceptions can be exist, because the parallel systems contain less operations than their serial counterparts, sometimes with an order of magnitude. But the fact that we have more kinematic chains does have a greater effect. But the difference of the parallisation rate stays very small.
We also see that systems with a shorter critical path have a higher parallelisation rate than their counterparts showing that there is a slight decrease of the number of cycles. Therefore reducing the length of the critical path is also useful when we not yet constrained by the length of the critical path.

In the saturation zone, the parallelisation rate systematically decreases as there is no possibility to parallelise any more the operations, we are constrained by the critical path. The order of the systems are here determined by their Y number. More specifically, if we had to order all systems between each other regarding the parallelisation rate for a number of PE in the saturation zone, this could be simply made by comparing the Y number.

Regarding the systems with a lower critical path, these will present a greater parallelisation zone, because as was explained earlier in section 3.3, the Y number will be shifted to the right. They therefore present a better parallelisation rate in

that conquered zone, which was previously a saturation zone.

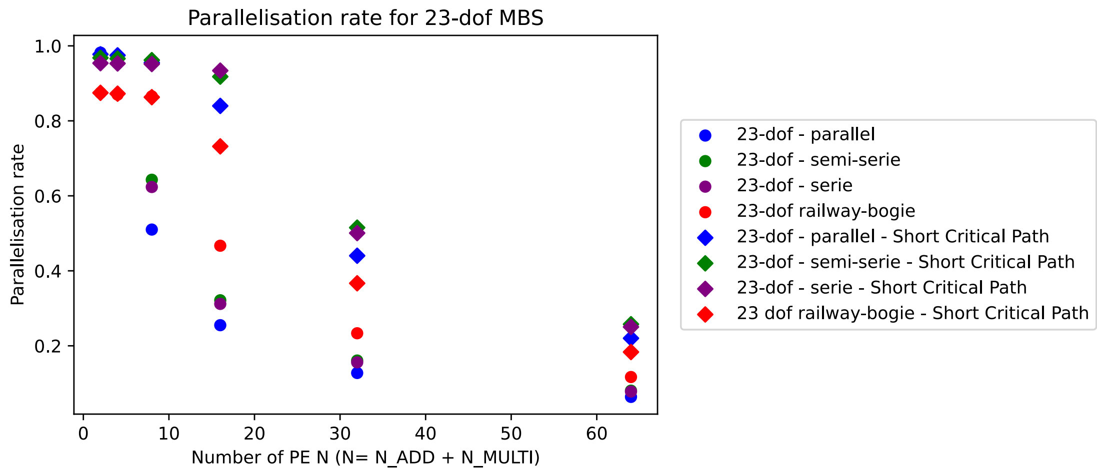


Figure 3.9: Parallelisation Rate 23-dof

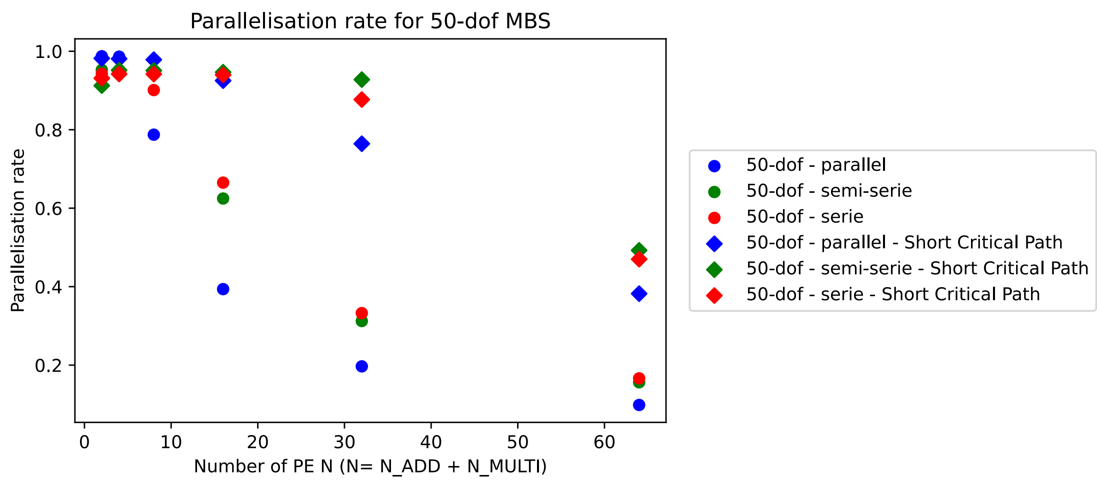


Figure 3.10: Parallelisation Rate 50-dof

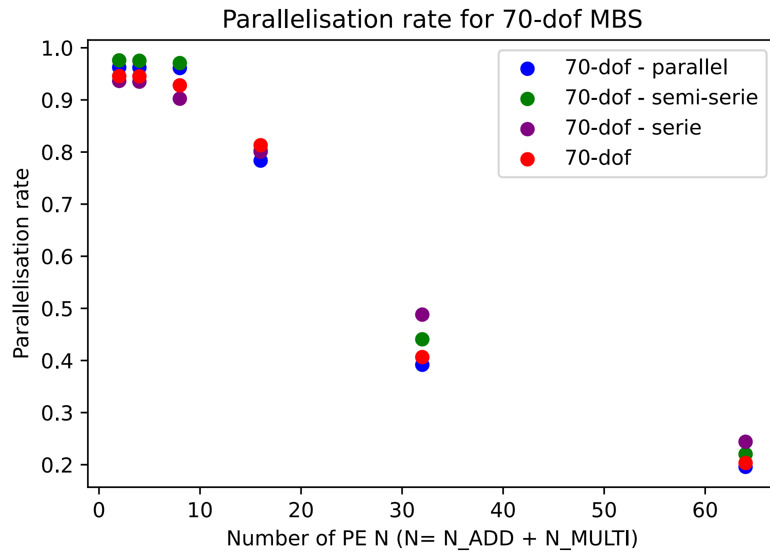


Figure 3.11: Parallelisation Rate 70-dof

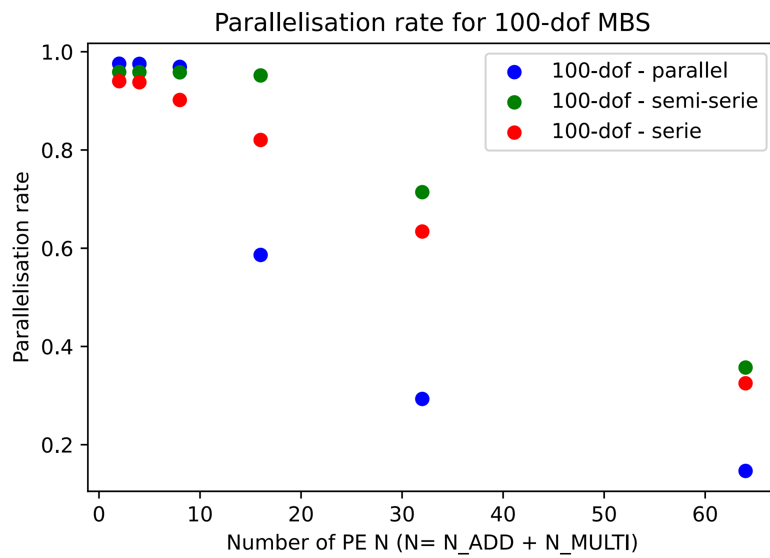


Figure 3.12: Parallelisation Rate 100-dof

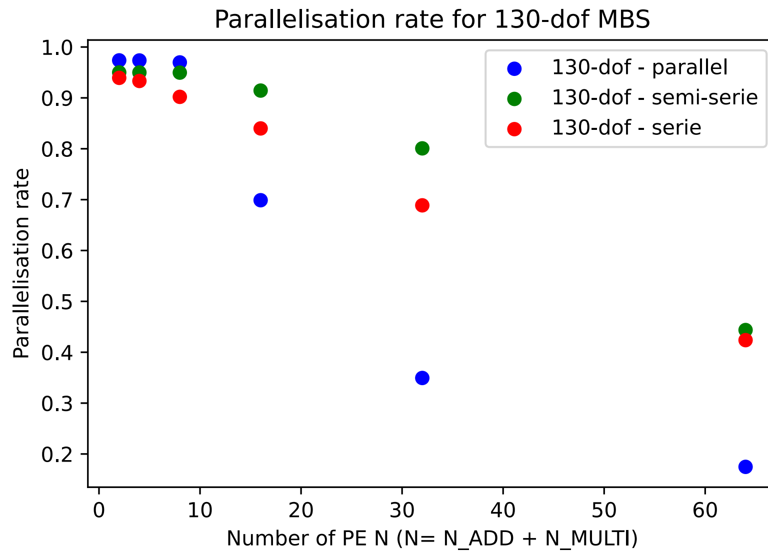


Figure 3.13: Parallelisation Rate 130-dof

3.5 Projection

As we have seen, the number Y gives the approximative number of PE we can have until reaching the maximal parallelisation possibilities offered by the system. We proved it both in sections 3.3 Minimal and effective Cycle Time and 3.4 Parallelisation Rate. Using the Y numbers we computed in 3.2, we are able to make the following figure 3.14. This figure shows the Y number for various systems given their size and their topology.

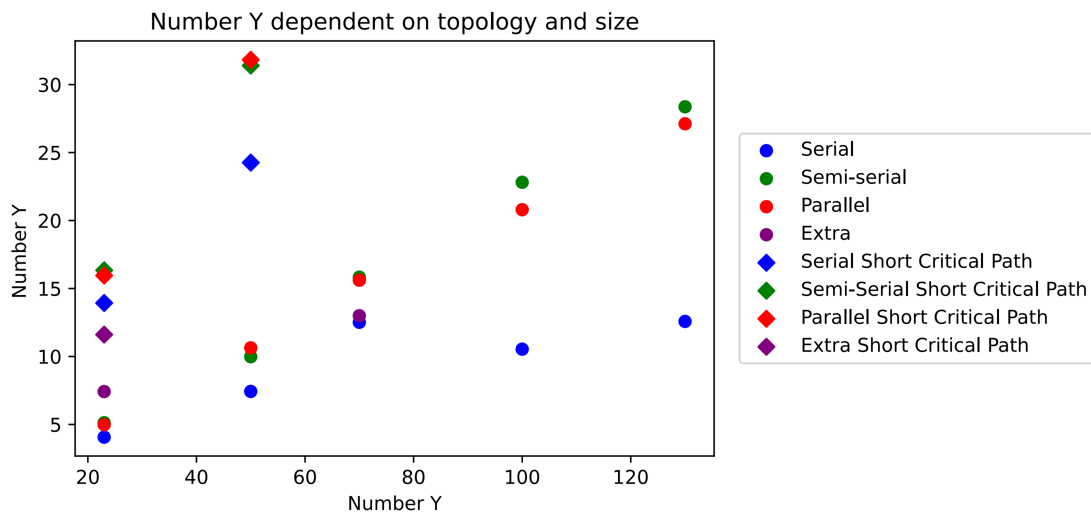


Figure 3.14: Number Y given topology and size

We can notice from the systems we studied two different projection curves, both linear. The first one is with the serial systems, the linear evolution of the number Y is slowly increasing, about 0.05 per dof. The second one is with both the semi-serial and the parallel systems, this time, the linear evolution increases faster, about 0.25 per dof.
This is only true for the systems of the topologies and sizes close to the ones we studied.

When implementing our improvement to have a shorter critical path, we improved the general number Y and its evolution. At least in theory because we only have two dof and we improved it manually. The increase can go up to three times the original number Y.

3.6 Occupation Rate of PE

In this section, we analyse the occupation rate with the different architecture configurations as in previous sections. For this purpose, we decided to plot for each MBS individually, the mean occupation rate of the PE. We took the rolling average over 50 cycle and smoothed it. This enables us to analyse properly and have readability on the repartition through time of the operations.
In the figures, the x-axis has a normalized timecycle

First, we will draw overall conclusions with a general analysis on the 23-dof railway-bogie and the 70-dof system. Second, we will make an analysis based specifically on the typology of the systems, we therefore have grouped together the systems with the same topologies: parallel, semi-serial and serial. Finally, we will conclude with the 23-dof and 50-dof systems where we reduced the length of the critical path.

We have the occupation rate for the 23-dof railways and the 70-dof MBS respectively in figure 3.15 and 3.16.
The first point is that we can once again see the three zones, we talk here about three types of cases which have the same meaning, parallelisation case, transitional case and saturation case.
The parallelisation cases are characterized by a rather flat occupation rate, with eventually a buildup phase, but they will usually be able to occupy the PE units until the very end. While the saturation case, is characterized by a more oscillating occupation rate. As in the parallelisation case, we have the same buildup phase, which can either be followed by ; a peak as for the 70-dof for N = 64, ending with a sharp decrease in the occupation rate; a plateau which will know a decrease after some time, as for the 23-dof railway-bogie for N=32; or a simple decrease after a build-up, as for the 23-dof railway-bogie for N=64. We also see that the transition zone is passed when we have at the very end of the cycletime a decrease of the occupation rate. This is due to the fact that all the operations out of the critical path are being computed and we are left with those on the critical path. In that scenario, we need to wait the end of the computations to be able to insert the following equations, which impacts the occupation rate.

When we increase the number of PE, we will have that phenomenon earlier and earlier on. This means that the operations which are not on the critical path, will be computed earlier. This leaves us earlier with only the critical path to compute. While decreasing the length of the critical path is crucial in order to further improve the parallelisation potential of the MBS, we need to point out that during the early stages of the build-up phase, the occupation rate can be quite low as illustrated in figure 3.16. Therefore we will need to compensate this later on, which could block

our configuration from reaching his optimal cycletime.

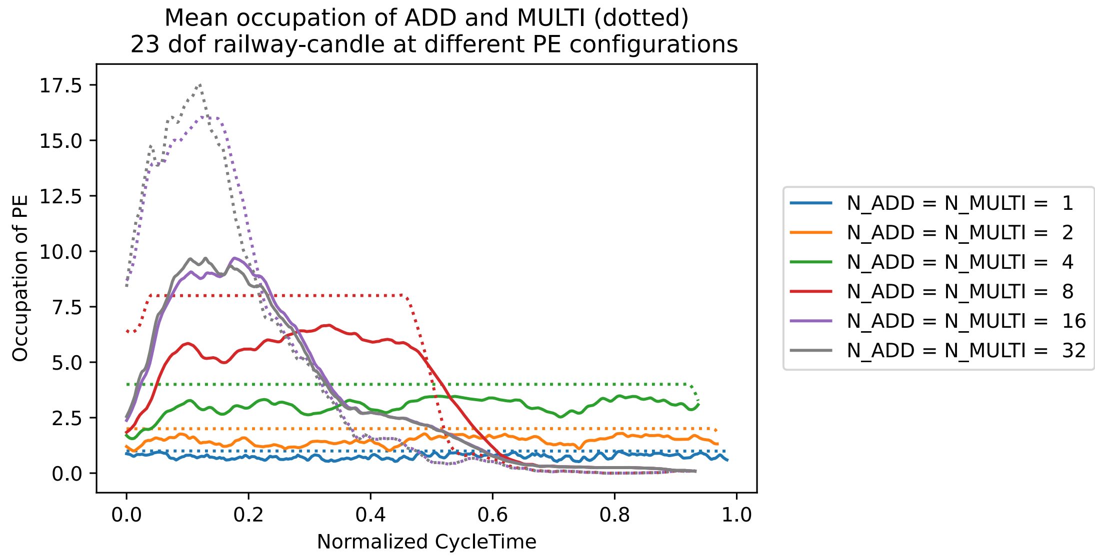


Figure 3.15: Occupation 23-dof railway-bogie

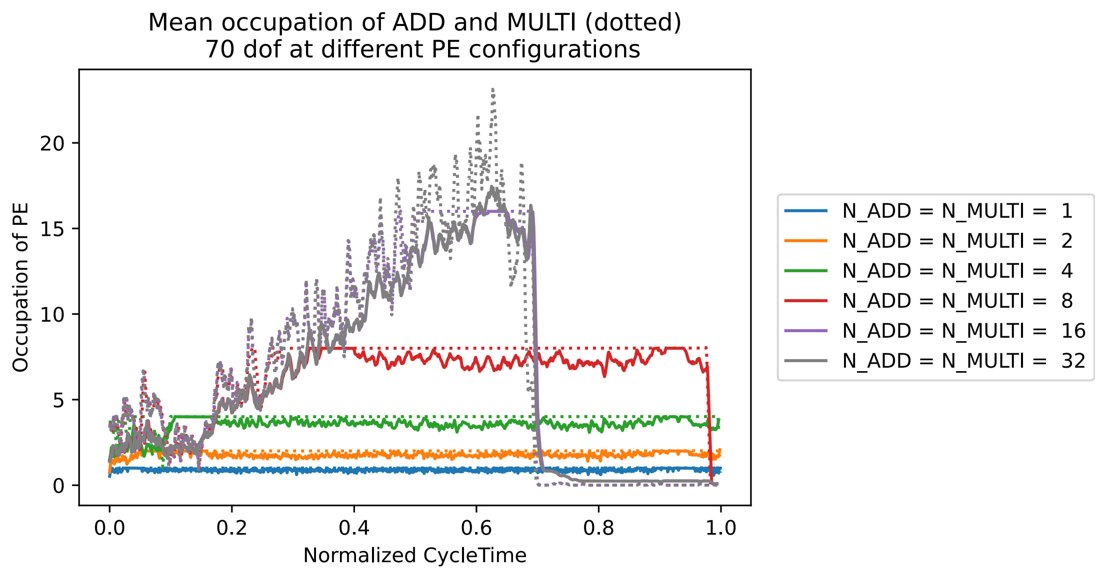


Figure 3.16: Occupation 70-dof

3.6.1 Serial Topological Systems

The mean feature in serial topological systems is the almost linear buildup we can observe during the saturation, eventually followed by a plateau phase where in fact the buildup is limited by the amount of PE of the configuration.
Therefore, when we increase the number of PE, we are able to continue the buildup until the severe drop we noticed in the section 3.6 Occupation Rate of PE. It should be noted that the occupation rate of this final part of the cycletime stays very low as we saw in the section 3.6 Occupation Rate of PE.
This is mainly visible in the figures 3.18 and 3.19, which represent the 50-dof and 70-dof serial MBS. These present both two configurations in the saturation zone.
For the 23-dof serial, illustrated in figure 3.17, we have in fact two buildup phases, first with a spike in the multiplication then with the linear build-up. One of the explanations could simply be the low number of equations which is about 20000.
For the 100-dof and 130-dof serial, illustrated in figures 3.20 and 3.21, although we do not have two saturation cases, the buildup is successively shifting to the right as we increase the number of PE because the number of cycles decreases.

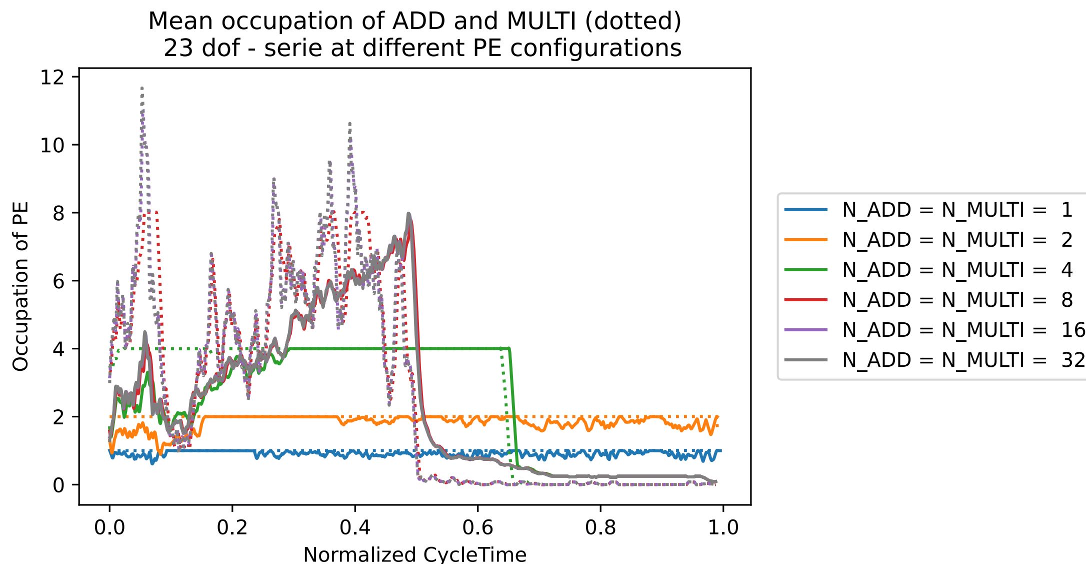


Figure 3.17: Occupation 23-dof serie

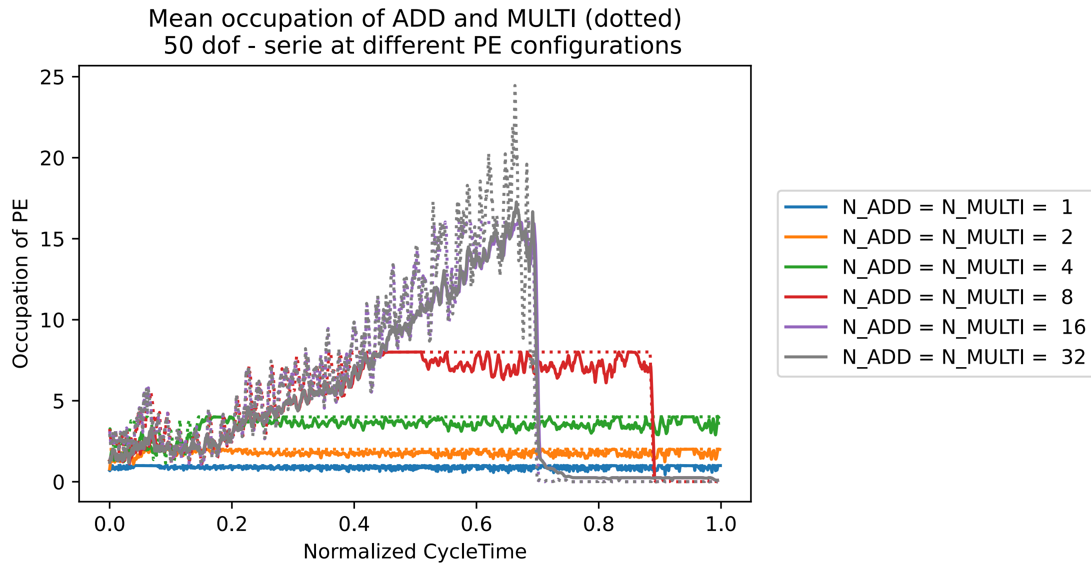


Figure 3.18: Occupation 50-dof serie

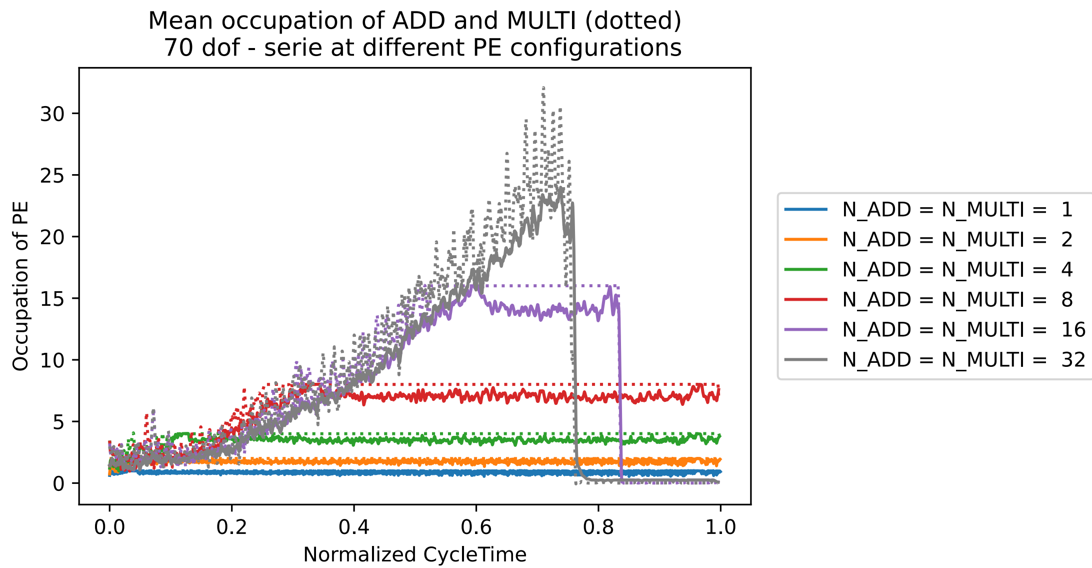


Figure 3.19: Occupation 70-dof serie

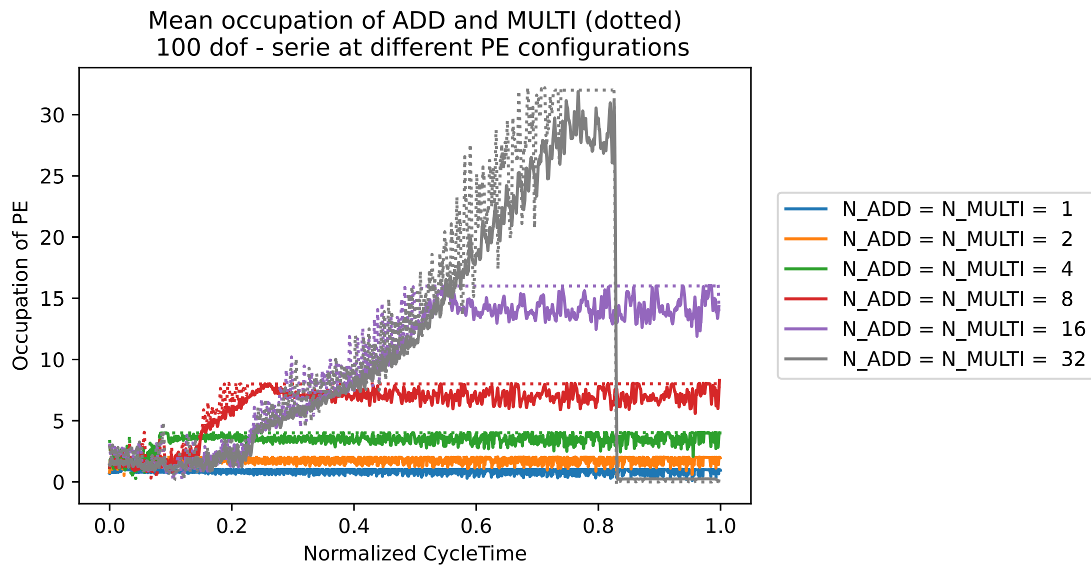


Figure 3.20: Occupation 100-dof serie

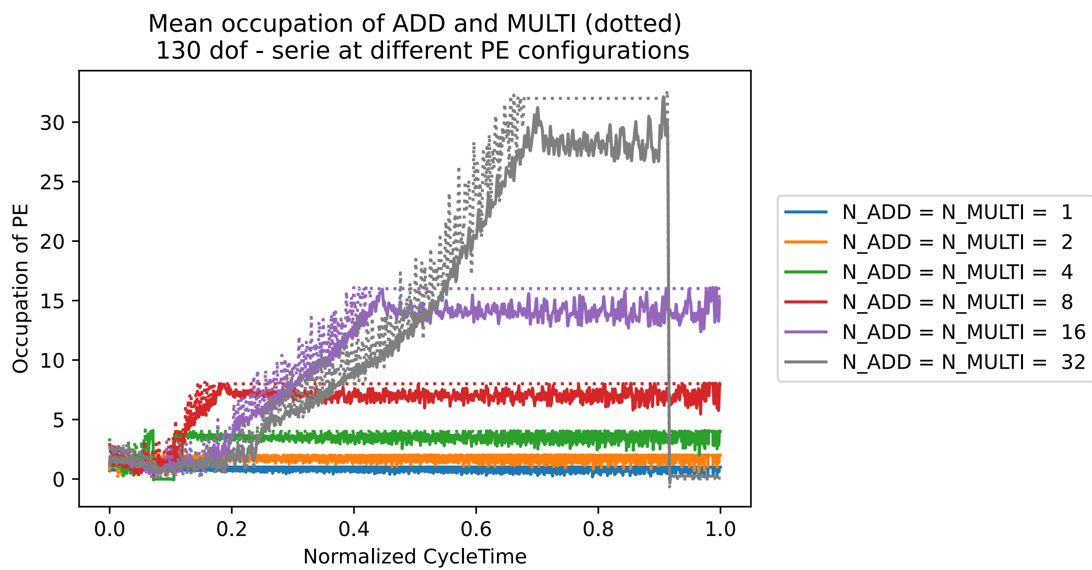


Figure 3.21: Occupation 130-dof serie

3.6.2 Semi-Serie Topological Systems

For semi-serial MBS, we see also an almost linear buildup, but this is preceded by a first build-up with a high spike in the occupation rate of the MULTI PE compared to the ADD PE.
We notice to some extend that the build-up is starting at a higher occupation rate compared to their serial counter parts and that the linear increase is less steep. This is caused by the existence of multiple shorter kinematic chains.
The occupation rate of the semi-serial systems; 23-, 50-, 70-, 100- and 130-dof, are illustrated respectively in figure 3.22, 3.23, 3.24, 3.25 and 3.26.

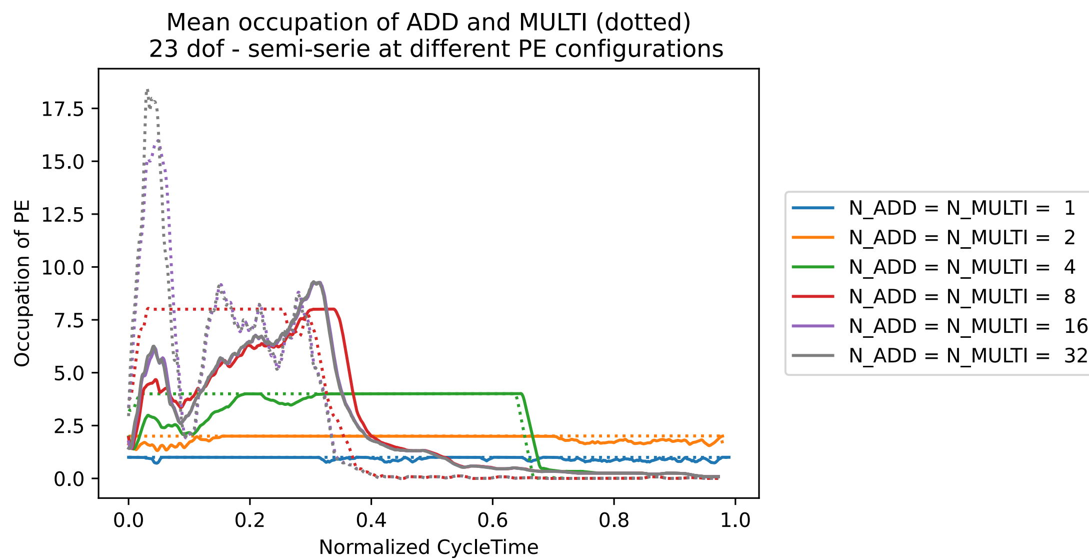


Figure 3.22: Occupation 23-dof semi-serie

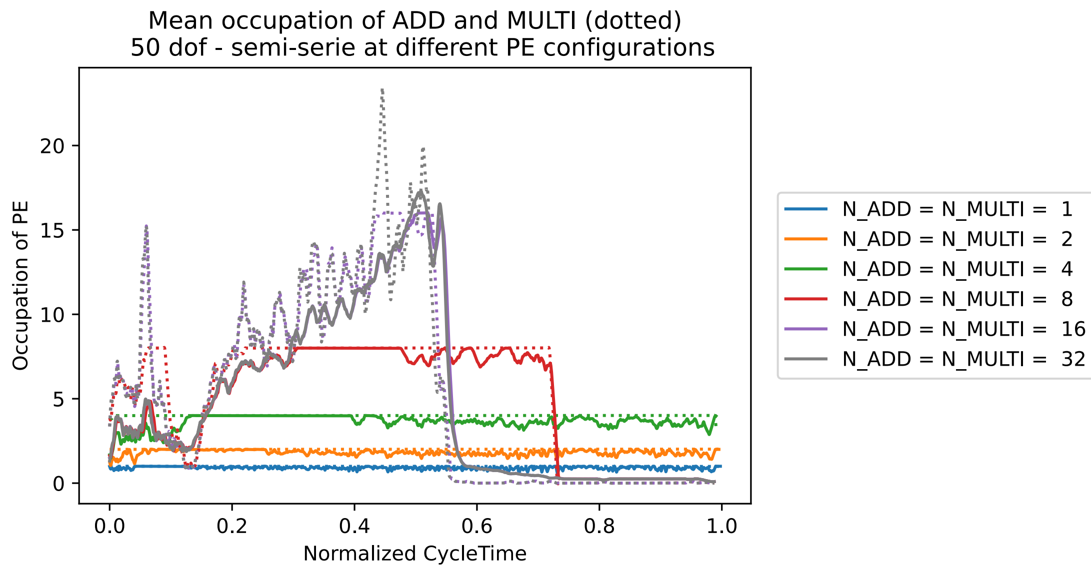


Figure 3.23: Occupation 50-dof semi-serie

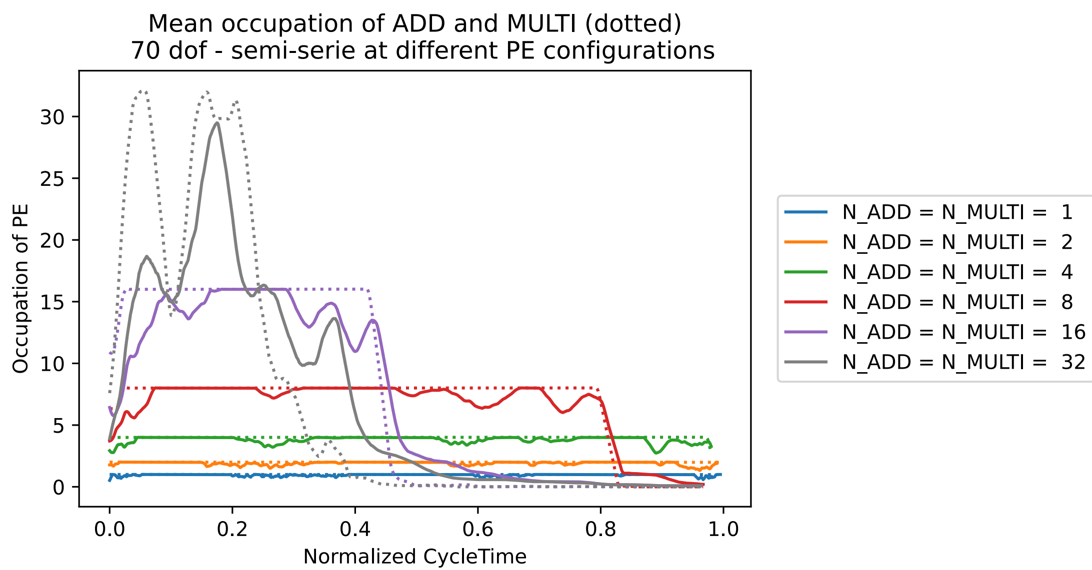


Figure 3.24: Occupation 70-dof semi-serie

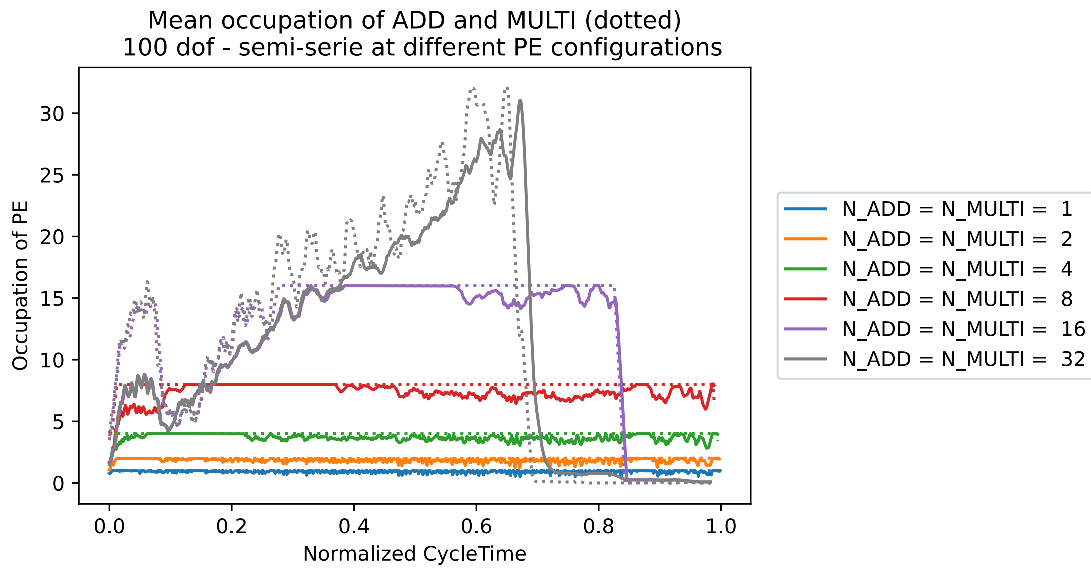


Figure 3.25: Occupation 100-dof semi-serie

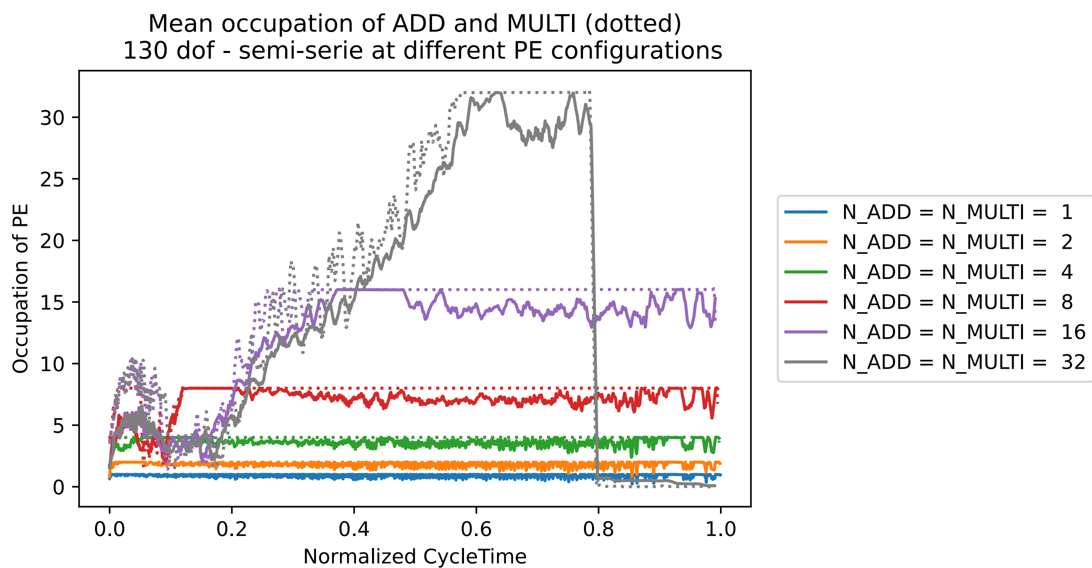


Figure 3.26: Occupation 130-dof semi-serie

3.6.3 Parallel Topological Systems

Regarding the parallel MBS, we have as we saw in 3.6.2 Semi-Serie Topological Systems, two distinct build-ups. First a small build-up as we had for the semi-serial MBS, followed now by a steep increase of the occupation rate that can either stay on a plateau when the amount of PE is close to the transition zone, or that can present oscillations when being in the saturation zone.
We have differences on the occupation rate between the MULTI PE and the ADD PE. The MULTI PE is more occupied than the ADD during the first small build-up: up to 3 times, which is greater than for the semi-serial MBS. And during the second build-up, where the MULTI PE oscillates more than the ADD PE.
The occupation rate of the parallel systems; 23-, 50-, 70-, 100- and 130-dof, are illustrated respectively in figure 3.27, 3.28, 3.29, 3.30 and 3.31.

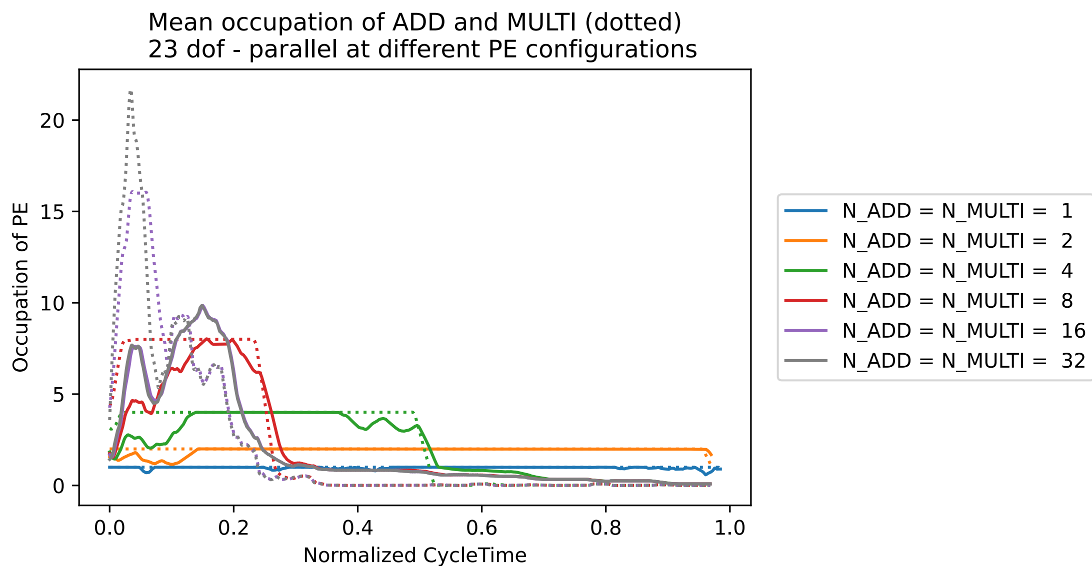


Figure 3.27: Occupation 23-dof parallel

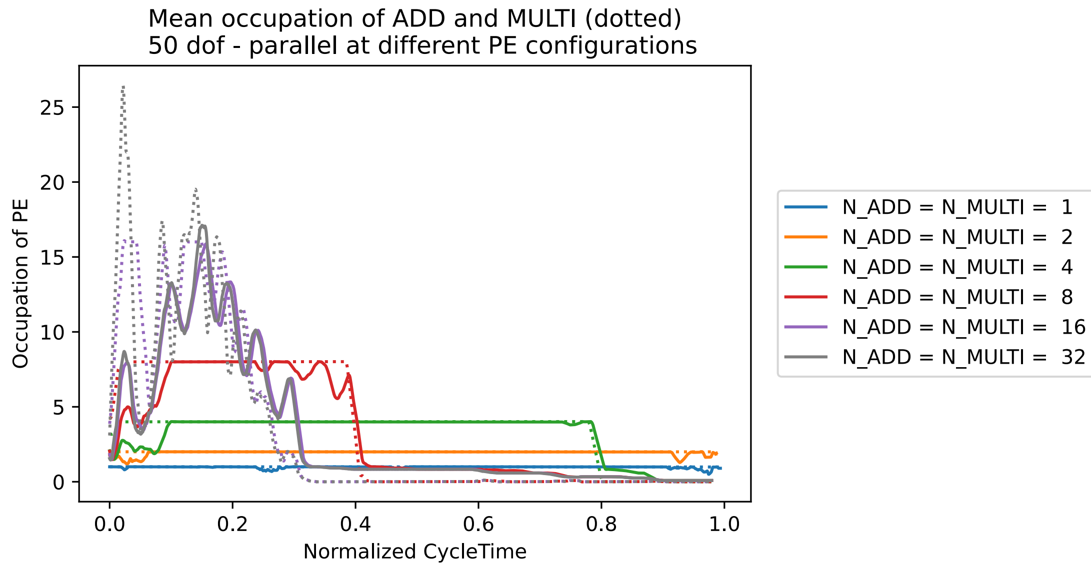


Figure 3.28: Occupation 50-dof parallel

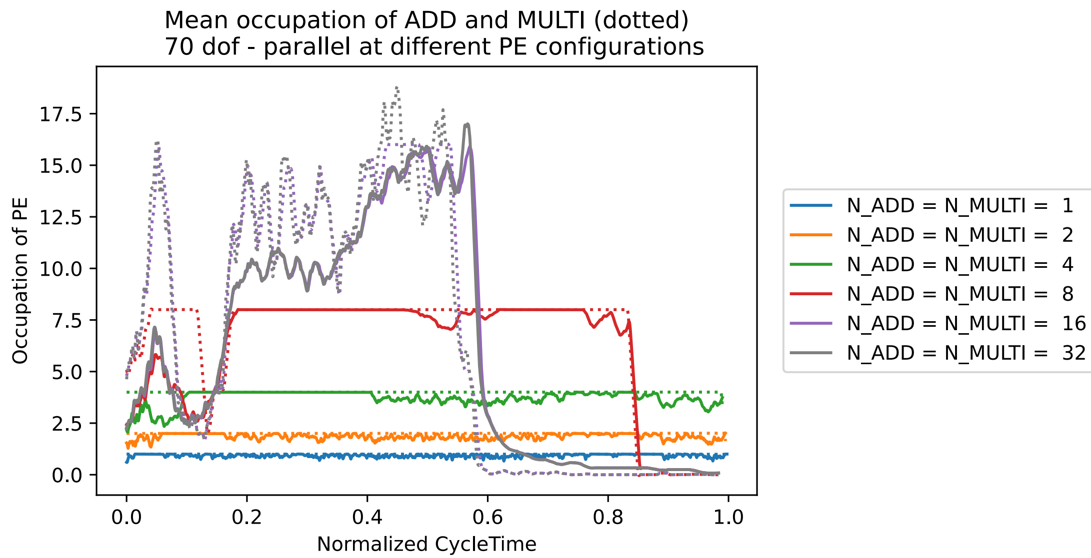


Figure 3.29: Occupation 70-dof parallel

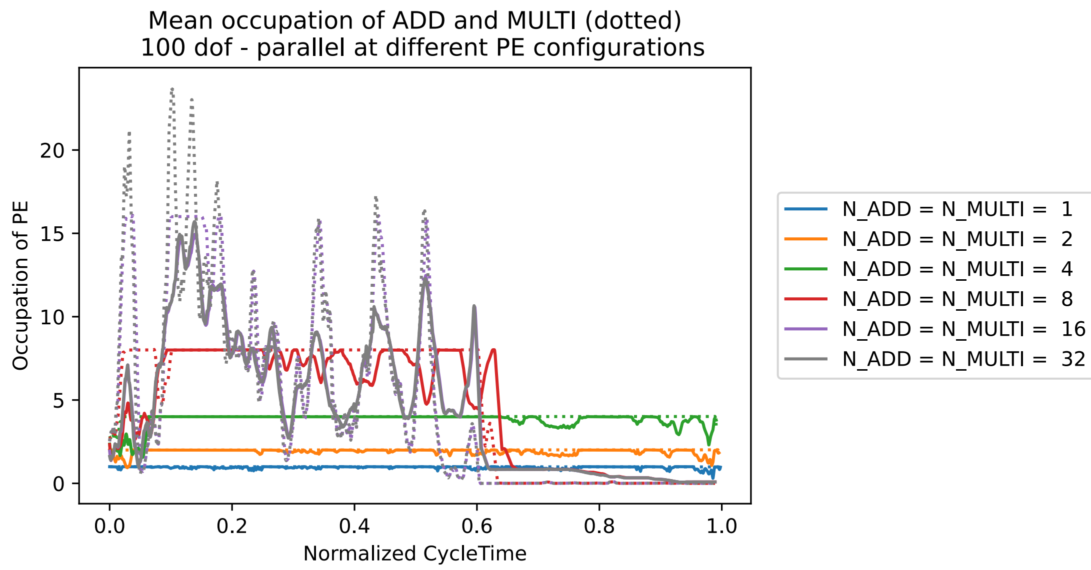


Figure 3.30: Occupation 100-dof parallel

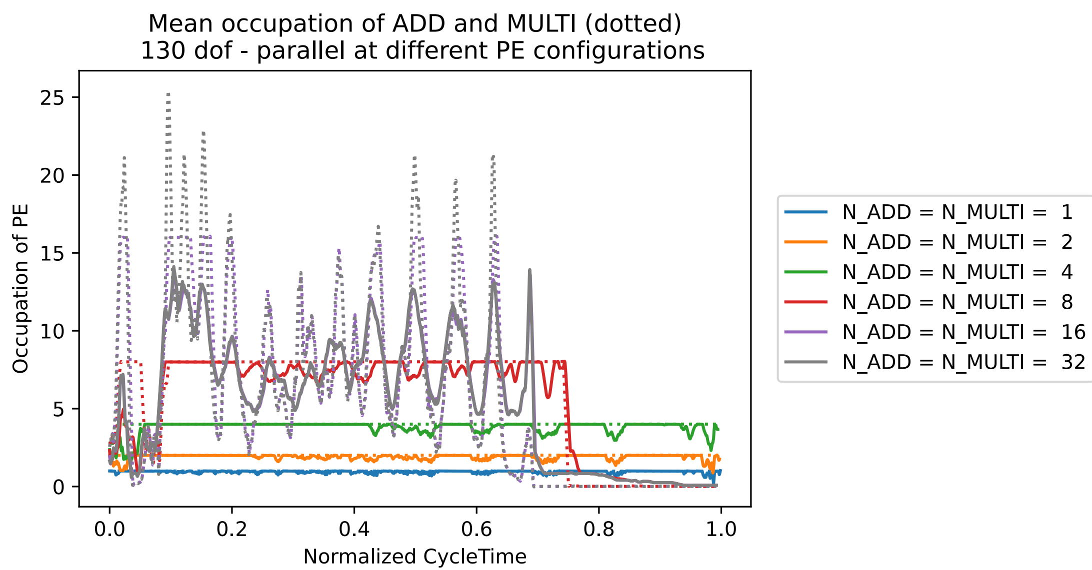


Figure 3.31: Occupation 130-dof parallel

3.6.4 Critical Path Reduction

We saw previously that the occupation rate of the systems presents two areas of improvement: the first stage of build-up and the critical path closing. Because of the reduction in critical path length, we expect the occupation rate to improve overall.
The occupation rate of the 23-dof railway-bogie, 23-dof serie, 23-dof semi-serial, 23-dof parallel, 50-dof serie, 50-dof semi-serial and 50-dof parallel MBS are illustrated respectively in figures 3.32, 3.33, 3.34, 3.35, 3.36, 3.37 and 3.38.
The most general observation on all occupation rates compared to their counterparts is that the curves aspect are the same eventhough the amount of PE we can setup is higher. Additionally, we have a shift of this curve towards the right as in some sense we are cutting off the end of the cycletime. More specifically, the occupation rate is proportionally higher at the start of the buildup and after the build-up drop, compared to the top of the occupation rate curve. The reason for the end of the cycletime to have a better occupation rate is that we reduced the length of the longest critical path and any other path that would take his place. Therefore in the end, we have more paths that have a length close to the current length of the critical part.

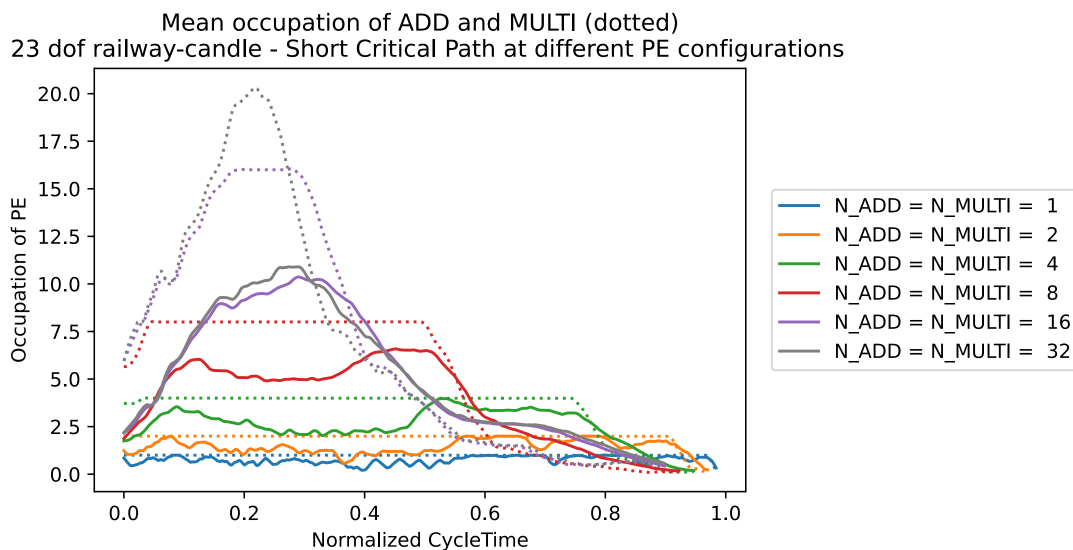


Figure 3.32: Occupation 23-dof - railway-bogie - lowered critical path

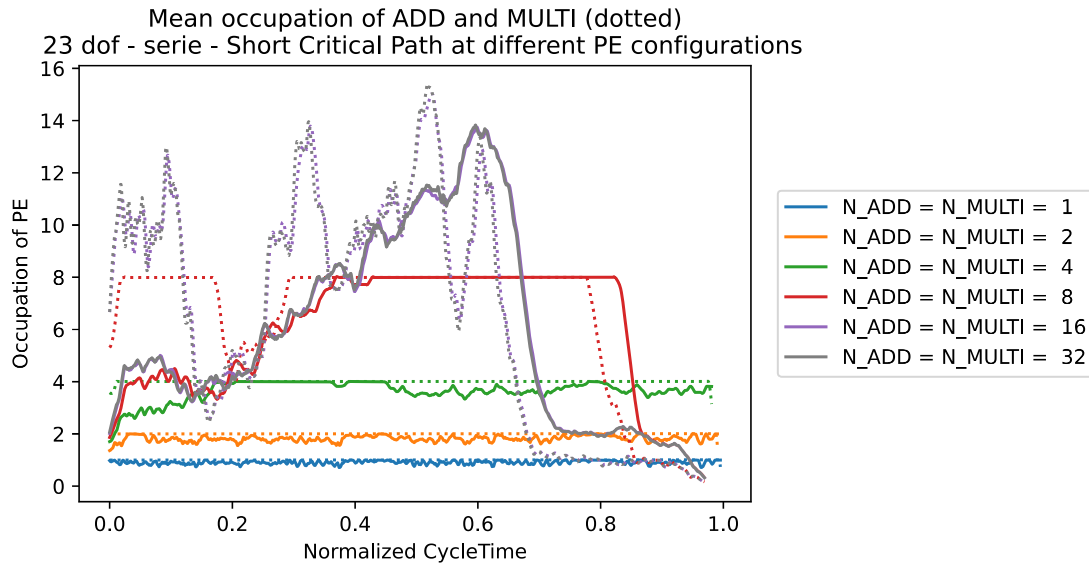


Figure 3.33: Occupation 23-dof - serie - lowered critical path

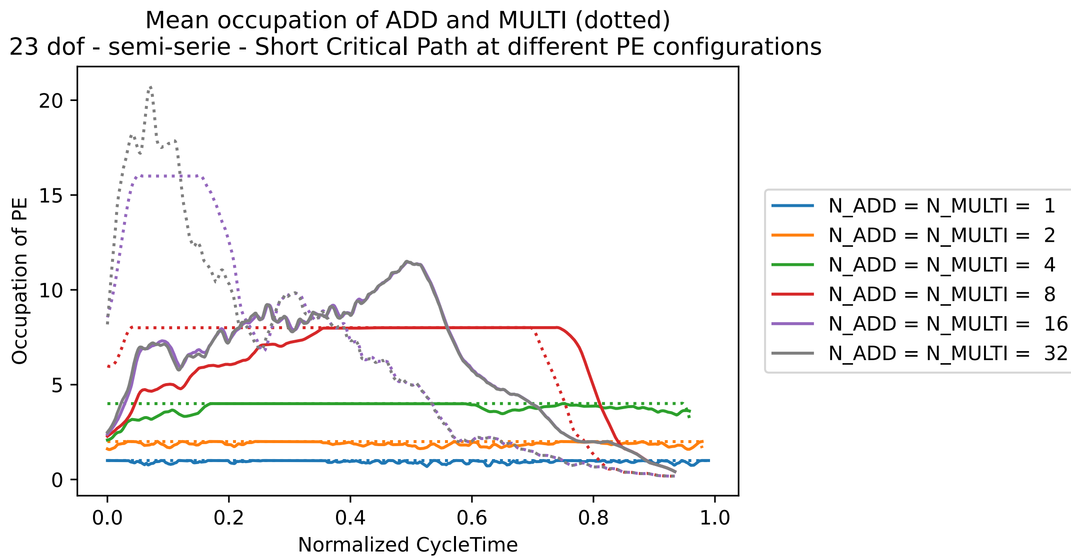


Figure 3.34: Occupation 23-dof - semi-serie - lowered critical path

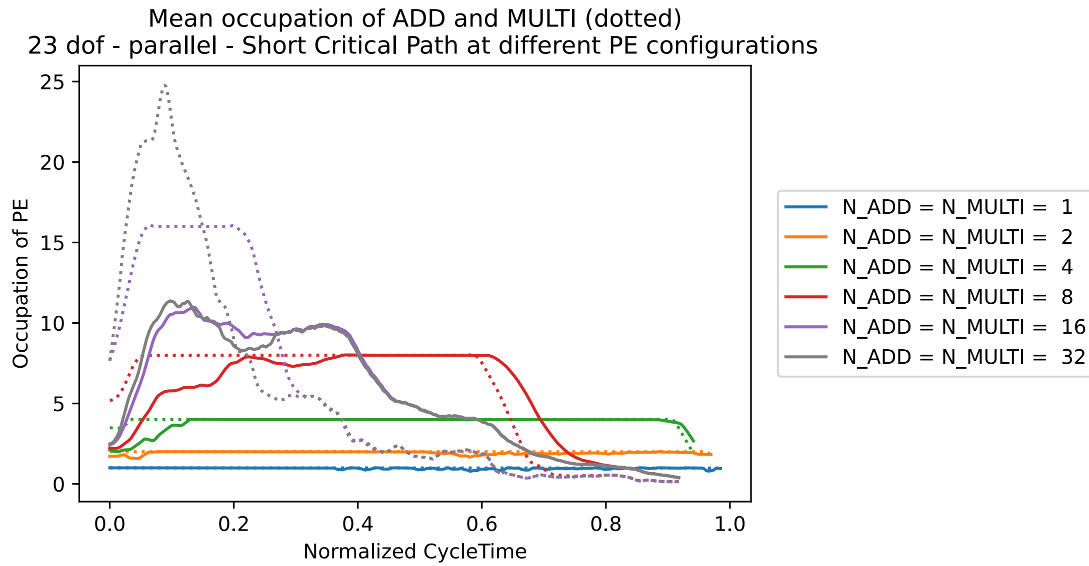


Figure 3.35: Occupation 23-dof - parallel - lowered critical path

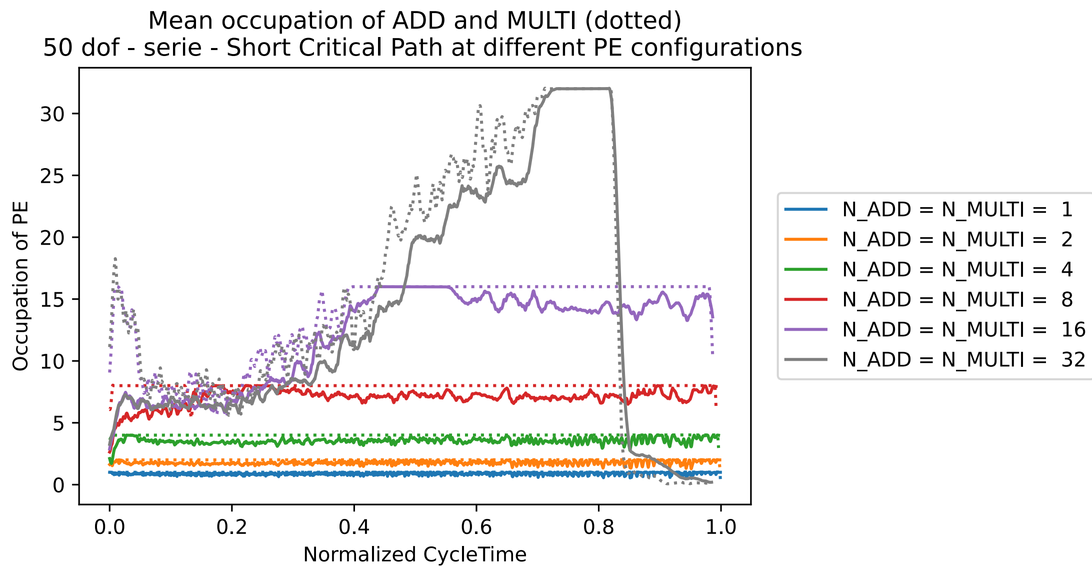


Figure 3.36: Occupation 50-dof - serie - lowered critical path

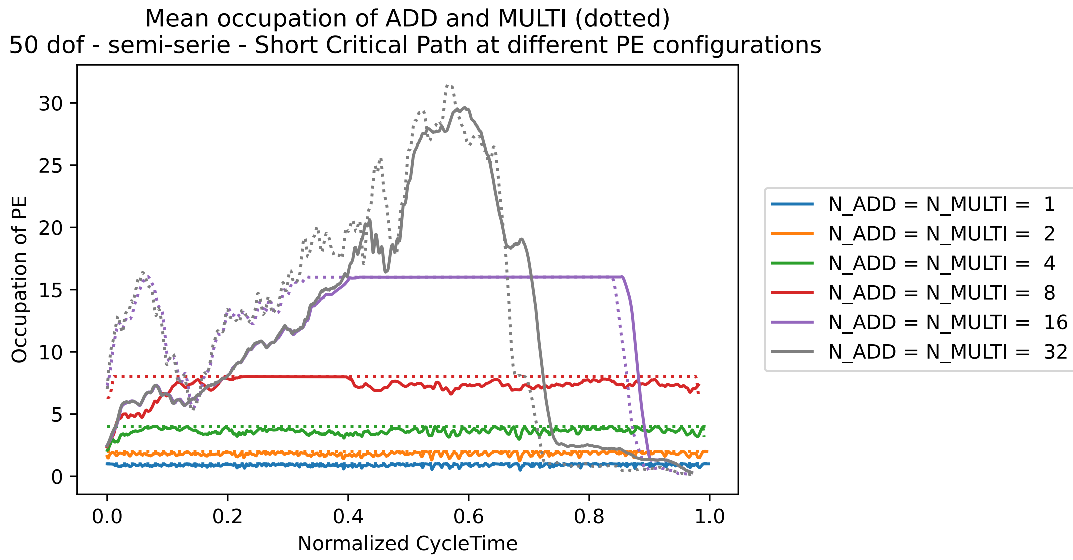


Figure 3.37: Occupation 50-dof - semi-serie - lowered critical path

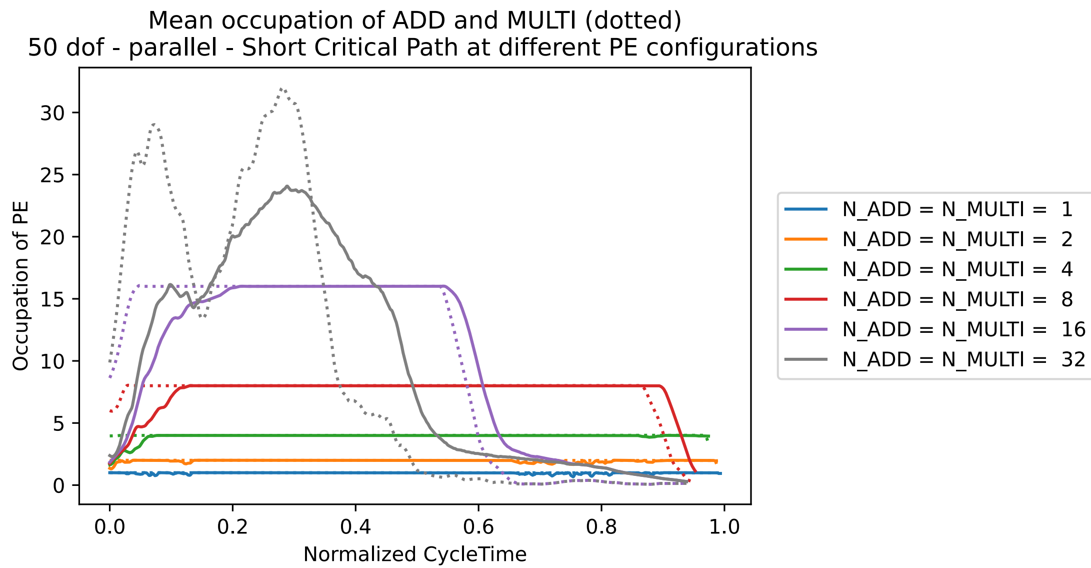


Figure 3.38: Occupation 50-dof - parallel - lowered critical path

Conclusion

During this thesis, we focused on the development and the characterization of parallel computation potential for multibody systems, specifically instruction-level parallelism, using as basis a dedicated architecture on a FPGA and the equations produced by the symbolic generator Robotran.

Synthesis

The equations of motion produced by Robotran by a symbolic generator allowed us to highlight the instruction-level parallelism present in these equations.

We have first presented in Chapter 1 an appropriate architecture on FGPA that allowed to accurately compute these equations in parallel. This architecture took advantage of the two operations we had in the equations to increase the number of PE in parallel. Indeed, we were limited to addition/subtraction and multiplication. We showed that pipelining was an effective way to have intrinsic instruction-level parallelism which would reduce the overall resource usage and increase the potential frequency.
The architecture design takes into consideration the constrains imposed by the DE10-Nano board as we were limited to two ports for the memory associated with the PE. From that constrain, we introduced a two clock design which limited the frequency of 50 MHz. We used for the connections between the different PE a dynamic interconnection, called Crossbar Switch, that enables us to choose for each data entry point of each PE Memory which result we want to write.
Finally, we showed that the DE10 could contain about 32 PE in parallel given the resources we have available on the board.

Based on this architecture, we developed in Chapter 2 methods to accurately extract the parallelism in the equations based on a list scheduling. We divided the equations into smaller parts to increase the parallelisation potential. Successively, with the aim to reduce the critical path and to increase the parallelisation rate even more, we presented three methods to expand the extraction of parallelism.

Regarding the placement algorithm we developed, we proposed a method to divide more equally the memory between the different PE. This reduced the memory size when increasing the number of PE. Effectively allowing for the FPGA design to have the same size of memory for the two types of PE.

Thanks to the development of both, an appropriate dedicated architecture for parallel computations and the writing of the instruction placement script, we were able to produce a study regarding the parallelisation potential for systems depending on their size and on their topological characteristics in Chapter 3.
Based on the length of the critical path and the number of operations in each multibody system, we were able to predict: the number of PE that would be needed to effectively have the minimal number of cycles or the number of cycles we will have given a amount of PE.
With regards to the size and the topological characteristics of the multibody system, we were able to present a projection for any other multibody system of the amount of PE they required for maximal reduction in computing time.
We studied the 23-ddl and 50-ddl system which equations had been modified to reduce the critical path, and we showed that we could divide the length of this path by at least two or three. Effectively increasing the parallelisation potential by the same magnitude.
Finally, we looked at the occupation rate of the PE throughout the cycletime and we highlighted that the occupation rate has similarities for systems of different sizes but sharing the same type of topology.

Limitations & Perspectives

In chapter 1, we had at our disposal a DE10, the implementation to overcome the difficulties caused by the 2-ports memory meant that the frequency was drastically limited. This issue is on top of a theoretical maximal frequency already low, of 120 MHz. While for modern CPU, eventhough they don't make an operation every cycle, have a clock of around 2.5 GHz. Combined with the possibility to have multi-core or multi-thread computations being produced through MPI or OpenMP, this leaves our dedicated architecture in a competitive disadvantage.
Fortunately, we are not limited to implement an architecture on a DE10-Nano. More recent FPGA present solutions to us. We have the possibility to take a FPGA from the Stratix 10 Family. Such a FPGA could provide a 4-port RAM Memory with a clock-speed between 400 and 600 MHz depending on the speed grade [33]. On top of that the Altera IP we are using for the ADD PE and MULTI PE would also know a proportional improvement in speed, therefore not becoming a limiting factor.
This would effectively enable us to reach much desired performances which are

needed in order to prove the interest and worth-while investment for such an architecture.

In the chapter 2, we showed that we could reduce the need for a buffer by preferring to select a PE for an operation based on the elements already present in the memory of each PE.
With the reduction of the buffer activity to a minimum low, we notice the possibility of removing the buffer as a whole. Although, this would necessitate the implementation of a new version of the placement algorithm, we could see a simplification of the architecture. Indeed, we would remove the presence of the associate memories and the number of inputs in the interconnection would be cut by half as the inputs linked to the buffer are no longer needed.

In Chapter 3, we applied the reduction of the critical path to the 23-dof and 50-dof systems. We applied manually the method named the tree swapping. We could write additional algorithms to implement the three methods to further reduce the length of the critical path.
In the end, for the 23-dof railway-bogie : from the 1812 equations grouped in 18 vectors, which results in 5469 operations for a critical path ALAP/ALAPCycle was 69/737. This critical path length was reduced to 47/471 using the tree swapping method.

Bibliography

[1] Nicolas Docquier, Sébastien Timmermans, and Paul Fisette. “Haptic Devices Based on Real-Time Dynamic Models of Multibody Systems”. In: *Sensors* 21.14 (2021), p. 4794.

[2] Paweł Malczyk et al. “Index-3 divide-and-conquer algorithm for efficient multibody system dynamics simulations: theory and parallel implementation”. In: *Nonlinear Dynamics* 95.1 (2019), pp. 727–747.

[3] Tony Postiau et al. “Génération et parallélisation des équations du mouvement de systemes multicorps par l’approche symbolique”. In: *Doctorat Université Catholique Louvain* (2004).

[4] Peter Eberhard and Werner Schiehlen. “Computational Dynamics of Multibody Systems: History, Formalisms, and Applications”. In: *Journal of Computational and Nonlinear Dynamics* 1.1 (May 2005), pp. 3–12. ISSN: 1555-1415. DOI: 10.1115/1.1961875. eprint: https://asmedigitalcollection.asme.org/computationalnonlinear/article-pdf/1/1/3/6686186/3_1.pdf. URL: https://doi.org/10.1115/1.1961875.

[5] Jean-Claude Samin and Paul Fisette. *Symbolic modeling of multibody systems*. Vol. 112. Springer Science & Business Media, 2003.

[6] Werner Schiehlen. “Research trends in multibody system dynamics”. In: *Multibody System Dynamics* 18.1 (2007), pp. 3–13.

[7] Werner Schiehlen. “Computational dynamics: theory and applications of multibody systems”. In: *European Journal of Mechanics - A/Solids* 25.4 (2006). 6th EUROMECH Solid Mechanics Conference, pp. 566–594. ISSN: 0997-7538. DOI: https://doi.org/10.1016/j.euromechsol.2006.03.004. URL: https://www.sciencedirect.com/science/article/pii/S0997753806000477.

[8] Jean-Claude Samin et al. “Multiphysics modeling and optimization of mechatronic multibody systems”. In: *Multibody System Dynamics* 18.3 (2007), pp. 345–373.

[9] Dan Negrut et al. "Parallel computing in multibody system dynamics: why, when, and how". In: *Journal of Computational and Nonlinear Dynamics* 9.4 (2014).

[10] Dan Negrut et al. "Leveraging parallel computing in multibody dynamics". In: *Multibody System Dynamics* 27.1 (2012), pp. 95–117.

[11] Gregorio Bernabé et al. "Exploiting Hybrid Parallelism in the Kinematic Analysis of Multibody Systems Based on Group Equations". In: *Procedia Computer Science* 108 (2017). International Conference on Computational Science, ICCS 2017, 12-14 June 2017, Zurich, Switzerland, pp. 576–585. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2017.05.041. URL: https://www.sciencedirect.com/science/article/pii/S1877050917305586.

[12] A. Celdran, M. Saura, and D. Dopico. "Computational structural analysis of spatial multibody systems based on mobility criteria". In: *Mechanism and Machine Theory* 176 (2022), p. 104985. ISSN: 0094-114X. DOI: https://doi.org/10.1016/j.mechmachtheory.2022.104985. URL: https://www.sciencedirect.com/science/article/pii/S0094114X22002361.

[13] Antonio J Rodríguez et al. "Hardware acceleration of multibody simulations for real-time embedded applications". In: *Multibody System Dynamics* 51.4 (2021), pp. 455–473.

[14] Roland Pastorino et al. "Hard real-time multibody simulations using ARM-based embedded systems". In: *Multibody System Dynamics* 37.1 (2016), pp. 127–143.

[15] Antonio Rodríguez et al. "Virtual sensing on automotive embedded heterogeneous platforms". In: Oct. 2017.

[16] Ladislav Mráz and Michael Valášek. "Solution of three key problems for massive parallelization of multibody dynamics". In: *Multibody System Dynamics* 29.1 (2013), pp. 21–39.

[17] Michael Valasek, Zbynek Sika, and Ondrej Vaculin. "Multibody formalism for real-time application using natural coordinates and modified state space". In: *Multibody System Dynamics* 17.2 (2007), pp. 209–227.

[18] Rudranarayan M Mukherjee and Kurt S Anderson. "Orthogonal complement based divide-and-conquer algorithm for constrained multibody systems". In: *Nonlinear Dynamics* 48.1 (2007), pp. 199–215.

[19] Marcin Pękal and Janusz Frączek. "Comparison of natural complement formulations for multibody dynamics". In: *Journal of Theoretical and Applied Mechanics* 54 (2016).

[20] Kristopher Wehage and Bahram Ravani. “A computational method for formulation and solution of dynamical equations for complex mechanisms and multibody systems”. In: *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*. Vol. 58172. American Society of Mechanical Engineers. 2017, V05AT08A031.

[21] Server Kasap and Khaled Benkrid. “Parallel Processor Design and Implementation for Molecular Dynamics Simulations on a FPGA-Based Supercomputer.” In: *J. Comput.* 7.6 (2012), pp. 1312–1328.

[22] Biagio Peccerillo et al. “A survey on hardware accelerators: Taxonomy, trends, challenges, and perspectives”. In: *Journal of Systems Architecture* 129 (2022), p. 102561. ISSN: 1383-7621. DOI: https://doi.org/10.1016/j.sysarc.2022.102561. URL: https://www.sciencedirect.com/science/article/pii/S1383762122001138.

[23] Subhi R. M. Zeebaree et al. “Design and Simulation of High-Speed Parallel/Sequential Simplified DES Code Breaking Based on FPGA”. In: *2019 International Conference on Advanced Science and Engineering (ICOASE)*. 2019, pp. 76–81. DOI: 10.1109/ICOASE.2019.8723792.

[24] Ramprasad Raghavan and Darshika G. Perera. “A fast and scalable FPGA-based parallel processing architecture for K-means clustering for big data analysis”. In: *2017 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)*. 2017, pp. 1–8. DOI: 10.1109/PACRIM.2017.8121905.

[25] Fernanda Kastensmidt and Paolo Rech. “FPGAs and parallel architectures for aerospace applications”. In: *Soft Errors and Fault-Tolerant Design*. Springer, 2016.

[26] Ahmed Ghazi Blaiech et al. “A Survey and Taxonomy of FPGA-based Deep Learning Accelerators”. In: *Journal of Systems Architecture* 98 (2019), pp. 331–345. ISSN: 1383-7621. DOI: https://doi.org/10.1016/j.sysarc.2019.01.007. URL: https://www.sciencedirect.com/science/article/pii/S1383762118304156.

[27] Matheus F Torquato and Marcelo AC Fernandes. “High-performance parallel implementation of genetic algorithm on fpga”. In: *Circuits, Systems, and Signal Processing* 38.9 (2019), pp. 4014–4039.

[28] Alexandre L. X. Da Costa et al. “Parallel Implementation of Particle Swarm Optimization on FPGA”. In: *IEEE Transactions on Circuits and Systems II: Express Briefs* 66.11 (2019), pp. 1875–1879. DOI: 10.1109/TCSII.2019.2895343.

[29] Intel Altera. *Cyclone V Device Overview*. [Online; accessed 30-July-2022]. URL: https://www.intel.com/content/www/us/en/docs/programmable/683694/current/cyclone-v-device-overview.html.

[30] Jean-Didier Legat. "A 400Kgates 8Mbytes SRAM multi-FPGA PCI system International Workshop on Logic and Architecture Synthesis (IWLAS'97), Grenoble, 16-18/12/1997, 1997, 5 pp." In: *Int. Workshop on Logic an Architecture Synthesis*. 1997.

[31] David Jean-Pierre and Legat Jean-Didier. "Implementation of very large dataflow graphs on a reconfigurable architecture for robotic application". In: *Parallel and Distributed Processing Symposium, International*. Vol. 4. IEEE Computer Society. 2001, 30143a–30143a.

[32] Intel Altera. *DE10 Specifications*. [Online; accessed 30-July-2022]. URL: https://www.intel.com/content/www/us/en/developer/topic-technology/edge-5g/hardware/fpga-de10-nano.html.

[33] Intel Altera. *Intel Stratix 10 Device Datasheet*. [Online; accessed 30-July-2022]. URL: https://www.intel.com/content/www/us/en/docs/programmable/683181/current/device-datasheet.html.

[34] Intel Altera. *Cyclone V Device Handbook: Volume 1: Device Interfaces and Integration*. [Online; accessed 30-July-2022]. URL: https://www.intel.com/content/www/us/en/docs/programmable/683375/current/logic-array-blocks-and-adaptive-logic-24877.html.

[35] Intel Altera. *Cyclone V Device Datasheet*. [Online; accessed 30-July-2022]. URL: https://www.intel.com/content/www/us/en/docs/programmable/683801/current/cyclone-v-device-datasheet.html.

[36] Dan C. Marinescu. "Chapter 5 - Cloud Access and Cloud Interconnection Networks". In: *Cloud Computing (Second Edition)*. Ed. by Dan C. Marinescu. Second Edition. Morgan Kaufmann, 2018, pp. 153–194. ISBN: 978-0-12-812810-7. DOI: https://doi.org/10.1016/B978-0-12-812810-7.00007-8. URL: https://www.sciencedirect.com/science/article/pii/B9780128128107000078.

[37] Wikipedia contributors. *Pipeline (architecture des processeurs)*. [Online; accessed 30-July-2022]. URL: https://fr.wikipedia.org/wiki/Pipeline_(architecture_des_processeurs).

[38] Paul Fisette and JM Peterkenne. "Contribution to parallel and vector computation in multibody dynamics". In: *Parallel Computing* 24.5-6 (1998), pp. 717–728.

[39] Paul Fisette. "Génération symbolique des équations du mouvement de systemes multicorps et application dans le domaine ferroviaire". PhD thesis. UCL-Université Catholique de Louvain, 1994.

[40] Robotran. *Robotran Basic*. [Online; accessed 28-July-2022]. URL: `https://robotran-doc.git-page.immc.ucl.ac.be/RobotranBasic/Robotran_basics.pdf`.

Appendix A

FPGA

A.1 ADD block

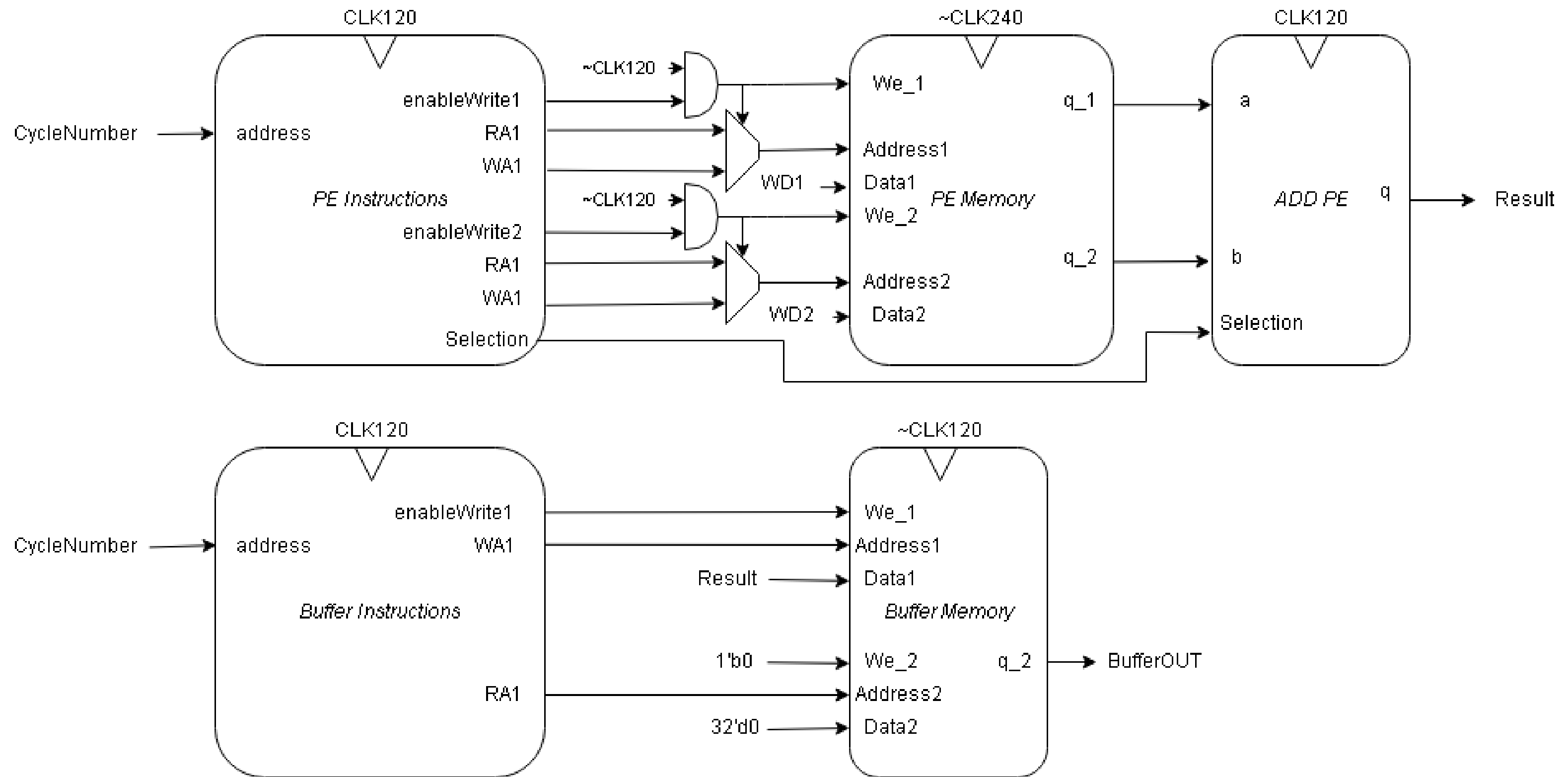


Figure A.1: Detailed ADD Block Architecture

A.2 MULTI block

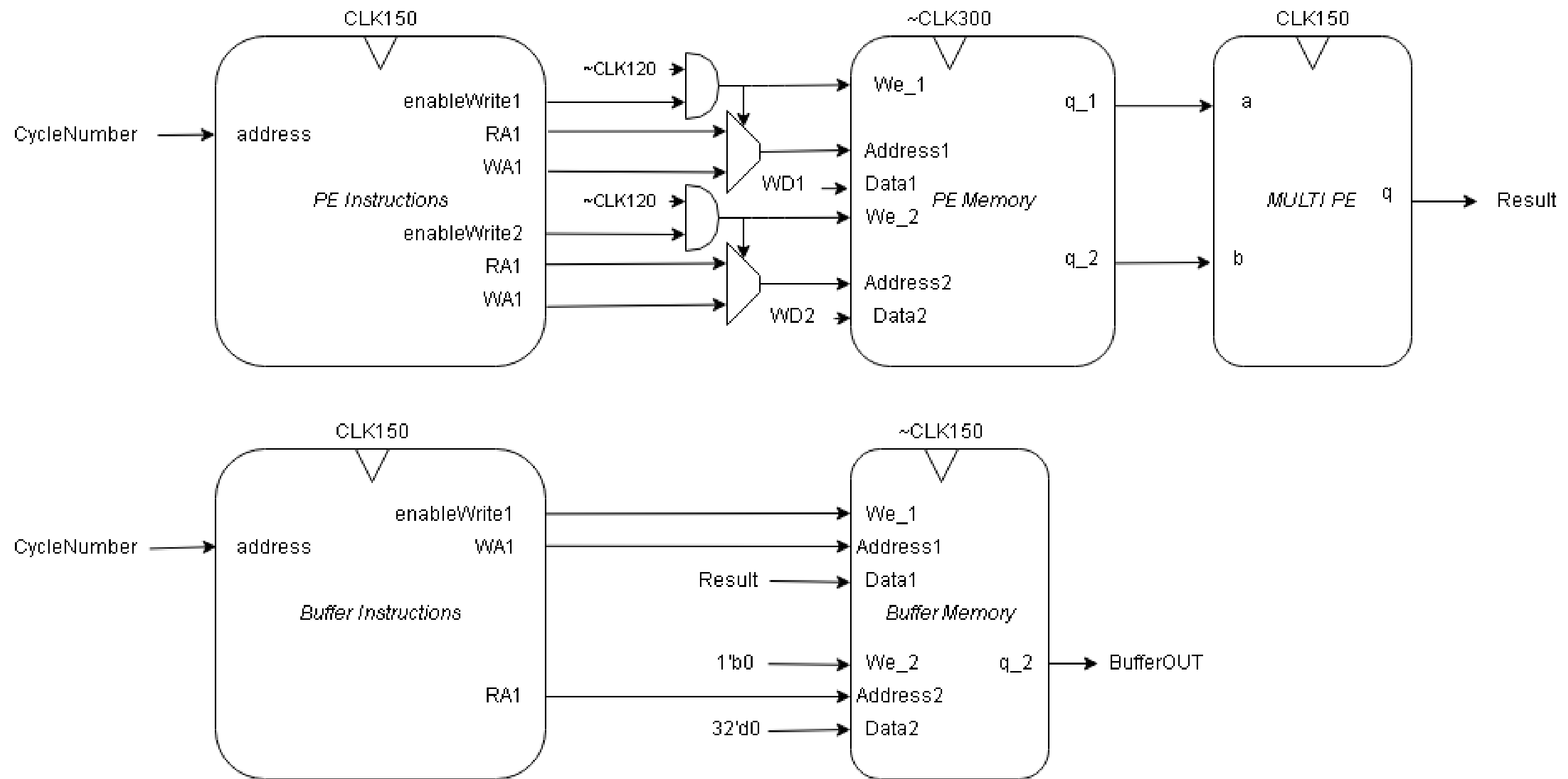


Figure A.2: Detailed MULTI Block Architecture

A.3 SignalTap

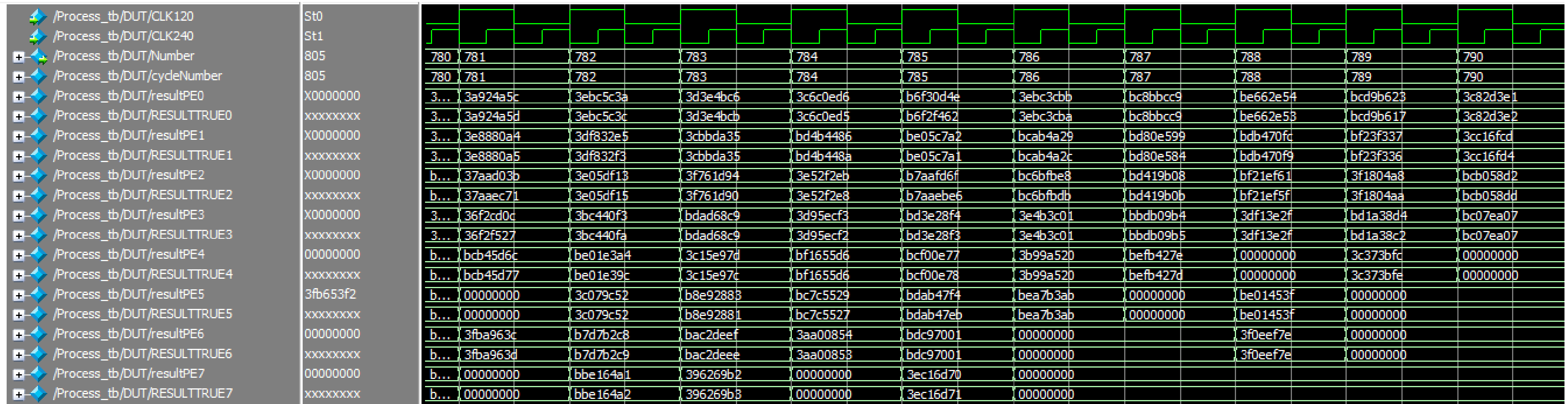


Figure A.3: SignalTap Screenshot :expected vs results

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**