Stefano Galvan

Perception-motivated parallel algorithms for haptics

Ph.D. Thesis

June 29, 2010

Università degli Studi di Verona
Dipartimento di Informatica

Advisor:
prof. Paolo Fiorini

Series N°: **TD-03-10**

Università di Verona
Dipartimento di Informatica
Strada le Grazie 15, 37134 Verona
Italy

Summary

In the last years the use of haptic feedback has been used in several applications, from mobile phones to rehabilitation, from video games to robotic aided surgery. The haptic devices, that are the interfaces that create the stimulation and reproduce the physical interaction with virtual or remote environments, have been studied, analyzed and developed in many ways. Every innovation in the mechanics, electronics and technical design of the device it is valuable, however it is important to maintain the focus of the haptic interaction on the human being, who is the only user of force feedback. In this thesis we worked on two main topics that are relevant to this aim: a perception based force signal manipulation and the use of modern multicore architectures for the implementation of the haptic controller.

With the help of a specific experimental setup and using a 6 dof haptic device we designed a psychophysical experiment aimed at identifying of the force/torque differential thresholds applied to the hand-arm system. On the basis of the results obtained we determined a set of task dependent scaling functions, one for each degree of freedom of the three-dimensional space, that can be used to enhance the human abilities in discriminating different stimuli.

The perception based manipulation of the force feedback requires a fast, stable and configurable controller of the haptic interface. Thus a solution is to use new available multicore architectures for the implementation of the controller, but many consolidated algorithms have to be ported to these parallel systems. Focusing on specific problem, i.e. the matrix pseudoinversion, that is part of the robotics dynamic and kinematic computation, we showed that it is possible to migrate code that was already implemented in hardware, and in particular old algorithms that were inherently parallel and thus not competitive on sequential processors. The main question that still lies open is how much effort is required in order to write these algorithms, usually described in VLSI or schematics, in a modern programming language. We show that a careful task decomposition and design permit a mapping of the code on the available cores. In addition, the use of data parallelism on SIMD machines can give good performance when simple vector instructions such as add and shift operations are used. Since these instructions are present also in hardware implementations the migration can be easily performed. We tested our approach on a Sony PlayStation 3 game console equipped with IBM Cell Broadband Engine processor.

Acknowledgments

"My journey to the haptic side of the force is almost complete"

The long and winding road of my PhD is at the end. I am completely satisfied with the results obtained, in particular because it has been hard to cross this finishing line. As for many other aspects of my life it took time. There has been a moment in which I realized that I had to slow down, to refocus, probably to wait a little bit. And to change things. However now it is done, and I am serene.

During this journey I crossed the paths of wonderful people, I have walked side by side with many persons, and luckily I am still travelling with some of them. I would like to thank them all.

In particular I wish to thank the internal commission and the external reviewers for their encouragement, their support and their valuable comments and opinion. And of course for the questions they asked during my defence ;). A warm thank goes to my advisor, Paolo, that never stopped telling me that "we can work it out". It is all thanks to him if some years ago I started considering robotics not only a cool topic for anime and sci-fi plots, but also an amazing field of studies, an incredible research opportunity, and a challenging bet. In the last months I learned from him the art of pushing the finishing line a little further, while adding some obstacles, to eventually increase the reward and, as a consequence, the gratification. Despite distances and conflicting schedules he gave me suggestions about all I've got to do for the PhD.

I spent good time in the Altair lab, with great people. We stayed until late while searchin' solutions such as fixing a hole in the teleoperation system or discovering the magic of electrical interference, and, of course, fightin' on multiplatform, free first-person simulations. Two of us started the PhD adventure in different moments, and I admit that I feel lucky to have shared it with the deadliest scallawag that ever swung a sword. Ezerb, look behind you, a three-headed monkey! And of course I faced all the good and the bad moments of the PhD with a little help from my friends. Actually with a big help from them. Boys and girls... you are really a treasure. Hope we will continue having good time together. A special thought is for Damiano and Sonia.

A big thank goes to my family: my mom, my great sister, my brother-in-law and my nephew. They never stopped trusting in me, supporting me, counseling

me and mainly bearing me, eight days a week. Wherever I may go you are always my reference point.

The biggest thank, and I have to shout it loud, goes to a very special girl: my beloved Francesca. She was not only the perfect proof-reader, but she was able to anticipate every little thing that I needed before I actually ask it. She really stands by me. And she always makes me laugh :).

That's all folks. Ob-La-Di, Ob-La-Da... life goes on, bra!

P.S.
I've got a feeling: I suppose that a lot of people expected some Star Wars citations. Actually I used the word *force* 235 times in this thesis: I think that is enough. Anyway it may seem that there is something strange in this Section. Yes it is. During my working, and writing and luckily relaxing hours, in these months I had a fantastic soundtrack: the songs of the Beatles. My passion for the Fab Four dates back to a nice evening with Francesca. However I decided to pay homage to them by hiding 21 song titles in these acknowledgments. The exercise to find them all is left to the willing reader. *Hint:* use wikipedia if you really need help!

Contents

1

Introduction

In this thesis, we show how to obtain better performance from haptic interfaces in teleoperation. In the last years much research was carried out in order to design, build and develop effective haptic devices for teleoperation systems. We also address this problem but from a task oriented point of view. In fact we first show how to exploit human force discrimination capabilities by a proper force signal manipulation based on pshychophysical evidences, to achieve personalized performance. Then, we propose to employ modern multicore parallel architectures for the implementation of the haptic controller to permit more complex operations while fulfilling teleoperation constraints. We show that the migration of code to parallel systems can be smooth and result in good performance when the original algorithm is inherently parallel. When a specific solution has been already implemented in hardware in the past, it is worth looking to the design of the code because it surely has a suitable programming model.

Teleoperation is defined as the remote manipulation of one or more devices by a human operator. In this context we assume that the controlled devices are robots. To ensure a better understanding of the remote environment and precise control of the robot, some feedback to the operator is needed. Visual and acoustic clues are widely used but it has been shown that operator performance is improved by providing her/him force feedback [27]. Force feedback improves the ability to perform complex tasks, and reduces the exertion of large forces that usually lead to operator fatigue and remote environment damage. This is an important issue in tasks such as robotic aided surgery when the interaction with the environment implies contact with the patient. When force feedback is present, the operator is said to be kinesthetically coupled to the slave and the teleoperated system is said to have bilateral control.

When we talk of bilateral teleoperation the distinctive part, with respect to automation, is the utilization of one or more haptic devices as masters, that is the instrument used by the operator in order to "perceive" the remote or simulated environment. A haptic tool is a more or less complex pointing interface equipped with actuators: they are responsible to provide the sensation of contact, weight and force to the human being.

1.1 Haptic devices

Haptic interfaces were developed to simulate the action of touching remote objects or to interact with virtual objects endowed with suitable dynamics (i.e. hardness and elasticity) to provide realistic sensations. The force feedback provided by a haptic device is usually defined as the sensation of weight or resistance felt by a human operator [19].
This technology has been used not only for teleoperation but also in other fields, such as virtual reality systems and computer simulation, with applications to diverse areas such as surgery, video games, exploration and molecule design.

The way humans sense forces is not yet completely understood. Different studies about human perception are addressing this point and special devices are needed to test, track and analyze different strategies and force profiles. Teleoperation needs fast response and stable control to be effective, whereas adjusting control parameters and generating forces at high speed is necessary for both perception studies and complete tele-presence experience. Addressing this points means to create a fast and robust haptic device easily configurable by the user. Since the goal is to combine hardware capabilities with software flexibility in order to exploit perception abilities, the needs for a new generation of human-centric configurable devices and controllers arise.

There are several good haptic devices available on the market. Phantom [100], Delta and Omega [35], Freedom6s [113] are widespread and often used for research purposes. Phantom is maybe the most famous one as it was the first desktop haptic device available at low cost. Usually these products come with an easy to use programming API and an ongoing effort is spent to make them compatible with a number of different operating systems. However the programmer and the user do not have the freedom to change the basic behavior of the device. There are also highly specialized haptic devices used in crucial task such as robotic assisted surgery, nuclear facility management, dangerous environment manipulation. In spite of the fact that they are extremely complex and optimized, they share the same basic principles of the "domestic" devices and they follow the very same workflow during the design, engineering and production development.

At the end of the process there is the development of drivers to use the device with the more common operating systems. Even for the highly specialized devices such as the Intuitive Surgical robot Da Vinci [50], the core of the application is driven by an operating system that needs a software handling of the device and often, when not always, a sets of programming API and software tools to handle kinematics and dynamics of the device, and to modify the parameters of the bilateral teleoperation loop. This organization is natural because it reflects the state of the practice of computer science that pushed towards powerful general purpose processors, delegating many tasks to software. In our opinion however, this seems somewhat limited since it does not reflect current changes in hardware architectures and specific software solutions.

In [45] Hayward and MacLean present an overview of the state of the art about haptic interfaces. They start with a description of the most common types of haptic displays, then they focus on specific components such as sensors, actuators and kinematic structures and they highlight rules for force feedback device characterization. Moreover, they explore control, stability, end modelling issues expressing considerations about hardware and software requirements. In particular they stress the importance of real-time software and loop synchronization pointing the attention, for instance, to the development of interesting middleware for haptics [85] or robotics and teleoperation in general. Although this is for sure a possible solution, and we worked in this area with the development of a real-time distributed framework for robotics [38], we think that its main application is about the interconnection of distributed devices in a teleoperation task. In fact, haptic displays require a more specific approach in terms of hardware/software integration. In [45] they analyze the topic in a classical engineering way, and in [67] the same authors continue their analysis form a different point of view, describing some basic concept of haptic interaction design together with several interesting applications based on this technology. In their comprehensive revision, the authors show the technology behind the physical devices but they also point out the importance of the design of an usable and effective interaction between the utility and thus the meaning of the haptic feedback and the hardware that provide it. They take into account also the psychophysical aspects of the haptic interaction, mainly with respect to the design phase of the device development. Although this approach is sound, in our opinion it is also crucial to add another step at the end of the developing process, that is the development of a suitable implementation of a perception driven, or perception-aware, controller.

There are several studies about human perception and robotics and haptics, but they are usually confined to specialized niches such as trauma rehabilitation, impairing disease analysis and perception evaluation [69]. Perception studies are also used in bio-mimetic robotics to better understand the physiology of the human interaction with the environment and to apply the results to robot design. This step is normally not considered as a necessary prerequisite for the design of a haptic device. Moreover it is very unusual to provide hardware/software support for force feedback adjustments based on psychometric functions, human perceptual skills exploitation and task oriented optimizations.

1.2 Objectives

We think that a haptic device should take into consideration the results of studies about human perception, and thus be more compatible with humans or ecological or human-centric. On the other hand, recent trends regarding the use of multicore processors, with dedicated hardware, could break the boundaries between the determinism, speed and reliability of ASICs and flexibility and easiness to customize of software implementation. With “customization” we mean the ability to vary the behaviour of the force interaction and thus the tele-experience of the operator using perceptual illusions while maintaining the fast and reliable real-time loops required for teleoperation stability.

With these issues in mind we propose a tighter integration between psychophysics and robotics and we suggest that this multi-modal approach should be supported by suitable hardware control architecture and parallel algorithmic programming, otherwise the theoretical advantages could be overwhelmed by the lack of integration between the device and its software.

This twofold approach to haptic interaction is getting more evident in the very last researches and this is the vision in which this work of thesis takes place. We demonstrate our idea using a configurable, high performance joystick with force feedback to carry out psychophysical experiments in order to gain new insight about human perception of forces. From this early results we propose enhancement to force reflection strategies. As a consequence, we propose the use of multicore architecture as fast, reliable, configurable controller for haptic applications. This requires the migration of part of the controlling software to parallel systems. In order to make this adaptation easy and to permit the achievement of good performance we analyze how to convert, when available, existing hardware implementation to parallel software. We show the feasibility of the approach by choosing a particularly crucial operation used in robotics, and to port the algorithm to a MIMD/SIMD architecture. We present an actual implementation on the IBM Cell Broadband Engine with performance results.

This thesis is organized as follows. In Chapter 2 we report on experiments about haptic perception aimed at measuring the force/torque differential thresholds applied to the hand-arm system. Then we present a force scaling for bilateral teleoperated system. We focus our attention on human perception capabilities and, on the base of previous psychophysical experiments, we exploit the human ability to perceive forces and torques differently along different directions. In Chapter 3 we explain what are the requirements that permits to obtain effective improvements in a real teleoperation systems making the haptic device reliably customizable and using the results of perception experiments. We present an overview of available multicore architectures and we survey programming models able to exploit these architectures. In particular, we highlight that for these systems we used to look for solution developed for ASICs and VLSI to develop good parallel algorithms. In Chapter 4 we identify the requirements for such an approach through the investigation of the multicore implementation of the solution to the inverse problem for ill-conditioned linear systems. By addressing this issue we solve a common problem in haptics: the Jacobian pseudoinversion. This is the topic of Chapter 5, where we show that the migration of code from hardware to parallel architectures is possible. We develop a design based on a good programming model and an implementation that mimics the solutions adopted in hardware achieving good performance without dealing with complex optimizations. We discuss the implementation for the IBM Cell Broadband Engine. The last Chapter highlights conclusions and future applications.

2

Perception Experiments

In this Chapter we describe our work on a perception-centric approach to force feedback manipulation. In order to make the remote experience realistic as well as to increase the abilities of the human operator during bilateral teleoperation, it is necessary to understand how forces are perceived and decoded by the human.

However since force feedback manipulation, and thus any enhancing strategy, depends on a specific teleoperation task, we develop a general methodology focusing on Minimally Invasive Robotic Surgery (MIRS).

In this application area there are highly variable forces. For example, in fine manipulation very low forces are involved. Furthermore, in minimally invasive interventions these forces are difficult to perceive. By means of robotic instruments we can overcome this limitation and provide a perception enhancement that would lead to a better surgical performance. Therefore, we investigate haptic perception of forces in order to quantify how well an user detects changes in force magnitude.The results of this work are presented in [117].

2.1 Prior work

The human haptic perception couples the user to any environment, either local, remote or virtual. Thus, a basic understanding of the biomechanical, sensorimotor, and cognitive abilities of haptic perception is a critical factor for the proper hardware and software specification of haptic devices. Moreover, the understanding of the limitations of human perception would also help in designing the interfaces and their associated applications [43].

Several techniques are especially relevant to the quantitative measurements of the human factors that affect the design of force-reflecting haptic interfaces. One of the most common measures is related to the *Just-Noticeable-Difference* (JND), which is the minimal difference in intensity between two stimuli (I vs. $I+\Delta I$) that leads to a change in the perceptual experience. The JND is an increasing function of the base input level, generally defined as a percentage value by

$$\text{JND}\% = \frac{(I + \Delta I) - I}{I} \times 100 \tag{2.1}$$

A number of works accounts for the human JND% in force perception. In experiments of hand and arm lifting of objects with masses between 1000 and 7000 g, [91] found a decreasing JND% from 16% to 11%. When users made judgments with their hands at rest, [16], they obtained a mean value of 13%. Further experiments were carried out with virtual objects, i.e. providing forces to the user by using different haptic devices: [52] reported a JND% of 15% for the human perception of forces applied to the hand-arm system. In [48], values from 6% to 50% during the human control of a pneumatically driven robotic arm by means of a haptic interface were obtained. In [121], while dealing with finger capabilities in terms of force perception, the authors report a JND% value of 16% for normal and tangential forces. The value for pinching between finger and thumb was found to be ranging between 5% and 10% of the reference force [84]. In [2] it was found that JND% was relatively constant over a range of different base force values between 2.5 and 10 N. In [84], the authors concluded that the force JND is essentially independent of reference force and displacement.

In two prime reviews [105, 107], these results are presented as a key in human force perception for the design of haptic devices. However, these reviews do not adequately stress the fact that the results are focused on finger capabilities. In addition, the experimental methodology often did not explicitly consider the elementary force perception. In fact, the perception of force vector angles was analyzed in the space as a whole and not with respect to the contribution of each Degree of Freedom (DoF) generating the target force [8]. The problem of enhancing the capability of the user to discriminate variations of the compliance of the remote environment is presented in [28], where a 4-channel control architecture is considered and the controllers are optimized to make the user more sensible to the variation of stiffness falling into a desired, tunable, range. It consider linear teleoperation systems and a linear (ideal spring) model of the remote environment.

Our aim is to identify whether force intensities and orientation are associated with different values of JND%. This finding could let us discover a force threshold felt by the human operator and distinguish the most sensitive directions for the arm. We could then identify one or more suitable scaling functions for force-feedback in haptic environments.

The purpose of the experiments described in this Section is to explore the differences in ability of a person to discriminate a wide range of force intensities applied along the axes of a reference frame positioned at the hand-grip. In particular, we investigate the capability of the arm to independently discriminate a force along the translational axis (x, y, and z) and the capability of the wrist to discriminate torques along each rotational axis (*roll*, *pitch*, and *yaw*).

The following Section illustrates the haptic device and its calibration. Also, it discusses the experimental phases of this work: the exploratory test and the experiment based on a psychophysical adaptive method.

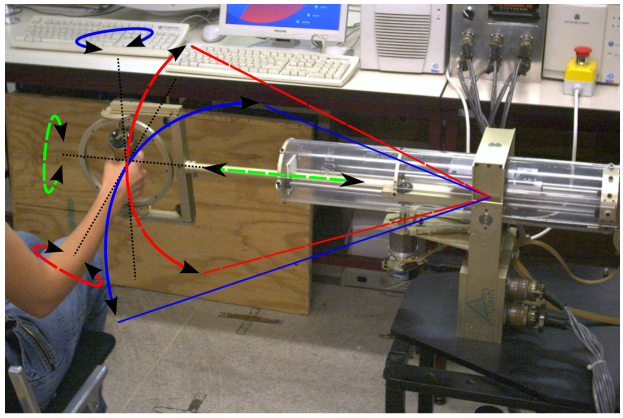


Fig. 2.1. The experimental setup with the NASA-JPL Force Reflecting Hand Controller haptic device. The arrows show the application of forces and torques to the hand of the subject. Arrows with the same color (line style) represent force and torque applied along the same axis of the subject reference frame placed at the hand-grip, under static conditions. Blue (continuous) arrows are respectively force and torque generated by the action of joints β_2 and β_4 on the Z-axis. Green (dashed) arrows are respectively force and torque generated by the action of joints R_3 and β_5 on the Y-axis; Red arrows are respectively force and torque generated by the action of joints β_1 and β_6 on the X-axis.

2.2 Experimental Setup

2.2.1 Haptic Device

In our experiments we used a Force Reflecting Hand Controller (FRHC) which meets our experimental requirements thanks to its particular structure. The FRHC was designed and developed at NASA's Jet Propulsion Laboratory [10]. It has been refurbished recently, in particular with the addition of a custom designed controller implemented mostly on a NI PCI-7831R FPGA board (National Instruments, Austin, TX), that drives the device motors [37].

This device has 6 Degrees-of-Freedom (DoFs) consisting of one translational and five rotational joints (see Fig. 2.1). It rotates and slides around a fixed support attached to the floor through two rotational (β_1, β_2) and one translational (R_3) joints. Its hand-grip has three intersecting axes $(\beta_4, \beta_5, \beta_6)$. With this structure, an operator can work with full dexterity in a cubic workspace of $30 \times 30 \times 30$ cm^3. Motion transmission is done by cables with pulleys of large curvature that reduce friction and increase the back-drivability. An idler mechanism translates on the direction opposite to R_3 to keep the manipulator always balanced, performing a mechanical gravity compensation. The power unit is placed on the floor to avoid unbalancing the structure. The hand-grip of the device is positioned so that it can be comfortably reached by the subject's dominant hand.

2.2.2 FPGA

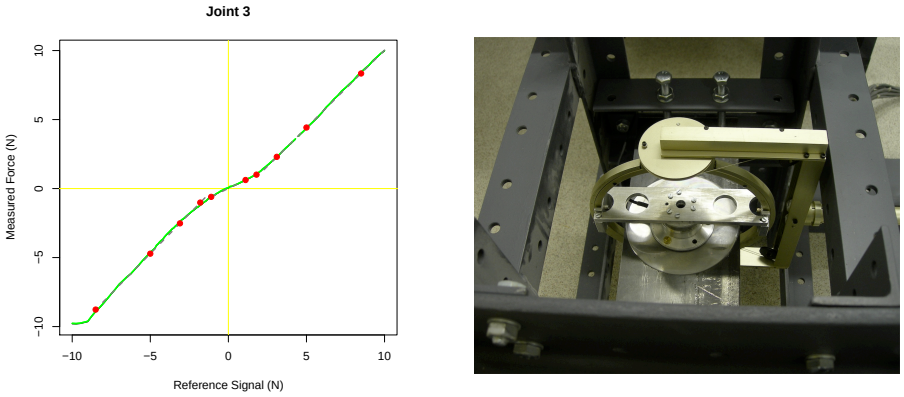
In [37] we presented an innovative hardware/software structure used to control the FRHC in a teleoperation task. To increase speed and reliability, parts of the kinematic calculation were embedded into the joystick controller. We used a one million gates National Instruments NI PCI7831R FPGA, based on a Xilinx Virtex II chip. The FPGA was initially used for handling sensors and actuators. We implemented quadrature decoders in order to obtain position information from incremental optical encoders, and we used analog outputs to give voltages to PWM generators for driving the joystick motors. In addition, we added a parallel communication interface to transfer data to/from a PC. We used a GNU/Linux PC with RTAI extension in order to give real-time capabilities to the overall system. Then we moved to the FPGA the algorithmic part that is usually handled in software. In particular we coded the forward kinematic of the FRHC and the force feedback computation. We designed the implementation in a modular way, to easy change between different kinds of control. In this way we obtained a fast and stable controller that, unlike industrial axis-board, includes both low and high level functionalities on the same hardware.

2.2.3 Subject reference frame

Since the kinematic structure of the FRHC permits a mapping between each DoF of the joystick and the reference frame placed at the hand-grip, in the following sections we refer to joints as the directions on which a stimulus is applied. When a subject uses the FRHC, her/his hand can be considered at the center of the reference frame. Results obtained using the hand reference frame can be related to other sensorial reference frames [95], and to several kinematic models of the hand-arm system [9].

The hand-grip gimbal ensures that the torques provided to the hand by the three upper DoFs ($\beta_4, \beta_5, \beta_6$, corresponding to extension/flexion, abduction/adduction and pronation/supination, respectively) are torques about the main axes of the hand reference frame. The same consideration holds for the lower joints with an assumption: while the prismatic joint R_3 moves in a linear way along the hand reference frame y axis, the lower joints β_1 and β_2 are rotational and their movement is not along the hand reference frame axes; in our setup this is not an issue since the design of the experiment asks the subject to maintain the position of the hand as firm as possible. Under this assumption (i.e. small movements) the forces exerted by the first two joints can be considered tangential to the rotational movement of the FRHC and thus aligned with the main axes.

The results of the experiment are then representative of every force stimuli presented at the hand of the subject and not related to the kinematic structure of the involved haptic device. When using a different joystick the kinematic and Jacobian transformations must be used to correctly express the forces exerted at the hand-grip in joint space.



(a) (b)

Fig. 2.2. (a) Prototypical data from the calibration phase: force measured by the F/T sensor over the expected force delivered by the motor in the calibration procedure of joint R_3. The 10 points refer to the reference forces identified for the experimental phase among the physical stimulus domain. (b) Calibration setup: the FRHC attached to the calibration cage. The hand-grip is replaced with an iron plate. The ATI F/T sensor connects the plate and the rigid cage.

2.2.4 Calibration

Before the experimental session, we calibrated each joint in order to associate the reference voltage given to the motor, and hence the *nominal* force/torque at the hand-grip, to the resulting *measured* force/torque. We wanted to obtain a function, in specific terms a curve, that fits the power provided by the motor with the stimuli at the hand, see Fig 2.2(a). The effective force is influenced by motors, links flexibility, friction, inertia, measurement errors and control errors; therefore a calibration is needed to overcome these disturbing effects and to accurately quantify the amount of force that will be provided during the experimental trials. We were especially concerned with the lower intensity stimuli since we wanted to identify the threshold with high precision and the variations involved were small.

We designed and built an iron "cage" large enough to contain the FRHC. Thanks to the structure and the thickness of the linkages we used, the cage was not deformable. We firmly attached the joystick inside the cage and replaced its hand-grip with an iron plate of the same weight in order to maintain the device right balanced. The plate was not deformable when loaded with forces in the rendering range of our haptic device. An ATI Mini 45 (ATI-Industrial Automation, Apex, NC) 6 DoFs force/torque sensor was mounted between the plate and a configurable rigid support of the cage, see Fig. 2.2(b). This sensor had higher resolution, precision and accuracy than humans. We were able to lock on up to five joints thus constraining the degrees of mobility of the FRHC.

We provided voltages to each motor and we collected the resulting forces at the hand-grip. We tried several voltages spread along the whole range of forces that the joystick could exert. For each value we used a high number of sensor

readings and we repeated the trial several times in different sequences, both linear and random. We considered the two directions of a motor independently thus obtaining two fitting curves. This was made in order to avoid asymmetries due to the cable driven actuation.

The joystick was static in the sense that it could not move when forces were applied. This is a correct assumption for the experimental setup since the considerations explained in Sec. 2.2.3 hold and the force signal is always instantaneous. The data collected were clean everywhere along the joystick force rendering range apart from in the values close to 0 N. This was due to the high values of friction and inertia with respect to the forces rendered. However, taking advantage of the results of the calibration phase, we did not consider for the experiments any force value close to 0 N, see Fig. 2.5. This strategy should ensure a precise and not corrupted threshold estimation. Calibration confirmed that the FRHC applied forces up to 9.9 N and torques up to 0.6 Nm.

2.2.5 Experimental Phases

The aim of the experimental design was to measure the capability of the human hand in terms of force perception. Well-known psychophysics methods were involved in order to measure the JND%. In Section 2.3 we used the Constant Stimuli procedure [39]: this very time-consuming procedure enables the accurate estimation of perceptual threshold, with only a minimal a-priori knowledge of stimuli dimensions. Only few subjects were involved in this first experiment: our aim was to identify plausible starting values for a more efficient adaptive psychophysics procedure [41] used in Section 2.4 with more subjects. The peculiarity of this procedure was to minimize the number of trials and consequently the duration of the session, while maintaining the same measurement properties. In order to identify reliable thresholds values, the first experiment was necessary due to the relevance that the initial starting value of the stimulus has on any adaptive procedure [65].

Here following, we describe the general set-up for the experiments. Experiment-specific details are presented along with the corresponding data.

2.2.6 Stimuli Presentation

For each joint, 10 reference forces or torques s_i were identified among the physical stimulus domain. The forces ranged from 0.4 to 9 N and the torques from 0.02 to 0.5 Nm. The selection is based on a logarithmic distribution of nominal set points to the hardware, with the aim of focusing especially on low force/torque because of denser samples at low values. In this way we can optimally use the capabilities of our hardware set up, and maintain a spread of data across the range of our devices consistent with the experiment goals. Forces were not accommodated with any ramp in order to let the participants perceive a unique well defined value.

Participants were instructed to firmly hold the hand-grip and to keep the hand movement as small as possible. The force and torque stimuli were applied to the subject's reference frame, lightly moving the hand in the direction of the applied force.

We used an n-Alternative Forced-Choice (nAFC) paradigm. Participants were given a choice of n alternatives, and they had to select the one containing the most intense stimulus. They knew that exactly one alternative contained the stronger stimulus $I + \Delta I$ (also called the comparison stimulus) and that the rest had the lower stimulus I (also called the standard or reference stimulus). The comparison stimuli were chosen according to the different psychophysics procedures discussed as follows.

During each trial, n stimuli were presented to the participants in sequence: $n-1$ times the reference stimulus and once the comparison stimulus. In each trial, the comparison stimulus was presented randomly in one of the n sequential positions, with an a-priori probability of $\alpha = 1/n$. There were n admissible responses: "1", "2", and so on. After stimuli presentation, participants were asked to judge which was the strongest one. Subject were instructed to respond "1" when they felt that the first stimulus was the strongest one, "2" for the second, and so on. Due to the nAFC-inherent guessing, we were expecting that participants gave n^{-1} correct responses; thus, the threshold was the force value with a number of correct responses equal to $(n\ +\ 1)/(2n)$.

Each stimulus was applied for 1,200 ms; the interval between different stimuli was 300 ms. No feedback was given during the experiment. The participants had to take a break among runs about every 7 minutes and whenever needed.

2.2.7 Statistical Analysis

The JND%s were obtained by calculating the respective percentage of "stronger than" responses and then fitting the psychometric curves to each data-set, obtaining the parameters of the assumed psychophysics function. That is, the cumulative responses to stimuli presentation were used to estimate the function which describes the probability that participants judged the stimulus as exceeding the standard stimulus. The resulting S-shaped curve formed the so called "psychometric function". It was conveniently defined by a logistic function and, given the parameter set $\Theta = \{\alpha, m, k\}$, its basic form was given by

$$\Phi(\Theta|x) = \alpha + (1 - \alpha)\frac{1}{1 + e^{-k(x-m)}} \tag{2.2}$$

where α is the a-priori probability, m is the threshold that is the point where the second derivative (curvature) is zero, and k is the slope of the psychometric function.

A statistical analysis was conducted separately for each subject and for aggregate data. Each analysis of variance (ANOVA) included a factor for individual subjects so that differences between subjects were not counted as a random variation; this made each analysis more sensitive to the parameter of the stimulus which is varied.

2.2.8 Participants

A total of 11 males and 6 females were examined (mean age of 26 years, age range from 19 to 36 years), almost all of them with no previous knowledge of the experiments. Five participants applied for the constant stimuli procedure, discussed

in Sec. 2.3, thirteen for the adaptive one, discussed in Sec. 2.4. One was tested in both experimental procedures.

Participants were recruited among the staff of the Altair laboratory of the University of Verona (Italy) by word of mouth and did not receive any compensation for their participation. All the participants were right-handed, had a normal sense of touch and used their dominant hand to perform the task.

2.3 Exploratory test

2.3.1 Design and stimuli

In the exploratory test we used the Constant Stimuli psychophysics procedure in a 3AFC paradigm [39]. This very time-consuming procedure had a fundamental advantage: it enabled accurate estimation of the psychometric function, its parameters and, consequently, the JND%, with only a minimal a-priori knowledge of the stimuli dimensions.

Based on a pilot test, 5 force increments were conveniently defined at a linear distance in the range +4% and +26%, where the lowest was a stimulus that could almost never be detected as the strongest, while the highest was a stimulus that was almost always detected as the strongest. For each of the 6 joints, the experimental block of trials consisted of 500 trials (10 stimuli $\times$ 5 force increments $\times$ 10 repetitions), randomly presented. Experimental sessions typically lasted more than 75 minutes. The presentation order of the experimental block was also randomized.

2.3.2 Results

In order to identify the psychometric function and its parameters, with the method described by [122], we fitted the logistic function defined in Eq. (2.2) using a ML-estimator and the minimization function *optim* provided within the R environment [112]. In order to assess the reliability of the estimates, we used two goodness-of-fit tests suitable for analyzing binomial data: the Deviance test and the Pearson χ^2 test. Estimates that did not fit the data with $p < 0.05$ were rejected from the analysis.

Thresholds and slopes were estimated with the same methodology for each joint, stimulus and subject. Results are reported in Table 2.1. These estimated thresholds and slopes were necessary to compute reliable starting parameters for the adaptive procedure used in the following experiment.

For each joint, 10 (stimulus references) $\times$ 5 (subjects) ANOVAs were performed, having the estimated thresholds and slopes as dependent variables and the subjects as the error term. Only for the joint β_2 the JND% did not significantly change among the stimuli ($F_{9,22} = 1.42$, p-value $= 0.24$), like drawing a linear function; for the other joints significant differences were observed (β_1: $F_{9,23} = 7.03$, R_3: $F_{9,28} = 3.20$, β_4: $F_{9,27} = 3.40$, β_5: $F_{9,28} = 8.76$, β_6: $F_{9,30} = 4.16$, p-value < 0.001), and the curve describing the JND% could be expressed by a bell curve.

Joints		Reference Stimuli									
		s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}
β_1	Reference [N]	-5.99	-3.28	-1.80	-0.87	-0.54	0.50	0.83	1.79	3.26	5.98
	JND%	15.38	17.84	27.62	33.60	42.85	74.56	49.48	24.40	20.79	15.64
	Slope	1.94	4.30	5.92	5.46	12.96	5.56	5.99	6.77	4.02	4.33
β_2	Ref. [N]	-7.21	-3.87	-2.06	-0.80	-0.51	0.41	1.07	2.29	4.07	7.35
	JND%	13.49	18.19	31.38	46.33	26.95	34.57	26.16	29.09	15.52	12.56
	Slope	1.92	3.12	5.23	7.00	14.32	9.55	7.41	5.56	5.17	1.98
R_3	Ref. [N]	-8.77	-4.71	-2.52	-1.01	-0.60	0.62	1.01	2.30	4.42	8.33
	JND%	34.00	17.14	12.52	60.86	48.61	51.34	50.96	20.93	10.98	9.97
	Slope	1.02	2.61	4.07	11.34	7.67	13.33	3.28	5.52	4.14	1.38
β_4	Ref. [Nm $\times 10^{-2}$]	-32.93	-16.87	-8.17	-4.50	-2.78	2.59	4.78	10.25	18.22	32.94
	JND%	12.22	19.78	34.07	38.73	36.30	29.90	38.98	20.82	18.81	15.66
	Slope	0.37	0.85	1.09	2.92	4.75	8.62	1.97	2.63	1.11	0.75
β_5	Ref. [Nm $\times 10^{-2}$]	-21.24	-12.02	-9.40	-6.50	-3.60	1.95	6.87	11.37	15.43	29.72
	JND%	13.95	12.32	15.82	22.12	39.03	82.78	31.77	25.92	14.41	30.96
	Slope	0.75	2.04	2.40	5.66	10.02	9.46	5.34	1.43	1.33	0.36
β_6	Ref. [Nm $\times 10^{-2}$]	-41.22	-20.86	-15.07	-8.67	-3.50	4.84	11.86	18.86	25.20	47.48
	JND%	26.72	36.70	37.64	48.73	65.59	59.24	60.10	32.64	23.75	16.61
	Slope	0.33	0.50	3.69	5.67	8.68	7.68	4.30	2.14	1.65	0.92

Table 2.1. Mean values of the threshold and slope distributions estimated in the exploratory test for the 10 reference forces s_i for each joint. These parameters were used as starting values for the adaptive procedure.

2.4 Experiment

2.4.1 Design and Methods

With the aim of collecting more experimental data, the Green's 2AFC Maximum-Likelihood adaptive procedure of measurement has been introduced in [41,42]; this procedure promised highly efficient trial placement and threshold estimation, minimizing the number of trials and thus the session duration. The Green's procedure is similar to the QUEST one but it does not carry so many prior assumptions [65].

The fundamental difference with respect to the procedure previously used was that the comparison values, which were presented to the participant, critically depended on his/her response. That is, while the reference force was constant during a trial, the comparison stimulus changed according to the participant's answer.

The final estimate of threshold was extracted from the most likely psychometric function after a number of trials. To improve the accuracy of these estimates, some catch trials were added. In these trials, chosen at random, the signal was presented at the lowest possible level given the range of assumed psychometric functions [42].

In the example shown in Fig. 2.3, the threshold level appeared to stabilize after about 20 trials. In order to obtain a more reliable estimation, the criterion for the termination of a run was set to 45 trials. With this procedure, the experimental sessions, articulated in 10 magnitude levels, lasted no more than 30 minutes per joint.

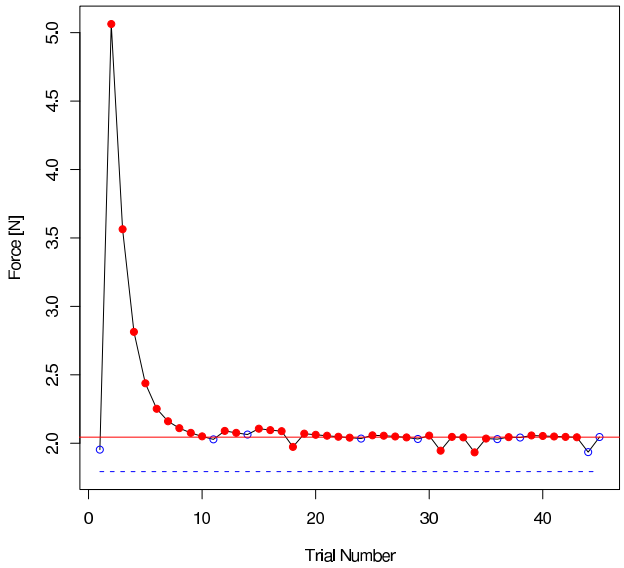


Fig. 2.3. Prototypical track following the Green's Maximum-Likelihood Adaptive procedure. Red filled points refer to positive responses; blue circles to negative ones. Red line refers to the estimated threshold; blue dashed line to the reference force. The estimated threshold appears to stabilize after a certain number of trials.

A critical situation arose when the stimulus level computed for the next trial was outside the range of the haptic device capabilities (i.e., the procedure computed a force value to be generated, but this value was outside the joint capabilities). In this situation this procedure would have converged to a value equal to the maximum force which could be generated by the motor and the session had to be discarded.

2.4.2 Results

Data were analyzed in order to estimate the perceptual threshold for the reference stimuli. Results are reported in Table 2.2. As expected, the adaptive procedure did not converge for some extreme stimuli, missing to estimate 6% of the force thresholds and 19% of the torque ones. The large number of missing data in torques is mainly due to the mechanical short range in torque rendering. Several points proposed by the tracking algorithm were outside the mechanical capabilities.

For each joint, a between-subjects ANOVA (factor: reference stimuli, error term: subjects) was conducted to determine if there were significant differences among the perceptual thresholds due to the different reference values with respect for the different subjects' thresholds. Thanks to the used procedure, which allowed more accurate estimations, and to the greater number of participants, we were expecting significant differences among the reference stimuli. Significant differences (p-value < 0.001) in the JND% along the stimulus continuum were observed for all the translational (β_1: $F_{9,97} = 5.42$, β_2: $F_{9,101} = 8.27$, R_3: $F_{9,103} = 24.56$) and rotational axes (β_4: $F_{9,74} = 6.39$, β_5: $F_{8,85} = 19.27$, β_6: $F_{9,91} = 28.46$).

Joints		Reference Stimuli									
		s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}
β_1	Reference [N]	-5.99	-3.28	-1.80	-0.87	-0.54	0.50	0.83	1.79	3.26	5.98
	JND%	15.3	16.3	18.0	18.1	25.7	31.6	22.7	15.7	17.0	9.6
β_2	Ref. [N]	-7.21	-3.87	-2.06	-0.80	-0.51	0.41	1.07	2.29	4.07	7.35
	JND%	11.62	12.56	19.29	36.93	25.33	29.58	32.66	24.87	13.25	12.16
R_3	Ref. [N]	-8.77	-4.71	-2.52	-1.01	-0.60	0.62	1.01	2.30	4.42	8.33
	JND%	15.73	16.95	19.92	56.47	57.06	48.57	32.47	20.31	12.96	11.25
β_4	Ref. [Nm$\times 10^{-2}$]	-32.93	-16.87	-8.17	-4.50	-2.78	2.59	4.78	10.25	18.22	32.94
	JND%	12.65	23.25	45.06	35.88	41.99	63.02	56.46	31.48	23.85	13.79
β_5	Ref. [Nm$\times 10^{-2}$]	-21.24	-12.02	-9.40	-6.50	-3.60	1.95	6.87	11.37	15.43	29.72
	JND%	11.13	16.20	18.56	17.56	66.73	>100	46.14	31.61	28.19	/
β_6	Ref. [Nm $\times 10^{-2}$]	-41.22	-20.86	-15.07	-8.67	-3.50	4.84	11.86	18.86	25.20	47.48
	JND%	12.11	19.30	21.16	24.02	> 100	46.04	30.86	27.48	20.49	12.21

Table 2.2. Median value of the JND% distribution, estimated with the ML-Adaptive procedure, for the 10 reference forces s_i for each joint.

In Fig. 2.4 the force JND% was plotted versus the reference force for all the joints. We observed a non linear relationship between the reference stimuli and the JND%. Considering the lower reference stimuli, the force stimulus and the JND% appeared to be inversely associated: the lower the force or torque applied, the higher the perceived JND% value. The perceptual thresholds for intense stimuli were higher than in the foregoing experiments: our findings provide some evidence that the perceptual threshold for intense forces or torques can be treated as linear, with an average value of about 15% of the reference force. A very high JND% was observable only for the lower intensity stimuli, in particular for joints β_5 and β_6: the perceptual thresholds were always greater than 45% and often over 100%.

Descriptive analysis suggested the presence of asymmetries in force and torque perception. We hypothesized that along each translational or rotational joint, given a positive or negative force/torque, asymmetric thresholds can be perceived both for low and high intensities. For each joint, we have analyzed the data from the opposite directions both conjointly and separately; as shown in figure 2.5, we have fitted the two exponential functions

$$F_1(x) = a + b_1 \cdot e^{c_1 \cdot x} \tag{2.3}$$

and

$$F_2(x) = a + b_2 \cdot e^{-c_2 \cdot x^2}. \tag{2.4}$$

where a refers to the asymptotical perceptual threshold, b and c are scaling factors. We have considered these functions for their different distribution close to zero: while F_1 grows exponentially, F_2 resembles the bell curve. It is not possible to know a-priori which distribution better accounts for perceptual data from stimuli of weak intensity; thus we were probing for an asymptotic behavior with F_1 or for the presence of a maximum threshold with F_2.

We have conducted this last analysis in order to look for a function which was the best accounting for data, to verify the plausibility of symmetries in the perceived thresholds and to search for the value of the estimated constant a defined

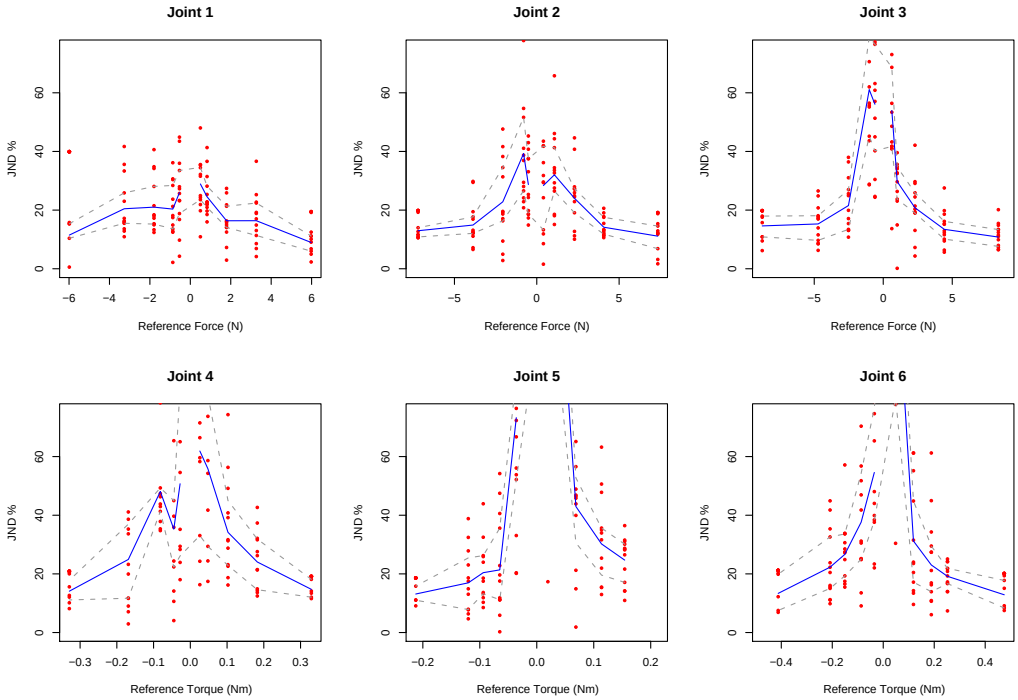


Fig. 2.4. JND% versus reference force and torque. Each point is the participants' threshold. The blue line maps the median values; the dashed grey lines map the first and third quartile.

in the functions, which asymptotically could approach the perceptual thresholds. In order to compare the goodness of these functions, we used one statistical test (the Pearson χ^2) and several information criteria: the R^2, AIC and BIC indexes [20].

Table 2.3 reports the functions which match data better and the values of the constant a, defined in both Eqn. (2.3) and (2.4). For the joints R_3, β_5 and β_6, the function which better accounts for data is F_1 when fitted with different parameters for the two directions; i.e., distinct parameters for the positive and negative direction better explain data variability than unique parameters for the two directions. Considering the remaining joints, data variability is better accounted by F_2, especially when fitted with the same parameters for the two directions.

2.5 Discussion

The goal of this preliminary phase was to understand what is the force range best perceived by a human arm and wrist. We obtained a mean threshold of 15% of the reference force for strong intensities (about forces > 3.0 N or torques > 0.20 Nm), using either of the psychophysical procedures.

We expected to find a similar asymptotic value for force perception: literature [68, 92] reports the JND% as constant for medium values, while it increases for

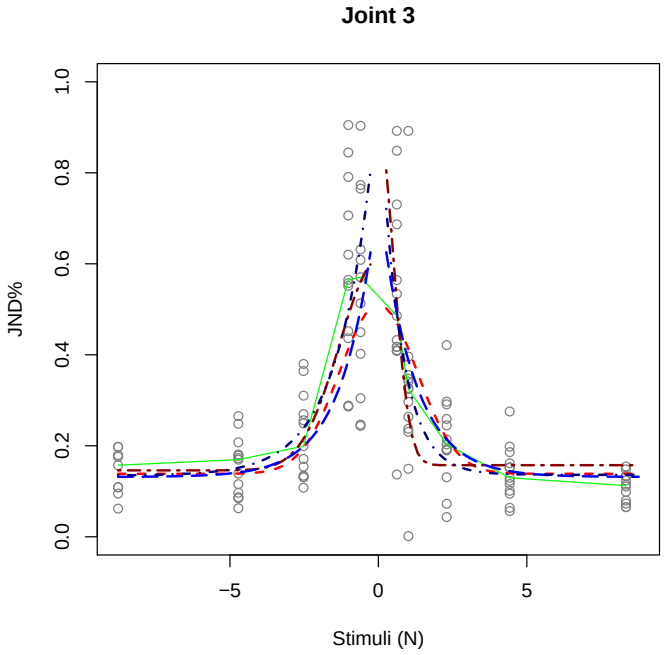


Fig. 2.5. Force JND% versus reference force for joint R_3. Each point is the partecipant' individual threshold. The green line maps the median values; for both the symmetric and asymmetric conditions, the blue curves (dot-dashed and long-dashed) refer to the exponential-type function F_1 while the red ones (dashed and two-dashed) to the bell-type function F_2.

Joints	Model	Pearson χ^2	R^2	Asymptotic JND%
β_1	F_2 Sym.	$\chi^2_{(94)} = 5.36$	$R^2 = 0.995$	$a = 10.90$
β_2	F_2 Sym.	$\chi^2_{(98)} = 7.89$	$R^2 = 0.997$	$a = 11.96$
R_3	F_1 Asym.	$\chi^2_{(97)} = 7.92$	$R^2 = 0.997$	$a_1 = 13.44$ $a_2 = 13.62$
β_4	F_2 Sym.	$\chi^2_{(71)} = 15.18$	$R^2 = 0.992$	$a = 13.49$
β_5	F_1 Asym.	$\chi^2_{(78)} = 32.56$	$R^2 = 0.983$	$a_1 = 17.09$ $a_2 = 25.60$
β_6	F_1 Asym.	$\chi^2_{(83)} = 11.44$	$R^2 = 0.992$	$a_1 = 18.99$ $a_2 = 17.46$

Table 2.3. Best accounting function from the adaptive procedure, the values of Pearson χ^2 and R^2, and the value of the parameter a, asymptotically approaching the perceptual threshold. The a parameter is unique whether the function is symmetric; otherwise a_1 refers to stimulus s_1 and a_2 to s_{10}.

low intensities. For example, using the Controlateral-Limb Matching method, [53] found that low forces were overestimated by an average of 44% whereas larger ones were slightly underestimated (by 9%). Our results were slightly higher in comparison to the literature about the finger force perception [84], but closer to the ones reported in recent works, even not aimed at primarily analyzing the hand-arm force perceptual thresholds [15,52,78]. [15], while investigating the effect of the visual feedback distortion, reported a mean force JND% for young subjects equal to 19% for reference forces greater than 1.5 N. [78] found that the thresholds for detecting changes in steady state steering-wheel force were determined at about 15% for force intensities greater than 5 N and that the JND% was inversely related to the reference force. However, these last works did not specifically considered the perceptual thresholds for low intensities.

Our findings led us to better understand the capabilities of the force perceptual system for low intensities (about forces < 1.0 N and torques < 0.10 Nm). Thanks to the calibration process, the fitted parametric functions seemed to account for the experimental data allowing prediction of the perceptual error. The force perception differed from joint to joint within the range of one joint and also along the movement directions of each joint. These were unexpected results which justify a deeper psychophysical work on force perception with the aim to better understand how the haptic perceptual system differentiate directions. To the best of our knowledge, only few recent works presented similar findings in haptics. For example, [93] reported differences in force discrimination along different directions: a JND% of about 40%, 20% and 30% along x, y, and z respectively.

It is likely that a subject has systematically perceptive distortions [33] and/or privileged directions [114] in the ability to compare forces, especially in near peripersonal space [54]. There exists accumulating evidence, both psychophysical and neurophysiological, that what is haptically parallel is decided in a frame of reference intermediate to an allocentric and an egocentric one. Based upon such demonstrations, it is also hypothesized that the haptic space may be non-Euclidian [79].

2.6 Scaling function

In the previous Sections , we carried out a perception experiment aimed exploring differences in ability of a person to discriminate a wide range of force intensities applied to a hand-grip along the axis of a reference frame positioned at the hand. In particular, differences in terms of force perception relatively to the stimulus intensity and among directions and orientations were observed and identified. This finding could let distinguish the most sensitive directions for the arm, allowing to determine a suitable scaling matrix for force-feedback in haptic environments. In the following we describe how it is possible to use this variable scaling matrix to improve performance in bilateral teleoperation. This is a new approach to force augmentation in teleoperation, since the acquainted concept of scaling in this field concerns mainly *constant* or *linear* amplification, such as the one proposed in [28]. Furthermore, the use of ad hoc scaling functions for each axis direction is also a novelty and its motivation comes from the human perception capabilities.

In Fig. 2.6 a graphical representation of our scaling idea is presented. The yellow area depicted is the one affected by the correct application of Eq. (2.5). The optimal case is to have a horizontal threshold (the green line of the figure) meaning that the JND%, not the force stimuli, is constant along the force domain. Our idea is to introduce the resulting function from the perception curve to scale the force signal, obtaining a constant perception curve which especially allows to better feel differences among low intensities. For example in surgical teleoperation, i.e. minimally invasive robotic surgery (MIRS) scenario, low intensity signals are very common but not well perceived by surgeons, therefore by improving the force perception we may augment the operation performance and reduce the possibility of tissue damages [120].

From the results of our experiments on the force and torque values vs. reference frame axis direction at the handle of the haptic device (see Fig. 2.4 and Table 2.2), we scale the force signal in a selective way, i.e. by using a specific scaling function per axis' direction, obtaining a scaling matrix that manipulates the forces at the master side.

We defined our perception-based force feedback scaling function on the basis of

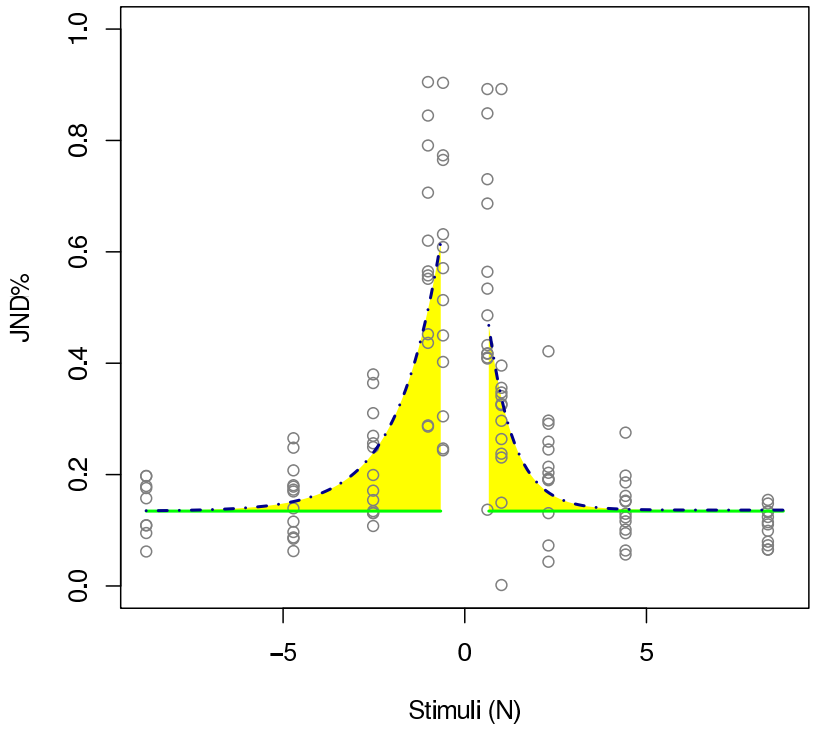


Fig. 2.6. Force threshold vs. reference force for one prototypical direction. The yellow surface represents a hypothetical manipulation of the force signal considering as scale factor an exponential functions, in order to reach a constant threshold along the stimuli continuum (green line).

Equations (2.3) and (2.4) as follows:

$$\alpha(F) = \begin{cases} 1 + e^{k_{pos}(-F+c_{pos})} & \text{if } F > 0 \\ 1 + e^{k_{neg}(F+c_{neg})} & \text{if } F < 0 \end{cases} \tag{2.5}$$

where F is the unscaled force (or torque) signal along a specific direction, k is a parameter referred to the human sensorial threshold, and c is a parameter referred to the accuracy of the calibration of the experimental setup and it is dependent to the human capability for a specific configuration. Tuning the parameters in the right way (i.e. parameters customized from the findings of the previous experiment) leads to a different scaling function for each directions.

This amplification of the force signal, with a non constant scaling, should erase the JND% degradation. The behavior of our scaling function is, quite obviously, amplifying the differences at lower forces without scaling the high intensity forces, that are already well discriminated by human beings, while maintaining the uniqueness of the force values.

The approach to the parametrization of the proposed scaling function is twofold: it is possible to use average values derived form the perception experiment knowing that the resulting scaling is well suited for everybody, or it is possible to use the thresholds obtained from the experiment by a specific subject in order to have a "personal" scaling. The drawback of the second strategy is that an entire experiment have to be performed in order to identify c and k. We are currently working on a simple and shorter system calibration procedure for an ad-hoc use of the model presented.

A side effect of our method can be an alteration of the distance among stimuli even if it does not compromise the order relation.

Since our goal is to make the telepresence experience more significant for the human, it is necessary to evaluate the goodness of the scaling with respect of the user perception. In order to verify this aspect we arranged a new psychophysics experiment and we tested the design with few subjects. We used the same setup of [118], while adding the scaling function. We omit the details since in the near future we will give more statistical relevance to the data we are collecting involving more subjects. However the underlying idea is that, if our assumption holds, the subjects should be able to discriminate small differences in the forces even at low intensities. As shown in Fig. 2.7 for the positive side of one direction, the application of Eq. (2.5) modifies the JND trend and results in a perceptual threshold that is constant along the whole force range, and it is characterized by lower standard deviation, similar to the green reference of Fig. 2.5. A constant and lower JND means higher ability to discriminate small variation of the forces. One possible shortcoming deriving from our variable force scaling approach is a "flattening" of the force intensity along the whole range. In a specific preliminary experiment we find evidences that our functions are an order-preserving scaling about intensities, and this early result make us confident about the applicability of different force scaling functions.

We would like to remark that this type of force signal manipulation is correct if the absolute value of the forces is not significant while it is important that the subject discriminates low intensity stimuli. Therefore the choice to use the force

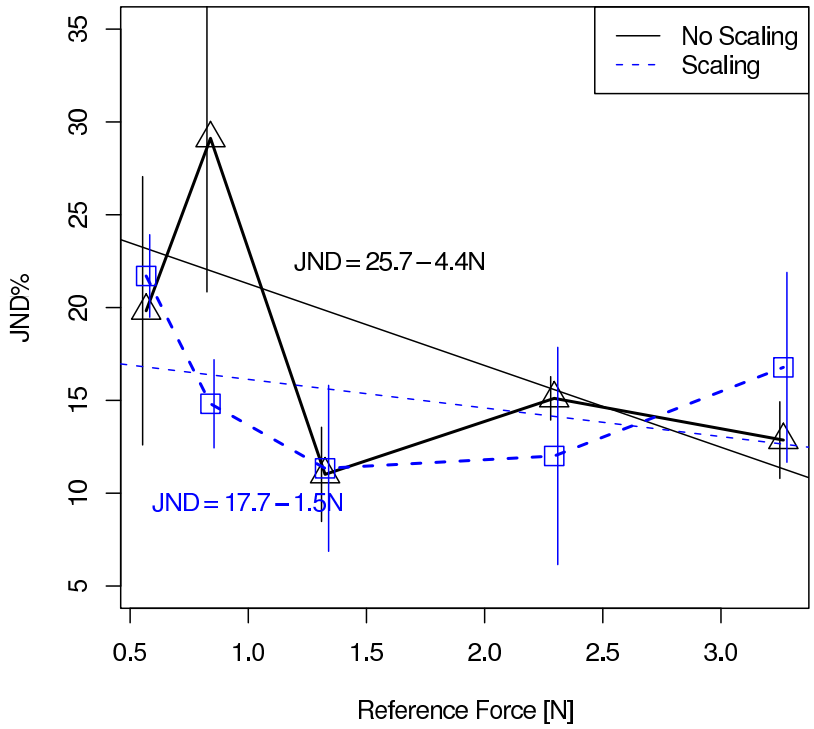


Fig. 2.7. Force JND versus reference force; data are referred to one prototypical direction (movement close-far). The black line refers to the experiment with non scaled forces, while the blue dashed line refers to the perceptual thresholds applying the force scaling function. A linear fit is depicted for both approaches in order to show the overall trend.

dependant scaling have to be compatible with the task that has to be performed in teleoperation. In Minimally Invasive Robotic Surgery, for instance, and in general in all macro/micro manipulation tasks, it is more important to let the operator perceive or discriminate between stimuli than render a "perfect" force. At the end we are proposing a different concept of transparency. In our perception-centric approach we consider "transparent" what is better perceivable, even if it is different from what it is registered at slave side.

Unfortunately, it is not granted that using a variable scaling factor in the interconnection between master and slave sides leaves the overall telemanipulation system stable.

In [13] a demonstration of the stability of the scaled teleoperation system in the case of negligible communication delay using Port-Hamiltonian system and the theory of passivity was presented.

Passivity theory has been widely used for the control of bilateral telemanipulators since it allows to guarantee a stable behavior of the system both in case of free motion and of contact with any, possibly unknown, environment thanks to impedance control techniques [46]. Port-Hamiltonian systems can be used for the design of passivity based bilateral teleoperators. They are passive systems and

they allow to model all physical systems and, furthermore, to represent very clearly the energetic structures and the power flows [99].

Our variable scaling can be used to improve performance in passivity based bilateral teleoperation. To this aim, it is necessary to join master and slave sides by a properly scaled interconnection. Nevertheless, a variable scaling is not a passivity preserving operation and, therefore, in principle it should not be used in passivity based bilateral teleoperation systems.

The problem of *constant* scaling of the velocities and of the forces exchanged between master and slave sides in passivity based teleoperators has been already solved in [98]. Nevertheless, the fact that the scaling factors are non constant makes the problem significantly harder and the results obtained in [98] not directly applicable. In [13] the authors, exploiting the port-Hamiltonian formalism, demonstrated how under reasonable assumptions it is possible to embed the variable scaling into a port-Hamiltonian teleoperation system while preserving a stable behavior of the system.

2.7 Conclusions

In this Chapter we focused on the role of human perception in haptics and teleoperation. We designed a setup for the execution of perception experiments by using a non commercial high performance haptic device. The NASA/JPL Force Reflecting Hand Controller was used to provide force stimuli in the Cartesian space to several subjects and collect results. The joystick was equipped with a FPGA board for the low level handling of the device and precise force generation. An important effort was spent to make the reflected force/torque signal correct and a complete calibration of the device was required. Then we collaborated, under the supervision of a psychologist, to the definition of a protocol that has been used to carry out experiments aimed at investigating the force threshold perceivable on the human hard-arm system when a haptic device acts as intermediary in the information transfer. The main goal was to study the low intensity force signals vs the human perception. The experimental results permitted the identification of force thresholds and the observation of asymmetries in the JND% plots. On this basis we determined a set of scaling functions, one for each degree of freedom of the three-dimensional space, that can be use to enhance the human abilities in discriminating different stimuli. The variable force scaling operates especially on low intensities forces and permits a stable behaviour while increasing performance.

3

Impact on haptics: requirements, issues and solutions

The scaling functions we found, described in the previous Chapter, or any other perception-based enhancement must bu used in the teleoperation control loop.

During bilateral teleoperation Cartesian positions and orientations are sent from the master device to the remote arm using a communication channel. The information about the interaction with the environment collected by the slave sensors are sent back to the master. Since master and slave may have different kinematic structures, specific coordinate conversions are used to transform joint positions and force feedback data from one reference frame to the other. In addition, information about the dynamics structure and configuration of the system can be used.

In order to make the overall architecture stable, these computations have to be performed at high speed and in a predictable manner. As described extensively in the technical literature, a 1 KHz computation speed should be achieved. In certain teleoperation tasks, especially with high approach velocities and stiff environments, high update rates and low delays are key requirements for stability and thus for realistic haptic perception [58]. Thus summarizing, at least once per millisecond the controller has to:

- read the encoders in order to obtain the current position of each joint of the haptic device;
- perform a forward kinematic transformation that calculates the Cartesian position of the hand-grip from the position of the joints. This information is sent to the slave device;
- receive the force feedback information from the slave. This is expressed in Cartesian space;
- modify the force signal in a perceptual "useful" way, for instance using our scaling functions;
- apply the transformation that maps modified force and torque data to the corresponding torques applied by the haptic device motors;
- implement a control strategy.

When the feedback is rendered on the haptic device it already has to contain the force signal manipulation. Moreover the control loop could use dynamics information, even considering the human arm, in order to track the force reference

with minimal error. The outcome is that any modification has to easily fit into the whole transformation process and elaboration chain and to be easily replaced. In fact, when talking about perception-based enhancement, the human-centric force signal manipulation is dependent on the human operator and on the task itself. While a mean and generic set of functions can be found one and for all through a specific psychophysic experiment, this set is related to the task analyzed, and even if good, it should be "fine-tuned" to the particular abilities of the operator. This great variability cannot be resolved just thinking at it as the setting of a number of parameters, maybe from a calibration-like procedure, since we forecast that the alternatives in force signal manipulation can be widely different. The underlying consideration, and the reason for the approach we present in the following Sections, is that a new block in the control loop of the haptic device controller has to be added without negatively affect the overall performance while maintaining stability.

The control loops for robotic devices usually require a rate higher than 1 KHz. For stability issues it is crucial to have real-time, or in other words strong deterministic behaviour and good speed, in order to meet control deadlines. Thus it is important to guarantee that the complete computation is fulfilled in 1 ms. In addition the control architecture has to permit an easy and fast variation and reconfiguration. In the aim of selecting a target platform for the implementation of the controller of a haptic device, the main point is to choose a hardware configuration that permits to obtain good performance in terms of speed, determinism and reliability and to grant the possibility of easy re-implementation.

Since we would like to exploit also hardware capabilities in order to achieve our goal, it is important to look at current trend in the development of CPUs, embedded systems and control systems. In particular recent literature seems to indicate that the new road about the development and use of processing systems leads to multicore architectures [71].

In the following we present an overview on multicore architectures and programming models able to exploit these architectures.

3.1 Multicore architectures

Computer science and programming cover many specialized fields in which the ability to perform high performance computation is crucial. From this continuously growing need, the demand for more powerful processing units arises. While manufacturing technology keeps improving, reducing the size of single gates, physical limits of semiconductor-based microelectronics have become a major design concern. Some effects of these physical limitations can cause significant heat dissipation and data synchronization problems. In the last few years moving the focus from increasing processors speed to introducing new form of parallelism has become the dominant mechanism for scaling processor performance. In particular the inclusion of multiple cores on a single chip has become the dominant trend.

Parallel computer architecture are not a novelty, but the widespread utilization due to cost affordability is new. However one of the most effective and used classification of computer architectures based upon the number of concurrent instructions

(or controls) and data streams available is the old Flynn's taxonomy [34]. Flynn describes four different types of categories:

Single Instruction, Single Data stream (SISD): a sequential computer which exploits no parallelism in either the instruction or data streams. Examples of SISD architecture are the traditional uniprocessor machines like a PC or old mainframes.

Single Instruction, Multiple Data streams (SIMD): a computer which exploits multiple data streams against a single instruction stream to perform operations which may be naturally parallelized. For example, an array processor or GPU.

Multiple Instruction, Single Data stream (MISD): multiple instructions operate on a single data stream. Uncommon architecture which is generally used for fault tolerance. Heterogeneous systems operate on the same data stream and must agree on the result.

Multiple Instruction, Multiple Data streams (MIMD): multiple autonomous processors simultaneously executing different instructions on different data. Distributed systems are generally recognized to be MIMD architectures; either exploiting a single shared memory space or a distributed memory space.

The taxonomy can be further enriched with the following categories:

Single Program, Multiple Data (SPMD): multiple autonomous processors, with a one's own thread of control each, simultaneously executing the same program at independent points on different data. Also referred to as 'Single Process, multiple data' [26]. SPMD is the most common style of parallel programming. An SPMD processor includes explicit support for coordinating threads.

Multiple Program Multiple Data (MPMD): multiple autonomous processors simultaneously operating at least two independent programs. Typically such systems pick one node to run one program that delivers data to all the other nodes which all run a second program. Those other nodes then return their results directly to the coordinator.

During the years other minor differentiations have been proposed.

The Flynn's taxonomy considers a processor as a single unit of computation, but nowadays other concepts are used.

A core is a processing element with an independent control flow. Cores can vary in computational power and can also present various types of internal parallelism.

A functional unit or execution unit is a part of a CPU that performs the operations and calculations called for by the computer program. It may have its own internal control sequence unit, some registers, and other internal units such as a FPU, or some smaller, more specific components. It is commonplace for modern CPUs to have multiple parallel execution units, referred to as scalar or super-scalar design.

It is not always correct to compare the amount of internal parallelism available on a specific architecture by just counting the number of cores, since usually a core might have hundreds of functional units. For instance a Cell BE has 9 cores

with 4 functional units per core and a Graphical Processing Unit can have up to 320 functional units. In addition a lead role in the ability of the processor to carry out several computations in parallel is given by the hierarchical structure of the cores, the interconnection among them and the memory organization and access.

Nowadays it is possible to find a broad range of parallel hardware architectures on the market, and they usually belong to one of the categories presented above. However, when several design solutions are applied to the same processor in order to maximize parallelism, it can be difficult to identify only one choice. Thus providing a simple classification is hard.

In recent multicore architecture it is common to find a combination of the above characteristics. For instance in a heterogeneous processor a coordinating core can provide multithreaded capabilities while the other cores could permit SIMD parallelism.

3.1.1 Architectural features and processors

In this Section, we will survey at a high level some specific parallel processor architectures. We focus on the various forms of parallelism exploited by these architectures. The processors are described with respect to their physical structure and, when possible, their category in the Flynn's taxonomy is given.

Simultaneous multithreading (for instance Intel's HyperThreading) was an early attempt to have a pseudo-multicore system. A processor capable of simultaneous multithreading has only one execution unit, but when that execution unit is idling (such as during a cache miss), it uses that execution unit to process a second thread. This type of processors is inexpensive and easy to obtain since fewer resources need to be replicated but it is hard to achieve as much of a speedup as using different cores. It is a MIMD processor.

A **vector processor** is a CPU or computer system that can execute the same instruction on large sets of data. "Vector processors have high-level operations that work on linear arrays of numbers or vectors. They are closely related to Flynn's SIMD classification. Cray computers became famous for their vector-processing computers in the 1970s and 1980s. Modern processor instruction sets do include some vector processing instructions, such as with AltiVec and Streaming SIMD Extensions (SSE).

A **multicore processor** implements multiprocessing in a single physical package. It replicates a single core design, duplicating it several times on a single chip. Each core can run a completely separate thread of control, so this is an MIMD processor. It can be described as an integrated circuit to which two or more cores have been attached. The cores are typically integrated onto a single integrated circuit die (known as a chip multiprocessor or CMP), or they may be integrated onto multiple dies in a single chip package. These processors differ from superscalar processors, which can issue multiple instructions per cycle from one instruction stream (thread); by contrast, a multicore processor can issue multiple instructions per cycle from multiple instruction streams. There are a number of different multicore design, and they are different in how they manage memory and intercore

communication. Cores in a multicore device may be coupled together tightly or loosely. For example, cores may or may not share caches, and they may implement message passing or shared memory intercore communication methods. Common network topologies to interconnect cores include: bus, ring, 2-dimensional mesh, and crossbar. All cores are identical in homogeneous multicore systems and they have different designs in heterogeneous multicore systems. Just as with single-processor systems, cores in multicore systems may implement architectures such as super-scalar, vector processor, SIMD, or multithreading.

Traditional central processing units (CPUs) have recently begun adding multiple cores and typically they adhere to the homogeneous paradigm.

The IBM Cell Broadband Engine is a heterogeneous multicore chip composed by one general purpose core and eight other smaller cores specialized for numerically intensive workloads. These smaller cores also have explicitly managed local memory and SIMD parallelism, and can communicate directly with each other via message passing over an on-chip ring network.

General-purpose computing on graphics processing units (GPGPU) is a fairly recent trend in computer engineering research. GPUs are co-processors that have been heavily optimized for computer graphics processing. Computer graphics processing is a field dominated by data parallel operations, particularly linear algebra matrix operations.

In the early days, GPGPU programs used the normal graphics APIs for executing programs, such as the OpenGL Shading Language (GLSL). This approach was hard form a programming point of view since all the computations have to be transformed into shading operation on texture. However, recently several new programming languages and platforms have been built to do general purpose computation on GPUs with both Nvidia and AMD releasing programming environments with CUDA [25] and CTM [5] respectively. Other GPU programming languages are BrookGPU, PeakStream, and RapidMind. GPU architectures are designed to support a massive number of threads running simultaneously but are able to suspend threads to hide the latency of certain operations, such as memory reads. In addition, SIMD computation is often exploited over a set of spatially coherent tiles, and on most GPUs, SIMD Within A Registe (SWAR) or Very long instruction word (VLIW) parallelism is also available. Since the threads (or kernels) are the same program in different moment of execution this is an example of SPMD architecture.

Within parallel computing, there are specialized parallel devices that are not recognizable as processors, since they have a completely different structure and functionality. However it is important to mention them since they represent an important niche of applications in which performance is the main goal.

A Field-Programmable Gate Array (FPGA) is, in essence, a computer chip that can rewire itself for a given task. It can be used standalone, usually for DSP and for controlling devices, or as reconfigurable computing, i.e. as a co-processor to a general-purpose computer. In this case usually it is part of a System on Chip

(SoC), where a CPU, an FPGA are put side by side on the same board and share connections with main memory and peripherals.

FPGAs can be programmed with hardware description languages such as VHDL or Verilog. However, programming in these languages can be tedious. Several vendors have created new languages that attempt to emulate the syntax and/or semantics of high level programming language, with which most programmers are familiar. One example is given by SystemC, that is based on C++.
Sometimes the term "stream processing" is used to refer to pipeline parallelism, where the output of one task is sent directly to another in a producer/consumer model. It can use multiple computational units without explicitly managing allocation, synchronization, or communication among those units. FPGA accelerators often make extensive use of this form of parallelism related to SIMD.

Several application-specific integrated circuit (ASIC) approaches have been developed for dealing with parallel applications. These were the almost unique solutions having full parallelism before the existence of multiprocessor and multicore systems. Because of his specificity for a given application, an ASIC can be optimized and tends to outperform a general-purpose computer. However, ASICs are created by an expensive lithography process. High initial cost, and the tendency to be overtaken by Moore's-law-driven general-purpose computing, has rendered ASICs unfeasible for most parallel computing applications. However, some have been built. For instance there are machines which use custom ASICs for molecular dynamics simulation. They often represent the term of comparison during the evaluation of algorithms and techniques implemented in FPGA or general purpose processors.

Furthermore the general trend in processor development has been from multicore to manycore; A manycore processor is one in which the number of cores is large enough that traditional multi-processor techniques are no longer efficient. This threshold is somewhere in the range of several tens of cores, and likely requires a network on chip. In addition, multicore chips mixed with simultaneous multithreading, memory-on-chip, and special-purpose "heterogeneous" cores promise further performance and efficiency gains, especially in processing multimedia, recognition and networking applications.

The most important example of multicore processors seen above are definitely GPUs and the Cell BE (and in very specific case FPGAs) because these are the consumer processors that currently support the most explicit parallelism. In addition the potentiality of these specialized processors is due to the large availability and the relatively inexpensiveness. CPUs, however, are clearly evolving in the same direction [81]: towards massive parallelism and "many-core" architectures. Since these architecture share same form of data stream, or at least some of them are equivalent to others it is possible to apply similar programming solution or approaches to all of them.

3.1.2 Programming approach

In order to really obtain important benefits by the adoption of a multicore system it is necessary to change the programming habits in order to identify and exploit the parallelism, when present, in the algorithm to be coded.

In the last decades, most programmers have been using a serial model of computation to design and implement programs, because of the SISD nature of general purpose CPUs. However, digital hardware is naturally parallel. In order to obtain increased performance, compiler and processor designers have struggled to automatically exploit implicit instruction-level parallelism (ILP) in serial code. For example, instructions can be rescheduled, either by the compiler or by the hardware, in order to best exploit pipeline parallelism in functional units or multiple instruction issue in superscalar processors.

However, experience has shown that most serial programs have limited implicit parallelism.Automatic extraction of parallelism without explicit assistance from the programmer, particularly in hardware but also in software, has reached the point of diminishing returns. Therefore, there is renewed interest in explicitly parallel programming models, languages, runtimes, and platforms.

We can identify mainly two types of parallelism:

Task parallelism that is based on the idea of decomposing a program into separate tasks and running these tasks at the same time on different processing elements. It is easy to see a relationship between task parallelism and MIMD computation. For instance, in a multicore system, task parallelism is achieved when each core executes a different process with the same or different code. In the general case, communication among processes permits the workflow data passage.

Data parallelism that is based on the idea that operations on collections of data, such as arrays, can themselves be performed in parallel. Usually this type of parallelism is exploited by SIMD and SPMD architecture. In a multicore system, data parallelism is achieved when each core performs the same task on different pieces of distributed data. In some situations, a single execution thread controls operations on all pieces of data. In others, different threads control the operation, but they execute the same code.

Task parallelism emphasizes the distributed (parallelized) nature of the processing, as opposed to the data (data parallelism). An algorithm can present both of them, and usually the program is built around these characteristics.

In order to get the best performance out of multicore processors, an explicit mechanism allowing the programmer to directly express a parallel computation is desirable. It is useful to design such mechanisms around programming models [102].

A programming model is an abstract model of computation that is used by the programmer to reason about how a program executes. A programming model is an abstraction and may not explicitly expose every parallelism mechanism available in the target processor.

A processing model is similar but describes how a physical machine actually performs computation. Or rather, the processing model is the programming

model exposed by the processor vendor and used in its instruction set architecture. The processing model is usually hardware or vendor specific and is designed to obtain the maximum in terms of performance.

The **programming language** has the role to efficiently translate an algorithm expressed relatively to the programming model used by the programmer into the processing model used by the target hardware.

In the ideal situation, the programming model expresses the most important aspects of the processing model and the programming language implementation simply automates the mapping of the desired computation onto the target hardware, relieving the programmer from the more tedious aspects of the procedure.

Programming and processing models are important because these models are what a programmer uses to reason about how a computation actually takes place on a physical computer. Programmers use a programming model to reason about how a computer operates and to design efficient algorithms. If a programming model is an accurate reflection of how a computer actually performs a computation in hardware, and if the primary features of the model as understood by the programmer reflect the most important architectural characteristics of the computer, then it will be possible for a programmer to design efficient algorithms for the computer. If the programming model does not allow manipulation of some important features of the processing model of the target hardware, or does not provide enough control over the final implementation on the real machine, then the programmer will be unable to optimize the code for the best performance.

The most important aspects of the processing model are those that have the most impact on performance. A programming language implementation should unburden the programmer from dealing with trivial questions, allowing the programmer to focus on making major policy decisions and on designing an efficient (and correct) algorithm. With a parallel programming model, the programmer is encouraged to think in parallel and give the system a large amount of latent parallelism to work with.

Unfortunately, the programming models of mainstream programming languages were developed a long time ago when computers were relatively simple and mainly SISD. In fact they lack native and easy to use support for exposing efficiently the computation parallelism. In addition to parallelism, there is also the issue of memory access. Poor use of the memory system can degrade the performance of a program by several orders of magnitude thus a good programming model should also allow the clear expression of memory locality and data movement. Again the memory organization and access have to be explicitly expressed by the programmer also when a native software development kit is available for a specific architecture. Thus the ability to handle it in an abstract way and have the programming language and the compiler perform the mapping is not yet possible.

In [71] a detailed survey of current programming models, language and platform is provided. The main conclusion is that many subjects and vendors are working on programming languages more oriented to multicore systems, and there are some valuable results. However it is still a very inhomogeneous field, and there are no available solutions able to target all the different architectures and to provide that transparency from hardware that is required. Thus it seems that the quality, in

terms of performance and simplicity, of a parallel implementation is mostly left to the programmer's skill.

3.1.3 Rediscovering old parallel algorithms

Computer science and programming are relatively new disciplines, while algorithms have existed in mathematics, algebra and calculus from centuries. In the continuous challenge of discovering solutions for an increasing number of application fields it is common practice to reuse existing algorithms and to apply them to new areas. In [111], dated 1984, the author presents a research overview, pointing out advantages, shortcomings and possible applications, of Residue Number System (RSN) due to the progress in technology of those days when VLSI was a promising and emerging technique. A RSN represents a large integer using a set of smaller integers, so that computation may be performed more efficiently. The description of the theoretical core of RSN (the Chinese remainder theorem) dates back to the 4th century AD in the Master Suns Arithmetic Manual written by Tsu Suan-Ching. The topic was reprised and analyzed in [108], mainly from a mathematical point of view, but unfortunately the technology of the 60's was insufficient to support the unique demands of the RNS. Nowadays RNS is widely used, especially in FPGA computing, in many cryptographic, fault tolerant, DSP systems and it is still an important research topic [24, 60, 74, 82, 115].

Referring to the definitions above, the translation from a "programming model", i.e. an abstract solution, into an existent "processing model" is not always practical. In addition, the evolution in the hardware capabilities of the processing units can encourage the adoption of a new algorithm in spite of another one that worked well until then. This does not mean that the algorithm is no more interesting or that in other scenarios cannot be used again but, in those specific conditions, it is not attractive. It is also quite common to try to apply any new solutions to a wider area of problems. For instance, when genetic algorithms and neural networks proved their goodness they were employed in many applications in several fields while traditional approaches were discarded.

Sometimes the old solution is not really put aside, but its use is continued in a specific niche of employment. A good example is the CORDIC algorithm. CORDIC (COordinate Rotation DIgital Computer) is a simple and efficient algorithm to calculate hyperbolic and trigonometric functions. It is commonly used when no hardware multiplier is available (e.g., simple microcontrollers and FPGAs) as the only operations it requires are addition, subtraction, bitshift and table lookup. The modern CORDIC algorithm was first described in [119] and it was developed to replace the analog resolver in the B-58 bomber's navigation computer. These days, CORDIC algorithm is used extensively for various applications, especially in the FPGA domain. Xilinx, one of the most famous manufacturer of FPGA chipsets provides CORDIC IP cores that can be customized and easily used on their boards.

A different consideration holds for algorithms that are state of the art in their domain, such as algorithms for ray-tracing in 3D rendering, but that are still considered batch solutions due to computational issues. When the algorithm has some parallelism, from the task or data point of view, then a parallel implementation can move the computation towards interactivity.

As stated above it is not uncommon that "old" algorithms remain useful in a specific domain or become useful again due to changes in processor technology. After an era dominated by SISD solutions caused by the intrinsically sequential nature of CPUs available on the mass market, in the new age of multithread and multicore systems it is worth looking for ASIC and VLSI solutions since they were inherently parallels.

The main question remains how much effort is required in order to write these algorithms, usually described in VLSI or schematics, in a modern programming language. In other terms it is necessary to extract the programming model from them and find a fast and easy way to translate it in the processing model of the target hardware. Of course any help that a modern programming language can give is important but, as stated above, no automatic translator nowadays can do all the work by themselves. Thus a methodology, or some guidelines, can help the programmer to obtain the expected results.

3.2 Previous work on robotics

In literature there are many works demonstrating the advantages of the use of multicore systems for complex application that require high performance and massive computations. There are several application areas in which this new parallel architecture can be applied with success, such as collision detection and responsive action computation, modeling and control of soft articulated characters, physically-based simulation of fluids and solids, crowd and multiagent simulation, interactive ray tracing, sound synthesis and propagation and scientific computations.

There are still many open challenges regarding hardware, programming, software and applications. The increasing interest of the scientific community in the potential of multicore systems is visible in the continuous proposal for workshops [96], special issues of journals [66], articles and thesis works.

Initially there has been a huge effort from the major enterprises involved in the hardware development. There are several technical reports from NVIDIA, ATI and IBM that show implementations of matrix multiplication, fast Fourier transform, imaging filters, ray tracing and so on.

Nowadays there are countless scientific applications that range from video decoding [6], DNA analysis [125], ray tracing cite [12, 89], image filtering [7], cryptography [44, 109], implemented on GPU, Cell and FPGA architecture.

Besides these consolidated results, it is important to understand if teleoperation, and hence robotics and haptics, can also benefit from the development of all these new hardware technologies.

It is easy to imagine that the main utilization of GPU-based solutions is in robotic applications that require accurate graphical and physical simulations. There is a strong interest in robotic aided surgery; usually it consists of a teleoperation setup where the surgeon uses a haptic device to give command to the robot that interacts with the patient. In this scenario, it is useful to have the possibility to use the haptic device with a virtual simulation instead than the real robot/patient for both training and pre-operative planning. In this case the simulation of

the human organs, fluids and surgical instruments and their physical interaction have to be rendered with the same constraints of real teleoperation. In [3, 29, 126] GPUs are used to make this kind of simulation possible and realistic.

Unfortunately this seems to be the main contribution of multicore systems to teleoperation. Until now the standard approach is to have the software handling communications, human interfaces, planning and supervision, and dedicated hardware for the real-time control of the devices.

In the last years several interesting software architectures for robotics were proposed. Smartsoft [94], Orca [17], Miro [57], Player [116] and CLARAty [77] are well known frameworks that rely on the concept of modular design and component-based software engineering. The main idea is to decompose a robotic system in modules while providing communication and synchronization systems as well as high level control strategies and algorithms. Almost all these frameworks are designed for mobile and cooperative robots, consequently with little focus on real-time constraints such as the ability to meet deadlines, to maintain specific control rates and to provide security and recovery functionalities.

The inherently heterogeneous nature of the teleoperation task requires to maintain a good level of abstraction in defining modules and, at the same time, to implement reliable low level controls. In addition, some real-time capability is needed to maintain transparency and stability while data are sent over the communication channel, dealing with unpredictable time delays and low bandwidth. This ambivalence in the solution approach excludes the utilization of control software architecture used in automation processes. Orocos [18], MCA2 [36] and Penelope [38] have good real-time capabilities while maintaining the modular and heterogeneous approach.

Since they are aimed at distributed systems they can be managed to work with multicore processors, splitting the execution among cores instead that among hosts. There are real-time versions or extensions of the Linux kernel that provide SMP/multicore support and thus the robotic software frameworks that rely on them can be used on parallel architectures. This is the case, for instance, of Orocos, Penelope and Orchestra [97] that use RTAI (Real-Time Application Interface). However they do not provide explicit mechanisms to handle data parallelism and to easily exploit SIMD architectures. YARP [73], that is a more recent architecture, is an almost unique example of robotics framework that permit to program and use GPGPU, through NVIDIA CUDA programming language, moving the most computational intense part of the controller on the graphic board.

Regarding the control of manipulators and haptic devices the main solution is still the utilization of industrial controller, axis board and ad-hoc electronics. From a software point of view highly optimized procedure based on the specific configuration of the robot are preferred to generic but computationally more intense approaches. Of course the drawbacks are that any change in the controller strategy requires complex code rewriting and that the solution is not reusable on different hardware.

An attempt to use FPGA for controlling a haptic device is presented in [37] combining the speed of the hardware with the ability to reconfigure it. In [88] the authors develop an embedded real-time controller for cable robot.

Beside the industrial controller and the embedded systems in literature there are examples of robotic arms controlled by PC using real-time operating systems, from the proprietary and expensive VxWorks to open source real-time Linux. In [83] is presented the classic evolution from the industrial controller to the Linux based solution for a PUMA 560 arm, one of the most used in universities.

While it is difficult to find robotic applications more aimed at multicore architectures than to distributed systems, it is however possible to find parallel solutions studied years ago for custom parallel ASIC and VLSI implementations. For instance in [63, 64] a comprehensive evaluation of algorithms for robotic arms forward and inverse dynamics computation is presented. In particular the authors identify the strategies that are more suitable to a parallel execution. In [23] a similar approach for the manipulator pseudo-inverse Jacobian computation is shown. This is an important topic since the Jacobian is one of the most important quantities in the analysis and control of robot motion, and its inversion is a crucial aspect in many controllers and applications. It can be interesting to implement these solutions on recent multicore architecture.

3.3 Conclusions

With all these consideration in mind, we want to check whether our field of interest, bilateral teleoperation, can take advantage from the current trend in the development of multicore systems. In order to do that, we decide to understand if there exist some algorithms, inherently parallel, that can exploit a multicore architecture and have good performance while solving a general problem with specific requirements that can also have an important impact on haptics, teleoperation and robotics. As a proof of feasibility we want to deal with the implementation on a multicore architecture. In the coding phase we will not care about specific details such as memory management. We leave it to smart multicore aware programming language. However we want to show that a careful choice of data organization, an appropriate task decomposition and a coding that mimics the ASIC implementation could give good results. The underlying idea is that since automatic parallelization is not yet achievable and programming language multicore aware are still in development, the programmer can improve the performance carefully choosing his/her programming style. When the algorithm has its origin in ASIC, an implementation that exploits low level instructions (or better shift-and-add) can easily outperform a high level approach, without dealing with machine dependent optimizations and fine tunings.

In the next Chapters we will focus on a generic application, that can be critical in robotics and teleoperation due to the requirements and constraints that we already presented: the matrix pseudo-inversion. In order to show the feasibility of our idea we will eventually describe the implementation on a heterogeneous multicore system: the IBM Cell Broadband Engine.

4

Strategies and approaches to multicore architectures

In the previous Chapters we presented our findings about human perception based functions that can be used in teleoperation to enhance the immersive experience. Since they have to be part of the control loop of the haptic device used in the teleoperation setup, we highlighted requirements and issues, aiming toward a multicore solution. In particular we stated that in order to obtain inherently parallel procedures it can be useful to look for solution developed for ASICs and VLSI. We found specific algorithms regarding parallel forward and inverse dynamics of manipulator. In order to show our approach that reuse this algorithms in modern multicore architecture we can focus, specifically, on their implementation. However they are almost completely related to robotic manipulation. We would like to show that the approach is valid for any type of porting. Thus we focus on a more generic problem, that is important for robotic as well for other fields. In addition, many existing solutions rely on very basic operations that can represent bottlenecks in a robotic or haptic system: matrix multiplication and inversion. There are several works about matrix multiplication on multicore systems, since this is an interesting topics and the solution can potentially fully exploit the parallel hardware capabilities. The matrix inversion is a more complex problem that has a number of possible alternative solutions, but it becomes particularly relevant when real-time requirements are added. In the following we present an overview of the algebraic problem represented by matrix inversion in one of its more complex form: the pseudoinversion.

4.1 Ill-conditioned linear systems: the inverse problem

A linear system of equations is a set of linear equations involving the same set of variables, also called unknowns. A solution to a linear system is an assignment of numbers to the variables such that all the equations are simultaneously satisfied.

A linear system is called inconsistent if it does not have a solution. There are three typical cases, based on the number of equations of the system, n, and the number of unknowns, k:

1. If $k < n$, then the system is (in general) overdetermined and there is no solution.

2. If $k = n$ and the system is consistent, then it has a unique solution in the n variables.
3. If $k > n$, then the system is underdetermined and there are infinite solutions.

Linear systems can be represented in matrix form as the matrix equation $Ax = b$, where A is the matrix of coefficients, x is the column vector of variables, and b is the column vector of solutions. When the system is consistent and $k = n$ the unique solution is given by $x = A^{(-1)}b$, where $A^{(-1)}$ is the inverse of matrix A. If the system is either overdetermined or underdetermined then $n \neq k$, thus the matrix A is not square and hence not invertible. This is congruent with the cases presented, since the system has no solution or infinite solutions. However in real cases this condition can happen and it is necessary to develop mechanism to deal with it.

In order to justify the last sentence let us present a particular class of problems.

Using a physical theory for predicting the results of observations corresponds to solving the **forward modelling problem**. The forward problem has a unique solution, because of the causality principle [110]. This methodology is used in particular for simulation purposes.

In an **inverse problem**, the values of some model parameter must be obtained from the observed data. The transformation from data to model parameters is the result of the interaction of a physical system, e.g. the Earth, the atmosphere, gravity etc. An inverse problem is to find m such that (at least approximately) $d = G(m)$ where G is an operator describing the explicit relationship between data d and model parameters m, and it is a representation of the physical system. The operator G is called observation function.

There are some issues related with this type of problems. Inverse problems are typically **ill posed**. In fact, inverse problems most often do not fulfill Hadamard's postulates of well-posedness: they might not have a solution in the strict sense, solutions might not be unique and/or might not depend continuously on the data. Hence their mathematical analysis is subtle, but they have many applications in engineering, physics and other fields.

In addition, while inverse problems are often formulated in infinite dimensional spaces, limitations to a finite number of measurements, and the practical consideration of recovering only a finite number of unknown parameters, may lead to problems being recast in discrete form. In this case the inverse problem will typically be **ill-conditioned**. The condition number associated with a problem is a measure of that problem's amenability to digital computation. A problem with a high condition number is said to be ill-conditioned. In an ill-conditioned problem the exact solution is extremely sensitive to small perturbations in the data; this makes iterating the solution to a small residual a tricky operation. Numerical round-off in the system can also present some challenges when it comes to solving a model having an ill-conditioned matrix.

It is easy to understand that any algorithmic solution to the inverse problem has to carefully address all these issues. We are interested in particular to the case in which the relationship is linear (or linearizable) and the equation that relates the data to the parameters reduces to $d = Am$, where A is a matrix. From a mathematical point of view one of the most common approaches to the inverse

problem when a set of possibly noisy observations of the physical system is given, this leading to an overdetermined system, is the linear least square method.

4.1.1 Linear least square

Linear least squares is the problem of approximately solving an overdetermined system of linear equations, where the best approximation is defined as that which minimizes the sum of squared differences between the data values and their corresponding modeled values [123]. The approach is called linear least squares since the solution depends linearly on the data. Linear least squares is a computational approach for fitting a mathematical or statistical model to data. It can be applied when the idealized value provided by the model for each data point is expressed linearly in terms of unknown parameters of the model. The resulting fitted model can be used to summarize the data, to predict unobserved values from the same system, and to understand the internal mechanisms of the system.

Given a system of linear equations

$$Ax = b.$$

in general we cannot always expect to find a vector x which will solve the system; even if there exists such a solution vector, then it may not be unique. We can however always ask for a vector x that brings Ax "as close as possible" to b, i.e. a vector that minimizes the Euclidean norm

$$\|Ax - b\|^2.$$

If there are several such vectors x, we could ask for the one among them with the smallest Euclidean norm. Thus formulated, the problem has a unique solution, given by the pseudoinverse:

$$x = A^+ b.$$

This description suggests the following geometric construction of the pseudoinverse of an $m \times n$ matrix A. To find $A^+ b$ for given b in $\Re^m$, first project b orthogonally onto the range of A, finding a point $p(b)$ in the range. Then form $A^{-1}(p(b))$, i.e. find those vectors in $\Re^n$ that A sends to $p(b)$. This will be an affine subspace of $\Re^n$ parallel to the kernel of A. The element of this subspace that has the smallest length (i.e. is closest to the origin) is the answer $A+b$ we are looking for. It can be found by taking an arbitrary member of $A^{-1}(p(b))$ and projecting it orthogonally onto the orthogonal complement of the kernel of A [124].

Using the pseudoinverse and a matrix norm, one can define a condition number for any matrix:

$$\mathrm{cond}(A) = \|A\|\|A^+\|.$$

A large condition number implies that the problem of finding least-squares solutions to the corresponding system of linear equations is ill-conditioned in the sense that small errors in the entries of A can lead to huge errors in the entries of the solution.

The pseudoinverse is one way to solve linear least squares problems.

4.1.2 Pseudoinverse

In mathematics, and in particular linear algebra, the pseudoinverse A^+ of an $m \times n$ matrix A is a generalization of the inverse matrix. The most commonly encountered pseudoinverse is the Moore-Penrose matrix inverse [75, 86], which is a special case of a general type of pseudoinverse known as a matrix 1-inverse.

A common use of the pseudoinverse is to compute a "best fit" (least squares) solution to a system of linear equations that lacks a unique solution. The pseudoinverse is defined and unique for all matrices whose entries are real or complex numbers.

Definition

The pseudoinverse A^+ of an m-by-n matrix A (whose entries can be real or complex numbers) is defined as the unique n-by-m matrix satisfying all of the following four criteria:

1. $AA^+A = A$ (AA^+ need not be the general identity matrix, but it maps all column vectors of A to themselves);
2. $A^+AA^+ = A^+$ (A^+ is a weak inverse for the multiplicative semigroup);
3. $(AA^+)^* = AA^+$ (AA^+ is Hermitian); and
4. $(A^+A)^* = A^+A$ (A^+A is also Hermitian).

where $M^* =$ is the Hermitian transpose (conjugate transpose) of M. For matrices of real numbers the relation $M^* = M^T$ holds.

4.2 Applications

We stated that the inverse problem is used in many different applications, specially when a model for a physical system is needed, and the use of the matrix pseudoinverse can be applied when the relationship between data and parameters is linear. The pseudoinverse in fact provides a least squares solution to a system of linear equations. The pseudoinverse, however, is used in many other fields. In general its utilization spans from signal processing to pattern recognition, from geophysics to astronomy, from image processing to robotics. It is in these last two research areas that the pseudoinverse keeps maintaining a strong importance and in the following we present a couple of specific examples.

4.2.1 Image processing

There are several example of the use of matrix pseudoinversion in the field of image processing. For instance it is used in image restoration and face recognition. However the most interesting and challenging utilization is in the field of image reconstruction, in particular in medical imaging. Medical imaging is the technique and process used to create images of the human body (or parts and function thereof) for clinical purposes or medical science. The possible applications span

from medical procedures seeking to reveal, diagnose or examine disease to the study of normal anatomy and physiology. Since medicine and surgery are possible target for perception based teleoperation, and computer aided surgery is one of our main motivation for this study, we briefly present an example of the use of linear least square, thus matrix pseudoinversion, in Diffusion Tensor Imaging.

The Diffusion Tensor Model

Diffusion MRI is a magnetic resonance imaging (MRI) method that produces in vivo images of biological tissues weighted with the local microstructural characteristics of water diffusion. Since 1965 it has been recognized that the diffusion properties of water in structured samples measured with NMR are more adequately described by a tensor than a scalar value. However, it was not until 1992 that Basser and colleagues showed how such a tensor can be estimated from a series of diffusion weighted signals using linear regression (See example in Figure 4.1). A key component in the estimation of the tensor is the calculation of the coupling between the signal attenuation and the elements of the diffusion tensor for a given gradient amplitude, duration, and separation. As the tensor is symmetric, there are only six unknown elements to determine. These are estimated from a series of diffusion-weighted images acquired with gradients applied in non-collinear and non-coplanar directions. The idea that to find n unknown variables in linear algebra, at least n simultaneous equations should be solved is a concept familiar to all, and the same applies when estimating the diffusion tensor from MR data. Thus, the minimum number of diffusion-encoding images required for estimating the elements of the tensor is six.

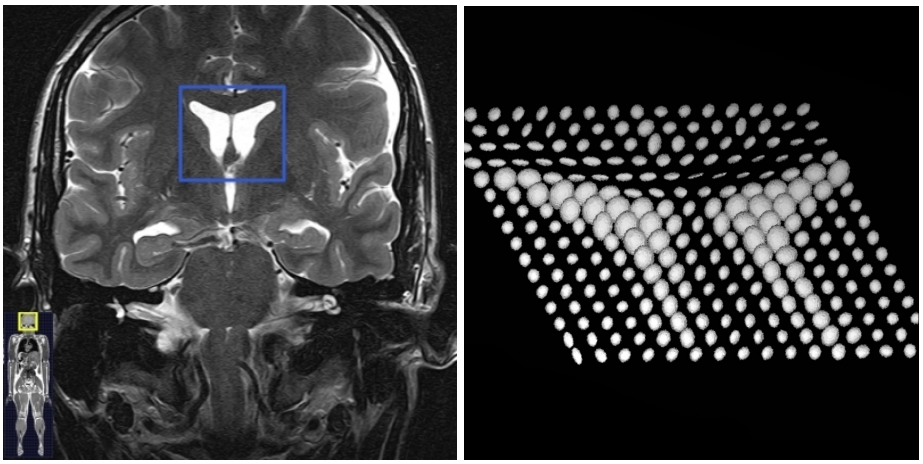


Fig. 4.1. Detail of the NMR image of the brain: on the left the structure as appear in the medical scan, on the right the representation of the tensors with ellipsoids, whose orientation reflects the neural fibers directions [51]

If X is a vector of the signal intensities, B is a matrix with the number of rows equal to the number of measurements of the signal, and six columns, and D is a

vector containing the elements of the diffusion tensor then we can summarize the relationship between the set of observed signals and the elements of the diffusion tensor via the expression: $X = BD$. The simplest estimation approach is simply to solve the equation by taking the inverse of B, i.e. $D = B^{(-1)}X$. This approach is fine if we have exactly six measurements such that the matrix B is square. However, the number of data we fit to the model is exactly the same as the number of fit parameters, so we will fit the data "exactly", and that includes the perturbations due to noise. It is usual, therefore, to acquire more than the bare minimum number of measurements in order to improve the signal to noise ratio. However this results in the matrix B no longer being square. In particular the matrix is rectangular with more equations n, than unknowns k. Thus we are in the first case of the list we presented above: if $k < n$, then the system is overdetermined and there is no solution. More precisely, there is no "exact" solution. In fact we do not expect that an exact solution exists, since the multiple measurements have the aim to lower the bad contribution due to noise. What we need is an approximation that best fit data collected. In this case, the tensor is found by computing the pseudo-inverse of the matrix B, i.e., $D = (B^T B)^{(-1)} B^T X$. This ordinary least squares approach, is extremely rapid and allows for vectorization of the estimation and is therefore prevalent in many popular software packages.

4.2.2 Robotics: the Jacobian matrix

The Jacobian matrix is the matrix of all first-order partial derivatives of a vector-valued function. Its importance lies in the fact that it represents the best linear approximation to a differentiable function near a given point.

$$\begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_n}{\partial x_1} & \cdots & \frac{\partial y_n}{\partial x_n} \end{bmatrix}$$

The Jacobian is one of the most important quantities in the analysis and control of robot motion. It arises in virtually every aspect of robotic manipulation: in the planning and execution of smooth trajectories, in the determination of singular configurations, in the execution of coordinated anthropomorphic motion, in the derivation of the dynamic equations of motion [104].

Moreover, interaction of the manipulator with the environment produces forces and moments at the end effector or tool. These, in turn, produce torques at the joints of the robot. The manipulator Jacobian has an important role in the quantitative relationship between the end effector forces and joint torques. This relationship is important for the development of path planning methods, the derivation of the dynamic equations and in the design of force control algorithms.

In robotics the Jacobian is a $6 \times n$ matrix, with n the number of joints. For a six joints manipulator, such as an antropomorfic arm, the Jacobian is a square matrix. Configurations for which rank $J(q)$ is less than its maximum value are called singularities or singular configurations.

- Singularities represent configurations from which certain directions of motion may be unattainable.

- At singularities, bounded end-effector velocities may correspond to unbounded joint velocities.
- At singularities, bounded joint torques may correspond to unbounded end-effector forces and torques.
- Singularities often correspond to point on the boundary of the manipulator workspace, that is, to points of maximum or minimum reach of the manipulator.
- Singularities correspond to points in the manipulator workspace that may be unreachable under small perturbations of the link parameters, such as length, offset, etc.

When the Jacobian is not full rank it is not invertible. Thus determining the rank of the Jacobian is crucial for the control of the robot. Unfortunately the rank cannot be always determined exactly because it can be easily altered by an arbitrary small perturbation. Using the terminology presented above the Jacobian is ill-conditioned. This happens in particular when the Jacobian is not full column or row rank, but also a full rank matrix can be affected by this problem.

There is another situation in which the inverse of the Jacobian is not defined, even if the robot is not in a singular configuration: when the manipulator has not six joints. In particular in the case of redundant robots the matrix is rectangular, and this happens independently from the current configuration of the robot.

Summarizing the robotic Jacobian matrix is:

- usually ill-conditioned
- not invertible at singular configuration (depending on the task) or in the case of redundant robots.

These conditions can be problematic in the presence of an inverse problem.

Kinematics

The kinematic description of a robotic manipulator is the relationship that relates velocity, position and orientation of the end effector to the joint variables of the robot. The relationship that determines the joint variables given the end effector information is called forward kinematics, the viceversa is known as inverse kinematics.

The Jacobian relationship

$$\xi = J\dot{q} \tag{4.1}$$

specifies the end-effector velocity that will result when the joints move with velocity $\dot{q}$. The inverse velocity problem is the problem of finding the joint velocities $\dot{q}$ that produce the desired end-effector velocity. When the Jacobian is square and nonsingular, this problem can be solved by simply inverting the Jacobian matrix to give

$$\dot{q} = J^{-1}\xi$$

For manipulators that do not have exactly six joints, the Jacobian cannot be inverted. In this case there will be a solution to Equation 4.1 if and only if ξ lies in

the range space of the Jacobian. This can be determined by the following simple rank test. A vector ξ belongs to the range of J if and only if

$$\text{rank}J(q) = \text{rank}[J(q)|\xi]$$

in other words, Equation 4.1 may be solved for $\dot{q} \in \mathbb{R}^n$ provided that the rank of the augmented matrix $[J(q)|\xi]$ is the same as the rank of the Jacobian $J(q)$.

For the case when $n > 6$ we can solve for $\dot{q}$ using the right pseudoinverse of J. To construct this pseudoinverse, we use the fact that when $J \in \mathbb{R}^{m\times n}$, in $m < n$ and rank $J = m$, then $(JJ^T)^{-1}$ exists. In this case $(JJ^T) \in \mathbb{R}^{m\times n}$, and has rank m. Using this result, we can regroup terms to obtain

$$\begin{aligned} I &= (JJ^T)(JJ^T)^{-1} \\ &= J[J^T(JJ^T)^{-1}] \\ &= JJ^+ \end{aligned}$$

Here, $J^+ = J^T(JJ^T)^{-1}$ is called a right pseudoinverse of J, since $JJ^+ = I$.

A solution to Equation 4.1 is given by

$$\dot{q} = J^+\xi + (I - J^+J)b \tag{4.2}$$

in which $b \in \mathbb{R}^n$ is an arbitrary vector.

Equation 4.2 indicates that the resultant joint velocities can be decomposed into a combination of the least squares solution of minimum norm, plus a homogeneous solution created by the action of a projection operator $(I - J^+J)b$, which describes the redundancy of the manipulator system. If the Jacobian is a square and nonsingular matrix then the projector operator is equal to the null operator and Equation 4.2 reduces to

$$\dot{q} = J^{-1}\xi$$

where J^{-1} is the 6×6 inverse Jacobian matrix.

In general, for $m < n$, $(I - J^+J) \neq 0$, and all vectors of the form $(I - J^+J)b$ lie in the null space of J. This means that, if $\dot{q}'$ is a joint velocity vector such that $\dot{q}' = (I - J^+J)b$, then when the joints move with velocity $\dot{q}'$, the end effector will remain fixed since $J\dot{q}' = 0$. Thus, if $\dot{q}$ is a solution to Equation 4.1, then so is $\dot{q}+\dot{q}'$ with $\dot{q}' = (I - J^+J)b$, for any value of b. If the goal is to minimize the resulting joint velocities, we chose $b = 0$ [104].

Dynamics

The dynamic model of a manipulator is the set of relationships that relates the motion of the robot to the actuation torques of the joints and, possibly, to the forces acting on the end effector. The dynamic equations can be used for simulating the motion of the manipulator or for controlling the real robot. The **forward dynamics** determines accelerations, velocities and positions of the joints when the joints torques and the forces exerted to the end effector are given. The **inverse dynamics** determines the joint torques that are needed in order to obtain a specific motion, in terms of position, velocities and accelerations, of the robot, given the

forces exerted to the end effector. Since we just want to provide another possible utilization of the pseudoinverse of the Jacobian matrix, a precise definition and explanation of the manipulator dynamics is out of the scope of this discussion.

For our purpose it is interesting to note that the dynamic equations can be expressed with respect to joint space or Cartesian space. The latter is used in particular when multiple robots have to be controlled. For instance, this can be the case when the operator uses an haptic device with each hand and this is a common setup in robotic aided surgery.

The dynamic model of a manipulator expressed in Cartesian space is:

$$B_A(x)\ddot{x} + C_A(x, \dot{x} + g_A(x) = \gamma_A - h_A \tag{4.3}$$

with

$$\begin{aligned} B_A &= J_A^{-T} B J_A^{-1} \\ C_A\dot{x} &= J_A^{-T} C\dot{q} - B_A \dot{J}_A \dot{q} \\ g_A &= J_A^{-T} g \end{aligned}$$

where J_A is the Jacobian, B is the mass matrix of the manipulator, C is a vector of centrifugal and Coriolis terms, g is a vector of gravity terms, h is the vector of the interaction forces between the end effector and the environment, and γ_A is the contribution to the end effector's forces due to joint torques. J_A^{-1} and J_A^{-T} are the inverse and the inverse transpose of the Jacobian respectively.

In [22] an example of the application of this approach to control is presented. In the paper the authors use two robots, one with the original industrial controller, the second with a connection between the controller and the PC, in order to afford the computation requested by this formulation.

However, since B_A exists only if the matrix J_A is full rank and hence invertible, Equation 4.3 can be applied only to non redundant manipulators that are not in singularity. The modifications to Equation 4.3 needed in order to consider also these cases lead, after several simplifications omitted for shortness to the following equation:

$$\tau = J_A^{T+}\gamma_A + (I - J_A^{T+}\overline{J}_A^T)\tau_a \tag{4.4}$$

where J_A^{T+} is the right pseudoinverse of $\overline{J}_A^T$.

Equation 4.4 indicates that the resultant torques can be decomposed into a combination of the least squares solution of minimum norm, plus a homogeneous solution created by the action of a projection operator, which describes the redundancy of the manipulator system. The choice of the value of τ_A defines the strategy for the handling of the internal motion of the joints, due to redundancy. It is easy to notice that this is the very same approach we found in Equation 4.2 for the kinematic problem.

These two examples illustrate that one of the most important quantities in robotics, the Jacobian matrix, requires efficient and deterministic pseudoinverse computation to be effective. Moreover the needs for the matrix pseudoinverse is particularly important in those cases, such as at singular configurations and for a good management of the redundancy, that are crucial in a teleoperated task.

4.3 Finding the pseudoinverse of a matrix

We have seen that the matrix pseudoinversion is an important instrument in the solution of the inverse problem and that its application is valuable in several fields, and in particular in robotics.

From its formalization, in the 50s many scientific articles about mathematical properties, relationships and special forms of the Moore-Penrose pseudoinverse have been proposed. At the same time there has been a specific research aimed at the algorithmic and computational approach to matrix pseudoinversion. Many efforts were spent in finding solutions for the specific application, exploiting the characteristics of the given matrix, but when considering more general approaches the most used method are:

1. The QR method
2. Iterative methods
3. Updating the pseudoinverse
4. The Singular Value Decompostion method

The first two solutions are more sensitive to ill-conditioning, thus let us consider the last two choices.

Updating the pseudoinverse

For the cases where A has full row or column rank, and the inverse of the correlation matrix (AA^* for A with full row rank or A^*A for full column rank) is already known, the pseudoinverse for matrices related to A can be computed by applying the Sherman-Morrison-Woodbury formula to update the inverse of the correlation matrix, which may need less work. In particular, if the related matrix differs from the original one by only a changed, added or deleted row or column, incremental algorithms exist that exploit the relationship. Similarly, it is possible to update the Cholesky factor when a row or column is added, without creating the inverse of the correlation matrix explicitly. However, updating the pseudoinverse in the general rank-deficient case is much more complicated.

Singular Value Decomposition

A computationally simpler and more accurate way to get the pseudoinverse is by using the Singular Value Decomposition (SVD). SVD is a factorization of a rectangular real or complex matrix.

Suppose A is an m-by-n matrix whose entries come from the field K, which is either the field of real numbers or complex numbers. Then there exists a factorization of the form

$$A = U\Sigma V^*,$$

where:
U is an m-by-m unitary matrix over K

Σ is m-by-n diagonal matrix with non negative numbers on the diagonal
V^* denotes the conjugate transpose of V, an n-by-n unitary matrix over K.

The SVD is typically computed by a two-step procedure: a bidiagonal reduction is computed first and than the resulting matrix is decomposed into SVD by applying an iterative method. The stop condition of the iterative procedure is usually the satisfaction of a certain precision. In many cases the precision required is the machine epsilon. Many variations and computation improvements have been proposed, resulting in very optimized software libraries and specific algorithmic procedures. In particular, implementations based on Givens and Jacobi rotations are used in order to make the process more parallel, and thus exploiting new architectures and improving performance.

The pseudoinverse of the matrix A with singular value decomposition $M = U\Sigma V^*$ is

$$A^+ = V\Sigma^+ U^*,$$

For a diagonal matrix such as Σ, we get the pseudoinverse by first transposing the matrix, and then taking the reciprocal of each non-zero element on the diagonal, and leaving the zeros in place. In numerical computation, only elements larger than some small tolerance are taken to be nonzero, and the others are replaced by zeros. Usually the tolerance is taken to be $t = \epsilon \max(m, n) \max(\Sigma)$, where ϵ is the machine epsilon. In presence of ill-conditioned matrix the tolerance can have bad influence on the correctness of the result.

From a performance point of view, the cost of computing the pseudinverse is largely dominated by the calculation of the SVD decomposition, when an optimized implementation of matrix-matrix multiplication such as LAPACK is used.

The above procedure shows why taking the pseudoinverse is not a continuous operation: if the original matrix A has a singular value 0 (a diagonal entry of the matrix Σ above), then modifying A slightly may turn this zero into a tiny positive number, thereby affecting the pseudoinverse dramatically as we now have to take the reciprocal of a tiny number. Special care has to be taken when the matrix is ill-conditioned.

A combination of this method and the precedent one, applied to the robotic field, has been proposed in [14].

Thanks to the application in other fields, such as statistics and regularization methods, and to the availability of good software libraries, the SVD is certainly the most known and used method for matrix pseudoinversion. Among its esteems there is in particular the fast execution time. The SVD approach has, however, some drawbacks and one in particular is crucial when dealing with robotic applications: SVD factorization is based mainly on iterative methods, and there is no guarantee that the procedure terminates in a given period. Of course the convergence speed depends on the present values of the matrix, and although on average the computation results fast with most of the configurations, slow convergence or divergence are also possible. This is an important issue in real-time systems where determinism and ability of meet deadlines is crucial, and teleoperation, haptics and robotics rely on these requirements.

4.4 Decell algorithm

The most used algorithms for the matrix pseudoinverse computation can present convergence issues and, in general, are difficult to parallelize. An alternative solution is to look for some less known algorithm, as we suggested in Section 3.1.3. Among others [11], in our research we found an interesting approach that dates back to the 60s: the Decell algorithm.

The work of Decell starts with the definition of the following theorem:

Theorem 4.1. *Let A be any n-by-m complex matrix and let*

$$f(\lambda) = (-1)^n(a_0\lambda^n + a_1\lambda^{n-1} + \cdots + a_k\lambda^{n-k} + \cdots + a_{n-1}\lambda + a_n)$$

with $a_0 = 1$ be the characteristic polynomial of AA^*. If $k \neq 0$ is the largest integer such that $a_k \neq 0$, then the generalized inverse of A is given by

$$A^\dagger = -a_k^{-1}A^*[(AA^*)^{k-1} + a_1(AA^*)^{k-2} + \cdots + a_{k-1}I].$$

If $k = 0$ is the largest integer such that $a_k \neq 0$, then $A^\dagger = 0$.

The proof uses the Cayley-Hamilton theorem [30, 31].

On the base of the mathematical result, Decell developed an algorithm for the generation of the pseudoinverse A^+, given a $p \times n$ matrix A with $n \geq p$: the algorithm consists in computing a sequence of matrices $A_0, A_1, \ldots, A_k$ as follows:

$$\begin{array}{lll}
A_0 = Z, & -1 = q_0, & B_0 = I; \\
A_1 = AA^*, & traceA_1 = q_1, & B_1 = A_1 - q_1 I; \\
A_2 = AA^*B_1, & \dfrac{traceA_2}{2} = q_2, & B_2 = A_2 - q_2 I; \\
\vdots & \vdots & \vdots \\
A_{k-1} = AA^*B_{k-2}, & \dfrac{traceA_{k-1}}{k-1} = q_{k-1}, & B_{k-1} = A_{k-1} - q_{k-1} I; \\
A_k = AA^*B_{k-1}, & \dfrac{traceA_k}{k} = q_k, & B_k = A_k - q_k I.
\end{array} \tag{4.5}$$

where $1 \leq i \leq k$, I is a $p \times p$ identity matrix and k is the rank of A and the * operation is the transposition for matrices of real numbers. Decell points out that $q_i = -a_i$ in Theorem 4.1.

Hence the pseudoinverse is given by

$$A^+ = -a_k^{-1}A^*B_{k-1} \tag{4.6}$$

Since the rank of A can be not known a priori, the iteration is continued until the matrix product A_1B_i, where $A_1 = AA^*$, becomes a zero matrix for some $i, i = 1, 2, \ldots, k$. The termination of the iteration will determine k as well. In any

case it is proven that the procedure requires exactly $(K + 1)$ iterations. This is a perfect approach to our problem. Let us consider in fact the Jacobian matrix of a redundant manipulator or a haptic device. The rank of the matrix is the minimum between 6 and the number of joints, at non singular configurations. During the motion of the robot, when a singularity is encountered the rank is lowered under the previous value. Thus, given a kinematic structure the time spent in the computation of the pseudoinverse of the Jacobian is at most the time spent in $\min(6, dof)$ iterations, plus the computation of Equation4.6, that can be carried out in fixed time. If the real-time deadline for the controller is correctly chosen or, the other way round given a specific controller rate, if the implementation is able to fulfill the computation of the "worst case" correctly, the algorithm is reliable.

The main drawback of this approach is that in order to determine the rank of the matrix, and hence in order to stop the iteration, the product $A_1 B_i$ has to be exactly zero. Unfortunately in presence of roundoff errors some elements of the products may be small but not equal to zero. The algorithm may suffer from numerical instability and this combines badly with the ill-conditioning of the Jacobian matrix.

In order to use the Decell algorithm while performing the computation of the exact solution of the pseudinversion, the application of some error free techniques is needed. In signal processes and filter design, the residue number system (RNS) is the most popular solution to the numerical instability problem [103].

4.4.1 Residue Number System

A residue number system (RNS) is used to represent a large integer using a set of smaller integers, so that the computation may be performed more efficiently. RNS have applications in the field of digital computer arithmetic. By decomposing a large integer into a set of smaller integers, a large calculation can be performed as a series of smaller calculations that can be performed independently and in parallel. This makes RNS particularly popular in hardware implementations. Moreover, since in each small computation the arithmetic range is reduced, the calculation has an higher numerical stability.

Definition 4.2. *Given any integer x and any modulus m, if $r \equiv x \mod m$ and $0 \leq r < m$, then we write $r = |x|_m$ and say r is a residue of x modulo m [23].*

This defines a single-modulus residue arithmetic that involves only non-negative integers. The definition of signed integers is possible through the *Symmetric Residues Modulo m*.

Definition 4.3. *Given any integer x and any modulus m, if $r \equiv x(\mod m)$ and*

$$-\frac{1}{2}m \leq r \leq \frac{1}{2}m$$

then we write $r = /x/_m$ and say r is a symmetric residue of x modulo m.

In order to perform all the computation needed by our algorithm it is necessary to define the multiplicative inverse as follows [108]:

Definition 4.4. *Assume x, y, and $m > 1$ are integers, and if $0 < y < m$ and $|xy|_m = |yx|_m = 1$, then we write $y = x^{-1}(m)$ and say y is a multiplicative inverse of x modulo m.*

The uniqueness of the multiplicative inverse is granted if m is a prime number. It is also possible to perform integer division to a limited extent [56].
These last properties are also suitable for symmetric residue representation.

A given integer number can be expressed with respect to several different moduli at once, leading to the definition of the concept of multiple-modulus arithmetic:

Definition 4.5. *Let $m_1, m_2, \ldots, m_L$ be the base for a residue number system, where $gcd(m_i, m_j) = 1$ for $i \neq j$, and $M = m_1 m_2 \ldots m_L = \prod_{i=1}^{L} m_i$.*

The unique L-tuple residue arithmetic representation of x is given by $x \sim \{|x|_{m_1}, |x|_{m_2}, \ldots, |x|_{m_L}\}$ where $r_i = |x|_{mi}$.
The representation also holds true for the symmetric residue system.

Arithmetic representation can be decomposed linearly into an equivalent one containing many components, each with a relatively small arithmetic range [61, 108].
This decomposition leads to a number of parallel and independent pseudoinverse computations, each of them applying the residue arithmetic which allows the use of short registers and simple arithmetic operations. Furthermore, since the decomposition is linear, the final pseudoinverse results for each parallel computation can be linearly recombined back together, yielding the desired pseudoinverse result (see Figure 4.2). To achieve this objective the multiple-modulus symmetric residue arithmetic defined above can be used. The use of RSN for the correct computation of matrix pseudoinverse was analyzed in [90, 106]. One important issue regarding the multiple-modulus representation of the numbers is due to the correct choice of the range $M = m_1 m_2 \ldots m_L$ and the selection of the bases or moduli m_i, $1 \leq i \leq L$, which are critical to the success of the method. These problems have been extensively studied and there are rules, based on the arithmetic range of the integer to be represented in moduli, that can be used [106].

The recombination phase can be implemented using different algorithms but mainly the Chinese Reminder Theorem and the Mixed Radix Recomposition procedure.

Summarizing, the advantages in the use of the residue number system are the ability to perform an error free computation even in presence of a numerical unstable algorithm, and the possibility to use simpler operation and, from a hardware point of view, short registers.

The main drawback of this approach is the requirements for a RNS arithmetic. Usually the libraries of mathematic functions available on modern PC do not provide native support for operations in moduli. In order to implement the Decell algorithm exploiting RSN and arithmetic decomposition the missed arithmetic has to be implemented from scratch. However it is worth noticing that RNS is used in several applications, from signal processing to cryptography although, for computation reasons, it was used mainly in hardware. With the introduction of

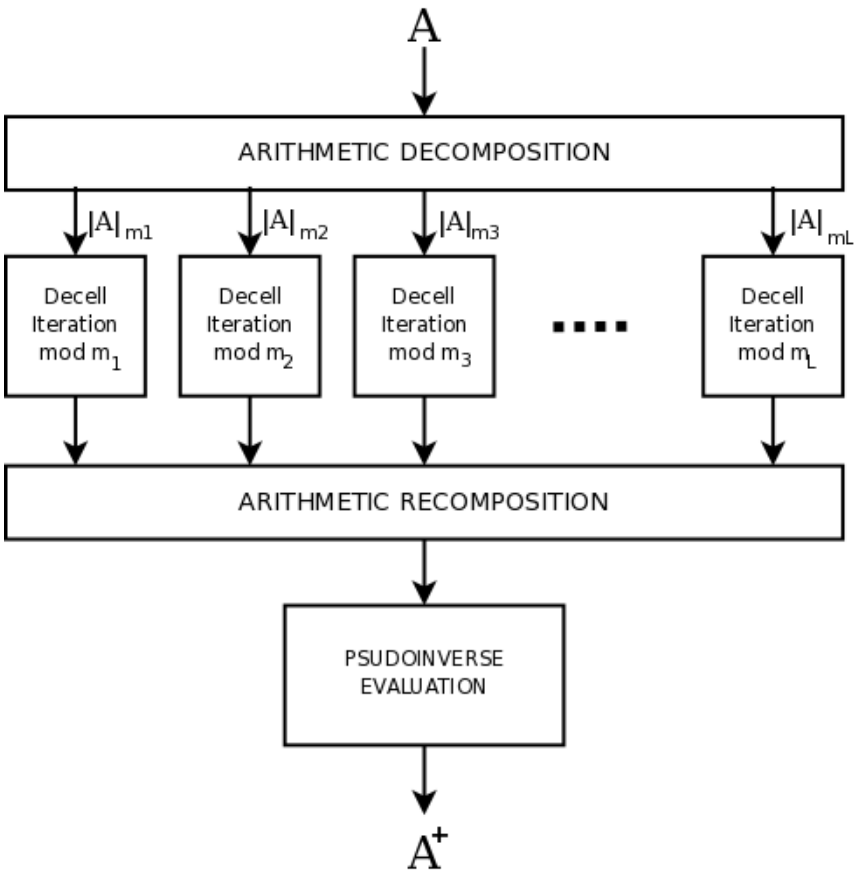


Fig. 4.2. Parallel Decell schema

new architecture capable of impressive performance, it is predictable that new software libraries will be available soon.

This implicit requirements of the RSN implementation made the Decell algorithm less attractive for traditional architecture, except for ASIC realizations. In fact in SISD processors the computation of the pseudoinversion represented in each base had to be performed sequentially and the advantage of short register operation had not a great impact in general purpose uniprocessor machines. In alternative, the utilization of high precision floating point operations was not a complete guarantee of error free computation and did not provide fast performance anyway. Considering all these problems it was natural to opt for a more efficient method, such as SVD, for determining the pseudoinverse matrix in those application in which determinism and real-time were not an issue. Meanwhile very optimized and suboptimal solutions where considered for robotic control and applications. With new multicore architectures and vector processors an efficient implementation of Decell algorithm is possible anew, and the use of short vec-

tor operation permits to use SIMD register to exploit the parallelism due to the presence of many matrix multiplications.

In [23] the authors presented an implementation in VLSI of the Decell algorithm in the residue number system. Our idea is that it is possible to implement the same algorithm in a modern multicore processor carefully designing the implementation and exploiting both task and data parallelism available. In order to obtain good performance without very specific target oriented fine tuning, we suggest to mimic the implementation in VLSI of the operations needed using the intrinsic functions of the programming language that are generic enough to be available for other platforms too.

4.5 Parallel hardware architectures

In the last Section we identified an algorithm that is useful for our robotic application and that seems to be suitable for parallel implementation. Now, in order to implement it we need to choose a hardware architecture. We would like to stress that we are not taking this decision on the base of the algorithm we want to implement. In fact, as stated previously, it is important to separate the programming model from the processing model, and think of the abstract organization of the program apart form the target machine. For this reason in the next Chapter we will analyze the parallelism present in the algorithm in order to understand how to exploit it. What we are describing here is the choice of an architecture that, in our opinion, can be useful in general for robotic applications.

In Chapter 3 we presented a classification of parallel architectures and we provided some examples of actual hardware realization. In particular we identified three platform that are the state of the practice of multicore systems and that represent a further step into the diffusion of parallel architecture thanks to their large availability and relative low price:

- Field Programmable Gate Array (FPGA)
- GPU (or GPGPU)
- Cell Broadband Engine

(a) FPGA (b) GPU (c) CBE

Fig. 4.3. Multicore architectures

These architectures share some parallelization strategies while thy are very different under other aspect.

4.5.1 FPGA

A field-programmable gate array (FPGA) is an integrated circuit designed to be configured after manufacturing. The FPGA configuration is generally specified using a hardware description language (HDL), similar to that used for an application-specific integrated circuit (ASIC). FPGAs can be used to implement any logical function that an ASIC could perform. FPGAs contain programmable logic components called "logic blocks", and a hierarchy of reconfigurable interconnects that allow the blocks to be wired together, somewhat like a one-chip programmable breadboard. Logic blocks can be configured to perform complex combinational functions, or merely simple logic gates like AND and XOR. In most FPGAs, the logic blocks also include memory elements, which may be simple flip-flops or more complete blocks of memory.

In Chapter 2.2.1 we explained that we used an innovative hardware/software structure to control an actuated 6 dof joystick. An FPGA was used to handle both low level tasks and algorithmic part that included forward kinematics and force feedback computations. Although we obtained very good results in terms of performance, we encountered some problems during the development of the system. The most critical issue was about the resources available on the board. Our reconfigurable board was based on a Virtex II FPGA with one million gates. That was a considerable number of logic components, however there were a limited number of specific blocks. For instance the 18 bit multipliers were only forty, thus it was difficult to fully exploit the parallelism present in the algorithm. The FPGA required a specific graphical programming language, LabVIEW, that clearly represents parallelism and data flow, without low-level hardware description languages or board-level design. It allows non skilled programmer to easily configure the FPGA but, on the other side, it hides many structures and prevent a low level access to the hardware. It permits to import VHDL code and IP cores but with limitations. One main problem was that LabVIEW for FPGA was not available under GNU/Linux platform. This not only prevent the development of the code on GNU/Linux machines but also limited the interaction with them, since all the communication among the FPGA and the host via PCI was controlled runtime by LabVIEW. Since our setup was based on RTAI and GNU/Linux it was necessary to use digital I/O ports in order to provide the board with parallel interface, at the cost of an high usage of logic blocks. As a result it was not possible to run the code for the forward kinematic and the code for the force feedback to once. In addition we needed to optimize a lot the algorithm making it customized for the specific haptic hardware.

Considering this past experience we think that FPGAs, that are growing in terms of resources, can give better results when combined with microcontrollers, memory and I/O modules on System on Chip board (SoC). There are not many SoC products available on the market at affordable price, thus it is still hard to find a good solution. In addition the programming languages used to program FPGA are still quite far from standard high level languages even if some interesting project

such as SystemC exists. In general it is difficult to compare FPGAs programming to general software development for PCs, thus the implementation of FPGA code still requires specific skills.

4.5.2 GPU or GPGPU

General-purpose computing on graphics processing units (GPGPU, also referred to as GPGP) is the technique of using a GPU, typically specialized in computer graphics elaboration, to perform computations for applications traditionally handled by the CPU. It is made possible by the addition of programmable stages and higher precision arithmetic to the rendering pipelines, which allows software developers to use stream processing on non-graphics data.

This type of parallel processor is used in several application fields and in particular in physical simulations and interactive photorealistic rendering. The wide availability of the boards, that nowadays are present in many home computers, their low cost and the push to development given by the entertainment industry, in particular for videogames, have largely contributed to the success of this multicore architecture. The results obtained in the early implementations have pushed towards the use of this board for many scientific applications, with stunning results in terms of speed.

In addition, several high-performance libraries have been implemented for GPGPU programming. The CUDA system from NVIDIA, for instance, is a high-level language based on an extension of C in which certain functions are identified using a special syntax as stream functions. Application of these functions to arrays then invokes parallel computations on an NVIDIA GPU. CUDA is based on an SPMD stream processing model but includes extensions that allow the specification of thread blocks that in practice will execute together on a single processor element using SIMD masking.

The main robotic applications are related to simulation, since the GPGPU can generate good physics simulations that combine well with the more classical three-dimensional graphics part, providing a complete virtual experience.

The main drawback of the use of GPGPU is due to the memory and cache management. For instance, in GPUs caches are supported, but there are separate read and write caches, and no guarantees are made that data written will be available immediately in a read from the same location unless specific steps are taken to flush the data from the write cache and invalidate the read cache. Generally, on a GPU, it is best to separate the computation into discrete passes and separate input and output memory locations for each pass. The need for synchronization and guards leads to a not really predictable execution timing. This is almos unnoticeable in 3D applications, due to the relatively low refresh rate, but can be a problem when real-time constraints and 1 KHz control loop are required.

However GPGPUs are one of the most promising parallel hardware architecture on the market.

4.5.3 IBM CBEA

The Cell Broadband Engine (CBE) processor is the first implementation of a new multiprocessor family conforming to the Cell Broadband Engine Architec-

ture (CBEA). The CBEA is a new architecture that extends the 64-bit PowerPC Architecture^{TM}. The CBEA and the CBE processor are the result of a collaboration between Sony, Toshiba, and IBM known as STI, formally begun in early 2001. Cell combines a general-purpose Power Architecture core with streamlined coprocessing elements which greatly accelerates multimedia and vector processing applications, as well as many other forms of dedicated computation.

The first implementation, the CBE processor, is a single-chip multiprocessor with nine processor elements operating on a shared, coherent memory, as shown in a high-level block diagram in Figure 4.4. In this respect, the CBE processor extends current trends in PC and server processors. The most distinguishing feature of the CBE processor is that, although all processor elements share memory, their function is specialized into two types:

- the Power Processor Element (PPE)
- the Synergistic Processor Element (SPE)

The CBE processor has one PPE and eight SPEs.

This configuration permits to classify the Cell processor as a heterogeneous multicore machine.

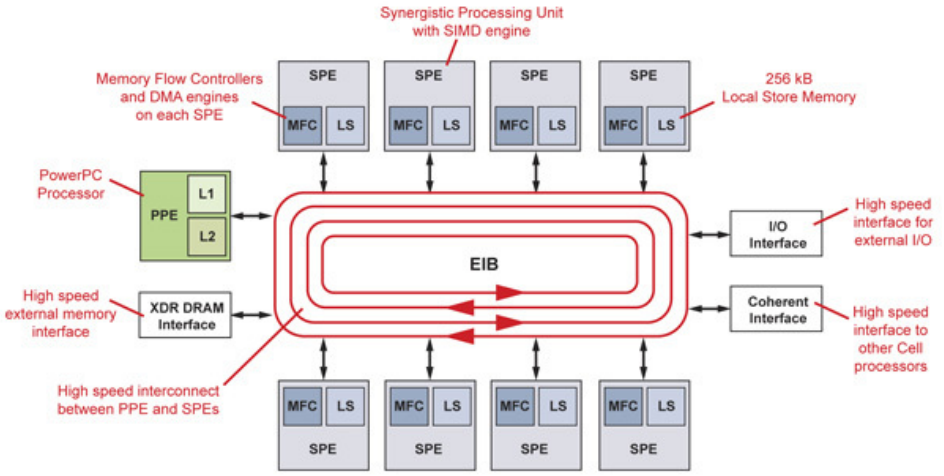


Fig. 4.4. Diagram of the Cell Broadband Engine Processor. It shows the connection topology of the cores and the location of different types of memory

The first type of processor element, the PPE, contains a 64-bit PowerPC Architecture core. It complies with the 64-bit PowerPC Architecture and can run 32-bit and 64-bit operating systems and applications. The second type of processor element, the SPE, is optimized for running computation-intensive SIMD applications; it is not optimized for running an operating system. The SPEs are independent processor elements, each running their own individual application programs or threads. Each SPE has full access to coherent shared memory, including the memory-mapped I/O space. There is a mutual dependence between the PPE

and the SPEs. The SPEs depend on the PPE to run the operating system, and, in many cases, the top-level thread control for an application. The PPE depends on the SPEs to provide the bulk of the application performance.

The SPEs are designed to be programmed in high-level languages, such as (but certainly not limited to) C/C++. They support a rich instruction set that includes extensive SIMD functionality. However, like conventional processors with SIMD extensions, use of SIMD data types is preferred, not mandatory. For programming convenience, the PPE also supports the standard PowerPC Architecture instructions and the vector/SIMD multimedia extensions.

To an application programmer, the CBE processor looks like a 9-way coherent multiprocessor. The PPE is more adept than the SPEs at control-intensive tasks and quicker at task switching. The SPEs are more adept at compute-intensive tasks and slower than the PPE at task switching. However, either processor element is capable of both types of functions. This specialization is a significant factor accounting for the order-of-magnitude improvement in peak computational performance and chip-area-and-power efficiency that the CBE processor achieves over conventional PC processors.

The more significant difference between the SPE and PPE lies in how they access memory. The PPE accesses main storage (the effective-address space) with load and store instructions that move data between main storage and a private register file, the contents of which may be cached. The SPEs, in contrast, access main storage with direct memory access (DMA) commands that move data and instructions between main storage and a private local memory, called a local store or local storage (LS) that has no associated cache. This 3-level organization of storage (register file, LS, main storage), with asynchronous DMA transfers between LS and main storage, is a radical break from conventional architecture and programming models, because it explicitly parallelizes computation with the transfers of data and instructions that feed computation and store the results of computation in main storage. In addition, since in SPE computation there are no cache misses, automatic memory paging and task preemption, if the programmer carefully handles LS the program execution is very predictable and hence "almost" real-time.

The element interconnect bus (EIB) is the communication path for commands and data between all processor elements on the CBE processor and the on-chip controllers for memory and I/O. The EIB supports full memory-coherent and symmetric multiprocessor (SMP) operations. Thus, a CBE processor is designed to be grouped with other CBE processors to produce a cluster.

By distinguishing and separately optimizing control-plane and data-plane processor elements, the CBE processor mitigates the problems posed by power, memory, and frequency limitations. The net result is a multiprocessor that, at the power budget of a conventional PC processor, can provide approximately ten-fold the peak performance of a conventional processor. Of course, actual application performance varies. Some applications may benefit little from the SPEs, whereas others show a performance increase well in excess of ten-fold. In general, compute-intensive applications that use 32-bit or smaller data formats (such as single-precision floating-point and integer) are excellent candidates for the CBE processor.

It is common to run a main program on the PPE that allocates threads to the SPEs. In such an application, the main thread is said to spawn one or more CBE tasks. A CBE task has one or more main threads associated with it, along with some number of SPE threads. An SPE thread is a thread that is spawned to run on an available SPE. The software threads are unrelated to the hardware multithreading capability of the PPE.

A main thread can interact directly with an SPE thread through the SPE's LS. It can interact indirectly through the main-storage space. A thread can poll or sleep, waiting for SPE threads. The operating system defines the mechanism and policy for selecting an available SPE. It must prioritize among all the CBE applications in the system, and it must schedule SPE execution independently from regular main threads. The operating system is also responsible for runtime loading, passing parameters to SPE programs, notification of SPE events and errors, and debugger support.

CBE combines MIMD and SIMD capabilities and thus permits to exploit task and data parallelism. As stated before the SPEs computation is very fast and reliable. In addition the Cell can be used as a very powerful co-processors, since it can be added as an expansion board to standard PC. Thus a combination of a CPU for the operating system, a Cell processor for the most computationally intense algorithms and GPU for simulations and graphics are a possible scenario of the future generation of robotic and haptic controllers. In this sense, and considering its heterogeneous nature, the Cell processor represents a good testbench for parallel algorithms. In another application area, the videogame console Sony Playstation 3 presents partially this configuration since it is equipped with a Cell processor and a NVIDIA GPU.

4.6 Conclusions

In this Chapter we presented an interesting algorithm for the matrix pseudoinversion that uses arithmetic decomposition and residue number system representation. The Decell algorithm can be used to exactly calculate the pseudoinverse of the Jacobian matrix while giving an upper bound limit to the execution time of the computation. This is useful in many robotic applications form kinematics to dynamics and control. Moreover we identified in the Cell Broadband Engine processor an appealing parallel architecture that in the future can be used to provide advanced computation capabilities for instance to the controllers of haptic and robotic devices. In the following we will present our approach that permits to implement an inherently parallel algorithm such as the RNS Decell on a multicore processor such as the Cell, using an high level language and intrinsic functions, obtaining good performance by mimicking the VLSI implementation of operations.

5

A case study: Matrix pseudo-inversion

In this Chapter we describe how we implemented the Decell algorithm on the Cell Broadband Engine. The original Decell is based mainly on matrix multiplications, subtractions and trace computations. In order to reduce numerical instability it is possible to use symmetric residue number system representation. When the bases to be used have been chosen the problem is divided into sub-problems by applying each modulo to the original matrix. This is the arithmetic decomposition. For each modulo the procedure described in 4.4 is applied. At the end the resulting matrices are recombined together producing the required result.

Given this description of the problem, the first step is to think to the programming model of the algorithm.

5.1 Programming model

As we stated in Chapter 3 the programming model is an abstract model of computation that is used by the programmer to reason about a program execution. Each algorithm is characterized by three parts: input, output and the process itself.

Input

The input is the matrix A that has to be pseudoinverted. We assume that A is a $p \times n$ matrix of integer values with $p \leq n$ otherwise, using the relation $A^+ = \left[\left(A^T\right)^+\right]^T$, we can compute $\left(A^T\right)^+$. It has to be noticed that we consider also the case $p = n$ in which, if the matrix is non singular, the inverse is defined but we apply the Decell algorithm anyway.

The use of residue arithmetic presumes that each element a_{ij} of the matrix A is an integer. Although fixed word-length computers store only rational numbers, they can be converted to integers by an appropriate scaling [23]. For example, in the radix u system, a_{ij} can be evaluated by

$$a_{ij} = \sum_{k=-s}^{s} \alpha_{ij}^{(k)} u^i = u^{-s} \left(\sum_{k=0}^{2s} \alpha_{ij}^{k-s} u^k \right) = u^{-s} \overline{a}_{ij}$$

where

$$\overline{a}_{ij} = \sum_{k=0}^{2s} \alpha_{ij}^{(k-s)} u^k$$

is the normalized integer and u^{-s} is the constant scaling factor. Thus, we can obtain the normalized matrix $\overline{A} = u^s A$. Since

$$A^+ = (u^{-s}\overline{A})^+ = \frac{1}{u^{-s}}\overline{A}^+ = u^s\overline{A}^+$$

one may apply the proposed residue arithmetic Decell algorithm to obtain the pseudoinverse $\overline{A}^+$ from which the desired pseudoinverse A^+ may be obtained by multiplying the scaling factor u^s. The need of conversion is not a big issue. In our previous FPGA implementation we faced the same problem, since we were forced to use 16-bits values. We realized that the primary input is given by sensors and raw sensor data are usually integer. Whatever computational strategy is chosen these values have to be transformed to a different representation, thus we are just moving this conversion at a later time. Moreover, all the trigonometric functions used to populate a forward kinematic matrix as well as a manipulator Jacobian can be performed in integer arithmetic by using, for instance, the CORDIC algorithm, and all the conversion/scaling can be computed in a fixed-point notation that is perfectly representable with integers, when carefully handled. Thus we do not consider this aspect as a limitation and, since the conversion is due anyway, we think that the conversion time has not to be accounted when comparing the Decell algorithm with other floating point approaches.

The other inputs of the algorithm are the bases or moduli, their number L and the dynamic range M they provide.

Let $m_1, m_2, \ldots, m_L$ be the pairwise relatively prime bases (or moduli, or radices) for the mixed-radix system. The dynamic range is given by:

$$M = \prod_{i=1}^{L} m_i. \tag{5.1}$$

A number x can be expressed in the mixed-radix form as

$$x = a_L \prod_{l=1}^{L-1} m_i + \cdots + a_3 m_1 m_2 + a_2 m_1 + a_1$$

where the a_i are the mixed-radix digits and $0 \leq a_i \leq m_i$.

The selection of the bases $m_i, 1 \leq i \leq L$, is critical to the success of the method.

A possible method for selecting the range M is:

$$M > \max\left\{X^p, p(p-K+1)X^{p-1}\right\} \tag{5.2}$$

where $X = \min\left\{\text{trace}(AA^T), ||AA^T||\right\}$, and $K = \text{rank}(A)$. For simplicity two practical criteria satisfying Equation 5.2 are

1.
$$M \geq 2 \prod_{i=1}^{p} \left(\sum_{j=1}^{n} a_{ij}^2 \right)^{1/2} \tag{5.3}$$
where a_{ij} is the (i, j) element A.

2.
$$M > p^{p^2/2} X^{p^2} \tag{5.4}$$
where X is the maximal absolute value of an element in the matrix AA^T.

In [106] the authors suggest that the bases $m_i, 1 \leq i \leq L$, should be chosen to be composed by large prime numbers greater than p and must satisfy Equation 5.1.

Output

The output of the algorithm is A^+, the pesudoinverse of the given matrix A. In addition Decell provides two other information at no computational cost: the rank of A and the coefficients of the characteristic polynomial associated with the matrix.

Process

We are not interested in describing details about the process at this point. In general it follows the steps of the algorithm indicated in Equation 4.6. In this part we would like to focus on some design aspects that have to be considered early during the development because they are necessary in the analysis of the algorithm and in particular in the parallelization. The more important facts are:

The first step is the choice of the moduli, following one of the rules indicated before. This choice depends only on the arithmetical range of the elements in the matrix that has to be pseudoinverted. In the case of robotic application this means that a profiling of the value of the Jacobian can reduce the required range to the minimum. This is important since the dynamic range $M = \prod_{i=1}^{i=L} m_i$ depends on the number of moduli and the number of bit length used to represent the numbers. Thus a desired range can be obtained by using few moduli with more bit length or more moduli with smaller bit length. Since the Jacobian reflects the structure of the manipulator this kind of tuning has to be done just once for each robot.

The moduli and the dynamic range are stored in the main memory together with the matrix A, in order to be easily accessed. The values of the matrix can be organized in the memory using several data structures. For our implementation we opted for a simple array representation, i.e. all values are stored in a monodimensional array in raw major. This approach is preferable to more complex data organization since it permits an easy data transfer among the processing units and does not require any particular functionality in the data fetching, since the values are simply stored contiguously.

The values, useful throughout the algorithms, that require some computation in order to be calculated and are related only to the moduli choice can be computed once for all and stored until needed. For instance, this is the case of the coefficients used in the arithmetic recomposition process with both Mixed Radix Recomposition and Chinese Reminder Theorem.

There are three different matrix operations involved in the Decell iteration: matrix multiplication, trace computation and matrix diagonal subtraction and they require symmetric RNS arithmetic.

The recombination can be carried out essentially in two ways: with the Mixed Radix Recomposition (MRR) or with Chinese Reminder Theorem (CRT). Both the procedures are effective and perform a weighted composition of the moduli; the main difference relies on the fact that MRR requires only modulo m_i arithmetic operations while CRT, based on the extended Euclidean algorithm, requires modulo M operations. For these reason, the latter was the choice in the VLSI Decell implementation in [23] since in that case CRT would have needed a different hardware design for the residue processor. In a software implementation, where the demand for higher precision can be met, the two alternatives are equivalent, with CRT performing a sligthly better.

5.1.1 Analysis of the algorithm

The Decell algorithm can exploit two types of parallelism:

- Task parallelism: the problem is arithmetically decomposed in independent sub-problems
- Data parallelism: when possible independent data can be processed at once

Task parallelism

Let us divide the algorithm presented in Equation 4.6 in four main parts:

1. Arithmetic decomposition
2. Decell iteration
3. Arithmetic recomposition
4. Evaluation of the pseudoinverse

In a multicore system, each phase can be assigned to a different core or, when parallelism is possible, to a set of cores. The four phases have to be computed sequentially and parallelism is allowed only inside each part, due to data dependency. It is however possible to have a pipelined approach to the problem by permitting the execution of phase 1 relative to the computation at time $t + 1$ at once with phase 2 at time t (see Figure 5.1). This approach maximizes the throughput of the algorithm and permits to reduce the time the cores remains idle.

In our processing model we supposed, without losing generality, that there is a main core that coordinates the other processing elements (PEs) and that it has access to a main memory for data storage. It has to decompose the matrix A into

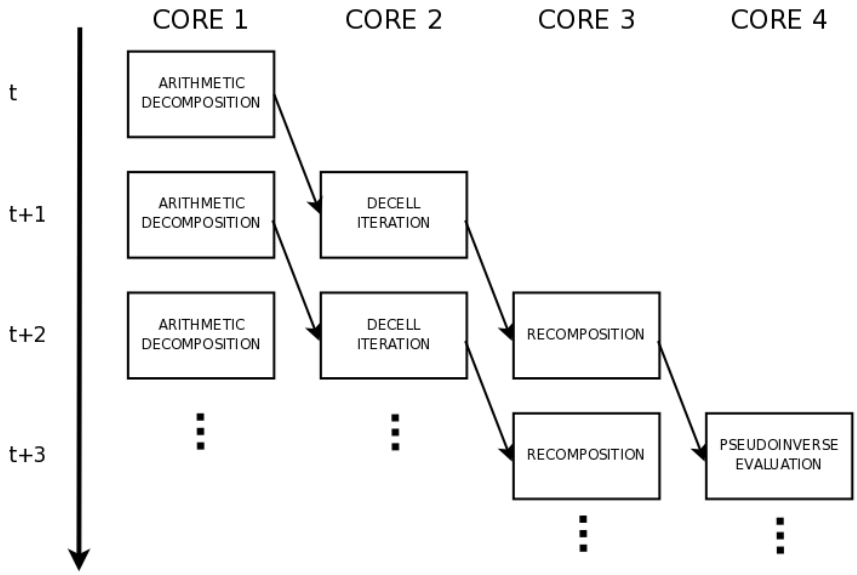


Fig. 5.1. Ideal pipeline for the concurrent computation of the four parts

L residue representation of A modulo m_i with $1 \leq i \leq L$. Since the process producing the residue representation of the matrix has to run L times, one for each modulo, it can represent a bottleneck if performed on the main core sequentially. In this planning phase it is better to place it in the other processing elements. In the translation from the programming to the processing model this choice has to be evaluated with respect to actual data representation and movement. Thus the matrix A, or part of it, has to be moved to each processing elements along with the modulo used in the specific residue computation, the precomputed coefficients and the dynamic range.

The main processor has also the responsibility to transfer or activate the code that has to be run on the other cores. At this point we can assume that the Decell iteration on the residue representation of A are carried out in parallel.

The recomposition phase, that recombines the resulting residue matrices into a single one, cannot start until all the PEs computations have been committed and the results stored in one place. Two alternative designs are:

1. The main core synchronizes the PEs, waiting for all the results to be moved back in main memory, and then perform the recomposition by itself. However this part of the algorithm is still computational extensive, since it requires to access and combines $n \times p$ values in L residue matrices, and relies on symmetric RNS arithmetic since both procedures involve modulo operations.
2. One of the PEs can assume the role of recombining the values after its Decell iteration ended. Since all the resulting residue matrices are needed, this solution requires that the results are moved from all the PEs to an idle processing element that can perform the recombination. The additional advantage is

that one data-move is no more required, since the PE already has the residue matrix it computed. Unfortunately in the meantime all the PEs remain idle since a synchronization among all of them is required before starting a new computation of Decell. The time spent in Decell iteration is the time of the longest computation on the PEs. Thus this alternative has to be carefully evaluated. However, this approach is particularly suitable if the number of PEs is greater than L, thus meaning that some PEs are not involved in the residue Decell iteration. In this case the pipelined approach presented in Figure 5.1 can increase the throughput.

The last phase of the Decell algorithm, that is the evaluation of the pseudoinverse in Equation 4.6, can be performed by the main core since the resulting values is the final result, the computation is somewhat limited and no modulo operation is involved.

Data parallelism

As stated earlier the Decell iteration is mainly based on matrix operations. In general, matrix operations are computationally intense but the values have a high degree of independence. In addition/subtraction operations, for instance, each value can be added/subtracted independently from the others. Multiplication is more complex, since a resulting value is given by the recombination of rows and columns of the factors, however the values in different rows (or columns) are not related and thus they can be computed at once. In literature there are many examples of parallel implementation of matrix multiplication, in particular when the factors have same specific structure. In general it is possible to exploit the independence available to speedup the computation splitting the operation in concurrent operations. Another interesting aspect related to matrix operations is the possibility to operate on large matrices by using block matrices. A block matrix or a partitioned matrix is a partition of a matrix into rectangular smaller matrices called blocks. A block partitioned matrix product can be formed involving operations only on the submatrices. This property is useful when it is not possible to have the whole matrix fit in memory, and especially in local store memory of smaller processing units. The block can be manipulated at once, leaving space for additional task parallelism.

The outcome from these considerations is that if the underlying architecture provides some sort of data parallelism, such as the one given by SWAR or SIMD capabilities, the Decell iteration can be designed in order to exploit this data independence.

5.1.2 The programming language

There are several systems developed in order to easily program GPUs hiding their graphics-specific nature while making their processing power available for general purpose computations. In some cases, these systems have also been mapped to additional targets, such as the Cell BE and multicore CPUs making these platform an important tool that can help in translating the programming model to the specific target [71].

The RapidMind platform [70] uses an SPMD stream programming model but generalizes it to multidimensional arrays. RapidMind uses an embedded programming model so that the kernels can be specified directly in the source code of a controlling C++ program. The RapidMind platform can target the Cell BE, both NVIDIA and ATI GPUs, and multicore CPUs with the same programming model. In order to express memory locality, local arrays may be declared and operated on inside other kernels, and arrays may be tiled into subarrays. The RapidMind array abstraction also encapsulates data strongly, allowing automatic management of remote data stored on accelerators.

Intel [40] is developing Ct, a data-parallel programming platform that targets x86 multicore CPUs. Ct supports the SPMD programming model and uses an embedded C++ interface similar to the RapidMind platform. Ct also includes an implementation of segmented collectives, which are useful for implementing algorithms using nested parallelism.

Programming models based on distributed memory models such as MPI can be mapped onto multicore processors such as the Cell BE that are essentially clusters on a chip, but the limited local memory space available relative to the requirements of existing MPI codes makes this difficult. MPI was primarily designed for large-grain tasks. However, the MicroMPI system [80] presents some modifications to MPI that make a mapping onto the small local memories in the Cell BE.

Although all these programming platforms are appealing and interesting we did not use them in our implementation. There are several motivations behind our choice and, in general, we agree that the modification of programming languages and paradigms is necessary to change the software development precess in presence of parallelism and multicore architectures. In addition a system that permits to expose the most critical parts of the program in terms of performance and complexity while hiding unnecessary target specific details is really important in order to stop thinking to the machine and focusing on the program itself. However these systems are still in development and usually they works well for their main target but they are less effective when targeting other platform. This depends mainly on the difficulty to map the programming model, for instance SPMD for RapidMind, to architectures that do not adhere perfectly to it.

The main reason for our choice is that we wanted to expose some mechanisms while drawing on the Decell VLSI implementation and this approach requires the use of specific intrinsic functions of the target platform (available also on other architecture with different syntax). In addition to better understand strong and weak points of these emerging multicore systems it is better to deal with the translation into the processing model directly.

Since in our implementation we left out some aspects that can be handled by new platforms, that still influence performance and the goodness of results but are not particularly relevant to the migration that we are analyzing, in the future a combination of parallel programming systems and custom implementations could be interesting.

In the following we will analyse the processing model and the implementation mainly for PS3 while keeping one eye on the other multi-core architecture that

we presented: GPUs and FPGA. It is difficult to put FPGA in the same category as Cell and GPUs, since the difference in the organization and programming are relevant. However , it is our idea that is possible to identify strategies and aspects that can be used in the programmable field array with some effort in adaptation and translation.

5.2 Sony PlayStation 3

The Cell broadband engine is available on the market in two main form: the IBM blade accelerator (for instance the QS21) and the Sony PlayStation 3 (PS3). The former expresses all the capabilities of the processor with a street price of more than 5000 Euros while the latter offers a limited version of the CPU at a cost of 400 Euros approximately, thus one order of magnitude lower. Moreover the first version of the PS3 allows to install an operating system in addition to the proprietary one. There are several versions of the PS3 which have minor differences such as the presence of fewer USB ports or the lack of back compatibility with PS2 games. Unfortunately the very last version, the PS3 slim, has lost the possibility to install the second operating system.

The first alternative operating system available was GNU/Linux and, since the PPE is mainly a PowerPC, Yellow Dog, a distribution that was targeted to Apple computers, released the first porting. The official IBM Cell SDK was included in the software equipping the distribution together with a GNU GCC compiler for PPE and SPE, permitting de facto the development of code for the Cell. Later on, several other distributions provided full support to PS3.

The limitations of the PS3 Cell are that not all the SPEs are available, since one of them runs the hypervisor of the system, that is a program which arbitrates access to the low level hardware from the operating system, and another one is simply disabled. Another difference is that it is not possible to obtain the position of the SPEs in the EIB ring and thus the topology of the connections. Less important for Cell development but interesting from an accessibility and openness point of view is that the hypervisor prevents any direct access to the PS3 graphic card, that is an NVIDIA RSX model. It is easy to speculate that this limitation is due to a successful attempt to prevent manoeuvres aimed at videogame development and/or violation. A bothersome side effect is that since the NVIDIA board supports GPU computation, a completely open PS3 could have been a valuable hybrid multicore testing system. Anyway the inexpensiveness of the console and the possibility to install GNU/Linux and IBM SDK permits not only the development of software but also the possibility to test the goodness of a Cell solution to a specific problem before affording the investment of an IBM Blade. Meanwhile, a hack to control the hypervisor under GNU/Linux has been found [47].

For these reasons the PS3 is an appealing platform for scientific research as stated for instance in [59]. In addition the possibility to easily connect several PS3 in a cluster assures supercomputing performance with low budget outlay [32, 55] with some limitations [21].

For our implementation we used the PS3, first version, equipped with Yellow Dog 6.0 and IBM Cell SDK version 3.1. We programmed in C using the GNU/GCC compiler.

5.2.1 Cell processing model

In Section 4.5.3 we presented the structure of the Cell Broadband Engine (CBE or simply Cell in the following) in details. We would like to call to mind here the important elements of the architecture, that are necessary to understand our approach. The Cell is composed by one main core, the Power Processing Element (PPE) that is basically a PowerPC processor, and by eight Synergistic Processing Elements (SPE). All the cores are connected, together with the memory controller and the Input/Output chips, through the Element Interconnect Bus (EIB), a circular ring comprising four unidirectional channels which counter-rotate in pairs. In the PS3 Cell six SPEs are available for programming.

We can draw a first parallel between what we defined earlier about the programming model and the Cell processing model by identifying two types of parallelism available in the architecture:

- Distribution of the computational load among the SPEs (Task or MIMD parallelism)
- Speed up of the computation using vector registers (Data or SIMD parallelism)

In our implementation we considered them separately.

MIMD parallelism

The role of main processor is naturally given to the PPE while the most intense computation is left to the SPEs.

We can rewrite the phases of the algorithm highlighting whose cores are assigned to each step:

1. Arithmetic decomposition (SPEs)
2. Decell iteration (SPEs)
3. Arithmetic recomposition (PPE)
4. Evaluation of the pseudoinverse (PPE)

One of the principal functions of the PPE is to coordinate the activity of the synergistic processors, performing a sequence of preliminary steps and waiting for completion. In details the PPE has to fill a structure that contains reference to the main memory area where information needed by SPEs is stored, and all the accessory data needed by the sub-process. In our case the structure is depicted in Figure 5.2.

The members `matrix_X` and `spe_result` are the addresses in main memory of the matrix to be pseudoinverted and of the results of the SPEs computations. The padding at the end of the structure is needed in order to maintain data quadword aligned. This is necessary in order to obtain good performance during data transfer. The PPE initializes the structure for each SPEs that has to be involved and then creates a thread for each SPEs passing the `struct` as an argument and a reference to the program to be executed. Since this is a MIMD architecture the program can be different or, as in our case, the same for all the synergistic processors. The main memory addresses are used by a SPEs to request a DMA transfer

```
typedef struct _control_block {
  addr64 matrix_X;
  addr64 spe_result;
  int spe_num;
  int size_m;
  int size_n;
  int mods;
  unsigned char pad[90];   /* pad to 128 bytes */
} control_block;
```

Fig. 5.2. PPE to SPE control block structure

in order to copy data to its Local Store (LS), that has no associated cache, and viceversa. After the initialization the PPE waits for a sync message from all the SPEs, via mailbox messages.

Figure 5.3 is a graphical representation of the data transfer among the cores. For the sake of simplicity the main memory is considered part of the PPE. This is not true from an hardware point of view, since there is a separated Memory Controller (MIC), but it is acceptable from a functional point of view since the main memory is directly addressable only by the PPE.

After the computation of the residue matrices, the execution of the algorithm continues in parallel during the Decell iteration phase as described in Equations 4.5. At the end of this step there is a resulting residue matrix, in row-major, in each LS that has to be transmitted back to a specific location in main memory. The consequence is that all the resulting matrices are stored sequentially in adjacent locations and are easily accessible by PPE for recombination. The SPEs communicates to the power processing element that they have complete their computation through messages. When all the messages are arrived, the PPE takes care of recombining these matrices into a non residue unique matrix by using the Chinese Reminder Theorem or the Mixed Radix Recomposition. Subsequently it computes the pseudoinverse as depicted in Equation 4.6.

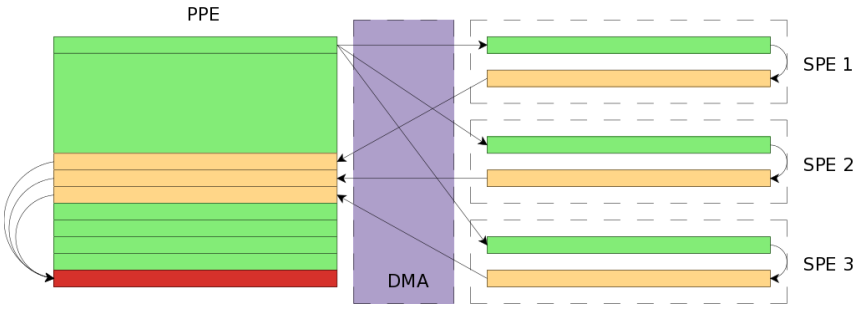


Fig. 5.3. Data storage and transfer among cores

We decided to maintain the recomposition phase in the PPE and not in one of the SPE. There are two main reasons.

1. First of all the recombination on the SPE requires intra SPEs communication and synchronization that have to be carefully implemented, otherwise it can represent a real bottleneck that can worse performance. In addition an effective communication requires to know the topology of the SPEs in the EIB ring. Of course the physical position of the SPEs does not change over the time but the ID assignment of the synergistic processors can vary among executions. One of the limitation of the PS3 Cell with respect to IBM board is that the PS3 does not have a function that returns the IDs order.
2. A second motivation relies on the fact that even if we worked with Cell intrinsic functions, we wanted to maintain a general approach to multicore programming. Our aim was to try to use strategies and operations that can be found also in GPU and FPGA or, at least, that are not conflicting with the processing models of these architectures. In a FPGA on a SoC the role of the PPE has to be taken by an external CPU, since the evaluation of the pseudoinverse requires non integer capabilities. In addition the modulo M operations involved in the Chinese Reminder Theorem that we wanted to compare with MRR, can be not feasible on FPGA.

However for a more application oriented approach, in an optimization point of view, if the PS3 is the target architecture of choice it is worth considering the speedup that a recombination on SPE can bring.

SIMD parallelism

Both the PPE and the SPE can exploit data parallelism using SIMD or vector instructions. A vector instruction operates on a set of data elements at once. The synergistic processors are described as more adept at compute-intensive tasks and slower than the PPE at task switching, thus perfect for calculations, and one of the reasons is their SIMD nature. However, since also the PPE and any recent processor can perform multimedia vector operations, it may seems that the emphasis on the SPE abilities is not completely justified. In our opinion the difference lies in the specialization of the core itself: while a general purpose processor can use a limited amount of SIMD operation while dealing with the operation system, the handling of devices, and a complex multitasking environment, the SPE is completely dedicated to a single task that can be executed without preemption. All the hardware design of the SPE is targeted to vector operation while, for instance, the AltiVec extensions share some processing elements with the general purpose registers. Moreover the lack of standard registers in the SPE makes the use of SIMD operations necessary.

Each SPE has a 128-bit 128-entry register file. An SPE can operate on sixteen 8-bit integers, eight 16-bit integers, four 32-bit integers, or four single-precision floating-point numbers in a single clock cycle, as well as a memory operation (see Figure 5.4). The SPU programming model introduces a set of fundamental vector data types to the C language. The vector data types are all 128-bit long and

contain from 2 to 16 elements, depending on the data type. Table 5.1 shows the supported vector types [101].

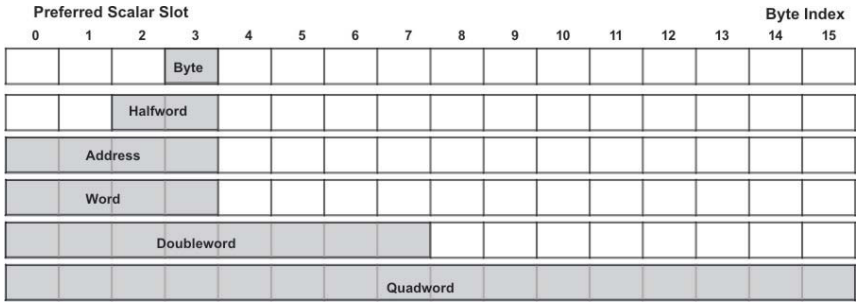


Fig. 5.4. Register layout of data types and preferred (scalar) slot (from IBM CBE Programming Tutorial v3.0)

Vector Data Type	Content
vector unsigned char	16 8-bit unsigned chars
vector signed char	16 8-bit signed chars
vector unsigned short	8 16-bit unsigned halfwords
vector signed short	8 16-bit signed halfwords
vector unsigned int	4 32-bit unsigned words
vector signed int	4 32-bit signed words
vector unsigned long long	2 64-bit unsigned doublewords
vector signed long long	2 64-bit signed doublewords
vector float	4 32-bit single-precision floats
vector double	2 64-bit double-precision floats

Table 5.1. Vector Data Types

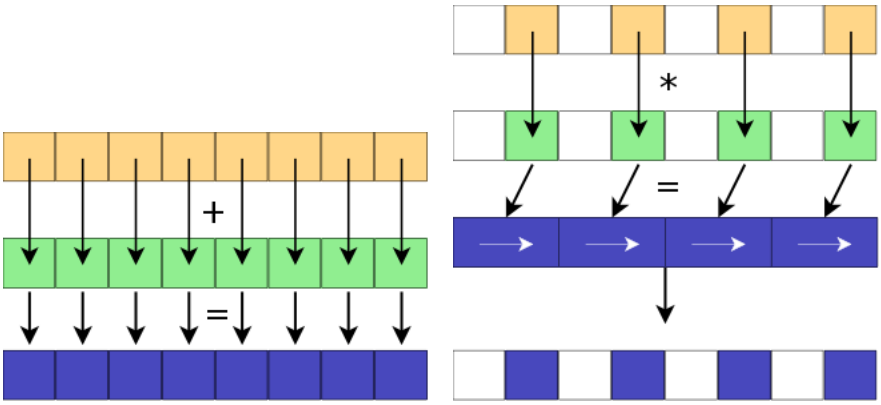
According to [1], the intrinsic functions are a large set of SPU C/C++ language extensions that make the underlying SPU Instruction Set Architecture and hardware features conveniently available to C programmers. These intrinsics can be used in place of assembly-language code when writing in the C or C++ languages.

In order to exploit the data parallelism present in the matrices used for Decell algorithms we decided to apply vectorization, that is to use vector operations, (also known as SIMDization) extensively. In fact since the values we worked on are almost independent it is possible to operate on them in parallel. We identified the intrinsic functions more useful for this purpose avoiding the operations that were exclusive for the Cell. We opted for classical shift and add operations, masked selections, comparison functions and, at some extend, multiplications.

The intrinsic for the addition of two vectors (`d = spu_add(a, b)`) is a good example of vector operations we used for the Decell algorithm. Each element of

vector `a` is added to the corresponding element of vector `b`. If `b` is a scalar, the scalar value is replicated for each element and then added to `a`. Overflows and carries are not detected, and no saturation is performed. The results are returned in the corresponding elements of vector `d`. This function accepts in input a vector of unsigned shorts as well as a vector of doubles. Figure 5.5 (a) shows graphically the operation applied to two vectors of 16-bit values.

The SPU uses 16×16 multipliers. The use of 16-bit values maximizes the speed. This is the same precision usually available on FPGA. Thus this fulfill our requirements. However the use of these multipliers has a drawback. In fact since the multiplication of two 16-bit values can generate a number that needs a 32-bit representation, the relative intrinsic function `spu_mulo` can operate only on the values of the vectors in odd positions because the results are extended using the even slots, erasing any pre-existent value as depicted in Figure 5.5 (b). Although we used modulo operations and thus we knew that at the end the result cannot exceed the 16-bit representation and thus we can cast the value back to 16-bit, the use of the fast multipliers requires this temporary extension. This limited de facto the parallelism of our SIMD implementation to four values per vector. We will see in the following that an alternative is possible in specific situations.



(a) SIMD addition (b) SIMD multiplication

Fig. 5.5. Vector operations

In the algorithm we focused on creating residue operations for a core matrix block that is a 4×4 matrix of 16-bit values. It has to be noticed that the dimension is completely due to the limitations introduced by the multipliers we described above. In our processing model each block matrix was composed by four vectors with four values each. The values presented in the even slot of the vectors can be considered undefined. They were never addressed directly and they were only used by the `spu_mulo` multiplication intrinsic function. Using standard block matrix multiplication (see Figure 5.6) it is possible to consider any generic $m \times n$ matrix and the limit is theoretically due to the LS dimension. In the case that the con-

sidered matrix is too big for it the same strategy of decomposing the problem can be used to partition the matrix and only one or more block can be sent to SPEs at ounce. This, of course, requires to deal with accumulation and recomposition in the PPE. Since our "building" block had fixed size, we implicitly assumed that the dimensions of input matrix A were multiple of the block ones. In order to be able to permit the application of the algorithm to any matrix we had to introduce zero padding.

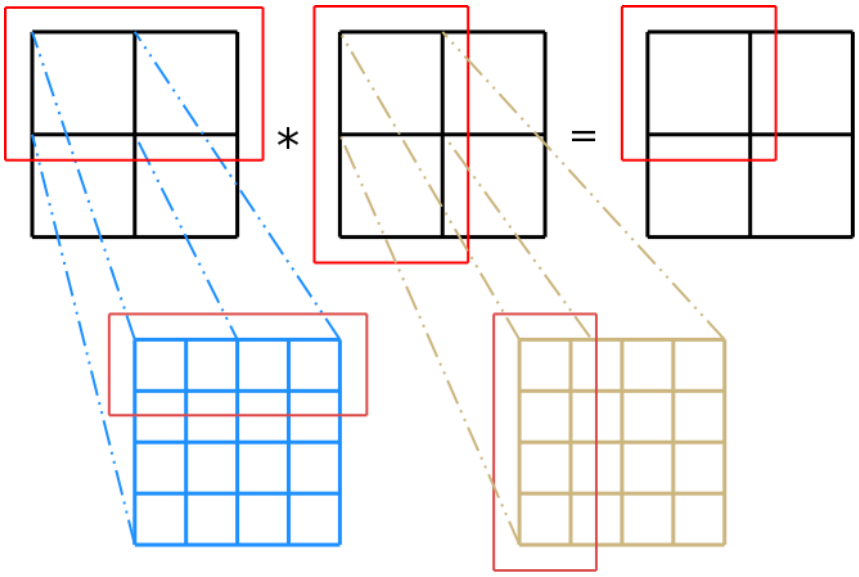


Fig. 5.6. Block matrix multiplication

Data vectorization can provide excellent results because it allows data parallelism but it also requires a careful design of the vector computations. This is due to the less flexible manipulation of data that have always to be considered together with an high latency cost for every single value access.

Let us examine, for instance, the data organization we chose for matrix multiplication, that was row-major for the first factor and column major for both the second factor and the product (see Figure 5.8). This approach permitted to use intrinsics for the row/column scalar product and only two matrix translation operation per Decell iteration, one for the AA^T evaluation and one for the final realignment, at the cost of a negligible slow down due to data rearrangement at the end of the multiplication. The last step was to sum all the elements in the resulting vector to obtain the result depicted in Figure 5.7. This is an horizontal operation, common in most SIMD/SPMD architecture but not available on the SPE, thus a combination of add and shift vector operations was used. The degradation of performance required from this approach was overcompensated by the reduction of heavy matrix transposition, in particular when the rank of the matrix is higher than 2, that is a plausible assumption.

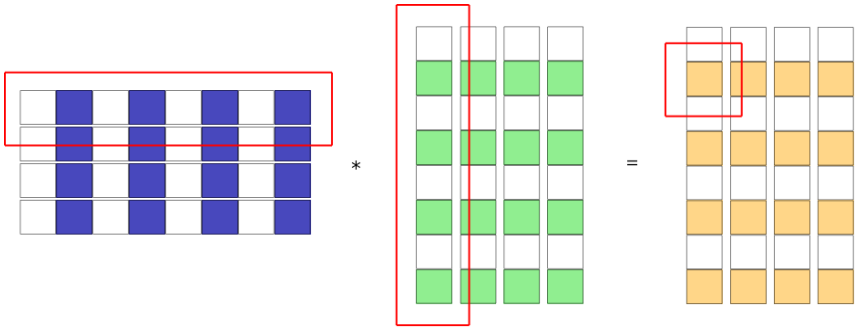


Fig. 5.7. Data organization for matrix multiplication

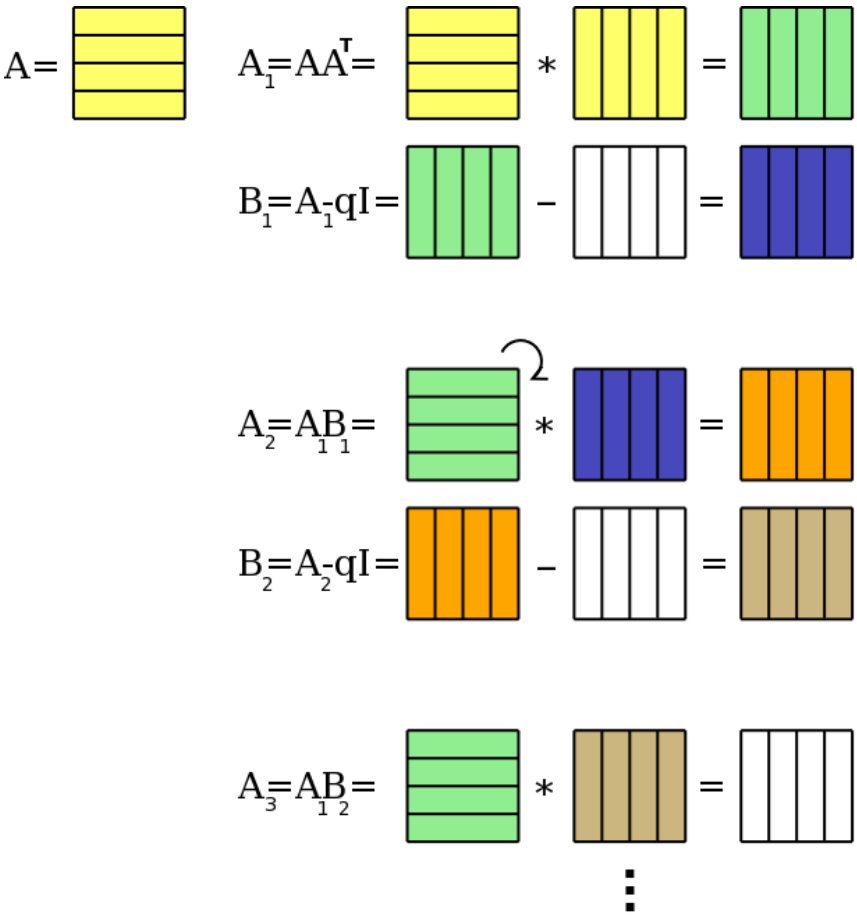


Fig. 5.8. Matrix row major and column major organization for Decell

Implementation

Initially we started the implementation of the Decell algorithm on the CBE with a conventional, high level approach. Given the programming model and with a light

overview of Cell intrinsic functions we begin focusing on the SPE coding, since for our purposes the PPE can be considered and hence programmed as a standard CPU.

Before considering the algorithm itself it was necessary to provide the SPEs with symmetric RSN support. In particular we needed a set of functions that were used throughout the computation and in details:

- a symmetric residue representation of a given integer number conversion function
- a residue add function
- a residue multiplication function

We designed the code for these function starting from the mathematical and algebraic definitions of modulo, residue number system and so on. Thus, for instance, to find the symmetric Residue representation of a given number we considered the Definition 4.3 and we used the relation described in [56]:

$$/x/ = x - \left\langle \frac{x}{m} \right\rangle * m$$

where $/x/$ is an integer such that $-[(m-1)/2] \leq /x/ \leq [m/2]$ and the symbols $[\ ]$ indicate a rounding operation. For m odd, the quantity $\langle x/m \rangle$ is the closest integer to x/m. If m is even, the quantity $\langle x/m \rangle$ is again the closest integer x/m except that if x is of the form $n * m/2$ where n is odd, the quantity $\langle x/m \rangle$ is the closest integer to $(x - 1/m)$.

Since our design already plans to exploit data parallelism, we worked on vectors using intrinsic. The transformation of a number in its residue representation is obviously an atomic and independent operation, thus it is easy to work on several values at once. We used intrinsic to implement the above relation and we succeeded. The result was a perfectly working function but the procedure used was a general purpose sequential approach ported and replicated to SIMD.

We mainly used the same approach also for the residue add and multiplication operation. The easiest way to perform a modulo addition is to compute the addition as usual and then apply the symmetric residue modulo conversion. Thus we combined the use of the vector addition and multiplication intrinsics depicted in Figure 5.5 with the above procedure. It is clear that the computation time spent in the arithmetic operations is negligible compared with the one used by the conversion routine. A poor performance in this procedure propagates to the complete algorithm.

After the completion of the residue arithmetic operations, we focused on the Decell iteration phase. The iteration can be split in four main parts:

1. initialization
2. residue matrix conversion
3. iteration step
4. exit condition check

The second point, that is not part of the Decell iteration but represents the arithmetic decomposition in the original algorithm, is included here according to our

programming model. In the following we are considering the synergistic element i.

The **initialization** is required in order to transform the values that represent the matrix in row major that are transferred to the SPE local store area in variables of type vector. For our core block multiplication we already designed that four vectors are needed. However, in order to deal with larger matrices composed by several blocks the vectors are grouped in arrays. This phase begins when the SPE thread is started and the control block structure with main memory addresses and all the accessory values, such as the modulo m_i, is fetched via DMA. The algorithm uses values that depend only on the moduli of choice and are independent from the matrix. Considering, for instance, the values used for the computation of the $q_j^i = ||j^{-i}|_{m_i}|traceA_j^i|_{m_i}|_{m_i}$ is clear that the $|j^{-i}|_{m_i}$ quantities can be precomputed once for all until the modulo m_i remains unvaried.

The **residue matrix conversion** transforms the given matrix A in its residue representation $|A_i|$, modulo m_i. We applied the `srns` function to the four vectors composing the core block. This is an intensive operation that has to be applied for each value and it is a good example of the potential limitation that the SIMD paradigm requires. The guidelines about SPE's programming suggest to avoid branching in programs. In fact there are no branch prediction mechanism on the SPE and each mispredicted branch can seriously degrade program performance. The typical SIMD solution is to compute both paths of the branches and then select the correct result by using some form of selection. In the symmetric residue system the value zero remains unvaried. Thus there is no need for elaboration on zero values and a sparse matrix theoretically can be converted faster than a generic one. In a SIMD architecture is better to perform the computation on all the values. The disadvantage is that the worst case has always to be considered in performance profiling, while the advantage is that the computation time is constant.

The **iteration step** contains the main part of the Decell pseudoinversion algorithm. This step can also be divided in three parts: the initialization, the iterative section and the exit test. The initialization, apart from the definition of the supporting structures, regards the computation of AA^T and in particular the necessity of the transpose computation. In order to exploit the SIMD mechanism of the SPE, we decided to implement the transposition of the core block using the `spu_shuffle` intrinsic that permits to rearrange the elements of two vectors easily and quickly. Of course this can apply to the core block, since the dimensions and the meaning of the vector slot are known, while extra work is needed when a multi-block matrix is considered. The other two parts are mainly composed by a sequence of residue operations. Special attention was paid to the trace computation and to the test $|A_1 B_i| = 0$ that ends the iteration, always applying, when possible, SIMD operations. Loop unrolling is another guideline suggested in order to increase the performance of the computation on the SPEs, since the exit test suffers of the same problem of the branch misprediction. In general we used loop unrolling when considering the vectors of the core block, since the number of vectors is known. We used a `while` statement for the iteration of the Decell procedure because of

the variability due to the current rank of the matrix.

The **exit condition check** verifies if all the elements of matrix $A_1 * B_i$ are zeros. Again we used a combination of comparison and selection operation in order to perform this check in parallel.

From a preliminary test about the execution performance of our code, we noticed that we were not exploiting the hardware capabilities of the platform completely; more details about these results will be presented in the next Chapter. In our opinion the part of our implementation that needed an improvement was the computation held by the SPEs. In the next Section we describe our approach based on a VLSI implementation of the Decell algorithm and we show later how it can affect performance.

5.3 Learning from the past

A VLSI implementation of the Decell algorithm was presented in [23]. In the article there are all the schematics describing the implementation of the various phases of the process. The author focused mainly on the Decell iteration phase, where the multiplications, addition and trace computation take place. They used an asynchronous, data-driven, wavefront processor array for the iteration. Although this approach was interesting and offered good performance, it does not fit well in the processing model of the CBE. The topology of the Cell is different from the one proposed and, in general, the wavefront array can be assumed to be very large, of the order of the dimensions of the matrix that has to be pseudoinverted. This design is more suitable to FPGAs than to GPUs or Cell. However the authors also presented the design of the residue arithmetic operations needed by the algorithm. We decided to check if the SIMD intrinsics of the SPE were adapt in order to easily replicate the structure of the VLSI components.

Symmetric residue conversion

We started with the symmetric residue conversion function. A possible implementation is based on the restoring division algorithm [61, 76]. It can be defined as follows:

- Assuming that the input is an n-bit positive number, we subtract from it the binary representation of the modulo m such that the highest order "1" bit of the representation is aligned with the highest order input bit. In the synergistic element this was done using the intrinsic `spu_sl`, that is the element-wise shift left by the number of bits that are the count of the leading zeroes of the modulo.
- The shifted version of m is then subtracted from the input, using the function that subtract each element of one vector from the corresponding element of the other one (`spu_sub`).
- If the result is negative, the original input is passed on (i.e., subtract 0). If the result is positive, the subtracted version is passed on. In either case, an $n-1$ bit number will result.

- This number is now treated as the input, m is shifted one less position to the left, and the above process repeats until m is no longer shifted.

The remainder after the last step is the result of the conversion.

The branch described in the procedure was substituted with the computation of the results for both branches and the selection `spu_sel` of the right one based on the outcome of a comparison `spu_cmpgt`. An example of the code is presented in Figure 5.9. The procedure length depends on the position of the leading "1" of the number to be converted. In order to avoid conditional statements we decided to repeat the shift/iterate step a number of times equal to the bit length of the input. This means that we always considered the worst case, that is an input value that uses all the bits available. If this is not the case the subtraction has to be avoided and we used the number of leading zeroes as the selection criterion. We then unrolled the loop and copied the shift/iterate procedure in sequence.

```
m = tottot; // current remainder

// Shift/iterate
shift=spu_add(shift,-1); // decrement the counter
b=spu_cmpgt(zeroes,(vector signed short)shift); // is negative?

m2=spu_sel(spu_rlmask(m2,-1),m2,b); // shift
tottot=spu_sel(spu_sub(m,m2),tottot,b); // subtract and check

// propagate the remainder if we finished the iterations
b=spu_cmpgt(zeroes,tottot); // is negative?
tottot=spu_sel(tottot,m,b); // select current or previous value
```

Fig. 5.9. The intrinsics used to code each shift/iterate step

If the number to be converted is negative its complement is considered as the input for the procedure, and the result of the procedure has to be complemented.

Figure 5.10 shows the application of the algorithm to find the modulo 5 representation of a given number.

Residue addition

We decided to apply the same strategy to the residue addition. We assume that the input of the adder are numbers already expressed in the residue number system and we expect that the result is represented in the same way. In [61] is explained that a conventional n-bit binary adder may be treated as a mod 2^n adder if the carry bit is ignored. If the result of the mod 2^n addition overflows, it can be reduced to the correct answer for mod m by adding m' mod 2^n, where m' is the additive inverse of m mod 2^n, to the result of the first addition. If the first addition does not overflow, it may fall int the range $[m, 2^n - 1]$, in which case it can also be

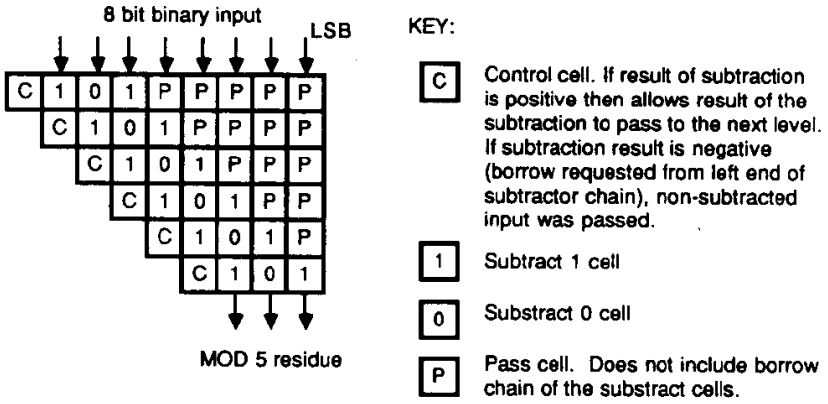


Fig. 5.10. Residue decomposer (modulo 5) floor plan

corrected by adding m' mod 2^n to the first addition. A multiplexer controlled by the carry selects the correct output Figure 5.11.

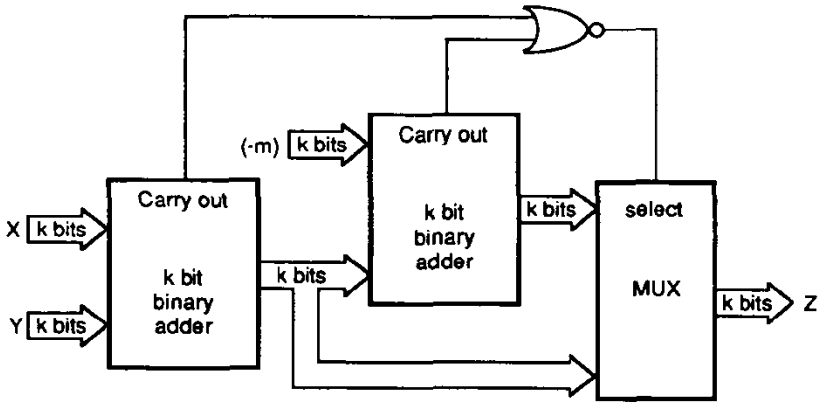


Fig. 5.11. Implementation of residue added

The residue adder proposed, and presented also in [23] works whit a standard residue system but it does not consider that the values can also be signed, as in our symmetric representation. In addition the design is based considering a specific type of binary adder and the capabilities of detect carries and overflow. Since these operation are slightly different on our SIMD architecture, we implemented our reside adder in a different way, always considering a decomposition of the task in simple vector operations.

Let s_k be the sum of the k-th elements of the addend vectors. Since the SRNS representation of the addends is given, we know that after the addition there are only three possible scenarios:

1. $m_i/2 < s_k$

2. $-(m_i - 1)/2_i \leq s_k \leq m_i/2$
3. $s_k < -(m_i - 1)/2$

In the second case nothing has to be done, since the result is still represented in modulo m_i. In the other cases it is sufficient to respectively subtract or add the value m_i to s_i. With this operation the result falls back in the range $[-(m - 1)/m, m/2]$. As usual we computed all the alternatives and then we selected the correct one in order to avoid branches.

Residue multiplication

The last operation we decided to recode was the residue multiplication. The main principle is based on "log transform-residue addition-antilog transform" [61]. The theory guarantees that there is at least one primitive element which generates all nonzero field elements. Given a finite field $GF(p)$, an element μ is a generator of the field if

$$1, 2, \ldots, (p-1) = \mu^0, \mu^1, \mu^2, \ldots, \mu^{p-2}$$

$i \Leftrightarrow \mu^e$ where $O \leq e_i < (p - l)$ The exponent e is the logarithm of the element $a = \mu^e \Rightarrow \log a = e$.

The multiplication of two field elements is equivalent to the addition modulo $(p - 1)$ of the corresponding exponents. If $a = \mu^{e_1}$ and $b = \mu^{e_2}$ then

$$|a \cdot b|_p = \mu^{|e_1 + e_2|_{(p-1)}}$$

and

$$|e_1 + e_2|_{(p-1)} = \log(a \cdot b)$$

Therefore, multiplications can be implemented by adding the appropriate exponents as determined from a logarithm table. The procedure finds the exponents e, from the logarithm table, adds indexes modulo $(p - l)$, and finds antilogarithm in table to give result. This procedure is shown in Figure 5.12. Particular attention has to be paid when one of the factors is zero. In fact zero is not part of $GF(p)$ and thus there is not corresponding exponent in the table. It has to be treated separately just forcing the result to zero.

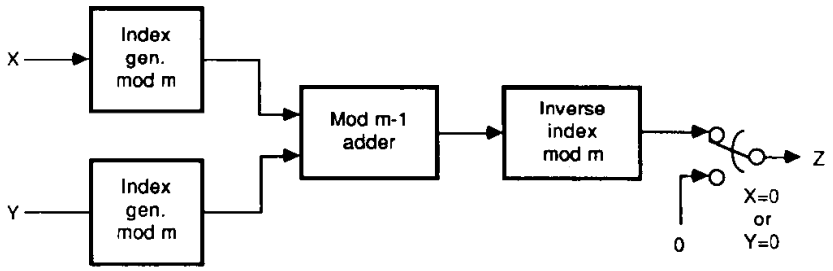


Fig. 5.12. Residue multiplier using index addition

We applied the same technique to our implementation. The first aspect is that we had to prepare the tables with the exponents. There is one table per modulo

and, in order to prove the goodness of the approach, we hardcoded them in the SPE program. However for the sake of portability the right strategy is to send them to the SPEs along with the matrix A. There are methods for filling out the table, given the modulo, and they are costly in terms of performance. However the tables have to be prepared just once for the entire computation, during the initialization of the setup, and thus no strict timing requirements are due for this phase.

The look up table can be defined in almost every architecture since it only requires memory space and access. FPGAs usually have memory modules that can be used for this purpose. As usual in the SPE the limit is given by the amount of LS free space, but the access mode had to be studied. We know that intrinsics work on vector data types so the idea was to use an array of vectors as memory table. The elements of the $GF(p)$ we are considering are all the values i with $1 \leq i \leq (m-1)$ and they can be implicitly stored in our table because they are the indexes for our vectors. The element in the position k is thus the exponent e_k and $k = \mu^{e_k}$. We had to assign a position to the element zero too, in order to have the procedure working in the same way even in presence of such an input. The first part of the procedure consisted in determining the position of a given element, and hence its exponent. The simplest implementation is to have a loop covering the entire length of the array of vectors.Unfortunately this approach requires loop and branch in order to work and we know that these statements are performance killers in the SPE, and in general in SIMD architectures.

To overcome this limitation we unrolled the loop and used comparison and selection intrinsics in order to match the target in the array of vectors. We used the supporting vectors showed in Figure 5.13.

```
vector short index = {1,2,3,4,5,6,7,8};
vector short incr = {8,8,8,8,8,8,8,8};
vector short ones = {1,1,1,1,1,1,1,1};
vector short comp_result ={0,0,0,0,0,0,0,0};
vector short test;
test = spu_splats(input);
```

Fig. 5.13. Supporting vectors for the exponent selection

Using the intrinsic `spu_cmpeq`, we compared all the elements of the first vector of our table with a vector that contained the value we are looking for, replicated (or splatted with `spu_splat`) in all the elements. The result was a vector that contained one "1" in the position of the match, if this occurred. Using a serie of selections `spu_sel` and masking operations, at the end, if a match was found the index was in one of the words of `comp_result`, otherwise `comp_result` was still all zero. The index was then extracted. The procedure has no branches and runs in constant time, since all the code is executed regardless of where the match was

found. Since the dimension of the array depends on the modulo it is not possible to completely get rid of the loop but is possible to use a combination of unrolled compare and loop with intermediate checks.

For the second step required by the residue multiplier, that is the residue addition modulo $(p-1)$, we recycle the code used for the symmetric residue addition with two main modifications:

1. In this case, since the addends are exponents there is no need of a symmetric residue representation, thus we changed the code accordingly.
2. We still have to deal with an input factor equal to zero. We resolved the problem by adding, at the end of the sum, a comparison with zero and a selection. Putting this control at the end we created a branch free function that runs completely even if the result of the multiplication is easily recognizable as zero. Again we were forced to program for the worst case scenario.

The last phase of the reside multiplier algorithm is to find the element of the finite field by using the exponent computed by the addition modulo $(p-1)$. This is easier, since the exponent is the index of our array of vectors. Thus we just had to extract the correct values.

Other functions

Given the promising results we obtained with the RNS arithmetic operations we applied the same strategies to the other parts of the Decell iteration phase such as, for instance, the trace computation. We do not present each implementation here but, in general, we noticed that replacing operations with add, shift, select and mask intrinsics when possible leads to an increase in performance.

5.4 Conclusions

In this Chapter we showed our idea about the possibility of easy code migration from hardware to multicore systems. We started defining a problem that can be useful for robotic and haptic applications and that, in those fields, requires some additional constraints that are not addressed by standard solutions. In the specific we were interested in matrix pseudoinversion carried out at constant time. Since we planned to use one of the emerging multicore architecture available on the market we decided to look for an algorithm that was inherently parallel, and thus easier to be ported to our system of choice: the Cell Broadband Engine equipping the Sony PlayStation 3 game console. We identify an algorithm, the Decell algorithm, that was implemented in VLSI in the '80s. We proposed and analyzed a programming and a processing models in order to exploit both the task and data parallelism available on the Decell procedure. Then we showed that a SIMD architecture such as the CBE have intrinsics function available for high level programming languages such as C/C++ that can easily mimic the VLSI implementation while operating on set of data. In the next Chapter we will show data results about the execution time of our code, giving special attention to the comparison among the alternative implementations we developed.

6

Experimental results

In the previous Chapter we described the implementation of the SRNS Decell algorithm on the CBE architecture. In the following we present a set of experimental results in order to demonstrate the goodness of our approach. With the help of specific tools and profiling instructions we analyze the time performance of our implementation of the Decell algorithm, considering the weight of each phase both in PPE and SPEs. We focus mainly on the SRNS operations and on the most crucial aspects of the algorithm. We perform comparison among the different strategies we described earlier and we highlight the improvement given by using the VLSI implementation as a reference. We examine also the execution results of the overall procedure and we provide considerations and motivations about the data we collected.

Early results

At the end of the implementation we wanted to have information about performance. There are several ways to check for this aspect, in both simulation and real system. In the second case the profiling tool may require additional libraries in order to obtain run time statistics. A fast and common approach to code analysis can be obtained using the code presented in Figure 6.1.

The method uses a counter that each SPU has, that count down at a fixed rate (decrementer). Of course the use of this extra code changes the timing of the overall program, but it can be used as a coarse timing reference when applied to small chunk of code. The precision of the result is limited by the granularity of the decrementer, however it can be considered good enough for standard implementations, in particular when it is used to compare two implementations for the CBE.

We wanted to have an idea of the time spent in the main phases of the computation and a comparison with a state of the art method. For this last purpose we considered other two Cell matrix pseudoinversion implementations based on SVD. The first one is derived from an algorithm presented in Numerical Recipes (NR) in C [87], while the second is made by using a function of LAPACK, with standard parameters, the most famous library for linear algebra computation [62].

```
//Code profiling from MIT lessons
// Start counting
spu_writech(SPU_WrDec, DECR_COUNT);
spu_writech(SPU_WrEventMask, MFC_DECREMENTER_EVENT);
int startMit = spu_readch(SPU_RdDec);

// Code to be profiled...

// Stop counting, print count
int endMit = spu_readch(SPU_RdDec);
printf("Time elapsed: %d\n", startMit - endMit);
printf("Time elapsed: %4.10f\n", (startMit - endMit)/decrementer);
spu_writech(SPU_WrEventMask, 0);
spu_writech(SPU_WrEventAck, MFC_DECREMENTER_EVENT);
```

Fig. 6.1. Code used to profile the performance of a part of the code

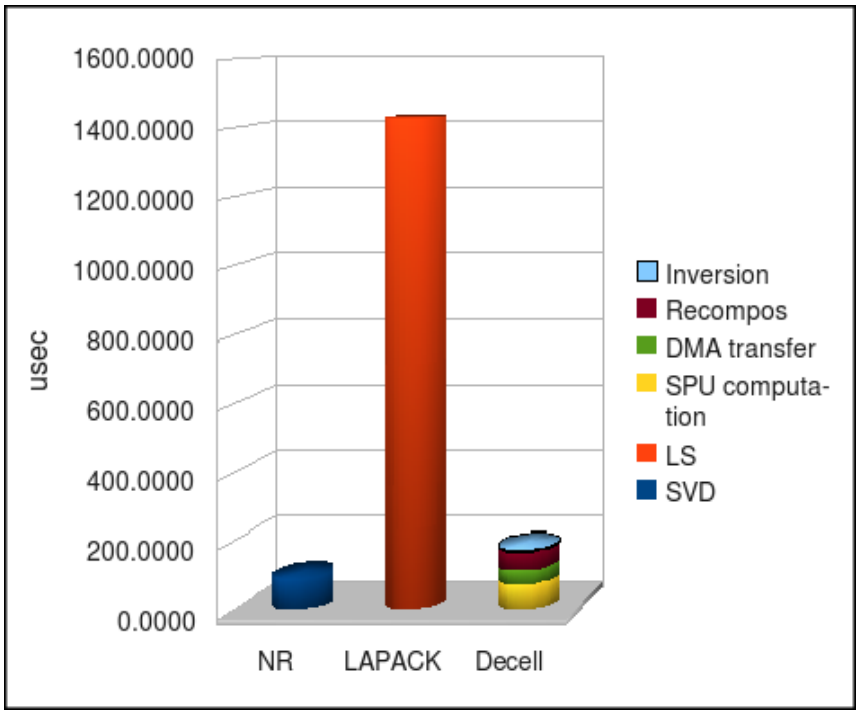


Fig. 6.2. NR, LAPACK and Decell performance comparison

Figure 6.2 depicts the result of our first implementation, obtained following the models, strategies and solutions described so far. The results regard the pseudoinversion of a 4×4 matrix. The first histogram shows that the NR pseudoinversion took 90 μseconds to complete. The second one report a computation time of about

1400 μseconds for LAPACK. The overall time needed by our implementation, that used a four moduli bases and thus split the computation on 4 SPEs, required 175 μseconds. LAPACK does not provide a function for the exact computation of the pseudoinverse of a matrix, mainly due to the numerical instability of the algorithm. However there are LAPACK functions that directly compute a system's least square solution. Since we used one of them in order to obtain the pseudoinverse it has to be noted that the function perform some extra computation with respect to the other two approaches. In addition LAPACK is not yet completely optimized for CBE nor is its vector implementation ScaLAPACK, and it is known to work better with large matrices. It should seems unfair to use LAPACK with standard parameters, and thus with high precision computation, for the comparison but theoretically the use of RSN and Decell leads to an error free computation of the *exact* solution to pseudoinversion, thus it is correct to look for the highest possible precision. We would also like to highlight that the LAPACK routine, with some matrix configuration took up to 1600 μseconds to execute. However we used this SVD performance only as references.

Let us consider each phase of our algorithm. The time spent by each SPE in computation was about 75 μseconds, the DMA transfers required 40 μseconds and the PPE consumed 55 μseconds in arithmetic recomposition, using the Chinese Reminder Theorem, and 5 μseconds in the pesudoinversion evaluation.

The first consideration is that the result depicted in the histogram considers the elaboration carried out in the PPE and in the SPE as sequential. Considering our application as a control loop, the Decell algorithm is continuously invoked. Thus the action of the two types of processors can be pipelined: when the PPE ends coordinating the SPEs for the pseudoinversion of the Jacobian matrix at time t, it can start computing the arithmetic recomposition and the evaluation of the matrix at time $t-1$. In addition, since we mainly focused on SPE implementation, the code used in the PPE was not programmed using SIMD instruction, thus the computation on the PPE was not able to exploit data parallelism during the arithmetical recombination. For what concerns the DMA transfer results it worth reminding that we did not optimize the data communication at all. Thanks to the 4 way EIB ring, the PPE-SPE communication can be organized using a quadruple buffering strategies that cut down the transmission latency. However this is really a tuning of the algorithm on the specific system and was not our main interest. The new generation of multicore aware programming systems such as RapidMind, can deal and optimize memory transfers when the programmer put information about data locality and usage in the pseudo-code. The system then chooses the best strategies on the base of the target architecture.

Considering the performance of a SPE implementation, it is interesting to know that each SPU has two pipelines. Into these pipelines, the SPU can issue and complete up to two instructions per cycle, one in each of the pipelines. Whether an instruction goes to the even or odd pipeline depends on its instruction type, which is related to the execution unit that performs the function. A good balance of these instructions permits the concurrent execution of two instruction at once. We stated that we are not interested in ILP optimization and in a heavy tuning of the code in

order to exploit these very specific abilities but we are rather interested in finding a good, general programming strategy for porting hardware implementation on multicore architectures. Nevertheless it can be important to understand how the high level approach we used so far impacted the organization of the instruction performed by the compiler. In particular we would like to know where are the stalls (instructions waiting for some dependencies) in the SPE instructions issuing, and if the core remains partially idle for longtime.

IBM provides a tool for the static analysis of the assembler generated by the compiler for the SPEs: the asmvis tool [49]. It is a graphical tool that shows how the assembler generated instructions are issued by the two pipelines, highlighting the stalls that are marked in red. When the pipelines work at their maximum, always issuing a couple of instructions at once, without stalls, the trace displayed by asmvis is composed by steep V shapes towards the center. Analyzing our code with asmvis we noticed that the two pipelines were unbalanced and that the instructions had high latency, also due dependencies requirements. An example is given in Figure 6.3. The consideration arising is that it is possible to obtain more performance from the SPEs.

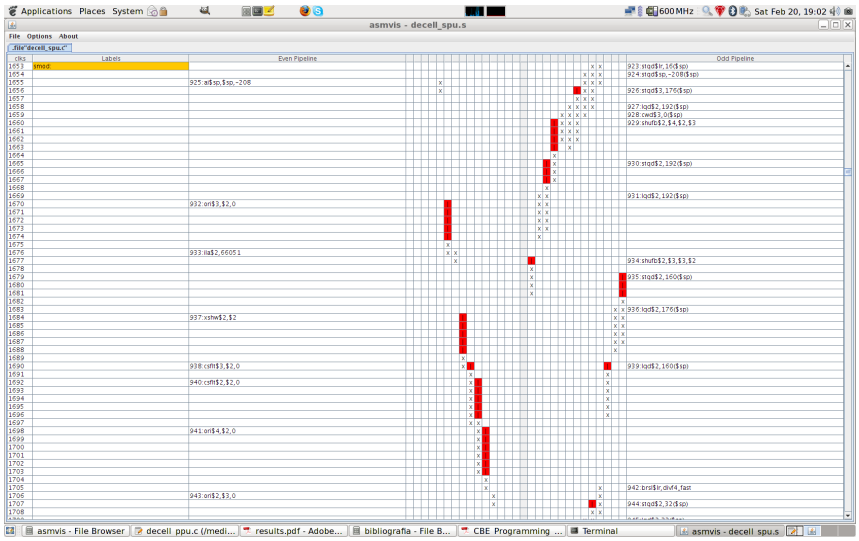


Fig. 6.3. Asmvis results for the residue system conversion function

Symmetric residue conversion

As described in the previous Chapter we used CBE intrinsics in order to improve the performance of our implementation. We substituted the code developed using a standard high level approach with a set of vector operations that mimic a VLSI implementation. The first residue operation we considered was the procedure that

permits to compute the symmetric modulo representation of a given integer. After the coding we used the profiling functions in order to check the difference in terms of performance between the first implementation of the symmetric residue converter and the last procedure borrowed form VLSI. We report the results in Figure 6.4.

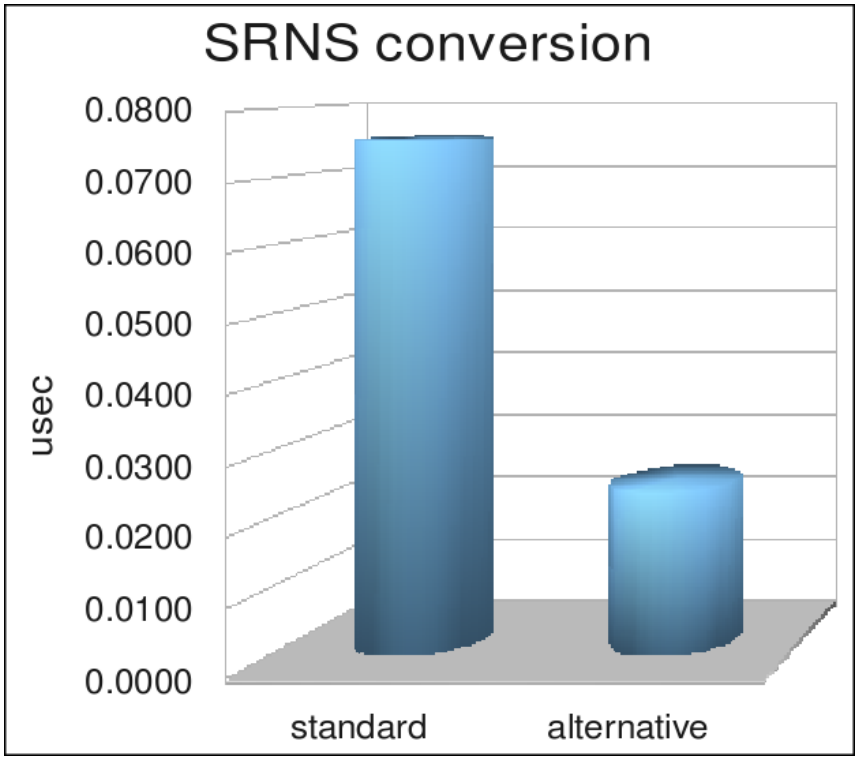


Fig. 6.4. SRNS conversion performance comparison

The computation time required in order to convert a number in its symmetric residue representation lowered from 0.0752 μseconds to 0.0240 μseconds, with an improvement of around 68%.

Residue addition

The second operation considered was the residue addition. After the implementation we collected data about the time required by our first implementation and this new residue adder. The outcome is that the first procedure took 0.0877 μseconds while the alternative version presented here took only 0.0251 μseconds with a gain of about 71%, as depicted in Figure 6.5.

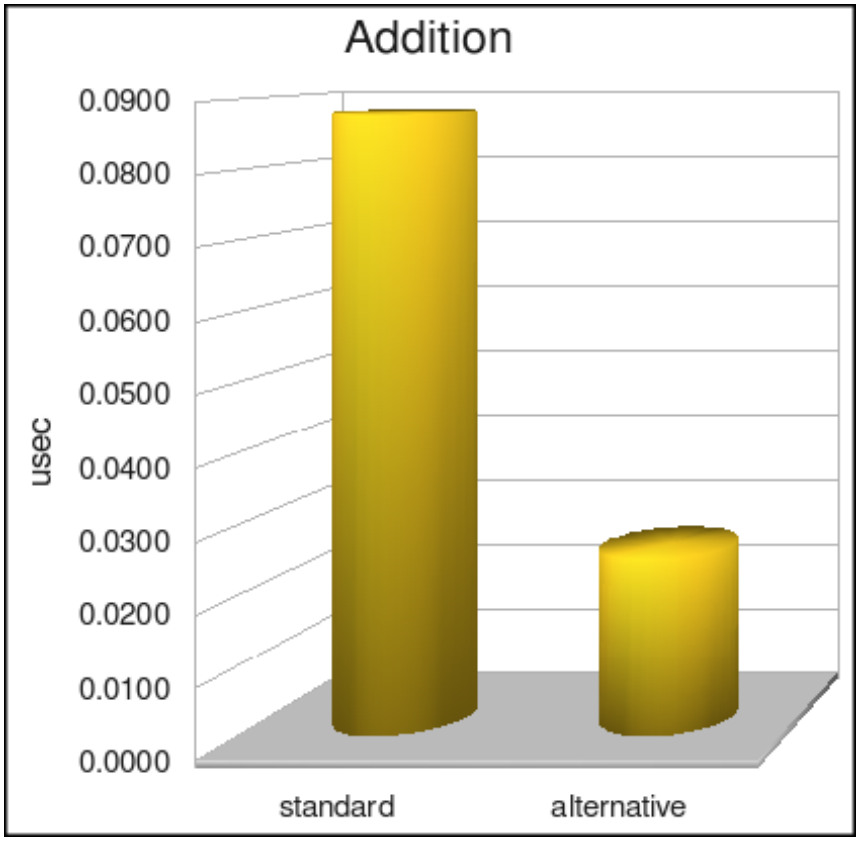


Fig. 6.5. Residue addition performance comparison

Residue multiplication

For our usual performance test we used a 2 vectors array, that is suitable to handle the tables of prime moduli up to the value 11. The results are depicted in Figure 6.6. The first histogram refers to the first implementation we did and the second one to this last coding. The third bar is an alternative solution. We just re-apply the initial idea of performing the multiplication as usual (with the intrinsic `spu_mulo`) and then apply the symmetric residue conversion algorithm in its fastest version. Looking at the values we obtained a computational time of about 0.0877 μseconds for the first approach, 0.0251 μseconds for the second one and approximately the same for the third.

Looking carefully to the last implementation of the residue multiplier we described a consideration arise. This procedure does not use any intrinsic that requires type extension. We stated at the beginning that we chose a 4×4 block matrix even if we use 16-bit precision values that fit in number of eight in the SPE registers. This limitation is mainly due to the use of `spu_mulo` functions that give the result with a 32-bit representation. In the "log transform-residue-addition-antilog transform" function all the operations are carried out on 16-bit values. Thus the

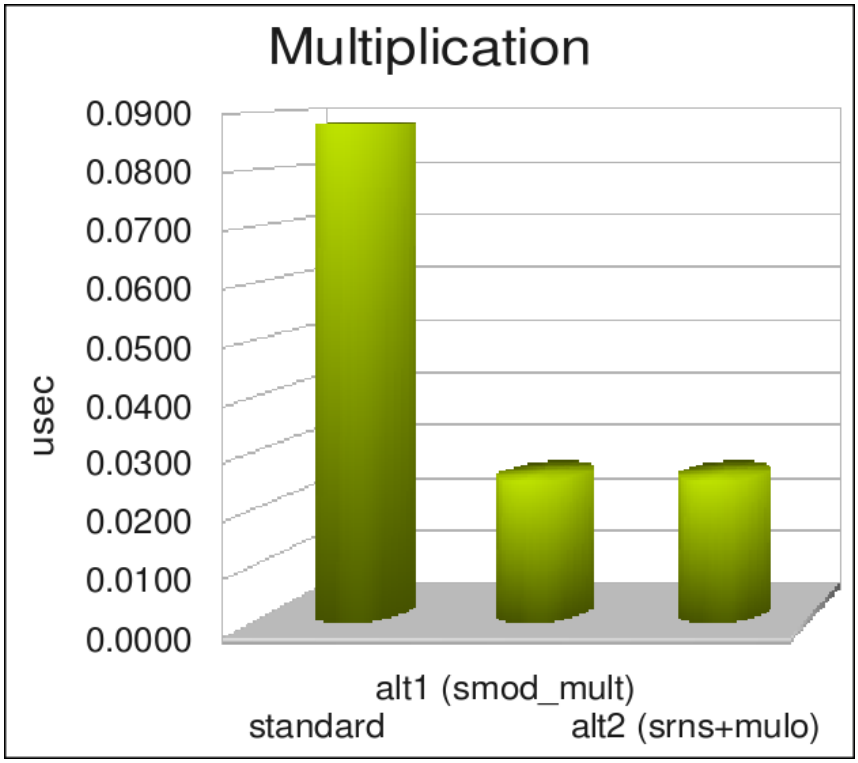


Fig. 6.6. Residue multiplication performance comparison

result we presented does not vary if we use eight 16-bit values in each vector. This means that using this approach we can relate all the timing results we obtained to a 4×8 block matrix, with a subsequent doubling of the performance, without decreasing the dynamic range. This would make our Decell implementation more competitive.

6.1 General results and discussion

In the previous Section we showed that using the available VLSI design as a reference when implementing parallel solutions can be effective in terms of performance without asking the programmer to deal with optimization based on ILP, instruction pipelines and so on: a correct approach can give good results. We tested the speed improvement due to the passage from a high level approach to an hardware mimicking one. However, since we worked always on the worst case scenario due to SIMD organization, we want to check the overall performance of our Decell implementation, comparing then with the early results presented in Figure 6.2. In our test we still considered a 4×4 block matrix.

The results depicted in Figure 6.7 represent the time spent in computation on the specific phases of the Decell algorithm form PPE and each SPE. The same considerations about the PPE code we did about the early results still hold, since no vectorization or optimization has been done. Thus again the most computational intensive part is due to arithmetic recomposition. In the test we used the

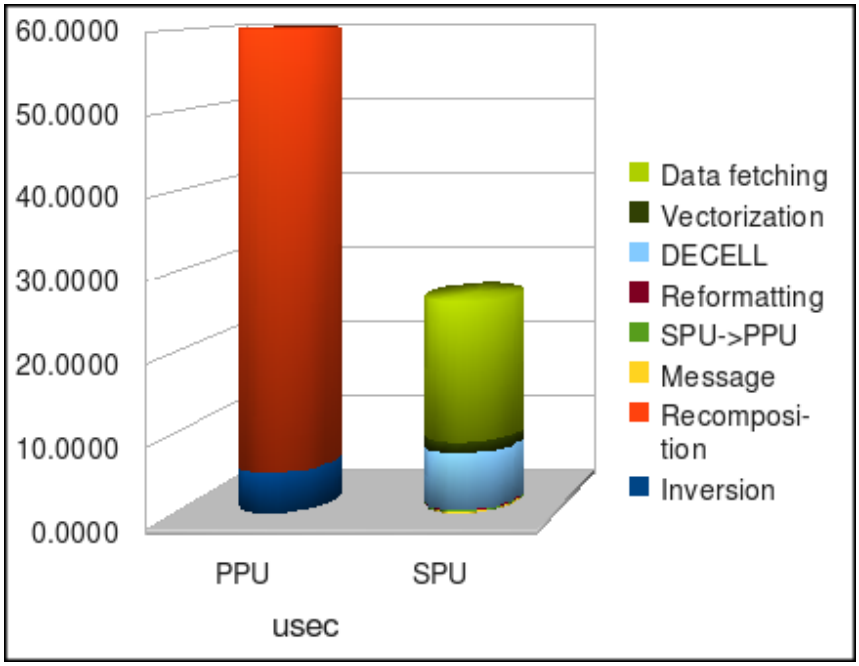


Fig. 6.7. Overall performance of the new implementation divided in phases and cores

CRT approach that is, in this implementation already 20% faster than MRR. The SPE computational time lowered from about 75 μseconds to about 26 μseconds. This last value includes the time required for data (and SPE program) fetching via DMA transfer, all the operations of vectorization and the initialization required by the procedure, the iterations (in light blue in the diagram) and the end message to the PPE. The time spent in the data transfer from SPE to PPE is not considered in the SPE side, but it is part of the PPE recomposition part of the diagram, because when the DMA transfer starts the SPE is idle and can be used, in a pipeline loop, for the processing of the next matrix. Of course if an optimization to the communication part, exploiting the quadruple buffer abilities of the EIB, is made, by hand or by using a multicore aware programming system, it is presumable that the DMA transfer time is masked by the pipelined computation of SPE and PPE.

In Figure 6.8 we just focus on the real execution time in the SPE. There are mainly three parts involved: the initialization of the SPE process once the matrix is stored in the LS (in yellow), the execution of the Decell iteration phase (in red) and the data rearrangement before the DMA transfer of the results (in blue); It is worth noticing that the elaboration per se takes only 7 μseconds when the block

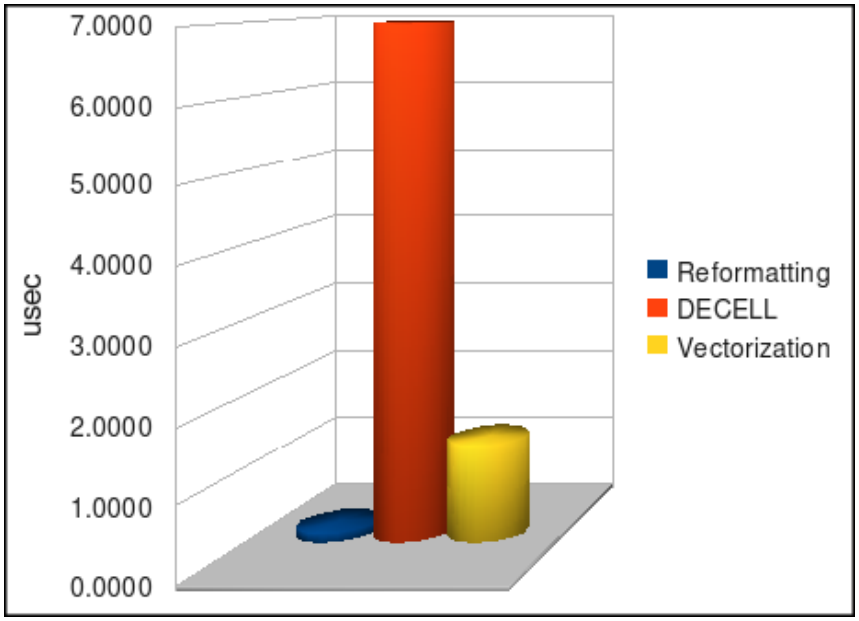


Fig. 6.8. Decell iteration phase timing details

matrix is full rank, thus with four iteration and four control of $A_1 B_i = 0$. The time consumed by the reformatting phase is negligible.

Finally, in Figure 6.9 we wanted to show a fast comparison between the implementation of SVD and Decell on a standard PC and on the CBE. The standard PC was an Intel Pentium M processor at 1.5 GHz. The diagram shows that the Decell sequential implementation, in SRNS, requires a huge computational time. This is a proof that this types of inherently parallel algorithms needs a multicore architecture in order to be again useful. The presence on the market of easily accessible parallel architecture may renew, as we suggest, old solution studied mainly for an hardware development. A straight porting if a SVD implementation on the Cell, on the other side, shows that it is difficult to exploit parallelism from an intrinsically sequential program. The last consideration is that the Decell algorithm for CBE, considering all the improvement margin we described, can be comparable to SVD for PC.

Generic matrix pseudoinversion

We extended the pseudoinversion to generic matrices by using the block matrix multiplication described in Section 5.2.1 and considering the 4×4 matrix as the building block. Our purpose was to test if the Decell algorithm scales well with respect to the size of the matrix considered, compared with the other available solutions. The block matrix multiplication was implemented in the SPEs without any specific optimization. In particular we did not exploit loop unrolling because we used simple nested loops. Of course the core 4×4 matrix multiplication that

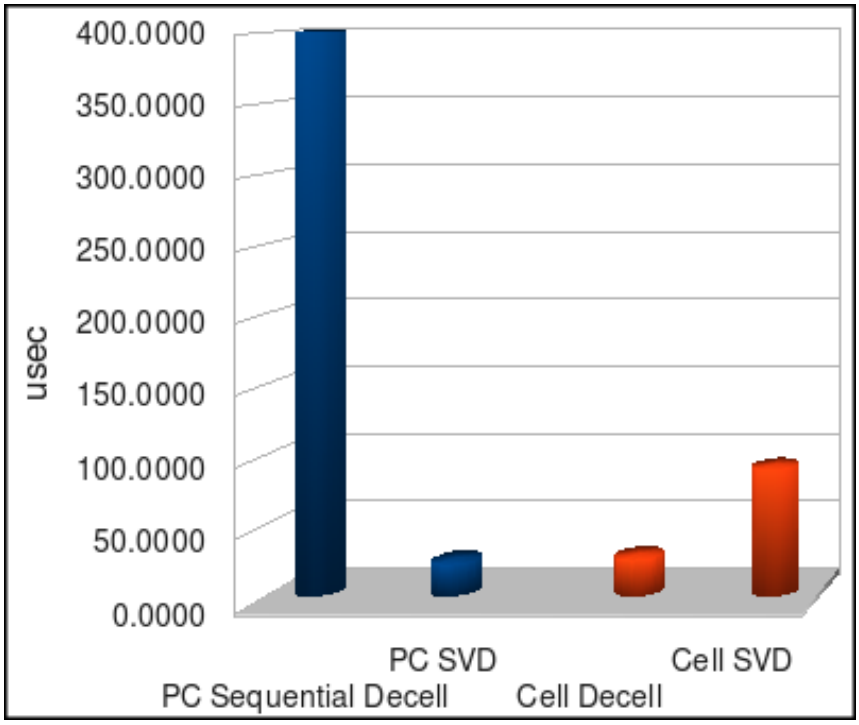


Fig. 6.9. SVD and Decell comparison among PC and CBE

span the entire matrix uses all the techniques described in the previous Section and thus takes advantages from the synergistic processors architecture.

In Figure 6.10 we present the time required by the execution of the SPE code when Decell is applied to different full rank square matrices. We are not considering the time required by DMA data transfers. We took into account the computation of the residue representation of the matrix, that linearly depends on the size of the matrix that has to be pseudoinversed, (in blue) and the execution of the Decell iteration phase (in red). The time spent in the decomposition is negligible when compared with the iteration phase. In general we can notice that the computation time grows exponentially with respect to the size of the matrix. This result was predictable because of the implementation of the block matrix multiplication we chose.

We wanted to compare this result we obtained with the time required by LAPACK and NR algorithms. In Figure 6.11 the values of the application on full rank square matrices of the three algorithms is presented. The solid lines connecting the values in the different sizes (4×4, 8×8, and so on) are just a visual guide rather than a measured interpolation. In order to better understand the intersections, and thus the algorithms better performing, Figure 6.12 depicts the same results with a logarithmic time scale. It is easy to notice that in general the Decell algorithm requires less computational time for matrices up to 32×32. NR is faster than LAPACK for small matrices while LAPACK outperforms the other two methods for

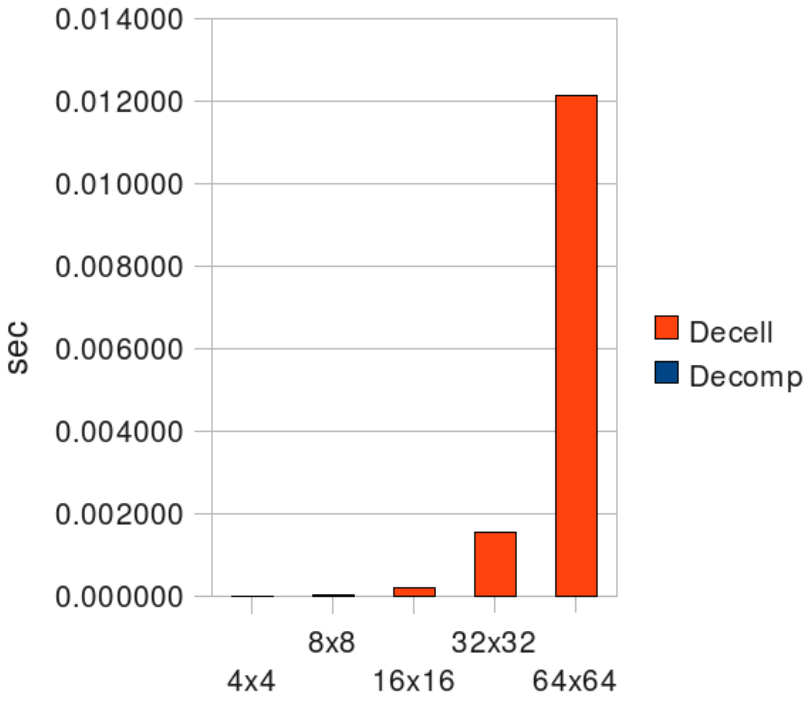


Fig. 6.10. Decell computation time required for the pseudoinversion of matrices of different sizes

a large 64×64 matrix. This was not a surprise since, as we stated above, LAPACK is optimized for large matrices.

Pseudoinversion is necessary in particular when the matrix considered is not invertible. Then we decided to test how the computation time required by the pseudoinversion changes in presence of rectangular matrices. Without loss of generality, we considered matrices where $m < n$, with m the number of rows and n the number of columns. This choice is general since when $n < m$ it is still possible to return this case by applying the relation presented in Section 5.1. For this specific test we compare the performance of Decell and NR algorithms.

We apply both algorithms to a series of matrices by varying the number of columns and rows. The behaviour of NR is depicted in Figure 6.13. Each coloured plot regards matrices that have the same number of rows, while in the abscissa there is the number of columns, and in the y-axis the computation time is depicted. It is possible to notice that the time required in NR computation is directly proportional to both the number of rows and column of the matrix that has to be pseudoinverted.

The results presented in Figure 6.14, that is the performance evaluation of the Decell algorithm, show that the computation time required is directly proportional to the number of rows of the matrix that has to be pseudoinverted, but the number of columns does not affect the overall calculation. The reason of this behaviour

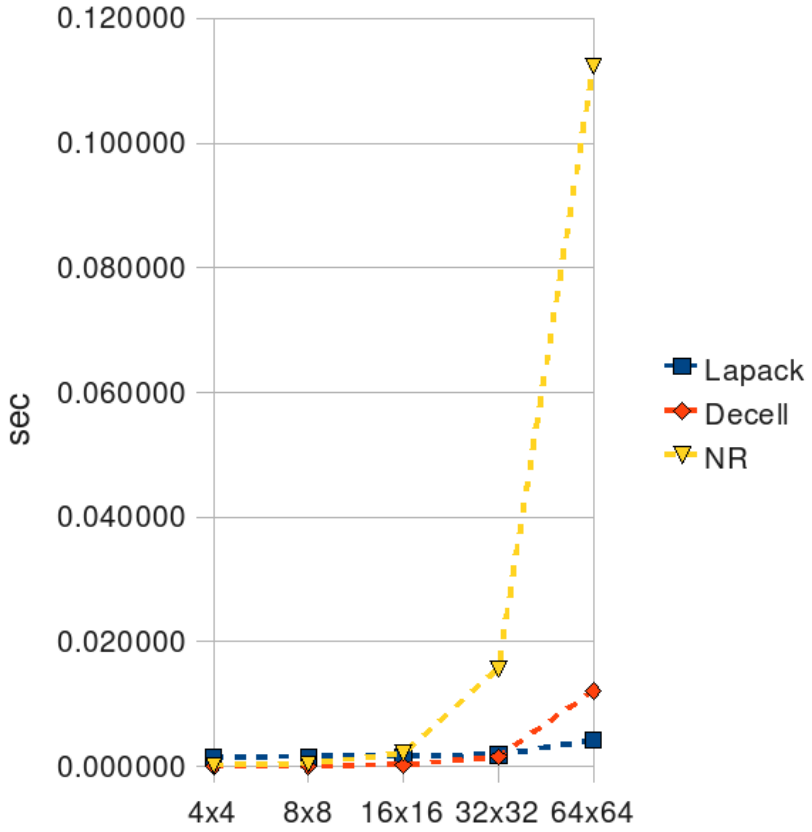


Fig. 6.11. Trend comparison among NR, LAPACK and Decell computation time for square generic matrices.

derives from the first operation that is performed in the Decell iteration phase (see Equation 4.6). In fact, at the beginning of the algorithm the multiplication $A_1 = AA^*$ ($A_1 = AA^T$ when the values of the matrix are real numbers) is performed. Considering that A is an $m \times n$ matrix, whatever is the number of columns n the operation produces an $m \times m$ A_1 matrix. Thus, despite the number of the columns of A, the most part of the algorithms works on a square $m \times m$ matrix, that is A_1. There is of course a performance penalization due to this first multiplication, that is proportional to the size of the matrix A, but its contribution to the overall computation time is not significant and it is evident just for very big matrices.

This aspect of the Decell algorithm is particularly interesting in robotics, since the Jacobian matrix has $6 \times n$ size, with n the number of the degrees of freedom of the manipulator. Thus, even for a high redundant robot the number of rows in the Jacobian matrix is fixed, while the number of rows can be big. Even in this case the computation time of the pseudoinverse is limited. This is another property that makes the Decell algorithm particularly suitable for haptics and robotics applications.

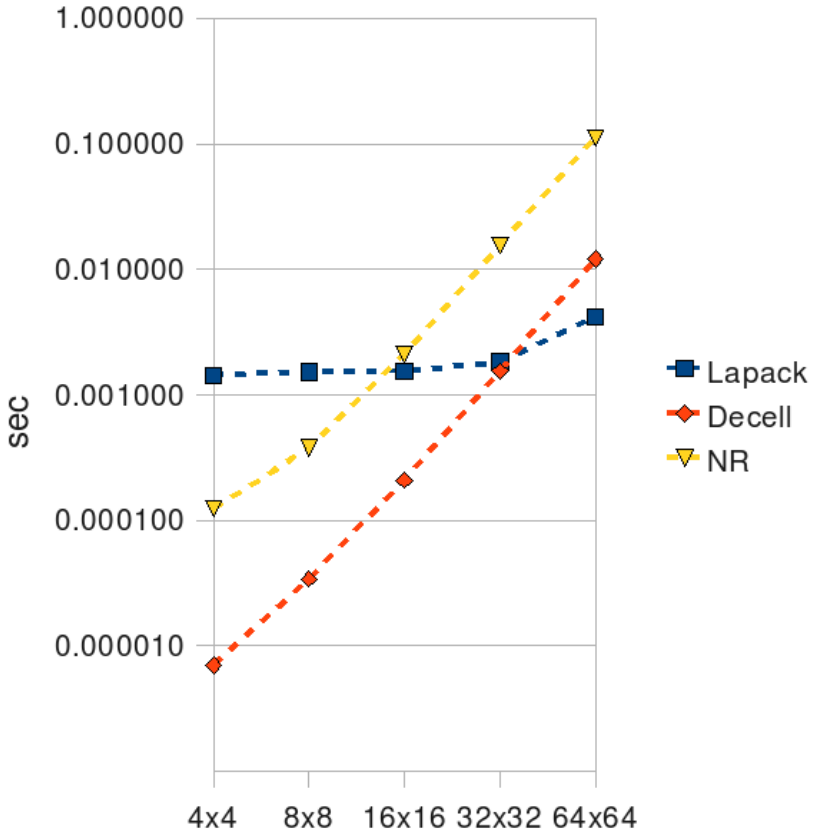


Fig. 6.12. Trend comparison among NR, LAPACK and Decell computation time for square generic matrices with time expressed in a logarithmic scale.

Further results

As utlimate test we used the asmvis tool in order to inspect the new implementation about the pipelines load and the stall existence. We observed that the red marks on the visualization were fewer, thus indicating that the number of instructions waiting for some dependencies decreased significantly (see Figure 6.15). There were almost entire functions in which stalls were absents. Usually this type of result is obtained by a direct optimization of the assembly code, usually made by hand. In [4], for instance, two microkernels developed in order to improve LAPACK matrix multiplication for the CBE were optimized manually. Our approach does not reach the same peak performance of these fine tuned applications, however it permitted a noticeable computation speedup, with an implementation based only on high level instructions.

In addition we noticed that the time required for SPE computation is almost constant. That is, since the synergistic element is designed to be used on a single

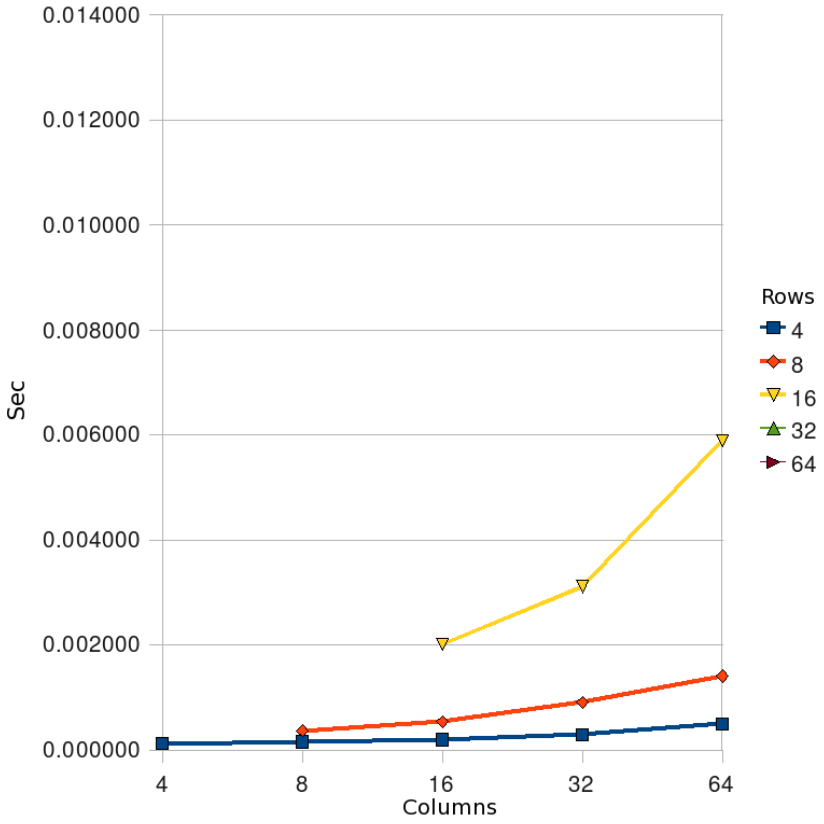


Fig. 6.13. Performance comparison of the NR implementation of pseudoinversion of non square matrices.

computational intense task, without preemption and interrupt, in standard condition it behaves in a very deterministic fashion. This may be useful because if a reasonable deadline is defined the computation can be considered reliable.

We stated that one of the main reason that leads us toward the choice of the Decell algorithm is the intrinsic upper bound limit to the computation of the matrix pseudoinverse, that is important in real-time applications. The same type of determinism is not guaranteed by other solutions for pseudoinversion such as the SVD decomposition.

On the base of the regularity in the code execution time of the SPEs, we verified this assumption by performing a set of tests. We collected the time required by LAPACK and Decell for the pseudoinversion. We repeated the computation on different matrices and different matrix sizes and we compared results. Figure 6.16 shows that when LAPACK is used the computation time required depend heavily on the specific configuration of the matrix. In fact the yellow diamonds represent the different runs of the algorithms on different matrices of the same size. The

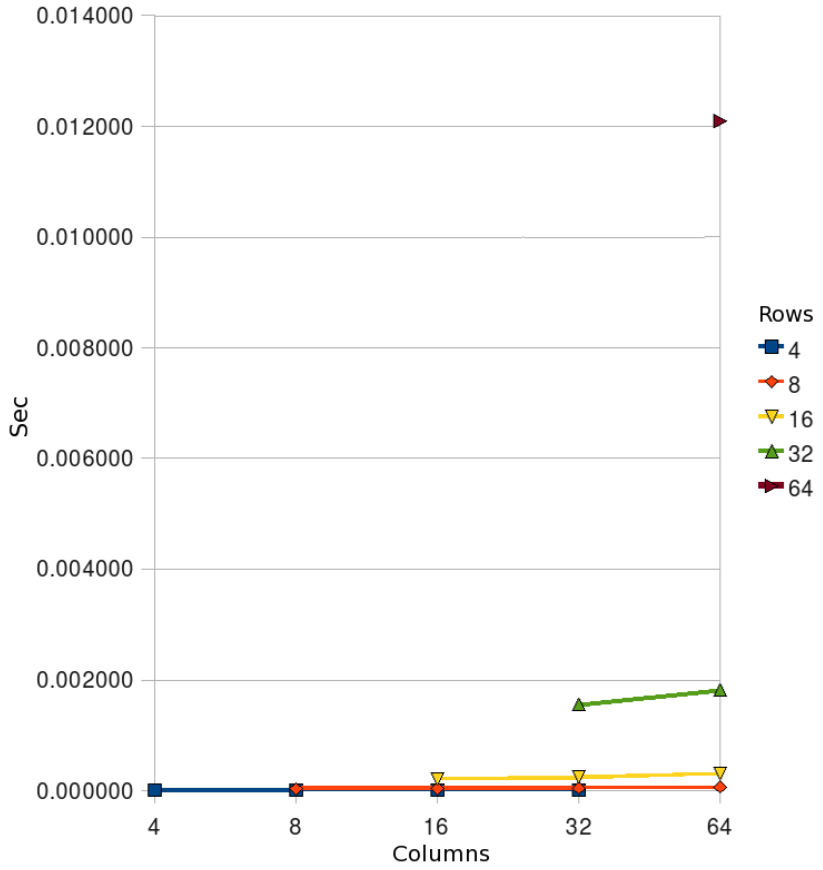


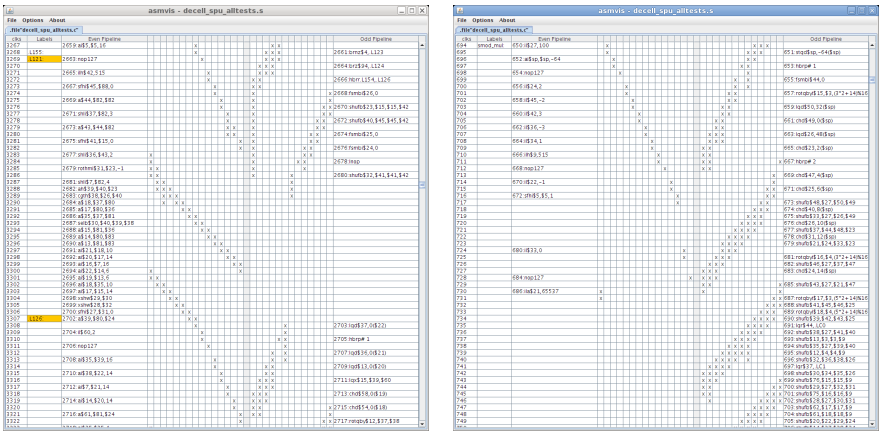
Fig. 6.14. Performance comparison of Decell pseudoinversion of non square matrices.

average time required (azure squares) is quite regular, but there are values that moves away significantly. The very same behaviour is present for the NR algorithm. It is also possible to find configurations that double the computation time, especially when the matrix is rectangular.

The same test was performed with the Decell algorithm and the results are depicted in Figure 6.17. There is only one type of value plotted, since the variability in the time required for the pseudoinversion of random generated matrices is negligible. The computation is performed in a very regular fashion and, if the architecture and the operating system support determinism in the execution of the code, and thus it is easier to identify correct deadlines and control frequencies.

Considerations

Regarding our assumption of staying as general as possible in our approach, in order to consider it useful also on others multicore architectures we would like to highlight that in our hardware inspired implementation of the residue arithmetic we used a limited set of intrinsics, mainly shift, add, comparison and selection.



(a) residue conversion (b) residue multiplication

Fig. 6.15. Asmvis results for the residue conversion and the multiplication functions

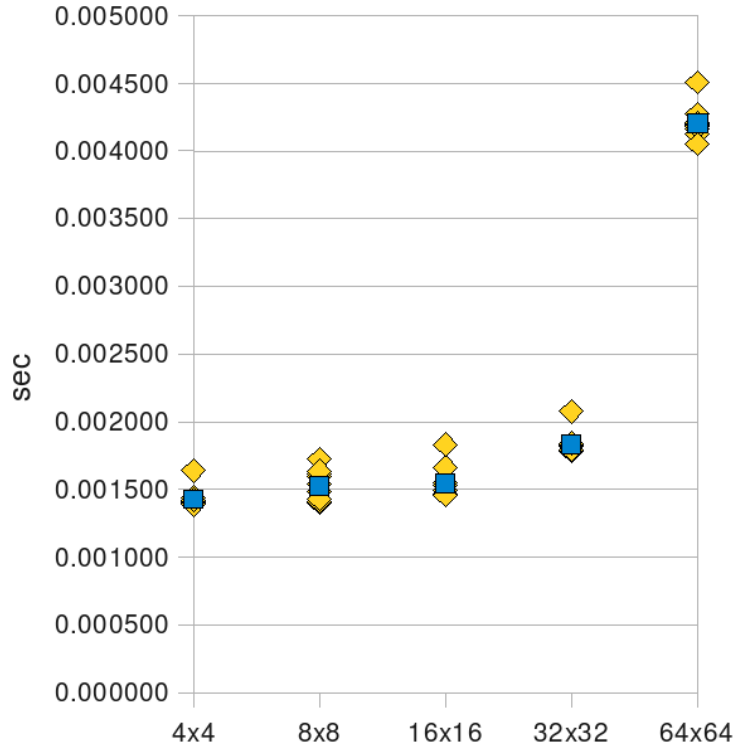


Fig. 6.16. LAPACK computation time for different matrices. The diamonds (yellow) are the result colleted in several runs of the algorithm while the squares (azure) are averages.

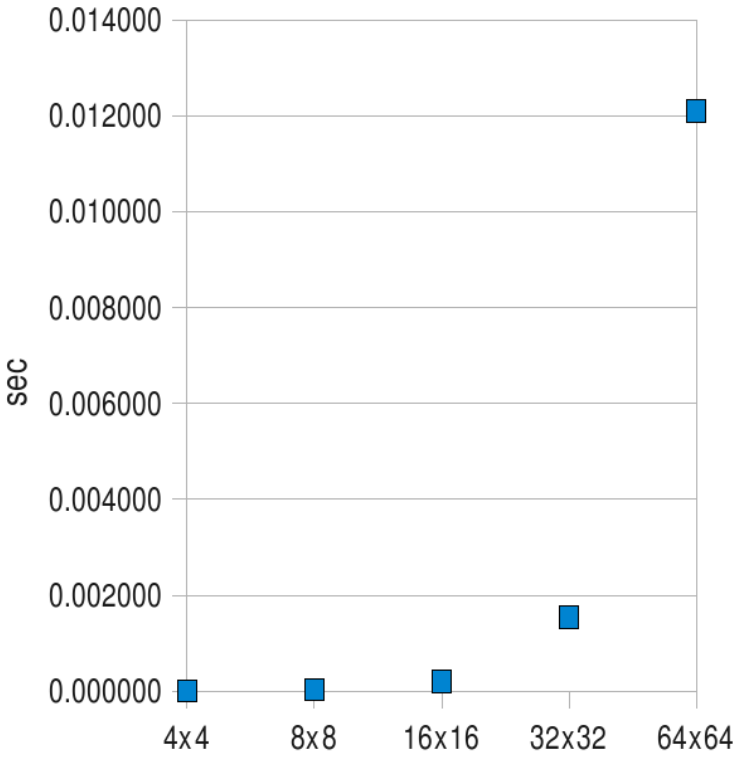


Fig. 6.17. Decell computation time for different matrices. The squares (azure) represent the actual performance in several runs of the algorithm.

This type of vector instructions are usually available on every SIMD or SPMD system, since the registers and the electronics they need is cheap and easy to be developed. Thus it is a reasonable assumption that our implementation can be ported, with the necessary modifications, to other architectures. Moreover some horizontal functions, such as the sum of all the elements of a vector are not available on the Cell but exists in other systems. It has to be noticed that the IBM SDK is constantly update and it is possible to find proposed tentatives of horizontal operation for the Cell in literature, for instance in [72].

The FPGA board is quite different and requires a more drastic redesign, however the basic operations are surely available. In addition we limited our data precision to 16-bit, that is a common standard for FPGAs.

It is of course always possible to use intrinsics and hardware oriented instructions, even on conventional uniprocessor systems and, if the microcode supports the operation used it will end up to a similar speedup. However, the two main points we found are: first, in a system supporting the SIMD paradigm it is easier to find these types of operation acting on data sets, since the SIMD or vector registers are closer to the hardware than general purpose ones. Second, when a hardware implementation that is based only on these types of operations already

exists, it is easier to port the code to a SIMD implementation.

It is important to make a distinction about our performance comparison and results. The aim of our study was not to find the best algorithm for pseudoinversion, nor to state that our parallel implementation is the fastest ever. Hence we are not proposing a tout court replacement for the SVD algorithm. We wanted to show that a specific approach for the porting of hardware implementation on recent multicore architecture is able to give, alone, a considerable speedup, while remaining comparable to standard, state of the art, approaches.

6.2 Conclusions

In this Chapter we presented the data we collected in order to evaluate the performance of our hardware inspired implementation of the Decell algorithm for matrix pseudoinversion. We analyzed the time spent in each phase of the algorithm, both in PPE and SPEs. Then we focused on the synergistic processors, since they are more suitable to exploit data parallelism. In particular we compared the execution time of the different implementations described in Chapter 5. We highlighted the improvement given by the use of CBE intrinsics that can easily mimic the VLSI implementation, while operating on sets of data at a time. In the following we will discuss our conclusions and possible future work.

7

Conclusions

Teleoperation indicates operation of a machine at distance. The standard setup for teleoperation requires a slave device, used to interact with the real or virtual remote environment, a communication link, responsible for data transfers, and a master device. This specific mechanism is used by the operator in order to provide commands to the slave device. In the simplest case it is just a pointing device, used to position and move the telecontrolled tool. Since the user has to "perceive" the remote environment as he/she is acting directly on it, a set of information has to be collected at master side and sent back to the operator. When this set includes force/torque data the teleoperation is said to have force feedback. The force feedback is known to improve the performance of the operator and is currently used for critic applications such as medical simulators and flight simulators for pilot training.

The master device that gives force feedback to the user is also called haptic device. Haptic devices are robots to all intents and purposes, since they have a kinematic structure, they are composed by sensors and actuators, and they are controlled. At the same time they differ from robots since they are constantly in contact with an human operator. They are the interface between the user and the whole teleoperated system and, in addition, they have to provide a convincing feedback. Vision is widely used in teleoperation due to the deep knowledge of physiological and psychophysical characteristics of human visual capabilities and due to the large availability of cameras and displays. The mechanisms of tactile and kinesthetic perception are still fields of research and complex and configurable devices are required in order to carry out experiments.

In this work we studied haptics with a multidisciplinary approach. We focused on two main aspects there are the role of human perception in haptics and teleoperation, and the use of multicore architectures for the implementation of hardware algorithms for dynamics.

In details:

- We designed a setup for the execution of perception experiments by using a non commercial high performance haptic device. The NASA/JPL Force Reflecting

Hand Controller was used to provide force stimuli in the Cartesian space to several subjects and collect results.

- The joystick was equipped with a FPGA board for the low level handling of the device and precise force generation. An important effort was spent in order to make the reflected force/torque signal correct and a complete calibration of the device was required. Then we collaborated in the design of the experiments.
- The analysis of the results leads towards the identification of the force/torque differential thresholds applied to the hand-arm system. On this basis we determined a set of scaling functions, one for each degree of freedom of the three-dimensional space, that can be use to enhance the human abilities in discriminating different stimuli. These variable force scaling operates especially on low intensities forces. We foresee that this can be useful in teleoperation tasks that require high precision and sensitivity, such as computer aided surgical operations.
- Since any real time force signal modification has to fit in the teleoperation and haptic device loops that have to strictly satisfy time constraints we worked in the identification of suitable high performance hardware that can be used in these fields. We focused on the migration of old inherently parallel algorithms implemented in hardware to new multicore architectures.
- We implemented an algorithm for the computation of the pseudoinverse of the Jacobian matrix, that was implemented in VLSI during the 80s successfully on a Cell broadband engine. We studied the programming model of the algorithm and found a match with the processing model of our target architecture. We showed that the implementation on a recent multicore system can be achieved. In particular we highlighted that good results in terms of performance can be obtained by exploiting tasks and data parallelism and by using instructions that can be easily transformed in elementary register operations, such as add, shift, mask and selection. These instructions are usually defined and accessible by high level languages in SIMD architecture, such as the CBE.
- We implemented and evaluated the SRSN Parallel Decell Algorithm for matrix pseudoinversion on a Sony PlayStation 3 console, equipped with CBE and GNU/Linux. We used intrinsic functions in order to easily access the vector registers without dealing with assembler and low level instructions. We let to the compiler the optimization of the code. The result was a consistent increase in performance, with respect to a standard C/C++ approach, even if no specific fine tuning was performed.

Haptics is becoming more accessible and diffuse, thanks to the availability of devices and applications. However, since they are the medium used to increase the immersion experience of the user during teleoperaration tasks, it is important to maintain a human-centric approach to force feedback. In particular it is necessary to deeply understand how the force signal are perceived and decoded by the human being. This understanding can be used to make the remote experience realistic as well as to increase the abilities of the operator. At the same time haptics and robotics have to exploit any actual technology in order to make these goals a reality. The use of multicore architectures and parallel programming is a solution, and it is possible easily to port algorithms and strategies once implemented in

hardware by using the correct programming approach.

7.1 Future work

Since this was a multidisciplinary work, there are several possible evolutions.

From a phychophysical point of view, our setup can be used to gain new insights about human force perception. In this thesis we focused on a specific aspect, that is the identification of the force/torque differential thresholds. This choice was given by a possible application for robotic surgery. However the approach we followed, the design of the experiments and the hardware and software used for this purpose can be easily adapted for other perception studies where the goal is the analysis of the hand-arm system and of its kinesthetic abilities. The choice of new experiments can be driven by the need of task specific enhanced human perception, in surgery as well in other teleoperated environments. An interesting side effect can be the identification of a set of guidelines that can be useful in the design of new haptic devices.

Regarding our results, we would like to design a new experiment in order to confirm that the proposed force scaling is good. As described in Chapter 2, we already checked that the underlying idea works and that our scaling permits the operator to discriminate small differences in the forces at low intensities, but we would like to involve more subjects in order to obtain statistical relevance to the data collected. Moreover the development of a simple and fast procedure for the ad-hoc calibration of the scaling function in order to obtain a "personal" enhancement is under development. At the same time we want to verify the stable variable force scaling strategy in a real teleoperation task.

For what concerns the parallel implementation of algorithms there are two main evolutions possible.

1. It would be interesting to implement the same SRNS parallel Decell algorithm in the other multicore architectures we presented, and in particular in GPU and in a System on Chip equipped with a new generation FPGA. A first analysis can regard the design modifications required by each porting, in order to exactly evaluate the portability of the approach. A second more direct and objective evaluation can refer to the comparison of the execution time required by the program on each architecture. The combination of our approach with the use of a multicore aware SPMD based programming language, such as RapidMind, can be interesting because it can further improve the performance while reducing the code adaptation efforts.
2. In order to evaluate our approach about parallel implementation we focused on the matrix pseudoinversion. Since the main goal is to implement all the control and transformation algorithms that can be used in teleoperation by the haptic interface, our proposal can be further broaden by porting entire kinematic and dynamic algorithms from hardware to software. There are several works about the implementation of both custom and general solutions for the dynamic

and kinematic computation of kinematic structures. In addition the controller currently used in axis boards and industrial robotic systems can be used. All these hard-coded programs can be renewed by using multicore architecture. Eventually the implementation of a complete haptic controller with perception based force signal manipulation can be considered.

The PS3 is an inexpensive alternative in order to check the goodness of an implementation on the Cell Broadband Engine. However the Sony console is somewhat limited for robotic/haptic development since it lacks custom I/O support. In order to couple the PS3 with the motors and sensors of a robotic device a complex electronic modification is required. Although these drawbacks in the future we would like to test the implementation with a real haptic device. The two alternatives are to use PS3 as a dedicated computing part in a distributed system, by using one of the real time architecture for teleoperation we presented in Section 3.2, overcoming all the limitations of an hypervisor filtered network communication, otherwise the use of an IBM Blade server if the investment is affordable.

Considering the interest in high performance error free computation, and its general use in field such as cryptography, a secondary topics can be the creation of a multiplatform library for the symmetric residue arithmetic, working on different SIMD platform. The residue operations can be implemented by using abstract basic add, shift, mask and selection vector operations that are then transformed in the target specific intrinsics.

References

1. Software development kit for multicore acceleration, version 3.0. Programming Tutorial, October 2007.
2. Sonya Allin, Yoky Matsuoka, and Roberta Klatzky. Measuring just noticeable differences for haptic force feedback: Implication for rehabiliatation. In *Haptic Interfaces for Virtual Environment & Teleoperator Systems*, pages 299–303. IEEE Computer Society, 2002.
3. M. Altomonte, D. Zerbato, D. Botturi, and P. Fiorini. Simulation of deformable environment with haptic feedback on gpu. In *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pages 3959–3964, Sept. 2008.
4. Wesley Alvaro, Jakub Kurzak, and Jack Dongarra. Fast and small short vector simd matrix multiplication kernels for the synergistic processing element of the cell processor. In *ICCS '08: Proceedings of the 8th international conference on Computational Science, Part I*, pages 935–944, Berlin, Heidelberg, 2008. Springer-Verlag.
5. AMD. Ctm guide: Thechnical reference manual, 2006.
6. Hyunki Baik, Kue-Hwan Sihn, Yun il Kim, Sehyun Bae, Najeong Han, and Hyo Jung Song. Analysis and parallelization of h.264 decoder on cell broadband engine architecture. In *Signal Processing and Information Technology, 2007 IEEE International Symposium on*, pages 791–795, Dec. 2007.
7. Zachary K. Baker, Maya B. Gokhale, and Justin L. Tripp. Matched filter computation on fpga, cell and gpu. In *FCCM '07: Proceedings of the 15th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 207–218, Washington, DC, USA, 2007. IEEE Computer Society.
8. Federico Barbagli, Ken Salisbury, Cristy Ho, Charles Spence, and Hong Z. Tan. Haptic discrimination of force direction and the influence of visual information. *ACM Transactions on Applied Perception*, 3(2):125–135, 2006.
9. Gabriel Baud-Bovy and Paolo Viviani. Pointing to kinesthetic targets in space. *The Journal of Neuroscience*, 18(4):1528–1545, 1998.
10. A. Bejczy and K. Salisbury. Kinematic coupling between operator and remote manipulator. *Advances in Computer Tecnology*, 1:197–211, 1980.
11. A. Ben-Israel and T.N.E. Greville. *Generalized Inverses: Theory and Applications (2nd edition)*. Springer Verlag, New York, 2002.
12. C. Benthin, I. Wald, M. Scherbaum, and H. Friedrich. Ray tracing on the cell processor. In *Interactive Ray Tracing 2006, IEEE Symposium on*, pages 15–23, Sept. 2006.
13. D. Botturi, S. Galvan, M. Vicentini, and C. Secchi. Perception-centric force scaling function for stable bilateral interaction. In *ICRA'09: Proceedings of the 2009 IEEE*

international conference on Robotics and Automation, pages 230–235, Piscataway, NJ, USA, 2009. IEEE Press.
14. Tracy D. Braun, Renard Ulrey, Anthony A. Maciejewski, and Howard Jay Siegel. Parallel approaches for singular value decomposition as applied to robotic manipulator jacobians. *Int. J. Parallel Program.*, 30(1):1–35, 2002.
15. Bambi R. Brewer, Matthew Fagan, Roberta L. Klatzky, and Yoky Matsuoka. Perceptual limits for a robotic rehabilitation environment using visual feedback distortion. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 13(1):1–11, 2005.
16. E. E. Brodie and Helen E. Ross. Sensorimotor mechanism in weight discrimination. *Perception & Psychophisics*, 36:477–481, 1984.
17. A. Brooks, T. Kaupp, A. Makarenko, A. Orebäck, and S. Williams. Towards component-based robotics. In *IEEE/RSJ International Conference on Intelligent Robot Systems*, August 2005.
18. H. Bruyninckx. Open robot control software: The OROCOS project. In *Proceeding of IEEE International Conference on Robotics and Automation (ICRA)*, pages 2523–2528, May 2001.
19. Grigore Burdea. *Force and touch feedback for Virtual Reality*. John Wiley & Sons, Inc., New York, NY, 1996.
20. K. P. Burnham and D. R. Anderson. Multimodel Inference: Understanding AIC and BIC in Model Selection. *Sociological Methods and Research*, 33(2):261–304, 2004.
21. Alfredo Buttari, Jack Dongarra, and Jakub Kurzak. Limitations of the playstation 3 for high performance cluster computing. Technical report, 2007.
22. F. Caccavale, C. Natale, B. Siciliano, and L. Villani. Achieving a cooperative behavior in a dual-arm robot system via a modular control structure. *Journal of Robotic Systems*, 18(12):691–699, 2001.
23. P.R. Chang and C.S.G. Lee. Residue arithmetic vlsi array architecture for manipulator pseudo-inverse jacobian computation. *Robotics and Automation, IEEE Transactions on*, 5(5):569–582, Oct 1989.
24. Rooju Chokshi, Krzysztof S. Berezowski, Aviral Shrivastava, and Stanislaw J. Piestrak. Exploiting residue number system for power-efficient digital signal processing in embedded processors. In *CASES '09: Proceedings of the 2009 international conference on Compilers, architecture, and synthesis for embedded systems*, pages 19–28, New York, NY, USA, 2009. ACM.
25. NVIDIA Corporation. Cuda. http://www.developer.nvidia.com/cuda/.
26. Frederica Darema, David A. George, V. Alan Norton, and Gregory F. Pfister. A single-program-multiple-data computational model for epex/fortran. *Parallel Computing*, 7(1):11–24, 1988.
27. H. Das, H. Zak, W. S. Kim, A. K. Bejczy, and P. S. Schenker. Operator performance with alternative manual control modes in teleoperation. *Presence*, 1(2):201–218, Spring 1992.
28. Gudrun De Gersem. *Kinaesthetic feedback and enhanced sensitivity in robotic laparoscopic telesurgery*. PhD thesis, Katholieke Universiteit Luven, Belgium, 2005.
29. Gilles Debunne, Mathieu Desbrun, Marie-Paule Cani, and Alan H. Barr. Dynamic real-time deformations using space & time adaptive sampling. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 31–36, New York, NY, USA, 2001. ACM.
30. H. P. Decell, Jr. An alternate form of the generalized inverse of an arbitrary complex matrix. *SIAM Rev.*, 7:356–358, 1965.
31. H. P. Decell, Jr. An application of the Cayley-Hamilton theorem to generalized matrix inversion. *SIAM Rev.*, 7:526–528, 1965.

32. EPFL Ecole Polytechnique Federale de Lausanne. Laboratory for cryptologic algorithms. http://lacal.epfl.ch/page75844.html.
33. Ernest D. Fasse, Neville Hogan, Bruce A. Kay, and Ferdinando A. Mussa-Ivaldi. Haptic interaction with virtual objects: Spatial perception and motor control. *Biological Cybernetics*, 82(1):69–83, 2000.
34. Michael J. Flynn. Some computer organizations and their effectiveness. *IEEE Transactions on Computers*, 21(9):948–960, September 1972.
35. Force Dimension. http://www.forcedimension.com.
36. Forschungszentrum Informatik (FZI). Modular controller architecture. mca2.sourceforge.net.
37. S. Galvan, D. Botturi, and P. Fiorini. FPGA-based controller for haptic devices. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Beijing, China, October 2006.
38. Stefano Galvan, Andrea Castellani, Debora Botturi, and Paolo Fiorini. Advanced teleoperation architecture. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1680–1685, Beijing, China, October 2006. IEEE.
39. G.A. Gescheider. The classical psychophysical methods. In *Psychophysics: the fundamentals*, Lawrence Erlbaum Associates, Mahwah, NY, 45-72, 1997.
40. A. Ghuloum, E. Sprangle, and J. Fang. Flexible parallel programming for tera-scale architectures with ct. Intel White Paper, April 2007.
41. David M. Green. A maximum-likelihood method for estimating thresholds in a yes-no task. *Journal of the Acoustical Society of America*, 93(4):2096–2105, 1993.
42. Xiang Gu and David M. Green. Further studies of a maximum-likelihood yes-no procedure. *Journal of the Acoustical Society of America*, 96(1):93–101, 1994.
43. Kelly S. Hale and Kay M. Stanney. Deriving haptic design guidelines from human physiological, psychophysical, and neurological foundations. *IEEE Computer Graphics and Applications*, 24(2):33–39, 2004.
44. Owen Harrison and John Waldron. Efficient acceleration of asymmetric cryptography on graphics hardware. In *AFRICACRYPT '09: Proceedings of the 2nd International Conference on Cryptology in Africa*, pages 350–367, Berlin, Heidelberg, 2009. Springer-Verlag.
45. V. Hayward and K.E. Maclean. Do it yourself haptics: part i. *Robotics & Automation magazine, IEEE*, 14(4):88–104, Dec. 2007.
46. N. Hogan. Impedance control: An approach to manipulation: Part I – Theory, Part II – Implementation, Part III – Applications. *ASME Journal of Dynamic Systems, Measurment and Control*, 107, 1985.
47. G Hotz. On the playstation 3. http://geohotps3.blogspot.com/.
48. Yildirim Hurmuzlu, Anton Ephanov, and Dan Stoianovici. Effect of a pneumatically driven haptic interface on the perception capabilities of human operators. *Presence*, 7(3):290–307, 1998.
49. International Business Machines Corp. IBM. Asmvis, assembly visualizer for cell broadband engine. http://www.alphaworks.ibm.com/tech/asmvis.
50. Intuitive Surgical Inc. http://www.intuitivesurgical.com.
51. Heidi Johansen-Berg and Timothy E. J. Behrens. *Diffusion MRI: From Quantitative Measurement to In-vivo Neuroanatomy*. Elsevier Science & Technology, 2009.
52. L. A. Jones and I. W. Hunter. A perceptual analysis of viscosity. *Experimental Brain Research*, 94(3):343–351, 1993.
53. Lynette A. Jones and I. W. Hunter. The relation of muscle force and EMG to perceived force in human finger flexors. *European Journal of Applied Physiology*, 50:125–131, 1982.
54. Amanda L. Kaas and Hanneke I. van Mier. Haptic spatial matching in near peripersonal space. *Experimental Brain Research*, 170:403–413, 2006.

55. G Khanna. Playstation 3 gravity grid. http://gravity.phy.umassd.edu/ps3.html.
56. E Kinoshita, H Kosako, and Y Kojima. General division in the symmetric residue number system. *Residue number system arithmetic: modern applications in digital signal processing*, pages 104–112, 1986.
57. Gerhard K. Kraetzschmar, Hans Utz, Stefan Sablatnög, Stefan Enderle, and Günther Palm. Miro - middleware for cooperative robotics. In *RoboCup 2001: Robot Soccer World Cup V*, pages 411–416, London, UK, 2002. Springer-Verlag.
58. Daniel Kubus, Ingo Weidauer, and Friedrich M. Wahl. 1khz is not enough: how to achieve higher update rates with a bilateral teleoperation system based on commercial hardware. In *IROS'09: Proceedings of the 2009 IEEE/RSJ international conference on Intelligent robots and systems*, pages 5107–5114, Piscataway, NJ, USA, 2009. IEEE Press.
59. Jakub Kurzak, Alfredo Buttari, Piotr Luszczek, and Jack Dongarra. The playstation 3 for high-performance scientific computing. *Computing in Science and Engg.*, 10(3):84–87, 2008.
60. Yeu-Pong Lai and Chin-Chen Chang. Parallel computational algorithms for generalized chinese remainder theorem. *Computers and Electrical Engineering*, 29(8):801 – 811, 2003.
61. B.W. Lamacchia and G.R. Redinbo. Rsn digital filtering structures for wafer-scale integration. *IEEE Journal of Selected Area Commun.*, SAC-4(1), Jan 1986.
62. LAPACK. Linear algebra package. http://www.netlib.org/lapack/.
63. C S G Lee and P R Chang. Efficient parallel algorithm for robot inverse dynamics computation. *IEEE Trans. Syst. Man Cybern.*, 16(4):532–542, 1986.
64. C.S.G. Lee and P.R. Chang. Efficient parallel algorithms for robot forward dynamics computation. *Systems, Man and Cybernetics, IEEE Transactions on*, 18(2):238–251, Mar/Apr 1988.
65. Marjorie R. Leek. Adaptive procedures in psychophysical research. *Perception & Psychophisics*, 63(8):1279–1292, 2001.
66. M. C. Lin and D. Manocha. Cutting-edge computing: Using new commodity architectures. *Proceedings of the IEEE*, 96(5):758–760, May 2008.
67. K.E. MacLean and V. Hayward. Do it yourself haptics: Part ii [tutorial]. *Robotics & Automation Magazine, IEEE*, 15(1):104–119, March 2008.
68. Lawrence E. Marks and George A. Gesheider. Psychophysical scaling. In Harold E. Pashler and Stanley S. Stevens, editors, *Stevens' Handbook of Experimental Psychology*, chapter 3, pages 91–138. John Wiley & Sons, Inc., 2002.
69. M. Mc Laughlin, A. Rizzo, Y. Jung, W. Peng, SC Yeh, and W. Zhu. Haptics-enhanced virtual environments for stroke rehabilitation. In *Proceedings on IPSI2005*, Cambridge, MA, 2005.
70. M. D. McCool. Data-parallel programming on the cell be and the gpu using the rapidmind development platform. In *in Proc. GSPx Multicore Applicat. Conf*, Oct.-Nov. 2006.
71. M.D. McCool. Scalable programming models for massively multicore processors. *Proceedings of the IEEE*, 96(5):816–831, April 2008.
72. C.H. Meenderinck and B.H.H. Juurlink. Intra-vector simd instructions for core specialization. In *Proceedings of the IEEE International Conference on Computer Design*, October 2009.
73. G. Metta, P. Fitzpatrick, and L. Natale. Yarp: Yet another robot platform. *International Journal of Advanced Robotics Systems, special issue on Software Development and Integration in Robotics*, 3(1), 2006.
74. Uwe Meyer-Bäse, Antonio García, and Fred Taylor. Implementation of a communications channelizer using fpgas and rns arithmetic. *J. VLSI Signal Process. Syst.*, 28(1/2):115–128, 2001.

75. E. H. Moore. On the reciprocal of the general algebraic matrix. *Bull. Amer. Math. Soc.*, 26:394–395, 1920. (Abstract).
76. Frederic J. Mowle. *A Systematic Approach to Digital Logic Design*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1976.
77. I.A. Nesnas, A. Wright, M. Bajracharya, R. Simmons, T. Estlin, and Won Soo Kim. CLARAty: An architecture for reusable robotic software. In *SPIE Aerosense Conference*, Orlando, Florida, April 2003.
78. A. C. Newberry, M. J. Griffin, and M. Dowson. Driver perception of steering feel. *Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering*, 221(4):405–415, 2007.
79. Roger Newport, Benjamin Rabb, and Stephen R. Jackson. Noninformative vision improves haptic spatial perception. *Current Biology*, 12:1661–1664, 2002.
80. M. Ohara, H. Inoue, Y. Sohda, H. Komatsu, and T. Nakatani. Mpi microtask for programming the cell broadband enginetm processor. *IBM Syst. J.*, 45(1):85–102, 2006.
81. Kunle Olukotun and Lance Hammond. The future of microprocessors. *Queue*, 3(7):26–29, 2005.
82. Amos Omondi and Benjamin Premkumar. *Residue Number Systems: Theory and Implementation*. Imperial College Press, London, UK, UK, 2007.
83. G. Palli, L. Biagiotti, and C. Melchiorri. An open source distributed platform for the control of the puma 560 manipulator. In *9th Real Time Linux Workshop*, Linz, Austria, November 2007.
84. X. D. Pang, H. Z. Tan, and N. I. Durlach. Manual discrimination of force using active finger motion. *Perception & Psychophisics*, 49(6):531–540, 1991.
85. J. L. Patton and F. A. Mussa-Ivaldi. Robot-assisted adaptive training: custom force fields for teaching movement patterns. *IEEE Transactions on Biomedical engineering*, 51(4):636–646, 2004.
86. R. Penrose. A generalized inverse for matrices. *Proc. Cambridge Philos. Soc.*, 51:406–413, 1955.
87. William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical recipes in C (2nd ed.): the art of scientific computing*. Cambridge University Press, New York, NY, USA, 1992.
88. Proctor and William P. Shackleford. Embedded real-time linux for cable robot control. In *ASME Design Engineering Technical Conference and Computers in Engineering Conference*, pages 200–2, 2002.
89. Timothy J. Purcell, Ian Buck, William R. Mark, and Pat Hanrahan. Ray tracing on programmable graphics hardware. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Courses*, page 268, New York, NY, USA, 2005. ACM.
90. T.M. Rao, K. Subramanian, and E.V. Krishnamurthy. Residue arithmetic algorithms for exact computation of g-inverses of matrices. *SIAM J. Numer. Anal*, 13(2):155–171, April 1976.
91. D. W. Rees and N. K. Copeland. Discrimination of differences in mass of weightless objects. WADD Tech Rep. 60-601, Wright-Patterson Air Force Base, Ohio, 1960.
92. Helen E. Ross and Eric E. Brodie. Weber fractions for weight and mass as a function of stimulus intensity. *The Quarterly Journal of Experimental Psychology Section A: Human Experimental Psychology*, 39(1):77–88, 1987.
93. Evren Samur, Fei Wang, Ulrich Spaelter, and Hannes Bleuler. Generic and systematic evaluation of haptic interfaces based on testbeds. In *IEEE Intl. Conf. on Intelligent Robots and Systems*, San Diego, CA, 2007.
94. Christian Schlegel and Robert Wörz. The software framework smartsoft for implementing sensorimotor systems. In *Proceeding of the 1999 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1610–1616, 1999.

95. Erik J. Schlicht and Paul R. Schrater. Impact of coordinate transformation uncertainty on human sensorimotor control. *Journal of Neurophysiology*, 97:4203–4214, 2007.
96. Bertil Schmidt and Douglas Maskell. Workshop on using emerging parallel architectures for computational science. In *ICCS '09: Proceedings of the 9th International Conference on Computational Science*, pages 861–863, Berlin, Heidelberg, 2009. Springer-Verlag.
97. Sintesi SCpA. Orchestra control engine. http://www.orchestracontrol.com.
98. C. Secchi, S. Stramigioli, and C. Fantuzzi. Power scaling in port-hamiltonian based bilateral telemanipulation. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, Edmonton, Canada, August 2005.
99. C. Secchi, S. Stramigioli, and C. Fantuzzi. *Control of Interactive Roboti Interfaces: a port-Hamiltonian Approach*. Springer Tracts in Advanced Robotics. Springer, 2007.
100. Sensable Technology. http://www.sensable.com/.
101. CBEA JSRE Series. Spu c/c++ language extensions version 2.1, October 2005.
102. David B. Skillicorn and Domenico Talia. Models and languages for parallel computation. *ACM Comput. Surv.*, 30(2):123–169, 1998.
103. Michael A Soderstrand, W Kenneth Jenkins, Graham A Jullien, and Fred J Taylor, editors. *Residue number system arithmetic: modern applications in digital signal processing*. IEEE Press, Piscataway, NJ, USA, 1986.
104. M.W. Spong, S Hutchinson, and M Vidyasagar. *Robot Modeling and Control*. John Wiley and Sons, Inc., New York, 2005.
105. Mandayam A. Srinivasan and Cagatay Basdogan. Haptics in virtual environments: taxonomy, research status and challenges. *Computer & Graphics*, 21(4):393–404, 1997.
106. W. T. Stallings and T. L. Boullion. Computation of pseudoinverse matrices using residue arithmetic. *SIAM Rev.*, 14:152–163, 1972.
107. Kay M. Stanney. Realizing the full potential of virtual reality: Human factors issues that could stand in the way. In *Virtual Reality Annual International Symposium*, pages 28–34, Los Alamitos, CA, 1995. IEEE Computer Society.
108. N.S. Szabo and R.I. Tanaka. *Residue Arithmetic and Its Application to Computer Technology*. McGraw-Hill, New York, 1967.
109. Robert Szerwinski and Tim Güneysu. Exploiting the power of gpus for asymmetric cryptography. In *CHES '08: Proceeding sof the 10th international workshop on Cryptographic Hardware and Embedded Systems*, pages 79–99, Berlin, Heidelberg, 2008. Springer-Verlag.
110. Albert Tarantola. Popper, Bayes and the inverse problem. *Nature Physics*, 2(8):492–494, August 2006.
111. F. J. Taylor. Residue arithmetic a tutorial with examples. *Computer*, 17(5):50–62, 1984.
112. R Development Core Team. R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria, 2008.
113. MPB Technologies Inc. (MPBT). http://www.mpb-technologies.ca/.
114. D. Toffin, J. McIntyre, J. Droulez, A. Kemeny, and A. Berthoz. Perception and reproduction of force direction in the horizontal plane. *Journal of Neurophysiology*, 90:3040–3053, 2003.
115. Tadeusz Tomczak. Residue arithmetic in fpga matrices. *Dependability of Computer Systems, International Conference on*, 0:297–305, 2006.
116. R.T. Vaughan, B. Gerkey, and A. Howard. On device abstractions for portable, resuable robot code. In *IEEE/RSJ International Conference on Intelligent Robot Systems*, Las Vegas, Nevada, USA, October 2003.

117. M. Vicentini, S. Galvan, D. Botturi, and P. Fiorini. Evaluation of force and torque magnitude discrimination thresholds on the human hand-arm system. *ACM Transactions on Applied Perception*, 2010. (Accepted - To be published).
118. Marco Vicentini, Maria Carla De Maggio, Debora Botturi, and Paolo Fiorini. Evaluation of directional force threshold through psychophysics experiments. In A. Luciani and C. Cadoz, editors, *Enactive/07. Proc. of the 4th Intl. Conf. on Enactive Interfaces*, pages 297–300, Grenoble, France, 2007. Association ACROE.
119. J. Volder. Binary computation algorithms for coordinate rotation and function generation. Convair report iar-1 148, Aeroelectrics Group, June 1956.
120. Christopher R. Wagner, Nicholas Stylopoulos, Patrick G. Jackson, and Robert D. Howe. The benefit of force feedback in surgery: Examination of blunt dissection. *Presence*, 16(3):252–262, 2007.
121. Heather E. Wheat, Lauren M. Salo, and Antony W. Goodwin. Human ability to scale and discriminate forces typical of those occurring during grasp and manipulation. *The Journal of Neuroscience*, 24(13):3394–3401, 2004.
122. Felix A. Wichmann and N. Jeremy Hill. The psychometric function: I. fitting, sampling and goodness of fit. *Perception & Psychophisics*, 63(8):1293–1313, 2001.
123. Wikipedia. Linear least squares. http://en.wikipedia.org/wiki/Linear_least_squaresm.
124. Wikipedia. Moore-penrose pseudoinverse. http://en.wikipedia.org/wiki/Moore-Penrose_pseudoinverse.
125. Adrianto Wirawan, Chee Keong Kwoh, and Bertil Schmidt. Parallel dna sequence alignment on the cell broadband engine. In *PPAM*, pages 1249–1256, 2007.
126. Wen Wu and Pheng Ann Heng. A hybrid condensed finite element model with gpu acceleration for interactive 3d soft tissue cutting: Research articles. *Comput. Animat. Virtual Worlds*, 15(3-4):219–227, 2004.

Curriculum Vitæ of Stefano Galvan

Department of Computer Science, University of Verona
CV-2, Strada le Grazie, 15, 37134 Verona (Italy)
(+39) 045 8027074 (voice), (+39) 045 8027068 (fax)
E-mail: stef@metropolis.sci.univr.it

Education:

- PhD in Computer Science, University of Verona, Italy (2010). Advisor: Prof. Paolo Fiorini. Dissertation Title: *Perception-motivated parallel algorithms for haptics*. Commission: Prof. Herman Bruyninckx, Prof. Vincent Hayward, Prof. Luigi Palopoli, PhD Cristian Secchi.
- "Cultore della materia" in Robotics and Systems Theory, University of Verona, Italy (2005).
- Laurea in Computer Science, University of Verona, Italy (2005). Thesis title: *Gestione integrata di dispositivi a ritorno di forza*. Graded 110/110, thesis awarded with exceptional credit. In this Thesis we develop an innovative hardware/software structure used to control an actuated 6 dof joystick in a teleoperation task. To increase speed and reliability parts of the kinematic calculation are embedded in the joystick controller. To implement this idea we used an FPGA to handle both the low level tasks and the algorithmic part of the problem.
- Diploma di Maturità Tecnica (56/60) as Ragioniere, Perito Commerciale e Programmatore (1994).

Activities:

(2008) Research assistant at Department of Computer Science, University of Verona (Italy) for the "Xpero - Learning by Experimentation" FP6-IST-29427 European project: human resources coordination, documentation, design and programming.

(2006-2008) Research assistant at Department of Computer Science, University of Verona (Italy) for the PRIN Cofin project in the research unit "Modellazione e resa aptico/visiva di oggetti soffici e deformabili";

(2005-2006) August-March Research assistant at Department of Computer Science, University of Verona (Italy).

(2002) September-December ERASMUS international exchange period at Center of Autonomous Systems, KTH Stockholm (Sweden);

(2001-2008) System administrator at ALTAIR Robotic Laboratory, Department of Computer Science, University of Verona (Italy);

(2001) Developed the robotics laboratory called *ALTAIR - A Laboratory for Teleoperation and Autonomous Intelligent Robots*, University of Verona (Italy);

(1997-2001) Web designer and programmer for e-commerce websites.

Other activities:

P. Fiorini, S. Galvan, L. Giuliari, L. Pighi. *It Takes a Village... to do Science Education*. Workshop at International Conference on Simulation, Modelling and Programming for Autonomous Robots (SIMPAR), Venezia, November 2008.

S. Galvan, L. Bertelli, F. Bovo, P.Fiorini. *Innovative Tools for Education in Mechatronics*. Workshop at 9th International Workshop on Research and Education in Mechatronics (REM), Bergamo, September 2008.

D. Moschini, A. Castellani, S. Galvan, D. Botturi, P. Fiorini. *Advanced Teleoperation Architecture*. Workshop at IEEE International Conference on Robotics and Automation (ICRA), Barcellona, April 2005.

(2005-2008) Scientific paper reviewer for international conferences in the fields of robotics, control theory and programming (ICAR, ICRA, IROS, CARS) and for international Journals (RAS and Mechatronics).

(2004-2008) *Robotics* courses for the TANDEM project in collaboration with high schools at University of Verona: course organization and preparation, gave lectures and laboratory exercises.

(2004-2007) Preparation and supervision of laboratory exercises in the following courses: *System theory*, *Systems and signals*, *Robotics*, *Introduction to control systems*.

Publications:

1. Marco Vicentini, Stefano Galvan, Debora Botturi, and Paolo Fiorini. *Evaluation of force and torque magnitude discrimination thresholds on the human hand-arm system*. ACM Transactions on Applied Perception, 2010. (Accepted - To be published)
2. Debora Botturi, Stefano Galvan, Marco Vicentini, and Cristian Secchi. *Perception-centric force scaling function for stable bilateral interaction*, ICRA 2009 Conference, IEEE International Conference on Robotics and Automation, Kobe, Japan, 12-17 May 2009
3. Davide Zerbato, Stefano Galvan, Paolo Fiorini, *Calibration of mass spring models for organ simulations*, IROS 2007 Conference, IEEE/RSJ International Conference on Intelligent Robots and Systems, San Diego, USA, October 2007
4. Lorenzo Bertelli, Francesco Bovo, Lorenzo Grespan, Stefano Galvan, Paolo Fiorini, *Eddy: an Open Hardware Robot for Education*, AMIRE 2007 Conference, 4th International Symposium on Autonomous Minirobots for Research and Edutainment, Buenos Aires, Argentina, 2-5 October 2007
5. Andrea Castellani, Stefano Galvan, Debora Botturi, Paolo Fiorini, *Advanced Teleoperation Architecture*, in *Software Engineering for Experi-*

mental Robotics, D. Brugali Ed., Springer Tracts on Advanced Robotics (STAR), XXII, Vol. 30, Springer Verlag Publisher 2007, ISBN: 978-3-540-68949-2.

6. Paolo Fiorini, Stefano Galvan, Debora Botturi, *Space teleoperation for the rest of us: the quest of low cost and high reliability software*, ASTRA 2006 Workshop, 9th ESA Workshop on Advanced Space Technologies for Robotics and Automation, Noordwijk, Olanda, November 2006
7. S. Galvan, D. Botturi, P. Fiorini. *FPGA-based Controller for Haptic Devices*, IROS 2006 Conference, IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Beijing, Cina, October 2006.
8. S. Galvan, A. Castellani, D. Botturi, P. Fiorini. *Advanced Teleoperation Architecture*, IROS 2006 Conference, IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Beijing, Cina, October 2006.
9. S. Galvan, D. Botturi, A. Castellani, P. Fiorini, *Innovative Robotics Teaching Using Lego Sets*, ICRA 2006 Conference, Special Session on Robotics Education IEEE International Conference on Robotics and Automation, Orlando, USA, May 2006.
10. S. Galvan, D. Botturi, P. Fiorini, *Perception and Computation in Miniature Surgical Robots*, BioRob 2006 Conference, IEEE/RAS-EMBS International Conference on Biomedical Robotics and Biomechatronics, Pisa, Italy, 20-22 February 2006.

Sommario

Negli ultimi anni l'utilizzo di dispositivi aptici, atti cioè a riprodurre l'interazione fisica con l'ambiente remoto o virtuale, si sta diffondendo in vari ambiti della robotica e dell'informatica, dai videogiochi alla chirurgia robotizzata eseguita in teleoperazione, dai cellulari alla riabilitazione. In questo lavoro di tesi abbiamo voluto considerare nuovi punti di vista sull'argomento, allo scopo di comprendere meglio come riportare l'essere umano, che è l'unico fruitore del ritorno di forza, tattile e di telepresenza, al centro della ricerca sui dispositivi aptici. Allo scopo ci siamo focalizzati su due aspetti: una manipolazione del segnale di forza mutuata dalla percezione umana e l'utilizzo di architetture multicore per l'implementazione di algoritmi aptici e robotici.

Con l'aiuto di un setup sperimentale creato ad hoc e attraverso l'utilizzo di un joystick con ritorno di forza a 6 gradi di libertà, abbiamo progettato degli esperimenti psicofisici atti all'identificazione di soglie differenziali di forze/coppie nel sistema mano-braccio. Sulla base dei risultati ottenuti abbiamo determinato una serie di funzioni di scalatura del segnale di forza, una per ogni grado di libertà, che permettono di aumentare l'abilità umana nel discriminare stimoli differenti.

L'utilizzo di tali funzioni, ad esempio in teleoperazione, richiede la possibilità di variare il segnale di feedback e il controllo del dispositivo sia in relazione al lavoro da svolgere, sia alle peculiari capacità dell'utilizzatore. La gestione del dispositivo deve quindi essere in grado di soddisfare due obbiettivi tendenzialmente in contrasto, e cioè il raggiungimento di alte prestazioni in termini di velocità, stabilità e precisione, abbinato alla flessibilità tipica del software. Una soluzione consiste nell'affidare il controllo del dispositivo ai nuovi sistemi multicore che si stanno sempre più prepotentemente affacciando sul panorama informatico. Per far ciò una serie di algoritmi consolidati deve essere portata su sistemi paralleli. In questo lavoro abbiamo dimostrato che è possibile convertire facilmente vecchi algoritmi già implementati in hardware, e quindi intrinsecamente paralleli. Un punto da definire rimane però quanto costa portare degli algoritmi solitamente descritti in VLSI e schemi in un linguaggio di programmazione ad alto livello. Focalizzando la nostra attenzione su un problema specifico, la pseudoinversione di matrici che è presente in molti algoritmi di dinamica e cinematica, abbiamo mostrato che un'attenta progettazione e decomposizione del problema permette una mappatura diretta sulle unità di calcolo disponibili. In aggiunta, l'uso di parallelismo a livello

di dati su macchine SIMD permette di ottenere buone prestazioni utilizzando semplici operazioni vettoriali come addizioni e shift. Dato che di solito tali istruzioni fanno parte delle implementazioni hardware la migrazione del codice risulta agevole. Abbiamo testato il nostro approccio su una Sony PlayStation 3 equipaggiata con un processore IBM Cell Broadband Engine.