DOI:10.15774/PPKE.ITK.2012.002

SOLVING NON-TOPOGRAPHIC PROBLEMS WITH TOPOGRAPHIC AND SYNCHRONIZATION ALGORITHMS AND ARCHITECTURES

Horváth András

A thesis submitted for the degree of Doctor of Philosophy

Pázmány Péter Catholic University

Faculty of Information Technology

Supervisor:

Miklós Rásonyi Ph.D.

Scientific advisor:

Roska Tamás D.Sc.

Budapest, 2012

DOI:10.15774/PPKE.ITK.2012.002

.

I would like to dedicate this thesis to my loving parents.

DOI:10.15774/PPKE.ITK.2012.002

„As the biggest library if it is in disorder is not as useful as a small but well-arranged one, so you may accumulate a vast amount of knowledge but it will be of far less value than a much smaller amount if you have not thought it over for yourself."
(Arthur Schopenhauer)

DOI:10.15774/PPKE.ITK.2012.002

Acknowledgement

First of all I would like to thank *Miklós Rásonyi* and *Tamás Roska* for their consistent help and guidance in very many ways, for their unbroken enthusiasm and fatherly guidance during my studies.

I thank my older and younger colleagues for their advices and with whom I could discuss all my ideas: *Gábor Tornai, Mihály Radványi, Tamás Fülöp, Tamás Zsedrovits, Miklós Koller, Attila Stubendek, Vilmos Szabó, László Füredi, István Reguly, Csaba Józsa.* It is not so hard to get a Doctoral Degree if you are surrounded with talented, motivated, optimistic, wise people.

I had some really great teachers who should be mentioned here, as well. I am grateful to Professors *Barna Garay* and *László Gerencsér* for their amazing lectures during my Phd studies.

The support of *Péter Pázmány Catholic University* is gratefully acknowledged and I am also thankful to the *University of Leuven* and to the *Politecnico di Torino* where I could spend one-one semester during my studies.

My work would be less without the discussions with *Csaba Rekeczky* and Eutecus Inc who could always assist my attempts and algorithms with practical problems.

I am indebted to *Fernando Corinto*, *Giovanni Pazienza*, *Csaba György* and *Jan D'hooge* for their kind help and advice.

I am especially grateful to *Timi* who has encouraged me and believed in me all the time. I am very grateful to my mother and father and to my whole family who always believed in me and supported me in all possible ways.

DOI:10.15774/PPKE.ITK.2012.002

Kivonat

Az elmúlt években megfigyelhető a többmagos architektúrák előtérbe kerülése. Ezen új eszközökön sajnos gyakran már nem elég a hagyományos eljárások egyszerű használata, új módszerekre, megoldásokra van szükség, mivel az eddigi mód -melynél a processzor órajelét emelték- s az eljárás lépéseinek gyorsítását célozták megváltozott. Jelenleg a processzáló egységek száma -melyek sokszor specializált egységek- emelkedik. Ezáltal felvetve új mérnöki kérdéseket, tervezési szemléleteket, kiemelve a lokalitás precedenciáját, hiszen napjainkban az egységek közti kommunikáció, mind fogyasztásban, mind a számítási sebesség meghatározásában jelentős részét teszi ki a teljes algoritmusnak.

Az így megjelent sok processzoros architektúrák rengeteg esetben megmutatták már hatékonyságukat, használhatóságukat. Számos topografikus eljárásban (legnagyobb részben a képfeldolgozás terén) tapasztalhattuk hatékonyságukat s újszerűségüket, melyek mind a lokalitás precedenciájának, elosztott s lokális kommunikációra épülő számításoknak köszönhető.

A jelenlegi kihívások egyik legfontosabbja azonban nem az, hogy miképpen tudjuk a már meglévő topografikus módszereket még hatékonyabb architektúrákon, optimális körülmények között végrehajtani, hanem, hogy felismerjük azon problémákat és lehetőségeket, melyek módosíthatóak, transzformálhatóak egy topografikus problémává, s ezáltal könnyedén implementálhatóak egy ilyen újszerű architektúrán.

Dolgozatomban szeretném megmutatni, hogy egy véletlen minta elemeinek kiválasztása hogyan befolyásolhatja különböző algoritmusok hatékonyságát. Valamint az algoritmusoktól eltekintve vizsgálnám két különböző szelekciós-mechanizmus a globális és lokális szelekció által eredményezett mintasorozatok minőségét, változatosságát.

Ezt egyszerűsített modelleken keresztül hajtanám végre, mellyel igazolom a lokális szelekció használhatóságát, s belátom, hogy ezen módszerrel helyettesíthető az algoritmusok egy adott csoportjában a globális szelekció, valamint a lokális mintavételezés további előnyös tulajdonságait is igazolom ezen modellben.

Dolgozatomban megpróbáltam általános szemszögből, a problémák reprezentációjától függetlenül megközelíteni a sztochasztikus optimalizációt. Az optimalizáció kulcskérdésének, s dolgozatom központi lépésének a szelekciót tekintettem.

Bevezettem a biológia által inspirált, lokális szelekció fogalmát, s összevetettem a hagyományosan használt globális eljárással. Az összevetést általánosnak tekinthető mod-

DOI:10.15774/PPKE.ITK.2012.002

elleken végeztem el, majd ezután igyekeztem ezen modellek apró módosításaival közelebb kerülni néhány gyakorlati problémához.

Mutattam két (egy statikus és egy dinamikus) nem-topografikus algoritmust melyek esetében a topografikus szemlélet jobban használható, melyet szimulációkkal is igazoltam.

Szimulációimat egy általam implementált celluláris sokprocesszoros virtuális architektúrán hajtottam végre.

A statikus eljárás, a genetikus algoritmus vizsgálatában három általános probléma: az utazó ügynök, a hátizsák és az N-királynő probléma esetében is megmutattam, hogy (bizonyos paraméterek esetében) a lokális mintavételezéssel gyorsabb konvergencia sebességgel kaphatunk eredményt, mint a globális eljárásnál.

A szimulációkat a gyakorlatban is használt, $Xenon_v3$ architektúrán is implementáltam, mely kihasználja az általam leírt topografikus módszerek egyik legnagyobb előnyét párhuzamosíthatóságukat és skálázhatóságukat.

A dinamikus probléma esetében Rejtett Markov Modellek állapotbecslését vizsgáltam. A probléma azért nehéz, mivel nem egy ismeretlen állapotot (optimumot) szeretnék becsülni, lehető legjobban megközelíteni, mint a sztochasztikus optimalizációban általában, hanem a markovi modell rejtett állapotait szeretnénk meghatározni, vagyis egy dinamikus, időben változó állapotsorozatot, trajektóriát szeretnék megbecsülni nemlineáris megfigyelések alapján. Ezen probléma esetében is három esettanulmányon keresztül mutattam be az eljárás használhatóságát, hatékonyságát. Valamint itt is mind a szimulációs eredmények, mind a konkrét implementáció a $Xenon_v3$ architektúrán megtalálható.

Dolgozatom második felében spinoszcillátorok szinkronizációjával foglalkoztam, valamint megpróbáltam ezen szinkronizációs jelenségeket a számítási képességek szemszögéből is megközelíteni. Készítettem egy általános szimulátort mellyel különböző típusú spinoszcillátorok tetszőleges hálózata szimulálható. Analitikus megoldást adtam az 'in-plane' oszcillációval rendelkező spinoszcillátorok differenciálegyenletének megoldására. Valamint a harmonic balance és discribing function módszerek alkalmazásával megmutattam, hogyan számítható ki egy ilyen elemekből készített tetszőleges hálózatban a különböző oszcillátorok közötti fázisszög. Ezáltal egy leképezést adtam, hogy miképpen alakítja át egy STO hálózat a bemenő, frekvencia-kódolt jelet a kimeneti, fáziskódolt jellé. Továbbá megmutattam,hogy ezen fázisszög, hogyan függ a csatolás erősségétől és a bemenő áramtól két oszcillátor esetén.

DOI:10.15774/PPKE.ITK.2012.002

Abstract

In this thesis I would like to examine how local topographic algorithms can be implemented and used in practical problems. Nowadays in engineering one of the most challenging task is the design of topographic algorithms that can be executed on topographic architectures.

As we have seen in the previous years the speed of Moore's law is further decreasing and the operating speed of processor has not increased significantly either. However the number of transistors that can be manufactured on a silicon wafer is increasing further and further. This creates a gap between high level/abstract algorithms and the many-core architectures. We can not yield for higher processing speed has decreased but we can have more parallel cores instead. Because of this phenomena in parallel algorithms on many-core architectures the wire delay became the most determining factor instead of the gate delay. The transfer of data between the cores can decrease the execution speed significantly. To avoid this we have to process all the data locally, and avoid global communication as much as possible. The only communication which is affordable in a fast, efficient way is the local topographic data exchange.

It is extremely complicated and difficult to examine all the algorithms from this point of view. The general theoretical investigation how general algorithms can be implemented on many-core architectures is out of the scope of this dissertation. Because of the theoretical complexity of many-core implementations I have selected only one type of algorithm: selection mechanism in stochastic processes and examined how they can be implemented in an efficient topographic way.

There are numerous different algorithms amongst the processes used in stochastic optimization. I also have to note that sometimes not only the problem representation, but also the algorithm itself depends on the problem (especially in case of different heuristic improvements). But we can also see, that in almost every stochastic optimization tasks there are common steps.

I have tried to see and describe these algorithms from a point of view which is relatively independent from the problems itself. One of the key steps in this processes is the selection step, when we have to generate a new sample from our current population. I have introduced and examined the biologically inspired selection, and compared these methods (in case of some problems) to some other algorithms containing global selection.

DOI:10.15774/PPKE.ITK.2012.002

I have tried to start these comparisons from a meta-level, where the program representation is not important, to show that the selection has the same properties in general. I have shown that in general the local selection can be considered as a generalized version of the global selection. I have zoomed from this general problem to the more specific problems, to show, that these can be used also not only in theory but also in practical problems.

Later I will show two sample algorithms (widely used in practical problems, which are originally non-topographic) in which case the topographic, cellular way of thinking works better, than the general methods. I will also try to underline this theory by different simulations.

These simulations were implemented on a virtual many core architecture.

The static algorithm (the genetic algorithm) was tested on three different problems on the knapsack, N-queen and on the travelling salesman problem.

I have tested my version of the algorithm not only by simulations but I have also implemented them on an existing chip on the $Xenon_v3$ architecture, which exploits the main characteristics of modern architectures namely the parallel procession, many core execution, scalability and local, cellular connections between the cores.

In case of the dynamic version of the algorithm I describe how the topographic method can be used for state estimation in case of Hidden Markov Models. This problem is more complex than the previously described optimization, because our aim is not to find the best parameters with numerous number of iterations, but to find a hidden state in every iteration, to identify a trajectory of a process. This means that we have only a limited number of steps for processing (usually only one) to identify a dynamically changing state.

Also in this case I have shown through three case studies how this algorithm can be implemented and used, and I have also examined the efficiency of the implementations. Also in this case I have examined the efficiency based on the simulations with the virtual cellular machine, and also on the existing architecture, the $Xenon_v3$ chip.

In the second part of my thesis I have examined how a cellular architecture can be realized by spin torque oscillators. In this architecture all the computation is performed by the physics of the oscillators and the interactions between the neighboring oscillators. Other different interactions are unfeasible, because of the underlying physics. The results a cellular architecture. I have investigated the synchronization of these oscillators, and how they can be used for computation, where the information is not the charge (as it is in the devices used nowadays), but the phase-shift between the neighboring, synchronized oscillators. This results a non boolean, nanoscale device.

In the second part of this thesis I will show an architecture, which is inherently topographic. This processor is made of Spin torque oscillators. The information exchange, interaction between these oscillators happens through the magnetic field. Because of this interaction only a cellular locally connected architecture is feasible.

To implement a processor we can not avoid to understand the behavior of spin torque

DOI:10.15774/PPKE.ITK.2012.002

oscillators. In the second part of this thesis I will describe how we can understand and simplify the synchronization of weakly coupled oscillator networks.

I have implemented a simulator in C, Python and Matlab. With this program I managed to investigate the behavior of the STO arrays in general. I have also calculated the equilibrium of an STO with a closed formula and this way the behavior can be calculated, without solving the differential equations.

Using the harmonic balance and the describing function technique I have shown, how the behavior of the synchronized oscillation can be calculated in any arbitrary array. Apart form the transient behavior, any phase shift, frequency and spin position can be examined in any arbitrary array regardless the boundary condition, initial condition, or coupling weights between the elements in the array.

I have also investigated more detailedly the case of two coupled oscillators. I have calculated how the phase shift between the two synchronized oscillator depends on the input current on the oscillators and on the coupling weight between the oscillators.

DOI:10.15774/PPKE.ITK.2012.002

Contents

DOI:10.15774/PPKE.ITK.2012.002

DOI:10.15774/PPKE.ITK.2012.002

List of Figures

DOI:10.15774/PPKE.ITK.2012.002

List of Tables

DOI:10.15774/PPKE.ITK.2012.002

Chapter 1

Introduction

One of the most interesting trend in computer engineering is how multi-parallel architectures have been brought into focus in the previous decade. Moore's law – implying that the clock frequency of a processor can be doubled in every second year – is slowing down continuously. Without increasing the clock speed, the obvious method to increase our computational power is to multiply the number of the processing elements. If inventing smarter or faster workers in a task proves to be infeasible, we are still capable of increasing the sheer number of workers. Nowadays we are creating architectures specialized for different tasks and further increasing the number of processing elements.

However, to exploit all the advantages of many-core devices it is not enough to use our old algorithms designed for single core chips, we have to change the paradigm of centralized processing. We need new methods and new algorithms. After a certain threshold the number of processing elements may be more than enough, but we still have to divide the work amongst them optimally. The optimization of such a task raises new questions in engineering, new points of view in algorithm design. These designs on many-core architectures underline the precedence of locality. These new topographic, many-core, cellular devices have shown their usability and effectiveness in various tasks. From these solutions we can see, that they can give new, effective solutions in case of topographic tasks (i.e image processing), thanks to the precedence of the locality, and the division of the operations based on local communication. Even on the most recent FPGAs data sharing and the communication between the elements could consume more time than the actual "processing". Communication considering time and power consumption became one of the main elements of an algorithm. Until this point this question was not investigated in a detailed way, although this is one of the most important engineering problem of the following decade.

The most important challenge is not further improving these existing methods on even better architectures [1] , but to identify problem classes, tasks and possibilities when the problems can be transformed into topographic algorithms and mapped on these cellular,

[1] however this is still a very challenging task of optimization

DOI:10.15774/PPKE.ITK.2012.002

many-core devices.

Unfortunately, the question of how to implement a proper algorithm on a many core architecture is too general and can not be answered. In this dissertation I will investigate a set of algorithms, Sequential Monte Carlo methods, and especially one, very important step of such processes the stochastic local selection of elements. I will show the effectiveness of randomly selected local elements and how they can affect the algorithms. I will investigate this selection independently from the problem representations and compare the selected sets with local and global selection regardless of the problems. I will do this through a general, simplified model, which is based only on the local selection of elements. With this model I will show that the local selection can be considered as a generalized version of the global selection and also a generalization of those methods where there is no selection at all.

Later in this dissertation I will show a few examples and algorithms how the local selection can be used on many-core chips. I will investigate the problem also in case of a dynamic environment, where the optimal state changes in time [2]. I will show, how this idea can be used in case of genetic algorithms and in particle filters.

With the genetic algorithm I will investigate three practical problems: the knapsack, N-queen and the travelling salesman problem. I will investigate the particle filter problem through commonly used benchmark models with one and three dimensional state spaces.

According to our current knowledge in physics after a certain point Moore's law and the downscaling of processors have to stop and they are hindered by the physical constraints in the atomic scale. It is unfeasible to build transistors using only a few atoms. In the second part of this dissertation I will investigate one possible architecture, considering in general what main criteria we have to fulfill beyond Moore's law and how a functioning device can be implemented, where the processing elements are single atoms. I will give an example of this device: an array of weakly coupled spin-torque oscillators and I will show some examples of the capabilities of this network.

All the mentioned algorithms, figures and also a digital version of this document can be found on the attached CD.

[2] state estimation of dynamic processes

DOI:10.15774/PPKE.ITK.2012.002

Chapter 2

Local and Global Selection Mechanism

2.1 Random Sampling and Locality in Stochastic Optimization

It is an interesting phenomenon, how limitless the freedom of thinking is. Maybe this could be the biggest advantage and present of our life, that our imagination allows us to create anything without time, space or any physical constraints. We can not feel the difference between imagining 10 or one billion state variables.

However, when we have to implement an algorithm we have to obey physical rules, constraints, because the functionality of our device is inherently bounded in time and space. With only one processing unit we only have to solve the problem of time, the space itself is not important [1].

In contrast, on a many-core architecture, even unwillingly, we have to consider how the processing units are placed and connected. We have to consider the well-defined place of implementation, we have to map our method onto a two dimensional wafer [2] and this implies the precedence of locality and in the end – even if we have not considered this before – our algorithm will be topographic, because it is bounded in a two dimensional state space, where only local connections are allowed and efficient. [3]

We can also observe locality and physical rules in nature. Every living creature is bounded to its territory and they apply inherently the precedence of locality. We can see that, in case of evolution, the improving of different generations depends on space, different entities at different places found different obstacles to overcome and find different mates to create new entities and generations. However, in most cases [4] the population itself is

[1] the placement of one element is indifferent

[2] We are not considering vertical integration for simplicity

[3] global connections are feasible as series of local connections

[4] not considering islands or physically separated animals, which would lead to island model genetic algorithms (see [11])

DOI:10.15774/PPKE.ITK.2012.002

global. During the evolution of populations the information [5] can spread out to every individual, but the selection and the comparison of the entities happens always locally. This is quite similar to our architecture regarding that they are both bounded in space.

It is an interesting task in biology to compare two different species with different mating customs, whether salmons or catfishes [6] have a better and more resistant gene-pool for environmental changes. These are the main idea, why I have decided to investigate local stochastic selection on many-core architectures.

One of the most commonly used operations in stochastic optimization is the selection of a set of elements from our observed data that will [7] represent and preserve correctly some important properties of the full cohort. This operation is a key step in almost every stochastic algorithm that determines the set of elements our process will work with.

This sampling is random, because our models, observations, or our estimations are stochastic. It is important to decide how we can select the best elements randomly from a general set to ensure that the distribution of this element would be the 'best' for us in some sense. Of course the definition of the 'best set' itself is especially difficult and a very interesting question, however, it is impossible to investigate this question in full depth in the present work.

This question will be even more interesting when it is applied to many-core architectures, namely: how we can select the best set, meanwhile we would like to distribute our effort amongst many elements uniformly. This problem is interesting in theory, but since many-core architectures [8] have been brought to focus in the previous decade it has also a large impact on the efficient solution of practical problems.

The problem in this form itself is too general and can not be investigated, because the algorithm [9] depends on the problem representation and on the problem itself. For this reason I have decided to avoid specifying the problem in this chapter and I will rather create a general model that considers only the main, crucial steps of the algorithm and those principles that are common in every solution, but also specific enough to contain random set of elements and the selection of entities based on certain principles.

The idea of local selection itself was motivated by biological selection. I also wanted to investigate, how biologically inspired computing [10] can help us in case of stochastic optimization.

In many areas and applications we can witness how biology inspired algorithmic principles can outperform other rivals. We can also see the wonders of biology: how mammals and insects can solve difficult tasks in a simply brilliant way, that can not be solved by

[5] in this case a good genome or gene

[6] Salmons mate at the same place every year and the population gathers globally, meanwhile catfishes select mating partners in their territory

[7] hopefully and according some heuristics or a priori information

[8] FPGAs, GPUs, CNNs

[9] or the optimized version of the algorithm

[10] the local selection and evolution of individuals

DOI:10.15774/PPKE.ITK.2012.002

thousands of engineers. In this section I would like to show how a simple example, what can be observed in real life: evolution can inspire us during algorithm development (especially in case of stochastic optimization) to build proper methods for many-core architectures.

In the literature we can find many articles examining the properties of global and local selection from different aspects, however, these questions are raised by biologists and philosophers, see [12]. [11]. In those papers where we can identify the aspect of engineering points of view, usually well specified, but special problems are examined based on this idea ([13], [14]) or sometimes some small classes or subsets of problems [15] without considering generalization or architectural planning or implementations.

Since the literature of this area is not comprehensive and the published papers are focusing on well-specified problems and not on algorithmic principles I have tried to find a general framework of the problem. Going against the tendencies I have not selected a specified problem to try to make general conclusions later on but I have tried to create a general model of the problem, to formulate general statements and later on I have tried to specialize, narrow down this model to practical problems without changing the main steps of the algorithm. Keeping the algorithm in focus during the whole time and handling the problem itself only as a tool, which is necessary to test our algorithm.

From an engineering point of view this comparison raises many questions. Which method is more effective? In what sense can we measure efficiency? Which method is faster, considering the number of iterations, or considering milliseconds? How we can divide the processing optimally amongst the processing units? Is it better to handle all entities of our sample together and we are creating a global order or applying a limit for every processing unit we store only a subset of the sample, and only local interactions are possible between the elements?

To investigate local constraints I have placed our processing elements in a homogeneous, rectangular grid. [12]. The structure of this grid can be seen on Fig. 2.1, which is the same as the structure of a CNN: a cellular network. A detailed description of the CNN architecture can be found in Appendix A. According to our hopes it can be easily seen that this can be mapped in a straightforward way on any multi-core system, like the $Xenon\_v3$ or an FPGA. A detailed desription of the Xenon architecture can be found at Appendix B.

On first read one could think, that the local selection is much simpler [13], than the global process, because we do not have to compare all the elements with each other and we do not have to order all the elements. Although this is true, because we will order only smaller sets in the population, and this will decrease the running speed. However, the algorithm will be more complex: we will have an extra parameter, the just introduced locality, the

[11] These point of views are not better or worse, but regarding engineering aspects can not be compared or used for algorithm development

[12] other grids with hexagonal or triangular positions could also be used, and especially heterogeneous grids can create interesting behaviors [11]

[13] in implementation and also in efficiency

DOI:10.15774/PPKE.ITK.2012.002

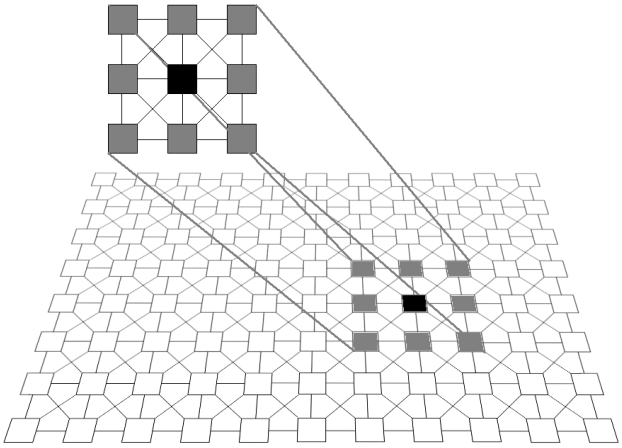


Figure 2.1: On this figure we can see the spatial distribution of the processors and also the interconnection between them. The locality in this case is defined by a neighborhood radius $r = 1$, which means the neighbors are those cells, which are closer than one unit. This neighborhood will define the communication between the elements and also how the information is divided and spreads out through the network.

size of the smaller sets. Although it is true, that this will result a more complex algorithm, and the theoretical investigation will be more difficult, but with this new parameter we did not create something totally new and different. We have only widened the possibility and the level of adaptation in our algorithm. If we set the neighborhood radius to a relatively large number [14] , then we will get back our global method, when all the elements are compared to each other and there is an interaction between any selected two elements. Also, if we set the neighborhood radius to zero [15], there will be no information exchange between the processing units, and we will get back the result when there is no selection at all. These two extreme situations are interesting and used in practice, however for us (from an engineering point of view) it is more challenging to examine with simulations the case when the neighborhood radius is between these two extrema and to see how information exchange can modify the local selection and the result of our algorithm.

Because the theoretical investigation of a general algorithm is problematic, I have decided to create experiments with a general model about stochastic selection. My aim was to create a framework that mimics all the properties of stochastic selection regarding practical problems, however it is general and in this sense independent from the representation of the problem itself.

[14] larger than the maximal distance between any two elements in the grid

[15] we will deny all interactions

DOI:10.15774/PPKE.ITK.2012.002

2.2 General Model of Stochastic Selection

2.2.1 The Model

In this subsection I will describe a simple model that grasps the essence of stochastic selection, without being to specific to have any connections with practical problems. I will start my investigation with this model and later on I will specify this general model, to show that it has a strong relation and can be applied in case of practical problems.

Examining stochastic selection and local searches in general we can identify that all these algorithms have the following common steps:

In the beginning of the process a sufficiently large amount of random samples was created by our previously given distribution. This distribution, if we do not use any problem specific heuristics, is usually uniform or normal and covers the state space. I have to note that the initial distribution should not play a crucial role in the algorithm, because the convergence of the algorithm should hold for an arbitrary -random- initial population.

We have an initial sample (sometimes called population) generated by normal distribution in the N dimensional state space [16]. We have to evaluate this initial population based on the fitness of every element. The fitness function is a scalar function, that maps the elements of the population to real numbers.

To avoid the over-representation of certain elements in the population, we will alter all the weights randomly. Later on we will recombine elements of the population and mutate them. This means, that based on a randomly driven function we will alter them (move, perturb them in the state space). This perturbation is made by a deterministic ϕ function, that maps state x, the state we would like to optimize and by random processes ($\gamma_{t.i}$), which is the stochastic part of the optimization. [17]

$$w_{t,i} = \phi(x_{t,i}) + \gamma_{t,i} \tag{2.1}$$

This weight, generated by the fitness function is the only element [18] that affect the selected elements. To get rid of the problem representation, we will ignore everything in the population, except these weights (w) in what follows.

$$w_{0,i} \sim N(0, \sigma_1), i = 1 \dots K \tag{2.2}$$

In the beginning we will create this K elements from independent samples. Intuitively we have to preserve the best elements in the population. To preserve these elements we have to create the next population randomly based on nothing but the fitness value of the elements. This will ensure, that the elements with higher fitness will have a larger chance to be a part of the next generation. This is called the selection step. However I also have to

[16] where N is the dimension of the problem

[17] here I use an additive noise, but this is not necessary

[18] apart from random Independent Identically Distributed (IID) variables

DOI:10.15774/PPKE.ITK.2012.002

note, that the best population is not necessarily a population containing only the clones of the best element, "diversity" of the population may also be desirable. We will investigate this problem later.

Here we have reached again the surface of the problem representation. We can see that this equation contains the state x again. To avoid the problem representation, we have to get rid of x. During the selection the state is not important for us, only the weight w itself. This is the other simplification of this general model. Lets assume, that this operation changes the weight by a normal additive noise. [19] This approximation that these operations are represented as an additive normal noise can be justified: large changes happening with low probability, and usually only small changes will occur during the mutation step. Also if the change is stochastic, we can assume that a random change in the genome will as likely increase its fitness as decrease it. This way we can concentrate only on the scalar weight function:

$$w_{t+1,i} = w_{t,i} + \xi_{t,i} \tag{2.3}$$

Where $\xi_{t,i} \sim N(0, \sigma_2)$ are independent. We can try to investigate the performance of local search based on this simple, general model and compare global and local selection. As we can see from this state transition it can happen that during the algorithm we will generate elements with negative weights (this problem can be avoided problem easily, because one could compare the relation of the weights instead of the direct weight or add a large sufficiently large constant to every weights). I chose zero as a minimal weight and I set all the negative values to zero before every selection step.

2.2.2 Results

The parameters of the model applied were the following: $\sigma_1 = 10$, $\sigma_2 = 3$

To compare the global and local selection mechanism I have generated a sample with $K = 400$ elements and these elements were placed on a 20×20 two-dimensional grid in case of the local sampling. In every experiments I examined the $10th$ population [20] and I have repeated every experiment 1000 time to reduce the noise of the simulations caused by the stochastic nature of the processes.

After the 10th iteration I have selected the best element [21]. The results of the global selection as the average of 1000 simulations can be seen in Table 2.1

While in case of global selection the number of selected parameters were determined directly by one, and only one parameter. In case of local selection we can set the neighborhood radius, in which distance we will examine the elements in a region of a cell. Even if it is not clear for the first read this is very similar to the parameter of the global selection.

[19] this is a generalization, but during stochastic optimization we generally use small perturbations in the state space. The effect of this small perturbation can be assumed to be normal additive noise

[20] the population after 10 selection and mutation steps

[21] the element with the highest fitness weight

DOI:10.15774/PPKE.ITK.2012.002

Table 2.1: Results of the simple model for global selection
The first column shows the amount of preserved elements during the selection step in percentage, The second column shows how many elements were selected during the selection step. The last column shows the highest value of the fitness function w obtained by this parameters. The fitness values were calculated as the average of 1000 independent simulations, the higher values mean better results.

Selected percentage	Selected elements	Value of Fitness function
1	4	58.24
2.5	10	52.36
5	20	48.24
7.5	30	45.40
10	40	43.62
12.5	50	42.13
15	60	39.58
17.5	70	38.17
20	80	38.21
22.5	90	36.03
25	100	36.11
27.5	110	33.31
30	120	33.31
32.5	130	33.33
35	140	28.82
37.5	150	28.79
40	160	28.76
42.5	170	28.77
45	180	28.75
47.5	190	28.70
50	200	28.60

DOI:10.15774/PPKE.ITK.2012.002

Previously I have described, how we can derive the global selection from the local version. If we observe this phenomena from an other point of view, it is even easier to realize the similarity. Lets imagine that we have one extremely good element in our population, with a very high weight . In the next step this element will be copied, cloned to the cells within the previously given neighborhood radius r. This parameter r will determine, how cells will see a good element: how visible it is and how it will be represented in the population. This means an upper bound, how many times the best element can be copied in one iteration, and in this sense this is very similar to the percentage of selected and preserved elements in case of the global mechanism. [22]

The result of the local selection mechanism can be seen in Table 2.2.

Table 2.2: Results of the simple model for local selection
The first column of the table contains the neighborhood radius r. The second column show how many elements are in a set defined by r. The third column is to help expressing the connection between the local and global selection mechanisms. This value can be calculated from r and represents the number of disjoint squares that can be placed on the grid used in the simulations. This means that in one step we will have at least this many independent selections for the middle element of these squares. The last column shows the value of the Fitness function (w, taken from 1000 independent simulations).

Neighborhood radius	Selection from elements	Disjoint squares	Value of Fitness function
1	9	44.44	40.55
2	25	16	47.17
3	49	8.16	50.98
4	81	4.93	53.41
5	121	3.30	55.03
6	169	2.36	56.27
7	225	1.77	57.24
8	289	1.38	58.03
9	361	1.10	58.67
10	441	0.90	59.30
11	529	0.75	59.63
12	625	0.64	59.77
13	729	0.54	60.14
14	841	0.47	60.40
15	961	0.41	60.39

As it can be seen from the comparison of the results, the general model of stochastic sampling with local sampling gives approximately the same results (for some parameters

[22] Of course we have to note, that the neighborhood radius will not only affect this property, it will also increase the section of elements seen by two cells in proximity.

DOI:10.15774/PPKE.ITK.2012.002

even better) than the global selection. in case of neighborhood radius 9, the maximal fitness value will be 58.67, we will have a similar value (58.24) with the global sampling when we select only the best 4 elements in every population of 400 entities. When we set the neighborhood radius to 1 [23], the fitness function will be 40.55, which is relatively high comparing the two different methods. [24]

Based on this results we can think, that the local selection can substitute its global counterpart, and the global algorithm is easily parallelizable and scalable and can be mapped easily to many-core architectures. Meanwhile collecting and comparing all the elements we would loose the biggest advantage of today's architectures: parallelism.

I have to mention, that in case of stochastic optimization our goal is dual. On one hand we would like to maximize the fitness value (we have investigated this scenario), on the other hand we would like to achieve the most diverse population possible, to avoid the local extrema in the state space and mutate our entities towards the global extremum.

Based on these properties I can state, that in case of the global selection, when we preserved only the best 4 elements (58.24 fitness value) the diversity of the population will be much lower than in case of the local selection. We can approximate this amount based on the lower bound on this property in table 2.2. As we will see it later the actual diversity is much larger. This would cause, that in case of a practical problem with many local extrema the global selection mechanism would perform more poorly than its local counterpart.

In this model local extrema do not occur, so the algorithm, in a sense, is trivial. For this reason we consider a slightly more complex model in the next section.

However, this can not be examined by this simple model, because it is too general and we can not add local extrema in it. Hence the the fitness function is linear.

2.3 A More Specific Model

2.3.1 Model

We have seen in the previous section, that the general model is not capable to reflect one -the biggest and most serious- problem of local stochastic searches: the local extrema. We have to add this crucial property into our model, without specializing it into practical problems. We have to stay as general as possible. To do this we have to consider the position of the elements in the state space in general, without creating a special state space related to some specific model. To avoid the implementation of a specific state space we can assume that every element in the state space will have a property p_1, to stuck in a local extremum [25] and with p_0 it will not find local bounds around itself in the state-space. This is of course again a generalization, because in case of a real problem, the state space

[23] and on every commonly used cellular architecture, this can be done easily

[24] considering that in case of the global method we would need a large global memory.

[25] which can be also global extremum

DOI:10.15774/PPKE.ITK.2012.002

is not homogeneous, and elements in different points will have different probabilities to stuck in local extrema. But in this dissertation my scope is not to compare or classify different state spaces, but to compare selection mechanisms, and this model is general but useful enough to reveal one main aspects of local stochastic searches: the diversity of the population. I also have to note that the other operations (recombination, mutation) are the same for both the global and local version of the algorithm, so eventual better results can only be explained by the selection mechanism.

We can also assume that elements close to each other will most likely be altered in the same way: If an element will stuck in a local extremum, we can presume, that all the elements around it will also stuck at the same place in the state space. Also if an element can move/mutate toward a better solution (without getting stuck) all the surrounding elements have the potential to move as well.

To achieve our goal we have to consider somehow the position of the elements in the state space, without specifying the state space itself. If we can label the positions in the state space we can calculate which elements will get stuck together. We also know that when an element with a relatively high weight is selected and copied multiple times there will be more identical elements (at the same position in the stat space) in the next generation before the mutation operator is applied. Based on this I have divided the elements into groups and each group is determined by the identity in the previous generation it was copied from. The p_0 and p_1 probabilities are applied for every group and they can determine whether a group can mutate (increase the weights of the elements) or not (they stuck in a local extrema)

During the resampling step I divide the elements into groups $\alpha_j, j = 1, \dots M$ according to the previously described rules. G_j-s are binary random (Bernoulli) variables with values zero or one. With p_0 probability $\alpha_j = 0$ and with $p_1 = 1 - p_0$ probability $\alpha_j = 1$.

Then for every i if $w_{t,i} \in G_j$ and $\alpha_j = 0$

$$w_{t+1,i} = w_{t,i} + \eta_{t,i} \tag{2.4}$$

if $w_{t,i} \in G_j$ and $\alpha_j = 1$

$$w_{t+1,i} = w_{t,i} \tag{2.5}$$

Where $\eta_{t,i} \sim N(0, \sigma_2)$ are independent and G_j represents a group of the elements in proximity.

Of course this model is a simplification of the position of the elements in the state space, and I have to note that elements can be in proximity caused by the mutation step as well, not only by the selection mechanism. The elements migrate in the state space and perform stochastic walks and even without a selection mechanism every element should visit all the points of the state space once in a while if the mutation factor is relatively large and we have to note that this model is not considering this temporal 'moving' of the elements. However in case of a relatively large state space and if the distribution of the

DOI:10.15774/PPKE.ITK.2012.002

elements is uniform in this state space the effect of resampling step driving the elements to the same places is much stronger.

2.3.2 Results

The parameters of the model were: $\sigma_1 = 10$, $\sigma_2 = 3$, $p_0 = 0.70$, $p_1 = 1 - p_0 = 0.30$

To compare the global and local selection mechanism I have generated a sample with $K = 400$ elements, and these elements were placed on a 20×20 two-dimensional grid in case of the local sampling. In every experiments I examined the $10th$ population [26] and I have repeated every experiments 1000 time, to reduce the noise on the simulations caused by the stochastic operators.

After the 10th iteration I have selected the best element [27]. The results of both the global and local selection as the average of 1000 experiments can be seen in Table 2.3 The graphical interpretation of these results can be seen on figure 2.2.

These result are very important regarding the local selection. This model is the most general, that is capable to represent the most challenging problem of local stochastic optimization. Namely, when we would like to maximize (or minimize) a function we have to have the most diverse set, and cover the whole state space to avoid all the local extrema except the only global extremum.

We can derive two conclusions from the results: It can be seen that the local algorithm outperforms its global counterpart. It can also be seen that the values of the fitness of the local selection for different parameters have a concave shape. It is true, that the extreme cases [28] are usually not efficient. This implies, that with the proper setting of the neighborhood radius we can optimize the performance of the localized algorithm. We can alter how the information spreads out through the processing elements and this way we can set the exploitation/exploration ratio of the algorithm. This way we can set either our aim is to maximize the fitness function and move towards the best value (case of global interactions) or to maintain the most diverse population possible (no interaction at all). In case of a specific problem we can set the best parameter considering the probability of finding a local extrema. Setting this parameter to an optimal value we can solve practical problems more easily and efficiently. Because this dissertation investigates primarily the implementation and design of many-core algorithms, I stress the importance of the fact, that the local selection mechanism can be easily parallelized and its execution time can be decreased to a fraction of the execution time of the global method.

[26] the population after 10 selection and mutation steps

[27] the element with the highest fitness weight

[28] small local interactions or large ,global interactions affecting the whole population

DOI:10.15774/PPKE.ITK.2012.002

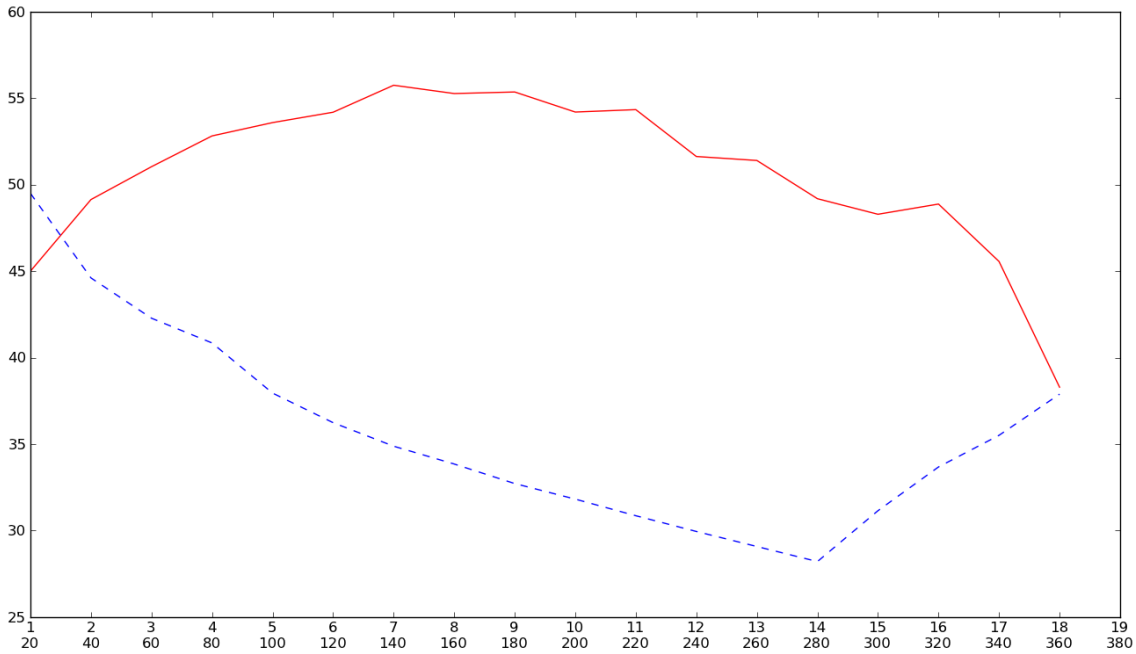


Figure 2.2: On this figure we can see the results of the local (red, continuous) and global selection (blue, dashed) mechanism. The numbers on the X-axis are the parameters, in every bracket the first is the neighborhood radius for the local method and the other is the number of the selected elements for the global method. The experiments were done with $K = 400$ elements. On the Y-axis we can see the fitness values, higher values mean better solutions. It can also be seen that the values of the fitness of the local selection for different parameters have a concave shape. This implies, that with the proper setting of the neighborhood radius we can optimize the performance of our algorithm by picking out the parameter value corresponding to the tip of this curve.

DOI:10.15774/PPKE.ITK.2012.002

Table 2.3: Results of the specific model for local and global selection
The first column of the table shows the neighborhood radius for local selection r. The second column is the result of the local method , the maximal value of the fitness function w (as an average of 1000 independent simulations). The third column is the result of the global method The last column represents the parameter of the global selection, the number of selected elements. As it can be seen the best result can be obtained with the local method, however the comparison of the two methods for one parameter-setup is not straightforward, because the parameters in the algorithms (the neighborhood radius and the number of preserved elements) are different.

neighborhood radius	fitness(L)	fitness(G)	selected elements
1	45.00	49.50	20
2	49.14	44.61	40
3	51.04	42.28	60
4	52.82	40.85	80
5	53.59	37.95	100
6	54.19	36.24	120
7	55.75	34.88	140
8	55.27	33.85	160
9	55.36	32.72	180
10	54.20	31.82	200
11	54.34	30.86	220
12	51.63	29.95	240
13	51.40	29.08	260
14	49.19	28.21	280
15	48.29	31.15	300
16	48.88	33.68	320
17	45.56	35.51	340
18	38.29	37.89	360

DOI:10.15774/PPKE.ITK.2012.002

Chapter 3

The Application of Stochastic Local Search in Practical Problems

Based on the models shown in the previous chapter one can feel that the local stochastic search can be used more efficiently [1] on many-core architectures than its global counterpart. But this reasoning was based only on a theoretical model, which can be too general and could have only a weak link to practical problems. A thorough theoretical investigation can be done only by generalization, however, the problem is too complex to investigate it in the scope of this dissertation. The previous steps were necessary to highlight the utility of the method, but to stress out the application I have to show, that they can be implemented on existing architectures and they can solve existing problems with the same accuracy but with a decreasing running time. It could happen that, because of the generality of the problems, we have forgot some important detail, that can be observed only in case of real problems. I will prove the applicability of the idea of the previous chapter with two commonly investigated and important algorithms.

3.1 Genetic Algorithms

The first practical algorithm that uses the local search is the genetic algorithm and this method is extremely closely related to the previously described models.

Numerous topographic and parallel algorithms have been described in previous years that can exploit the high performance of multi-parallel, Cellular Neural Network (CNN) architecture [1], [2]. These algorithms can be executed with an extraordinary speed; they have been used in image-processing [16], for analyzing three-dimensional surfaces [17], [3], for solving differential equations [18] or for various retina modeling tasks [19].

Genetic algorithms (GA) and their utility in practical problems have been introduced

[1] Determining the efficiency of an algorithm is an extremely difficult task because one can rank these methods by many metrics: power consumption, running speed,accuracy, prices, area of the implemented chip etc...In this dissertation efficiency is measured as a comparison of execution speed and accuracy on a virtual machine and the $Xenon\_v3$ architecture

DOI:10.15774/PPKE.ITK.2012.002

by John H. Holland in 1975, [20]. Since then these methods (with various alterations) have been applied to a large class of problems. These heuristic search algorithms, inspired by natural selection and evolutionary mechanisms, can provide solutions faster than exhaustive search, and give better results than greedy algorithms.

It can be observed in biological networks and among organisms that mate selection and genetic inheritance between generations happens locally, according to a topographic rule. Genomes that are able to overcome the challenges of nature in a well-defined environment (in the territory of the individual) can be inherited to the next generation, and spread out in a diffuse-topographic way in the population.

Examining evolution and natural selection we can see that a parallel and topographic approach is more realistic and similar to the original, motivating idea than the commonly used "global", non-topographic selection rules. This also refers to GAs, and a parallel topographic implementation can outperform its "regular" single-core ancestors.

Favorable convergence speed of the cellular algorithm has been proved, and the utility of these methods has been demonstrated theoretically in [21], but the ideal implementation on the Cellular Neural Network Universal Machine (CNN-UM) [2], has not yet been studied previously.

3.1.1 The Nonparalellized Genetic Algorithm

GAs are stochastic algorithms for search and optimization inspired by natural selection. A more detailed description of the Genetic Algorithm can be found at Appendix C. Many alterations and changes are known but the following four steps are the fingerprints that determine genetic algorithms and they can be found in all variants and alterations.

- (1) Initialization of the populations
- (2) Fitness (weight) calculation for every genome
- (3) Selection and recombination
- (4) Mutation

Steps 2, 3, 4 are repeated until a previously given time constraint, or until the optimal solution can be found [3].

I provide a brief description on these four steps that are relevant to my settings. A more detailed description can be found in [22]

(1) *Initialization of the population:*

at this step we will create random [4] vectors with values, genomes according to the problem representation.

[2] which is a true topographic mapping of the algorithm

[3] when the optimal solution can be identified based on the constraints

[4] usually uniformly distributed

DOI:10.15774/PPKE.ITK.2012.002

(2) *Fitness (weight) calculation:*

One has to calculate a fitness, weight value for every genome. This value represents a distance between our solution candidate and an optimal solution in the state space. The metric of the distance is based on problem dependent heuristics. Selecting an appropriate metric is a key question, I base my choice on well-known, published suggestions.

(3/a) *Selection of parents*:

In case of the panmictic GA, selection is calculated globally, i.e. using all the genomes generated in the previous step. We select the part of the genome we want to conserve and inherit for the next generation, and replace unnecessary elements.

There are many different methods for the selection of the parents because of practical considerations we have selected Stochastic Universal sampling [23]. With this method we need only a relatively low amount of random numbers on the chip [5]. The probability of selecting a genome as a parent is proportional to its fitness value. This is one of the most simple step in the algorithm for us and in this selection we can use all the theories and simulation results from the previous chapter, since this is the most similar to stochastic sampling.

(3/b) *Recombination:*

In this step we will recombine the genes of the selected parents and create a new entity combining their properties. In the first case, we will choose one point in the genome, and until this point all the genes will be inherited from one parent and after this point all the genes will be copied from the other parent. Ideally, the offspring solution obtained through the recombination is not identical to any of the parents, but contains combined building blocks from both.

(4) *Mutation:*

Mutation performs a random jump in the state space in the neighborhood of a candidate solution. There exist many mutation variants, which usually affect one or more loci [6] of the individual. The mutation randomly modifies a single solution whereas the recombination acts on two or more parent chromosomes.

The pseudo-code of a GA based on the previous steps can be seen at Algorithm 1.

[5] which will be a bottleneck of the algorithm as we can see it in section 3.1.5

[6] genes or components

DOI:10.15774/PPKE.ITK.2012.002

Algorithm 1 Pseudo Code of the Genetic Algorithm using deterministic sampling Parameters $PopsSize$, $MutFact$, $UsedPop$, $MaxIter$ are the following:

$PopsSize$: the size of the population, i.e. the number of genomes used in an iteration.
$MutFact$: mutation factor. The probability that a randomly selected gene will change its value.
$UsedPop$: the ratio that determines how many elements will be stored in every iteration, generation. The number of deleted elements is: $PopsSize * (1 - UsedPop)$.
$MaxIter$: the number of maximal iterations. After the $MaxIter$-th iteration the algorithm will stop regardless we have found the optimal solution or not.

Require: $MutFact\ UsedPop\ PopsSize\ MaxIter$
Ensure: $gmin$
  $gmin \Leftarrow 1$
  $Iter \Leftarrow 0$
  {/}/1- initialization
  for $i = 0$ to $PopSize$ **do**
    for every $gene$ in g_i **do**
      $gene \Leftarrow randomgene()$
    end for
  end for
  while $Iter < MaxIter$ AND $gmin \neq Fittnes(optimum)$ **do**
    {/}/2- selection-ordering genes
    for $i = 0$ to $PopSize$ **do**
      $Order(g_i, Fittnes(g_i))$
    end for
    $gmin \Leftarrow g_0$
    {/}/3-recombination
    for $i = 0$ to $PopSize$*$UsedPop/2$ **do**
      for $j = 0$ to $PopSize$*$UsedPop$ **do**
        $gnew_j \Leftarrow recombine(g_{i*2}, g_{i*2+1})$
      end for
    end for
    $g \Leftarrow gnew$
    {/}/4- mutation
    for $i = 0$ to $PopSize$ **do**
      for every $gene$ in g_i **do**
        $a \Leftarrow randomnumber(0, 1)$
        if $a < MutFact$ **then**
          $gene \Leftarrow randomgene()$
        end if
      end for
    end for
    $Iter \Leftarrow Iter + 1$
  end while

DOI:10.15774/PPKE.ITK.2012.002

3.1.2 Cellular Version of the Genetic Algorithm

The panmictic genetic algorithm can not be effectively implemented on a parallel architecture because for the selection step we need to collect fitness values from all the genomes. All the other steps could be easily implemented on a fine-grained (single instruction multiple data) architecture, where every processing unit represents one genome.

I will use an altered version of GA, a two dimensional variant of the so-called cellular genetic algorithm. The theory of these algorithms was introduced in [24], [25]. Here I will implement a two-dimensional variant on an $n \times n$ array, where the radius of the communicating neighborhoods can be set arbitrarily, by repeating the parent selection step with one neighborhood radius. We can set the exploration/exploitation ratio with this parameter, instead of using an $n \times m$ grid and varying the n/m ratio, as in [26]. In this method we will determine the parents locally, the selection of the fittest genomes and the recombination is done in a topographic way, just like in the case of real organisms.

In an iteration every genome will select the fittest genomes in a neighborhood of radius $NeighSize$, and these parents will create the gene pool of the respective genome for the next generation. With this I can ensure parallel execution, and a prefect mapping to a cellular architecture.

In this dissertation I will use only the simplest, but also most general variation of cGA, using single point mutation and single point recombination between two parents and selection based on Stochastic Universal Sampling [23] (This operations are described in Appendix C). This contains all the important features from the implementation point of view. This implementation can be used for solving other types of problems, and can be adapted to improved versions of GAs.

Because heuristic improvements like [27] or [28] nearly always intervene at the stage of mutation, fitness calculation or at other genome-dependent stages they can also be realized in the topographically distributed, multi-parallel implementation on CNN architecture.

After this general description one could easily modify the global algorithm to have the pseudo code of the cGA (the algorithm can be seen at Algorithm 2).

3.1.3 Three Practical Test Cases

For testing the algorithm, I have selected three typical, well documented problems: the N-queen problem (see [29]) for $N = 16$,the knapsack problem ([30],[31], [32]) and the travelling salesman problem ([33], [34], [35]).

These are fundamental problems from the field of combinatorial optimization. I have to note that my aim is not to compare the genetic algorithm with other possible solutions but to show the applicability of my implementation.

DOI:10.15774/PPKE.ITK.2012.002

Algorithm 2 Pseudo Code of Cellular Genetic Algorithm Parameters $PopsSize$, $MutFact$, $NeighSize$, $MaxIter$ are the following:

$PopsSize$: the size of the population, i.e. the number of genomes used in an iteration.

$MutFact$: mutation factor. The probability that a randomly selected gene will change its value.

$NeighSize$: Neighborhood size. This parameter determines the size of neighborhood in which the parents are searched. The larger value means more possible parent candidates.

Require: $MutFact\ NeighSize\ PopsSize\ MaxIter$

Ensure: $gmin$

$gmin \Leftarrow 1$
$Iter \Leftarrow 0$
{1- initialization};
for $i = 0 toPopSize$ **do**
  for every $gene$ in g_i **do**
    $gene \Leftarrow randomgene()$
  end for
end for
while $Iter < MaxIter$ AND $gmin \neq 0$ **do**
  {2- weight calculation};
  for $i = 0 toPopSize$ **do**
    $w_i \Leftarrow CalcWeight(g_i)$
  end for
  {3- selection recombination};
  for $i = 0 toPopSize$ **do**
    for each $neighbour$ in $LocalNeighbourhood(g_i, NeighSize, w_i)$ **do**
      $Parent1$, $Parent2 \Leftarrow SelectParents(neighbour, w_i)$
    end for
    {recombination};
    $gnew_{i+j} \Leftarrow recombine(Parent1, Parent2)$
  end for
  $g \Leftarrow gnew$
  {4- mutation};
  for $i = 0$ to $PopSize$ **do**
    for every $gene$ in g_i **do**
      $a \Leftarrow randomnumber(0, 1)$
      if $a < MutFact$ **then**
        $gene \Leftarrow randomgene()$
      end if
    end for
  end for
  $Iter \Leftarrow Iter + 1$
end while

DOI:10.15774/PPKE.ITK.2012.002

The N-queen Problem

In the N-queen problem our task is to place N queens on an $N \times N$ chessboard in an arrangement where no two queens can attack each other [7]. To put it differently: we have to place the queens in such a way, that each of them has to be in different rows, columns and diagonals.

The most widely used and most efficient representation of this problem is the following: the positions of the queens are represented by an array containing N [8] elements. The i-th element represents the queen in the i-th row, and its value represents the column in which the queen can be found. This way the representation ensures that every queen will be in different rows. The positioning in different columns is ensured whenever all the numbers are different in the arrays, and this can be checked easily. Only the positioning with respect to the diagonals is more difficult to detect and takes further checking during the weight calculation.

With exhausting search we should check all the possible constellations [9] and the number of good solutions is only $14,772,512$ [10]. Finding a solution with exhaustive search or with backtracking takes a lot of time.

For the N-queen problem the fitness is the number of those queens, which are attacking each others [11]. If a queen is able to attack more than one queen, its weight is calculated multiple times.

The representation of the data is also easy on this architecture as the values can be stored digitally. For the N-queen problem we have to store sixteen numbers with values from 1 to 16, the position of one queen could be stored on 0.5 byte. The storage of 16 queens require a minimal of 8 bytes.

We decided to store the position of one queen on four bits, and thus one possible arrangement can be stored on 8 bytes (64 bits).

The Knapsack Problem

The knapsack problem [36] is well-known from the field of combinatorial optimization: we are given a set of items with different predefined properties and one global constraint for every property. Our aim is to determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible.

The number of properties [12] gives the dimension of the problem. For simplicity, here we will discuss the one-dimensional problem where all the items have only one property [13].

[7] the moves of the queens are the same as regular chess moves

[8] in this case sixteen

[9] 10^{25} of them

[10] considering symmetry and eliminating symmetrical solutions only $1,846,955$

[11] in the case of the optimal solution this number is zero

[12] also the number of constraints

[13] in this case weight

DOI:10.15774/PPKE.ITK.2012.002

The solution of multidimensional knapsack problems is analogous and straightforward, we just have to check more constraints, one after the other. Apart from the relevant constraint, every item has a value. Our aim is to maximize the value of selected items, while satisfying all the constraints.

I have examined the bounded knapsack problem, where we have a limit for every element [14].

The most common representation of this problem is the following: use a vector of size N to store the objects [15]. If the value of the i-th element is set to x in the vector, it means that we have selected x pieces from the i-th element. After summing the properties for every selected element we can easily check whether a constraint is satisfied or not and by adding up the values of all the selected elements, we can easily calculate the value of our selection.

For the knapsack problem fitness is the summed value of the selected items, if they fulfill all the constraints. If there is an unsatisfied constraint then the weight is zero. In case if this problem the fitness value of the optimal solution is unknown, our aim is to maximize the fitness value itself.

The knapsack problem is an NP-complete problem and greedy algorithms do not work.

The knapsack problem I have used for simulations is the following: weights are $[100, 50, 45, 25, 20, 10, 15, 5, 22, 42, 50, 61, 71, 20, 78, 56]$ and values are represented as $[40, 35, 18, 17, 4, 10, 9, 2, 5, 7, 9, 5, 2, 50, 10, 3]$ and I have tested a bounded version, where the limit for every element was 2.

The number of possible selections is $3^{16} = 43046721$. Finding the optimal solution with exhaustive search required more than five hours on a regular CPU (intel T6670). The optimal solution for weight limit 100 is with value 145.

In case of the bounded knapsack problem, we only have to store amount of the selected elements. To store the chosen number for every element, I have required 2 bits. For 16 different elements the algorithm would require 32 bits, 4 bytes to store one possible solution.

According to the chip parameters given in Appendix B, it is easily feasible to implement a population with 1024 genomes. For one iteration with these parameters we will need: 5 Bernoulli variables for recombination 32 Bernoulli variables for mutation altogether 37 variables. To implement this we have to read one additional input-image in every iteration.

With this data representation we can easily read, write, compare and switch data in the population.

[14] usually we can select only one of each

[15] where N is the number of different elements

DOI:10.15774/PPKE.ITK.2012.002

The Travelling Salesman Problem

The Traveling salesman problem (TSP) is a well known NP-hard problem from the field of stochastic optimization. This problem is one of the most intensively studied in optimization [37].

TSP can be considered as a search of a Hamiltonian path in an undirected weighted graph, where the sum of the edge's lengths (weights) are minimal. The TSP has several applications even in its purest formulation, such as planning and the manufacture of microchips, it also appears as a sub-problem in many areas, such as DNA sequencing. Our problem can be considered as a planning problem for a mobile robot, where the robot has to choose the shortest path in an environment, visiting sixteen different objects in sequence.

The distance matrix of the objects was the following:

$$\begin{pmatrix}
0,29,82,46,68,52,72,42,51,55,29,74,23,72,46,23\\
29,0,55,46,42,43,43,23,23,31,41,51,11,52,21,53\\
82,55,0,68,46,55,23,43,41,29,79,21,64,31,51,42\\
46,46,68,0,82,15,72,31,62,42,21,51,51,43,64,28\\
68,42,46,82,0,74,23,52,21,46,82,58,46,65,23,33\\
52,43,55,15,74,0,61,23,55,31,33,37,51,29,59,17\\
72,43,23,72,23,61,0,42,23,31,77,37,51,46,33,14\\
42,23,43,31,52,23,42,0,33,15,37,33,33,31,37,36\\
51,23,41,62,21,55,23,33,0,29,62,46,29,51,11,52\\
55,31,29,42,46,31,31,15,29,0,51,21,41,23,37,43\\
29,41,79,21,82,33,77,37,62,51,0,65,42,59,61,37\\
74,51,21,51,58,37,37,33,46,21,65,0,61,11,55,12\\
23,11,64,51,46,51,51,33,29,41,42,61,0,62,23,16\\
72,52,31,43,65,29,46,31,51,23,59,11,62,0,59,51\\
46,21,51,64,23,59,33,37,11,37,61,55,23,59,0,43\\
23,53,42,28,33,17,14,36,52,43,37,12,16,51,43,0
\end{pmatrix} \tag{3.1}$$

In case of the TSP we have a vector, a solution candidate containing the order of the visiting of the objects. So we have to store a a vector with 16 elements, each element containing the objects ID, a value between 1 and 16. The storage of this vector will require 64 bits, 8 bytes.

In size the representation of this problem (in case of the stored values) is the same as the 16-queen problem. Apart from the fitness calculation all the other parts will require the same resources on the chip. In this case we will also need two additional input images.

The calculation of the fitness function of a solution candidate is: - the sum of the weights of the edges, if all the cities are visited only once in the path - 1000 if any of the cities is visited more than once in the path.

DOI:10.15774/PPKE.ITK.2012.002

The weight of the optimal solution for this distance matrix is 303.

3.1.4 Simulation Results on a Virtual Cellular Machine

Before the actual implementation I would like to show, that the theory described in the previous section is right and it can be used in case of real life and implemented on existing architecture.

I simulated a virtual version of the CNN-UM in order to test my algorithm on the previously described test case. The simulated CNN-UM was a virtual machine containing 64 processing cores similarly to the Xenon architecture. The neighborhood radius of every processor could be set to any arbitrary number. The core of the virtual machines were able to execute regular CNN operations (templates), but also arithmetical and logical operations as well. This way we could easily check the validity of the architecture, without facing the difficulties of random number generation or the limited number of supported operations in the cores.

As a measure of performance I used the average number of iterations until the best entity reaches the previously known solution of the problem. I have repeated every setup $1,000$ times and calculated the average from as the result of these independent experiments. I carried out the calculations for various parameter values (neighborhood radius, mutation factor for single point mutation, population size) to find out optimal settings. I identified the best combination for mutation factor/neighborhood radius parameters for our test cases.

The qualitative measurement of this results is quite complex, because this kind of convergence can be measured in more ways: On one hand we can measure the average convergence speed [16]. On the other hand, and this is a more practical consideration, we can measure the average of the best element after a given number of iterations. In practice we usually have a constraint about the execution time, also about the number of iterations and our aim is to achieve the best result possible in this time.

I have made the simulations with the following parameters: 1024 elements were simulated in every population in case of the N-queen and knapsack problem, this number was 2048 for the TSP. The initial distribution was generated according to uniform distribution in the state space.

As a metric to the measurements I have used the average number of iterations until the best entity will reach the optimal solution of the problem to measure the quality of a parameter setup.

I have tried to identify the best mutation factor, and the best Neighborhood radius parameters for every test case. As it can be seen from Table 3.1, 3.2, 3.3 the exploitation/exploration ratio can be set by the tuning of this two parameters and larger neighborhoods induce a higher level of implicit migration.

[16] the number of iterations in average until we will find the optimal solution

DOI:10.15774/PPKE.ITK.2012.002

Table 3.1: Average number of iterations for the 16-queen problem
The table contains the average number of iterations for the 16-queen problem for different parameters as a function of the neighborhood radius and the mutation factor for single point mutation. The results are obtained by taking the average of 1000 independent runs. The columns represents different mutation factors while the rows are showing different neighborhood radii. As it can be seen from this results (with these parameters) the best convergence speed can be obtained with mutation factor 0.2 and neighborhood radius 3 and in this case the algorithm requires 52.23 iterations in average to find the optimal solution.

N/pm	0.1	0.2	0.3	0.4	0.5	0.6
1	143.47	72.15	75.93	87.37	95.83	113.42
2	120.30	58.95	71.66	85.48	89.83	102.39
3	125.28	52.23	64.97	83.82	85.87	95.56
4	132.67	54.16	54.81	81.91	83.24	87.00
5	147.53	54.32	57.57	79.91	82.54	84.56
6	158.67	54.73	57.94	75.20	80.13	81.64

Table 3.2: Average number of iterations for the Knapsack problem
The table contains the average number of iterations for the Knapsack problem for different parameters as a function of the neighborhood radius and the mutation factor for single point mutation. The results are obtained by taking the average of 1000 independent runs. The columns represents different mutation factors while the rows are showing different neighborhood radii. As it can be seen from this results (with these parameters) the best convergence speed can be obtained with mutation factor 0.4 and neighborhood radius 4 and in this case the algorithm requires 29.06 iterations in average to find the optimal solution.

N/pm	0.1	0.2	0.3	0.4	0.5	0.6
1	162.14	104.26	87.74	56.73	61.59	70.64
2	154.22	92.59	80.69	37.02	58.68	52.70
3	148.85	85.25	61.63	35.80	41.70	40.06
4	140.39	76.52	54.46	29.06	34.14	42.17
5	138.91	68.31	46.93	33.07	38.24	43.83
6	133.12	62.39	51.76	38.35	42.88	45.32

DOI:10.15774/PPKE.ITK.2012.002

Table 3.3: Average number of iterations for the traveling salesman problem
The table contains the average number of iterations for the traveling salesman problem for different parameters as a function of the neighborhood radius and the mutation factor for single point mutation. The results are obtained by taking the average of 1000 independent runs. The columns represents different mutation factors while the rows are showing different neighborhood radii. As it can be seen from this results (with these parameters) the best convergence speed can be obtained with mutation factor 0.7 and Neighborhood radius 4and in this case the algorithm requires 129.20 iterations in average to find the optimal solution.

N/pm	0.3	0.4	0.5	0.6	0.7	0.8
1	267.05	232.97	194.50	167.86	156.87	152.69
2	237.10	215.90	189.05	163.18	144.57	141.79
3	224.33	190.02	181.38	157.48	137.93	134.33
4	204.87	178.77	139.17	133.00	129.20	137.82
5	229.69	182.71	164.28	133.01	124.32	136.73
6	225.92	188.23	172.85	137.24	127.92	135.05

After identifying the best parameters, from a more theoretical point of view I have examined the average fitness of the population for the best parameters.

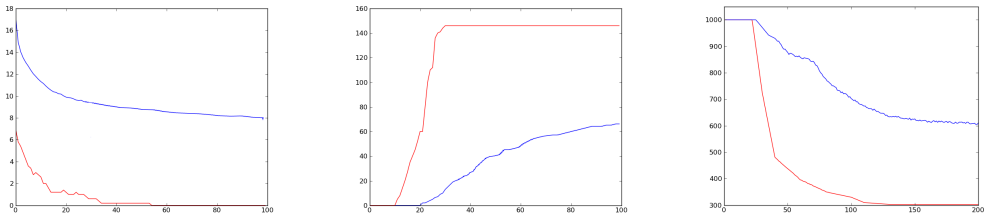
As it can be seen in Figure 3.1 the average entity is getting closer and closer to the optimal solution the value of its fitness is getting closer to the value of the solution, this means that this setup and the virtual cellular UM is capable of solving this type of optimization problems, which makes it a good architecture candidate in certain optimization tasks. I have to note, that the fitness of the average entity will not reach the optimal solution, hence the mutation and recombination operators, but the convergence of the trajectories are easily detectable. The function of the fitness values is almost monotonic. More experiments [17] would probably result a monotonic function. I do not think such a large number of experiments is necessary because the improvement of the average population can clearly be seen from the figures.

3.1.5 Implementation on the Xenon Architecture

After the quantitative measurements it can be seen that the algorithm is capable of solving optimization problems, but for its applications on a real CNN-UM instead of a virtual machine further examinations are needed. The theoretical solution is the same in the case of the virtual machine and on an existing CNN chip, however in an actual implementation we can benefit form the advantages of the CNN architecture like low power consumption. To verify the results with measurements I have tested the performance with the $xenon_v3$ chip. A detailed description of the Xenon architecture can be found in

[17] possibly 10000 instead of 1000 measurements

DOI:10.15774/PPKE.ITK.2012.002



(a) N-queen (b) Knapsack (c) TSP

Figure 3.1: The images show the results of the cellular genetic algorithm for the 16-queen, knapsack and TSP problems with the previously identified parameters. The blue line shows the average fitness of the population, while the red line marks the fitness value of the best entity in the population. On the X-axis the average number of iterations can be seen.

Appendix B.

I have selected the Xenon chip because I have already knew how it ca be programmed and also because it is able to execute simple arithmetical and logical operations which are extremely useful during the implementation of fitness calculation and the selection operators.

The Xenon chip [38] is a two dimensional digital CNN architecture that combines the advantages of cellular structures and bit-wise arithmetic/morphological units of the CNN-UM model. It contains 64 digital processing units, each one of them operates with $100Mhz$. The cores are also integrated with a 8×8 focal plane sensor array [39] through which the input can easily be uploaded directly to the memory of the processing cores. This makes this device perfect for the implementation of the cellular genetic algorithm. The schematic architecture of the Xenon chip can be seen on Figure 3.2.

Every processing unit has a relatively large (512 byte) memory that can be addressed both bit-wise and byte-wise. This memory enables us to implement a multi-layered CNN, where the different steps of the cGA can be executed in respective layers; for instance, the fitness values of genomes can be calculated in the first layer. This value can be stored there and sent forward as an input for the next layer, the parent selecting one (Figure 3.3).

On the Xenon chip every processing core contains an arithmetical and morphological unit, with these units the necessary operations can easily be implemented. Since this architecture was designed for image processing purposes, the available operations in the arithmetic unit are restricted to addition, subtraction and multiplication. These are sufficient for performing mutation, selection, recombination and they are usually enough for fitness function calculation as well. However, to implement a problem with a more complex fitness function [18] these operations could be implemented with successive approximation;

[18] containing division or square root calculation

DOI:10.15774/PPKE.ITK.2012.002

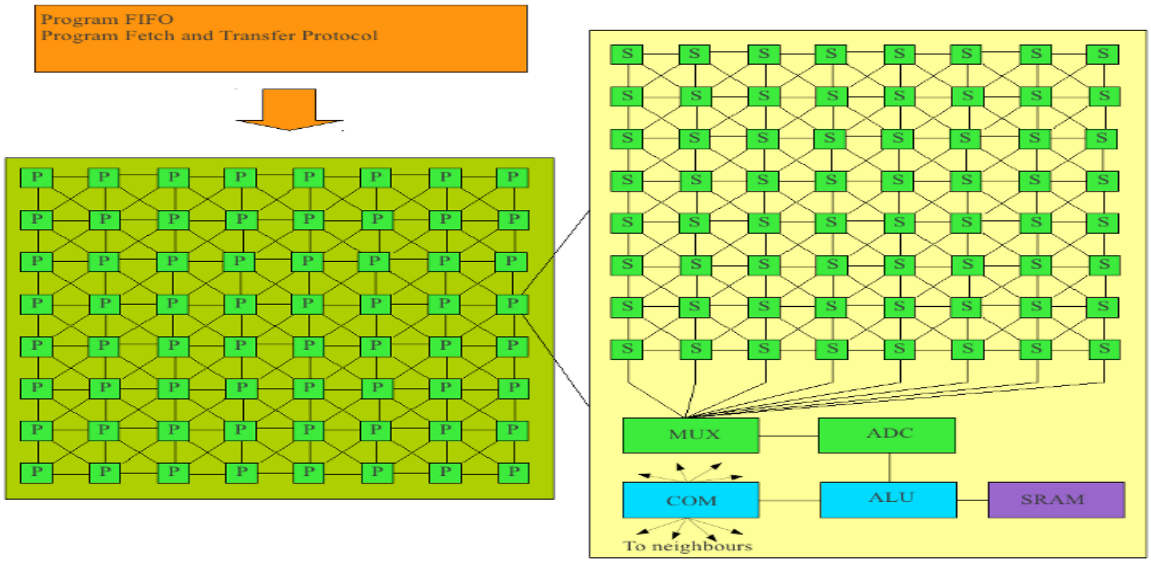


Figure 3.2: The schematic architecture of the $Xenon\_v3$ chip. The connection of the Xenon cores can be seen on the left side of the image (the cores are noted as c and the connections are represented by the lines). Every core contains an 8×8 focal plane sensor array, this can be seen on the right side of the image. Apart from the sensors every core contains a multiplexer, an analog-digital converter, an arithmetic logical unit, and a relatively large memory implemented by SRAMs.

the execution of these elementary operations being relatively fast because of the eight bit precision. The morphological unit is able to execute bit-wise logical operations [19]. With these one can easily select and alter bits in the genomes, resulting in a simple implementation for recombination and mutation. The Xenon chip is capable of operating with 30 giga operation per second (GOPS), which is an amazing performance considering its size and power consumption [20].

In the sequel I will describe the main characteristics of my implementation and also mention some general considerations about implementing cGA on the CNN-UM for arbitrary problems.

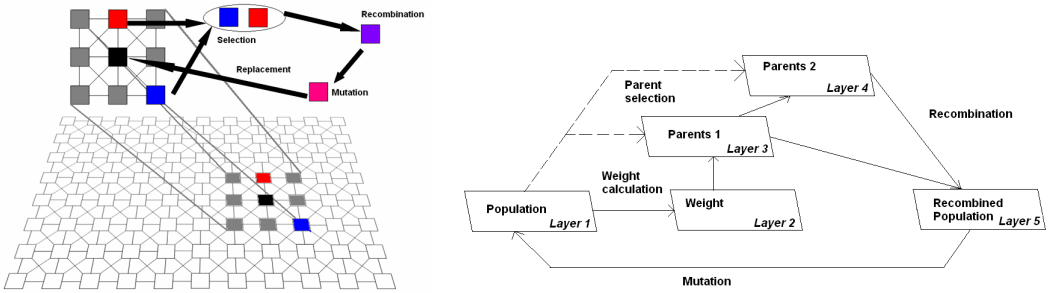
Initialization of the Population

Unfortunately, the CNN-UM lacks a simple, built-in random number generator since these operations are usually not required for image processing. It is possible to generate random numbers with CNN according to the state equations [40], however these methods are time consuming and too complex to use as part of my algorithm. I generated the initial population, along with the other necessary random numbers off-line, before the execution. Because the distribution of the necessary random numbers is known, It is enough to upload a sufficient amount of random numbers with uniform distribution.

[19] e.g. AND, OR, XOR, NOR...

[20] less than 20 mW approximately 5mW in average

DOI:10.15774/PPKE.ITK.2012.002



(a) Topographic implementation of the steps of the algorithm

(b) The implementation on the different layers of a CNN-UM

Figure 3.3: The image on the left represents the grid of the processors and the different genomes of the population, each square represents one processor (one genome). The magnified small set is the neighborhood of a given processor , which defines the calculations. The grid structure represents how the information spreads out from processor to processor by local neighborhoods. The topographic steps of the algorithm (listed in Sec. 3.1.2) are also illustrated for a particular processor (marked as black). On the other image the mapping of the algorithm can be seen and how it is divided according to the different layers in the CNN-UM. This method can be implemented on a $3 + P$ layered CNN, where P represents the selected number of parents for each recombination. Without recombination we select only one parent ($P = 1$) and the method can be implemented on a 4 layered CNN architecture. In case of regular recombination with two parents P equals two and we require 5 layers for the implementation. We will also need additional memory segments to store the previously generated random numbers. The lines are representing operations that can be done by nonlinear, multi-layered CNN templates, the dashed line is a selection, that simply copy the values from the other layer, this can also be implemented easily and efficiently on the $Xenon\_v3$ architecture.

We have to load the values of the previously generated input image into the states of the processing elements [21], according to the CNN state equation.

Calculation of the fitness function

We can calculate fitness values in another layer with an uncoupled non-linear CNN template implementing the operation described at the problem representations. This value represents a distance between our solution candidate and an optimal solution in the state space. The metric of the distance is based on problem dependent heuristics. Selecting an appropriate metric is a key question, I based my choices on well-known, published suggestions.

On the general CNN architecture we have to store these values in a different layer; on

[21] into the bottom layer in case of a multi-layered CNN architecture

DOI:10.15774/PPKE.ITK.2012.002

the Xenon architecture we can store it in another segment of our memory.

Selection of the parents

For the selection step we do not need to order the values on the grid, we just have to find local extrema [22]. There are well-known, commonly used templates for local extremum searches hence they can be implemented easily on the CNN architecture. To find both parents with the best weights in a given neighborhood, first we have to find the best parent with local maximum search, store its value, than lower its value on the original image to a minimal value by masking, and then repeat the maximum search.

On a one-layered CNN we can not store the previously found parent candidates, however, on a multi-layer one, and also on the Xenon architecture, we can do it easily [23].

If the cGA requires neighborhoods with a radius larger than one we have to iterate the local maximum search on a regular CNN architecture, however this can be done easily on the Xenon chip, because each processor can reach the neighboring elements of every genome in a 15×15 sized kernel. The CNN-UM has local connections only to the closest neighboring cells but with repeated applications we can spread out the range of the extremum search.

The implementation of other parent-selecting mechanisms [24] is more complex, involving random numbers which cannot be generated on-chip. So they need to be uploaded through the sensor array. However, the resolution of this array is limited, and iterated uploads would slow down the computations, it is advisable to use as few random numbers as possible.

Recombination – generating new genomes from the previously selected parents

The previously described operator: single-point recombination can be implemented by randomly selecting bits from one or more parents. With a non-linear template this could also be easily implemented on a multi-layered CNN.

On the Xenon architecture the parents can be selected by the morphological unit, the only problem is again the uploading of random numbers. In this case we have to use the previously generated random numbers from the input picture. Since the amount of random numbers is strictly limited [25], I have chosen the implementation of single point recombination, because for this method we will only need a small amount of random bits with Bernoulli distribution ($P(x=0)=p$, $P(x=1)=1-p$ for some $p>0$). If the length of a genome is G bit, we require $\lceil log_2(G)) \rceil$ bits to encode the position of the recombination using Bernoulli random variables.

[22] maxima or minima according to the fitness function

[23] just like in the case of fitness function calculation

[24] e.g. Stochastic Tournament Selection or Remainder Stochastic Sampling

[25] unless two input pictures are uploaded, which would increase the execution time by another iteration

DOI:10.15774/PPKE.ITK.2012.002

Mutation

The only task during this step is to detect the eventual change of a selected gene. For every gene we will need a Bernoulli random number to determine whether its value has been altered or not. This can be done again by reading out bits from the input image, one bit for every gene. I opted for the value dependent change of the gene, because this implementation requires only G Bernoulli variables per genome.

As we can see from the above discussions, during the implementation the quantity of random numbers used is a key factor and we have only one fast way to upload these numbers during an iteration: using the sensor array. One iteration of my algorithm requires $G + \lceil log_2(G)) \rceil$ random variables, which has to be considered during the implementation.

If the representation of the problem were different and one input problem were not enough, one could still execute the algorithm with multiple input images per iteration. Because of the parallel execution one could still decrease running time, while having low energy consumption. Hence in mobile low power applications with strict running time even much more complex problems would be worth being implemented on a cellular architecture.

Detecting the optimal solution

After we have implemented and executed the algorithm with a previously given iteration number we arrive at an optimal or sub-optimal solution for the problem. But we will have to find this on our cellular array by a global extremum search on the fitness function. This is a key step, although strictly speaking it is not part of the cGA algorithm. We can find the global extremum easily, by spreading out a local anisotrope maximum or minimum search for one side of the array. This operation will find the extremum in every row, after this with a vertical wave calculation we can find the exteremum in every column. In the last column the extremum will be calculated from the previously detected extrema of the rows. In this way the global minimum or maximum can be found easily, with $O(\sqrt{N})$ operations, if the number of genomes is N and they are arranged in a $\sqrt{N} \times \sqrt{N}$ grid.

When the optimal solution/fitness are unknown we can execute this global detection after the last iteration. If the fitness of the optimal solution is known and we want to detect a solution and terminate execution then we may use the global search after each iteration.

With this data representation we can easily read, write, compare and switch data in the population.

The performance and the optimal/suboptimal solutions were checked also on the real architecture and they were identical to the simulated results.

DOI:10.15774/PPKE.ITK.2012.002

Table 3.4: The distribution of the total running time amongst the operations
The first row of the table describes how many elements were used during the solution of the problems (16-queen, knapsack and traveling salesman). The next five rows contain the required clock cycle for the given operations. At the selection row the number in brackets represents the neighborhood radius which was used during the implementation (e.g 23600 (4) means that a neighborhood radius of 4 was used). At row " Additional input image" the number in brackets show the number of additional input images used (e.g 1824 (2) means that two additional input images were processed in every iteration).

	16-queen	knapsack	traveling salesman
Number of elements	1024	1024	2048
Calculation of fitness value	26832	3110	39640
Selection of two parents	18500 (5)	23600 (4)	23600 (4)
Crossover	5720	5720	5720
Mutation	192	192	192
Additional input image	1824 (2)	912 (1)	3648 (4)
Sum (clock cycle)	53068	33534	72800
Execution time of one iteration (μsec)	530	335	728
iterations/sec	1884	2982	1373

3.1.6 Performance Analysis

Because of the high performance of the chip the execution time of the actual implementation is hard to be measured in an accurate way. I have used the Xenon emulator and the output of the compiler to determine the clock cycles needed for execution. This gives us accurate measurements for the running times and also a code that can be executed on the chip for quantitative measurements. The performance and the optimal/suboptimal solutions were checked also on the real architecture and they were identical to the simulated results.

According to the compiler results for the 16-queen problem for a population with 1024 elements one iteration takes 53068 clock cycles [26]. With this performance we can execute 1884 iterations in one second. For this problem in average 83 iterations are enough to find the optimal solution as we could see it from table 3.1. With 1884 iterations one can expect that even much more difficult problems could be solved in less than a second.

For the knapsack problem with 1024 elements: 33534 clock cycles (335 μsec) per iteration, which means 29821 iterations per second.

For the traveling salesman problem with 2048 entities in the population the algorithm required 72800 clock cycles, which means 1373 iterations per second.

Table 3.4describes how running time was distributed amongst the operations.

[26] the cores on the Xenon chip are operating with 100 Mhz, this means 530 μsec per iteration

DOI:10.15774/PPKE.ITK.2012.002

It would be unfair to compare these results with non-parallel implementations of the algorithms where one iteration takes seconds. Nevertheless I have to underline that using the Xenon chip one can find an optimal solution in obviously less time, and also with less power, than using a single core architecture like CPUs.

It is almost impossible to compare my results with the same test-cases and other implementations on different chips, using hardware accelerations, because in most cases the exact details of the optimization problems are not published entirely [27].

For the $16-queen$ completely the same problem can be found, hence the few parameters in the problems, and it can be seen, that my algorithm, even comparing it to computer clusters [41] [28] is lower. A heavily optimized C program implemented by Jeff Sommers could solve the $16-queen$ in 23 seconds, which is much higher than the time required by the Xenon implementation ($0.045sec$ / 511 times speedup).

For the knapsack problem I could not find a recent reproducible experiment, with hardware acceleration and parallelism, but again comparing my method on a slightly different problem (multidimensional knapsack problem [42]), my method still has usability, because multidimensional knapsack problems were solved within seconds (the solutions of the simplest test case in the article (with 6 items, 180 iterations, and a population of 250 elements) can be found in an average 0.92 or 4.21 seconds depending on the algorithm) meanwhile my method was able to solve an almost identical problem with eight variables within milliseconds and also from a theoretical point of view with less average iterations (I have to note that the two problems were not exactly the same, however the algorithmic steps for every knapsack problems are similar, and from the details it can be easily calculated, that the execution of one iteration is much faster on the $Xenon_v3$ chip, meanwhile the power consumption is still less than one thousand on the CNN architecture. Strictly speaking, we have to add the consumption of the device that would generate the random numbers, but such a device can be implemented from shift registers with low energy consumption.

In case of he TSP, which is the most important problem from a practical point of view, because it can be considered as a path planing task for a mobile robot comparing my method with recent results [43], we can see, that my result gives the same quantitative results as other simulations. A desired suboptimal solution could be reached in 50-100 iterations, with a population containing 20 entities and using fixed crossover and mutation rate (without identifying the best values). From this point of view, considering the average number of iterations my solution is similar in complexity to a practical path planning problem.

Comparing my method with an other current result [44], which still does not contain any implementation, only simulations on complex environments [29] 100 and 80 iterations

[27] usually only the type and main characteristics of the problems are listed

[28] where power consumption is thousands of times more than in my case

[29] Indoor-like environment and Complex scattered environment - table 6 and table 5 in the cited article

DOI:10.15774/PPKE.ITK.2012.002

were used with a lower population to solve path planing tasks.

Considering this two examples and the fact that the exact test-cases are usually not listed in articles, I can claim that even if my test-case was not based on a problem from the field of mobile robotics, it has the same complexity, examining the average number of iterations.

The comparison of my method with a hardware accelerated version [45] using a GPU implementation, and a CPU+GPU implementation results that my method required less execution time to solve a similar task. In the article they have examined many different setup, for comparison I have selected the most similar, which was also a path planning problem. They have used 100 iterations with fixed crossover and mutation probabilities using the cooperative Island Model of evolutionary algorithm, and used the average of only 50 experiments. The 100 iterations for a population with 1024 entities required 0.20 sec with GPU implementation and 0.19 sec with CPU-GPU implementation. We can not say that the comparison of this two similar but different tasks, will give a fair result, or that the comparison of two different architectures is possible, especially considering that on the GPU implementation memory latency is one of the bottlenecks of the algorithm [30]. In the article they did not measure the exact power consumption of the graphic card, but according to the manufacturer, the NVIDIA GTX 280, consumes 310 watts [31]. But the execution of 100 iterations (of the problem) on the Xenon chip would require $93msec$, which means an almost 2.1 times speed up with an amazingly approximately 1500 times lower power consumption, which can underline not only the utility of my implementation and the low power/high performance probabilities of the xenon cores, but also the amazing architecture of the cellular neural networks, and its possibility to solving not only image processing tasks, but also other topographic problems, like certain combinatorial optimization problems.

3.2 The Applicability of Local Selection in cGAs on Multi-parallel Architectures

Although results of these implementations are dependent on the particular problem and on the given data representation, the general steps can be executed on a CNN architecture in similar types of problems. In this way one can benefit from the immense multi-parallel performance of the CNN chip that has been demonstrated in numerous other cases.

I hope that with this implementation we could exhibit a new, more abstract class of algorithms: topographic development of different metaheuristics [46] that can be mapped easily and with high performance on the different versions of the CNN-UM architecture.

These algorithms could be used in mobile applications where parameter optimization is needed within strict time constraints and with a low energy consumption considering

[30] reading global memory incurs an additional 400 to 600 clock cycles

[31] this is the power consumption of the processor only,not counting the surrounding environment

DOI:10.15774/PPKE.ITK.2012.002

the advantages of the $xenon_v3$ chip.

The number of elements simulated and tested on the virtual machine are enough to solve complex optimization problems in practice. The virtual machine and the Xenon architecture are both equivalents of the CNN-UM architecture, however the implementation of nonlinear templates [32] and using neighborhood radii larger than one is not possible on all the CNN architectures.

The topographic thinking and implementation of parallel, localized algorithm is getting more and more important and popular. Within this new algorithm development the cellular architectures are playing a key role, as it can be observed how the number of processors are increasing on a chip in every year. I am happy, that I could present here my version of the cellular genetic algorithm that can be mapped perfectly into a cellular machine, keeping the convergence of the algorithm as it was evidenced by three different practical test cases.

As it can be seen from the results larger neighborhoods induce a higher level of implicit migration and with proper parameter settings the exploitation/exploration ratio can be tuned to reach faster convergence speed.

[32] it is proven, that all nonlinear templates can be implemented as a series of linear templates

DOI:10.15774/PPKE.ITK.2012.002

Chapter 4

State Estimation of Hidden Markov Models

As we have seen in the previous chapters, the many-core and cellular architectures are offering brand new possibilities and also new challenges in algorithm development. In the previous chapters I have shown how the local selection can be used in case of static problems, when the optimal state (the solution) is constant in time and how it can be implemented and mapped on many-core architectures. In this chapter I will introduce the cellular particle filter (Pfilter, PF) to estimate the hidden state of Hidden Markov Models (HMM) and how this algorithm can be mapped to modern cellular architectures.

In this chapter I will introduce Hidden Markov Models and methods that can be used for state estimation. This entire chapter and the description of the theory were based on the lecture notes of Ramon van Handel used at Princeton University [47]. I will also refer to [48] and [49] for the relevant mathematical theory.

The applications of this kind of processes is extremely versatile, however they can be divided into different subgroups:

- processes where the original state [1] can not be observed directly, only through distortion and noise. E.g: the general channel model: In this case our aim is to approximate X based on the values of Y.

- The other case is, when we would like to estimate and predict Y [2], however the state transition of Y can not be seen or derived directly. Sometimes it is useful to introduce a hidden value X in the background, which has a simple state transition and also effects/determine the values of Y E.G: A stock process (Y) can have difficult dynamics, and usually we are interested in stock prices, however to introduce the manufacturing process (X) will simplify the state transition and determine [3] also the stock price.

[1] X_t that we would like to process or estimate
[2] the current and previous values can be observed
[3] with an additional noise

DOI:10.15774/PPKE.ITK.2012.002

In this dissertation I will examine the first case.

4.1 Filtering, Smoothing, Prediction

We can divide the main tasks and usual problems with HMMs into three different subgroups, however in all cases our aim is to approximate the conditional probability of the hidden state based on the observations:

$$P(X_0X_1...X_n \in S|Y_0, Y_1 \dots Y_k) \tag{4.1}$$

Our aim is always to maximize this value: identify the conditional distribution of the hidden states. The three different problems depend on the values of n and k. [4] The problems can be divided into three cases which are the following:

- filtering $k = n$
- smoothing $k < n$
- prediction $k > n$

In this dissertation I will introduce filtering ($k = n$) and I will work with this problem, however all the other problems will raise similar questions and calculations.

To solve the problem we can derive a filtering equation (described in Appendix D). It is proven that this equation gives the optimal solution of the problem the hidden trajectory with the lowest mean square error based on the observation. Based on Appendix D the filtering of HMMs can be done in theory and the equations are clear to maximize the conditional expectation. Although there are different problem classes, with special dynamics, where this calculation can be done relatively easily, in case of the general problem the calculation of the integral in (10.9) is computationally expensive. Tn practice it can be difficult to calculate the values in (10.9) and to obtain the results one has to calculate a computationally expensive integration in equation (10.9).

In some worse cases this integration can not be calculated at all and we can only apply different approximations.

4.1.1 The General Case

In the general case, covering many practical problems the state space can be infinite, continuous and the state transition and observation are arbitrary and non-linear. In this case we run into formidable computational difficulties when we wish to apply this technique in practice to calculate the filtering recursion.

[4] Usually we calculate the conditional distribution because this is the best approximation in the L^2 space. To minimize the error according to other metric we should calculate the conditional distribution according to some other arbitrary function.

DOI:10.15774/PPKE.ITK.2012.002

The problem in this case is not only, that with certain problems the calculation of the integral is extremely time consuming, but with the increase of the dimension of the problem the computational need will increase exponentially. We will have to face the curse of dimensionality (first described by Richard E. Bellman [50]) .In lower dimensions my solution could work but the resources required by the algorithm are increasing exponentially with the increase of the dimension.

The first solution that comes to our mind is to discretize the continuous state space to avoid the continuous state space [5]: this will lead to the usage of a finite grid to approximate our integral, however this grid helps us to approximate the continuous state-space, but it will not help to avoid the curse of dimensionality and the exponentially increasing need of computation.

One can easily see, that a finite grid with uniform spatial resolution can not be approximate efficiently all the possible state spaces. Certain parts of the state space has to be approximated with high resolution and in some cases even a much lower resolution can be satisfactory. We can use Monte Carlo methods to avoid unnecessary computation, because it was used in many similar problems and with this method we can avoid the curse of dimensionality. With random sampling we will still see, that not all of the used samples, lets call them particles will be used. We will see that many of them will not approximate our functional (when the observations are not identical). We will not need these samples in our recursion anymore and we can throw them out from the cohort. One can intuitively see, that it would be good to substitute these particles by other samples which were proved to be useful according to the previous observations.

These methods are called particles filters with resampling.

In the following section I will describe these methods more detailedly.

4.2 Dynamical Optimization - Particle Filtering

Particle filtering is state-of-the-art method for the state estimation of non-linear stochastic systems.

Sequential Monte Carlo methods were introduced for the computation of optimal state estimates in non-linear and non-Gaussian state-space models where analytic solutions are not available. They found applications in diverse areas such as navigation, tracking and image processing, see e.g. [51] for a representative sample. More recently they have also been applied in financial mathematics [6], see [52] and [53].

Here I examine particle filters with resampling [7], introduced in [54]. These were the prototypes whose various modifications (see e.g. [55]) proved to be efficient tools for recursive filtering. In this dissertation I will not deal with their variants or implementations

[5] forming the integration into summation

[6] e.g. to stochastic volatility models and to the computation of credit losses

[7] the so-called bootstrap filters or SIR filters

DOI:10.15774/PPKE.ITK.2012.002

but restrict ourselves to the simplest bootstrap filter and present a novel idea (initiated in [4]) for their acceleration and parallelization while keeping the same level of accuracy (or even doing better). This method can be applied to more complex versions of the particle filter algorithms, too [5].

Studying particle filters is of particular interest for distributed computing as they are inherently unsuitable for parallelization and hence pose considerable challenge. There have been attempts into this direction, e.g. [56] or [57], the approach I propose differs significantly from theirs.

The new algorithm presented here could be implemented on an array of processors and, using parallelism and local communication, could greatly enhance computational speed with similar (or even improved) precision.

4.2.1 Hidden Markov Model and Particle Filtering

It is often the case that (10.9) is difficult to calculate and q is not available in an explicit form. Then, instead of the unfeasible numerical integration, one often resorts to particle filters which provide an effective method for computing (10.9) and thus also

$$E[x_t|y_t, \ldots, y_1] = \int_{\mathbf{R}} u \, \mu_t(du).$$

There are various implementations of particle filters, but they usually contain the four steps explained below. A more detailed description can be found in [55] or [52]. We will simulate K particles whose trajectories follow the state dynamics but are subject to a selection mechanism based on observations.

The Particle Filter algorithm consist of four steps:

- 0, Initialization
- 1, Error Calculation
- 2, Resampling
- 3, Iteration

At the first step we initialize the distribution. The initial distribution can be any arbitrary distribution. The algorithm will converge with probability one apart from the chosen distribution. Usually uniform distribution is used to 'cover' the state-space as much as possible. The elements of this initial distribution will form the set of particles where we will make our measurements. This means we measure at random points in the state space.

At the previously defined random point we will calculate a probability [8] for each particle that the particle is an accurate representation of the system based on that observation. This calculation can be done easily based on the model of the system and on the observation kernel. These likelihood values are usually called particle weights (or particle's importance weights).

[8] or proportional likelihood

DOI:10.15774/PPKE.ITK.2012.002

At the next step we normalize the particle weights.I have to note that this step is not necessary from an engineering point of view ,however this ensures that this weights will represents probabilities [9] and ease the formal mathematical description of the selection step.

During the selection step (or usually called rasmpling) we resample the distribution of the particles to get a new distribution. A particle is selected at a frequency proportional to its importance weight. This is the key step of the algorithm because this can prevent particle impoverishment and cause an efficient use of then random measures, particles. This step is also closely related to stochastic selection, because we need to select the best set of particles based on random variables. This operation is computationally expensive. Generally every particle is compared to the others, or the weights of the particles ordered by some algorithms, but none of these method can be implemented efficiently on a multi-parallel architecture.

After the resampling step the new cohort of the particles are iterated according to the Hidden Markov Model, than the Error Calculation and the Resampling step are iterated during the trajectory.

A more detailed and more formal, mathematical description of the algorithm can be found in Appendix D..

4.3 Cellular Particle Filter

The algorithm sketched in the previous section has the drawback that comes from the resampling step, where all the weights/states of the particles has to be collected. This makes the algorithm non-parallelizable and hence slow, especially when a large number of particles need to be simulated (e.g. in the case of a high-dimensional state vector x_t). Simulations show that resampling usually consumes about 90% of the total computation time on regular architectures.

I am now introducing an algorithm where the resampling step is based only on local communication. I assume that each particle is represented by a processor [10] and these are imagined to sit on a rectangular grid .

In this novel algorithm resampling of particle i is performed using information from small sets N_i of particles only. Every particle i communicates with its $|N_i| - 1$ neighboring/surrounding particles [11]. When particles [12] are on a rectangular grid, a generic processor (particle) will communicate with 8 neighbors, so $|N_i| = 9$ [13]. Just like in the case of local selection, which was more detailedly described in the first chapter and the grid of the processors can be seen on Fig. 2.1, similarly to the spatial distribution of the pro-

[9] their sum has to be zero

[10] i.e. I am using with a fine-grained architecture

[11] that can be reached, say, in one clock cycle

[12] represented by one processor each

[13] Those on the edges of the rectangle obviously communicate with less neighbors

DOI:10.15774/PPKE.ITK.2012.002

cessors in the previous chapters . Initialization, error calculation and iterations steps are carried out in exactly the same way as described before, simultaneously in every processor.

The only, but crucial alteration takes places at Reasmpling: step 2. This modification is extremely important, because it makes the algorithm parallelizable.

4.3.1 Resampling (step 2) in the cellular particle filter

We have seen in the previous chapter how the parallel local selection method can be applied in computation to create a scalable, parallelizable algorithm where the selection happens according to topographic rules. The Cellular Particle Filter method was inspired by the previous results. Let's us distribute the particles on a two-dimensional grid [14] and define local connections and interactions between the particles. This means that the neighbors of the particles are defined and will not change during the algorithm. However this placement will define the actual position, or indexing in the particles and has no connection to their position in the state-space. A position of the particle in the state-space will change during the algorithm, however its position on the grid is constant. This constant placement will result a cellular structure of the particles and from this cellular structure we can easily derive neighborhoods with different radii, just like it is on Figure 2.1.

Based on these neighborhoods one can calculate the sum of the weights for every particle in its own neighborhood (this takes place in processor i based on the information sent to it from processors in N_i):

$$W_t^i = \sum_{j \in N_i} r(E_t^j), \tag{4.2}$$

and set the new weights $h_t^j(i) := r(E_t^j)/W_t^i$ for $j \in N_i$.

Processor i will select a new particle from ξ_t^j, $j \in N_i$ using the weights $h_t^j(i)$:

$$\hat{\xi}_{t+1}^i = \xi_t^{\theta_i}, \tag{4.3}$$

where θ_i are independent, take values in N_i with distribution $P(\theta_i = j) = h_t^j(i)$, $i = 1, \ldots, K$ and $j \in N_i$. Thus θ_i selects one of the particles from the neighborhood. The "best" or "fittest" particles will hence be diffused gradually over the whole grid.

The other operations and steps of the algorithms are completely the same as they are applied during the regular Particle Filter method.

Remark: Further advantages of the cellular particle filter are revealed after the previous steps and calculations are carried out. At this point we have all the particles $\hat{\xi}_T^i$ suitably distributed to approximate the distribution μ_T, but to get the desired result we have to count the average of the states, see (10.17). This step is unavoidable, regardless of the method used to generate ξ_T^i.

[14] in general it can be also a higher dimensional grid

DOI:10.15774/PPKE.ITK.2012.002

Such a calculation can be done fast and easily on a cellular-like architecture or other fine-grained parallel architectures [57]. One can start a "horizontal" wave from the left border of the grid, and spread it to the other end of the chip summing all the states, producing an average at the right border. After this one can spread a similar "vertical" wave that will calculate the average in one of the corners. With this method the calculation of (10.17) can be done in $O(\sqrt{N})$ steps, which is much better than the usual $O(N)$. This $O(N)$ collection can be done even with $O(log(N))$ in case of a tree structured topographic fine-grained architecture. However this would increase the average distance between two randomly selected particle, and change the definition of neighborhoods, so I did not investigate this possibility in the case of this algorithm.

DOI:10.15774/PPKE.ITK.2012.002

Chapter 5

Application of the Cellular Particle Filter

I have selected a few models to examine the performace of the Particle Filter with some test cases also to check the implementation on many core architecture.

5.1 Models

I tested the new algorithm on three commonly used benchmark models, well described in other papers, e.g. [55]. All represent non-linear systems with continuous state spaces, which makes the state estimation difficult since the Kalman filter is not applicable.

The first model I consider is described by the following equations:

$$x_{t+1} = \frac{x_t}{2} + \frac{25x_t}{1+x_t^2} + 8\cos(1.2t) + n_t, \tag{5.1}$$

$$y_t = \frac{x_t^2}{20} + u_t, \tag{5.2}$$

where x_t is the (unobserved) state of the system and y_t is its noisy observation; n_t and u_t are Gaussian IID sequences, independent of each other. In the simulations below I took $n_t \sim N(0, 10)$, $u_t \sim N(0, 1)$.

With the same variables the second model can be written as:

$$x_{t+1} = 0.9(x_t + 0.2x_t^3) + n_t, \tag{5.3}$$

$$y_t = x_t + u_t, \tag{5.4}$$

this time I took $n_t \sim N(0, 0.1)$ and $u_t \sim N(0, 0.05)$.

I cannot help mentioning one serious problem about the system described in (5.3) and (5.4). The stability of this model is not ensured, this can be seen easily from the state equation. However, during the simulation this occurred only with about 0.06% of the trajectories. I ignored these samples, and substituted them with stable ones.

DOI:10.15774/PPKE.ITK.2012.002

Even these previously introduced benchmark models can justify the usage of my method, however the equations that describe real problems are usually more complex, containing more dimension. To validate my approach in the case of real life problems, I wanted to test it with a more complex system-model.

To prove this I created a general 3 dimensional model, that can be described as the following:

$$X_{t+1} = \begin{bmatrix} x_{t+1}^1 \\ x_{t+1}^2 \\ x_{t+1}^3 \end{bmatrix} = AX_t + \frac{C}{\|X_t\|} + \eta_{t+1}, \tag{5.5}$$

Where:

Xt is the three dimensional state vector and $\|X_t\|$ notes the length of the vector.

$$A = \begin{bmatrix} 0.4875 & -0.0375 & 0.1875 \\ -0.0375 & 0.4875 & -0.1875 \\ 0.1875 & -0.1875 & 0.7125 \end{bmatrix}$$

is a stable matrix,

$$C = \begin{bmatrix} 1 \\ 5 \\ 10 \end{bmatrix}$$

and

$$\eta_t = \begin{bmatrix} \eta_t^1 \\ \eta_t^2 \\ \eta_t^3 \end{bmatrix}$$

is a three dimensional IID sequence where η_t is with law $N(0, 10)$.

The observations can be calculated as:

$$Y_{t+1} = \begin{bmatrix} y_{t+1}^1 \\ y_{t+1}^2 \\ y_{t+1}^3 \end{bmatrix} = X_t + \mu_{t+1}, \tag{5.6}$$

Where μ is a three dimensional IID sequence where μ_t is with law $N(0, 2)$.

In both cases I aimed at estimating the current state from past observations, i.e. at computing the conditional expectations $E[x_t|y_{t-1}, \ldots, y_1]$ using (10.17).

The algorithms I compared were the particle filter with resampling (also known as bootstrap filter, see section Appendix D); the new cellular variant of it in section 4.3 and the method of Miodrag Bolic [58], which is an other quasi-parallelized version of the bootstrap filter.

DOI:10.15774/PPKE.ITK.2012.002

5.2 Results

I have implemented virtual machines on a simple dual-core processor PC [1] to measure the performance and the accuray of the algorithms. The virtual machine could physically be realized as a square [2] array of appropriate processors. Ideally, the number of processors should coincide with the number of particles to be simulated.

The tests were carried out and averaged on 1000 different simulated trajectories of (x_t, y_t) with $t = 0, \ldots, 100$. I took as a measure of error the sum of the squared differences [3] between the true trajectories and their estimated counterparts generated by the respective algorithms.

The test runs were also carried out on an existing digital cellular neural network type (CNN) architecture: the emulated version of the Xenon-v3 chip, see [59] and [60]. A more detailed description about the Xenon chip can be found in section 3.1.5. Although this chip is designed for image processing purposes, and can process data only with 8-bit accuracy, the given two examples shows, that this still gives comparable performance thanks to the robustness of the algorithm, compared to other methods. This chip also misses the ability to generate pseudo-random numbers, so they were pre-created off-line and uploaded with the input. It also misses complex operations like square-root or division calculation, however these operations can be done fast enough using approximations.

The error rates of these different implementations can be seen in Table 5.1 for the model described by (5.1) and (5.2). The results for the model of (5.3) and (5.4) are in Table 5.2. The results of the three dimensional (5.5) can be seen in Table 5.3.

As it can bee seen from the results the new algorithm [4] outperforms all the other variants for these models.

The main advantage of this method is that if it were realized on an array of processors with a suitable computing capacity then its speed would not depend on the number of particles while it would provide an even better accuracy [5]. The number of particles is limited only by the architecture of the chip [6]. Calculations of individual particles can be done separately in a parallel way and the information can be locally distributed amongst the cellular connections for the censoring step.

The runtime of the above calculations is given in Table 5.4 [7].

It can be seen that with a small number of particles the calculation of the "regular" particle filter method takes less time. Let us not forget, however, that the presented runtimes for cellular particle filter are those of *emulations*, performed on a single core. Dis-

[1] intel T6570

[2] or rectangular

[3] mean square error, MSE

[4] with certain parameters

[5] however existing Xenon chips are not developed enough to compare them with general purpose processors

[6] by the number of processors

[7] the time measurements are basically the same for all the models

DOI:10.15774/PPKE.ITK.2012.002

Table 5.1: Results of the first model with different approaches:
The first column contains the number of the particles, the rest contains the mean-square errors of the previously mentioned methods (in order from left to right: "distributed resampling algorithm with non-proportional allocation and local exchange" (Bolic); the "classical" particle filter (PFilter), the simulation of the localized particle filter on a 32-bit architecture (Cellular PFilter); the results of the localized particle filter on the Xenon architecture ($Xenon\_v3$) with 8 bit precision)

Number of Particles	Bolic	PFilter	Cellular PFilter	$Xenon\_v3$
16	87.43	66.42	77.38	90.59
36	75.75	55.44	61.61	71.70
49	67.55	53.66	57.17	68.56
64	64.41	52.56	54.65	66.44
81	61.02	51.65	53.00	65.04
100	59.14	50.98	51.58	64.42
144	56.31	50.51	49.69	63.39
225	53.83	50.01	48.44	62.16
400	51.82	49.49	47.61	62.14
625	51.09	49.16	47.08	62.11
900	50.60	49.13	47.01	62.10

Table 5.2: Results of the second model with different approaches:
The first column contains the number of the particles, the rest contains the mean-square errors of the previously mentioned methods (in order from left to right: "distributed resampling algorithm with non-proportional allocation and local exchange" (Bolic); the "classical" particle filter (PFilter), the simulation of the localized particle filter on a 32-bit architecture (Cellular PFilter); the results of the localized particle filter on the Xenon architecture ($Xenon\_v3$))

Number of Particles	Bolic	PFilter	Cellular PFilter	$Xenon\_v3$
16	1.120	0.618	0.584	0.659
36	0.888	0.564	0.538	0.582
49	0.835	0.557	0.529	0.570
64	0.797	0.550	0.522	0.562
81	0.785	0.547	0.518	0.554
100	0.762	0.545	0.516	0.551
144	0.706	0.542	0.511	0.5467
225	0.702	0.539	0.508	0.5442
400	0.701	0.538	0.505	0.5440
625	0.693	0.537	0.504	0.5440
900	0.680	0.536	0.502	0.5438

DOI:10.15774/PPKE.ITK.2012.002

Table 5.3: Result of the three dimensional model with different approaches:
The columns from left to right are the following: number of particles, error of the regular particle filter, error of the cellular particle filter method (with neighborhood radius one).

Number of Particles	PFilter	Cellular PFilter
16	467.354	188.073
36	341.610	162.397
49	299.706	156.643
64	263.921	152.800
81	237.344	150.416
100	208.303	148.110
144	177.035	145.360
225	148.766	143.131
400	116.007	141.152
625	103.033	140.115
900	93.9301	139.353
1225	88.482	138.967
1600	85.614	138.730
2500	81.689	138.276
4096	78.104	137.973

Table 5.4: The running times of different particle filter algorithms on different architectures:
The first column contains the number of the particles (NumP), the second column (T PFR) contains the estimation time (in seconds) for 100 steps for one trajectory with the "regular" particle filter, the third column (T LPF) contains the same time for the new particle filter emulated on a regular architecture, without any parallelization (running the algorithm on a coarse-grained architecture with K cores the processing time would be approximately decreased to TLPF$/K$, because the algorithm is fully parallelizable). The fifth (T XEN) contains the run-time of the algorithm if it were implemented on a real Xenon chip (Using the same code on the device independently of the number of particles and increasing only the number of cores used). According to the compiler of the chip, one step of the calculation could be done with 1248 clock cycles, which is approximately $120ms$, so proceeding through 100 steps would take about 12 seconds.

NumP	T PFR	T NEW	T PFR/NEW	T XEN	T PFR/XEN
36	0.143	0.215	0.67	12	0.01
100	0.421	0.615	0.68	12	0.04
225	0.901	1.330	0.68	12	0.08
4096	16.417	24.495	0.67	12	1.37

DOI:10.15774/PPKE.ITK.2012.002

tributing the calculations amongst more processing units on an appropriate fine-grained architecture (FPGA, GPU) would lead to significant speed gain.

This new method outperforms "regular" particle filter in tasks where high number of particles should be used, because the processing time is independent from the number of particles, it depends only on the architecture. The estimation of a quite complex system [8] requires an increased number of particles to preserve the accuracy, and this is the most serious drawback of the original particle filter algorithm. That's why I have also included the case of 4096 particles in table 5.4. Such large numbers may become necessary when the state x_t is high-dimensional [9]. The present paper is about highlighting the novelty of the algorithm, real-life multi-dimensional models will be subject of my future research. Nonetheless one can see that even on the emulator [10] a runtime comparable to the original algorithm can be observed [11]. This shows that, in the case of a large number of particles, the cellular algorithm is worth implementing.

I also have to underline that the Xenon chip was not designed for state approximation. With a special-purpose processor containing all the necessary operations in an optimized form the processing time in column four of table 5.4 could be decreased drastically.

5.3 Setting the Speed of Information Propagation

Further observing the trajectories, it can be seen (especially in case of the third model, Table 5.3), that average approximation of the new method is fine, but when the model moves 'quickly' – especially when the gradient is changing sign – approximation slows down, because the local spreading of the information takes time. This can be seen from the comparison of the size of the neighborhood and the increasing size of the whole grid. To avoid this, we have to increase the speed of how the information (the weights and the "good" states) can be spread out on the grid.

This can be done easily by increasing the neighborhood of a given processor, for example, communicating with 24 neighbors instead of only 9. One has to be careful here, not to increase it too much, because in this way we would get back to the 'regular' method where information is propagated to all particles.

After this observation it is easy to create a new modified version of the previously described algorithm, where we can set the size of the surrounding neighborhood of every particle.

This is just one simple alteration in (4.2). Let us introduce a constant s, denoting the neighborhood size.

$$W_t^i = \sum_{j \in N_i(s)} r(E_t^j), \tag{5.7}$$

[8] especially in several dimensions

[9] see e.g. [61] where a real-life example requires a 27 dimensional state vector

[10] where there are no actual parallel chips

[11] especially with high number of particles

DOI:10.15774/PPKE.ITK.2012.002

Table 5.5: Results of the three dimensional model with different neighborhood radii using a three dimensional grid

This table shows the error rates for the 3-dimensional model I considered (it is just a coincidence that both the grid and the system have the same dimension here). The first column contains the number of particles, the following four columns represent the number of resamplings applied when $s = 1, 2, 3, 4$.

Number of Particles	Neigh size 1	Neigh size 2	Neigh size 3	Neigh size 4
64	127.046	121.145	123.461	-
216	111.830	88.793	88.645	89.213
343	108.174	83.678	80.947	80.854
512	106.143	80.460	75.260	75.343
729	104.441	78.565	72.355	71.947
1000	103.329	77.407	70.584	68.849
1728	101.603	75.626	68.573	66.103
3375	100.167	74.059	67.021	64.287
8000	98.882	72.833	65.767	63.125
15625	97.830	72.220	64.211	62.743

where $N_i(s)$ is the set of processors in a neighborhood of processor i which has radius s.

This tiny modification makes my method applicable in case of more complex systems, and all the previously mentioned advantages remain. The running time will stay low, even on a conventional (single-core) architecture.

The error rates can be seen in Table 5.5: with the increase of neighborhoods this algorithm can outperform the regular method, preserving the parallelizabilty, and low running time. For instance, the error of the common particle filter with 4096 particles is higher than cellular version using only 1600 particles and neighborhood radius $s = 4$.

The actual physical realization of larger neighborhoods may be problematic, hence another remedy for slow information propagation is suggested in section 5.5

As we can see from the results my method works better if the number of particles is relatively low, but this is not the case in real-life problems. When we want to ensure a previously given error level, we have to use a high number of particles, and in this case, the previously described method gives worse results than the 'regular' particle filter. This shows that increasing the number of particles will not be enough.

As it can be seen with the increase of neighborhoods this algorithm can outperform the regular method, preserving the parallelizabilty, and low running time. An other remedy for slow information propagation is suggested in section 5.5

DOI:10.15774/PPKE.ITK.2012.002

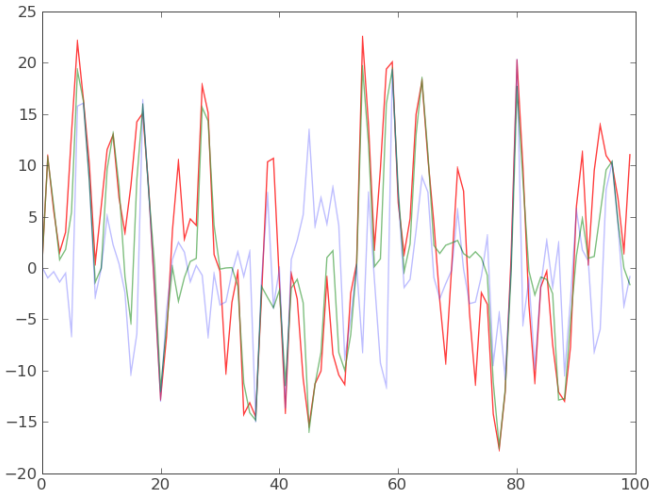


Figure 5.1: An example to demonstrate the accuracy of the regular Particle filter method and the grid based approach. The original series can be seen as red, and the approximated state by the Cellular Particle filter is marked with the green line and by the regular Pfilter is marked by light blue. As it can be seen, the approximation is extremely good and much better tahn the approximation of the regular Pfilter algorithm. On the x-axis we can see the iterations (from 0 to 100) and on the y-axis the value of the hidden state is shown.

5.4 Diversity of the particles - The Reason for a Lower Error Rate

As we could see in the first model given by (5.1) and (5.2), my method gives lower error rate (as an extra to parallelization and less running time). To identify the cause of this improvement, I analyzed a case where the error of the grid-based approach is almost half as low as in case of the original algorithm (This trajectory and its estimations can be seen on Figure 5.1).

I used 400 particles, and the results gave mean square error 97 in case of the original particle filter and 49 in case of the grid-based method.

To examine the root of this improvement I have checked the particles in a narrow band around the true state of the system. At each time step I calculated the number of particles in ε distance from the estimated series.

I took $\varepsilon = 0.3$, this can be considered as a small value, because from the previous results in section 5.2 we can see that the states are oscillating between ± 30.

With this choice I got that the number of particles close to the solution is 4118 for the 'regular method', and 3932 for the grid-based method. So here we can not see much difference. (The comparison of the number of particles in ε distance from the hidden state for both methods can be seen on Figure 5.2)

If we alter the previous measurement and count only the particles *in different states*,

DOI:10.15774/PPKE.ITK.2012.002

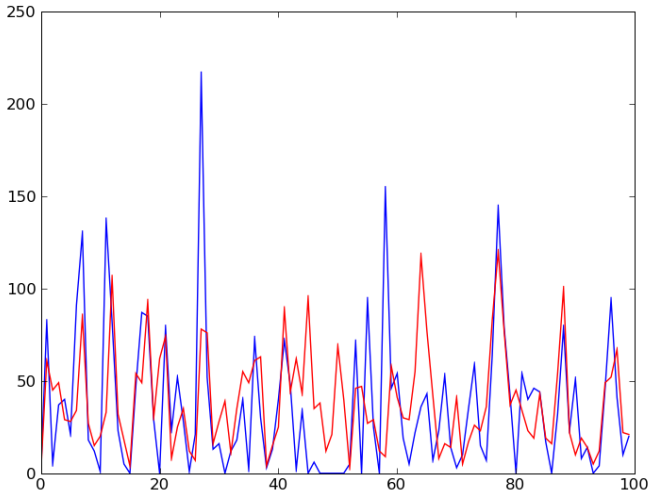


Figure 5.2: In this Figure we can see the number of particles in ε (0.3) distance from the hidden state for the regular method with blue and for the cellular version red. On the x-axis we can see the iterations (from 0 to 100) and on the y-axis the number of particles is shown. We can see, that in average the two solutions are close, however the regular method has more 'peaks'.

we can measure the diversity of our estimation. Since the process is stochastic, one can expect that a good solution is not only close to the real state, but also contains a lot of different particles: in this way we can have a substantial diversity that makes the algorithm able to follow random changes, and even after a wrong estimation helps to "find the way back" to the true hidden state and decrease the error again.

In this case the number of different particles in ε distance was: 645 for the regular method and 1170 for the grid based method, which means almost twice as many types of particles, and they are all close to the good solution. So they will form a better estimation when the weighted average (10.17) is calculated. (The comparison of the different number of particles in ε distance from the hidden state for both methods can be seen on Figure 5.3)

As it can be seen, the wrong estimations [12] are rare for the novel method because of the diffusion on the grid keeps more particles in different states 'alive'.

Figures 5.2 and 5.3 explain how particle diversity can enhance estimations, but I also have to show that this phenomenon can be observed in general, not only in case of a few particular trajectories.

The average number of 'close' particles and different particles for 1000 trajectories can be seen in Table 5.6: there is no remarkable difference between the number of particles that are close to the real state, however, the diversity amongst the particles is greatly increased

[12] where the algorithm loses track of the state

DOI:10.15774/PPKE.ITK.2012.002

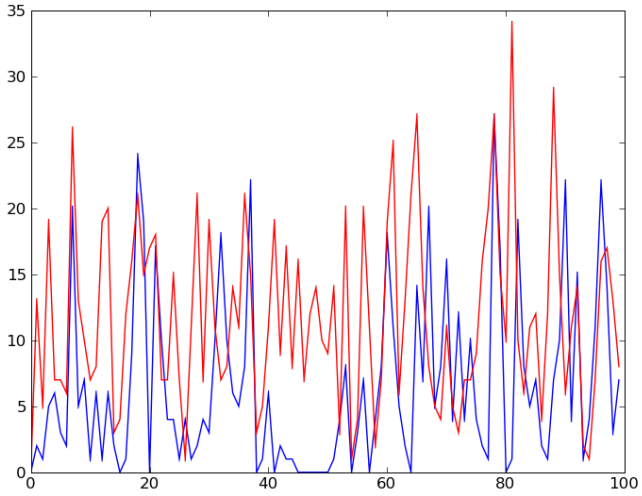


Figure 5.3: In this Figure we can see the number of different particles in ±0.3 distance for the same trajectory as in Figure 5.2. Red is the grid based method , blue is the original method. Comparing this image with Figure 5.2 one can explain the reason for the lower error rate of CPF easily.

Table 5.6: The table shows the average diversity increase of the particles for model (5.1),(5.2) for 1000 trajectories.The first column shows the total number of particles during the whole estimation (100 steps), 100 times more, than the used number of particles at one step. The second column shows the particles inside the given (±0.3) distance for the common particle filter, the third column is the same for the cellular version. The fourth and fifth columns shows the number of *different* particles inside the given corridor for the two methods.

Num of Part	Inside PFR	Inside CPF	Different PFR	Different CPF
1600	86.316	141.147	23.611	46.085
3600	246.123	331.78	54.381	107.58
4900	355.57	458.132	74.995	146.503
6400	500.074	600.766	99.729	192.65
8100	655.967	764.91	126.101	243.893
10000	844.41	946.337	158.067	301.202
14400	1278.998	1373.54	229.656	435.701
22500	2085.942	2153.456	362.832	682.617
40000	3887.569	3849.08	654.209	1213.158
62500	6183.828	6025.491	1020.445	1899.218
90000	9009.938	8709.272	1473.049	2740.171

DOI:10.15774/PPKE.ITK.2012.002

by my method. This helps the tracking of swift changes of trajectories and provides a certain type of resilience.

5.5 Another Possible Solution to Improve Information Propagation

Increased neighborhoods (see section 5.3) are easy to handle in an emulation, but are hard to be actually implemented on a real architecture because processing units are usually not wired with enough interconnections to create large neighborhoods. In FPGA designing and in new trends of processor development local connections are preferred.

With a neat trick – repeating the resampling step – the propagation of the information can be accelerated. It takes only a short time to collect all particles from the neighboring processors, in a few clock cycles they can be escalated into the centers of the neighborhoods. Hence I propose, instead of increasing the neighborhood radius of particles, to perform resampling (just as it was described in equation (4.2)) s times at each iteration of the cellular particle filter algorithm. This procedure can be imagined as equivalent to an iterative increase of neighborhoods [13].

The results obtained for the three-dimensional model (5.5) are shown in Table 5.7: resampling four times on a 35×35 grid (i.e. 1225 particles) is just as accurate as the regular method on 4096 particles and my method preserves its parallel features.

Table 5.7: Result of the three dimensional model with iterative resampling
The first column contains the radius of the neighborhood (Neigh); s in (5.7). (1-means the previously mentioned localized Particle Filter ref. -higher radius means increased neighbourhood localized particle filter ref) Than the other columns are the following, from left to right: error with 25×25, 30×30, 35×35, 40×40 particles.

Number of resamplings	Err CPF (25)	Err CPF (30)	Err CPF (35)	Err CPF (40)
1	140.019	139.420	138.951	138.707
2	102.264	101.415	100.826	100.423
3	87.907	86.637	85.955	85.449
4	80.993	79.448	78.715	77.947
5	77.171	75.388	74.398	73.605
6	74.730	72.982	71.725	71.045
7	73.512	71.412	70.094	69.260
8	72.662	70.438	68.899	68.026
9	72.491	69.769	68.234	67.196
10	72.292	69.367	67.683	66.640

Even more efficient solutions can be found using architectures where the processing

[13] expanding its size step by step from 1 to s

DOI:10.15774/PPKE.ITK.2012.002

units are organized in a higher dimensional topographical grid. A higher dimensional grid naturally brings about the improvement of neighborhood sizes. This method can be useful when we have difficult calculations, and the system is implemented on an actual computer grid. Grids can be easily transformed into 3 or even higher dimensional hypercubes, because the neighborhoods are determined by the network topology.

Let us first have a look at Table 5.3 in section 5.2 above. By performing further simulations one can check that the saturation point of the regular method is at the error level 78, one cannot go below this by further increasing the number of particles. It is quite astonishing that in case of iterative resampling the error rate (see results in Table 5.7) can reach 66, so this solution can be ideal when we want to make extremely precise estimations and we have fairly large processing source or processing time.

I also have to underline that all the experiments in this section were performed by emulations on a single core. Even in this case [14] this novel method can be successfully applied. But using a dual-, quad-, or many-core architecture, and distributing the calculations among the cores [15], the running time can be divided approximately by the number of cores. I have in mind fine-grained architectures such as some advanced versions of existing cellular neural networks.

5.6 Distribution of the Particles

In the previous sections I have shown the utility of this new, novel method by several examples. It turned out that, by suitably tuning the neighborhood radius, we may even outperform the standard bootstrap filter the "regular" method with global resampling in certain cases, when the measure of performance is the error of the estimation of the hidden state.

In practical problems in most cases we are interested in the approximation of the hidden state, and the distribution of the particles is less important for us. From a theoretical point of view, however, it is natural to ask whether my localized particle filter converges to the right conditional distribution. The convergence of the original (bootstrap) method was rigorously shown under suitable assumptions (see [62], [63], [64], [65]). My cellular resampling mechanism seems rather complex to be investigated analytically, so I have to leave this for future work.

In the present section I would like to point out the localized filter can be regarded as a certain generalized version of known methods. Moreover, I have performed specific experiments indicating that the filter converges, just like the bootstrap method. This provides further support that the localized filter can be recommended for practical problems.

The distribution of the particles on an infinite state-space can not be calculated (except some very particular cases), only discretized approximations can be used. So I have

[14] without its main advantage of parallelism

[15] which can be done easily, because the algorithm is scalable

DOI:10.15774/PPKE.ITK.2012.002

examined a problem with finite state-space. Such problems are not too interesting in practice, because all the necessary probabilities can be calculated by simple algebra (using the Baum equation for hidden Markov chains). They are ideal, however, for testing whether the method is biased or not.

I considered a hidden Markov model on a finite state space with a finite read-out space.

The table below refers to test runs with a concrete, fixed model. I have investigated the problem with many other parameter specifications [16] and the results were always the same.

This discrete model had six states $\{0, 1, 2, 3, 4, 5\}$ and I have taken a transition matrix with positive entries, i.e. all the states could be reached directly from all the other states. When the state was x, observations took the values $\{x - 1, x, x + 1\}$ (modulo 6) with positive probabilities [17].

For every parameter setup, I have calculated the error as the average of results for 1000 distinct, randomly generated observation sequences. My purpose was to investigate the distribution of the particles. I used the bootstrap method as 'benchmark' and compared the new method to this. The metric of the comparison was the following: I have calculated the number of particles in every state at a fixed time instant [18]. I looked at the number of particles in each of the 6 states in the bootstrap filter and noted the difference between this number and the number of particles in the same state for the localized filter. I added up these differences along the six states and divided this by the total number of particles.

I have also compared the distribution between two test runs with the bootstrap method (i.e. the regular method with global resampling). As the model is stochastic there is always a small difference between the distribution of the particles between two distinct test runs. I have then compared the number of particles in different states between the regular method with global resampling and my method with local resampling. The results can be seen in Table 5.8.

As it can be seen my method can be considered as an intermediate step between the particle filter without resampling ("0-radius neighborhood resampling") and the bootstrap filter (with global resampling). By choosing the neighborhood one can introduce various degrees of information propagation speed in the grid. It can be seen from the results that, when the neighborhood radius is large, the performance of the local resampling is similar to that of global resampling. When the neighborhood radius is smaller we are closer in performance to the filter without resampling, which is also a valid method, but it performs poorly due to particle impoverishment.

Our new parameter is the neighborhood size (s) which determines the information propagation speed among distant particles. If this is small, the information propagates slowly, and the diversity of the particles will be large, but they will converge/'move' slowly

[16] randomly chosen

[17] that were chosen randomly before the test runs

[18] after the 100th observation

DOI:10.15774/PPKE.ITK.2012.002

Table 5.8: In the table above the ratio of the discrepancy (i.e. the number of particles in different states in the two compared methods) and of the total number of particles can be seen, each column representing different size of population (100, 400, 1089, 1600). The first row is a comparison between the regular method with global resampling and the particle filter without resampling. The next five rows are comparisons between the global method and the cellular method with different radii of neighborhoods (1-5). The last row contains the average difference between two distinct test runs created with global resampling. I have also measured the variance for every value in the table, and the variance is low e.g.: 0.03 for the comparison of the global method and my method with a neighborhood radius of 5 with 400 particles (Global-Neigh5).

Neighborhood radius	P100	P400	P1089	P1600
Global-Neigh0	0.1294827	0.149382114	0.3453219324	0.454483990
Global-Neigh1	0.0754324	0.048415175	0.0384909734	0.036523678
Global-Neigh2	0.0521779	0.026664575	0.0175635996	0.024453972
Global-Neigh3	0.0478223	0.0236077	0.014340433	0.0144483921
Global-Neigh4	0.0466132	0.02275205	0.0144327502	0.0125891234
Global-Neigh5	0.0462745	0.02252605	0.013946832	0.011498632
Global-Global	0.0436323	0.021722355	0.0131806792	0.011053025

to the direction of the hidden state. If the neighborhood size (s) is large our filter is closer to global resampling and we will move towards the global state faster [19], but the diversity of the particles will be lower. I refer, however, to section 5.4 so as to stress out that the localized filters still maintain a much greater degree of diversity than the bootstrap filter.

We have already seen in previous examples [20], by setting neighborhood size (s) in an optimal way, we may find solutions that are more accurate than the ones one can obtain from the bootstrap filter and knowing the model this calculation can be done off-line,

5.7 The Applicability of Cellular Particle Filter in Practical Problems

My method operates with parallelism and local interactions. Better approximations could be reached than with the usual particle filter and the runtime can be decreased drastically when the method is implemented on an appropriate processor array. The algorithm takes advantage of the structure of cellular neural networks, see [66] and [67].

The present study shows, that this algorithm – implemented on cellular architecture like the $Xenon_v3$ or on a more specific locally coupled chip network – could provide a solution to many problems where the state of a complex system should be estimated with

[19] and will converge to the true distribution

[20] especially in section 5.5

DOI:10.15774/PPKE.ITK.2012.002

high accuracy within a very strict time limit, e.g. when estimating the position of an aircraft.

This method shows that with the proper parameters (number of particles, neighborhood size), one can create an extremely good balance between information preserving (to avoid particle impoverishment), and information spreading (to create good estimations). The optimal parameters can be easily found with a few simulations for a known model. With this method one can outperform the regular particle filters, ensuring a lower mean square error; we can make our estimations faster or achieve better estimation with the same temporal resolution.

The algorithm can be used even in case of regular single-core architectures, but they are inherently designed for multi-core, parallel architectures with local, cellular interconnections.

As it can be seen in chapter 2, 3, 4 and 5 the idea of local, topographic selection can be applied not only in theoretical models, but also in the solution of practical problems. In certain cases -with optimized parameters- the local method can even outperform its global counterparts The examples in these chapters are case studies only, because it would be unfeasible to examine these broad set of problems in only one dissertation. But hopefully these case studies could be enough to show and stress out (based on the general models) that the statements can be applied in practice as well. I hope, the case studies listed in these chapter can encourage the readers and other engineers to implement topographic algorithms to solve different practical tasks from the field of stochastic optimization.

The models investigated in this chapter are comparable in complexity with practical problems. Practical problems in mobile environments are usually using system with around 1000 particles [68]. This reasons the applicability of my method in practical problems. Some problems (especially) in financial mathematics could require even more particles because the noise of the model is and the required accuracy is higher. However the power consumption and the size of the architecture is usally not an issue in these problems.

The virtual machine and the Xenon architecture are both equivalents of the CNN-UM architecure, however the implementation of nonlinear templates [21] and using neighbourhood raddi larger than one is not possible on all the CNN architectures. Since my method is easily scalable it could be used also on computer clusters FPGAs or GPUs [6] and a higher number of applied particles will not change any operations used in the algorithm.

[21] it is proven, that all nonlinear templates can be implemented as a series of linear templates

DOI:10.15774/PPKE.ITK.2012.002

Chapter 6

Dynamics of Spin Torque Oscillators

In this chapter I will describe the basic dynamics of "spin torque nano-oscillators" (STOs).

It has been experimentally demonstrated that, under certain conditions of applied field and current density, a spin-polarized DC current induces a steady precession of the magnetization at GHz frequencies, i.e. the magnetic precession is converted into microwave electrical signals. I will refer to these nonlinear oscillators as STOs. They emit at frequencies which depend on field and dc current, and can present very narrow frequency line-widths.

This chapter and the theoretical description is based on the book of Russek and Rippard and a more detailed description can be found in [69].

6.1 Spin Torque nano-Oscillator

Spintronics is an emerging technology having its origins from ferromagnet/superconductor tunneling experiments and initial experiments on magnetic tunnel junctions by Julliere in the 1970s.

Spin-polarized current can be generated easily by passing the current through a ferromagnetic material. Spintorque oscillation was first predicted by John Slonczewski and Luc Berger in 1996 ([70], [71]). They have derived, that under certain conditions a sustained oscillation of the magnetization can occur at microwave frequencies. The conditions to sustain such oscillations are the following:

-the spacer layers between the magnetic layers has to be thin, less than 50 nanometers, so that spins do not depolarize as they go from one layer to another

-the device has to be sufficiently small (smaller than 100 nanometers) so that the amount of spin momentum transported by the electron current is a significant fraction of the angular momentum of the magnetic element

DOI:10.15774/PPKE.ITK.2012.002

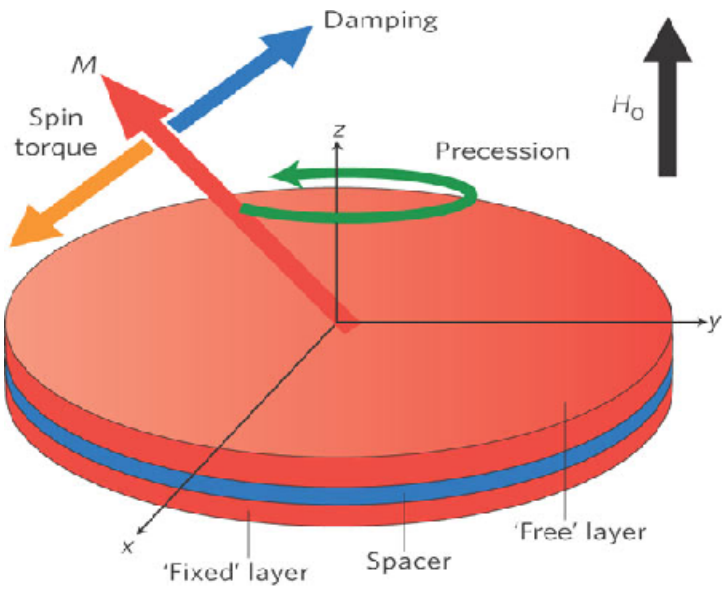


Figure 6.1: Schematic of a spin device, electrons are spin polarized. The oscillation is given by **M**. The layers of the device are (from top): free oscillating layer, nanomagnetic spacer, pinned ferromagnetic layer ('fixed layer'). The tree forces acting on the magnetic field are listed and their directions are showed by vectors: precession, dumping, spin-torque. Also the directions of the three components (x,y,z) are listed.

-there has to be sufficient nonlinearities in the configuration to stabilize the precessional orbits of the oscillation.

These structures are commonly referred to as spin-torque oscillators, spin-transfer oscillators, or spin- transfer nano-oscillators (STNOs). The inner structure of such a device can be seen on Fig. 6.1.

These microwave oscillations in magnetic multilayers were first measured by Tsoi ([72], [73]) in 1998 and 2000.

As we can see from its origins spintronic is a current field of science. The advantages of spin-transfer oscillators are that they are highly tunable by current and magnetic field, they are among the smallest microwave oscillators yet developed, they are relatively easy to fabricate in large quantities [1], they are compatible with standard silicon processing, and they operate over a broad temperature range. STOs are closely related to giant magnetoresistance and tunneling magnetoresistance devices that have been developed for magnetic recording read heads and magnetic random access memory. However physical challenges still remain before widespread applications of STNOs will be possible. These challenges include increasing the output power of these devices (consult [74]). Based on these advantages and disadvantages I think that the examination of the dynamics of these oscillators are extremely important. Recently, this phenomenon has been the subject of extensive experimental and theoretical studies.

[1] however the manufacturing of the readout circuits are difficult and complex

DOI:10.15774/PPKE.ITK.2012.002

In this chapter I consider a simple model [2] (taken from [75], [7]) that describes one nano-particle (permalloy) with a spin-polarized current.

First I have to introduce some parameters:

- the geometry of the device is described by the vector

$$\mathbf{N} = (N_x, N_y, N_z)^T \tag{6.1}$$

  The transpose of a vector is denoted by $(\cdot)^T$. The elements of $\mathbf{N}$ vector define the ratios of the state space according to each other, i.e. $N_x + N_y + N_z = 1$ has to be fulfilled. In the following I assume: $N_z > N_x, N_x = N_y$ describes a cylindric disc having $N_x = 0.2$ and $N_z = 0.6$;

- the polarization of the spin is described by the vector $\mathbf{S} = (S_x, S_y, S_z)^T$. Without losing any generality, I consider $\mathbf{S} = [0, 0, 1]^T$ (the opposite case would be $S_z = -1$, i.e. the vector of the spin points to the opposite direction);

- the spin is defined by the vector

$$\mathbf{M}(t) = (M_x(t), M_y(t), M_z(t))^T \tag{6.2}$$

  To simplify the notation[3] I denote $\mathbf{M}(t)$ by $\mathbf{M} = (M_x, M_y, M_z)^T$. The length of the vector $\mathbf{M}(t)$ has to be equal to 1, i.e. the following relation holds[4]

$$M_x^2(t) + M_y^2(t) + M_z^2(t) = 1, \qquad \forall t \tag{6.3}$$

- the magnetic field $\mathbf{H}$ can be defined in terms of the vectors $\mathbf{M}$ and $\mathbf{N}$ as follows:

$$\mathbf{H} = -M_s \begin{pmatrix} N_x M_x \\ N_y M_y \\ N_z M_z \end{pmatrix} \tag{6.4}$$

  where M_s is a parameter related to the saturation magnetization of the material. Permalloy is characterized by $M_s = 8.6 \cdot 10^5$. It is worth pointing out that $\mathbf{H}$ is not needed directly, because it is a function of $\mathbf{M}$, i.e. (6.4) may yield a more compact form useful to understand the physical properties of the system.

The equations of the motion of the spin torque nano-oscillator turn out to be:

$$\frac{d\mathbf{M}}{dt} = \gamma(\mathbf{M} \times \mathbf{H}) - \gamma\alpha\mathbf{M} \times (\mathbf{M} \times \mathbf{H}) - \gamma A\mathbf{M} \times (\mathbf{M} \times \mathbf{S}) \tag{6.5}$$

[2] created by Prof. Wolfgang Porod's group in Notre Dame

[3] Time dependency is explicitly reported if it is needed.

[4] This can be derived by the equations and also by the physics of the system.

DOI:10.15774/PPKE.ITK.2012.002

where $\times$ denotes the cross product between the vectors, A is the normalized current[5] and the physical constants γ (gyro-magnetic ratio) and α (magnetic efficiency) are physical parameters with values $\gamma = 2.21 \cdot 10^5$ and $\alpha = 8 \cdot 10^{-3}$, respectively.

The first term in (6.5) is the precession of the system, the second one denotes the damping and the last is spin torque caused by the current A.

Equation (6.5) is completely defined with initial conditions $\mathbf{M}(0)$. Equation (6.3) implies we have to select a unit-vector in space. For the sake of simplicity, I have decided to determine $\mathbf{M}(0)$ by two angles as follows [6]: ϕ and θ;

$M_x(0) = cos(\phi)\, sin(\theta)$

$M_y(0) = sin(\phi)\, sin(\theta)$

$M_z(0) = cos(\theta)$

Oscillations of actual STO are in the GHz range, so I can consider a time scale in ns (i.e. in numerical simulations time is meant in the ns units).

Fig. 6.2 shows oscillations, obtained by means of numerical simulations, in STOs described by of (6.5).

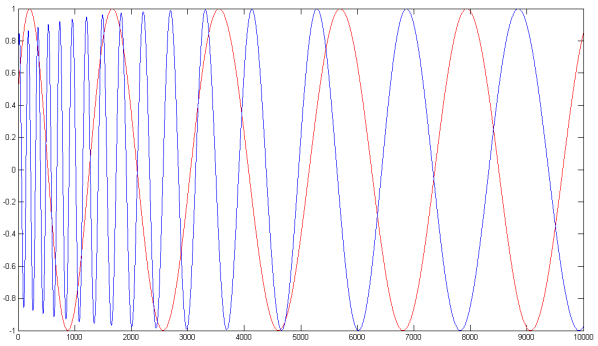


Figure 6.2: Oscillations in the GHz region of two STOs (6.5) with the same parameters but with different initial conditions (noted with red and blue curves). As it can be observer both oscillators have sinusoidal waveforms. The first components $M_x(t)$ is just displayed on this figure. The y axis shows the magnitude of the $M_x(t)$ component.

6.2 Spin Torque nano-Oscillator arrays

Electron currents in magnetic multilayer devices can transport angular momentum from one magnetic layer to another, thereby exerting a torque on the local magnetization. The interaction among STOs occurs by means of the magnetic field. This can be modeled by modifying $\mathbf{H}$ given in (6.4) as follows

[5] This is a number proportional to the current of the oscillator and not the actual current itself (for instance $A = 100$ corresponds to $1mA$).

[6] this ensures that the length of the vector is 1

DOI:10.15774/PPKE.ITK.2012.002

$$\begin{aligned} \mathbf{H}_{eff} &= -\mathbf{H}_i + \mathbf{G}_j \\ &= -M_s \begin{pmatrix} N_x M_{xi} \\ N_y M_{y_i} \\ N_z M_{zi} \end{pmatrix} + M_s \begin{pmatrix} C_x^j M_{xj} \\ C_y^j M_{y_j} \\ C_z^j M_{zj} \end{pmatrix} \end{aligned} \tag{6.6}$$

where the first term is the magnetic field of the i^{th} STO (i.e. $\mathbf{H}_i = M_s(N_x\, M_{xi}, N_y\, M_{y_i}, N_z\, M_{zi})^T$) and the second addendum describes the influence of the j^{th} STO on the magnetic field of the i^{th} STO. This influence is defined by the vector $\mathbf{C}^j = (C_x^j, C_y^j, C_z^j)^T$.

Using $\mathbf{H}_{eff}$ in (6.5), I obtain the following model describing the dynamics of one–dimensional STO arrays ($i = 1, \ldots, N$):

$$\begin{aligned} \frac{d\mathbf{M}_i}{dt} &= \gamma(\mathbf{M}_i \times \mathbf{H}_i) \\ &- \gamma\alpha\mathbf{M}_i \times (\mathbf{M}_i \times \mathbf{H}_i) \\ &- \gamma A\mathbf{M}_i \times (\mathbf{M}_i \times \mathbf{S}) \\ &+ \sum_{j=1}^{N} \gamma(\mathbf{M}_i \times \mathbf{G}_j) \\ &- \sum_{j=1}^{N} \gamma\alpha\mathbf{M}_i \times (\mathbf{M}_i \times \mathbf{G}_j) \end{aligned} \tag{6.7}$$

DOI:10.15774/PPKE.ITK.2012.002

Chapter 7

The phase equation of Synchronized network of Spin oscillators

7.0.1 Dynamical properties of STO

In order to unfold the dynamical properties of (6.5) it is useful to write the equations of each component of $\mathbf{M}(t)$:

$$\begin{aligned}
\frac{dM_x}{dt} &= M_y\left[M_z\gamma M_s(N_y-N_z)\right] \\
&\quad +M_x\left[\gamma\alpha M_s(N_z-N_x)M_z^2-\gamma AM_z\right] \qquad (7.1)\\
\frac{dM_y}{dt} &= M_x\left[M_z\gamma M_s(N_z-N_x)\right] \\
&\quad +M_y\left[\gamma\alpha M_s(N_z-N_y)M_z^2-\gamma AM_z\right] \qquad (7.2)\\
\frac{dM_z}{dt} &= \gamma(M_xH_y-M_yH_x) \\
&\quad -\gamma\alpha({M_x}^2H_z+{M_y}^2H_z-M_xM_zH_x-M_yM_zH_y) \\
&\quad -\gamma A(x^2+y^2) \qquad (7.3)
\end{aligned}$$

It can be seen easily that cylindrical geometry, i.e. $\mathbf{N}$ such that $N_z > N_x, N_x = N_y$, makes possible further simplification in (7.1)–(7.3). In addition, the constrain (6.3) implies that the last equation (7.3) can be replaced with the following algebraic condition $M_z(t) = \sqrt{1-M_x^2(t)-M_y^2(t)}$.

Simulations reveal that trajectories of (7.1)–(7.3) converge toward a plane[1] defined by $M_z = const$. This can be easily shown by investigating the equilibria of (7.3). Substituting the magnetic vector $\mathbf{H}$ (given in (6.4)) in the equation (7.3), the following equation is obtained:

[1]With cylindric geometry the third component $M_z(t)$ always converges toward a constant value.

DOI:10.15774/PPKE.ITK.2012.002

$$\begin{aligned}\frac{dM_z}{dt} = F(M_z) &= M_s(M_xM_yN_y - M_xM_yN_x) \\ &- \alpha M_s M_z \left[(N_x - N_z){M_x}^2 + (N_y - N_z){M_y}^2\right] \\ &- A({M_x}^2 + {M_y}^2)\end{aligned} \tag{7.4}$$

The first addendum of the equation above is zero, because we have $N_x = N_y$, i.e. cylindric geometry. Replacing $(N_x - N_z)$ with $(N_y - N_z)$ (as $N_x = N_y$), we can collect $M_x^2 + M_y^2 = 1 - M_z^2$ in the second term. Thereby, (7.4) becomes

$$\frac{dM_z}{dt} = F(M_z) = (-\alpha M_s(N_x - N_z)M_z - A)\left(1 - {M_z}^2\right) \tag{7.5}$$

Defining the parameters $\delta = N_z - N_x = N_z - N_y$, the only equilibrium M_z^* is obtained by imposing $F(M_z) = 0$

$$M_z^* = -\frac{A}{\alpha\, M_s(N_x - N_z)} = \frac{A}{\alpha\, M_s\, \delta} \tag{7.6}$$

It is worth observing that $M_z^* = \pm 1$ are trivial equilibria corresponding to no oscillations.

For the case under consideration ($\alpha = 8 \cdot 10^{-3}$, $M_s = 8.6 \cdot 10^5$, $N_x = 0.2$ and $N_z = 0.6$)

$$M_z^* = 0.363$$

which matches the numerical simulations (see Figs. 7.1 and 7.2).

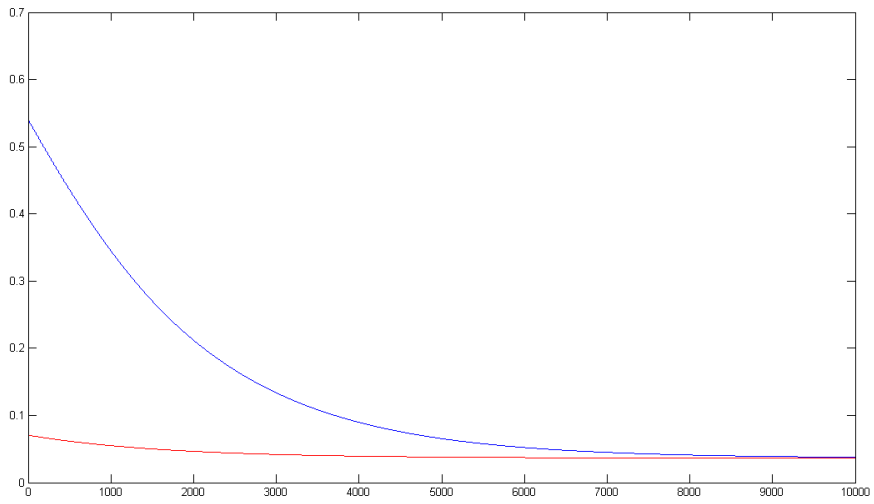


Figure 7.1: The evolution of $M_z(t)$ of two STOs with the same parameters but with different initial conditions (noted with red and blue curves). As it can be observer $M_z(t)$ converges to a fixed point M_z^*.

It can be seen that

DOI:10.15774/PPKE.ITK.2012.002

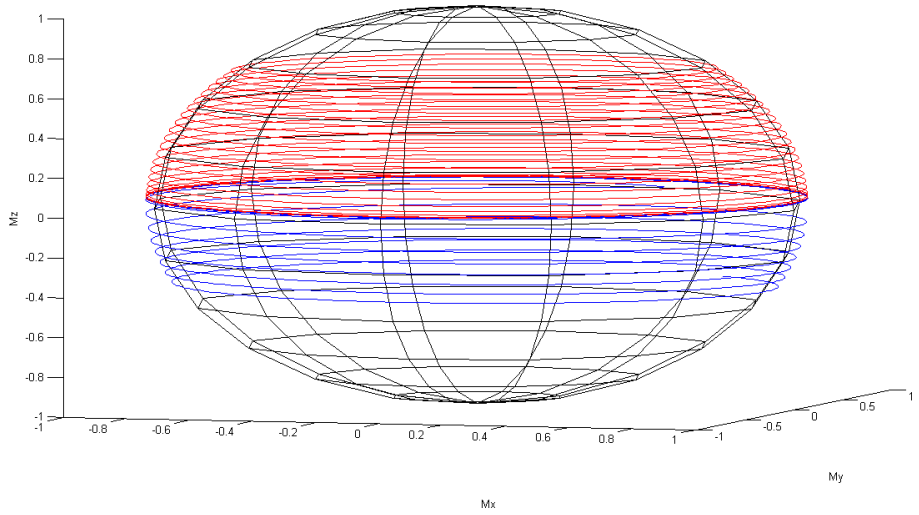


Figure 7.2: The magnetization precession vector $\mathbf{M}(t)$ of two STOs with the same parameters but with different initial conditions (noted with red and blue curves). $\mathbf{M}(t)$ converges toward a plane defined by M_z^*.

- M_z^* depends on the geometry of the cylinder, the material and the current of the STO
- M_z^* increases linearly by increasing the current A
- M_z^* is the only stable equilibrium point of (7.5)

$$\frac{F(M_z)}{dM_z}|_{M_z^*} = \alpha M_s \delta (1 - M_z^*) < 0 \tag{7.7}$$

We can also introduce the parameters $\tilde{M}_z = M_z/M_s$ and $\eta = \gamma\, M_s$ to describe the equations (7.1)–(7.3) in a more compact form:

$$\frac{dM_x(t)}{dt} = M_y \left[M_z \eta(-\delta)\right] + M_x \left[\alpha\eta\delta M_z^2 - \eta A \tilde{M}_z\right] \tag{7.8}$$

$$\frac{dM_y(t)}{dt} = M_x \left[M_z \eta\delta\right] + M_y \left[\alpha\eta\delta M_z^2 - \eta A \tilde{M}_z\right] \tag{7.9}$$

$$M_z(t)^2 = 1 - \left[M_x(t)^2 + M_y(t)^2\right] \tag{7.10}$$

Equations (7.8)–(7.9) allow us to identify the amplitude and the frequency of the oscillations when $z(t)$ approaches z^*.

7.0.2 Frequency and amplitude of the oscillation

Substituting M_z^* in (7.8)–(7.9), the simplified equations governing the evolution of $M_x(t)$ and $M_y(t)$ on the plane $M_z(t) = M_z^*$ result to be:

DOI:10.15774/PPKE.ITK.2012.002

$$\frac{dM_x(t)}{dt} = C_2 M_x - C_1 M_y \tag{7.11}$$

$$\frac{dM_y(t)}{dt} = C_1 M_x + C_2 M_y \tag{7.12}$$

where $C_1 = \eta M_z^* \delta$ and $C_2 = \eta(\alpha\delta M_z^{*2} - A\tilde{M_z}^*)$. These two equations represent a dumped harmonic oscillator on the plane $M_z(t) = M_z^*$. The constrain $M_x^2 + M_y^2 + M_z^2 = 1$ implies that the trajectory of the STO, described by (7.1)–(7.3), approaches a limit cycle on the plane $M_z(t) = M_z^*$ (see Fig. 7.3). The angular frequency of the limit cycle is defined by C_1, that is $\omega = C_1$. As a consequence the angular frequency ω is

$$\omega = \frac{\gamma A}{\alpha} = 2.76 \cdot 10^9 \, Hz = 2.76 \, GHz \tag{7.13}$$

In addition, it can be easily seen that the amplitude of the oscillations for $M_x(t)$ and $M_y(t)$ (denoted as B) is

$$B = \sqrt{(1 - M_z^{*2})}$$

It follows that the steady state behavior of $M_x(t)$ can be written as

$$M_x(t) = B \, \sin(\omega t); \tag{7.14}$$

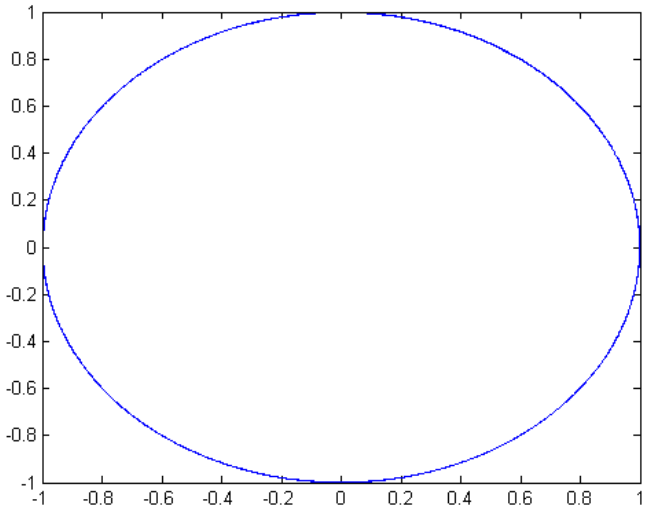


Figure 7.3: Limit-cycle on the plane $M_z(t) = M_z^*$, i.e. steady state behavior of a STO when M_z approaches the equilibrium M_z^*.

7.1 Spin Torque nano-Oscillator arrays

The interaction among STOs occurs by means of the magnetic field. This can be modeled by modifying **H** given in (6.4) as follows

DOI:10.15774/PPKE.ITK.2012.002

$$\begin{aligned}\mathbf{H}_{eff} &= -\mathbf{H}_i + \mathbf{G}_j \\ &= -M_s \begin{pmatrix} N_x M_{xi} \\ N_y M_{y_i} \\ N_z M_{zi} \end{pmatrix} + M_s \begin{pmatrix} C_x^j M_{xj} \\ C_y^j M_{y_j} \\ C_z^j M_{zj} \end{pmatrix}\end{aligned} \tag{7.15}$$

where the first term is the magnetic field of the i^{th} STO (i.e. $\mathbf{H}_i = M_s(N_x\, M_{xi}, N_y\, M_{y_i}, N_z\, M_{zi})^T$) and the second addendum describes the influence of the j^{th} STO on the magnetic field of the i^{th} STO. This influence is defined by the vector $\mathbf{C}^j = (C_x^j, C_y^j, C_z^j)^T$.

Using $\mathbf{H}_{eff}$ in (6.5), I obtain the following model describing the dynamics of one–dimensional STO arrays ($i = 1, \ldots, N$):

$$\begin{aligned}\frac{d\mathbf{M}_i}{dt} &= \gamma(\mathbf{M}_i \times \mathbf{H}_i) \\ &- \gamma\alpha\mathbf{M}_i \times (\mathbf{M}_i \times \mathbf{H}_i) \\ &- \gamma A\mathbf{M}_i \times (\mathbf{M}_i \times \mathbf{S}) \\ &+ \sum_{j=1}^{N} \gamma(\mathbf{M}_i \times \mathbf{G}_j) \\ &- \sum_{j=1}^{N} \gamma\alpha\mathbf{M}_i \times (\mathbf{M}_i \times \mathbf{G}_j)\end{aligned} \tag{7.16}$$

Considering local (cellular), space-invariant coupling $\mathbf{G}_j$ can be written as:

$$\mathbf{G}^j = M_s \begin{pmatrix} C_x^- M_{xi-1} \\ C_y^- M_{y_{i-1}} \\ C_z^- M_{zi-1} \end{pmatrix} + M_s \begin{pmatrix} C_x^0 M_{xi} \\ C_y^0 M_{y_i} \\ C_z^0 M_{zi} \end{pmatrix} + M_s \begin{pmatrix} C_x^+ M_{xi+1} \\ C_y^+ M_{y_{i+1}} \\ C_z^+ M_{zi+1} \end{pmatrix} \tag{7.17}$$

The middle part in the equation (the self coupling of a STO) is always zero, because $\mathbf{M}_i \times \mathbf{M}_i = 0$ for every $\mathbf{M}_i$.

Considering the space-invariant coupling I simplify the notation as follows: $\mathbf{C}^+ = \mathbf{r} = (r_x, r_y, r_z)^T$ (the coupling from the i^{th} STO to its right neighbor) and $\mathbf{C}^- = \mathbf{l} = (l_x, l_y, l_z)^T$ (the coupling from the i^{th} STO to its left neighbor), i.e.

$$\mathbf{G}^j = M_s \begin{pmatrix} l_x M_{xi-1} \\ l_y M_{y_{i-1}} \\ l_z M_{zi-1} \end{pmatrix} + M_s \begin{pmatrix} r_x M_{xi+1} \\ r_y M_{y_{i+1}} \\ r_z M_{zi+1} \end{pmatrix} \tag{7.18}$$

With these notations and expanding $\mathbf{H}_i$ and $\mathbf{G}$, we get the following equations (governing the evolution of the components of $\mathbf{M}_i$):

DOI:10.15774/PPKE.ITK.2012.002

$$\begin{aligned}\frac{dM_{xi}}{dt} = &-\eta\delta M_{y_i}M_{zi} + \alpha\eta\delta M_{xi}{M_{zi}}^2 - cAM_{zi}M_{xi}+\\ &+\eta r_z M_{y_i}M_{zi+1} - \eta r_y M_{zi}M_{y_{i+1}}\\ &-\eta\alpha r_y M_{xi}M_{y_i}M_{y_{i+1}} - \eta\alpha r_z M_{xi}M_{zi}M_{zi+1}\\ &+\eta\alpha r_x {M_{y_i}}^2 M_{xi+1} + \eta\alpha r_x {M_{zi}}^2 M_{xi+1}\\ &+\eta l_z M_{y_i}M_{zi-1} - \eta l_y M_{zi}M_{y_{i-1}}\\ &-\eta\alpha l_y M_{xi}M_{y_i}M_{y_{i-1}} - \eta\alpha l_z M_{xi}M_{zi}M_{zi-1}\\ &+\eta\alpha l_x {M_{y_i}}^2 M_{xi-1} + \eta\alpha l_x {M_{zi}}^2 M_{xi-1}\end{aligned} \tag{7.19}$$

$$\begin{aligned}\frac{dM_{y_i}}{dt} = &\eta\delta M_{xi}M_{zi} + \alpha\eta\delta M_{y_i}{M_{zi}}^2 - cAM_{zi}M_{y_i}+\\ &+\eta r_x M_{zi}M_{xi+1} - \eta r_z M_{xi}M_{zi+1}\\ &-\eta\alpha r_x M_{xi}M_{y_i}M_{xi+1} - \eta\alpha r_z M_{y_i}M_{zi}M_{zi+1}\\ &+\eta\alpha r_y {M_{xi}}^2 M_{y_{i+1}} + \eta\alpha r_y {M_{zi}}^2 M_{y_{i+1}}\\ &+\eta l_x M_{zi}M_{xi-1} - \eta l_z M_{xi}M_{zi-1}\\ &-\eta\alpha l_x M_{xi}M_{y_i}M_{xi-1} - \eta\alpha l_z M_{y_i}M_{zi}M_{zi-1}\\ &+\eta\alpha l_y {M_{xi}}^2 M_{y_{i-1}} + \eta\alpha l_y {M_{zi}}^2 M_{y_{i-1}}\end{aligned} \tag{7.20}$$

$$\begin{aligned}\frac{dM_{zi}}{dt} = &-\alpha\eta\delta M_{zi}({M_{xi}}^2 + {M_{y_i}}^2) + cA({M_{xi}}^2 + {M_{y_i}}^2)\\ &+\eta r_y M_{xi}M_{y_{i+1}} - \eta r_x M_{y_i}M_{xi+1}\\ &-\eta\alpha r_y M_{y_i}M_{zi}M_{y_{i+1}} - \eta\alpha r_x M_{xi}M_{zi}M_{xi+1}\\ &+\eta\alpha r_z {M_{y_i}}^2 M_{zi+1} + \eta\alpha r_z {M_{xi}}^2 M_{zi+1}\\ &+\eta l_y M_{xi}M_{y_{i-1}} - \eta l_x M_{y_i}M_{xi-1}\\ &-\eta\alpha l_y M_{y_i}M_{zi}M_{y_{i-1}} - \eta\alpha l_x M_{xi}M_{zi}M_{xi-1}\\ &+\eta\alpha l_z {M_{y_i}}^2 M_{zi-1} + \eta\alpha l_z {M_{xi}}^2 M_{zi-1}\end{aligned} \tag{7.21}$$

In every equations the first rows describe the dynamics of the i^{th}oscillator, whereas the remaining rows take into account the couplings. As it can be seen the first two equations are symmetrical for M_x and M_y.

In the following wI consider couplings involving only $M_z(t)$ in order to derive qualitative properties of STO arrays.

7.1.1 Cellular STO arrays with interactions in $M_z(t)$ only

I focus on a simplified STO array model with couplings given by (7.18) such that

$$r_x = l_x = r_y = l_y = 0 \tag{7.22}$$

Using condition (6.3), equation (7.21) becomes (for $i = 1, \ldots, N$):

$$\begin{aligned}\frac{dM_{zi}}{dt} = &(-\alpha\eta\delta M_{zi} + cA+\\ &\eta\alpha r_z M_{zi+1} + \eta\alpha l_z M_{zi+1})(1 - {M_{zi}}^2)\end{aligned} \tag{7.23}$$

The condition (7.22) implies that the equations above do not depend on M_x or M_y, i.e. they can be studied separately from (7.19)–(7.20).

I introduce the following notation to write (7.21) in matrix form:

DOI:10.15774/PPKE.ITK.2012.002

- $f(M_{zi}) = 1 - {M_{zi}}^2$ is a nonlinear-function of M_{zi}
- $\mathbf{M_z} = (M_{z1}, M_{z2}...M_{zN})^T$ is the vector containing the variables M_{zi} of the i^{th} STO
- $diag(f(\mathbf{M_z}))$ is a diagonal matrix containing the following elements:

$$diag(f(\mathbf{M_z})) = \begin{bmatrix} 1-{M_{z1}}^2 & 0 & ... & 0 \\ 0 & 1-{M_{z2}}^2 & ... & 0 \\ ... & ... & ... & ... \\ 0 & 0 & ... & 1-{M_{zN}}^2 \end{bmatrix}$$

- $\mathbf{w} = (A_1, A_2...A_N)^T$ is the vector containing the input current A_i of the i^{th} STO
- $\mathbf{P}$ represents the coupling tridiagonal matrix

$$\mathbf{P} = \begin{bmatrix} -\delta & r_z & 0 & ... & 0 & 0 \\ l_z & -\delta & r_z & ... & 0 & 0 \\ ... & ... & ... & ... & ... & ... \\ 0 & 0 & 0 & ... & l_z & -\delta \end{bmatrix}$$

With these notations (7.23) for $i = 1, \ldots, N$ can be written as:

$$\frac{d\mathbf{M_z}}{dt} = diag(f(\mathbf{M_z}))(c\mathbf{w} + \eta\alpha\mathbf{P}\mathbf{M_z}) \tag{7.24}$$

Using (7.24) one can investigate the equilibrium $\mathbf{M_z^*}$.
The trivial equilibria $M_{z\,i}^* = \pm 1, \forall i$ correspond to no oscillation in the STO array.
Other equilibrium points are obtained by

$$\mathbf{M_z^*} = -\frac{\gamma}{\alpha\eta}\mathbf{P^{-1}}\mathbf{w} \tag{7.25}$$

Remark: The equation above can be easily generalized for two dimensional arrays of STO having space-varying cellular architectures and fully connected architecture as well.

7.1.2 Cellular STO arrays with general interactions

I would like to examine synchronized oscillators in a network. In this case synchronization means, that after a transient behavior every oscillator will reach a limit cycle and they will oscillate with the same frequency and with arbitrary phase-shift compared to each other.

From the simulations, we can see that under certain conditions the third component converges to a fixed-point. I have to note the fixed point of M_{z_i} as K_i. This fixed point can be different for every oscillator, however the frequency of the synchronization is the same

DOI:10.15774/PPKE.ITK.2012.002

for every synchronized oscillator. (If M_z is not a constant, there is a small oscillation in this component, however this oscillation is 2-3 orders smaller, than the oscillation of the other components, so I approximated this as a constant. The magnitude of this oscillation can be seen on Figure 7.2) In Section 7.0.2 we have also seen that the spin of the uncoupled oscillators has a harmonic oscillation in two component, meanwhile the third component is a constant.

Knowing that K_i is a fixed point of the M_z component of the ith oscillator I can approximate the components of our system the following way: (because in this case the motion of the spin vector can be modeled as a circular motion in the first and the second components)
$M_{x_i} = \sqrt{(1 - {K_i}^2)}cos(\omega t + \phi_i)$,
$M_{y_i} = \sqrt{(1 - {K_i}^2)}sin(\omega t + \phi_i)$
$M_{z_i} = K_i$

where ω is the frequency of the oscillation and ϕ_i is the phase shift between the reference and the ith oscillator [2].

(Remark: I have selected this model based on the simulation, because I saw, that even when there is a small difference between different fixed-points K_i and K_{i+1} they will oscillate with the same frequency, when the coupling is 'strong enough' to compensate this small difference, if I consider all the fixed-points the same, i will get a less accurate, but much simpler model). However I will make this approximation, in case of two oscillators, but in that case this approximation is more accurate, than in general.)

Furthermore with the following transformation we can have a simpler system of our network: if we multiply Equation 7.20 by M_x and subtract this from Equation 7.19 multiplied by M_y we will have the following equation:

$$\begin{aligned} &\tfrac{dM_{x_i}}{dt}M_{y_i} - \tfrac{dM_{y_i}}{dt}M_{x_i} = \\ &-2\eta\delta M_{y_i}M_{x_i}M_{z_i} + \eta r_{z_i}M_{z_{i+1}}(M_{x_i}^2 + M_{y_i}^2) \\ &-\eta\alpha r_{y_i}M_{x_i}M_{y_{i+1}}(M_{x_i}^2 + M_{y_i}^2 + M_{z_i}^2) \\ &+\eta\alpha r_{x_i}M_{y_i}M_{x_{i+1}}(M_{x_i}^2 + M_{y_i}^2 + M_{z_i}^2) \\ &-\eta r_{x_i}M_{z_i}M_{x_i}M_{x_{i+1}} - \eta r_{y_i}M_{z_i}M_{y_i}M_{y_{i+1}} \end{aligned} \tag{7.26}$$

Using my approximation, from this equation I can get, noting $\sqrt{(1 - {K_i}^2)}$ as P_i for simpler notation.

$$\begin{aligned} &{P_i}^2\omega = -2\eta\delta K_i{P_i}^2 cos(\omega t + \phi_i)sin(\omega t + \phi_i) \\ &+\eta r_{z_i}K_j{P_i}^2 - \eta r_y K_i P_i P_j sin(\omega t + \phi_i)sin(\omega t + \phi_j) \\ &-\eta\alpha r_{y_i}P_iP_j cos(\omega t + \phi_i)sin(\omega t + \phi_j) \\ &+\eta\alpha r_{x_i}P_iP_j sin(\omega t + \phi_i)cos(\omega t + \phi_j) \\ &-\eta r_{x_i}K_iP_iP_j cos(\omega t + \phi_i)cos(\omega t + \phi_j) \end{aligned} \tag{7.27}$$

[2] we can select the reference oscillator arbitrary

DOI:10.15774/PPKE.ITK.2012.002

or in an other form:

$$\begin{aligned} P_i\omega = -2\eta\delta K_i P_i \tfrac{sin(2\omega t+2\phi_i)-1}{2} \\ +\eta r_{z_i} K_j P_i - \eta r_y K_i P_i P_j \tfrac{cos(\phi_i-\phi_j)-cos(2\omega t+\phi_i+\phi_j)}{2} \\ -\eta\alpha r_{y_i} P_j \tfrac{sin(2\omega t+\phi_i+\phi_j)-sin(\phi_i-\phi_j)}{2} \\ +\eta\alpha r_{x_i} P_j \tfrac{sin(2\omega t+\phi_i+\phi_j)+sin(\phi_i-\phi_j)}{2} \\ -\eta r_{x_i} K_i P_i P_j \tfrac{cos(\phi_i-\phi_j)+cos(2\omega t+\phi_i+\phi_j)}{2} \end{aligned} \tag{7.28}$$

and using the Discribing function method on Equation 7.21, noting the phase shift between oscillator i and j as θ_{ij}:

$$\begin{aligned} \tfrac{\alpha\eta\delta K_i P_i - cA_i P_i - \eta\alpha r_{z_i} K_j P_i}{\eta P_j} \\ = sin(\theta_{ij})(\tfrac{-r_{x_i}-r_{y_i}}{2}) \\ +cos(\theta_{ij})\alpha K_i(\tfrac{-r_{x_i}-r_{y_i}}{2}) \end{aligned} \tag{7.29}$$

As we can see from the previous two equations, the coupling of the third coordinate does not depend on the phase-shift, it depends only on the coupling strength of the third coordinate. This means, that this coupling will not effect the phase-shift between the oscillators. As I have shown in Section 7.1.1 this coupling will change only the 'height' of the planes of the limit-cycles (however I have to note, that the planes are depending on the other variables too). With this coupling (in the M_z component) one can change the heights of the planes independently from the phase shift, and the phase-shift will be determined by the coupling in the two other components as it can be seen on Figure 7.1.2

7.2 When r_x equals r_y

If the coupling strength in the first two components are equal ($r_x = r_y = r_{xy}$) we will have the following simpler equations:

$$\begin{aligned} \tfrac{\alpha\eta\delta K_i P_i - cA_i M_i - \eta\alpha r_z K_j P_i}{\eta P_j} \\ = -r_{xy} sin(\theta_{ij}) \\ -r_{xy} cos(\theta_{ij})\alpha K_i \end{aligned} \tag{7.30}$$

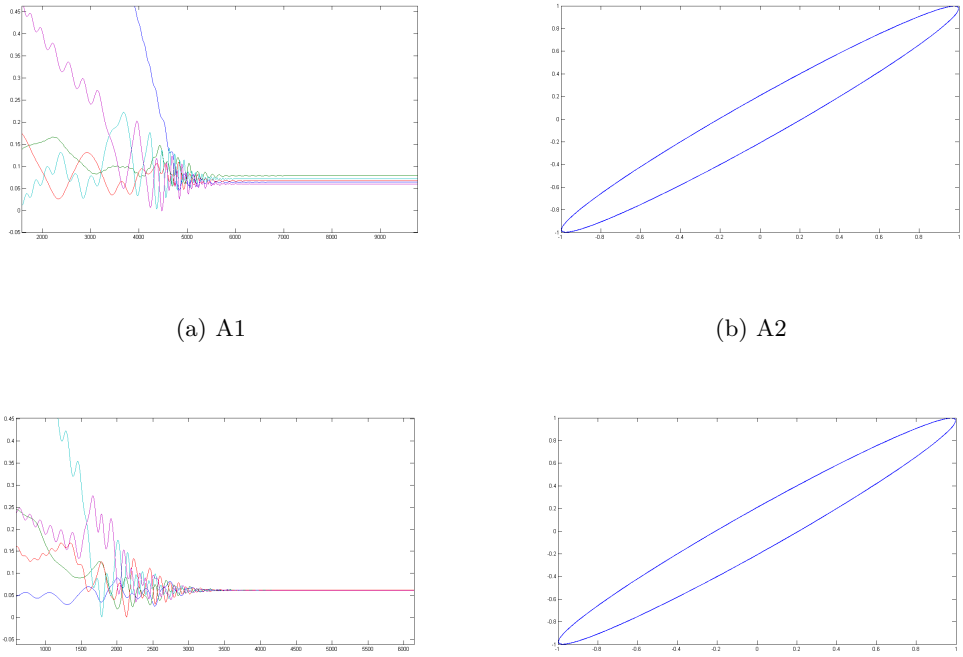
and:

$$\begin{aligned} P_i\omega = \eta\delta K_i P_i + \eta r_z K_j P_i \\ -\eta r_{xy} K_i P_i P_j cos(\theta_{ij}) + \eta\alpha r_{xy} P_j sin(\theta_{ij}) \end{aligned} \tag{7.31}$$

If we would like to put these equations in a more compact form generally for N oscillators we can introduce:

- $\mathbf{P} = (P_1, P_2 ... P_N)'$
- $\mathbf{K} = (K_1, K_2 ... K_N)'$
- $\mathbf{\Theta} = (0, \theta_{0-1} ... \theta_{0-N})'$

DOI:10.15774/PPKE.ITK.2012.002



(a) A1 (b) A2

(c) B1 (d) B2

Figure 7.4: On A1 we can see the initial fixed-point of the third component with some arbitrary selected weights and inputs in the array. As it can be these fixed point are slightly different fir every oscillator. On A2 the phase shift can be seen between the third and the second oscillators. According to the theory described in Section 7.1.2 I have altered the coupling strengths in the M_z components. As a result the fixed point of the third components with my calculated weights can be seen on B1. As it can be observed the planes of the oscillation are the same. Meanwhile I have changed the plane of oscillation using different weight in the third component the phase shift between the oscillators remained the same, as it is shown on B2.

- $\mathbf{A} = (A_1, A_2 ... A_N)'$

Where $\mathbf{K}$ is the vector of the fixed-point of the third component ($\mathbf{P} = \sqrt{1 - \mathbf{K}^2}$), $\mathbf{\Theta}$ is the vector of the phase-shifts between the oscillators. And we can introduce $\mathbf{R_{xy}}$ as the matrix of the coupling strengths, using cellular (but heterogeneous) connections this matrix will be a sparse matrix.

$$\mathbf{R_{xy}} = \begin{bmatrix} 0 & r_{xy_1} & 0 & ... & 0 & r_{xy_N} \\ r_{xy_1} & 0 & r_{xy_2} & ... & 0 & 0 \\ ... & ... & ... & ... & ... & ... \\ r_{xy_N} & 0 & 0 & ... & r_{xy_{N-1}} & 0 \end{bmatrix} \tag{7.32}$$

We can also introduce an other coupling matrix for the third coordinate $\mathbf{R}_z$. With this notation the equations will be the following:

DOI:10.15774/PPKE.ITK.2012.002

$$\eta\alpha\delta\mathbf{K}\circ\mathbf{P}-c\mathbf{A}\circ\mathbf{P}-\eta\alpha(\mathbf{R_z}\mathbf{K})\circ\mathbf{P}= \\ =-\eta\mathbf{R_{xy}}(\mathbf{P}\circ sin(\mathbf{\Theta}))-\eta\alpha(\mathbf{R_{xy}}(\mathbf{P}\circ(cos\Theta)))\circ\mathbf{P} \tag{7.33}$$

$$\mathbf{P}\omega=\eta\alpha\mathbf{K}\circ\mathbf{P}+\eta(\mathbf{R_z}\mathbf{K})\circ\mathbf{P} \\ -\eta\mathbf{R_{xy}}(\mathbf{P}\circ cos(\mathbf{\Theta}))\circ\mathbf{P}\circ\mathbf{K}+\eta\alpha\mathbf{R_{xy}}(\mathbf{P}\circ sin(\mathbf{\Theta})) \tag{7.34}$$

where $\circ$ denotes the Hadamard product between two vectors.

This way we can have a system with $2N$ independent equations and $2N$ unknowns. This way, with the numerical solution of equation 7.33 and 7.34 I can calculate the phase between the oscillators, without the time consuming simulation of the differential equations.

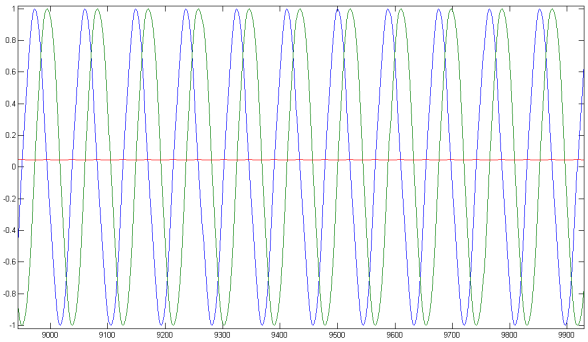


Figure 7.5: In this image we can see the oscillations in every component, when the coupling in M_x and M_y are different. As we can see the coupling in the third component is three-four orders of magnitude smaller, than the oscillations in the first two components, this way it's elimination will not affect the results significantly, and with the solution of equations 7.33 and 7.34 I can get the same results as I have obtained by the simulations.

7.3 Two coupled oscillator

7.3.1 Coupling only in the M_z component

As a special case I have examined two STOs coupled only by M_z. The equilibrium is determined by the following equation:

$$\begin{pmatrix} areM^*_{z\,1} \\ M^*_{z\,2} \end{pmatrix}=-\frac{\gamma}{\alpha\eta}\begin{pmatrix} -\delta & \frac{1}{l} \\ \frac{1}{r} & -\delta \end{pmatrix}\begin{pmatrix} A_1 \\ A_2 \end{pmatrix} \tag{7.35}$$

Given two different input current A_1 and A_2, the two uncoupled STOs have different angular frequencies (see equation (7.13)). By exploiting the couplings l_z and r_z it is possible to drive the STOs at the same angular frequency despite their different currents. This is easily achieved by imposing $M^*_{z\,1}=M^*_{z\,2}$, i.e. from the previous equation the following condition is derived:

DOI:10.15774/PPKE.ITK.2012.002

$$\frac{A_2}{A_1} = \frac{\frac{1}{r} + \delta}{\frac{1}{l} + \delta} \tag{7.36}$$

I have tested this case of two coupled STOs also with the simulator and the measurements of the simulations are matching the theoretical data.

Even in case of large STO arrays, it is possible to design the coupling matrix $\mathbf{P}$ (or equivalently its inverse $\mathbf{P}^{-1}$) and the input currents $\mathbf{w}$ in such a way that the STOs drop into different clusters. Each cluster results to be defined by the frequency of the oscillations within a group. This may enhance the synchronization between oscillators in the same groups and prevent the synchronization between oscillators in different groups. Similarly, with the design of the matrix $\mathbf{P}$ I can create different clusters of STOs, with different frequencies considering a cellular array with input (different currents) only on the border cells, when the range of the inputs is known.

A simple example containing six STOs and their division into two groups can be seen in Fig. 7.6. Fig. 7.7 shows the limit cycles associated to each cluster.

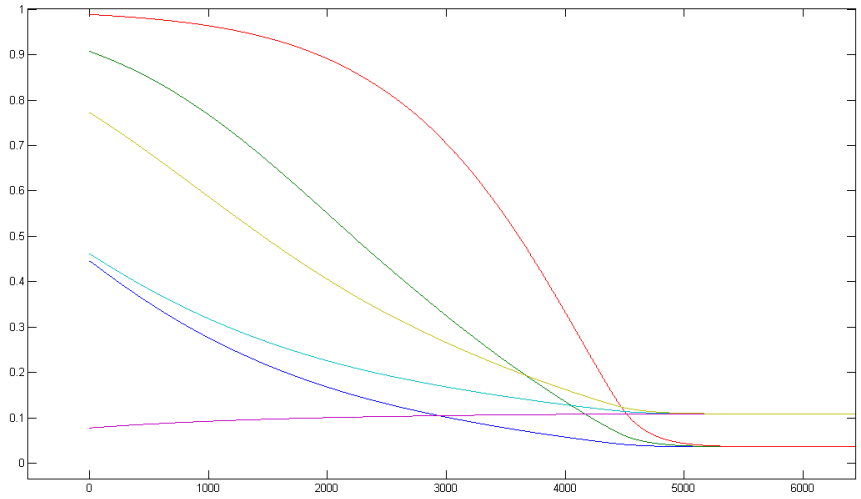


Figure 7.6: The $M_z(t)$ variables of six different STOs converge into two different clusters.

7.3.2 General coupling of two Oscillators

I would like to investigate the general coupling of two STOs, especially the effect of the input current and the coupling strength on the phase-shift after synchronization. As we have seen previously it is enough to solve equations 7.33 and 7.34 two calculate every parameters of an oscillator after the transient behavior. However these equations are not good to design an array with given properties, because they do not reveal the dependency of the phase-shifts on the input current. But as we can seen the input current can be found only in equation 7.33. If I substitute the proper parameters in this equation for the case of two, generally coupled oscillator we will have the following equations:

DOI:10.15774/PPKE.ITK.2012.002

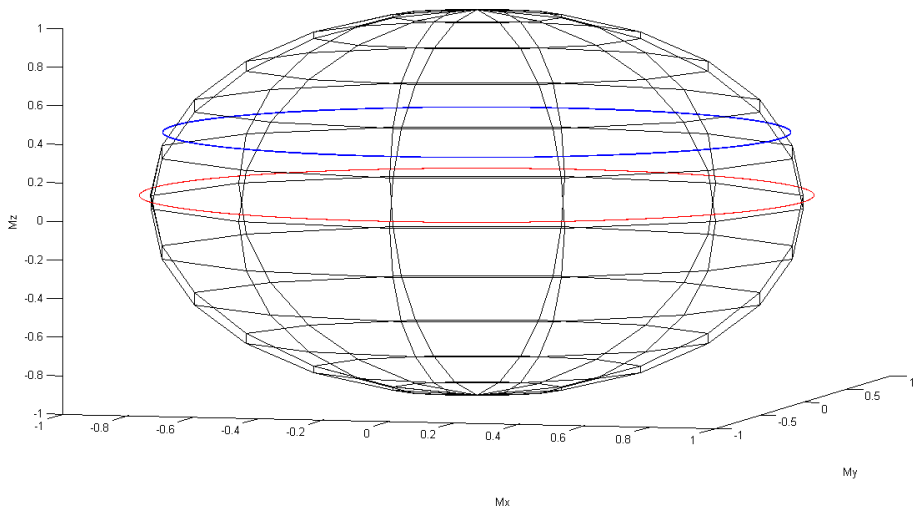


Figure 7.7: The two different limit cycles of the **M** vector can be seen according to the designed clusters. After transient time, the $M_z(t)$ variables of six different STOs converge into two different groups, i.e. the STOs synchronize in two clusters having different angular frequencies.

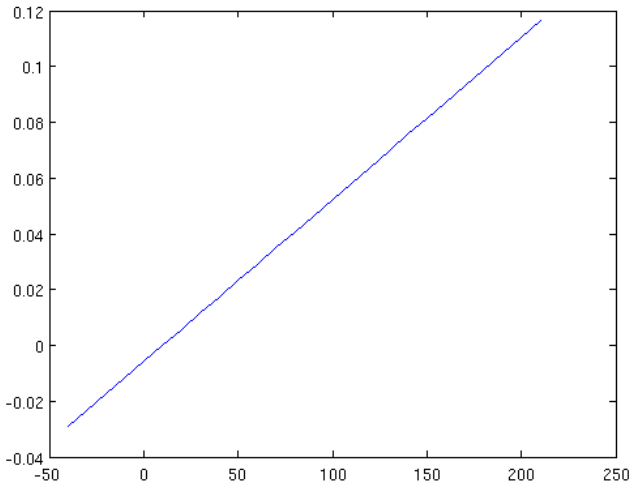


Figure 7.8: The dependency of the phase-shift based on the current. As it can be seen the dependency of the phase-shift is linear with respect to the change of the current difference between the oscillators.

DOI:10.15774/PPKE.ITK.2012.002

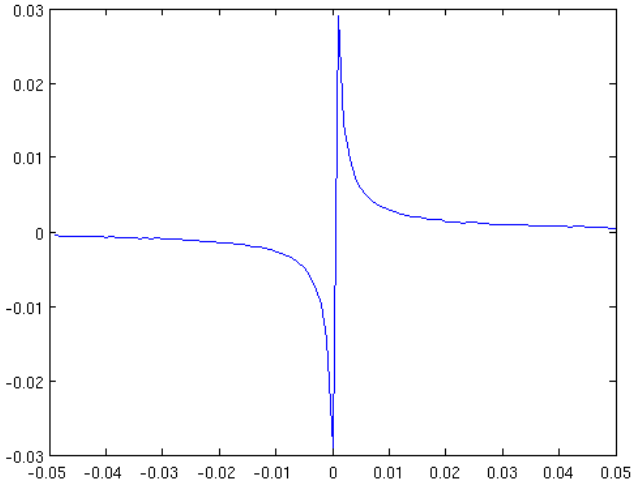


Figure 7.9: The dependency of the phase-shift based on the coupling strength. I have ignored the case when the coupling strength is zero. As it can be seen the dependency of the phase-shift is hyperbolic.

$$\begin{aligned} \alpha\eta\delta K_1 P_1 - cA_1 P_1 - \eta\alpha r_z K_1 P_1 = \\ sin(\theta) r_{xy}\eta P_2 + cos(\theta) r_{xy}\alpha\eta K_2 P_1 \end{aligned} \tag{7.37}$$

Because $\theta_{ij} = -\theta_{ji}$, for the other oscillator I can write:

$$\begin{aligned} \alpha\eta\delta K_2 P_2 - cA_2 P_2 - \eta\alpha r_z K_2 P_2 = \\ -sin(\theta) r_{xy}\eta P_1 + cos(\theta) r_{xy}\alpha\eta K_1 P_2 \end{aligned} \tag{7.38}$$

Since, with an oscillation with a common frequency the planes of oscillation is close to each other I can approximate $K_1 = K_2$ as K and $P_1 = P_2$ as P Noting the difference of the input current on the two oscillators as $A_2 - A1 = A_\Delta$ and subtract equation 7.3.2 from 7.3.2. We will have:

$$\theta = asin(\frac{A_\Delta}{2r_{xy}M_s}) \tag{7.39}$$

Since $\frac{A_\Delta}{2r_{xy}M_s}$ is between ± 0.1 according to the physical parameters and the sinus function is quasi-linear on this interval, I can also write:

$$\theta = \frac{A_\Delta}{2r_{xy}M_s} \tag{7.40}$$

This way I can examine the phase-shift as a function of the coupling strength and the input current, this equation is also suitable to design phase shift for previously given coupling strengths and/or input currents [8]. As it can be seen on Figure 7.3.2 and 7.3.2 the phase-shift depends linearly on the input current and hyperbolically on the coupling weight.

DOI:10.15774/PPKE.ITK.2012.002

7.4 Applications of spin torque oscillators

In this section I will show two simple examples, how a two dimensional cellular array of spin torque oscillators based on the dynamics described in equation (7.33) and (7.34) can be used for edge and change detection.

I can easily define a mapping between the pixel intensities of the image and the input currents of the oscillators. This way the input current of every oscillator will be proportional to the intensity of a pixel on a two dimensional grayscale image. Using (7.33) and (7.34) I can calculate the coupling weights in the array in a way, that after the synchronization of the array (when all oscillators are synchronized), the phase shifts between the oscillators will depend only on the input current differences between neighboring oscillators.

Although the differential equations in Chapter 6 and 7 do not contain noise and their investigation was done without considering a stochastic part, the simulations in this chapter were examined with noise in the differential equation and also when the parameters of the oscillators were altered by a stochastic noise. This results were qualitatively the same with or without the noise, however the quantitative investigation of the noise is out of the scope of this dissertation and requires further simulations.

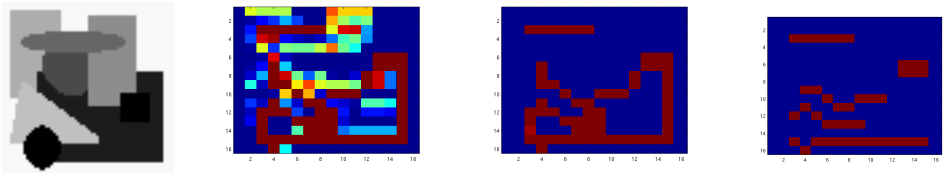
However there are many open questions about the feasibility and applicability of spin devices the development of spin-torque nano devices is constantly in progress. Although the largest network implemented contains only 5 coupled spin-torque oscillators and it is not feasible to implement devices with 16×16 oscillators, and create architectures useful in solving practical problems, the coupling models and the macro-models of the oscillators will remain the same even for larger networks. These smaller networks and preliminary results can confirm the applicability of spin-torque dynamics in computation and with simulations I can check how larger networks (even with 64×64 or more oscillators) would operate. This investigation can give a boost to the design of spin devices and hopefully the physical and engineering problems of implementing large networks of oscillators will be overcame in the following years.

7.4.1 Application example: edge detection

Using this coupling I can detect the color differences between neighboring oscillators. With the tuning of the coupling weights I can also adjust the level how the phase shift will depend on the intensity differences and also on the spatial density of intensity differences. This means I can detect intensity changes on images considering not only the differences in values but spatial changes as well. This gives possibility to perform edge detection on grayscale images and considering the spatial changes I can implement horizontal or vertical edge detection as well. A simple example of this task can be seen on Fig. 7.10

The computation can be done with the usage of STOs exclusively: without using any CMOS logic or non nano-devices.

DOI:10.15774/PPKE.ITK.2012.002



(a) Input (b) Intermediate result (c) Thresholded output (d) Horizontal edge detection

Figure 7.10: We can see the input image on the upper left figure. On figure (b) we can see an intermediate result, the synchronization of the oscillators. I have selected the first (most upper left oscillator as a reference) the pseudo colors are reflecting the phase shifts between the oscillators, the blue oscillators are synchronized in phase with the first oscillator. The red oscillators have a phase-shift around 12 degrees. On figure (c) we can see a thresholded version of figure (b). On the last image we can see a different spatial coupling, which detects the horizontal edges only.

7.4.2 Application example: spatial change detection

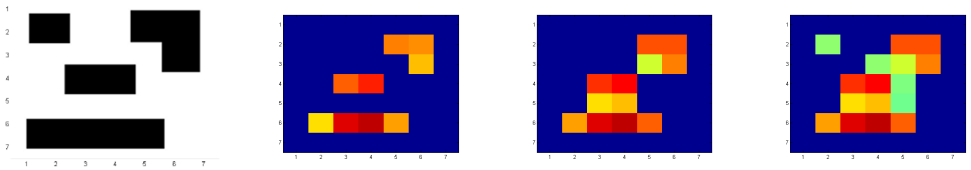
In case of object detection [3] and in general tasks of image processing, one extremely important question is the handling of noise. In case of object detection/classification not only the the intensity of the noise will determine the output, but also the topography of the noise: a group of low level noises in close proximity could be an object, because one can presume that the appearance of the noise at different pixels is independent from each other. Because of this phenomena we will have to process not only the intensity differences but also the topology of different intensities at the same times. Based on equation (7.33) and (7.34) I have designed a simple array, which is taking account both values with one computation: the output is determined not only by the difference level, but also how far the dissonant pixels are. The sensitivity of the network can be tuned by the coupling weights. By decreasing the weights the amount of intensity difference will be more determining, and by increasing the weights the effect of the pixels in proximity will be more determining, and the computation will consider the topology of the objects. Different outputs of this array can be seen on Figure 7.11 with different coupling weights.

As it can be seen this way the network sensitivity to the noise can be set. And it can be set according to the change level and according to the distance. Based on this example a cellular array of spin oscillators could be used for noise reduction, filtering, change detection or edge detection in image processing tasks. The implementation of change detection for grayscale images is straightforward when the input current is proportional to the color intensity.

Based on this example and the equations, a cellular array of spin oscillators could be

[3] even in case of binary object

DOI:10.15774/PPKE.ITK.2012.002



(a) Input (b) Output 1 (c) Output 2 (d) Output 3

Figure 7.11: We can see the input image on the upper left figure, and on the other figures the outputs with different coupling weights are shown. The weights were 0.01, 0.003 and 0.0004. As it can be seen with weights 0.01 the differences were calculated based on individual pixels, and objects with are of one pixel are considered as noise. With weights 0.003 the to objects in the bottom are considered as one object, white pixels with area of 1 are considered as noise. With weights 0.0004 all the three objects on the left are considered as one object.

used for noise reduction, filtering, change detection or edge detection in image processing tasks. The implementation of change detection for grayscale images is straightforward when the input current is proportional to the color intensity. Oscillatory behavior could be used also in associative memories as it was shown for general oscillators in [76], [9].

DOI:10.15774/PPKE.ITK.2012.002

Chapter 8

Summary

8.1 Methods used in the experiments

Unfortunately stochastic sampling is too complex to create a thorough theoretical examination and estimate bounds how an implementation would perform on a many-core architecture. Also the number of existing architectures is tremendous, this way it is almost impossible to create theories covering every architecture and problems.

To avoid this, I have introduced a general model of the stochastic optimization, which is able to mimic all the important characteristics. These models reveal different properties of stochastic optimization in different depths.

I have expanded these models to practical problems to show they are capable of simulating practical problems, too, apart from the theoretical results. Through this I have linked the general models to some practical problems and case studies. I have shown through several case studies (similarly to how it is usually done in the literature) how cellular implementations and topographic algorithms can be used in practice.

I have selected two algorithms to measure the performance of the cellular implementation. The stochastic sampling plays a crucial role in both of the methods. These two problems are the genetic algorithm and the particle filter algorithm. The test problems for the genetic algorithm were the N-Queen, Knapsack and Travelling Salesman problems. All of these are NP-hard, and general problems, and they can be extended to various practical problems.

To average out the randomness in the algorithms I have repeated all the experiments at least 1000 times, this way I can have reliable results. I have also measured the variance between the 1000 averages and it could be neglected.

Apart from this the most important part of the experiments was the architecture. Because the commercial computers are usually not many-cores and the existing many-core architectures show extreme diversity regarding the basic architecture (FPGAs, GPUs) I have implemented a virtual cellular machine, that mimics all the important characteristics. In this architecture the processing units are placed on a two-dimensional rectangular grid.

DOI:10.15774/PPKE.ITK.2012.002

This architecture and the measurements were implemented in Python.

I also wanted to measure the results on an existing chip and architecture. I chose the $Xenon_v3$ device, because I have had some experience about its capabilities and programming. The programming of the device can be done in Assembly language.

The other part of this dissertation, the investigation of the synchronization of Spin-torque oscillators contains mostly theoretical results for this reason the necessary devices, simulation and experiments was limited. But it is always a useful extension to support theoretical results by simulations and/or measurements. This was the reason why I have created a general simulator, which is capable to simulate the behavior of arbitrary STO arrays with various parameters. The implementation was done in MATLAB, C and PYTHON languages. The topography of the STO array can be arbitrary: we can simulate even globally coupled (in spite of the underlying physics is unfeasible) arrays. I have received the macro-model of the oscillators from physicists from the University in Notre Dame.

This simulator was a useful tool to test the two computational case-studies of STO array and the theoretical results in simulations (even if there were no possibilities to measure them in practice with an actual device).

8.2 New scientific results

1. Thesis:

1.1. Thesis:

I have investigated through general models, how the biology-motivated, localized stochastic selection affect the diversity and the quality of the generated sets. I have shown that by selecting optimal parameters (neighborhood radius, mutation factor) the quality of the generated sets (in case of the general models) is comparable to the quality of the global selection method, meanwhile the localized method can easily mapped into a many-core architecture and fits perfectly on multi-parallel, cellular devices, this way it can be executed with significant speedup. I have shown a way, how a non-topographic algorithm -stochastic selection- can be transformed into a topographic problem, and in what kind of advantages this can result.

I have introduced, the biology-inspired, localized selection and I have compared it to its original, global counterpart through two general (and some problem dependent) models. I have done the comparison through general models, and from these models I have specified some practical problems by modifying the characteristics of these models. These general models are able to mimic the most important characteristics of stochastic selection. Through them I have shown, how the localized selection can substitute its global counterpart, and I have shown through simulations and measurement how the information propagation speed amongst elements in the selected set depends on the tuning of the

DOI:10.15774/PPKE.ITK.2012.002

neighborhood radius.

I have also shown, that the localized version is not a brand new, completely different method, it can be considered as a generalization of the global method, where we can set the speed of information propagation amongst the elements through the neighborhood radius. (If the neighborhood radius is larger than the maximal distance between any two selected elements, we will have the same algorithm as the global method; and if the neighborhood radius is set to zero we will implement the original simple Monte Carlo method, where there are no selection mechanism at all.)

I have also investigated, how the neighborhood radius can affect the diversity of a set and the distribution of the weights of the selected elements (exploitation/exploration ratio): this phenomena can be seen on Fig. 2.2.

1.2. Thesis:

I described general rules and guidelines about mapping the cellular genetic algorithm on a CNN architecture. I have also implemented this modified, cellular version of algorithm on a general virtual machine and an existing cellular, many-core system: on the $Xenon_v3$, CNN chip. I have measured the efficiency of this implementation through simulations and measurements.

I have shown through three different case studies (the N-queen, the Knapsack and the Travelling salesman problem), how a CNN implementation of the genetic algorithm can solve difficult optimization problems in an efficient and elegant way with power consumption in the milliwatt range. The implementation gives a possibility to execute one iteration of the genetic algorithm in milliseconds (for the exact problems: $1884, 2982, 1373$ iterations per second could be executed for the N-queen, Knapsack and Travelling salesman problems) on the $Xenon_v3$ architecture. This execution times are orders of magnitudes better than other current results. I have compared these times with similar current results. In case of the Knapsack problem the fastest current result, what I have found for a problem with same complexity needed 0.92 second, which is much larger and would not be fair to be compared with the speed of the CNN implementation. In case of the Traveling Salesman problem I have compared my method with a CPU-GPU implementation, where the execution time was 0.20 seconds, meanwhile the power consumption of the GPU-card they have used (an NVIDIA GTX 280) is 310 W , not considering the additional consumption of the computer and this can not be compared to the power consumption of the CNN chip, which consumed less then 5 milliwatts and the execution time was 93 milliseconds. This shows that a two-times speed up could be reached meanwhile the power consumption remained 1500 times lower than the CPU-GPU implementation. This shows, how this implementation could be useful in tasks where complex optimization tasks have to be solved with lower power consumption within a strict time limit like in case of naviga-

DOI:10.15774/PPKE.ITK.2012.002

tion, speech-processing, parameter optimization or in case of other problems generated by mobile devices.

Apart from the implementation I have describe a general method, how a cellular genetic algorithm can be implemented on a multi-layered CNN architecture. This description can give help and hints to implement the algorithm on other similar devices. The sketch of the general implementation divided into different steps and layers can be seen on Fig 3.3.

1.3. Thesis:

I have shown how the localized selection mechanism can be used in case of a dynamic state estimator: in the particle filter algorithm. I have introduced the Cellular Particle Filter algorithm. I have implemented the algorithm on a virtual cellular machine and also on an existing architecture, the $Xenon_v3$ *chip.*

Also in the case of this problem I have investigated three case studies, to show how the algorithm can be implemented and utilized. Also the simulation results and the real measurements on the $Xenon_v3$ architecture were examined. I have shown in case of Hidden Markov models, how the cellular algorithm can be used for state estimation, even in problems when neither the Kalman-filter nor the Baum-equation can be applied, because our state-transition is non-linear and the stat-space is continuous (or infinite). I have shown that this algorithm approximates well not only the expected value of the hidden state, but also the distribution of hidden states as well, this can be used later on to approximate probabilities and conditional expectations as well. I have shown through commonly used case studies that the cellular version can be implemented with a faster execution speed, it is easily scalable and with the proper setting of the parameters we can approximate the hidden state with a lower error rate. I have also shown through the same case studies the reason behind this lower error rate: I have measured that in the cellular, topographic method the diversity of the particles is higher than in the global method, (meanwhile the approximation is the same) because the information propagates amongst the particles in a cellular, local way.

Some example results can be seen in Table 5.1, for a commonly used model. As it can be seen from the results in case of certain parameters, the error of the cellular method is lower (especially, if the umber of the particles is relatively high).

2. Thesis:

2.1. Thesis:

I gave an analytic solution, a closed formula to calculate the equilibrium of the oscillation of the differential equation of spin torque oscillators. Thanks to this solution the equilibrium of the oscillation can be calculated as a function of the geometry, input current and magnetic permeability of the oscillator, without the time-consuming numerical

DOI:10.15774/PPKE.ITK.2012.002

simulation of the differential equation.

The simple macro-model of spin-torque oscillators can be be described as:

$$\frac{d\mathbf{M}}{dt} = \gamma(\mathbf{M} \times \mathbf{H}) - \gamma\alpha\mathbf{M} \times (\mathbf{M} \times \mathbf{H}) - \gamma A\mathbf{M} \times (\mathbf{M} \times \mathbf{S}) \tag{8.1}$$

Where $\mathbf{M}$ is the spin-vector, $\mathbf{H}$ is the magnetic field, $\mathbf{S}$ is the direction of the input current, A is the strength of the input current $\times$ notes the cross product between the vectors, γ and α are physical constants,the gyromagnetic-constant and the magnetic efficiency.

From this equation I have derived the fix-point of the third component of $\mathbf{M}$ (M_z), which determines the plane of the oscillation:

$$M_z^* = -\frac{A}{\alpha\, M_s(N_x - N_z)} = \frac{A}{\alpha\, M_s\, \delta} \tag{8.2}$$

Where $\mathbf{N} = (N_x, N_y, N_z)^T$ is a vector determining the geometry of the oscillator, with the following constraints: $\delta = N_z - N_x = N_z - N_y$.

From this the oscillation can be reduced to a simple harmonic, circular motion, with the following parameters:

The frequency of the oscillation:

$$\omega = \frac{\gamma\, A}{\alpha} \tag{8.3}$$

The amplitude of the oscillation:

$$B = \sqrt{(1 - M_z^{*2})} \tag{8.4}$$

I have checked these result with the simulator, and the results match the analytical solutions.

Because from an engineering point of view we are interested in only the equilibrium and not at all in the transient states, this analytic solution can be a huge help in the investigation of spin-torque oscillators, because thanks to this solution there is no need for the time-consuming numerical simulation of the oscillations, we can derive them easily and efficiently by a closed formula.

2.2. Thesis:

I created a mapping between a current-coded input and the phase-coded output of a cellular STO array. I gave an approximation with the help of the "harmonic Balance technique" to the phase-shift of synchronized,weakly coupled STOs in a general network. I have shown that, when the coupling strength in the x and y components are the same, this method is an exact solution and not an approximation. And even when the coupling strength in the two components are different the error of the approximation is four orders smaller, than the error of the amplitude of the spin. With the help of this method the phase

DOI:10.15774/PPKE.ITK.2012.002

shifts in a general network of oscillators can be calculated, without solving a difficult and complex differential equation system. We can calculate the phase shifts by solving only an algebraic equation system. With the help of these equations I have given an example how an architecture can be built, when the processing elements are small oscillators, consisting only a few atoms (or only one), and I have also given two simple examples which kind of topographic calculations can be done by such a device.

The connection, and this way the synchronization between weakly coupled spin torque oscillators happens through the magnetic filed:

$$\begin{aligned} \mathbf{H}_{eff} &= -\mathbf{H}_i + \mathbf{G}_j \\ &= -M_s \begin{pmatrix} N_x M_{xi} \\ N_y M_{y_i} \\ N_z M_{zi} \end{pmatrix} + M_s \begin{pmatrix} C_x^j M_{xj} \\ C_y^j M_{y_j} \\ C_z^j M_{zj} \end{pmatrix} \end{aligned} \tag{8.5}$$

Where the coupling strength between oscillators i and j is determined by the vector $\mathbf{C}$. And from this the effective magnetic field can be calculated: $\mathbf{H}_{eff}$.

The approximation of any general network of STOs can be done by these equations, I have obtained with the usage of spectral techniques ("Harmonic Balance" and "Describing Function") on the differential equation system. This way we can easily calculate the phase-shift between synchronized oscillators.

We can introduce the following vectors:

- $\mathbf{P} = (P_1, P_2 ... P_N)'$
- $\mathbf{K} = (K_1, K_2 ... K_N)'$
- $\mathbf{\Theta} = (0, \theta_{0-1} ... \theta_{0-N})'$
- $\mathbf{A} = (A_1, A_2 ... A_N)'$

Where $\mathbf{K}$ contains the plane of oscillations, also the fixed-point in the third component, $\mathbf{A}$ is the strength of the input current of the oscillators and ($\mathbf{P} = \sqrt{1 - \mathbf{K}^2}$) and $\mathbf{\Theta}$ is the relative phase difference between the oscillators.

$\mathbf{R_{xy}}$ is the coupling matrix in x and y components, and similarly $\mathbf{R_z}$ is the coupling weights in the third component.

The equation system obtained with this notation:

$$\begin{aligned} \eta\alpha\delta\mathbf{K} \circ \mathbf{P} - c\mathbf{A} \circ \mathbf{P} - \eta\alpha(\mathbf{R_z K}) \circ \mathbf{P} = \\ = -\eta\mathbf{R_{xy}}(\mathbf{P} \circ sin(\mathbf{\Theta})) - \eta\alpha(\mathbf{R_{xy}}(\mathbf{P} \circ (cos\Theta))) \circ \mathbf{P} \end{aligned} \tag{8.6}$$

$$\begin{aligned} \mathbf{P}\omega = \eta\alpha\mathbf{K} \circ \mathbf{P} + \eta(\mathbf{R_z K}) \circ \mathbf{P} \\ -\eta\mathbf{R_{xy}}(\mathbf{P} \circ cos(\mathbf{\Theta})) \circ \mathbf{P} \circ \mathbf{K} + \eta\alpha\mathbf{R_{xy}}(\mathbf{P} \circ sin(\mathbf{\Theta})) \end{aligned} \tag{8.7}$$

DOI:10.15774/PPKE.ITK.2012.002

I have shown, that if the coupling strengths in the C_x and C_y components are equal, the previous equation system is an exact solution of the phase shift defined by the differential equation system of the STOs. Also when the coupling strength are not equal in the two components a minor oscillation will appear in the third, z component, but the magnitude of this oscillation is four orders smaller as the amplitude of the oscillations in the two other components.

I have also derived the simpler form of the previous equation system for only two oscillators:

$$\theta = \text{asin}(\frac{A_\Delta}{2rM_s}), \tag{8.8}$$

where A_Δ is the different between the input current on the two oscillators, r is the coupling weight between the oscillators (which is proportional to the distance between the oscillators), M_s is a physical constant, the magnetic saturation.

From this equation in can be seen, that the phase shift between the oscillators depends linearly from the input current difference, and hyperbolically from the distance between the oscillators.

Based on these equations I have designed and simulated two examples, showing how the an cellular network of STOs can be used for computation. One example is a simple grayscale edge detection on two-dimensional input images. The other example is a more complex spatial-difference detection where not only the pixel differences but also the spatial distribution of the distances are considered during the difference calculation. This preliminary examples are showing how an STO network could be used for noise filtering and object segmentation.

8.3 Application of the results

The examples shown in the dissertation clearly highlight the main characteristics of the stochastic selection. We can claim that they are generally valid. Because of this, the methods and implementations are also valid, and the cellular, localized selection can be used as a substitution of the global method. Considering this, my method as an example can also help to show, how a non-topographic problem can be transformed into a topographic one, and how it can be solved by topographic algorithms on topographic architectures.

Not only the general method on the virtual machine, but also the implementation on the $Xenon_v3$ architecture can help to solve optimization problems. This way the commonly used (mostly in image processing tasks) CNN chips can also applied in tasks where parameter optimizations and/or state estimations are required. These topographic algorithm are ideal, where complex problems have to be solve with strict time limits and low power consumption. The extremely good low power consumption of the $Xenon_v3$ (under 20mW) could be ideal in mobile application such as robotics, mobile-communication or navigation.

DOI:10.15774/PPKE.ITK.2012.002

Also not only the implementation can be applied and useful, but the similar implementation of these (or related) algorithms on other devices and architectures can be crucial e.g: implementation on FPGA-s or GPU-s.

The architectures described in the second part of the dissertation could be useful in designing computers with extremely low power consumption. These simple computational models can be extended in cases where the storing essence of the information is the spin and not the charge. The cores of these architectures are feasible even when only a few atoms are used, this makes them as a true alternative for Beyond Moore's low computation. The analytic solution of the differential equation system could help the investigation of spin-torque oscillators. Apart from the theoretical results I have shown through two simple examples how an STO array can be used for simple computational problems. Although these networks are not feasible in these days because of the physical constraints, but I hope in the future I can measure the dynamics of STOs in practice and the development of STO devices it could speed up the process since only the equation system has to be solved instead of simulating a complex differential equation system.

In the dissertation I wanted to show, how obtaining the topographic constraints can modify the algorithm development, and what we can do to avoid these limitations, also how this limits can help to solve non-topographic problems. I hope in this way topographic thinking, and considering two dimensional constraints and the precedence of locality can help in the solution of new problem classes and in their mapping on many-core architectures as well.

DOI:10.15774/PPKE.ITK.2012.002

Chapter 9

Conclusion

In this dissertation I have shown through general models, how local selection can be applied instead of the global selection mechanism. I have shown, that the local selection is not a completely different or specialized algorithm, but a more generalized version of the global process, where by setting the neighborhood radius optimally the exploitation/exploration ratio of stochastic optimization can be set. In case of the case studies I have examined how the neighborhood radius can tune the speed of information propagation. This can result more diverse cohorts, helping us to avoid local extrema, which is one of the most serious problems in stochastic optimization.

The dissertation also shows how these general models can be applied in practice. In case of the practical problems I have considered how this method can be implemented on existing CNN chips and also I have shown a sample implementation on the $Xenon_v3$ architecture.

I examined the idea of local selection mechanisms in two practical algorithms: in the genetic algorithm and also with the particle filter algorithm. In both cases I have pointed out how easily this method is parallelizable and scalable. This means that in case of difficult problems we do not have to alter our method our increase the processing speed, the addition of extra computing cores – what can be done easily– is enough.

Later on it would be useful to examine other problems, problem classes which can be mapped ideally on topographic many-core architectures. I would also like to use this method in practice, there are promising problems from the field of tracking, navigation and speech processing. Mobile applications with strict time constraint could be the best environment, because the low power consumption and high computational power of the CNN chip could result optimal implementation of such algorithms. It would be also useful to enhance the current implementation in those parts which are not ideal because of the design of the $Xenon_v3$ architecture. Implementation on other architectures like FPGA, GPU implementations could be also useful in case of complex problems from the field of financial mathematics.

In the second part of this thesis I have also investigated special topographic architec-

DOI:10.15774/PPKE.ITK.2012.002

tures. I briefly describe, how topographic architectures can be applied beyond Moore's law. I have shown a simple example how spin (the magnetization vector of a nano-device) can be used instead of the charge as a unit to store information dynamically. I have shown who the steady-state frequency and plane of oscillation of an STO can be calculated analytically from the differential equation system. I have done an approximation about a general coupled network of STOs, with this giving a mapping between the frequency-encoded input and the phaseshift-encoded output of an STO array.

I have examined how the phase shift between two oscillators depend on the input current difference and the coupling weight. these equations can be used to design simple coupling of oscillatory network.

I have shown through two examples how these methods can be used in practice. I have calculated and simulated two different networks for topographic, image processing tasks: one for simple edge detection and one for a complex spatial change detection.

In this dissertation I wanted two show, how we have to consider topographic constraints during algorithm development. What we can do to avoid these constraints, and also how they can help us to design topographic algorithms to solve even non-topographic problems. In the following year topographic thinking and topographic algorithm will be extremely important, because on the nanoscale only cellular, local couplings are feasible in an efficient way.

DOI:10.15774/PPKE.ITK.2012.002

Chapter 10

Appendix

Appendix A CNN summary

This section is based on the more detailed description of Tamás Roska and Leon Chua [77]. The images were also taken from their book.

Cellular nonlinear/nanoscale/neural networks (CNNs) are built of parallel processing structures with nonlinear dynamic units (cells) arranged topographically in a single or multi-layer regular grid. The connectivity of each processor is local in space. The programming (and the computation) of the network is done by defining the local interactions. These programming operators are called templates.

The time-evolution of the analog transient, "driven" by the template operator and the cell dynamics, represents the elementary computation in CNN (results can be defined both in equilibrium or non-equilibrium states of the network). The standard CNN equation contains only first order cells placed on one layer of a regular grid and the interconnection pattern is linear.

A cellular wave computer architecture that includes CNN dynamics as its main instruction, is the CNN Universal Machine (CNN-UM). The CNN-UM makes it possible to efficiently combine analog array operations with local logic. Since the reprogramming time is approximately equal to the settling time of a non-propagating analog operation it is capable of executing complex analogic (analog and logic) algorithms. To ensure stored programmability, a global programming unit is added to the array and for an efficient reuse of intermediate results, each computing cell is extended by local memories. In addition to local storage, every cell might be equipped with local sensors and additional circuitry to perform cell-wise analog and logical operations.

Using the CNN-UM we are able to design and run analog and logic CNN wave algorithms. It is known that CNN-UM is universal as a Turing Machine andwe as a nonlinear operator. Therefore many problems can be solved by this machine.

Its structure suggests using it for image processing in numerous applications. Beyond the classical image processing there are a lots of new methods of solving problems based

DOI:10.15774/PPKE.ITK.2012.002

on partial differential equations which need huge computational power. Most of these kind of problems can be transformed into CNN algorithm too.

Another important scope is the biological modeling. The researchers found in early times that CNN can be used for modeling some parts of the human visual system, mainly the outer retina. Recently, a multilayer, multichannel retina model has been developed [78]. Because of the simple structure of CNN, it is realizable in real hardware.

Standard CNN Dynamics

The cellular nonlinear network (CNN) is a locally connected, analog processor array which has two or more dimensions. A standard CNN architecture consists of an $M \times N$ rectangular array of cells $C(i,j)$ with Cartesian coordinate $(i,j) i = 1..M, j = 1..N$ (Figure 10.1)

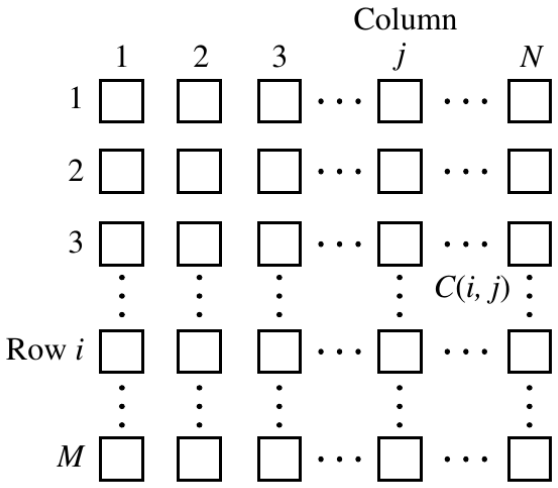


Figure 10.1: MxN representation of CNN structure

MxN representation of CNN structure.

The sphere of influence, $S_r(i,j)$, of radius of r of cell $C(i,j)$ is defined to be the set of all neighboring cells satisfying the following property:

$$S_r(i,j) = \left\{ C(k.l) | \max_{1 \le k \le M, 1 \le l \le N} \{ |k-i|, |l-j| \} \le r \right\} \tag{10.1}$$

where r is a positive integer.

CNN Templates

The state of a cell depends on interconnection weights between the cell and its neighbors. These parameters are expressed in the form of the template:

DOI:10.15774/PPKE.ITK.2012.002

$$A = \begin{bmatrix} a_{i-1j-1} & a_{i-1j} & a_{i-1j+1} \\ a_{ij-1} & a_{i1j} & a_{ij+1} \\ a_{i+1j-1} & a_{i+1j} & a_{i+1j+1} \end{bmatrix} B = \begin{bmatrix} b_{i-1j-1} & b_{i-1j} & b_{i-1j+1} \\ b_{ij-1} & b_{i1j} & b_{ij+1} \\ b_{i+1j-1} & b_{i+1j} & b_{i+1j+1} \end{bmatrix} z = z_{ij} \tag{10.2}$$

A template has two main parts, a feedforward and feedback matrixes. These parts are called A and B templates. The z on Equation (10.2) is the offset (bias) term. In the simplest case the template is given by 19 numbers, 9 feedback, 9 feedforward and one bias terms. This 19 number template is an elementary operation of CNN-UM and codes a complex spatial-temporal dynamics. An analogical algorithm might contain some templates and logical operations. The following differential equation system describes the dynamics of the network:

$$\begin{aligned} C_x \frac{dv_{x_{ij}}(t)}{dt} = -\frac{1}{R_x} v_{x_{ij}}(t) + \sum_{C(k,l)\in S_r(i,j)} A_{ij;kl} v_{y_{kl}}(t) + \\ \sum_{C(k,l)\in S_r(i,j)} B_{ij;kl} v_{u_{kl}}(t) + z_{ij} \\ v_{y_{ij}}(t) = f(v_{x_{ij}}(t)) = 1/2(\left|v_{x_{ij}}(t)+1\right| - \left|v_{x_{ij}}(t)-1\right|), \\ i = \overline{1,M}; j = \overline{1,N} \end{aligned} \tag{10.3}$$

The figure of the given function can be seen on Figure 10.2. This is called standard nonlinearity.

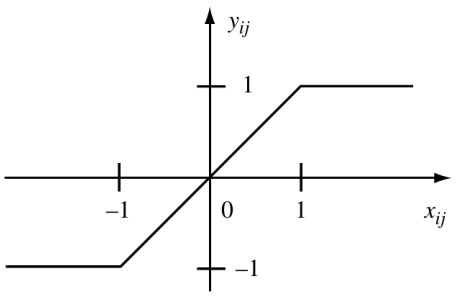


Figure 10.2: The output characteristic function of a CNN cell.

In the case where the values of $A_{ij;kl}$; $B_{ij;kl}$ do not depend on i and j, the template is space invariant. In most cases the value of the offset current does not depends on space $z_{ij} = z$. Because of the regular 2D shape of the CNN, the value of a cell can be represent by a pixel of a picture. This gray-scale value can be between white (-1) to black (1). Sometimes we use fixed state mask whose values allow or permit the change of the values of their cells. 3D CNN networks can connect like layers and this gives multi-layer CNN

DOI:10.15774/PPKE.ITK.2012.002

networks. Its differential equation is similar to Eq. (10.3):

$$C_{xm}\frac{dv_{x_{mij}}(t)}{dt} = -\frac{1}{R_{xm}}v_{x_{mij}}(t)+ \sum_{n=1}^{P}\Big(\sum_{C(k,l)\in S_r(i,j)} A_{mn;ij;kl}v_{y_{nkl}}(t) + \sum_{C(k,l)\in S_r(i,j)} B_{mn;ij;kl}v_{u_{nkl}}(t)\Big) + z_{mij} \tag{10.4}$$

where p is the number of layers, m is the current layer, and A_{mn} and B_{mn} give the connection between n and m layers. For the solution of a given example, we have to give the input U, $x(0)$ initial state and the templates with the algorithm. The result is Y after running the transient. In most cases we can work with predefined templates.

CNN Universal Machine

The CNN Universal Machine (CNN-UM) is based on a CNN. The architecture of and extended CNN-UM cell can be seen on Figure 10.3. This is the first programmable analog processor array computer with own language and operation system whose VLSI implementation has same computing power as a supercomputer in image processing applications [66]. The extended universal cells of CNN-UM are controlled by global analogic programming unit (GAPU), which has analog and logic parts: global analog program register, global logic program register, switch configuration register and global analogic control unit. Every cell has analog and logical memories as well.

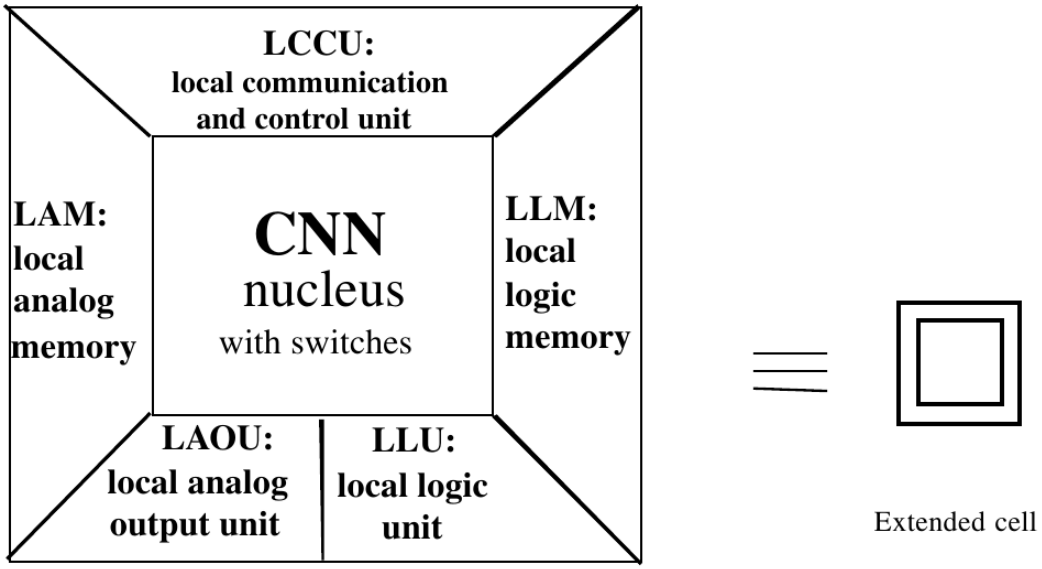


Figure 10.3: The architecture of a CNN Universal Machine cell

Appendix B Xenon Architecture

The $Xenon\_v3$ chip [38] is a 3D integrated scalable focal-plane processor array based on digital CNN technology. The chip contains 8×8 cells, processing cores. The chip contains

DOI:10.15774/PPKE.ITK.2012.002

On-chip sensors made by Indium-Gallium-Arsenide and each cell is prepared to process an 8×8 pixel array[1]. Altogether the chip can execute computations on a 64×64 image. The Neighboring cells are directly interconnected which gives the possibility to read from the memory of the neighboring cells/cores in one clock cycle. This Gives the possibility to implement operations based on large kernels (even 15×15 efficiently). The structure of the Xenon chip can be seen on Figure 10.4.

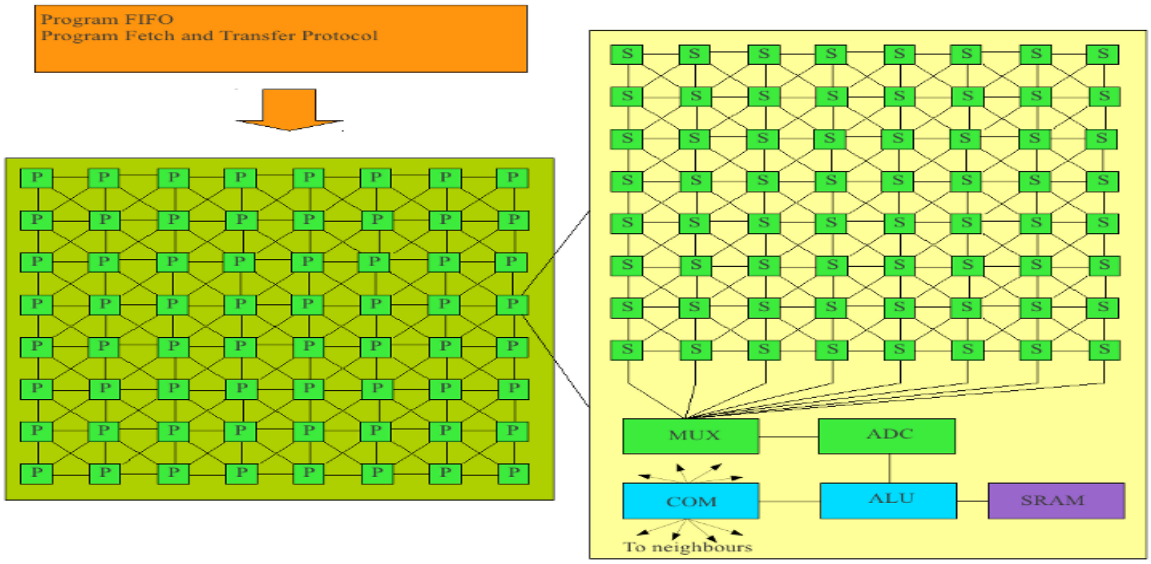


Figure 10.4: The schematic architecture of the $Xenon\_v3$ chip. The connection of the Xenon cores can be seen on the left side of the image (the cores are noted as c and the connections are represented by the lines). Every core contains an 8×8 focal plane sensor array, this can be seen on the right side of the image. Apart from the sensors every core contains a multiplexer, an analog-digital converter, an arithmetic logical unit, and a relatively large memory implemented by SRAMs.

The processing cells/cores can be operated by a $100Mhz$ core clock. Every processing unit has a relatively large (512 byte) memory that can be addressed both bit-wise and byte-wise and both signed and unsigned values can be handled in the data path. The structure of the processing cells can be seen on Figure 10.5.

Each of the cores has a digital arithmetic core that is capable of solving simple arithmetical tasks by 16 bit precision like: addition, subtraction, multiplication, comparison, absolute value calculation, barrel shifting. The arithmetic unit contains a 9×9-bit signed hardware multiplier, a 24-bit accumulator and 8-bit general purpose registers. The structure of the arithmetic unit can be seen on Figure 10.6.

Each core has also a dedicated morphology unit, which can process 8 elements (8 pixels) in a parallel way. The morphological unit is able to execute bit-wise logical operations like: AND, OR, XOR, NOR.... The structure of the morphological unit can be seen on Figure

[1]this amount is scalable

DOI:10.15774/PPKE.ITK.2012.002

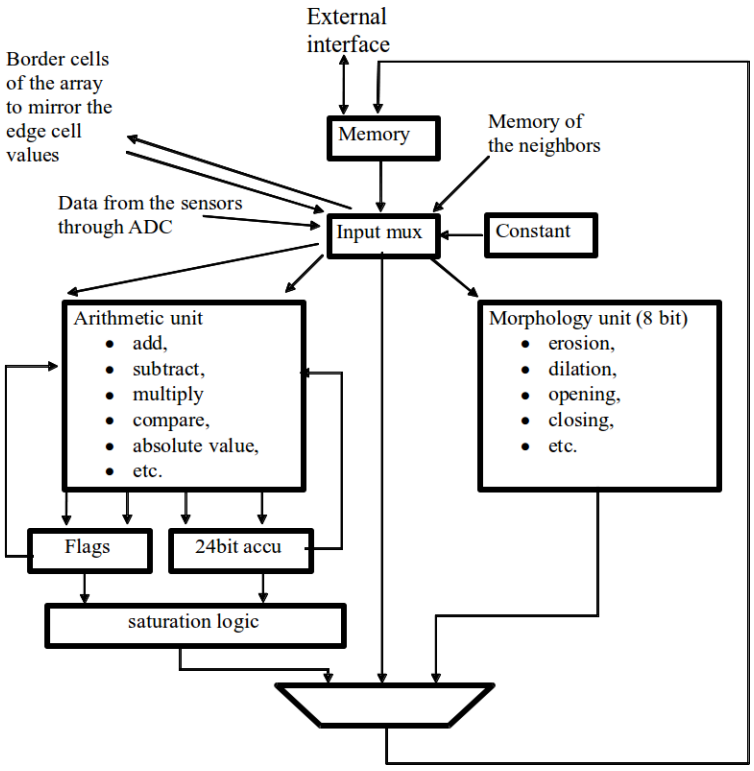


Figure 10.5: The schematic architecture of a $Xenon\_v3$ core.

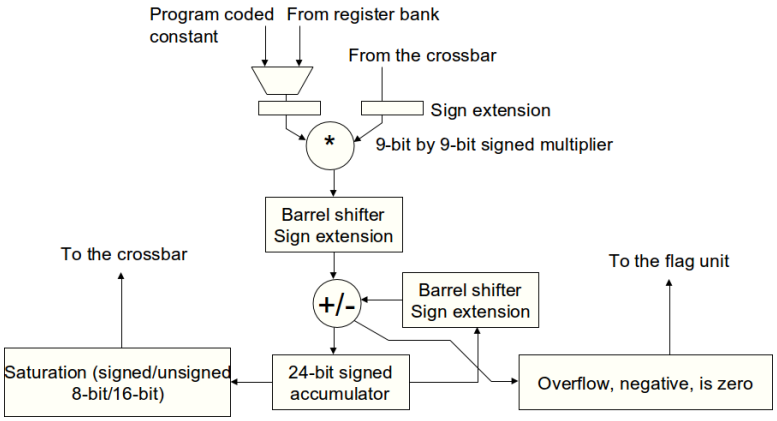


Figure 10.6: The structure of the arithmetic unit of a Xenon core

DOI:10.15774/PPKE.ITK.2012.002

10.7.

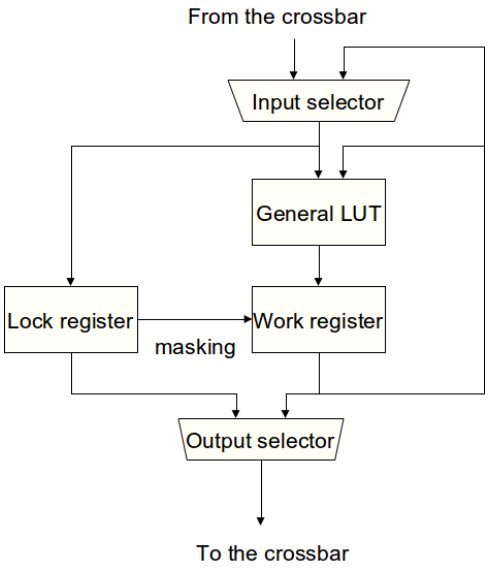


Figure 10.7: The structure of the morphological unit of a Xenon core

The cores are also integrated with a 8×8 InGaAs/InP diode focal plane sensor array [39] with 1 nAmp to 1 uAmp range integration type diode interface through which the input can easily be uploaded directly to the memory of the processing cores.

The Program scheduler of the chip contains a 1024 instructions deep FIFO (with stall, standby, full-empty flags). The chip contains approximately 600,000 transistor (40% for logic, 40% for memory, 20% for digital analog conversion). The size of the chip is $0.23mm^2$ with ($5 \times 5mm^2$ die size). The chip was manufactured with UMC 0.18 micron technology. The peak power consumption of the chip is between $20 - 100mW$. The Input/Output speed of the chip is 32 bit at $80MHz$. The device designed for on-chip image acquisition and preprocessing, local adaptation (masking based on image content) and Pixel-wise operations (absolute value calculation, gamma correction, squaring).

Some example operation which can be implemented in an efficient way on the device: linear convolution (filtering up to 15×15 kernel), nonlinear operations (like: grayscale morphology, median, other rank-order filters, anisotropic diffusion) binary morphology (dilation, erosion, skeleton (up to 15×15 kernel)).

The Xenon chip is capable of operating with 30 giga operation per second (GOPS), which is an amazing performance considering its size and power consumption [2]. Which makes it ideal not only for image processing but also to solve optimization task in mobile environments.

[2]less than 20 mW approximately 5mW in average

DOI:10.15774/PPKE.ITK.2012.002

Appendix C Genetic Algorithm

This section is based on [22], [24] and [79].

Genetic algorithm (GA) is a heuristical search method that mimics natural evolution. This heuristic is routinely used to generate useful solutions to optimization and search problems. Genetic algorithms can generate solutions to optimization problems using techniques inspired by natural evolution, such as inheritance, mutation, selection, and crossover. However these operations are not always used together in every version of genetic algorithms.

In a genetic algorithm, a population of strings (called chromosomes, genotype or genomes), which encode candidate solutions representing them in the state space of an optimization problem, evolves toward better solutions. Traditionally, solutions are represented in binary as strings of 0s and 1s [3], but other encodings are also possible. The evolution usually starts from a population of randomly generated individuals and happens in generations. In each generation, the fitness of every individual in the population is evaluated (based on heursitcis), multiple individuals are stochastically selected from the current population (based on their fitness), and modified (recombined and possibly randomly mutated) to form a new population. The new population is then used in the next iteration of the algorithm. Commonly, the algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population. If the algorithm has terminated due to a maximum number of generations, a satisfactory solution may or may not have been reached.

Genetic algorithms find application in bioinformatics, phylogenetics, computational science, engineering, economics, chemistry, manufacturing, mathematics, physics and other fields.

A typical genetic algorithm requires:

- a genetic representation of the solution domain (the state space of the problem),
- a fitness function to evaluate the solution domain (a heuristical evaluation of the genomes).

A standard representation of the solution is as an array of bits. Arrays of other types and structures can be used in essentially the same way. The main property that makes these genetic representations convenient is that their parts are easily aligned due to their fixed size, which facilitates simple crossover operations. Variable length representations may also be used, but crossover implementation is more complex in this case. Tree-like representations are explored in genetic programming and graph-form representations are explored in evolutionary programming; a mix of both linear chromosomes and trees is explored in gene expression programming.

[3] hence these are the straight forward represnetation on a common computer

DOI:10.15774/PPKE.ITK.2012.002

The fitness function is defined over the genetic representation and measures the quality of the represented solution. The fitness function is always problem dependent. For instance, in the knapsack problem one wants to maximize the total value of objects that can be put in a knapsack of some fixed capacity. A representation of a solution might be an array of bits, where each bit represents a different object, and the value of the bit (0 or 1) represents whether or not the object is in the knapsack. Not every such representation is valid, as the size of objects may exceed the capacity of the knapsack. The fitness of the solution is the sum of values of all objects in the knapsack if the representation is valid, or 0 otherwise. In some problems, it is hard or even impossible to define the fitness expression; in these cases, interactive genetic algorithms are used.

Canonical operations of the Genetic Algorithm

The following four methods are the fingerprints that determine genetic algorithms They can be found in every variants and alterations.

- 1- Initialization of the population
- 2- Fitness (weight) calculation for every entity
- 3- selection and recombination
- 4- mutation

And after the fourth step the iteration of steps 2,3,4 till a previously given time constraint, or until we can find the optimal solution. (the N-queen problem uses the second version because the fitness of the optimal solution is known. This weight is not known in every case, however this will not effect or change the steps in the iterations)

During each successive generation, a proportion of the existing population is selected to breed a new generation. Individual solutions are selected through a fitness-based process, where fitter solutions (as measured by a fitness function) are typically more likely to be selected. Certain selection methods rate the fitness of each solution and preferentially select the best solutions. Other methods rate only a random sample of the population, as the former process may be very time-consuming.

The next step is to generate a second generation population of solutions from those selected through genetic operators: crossover (also called recombination), and/or mutation.

For each new solution to be produced, a pair of "parent" solutions is selected for breeding from the pool selected previously. By producing a "child" solution using the above methods of crossover and mutation, a new solution is created which typically shares many of the characteristics of its "parents". New parents are selected for each new child, and the process continues until a new population of solutions of appropriate size is generated. Although reproduction methods that are based on the use of two parents are more "biology

DOI:10.15774/PPKE.ITK.2012.002

inspired", some research [80] [81] suggests that more than two "parents" generate higher quality chromosomes.

The detailed description of the operators are the following:

Initialization

Once the genetic representation and the fitness function are defined, a GA proceeds to initialize a population of solutions (usually randomly) and then to improve it through repetitive application of the mutation, crossover, inversion and selection operators. Initialization At this step we will create random strings (with genomes according to the problem representation). Each genome codes a possible solution candidate, representing a point in the state-space Our aim is to create the most diverse population possible and cover the entire stat-space. The optimization of this operation is always problem dependent. The convergence of the algorithm does not depend on the initial population if the mutation rate and the number of entities is relatively high, in this case the algorithm will always converge to the optimal solution with probability one.

Initially many individual solutions are (usually) randomly generated to form an initial population. The population size depends on the nature of the problem, but typically contains several hundreds or thousands of possible solutions. Traditionally, the population is generated randomly, allowing the entire range of possible solutions (the search space). Occasionally, the solutions may be "seeded" in areas where optimal solutions are likely to be found.

Fitness Calculation

We will calculate a fitness (weight) value for every genome. This value represents a distance between our point and an optimal point in the state space. The metric of the distance is based on a problem dependent heuristic.

The selection of this heuristic is a key part of the algorithm, and an extremely difficult task. however we do not want to investigate this problem in this article. There are published and well known fitness functions, heuristic metrics for a large number of problems. e will calculate a fitness (weight) value for every genome. This value represents a distance between our solution candidate and an optimal solution in the state space. The metric of the distance is based on problem dependent heuristics. Selecting an appropriate metric is a key question, I base our choice on well-known, published suggestions.

Selection of Parents

In case of the 'general' GA selection is calculated globally. During this step we select the genome we want to conserve, and use in the next iteration, and overwrite the unnecessary elements.

For selection the following methods are used:

DOI:10.15774/PPKE.ITK.2012.002

Deterministic Sampling The best genome, genomes with highest fitness values are selected. The genomes are ordered, and after this the first X % is selected, and the other genomes are cleared. X is previously declared as a parameter.

The commonly used sampling methods are:

-Stochastic Universal Sampling also known as roulette wheel sampling.

During the generation of the new population ever genome has a probability to be chosen relational to its fitness.

-Stochastic Tournament Selection

A composite method from the previous two version. First we select n genomes with Stochastic Universal Sampling, and after this from the n elements we will chose k genomes (usually k=1 or k=2) and these genomes will be conserved as parents for the next population. We will repeat the sampling until the new population will reach the size of the previous population

-Remainder Stochastic Sampling

Also a composite method created from the first two selection mechanism. We normalize the weights for every entity, in a way that the sum of the weight will be number of genomes in the population. the selection is based on these normalized weights. First we will select every genomes deterministically as many times as the integer part of their fitness. After this we will create an other weight from the remainder part of the fitness values and perform a stochastic resampling based on them. e.g.: after the normalization the fitness value is 2.65, it means that the genome will be used twice in the gene pool of the next iteration, and it will have a 0.65 likelihood (not probability) for the stochastic selection.

Recombination

There are two different versions of this operator: single-point and multi-point recombinations.

In case of a single-point mutation we will choose one gene in the genome, and until this point all the genes will be taken from one parent and after this point all the genes will be copied from the other parent. Single-point mutation can only be used in case of two parents.

Multi-point mutation is a repeated version of single-point mutation for more parents. we will select more genes in the genomes, and the genes between this selection are determined by one parent. In an extreme case it is also possible the that all the genes are inhereted from a different father.

Mutation

Mutation performs a random jump in the state space of problem in the neighborhood of a candidate solution

DOI:10.15774/PPKE.ITK.2012.002

There exist many mutation variants, which usually affect one or more loci (genes or components) of the individual. The mutation randomly modifies a single solution whereas the recombination acts on two or more parent chromosomes.

There are also two different, simple version for this step: We have an upper limit for the number of changing genes in a genome, or we do not have an upper limit,and the number of changing genes is arbitrary. For every gene we have a probability, that the selected gene will change its value. There are a large number of heuristic and non heuristic version for improving the mutation, however all these steps are usually based on the problem, and its representation. The mutation can be value dependent or independent, in the value dependent case the new gene will be an altered version of the previous gene (larger/smaller with a predefined value) in case of the value independent case the new gene will be a randomly selected value.

Termination Criterion

These processes ultimately result in the next generation population of chromosomes that is different from the initial generation. Generally the average fitness will have increased by this procedure for the population, since only the best organisms from the first generation are selected for breeding, along with a small proportion of less fit solutions, for reasons already mentioned above.

Although Crossover and Mutation are known as the main genetic operators, it is possible to use other operators such as regrouping, colonization-extinction, or migration in genetic algorithms [82].

This generational process is repeated until a termination condition has been reached. Common terminating conditions are:

- A solution is found that satisfies minimum criteria
- Fixed number of generations reached
- Allocated budget (computation time/money) reached
- The highest ranking solution's fitness is reaching or has reached a plateau such that successive iterations no longer produce better results
- Manual inspection
- Combinations of the above

Appendix D Particle Filter

This entire chapter and the description of the theory were based on the lecture notes of Ramon van Handel used at Princeton University [47]. I will also refer to [48] and [49] for the relevant mathematical theory.

DOI:10.15774/PPKE.ITK.2012.002

Particle filter (PF), also known as a sequential Monte Carlo method (SMC), is a sophisticated model estimation technique.

Particle filters are usually used to estimate Bayesian models in which the latent variables are connected in a Markov chain — similar to a hidden Markov model (HMM), but typically where the state space of the latent variables is continuous rather than discrete, and not sufficiently restricted to make exact inference tractable (as, for example, in a linear dynamical system, where the state space of the latent variables is restricted to Gaussian distributions and hence exact inference can be done efficiently using a Kalman filter). In the context of HMMs and related models, "filtering" refers to determining the distribution of a latent variable at a specific time, given all observations up to that time; particle filters are so named because they allow for approximate "filtering" (in the sense just given) using a set of "particles" (differently weighted samples of the distribution).

Particle filters are the sequential (online) analogue of Markov chain Monte Carlo (MCMC) batch methods and are often similar to importance sampling methods. Well-designed particle filters can often be much faster than MCMC. They are often an alternative to the Extended Kalman filter (EKF) or Unscented Kalman filter (UKF) with the advantage that, with sufficient samples, they approach the Bayesian optimal estimate, so they can be made more accurate than either the EKF or UKF. However, when the simulated sample is not sufficiently large, they might suffer from sample impoverishment. The approaches can also be combined by using a version of the Kalman filter as a proposal distribution for the particle filter

Markov Process

Memoryless stochastic processes are called Markov processes, they all have the markovian property:

$$P(X_{t+1} \in S | X_0 X_1 \dots X_t) = P(X_{t+1} \in S | X_t) \tag{10.5}$$

The probability that during the next experiment S will occur depends only on the current state X_t, and independent from the previous states. [4]

This models are commonly used in modeling. The main reason behind this is that our physical world is memoryless. The physical laws are memoryless and they are all defined only by current states and the previous states have no effect[5] on the current state transition. This model is also a general and strong tool to develop generally applicable mathematical models and robust methods.

The state transition can be defined by a transition matrix:

$$Q_{ij} = P(X_{t+1} = j | X_t = i) \tag{10.6}$$

[4] X_t can be a multidimensional variable.

[5] directly

DOI:10.15774/PPKE.ITK.2012.002

Q_{ij} defines a probability, that our process will step from state i to state j. When Q is independent from t we call the process a homogeneous Markovian process, if it depends on t the process is heterogeneous.

Here we will restrict ourselves to homogeneous Markovian processes. [6]

Hidden Markov Models

A discrete time Markovian process containing two state variables is called a Hidden Markov Model. One of this processes is the observable and the other is the hidden state [7]. The process can be noted by (x_t, y_t), where x is the hidden and y is the observable state.

We assume that x_t follows Markovian dynamics given by the recursion

$$x_{t+1} = \varphi(x_t, e_1(t+1)) \tag{10.7}$$

where $\varphi : \mathbf{R}^2 \to \mathbf{R}$ is a fixed (non-linear) function and $e_1(t)$ is an IID (Independent identically distributed) sequence independent of the initial state x_0. The transition density of x_t will be denoted by $q(v, w)$, that is,

$$P(x_1 \in A | x_0 = v) = \int_A q(v, w) dw$$

for sets $A \subset \mathbf{R}$ and for any $v \in \mathbf{R}$.

The observations are assumed to be a function of the system state blurred by additive noise, that is

$$y_t = \psi(x_t) + e_2(t) \tag{10.8}$$

for some (non-linear) function $\psi : \mathbf{R} \to \mathbf{R}$ and an IID noise sequence $e_2(t)$, independent of $e_1(t), x_0$. We denote by $r(w)$ the density function of the law of $e_2(t)$, i.e.

$$P(e_2(t) \in A) = \int_A r(w) dw, \quad A \subset \mathbf{R}.$$

Then $r(y_t - \psi(x))$ is the "likelihood function" expressing how x is likely to be the state of the system at time t when the observation is y_t.

The applications of this kind of processes is extremely versatile, however they can be divided into different subgroups:

- processes where the original state [8] can not be observed directly, only through distortion and noise. E.g: the general channel model: In this case our aim is to approximate X based on the values of Y.
- The other case is, when we would like to estimate and predict Y [9], however the state transition of Y can not be seen or derived directly. Sometimes it is useful to

[6] however the theories and methods are easily applicable to heterogeneous processes too

[7] which usually can not observed directly

[8] X_t that we would like to process or estimate

[9] the current and previous values can be observed

DOI:10.15774/PPKE.ITK.2012.002

introduce a hidden value X in the background, which has a simple state transition and also effects/determine the values of Y E.G: A stock process (Y) can have difficult dynamics, and usually we are interested in stock prices, however to introduce the manufacturing process (X) will simplify the state transition and determine [10] also the stock price.

Filtering Equation-Filtering Recursion

The pair of processes (x_t, y_t) is a typical example of a hidden Markov model, see [83]. In applications one tries to compute

$$E[x_t|y_t, \ldots, y_1],$$

which is the best least-squares estimate of the hidden state x_n based on the information $y_k, \ldots, y_1$ available at time t.

To do this, one needs the conditional law of x_n knowing $y_k, \ldots, y_1$, that is

$$\mu_{n|k}(A) := P(x_n \in A|y_k, \ldots, y_1).$$

for $A \subset \mathbb{R}$.

To simplify our notation we will use $\mu_{k|k}$ (approximate the current state) if $k \geq 0$ we can call $\mu_{k|k}$=μ_k.

According to this notation:

If we examine μ_k and μ_{k-1} we can obtain a recursion using Bayesian updating to calculate μ_k, namely

$$\mu_{k+1}(A) = \int_A \frac{r(y_{k+1} - \psi(u))}{M_{k+1}} \int_{\mathbf{R}} q(v, u)\mu_k(dv)du \tag{10.9}$$

where M_{k+1} is the normalizing constant

$$M_{k+1} := \int_{\mathbf{R}} r(y_{k+1} - \psi(u)) \int_{\mathbf{R}} q(v, u)\mu_k(dv)du$$

ensuring that μ_{k+1} is a (conditional) probability.

With the initial condition: $\mu_0(y_0, A) = \int_A \frac{r(y_0 - \psi(u_0))}{M_0} \int_{\mathbf{R}} q(v, u)du$ Based on this equation we can derive a filtering recursion for any arbitrary HMM. This recursion is extremely important in practice, because this way we can minimize our calculations in every step and this gives us hope to implement online methods, capable of working real-time.

Discrete state space

When our state space is finite and discrete it is easy to calculate the probability of the hidden state based on the observations, because the integral in (10.9) will be a discrete summation and this can be calculated easily. We can also derive a simpler closed formula

[10] with an additional noise

DOI:10.15774/PPKE.ITK.2012.002

for the recursion equation based on the state transition and observation matrices, called the Baum equation [84], determining the hidden values based on the previous observations.

If we would like to obtain not only one value, but more consecutive values in a trajectory we can use the recursion [11] in both direction, resulting the forward-backward or the Baum-Welch algorithm [85]. If we would like to maximize the conditional probability based on Transition counting and occupation time for the whole trajectory, considering the maximization of not one element at a time, but the probability of a trajectory we will have the famous Viterbi algorithm [86].

Although in this case all the calculations are relatively simple and can be described as matrix products and summations, in case of extremely difficult models, where in spite of the discrete state space the problem representation can be too large to compute [12]. In this case we have to find an other method to avoid the calculation in every state (usually the state transition affects only a few states, the state transition matrix is usually sparse; and the observation/noise affects more states)

Linear Gaussian state space models

At some cases we can handle problems with infinite and continuous state spaces. In these special case our state space is general, but our model is a special subclass of all the possible problems. If we now that our model is a linear combination of Gaussian processes [13]:

$$X_k = a + AX_{k-1} + B\xi_k \tag{10.10}$$

and

$$Y_k = b + CX_{k-1} + D\eta_k \tag{10.11}$$

In the previous equations X_k represents the hidden state, Y_k the observations, ξ_k and η_k are independent random variables with normal distribution and A, B, C, D, a and b are arbitrary constants in the model (or constants matrices or vectors in higher dimension). Because the linear combination of Gaussian processes will result a Gaussian process, we are working in a subset of all the possible processes [14] Every conditional distribution in our model will be Gaussian, all we have to do is to approximate the properties of this Gaussian distribution. The only parameters are the mean vector and the covariance matrix. Based on the dimension of the problem we are facing a subclass of the general problem again, where our aim is not to approximate the hidden values directly but first to approximate all the parameters. This is indeed a finite dimensional problem and it can be calculated by a finite dimensional recursion, because our approximations are linear and considering

[11] or the Baum equation

[12] also in case of discrete,but infinite state-spaces

[13] or can be approximated with linear Gaussian processes

[14] the subset of Gaussian processes

DOI:10.15774/PPKE.ITK.2012.002

the previously described properties and the filtering equation will lead us to the famous Kalman-filter [87].

If we have a nonlinear model, but it can be linearized in certain intervals, or can be approximated as a series of linear functions we can use the extended Kalman-filter [88] instead of the general filtering equation (10.9).

Hidden Markov Model and Particle Filtering

It is often the case that (10.9) is difficult to calculate and q is not available in an explicit form. Then, instead of the unfeasible numerical integration, one often resorts to particle filters which provide an effective method for computing (10.9) and thus also

$$E[x_t|y_t,\dots,y_1] = \int_{\mathbf{R}} u\,\mu_t(du).$$

There are various implementations of particle filters, but they usually contain the four steps explained below. A more detailed description can be found in [55] or [52]. We will simulate K particles whose trajectories follow the state dynamics but are subject to a selection mechanism based on observations.

Initialization (step 0)

We first draw initial values

$$\xi_0^i = \zeta_i \tag{10.12}$$

where $i = 1,\dots,K$ is the index for the different particles and ζ_i is an IID sequence with a law of our choice, a guess for the true law of x_0. [15]

The time parameter t will range over $t = 0, 1, \dots, T$, where T is the time horizon for our observations.

Each particle will have a weight that represents its accuracy/distance from the real state. We first set $w_0^i = 1/K$ so all the initial weights are assumed to be equal. [16]

Error calculation (step 1)

Let us assume that we have already generated the trajectories of the particles ξ_s^i, $i = 1,\dots,K$ for $s \le t$. We now have to calculate the "fitness" of each particle, based on the next observation. We set $E_t^i = y_t - \psi(\xi_t^i)$ for the "error" of particle i in the light of the observation y_t.

Resampling (step 2)

Set new weights for every particle according to the likelihood function above:

$$\tilde{w}_t^i = r(E_t^i) = r(y_t - \psi(\xi_t^i)), \tag{10.13}$$

[15] Here I am not dealing with the question of how to choose this law optimally.

[16] We do not have any observation about the system yet, hence this seems to be a reasonable choice

DOI:10.15774/PPKE.ITK.2012.002

then normalize the weights:

$$w_t^i := \frac{\tilde{w}_t^i}{\sum_{j=1}^N \tilde{w}_t^j}. \tag{10.14}$$

We choose our new set of particles by drawing from our sample based on the discrete probability distribution determined by the weights w_t^i:

$$\hat{\xi}_t^i := \xi_t^{\eta_i}, \tag{10.15}$$

where the η_i are IID random variables taking values in $\{1, \dots, K\}$ such that $P(\eta_i = j) = w_t^j$, for $i, j = 1, \dots, K$.

Iteration (step 3)

Go one step ahead with all the particles according to the rule in model (10.7):

$$\xi_{t+1}^i := \varphi(\hat{\xi}_t^i, e_1^i(t+1)). \tag{10.16}$$

here the $e_1^i(t+1)$ are K IID copies of $e_1(t+1)$.

Often it is more efficient to generate ξ_{t+1}^i using a law other than that of $e_1(t+1)$ and then correct this bias by assigning appropriate weights to these new particles. With this trick (called "importance sampling") we can "lead" the particles towards a prescribed region where we are the most interested in their behavior, see e.g. [55]. This method fits our algorithm described below and it points to numerous further enhancements. However, these are not in the scope of the present dissertation so for simplicity we stick to the above setting and assume that we generate $e_1^i(t+1)$ according to the law of $e_1(t+1)$.

After this we return to steps $1-3$ and eventually ξ_t^i for all the time points $0, \dots, T$ and for all $i = 1, \dots, K$ will be generated. If T and K are large enough then the (discrete) distribution of the particles ξ_T^i, $i = 1, \dots, K$ is hoped to approximate μ_T fairly well. (This can be rigorously proven under suitable assumptions, see e.g. [62, 63, 64, 65, 89].) In formulas, denoting by δ_w the one-point mass at w, we have

$$\frac{1}{K} \sum_{i=1}^K \delta_{\hat{\xi}_T^i} \approx \mu_T.$$

Hence one may take

$$\frac{1}{K} \sum_{i=1}^K \hat{\xi}_T^i \approx E[x_T | y_T, \dots, y_1]. \tag{10.17}$$

to estimate the hidden state x_T.

[10]

DOI:10.15774/PPKE.ITK.2012.002

The Author's Publications

[1] **András Horváth** and Miklos Rasonyi. Topographic implementation of particle filters on cellular processor arrays. *Elsevier Signal Processing (Submitted)*, 2012.

[2] **András Horváth** and Miklos Rasonyi. Implementation of cellular genetic algorithms on a cnn chip: Simulations and experimental results. *International Journal of Circuit Theory and Applications*, 2012.

[3] Elen An, **András Horváth**, and Loeckx Dirk. 3d cardiac strain estimation using spatio-temporal elastic registration : in-vivo application. In *IEEE International Ultrasonics Symposium*, Beijing,China, November 2008. IEEE.

[4] **András Horváth** and Miklos Rasonyi. Fast computation of particle filters on processor arrays. In *12th IEEE international workshop on cellular nanoscale networks and applications*, Berkeley, California, February 2010. IEEE.

[5] **András Horváth** and Miklos Rasonyi. Maximum likelihood estimation of quantized gaussian autoregressive processes using particle filters with resampling. In *International Symposium on Nonlinear Theory and its Applications*, Palma di Majorca, Spain, October 2012.

[6] Horváth Anna, Tornai Gábor, **Horváth András**, and Cserey György. Gpu-ra adaptált celluláris particle filter. *OTDK paper*, 2011.

[7] György Csaba, Matt Pufall, Dmitri Nikonov, George Bourianoff, **András Horváth**, Tamás Roska, and Wolfgang Porod. Spin torque oscillator models for applications in associative memories. In *13th IEEE international workshop on cellular nanoscale networks and applications*, Turin, Italy, August 2012. IEEE.

[8] **András Horváth**, Fernando Corinto, György Csaba, Wolfgang Porod, and Tamás Roska. Synchronization in cellular spin torque oscillator arrays. In *13th IEEE international workshop on cellular nanoscale networks and applications*, Turin, Italy, August 2012. IEEE.

[9] Tamás Roska, **András Horváth**, Attila Stubendek, Fernando Corinto, Gyorgy Csaba, Wolfgang Porod, Tadashi Shibata, and George Bourianoff. An associative memory with oscillatory cnn arrays using spin-torque oscillator cells and spin-wave

DOI:10.15774/PPKE.ITK.2012.002

interactions – architecture and end-to-end simulator. In *13th IEEE international workshop on cellular nanoscale networks and applications*, Turin, Italy, August 2012. IEEE.

[10] **András Horváth**, Tamas Roska, Attila Stubendek, Danny Voils, Fernando Corinto, Gyorgy Csaba, Wolfgang Porod, Tadashi Shibata, Dan Hammerstrom, and George Bourianoff. O-cnn vice spin torque oscillator cells and cellular spin-wave interactions in an associative memory. *IEEE Transactions on Nanotechnology*, 2012.

DOI:10.15774/PPKE.ITK.2012.002

Other Related Publications

[11] Robert B. Heckendorn Darrell Whitley, Soraya Rana. The island model genetic algorithm: On separability, population size and convergenc. *Journal of Computing and Information Technology*, 1998.

[12] Matthew Wolf. Global versus local optimization in allomorph selection. *International Journal of Intelligent Information Processing*, 2009.

[13] Mohammad Alrifai and Thomas Risse. Combining global optimization with local selection for efficient qos-aware service composition. *Proceedings of the 18th international conference on World wide web*, 2009.

[14] J. Dy Y. Guan, M. I. Jordan. In L. Getoor, and T. Scheffer. A unified probabilistic model for global and local unsupervised feature selection. *Proceedings of the 28th International Conference on Machine Learning (ICML)*, 2011.

[15] Efrén Mezura-Montes and Carlos A. Monterrosa Lópe. Global and local selection in differential evolution for constrained numerical optimizationm. *Journal of Computer Science and Technology*, 2009.

[16] Victor M. Brea1, Mika Laiho, David López Vilariño, Ari Paasio, and Diego Cabello. A binary-based on-chip cnn solution for pixel-level snakes. *International Journal of Circuit Theory and Applications*, 34(4):383–407, July/August 2006.

[17] Dániel Hillier, Viktor Binzberger, David Lopez Vilariño, and Csaba Rekeczky. Topographic cellular active contour techniques: theory, implementations and comparisons. *International Journal of Circuit Theory and Applications*, 34(2):183–216, July/August 2006.

[18] Vu Duc Thai and Pham Thuong Cat. Equivalence and stability of two-layered cellular neural network solving saint venant id equation. In *Proceedings of the 11th International Conference on Control Automation Robotics and Vision (ICARCV)*, pages 704 – 709, Singapore, 2010. IEEE, IEEE.

[19] Frank Werblin, Tamás Roska, and Leon O. Chua. The analogic cellular neural network as a bionic eye. *International Journal of Circuit Theory and Applications*, 323(6):541–569, November/December 1995.

DOI:10.15774/PPKE.ITK.2012.002

[20] John Henry Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology.* MIT Press, 1975.

[21] Nguyen Quang Huy, Ong Yew Soon, Lim Meng Hiot, and Natalio Krasnogor. Adaptive cellular memetic algorithms. *Journal of Evolutionary Computation*, 17(2), 2009.

[22] Enrique Alba and Bernabe Dorronsoro. *Cellular Genetic Algorithms - Series: Operations Research-Computer Science Interfaces Series, Vol. 42.* SpringerLink, 2008.

[23] Nguyen Quang Huy, Ong Yew Soon, Lim Meng Hiot, and Natalio Krasnogor. Convergence models of genetic algorithm selection schemes. *Lecture Notes in Computer Science - Parallel Problem Solving from Nature*, 866:119–129, 1994.

[24] Holger H. Hoos and Thomas Stützle. *Stochastic Local Search Foundations and Applications.* Morgan Kaufmann - Elsevier, 2004.

[25] Mariusz Nowostawski and Riccardo Poli. Parallel genetic algorithm taxonomy. In *Third International Conference Knowledge-Based Intelligent Information Engineering Systems*, pages 88–92, Adelaide, South Australia, 1999.

[26] Enrique Alba and Marco Tomassini. Parallelism and evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 6(5):443–462, October 2002.

[27] Günter Rudolph and Joachim Sprave. A cellular genetic algorithm with self-adjusting acceptance threshold. In *First International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications*, pages 365–372, Sheffield, UK, 1995.

[28] Lakhdar Loukil, Malika Mehdi, Nouredine Melab, El-Ghazali Talbi, and Pascal Bouvry. A parallel hybrid genetic algorithm-simulated annealing for solving q3ap on computational grid. In *International Symposium on Parallel and Distributed Processing*, pages 1–8, Chengdu, China, 2009. IEEE, IEEE.

[29] Craig Letavec and John Ruggiero. The n-queens problem. *INFORMS Transactions on Education*, 2(3):101–103, 2002.

[30] Konstantin Kogan. Unbounded knapsack problem with controllable rates: the case of a random demand for items. *Journal of the Operational Research Society*, 54(6):594–604, 2003.

[31] Jakob Puchinger, Güntehr R. Raidl, and Ulrich Pferschy. The multidimensional knapsack problem: Structure and algorithms. *INFORMS Journal on Computing*, 22(2):250–265, 2010.

[32] Umesh Chandra Jaiswal, Ankit Singh, Omkar Maurya, and Anil Kumar. Unified knapsack problem. *Communications in Computer and Information Science*, 142(2):325–330, 2011.

DOI:10.15774/PPKE.ITK.2012.002

[33] Sylvia Boyd, René Sitters, Suzanne van der Ster, and Leen Stougie. The traveling salesman problem on cubic and subcubic graphs. *Data Structures and Algorithms*, 2011.

[34] Akshay Gupta and Khushboo Aggarwal. Multiple neural network architecture for the travelling salesman problem. *International Transactions on Applied Sciences and Technology (ITAST)*, 1(1), 2011.

[35] Yll Haxhimusa, Edward Carpenter, Joseph Catrambone, David Foldes, Emil Stefanov, Laura Arns, and Zygmunt Pizlo. 2d and 3d traveling salesman problem. *The Journal of Problem Solving*, 3(2), 2011.

[36] Hans Kellerer, Ulrich Pferschy, and David Pisinger. *Knapsack Problems*. Springer Verlag, 2005.

[37] Donald Davendra, editor. *Traveling Salesman Problem, Theory and Applications*. InTech, 2010.

[38] Peter Földesy, Ákos Zarándy, Csaba Rekeczky, and Tamas Roska. Digital implementation of the cellular sensor-computers. *International Journal of Circuit Theory and Applications*, 34:409–428, 2006.

[39] Peter Földesy, Ákos Zarándy, and Csaba Rekeczky. Configurable 3d-integrated focal-plane sensor-processor array architecture. *International Journal of Circuit Theory and Applications*, 36(5-6):573–588, 2008.

[40] Maria Ercsey-Ravasz, Tamas Roska, and Zoltan Néda. Random number generator and monte carlo type simulations on the cnn-um. In *Proceedings of the 10th IEEE International workshop on Cellular Neural Networks and their applications*, pages 47–52, Istanbul, Turkey, 2006. IEEE, IEEE.

[41] Kenji Kise, Takahiro Katagiri, Hiroki Honda, and Toshitsugu Yuba. Solving the 24-queens problem using mpi on a pc cluster. *Technical Report UEC-IS-2004-6*, 2004.

[42] C. B. Posada, V. D. Ayala, S. J. Garcia, L. S. Silverio, and T. M. Vidal. A solution to multidimensional knapsack problem using a parallel genetic algorithm. *International Journal of Intelligent Information Processing*, 1(2):47–54, December 2010.

[43] Meijuan Gao, Jin Xu, Jingwen Tian, and Hao Wu. Path planning for mobile robot based on chaos genetic algorithm. In *Proceedings of the Fourth International Conference on Natural Computation*, pages 409 – 413, Jinan, China, 2008. IEEE, IEEE.

[44] Ismail AL-Taharwa, Alaa Sheta, and Mohammed Al-Weshah. A mobile robot path planning using genetic algorithm in static environment. *Journal of Computer Science*, 4(4):341–344, 2008.

DOI:10.15774/PPKE.ITK.2012.002

[45] Thé Van Luong, Nouredine Melab, and El-Ghazali Talbi. Gpu-based island model for evolutionary algorithms. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, Portland, USA, 2008. ACM, ACM.

[46] El-Ghazali Talbi. *Metaheuristics: From Design to Implementation*. Wiley, 2009.

[47] Ramon Van Handel. Hidden markov models (http://www.princeton.edu/ rvan/orf557/hmm080728.pdf). *Lecture notes of Hidden Markov Models at Princeton University*, 2008.

[48] P. Del Moral. *Feynman-Kac Formulae: Genealogical and Interacting Particle Systems with Applications*. Springer, 2004.

[49] Molnar-Saska Gabor. Rejtett Markov Modellek Statisztikai Vizsgálata. *Phd Dissertation*, 2005.

[50] Richard Ernest Bellman. *Dynamic programming*. Rand Corporation/Princeton University Press, 1957.

[51] P.M. Djuric and S. J. Godsill. Special issue on monte carlo methods for statistical signal processing. *IEEE Transactions on Signal Processing*, 50, 2002.

[52] A. Doucet and A. M. Johansen. A tutorial on particle filtering and smoothing: fifteen years later. *Oxford Handbook of Nonlinear Filtering*, pages 4–6, 2008.

[53] R. Carmona, J.-P. Fouque, and D. Vestal. Interacting particle systems for the computation of rare credit portfolio losses. *Finance and Stochastics*, 13:613–633, 2009.

[54] Salmond D. Gordon, N. and A. F. M Smith. Novel approach to non-linear and non-Gaussian Bayesian state estimation. *Radar and Signal Processing, IEEE Proceedings F*, 140:107–113, 1993.

[55] M.S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp. A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking. *Signal Processing, IEEE Transactions on*, 50(2):174–188, 2002.

[56] M. Bolic. Architectures for the implementatin of particle filters. *PhD thesis, Stony Brook University*, 2004.

[57] S. Maskell, B. Alun-Jones, and M. Macleod. A single instruction multiple data particle filter. *Proceedings of Nonlinear Statistical Signal Processing Workshop*, pages 51–54, 2006.

[58] M. Bolic, P.M. Djuric, and Sangjin Hong. Resampling algorithms and architectures for distributed particle filters. *Signal Processing, IEEE Transactions on*, 53(7):2442–2450, 2005.

DOI:10.15774/PPKE.ITK.2012.002

[59] P. Földesy, Á. Zarándy, Cs. Rekeczky, and T. Roska. Digital implementation of the cellular sensor-computers . *International Journal of Circuit Theory and Applications*, 34(4):409–428, 2006.

[60] P. Földesy, Á. Zarándy, Cs. Rekeczky, and T. Roska. High performance processor array for image processing. *IEEE International Symposium on Circuits and Systems*, pages 1177–1180, June 2007.

[61] Gustaffson F. Schön, T. and P.-J Nordlund. Marginalized particle filters for mixed linear/nonlinear state-space models. *IEEE Transactions on Signal Processing*, pages 2279–2289, 2005.

[62] N. Oudjanea and S. Rubenthaler. Stability and uniform particle approximation of nonlinear filters in case of non ergodic signals. *Stochastic Analysis and Applications*, 48(6):421–448, 2002.

[63] F. Le Gland and N. Oudjanea. Stability and uniform approximation of nonlinear filters using the Hilbert metric and application to particle filters. *Annals of Applied Probability*, 14(1):144–487, 2004.

[64] Oudjane N. Le Gland, F. A robustification approach to stability and to uniform particle approximation of nonlinear filters: the example of pseudo-mixing signals. *Stochastic Process. Appl.*, 106:279–316, 2003.

[65] D. Crisan and A. Doucet. A survey of convergence results on particle filtering methods for practitioners. *Signal Processing, IEEE Transactions on*, 50(3):736–746, 2002.

[66] T Roska and L.O Chua. The cnn universal machine: an analogic array computer. *IEEE Transactions on Circuits and Systems II-Analog and Digital Processing*, pages 163–173, 1993.

[67] L. O. Chua and T. Roska. The cnn paradigm. *IEEE Transactions on Circuits and Systems I - Fundamental Theory and Applications*, pages 147–156, 1993.

[68] Hoffmann G.M. and Tomlin C.J. Mobile sensor network control using mutual information methods and particle filters. *IEEE Transactions on Automatic Control*, 55(1):32–47, 2010.

[69] M. R. Pufall W. H. Rippard, editor. *Microwave Generation in Magnetic Multilayers and Nanostructures in Handbook of Magnetism and Advanced Magnetic Materials.* John Wiley (Sussex, U.K.), 2010.

[70] John C. Slonczewski. Current-driven excitation of magnetic multilayers. *Journal of Magnetism and Magnetic Materials*, vol 159:1–7, 1996.

DOI:10.15774/PPKE.ITK.2012.002

[71] L Berger. Emission of spin waves by a magnetic multilayer traversed by a current. *Physical Review B*, 1996.

[72] M. et al. Tsoi. Excitation of a magnetic multilayer by an elec tric current. *Phys. Rev. Letters*, 1998.

[73] M. et al. Tsoi. Generation and detection of phase-coherent current-driven magnons in magnetic multilayers. *Nature*, 2000.

[74] A. M. et al. Deac. Bias-driven high-power microwave emission from MgO-based tunnel magnetoresistance devices. *Nature Pgysics*, vol 4:803–809, 2008.

[75] Gyorgy Csaba, Wolfgang Porod, and Arpad I. Csurgay. A computing architecture composed of field-coupled single domain nanomagnets clocked by magnetic field. *International Journal of Circuit theory and applications*, vol 31:67—-82, 2003.

[76] M. Bonnin F. Corinto and M. Gilli. Weakly connected oscillatory networks models for associative and dynamic memories. *International Journal of Bifurcation and Chaos*, vol 17:4365–4379, 2007.

[77] Leon O. Chua and Tamas Roska. *Cellular Neural Networks and Visual Computing: Foundations and Applications*. Cambridge University Press, 2002.

[78] D. Bálya, B. Roska, T. Roska, and F. S. Werblin. A cnn framework for modeling parallel processing in a mammalian retina. *International Journal of Circuit Theory and Applications*, (30):363–393, 2002.

[79] Wolfgang Banzhaf, Peter Nordin, Robert Keller, and Frank Francone. *Genetic Programming – An Introduction*. Morgan Kaufmann, San Francisco, CA, 1998.

[80] A. E. et al Eiben. Genetic algorithms with multi-parent recombination. In *Proceedings of the International Conference on Evolutionary Computation The Third Conference on Parallel Problem Solving from Nature*, page 78–87, Jerusalem, Israel, 1994.

[81] Chuan-Kang Ting. *Advances in Artificial Life: On the Mean Convergence Time of Multi-parent Genetic Algorithms Without Selection*. Springer Verlag, 2005.

[82] Ziarati Akbari. A multilevel evolutionary algorithm for optimizing numerical functions. *International Journal of Industrial Engineering Computations*, pages 419–430, 2010.

[83] Y. Ephraim and N Merhav. Hidden markov processes. *IEEE Transactions on Information Theory*, 48:1508–1569, 2002.

[84] Leonar E. Baum and Ted Petrie. Statistical inference for probabilistic functions of finite state markov chains. *The Annals of Mathematical Statisticsb*, 37:1559–1563, 1966.

DOI:10.15774/PPKE.ITK.2012.002

[85] George Soules Leonar E. Baum, Ted Petrie and Norman Weiss. A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains. *The Annals of Mathematical Statisticsb*, 41(3):164–171, 1970.

[86] Andrew James Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13(2):260–269, 1967.

[87] Rudol E. Kálmán. A new approach to linear filtering and prediction problem. *Journal of Basic Engineering*, 82(1):35–45, 1960.

[88] Julier S.J. and Uhlmann J.K. Unscented filtering and nonlinear estimation. *Proceedings of the IEEE*, 92(3):401–422, 2004.

[89] Del Moral P. Lyons T Crisan, D. Discrete filtering using branching and interacting particle systems. *Markov Process. Related Fields*, 5:293–318, 1999.