Cellular Neural Networks and Least Squares for partial differential problems parallel solving

Nicolas Fressengeas, Hervé Frezza-Buet

Cellular Neural Networks and Least Squares for partial differential problems parallel solving

Nicolas Fressengeas

Laboratoire Matériaux Optiques, Photonique et Systèmes
University Paul Verlaine of Metz and Supélec
2, rue Edouard Belin, 57070 Metz Cedex, France
email: nicolas.fressengeas@univ-metz.fr

Hervé Frezza-Buet

Information, Multimodality and Signal, Supélec
2, rue Edouard Belin, 57070 Metz, France
email: Herve.Frezza-Buet@supelec.fr

Abstract

This paper shows how Cellular Neural Networks (CNN) can be harnessed into solving partial differential problems through an adaptation of the Least Squares Finite Elements Method. As CNNs can be implemented on distributed parallel architectures, this method allows the distribution of a resource demanding differential problem over a computer network.

Key words: Partial Differential Systems, Cellular Automata, Cellular Neural Networks, Formal computing, Least squares
PACS: 02.30.Jr, 02.60.Cb, 02.60.Jh, 02.60.Lj

1 Partial Differential Equations and Cellular Neural Networks

1.1 Automata and calculus

Ever since von Neumann (1), the question of modeling continuous physics with a discrete set of cellular automata has been raised, whether they handle discrete or continuous values. Many answers have been brought forth through, for instance, the work of Stephen Wolfram (2) summarized in a recent book (3).

This problem has been mostly tackled by rightfully considering that modeling physics through Newton and Leibniz calculus is fundamentally different from a discrete modelisation as implied by automata.

Indeed, the former implies that physics is considered continuous either because materials and fields are considered continuous in classical physics or because quantum physics wave functions are themselves continuous. On the contrary, modeling physics through automata implies modeling on a discrete basis, in which a unit element called *a cell*, interacts with its surroundings according to a given law derived from local physics considerations.

Such discrete automaton based models have been successfully applied to various applications ranging from reaction-diffusion systems (4) to forest fires (5), through probably one of the most impressive achievements: the Lattice Gas Automata (6), where atoms or molecules are considered individually. In this frame, simple point mechanics interaction rules lead to complex behaviors such as phase transition and turbulence. This peculiar feature of automata, making complex group behavior emerge from fairly simple individual rules aroused the interest around them for the past decades (7).

1.2 Cellular Neural Networks

Cellular automata-based modeling attempts have also concerned the theory of circuits for a few decades, because the Very-Large-Scale Integration (VLSI) components offer a large amount of configurable processors, spatially organized as a locally connected array of analogical and numerical processing units. In this field, the concept of Cellular Neural Network (CNN) has been proposed (8), which extends cellular automata, allowing local cells, that are dynamical systems, to deal with several continuous values and local connections.

CNNs are good candidates for the numerical resolution of partial differential equations (PDE), and a methodology for the design of a CNN from a given PDE has been proposed in (9). This approach consists in performing a spatial discretisation of the PDE through the finite difference scheme, yielding an Ordinary Differential Equation (ODE) on time that can be numerically solved by standard methods like Runge-Kutta.

This approach is widely used in this field, and drives the design of simulators like SCNN 2000 (10), as well as the design of actual VLSI components (11). The partial differential system is there implemented using analogical VLSI components, the circuit temporal evolution being then the temporal evolution of the initial PDE.

Two main difficulties arise in this framework. The first one concerns the stabil-

ity of the CNN dynamical system. Some stability studies of CNNs for classical PDEs can be found in (12) but stability has still to be analyzed when dealing with new specific problems, as it has been done, for example, for the dynamics of nuclear reactors (13). The second difficulty raised by transforming PDE to ODE for resolution by CNNs is the actual fitting of the CNN to the PDE, since the method is more a heuristic one than a formal derivation of the CNN from the PDE, as mentioned in (14). Furthermore, the features of the CNN cannot be easily associated to the physical parameters involved in the PDE.

To cope with the lack of methods to formally derive a CNN from a PDE, some parameter tunings can be performed once the CNN is designed. This tuning can be driven by a supervised learning process, as in (10; 14). Some other a posteriori checks can be achieved if some analytical solution of the PDE is known for particular cases, as in (12), or if some behavior of the CNN can be expected, as traveling waves or solitons (9; 15; 16). In the latter case, validation is based on a qualitative criterion.

Some other methods to derive automata from particular differential problems such as reaction-diffusion systems (4) or Maxwell's equations (17) have been presented. In the former, the automaton is constructed from a moving average paradigm, while the latter is a modified version of the Lattice Gas Automaton (6).

1.3 CNNs as quantitatively exact PDE solvers

In most cases, the predictions of calculus based, continuous models and those of discrete, automata based ones, are seldom quantitatively identical, though qualitative similarity is often obtained. This is mostly explained by the fact that the two drastically different approaches are applied to their own class of problems.

Some attempts have recently been made to set up the solution of a PDE by using a regression method (18). The idea there is to measure an error at each discrete point of the system, and to drive an optimization process in order to find the continuous function that minimizes this error, this function being taken in a parametrized set of continuous functions defined by a multilayer perceptron. This error is null if the function that is found meets the EDP requirements. Such regression processes, based on classical empirical risk minimization, are known to be sensitive to over fitting (19).

Other attempts at a quantitative link have however been made by showing connections between an automaton and a particular differential problem (20) or by designing methods for describing automata by differential equations (21; 22; 23) allowing in the way to *assess* the performance of two different

implementations of the same problem, which are in fact basically two different automata for the description of the same physics.

The interest of solving PDEs with cellular automata is of course not limited to physics, since PDEs are also intensively used in image processing (24). Some CNN-based solutions have also been proposed in that field (25). This stresses the need for generic tools for simulating PDEs in many areas. In (25), an attempt has been made to provide ready-to-use programming templates for the design of CNNs, and previously mentioned software (10) help to rationalize the design of CNNs for PDEs.

In this paper, the problem of designing a Cellar Neural Network from a given partial differential problem is addressed in an attempt to bridge the present gap (14) between continuous PDE and discrete CNN. To this aim, we have adapted the Least Squares Finite Elements Method (26) (LSFEM). In the following, section 2 describes our adaptation of the LSFE Method. Section 3 shows that the proposed algorithm can be made purely local and thus implemented thanks to CNNs. Section 4 provides implementation and application examples.

2 Adaptation of LSFEM to CNNs

In the following, we will reformulate the LSFE Method in a mathematical formalism which can then be used for the description of CNNs. Subsection 2.1 sets the necessary mathematical basis where one particular state of a CNN is viewed as a function from a discrete set (the cells) into a vector space (each cell hosts a vector of reals). Subsection 2.2 describes how the LSFEM functional is set and minimized. Finally, subsection 2.3 suggests that stochastic gradient descent (27) can help at making the computations local only. This will be proved in section 3.

Unfortunately, the necessary mathematical formalism used in this section can seem quite abstract. To overcome this difficulty, we will provide, at each step, a simple example: a normalized mono dimensional Poisson equation, $\triangle V(x) = \frac{\partial^2 V}{\partial x^2} = \rho(x)$, V being the unknown electrostatic potential and ρ a given repartition of charges. The example chosen has of course a straightforward solution but it is simple enough so that each step can be detailed in the paper.

The very characteristic of continuous physics is its intensive use of *fields*. If we note $(B)^A$ the set of functions from A to B, a field $\xi \in (\mathbb{R}^n)^{\mathbb{R}^m}$ is a mapping of a given vector physical quantity —belonging to $\mathbb{R}^n$— over a given physical space $\mathbb{R}^m$, for $(m, n) \in \mathbb{N}^2$. For instance, our example electrostatic potential field in a 1D space is a scalar mapping over $\mathbb{R}$, as an electric field over a 3D space would be a 3D vector mapping over $\mathbb{R}^3$. Furthermore, if time were present in this example, it would be treated equally as just an additional dimension. For instance, a time-resolved 3 dimensional problem is considered as having 4 dimensions.

Therefore, a particular local differential problem $\mathbb{P}$ stemming from local relationships, can be expressed in terms of a functional equation $\Phi^{(\xi)} = 0$, where the field ξ is the unknown, and where Φ represents the differential relationships derived from physical considerations, that a field ξ should satisfy to be the solution of $\mathbb{P}$. Let us note here that the functional equation $\Phi^{(\xi)} = 0$ merely represents any differential equation, or system of equations, over a field ξ of one or more dimensions. In our example, the field ξ is the association of an electrostatic potential $V(x)$ to each point x. The equation that has to be satisfied is $\forall x,\ \triangle V(x) - \rho(x) = 0$. This is better expressed by the corresponding functional equation, $\triangle V - \rho = 0$, where the whole mapping V is the unknown.

Φ can thus be defined as follows, where $p \in \mathbb{N}$ can be thought of as the number of independent real equations necessary to express the local relationships which are to be satisfied at any point of $\mathbb{R}^m$ ($p = 1$ in our example since only one scalar equation describes the problem):

$$\begin{aligned} \Phi : (\mathbb{R}^n)^{\mathbb{R}^m} &\mapsto (\mathbb{R}^p)^{\mathbb{R}^m} \\ \xi &\to \Phi^{(\xi)} \end{aligned} .$$

In other words, $\Phi^{(\xi)}$ is a mapping of a vector of p real values over the physical space $\mathbb{R}^m$. For any point $x \in \mathbb{R}^m$ in space, the i^{th} component of $\Phi^{(\xi)}(x) \in \mathbb{R}^p$, which would be zero if ξ was the solution of $\mathbb{P}$, actually corresponds to the local amount of violation of the i^{th} real local equation used to describe the problem at x: in our example, $\triangle V\,(x) - \rho\,(x)$, if not null, is the violation of Poisson equation at x. This is the heart of LSFEM: all violations, or errors, on all points can be summed up to a global error, which can itself be minimized. This is developed in the next paragraphs.

Using a functional equation instead of considering ξ as a given numerical instantiation as is usually done, allows to point out that the differential problem expressed by the mapping Φ depends intrinsically on the unknown field

ξ, whatever its actual instantiation, or value, is. The functional formalism allows to handle the dependency itself, *i.e.* the way all violations $\Phi^{(\xi)}$ over the physical space $\mathbb{R}^m$ depend on the whole field ξ.

With these notations, finding the solution to the differential problem Φ means finding $\xi^\star$ for which $\Phi^{(\xi^\star)} = 0$. This can be done by conventional approaches such as the multidimensional Newton minimization method or well known gradient descent such as conjugate gradient. All such methods could be used in the framework of our paper, each having its own advantages and disadvantages. For the sake of clarity and illustration purposes, we have chosen to develop our paper on the Newton Minimization Method but all concepts and demonstrations can be generalized to the other minimization methods.

We will express this method using functional derivatives of the mapping Φ with respect to ξ to set the basis for understanding how we can make it local only. Let us note here however that the functional derivatives are not as mathematically exotic as they may seem: they simply correspond to the derivative of one side of a differential equation with respect to the unknown field itself. In our example, this means deriving $\triangle V - \rho$ with respect to V.

To make this approach computationally tractable, we need to discretize the problem. This is performed by discretizing Φ on a finite mesh $\Omega \subset \mathbb{R}^m$, the discretized problem being then expressed as $\tilde{\Phi}^{(\tilde{\xi})} = 0$, where $\tilde{\xi}$ is the unknown and $\tilde{\Phi}$ is defined as follows:

$$\begin{array}{rcl} \tilde{\Phi} : (\mathbb{R}^n)^\Omega & \mapsto & (\mathbb{R}^p)^\Omega \\ \tilde{\xi} & \rightarrow & \tilde{\Phi}^{(\tilde{\xi})} \end{array} . \tag{1}$$

We will not address, in this paper, the difficult question of the optimum mesh Ω which allows the discrete solution to be the closest to the continuous one. We will thus assume that Ω is correctly chosen with respect to the differential problem itself so that conventional methods would give satisfactory results.

In contrast, the question of of the treatment of boundary conditions, whether they be of the Dirichlet or Neumann type is of primary importance. A Dirichlet condition expressed as some $\omega \in \Omega$ is to expressed as $\forall \tilde{\xi}$, $\tilde{\Phi}^{(\tilde{\xi})}(\omega) = 0$: in other words, the restriction of $\tilde{\Phi}$ to $\{\omega\}$ is null. Equally, specifying that Neumann conditions are to be satisfied on a subset of Ω is equivalent to giving a specific definition of $\tilde{\Phi}$ over this subset. This idea can be further generalized by considering several subsets of Ω over which the general expression of $\tilde{\Phi}$ changes. This would allow to take into account subdomains of Ω, each of which having its own differential problem. However and whatever the precise differential problem and thus the actual definition of $\tilde{\Phi}$, this shows that boundary conditions are not to be *added* to the discretised differential problem $\tilde{\Phi}$ but

are inherently *part of it*, as it should mathematically be.

To clarify this, let us go back to our example: the solving of the 1D Poisson equation on a mono dimensional mesh Ω of N regularly spaced points $x_1, \cdots, x_N$. In the following, the value $V(x_i)$ associated by the mapping V at the point x_i will be shortened to V^i. The same stands for the charge $\rho(x_i)$ at the point x_i, that will be written as ρ^i. The discretized problem can then be found by finite difference as the following, provided V^1 and V^N are defined as Dirichlet boundary conditions and d is the sampling step :

$$\forall i \in \mathbb{N}, 1 < i < N, \frac{1}{d^2}\left(V^{i-1} - 2V^i + V^{i+1}\right) - \rho^i = 0.$$

Once again, let us us stress here that the whole expression, including all space points is seen as depending on a *single* functional parameter V, which is a function over the discrete set $\{x_1, \cdots, x_N\}$. This function V is what is actually generally formalized above as $\tilde{\xi}$.

2.2 General method

Getting back to the general case and as was already discussed, solving the problem means finding $\xi^\star$ for which $\Phi^{(\xi^\star)} = 0$, which means finding a field $\tilde{\xi}$ for which $\tilde{\Phi}^{(\tilde{\xi})}$ is as close to the 0 mapping as possible given a distance on the functional space $(\mathbb{R}^p)^\Omega$. This, in turn, is equivalent to zeroing all p relations $\tilde{\Phi}^{(\tilde{\xi})}(\omega)$ for *all* $\omega \in \Omega$. Finally, this can be equivalently done by similarly minimizing a the functional expression $\mathcal{E}^{(\tilde{\xi})}$ as is done in the LSFE Method (26):

$$\mathcal{E}^{(\tilde{\xi})} = \sum_{\omega \in \Omega} \left| \tilde{\Phi}^{(\tilde{\xi})}(\omega) \right| \tag{2}$$

where $|\quad|$ is any given norm on $\mathbb{R}^p$. The usual LSFEM continuous integral is here replaced by a discrete sum because we have already discretized the differential problem so as to formalize the use of CNNs.

In our example, if the norm is chosen as the simple square, equation (2) translates to

$$\mathcal{E}^{(V)} = \sum_{i=2}^{N-1} \left(\frac{1}{d^2}\left(V^{i-1} - 2V^i + V^{i+1}\right) - \rho^i \right)^2.$$

As mentioned previously, we have to set $\tilde{\Phi}$ so that it *includes* the satisfaction of the differential equations at boundary conditions. This has been done here easily for the Dirichlet type by just a priori removing boundary terms 1 and

N from the sum, because their values are known from the Dirichlet conditions and thus no error can be committed on them.

Now that the error functional $\mathcal{E}^{(\tilde{\xi})}$ is defined, the LSFE Method prescribes that it be minimized so as to find the value $\tilde{\xi}^{\star}$ which produces the best solution to the initial problem. This can be done by numerous numerical methods such as the steepest or conjugate gradient (see for instance (28)). As we chose not to restrain our study to a specific differential problem, we have no particular reason to chose one particular minimization method. Thus, for illustration and demonstration purposes, we have chosen the standard Newton minimization method applied to multidimensional problems. The following considerations are valid whatever the method chosen. Let us note here however that this minimization process does not ensure the zeroing of $\mathcal{E}^{(\tilde{\xi})}$, which is to be verified *a posteriori* by evaluating $\mathcal{E}^{(\tilde{\xi}^{\star})}$.

The computation of $\mathcal{E}^{(\tilde{\xi})}$ produces a scalar from a given state $\tilde{\xi}$ of the discretized problem variables. This scalar can be viewed as an *evaluation* of this state. For further purpose, let us define more generally an evaluation as a function $\zeta \in (\mathbb{R})^{(\mathbb{R}^n)^{\Omega}}$. $\mathcal{E}$ is precisely an *evaluation* that is suited for quantifying the quality of a particular instantiation of $\tilde{\xi}$ as a solution to the discretized differential problem $\tilde{\Phi}$.

To undertake this optimization task, we previously need to define a canonical basis of the functional space $(\mathbb{R}^n)^{\Omega}$ with respect to which the *gradient* and *Hessian* will be taken. This basis is the set of the Cellular Network states in which each state is totally null except one given component of one given cell, which is set to 1. The number of basis elements is thus equal to the number of cells multiplied by the number of reals in each cell. This is mathematically defined as the following: if δ is the Kronecker symbol and $\{r\}_i$ is the canonical basis of $\mathbb{R}^n$, let us define $\{e\}_{(\omega,i)}$, the canonical basis of $(\mathbb{R}^n)^{\Omega}$ as the set of functions $e_{(\omega,i)}$, for all $\omega \in \Omega$ and all $1 \leq (i \in \mathbb{N}) \leq n$:

$$\begin{aligned} e_{(\omega,i)} \,:\, & \Omega \mapsto \mathbb{R}^n \\ & \omega' \rightarrow \delta_{\omega\omega'} r_i \end{aligned} \tag{3}$$

The partial derivative of an evaluation ζ at point $\tilde{\xi}$ according to basis vector $e_{(\omega,i)}$ is by definition $\lim_{h\in\mathbb{R}\to 0} \left(\zeta\left(\tilde{\xi} + he_{(\omega,i)}\right) - \zeta\left(\tilde{\xi}\right)\right)/h$. This value is, by definition of the gradient, the actual (ω, i) component of $\mathrm{grad}^{(\zeta)}_{|\{e\}_{(\omega,i)}}\left(\tilde{\xi}\right)$.

In our example, the basis vector $e_{(\omega,i)}$ is reduced to $e_{(\omega,1)}$ since $p = 1$. As ω is a given x_i, this basis vector is the mapping with 0 potential everywhere, except at x_i where the value V^i equals 1. Let us write this $e_{(x_i,1)}$ as v_i for our example.

Using these definitions of derivation and getting back to the general case, the Newton method consists in building a series $\tilde{\xi}_t$ defined as follows, the limit of which should be the sought solution $\tilde{\xi}^\star$ to $\tilde{\mathbb{P}}$, the field which is the solution of our initial differential problem:

$$\begin{aligned} \tilde{\xi}_{t+1} &= \tilde{\xi}_t - \mu^{(\mathcal{E})}_{\left|\{e\}_{(\omega,i)}\right.}\left(\tilde{\xi}_t\right) \\ \mu^{(\mathcal{E})}_{\left|\{e\}_{(\omega,i)}\right.}\left(\tilde{\xi}_t\right) &= \bar{\mathrm{H}}^{(\mathcal{E})}_{\left|\{e\}_{(\omega,i)}\right.}\left(\tilde{\xi}_t\right).\mathrm{grad}^{(\mathcal{E})}_{\left|\{e\}_{(\omega,i)}\right.}\left(\tilde{\xi}_t\right) \end{aligned} \tag{4}$$

where $\bar{\mathrm{H}}$ is the inverse of the Hessian matrix.

The above expression requires some derivability conditions on $\mathcal{E}$, and thus on both $\tilde{\Phi}$ and the chosen norm on $\mathbb{R}^n$. The former is assumed, since it stems from the problem $\mathbb{P}$ itself: the differential problem is here assumed to be derivable with respect to the unknown field. The latter is ensured by the appropriate choice of the used norm. As another precaution to be taken on that choice, the used norm must ensure that no component of the gradient – and thus of the Hessian inverse – neither supersedes the others nor is superseded by them, for this is known to create stability problems in the iteration defined by (4). The conventional $|\quad|_2$ norm, or its square, is for instance a good choice, provided $\mathbb{P}$ is conveniently normalized, *i.e.* that the unknown of the initial differential problem is a normalized quantity which has an order of magnitude around 1.

Equation (4) can be applied to our example by simply replacing $\tilde{\xi}_t$ by V_t and $\{e\}_{(\omega,i)}$ by the set of all $\{v\}_i$. This yields a complex expression for $\mu^{(\mathcal{E})}_{\left|\{v\}_i\right.}(V_t)$, too complicated too show here, that involves all $V^i, 1 \leq i \leq N$.

2.3 Local only computations

The effective computation of such a series as defined by (4) implies to compute, for each step t, the gradient and inverse Hessian with respect to $\{e\}_{(\omega,i)}$, which implies getting access to the whole Ω. This is in contradiction with our initial goal which was to design a computational method which can be implemented thanks to CNNs. Indeed, this requires that the method be *local-only*. This means that the evaluation of a particular cell of the mesh requires only the knowledge of the values in a few neighboring cells instead of the whole Ω.

To overcome this limitation, we present in the following a method inspired from the stochastic gradient descent method (27), the locality of which will be established in the next section.

$i = 1 \quad \text{or} \quad i = N$	$V_{t+1}^i = V_t^i$
$i = 2$	$V_{t+1}^i = \frac{1}{5}\left(2V_t^{i-1} + 4V_t^{i+1} - V_t^{i+2} + d^2 \times \left(\rho^{i+1} - 2\rho^i\right)\right)$
$i = N - 1$	$V_{t+1}^i = \frac{1}{5}\left(2V_t^{i+1} + 4V_t^{i-1} - V_t^{i-2} + d^2 \times \left(\rho^{i-1} - 2\rho^i\right)\right)$
$3 \leq i \leq N - 2$	$V_{t+1}^i = \frac{1}{6}\left(-V_t^{i-2} + 4V_t^{i-1} + 4V_t^{i+1} - V_t^{i+2} + d^2 \times \left(\rho^{i-1} - 2\rho^i + \rho^{i+1}\right)\right)$

Table 1
Update rules for the mono dimensional automaton which solves the mono dimensional Poisson equation, as computed from (5).

The stochastic gradient method consists in updating $\tilde{\xi}$ by considering only a few of its components at a time. We choose to consider a single ω in Ω at each step, thus modifying only $\tilde{\xi}^\omega$, the ω-related components of $\tilde{\xi}$, *i.e.* the values of the field at a given point in the mesh. Therefore, the gradient and Hessian appearing in (4) are taken not with respect to the whole $\{e\}_{(\omega,i)}$ but rather with a subset $\{e\}_\omega$ of it, restricted to ω, defined as the set $\left\{e_{(\omega',i)} : \omega' = \omega \text{ and } 1 \leq i \leq n\right\}$.

The system of interdependent equations resulting from the problem discretization is thus derived with respect to the field values at a given point at a time only. One such step is therefore defined as follows:

$$\tilde{\xi}_{t+1}^\omega = \tilde{\xi}_t^\omega - \mu^{(\mathcal{E})}_{|\{e\}_\omega}\left(\tilde{\xi}_t\right), \tag{5}$$

which, in the frame of our example, translates to:

$$V_{t+1}^i = V_t^i - \mu^{(\mathcal{E})}_{|v_i}(V_t).$$

The above relationship describes a series for a given point ω of Ω. For the series (4) to be completely approximated by the stochastic method, the relationship (5) is to be iterated over Ω with a random choice of $\omega \in \Omega$ at each step: for the derivative to be complete, it is here taken successively with respect to the field values at each point in the mesh.

Thus, provided $\mu^{(\mathcal{E})}_{|\{e\}_\omega}$ is somehow local, an issue that will be addressed in the next section, the above considerations allow to consider (5), at ω, as the definition of a *continuous* automaton, or CNN, which is an extension of classical cellular automata for which the cell states are allowed to take their values in $\mathbb{R}^n$. This automaton can be implemented for any given differential problem $\mathbb{P}$ by evaluating $\mu^{(\mathcal{E})}_{|\{e\}_\omega}$ for this particular problem.

Evaluating $\mu^{(\mathcal{E})}_{|\{e\}_\omega}$ can be done, as equation (4) suggests, by taking the proper

gradient and Hessian of the discretized problem at each point in the mesh. Applied to our example, this method allows to calculate, for each point in the mono dimensional mesh, the update rule to be applied to that point. 4 different rules are found, which are given in table 1.

As can also be inferred from table 1, the automaton described by (5) for each ω departs from the strict definition of a cellular automaton by the fact that the update rule for all cells ω are only the same for a vast majority of them, but not strictly all. Indeed, because of the existing boundary conditions, the H and grad operators will not give the same result for all points, since the boundary conditions are considered as constants. Hence, a Dirichlet boundary is described in the automaton by a constant cell, the value of which is given by the automaton initial state.

At this point of the paper, we have defined a CNN that can be automatically generated from any given differential problem, thanks to automated formal derivative computing, by evaluating the stochastic gradient descent (5) at each point of the mesh. We have done so by assuming that the update rules thus computed is local. To formally establish that a CNN is indeed generated, we now need a proof of this assumption. It is the subject of the next section.

3 Localization of each cell neighborhood

The demonstration of the locality of the variant of the LSFE Method we have presented is a key point in this paper, as the computation thanks to CNNs that can be implemented on parallel architecture is our essential goal. It is done in a two step procedure detailed in the next two subsections. The first step is the definition of the neighborhood of a given cell ω in the mesh: it is the set of the cells whose values are needed in the computation of the update of ω.

The second step consists in evaluating the size of this neighborhood by determining which cells are elements of it. This demonstration is done by considering the particular Newton method for minimization but it would be valid for any other method as only the properties of the derivatives are used.

3.1 Neighborhood definition

The definition of the *neighborhood* $\mathcal{V}(\zeta)$ of a given *evaluation* ζ (see section 2.2 for a definition) is thus to be understood as being the set of all the points ω

$i = 1 \quad \text{or} \quad i = N$	$\{x_i\}$
$i = 2$	$\{x_{i-1}, x_{i+1}, x_{i+2}\}$
$i = N - 1$	$\{x_{i-2}, x_{i-1}, x_{i+1}\}$
$3 \leq i \leq N - 2$	$\{x_{i-2}, x_{i-1}, x_{i+1}, x_{i+2}\}$

Table 2
Neighborhoods $\mathcal{V}\left(\left|\mu^{(\mathcal{E})}_{|\{v\}_i}\right|\right)$ for all points of a CNN which solves the mono dimensional Poisson Equation

needed in the computation of ζ.

$$\begin{aligned} \mathcal{V} : (\mathbb{R})^{(\mathbb{R}^n)^\Omega} &\mapsto \mathcal{P}(\Omega) \\ \zeta &\to \left\{\omega \in \Omega \ : \ \exists \tilde{\xi} \ : \ \text{grad}^{(\zeta)}_{|\{e\}_\omega}\left(\tilde{\xi}\right) \neq 0\right\} \end{aligned} \tag{6}$$

The CNN described in the previous section is thus practically usable if the calculations needed to evaluate it are *local*, *i.e.* expression (5) can be evaluated without requiring access to Ω as a whole. This can be formally stated as $\mathcal{V}\left(\left|\mu^{(\mathcal{E})}_{|\{e\}_\omega}\right|\right) \neq \Omega$. This can happen only if some kind of locality condition on $\mathbb{P}$ is assumed, *i.e.* if the initial differential problem is expressed in a local manner, as it is usually the case.

In the frame of our example, the values of $\mathcal{V}\left(\left|\mu^{(\mathcal{E})}_{|v_i}\right|\right)$ for all points in the mesh are given in table 2. For instance, the last row of table table 1 show that the value of V at points x_{i-2}, x_{i-1}, x_{i+1} and x_{i+2} are required to evaluate x_i, thus leading to the neighborhood $\{x_{i-2}, x_{i-1}, x_{i+1}, x_{i+2}\}$ shown on the last row of table 2.

3.2 Neighborhood size

To show that the automaton is indeed local in the general case, let us first consider the specific case of the evaluation $\left|\tilde{\Phi}(\omega)\right|$, that is the error measurement at point ω. The global error evaluation $\mathcal{E}$ is a summation of such terms (see (2)).

For further use, we now need to define an enhancement of the neighborhood concept we have called the *dependency* $\mathcal{D}\left(\omega, \tilde{\Phi}\right)$ of a given $\omega \in \Omega$ involved in a problem $\tilde{\Phi}$. It is the set of point ω' for which ω belongs to the neighborhood of ω':

$$\mathcal{D}\left(\omega, \tilde{\Phi}\right) = \left\{\omega' \ : \ \omega \in \mathcal{V}\left(\left|\tilde{\Phi}(\omega')\right|\right)\right\}.$$

Given the definition (4) of $\mu^{(\mathcal{E})}_{\left|\{e\}_\omega\right.}$, the gradient can be linearly distributed over the additive components of $\mathcal{E}$ as in (8). The summation term appearing in (7) has been restricted to those ω' in Ω for which the gradient does not vanish, *i.e.* those $\omega' \in \mathcal{D}\left(\omega, \tilde{\Phi}\right)$. The summations product in (8) is obtained by similarly distributing the Hessian.

$$\begin{aligned} \mu^{(\mathcal{E})}_{\left|\{e\}_\omega\right.} &= \sum_{\omega' \in \mathcal{D}\left(\omega,\tilde{\Phi}\right)} \bar{\mathrm{H}}^{(|\mathcal{E}|)}_{\left|\{e\}_\omega\right.} \mathrm{grad}^{\left(\left|\tilde{\Phi}(\omega')\right|\right)}_{\left|\{e\}_\omega\right.} \qquad (7) \\ &= \bar{\mathrm{H}}^{(|\mathcal{E}|)}_{\left|\{e\}_\omega\right.} \sum_{\omega' \in \mathcal{D}\left(\omega,\tilde{\Phi}\right)} \mathrm{grad}^{\left(\left|\tilde{\Phi}(\omega')\right|\right)}_{\left|\{e\}_\omega\right.} \\ = \left(\sum_{\omega' \in \mathcal{D}\left(\omega,\tilde{\Phi}\right)} \mathrm{H}^{\left(\left|\tilde{\Phi}(\omega')\right|\right)}_{\left|\{e\}_\omega\right.} \right)^{-1} & \sum_{\omega' \in \mathcal{D}\left(\omega,\tilde{\Phi}\right)} \mathrm{grad}^{\left(\left|\tilde{\Phi}(\omega')\right|\right)}_{\left|\{e\}_\omega\right.} \qquad (8) \end{aligned}$$

The neighborhood of a product being included in the union of its operands neighborhoods, from (6), the neighborhood of $\left|\mu^{(\mathcal{E})}_{\left|\{e\}_\omega\right.}\right|$, according to (8), can be limited to

$$\begin{aligned} \mathcal{V}\left(\left|\mu^{(\mathcal{E})}_{\left|\{e\}_\omega\right.}\right|\right) \subset \mathcal{V}\left| \left(\sum_{\omega' \in \mathcal{D}\left(\omega,\tilde{\Phi}\right)} \mathrm{H}^{\left(\left|\tilde{\Phi}(\omega')\right|\right)}_{\left|\{e\}_\omega\right.} \right)^{-1} \right| \\ \bigcup \mathcal{V}\left| \sum_{\omega' \in \mathcal{D}\left(\omega,\tilde{\Phi}\right)} \mathrm{grad}^{\left(\left|\tilde{\Phi}(\omega')\right|\right)}_{\left|\{e\}_\omega\right.} \right| \qquad (9) \end{aligned}$$

From definition (6), it can be shown that the neighborhood of a derivative, or a gradient, is included in the neighborhood of its operand. The same holds for the neighborhood of a Hessian since any line or column of the Hessian is a derivative of the gradient. Therefore, the right-hand term of the above union is included in $\bigcup_{\omega' \in \mathcal{D}\left(\omega,\tilde{\Phi}\right)} \mathcal{V}\left(\left|\tilde{\Phi}(\omega')\right|\right)$.

Furthermore, the neighborhood of a matrix norm $|M|$ is obviously included in the union of the neighborhoods of all its components. The same holds for the inverse matrix $\left|(M)^{-1}\right|$ since each of its components can be obtained by a combination of the components of M. We can therefore conclude that the left-hand term of the union in (9) is also a subset of $\bigcup_{\omega' \in \mathcal{D}\left(\omega,\tilde{\Phi}\right)} \mathcal{V}\left(\left|\tilde{\Phi}(\omega')\right|\right)$.

Therefore, provided we can assume that $\mathcal{V}\left(\left|\tilde{\Phi}(\omega)\right|\right)$ is small enough for all $\omega \in \Omega$ —which is ensured if the differential problem $\mathbb{P}$ is defined locally—,

the calculations to be undertaken to evaluate $\mu^{(\mathcal{E})}_{\big|_{\{e\}_\omega}}$ for each cell ω are local to some extended neighborhood of that cell:

$$\mathcal{V}\left(\left|\mu^{(\mathcal{E})}_{\big|_{\{e\}_\omega}}\right|\right) \subset \bigcup_{\omega' \in \mathcal{D}\left(\omega, \tilde{\Phi}\right)} \mathcal{V}\left(\left|\tilde{\Phi}(\omega')\right|\right)$$

From the definition of the neighborhood, the above inclusion means that the calculations involved in computing the update rule $\mu^{(\mathcal{E})}_{\big|_{\{e\}_\omega}}$ at a given point in the mesh only involve the field values of the dependent points ω' in the sense of $\mathcal{D}$, the actual number and repartition of those points being dependent on the differential problem itself. (In the frame of our example, the automaton obtained by our formal resolution process is given in table 1.)

We have now proven that the stochastic gradient descent applied to the Newton minimization in LSFEM leads to a local CNN, provided that the initial differential problem is itself local as it is generally the case. We have therefore proven that LSFEM can be efficiently implemented using CNNs. This is particularly interesting if a programming environment using CNNs, analogous to those described in (29; 30), is available not only on shared memory multi-processor computers but also on distributed memory architectures such as clusters (31; 32; 33).

4 Application examples

This next section is thus devoted to the presentation of application examples on two classical Dirichlet boundary value problems. We will however not present any performance analysis in terms of computing time until convergence since this highly depends on the actual parallel implementation of the CNNs (31; 32). This work is currently in progress and an analysis of the obtained computing time has already been done (33).

As hinted before, we have implemented the continuous automaton described in the previous sections with the help of an off-the-shelf formal computing software [1], essentially used to formally evaluate the update rule from the differential problem through equation (5), and a cellular automata environment analogous to those reported in (29; 30). We have thus automated the computation from the specification of the discrete differential problem $\tilde{\mathbb{P}}$ to the design of the adequate continuous automaton solving the differential problem

[1] We have used *Mathematica* from Wolfram Research

through LSFEM [2] .

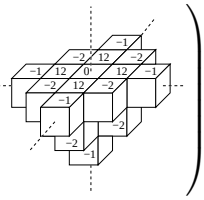
Let us now illustrate this process with two examples. The first one is the generalization to 3 dimensions of the example used in the previous sections. The mono dimensional example was trivial, as it possessed a straightforward solution. The 3D one is a little trickier as it is a 3D boundary value problem. The second example is the application of strictly the same piece of software to the non-paraxial beam propagation equation, which is not so easy to solve numerically.

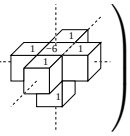
4.1 Poisson equation

The first example is thus the solving of a normalized Poisson Equation for V in the three dimensions of space: $\triangle V(x,y,z) = \rho(x,y,z)$ for any given ρ, the Dirichlet boundary conditions being set on the sides of the computing cube window. The corresponding discrete problem is straightforward and is obtained through finite difference centered second derivatives on each dimensions of space, for the same space step d.

The automaton obtained through the evaluation of (5) on each point of the mesh has 28 different update rules. The update rule obtained for ω such as the boundaries conditions are not in $\mathcal{V}\left(\left|\mu^{(\mathcal{E})}_{|\{e\}_\omega}\right|\right)$ concerns the vast majority of the mesh nodes ω and is shown below. It is a centro-symmetric three dimensional convolution kernel involving V and ρ. Only middle and lower parts of these kernels are shown, the upper part being obtained by symmetry.

$$V \leftarrow \frac{1}{42}\left[V\left(\right. \right. \left. \left. \right) + d^2.\rho\left(\right. \right. \left. \left. \right)\right].$$





The 27 other update rules account for the boundary conditions. When launched, the system converges to a fixed point, corresponding to the result that, in that case, can also be obtained with other methods.

As stated above, the result has to be checked valid *a posteriori* by evaluating the remaining error as defined by (2). For a better assessment of the performance, we will provide two values of the remaining error. the first one is the *mean error*, which is simply $\mathcal{E}^{(\tilde{\xi})}$ when $\tilde{\xi}$ takes the value of the solution found,

[2] The corresponding piece of software is available under GPL license on the authors web sites.

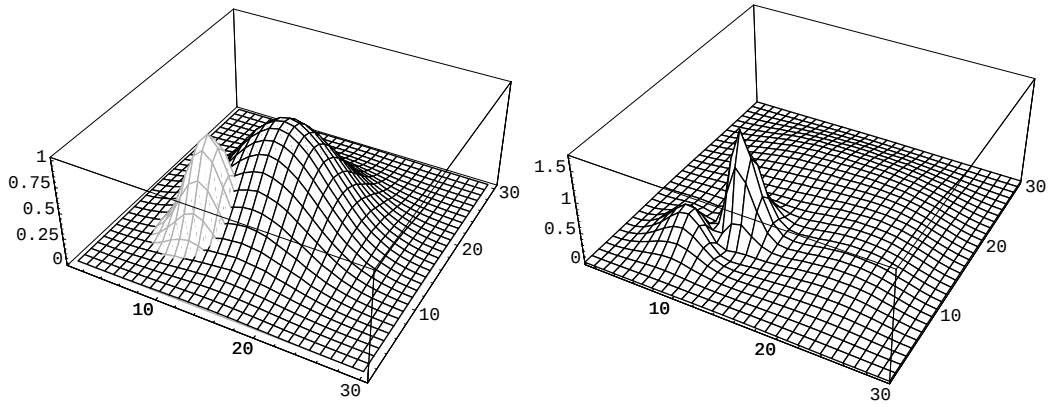


Fig. 1. Left : laser beam Gaussian profile to be coupled in the waveguide (black) and waveguide (grey) (A.U.). Right : beam profile after a 3mm propagation. The window size is 30μm and the beam wavelength is 250μm. The highest peak on the right evidences the light which is coupled into the waveguide.

all divided by the number of points, to get the mean error per mesh point. The other one, the *maximum error*, is defined as $\mathcal{E}_M{}^{(\tilde{\xi})} = \max_{\omega\in\Omega} \left|\tilde{\Phi}^{(\tilde{\xi})}(\omega)\right|$ and yields the maximum error per point. Both will be normalized to the maximum component of $\tilde{\xi}$.

For a $20 \times 20 \times 20$ mesh and for a value of ρ varying from 1 to 0 from one side of the cube to the other, the *a posteriori* computed mean and maximum errors are 7×10^{-3} and 4×10^{-1} respectively for $d = 1$ and decrease with it. The maximum error, that can seem large, is due to strong gradients in the solution close to the boundary conditions and the very crude mesh used. The strong gradients are caused by the non realistic values taken for ρ. However, the mean error shows that the solution found, aside from a few points, is still acceptable, despite the sparse mesh.

4.2 Non paraxial laser beam propagation

The second example is a well know difficulty in the domain of electromagnetic propagation: the removal of the paraxial approximation. Indeed, we now aim to compute the coupling of a Gaussian laser beam of width W into a Gaussian shaped waveguide of width $\frac{W}{2}$ and modulation depth 10^{-4}. The centers of both beam and waveguide are set to a distance of $\frac{W}{2}$, both being aligned in the same direction. In the computation process, we will *not* make the standard simplifying paraxial approximation, which makes our problem difficult to solve by conventional methods.

The non-paraxial propagation equation to be solved is thus the following, where A is the wave electric field to be found, z is the propagation direction,

k is the wave vector, n and δn are the given refraction index and a small variation of it:

$$\frac{\partial A}{\partial z} - \frac{i}{2k}\triangle A = \frac{ik}{n}\delta n A. \tag{10}$$

The problem is solved by deriving two real equations from (10), discretizing them with finite difference centered derivatives except along z where left-handed derivatives are needed because of the impossibility to give a boundary condition on one side of the propagation axis.

The adequate continuous automaton (it also has 28 update rules but is too complicated to show here) is then computed from the discretized problem. When launched on a $30 \times 30 \times 30$ network, it stabilizes to a fixed point shown on figure 1, where the light is found to be coupled into the waveguide. The *a posteriori* remaining mean and maximum errors are now computed to be 2×10^{-12} and 9×10^{-12} respectively, proving that the obtained solution does indeed meet the differential problem requirements.

5 Conclusion

We have shown how the Least Squares Finite Elements Method can be adapted for its implementation using CNNs. This is of particular interest as CNNs can be efficiently implemented on parallel hardware (31; 32; 33), paving the way for the distribution of large scale differential problems on computer networks.

A side effect of the method is the possibility to automate the design of the CNN, thanks to formal computing, from the differential problem specification down to its solution, sparing the user the need to get involved in actual numerical mathematics and computer programming, thus sparing code development time.

6 Acknowledgments

This work was supported by the InterCell MISN program of the French State to Lorraine region 2007-2013 plan. CNN implementation was done and run on hardware funded from this grant.

References

[1] A. W. Burks, Von Neumann's Self-reproducing Automata, University of Michigan, 1969.

[2] S. Wolfram, Statistical mechanics of cellular automata, Rev. Mod. Phys. 55 (1983) 601–444.

[3] S. Wolfram, A new kind of science, Wolfram Media, Champaign, 2002.

[4] J. R. Weimar, J. P. Boon, Class of cellular automata for reaction-diffusion systems, Phys.Rev.E 49 (2) (1994) 1749–1752.

[5] B. Drossel, F. Schwabl, Self-organized critical forest-fire model, Phys. Rev. Lett. 69 (11) (1992) 1629–1632.

[6] D. H. Rothman, S. Zaleski, Lattice-gas models of phase separation: interfaces, phase transitions, and multiphase flow, Rev. Mod. Phys. 66 (4) (1994) 1417–1479.

[7] B. Chopard, M. Droz, Cellular Automata and Modeling of Physical Systems, Cambridge University Press, Cambridge, 1998.

[8] L. O. Chua, L. Yang, Cellular neural networks: Theory, IEEE Transactions on Circuits and Systems 35 (1988) 1257–1272.

[9] T. Roska, L. O. Chua, D. Wolf, T. Kozek, R. Tetzlaff, F. Puffer, Simulating nonlinear waves and partial differential equations via cnn–part i: Basic techniques, IEEE Transactions on Circuits and Systems–I: Fundamental theory and applications 42 (10) (1995) 807–815.

[10] A. Lonkar, R. Kuntz, R. Tetzlaff, Scnn 2000 - part i: Basic structure and features of the simulation system for cellular neural networks, in: 6th EEE International Workshop on Cellular Neural Networks and Their Applications, 2000, pp. 123–128.

[11] F. Sargeni, V. Bonaiuto, Programmable cnn analogue chip for rd-pde multi-method simulation, Analog Integrated Circuits and Signal Processing 44 (2005) 283–292.

[12] A. Slavova, Application of some mathematical methods in the analysis of cellular neural networks, Journal of Computational and Applied Mathematics 114 (2000) 387–404.

[13] K. Hadad, A. Piroozmand, Application of cellular neural networks (cnn) method to the nuclear reactor dynamics equation, Annals of Nuclear Energy.

[14] O. Bandmann, Cellular-neural automaton: an hybrid model for reaction-diffusion simulation, Future Generation Computer Systems 18 (2002) 737–745.

[15] T. Kozek, L. O. Chua, T. Roska, D. Wolf, R. Tetzlaff, F. Puffer, K. Lotz, Simulating nonlinear waves and partial differential equations via cnn–part ii: Typical examples, IEEE Transactions on Circuits and Systems–I: Fundamental theory and applications 42 (10) (1995) 807–815.

[16] A. Slavova, P. Zecca, Cnn model for studying dynamics and travelling wave solutions of the fitzhugh-nagumo equation, Journal of Computational and Applied Mathematics 151 (2003) 13–24.

[17] N. R. S. Simons, G. E. Bridges, M. Cuhaci, A lattice gas automation capable of modeling three-dimensional electromagnetic fields, J. Comput. Phys. 151 (2) (1999) 816–835.
[18] X. Zhou, B. Liu, B. Shi, Neural networks for solving partial differential equations, in: Proceedings of the 7th World Multiconference on Systemics, Cybernetics and Informatics, Vol. V of Computer Science and Engineering: I, 2003, pp. 240–244.
[19] V. N. Vapnik, The Nature of Statistical Learning Theory, Statistics for Engineering and Information Science, Springer, 2000.
[20] T. Tokihiro, D. Takahashi, J. Matsukidaira, J. Satsuma, From soliton equations to integrable cellular automata through a limiting procedure, Phys Rev. Lett. 76 (18) (1996) 3247–3250.
[21] A. Doeschl, M. Davison, H. Rasmussen, G. Reid, Assessing cellular automata based models using partial differential equations, Math. Comp. Mod. 40 (2004) 977–944.
[22] W. Kunishima, A. Nishiyama, H. Tanaka, T. Tokihiro, Differential equations for creating complex cellular automaton patterns, Journ. Phys Soc. Japan 73 (8) (2004) 2033–2036.
[23] S. Omohundro, Modelling cellular automata with partial differential equations, Physica D 10D (1984) 128–134.
[24] G. Aubert, P. Kornprobst, Mathematical Problems in Image Processing Partial Differential Equations and the Calculus of Variations, 2nd Edition, Vol. 147 of Applied Mathematical Sciences, Springer, 2006.
[25] C. Rekeczky, Cnn architecture for constrained diffusion based locally adaptive image processing, International Journal of CVircuit Theory and Applications 30 (2002) 313–348.
[26] B.-N. Jiang, The Least-squares Finite Element Method: Theory and Applications in Computational Fluid Dynamics and Electromagnetics, Springer, 1998.
[27] J. C. Spall, Introduction to Stochastic Search and Optimization: Estimation, Simulation, and Control, Wiley-Interscience, 2003.
[28] C. M. Bishop, Neural Networks for Pattern Recognition, Oxford University Press, 2004.
[29] M. Cannataro, S. D. Gregorio, R. Rongo, W. Spataro, G. Spezzano, D. Talia, A parallel cellular automata environment on multicomputers for computational science, Parallel Computing 21 (1995) 803–823.
[30] G. Spezzano, D. Talia, The CARPET programming environment for solving scientific problems on parallel computers, in: Virtual shared memory for distributed architectures, Nova Science Publishers, Inc., Commack, NY, USA, 2001, pp. 51–68.
[31] J. Gustedt, S. Vialle, A. De Vivo, The parXXL Environment: Scalable Fine Grained Development for Large Coarse Grained Platforms, in: Applied Parallel Computing. State of the Art in Scientific Computing PARA-06: Worshop on state-of-the-art in scientific and parallel computing, Vol. 4699 of Lecture Notes in Computer Science, Umea Suède, 2007, pp. 1094–

1104.

[32] J. Gustedt, S. Vialle, A. De Vivo, parXXL: A Fine Grained Development Environment on Coarse Grained Architectures, in: Workshop on State-of-the-Art in Scientific and Parallel Computing - PARA'06, Umeå/Sweden Suède, 2006.
URL http://hal.inria.fr/inria-00103772/en/

[33] N. Fressengeas, H. Frezza Buet, J. Gustedt, S. Vialle, An Interactive Problem Modeller and PDE Solver, Distributed on Large Scale Architectures, in: Third International Workshop on Distributed Frameworks for Multimedia Applications - DFMA '07, IEEE, Paris France, 2007, http://lifc.univ-fcomte.fr/dfma07/ CPER Région Lorrain MIS - InterCell.
URL http://hal.inria.fr/inria-00139660/en/