Mass-parallel Sleptsov Net-Based Solving PDEs on FPGA for Embedded Control

Dmitry A. Zaitsev[a*], Alexander A. Kostikov[b]

[a] *The University of Derby, Derby, UK, ORCID: 0000-0001-5698-7324*
[b] *Technical University "Metinvest Polytechnic" LLC, Zaporizhzhe, Ukraine, ORCID: 0000-0002-1007-7912*

ABSTRACT

Continuous processes, significant for real-time control of embedded applications, are specified by partial differential equations (PDEs) and systems with certain boundary conditions. The paper develops techniques of computing structures design for fast mass-parallel numerical solving of PDEs. We compose specialized computing lattices based on the integer number approximation specified with Sleptsov nets to be implemented as dedicated hardware, which we prototype on FPGA. For mass-parallel solving of PDEs, we employ ad-hoc finite-difference schemes and iteration methods that allow us to recalculate the lattice values in a single time cycle with appropriate accuracy suitable for control of hypersonic objects and thermonuclear reactions.

CCS Concepts

• Computing methodologies → Concurrent computing methodologies • Computer systems organization → Embedded and cyber-physical systems • Mathematics of computing → Continuous mathematics.

Keywords

Partial-differential equation; mass-parallel solution; Sleptsov net; FPGA; embedded control

1. INTRODUCTION

Solving a Partial Differential Equation (PDE) [1-3] represents the basic technique of continuous system modeling in a wide range of practical applications. Some specific forms of PDE allow us to obtain analytical solutions, which simplify the process of their application. In the general case, a given equation is resolved numerically; either finite difference or finite element techniques are applied. We focus on specific requirements and techniques for fast solving of PDEs for control purposes in embedded applications [4]. For example, control of airplane engine turbines based on resolving gas dynamics and thermodynamics problems, control of hypersonic vehicle trajectories, etc. For these applications, iteration methods [2] yield utmost performance, allowing us to apply directly a mass-parallel computing approach.

For embedded applications, we offer an approach that implements a specialized hardware lattice on FPGA [5] via generating the corresponding Verilog [6] code. For the lattice specification, we apply a graphical language of Sleptsov nets [7,8]. We considerably simplify the computing structure based on integer number approximation of a computing lattice, converging with the cellular automata techniques [9] for dedicated hardware implementation, which we prototype on FPGA [5]. Obtained for FPGA benchmarks, acknowledge the applicability of the approach for fast control based on the ongoing process of numeral solving boundary problems for PDE, where the boundary is actually mapped into sensors and actuators.

2. NUMERICAL SOLVING PDE

Partial differential equations describe physical processes in mechanics, hydrodynamics, acoustics, heat transfer, electricity, magnetism, and other domains. When it is impossible to find analytically the solution of these equations, numerical methods, in particular, the finite difference method, are applied for their solution [1-3]. The original partial derivative equation is reduced to a system of difference equations, for which solution direct and iterative methods are applied. In many cases, this is the only way to find the solution of PDEs.

As a case study, we apply the finite difference method for solving the Laplace equation based on the technique described in [2]

$$\frac{\partial^2 u(x,y)}{\partial x^2} + \frac{\partial^2 u(x,y)}{\partial y^2} = 0, \qquad (1)$$

which is replaced by the following difference relation

$$\frac{u(x-h,y) - 2u(x,y) + u(x+h,y)}{h^2} + \frac{u(x,y-h) - 2u(x,y) + u(x,y+h)}{h^2},$$

where $h > 0$ is the discretization step.

With this substitution, we arrive at an equation that relates the values of the desired function at separately taken points, which are usually chosen so that they form a square mesh. We construct the difference approximations for the case of a function of two independent variables, $u(x,y)$ on a rectangular mesh. Without restricting the generality, we will assume that the change area of the argument x is the segment $0 \le x \le 1$, and the change area of the argument y is the segment $0 \le y \le 1$. Let us divide the segments $0 \le x \le 1$, $0 \le y \le 1$ by points $x_i = ih$ $(i = 0,1,2 \dots, N)$, $y_j = jh$ $(j = 0,1,2, \dots, N)$ into N equal parts of length $h = \frac{1}{N}$ each. The set of points (x_i, y_j) with coordinates $x_i = ih$ $(i = 0,1,2 \dots, N)$ and $y_j = jh$ $(j = 0,1,2, \dots, N)$ is called a mesh in the square $0 \le x \le 1$, $0 \le y \le 1$. By $u_{i,j}$ we denote the value of function u at the points with coordinates (x_i, y_j): $u_{i,j} = u(x_i, y_j)$. The value of the sought function at the boundary nodes is determined for $i = 0,1,2, \dots, N, j = 1,2, \dots N-1$ as

$$u_{i,0} = \varphi(x_i, 0)$$
$$u_{i,N} = \varphi(x_i, 1)$$
$$u_{0,j} = \varphi(0, y_j)$$
$$u_{N,j} = \varphi(1, y_j) \quad (2)$$

Let us set the initial approximation $u_{i,j}^{(0)}$. For internal points of the domain, we choose the initial approximation arbitrarily; at the boundary, the initial approximation is determined by expressions (2). The next approximation is determined by the formula

$$u_{i,j}^{(s+1)} = \frac{4\left(u_{i,j-1}^{(s)} + u_{i-1,j}^{(s)} + u_{i,j+1}^{(s)} + u_{i+1,j}^{(s)}\right) + u_{i-1,j-1}^{(s)} + u_{i+1,j+1}^{(s)} + u_{i-1,j-1}^{(s)} + u_{i+1,j-1}^{(s)}}{20}, \quad (3)$$

where s is the iteration number s=0, 1, 2, …, $i = 1, 2, \dots, N-1$, $j = 1, 2, \dots, N-1$.

The calculations (3) continue until the condition (4) is satisfied.

$$\max_{i,j} \left| u_{i,j}^{(s+1)} - u_{i,j}^{(s)} \right| < \varepsilon. \quad (4)$$

As an example, we compute the solution of equation (1) in a square with sides of unit length and the following boundary conditions:

$$u(x, 0) = 1, \quad 0 \le x \le 1$$
$$u(0, y) = 0, \quad 0 < y \le 1$$
$$u(x, 1) = 0, \quad 0 \le x < 1$$
$$u(1, y) = 1, \quad 0 \le y \le 1$$

The results of calculations are shown in Fig. 1, from where it can be concluded that the steady-state temperature distribution inside the bar takes values between the minimum and maximum temperature values at the boundary, which agrees with the maximum principle satisfied by the solution of Laplace's equation.

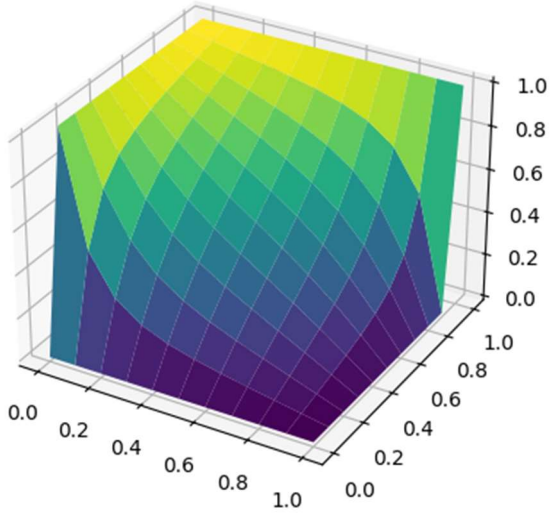


Fig. 1. The surface plot of the temperature distribution inside the beam cross-section.

Using the technique described in [2], we conclude that the solution obtained with (3) and (4) converges to the solution of a problem (1), (2) at the rate of $O(h^4)$ at any initial approximation. Given an accuracy of approximation ε, we compute the mesh step as $h = \sqrt[4]{\varepsilon}$ and the mesh size as $N = \lfloor 1/h \rfloor$ to run the iteration process.

3. SLEPTSOV NET COMPUTING

A Sleptsov net (SN) [7], where a transition fires in multiple instances at a step, generalizes a Petri net [10], making it Turing-complete [11] and running exponentially faster [7].

Carl Petri supplied a bipartite graph, Gill [12] was using for modeling parallel computations, with dynamic elements, called tokens [13], situated within vertices of the graph's first part, called places and depicted as circles or ovals. Vertices of the graph's second part, called transitions and depicted as squares or rectangles, fire consuming tokens from their input places and producing tokens within their output places. The discrete time process of firing transitions represents the behavior of net. An example of computing an expression on integer numbers applying a Sleptsov-Salwicki transition firing rule [11] is shown in Fig. 2.

SN is a tuple $N = (P, T, A, \mu^0)$, where P and T are finite sets of places and transitions, respectively, connected via arcs given by the following mapping $A: (P \times T) \cup (T \times P) \to \mathbb{Z}^{\ge 0}$, where zero value means the arc absence and nonzero value specifies the arc multiplicity. The mapping $\mu^0 \to \mathbb{Z}^{\ge 0}$, gives the initial situation of tokens within places called a marking.

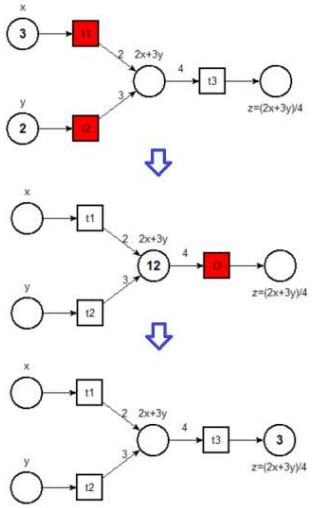


Fig. 2. Computing an expression $z = (2x + 3y)/4$ with an SN.

We use multisets [14] for SN behavior specification. At a step, a maximal submultiset F' of the fireable transition multiset F fires. The arc and the transition firing multiplicities are specified by formulae (5) and (6), respectively.

$$C(p, t) = \mu(p)/A(p, t) \quad (5)$$
$$C(t) = \min_{A(p,t)>0} C(p, t) \quad (6)$$

Here, at the maximal multiset F' choice, we do not split the firing multiplicity of a transition and observe obtaining of a nonnegative next marking, specified by the SN state equation (7) for $p \in P$, $t \in F'$.

$$\mu(p)^{k+1} = \mu(p)^k - \sum_{A(p,t)>0} C(t) \cdot A(p,t) + \sum_{A(t,p)>0} C(t) \cdot A(p,t) \qquad (7)$$

In (3), we consider a division operation as the whole division. Thus, an SN implements directly such operations as addition, subtraction, and also multiplication and division by a constant.

Note that within this paper, in contrast to a series of publications on SN, we use the data-flow approach introduced in [7] for solving the Laplace equation using a stencil with a von Neumann neighborhood that provided the second degree of accuracy compared to the fourth degree of (3), (4) using a stencil with a Moore neighborhood. The Sleptsov-Salwicki rule (5)-(7) yields the maximal parallelism for subsequent hardware implementation. We avoid alternatives, composing structural conflict-free nets [15] where $F' = F$.

Thus, in Fig. 2, we fire two transitions, t_1 and t_2, at the first step, transition t_1 in 3 instances and transition t_2 in 2 instances, then, at step 2, transition t_3 fires in 3 instances. The expression is calculated in 2 steps: the multiplication by a constant is specified by the multiplicity of the transition incoming arc, and the division by a constant is implemented by the transition incoming arc.

We specify the corresponding transition firing sequence as

$$\sigma = \begin{Bmatrix} t_1^3 \\ t_2^2 \end{Bmatrix} t_3^3.$$

In [7], SNs that implement basic arithmetic and logic operations are composed; aspects of processing real numbers and other data structures are studied in [16].

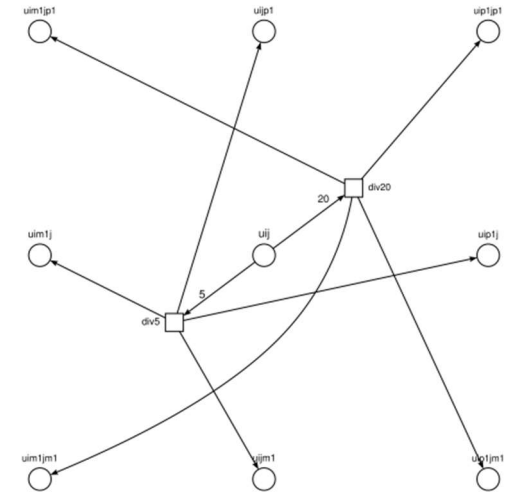
Sleptsov Net Computing (SNC) [17] resolves problems of modern HPC. In this paper, we use SNC as the basic approach to represent mass-parallel computations. We implement machines to run programs drawn in the SN graphical language either in the form of software as virtual machines [18] or in hardware that yields the utmost performance as described in [19] and the present paper.

4. SN PROGRAMS FOR SOLVING PDE

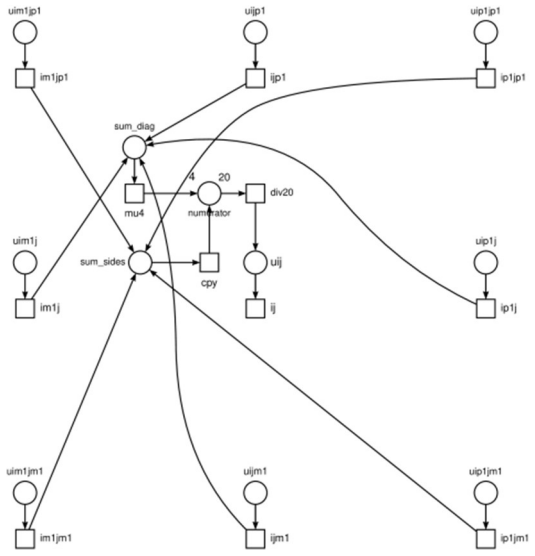
For the most significant tasks of real-time embedded control, we develop a model of computations that is subsequently implemented in a semi-hardware form using FPGA [5,20], where generated Verilog [6] code defines the specialized hardware. We follow a traditional for embedded applications [4,5] way of approximating real numbers as fixed-point numbers or, in even more simplified form, as integer numbers, supposing an initial scaling of numbers with respect to the working range and required accuracy of the function approximation. For this purpose, a UNIT_VALUE is selected; for instance, if a unit equals 0.001, then all the values are expressed with respect to the chosen unit. For using unsigned numbers, the corresponding shift of values is applied; for instance, having a range from -10 to 10 and the above unit, we implement a shift by 10, having the working range of values from 0 to 20000.

Formal issues of convergence for nonnegative integer approximation require further investigation, and with such an approximation we can directly apply Sleptsov nets, which are easy for hardware implementation [19]. Note that there is an intrinsic connection of cellular automata (CA) [9] with the iteration technique, especially when approximated with nonnegative integer numbers. Indeed, the node value, together with a given neighborhood and iteration formula, represents the CA rule [9]. Thus, we can apply CA theory [9] to study the convergence issues [2]. An unchanged CA configuration directly represents convergence, though some cycles of configurations with low differences of the node values are possible to achieve as well.

In Fig. 3, we present two different approaches to modeling iteration processes by SNs; the first approach (Fig. 3a) is fast and rough, though it yields simpler constructs; the second approach (Fig. 3b) is more precise, though it runs three times slower and contains more graphical elements. The node model represents a clan (functional subnet) assuming applicability of the clan composition technique for the model properties analysis.



(a) one time cycle per iteration;



(b) three time cycles per iteration.

Fig. 3. Node model for integer number approximation lattice for embedded systems design.

Both nodes of Fig. 3, implement the iteration formula (3), the node shown in Fig. 3b implements it directly, while the node shown in Fig. 3a implements it with transformations (5) as a sum of 8 addendums in the third line; an intermediate variant is possible corresponding to the second line and requiring two time cycles.

$$u_{i,j}^{(s+1)} == \frac{4\left(u_{i,j-1}^{(s)} + u_{i-1,j}^{(s)} + u_{i,j+1}^{(s)} + u_{i+1,j}^{(s)}\right) + u_{i-1,j-1}^{(s)} + u_{i+1,j+1}^{(s)} + u_{i-1,j-1}^{(s)} + u_{i+1,j-1}^{(s)}}{20}$$

$$= \frac{u_{i,j-1}^{(s)}+u_{i-1,j}^{(s)}+u_{i,j+1}^{(s)}+u_{i+1,j}^{(s)}}{5} + \frac{u_{i-1,j-1}^{(s)}+u_{i+1,j+1}^{(s)}+u_{i-1,j-1}^{(s)}+u_{i+1,j-1}^{(s)}}{20} =$$

$$\frac{u_{i,j-1}^{(s)}}{5} + \frac{u_{i-1,j}^{(s)}}{5} + \frac{u_{i,j+1}^{(s)}}{5} + \frac{u_{i+1,j}^{(s)}}{5} + \frac{u_{i-1,j-1}^{(s)}}{20} + \frac{u_{i+1,j+1}^{(s)}}{20} + \frac{u_{i-1,j-1}^{(s)}}{20} + \frac{u_{i+1,j-1}^{(s)}}{20}. \quad (8)$$

For the node shown in Fig. 3a, a 5×5 lattice is composed in Fig. 4. It runs one iteration in one time cycle for the entire matrix of values for the synchronous implementation of SN (Sleptsov-Salwicki rule [11]). The lattice design is based on the transformations of formula (3) represented with (8).

Each of eight addendums is implemented by a separate transition of a neighboring node in Fig. 3a; separating division operations affects the accuracy of computations, though leads to a one-time-cycle implementation with enhanced performance. Actually, a transition divides the node value by 5 or 20 and adds the obtained values to the neighboring nodes; the division is implemented using the corresponding weight of the transition incoming arc, while the addition is implemented implicitly with a few incoming arcs of the corresponding place. Actually, the boundary nodes are not recalculated, while each internal node place has 8 incoming arcs bringing the corresponding addendums of the final expression in the transformations (24).

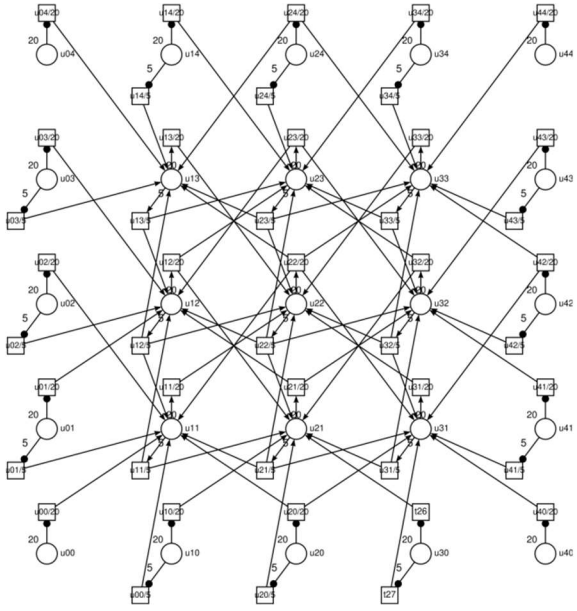


Fig. 4. An integer number approximation lattice solving PDE for embedded systems design with a node shown in Fig. 3a.

We avoid conflicts to omit the choice of fireable multiset of SN transitions and use formula (5) – (7) directly to simulate the SN step. Actually, in Fig. 3a, we replace place $u_{i,j}$ by a pair of places $u_{i,j}$ and $u_{i,j}{}'$ (one place for each outgoing arc) to avoid conflicts, directing the same incoming arcs to each of them that leads to a rather tangled picture. For thorough implementations, we also need to clear the division reminder from places $u_{i,j}$ and $u_{i,j}{}'$.

If this fastest scheme is not suitable from the error of integer division point of view, a three time-cycle implementation strictly following formula (8) will be the best choice for nonnegative integer approximation. The corresponding node model is represented in Fig. 3b, yielding a lattice shown in Fig. 5.

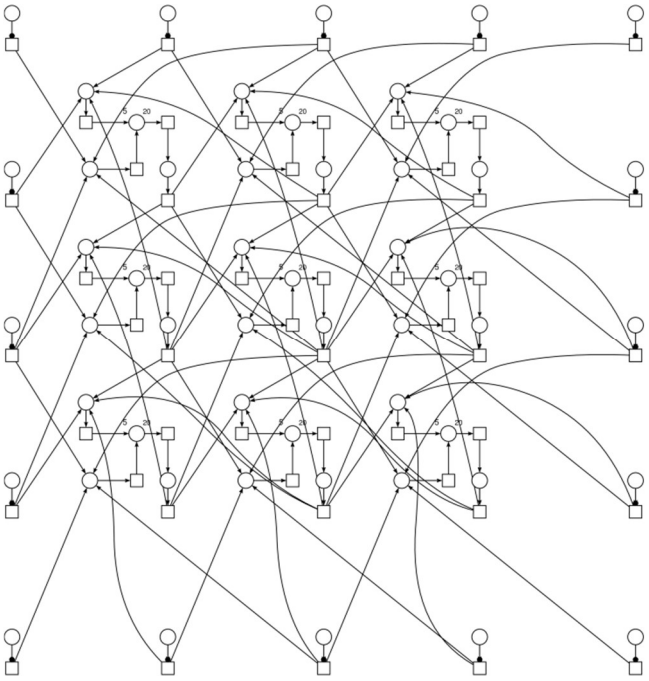


Fig. 5. An integer number approximation lattice solving PDE for embedded systems design with a node shown in Fig. 3b.

For formal specification of SN lattices, we actually apply the multiset rewriting system based technique [14], which yields expression (9) specifying the lattice shown in Fig. 4 with duplicated places $u_{i,j}$ and $u_{i,j}{}'$ and expression (10) specifying the borders to obtain a structural conflict-free net [15]. For each line, a prefix gives the transition name; its input places are written to the left of the arrow symbol; its input places are written to the right of the arrow symbol. To avoid a bulky expression (10) with a separate line for each border and each corner of the domain, we suppose that the outgoing arcs, leading to borders, are omitted; thus, we filter the right part of (10) with the following conditions: $i > 0, i < N-1, j > 0, j < N-1$.

$$\begin{pmatrix} div5_{i,j}\colon 5 \cdot u_{i,j} \to u_{i-1,j}, u'_{i-1,j}, u_{i+1,j}, u'_{i+1,j}, u_{i,j-1}, u'_{i,j-1}, u_{i,j+1}, u_{i,j+1}{}' \\ div20_{i,j}\colon 20 \cdot u'_{i,j} \to u_{i-1,j-1}, u'_{i-1,j-1}, u_{i-1,j+1}, u'_{i-1,j+1}, u_{i+1,j-1}, u'_{i+1,j-1}, u_{i+1,j+1}, u_{i+1,j+1}{}' \end{pmatrix}$$

$$1 \le i \le N-1, 1 \le j \le N-1 \quad (9)$$

$$\begin{pmatrix} div5_{i,j}\colon 5 \cdot u_{i,j} \to 5 \cdot u_{i,j}, u_{i-1,j}, u_{i+1,j}, u_{i,j-1}, u_{i,j+1}, \\ div20_{i,j}\colon 20 \cdot u'_{i,j} \to 20 \cdot u_{i,j}{}', u'_{i-1,j-1}, u'_{i-1,j+1}, u'_{i+1,j-1}, u_{i+1,j+1}{}' \end{pmatrix}$$

$$i = 0, i = N-1, j = 0,\ j = N-1 \quad (10)$$

For verification of SN programs, we apply system Tina [21], which recently implements SNs and represents an IDE [18] for the novel SNC paradigm of computations [17-19].

5. GENERATING VERILOG CODE ON SN PROGRAM

The composed SN model represents the lattice specification for further implementation of the approach [19] to compile an SN into Verilog code [6] for its subsequent implementation on FPGA [5]. We employ either direct mapping of the lattice into explicit Verilog statements or using loops as a template with respect to the lattice size parameter *N*. The corresponding code snippet is shown in Fig. 6. On the growing edge of the clock signal, we recalculate the entire lattice for *K* iterations and then indicate with the built-in LED array the result for an internal node. We run a single iteration during a time cycle; the subsequent fine-tuning of Verilog code supposes adjusting the iteration time with the device clock with the purpose of running the maximal possible number of iterations during a single time cycle. Since the conventional division of integer numbers just omits the fractional part, we need to provide a correction for better rounding. Because of this reason, we compare the division reminder *num%20* with a given constant *ROUND_EDGE* to add an extra unit. As a result of fine-tuning, we can recommend using the constant *ROUND_EDGE* values from 10 to 12.

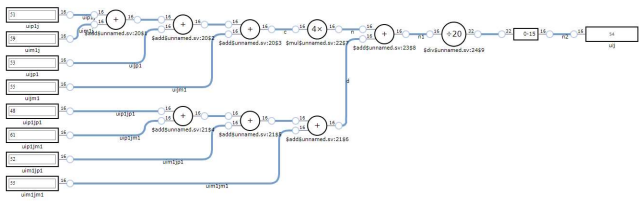
```
always @(posedge sys_clk) begin
     if (counter < `K ) begin
        for(i = 1; i < `N-1; i = i+1) begin // iterate
          for(j = 1; j < `N-1; j = j+1) begin
            cross=u[i+1][j]+u[i-1][j]+u[i][j+1]+u[i][j-1];
            diag=u[i+1][j+1]+u[i+1][j-1]+
                              u[i-1][j+1]+u[i-1][j-1];
            num=4*cross+diag;
            u1[i][j] = num/20+((num%ROUND_EDGE)<11)?0:1;
          end
        end
        for(i = 1; i < `N-1; i = i+1) begin // u=u1;
          for(j = 1; j < `N-1; j = j+1) begin
            u[i][j] = u1[i][j];
          end
        end

     end
     else
       led = ~ u[2][2][5:0];
     end
```

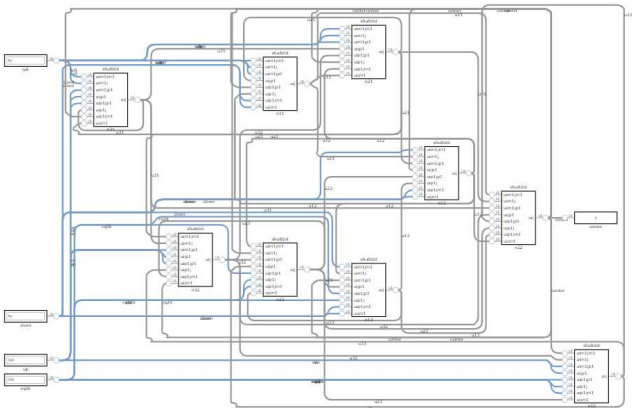
Fig. 6. Listing of Verilog code snippet of the iteration (8) hardware implementation.

We tested the node (Fig. 7b) and composed a lattice for *N=5* that recalculates the value of its internal part represented by a 3×3 lattice using the perimeter nodes as constant boundary conditions. The lattice layout is shown in Fig. 7b and represents the connection of 9 nodes. Here we do not rearrange the automatically obtained layout; though, when optimizing it for production, the lattice structure will be imposed, providing the minimal wire (connections) length and the minimal number of intersections to trace on additional layers.

An adjusted constant number of iterations, for a certain range of considered magnitudes, represents a good compromise with respect to the limited resources of embedded applications. A layout of a node, obtained as a result of Verilog code processing, is shown in Fig. 7; here we use neighboring node values according to the names specified in Figs. 3 – 5; it just describes the data flow of formula (3), we use a 16 bits unsigned integer approximation.



a) layout of a node obtained on Verilog code (automatic);



b) layout of lattice with node substitution (automatic).

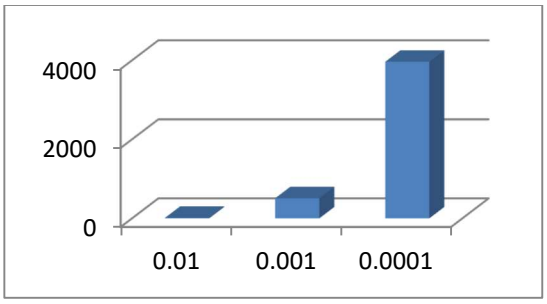
Fig. 7. Layout of 5x5 lattice with constant values on borders.

6. BENCHMARKS FOR FPGA

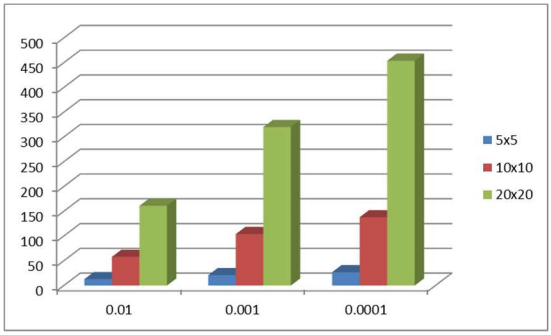
For preliminary benchmarks, we were using the Tang Nano 9k FPGA operated by Gowin FPGA Designer [19]. The dependence of solution time on the lattice size for required precision is shown in Fig. 8. For 50MHz FPGA clock, we run an iteration in about 20 ns; thus, we calculate about *K=50* iterations in 1 us that provides a rather good approximation of the optimal control specified by PDE (a system of PDEs). Benchmarks obtained for the specified FPGA completely correspond to simplified evaluations by multiplying the number of iterations by the device's basic time cycle. Modern advanced FPGAs work on frequencies higher that 1 GHz, which allows us to run *K=1000* iterations in 1 us or to have a nanosecond reaction with fewer numbers of iterations.

One could think that the chosen number of steps within the figures, $N = 4$ ($N + 1 = 5$ dots) serves for illustrative purposes only, though we will show that it provides rather good approximation as well. Having a step *h=1/4=0.25*, the scheme (3), (4) provides the accuracy of approximation $\varepsilon \approx h^4 = 0.25^4 \approx 0.004$, using the same value in formula (9), that requires about 20

iterations to run. Tang Nano 9k FPGA, having a modest cost of some ten pounds, can handle a mesh of $N = 10$, that provides theoretical accuracy of computations of about 0.0001.



a) real numbers;



b) integer approximation on FPGA.

Fig. 8. Dependence of the solution time (number of iterations) on the actual accuracy of approximation.

For computations represented with Fig. 7, we were using fine-tuning which includes, for the embedded implementation, adjusting the following set of parameters: *N*, *UNIT_VALUE*, *K*, and *ROUND_EDGE*. We are reaching the required accuracy by increasing *N* and decreasing *UNIT_VALUE*, which requires increasing *K* to achieve the theoretical accuracy, though because of the rounding error accumulation, it shifts the result with respect to the precise solution that is adjusted by the appropriate *ROUND_EDGE* choice.

As for the discussion of the presented approach prospects, we would like to note that in the present paper, we consider (for the case study) constant boundary conditions and, as a goal, the convergence to an unmovable or pulsing with low error configuration of the nonnegative integer approximation (a cellular automaton). For real-life embedded applications, extended systems are of certain interest where sensors are attached in the form of changing border conditions, and the goal values for the optimal control are taken from certain specified locations of the lattice. In this case, we are not interested in halting the computing structure but keeping it running and processing possible updates of sensors. For control systems that change time scale and have prolonged idling periods, switching to energy-saving modes of functioning is possible, when achieving good convergence, to awake on a change of sensor inputs.

7. CONCLUSIONS

In this paper, we have applied infinite Sleptsov (generalized Petri) nets for computing structure design aimed at fast mass-parallel numerical solving of practically significant problems specified by PDE or systems of PDE focusing on the domain of embedded applications with specialized hardware prototyped with FPGA.

We start with a given PDE or a system of PDEs and boundary conditions, then apply finite difference methods using a mesh of appropriate shape for its numerical solution with required accuracy (error). For an efficient solution, in an ideal case, the task spatial structure should be directly mapped into the corresponding computing-communication structure, which is possible for dedicated hardware implementation or its prototyping with FPGA for embedded control applications. We focus on the connection mesh obtained using Moore's [9] for considerably better approximation (as a fourth degree compared to the second degree depending on the finite difference mesh step size).

Composing SN data-flow programs, we considered the integer number approximation of the iteration technique that represents, in essence, a cellular automaton approach. SN graphical language has been employed as an intermediate construct to generate Verilog code for dedicated hardware implementations or prototyping on FPGA. For the FPGA implementation, benchmarks and prospects of application for fast process control have been discussed. It concerns plane and rocket engines, (thermo) nuclear power stations, the trajectory of hypersonic moving objects, etc.

8. REFERENCES

[1] Gardner, C. L. *Applied Numerical Methods for Partial Differential Equations*, Springer Nature Switzerland, 2024.

[2] Samarskii, A. A. *Theory of Difference Schemes*, Taylor & Francis Group, 2001.

[3] Strikwerda, J. *Finite Difference Schemes and Partial Differential Equations*, Society for Industrial Mathematics, 2007.

[4] Adam Taylor, Dan Binnun, Saket Srivastava. *A Hands-On Guide to Designing Embedded Systems*, Artech House, 2021.

[5] *Principles and Structures of FPGAs*, Hideharu Amano, ed., Springer, 2018.

[6] Joseph Cavanagh. *Verilog HDL: Digital Design and Modeling*, CRC Press, 2017.

[7] Dmitry A. Zaitsev. *Sleptsov Nets Run Fast*, IEEE Transactions on Systems, Man, and Cybernetics: Systems 46(5), 682–693.

[8] A.I. Sleptsov, A.A. Yurasov. *Computer-Aided Design of Flexible Computer-Aided Manufacturing Systems*, Tekhnika, Kiev, 1986.

[9] J. Kari. *Theory of cellular automata: a survey*, Theoret. Comput. Sci. 334 (2005) 3–33.

[10] Tadao T. Murata. *Petri Nets: Properties, Analysis and Applications*, Proc. of the IEEE 77(4), 541-580 (1989).

[11] Dmitry A. Zaitsev. *Strong Sleptsov nets are Turing complete, Information Sciences*, Vol. 621, 2023, 172-182.

[12] S. Gill. *Parallel Programming*, The Computer Journal 1 (1) (1958) 2–10, https://doi.org/10.1093/comjnl/1.1.2.

[13] C.A. Petri, *Kommunication mit Automaten*, Technischen Hoschule Darmstadt, 1962, Ph.D. thesis.

[14] A. Alhazov, S. Verlan. *Minimization strategies for maximally parallel multiset rewriting systems*, Theoretical Computer Science 412 (17) (2011) 1581-1591,

[15] Dmitry A. Zaitsev, Anatoly I. Sleptsov. *State equations and equivalent transformations for timed petri nets*, Cybern. Syst. Anal. 33 (1997) 659–672.

[16] A.A. Kostikov, N.D. Zaitsev, O.V. Subotin. *Realisation of the double sweep method by using a Sleptsov net*, Int. J. Parallel, Emergent Distributed Systems 36 (6) (2021) 516–534.

[17] Dmitry Zaitsev. *Sleptsov Net Computing resolves problems of modern supercomputing revealed by Jack Dongarra in his Turing Award talk in November 2022*, International Journal of Parallel, Emergent and Distributed Systems, 2023.

[18] Dmitry A. Zaitsev, Tatiana R. Shmeleva, Qing Zhang, and Hongfei Zhao. *Virtual Machine and Integrated Developer Environment for Sleptsov Net Computing*, Parallel Processing Letters, Vol. 33, No. 03, 2350006, 2023.

[19] Ruiyao Xu, Si Zhang, Ding Liu, and Dmitry A. Zaitsev. *Sleptsov net based reliable embedded system design on microcontrollers and FPGAs*, Proc. of 2024 IEEE International Conference on Embedded Software and Systems (ICESS), Wuhan, China, Dec. 13-15, 2024.

[20] R. Baxter et al. *Maxwell - a 64 FPGA Supercomputer*, Second NASA/ESA Conference on Adaptive Hardware and Systems (AHS 2007), Edinburgh, UK, 2007, pp. 287-294.

[21] Berthomieu, B., O.-P. Ribet, and F. Vernadat. *The tool TINA – construction of abstract state spaces for Petri nets and time Petri nets*, Int. J. Prod. Res. 42(4), 2741–2756 (2004).