INTIACC: A Programmable Floating-Point Accelerator for Partial Differential Equations

Paul Xuanyuanliang Huang, *Student Member, IEEE*, Yannis Tsividis, *Life Fellow, IEEE*, and Mingoo Seok, *Senior Member, IEEE*

***Abstract*— This article presents a 32-bit floating-point (FP32) programmable accelerator for solving a wide range of partial differential equations (PDEs) based on numerical integration methods. Compared to prior works that have fixed-point systems and are only applicable to specific types of PDEs, our proposed, integration accelerator for PDEs, named INTIACC, accelerator consists of 16 locally interconnected processing elements (PEs) where each PE is a fully programmable reduced instruction set computer (RISC) processor with an FP32 arithmetic logic unit (FP32 ALU) and a custom-designed instruction set architecture (ISA). These features enable INTIACC to generate solutions with high precision and a wide dynamic range and also allow users to implement different numerical algorithms to perform high-order integration methods and to evaluate nonlinear functions. In addition, we create a novel slow-global-fast-local clocking scheme in which PEs operate asynchronously with each other most of the time. We prototype the INTIACC test chip in 65 nm, with a core area of 0.975 mm^2. Running at an average local clock frequency of 570 MHz at 1 V, it offers a single-precision computation throughput of 9.12 GFLOPS. Testing results show that with a similar energy-delay product, INTIACC is up to 40× faster than the prior state-of-the-art PDE solver.**

***Index Terms*— 32-bit floating point, boundary conditions (BCs), custom instruction set architecture (ISA), hybrid global–local clocking scheme, numerical integration, partial differential equations (PDEs), programmable accelerator.**

I. Introduction

HIGH-PERFORMANCE scientific computing is at the heart of many modern technological achievements that rely on computer simulations. These include integrated circuit design, new drug development, aerodynamics design for new cars and planes, and accurate weather forecasting. In many of these simulations, the bulk of the computational load is dedicated to solving partial differential equations (PDEs) [1], [2], [3], [4]. PDEs fundamentally describe many physical phenomena, such as electric charge distribution or airflow around a moving object [5]. Without PDE-based simulations, many modern engineering products we enjoy would have been impossible.

To solve PDEs on computers, numerical approximations replace the infinitesimally small differentials with finite quantities that a computer can handle [6]. Then, the PDE is solved on each discretized point in (most commonly) time and space. These processes can take hours to days and cost a large amount of energy on multi-core CPUs and GPUs, which are the common platforms used to tackle PDEs nowadays [7]. Therefore, researchers developed hardware accelerators [8], [9], [10], [11], [12], [13], [14], [15], [16], [17] to improve speed and lower energy consumption. The accelerators use different approaches to achieve these goals.

One interesting direction is to use analog computing techniques to seek higher energy efficiency. This was done by Guo et al. [13], where they combined analog and continuous-time digital circuits on a chip to map ordinary differential equations (ODEs) onto a network of interconnected analog multipliers and adders. More recently, Liang et al. [14] proposed another analog chip to solve the 1-D wave equation. One of the main advantages of solving differential equations with an analog computer is that the solution, which is represented in analog voltage or current, does not have convergence issues as one may have when using numerical approaches. However, the drawback lies in the device variations inherent in analog circuits and the difficulty of scaling to larger problems.

Also, analog but following a completely different approach was the work from [15]. Instead of mapping the equations directly, they built analog in-memory-computing (IMC) hardware to perform reduced-precision matrix–vector multiplications required in the numerical algorithms for solving PDEs. As the first paper to employ IMCs in a PDE accelerator, their work demonstrated algorithmic optimizations that can be used to design PDE solvers with limited arithmetic precision.

More recently, Mu and Kim [16] proposed to map Poisson's equation, a PDE used in physics modeling, onto a 2-D array of digital processing elements (PEs) where each PE in the array corresponds to a discretized grid point in the spatial domain where the problem is defined. The PEs compute in a bit-serial fashion with variable arithmetic precision, ranging from 4 to 16 bits fixed point. They proposed a variable-precision technique and a residue-based algorithm

Manuscript received 8 October 2023; revised 30 January 2024 and 14 March 2024; accepted 15 March 2024. This article was approved by Associate Editor Sanu Mathew. This work was supported in part by NSF under Grant 1840763. *(Corresponding author: Paul Xuanyuanliang Huang.)*

The authors are with the Department of Electrical Engineering, Columbia University, New York, NY 10027 USA (e-mail: xh2373@columbia.edu).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/JSSC.2024.3379308.

Digital Object Identifier 10.1109/JSSC.2024.3379308

Equation	Class	Boundary Conditions	JSSC16 [13]	JSSC20 [15]	JSSC21 [14]	JSSC23 [16]	INTIACC
Ordinary differential equations	ODE	Dirichlet	✓	✗	✗	✗	✓
		Neumann	✗				✓
		Time-dependent	✗				✓
Advection equation $\frac{\partial u}{\partial t}=-v\left(\frac{\partial u}{\partial x}+\frac{\partial u}{\partial y}\right)$	1st-order hyperbolic	Dirichlet	✗	✗	✗	✗	✓
		Neumann					
		Time-dependent					
Wave equation $\frac{\partial^2 u}{\partial t^2}=c^2\left(\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}\right)$	2nd-order hyperbolic	Dirichlet	✗	✗	✓	✗	✓
		Neumann					
		Time-dependent					
Heat equation $\frac{\partial u}{\partial t}=D\left(\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}\right)$	Parabolic	Dirichlet	✗	✗	✗	✗	✓
		Neumann					
		Time-dependent					
Conv-diff equation $\frac{\partial u}{\partial t}=D\left(\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}\right)-v\left(\frac{\partial u}{\partial x}+\frac{\partial u}{\partial y}\right)+R$	Hyperbolic-parabolic	Dirichlet	✗	✗	✗	✗	✓
		Neumann					
		Time-dependent					
Poisson's equation $\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}=b$	Elliptic	Dirichlet	✗	✓	✗	✓	✓
		Neumann		✗		✗	✓
Laplace's equation $\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}=0$	Elliptic	Dirichlet	✗	✓	✗	✓	✓
		Neumann		✗		✗	✓

Fig. 1. Compared with prior works, INTIACC supports a much wider range of PDE problems that have various BCs and parameter settings.

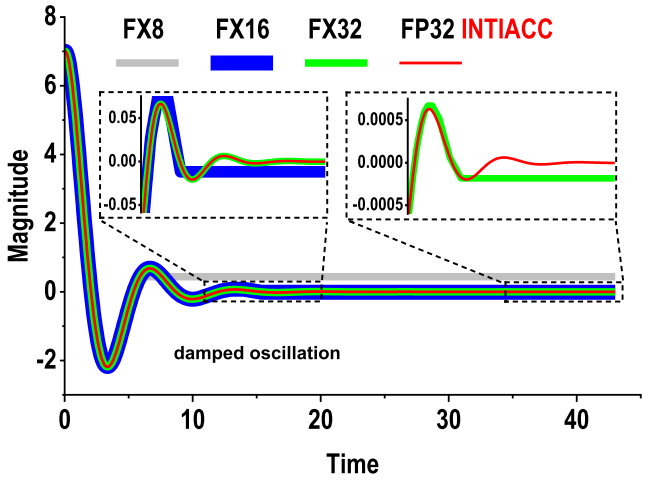


Fig. 2. Oscillation magnitude versus time solution of a damped oscillation problem, computed with 8-, 16-, and 32-bit fixed-point (FX8, FX16, and FX32) arithmetic and with FP32 arithmetic. Among the four arithmetic systems, FP32 has by far the largest dynamic range and hence can produce accurate solutions as the oscillation magnitude diminishes. In contrast, the FX systems are not able to adapt to the magnitude of change as effectively.

to enable more energy-efficient PDE solving on bit-serial hardware.

However, despite the novelties of the prior works, several aspects limit their practical applicability. First, the types of equations they map onto hardware are constrained to ODEs [13] or a single type of PDEs such as the Poisson's, Laplace's [15], [16], or the wave equation [14]. Second, the boundary conditions (BCs), an essential specification of every PDE, that the prior accelerators support are mostly limited to the constant type (called the Dirichlet BC). Yet, for real-world problems, there are many PDEs with non-constant BCs. Third, most of the prior accelerators lack the capability of mapping nonlinear terms such as x^2, $\sin x$, and $1/x$, which frequently appear in problems with practical applications [23]. Fourth, the systems in prior works have limited arithmetic precision and dynamic range because they use analog signals or short-length (less than 16 bits) fixed-point numbers to represent numerical values. PDEs arising in practical problems require much higher precision and dynamic range.

Targeting these challenges, we propose an **int**egrat**i**on **acc**elerator for PDEs, named INTIACC, that can solve a wide range of PDEs. This wide applicability is illustrated in Fig. 1, where we compare INTIACC with prior accelerators. INTIACC can also solve problems outside the listed examples if the target PDE can be mapped to INTIACC-supported algorithms (see Section II). Architecture-wise, the accelerator consists of 16 locally interconnected PEs, each containing a programmable custom instruction set architecture (ISA) reduced instruction set computer (RISC) processor. This architecture allows the user to program the PEs to map different types of PDEs with different BCs (i.e., Dirichlet, Neumann, and time dependent). Also, the architecture improves computation efficiency by limiting data communication to only among neighboring PEs. For the computing unit in each PE, we employ a 32-bit floating-point ALU (FP32 ALU), which allows INTIACC to offer high-precision solutions with a much larger dynamic range than those of prior fixed-point systems. We demonstrate the importance of a large dynamic range using a damped oscillation problem as an example (Fig. 2).

INTIACC's programmability enables users to implement many algorithms that are unattainable in systems without programmability. For example, we can implement high-order integration algorithms such as the Runge–Kutta fourth-order method (RK4) [24], which offers much better accuracy than the first-order Euler's method. Also, users can map nonlinear terms by employing numerical algorithms in each PE. For instance, there are the Goldschmidt fast division algorithm [25] for computing $1/x$, Newton–Raphson for $(x)^{1/2}$, and Taylor approximation for $\sin x$. Compared to using lookup tables (LUTs) to map these functions [10], [13], INTIACC's algorithmic approach is much more memory-efficient and also offers much higher result precision due to the algorithm's iterative refinement.

In addition, we designed a hybrid global and local clocking scheme for INTIACC. Under this scheme, each PE operates at its own local clock's frequency (around 570 MHz), while the PEs only synchronize using a slow global clock (10–50 MHz) when necessary. This scheme takes advantage of the algorithmic nature of numerical integration, that is, the PEs only need to exchange data at the beginning of each integration step, and hence, they can operate on mutually independent clocks within each such step. From a hardware perspective, the scheme is designed to relieve design constraints on clock trees and also save global clock distribution power.

In this article, we will first introduce some of the main algorithms supported by INTIACC in Section II. Then, in Section III, we will present design details, including the chip's top architecture, programming model, and the microarchitecture of the PE. Section IV presents the measurement results of the prototype chip and our test-case PDE settings. We use various equation settings to thoroughly test the INTIACC's performance under different computing tasks and compare it with the prior art and CPU/GPU.

Then, in Section V, we analyze the scalability of the INTIACC architecture in terms of memory size and latency. Section VI concludes this article.

II. Algorithmic Background

This section describes some numerical approaches that INTIACC supports in solving PDEs. The discussion includes how we discretize the spatial domain, integration methods to perform time stepping, methods for solving time-independent PDEs, and the various BCs for PDEs.

A. Domain Discretization

One of the most popular numerical discretization methods used in solving PDEs is the finite difference method (FDM). This method approximates the differentials in a PDE by the differences between neighboring discretization points. For example, we show the heat equation (1) commonly used to model heat transmission. In this equation, the variable u is the temperature. Under a set of initial and BCs (e.g., the temperature at the domain boundary at $t = 0$ is 100 °C), u becomes a function of time and space. However, instead of having an explicit functional form, the relation is specified in terms of partial derivatives

$$\frac{\partial u}{\partial t} = D\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right). \tag{1}$$

Here, we show how to use the FDM to solve the heat equation. First, we apply the FDM to discretize the spatial dimensions, which is to approximate the second-order differentials $(\partial^2 u/\partial x^2)$ and $(\partial^2 u/\partial y^2)$ with $(u_{i-1,j} - 2u_{i,j} + u_{i+1,j}/(\Delta x)^2)$ and $(u_{i,j-1} - 2u_{i,j} + u_{i,j+1}/(\Delta y)^2)$, respectively [22]. Here, Δx and Δy represent the discrete increments in the two spatial dimensions. We use the subscript letters i and j to denote the x- and y-coordinates of each discretized grid point in the 2-D domain. This approximation originates from the Taylor series, and here, we use only the nearest two neighboring grid points in each approximation

$$\left(\frac{du_{i,j}}{dt}\right) = D\left(\frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{(\Delta x)^2} + \frac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{(\Delta y)^2}\right). \tag{2}$$

Using these approximations, we convert the time- and space-dependent PDE (1) into equations only dependent on the time variable t, shown in (2). These equations are ODEs because the temperature variable $u_{i,j}$ at each grid point is only differentiated with respect to one independent variable, the time t. Due to this reason, we can replace the partial differential $(\partial/\partial t)$ with the ordinary differential (d/dt). Note that the number of independent equations represented in (2) equals the number of grid points in the domain, which should be determined based on how much spatial granularity is needed for the solution.

B. Time Integration—Euler's Method

To solve ODEs such as (2), we need to integrate both sides of the equations with respect to time. Numerically, the integration process is illustrated in (3). Here, we use the superscript letter n to denote the current timestep index, and Δt is the timestep size. Next, we must find a suitable approximation for the slope (du/dt) at each timestep. The most basic approach in approximating this slope is Euler's method. For example, if we use Euler's method for the space-discretized heat equation from (2), the slope (du/dt) is directly equal to the right-hand side of the equation. Then, (3) becomes (4)

$$u^{(n+1)} = u^{(n)} + \left(\frac{du}{dt}\right)\Delta t \tag{3}$$

$$u_{i,j}^{(n+1)} = u_{i,j}^{(n)} + D\left(\frac{u_{i-1,j}^{(n)} - 2u_{i,j}^{(n)} + u_{i+1,j}^{(n)}}{(\Delta x)^2} + \frac{u_{i,j-1}^{(n)} - 2u_{i,j}^{(n)} + u_{i,j+1}^{(n)}}{(\Delta y)^2}\right)\Delta t. \tag{4}$$

To start the evaluation, we use a set of initial conditions (i.e., $u_{i,j}$'s value $u_{i,j}^{(0)}$ at $t = 0$) and BCs (i.e., $u_{i,j}$'s values at domain boundaries). The solution then consists of the temperature values at each grid point of the domain at every timestep.

Euler's method suffers from relatively large numerical errors compared to other higher order integration methods. Specifically, it can be shown that the accumulated error (which describes the accumulated difference between the numerical and the exact solutions) is proportional to the step size Δt. In contrast, higher order integration methods make the error proportional to a higher power of Δt so that these methods produce much more accurate solutions than Euler's using the same integration step size.

C. Time Integration—Runge–Kutta Fourth-Order Method

One such higher order method is RK4 [24]. It is one of the most commonly used methods for numerical integration nowadays. It is a fourth-order method because the accumulated error mentioned above is proportional to the fourth power of the step size Δt. In other words, if the step size decreases by a factor of 10, the error decreases by 10 000. To compare, Euler's method would only reduce the error by the same factor of 10

$$k_1 = \frac{du}{dt} = f(t, u) \tag{5}$$

$$k_2 = f\left(t + \frac{1}{2}\Delta t, u + \frac{1}{2}\Delta t k_1\right) \tag{6}$$

$$k_3 = f\left(t + \frac{1}{2}\Delta t, u + \frac{1}{2}\Delta t k_2\right) \tag{7}$$

$$k_4 = f(t + \Delta t, u + \Delta t k_3) \tag{8}$$

$$k_{\mathrm{RK4}} = \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4). \tag{9}$$

The RK4 differs from Euler's method in the formula used to approximate the slope (du/dt) in (3). For example, instead of directly using the right-hand side of (2) to serve as the slope approximation, RK4 performs extra calculations to generate a slope that leads to a much more accurate solution for the next step. To illustrate, we first generalize (2) to the form in (5), where k_1 denotes the slope approximation used by

Euler's method and $f(t, u)$ represents the right-hand side of the spatially discretized equation.

What RK4 does is to compute three more slopes, i.e., k_2–k_4, and then use all four slopes to generate a weighted final slope value to be used in the integration process of (3). These slope calculations are shown in (6)–(9) in which the final slope is denoted with k_{RK4}.

D. Time-Independent PDEs

Unlike the heat equation, some PDEs are only dependent on the spatial variables and independent of time and hence require a different solution method. Two of the most notable equations of this kind are Poisson's and Laplace's equations, where the former is a generalization of the latter. These equations are widely used in physics. For example, they can model the electric potential distribution caused by an electric charge. The Poisson's equation is specified in the following equation:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = b \tag{10}$$

where u is the variable of interest, while the right-hand side $\mathbf{b}$ is a known quantity

$$\frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{(\Delta x)^2} + \frac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{(\Delta y)^2} = b \tag{11}$$

$$u_{i,j}^{(n+1)} = \frac{1}{4}\left(u_{i-1,j}^{(n)} + u_{i+1,j}^{(n)} + u_{i,j-1}^{(n)} + u_{i,j+1}^{(n)} - h^2 b\right). \tag{12}$$

Similarly, as presented before, we use FDM to discretize the spatial domain and convert the equation into (11). Then, we use Jacobi's iteration to refine an initial-guess solution to approach the final solution iteratively. The iteration formula is shown in (12), where, for simplicity, we assume that Δx and Δy are both equal to h.

Note that in (12), the superscript n no longer represents the timestep index as in time-dependent equations but is an iteration index. We start from $n = 0$ where the right-hand side u values of (12) are either given by initial guess or from the BCs. As n becomes larger, the solution gets closer to the exact solution.

E. Boundary Conditions

BCs are essential for a PDE. They specify the relations that the target variable satisfies on the domain's boundaries. There are several types of BC. The first is called the Dirichlet BC. This type of BC has constant values. The second is the Neumann BC, which specifies that the derivative of the variable is a constant. Furthermore, there could be time-dependent BC, where either the variable or its spatial derivative changes over time. We illustrate these BCs in Fig. 3 in the context of the heat equation specified over a rectangular domain.

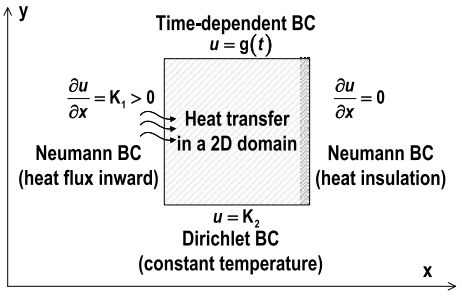


Fig. 3. Illustration of different BCs for the heat equation.

III. Hardware Implementation

A. Top Architecture

As shown in Fig. 4(a), the INTIACC accelerator consists of 16 interconnected programmable PEs and circuits for global clock generation and interfacing with off-chip. The PE interconnect bit width is 32 since we use the FP32 number representation. We build bidirectional interconnects since the wires are locally distributed and have very small lengths, leading to low-cost physical design. Compared to [16], INTIACC does not have special-purpose components at the chip boundaries to set the BCs since the user programs the BC in boundary PEs.

We chose to implement a 4 × 4 PE array because it allows us to have an equal number of PEs on each boundary. The 4 × 4 architecture also has four non-boundary PEs interacting with each other in the center. More PEs could be added at the cost of a larger chip area. We chose FP32 arithmetic precision because it is widely used in scientific computing applications. It can also be extended to 64-bit counterparts at the cost of silicon area and latency.

We chose to implement local communication only because it is sufficient for our target FDM with explicit integration algorithms. Supporting only local communication greatly reduces the amount of hardware dedicated to inter-PE communication and thus greatly improves energy efficiency for our target applications. The main tradeoff is that it may not be straightforward to program other numerical methods (e.g., implicit integration) that require data transfer between non-neighboring PEs. We can make the current hardware perform such data transfer only over many global clock cycles. However, it would degrade the performance and energy efficiency.

B. Programming Method

To program INTIACC for a PDE problem, the user should first discretize the PDE using methods described in Section II. As an example, Fig. 4(b) shows how we can discretize the convection–diffusion equation (which is a generalized form of the heat equation) and convert each discretized equation into a series of arithmetic operations. Then, we write these operations into a PE program. The general form of such a program is illustrated in Fig. 4(c), which consists of five main stages: acquiring solutions from neighboring PEs, evaluating slope, integrating, updating BCs, and finally sending solutions to neighboring PEs.

To support varying BCs, we add instructions to the programs of corresponding boundary PEs, which can evaluate the mathematical operations required for the update of the BCs. These PEs have part of their programs perform computations required for the updates. For example, if the BC specifies that

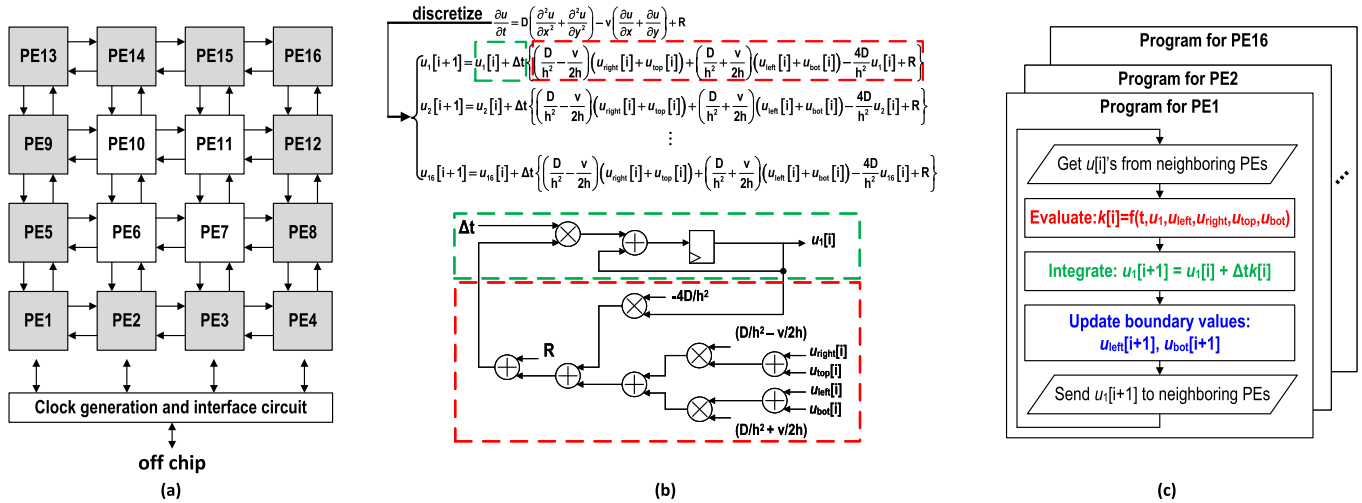


Fig. 4. (a) Top architecture of INTIACC; the PEs have the same hardware but are programmed independently based on the target PDE problem. Boundary PEs are shaded, which also have the same hardware but different software from non-boundary PEs. (b) Discretization of the convection–diffusion equation and a graphical representation of the data flow. (c) Program flowchart for each PE.

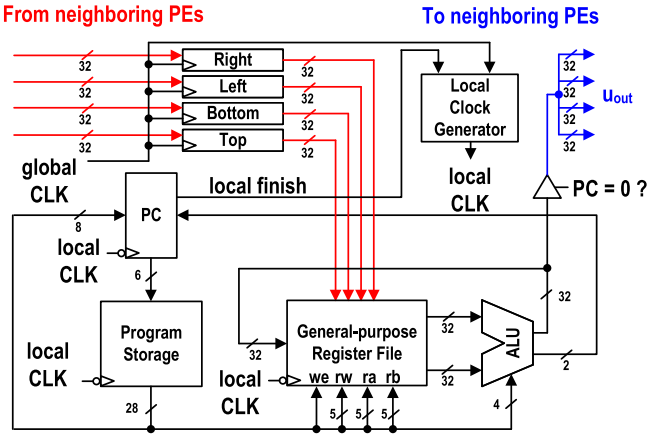


Fig. 5. PE consists of an RISC data path with four special registers and a local clock generator.

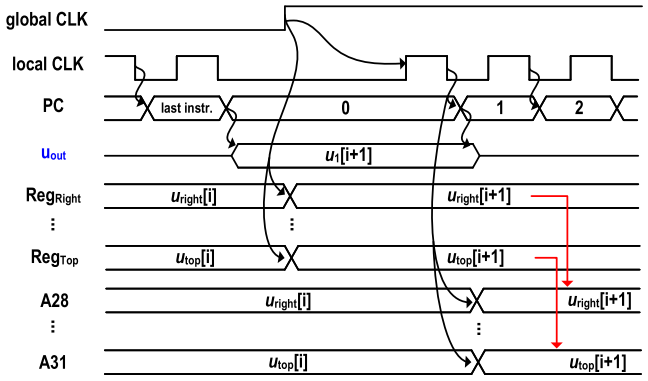


Fig. 6. Timing diagram of the PE operation.

$u^{(i+1)}_{\text{boundary}} = 1.1 \cdot u^{(i)}_{\text{boundary}} + 0.2$, then the program needs a multiply and an add instruction for the update in each iteration. As a result, this boundary PE needs to perform two more instructions, taking two more local clock cycles than non-boundary PEs.

After writing the program using INTIACC's custom instruction set (see Section III-D), the user converts the instructions to a binary form and loads them to the instruction memory in the PEs. Also, the user loads the data memory in the PEs with data needed during the program execution.

C. PE Architecture

As shown in Fig. 5, the PE uses an RISC architecture that consists of an FP32 ALU, instruction, data register files, and a program counter. In addition to these components, we incorporate four special registers in the datapath to capture the solutions from neighboring PEs. The PE can store up to 224 bytes of program instructions and 128 bytes of data. We determined them based on the requirements of the target PDE problems and the available silicon area.

Also, each PE is individually clocked by its local clock, while only the special registers are synchronized to a global clock shared by all PEs. We implemented the local clock generator because we want to keep the local clock fast and the global clock slow. The main tradeoff is the overhead of the generator versus a fast global clock distribution power and the associated timing constraints. By making this tradeoff, we are able to distribute the fast clock only locally within each PE.

We use a simple timing diagram (Fig. 6) to illustrate the PE operation. At each rising edge of the global clock, all PEs capture the solutions from their neighbors. Then, the local clock generator in each PE is activated. Using their local clock, each PE starts to execute instructions. One iteration in a typical PE program first transfers the data captured in the special registers to the last four locations (A28–A31) in the general-purpose register file. Then, the computation part of the program is executed. Finally, at the end of the program execution in each iteration, the local clock is deactivated, and each PE sends its neighbors the solution it just computed. The PEs capture these solutions at the next rising edge of the global clock.

Note that if the PE program uses Euler's method, each global clock cycle corresponds to one timestep in the

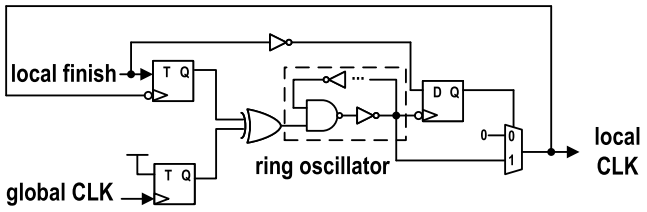


Fig. 7. Local clock generator circuit.

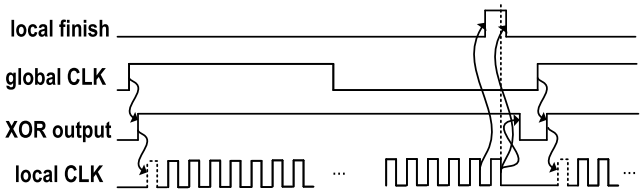


Fig. 8. Operation of the local clock generator.

27 24	22	18 17	13 12	8 7	4 3	0
opcode	we	rw	ra	rb	pcy	pcn

Name	Mnemonic	Operation (in Verilog) (number format: +-MAN*2^EXP)	opcode (binary)
Add	add	R[rw]=R[ra]+R[rb]	0000
Subtract	sub	R[rw]=R[ra]-R[rb]	0001
Multiply	mult	R[rw]=R[ra]*R[rb]	0010
Compare and branch	cpbr	if(R[ra]>R[rb]) R[rw]=-1.0, PC=PC+pcy else R[rw]=1.0, PC=PC+pcn	0011
Get exponential unsigned	gexpu	R[rw]=1.0*2^(EXP of R[ra])	0100
Get exponential reciprocal signed	gexprs	R[rw]=+-1.0*2^(-EXP of R[ra])	0101
Get mantissa unsigned	gmanu	R[rw]=(MAN of R[ra])*2^0	0110
Get fractional	gfrac	R[rw]=fractional part of R[ra]	0111
Get sign	gsign	R[rw]=sign of R[ra]	1000
Absolute value	abs	R[rw]=absolute value of R[ra]	1001
Pass	pass	R[rw]=R[ra]	1010
Jump	jump	PC=PC+R[ra]	1011
Square root initial guess	sqrig	If(EXP of R[ra] is even) R[rw]=(MAN of R[ra])*2^((EXP of R[ra])/2) else R[rw]=1.0*2^(((EXP of R[ra])+1)/2)	1100

Fig. 9. Custom ISA.

PDE program. However, if a higher order integration method is used, each timestep requires multiple rounds of data exchange among neighboring PEs, hence requiring multiple global clock cycles. For example, each timestep in the RK4 method requires four global clock cycles to evaluate the four intermediate slopes.

We show the details of the local clock generator in Fig. 7 and illustrate its operation in Fig. 8. It has a ring oscillator controlled by the XOR result of two toggle flip flops (TFFs), where one is synchronized to the global clock, while the other is synchronized to the generated local clock. For the two TFFs, while the global clock one always toggles at the rising edge, the local clock one only toggles when a local finish signal is high. This local finish signal is asserted when the program counter reaches a predetermined value that indicates the end of the iteration. We use a negative edge triggered D flip-flop and a mux at the output of the ring oscillator. The purpose is to ensure that the generated clock signal always starts from a low state so that any potential setup time violation can be avoided.

D. Custom ISA and Algorithms

We designed a custom ISA for INTIACC, summarized in Fig. 9. The purpose for the custom ISA is to map many essential nonlinear functions required in solving PDEs in a small

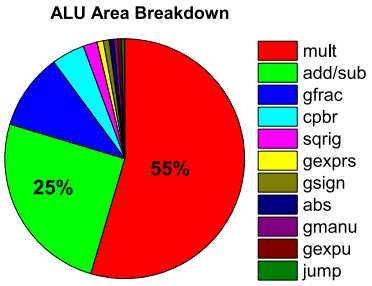


Fig. 10. ALU area breakdown based on instructions.

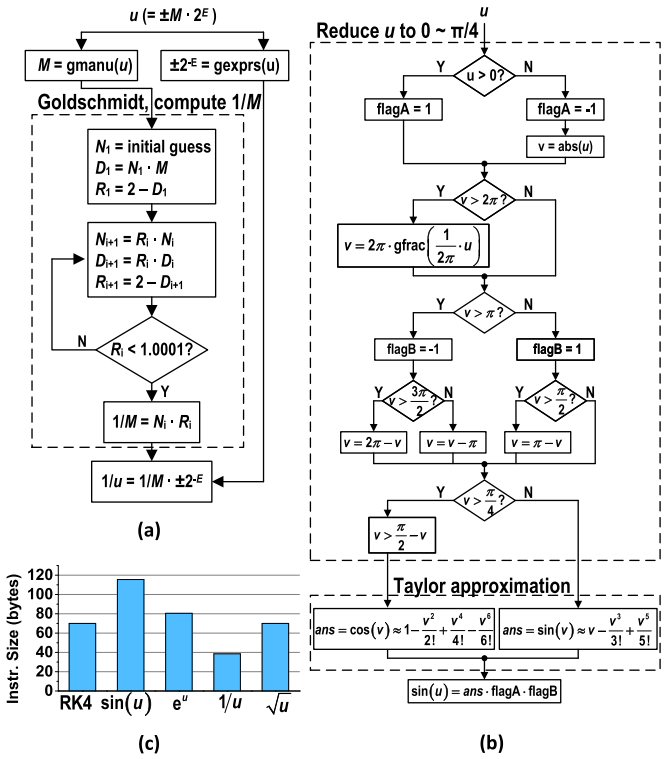


Fig. 11. Implementations of (a) $1/u$ and (b) $\sin u$ functions. (c) Instruction sizes for implementing different nonlinear functions.

number of instructions. The ISA uses a 28-bit instruction format composed of a 4-bit opcode, 1-bit write enable, three 5-bit register index fields, and finally 8 bits for branch operations. The instruction set consists of basic add/mult operations and more complex instructions performed by custom hardware in the ALU. For example, some instructions acquire certain parts of a floating-point number (i.e., gexpu, gexprs, and gmanu). These are designed to implement floating-point division. The ALU hardware overhead for implementing these instructions is about 20%. As shown in Fig. 10, the ALU area is dominated by the floating-point multiplier and adder.

To demonstrate the effectiveness of the ISA, we show two examples of nonlinear function implementation. Fig. 11(a) shows the reciprocal function $1/u$. For this function, we use the Goldschmidt fast division algorithm, which is one of the most commonly used algorithms in performing division. A challenge in implementing this algorithm is that it requires an initial guess of the division result, and this guess should be

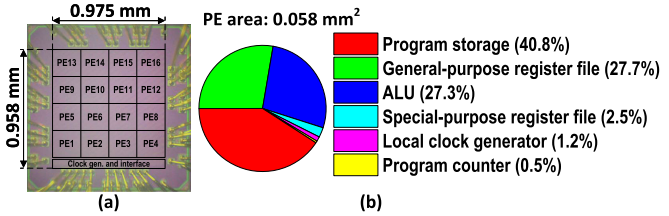


Fig. 12. (a) INTIACC chip die photograph. (b) PE area breakdown.

close enough to the exact result for the algorithm to converge. To generate such an initial guess for $1/u$, where u is an arbitrary FP32 number, we propose the following approach. First, we separate the input floating-point number u's mantissa and exponential parts using specialized hardware. Then, for the exponential part, i.e., the 2^E with the $\pm$ sign in Fig. 11(a) (top), the reciprocal is $\pm 2^{-E}$, which can be computed with the *gexprs* instruction using simple integer hardware. For the mantissa part (the M in the figure) since the floating-point format guarantees that it is always between 1 and 2, we can use any number between 0.5 and 1 as the initial guess of its reciprocal. Finally, we get the initial guess for $1/u$ by multiplying the above results for the exponential and mantissa parts.

The second example is the sine function shown in 11(b). First, we use trigonometric identities to reduce the argument u of the sine function to a value between 0 and $\pi/4$. This process makes use of the *gfrac* instruction that produces the fractional part of the input number. After u is reduced, we then use the Taylor approximation to evaluate $\sin u$.

The Goldschmidt and Taylor approximation methods shown are important examples because they can be used to implement many other nonlinear functions. Specifically, our Goldschmidt routine can be used for general floating-point division, while the Taylor approximation can be used to approximate non-standard nonlinear functions.

In Fig. 11(c), we show the instruction memory space needed for various algorithms. These algorithms can be implemented in a memory-efficient manner using INTIACC's custom ISA. For example, performing Taylor approximation requires finding the fractional part of a floating-point number. Using the conventional RISC ISA, this operation can be performed in about 30 instructions. In comparison, INTIACC can perform this in a single instruction (named *gfrac*) in the custom ISA.

IV. Measurement Results and Analysis

We prototyped INTIACC in the TSMC 65-nm CMOS process. The chip die photograph and an area breakdown of the PE are shown in Fig. 12. To test the INTIACC's performance thoroughly, we designed a set of test cases that are based on different types of PDEs with different BCs and parameter settings. Then, we compare INTIACC's solution time and energy against a CPU (Intel i9-9920X) and GPU (NVIDIA Quadro RTX6000) in which the same problems are solved.

A. Frequency and Power Measurement Results

In Fig. 13, we show the power and average local clock frequency scaling across different supply voltages. We measure

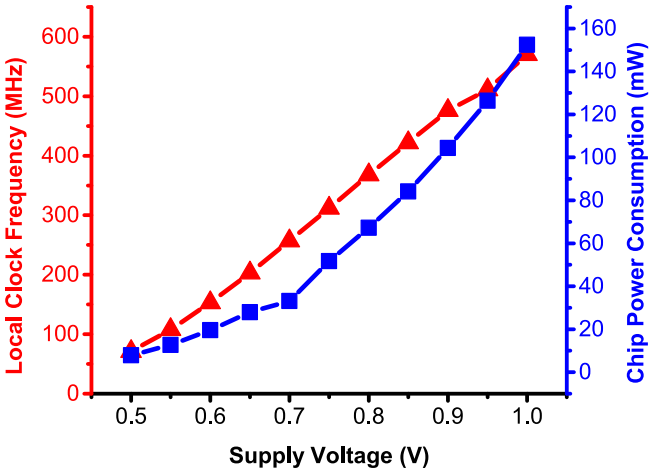


Fig. 13. Average local clock frequency and chip power consumption measurement results across different supply voltages.

the average power by running a typical test program in each PE. Also, the local clock generators are set to a free-running mode in which the global clock positive edges do not deactivate them. In this way, we obtain the maximum power consumption of the chip. Under this setup, at 1-V supply, the chip runs at an average local clock frequency of 572 MHz, offering an aggregate single-precision computation throughput of 9.12 GFLOPS while consuming 152.4 mW.

B. Test Cases

We design seven test cases to test INTIACC's performance. Their configuration details are listed in Table I. We started from a fully fledged convection–diffusion equation in test case 1 (TC1) and made simplified versions of it in TC2–TC4. The physical context of this equation is that of a chemical substance diffusing in flowing water. The variable of interest u is the concentration of the substance. The equation's solution describes the chemical's concentration across time and locations in the water. Using TC1 as an example, the parameter $D = 0.4$ m^2/s is the diffusion coefficient. The water flow velocity field v is a function of time and space and is equal to $0.1e^{-0.1t} + 0.2x + 0.1y$ m/s. Besides, the chemical is constantly generated when diffusing, and the amount generated is a function of the concentration. Here, we assume a sinusoidal form for this function, i.e., $R = \sin u$. Moreover, we specify that the domain boundaries have an exponentially decreasing concentration of this chemical, as given in the BC $u = e^{-0.2t}$.

TC5, TC6, and TC7 target other types of PDEs. The wave equation in TC5 models the transmission of waves; TC6 and TC7 test Poisson's and Laplace's equations, which can be used for modeling thermal or electric phenomena. We tested these PDEs because they use different domain quantization formulas or different evaluation algorithms than those used in the convection–diffusion equation.

We solve the test-case equations with a grid size of 4×4. For the time-dependent cases TC1–TC5, we use an integration timestep size of 0.001 s. We illustrate the solutions of two of the test cases in Fig. 14(a) and (b). Each solution plot contains 16 curves corresponding to the solutions obtained in 16 PEs across iterations.

TABLE I
TEST-CASE PROBLEM SPECIFICATIONS

Test Case	Name	Equation	Parameter Setting	BC
TC1	Convection-diffusion equation	$\frac{\partial u}{\partial t} = D\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) - v\left(\frac{\partial u}{\partial x} + \frac{\partial u}{\partial y}\right) + R$	$D = 0.4, v = f(t,x,y), R = \sin u$	$u = e^{-0.2t}$
TC2			D, v same as TC1, $R = 0$	$u = e^{-0.2t}$
TC3			$D = 0.4$, $v = 0.1$, $R = 0$	Dirichlet
TC4			$D = 0.4$, $v = 0.1$, $R = 0$	Neumann
TC5	Wave equation	$\frac{\partial^2 u}{\partial t^2} = c^2\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right)$	$c = 1$	Dirichlet
TC6	Poisson's equation	$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = b$	$b = -1$	Dirichlet
TC7	Laplace's equation	$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = b$	BC. only	Neumann

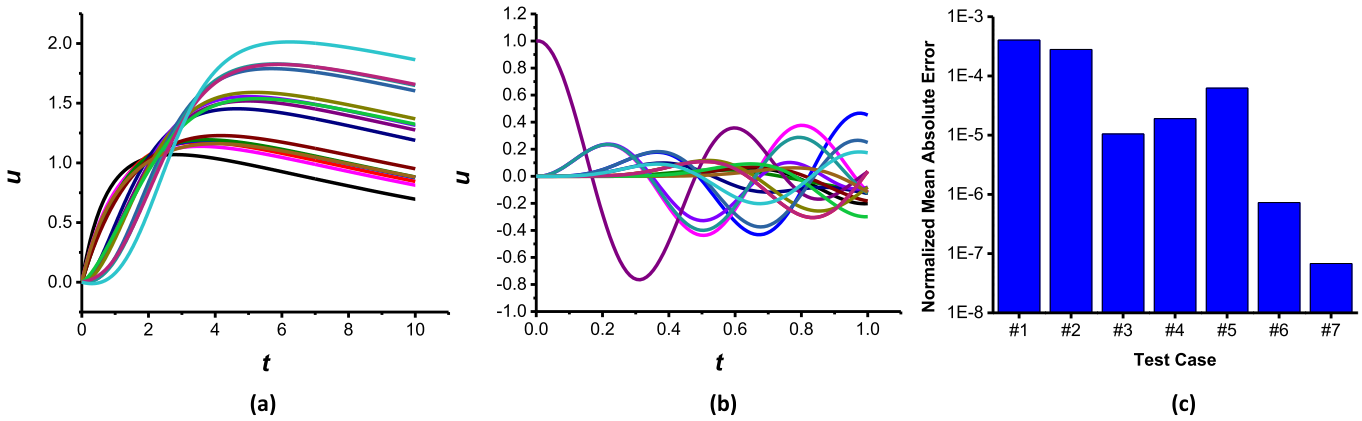


Fig. 14. INTIACC solutions for (a) TC1 and (b) TC5. Each plot consists of 16 curves representing the solutions obtained over iteration steps in the 16 PEs. (c) Solution error of each test case compared to the corresponding 64-bit floating-point solution.

Before solving a PDE problem, INTIACC's global clock frequency (f_{global}) needs to be configured. We determine f_{global} as follows. First, based on the specifications of the PDE problem and the solving algorithm, we write a program for each PE using INTIACC's instruction set. Then, we count the number of instructions for one iteration for each PE, which equals some number N. Assuming that the largest N out of the 16 Ns (for 16 PEs) is N_{max} and the local clock frequency is f_{local}, then f_{global} should satisfy $f_{\text{global}} < f_{\text{local}}/N_{\text{max}} - \tau$, where τ is a margin. The reason for this relation is that each PE must finish executing instructions for one iteration within one global clock cycle. The margin τ accounts for the latency of turning the local clock on and off at the beginning and the end of a global clock cycle, respectively.

Although INTIACC uses FP32 arithmetic, the solutions are still less precise than those from a CPU since the latter uses double-precision floating-point (FP64) arithmetic. Therefore, we plot the mean absolute errors of the solutions obtained in INTIACC across all the test cases, as shown in Fig. 14(c). We note that the errors depend on the specific math operations performed in solving each test case. For example, TC1 and TC2 have the largest normalized errors due to the extra function evaluations in their equation settings. In contrast, the errors are much smaller in TC6 and TC7 since their solutions are obtained through iterative refinement instead of time integration as in TC1–TC5.

C. Performance Comparison

We compare INTIACC's performance with CPU/GPU and the prior accelerator works. For comparison with CPU/GPU,

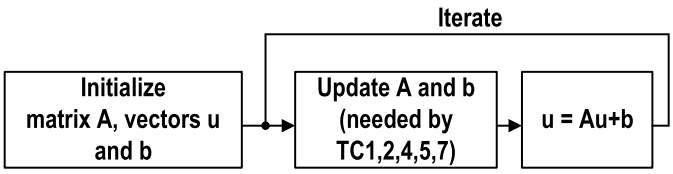


Fig. 15. Program structure used in CPU/GPU for solving the test-case problems.

we program the CPU/GPU to solve the same test-case PDEs with a grid size of 16 using a standard matrix–vector multiplication approach. Specifically, as shown in Fig. 15, the matrix $\mathbf{A}$ contains information about the PDE's parameter settings, while the vector $\mathbf{b}$ contains BCs, and the vector u is the solution. The power measurements of the CPU are performed using real-time CPU power measuring software. An example of this type of software can be found in [27].

The performance improvements of INTIACC over the CPU and GPU are summarized in Fig. 16. We note that the solution times of the GPU are larger than those of the CPU for our test cases. This is because the GPU has latency overhead in launching parallel computing kernels and, therefore, is only faster than the CPU when the problem size is large enough to amortize this overhead [28]. For this reason and the GPU's higher power consumption, the GPU is less energy efficient than the CPU in solving our test cases, so we only compare INTIACC with the CPU for energy consumption. However, we note that as the problem size becomes larger, the GPU solution speed will exceed that of the CPU, and hence, we should compare the performance of INTIACC with that of the GPU.

TABLE II
COMPARISON WITH PRIOR WORKS

		JSSC 2016 [13]	JSSC 2020 [15]	JSSC 2021 [14]	JSSC 2023 [16]	INTIACC
Technology		65nm CMOS	180nm CMOS	180nm CMOS	65nm CMOS	**65nm CMOS**
Analog/Digital		Mixed-signal	Mixed-signal	Analog	Digital	**Digital**
Number Precision		8bit Fixed-Point	5bit Fixed-Point	7.9bit Fixed-Point(1)	16bit Fixed-Point	**32bit Floating-Point**
Applicable to different types of PDEs		No	No	No	No	**Yes**
Problem Dimension(2)		1+1D	2D	1+1D	2D	**2D, 2+1D, 1D, 1+1D**
Variable Boundary Conditions		No	No	Yes	No	**Yes**
Can Compute Nonlinear Terms		Yes	No	No	No	**Yes**
Numerical Routine Support		N/A	No	N/A	No	**Yes**
Core Area		2.0 mm^2	1.868 mm^2	N/A	0.462 mm^2	**0.975 mm^2**
Supply Voltage		1.2 V	1.8 V	2.2 V	0.6-1.2 V	**0.5-1 V**
Clock Frequency		N/A	200 MHz	N/A	25.6 MHz	**570 MHz Local**
Energy per Clock Cycle		N/A	332 pJ	N/A	94 pJ	**263 pJ**
Laplace's Equation(3)	Number of Iterations(4)	Not mappable	Time per iteration not mentioned	Not mappable	473	**390**
	Solution Time(4)				311.7 us	**7.8 us**
	Solution EDP(4)				234 uJ·us	**252 uJ·us**
	Solution NRMS Error				1.00E-04	**3.43E-08**
Conv-diff Equation(5)	Number of Time Steps	Not mappable	Not mappable	Not mappable	Not mappable	**1000**
	Solution Time					**86.2 us**
	Solution EDP					**1.11 mJ·us**
Wave Equation	Solution Time(6)	Not shown	Not mappable	1 ms	Not mappable	**35 us**
	Solution EDP(6)			560 uJ·us		**4.6 uJ·us**

(1) ENOB of the ADC is 7.9 bits. (2) Space dimensions + time dimension (e.g., 2+1D means 2 space dimensions and 1 time dimension). (3) FX16 (range -128 to 128) vs. FP32 simulation results, solving 21x21 grid points, boundary conditions: 1.992, -2.66, 56.33, -101.5. (4) For achieving the same NRMS error (i.e., 1.00E-04), with INTIACC's performance data extrapolated to 21x21 PEs. (5) Equation setup specified in test case 1. (6) Assuming time step size = 1 us, physical time of wave propagation = 1 ms.

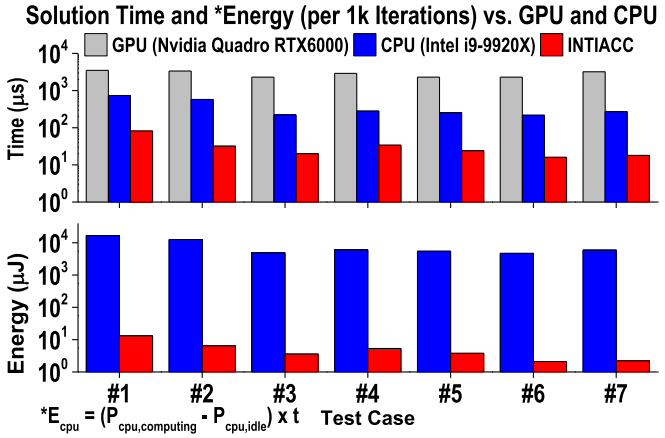


Fig. 16. Solution time and energy of CPU, GPU, and INTIACC. The performance data for INTIACC do not include the overhead of being configured and launched by a host.

We observe the largest speedup in TC2. The main reason is that TC2 requires a large number of updates in the BCs and equation parameters for each iteration. In CPU/GPU, these updates happen to individual elements in the matrix **A** (for equation parameters) and vector **b** (for BCs), which could lead to expensive memory accesses. In contrast, INTIACC performs these updates locally within each PE and mostly in parallel, which enables the speedup. We also note that although TC1 has one more parameter update, i.e., the nonlinear term $R = \sin u$, than TC2 does, INTIACC's performance advantage over the CPU is less in TC1 than in TC2. We think that this is because, in TC1, the evaluation of the sine function is faster on the CPU than on INTIACC due to the much higher clock frequency of the CPU.

We also compare our works with the prior state-of-the-art PDE accelerators, as shown in Table II. We show that INTIACC offers more than 40× speedup while maintaining a similar energy-delay product compared to [16] in solving Laplace's equation. Yet, we note that comparing INTIACC with prior fixed accelerator works is difficult because they use much lower numerical precision and are not programmable (except for [13]) for solving a wide range of problems. We also note that the prior systolic array processors [18], [19], [20], [21] do not focus on the PDE problems. To the best of our knowledge, INTIACC is the first to optimize the systolic floating-point processor array for solving various classes of PDE problems.

V. SCALABILITY ANALYSIS

In this section, we offer a scalability analysis for using the INTIACC architecture to compute larger problems. First, a larger 2-D problem can be mapped by dividing the problem domain and assigning multiple grid points to each PE. For example, a problem with 16 × 16 grid points can be divided into chunks of 4 × 4 grid points where each chunk will be mapped to a PE. The BCs can be processed by the boundary PEs in a similar manner. In the above example, a boundary PE will have four boundary points to update (or eight boundary points if it is at a corner). These updates will be performed in program order based on the mathematical formula associated with the corresponding BCs.

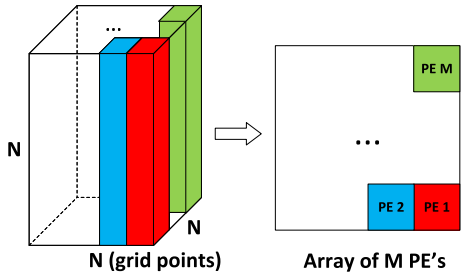


Fig. 17. Map a 3-D problem of size $N \times N \times N$ grid points to an array of M PEs. Each PE is assigned with a column of grid points.

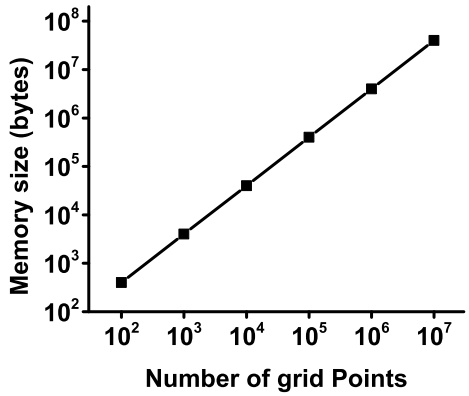


Fig. 18. Total on-chip memory size versus the number of grid points for Laplace's equation.

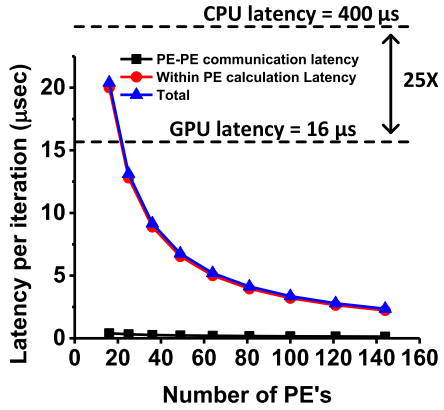


Fig. 19. Latencies of CPU/GPU and INTIACC for solving a 200 × 200 grid size 2-D Laplace's equation. The latency for INTIACC does not include the overhead of being configured and launched by a host.

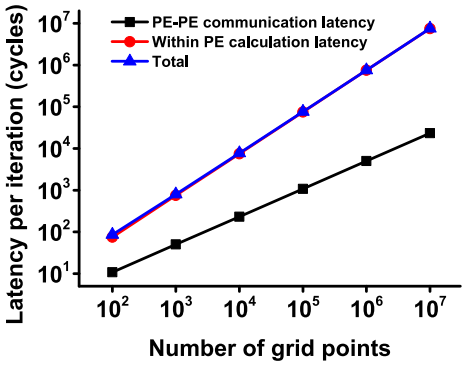


Fig. 20. Latency versus the number of grid points for Laplace's equation, assuming that there are 64 PEs.

For 3-D problems, assume its domain is rectangular and discretized into N grid points on each dimension. To map the problem, each PE can be assigned with a column region of grid points, as illustrated in Fig. 17, where each of the M PEs computes the solutions for N^3/M grid points.

In each iteration, PEs first exchange the values of grid points at the column boundaries with their corresponding neighbors. After doing so, each PE possesses the complete grid-point data needed for updating solutions in the current iteration. Then, each PE performs these updates for all its associated grid points. We note that each PE can map multiple grid points because of the programmable PE architecture, which allows the PE to compute multiple grid-point solutions by sequentially executing a user-defined program.

To map more grid points per PE, we need more on-chip memory. Using Laplace's equation as an example, the PE's memory area would be dominated by the storage of grid-point solutions (each in FP32 format). The total memory size required versus the number of grid points can be plotted as in Fig. 18. Therefore, mapping a ten million grid-point size problem requires about 40 MB of on-chip memory, which we believe is feasible in today's technology.

To solve the 3-D problem, each PE must first exchange data (of the boundary grid-point solutions) with their neighbors. Therefore, the total computation latency for one iteration consists of the communication and calculation latency. We assume that each PE uses a single-port memory and transferring one grid-point solution takes one clock cycle. Then, for Laplace's equation, the communication latency can be calculated as $4 \cdot N^2/(M)^{1/2}$, where M is the number of PEs, which we assume is the square of an integer (e.g., 16 and 64). For Laplace's equation, updating each grid point requires summing the values of its six neighbors in the x-, y-, and z-directions and then multiplying by a parameter. Therefore, it takes six clock cycles to update each grid point within a PE. This leads to a calculation latency of $6 \cdot N^3/M$.

We now compare the performance of the scaled-up versions of INTIACC with GPU for a 2-D Laplace's equation with a grid size of 200 × 200. We compare it to the GPU since, with this problem size, the GPU's performance is more than 20 times better than that of the CPU. This is because the GPU kernel launching overhead is amortized. The comparison is shown in Fig. 19. For each iteration, the GPU has a computation latency of 16 μs, while the CPU is much slower, at 400 μs. The figure also shows the INTIACC's performance across different numbers of PEs without the overhead of being configured and launched by a host. We note that it is not necessary for INTIACC to use off-chip memory since each PE can store the values of many grid points. For example, when the problem of size 200 × 200 is solved on a 10 × 10 PE array, each PE stores and updates 400 grid-point values, which consume 1.6 kB of on-chip memory per PE.

Figs. 20 and 21 summarize the latency versus the number of grid points and the number of PEs, respectively. The total computation latency consists of the communication latency between neighboring PEs and the calculation latency inside

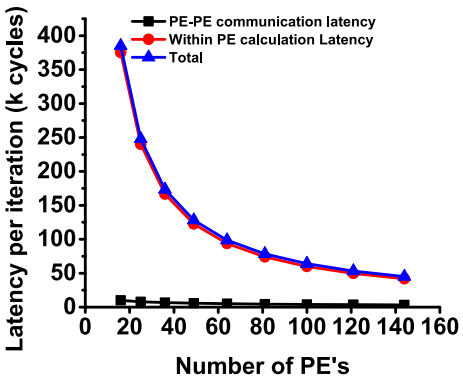


Fig. 21. Latency versus the number of PEs for Laplace's equation, assuming one million grid points.

each PE. The plots show that the total latency is dominated by the calculation latency, especially for large problem sizes with a relatively smaller number of PEs. As the PE array becomes larger, the calculation latency decreases due to increased parallelism, while the communication latency also decreases. We note that we are showing the ideal scaling behavior since at each analysis point, i.e., the number of PEs and the number of grid points, we assume that PEs have sufficient memory to store computing data and, therefore, the PEs fully utilize the memory and computing resources.

The current INTIACC prototype can map 3-D problems if each PE has a larger memory. However, it is not preferable for our current microarchitecture implementation because we optimize it for problems that mostly require data exchange among neighboring PEs. For example, we design the PE to exchange one solution value per iteration with each neighbor, which is sufficient for a 16-point 2-D problem. To map 3-D problems, however, more solution values must be exchanged among non-neighboring PEs. We can make the current hardware perform these exchanges over many global clock cycles. However, this can be done more efficiently if we add additional hardware to perform these exchanges.

VI. Conclusion

Practical PDE problems involve various equation types, BCs, and parameter settings. This requirement can be best satisfied by programmable hardware. The proposed INIACC accelerator is, therefore, designed to have programmable PEs to offer flexibility for users to configure different equations and numerical algorithms. We tested our prototype chip with equations from different PDE categories and compared its performance with that of CPU/GPU and the state-of-the-art PDE accelerator. INTIACC exceeds the state-of-the-art PDE solver by 40× in speed and has a much wider applicable problem range and much higher numerical precision. Moreover, we show that the INTIACC architecture can be easily scaled to map larger problems by incorporating more on-chip memory in each PE and/or increasing the number of PEs.

References

[1] R. E. Bank, W. M. Coughran, W. Fichtner, E. H. Grosse, D. J. Rose, and R. K. Smith, "Transient simulation of silicon devices and circuits," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. CAD-4, no. 4, pp. 436–451, Oct. 1985, doi: 10.1109/TCAD.1985.1270142.

[2] W. Materi and D. S. Wishart, "Computational systems biology in drug discovery and development: Methods and applications," *Drug Discovery Today*, vol. 12, nos. 7–8, pp. 295–303, Apr. 2007, doi: 10.1016/j.drudis.2007.02.013.

[3] A. Jameson, L. Martinelli, and N. A. Pierce, "Optimum aerodynamic design using the Navier–Stokes equations," *Theor. Comput. Fluid Dyn.*, vol. 10, nos. 1–4, pp. 213–237, Jan. 1998, doi: 10.1007/s001620050060.

[4] M. Ghil, S. Cohn, J. Tavantzis, K. Bube, and E. Isaacson, "Applications of estimation theory to numerical weather prediction," in *Dynamic Meteorology: Data Assimilation Methods* (Applied Mathematical Sciences), L. Bengtsson, M. Ghil, and E. Källén, Eds. New York, NY, USA: Springer, 1981, pp. 139–224, doi: 10.1007/978-1-4612-5970-1_5.

[5] W. A. Strauss, *Partial Differential Equations: An Introduction*. Hoboken, NJ, USA: Wiley, 2007.

[6] W. F. Ames, *Numerical Methods for Partial Differential Equations*. New York, NY, USA: Academic, 2014.

[7] C. Barajas, M. K. Gobbert, G. C. Kroiz, and B. E. Peercy, "Challenges and opportunities for the simulation of calcium waves on modern multi-core and many-core parallel computing platforms," *Int. J. Numer. Methods Biomed. Eng.*, vol. 37, no. 11, p. e3244, Nov. 2021, doi: 10.1002/cnm.3244.

[8] Y. Huang, N. Guo, M. Seok, Y. Tsividis, K. Mandli, and S. Sethumadhavan, "Hybrid analog-digital solution of nonlinear partial differential equations," in *Proc. 50th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, New York, NY, USA, Oct. 2017, pp. 665–678.

[9] G. E. R. Cowan, R. C. Melville, and Y. P. Tsividis, "A VLSI analog computer/digital computer accelerator," *IEEE J. Solid-State Circuits*, vol. 41, no. 1, pp. 42–53, Jan. 2006, doi: 10.1109/JSSC.2005.858618.

[10] J. Kung, Y. Long, D. Kim, and S. Mukhopadhyay, "A programmable hardware accelerator for simulating dynamical systems," in *Proc. ACM/IEEE 44th Annu. Int. Symp. Comput. Archit. (ISCA)*, New York, NY, USA, Jun. 2017, pp. 403–415, doi: 10.1145/3079856.3080252.

[11] M. A. Zidan et al., "A general memristor-based partial differential equation solver," *Nature Electron.*, vol. 1, no. 7, pp. 411–420, Jul. 2018, doi: 10.1038/s41928-018-0100-6.

[12] B. Asgari, R. Hadidi, T. Krishna, H. Kim, and S. Yalamanchili, "ALRESCHA: A lightweight reconfigurable sparse-computation accelerator," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, San Diego, CA, USA, Feb. 2020, pp. 249–260, doi: 10.1109/HPCA47549.2020.00029.

[13] N. Guo et al., "Energy-efficient hybrid analog/digital approximate computation in continuous time," *IEEE J. Solid-State Circuits*, vol. 51, no. 7, pp. 1514–1524, Jul. 2016, doi: 10.1109/JSSC.2016.2543729.

[14] J. Liang, N. Udayanga, A. Madanayake, S. I. Hariharan, and S. Mandal, "An offset-cancelling discrete-time analog computer for solving 1-D wave equations," *IEEE J. Solid-State Circuits*, vol. 56, no. 9, pp. 2881–2894, Sep. 2021, doi: 10.1109/JSSC.2021.3074003.

[15] T. Chen, J. Botimer, T. Chou, and Z. Zhang, "A 1.87-mm^2 56.9-GOPS accelerator for solving partial differential equations," *IEEE J. Solid-State Circuits*, vol. 55, no. 6, pp. 1709–1718, Jun. 2020, doi: 10.1109/JSSC.2019.2963591.

[16] J. Mu and B. Kim, "A dynamic-precision bit-serial computing hardware accelerator for solving partial differential equations using finite difference method," *IEEE J. Solid-State Circuits*, vol. 58, no. 2, pp. 543–553, Feb. 2023, doi: 10.1109/JSSC.2022.3174354.

[17] P. X. Huang, D. Jang, Y. Tsividis, and M. Seok, "INTIACC: A 32-bit floating-point programmable custom-ISA accelerator for solving classes of partial differential equations," in *Proc. IEEE 48th Eur. Solid State Circuits Conf. (ESSCIRC)*, Milan, Italy, Sep. 2022, pp. 349–352, doi: 10.1109/ESSCIRC55480.2022.9911441.

[18] S. Kim et al., "Versa: A 36-core systolic multiprocessor with dynamically reconfigurable interconnect and memory," *IEEE J. Solid-State Circuits*, vol. 57, no. 4, pp. 986–998, Apr. 2022, doi: 10.1109/JSSC.2022.3140241.

[19] G. Peng, L. Liu, S. Zhou, S. Yin, and S. Wei, "A 2.92-Gb/s/W and 0.43-Gb/s/MG flexible and scalable CGRA-based baseband processor for massive MIMO detection," *IEEE J. Solid-State Circuits*, vol. 55, no. 2, pp. 505–519, Feb. 2020, doi: 10.1109/JSSC.2019.2952839.

[20] M. Anders et al., "2.9 TOPS/W reconfigurable dense/sparse matrix-multiply accelerator with unified INT8/INTI6/FP16 datapath in 14 NM tri-gate CMOS," in *Proc. IEEE Symp. VLSI Circuits*, Honolulu, HI, USA, Jun. 2018, pp. 39–40, doi: 10.1109/VLSIC.2018.8502333.

[21] S. Shanmuga Sundaram, Y. Khodke, Y. Li, S.-J. Jang, S.-S. Lee, and M. Kang, "FreFlex: A high-performance processor for convolution and attention computations via sparsity-adaptive dynamic frequency boosting," *IEEE J. Solid-State Circuits*, vol. 59, no. 3, pp. 855–866, Mar. 2024, doi: 10.1109/JSSC.2023.3341348.

[22] W. E. Schiesser, *The Numerical Method of Lines Integration of Partial Differential Equations*. San Diego, CA, USA: Academic, 1991.

[23] X. Lü and S.-J. Chen, "Interaction solutions to nonlinear partial differential equations via Hirota bilinear forms: One-lump-multi-stripe and one-lump-multi-soliton types," *Nonlinear Dyn.*, vol. 103, no. 1, pp. 947–977, Jan. 2021, doi: 10.1007/s11071-020-06068-6.

[24] J. C. Butcher, *Numerical Methods for Ordinary Differential Equations*. New York, NY, USA: Wiley, 2008.

[25] R. E. Goldschmidt, "Applications of division by convergence," M.S. dissertation, MIT, Cambridge, MA, USA, 1964.

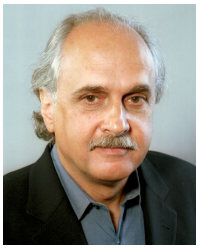
[26] T. Stocker, *Introduction to Climate Modelling*. Berlin, Germany: Springer, 2011, p. 57.

[27] D. Beyer and P. Wendler, "CPU energy meter: A tool for energy-aware algorithms engineering," in *Tools and Algorithms for the Construction and Analysis of Systems* (Lecture Notes in Computer Science), vol. 12079, A. Biere and D. Parker, Eds. Cham, Switzerland: Springer, 2020, doi: 10.1007/978-3-030-45237-7_8.

[28] S. Kim, S. Oh, and Y. Yi, "Minimizing GPU kernel launch overhead in deep learning inference on mobile GPUs," in *Proc. 22nd Int. Workshop Mobile Comput. Syst. Appl.* New York, NY, USA: Association for Computing Machinery, Feb. 2021, pp. 57–63, doi: 10.1145/3446382.3448606.

Paul Xuanyuanliang Huang (Student Member, IEEE) received the bachelor's degree in electrical engineering from the University of Electronic Science and Technology of China (UESTC), Chengdu, China, in 2018, and the master's degree in electrical engineering from Columbia University, New York, NY, USA, in 2020, where he is currently pursuing the Ph.D. degree with the VLSI Laboratory led by Prof. Mingoo Seok.

His research focuses on the design of high-performance accelerator circuits for computing applications.



Yannis Tsividis (Life Fellow, IEEE) received the B.S. degree from the University of Minnesota, Minneapolis, MN, USA, and the M.S. and Ph.D. degrees from the University of California at Berkeley, Berkeley, CA, USA.

He is currently the Edwin Howard Armstrong Professor of Electrical Engineering at Columbia University, New York, NY, USA. He has worked on analog and mixed-signal integrated circuits at the device, circuit, system, signal processing, and computer simulation level.

Dr. Tsividis received the 1984 IEEE W. R. G. Baker Award for the best IEEE publication, the Columbia's Presidential Award for Outstanding Teaching in 2003, the IEEE Undergraduate Teaching Award in 2005, the IEEE Circuits and Systems Education Award in 2010, the IEEE Gustav Robert Kirchhoff Award in 2007, and the Outstanding Achievement Award of the University of Minnesota in 2013. He was a recipient or co-recipient of best paper awards from the European Solid-State Circuits Conference in 1986, the IEEE International Solid-State Circuits Conference in 2003, and the IEEE Circuits and Systems Society (the Darlington Award in 1987 and the Guillemin-Cauer Award in 1998 and 2008). In 2012, he was an elected Professor Honoris Causa at the University of Patras, Patras, Greece.

Mingoo Seok (Senior Member, IEEE) received the B.S. degree (summa cum laude) in electrical engineering from Seoul National University, Seoul, South Korea, in 2005, and the M.S. and Ph.D. degrees from the University of Michigan, Ann Arbor, MI, USA, in 2007 and 2011, respectively, all in electrical engineering.

He was a member of the Technical Staff with Texas Instruments Inc., Dallas, TX, USA, in 2011. Since 2012, he has been with Columbia University, New York, NY, USA, where he is currently an Associate Professor of electrical engineering and the Chair of Computer Engineering. His current research interests include ultra-low-power system-on-chip (SoC) design for emerging intelligent systems, machine-learning VLSI architecture and circuits, variation, voltage, aging, thermal-adaptive circuits and architecture, on-chip integrated power circuits, and nonconventional hardware design, including in-memory computing and analog and mixed-signal computing hardware.

Dr. Seok received the 1999 Distinguished Undergraduate Scholarship from Korea Foundation for Advanced Studies, the 2005 Doctoral Fellowship from Korea Foundation for Advanced Studies, the 2008 Rackham Pre-Doctoral Fellowship from the University of Michigan, the 2009 AMD/CICC Scholarship Award for picowatt voltage reference work, the 2009 DAC/ISSCC Design Contest for the 35-pW sensor platform design, the 2015 NSF CAREER Award, and the 2019 Qualcomm Faculty Award. He is a Technical Program Committee Member for several conferences, including IEEE International Solid-State Circuits Conference (ISSCC) and ACM/IEEE Design Automation Conference (DAC). He serves/served as an Associate Editor for IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—I: REGULAR PAPERS from 2013 to 2015, IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS since 2015, and IEEE SOLID-STATE CIRCUITS LETTER since 2017. He also served as a Guest Editor for IEEE JOURNAL OF SOLID-STATE CIRCUITS (JSSC).