ESSCIRC 2022- IEEE 48th European Solid State Circuits Conference (ESSCIRC) | 978-1-6654-8494-7/22/$31.00 ©2022 IEEE | DOI: 10.1109/ESSCIRC55480.2022.9911441

INTIACC: A 32-bit Floating-Point Programmable Custom-ISA Accelerator for Solving Classes of Partial Differential Equations

Paul Xuanyuanliang Huang, Daniel Jang, Yannis Tsividis, and Mingoo Seok
Department of Electrical Engineering
Columbia University
New York, NY, USA
Email: xh2373@columbia.edu

***Abstract*— We propose a numerical integration accelerator (INTIACC) that speeds up the solution of partial differential equations (PDEs) for scientific computing. In contrast to recent works, INTIACC applies to a variety of PDEs and boundary conditions, has enhanced nonlinear function capability, supports high-order integration algorithms, and uses floating-point arithmetic for orders of magnitude smaller solution error. With all the benefits, our test chip still achieves 40X speed-up over prior accelerators and orders of magnitudes over CPU and GPU based systems.**

Keywords—Digital Accelerator, Scientific Computing, Partial Differential Equations

I. Introduction

PDEs are the core mathematical tool for modeling a wide range of scientific and engineering phenomena, making it of great interest to accelerate their numerical solution. Today's primary choices for solving PDEs are multi-core CPUs [5, 6] and GPUs, but it still takes a long time and a large amount of energy to solve them. This limitation has motivated multiple accelerator architectures [1-4, 7]. However, they are still severely limited in mainly four ways. First, they can only handle ordinary differential equations (ODEs) [3] or a single type of PDE (Poisson's equation) [1, 2], leaving a large number of essential PDEs inapplicable. Second, they only support constant (i.e., Dirichlet) boundary conditions [1-4]. Third, some accelerators do not support the computation of nonlinear terms such as x^2, sin(x), and 1/x, which frequently arise in real-world problems [1, 2]. Fourth, they have low computation precision and limited dynamic range, using analog current signals [3] or 4-16-bit fixed-point arithmetic [1, 2].

II. Proposed Accelerator

A. Architectural Features

INTIACC contains 16 processing elements (PEs), each of which is a fully programmable custom-ISA RISC processor. The programmable architecture allows INTIACC to solve a wide range of PDEs (Fig. 1) and to support Dirichlet, Neumann, and time-dependent boundary conditions. Also, each core employs the 32-bit floating-point ALU and multiplier, allowing it to compute high-precision solutions without dynamic range issues (Fig. 2). Also, the accelerator supports high-order integration algorithms such as the Runge-Kutta 4th order (RK4). Furthermore, it can compute almost any nonlinear terms by evaluating them with numerical routines (i.e., algorithms) programmed in each PE, such as the Goldschmidt fast division for 1/x, Newton-Raphson for √x, and Taylor approximation for sin(x). As compared to using look-up tables (LUTs) [3, 4], the algorithmic method is orders-of-magnitude more efficient in memory usage (Fig. 3). We also adopt a hybrid local and global clock scheme, where each PE operates at a fast local clock (570 MHz), with only the slow global clock (10-50 MHz) distributed across PEs, significantly saving clock power dissipation. We prototyped INTIACC in 65nm (Fig. 12). It vastly outperforms the previous accelerators in applicability and precision while improving speed by 40X (Fig. 14). As compared to the CPU/GPU-based system, INTIACC achieves one to three orders of magnitude improvement both in speed and energy (Fig. 13 bottom right).

	Equation	Class	Boundary Conditions	JSSC16 [3]	JSSC20 [2]	ISSCC21 [1]	INTIACC
	Ordinary differential equations	ODE	Dirichlet	✓	✗	✗	✓
			Neumann	✗			✓
			Time-dependent	✗			✓
PDEs	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 $\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}=f$	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. Applicability compared to prior arts.

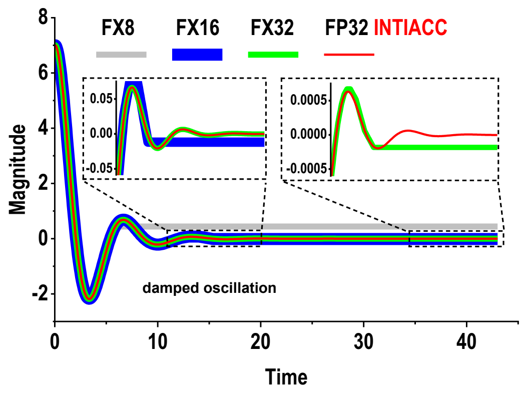


Fig. 2. FP vs. FX in dynamic range.

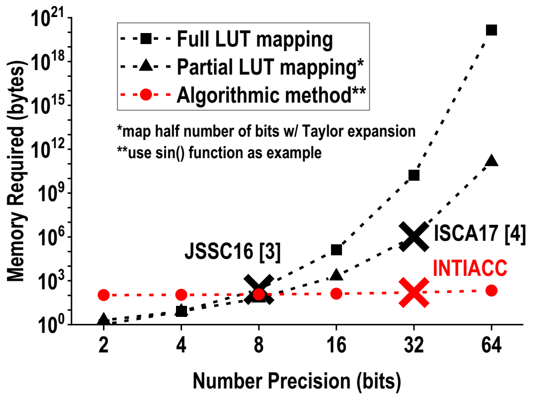


Fig. 3. Methods for mapping nonlinear functions and their memory usage.

B. Programming Model

Fig. 4 illustrates the programming model of INTIACC. First, we discretize a target PDE by applying the finite-difference method to the space dimensions and a numerical integration method (e.g., Euler, RK4) to the time dimension [8] (Fig. 4, top left). The discretization produces 16 coupled equations, each leading to a time-evolving PDE solution at one of the 16 discretized spatial grid points in the domain of interest. Then, we formulate a data flow for each discrete equation. Fig. 4 top right depicts the exemplary data flow of the first equation. It performs Euler's method (the green box), i.e., computing the solution $u_1[i+1]$ for the (i+1)-th time step from the solution $u_1[i]$ and a slope value k[i] (the red box). We can compute k[i] using u[i] 's from neighboring grid points or boundary conditions. Next, we write a program based on each data flow and map it on each PE (Fig. 4 bottom). The figure shows the program for PE1, which starts with getting u[i] values from neighboring grid points, then evaluating k[i], updating $u_1[i+1]$, and sending $u_1[i+1]$ to its neighbors. If the PE is at a boundary that has time-dependent or Neumann boundary conditions, then we add to the program the additional step of updating the boundary values (blue box in Fig. 4). Note that for PDEs that do not have the time dimension (e.g., Poisson's or Laplace's equation), we can create a similar program that performs an iterative algorithm (e.g., Jacobi's) without the time integration. Finally, we write the programs to the PEs through a test circuit consisting of scan chains and a global clock generator, etc.

The work is supported in part by NSF (1840763).

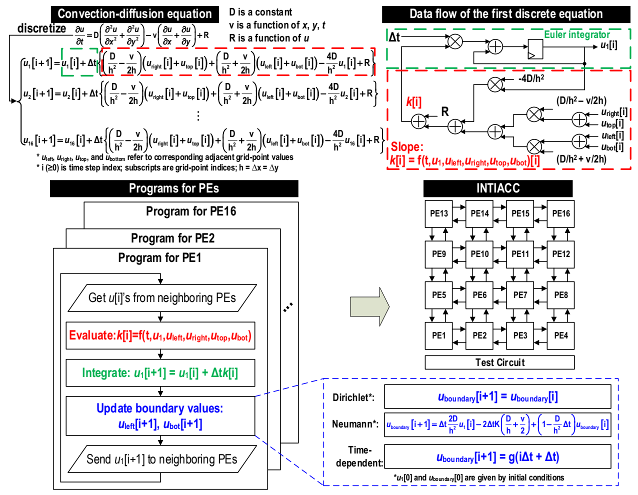

Convection-diffusion equation
D is a constant
v is a function of x, y, t
R is a function of u
discretize
$\frac{\partial u}{\partial t}=D\left(\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}\right)-v\left(\frac{\partial u}{\partial x}+\frac{\partial u}{\partial y}\right)+R$
$u_1[i+1]=u_1[i]+\Delta t\left\{\left(\frac{D}{h^2}-\frac{v}{2h}\right)(u_{right}[i]+u_{top}[i])+\left(\frac{D}{h^2}+\frac{v}{2h}\right)(u_{left}[i]+u_{bot}[i])-\frac{4D}{h^2}u_1[i]+R\right\}$
$u_2[i+1]=u_2[i]+\Delta t\left\{\left(\frac{D}{h^2}-\frac{v}{2h}\right)(u_{right}[i]+u_{top}[i])+\left(\frac{D}{h^2}+\frac{v}{2h}\right)(u_{left}[i]+u_{bot}[i])-\frac{4D}{h^2}u_2[i]+R\right\}$
$u_{16}[i+1]=u_{16}[i]+\Delta t\left\{\left(\frac{D}{h^2}-\frac{v}{2h}\right)(u_{right}[i]+u_{top}[i])+\left(\frac{D}{h^2}+\frac{v}{2h}\right)(u_{left}[i]+u_{bot}[i])-\frac{4D}{h^2}u_{16}[i]+R\right\}$
* u_{left}, u_{right}, u_{top}, and u_{bottom} refer to corresponding adjacent grid-point values
* i (≥0) is time step index; subscripts are grid-point indices; h = Δx = Δy
Data flow of the first discrete equation
Δt
Euler integrator
$u_1[i]$
$-4D/h^2$
k[i]
R
$(D/h^2 - v/2h)$
$u_{right}[i]$
$u_{top}[i]$
$u_{left}[i]$
$u_{bot}[i]$
$(D/h^2 + v/2h)$
Slope:
$k[i] = f(t,u_1,u_{left},u_{right},u_{top},u_{bot})[i]$
Programs for PEs
Program for PE16
Program for PE2
Program for PE1
Get $u[i]$'s from neighboring PEs
Evaluate: $k[i]=f(t,u_1,u_{left},u_{right},u_{top},u_{bot})$
Integrate: $u_1[i+1] = u_1[i] + \Delta t k[i]$
Update boundary values: $u_{left}[i+1]$, $u_{bot}[i+1]$
Send $u_1[i+1]$ to neighboring PEs
INTIACC
PE13 PE14 PE15 PE16
PE9 PE10 PE11 PE12
PE5 PE6 PE7 PE8
PE1 PE2 PE3 PE4
Test Circuit
Dirichlet*: $u_{boundary}[i+1] = u_{boundary}[i]$
Neumann*: $u_{boundary}[i+1]=\Delta t\frac{2D}{h^2}u_1[i]-2\Delta tK\left(\frac{D}{h}+\frac{v}{2}\right)+\left(1-\frac{D}{h^2}\Delta t\right)u_{boundary}[i]$
Time-dependent: $u_{boundary}[i+1] = g(i\Delta t + \Delta t)$
*$u_1[0]$ and $u_{boundary}[0]$ are given by initial conditions

Fig. 4. INTIACC programming methodology.

C. Microarchitecture

Fig. 5 depicts the microarchitecture of the PE. The PE features the proposed hybrid local (570 MHz) and global (10-50 MHz) clock scheme for saving clock power dissipation. In this scheme, each positive edge of the global clock triggers the local clock generator to produce a local clock signal using a ring oscillator (Fig. 6-7). With this active local clock, the PE executes the instructions of the program. At the end of the program execution, as determined by the instruction, the local finish signal is asserted, which disables the local clock and resets the PC to zero, getting the PE ready for the next global clock positive edge. The D flip-flop and mux in the local clock generator ensure that the local clock always starts with the logic-0 value to avoid any setup time violation. We designed the special register file (SRF) in each PE to capture the computation results from neighboring PEs at every global clock positive edge. The captured data are transferred to the last four registers (A28~31) of the general-purpose register file (GRF), which the PE can access with the local clock (Fig. 8).

D. Custom ISA and Algorithms

We designed a custom ISA for the PE to implement numerical routines to perform high-order integration and evaluate nonlinear functions. Fig. 9 shows INTIACC's 28-bit instruction format and a summary of the custom ISA. Fig. 10 shows the pseudo-program implementation of RK4, which INTIACC can perform across four global clock cycles for each time step (it computes one slope [k_1 to k_4] per cycle). RK4 is the de facto standard method for time integration since it is significantly more accurate than the Euler method. On the other hand, Fig. 11 left shows the pseudo-program for computing the sin(u) function using Taylor approximation. Here, the variable u is first reduced to a value between 0 and $\pi/4$ based on trigonometric identities, and Taylor expansion is applied to approximate the function. Meanwhile, an iterative algorithm gives higher accuracy for some nonlinear functions, e.g., the Goldschmidt fast division is optimal to compute the reciprocal function 1/u. For this algorithm to converge, however, it is critical to give an approximately correct initial guess. Therefore, we propose a new procedure (Fig. 11 top right), which exploits the fact that the magnitude of the mantissa part of the floating-point number (i.e., M) is between 1 and 2. Hence, the initial guess for its reciprocal (i.e., 1/M) can be any number from 0.5 to 1. While those programs look complex, the custom ISA allows us to implement them with small numbers of instructions, typically 11-33, consuming 38.5-115.5 bytes of memory (Fig. 11 bottom right).

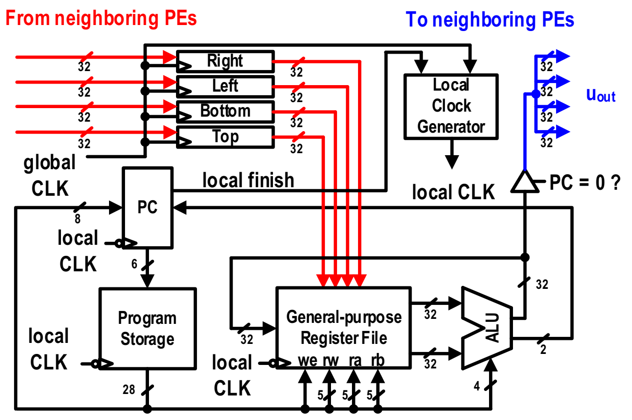


Fig. 5. Microarchitecture of INTIACC's PE.

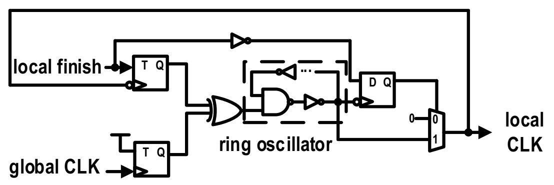


Fig. 6. Local clock generator circuit.

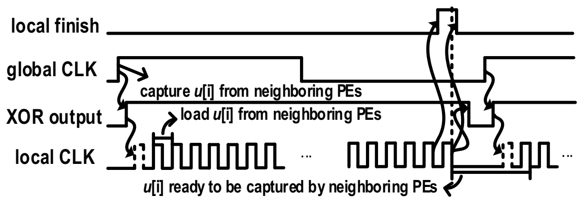

local finish
global CLK
XOR output
local CLK
capture u[i] from neighboring PEs
load u[i] from neighboring PEs
u[i] ready to be captured by neighboring PEs

Fig. 7. Timing diagram of local clock generation.

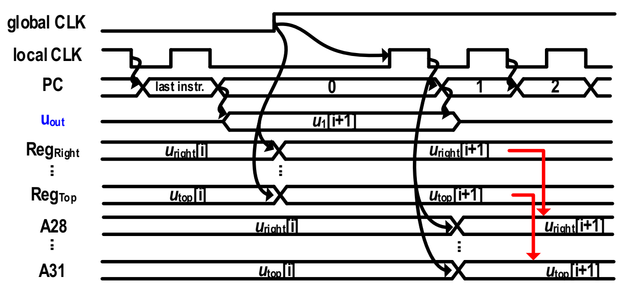
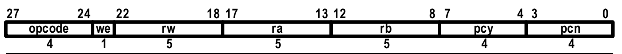


Fig. 8. Timing diagram of PE operation.



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

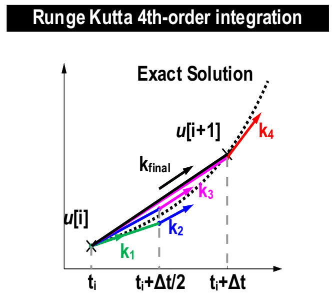
Compare and branch	cpbr	if(R[ra]>R[rb]) R[rw]=-1.0, PC=PC+pcy
		else R[rw]=1.0, PC=PC+pcn
			0011

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

Special notes:
we: write enable (if we=1); PC = PC + pcy by default; If pcn = 1000_{binary}, PC = 0 and local finish = 1;

Fig. 9. instruction format and instructions of the custom instruction set architecture (ISA).



Global CLK Cycle	Computation Performed*
1	$k_1 = f(t_i, u[i])$ Euler stops here
2	$k_2 = f\left(t_i + \frac{1}{2}\Delta t, u[i] + \frac{1}{2}\Delta t k_1\right)$
3	$k_3 = f\left(t_i + \frac{1}{2}\Delta t, u[i] + \frac{1}{2}\Delta t k_2\right)$
4	$k_4 = f(t_i + \Delta t, u[i] + \Delta t k_3)$ $k_{final} = \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$ $u[i+1] = u[i] + \Delta t k_{final}$

Fig. 10. Implementation of RK4 scheme on INTIACC.

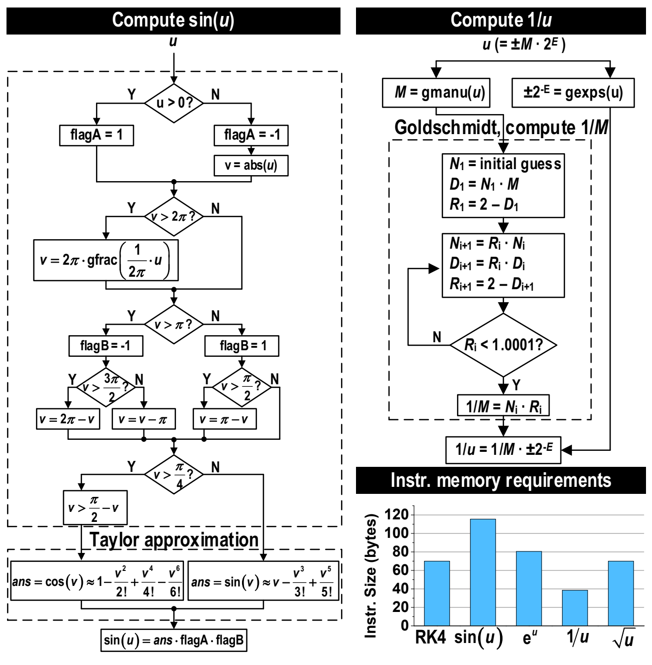


Fig. 11. Examples of numerical routines and the memory requirements.

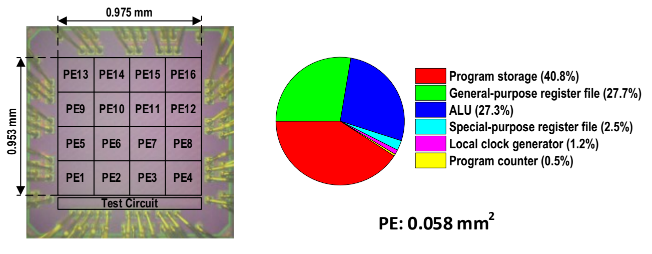
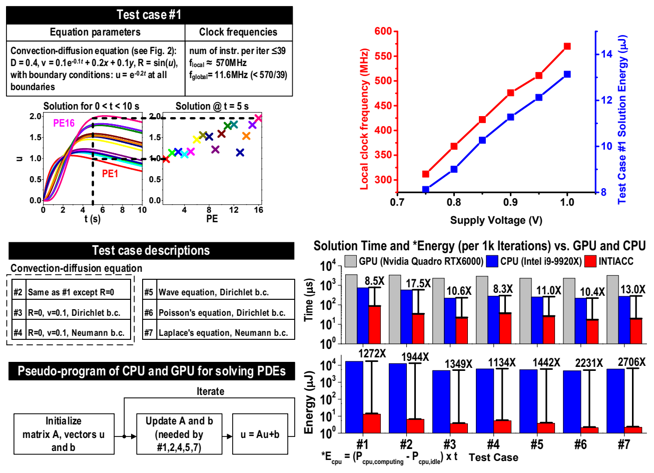


Fig. 12. Die photo and PE area breakdown.



Test case #1	
Equation parameters	Clock frequencies
Convection-diffusion equation (see Fig. 2): D = 0.4, v = 0.1e$^{-0.1t}$ + 0.2x + 0.1y, R = sin(u), with boundary conditions: u = e$^{-0.2t}$ at all boundaries	num of instr. per iter ≤39 f_{local} ≈ 570MHz f_{global} = 11.6MHz (< 570/39)

Test case descriptions

Convection-diffusion equation

#2	Same as #1 except R=0
#3	R=0, v=0.1, Dirichlet b.c.
#4	R=0, v=0.1, Neumann b.c.
#5	Wave equation, Dirichlet b.c.
#6	Poisson's equation, Dirichlet b.c.
#7	Laplace's equation, Neumann b.c.

Pseudo-program of CPU and GPU for solving PDEs

Initialize matrix A, vectors u and b → Update A and b (needed by #1,2,4,5,7) → u = Au+b

Fig. 13. Test case specifications and measurement results vs. CPU and GPU.

III. Measurement Results

We prototyped INTIACC in a 65nm CMOS with a core area of 0.975 mm^2 (Fig. 12). At 1 V, it operates at 570 MHz locally, offering an aggregated computation performance of 9.12 GFLOPS (FP32). Fig. 13 top left shows the equation parameters and measured solutions of our first test case of seven in total (Fig. 13 bottom left). For case #1, we chose the parameter v to depend on both space and time and the boundary conditions to change over time. We solved the equation from t = 0-10 s with a step size of 0.01 s. The global clock frequency (11.6 MHz) is set based on the maximum number of instructions required per iteration. For case#1, INTIACC consumes 13 µJ at 1 V (Fig. 13 top right).

For all test cases, we also compared INTIACC's performance with MATLAB running on an Intel i9-9920X CPU and with CUDA running on an NVIDIA Quadro RTX6000 GPU. For the test-case equations, we chose them to represent different types of PDEs with various parameter settings and boundary conditions. For instance, as compared to case #1, case #2 sets the nonlinear term R to zero, thus eliminating the computation time for evaluating sin(u). Then in cases #3 and #4, we set the parameter v to a constant and used different boundary conditions. For the programs running on the CPU and GPU, we used a conventional matrix/vector approach (Fig. 13 bottom left) [9]. The matrix A (16-by-16) contains the parameter information of the equation, while the vector b (16-by-1) includes the boundary conditions. We summarize the test results in Fig. 13 bottom right. We observed up to 17.5X speed and 2,706X energy improvements over the CPU. The speed-up stems from the efficient in-place computing of INTIACC's PEs. Finally, Fig. 14 compares INTIACC with recently published accelerators. INTIACC significantly expands the diversity of accelerable problems in equation types, boundary conditions, and applicable numerical algorithms with orders of magnitude better solution errors, dynamic range, and speed.

		JSSC 2016 [3]	JSSC 2020 [2]	ISSCC 2021 [1]	INTIACC
Technology		65nm CMOS	180nm CMOS	65nm CMOS	**65nm CMOS**
Analog/Digital		Mixed-signal	Mixed-signal	Digital	**Digital**
Number Precision		8bit Fixed-Point	5bit Fixed-Point	16bit Fixed-Point	**32bit Floating-Point**
Applicable to different types of PDEs		No	No	No	**Yes**
Problem Dimension[1]		1+1D	2D	2D	**2D, 2+1D, 1D, 1+1D**
Variable Boundary Conditions		No	No	No	**Yes**
Can Compute Nonlinear Terms		Yes	No	No	**Yes**
Energy Efficiency Against CPU		N/A	N/A	N/A	**1,100X-2,700X**
Speedup Against CPU		N/A	N/A	N/A	**8-17X**
Numerical Routine Support		N/A	No	No	**Yes**
Core Area		2.0 mm^2	1.868 mm^2	0.462 mm^2	**0.975 mm^2**
Supply Voltage		1.2 V	1.8 V	0.6-1.2 V	**0.5-1 V**
Clock Frequency		N/A	200 MHz	25.6 MHz	**570 MHz Local**
Energy per Clock Cycle		N/A	332 pJ	94 pJ	**263 pJ**
Laplace's Equation[2]	Number of Iterations[3]	Not mappable	Time per iteration not mentioned	473	**390**
	Solution Time[3]			311.7 us	**7.8 us**
	Solution EDP[3]			234 uJ·us	**252 uJ·us**
	Solution NRMS Error			1.00E-04	**3.43E-08**
Conv-diff Equation[4]	Number of Time Steps	Not mappable	Not mappable	Not mappable	**1000**
	Solution Time				**86.2 us**
	Solution EDP				**1.11 mJ·us**

[1]space dimensions + time dimension (e.g., 2+1D means 2 space dimensions and 1 time dimension)
[2]FX16 (range -128 to 128) vs. FP32 simulation results, solving 21x21 grid points, boundary conditions: 1.992, -2.66, 56.33, -101.5
[3]For achieving the same NRMS error (i.e., 1.00E-04), with INTIACC's performance data extrapolated to 21x21 PEs
[4]equation setup specified in Fig. 5a

Fig. 14. Comparison with prior arts.

References

[1] J. Mu et al., "A 21x21 Dynamic-Precision Bit-Serial Computing Graph Accelerator for Solving Partial Differential Equations Using Finite Difference Method," ISSCC, pp. 230-231, 2021.

[2] T. Chen et al., "A 1.87-mm2 56.9-GOPS Accelerator for Solving Partial Differential Equations," IEEE JSSC, vol. 55, no. 6, pp. 1709-1718, June 2020.

[3] N. Guo et al., "Energy-Efficient Hybrid Analog/Digital Approximate Computation in Continuous Time," IEEE JSSC, vol. 51, no. 7, pp. 1514-1524, July 2016.

[4] J. Kung et al., "A Programmable Hardware Accelerator for Simulating Dynamical Systems," IEEE/ACM ISCA, pp. 403-415, 2017.

[5] P. Vivet et al., "IntAct: A 96-Core Processor With Six Chiplets 3D-Stacked on an Active Interposer With Distributed Interconnects and Integrated Power Management," IEEE JSSC, vol. 56, no. 1, pp. 79-97, Jan. 2021.

[6] K. Bryson, "NVIDIA Announces CPU for Giant AI and High Performance Computing Workloads", NVIDIA, Apr. 12, 2021.

[7] M. Zidan et al., "A General Memristor-Based Partial Differential Equation Solver," Nature Electronics, pp. 411-420, July 2018.

[8] Schiesser, W. E. (1991), The Numerical Method of Lines Integration of Partial Differential Equations, Academic Press, San Diego.

[9] Bruaset, Tveito, A., & Bruaset, A. M. (Are M. (2006). Numerical solution of partial differential equations on parallel computers / Are Magnus Bruaset, Aslak Tveito (eds.). Springer. https://doi.org/10.1007/3-540-31619-1