ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/322135343

High-Performance Hardware Accelerators for Solving Ordinary Differential Equations

Conference Paper · June 2017

DOI: 10.1145/3120895.3120919

CITATIONS
0

READS
30

6 authors, including:

Christoforos Kachris
National Technical University of Athens
87 PUBLICATIONS **947** CITATIONS

SEE PROFILE

Dimitrios Soudris
National Technical University of Athens
537 PUBLICATIONS **2,059** CITATIONS

SEE PROFILE

Some of the authors of this publication are also working on these related projects:

Cognitive Heterogeneous Reconfigurable Optical Network (CHRON) View project

SWAN-iCare View project

All content following this page was uploaded by Christoforos Kachris on 11 January 2018.

The user has requested enhancement of the downloaded file.

High-Performance Hardware Accelerators for Solving Ordinary Differential Equations

***Abstract*—Ordinary Differential Equations (ODEs) are widely used in many high-performance computing applications. However, contemporary processors generally provide limited throughput for these kinds of calculations. A high-performance hardware accelerator has been developed for speeding-up the solution of ODEs. The hardware accelerator has been developed both for single and double floating-point precision types and a design-space exploration has been performed in terms of performance and hardware resources. The hardware accelerator has been mapped to an FPGA board and connected through PCIe to a typical processor. The performance evaluation shows that the proposed scheme can achieve up to 14x speedup compared to a reference, single-core CPU solution.**

I. Introduction

Systems of ordinary differential equations (ODEs) play an important role in modeling dynamically changing phenomena and evolutionary processes mathematically. Many ODEs of practical interest are nonlinear and, hence, they lack a closed-form solution so that numerical methods are required to compute approximate solutions [1].

High-performance computing (HPC) applications need powerful infrastructures to solve millions of ODEs. Contemporary processors have limited performance in solving these kinds of equations. Specialized accelerators can be employed for speeding up execution. However, most of the hardware accelerators for ODE solvers are fixed and require significant effort to be customized to specific applications.

In [2], a custom FPGA processor has been presented that is used for the efficient solution of physical model ODEs on FPGAs. A single differential equation processing element on a Xilinx Virtex-6 FPGA executes several physiological models faster than real-time while requiring only a few hundred FPGA look-up tables (LUTs). The presented architecture shows that a single differential equation processing element is 5–50 slower than High-Level Synthesis(HLS)-based ODE-solver circuits, however the differential equation processing element is 10–200 smaller in terms of hardware resources. However, the FPGA processor cannot meet the high-throughput requirements that are required by HPC applications.

In this paper, we present a family of high-performance and configurable hardware accelerators that can reduce significantly the execution time of the ODE solvers and can be customized to meet the requirements of several ODE solvers. The accelerators for the ODE solvers have been implemented in reconfigurable logic and have been mapped to an FPGA connected via PCIe with an contemporary Intel processor. Performance evaluation shows that the hardware accelerator can achieve up to 14x speedup compared to a typical desktop processor.

II. ODE Solvers

We will demonstrate our proposed accelerators by solving the Lotka–Volterra equations, also known as the predator-prey equations [3]. These are a pair of first-order, nonlinear, differential equations forming a 2x2 ODE system and are frequently used to describe the dynamics of biological systems in which two species interact, one as a predator and the other as prey. The species populations u and v change through time and their dynamic interplay is given by the following equations:

$$\frac{du}{dt} = f(u, v, t) \tag{1}$$

$$\frac{dv}{dt} = g(u, v, t) \tag{2}$$

Here, the f and g functions are given by:

$$f(u, v, t) = 0.1 \cdot u - 0.2 \cdot u \cdot v \tag{3}$$

$$g(u, v, t) = -0.2 \cdot v + 0.4 \cdot u \cdot v \tag{4}$$

Other 2x2 ODE systems describing variations on this Predator–Prey system, such as the dynamic interaction between two species of bacteria that compete for the same supply of food [4], are expected to have similar performance results.

For the numerical solution of the above model, we adopt four widely-used ODE solvers; namely, *forward Euler* (FwdEuler), *modified Euler* (ModEuler), and *strong stability-preserving Runge-Kutta schemes* of order two (SSP-RK2) and three (SSP-RK3) [5]. All four methods are fully explicit and thus, lend themselves to parallel implementations. The computational complexity increases from the single-step, forward Euler method to the two- and three-step Runge-Kutta methods, so that the speedup of the FPGA implementation over the software solution is expected to be highest for the FwdEuler but still considerable for the ModEuler, the SSP-RK2 and the SSP-RK3. The benefit of the latter is, however, their higher temporal accuracy. Furthermore, the selected ODE solvers are used as time integrators for partial differential equations (PDEs), especially hyperbolic conservation laws, and therefore, the development of efficient FPGA-accelerated solution procedures has the potential to significantly speed-up

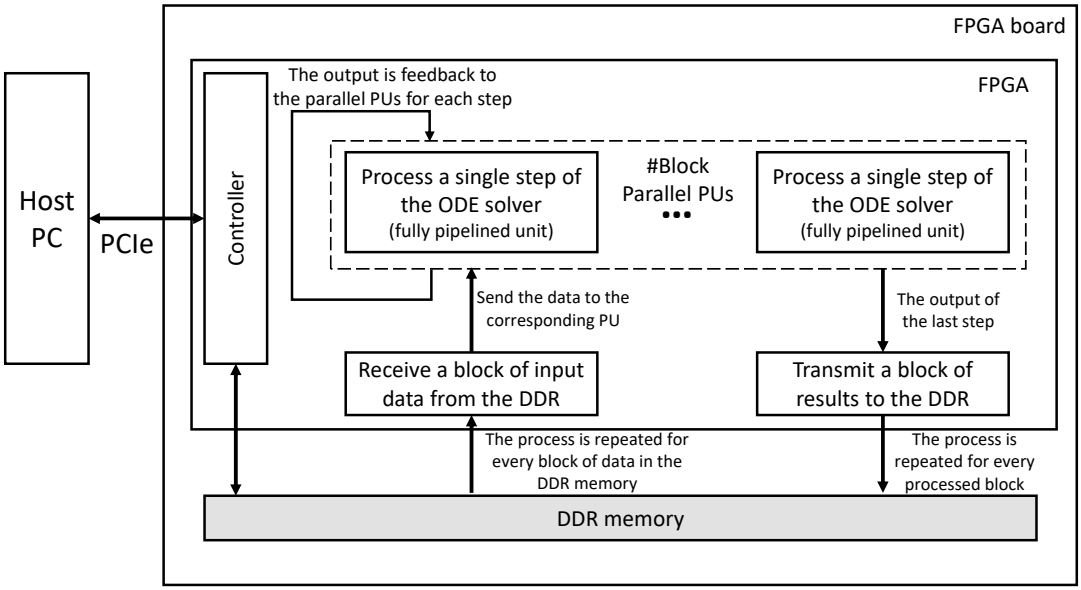

Fig. 1. Architecture of the ODE solvers hosted in a FPGA board.

the numerical simulations of PDE problems as well, at a later stage.

The main contributions of this paper are the following:

- A high-performance accelerator for solving ODEs that can achieve up to 14x speedup;
- A design-space exploration of single and double floating-point architectures; and
- A performance evaluation of 4 different solvers in terms of throughput, speedup and hardware resources.

III. Implementation

A. Kernel

The input variables of the kernel on the FPGA are: step size dt, the number of steps, the initial values for each u and v and the coefficients of the f and g functions. The output variables are the values of each u and v after the number of steps. This leads to an implementation which can be executed for different scenarios.

The selected ODE solvers were implemented using HLS in single- and double-precision floating-point (FP) arithmetic. In the kernel on the FPGA single processing units (PUs) are created which calculate the values of the next step for a pair of u and v, with the use of the f and g functions. To increase the performance multiple units process the data in parallel for multiple pairs of u's and v's. The pairs of u's and v's which can be executed in parallel will be referred to as a *block*. Per block, the elements are stored to the DDR memory of the FPGA board. Because of the limited amount of resources on the FPGA it is possible that not all elements are able to fit into a single block. If this is the case, the elements are divided to multiple blocks. First the final values of the current block are calculated after which the next block is retrieved and processed. This continues until the total number of blocks is reached. If not all the block of elements are able to fit in the DDR memory, they will be replaced with new ones and the whole cycle will repeat until all elements are processed.

For each ODE solver the maximum number of PUs is determined, which is constrained by limited resources on the FPGA. For each of the presented experiments we choose the optimal amount of blocks that fits the amount of elements of the simulation. Additionally, the communication overhead is minimized by transferring at once as much blocks as possible from the Host CPU to the FPGA board.

Figure 1 presents the architecture of the implemented system. Between the four different ODE solvers only the parallel PUs differ. The Host PC continues to send sets of blocks to the DDR memory until the entire amount of the element is processed. The directive used in the Xilinx Vivado HLS development platform to achieve the described architecture is `#pragma HLS PIPELINE` in each processing unit, `#pragma HLS UNROLL` for the loop of the elements at the current block, `# pragma HLS PIPELINE` for the loop of the steps and `#pragma HLS ARRAY PARTITION dim=1` for the memory that stores the input and output data. Additionaly, directives are used for the interface of the top function to be implemented with the Xilinx AXI protocol.

B. Host

The access of the accelerator kernels from the Host CPU can be achieved by using six functions that were developed using C++. Those functions are responsible for creating the CPU-FPGA connection, programming the FPGA board with the correct kernel, initializing the memories required for the communication, sending the element values, receiving the results, breaking the link between the CPU and the FPGA and freeing the reserved CPU resources.

At system startup the initialization function must be executed once. This function creates the device descriptors used for reads/writes to the accelerator and reserves the memory required for the packets that are going to be communicated between the CPU and the FPGA. This process should be avoided to run at each call of the accelerator functions due to the large execution overhead (6-7 seconds). To process data with one of the accelerators (FwdEuler, ModEuler, SSP-RK2 or SSP-RK3) the corresponding function must be called that transmits the data to the kernel and receives the results. During the execution of the accelerator functions a set of blocks is transmitted to the DDR memory of the FPGA board; then the kernel processes each block sequentially. When all the elements of a block are processed the results are transmitted to the DDR memory and a next block is received for process. When all blocks are processed the results are transmitted back to the Host and a new batch of blocks is received. When the system is about to be terminated or the accelerators are no longer needed the terminating function releases the buffers and closes the device descriptors, this process takes around 0.06ms.

The SDAccel tool from Xilinx was used for the integration of the kernels to a final system [6]. The SDAccel tool provides a framework for the development of an entire system consisting of the Host running on a CPU and the kernels running on a FPGA. A PCIe connection is utilized for the communication of the Host and the Kernel.

IV. Results & Evaluation

An exact solution to the problem does not exist, therefore an approximate solution based on the simplest among the considered ODE solvers (FwdEuler) will be used as a reference solution. The parameters of the reference execution are given in Table I, where the step size dt has been selected very small so as not to break the system of non-linear equations. Based on this step size, 10 million solver steps have been run to calculate the solution for 10 simulated seconds. The global accuracy error of the ODE solvers has be calculated as the averaged sum of the differences compared to the reference solution between the u's and v 's at 10 seconds of simulated time.

As a next step, the FwdEuler solver is run again for a larger (but sufficient) step size which leads to a global accuracy error, as shown in Table II, which will be used as the maximum allowable global accuracy error. Of course, now a smaller number of steps is required for simulating 10 seconds of the solution. We repeat this process for each of the other three more advanced solvers (ModEuler, SSP-RK2, SSP-RK3), without exceeding the maximum allowable global accuracy error, their results are also reported in Table II. We can readily see that the more advanced the solver used, the more relaxed the step size dt permitted and, thus, the smaller the number of steps required for achieving a similar accuracy error (at the end of 10 simulated seconds).

The Host CPU used is an Intel Core i5-4590 @ 3.30GHz with 16GB RAM and CentOS 7 x64 operating system. The FPGA board is the Alpha Data ADM-PCIE-KU3 board, featuring a Xilinx Kintex Ultrascale (XCKU060 - FFVA1156) FPGA. The targeted operating frequency for the systems in the FPGA is set at 200 MHz.

TABLE I
Values used to calculate reference solution using the Forward Euler solver.

Number of steps	10^7
dt	10^{-5}
u(t = 0)	0.20
v(t = 0)	1.10

TABLE II
Number of steps, dt, and average global error for each solver.

Solver	**Number of steps** (#)	**dt** (sec)	**Accuracy error (avg.)** (–)
FwdEuler	10,000	0.001	$2.01 \cdot 10^{-5}$
ModEuler	60	0.170	$1.86 \cdot 10^{-5}$
SSP-RK2	60	0.170	$1.86 \cdot 10^{-5}$
SSP-RK3	12	0.830	$1.92 \cdot 10^{-5}$

For each solver the maximum number of parallel processing units and the maximum number of blocks stored in the DDR memory of the FPGA board are determined for both single as double FP precision. Those results together with the resources used on the FPGA are shown in Table III, Table IV, Table V and Table VI for each of the ODE solvers, FwdEuler, ModEuler, SSP-RK2 and SSP-RK3 respectively. The resources are estimations for the kernels, as produced from the SDAccel tool; the PCIe controller was not considered. From those tables we can see the increased resources requirements between the single and double precision implementations and between the ODE solvers. Additionally, for each simulation the amount of blocks stored in the DDR memory can be seen in Table VII for FwdEuler, Table VIII for ModEuler, Table IX for SSP-RK2 and Table X for SSP-RK3. All simulations use the maximum amount of parallel processing units.

As expected, when double precision is used the amount of parallel processing units decreases in comparison to single precision, due to higher FPGA resources requirements, which limit the achievable speedup of the accelerator. The same holds for all the ODE accelerators. Because the second-order and third-order strong-stability-preserving Runge-Kutta ODE solvers require more operations in comparison with the Euler solvers, the usage of FPGA resources for the Runge-Kutta ODE solvers per PU is higher. As a result the amount of parallel PUs decreases when more operations are needed for the ODE solver. Any further increase on the parallel PUs or the number of blocks that are stored to the DDR memory, for each of the ODE solvers, will cause the operating frequency to drop under 200 MHz.

We measure the execution time of each solver from one thousand up to one hundred million elements. For the measure-

TABLE III
FPGA RESOURCES FOR THE FWDEULER SOLVER IN SINGLE AND DOUBLE FP PRECISION.

	FF	LUT	DSP	BRAM (x36Kb)	Parallel PUs	Blocks in DDR
FPGA Single FP	53k	36k	182	2	120	240
FPGA Double FP	72k	46k	372	4	60	140

TABLE IV
FPGA RESOURCES FOR THE MODEULER SOLVER IN SINGLE AND DOUBLE FP PRECISION.

	FF	LUT	DSP	BRAM (x36Kb)	Parallel PUs	Blocks in DDR
FPGA Single FP	54k	38k	187	2	120	170
FPGA Double FP	57k	41k	406	4	40	220

TABLE V
FPGA RESOURCES FOR THE SSP-RK2 SOLVER IN SINGLE AND DOUBLE FP PRECISION.

	FF	LUT	DSP	BRAM (x36Kb)	Parallel PUs	Blocks in DDR
FPGA Single FP	49k	36k	271	2	80	220
FPGA Double FP	80k	54k	594	4	60	167

TABLE VI
FPGA RESOURCES FOR THE SSP-RK3 SOLVER IN SINGLE AND DOUBLE FP PRECISION.

	FF	LUT	DSP	BRAM (x36Kb)	Parallel PUs	Blocks in DDR
FPGA Single FP	57k	41k	370	2	80	200
FPGA Double FP	69k	50k	916	4	1	10^8

TABLE VII
USED BLOCKS (#) IN THE SIMULATIONS FOR THE FWDEULER SOLVER IN SINGLE AND DOUBLE FP PRECISION.

Elements	1k	10k	>10k
Single FP Blocks in DDR	9	84	240
Double FP Blocks in DDR	17	140	140

TABLE VIII
USED BLOCKS (#) IN THE SIMULATIONS FOR THE MODEULER SOLVER IN SINGLE AND DOUBLE FP PRECISION.

Elements	1k	10k	>10k
Single FP Blocks in DDR	9	84	170
Double FP Blocks in DDR	25	140	220

ments the function *gettimeofday* is used. Those measurements are done with both single and double FP precision entirely on the Host CPU and on the FPGA using the accelerators.

TABLE IX
USED BLOCKS (#) IN THE SIMULATIONS FOR THE SSP-RK2 SOLVER IN SINGLE AND DOUBLE FP PRECISION.

Elements	1k	10k	>10k
Single FP Blocks in DDR	13	125	220
Double FP Blocks in DDR	17	167	167

TABLE X
USED BLOCKS (#) IN THE SIMULATIONS FOR THE SSP-RK3 SOLVER IN SINGLE AND DOUBLE FP PRECISION.

Elements	1k	10k	>10k
Single FP Blocks in DDR	13	125	200
Double FP Blocks in DDR	10^3	10^4	# elements

TABLE XI
EXECUTION TIME (MS) FOR THE FWDEULER SOLVER IN SINGLE AND DOUBLE FP PRECISION.

Elements	1k	10k	100k	1m	10m	100m
CPU Single FP	$1.75{\cdot}10^2$	$1.75{\cdot}10^3$	$1.75{\cdot}10^4$	$1.75{\cdot}10^5$	$1.75{\cdot}10^6$	$1.75{\cdot}10^7$
CPU Double FP	$1.51{\cdot}10^2$	$1.51{\cdot}10^3$	$1.51{\cdot}10^4$	$1.51{\cdot}10^5$	$1.51{\cdot}10^6$	$1.51{\cdot}10^7$
FPGA Single FP	14.33	$1.28{\cdot}10^2$	$1.45{\cdot}10^3$	$1.27{\cdot}10^4$	$1.26{\cdot}10^5$	$1.26{\cdot}10^6$
FPGA Double FP	41.80	$6.76{\cdot}10^2$	$4.07{\cdot}10^3$	$4.07{\cdot}10^4$	$4.04{\cdot}10^5$	$4.05{\cdot}10^6$

The results can be found in Table XI, Table XII, Table XIII, and Table XIV for respectively the FwdEuler, the ModEuler, the SSP-RK2, and the SSP-RK3 ODE solvers. Those results show that a higher order solver need less time to execute. This is because a higher order solver need less steps to simulate the same amount of time, while staying below a certain threshold accuracy error. Interestingly, the execution times on the CPU are higher with the single FP implementation than the double FP implementation. This may be explained by the 64 bit architecture of the CPU, which uses for both single and double FP variables the same floating point unit (FPU) for computations. The FPU uses double FP variables, therefore, single FP variables need to be converted to a double FP variables [7]. This creates a penalty for using single FP variables in comparison with using double FP variables. This penalty is likely higher than the faster memory operations.

Figure 2 presents the speedup achieved through hardware acceleration of each ODE solver for the single-precision FP FPGA implementation, with respect to the single-precision FP CPU implementation. As the complexity of the ODE solver increases the achieved speedup decreases because less parallel processing units can fit in the targeted FPGA and/or less blocks can be stored in the DDR memory of the board. Figure 3

TABLE XII
EXECUTION TIME (MS) FOR THE MODEULER SOLVER IN SINGLE AND DOUBLE FP PRECISION.

Elements	1k	10k	100k	1m	10m	100m
CPU Single FP	2.50	24.67	$2.46{\cdot}10^2$	$2.46{\cdot}10^3$	$2.46{\cdot}10^4$	$2.46{\cdot}10^5$
CPU Double FP	2.05	19.84	$1.92{\cdot}10^2$	$1.92{\cdot}10^3$	$1.92{\cdot}10^4$	$1.92{\cdot}10^5$
FPGA Single FP	0.84	3.00	24.59	$2.43{\cdot}10^2$	$2.37{\cdot}10^3$	$2.39{\cdot}10^4$
FPGA Double FP	1.57	10.61	98.29	$9.06{\cdot}10^2$	$9.10{\cdot}10^3$	$9.15{\cdot}10^4$

TABLE XIII
EXECUTION TIME (MS) FOR THE SSP-RK2 SOLVER IN SINGLE AND DOUBLE FP PRECISION.

Elements	1k	10k	100k	1m	10m	100m
CPU Single FP	2.18	22.25	$2.17{\cdot}10^2$	$2.17{\cdot}10^3$	$2.16{\cdot}10^4$	$2.16{\cdot}10^5$
CPU Double FP	1.75	17.89	$1.74{\cdot}10^2$	$1.74{\cdot}10^3$	$1.74{\cdot}10^4$	$1.73{\cdot}10^5$
FPGA Single FP	1.04	3.79	33.7	$3.02\ {\cdot}10^2$	$3.02{\cdot}10^3$	$3.06{\cdot}10^4$
FPGA Double FP	1.33	6.6 3	59.47	$5.86\ {\cdot}10^2$	$5.85{\cdot}10^3$	$5.91{\cdot}10^4$

TABLE XIV
EXECUTION TIME (MS) FOR THE SSP-RK3 SOLVER IN SINGLE AND DOUBLE FP PRECISION.

Elements	1k	10k	100k	1m	10m	100m
CPU Single FP	0.70	6.69	66.87	$6.60{\cdot}10^2$	$6.61{\cdot}10^3$	$6.59{\cdot}10^4$
CPU Double FP	0.52	5.57	51.49	$5.16{\cdot}10^2$	$5.16{\cdot}10^3$	$5.15{\cdot}10^4$
FPGA Single FP	0.82	2.218	18.28	$1.50{\cdot}10^2$	$1.48{*}{\cdot}10^3$	$1.48{\cdot}10^4$
FPGA Double FP	9.73	94.79	$9.41{\cdot}10^2$	$9.48{\cdot}10^3$	$9.50{\cdot}10^4$	$9.49{\cdot}10^5$

presents the speedup achieved for the double-precision FP FPGA implementation, with respect to the double-precision FP CPU implementation. Here, the achieved speedups are significantly lower than the speedup of the single-precision FP implementations, with the SSP-RK3 accelerator requiring more execution time than the software counterpart. Although, for the double-precision SSP-RK3 accelerator the entire set of the elements can be transferred to the DDR memory, the lack of parallelism results in higher execution times. Additionally, the synchronization barrier that exists between the calculation of each step negatively affects the performance of each implementation, because, it does not utilize the fully pipelined PUs.

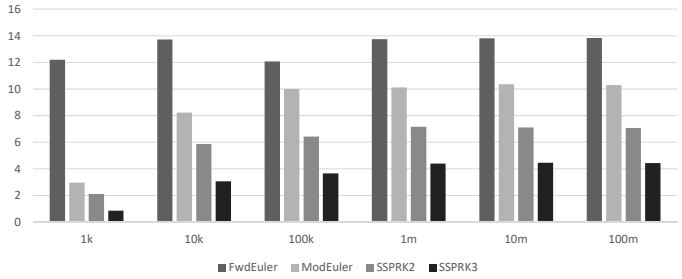

Fig. 2. Achieved speedup of the accelerated ODE solvers for single FP precision.

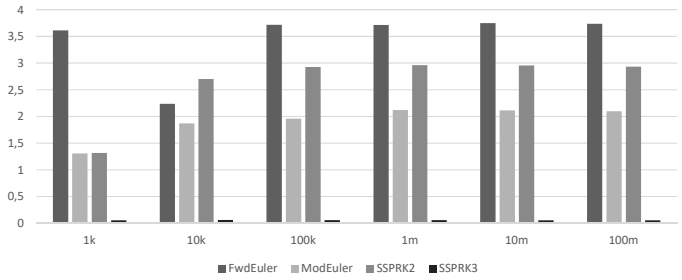

Fig. 3. Achieved speedup of the accelerated ODE solvers for double FP precision.

V. CONCLUSIONS

In this work we have implemented and evaluated the performance of four different ODE solvers, which can be highly parallelized by design. The hardware accelerators that have been implemented can be tuned to meet the application requirements of several ODE solvers. The performance evaluation shows that the proposed hardware accelerators can achieve up to 14x speedup compared to their sequential counterparts when run on contemporary processors, thus reducing the execution time of the predator-prey application significantly. It can be seen that when using more resources of the FPGA with increasing ODE-solver complexity or increasing precision, the achieved speedup decreases because less parallelism can be achieved on the FPGA.

VI. FUTURE WORK

Interesting future work involves pipelining the process of the blocks that are stored to the DDR memory of the FPGA, using as intermediate buffers extra registers of the FPGA. This change in the design of the accelerator, shall lead to the removal of the synchronization barrier, which can further increase the performance of the hardware solvers. Additionally, other applications with larger systems of ODEs can be accelerated with the use of FPGAs.

REFERENCES

[1] J.C. Butcher. *Numerical Methods for Ordinary Differential Equations*. Wiley, 2003.

[2] C. Huang, F. Vahid, and T. Givargis. A custom fpga processor for physical model ordinary differential equation solving. *IEEE Embedded Systems Letters*, 3(4):113–116, Dec 2011.

[3] Fred Brauer, Carlos Castillo-Chavez, and Carlos Castillo-Chavez. *Mathematical models in population biology and epidemiology*, volume 40. Springer, 2001.
[4] James Dickson Murray. *Mathematical biology. I. , An introduction*. Interdisciplinary applied mathematics. Springer, New York, 2002.
[5] Sigal Gottlieb, Chi-Wang Shu, and Eitan Tadmor. Strong stability-preserving high-order time discretization methods. *SIAM Rev*, 43:89–112, 2001.
[6] The Xilinx SDAccel Development Environment: Bringing The Best Performance/Watt to the Data Center. Technical report, 2015.
[7] Intel Corparation. Intel(R) 64 and IA-32 Architectures Software Developer's Manual, Combined Volumes, Dec 2016.

View publication stats