FIRST SYCL IMPLEMENTATION OF THE THREE-DIMENSIONAL SUBSURFACE XCA-FLOW CELLULAR AUTOMATON AND PERFORMANCE COMPARISON AGAINST CUDA

Donato D'Ambrosio[1], Giovanni Terremoto[1], Alessio De Rango[2], Luca Furnari[2], Alfonso Senatore[2] and Giuseppe Mendicino[2]

[1]*Department of Mathematics and Computer Science, University of Calabria, Ponte Pietro Bucci, Cubo 30B, 87036 Rende, Italy*

[2]*Department of Environmental Engineering, University of Calabria, Ponte Pietro Bucci, Cubo 44A, 87036 Rende, Italy*

ABSTRACT

We present the results of a first SYCL vs CUDA performance assessment for the case of the three-dimensional XCA-Flow subsurface Extended Cellular Automata model. A grid domain of **100 x 100 x 50** cubic cells of 0.3 m side was considered, where two different heterogeneous hydraulic conductivity fields were imposed, resulting in different computational loads. For each conductivity field, a 10 days test case simulation with a constant infiltration rate over the central part of the upper domain's interface and no-flow condition at other boundaries was designed as a benchmark for performance assessment. The stencil-based kernels of the XCA-Flow model were implemented by considering the one-thread-one-cell thread-to-cell mapping and global memory accesses. A global reduction, needed by the algorithm at each computational step to find the minimum of a domain's state variable, exploited the device's on-chip local memory (shared memory in the CUDA nomenclature). The CUDA back-end featured SYCL compiler adopted was the Intel DPC++. The experiments were performed on an Nvidia Titan Xp GPU by considering different configurations of SYCL/CUDA thread blocks. Each simulation was re-executed four times by selecting the minimum elapsed time. As expected, the CUDA implementation performed slightly better. Nevertheless, SYCL provided satisfying results, with a limited mean gap of approximately 8%.

KEYWORDS

XCA-Flow 3D Subsurface Model, Extended Cellular Automata, SYCL vs CUDA, Performance Assessment

1. INTRODUCTION

In recent years, new shared memory data-parallel architectures and APIs were proposed. Hardware vendors like Nvidia, AMD, and Intel, offered GPUs (Graphics Processing Units) and many-core computing accelerators that rapidly arose as reference devices in the High-Performance Computing (HPC) field (Owens et al., 2007).

The Nvidia Compute Unified Device Architecture (CUDA), proposed in 2008, rapidly became mainstream (see, e.g., Blecic et al., 2013; Arca et al., 2015; D'Ambrosio et al., 2012; D'Ambrosio et al., 2013 for some examples of application) due to, among others, the higher performance/cost ratio of CUDA devices compared to the alternative CPU-based solutions, and the C-like CUDA single source CPU/GPU heterogeneous programming model that made it easy to port existing codes to the new architecture. The main CUDA drawback is portability, since CUDA applications can only run on Nvidia hardware.

Aiming at overcoming the CUDA portability issue, Khronos Group released the OpenCL (Open Computing Language) specifications in 2009 (Stone et al., 2010) as an open standard. OpenCL offered a programming model similar to CUDA, though targeting heterogeneous devices like CPUs, GPUs, and other accelerators (see, e.g., Du et al., 2012; Su and Keutzer, 2012; Cercos-Pita, 2015; Macri et al., 2015; Senatore et al., 2016; De Rango et al., 2019 for some examples of application). Alternative solutions to CUDA and OpenCL were also proposed like OpenACC (see, e.g., Wienke et al., 2012) and OpenMP (starting from the release 4.0 - see, e.g., De Rango et al., 2018).

One of the most recent and promising shared memory data-parallel technologies is SYCL, a set of specifications by Khronos Group defining a C++ compiler-embedded abstraction layer for data-parallel programming. Modern C++ is supported, and the single-source programming approach permits mixing host and device code in the same translation units. SYCL was initially based on OpenCL, but the last specifications allow for different back-ends, including CUDA (Reyes et al., 2020). The interest in SYCL has become very high and its adoption is growing (see, e.g., Shin et al., 2020; Goli et al., 2020; Christgau and Steinke, 2020; Reguly et al., 2021). Several supercomputers in the Top500 list support SYCL (Thoman et al., 2019).

In this paper, we performed a first comparative analysis to assess the performance of the CUDA back-end featured Intel DPC++ SYCL compiler with respect to the Nvidia nvcc CUDA compiler. To this end, we considered the Extended Cellular Automata class of computational models (von Neumann, 1966; Di Gregorio and Serra, 1999). Specifically, we selected the three-dimensional XCA-Flow subsurface hydrological model, already applied to real and synthetic case studies in previous works (De Rango et al., 2020a; De Rango et al., 2020b; De Rango et al., 2021a; De Rango et al., 2021b; Furnari et al., 2021). This paper considered two 10-day long simulations differing by the heterogeneous hydraulic conductivity parameters to have different infiltration behaviors and computational loads. The simulations are characterized by a 100 × 100 × 50 grid domain with cubic cells of 0.3 m side, a constant infiltration rate over the central part of the upper domain's surface, and no-flow condition at other boundaries. SYCL and CUDA implementations of the XCA-Flow kernels only referred to GPU's global memory, even if a global reduction needed by the algorithm to find the minimum of a domain's state variable also exploited the on-chip local memory (shared memory in the CUDA nomenclature). The simulation experiments were performed on an Nvidia Titan Xp GPU.

The paper is organized as follows: Section 2 provides the theoretical foundations of the subsurface flow XCA-Flow model, together with details regarding computational aspects, while Section 3 describes the case-study simulations; Section 4 briefly characterizes the XCA-Flow kernels and presents the achieved computational results, while Section 5 eventually concludes the paper with a general discussion, by also outlining possible future developments.

2. THE XCA-FLOW HYDROLOGICAL MODEL

The XCA-Flow subsurface hydrological model is based on the direct discrete formulation of the Richards' equation as formalized by Mendicino et al. (2006). The Richards' equation, governing the subsurface water flow process, is a nonlinear degenerate elliptic-parabolic partial differential equation (List and Radu, 2016). It describes double-phases flow in porous media by combining the mass conservation equation with the Darcys' law, representing the momentum conservation equation. Considering the pressure head ψ as the dependent variable, the Richards' equation for an isotropic porous medium is written as (Celia et al., 1990):

$$(1) \quad C_c(\psi) = \frac{\partial \psi}{\partial t} - \nabla[K(\psi)\nabla\psi] - \frac{\partial K(\psi)}{\partial z} = 0$$

where the pressure head ψ [m] is related to the hydraulic head h [m] by the equation $\psi = h - z$, being z [m] the elevation, $C_c(\psi)$ *i*s the specific retention capacity [m^{-1}], given by the relation $C_c(\psi) = \frac{d\theta}{dh}$, where in turns θ is the moisture content [$m^3\ m^{-3}$] and $K(\psi)$ is the hydraulic conductivity [$m\ s^{-1}$].

According to the Cellular Automata computational paradigm, XCA-Flow is a space-time discrete simulation model based on local transition rules of evolution, called kernels or elementary processes. The new cell's state depends on the current state of the cell itself (called central cell) and those of a limited set of adjacent cells. The central cell together with its adjacent cells forms the so-called neighborhood, whose geometrical pattern is referred to as stencil. In XCA-Flow, the three-dimensional von Neumann stencil is adopted, defined below in terms of relative coordinates with respect to the central cell, which belongs to its own neighborhood:

$$X = \{(0,0,0), (-1,0,0), (0,-1,0), (0,1,0), (1,0,0), (0,0,-1), (0,0,1)\}$$

The three computing kernels, $\sigma_\rho, \sigma_\beta$, and σ_τ, together with a global reduction used to assess the physical time corresponding to the XCA-Flow computational step, γ_t, are formally defined below.

- $\sigma_\rho: Q_h \times Q_{C_c} \rightarrow Q_h$ accounts for input rain, where Q_h and Q_{C_c} represent the sets of values for the hydraulic total head and specific retention capacity state variables, respectively. Specifically, if $h, C_c, r_{ir}, \Delta t$ and Δs^2 denote hydraulic head, specific retention capacity, rain intensity rate, time interval corresponding to an XCA-Flow transition step, and surface of the cell side, respectively, σ_ρ updates the hydraulic head within the cell as:

$$h' = h + \frac{r_{ir} \cdot \Delta t}{\Delta s^2 \cdot C_c}$$

- $\sigma_\beta: Q_h \rightarrow Q_h$ sets the boundary condition on the boundary cells that are not affected by rain. The Neumann boundary conditions, which fix the water flow to a constant value, for instance $h' = h$ represent a no flow condition; the Dirichlet boundary conditions, which fix the hydraulic head to a constant value, can be adopted.
- $\sigma_\tau: Q^{|X|} \rightarrow Q$, where Q and $|X|$ represent the set of all possible cell's states and the number of neighboring cells, respectively, corresponds to the discrete time explicit resolution of Eq. 1 and can be written as:

$$h'_c = h_c + \frac{\Delta t\left[\sum_{\alpha=1}^{6} \overline{K}_\alpha (h_\alpha - h_c)\ \right]}{\Delta s^2 C_c}$$

Here, $\overline{K}_\alpha$ is the average hydraulic conductivity between the current cell c and the cell in the α neighbor calculated using:

$$\overline{K}_\alpha = \frac{2\Delta s^3}{\frac{\Delta s^3}{K_\alpha} + \frac{\Delta s^3}{K_c}}$$

Also, the q_θ, q_K and q_{c_c} state variables are updated according to the constitutive equations between ψ, θ and K, as proposed by Mualem (1976) and Van Genuchten (1978).
Finally, the state variable $q_{conv} \in Q_{conv}$, is updated adopting the Courant-Friedrichs-Lewy condition to achieve numerical convergence:

$$\Delta t \leq \frac{\Delta s^2 C_c}{\sum_{\alpha=1}^{6} \overline{K}_\alpha}$$

- $\gamma_t : Q_{conv}^{|D|} \rightarrow \mathbb{R}$ evaluates a reduction over the Q_{conv} substate in order to evaluate the physical time corresponding to a state transition of the automaton. Specifically, if Δt denotes the time step size, we have:

$$\Delta t = \min_{\iota \in D} q_{conv_\iota}$$

The complete XCA-Flow model's formalization, here omitted for the sake of brevity, can be found in Furnari et al. (2021) - Appendix A.

3. THE TEST CASE SIMULATIONS

The XCA-Flow model was applied to three-dimensional heterogeneous test cases. Two different pseudo-random synthetic fields of saturated hydraulic conductivity, K_s, were generated through the application of simple kriging techniques with an exponential semivariogram model (Pebesma, 2004), as already done in Furnari et al. (2021). The field generation method required the assignment of both mean μ and variance σ^2 values of the K_s field. Due to the wide range of variability of such parameter, the logarithmic transform of the variable was used. A value of μ equal to -4.19 and σ^2 equal to 2.0 were imposed, used for generating the two K_s fields shown in Figure 1. Choosing σ^2 = 2.0, imposed a large variability in the hydrological soil properties.

Table 1. Hydrological parameters in the test case. θ_r is the residual moisture content, θ_s is the saturated moisture content, α and n are soil parameters used in the Van Genuchten model, ψ_0 is the continuity parameter to ensure the passage between unsaturated and saturated conditions (Paniconi et al., 1991), S_s is the specific storage

θ_r	θ_s	$\alpha(m^{-1})$	n	$\psi_0\ (m)$	$S_s(m^{-1})$
0.095	0.348	3.473	1.729	0.001	10^{-6}

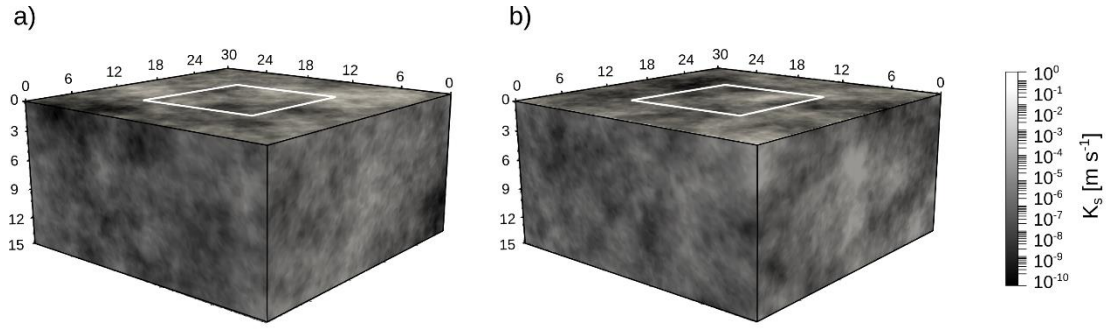

Figure 1. The three-dimensional computational domains used in a) test case simulation Sim_1, b) test case simulation Sim_2. Colors represent the saturated hydraulic conductivity K_s. The surface delimited by the white square on the top part of the domain is affected by infiltration of 0.02 md^{-1}. The domain extends 30 meters long and wide, with a depth of 15 meters, resulting in a grid of 100 × 100 × 50 with a cubic cells side of 0.3 m

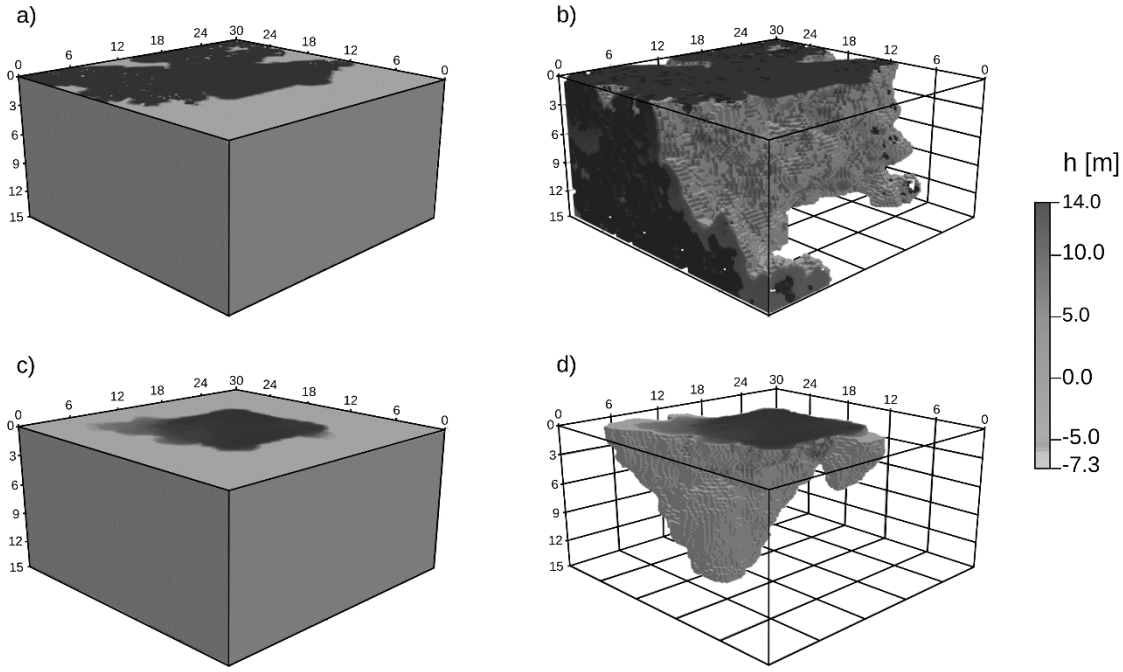

Figure 2. Three-dimensional simulation test cases: overview of the computational domains and wet front propagation. Colours represent hydraulic head values at the end of the simulations. a), b) refer to the test case simulation Sim_1; c), d), refer to the test case simulation Sim_2. In b) and d), hydraulic head values less than or equal to the initial total hydraulic head, namely -7.34 m, are not displayed. In particular, b) and d) highlight the effective wet front propagation

The space domain was composed of a parallelepiped 30 meters long and wide, with a depth of 15 meters. The mesh size was fixed to 0.3 m, generating 100 × 100 × 50 grid points. Neumann's no-flow conditions were set for all boundaries, except a constant infiltration rate of 0.02 md^{-1} interesting the central part of the upper border of the domain. Specifically, the infiltration affected a 12×12 m^2 square area located in the centre of the upper face of the domain, as shown in Figure 1. The experiments simulated a physical time of 10 days, with a constant initial total hydraulic head of -7.34 m. The resulting simulation is shown in Figure 2. The hydrological parameters are reported in Table 1.

4. KERNELS IMPLEMENTATION AND COMPUTATIONAL RESULTS

The XCA-Flow model is based on the three computational kernels and the global reduction described in Section 2., which are executed at each iteration of the simulation loop. In this first work, we considered a basic implementation of the computational kernels, with global memory accesses, and monolithic (one-thread/one-cell) thread-to-cell mapping. Conversely, the reduction kernel was implemented by applying a classical sequential addressing parallel reduction algorithm, whit local memory (shared memory in the CUDA nomenclature) usage (Kirk and Hwu, 2017).

As stated, we exploited an Nvidia Titan XP GPU for the experiments, which belongs to the Pascal architecture. The GPU is connected through the PCI express channel to a scientific workstation running Arch Linux. The adopted SYCL compiler was the Intel DPC++. In particular, we used the sycl llvm fork from Intel at https://github.com/intel/llvm, commit hash 68b089f. Nvcc version 11.6 was used to compile the CUDA code.

Regarding the experiments, we ran each test case simulation (see Figure 2) on eight different block configurations, ranging from 8x8x1 to 16x16x2 threads. Specifically, we repeated each experiment four times, by registering the fastest execution. The computational results, expressed in terms of simulation elapsed time, are shown in Figure 3 and Figure 4 for the test case simulations Sim_1 and Sim_2, respectively.

As expected, CUDA performed slightly better than SYCL for both the $Sim_1$and Sim_2 simulations. In the case of Sim_1, the best SYCL result was obtained in correspondence with the 16x16x2 block configuration, with a gap of 5.57% with respect to CUDA. On average, the SYCL gap was 8.2%, with a variance of 2.89%. Similar results were achieved for Sim_2. The best SYCL result was obtained in correspondence with the 16x16x2 block configuration, with a gap of 4.96%. On average, the SYCL gap was 7.63%, with a variance of 2.62%. Results are summarized in Table 2.

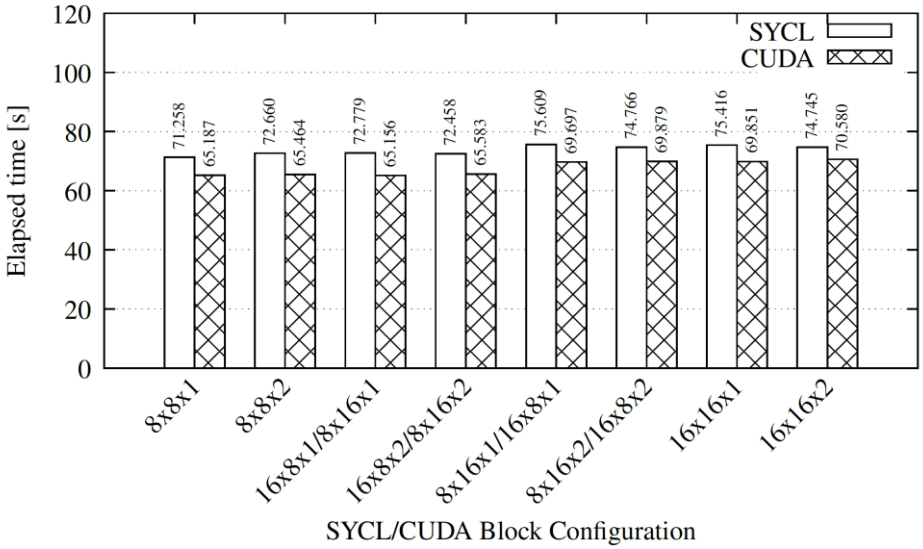

Figure 3. SYCL/CUDA elapsed times of the test case simulation Sim_1 (i.e., the one considering the K_s field in Figure 1a) executed on the Nvidia Titan Xp GPU with different thread block configurations. The exact elapsed time is reported on top of each bar. Statistics are reported in Table 2

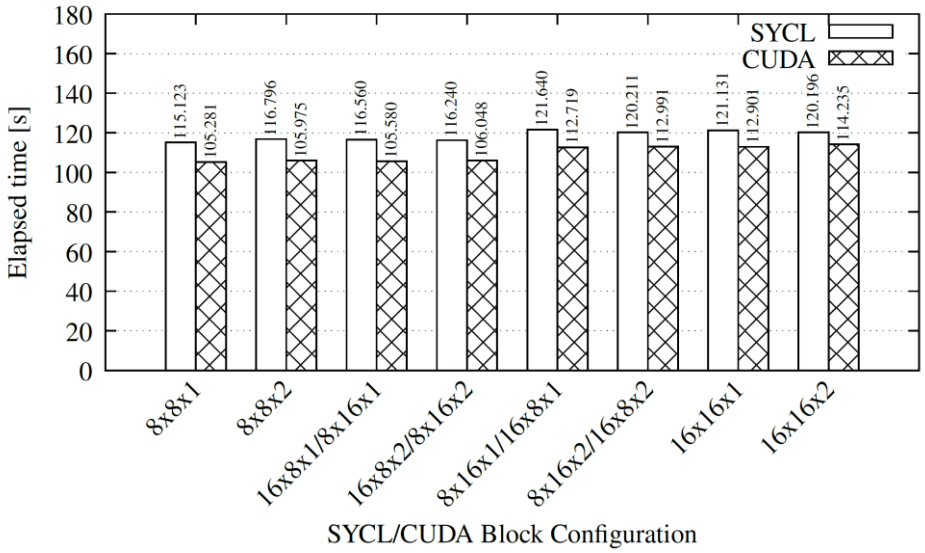

Figure 4. SYCL/CUDA elapsed times of the test case simulation Sim_2 (i.e., the one considering the K_s field in Figure 1b) executed on the Nvidia Titan Xp GPU with different thread block configurations. The exact elapsed time is reported on top of each bar. Statistics are reported in Table 2

Table 2. Comparison between SYCL and CUDA elapsed times of the two test case simulations expressed in terms of percentage gaps. Minimum, average, maximum and variance values are reported

Simulation	Minimum gap [%]	Average gap [%]	Maximum gap [%]	Variance gap [%]
Sim_1	5.57	8.20	10.47	2.89
Sim_2	4.96	7.63	9.42	2.62

5. CONCLUSION

In this work, we presented a first SYCL implementation of the XCA-Flow three-dimensional subsurface Cellular Automata model by assessing its computational performance on an Nvidia Titan Xp GPU compared to an equivalent CUDA implementation of the same model. The SYCL compiler used is the CUDA back-end featured Intel DPC++.

We considered two test case simulations characterized by different hydraulic conductivity fields, resulting in different computational loads for the GPU. We tested different thread block configurations by running each experiment four times and taking the minimum elapsed time. As expected, the CUDA implementation performed slightly better, with SYCL limiting the gap to approximately 8%. The minimum gap registered was approximately 5%, with a variance of less than 3%.

Considering that the CUDA back-end of DPC++ is still experimental, the achieved results can be regarded as satisfying. Nevertheless, better performance is expected with the successive revisions of the compiler.

Future developments will consider the XCA-Flow kernels characterization regarding operational intensity, which is needed to perform a comprehensive Roofline analysis (Williams et al., 2009; 38. Yang et al., 2020). This preliminary analysis will give us information about the kernels that could take advantage of local memory (shared memory in the CUDA nomenclature) tiled implementation. Therefore, a tiled implementation of these kernels will be developed. As a further development, a multi-GPU version of the XCA-Flow simulation model, with applications to extended domains, will be designed and implemented. In this phase, a load balancing algorithm (e.g., Giordano et al., 2021, Giordano et al., 2020) will be adapted and applied to the new multi-GPU

context. All the above-cited developments will be implemented in SYCL and CUDA for comparison. Eventually, distributed MPI/SYCL and MPI/CUDA versions of XCA-flow will be developed. Even in this case, a load balancing algorithm will be adopted to distribute the simulation load to the involved computing resources evenly. The software developed in this study is available on request by writing to the first author.

REFERENCES

Arca, B., Ghisu, T., Trunfio, G., (2015). GPU-accelerated multi-objective optimization of fuel treatments for mitigating wildfire hazard. *In Journal of Computational Science*, Vol. 11, pp 258-268.

Blecic, I., Cecchini, A., Trunfio, G., (2013). Cellular automata simulation of urban dynamics through GPGPU. *In Journal of Supercomputing*, Vol. 65, No. 2, pp 614-629.

Celia, M.A., Bouloutas, E.T., Zarba, R.L., (1990). A general mass-conservative numerical solution for the unsaturated flow equation. *In Water Resources Research*, Vol. 26, No. 7, pp 1483-1496.

Cercos-Pita, J., (2015). AQUAgpusph, a new free 3D SPH solver accelerated with OpenCL. *In Computer Physics Communications*, Vol. 192, pp 295-312.

Christgau, S. and Steinke, T., (2020). Porting a Legacy CUDA Stencil Code to oneAPI. *International Parallel and Distributed Processing Symposium Workshops (IPDPSW).* New Orleans, USA., pp. 359-367.

D'Ambrosio, D., Filippone, G., Marocco, D., Rongo, R., Spataro, W., (2013). Efficient application of GPGPU for lava flow hazard mapping. *In Journal of Supercomputing*, Vol. 65, No. 2, pp 630-644.

D'Ambrosio, D., Filippone, G., Rongo, R., Spataro, W., Trunfio, G., (2012). Cellular Automata and GPGPU: An Application to Lava Flow Modeling. *In International Journal of Grid and High Performance Computing*, Vol. 4, No 3, pp 30-47.

De Rango, A., Furnari, L., Giordano, A., Senatore, A., D'Ambrosio, D., Spataro, W., Straface, S., Mendicino, G., (2021). OpenCAL system extension and application to the three-dimensional Richards equation for unsaturated flow. *In Computers and Mathematics with Applications*, Vol. 81, pp 133-158.

De Rango, A., Furnari, L., Senatore, A., D'Ambrosio, D., Straface, S., Mendicino, G., (2021). Massive simulations on GPGPUs of subsurface flow on heterogeneous soils. *29th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP),* Valladolid, Spain, pp. 249-252.

De Rango, A., Napoli, P., D'Ambrosio, D., Spataro, W., Di Renzo, A., Di Maio, F., (2018). Structured Grid-Based Parallel Simulation of a Simple DEM Model on Heterogeneous Systems. *26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP),* Cambridge, UK, pp. 588-595.

De Rango, A., Furnari, L., D'Ambrosio, D., Senatore, A., Straface, S., Mendicino, G., (2020). The quantization algorithm impact in hydrological applications: Preliminary results. *20th International Conference on Computational Science, ICCS 2020*, Amsterdam, The Netherlands, pp. 191-204.

De Rango, A., Furnari, L., Giordano, A., Senatore, A., D'Ambrosio, D., Straface, S., Mendicino, G., (2020). Preliminary model of saturated flow using cellular automata. *Numerical Computations: Theory and Algorithms*, Le Castella Village, Italy, pp. 256-268.

De Rango, A., Spataro, D., Spataro, W., D'Ambrosio, D., (2019). A first multi-GPU/multi-node implementation of the open computing abstraction layer. *In Journal of Computational Science*, Vol. 32, pp. 115-124.

Di Gregorio, S., Serra, R., (1999). An empirical method for modelling and simulating some complex macroscopic phenomena by cellular automata. *In Future Generation Computer Systems*, Vol. 16, pp. 259-271.

Du, P., Weber, R., Luszczek, P., Tomov, S., Peterson, G., Dongarra, J. (2012) From CUDA to OpenCL: Towards a performance-portable solution for multi-platform GPU programming. *In Parallel Computing*, Vol. 38, No. 8, pp. 391-407.

Furnari, L., Senatore, A., De Rango, A., De Biase, M., Straface, S., Mendicino, G., (2021). Asynchronous cellular automata subsurface flow simulations in two and three-dimensional heterogeneous soils. *In Advances in Water Resources*, Vol. 153, 103952.

Giordano, A., De Rango, A., Rongo, R., D'Ambrosio, D., Spataro, W., (2021). Dynamic Load Balancing in Parallel Execution of Cellular Automata. *In IEEE Transactions on Parallel and Distributed Systems*, Vol. 32, No. 2, pp. 470-484.

Giordano, A., De Rango, A., Rongo, R., D'Ambrosio, D., Spataro, W., (2020). A Dynamic Load Balancing Technique for Parallel Execution of Structured Grid Models. *Numerical Computations: Theory and Algorithms*, Le Castella Village, Italy, pp. 278-290.

Goli, M., Narasimhan, K., Reyes, R., Tracy, B., Soutar, D., Georgiev, S., Fomenko, E., Chereshnev, E., (2020). Towards Cross-Platform Performance Portability of DNN Models using SYCL. *IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC)*, GA, USA, pp. 25-35.

Kirk, D.B., mei W. Hwu, W., (2017). Chapter 5 - Performance considerations. *Programming Massively Parallel Processors (Third Edition)*. Morgan Kaufmann Publishers, San Francisco, USA.

List, F., Radu, F.A., (2016). A study on iterative methods for solving Richards' equation. *In Computational Geosciences* Vol. 20, No. 2, pp. 341-353.

Macri, M., De Rango, A., Spataro, D., D'Ambrosio, D., Spataro, W., (2015). Efficient Lava Flows Simulations with OpenCL: A Preliminary Application for Civil Defence Purposes. *Proceedings of the 10th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, 3PGCIC*. Krakow, Poland, pp. 328-335.

Mendicino, G., Senatore, A., Spezzano, G., Straface, S., (2006). Three-dimensional unsaturated flow modeling using cellular automata. *In Water Resources Research*, Vol. 42, No. 11, pp. 2332-2335.

Mualem, Y., (1976). A new model for predicting the hydraulic conductivity of unsaturated porous media. *In Water Resources Research*, Vol. 12, No. 3, pp. 513-522.

von Neumann, J., (1966). *Theory of Self-Reproducing Automata*. University of Illinois Press, Champaign, IL, USA.

Owens, J., Luebke, D., Govindaraju, N., Harris, M., Kruger, J., Lefohn, A., Purcell, T., (2007). A survey of general-purpose computation on graphics hardware. *In Computer Graphics Forum*, Vol. 26, No. 1, pp. 80-113.

Paniconi, C., Aldama, A.A., Wood, E.F., (1991). Numerical evaluation of iterative and noniterative methods for the solution of the nonlinear Richards equation. *In Water Resources Research*, Vol. 27, No. 6, pp. 1147-1163.

Pebesma, E.J., (2004). Multivariable geostatistics in S: the gstat package. *Computers & Geosciences* 30(7), pp. 683-691.

Reguly, I., Owenson, A., Powell, A., Jarvis, S., Mudalige, G., (2021). Under the Hood of SYCL – An Initial Performance Analysis with An Unstructured-Mesh CFD Application. *36th International Conference, ISC High Performance 2021*, Virtual Event.

Reyes, R., Brown, G., Burns, R., Wong, M., (2020). Sycl 2020: More than meets the eye. *Proceedings of the International Workshop on OpenCL*, Association for Computing Machinery, New York, NY, USA.

Senatore, A., D'Ambrosio, D., De Rango, A., Rongo, R., Spataro, W., Straface, S., Mendicino, G., (2016). Accelerating a three-dimensional eco-hydrological cellular automaton on GPGPU with OpenCL. *AIP Conference Proceedings*, Vol. 1776, 080003.

Shin, W., Yoo, K.H., Baek, N., (2020). Large-Scale Data Computing Performance Comparisons on SYCL Heterogeneous Parallel Processing Layer Implementations. *In Applied Sciences* (Switzerland), Vol. 10, No. 5, 1656.

Stone, J., Gohara, D., Shi, G., (2010). OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems. *In Computing in Science and Engineering*, Vol. 12 No. 3, pp. 66-72.

Su, B.Y., Keutzer, K., (2012). clSpMV: A Cross-Platform OpenCL SpMV Framework on GPUs. *Proceedings of the 26th ACM international conference on Supercomputing*. Venice, Italy, pp. 353-364.

Thoman, P., Salzmann P., Cosenza, B., Fahringer T., (2019). Celerity: High-Level C++ for Accelerator Clusters. *Euro-Par 2019: Parallel Processing*, Göttingen, Germany, pp. 291-303.

Van Genuchten, M.T., (1978). Calculating the unsaturated hydraulic conductivity with a new closed-form analytical model. Researh Reprot - Water Resour. Program, Dep. of Civ. Eng., Princeton Univ., Princeton, NJ.

Wienke, S., Springer, P., Terboven, C., an Mey, D., (2012). OpenACC — First Experiences with Real-World Applications. *Euro-Par 2012 Parallel Processing*, Göttingen, Germany, pp. 859-870.

Williams, S., Waterman, A., Patterson, D., (2009). Roofline: An insightful visual performance model for multicore architectures. *Communications of the ACM*, Vol. 52, No. 4, pp. 65-76.

Yang, C., Kurth, T., Williams, S., (2020). Hierarchical Roofline analysis for GPUs: Accelerating performance optimization for the NERSC-9 Perlmutter system. *Concurrency and Computation: Practice and Experience*, Vol. 32, No. 20, e5547.