Received: 29 June 2020 | Accepted: 12 November 2020

DOI: 10.1002/pamm.202000114

GPU-accelerated simulation of a non-local conservation law

Simone Göttlich[1,*], **Jann Müller**[2,**], and **Jennifer Weissen**[1,***]

[1] University of Mannheim, 68131 Mannheim, Germany
[2] Avatar Engines; B6, 23; 68159 Mannheim; Germany

The model under consideration is a non-local conservation law in two space dimensions that might be used to model material or pedestrian flow. We accelerate the simulation of the non-local model using parallelization of the non-local terms in the programming language Haskell. The implementation allows for the computation of the solution to the conservation law with a finite volume method at reasonable computational cost even for large domains and fine discretizations.

1 Material flow model

A possible application for a non-local conservation law might be material flow on a conveyor belt [1]. The model is of non-local nature, i.e. the movement of material is determined by the conditions in the neighbourhood of its position. Given initial conditions ρ_0, the evolution of the density $\rho = \rho(x,t), x \in \mathbb{R}^2$ is given by

$$\partial_t \rho + \nabla \cdot (\rho(v^{dyn}(\rho) + v^{stat}(x)) = 0, \qquad \rho(x,0) = \rho_0(x). \tag{1}$$

The velocity component v^{stat} describes a time independent velocity field where the geometry of the belt, i.e. obstacles and walls which influence the material movement, are included. The dynamic velocity field v^{dyn} depends on the non-local density conditions in the sorrounding and is modelled as follows

$$v^{dyn}(\rho) = H(\rho - \rho_{max}) \cdot I(\rho), \qquad I(\rho) = -\epsilon \frac{\nabla(\eta * \rho)}{\sqrt{1 + \|\nabla(\eta * \rho)\|}}. \tag{2}$$

At a position x, the dynamic velocity $v^{stat}(x)$ is determined by averaging the density conditions with the spatial convolution $\eta * \rho$ of the kernel η with the density $\rho(\cdot, t)$. The collision operator $I(\rho)$ represents repulsion and is activated whenever the local density exceeds the maximum density ρ_{max} and leads to movement of the material from congested regions into the direction of decreasing material gradient. The parameter $\epsilon > 0$ weights the collision impact and $H(u) = \mathbb{1}_{(u>0)}$ denotes the Heaviside function.

1.1 Numerical treatment

To compute the solution to the conservation law (1), we use a finite volume scheme based on a modified Roe flux [1]. While its approximate solutions converge to the solution of the conservation law, the applied Roe scheme with dimensional splitting is known to give less diffusive solutions than other schemes [2]. We discretize the domain $\Omega \subset \mathbb{R}^2$ in rectangular cells with step sizes $\Delta x_1, \Delta x_2$ and discretize the time with step size Δt. Let $x_{1,i} = i\Delta x_1,\ i = 1, \dots N_{x_1}, x_{2,j} = j\Delta x_2,\ j = 1, \dots N_{x_2}, t_k = k\Delta t,\ k = 1, \dots N_t$. We compute the piecewise constant solution

$$\rho(x,t) = \rho^k_{i,j}\,(x,t) \in [x_{1,i-1/2}, x_{1,i+1/2}) \times [x_{2,j-1/2}, x_{2,j+1/2}) \times [t_k, t_{k+1}), \tag{3}$$

where we use $x_{l,i\pm1/2} = (i \pm \frac{1}{2})\Delta x_l, l = 1, 2$. For details on the computation to obtain the discretized solution (3), we refer to the algorithm `macro_solver` [1]. In each iteration k, the collision operator $I(\rho^k)$ has to be evaluated to compute the solution $\rho(\cdot, t^{k+1})$, see routine `compute_velocityfield` [1]. We remark that the computation of the gradient of the convolution $\nabla(\eta * \rho^k)$ is extremely expensive when the domain is large and/or the discretization is fine.

2 GPU-accelerated implementation

We implement the Roe scheme in the programming language Haskell and use the `accelerate` library, a mini-language embedded in Haskell [3], that enables on-the-fly compilation of programs for GPU and CPU backends via the LLVM compiler framework. With `accelerate` it is possible to benefit from the expressive type system of the Haskell language, the automated memory management and the parallelisation across multiple CPU or GPU threads. The Roe scheme is implemented in

* Corresponding author: e-mail goettlich@uni-mannheim.de, phone +49 621 181-2540
** jann.mueller@avatar-engines.com
*** jennifer.weissen@uni-mannheim.de

Table 1: Time to compute one step (ms) - with and without GPU acceleration.

Grid size $(N_{x_1} \cdot N_{x_2})$	F4s_v2[1] (CPU)	F8s_v2[2] (CPU)	NC12[3] (CPU)	NC6[4] (GPU)	NC12[5] (GPU)
2200	8.55	7.24	5.12	- [6]	2.61
8800	27.15	18.02	14.79	2.90	2.61
35200	96.06	54.53	27.92	5.05	2.55
97778	258.50	146.70	68.66	13.33	2.66
220000	576.30	296.20	142.00	31.54	2.96
880000	2241.00	1150.00	538.00	120.9	10.44

[1] 4 vCPUs 3.4Ghz, 8GB RAM [2] 8 vCPUs 3.4 Ghz, 16GB RAM
[3] 12 vCPUs 2.6Ghz, 224GB RAM
[4] 1 NVidia Tesla K80 GPU, 56GB RAM, 12GB GPU memory
[5] 2 NVidia Tesla V100 GPUs, 224 GB RAM, 32GB GPU memory
[6] not available

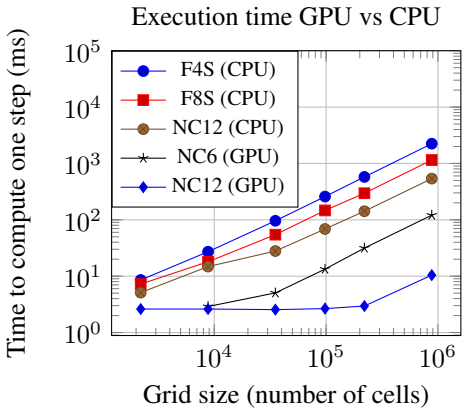

Fig. 1: Visualisation of execution times

the Haskell environment to allow for parallel computation of the convolution $\nabla(\eta * \rho)$. With matrices η and ρ stored in arrays. Since we are interested in large grid sizes $N_{x_1} \cdot N_{x_2}$, the resulting matrices become very large, cf. Table 1. For these array sizes, the convolution is not predefined in Haskell. Therefore, we propose the following: The convolution of two matrices $A \in \mathbb{R}^{a_1 \times a_2}$, $B \in \mathbb{R}^{b_1 \times b_2}$ is defined as $(A * B)_{i,j} = \sum_{p,q} A_{p,q} \cdot B_{i-p,j-q}$ which links the entries with indices i, j in the target array to entries in the source arrays with valid indices p, q and $i - p, j - q$. In our implementation, we first compute the cross product of the input arrays A and B, a four-dimensional array $Z(p, q, r, s) = A(p, q) \cdot B(r, s)$. We use the accelerate function `generate` to fill each cell (p, q, r, s) and store the result to the array `productArray` of size `Pshape` = (a_1, a_2, b_1, b_2). To compute the convolution, we apply an index transformation function $f : (p, q, r, s) \to (p, q)$ and map each cell from the four-dimensional source array to a cell of the two-dimensional target array. If multiple source cells are mapped to the same target cell, their values are added. To compute the summation, we use the accelerate function `permute`. A sketch of the implementation is given below

```
-- Array with cross product
productArray =
  let item (p, q, r, s) = kernel ! (p, q) * density ! (r, s)
  in A.generate PShape item

-- Index transformation function
f :: (Int, Int, Int, Int) -> (Int, Int)
f (p, q, r, s) = (r + p - 1, s + q - 1)

-- Array of the target shape filled with zeros
initialTarget :: Acc (Array DIM2 Double)

-- Array containing the convolution
result = A.permute (+) initialTarget f productArray
```

`accelerate` compiles the array code for two target architectures, CPU (host) and GPU (client). Compilation happens at runtime of the Haskell program. `accelerate` generates bytecode for the target architecture. A scheduler moves data back and forth between client and host, allocating and freeing memory as required, see [3] for details. Table 1 and Figure 1 show the execution time to compute the density $\rho(\cdot, t^k)$ from $\rho(\cdot, t^{k-1})$ with one iteration of the Roe scheme on different sized virtual machines in the Microsoft Azure cloud, for a number of grid sizes and illustrate the speed up using GPU architectures.

Acknowledgements The authors are grateful for the support of the German Research Foundation (DFG) within the project “OptiPlant”, GO 1920/7-1.
Open access funding enabled and organized by Projekt DEAL.

References

[1] S. Göttlich, S. Hoher, P. Schindler, V. Schleper and A. Verl, Modeling, simulation and validation of material flow on conveyor belts, Applied Mathematical Modelling **38**, pp. 3295-3313 (2014).
[2] E. Rossi, J. Weissen, P. Goatin and S. Göttlich, Well-posedness of a non-local model for material flow on conveyor belts, ESAIM:M2AN **54**, pp. 679-704 (2020).
[3] M.M.T. Chakravarty, G. Keller, S. Lee, T.L. McDonell and V. Grover, Proceedings of the sixth workshop on declarative aspects of multicore programming, Austin, Texas, United States, Accelerating Haskell array codes with multicore GPUs (ACM, 2011), pp. 3-14

