Dynamic Domain Expansion in Smoke Spread Simulations with ARTSS: Speedup and Overhead

My Linh Würzburger[a*], Lukas Arnold[a,b]

[a]Institute for Advanced Simulation, Forschungszentrum Jülich, Jülich, Germany
[b]Computational Civil Engineering, University of Wuppertal, Germany

[*]Corresponding author (m.wuerzburger@fz-juelich.de)

Highlights:

- Real-time smoke spread simulation
- Dynamic domain expansion in CFD simulations
- Analysis of domain expansion overhead
- Speedup of computation on GPUs and multicore

Abstract:

This paper describes the impact and consequences of a dynamic domain expansion in a smoke simulation performed in the software ARTSS. This software is developed with the aim to conduct real-time or even prognosis computations by using GPUs as the main computational architecture. Further runtime acceleration is proposed by means of a dynamic extension of the computational domain. This approach is based on the reduction of the computational domain, which is dynamically adopted to calculate only the domain of interest, e.g. regions containing smoke. Here, the domain starts as a localised region and is extended based on prescribed criteria. This contribution outlines the initial implementation. However, to understand the impact of an expansion, the overhead caused by the expansion process, the influence on the numerical result and on the runtime, as well as the used expansion parameters, are investigated. In general, an increased acceleration can be eventually observed at the costs of accuracy due to the reduced domain. The overhead and accuracy can be controlled by the method's parameters. The loss of accuracy depends strongly on which expansion methods and setting are used. With more complex expansion methods, the loss of accuracy can be reduced.

Keywords: smoke; modeling; CFD; fluid dynamics; dynamic computational domain; GPU computation

1. Introduction

The numerical computation of smoke spread in compartment fires has become a common approach in fire safety engineering. With new computer technologies the computational power allows new application fields for the simulation models. Eventually, real-time

applications in evacuation modelling are already achievable [1] and envisioned in fluid dynamics in the context of buildings [2]. The aim of the software ARTSS (accelerator-based real time smoke simulator), former name JuROr [3], is to contribute to this new field with a new approach for a fast computational fluid dynamics (CFD) code to be used for fire, especially smoke, dynamics.

As stated above, the purpose of ARTSS is to create a real-time and eventually even prognosis simulation of smoke propagation in complex buildings. The approaches to achieve this goal are to adopt the modelling and numerical algorithms as well as the implementation to use highly parallel systems, like graphics processing units (GPU). Presently, none of the common simulation tools has the flexibility and performance portability to run efficiently on GPUs. In the course of the development, simplifications w.r.t. the numerical and physical modelling are conducted in order to provide an efficient execution. Nevertheless, there are application cases which do not demand a high level of accuracy. One application is a quick scan of parameter spaces during the design of fire protection measures or the assessment of life safety. The resulting scenarios of interest can thereafter be evaluated with enhanded models, like the commercial software ANSYS CFX and open source software like FDS (Fire Dynamic Simulator) [4] amongst others. Another one, assuming a real-time capability, is to provide a simulation core in support decision systems, which can be used by firefighters to adopt their strategies in complex building structures or to steer technical systems such as smoke extraction systems or adaptive escape routes. These goals require a coupling to sensor data and a data assimilation methodology, which may be based on existing approaches [5]. So far, these approaches have only be applied to zone models, due to their short execution times. Finally, the inclusion in serious games to create a dynamic smoke behaviour in virtual fire fighting training systems [6].

In order to accelerate the execution of ARTSS even further, the approach of a dynamic domain adaptation is developed. In this paper only the expansion is considered for a case where the smoke can spread undisturbed. The general idea of dynamic domain expansion is based on a flexible computational domain. Adoption of its size will directly impact the execution time, as only the selected regions will be considered by the solver. During the computation the adoption can be conducted dynamically to react on the current simulation state. The initial domain would include only the initial locations around the heat and smoke source and be extended as the smoke spreads. This way only regions which are affected by the smoke, e.g. given by the smoke concentration, are computed. In general, the extension strategy can be room or volume oriented. The first one would dynamically add adjacent rooms, while the latter one extends the computed domain in a room. Both strategies introduce new boundary conditions, here open boundary conditions, at the virtual boundaries of the dynamic domain. This is the main reason for a loss of modelling accuracy.

The following section 2 gives an overview of ARTSS and the implementation of the proposed dynamical domain extension. This is followed by an example application (section 3) of the smoke spread in a tunnel geometry. Based on this example case, the

impact of the extension model's parameters on the runtime is analysed in section 4. Finally, the results are concluded in section 5.

2. Concept and implementation in ARTSS

2.1 Overview of ARTSS

ARTSS calculates numerically the smoke and heat propagation in a given building structure. In favour of acceleration, models for thermal radiation, pyrolysis, combustion etc. are neglected. The open-source software uses a set of solvers for the Navier-Stokes equations including a solver for advection, diffusion and pressure. Due to its modular software design, all solvers can be replaced by other implementations. The code is written in C++ and uses OpenACC to be executed on a GPU or a multicore CPU (central processing unit). OpenACC is a pragma-based parallelisation approach which allows to create parallelisable tasks that are distributed amongst the computing units of the target architecture. In order to use various architectures the code does not have to be adopted but the same source code can be used. Especially the usage of GPUs requires code adoptions in other programming techniques. The efficiency and portability of ARTSS was shown in [7], while [3] demonstrates validation cases and the codes scalability.

ARTSS solves the mass and heat transport equations with the following assumptions [3]:

1. Low velocity: The flow velocity is assumed to be low, i.e. $Ma := u/c < 1/3$, whereby the Mach number relates the local flow velocity u with the speed of sound in the medium c.
2. Incompressible flow: Fluid mass density $\rho(\mathbf{x}, t) = \rho_0$ is assumed to be constant.
3. Small changes of density: For the buoyancy force it is assumed that the changes of density, $\Delta\rho = \rho - \rho_0$, as a function of temperature, are small.
4. Constant properties: The specific heat capacity c_p, thermal conductivity k, and the viscosity μ are constant with respect to temperature.
5. Isobaric condition: The pressure in the thermodynamic process is assumed to be constant ($\Delta p = 0$).
6. Zero viscous dissipation: The contribution of viscous dissipation is ignored in the energy equation, i.e. $\mathbf{\Phi} = 0$.

These simplifications lead to the following system of equations:

$$\nabla \cdot \mathbf{u} = 0 \tag{1}$$

$$\partial_t \mathbf{u} + (\mathbf{u} \cdot \nabla)\,\mathbf{u} - \nu \nabla^2 \mathbf{u} + \frac{1}{\rho_0} \nabla p = \frac{1}{\rho_0} \mathbf{f}_B(T) \tag{2}$$

$$\partial_t T + (\mathbf{u} \cdot \nabla)\,T - \alpha \nabla^2 T = S_T \tag{3}$$

where $\nu = \frac{\mu}{\rho_0}$ describes the kinematic viscosity, $\alpha = \frac{k}{\rho_0 c_p}$ the thermal diffusivity, and S_T denotes a volumetric energy source.

ARTSS uses the finite difference method (FDM) on a structured grid to approximate derivatives. In order to represent obstacles and boundaries, ARTSS uses index lists, e.g. a list that contains only the indices of inner grid cells to be considered by the solver. All variables, here velocity, temperature, and pressure, are stored as cell centered values (collocated grid cf. Fig. 1). With collocated grids, it is necessary to introduce a layer of so called ghost cells on the domain's boundary to apply the boundary conditions of choice. This allows discretisation operators, or stencils, to be used at cells adjacent to the boundary as they are used in the interior of the domain. In order to quickly access the boundary cells their indices are again stored in a list. Therefore, the relocation of boundaries is done by adopting the according index lists. The adoption of this data structure is employed in the domain extension method.

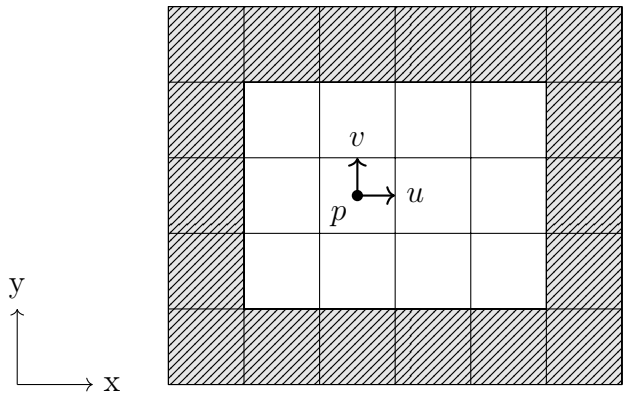


Fig. 1. 2D collocated grid with ghost cells (marked in gray with line pattern). Velocities (u,v), temperature T and pressure p are stored at cell center.

2.2 Domain adoption method

Starting with the technical implementation, a distinction between the computation domain and the complete geometry is required. Ideally, the smoke and heat source can be located and correspondingly the starting selection of the domain is constructed around it.

ARTSS features an implementation for distinguishing between the total geometry (physical domain) and its subdomain to be calculated (computational domain). Both domains are mandatory cuboid and since ARTSS has no implementation of the concept of rooms, the expansion is done by adding planes of mesh cells in the extension direction. The size of the expansion can be defined by the variable `expansion size`. However, there is a restriction on the value of this variable due to the applied multi-grid method. This method requires the number of mesh points in a direction to fit the underlying restriction and prolongation steps. Accordingly, `expansion size` $= 2^{\text{level}}$ applies, where level stands for the number of levels of the multigrid method. The variable `check value` is used as a threshold and therefore determines if a cell is of interest and to become part of the computational domain. In the following examples, just the temperature is evaluated and not the smoke concentration. The cells, which are checked for smoke are specified by the variable `buffer size`. The complete plane is examined, in Fig. 2 marked by the gray level. `buffer size` determines the distance to the end of the computational domain. The `time counter` parameter is used to set the interval for the expansion check, e.g. every other time step.

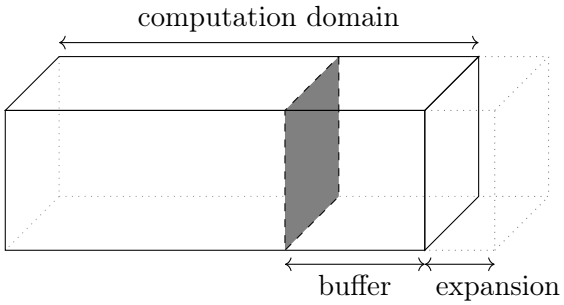


Fig. 2. Parameter `buffer size` and `expansion size` in an exemplary scenario

On the implementation side, the expansion process is divided into three steps (cf. Fig. 3): The control step, which contains the update method, where the check whether to expand or not occurs, the perform step, which contains the methods `resize` and `applyChanges`, where the domain is expanded and new values are set and the third step adjust, where everything else is adjusted to fit the new size of the domain, e.g. setting new boundaries. The `update` and `applyChanges` methods, as well as the variables `check value`, `buffer size`, are part of an interface and can be easily replaced by the user with their own methods.

In the approach used, the control step passes through the plane indicated by the `buffer size` cell by cell and checks if a cell exceeds the `check value`. If this is the case, an expansion is ordered and in step perform the values of the corresponding last cell are copied into the cells of the new domain.

The control step contains a check of the plane defined by the `buffer size`. If a cell is found that exceeds the `check value`, an expansion is initiated. In the perform step the outermost cell (excluding the ghost cells) is used as reference cell. From this the containing

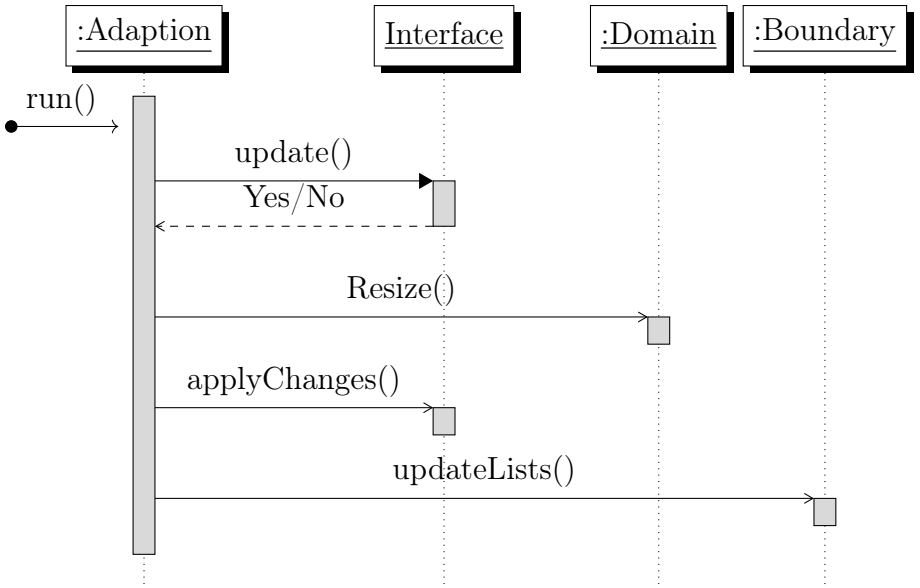


Fig. 3. UML sequence diagram of expansion process in ARTSS. The class `Adaption` manages everything regarding the expansion/adaption process. The `Domain` class contains data about the domain (e.g. start position of domain, how many cells in a direction etc.). The `Boundary` class is responsible for everything regarding the boundary. `Interface` is the interface to which alternative approaches to adaptation can be inserted.

values – except the velocities – are taken and set for all new cells of the expansion as shown in Fig. 4.

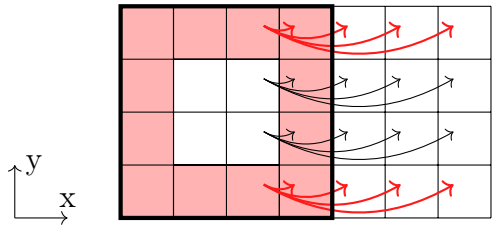


Fig. 4. Handling of new cells after an expansion in x-direction. The cells enclosed by the thick black line corresponds to the computational domain. The red marked cells are ghost cells.

3. Simulation setting of an example application

3.1 Setup details

The computational domain is a elongated rectangular geometry with open boundary conditions at the boundaries in x-direction, respectively a tunnel. We deliberately chose a

simple geometry to demonstrate the domain expansion without being distracted by complex obstructions. In the first third of the tunnel (x = 30 m) the heat source is located, which is represented as a volumetric source term in the energy equation. For demonstration purposes very high gas temperatures were created, which do not correspond to real temperatures.

Over time, the heated medium spreads across the tunnel, i.e. it rises to the ceiling and spreads along it. The spread is symmetrical until it reaches the closer left ending of simulation domain. Snapshots of the temperature field for the static and dynamic cases are shown in Fig. 5.

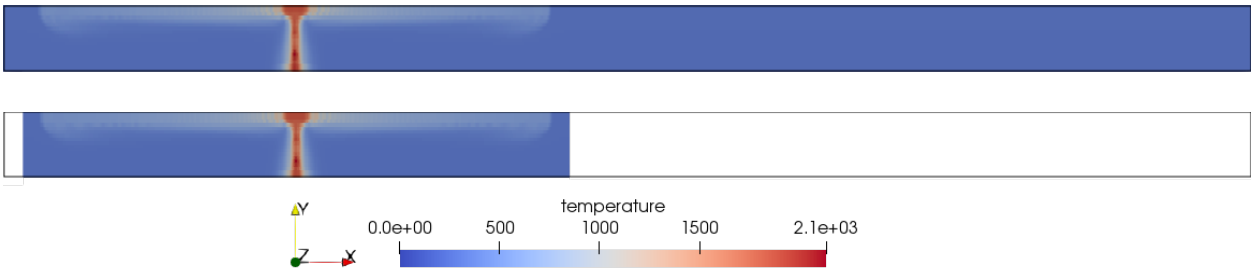


Fig. 5. Temperature [K] field of tunnel scenario without (top) dynamic expansion, i.e. static case, and with (bottom) a dynamic expansion, i.e. dynamic case. The bottom figure indicates the computational domain at the shown simulation time.

For the open ends in x-direction (left and right boundary) an outflow boundary condition was chosen. No-slip boundary conditions are applied for the enclosing walls in the y- (top and bottom boundary) and z-direction (front and back boundary), see Fig. 6.

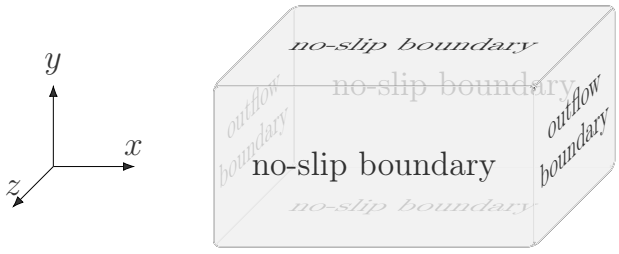


Fig. 6. Boundary conditions in the tunnel example.

The used settings are enlisted in Tab. 1 and Tab. 2. If variations were used, e.g. for speedup measurements (dt), it is explicitly indicated. For the static cases the computational domain is the same as the physical one.

The hardware details of the machine on which the performance calculations were made can be found in Tab. 3. For the multicore version, only one of the two sockets with an eight core CPU was used.

Physical parameters		Value	Unit
Simulation time	t_{end}	30	s
Kinematic viscosity	ν	2.44139e-05	m^2/s
Computational domain	(x_1, x_2)	$(26, 34)$	m
	(y_1, y_2)	$(-3, 3)$	m
	(z_1, z_2)	$(-4, 4)$	m
Physical domain	(X_1, X_2)	$(0, 128)$	m
	(Y_1, Y_2)	$(-3, 3)$	m
	(Z_1, Z_2)	$(-4, 4)$	m
Domain boundary condition			
- front, back, top, bottom	u, v, w	0	m/s
- left, right	$\partial_n u, \partial_n v, \partial_n w$	0	1/s
- front, back, top, bottom	p	0	Pa
- left, right	$\partial_n p$	0	Pa/m
- all walls	$\partial_n T$	0	°C/m
Numerical parameters			
Time resolution	dt	0.001	s
Number of inner cells	$N_x - 2$	64	
	$N_y - 2$	16	
	$N_z - 2$	32	

Tab. 1. Tunnel test case: Physical and numerical parameters

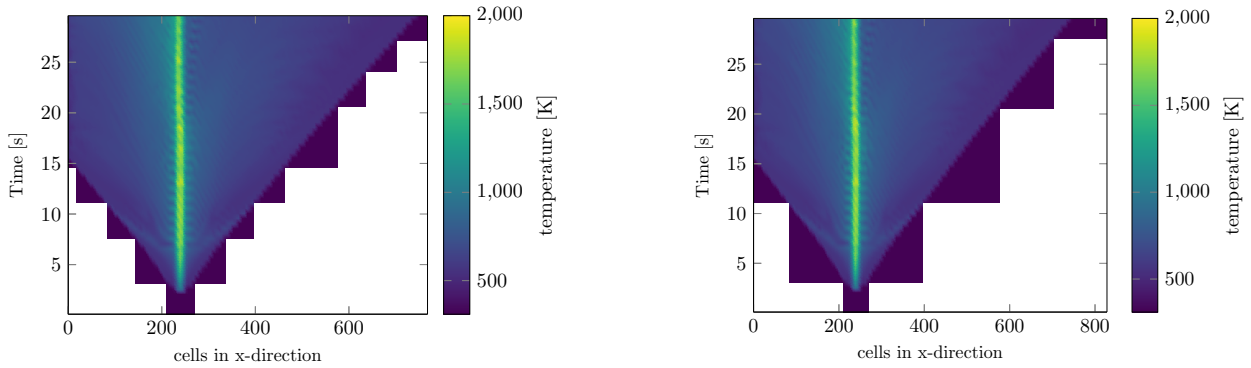
Fractional step	Method	Parameter	
Advection	Semi-Lagrangian		
Diffusion	Jacobi	100 iter's or	$\delta_{\text{tol}} = 1 \times 10^{-7}$
Turbulence	Smagorinsky	constant	$C_s = 0.2$
Pressure	Multigrid	2 cycles	6 levels
	- pre-conditioning	100 cyc's/ iter's or	$\delta_{\text{tol}} = 1 \times 10^{-7}$
	- Jacobi relaxation	$\omega = 2/3$	4 iterations
	- Jacobi solver	$\omega = 2/3$, 100 iter's or	$\delta_{\text{tol}} = 1 \times 10^{-7}$

Tab. 2. Tunnel test case: Solution method and parameters

The temporal dynamics of the domain adoption is illustrated in Fig. 7. The figures show the temperature profile at a fixed height and $z = 0$ along the x-direction over time. Here, the temperature field is evaluated and a threshold value determines if the domain needs to be extended. The graphs demonstrate the progress over time of the domain expansion and the temperature development. These cases visualise the dynamic adoption with two different values for the expansion size resulting in different regions that are added to the computational domain whenever the temperature threshold was met.

Processor	Details
CPU	Intel® Xeon® CPU E5-2623 v4 @ 2.60GHz, 2x8 cores
GPU	NVIDIA Pascal P100, PCIe, 1382 MHz, 720 GB/s, 16 GB, 3584 cores

Tab. 3. Hardware details



(a) `expansion size` = 1 (b) `expansion size` = 2

Fig. 7. Visual representation of the simulation with dynamic expansion of a domain slice over time. The data below the stairs shows the domain that is not yet part of the computational domain.

3.2 Comparison of static case with FDS

To give a brief overview of the accuracy of ARTSS, the static case is compared with data simulated by FDS. As this is a generic setup, there is no experimental data to be compared with. A more detailed validation and verification, i.e. including more cases, of the models used in ARTSS can be found in [3].

For the comparison with FDS the values of the heat release rate (HRR), the simulation time t_{end} and the time step dt were adjusted to fit each other (cf. Tab. 4).

	HRR [MW]	t_{end} [s]	dt [s]	time steps
ARTSS	3.0	60	0.02	3000
FDS	3.6	60	dynamic	3278

Tab. 4. Adapted parameter for a comparison between ARTSS and FDS: Given the pure convective heat release in ARTSS, 20% of the HRR was added to the FDS value to achieve a similar effective HRR. In addition, unlike ARTSS, FDS uses a dynamic time step, which is why the time step of ARTSS has been adapted to the number of time steps in FDS.

The first comparison was made at the end of the simulation, i.e. at $t = 60$ s, and the propagation of the ceiling jet was compared. As can be seen in Fig. 8, both simulations have similar smoke propagation.

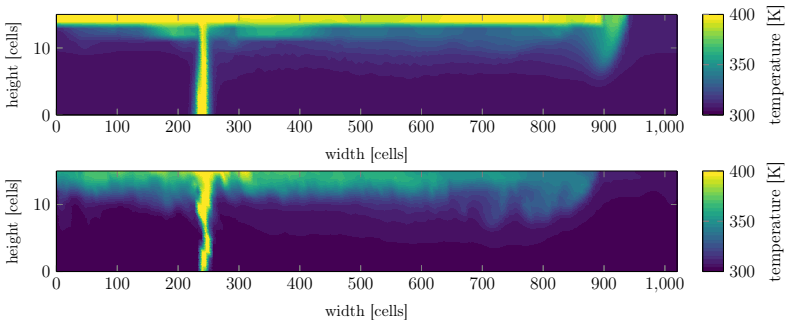


Fig. 8. Cross section of temperature profile at the end of the simulation $t = 60$ s. The temperature values are limited to 400 K for better visual evaluation of the ceiling jet.

In general, the predicted temperature in ARTSS is higher than in the FDS simulation. Fig. 9 visualises the time evolution of the temperature at a fixed position. Furthermore, the course clearly shows that the data of the FDS simulation is much more dynamic than that of ARTSS, but the rough course is consistent. Given that ARTSS specialises in fast calculation of smoke propagation, these deviations are within reasonable limits.

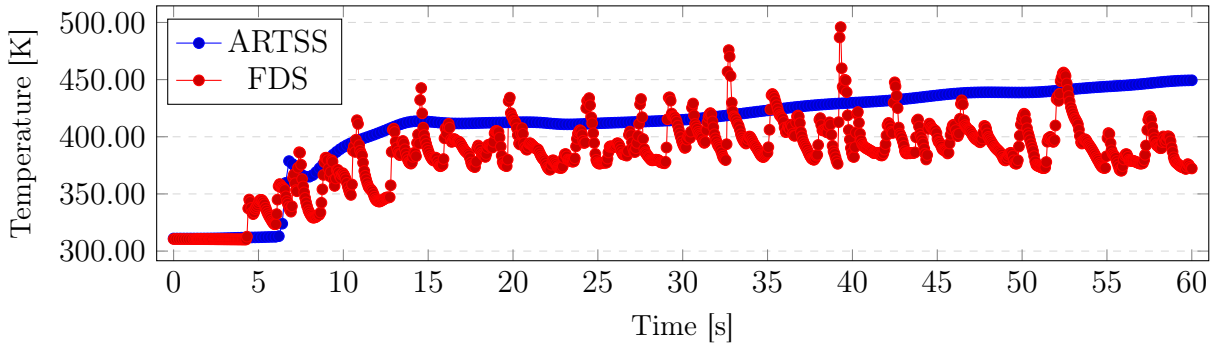


Fig. 9. Temperature at point (height = 15, width = 200) over the whole simulation time

4. Analysis of execution time

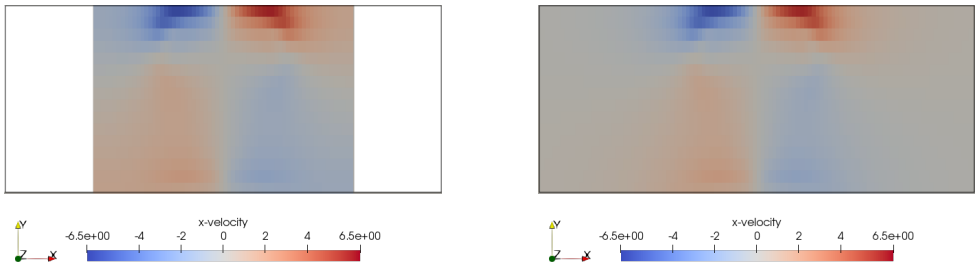
In the following, the effects of the expansion on the execution time are examined, how the individual parameters contribute to it and to what extent the result with dynamic expansion differs from the static simulations.

4.1 Accuracy

In many cases, the acceleration due to a dynamic domain expansion is at the cost of accuracy. The later the domain is expanded, the less has to be calculated, the shorter is the

computing time. Overhead time is caused by the control step and the resulting perform and adjust steps, if an expansion is necessary. The methods of step control and step perform are designed in a very simple fashion to keep the additional time low and thus achieve as much acceleration as possible. Correspondingly, parameters that change the accuracy also influence the runtime and vice versa. The correlation between runtime and accuracy does not get further mentioned in the following, but is regarded as a generally resulting effect.

To compare the static version with the dynamic version, the L^2 norm for the entire domain at the end of the simulation was calculated. The difficulty is to determine, if an expansion is considered unsuccessful. This is the case if data was lost due to an delayed domain expansion. Fig. 10 shows an example of the x-velocity profile just before an expansion is triggered by the temperature field. This illustrates that information of the velocity field is already missing and thus may eventually change the simulation results.



(a) Dynamic case (b) Static case

Fig. 10. x-velocity profile for static and dynamic case.

Therefore the validation is partly visual and the L^2 norm is used to compare the loss of accuracy of the dynamic versions. For a result to be considered valid, the temperature development of the dynamic and static version needs to advance similarly. If the difference is plotted, images similar to Fig. 11 (left) are obtained for valid cases and for invalid images similar to Fig. 11 (right). Invalid cases are not further considered in the following examinations.

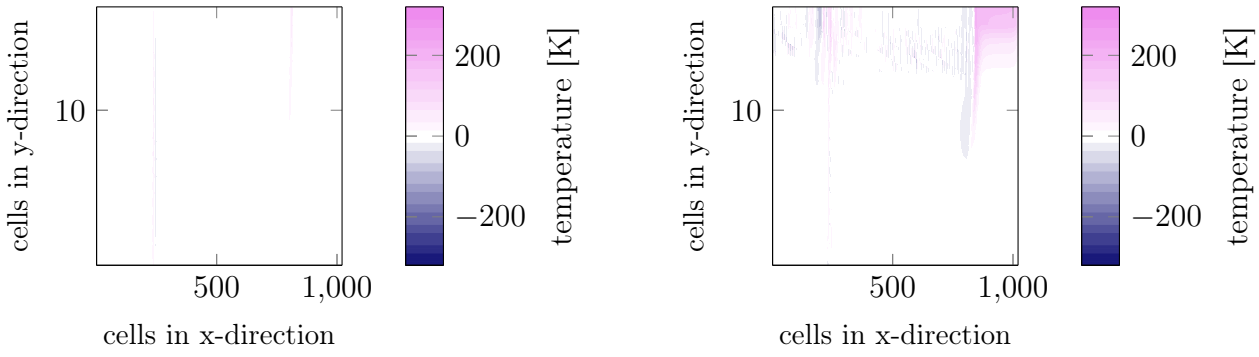


Fig. 11. Temperature difference between dynamic and static version. Left: successful expansion. Right: failed expansion.

Accordingly, all simulations shown in the following sections are considered as valid — w.r.t. the expansion procedure – and no large temperature differences were determined. After all, the user decides which deviation from the static version is acceptable.

4.2 Parameter Influence on Accuracy

A correlation can be observed in the inverse proportionality of `buffer size` and `check value` as can be seen in Fig. 12. The smaller the `check value` or the larger the `buffer size`, the earlier the expansion occurs and the difference to the static version becomes smaller.

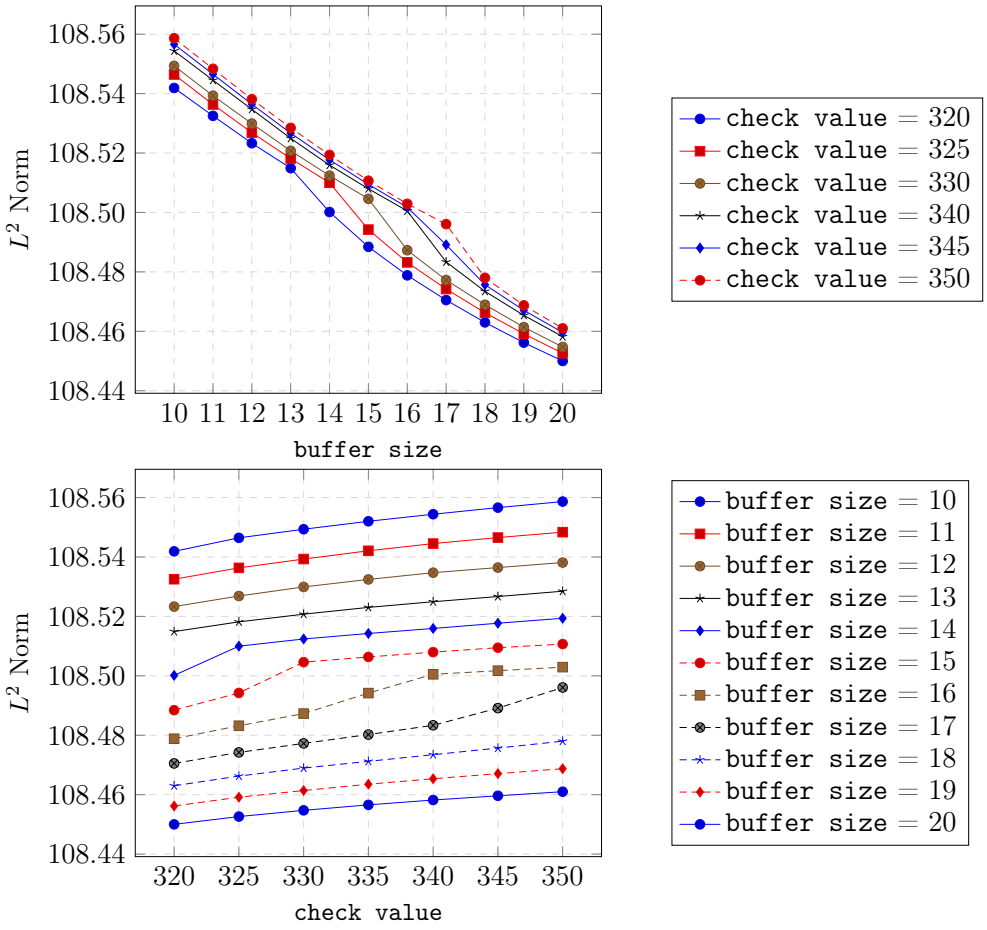


Fig. 12. Correlation between parameter `check value` and `buffer size`.

The earlier the domain is expanded, the longer the computing time, accordingly the choice of the two parameters affects the runtime. However, this follows from the general compromise between performance and accuracy, therefore it can be assumed that the two parameters only influence the performance indirectly as shown in Fig. 13. An earlier

expansion leads to a bigger domain, which results in a slower runtime. In the graph, the runtime differences are very small, but the trend shows that it does make a difference even if it is small in percentage terms. This means that if a more complex algorithm were chosen for adaptation, the percentage could increase and the choice of check value and buffer size would have a much greater effect on the runtime.

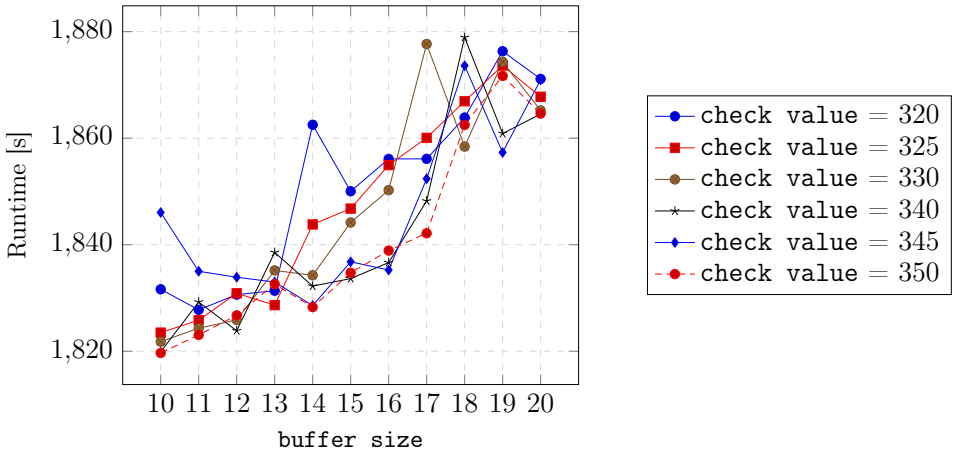


Fig. 13. Correlation between parameter `check value` and `buffer size` regarding runtime.

4.3 Performance

For the performance analysis a test scenario was chosen where at the end of the simulation the domain size of the dynamic case corresponds to the domain size of the static case. The performance showed a speedup of 1.38 for the GPU, 1.51 for the multicore and 1.56 for the serial version with time step $dt = 0.5\,\mathrm{s}$ (cf. Fig. 14 and Tab. 5), this time step differs from the described simulation setting in Tab. 2. In Tab. 5, the runtime of the GPU execution is higher than that of the multicore execution in the dynamic case. This is because in the current development stage, the routines for the domain extension still accesses serial parts in the code, which slows down the execution on a GPU considerably. This is due to the fact, that changes of data structure sizes are not handled well within OpenACC. Therefore data structures have to be exchanged between the GPU and CPU in the dynamic cases, while the static ones run fully on the GPU.

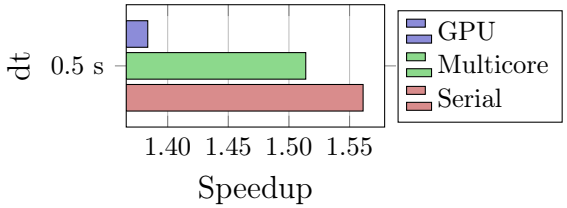


Fig. 14. Speedup of serial, multicore and GPU version with time step $dt = 0.5\,\mathrm{s}$.

	Serial	Multicore	GPU
Dynamic	605.0	97.5	98.1
Static	944.5	147.5	135.7

Tab. 5. Runtime [s] of serial, multicore and GPU version with time step $dt = 0.5\,\mathrm{s}$.

The advantage of the dynamic expansion lies in the time it takes for the dynamic version to reach the same domain size as the static one. If the calculation time is increased, for example by reducing the time step, the speedup increases (cf. Fig. 15 and Tab. 7).

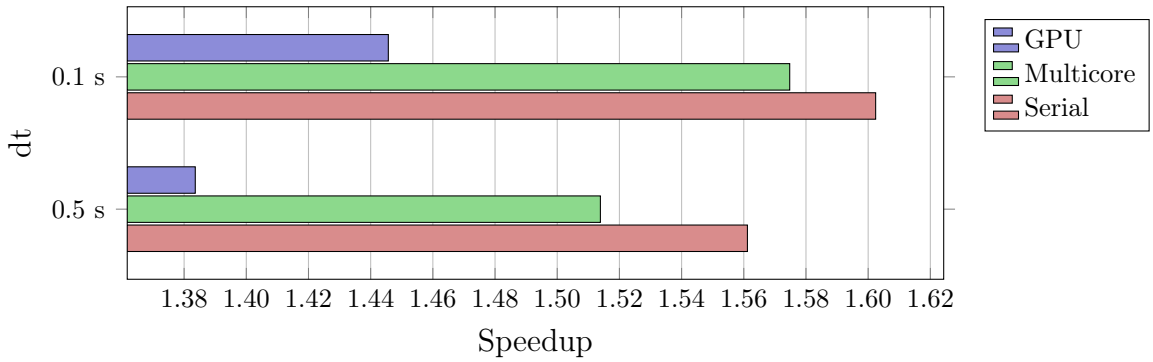


Fig. 15. Comparison of speedup of serial, multicore and GPU version for simulation with time step $dt = [0.5\,\mathrm{s}, 0.1\,\mathrm{s}]$.

	Serial	Multicore	GPU
$dt = 0.5$ s	1.56	1.51	1.38
$dt = 0.1$ s	1.60	1.58	1.45

Tab. 6. Speedup of serial, multicore and GPU version with time step $dt = [0.5\,\mathrm{s}, 0.1\,\mathrm{s}]$.

	Serial	Multicore	GPU
Dynamic	450.4	71.3	67.7
Static	721.7	112.3	98.9

Tab. 7. Runtime [s] of serial, multicore and GPU version with time step $dt = 0.1\,\mathrm{s}$.

How exactly the dynamic expansion reduces the execution time is demonstrated by plotting the time for the individual time steps as shown in Fig. 16.

It can be seen from the left subfigure that the runtimes of the individual time steps of the dynamic version, given by the smaller domain, are faster than those of the static version. The right image is a section of the left graph. The focus is on the runtime at the time of expansion (71st step). The running time is slightly higher than that of the previous points, but significantly lower than that of the following points. The difference of the runtime between the 71st step and the previous step corresponds to the additional effort resulting from the expansion procedure. Respectively, the combined time of step perform and step adjust. The form resembles the number of iterations of the Jacobi iteration, see Fig. 17. This is due to the fact that at the beginning of the simulation there is little dynamics with low velocities, as there is a smooth increase in heating power in the beginning. Therefore the calculation effort is lower in the first phase of the simulation.

4.4 Parameter Influence on Performance

The parameters `expansion size` and `time counter` have an indirect influence on the runtime, through their role within the steps control, perform and adjust. In order to

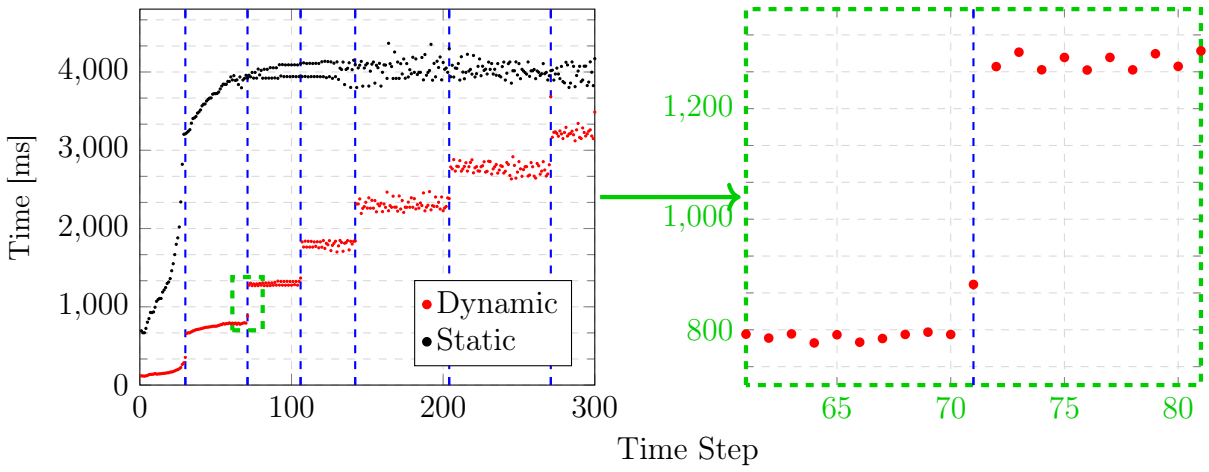


Fig. 16. Runtime of the individual time steps. The times where an expansion occurred are marked with blue dashed vertical lines. The right plot shows the section marked in the right plot in a higher scale, at about time step 70.

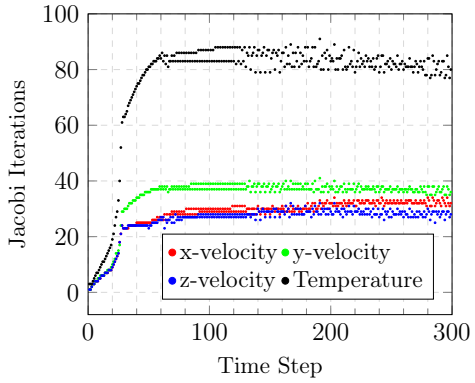


Fig. 17. Number of Jacobi iterations per time step.

investigate the impact of `time counter` on the control step, a timing of the according code sections can done. Since the time required for control is in the microsecond range and varied in different time measurements, the method was artificially extended by one second for measurement purposes, otherwise the runtime could not be distinguished from measurement inaccuracies. As can be seen in Fig. 18, doubling the value of the `time counter` parameter halves the time spent in the method. If the term of control is considered without artificial extension, the ratio is too small for `time counter` to influence the global runtime (cf. Tab. 8). But even here the ratio shows a reversed proportionality to `time counter`. Thus, if a more time-consuming control method is used, the total runtime can be reduced by `time counter`. However, when choosing the `time counter`, it must be considered that a necessary expansion may be recognized too late and thus a falsified result is created (cf. Fig. 11 right).

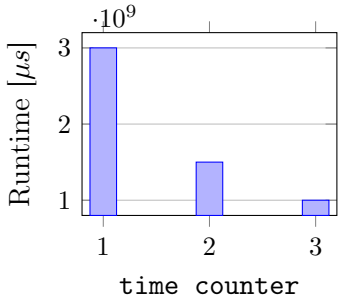
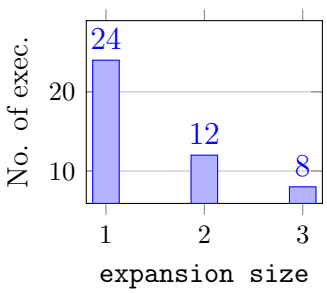


Fig. 18. Runtime of step control with different values for `time counter`.

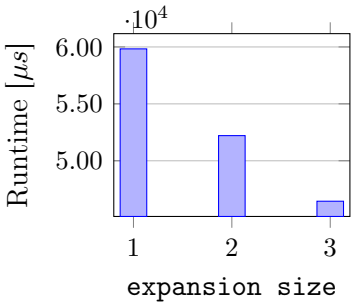
`time counter`	Global Time [s]	control [μs]	Ratio [%]
1	1690.76	71010	4.2e-05
2	1689.31	35864	2.1e-05
3	1691.43	29422	1.7e-05

Tab. 8. Influence of step control on the global runtime with different values for `time counter`.

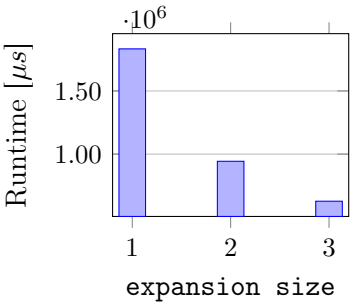
And with the `expansion size` the total running time of the steps perform and adjust changes. The larger the `expansion size`, the less frequently expansion is required, thus the total runtime of perform and adjust decreases (cf. Fig. 19). The parameters `expansion size` and `time counter` are therefore mainly a tool to control the additional time effort that emerges from the expansion.



(a) Number of executions of perform and adjust



(b) Total runtime of step perform



(c) Total runtime of step adjust

Fig. 19. Influence of parameter `expansion size` on runtime of step perform and adjust.

5. Conclusions

The aim of this contribution is to outline the impacts and consequences of a dynamic domain expansion for a smoke simulation in ARTSS, which can be summarised as follows: an acceleration of about 1.5 could be achieved by neglecting domains with lower temperature values. The approach for the type of examination and how the values of the new cells were substantiated after an expansion is kept very simple, but extendable in order to be able to implement complex extension criteria. It has been shown that the overhead caused by the expansion is negligible and can be effectively controlled by the parameters. Accordingly, more complex expansion methods can be used than those applied

here. Instead of the current handling for new cells to copy from inner cells, e.g. a combined use of the temperature and velocity gradient might be a better approach to determine the new values. The ideal approach would be an adaptive one in which the parameters change with the propagation speed of the smoke. At the moment it is not possible to make a dynamic expansion with changing boundary conditions, which would be necessary, for example, if the scenario takes place in a closed environment. Furthermore, the GPU version is slower as the multicore version because of the forced GPU and CPU data exchange. However, this issue will be addressed in future code and compiler versions. It can be summarised that a significant runtime improvement can be achieved despite a simply chosen approach that offers a lot of potential for further acceleration.

6. References

[1] B. Steffen, U. Kemloh, M. Chraibi, and A. Seyfried. Parallel Real Time Computations of Large Scale Pedestrian Evacuation. In *Proceedings of the Second International Conference on Parallel, Distributed, Grid and Cloud Computing for Engineering, /ed: P. Ivanyi, B.H.V. Topping, Civil-Comp Press, 2011. - 978-1-905088-44-7. - S. 95*, 2011.

[2] Tobias Kempe and Andreas Hantsch. Large-eddy simulation of indoor air flow using an efficient finite-volume method. *Building and Environment*, 115:291–305, 01 2017.

[3] Anne Küsters. *Real-Time Simulation and Prognosis of Smoke Propagation in Compartments Using a GPU*. Dissertation, Bergische Universität Wuppertal, Jülich, 2018.

[4] Kevin McGrattan, Simo Hostikka, Randall McDermott, Jason Floyd, Craig Weinschenk, and Kristopher Overholt. *Fire Dynamics Simulator User's Guide*, 2019.

[5] Wolfram Jahn, Guillermo Rein, and José L Torero. Forecasting fire dynamics using inverse computational fluid dynamics and tangent linearisation. *Advances in Engineering Software*, 47(1):114–126, 2012.

[6] F Michael Williams-Bell, B Kapralos, A Hogue, BM Murphy, and EJ Weckman. Using serious games and virtual simulation for training in the fire service: a review. *Fire Technology*, 51(3):553–584, 2015.

[7] Anne Küsters, Sandra Wienke, and Lukas Arnold. Performance Portability Analysis for Real-Time Simulations of Smoke Propagation Using OpenACC. volume 10524 of *Lecture Notes in Computer Science*, pages 477 – 495, Cham, 2017. Springer International Publishing.

Figure captions