Three practical ways to speed up the method of characteristics

Dr. Ir. F.J. Lingen
Dynaflow Research Group

ABSTRACT
Although the Method of Characteristics (MoC) has proven to be robust and accurate, it may involve long execution times when the ratio between the spatial and temporal resolution is at odds with the geometry of the piping system. This paper therefore describes three methods that are aimed at reducing the execution time of the MoC. The first two seek to obtain a coarse and accurate space-time grid, while the third is aimed at running the MoC on multiple processor cores. This paper shows that these three methods can reduce the execution time of the MoC by two orders of magnitude.

1 INTRODUCTION

The Method of Characteristics (MoC) has been widely used for multiple decades to simulate transient flow conditions in piping systems and networks. It is accurate, robust and straightforward to implement because its mathematical formulation embodies the way that pressure waves propagate through space and time (9,11,2). This, however, also imposes restrictions on the space-time grid that determines at which locations the flow rate and pressure need to be calculated. In particular, the MoC may require a large number of grid points when the piping system involves both short and long pipeline sections, or when the system involves dynamic boundary conditions and equipment — such as check valves — that change the flow conditions on short time scales. As a consequence, the execution time of the MoC may be in the order of tens of minutes to hours. While this is not spectacular when compared to the execution times of, say, typical CFD computations, it is certainly less than ideal, especially when multiple transient simulations are required to explore and understand the relevant problem space.

This paper therefore presents three methods for reducing the execution time of the MoC. The first method is actually a collection of methods and heuristics that are aimed at obtaining an optimised base space-time grid that comprises a near minimum number of grid points, and that still leads to sufficiently accurate results. The second method selectively coarsens the base grid to reduce the number of grid points in long pipeline sections. This method leads to a non-uniform grid spacing in time, but in such a way that its implementation is far from complicated. The third method is aimed at using the combined processing power of multiple processor cores to reduce the execution time of the MoC. The three methods can be applied independently from each other and their implementation does not require radical changes to the structure of typical computer programs implementing the MoC.

The effectiveness of the three methods is examined for three realistic piping systems with different traits. Not only the execution time of the MoC is an important aspect, but also the accuracy of the results. Indeed, a spectacular reduction in execution time is not worth much if the results are not accurate enough.

The methods presented in this paper are, to a large extend, variations, combinations and extensions of a multitude of methods and ideas that have been presented in earlier research. However, they do incorporate some twists that make them unique and that help to make the MoC more efficient.

2 ESSENTIALS OF THE MOC

The core idea underpinning the MoC is to evaluate the partial differential equations of mass and momentum conservation along the characteristic lines in the space-time domain. This yields a set of ordinary differential equations that can be transformed into a series of algebraic equations that, in turn, yield the pressure and flow rate at discrete locations in the space-time domain. Those locations will be referred to as the grid points that make up the space-time grid.

As the MoC has been described extensively in literature, not all the details will be repeated here. Instead, this paper focuses on the way that the MoC updates the pressure and flow rate in each grid point as it marches through time. The procedure is summarised in Figure 1 (left): the pressure and flow rate in point *P* is obtained from the pressures and flow rates in points *U* and *D* that are located at a previous time level. Both points are also located at the starting points of the characteristic lines C^+ and C^- that end at point *P*. The characteristic lines are described by the following equations:

$$x = x_p \pm a(t - t_p)$$

where a is the effective wave speed, and (x_p,t_p) denote the space-time coordinates of point *P*.

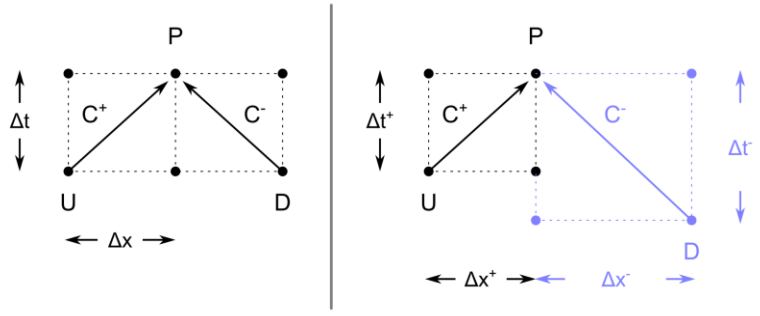

Figure 1: illustration of the way in which the MoC updates the pressure and flow rate in the point *P* by using the pressures and flow rates in the points *U* and *D*. The left figure shows a symmetric update while the right figure shows a skewed update.

The points *U* and *D* do not necessarily have to be located at the same time level, as long as they are the starting points of the positive and negative characteristic lines, respectively, that end at point *P*; see Figure 1 (right). This observation forms the basis of different interpolation schemes (3) that need to be applied when adjacent grid points are not joined by characteristic lines.

It is attractive to use a rectangular space-time grid that conforms to the characteristic lines. That is, a grid in which the ratio between the spatial and temporal distance between adjacent grid points equals the (local) wave speed of the fluid. Such a conforming grid not

only simplifies the implementation of the MoC, but also avoids adverse numerical effects, such as dispersion and damping (10), that typically arise when interpolation is needed to update grid points that are not joined by characteristic lines. On the other hand, the fixed ratio between the spatial and temporal resolution can be at odds with the geometry of the piping system to be modelled, especially when the system includes both short and long pipeline sections. In such a case the smallest pipeline sections will dictate the spatial resolution and therefore the temporal resolution. The methods described in the next two sections are aimed at addressing this particular problem without giving up all the advantageous properties of a rectangular, conforming grid.

Note that the above, and also the remainder of this paper, assumes that the effective wave speed at any point in space does not change (significantly) over time. This assumption is normally true when considering single-phase, incompressible fluids.

3 GRID AND MODEL OPTIMISATION

While the use of a rectangular, conforming space-time grid clearly has its advantages, generating such a grid is not trivial when the total number of grid points is to remain manageable. To illustrate this, consider the (contrived) piping system shown in Figure 2. Assuming a uniform wave speed, this seemingly trivial system requires at least 11,607 + 18,469 + 41,393 = 71,469 grid points in the spatial dimension in order to correctly represent the lengths of all three pipeline sections. It is not hard to imagine that the required number of grid points becomes way too large for a piping system of any complexity and with a non-uniform wave speed. Clearly, some concessions need to be made if the number of grid points is to remain manageable. One possibility is to abandon the idea of using a conforming grid and apply an interpolation scheme (10) to match the grid with the pipeline sections. Another possibility, and the one that is pursued here, is to allow the effective pipeline lengths and wave speed, as defined by the space-time grid, to deviate from the true pipeline lengths and wave speed. This is not an unreason able approach as the pipeline lengths and the wave speed are not known with an infinite precision (11).

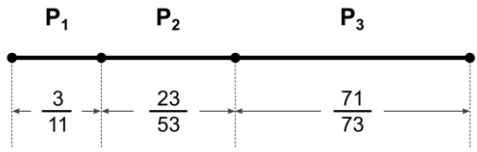

Figure 2: example piping system consisting of three pipeline sections. The indicated lengths of the sections are relative to each other.

In addition to the geometry of the piping system and the wave speed, there are other factors that affect the resolution of the space-time grid. These include: time-dependent boundary conditions; models of highly dynamic equipment, such as safety relief valves, that are solved together with the flow equations; and the stability criterion that is related to the way that the pipe-wall friction is incorporated into the MoC (11). The latter depends on the flow rate that is not known beforehand. However, one can make an educated guess based on the fluid type and typical flow velocities.

Here follows a basic algorithm for generating a rectangular, conforming space-time grid:

1. Determine a maximum time step size for each pipeline section. This depends on the particular implementation of the MoC and the minimum number of spatial grid points per section; this typically equals two or three. The maximum time step also depends on the other factors mentioned above.
2. Set the initial time step size equal to the smallest maximum time step size associated with each pipeline section.
3. Check whether an acceptable grid can be obtained by varying the lengths and wave speed between specified limits.
4. If this is the case, then the algorithm terminates. Otherwise, the time step size is decreased and the algorithm continues with the previous step.

The faster the time step size is decreased, the faster the algorithm will terminate. On the other hand, the slower the time step size is decreased, the less points the final space-time grid will contain, and the less time a transient simulation will take.

The basic algorithm can be improved in various ways. One way is to decrease the time step size in a more intelligent way by keeping track of an upper and lower bound on the time step size for each pipeline section, and by trying to nudge the time step to a value that lies between the lowest upper bound and the largest lower bound. Another way is to keep raising the time step size once an acceptable time step size has been found. This is only useful if the rate with which the step size is increased is lower than the rate with which it is decreased. This approach makes it possible to set a fast, decreasing rate to start with, and still end with a time step size that is near its optimum. It can be improved further by applying some random perturbations to the time step size in order to escape from a local optimum.

Consider, again, the piping system shown in Figure 2 and assume that section P_1 has an actual length of $100m$, and that the wave speed equals $1000m/s$. Further assume that the effective pipeline lengths and wave speed, as represented by the space-time grid, must not deviate more than 0.1% and 5%, respectively, from the actual lengths and wave speed. Then, when using the basic grid generation algorithm together with the above improvements, and when ignoring any restrictions on the time step size, the resulting grid contains only 22 points in the spatial dimension. This is a dramatic improvement when compared to the original 71,469 grid points.

Model optimisation

The grid generation algorithm can still yield a dense space-time grid when the piping system involves one or more sections that are very small in comparison with the overall system length. There are several solutions to this problem, three of which are considered here. This first solution is to view (very) small sections as “lumped” elements (11) that are handled in a similar way as orifices and valves. That is, the wave speed in such a small section is considered to be infinite and the pressure drop over the section is considered to be proportional to the square of the flow velocity. The length of a lumped section can be added to an adjacent pipeline section in order to preserve the total length of the system.

A slight variation of this solution is to impose a lower limit on the length of each pipeline section. If a section is smaller than this limit, then the section is essentially stretched. Again, the lengths of adjacent sections can be adjusted to preserve the total length of the system. Although this approach is a bit blunt, it can be effective when small sections have

been included solely in order to obtain an accurate model of a piping system and when those sections have a negligible impact on the calculated flow rate and pressure.

A third, and effective solution involves aggregating adjacent pipeline sections with matching properties – such as inner diameter and pipe-wall stiffness – into larger sections so that small sections are effectively removed from the piping model. This approach makes the implementation of the MoC more complex as the end points of the original pipeline sections do not necessarily coincide with the grid points in the optimised model. This means that additional data structures are required to determine how the pipeline sections in the original model are related to the sections in the optimised model. This also means that interpolation is required to determine the flow rate and pressure at the end points of the original pipeline sections. The gain, on the other hand, can be a significant reduction in the number of grid points, especially when the optimised model is made up of significantly longer pipeline sections than the original model.

Accuracy assessment

Any change to the space-time grid affects the numerical accuracy of the computed flow rate and pressure. A coarser grid will, in general, lead to less accurate results. Conversely, an increasingly finer grid will lead to more and more accurate results. This, then, suggests a pragmatic way to determine the accuracy of the results: first run a transient flow simulation with a coarse space-time grid; then reduce the time step size and run a second simulation with a finer grid. If the results obtained with the two runs are sufficiently close, then it seems reasonable to assume that the results are accurate. Otherwise, the initial grid is too coarse and the finer grid should be used as the starting point for another iteration of this procedure.

While this is not a rigorous approach, it can spot situations in which the initial grid is too coarse to obtain sufficiently accurate results. The second run does increase the total execution time significantly, but this increase may be offset by the gain that is obtained with a more efficient space-time grid. This is especially the case when applying the grid coarsening method described in the next section.

Complexity tends to lurk in the details, and in this case one of those details concerns the comparison between the results obtained with two different space-time grids. If the comparison is too strict, then the final space-time grid might be too fine for the level of accuracy that is deemed to be sufficient. The approach that has been adopted here removes the temporal dimension by comparing the maximum and minimum flow rate, pressure and unbalanced (fluid-induced) force that is obtained per pipeline section over the entire simulation time. This not only simplifies the comparison, but also bases the comparison on the quantities that tend to be the most relevant in transient analyses. To make the method more robust, the maximum and minimum values are compared for overlapping clusters of adjacent pipeline sections. Thus, if a maximum or minimum occurs very near an end point of a section, then a slight shift of that maximum or minimum to the adjacent section will not cause the comparison to yield a negative result.

4 GRID COARSENING

The number of grid points in a rectangular space-time grid will, in general, be determined by the smallest pipeline section in the piping system, even when applying the optimisation methods presented in the preceding section. The grid coarsening method presented here therefore gives up the idea of using the same time step size for all pipeline sections while

aiming to preserve the advantages of a rectangular, conforming grid in which the ratio between the temporal and spatial resolution equals the wave speed. The method is based on earlier work (7,8,3) and adds a little twist that simplifies its implementation.

The grid coarsening method starts with a rectangular, conforming grid with a fixed time step size that is referred to as the base time step size. Then, for each pipeline section, it repeatedly doubles both the time step size and the spatial distance between the grid points until the time step size exceeds the maximum time step size associated with the section, or until the effective, grid-spanned length of the section deviates too much from its actual length. In this way the time step size associated with each section is equal to the base time step size times a power of two. The ratio between the section-specific time step size and the base time step size is called the section *grid level*. Note that sections with a time step size equal to the base step size have a grid level equal to one. Figure 3 shows an example of the resulting grid for three pipeline sections that have grid levels one, four and two, respectively. The symbol Δt_1 denotes the base time step size.

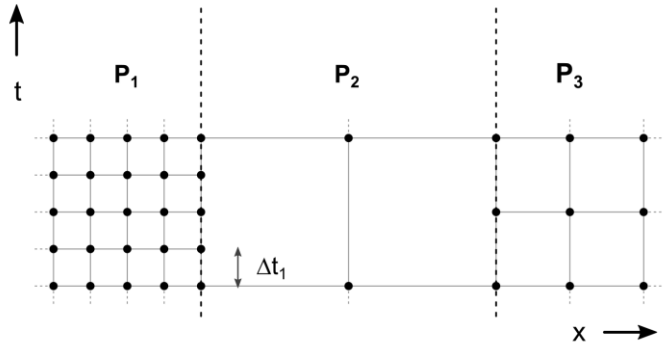

Figure 3: example of a space-time grid obtained after coarsening the base grid with base time step size Δt_1. The pipeline sections P_1, P_2 and P_3 have grid levels equal to one, four and two, respectively.

After generating a non-uniform grid in this way, the MoC proceeds more or less in the usual way by marching through time with the base time step size. However, the pressure and flow rate are only calculated for the grid points within a pipeline section when its grid level is a multiple of the current time step number. If that is not the case, then the section is simply skipped; it is deemed to be inactive during the current time step.

The application of grid coarsening does not require different data structures than those that are required when using a rectangular grid. In particular, it is sufficient to store the flow rate and pressure for two successive time steps, just like when using a rectangular grid. The only difference is that those time steps will correspond with different base time steps for different pipeline sections, depending on the grid levels associated with those sections.

Interpolation between grid levels

Interpolation is required when two or more adjacent pipeline sections have different grid levels. This is not very complicated, however, because the time step sizes associated with those sections are always an exact multiple of the base time step and of each other. In particular, there is no need for an extrapolation step (7); a simple time-line or space-line interpolation scheme (3) is sufficient. The current implementation uses a first-order space-

line interpolation scheme as that proved more convenient to embed into an existing MoC implementation.

Figure 4 shows how the pressure and flow rate are updated at the boundaries of three pipeline sections and for four consecutive time steps. The time steps are marked with different colours and symbols. The slanted arrows in the figure indicate the characteristic lines that end at the grid points. As not all starting points of the characteristic lines correspond with a grid point, the flow rate and pressure are linearly interpolated between the grid points A_1 and B_1 to obtain the flow rate and pressure at the points A_2, A_3 and A_4. Likewise, the flow rate and pressure need to be interpolated between the points B_1 and C_1 to update point C_3. This is similar to the skewed update shown in Figure 1 (right). Note that because the effective section-local time step sizes are always an integer multiple of each other, the interpolation coefficients are trivial to determine and do not need to be stored explicitly.

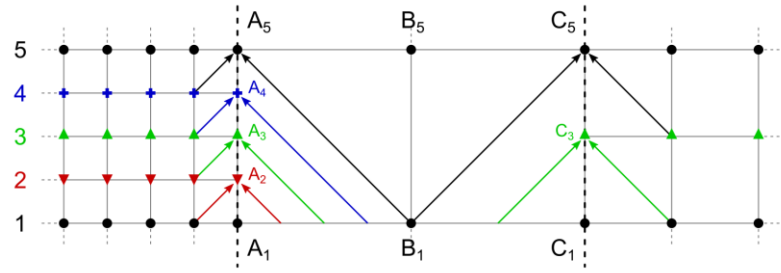

Figure 4: schematic representation of the method for interpolating results between grid levels. The colours and symbols correspond with different time steps, and the slanted arrows indicate the characteristic lines along which the flow rate and pressure are updated.

Impact on execution time and accuracy

When grid coarsening is applied, the grid resolution is no longer dictated by small pipeline sections. Indeed, the grid resolution is much more determined by stability and accuracy requirements than the geometry and topology of the piping system. As will be shown in Section 6 the reduction in execution time can be spectacular, especially when the system contains relatively long pipeline sections.

As there is still no such thing as a free lunch, the reduction in grid resolution is inescapably tied to some reduction in accuracy. The interpolation between grid levels also introduces some numerical artefacts such as dispersion. This is illustrated for a simple, water-filled pipeline system consisting of three adjacent pipeline sections with lengths of $1km$, $10m$ and $1m$. Next comes a valve and another pipeline section with a length of $100m$. The valve is closed in $0.5s$ with a classic water hammer event as a result. The pressure and flow rate have been computed without and with grid coarsening. When no grid coarsening is applied, then the space-time grid contains about 1,500 points in the spatial dimension. This number is reduced dramatically to about 60 when grid coarsening is enabled. Note that no model optimisation has been enabled to avoid that the three pipeline sections preceding the valve are merged into a single section.

Figure 5 shows the pressure just upstream of the valve as function of time. Grid coarsening has almost no effect on the computed maximum and minimum pressures, even though the

number of grid points is much lower. Grid coarsening does affect the effective speed with which the pressure waves travel through the system. The reason for this is that the (linear) interpolation algorithm (temporarily) speeds up the transmission of the wave front when it passes from a higher to a lower grid level. This is essentially a manifestation of numerical dispersion that is known to be associated with interpolation between grid points.

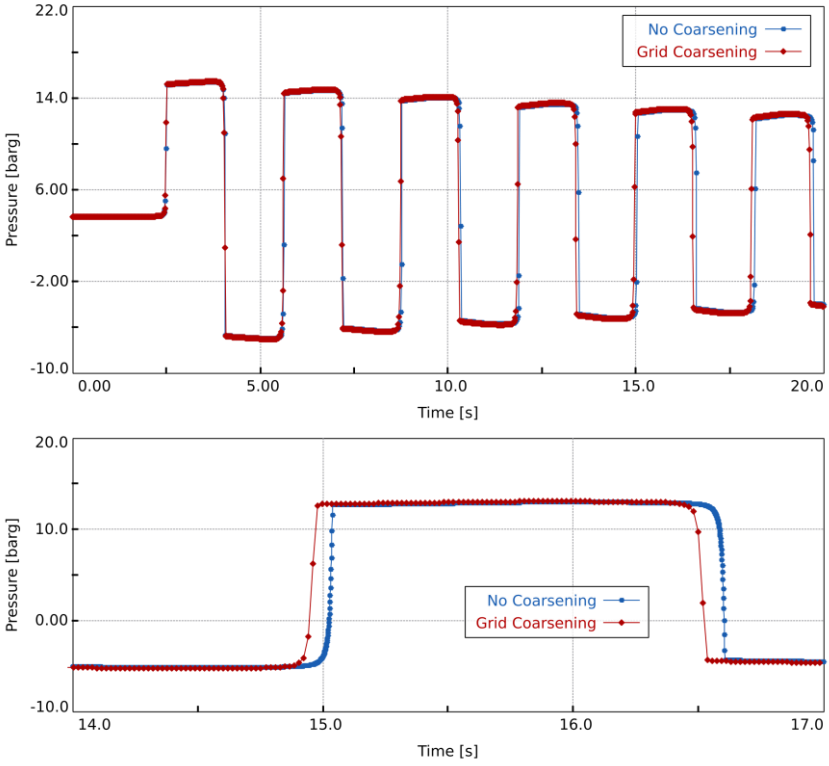

Figure 5: the pressure upstream of the valve as function of time with and without grid coarsening. The lower graph shows a close-up view of the upper graph.

Even when grid coarsening does not significantly affect the calculated pressure and flow rate, it can still have a noticeable impact on the calculated unbalanced forces. The reason is that a small change in the effective travel speed of pressure waves can have a significant effect on the difference in pressure at opposing elbows at the end points of long pipeline sections. This will be illustrated in Section 6.

5 PARALLEL EXECUTION

Instead of, or in addition to, changing the space-time grid or the MoC itself, one can also make use of faster hardware in order to reduce the execution time. As the sequential speed of processors (CPUs) is not increasing as fast as in the past, it makes sense to take advantage of the combined processing power offered by the increasing number of

processor cores that make up contemporary CPUs. Indeed, any recent workstation, notebook and even phone comes with a couple to tens of processor cores.

An alternative approach involves making use of special-purpose accelerators such as graphics processing units (GPUs). While their use can reduce the execution time significantly (6), they do have some disadvantages. To begin with, they require significant changes to the way that the MoC is implemented in order to fully exploit the hardware. While this might not be too problematic as far as the MoC itself is concerned, the impact will be greater on models of equipment such as check valves, surge vessels and pumps. Another disadvantage is that the use of accelerators typically involves proprietary programming languages and tools with more or less vendor lock-in effects.

The present research focuses on the use of multiple processor cores in parallel to speed up the MoC. This involves distributing the computations and the data over the cores, and coordinating their execution. How this is done depends to a large extend on the parallel programming model that is used. When using the message passing programming model, for instance, the data structures need to be divided into multiple chunks that are assigned to different processor cores. Messages must be exchanged between processor cores to handle dependencies between computations. While this is certainly a viable approach, it makes the implementation of the code quite a bit more complex.

An alternative approach, and the one adopted here, is to make use of the shared memory programming model. In this case the data structures do not need to be divided as all processor cores can access those data structures concurrently. Dependencies between computations are handled by means of synchronisation constructs that coordinate the execution of the processor cores and that prevent multiple cores from writing to the same memory location at the same time.

Outline of the implementation

The parallel implementation of the MoC assumes that the program is executed by multiple, concurrent *threads* (1), each one running on a different processor core, and each one processing a different chunk of the space-time grid. The initial thread, or main thread, coordinates the execution of the other threads and makes sure that dependencies between computations are met. The main thread also handles operations that are to be executed sequentially. This approach has the advantage that it does not require complicated and error-prone locking mechanisms for avoiding data race conditions.

When the program starts it is executed only by the main thread, just like a sequential implementation of the MoC. The main thread sets up the main data structure, including the *current* and *next* arrays that store the pressure and flow rate associated with the current and next time step, respectively. The main thread then divides the space-time grid into non-overlapping virtual domains that indicate which grid points are to be processed by which thread. It also creates a synchronisation data structure, referred to as the *thread arena*, that is used to coordinate the execution of the other threads; more about this later. After all this, the main thread spawns the other threads, passing each of them a virtual domain, a pointer to the main data structure, and a pointer to the shared thread arena. The other threads then wait for the main thread to indicate how to proceed.

The main thread starts the execution of the MoC that proceeds in parallel as follows. First, the main thread signals the other threads that they should calculate the flow rate and pressure at the grid points within their domain and at the next time step. The main thread performs the same operation concurrently with the other threads. Because the domains do

not overlap, there is no danger that different threads write to the same memory location; see Figure 6. The threads do need to read the pressure and flow rate associated with the domains of other threads, but this can be done in a safe way. All threads, including the main thread, then wait until the last thread has completed the operation. When that is the case, the main thread advances the time step number and signals the other threads that they should copy the data from the *next* arrays to the *current* arrays. Again, this can be done safely because the domains do not overlap. These two operations – calculate the next data and then update the current data – are repeated until the final time step has been reached.

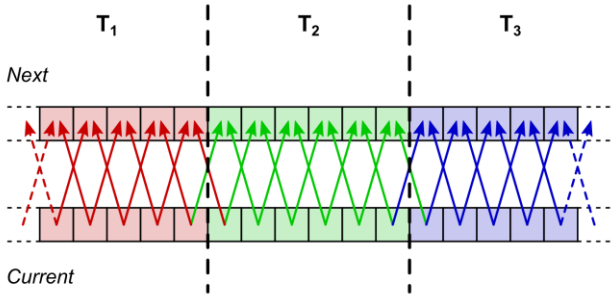

Figure 6: schematic representation of the parallel update procedure. The threads T_1, T_2 and T_3 concurrently read from the *current* array and write to the *next* array.

The above algorithm works only for the bare-bones MoC applied to a single string of pipeline sections. However, it can be extended in a straightforward way to include (tee) junctions and special, non-pipe elements like pumps and check valves. This involves extending the virtual domains with non-overlapping ranges of junctions and lists of special elements. Each time step then proceeds as follows: calculate the pressure and flow rate at the grid points and at the next time step; update the pressure, flow rate and MoC parameters at the (tee) junctions; update the state variables associated with the non-pipe elements; and copy the *next* data to the *current* data. Each operation starts with the main thread signalling the other threads and ends with all threads waiting until the operation has been finished. In other words, the operations are executed concurrently and in lock step with each other. This approach avoids data race conditions and makes sure that dependencies between computations are met. It does this without explicitly locking shared memory locations in the core code that implements the MoC.

Implementation of the thread arena

The thread synchronisation mechanism, referred to as the *thread arena*, essentially functions as a barrier construct that blocks the threads until they have all finished a particular operation. Its implementation largely determines the performance of the proposed parallel execution method because the execution of the threads must be synchronised multiple times per time step and because a transient flow simulation may require hundreds of thousands of time steps. This means that the thread synchronisation overhead must be very small in order to achieve an execution time in the order of minutes.

The first attempt to implement an efficient thread arena failed miserably. This implementation involved a shared data structure containing an integer together with a standard mutex and condition variable. The integer was used by the main thread to indicate what kind of operation was to be executed next, while the mutex and condition variable

were used to implement a barrier construct. This approach turned out to be much too inefficient because of memory contention, and system call and thread scheduling overhead.

The second, and final, attempt adopted a *busy-wait* construct with local-only spinning, very similar to the algorithm underlying the MCS Lock construct (5,4). This implementation proved to be much more efficient because it avoids memory contention and does away with the system call and scheduling overhead. In this implementation of the thread arena each thread has its own small data structure containing an integer that signals the next operation to be executed. These structures are organised in a linked list that is traversed by the main thread each time that the next operation is to be executed; see Figure 7. The linked list is composed and broken down with a lockless algorithm that is based on atomic compare-and-swap and atomic swap operations. The structures that form the list are pre allocated to avoid memory allocation overhead during the execution of the MoC.

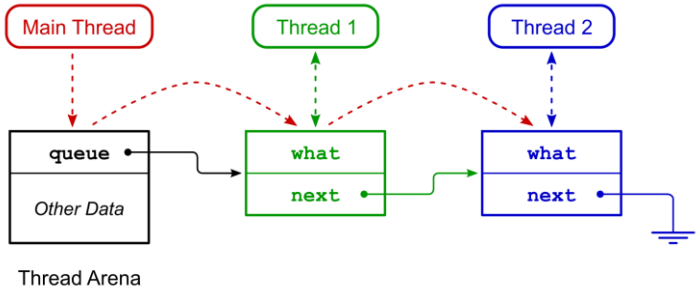

Figure 7: schematic representation of the thread arena. The main thread signals the other threads by writing to the `what` variables that are monitored in a busy-wait loop by the other threads.

Although the second implementation of the thread arena is much more efficient than the first one, it still involves some thread synchronisation overhead. This means that a significant reduction in execution time can be obtained only when the space-time grid contains a substantial number of grid points in the spatial dimension. This is shown in the next section.

6 EFFECTIVENESS ASSESSMENT

The three optimisation methods presented here – grid/model optimisation, grid coarsening and parallel execution – have been implemented in a commercial surge analysis application. They have consequently been used in numerous transient flow analyses involving a wide range of models of existing piping systems. Three of those models have been selected to show how the optimisation methods affect the execution time of the MoC. These models represent an LPG loading system (Model A), a firewater system (Model B) and a water distribution network (Model C). Their most relevant properties are listed in Table 1. Note that Model A (the LPG system) involves a discrete cavitation model (11) for simulating the formation and collapse of vapour cavities.

Table 1: summary of the relevant properties of the three piping models.

	Model A	Model B	Model C
Pipeline sections	1008	2048	718

Total length	3.1*km*	18*km*	99*km*
Shortest section	42*mm*	0.10*m*	0.25*m*
Longest section	10.1*m*	1.2*km*	17k*m*

Model A contains a few very short pipeline sections that represent the flanges connecting adjacent pipes. Although this level of detail is not needed in a flow analysis, the same model has been used to perform a mechanical analysis involving the unbalanced forces resulting from the transient flow analysis. Models B and C also contain some short pipeline sections because of the way that they incorporate some details of the actual piping systems.

The execution times shown in the remainder of this section have been obtained with an optimised implementation of the MoC in C++ that uses raw, restricted pointers to help the compiler generate efficient machine code. This code has been executed on a workstation sporting two Intel Xeon X5670 CPUs with a total of twelve processor cores.

Model optimisation and grid coarsening

Table 2 shows how model optimisation and grid coarsening affect the execution time for each of the three models. The first row lists the execution times that are obtained when using only the basic grid optimisation method described in Section 3. In order to keep the execution times somewhat reasonable, the effective pipe section lengths and wave speed (as represented by the grid) are allowed to deviate 1% and 10%, respectively, from their specified values. The second row lists the execution times when using the model optimisation method that aggregates adjacent pipe sections into larger sections. The third and final row lists the execution times when using both grid/model optimisation and grid coarsening.

Table 2: the execution time (in seconds) of the MoC when different optimisation methods are applied.

	Model A	Model B	Model C
Grid optimisation	2,680	21,600	784
Model optimisation	572	4,860	305
Grid coarsening	66.7	75.5	342

Model optimisation lowers the execution time substantially, especially for Models A and B because of the (very) short pipeline sections that are present in those models. When these sections are merged into longer sections, they have a much lesser impact on the resolution of the space-time grid. This effect is less pronounced for Model C as that model contains much less details than the other two.

Grid coarsening further lowers the execution time significantly, except for Model C. The reason is that a maximum time step size must be imposed on Model C in order to obtain sufficiently accurate results. This time step size is approximately equal to the time step size that is used when only model optimisation is applied. Hence, there is no scope anymore to increase the time step size by means of grid coarsening. Somewhat surprisingly, perhaps, the execution time for Model C increases when grid coarsening is applied. This has to do with the way that a user-imposed time step size is taken into account. Rather than reducing the maximum grid level, the base time step size is reduced so that a smaller imposed time step size results in a finer grid for all pipeline sections and not only the sections with the highest grid level.

Grid coarsening has a noticeable effect on the unbalanced forces that are calculated with Model B. These forces occur because the closure of multiple check valves leads to pressure waves repeatedly traversing the system. The dispersion introduced by grid coarsening slightly modifies the effective speed with which those waves travel through the system (see Figure 5). Although this effect is small when looking at the pressure or flow rate, it can become significant when looking at the difference in pressure at two opposing elbows.

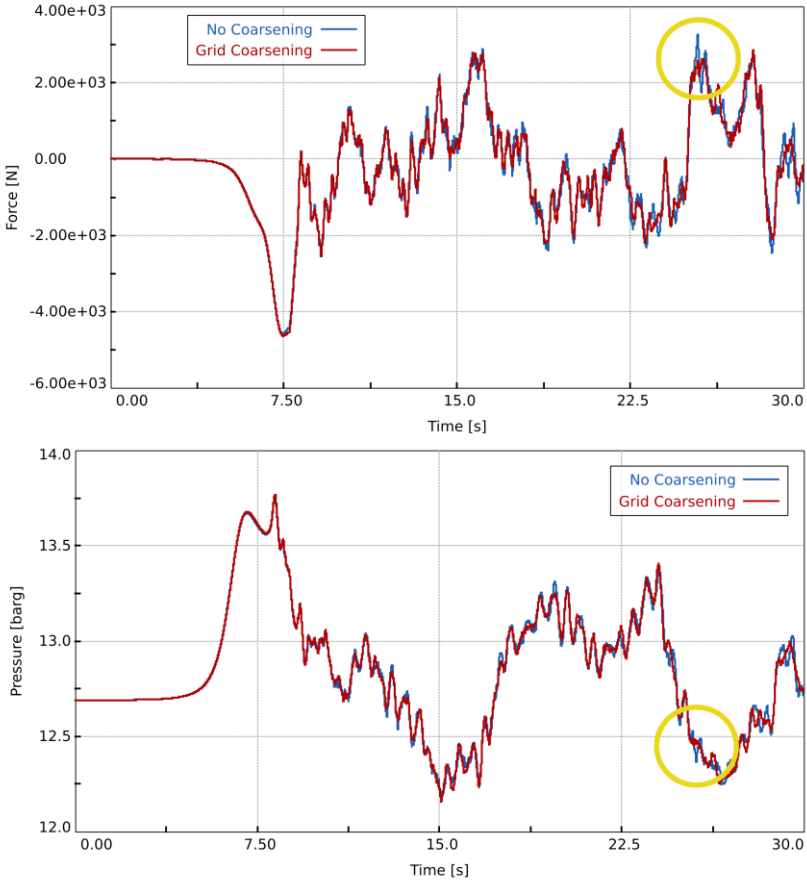

Figure 8: the unbalanced force (upper graph) acting on a pipeline section in Model B as function of time with and without grid coarsening. The lower graph shows the pressure at one of the end points of the pipeline section.

Table 3: the execution time (in seconds) of the MoC when an increasing number of processor cores is used.

Number of cores	Model A	Model B	Model C
1	66.7	75.5	342
2	53.4	71.0	329
4	34.2	48.7	223
6	32.3	43.1	195

The upper graph in Figure 8 shows the unbalanced force acting on a relatively long pipeline section with and without grid coarsening. The results are similar to begin with, but later they deviate by more than 20%. When looking at the pressure at one of the end points of the section (the lower graph), however, the results deviate much less. Fortunately, the unbalanced force is not a quantity of great interest in this case as the firewater system represented by Model B is mostly buried.

Parallel execution

Table 3 shows how the use of multiple processor cores affects the execution time of the MoC when both model optimisation and grid coarsening are applied. Compared to the gains obtained with those two optimisation methods the results are underwhelming. Indeed, the execution time is reduced at most a factor two when using six processing cores. There are various reasons for this, two important of which have to do with grid coarsening. First, grid coarsening leads to a large reduction in the number of grid points so that the thread synchronisation overhead becomes significant. Second, the threads need to synchronise their execution at every base time step though a large fraction of the work is associated with higher grid levels.

Table 4 shows the parallel execution time of the MoC when only model optimisation is applied. In this case the use of multiple processor cores results in a greater reduction in the execution time. However, the results are still not ideal because of thread synchronisation overhead and insufficient memory bandwidth. The overhead plays a significant role with Model A because this model contains an order of magnitude less grid points (17,000) in the spatial dimension than Models B and C. The synchronisation overhead is less relevant for these latter two models, but here the total memory bandwidth of the workstation is insufficient to sustain more than a few cores. This becomes apparent when multiple, independent, single-core transient simulations with Model C are executed concurrently. The execution time then remains about 305*s* with two concurrent simulations, and then starts to climb to 551*s* with four, and 865*s* with six simulations. As these simulations are completely independent, their execution time is only limited by the memory bandwidth; synchronisation overhead plays no role.

Table 4: the parallel execution time (in seconds) of the MoC without grid coarsening.

Number of cores	Model A	Model B	Model C
1	572	4,860	305
2	426	2,399	163
4	279	1,964	127
6	211	1,906	127

7 CONCLUSIONS

The three optimisation methods presented here – grid/model optimisation, grid coarsening and parallel execution – can overcome the limitation that is imposed on the space-time grid by the Method of Characteristics (MoC). Consequently, they can reduce the execution time of the MoC significantly. The first two methods can have a very large impact, especially when the piping model combines a high level of detail with long pipeline sections. Indeed, the execution time associated with two such models has been shown to be reduced two orders of magnitude when model optimisation and grid coarsening are applied. When the

model contains less details the two methods still help, but the reduction in execution time tends to be less spectacular.

Model optimisation and grid coarsening affect the accuracy of the results. However, because they can lower the execution time significantly, it becomes feasible to check the accuracy automatically by running a second transient analysis with a smaller time step size and by comparing the results obtained with both analyses.

Parallel execution on multiple processor cores can certainly reduce the execution time of the MoC, but the gain is less spectacular, in general, when compared to the gain obtained with the first two optimisation methods. One reason is that those two methods tend to reduce the number of grid points to such a degree that the synchronisation overhead increases significantly with an increasing number of cores. Another reason is that the available memory bandwidth must be large enough to sustain all processor cores. This, unfortunately, is not always the case as has been demonstrated in the previous section. One possible solution would be to make better use of the memory cache associated with each core by updating the space-time grid in triangular-shaped chunks instead of one time step after another. This could not only lower the required memory bandwidth, but also the synchronisation overhead, and could therefore be an interesting subject of future research.

REFERENCES

(1) D.R. Butenhof. *Programming with POSIX threads*. Addison-Wesley Professional, 1997.

(2) M.H. Chaudhry. *Applied hydraulic transients*. Springer, New York, third edition, 1993.

(3) D.E. Goldberg and E.B. Wylie. Characteristics method using time-line interpolations. *Journal of Hydraulic Engineering*, 109(5):670–683, 1983.

(4) T. Johnson and K. Harathi. A simple correctness proof of the MCS contention-free lock. *Information Processing Letters*, 48(5):215–220, 1993.

(5) J.M. Mellor-Crummey and M.L. Scott. Algorithms for scalable synchronization on shared- memory multiprocessors. *ACM Transactions on Computer Systems*, 9(1):21–65, Feb 1991.

(6) W. Meng, Y. Cheng, J. Wu, Z. Yang, Y. Zhu, and S. Shang. GPU acceleration of hydraulic transient simulations of large-scale water supply systems. *Applied Sciences*, 9:91, 12 2018.

(7) A.K. Trikha. Variable time steps for simulating transient liquid flow by method of characteristics. *Journal of Fluids Engineering*, 99(1):259–261, Mar 1977.

(8) J.B. Turpin. Variable step integration coupled with the method of characteristics solution for water-hammer analysis, a case study. Technical report, NASA Marshall Space Flight Center, Huntsville, AL, 2013.

(9) G.Z. Watters. *Analysis and control of unsteady flow in pipelines*. An Ann Arbor science book. Butterworths, 1984.

(10) David C. Wiggert and Mark J. Sundquist. Fixed-grid characteristics for pipeline transients. *Journal of the Hydraulics Division*, 103(12):1403–1416, 1977.

(11) E.B. Wylie, V.L. Streeter, and L. Suo. Fluid transients in systems. Prentice Hall, 1993.