Parallel computing in conceptual sewer simulations

G. Burger, S. Fach, H. Kinzel and W. Rauch

ABSTRACT

Integrated urban drainage modelling is used to analyze how existing urban drainage systems respond to particular conditions. Based on these integrated models, researchers and engineers are able to e.g. estimate long-term pollution effects, optimize the behaviour of a system by comparing impacts of different measures on the desired target value or get new insights on systems interactions. Although the use of simplified conceptual models reduces the computational time significantly, searching the enormous vector space that is given by comparing different measures or that the input parameters span, leads to the fact, that computational time is still a limiting factor. Owing to the stagnation of single thread performance in computers and the rising number of cores one needs to adapt algorithms to the parallel nature of the new CPUs to fully utilize the available computing power. In this work a new developed software tool named CD3 for parallel computing in integrated urban drainage systems is introduced. From three investigated parallel strategies two showed promising results and one results in a speedup of up to 4.2 on an eight-way hyperthreaded quad core CPU and shows even for all investigated sewer systems significant run-time reductions.

Key words | CD3, conceptual model, integrated urban drainage modelling, multi-core, parallel computing, parallel strategy

G. Burger
Institute of Computer Science,
University of Innsbruck,
Technikerstr. 21A,
Innsbruck A-6020,
Austria
E-mail: *gregor.burger@uibk.ac.at*

S. Fach
W. Rauch
Unit of Environmental Engineering,
University of Innsbruck,
Technikerstr. 13,
Innsbruck A-6020,
Austria
E-mail: *stefan.fach@uibk.ac.at*;
wolfgang.rauch@uibk.ac.at

H. Kinzel
hydro-IT GmbH,
Technikerstr. 13,
Innsbruck A-6020,
Austria
E-mail: *Kinzel@hydro-it.com*

INTRODUCTION

Whilst in the past the processors (CPUs) got significantly more powerful (and thus faster), nowadays it is not the single CPU that can be improved further but instead the number of processors is increased. Multi-core systems are the future of desktop computing. Single thread performance is stagnating, but the number of cores is rising (Kongetira *et al.* 2005). To fully utilize that available computing power one needs to adapt algorithms to the parallel nature of these new CPU-architectures. Therefore a framework for integrated urban drainage models named CITY DRAIN 3 (CD3) was developed that exploits these additional computational resources of multi-core CPU-architectures. CD3 is a further development of the existing non multi-core capable CITY DRAIN 2 (Achleitner *et al.* 2007). The objective was to program a framework capable to be extended for all kind of integrated urban drainage models, like waste water treatment plant processes and river quality models. In a first step CD3 is limited to conceptual sewer systems.

The need of computational power for urban drainage simulations

Integrated urban drainage modelling (IUDM) combines the main subsystems of urban drainage systems (e.g. natural and urban catchments, sewers, receiving water bodies and waste water treatment plants) of the urban (waste) water cycle into one single model (Rauch *et al.* 2002; Butler & Schütze 2005). The main use of those models is to analyze how existing urban drainage systems respond to particular conditions (Butler & Davies 2004). Generally, deterministic models are used which always produce the same output for

doi: 10.2166/wst.2010.798

a specific set of input data. With these models engineers and scientists are able to fully reason about sewer system performance, discharge to the receiving water body and river water quality. Based on these integrated models, researchers are able to e.g. estimate long-term pollution effects (Rauch *et al.* 1998), optimize the behaviour of a system by comparing impacts of different measures on the desired target value or get new insights on systems interactions.

IUDM models are often formulated in a simplified manner applying e.g. hydrological routing instead of hydrodynamic wave equations to calculate the waste water transport in the sewer. Supplementary the originally complex conversion process of a rainfall hyetograph into a surface runoff hydrograph is often reduced to a simplified model with initial and continuing losses. The resulting effective rainfall hyetograph is then transformed into a surface runoff hydrograph using also simple models, e.g. (synthetic) unit hydrographs, time-area diagrams or reservoir models. Using these simplified conceptual models reduces the computing time significantly. A simulation run of a moderate sized system, over several decades with time steps in the order of minutes, only takes few seconds on recent computers. Nevertheless searching the enormous vector space, that is given by comparing different measures (e.g. spatial configuration of CSOs or tanks or stormwater infiltration devices) or that the input parameters span (e.g. for auto calibration or Monte Carlo simulation for uncertainty analysis), leads to the fact, that computing time is still a limiting factor, even with sophisticated searching algorithms. So speed is the limiting factor for an efficient use of existing auto calibration tools, such as CALIMERO (Kleidorfer *et al.* 2009*a*) or PEST (Doherty *et al.* 1994). Hence the development of simulation code that can be executed faster is still an important issue in integrated urban drainage modelling. For example Feyen *et al.* (2007) used parallel computing to implement a conceptual rainfall runoff model named LISFLOOD which simulates the river discharge in large drainage basins as a function of spatial information on topography, soils and land cover.

Parallel computing

Parallel Computing is a term used in computer science which describes a way to solve a computationally expensive problem by dividing it into subtasks. These subtasks are then distributed on different independent computational units and run concurrently (in parallel). Splitting up problems into parallel parts is called decomposition. There exists a huge variety of decomposition techniques, like recursive decomposition, data decomposition, exploratory decomposition and speculative decomposition (Grama *et al.* 2003).

The performance of parallel implementations is calculated using speedups. Speedup is defined as the ratio between the computational time needed for the best sequential algorithm divided by the computational time required for the parallel algorithm (Akhter 2006):

$$\text{speedup } (n_t) = \frac{\text{time}_{\text{bestseqalg}}}{\text{time}_{\text{parallelalg}(n_t)}} \tag{1}$$

Scalability of an algorithm is how far it can be parallelized and how well it works on more parallel entities. Linear scalability is the theoretically best achievable condition, it means adding n entities makes the run-time n-times better.

Different parallel computing entities exist depending on the parallel computing environment, e.g. cores of a multi-core CPU or servers in a clustered environment, for which the algorithm was designed for. Martins *et al.* (2001) give an overview and compare several common used parallel computing environments. Choosing the suited platform is generally critical and depends on several impact factors. Urban drainage simulations are characterized by many small computations with a high amount of dependencies. The architecture of single chip multiprocessors (commonly known as multi-core processors) fulfils the hope of having the best outcome with respect to parallel performance (Olukotun *et al.* 1996). Furthermore these multi-core systems are cheap available at the consumer market. With regard to the software OpenMP (OpenMP Architecture Review Board 2008) and the standardized portable operating system interface [for Unix] threads (POSIX-threads) were used to implement the parallel strategies. POSIX is set of standardized libraries and tools that allow writing portable applications.

Communication and synchronization in parallel computing is the exchange of information between concurrent tasks. Depending on the need of synchronization, a programmer can choose between locks, semaphores, monitors, conditional-variables, messages, fences and barriers.

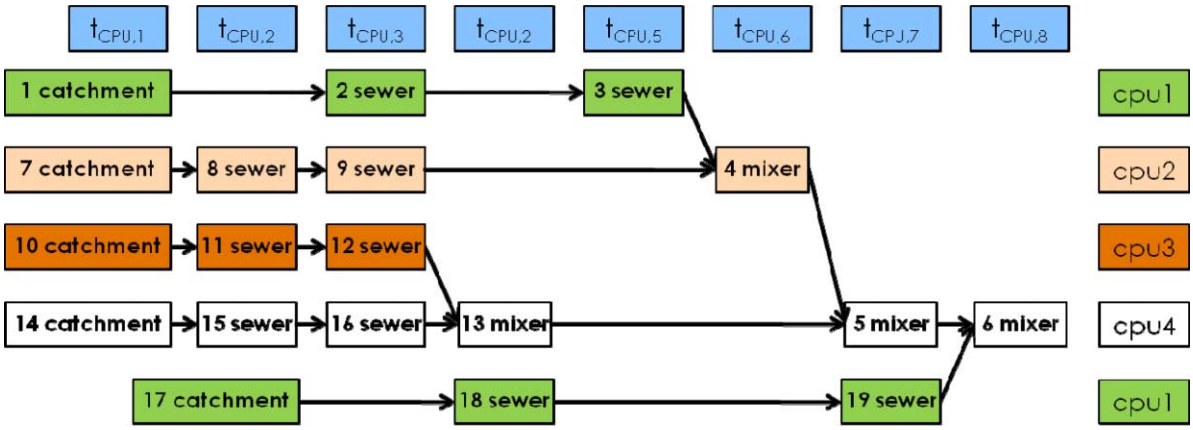

Figure 1 | Flow parallel strategy realized on a quad core CPU.

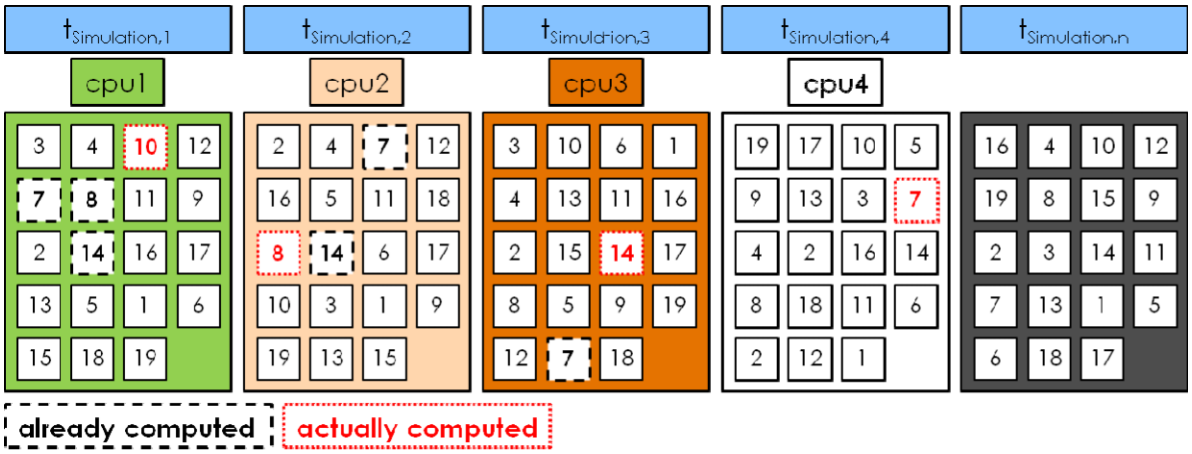

Figure 2 | Pool pipeline strategy realized on a quad core CPU.

Synchronization errors are subtle, hard to find and hard to fix, because a near infinite number of situations can occur depending on the race of the threads. These errors are mostly unknown in classical sequential programming and have names like "race condition", "dead locks" and "live locks" (Akhter 2006).

Finding parallel strategies with good communications and avoiding concurrency errors are critical for high performance of parallel systems.

METHODS

In this chapter the parallel strategies developed for conceptual sewer systems are described that were used to accelerate the computation of the processes. For demonstration purposes the conceptual sewer system consisted of a reduced set of nodes: a catchment for the constant dry weather flow, a sewer for the routing process, a mixer and a file-out node for writing the simulation results into a file. In total three strategies were found. The first one is the flow parallel strategy (FPS) which uses a data parallel decomposition. The second, pool pipeline strategy (PPS) and third one, ordered pipeline strategy (OPS) are based on a pipelined method in which the nodes are pipelined through the threads.

Flow parallel strategy

The flow parallel strategy combines data parallel and task parallel model described in Grama *et al.* (2003). The flow chart of Figure 1 illustrates the depending computations of

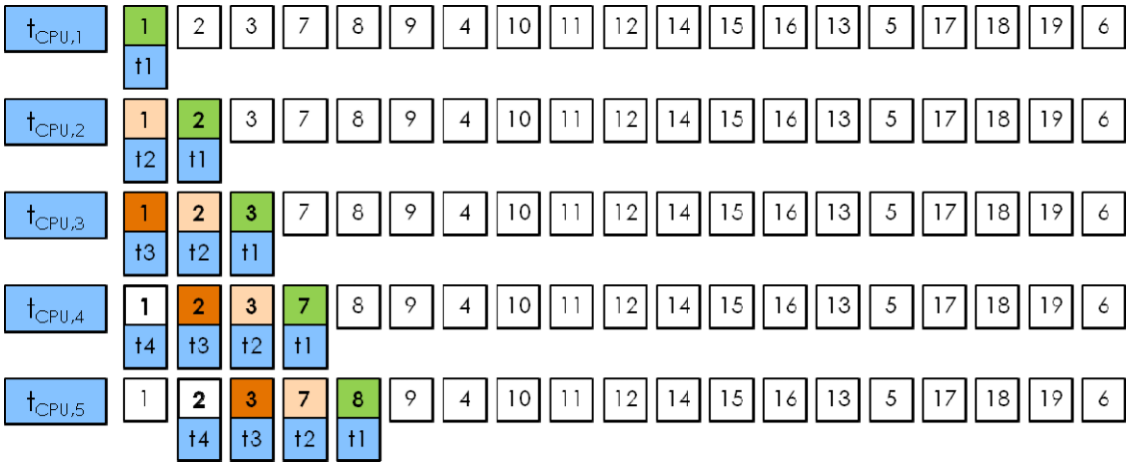

Figure 3 | Ordered pipeline strategy realized on a quad core CPU.

the sewer system for one simulation time step. On each input flow originated by a catchment a new thread is started (see Figure 1). Each sequential node after the input node is then calculated by this thread. If several flows are merged due to a mixer node all threads except one are shut down. This thread continues to compute the merged downstream flow.

At the mixer node which functions as a junction, synchronization is required. Each mixer node contains a counter starting with the number of input flows, i.e. number of connected links. If a thread reaches a mixer node its counter is decremented by one. The thread which decrements the counter to zero continues the calculation of the nodes following the mixer node downstream.

The number of threads is limited by the number of input flows. At each mixer node at least two flows are merged. This effect impacts the possible parallel streams to be calculated, i.e. the more the flow of sewer sections downstream is already calculated the less parallelization is possible. Due to this fact this strategy is not able to fully utilize all cores over the entire sewer system.

The arrows in Figure 1 are symbolizing the data transfer between the nodes. The ones which start and end in different colours are data transfers between the CPU cores. The ones which start and end in the same colour are data transfers in the cores. Data transfers between CPU cores causes CPU flushes and memory stalls which are expensive with regard to CPU cycles (Drepper 2007). The advantage of this strategy is that the data needed for downstream computations is more likely to remain in the cores, as can be seen in Figure 1.

Pool pipeline strategy

The second strategy starts a thread per simulation time step (see Figure 2). The goal for the thread is to get all nodes of the sewer system executed. A node can execute the implemented algorithms, e.g. Muskingum routing, if all its upstream nodes are in the same time step. Each thread maintains a private set of nodes, called a pool, which were not yet processed. The next node to be processed is chosen randomly from this pool and gets executed if the dependencies are figured out. If the pool is empty the simulation time step is finished and the thread computing the step can be reallocated for the next time step.

Ordered pipeline strategy

The goal of the OPS was to get rid of the non deterministic behaviour of the PPS. As in the PPS a thread is responsible for a time step. Instead of randomly choosing the next node to be executed, the execution order is determined prior the simulation run (see Figure 3). The execution order is calculated by applying a topological sorting algorithm onto the sewer system (Kahn 1962). The topological sorting assures that the nodes are executed with all dependencies satisfied. Each thread is connected to the thread executing the next time step by a first in first out (FIFO) queue. If a

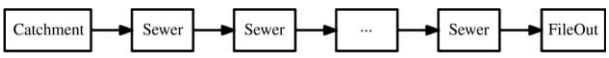

Figure 4 | Artificial testing system consisting of a sequential line of sewer sections.

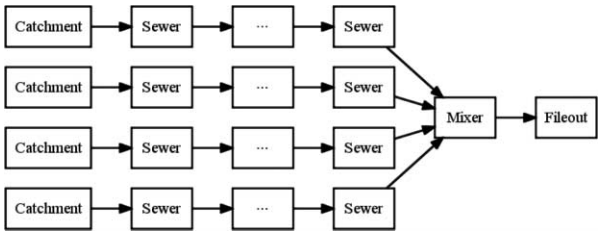

Figure 5 | Artificial testing system consisting of a several sequential sewer sections in parallel.

node is finished in time step dt it is fed into the queue of the time step $dt + 1$. An empty queue means that the responsible thread is finished with this time step. The thread can then move on to calculate a new pending time step.

In the pipelined strategies PPS and OPS the number of threads to be utilized is the highest number of sequential nodes in the sewer system. OPS can even run all nodes in parallel by intelligently buffering the nodes data exchange. This means that the pipelined strategies are able to handle more cores than the FPS. The disadvantage of these pipelined strategies is that every single data exchange between nodes is carried out between threads and therefore CPU cores.

URBAN DRAINAGE SYSTEM USED FOR TESTING

CD3 is a reimplementation of the MATLAB based CITY DRAIN software tool (Achleitner *et al.* 2007) with a focus on performance. Therefore in CD3 the same mathematical models are used as in CITY DRAIN. On the other hand the benchmark of CD3 against CITY DRAIN is meaningless due to the fact that CD3 has been realised outside the MATLAB environment which has a significant impact on the computational performance. On the other hand CD3 cannot be benchmarked with other software tools because of the different model approaches, i.e. runoff generation, surface routing and sewer flow implemented. Artificial testing systems were generated to highlight the advantages and disadvantages of the different parallel strategies. Furthermore, one converted and adapted system of the city of Innsbruck was used to show how well the parallel implementation can handle real-world scenarios. The software was benchmarked to demonstrate the increase of computational performance with the above mentioned testing systems.

Artificial urban drainage systems

Generally the structure of urban drainage systems complies with an inverted tree. The artificial testing systems chosen differ in complexity and degree of reality. The first artificial testing system is a sequential line of sewer sections starting with an input node and ending in a file out node delivering the results in a CSV format (see Figure 4). With this testing system it is possible to demonstrate the benefits of parallelization even for simple conceptual models. The sequential nature of this testing system is well suited to show the communication overhead of the FPS. The second testing system is used to point out the theoretical upper

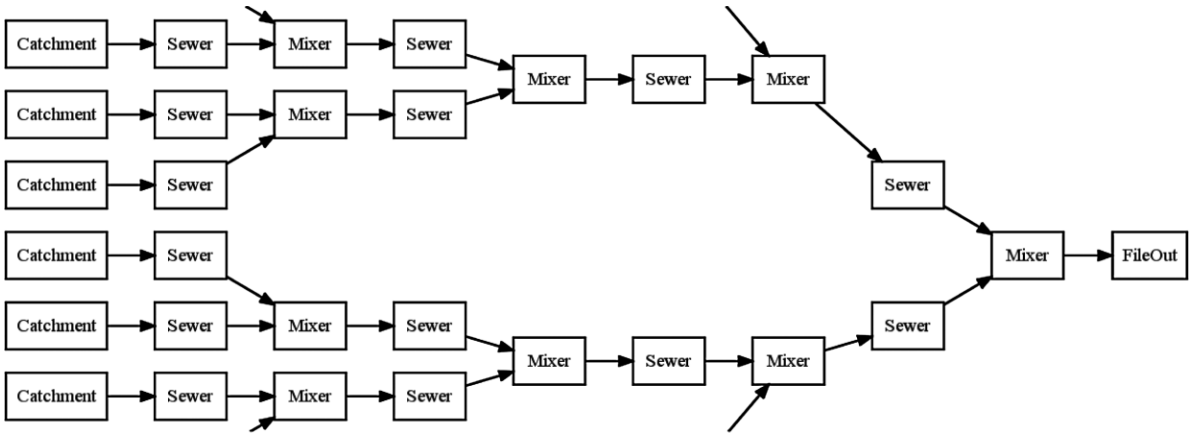

Figure 6 | Detail of the artificial testing system with a binary tree structure.

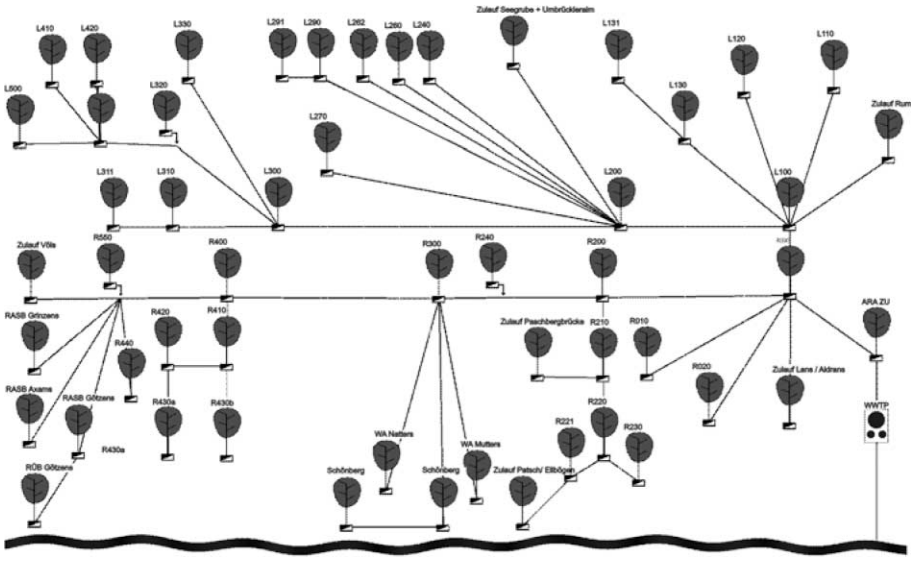

Figure 7 | Sewer system of the city of Innsbruck implemented in KAREN (Kleidorfer *et al.* 2009*b*).

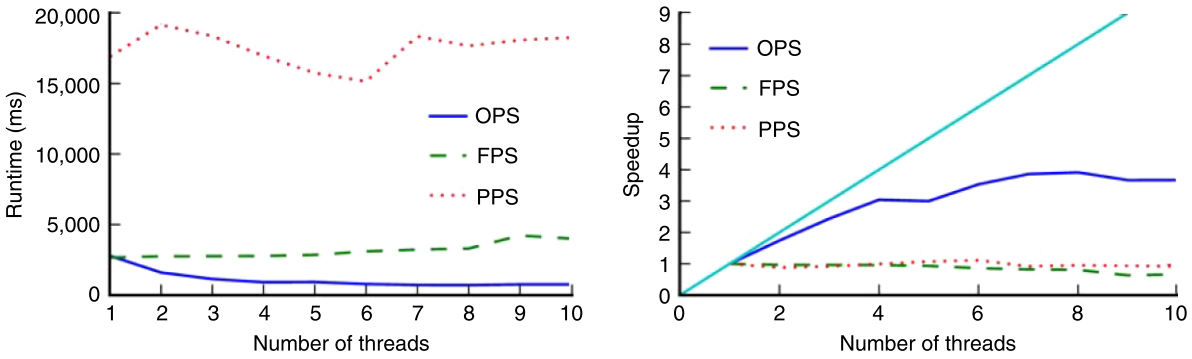

Figure 8 | Results of 1000 sequential sewer sections.

bounds reached by the FPS. Several sequential sewer sections in parallel are mixed together at the end (see Figure 5). The third testing system is chosen due to its close scheme to natural drainage structures which have the shape of an inverted binary tree (see Figure 6). Two sewers sections are mixed up and connected to a sewer section downstream.

Real world case study of Innsbruck

To have one real-world example as testing system, the sewer system of the city of Innsbruck used by Kleidorfer *et al.* (2009*b*) was converted into the native CD3 input format. This combined sewer system consists of 53 catchments, a total runoff effective area of 915 ha and a total CSO tank volume of 5,100 m^3. In the city of Innsbruck live 165,000 population equivalents (PE) with a daily dry weather flow (DWF) of 200 l/(PE d) Figure 7.

Looped systems

Although looped systems can occur in real-world structures the framework of CD3 was evaluated firstly on systems based on more or less tree like structures. In principle the algorithms used in CD3 should be capable to compute looped systems by breaking up the circular structure.

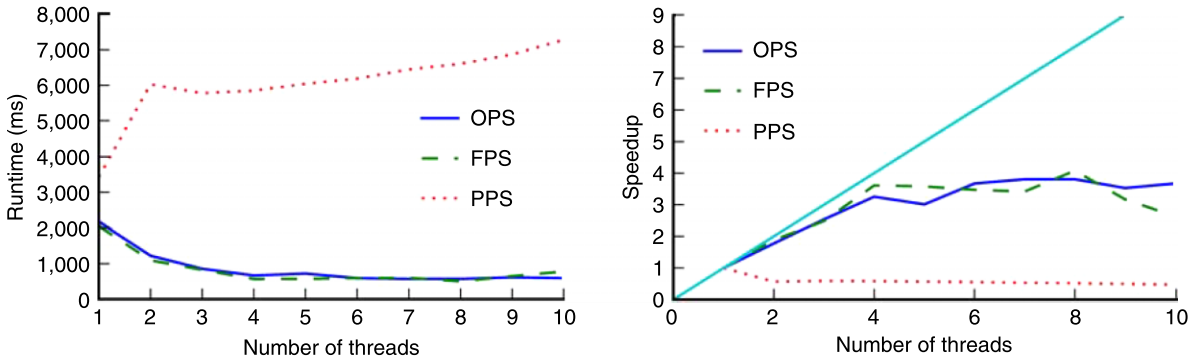

Figure 9 | Results of eight parallel streams each 100 sewer sections long.

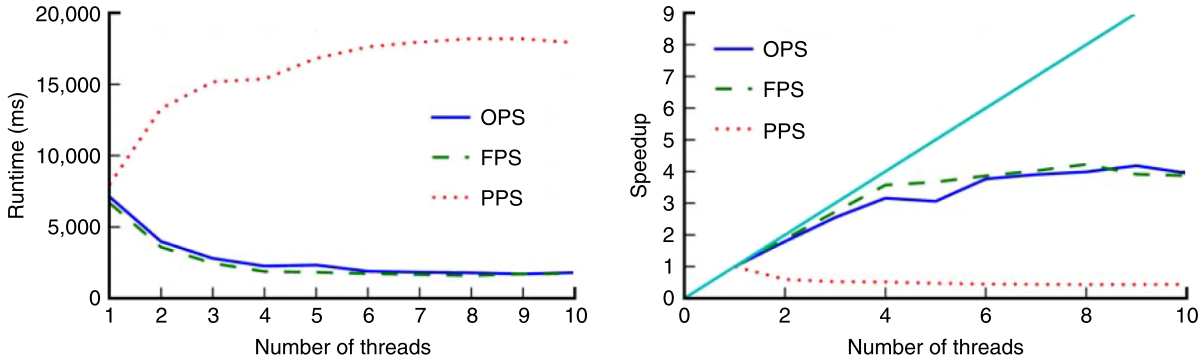

Figure 10 | Results of a binary tree system with ten generations.

RESULTS AND DISCUSSION

Benchmark environment

The hardware on which CD3 was benchmarked is an Intel (R) Core(TM) i7 CPU 920 @ 2.67 GHz equipped with four hyperthreaded cores. The software was compiled with the Intel (R) Compiler Suite v11. Each testing system was run with a different count of allowed threads from one to ten. As benchmark results the minimum of four simulation runs was taken. The processor has eight virtual threads, two hyperthreaded per core. Due to parallel strategies are based on structural decomposition and not on time decomposition the simulation time of the runs was two hours with five minutes time steps. The results are presented using two kinds of diagrams. The diagrams on the left depict the total time the simulation takes. The diagrams on the right show the speedups. For speedup calculations the single thread performance is equivalent to the best sequential algorithm as is apparent from Equation (1). The x-axis is always the number of threads that a strategy is allowed to use.

Computational calculation time for artificial urban drainage systems

The testing system of Figure 8 consists of 1000 sequential sewer sections. Despite the fact that this sewer system is intrinsic sequential the implementation of the OPS seems to gain a speedup using more threads. Although the implementation of the FPS cannot scale in this testing system the communication overhead in such situations is insignificant.

Figure 9 shows the results of the parallel testing system with eight parallel streams each 100 sewer sections long. This testing system should give particularly good results for the FPS, but also the implementation of the OPS seems to scale pretty well to the four available cores. Because of the

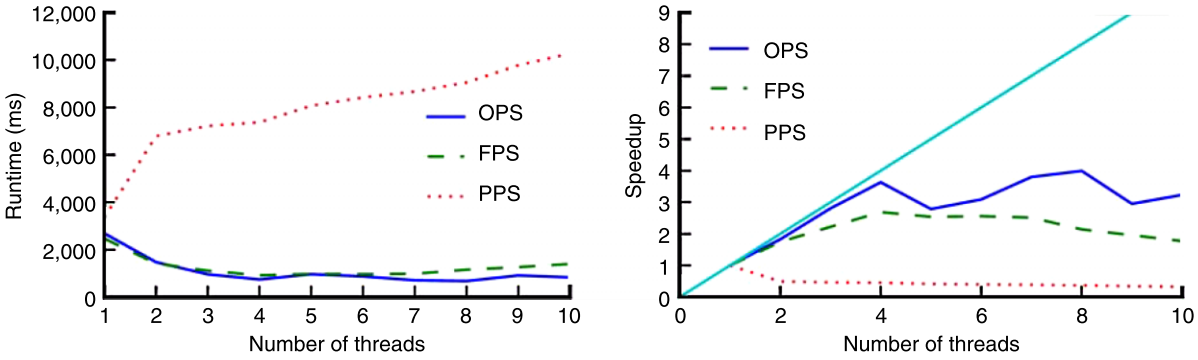

Figure 11 | Results of the real-world sewer system of the city of Innsbruck.

unusable bad computational performance of the PPS the results are not explained further in the discussion chapter.

In Figure 10 the results of the binary tree with ten generations and 2047 nodes are depicted. This tree offers enough parallel streams that the FPS and OPS are able to reduce the run-time on more cores, although the FPS has a slight edge ahead. Even with a smaller testing system consisting of two generations and seven nodes (not depicted here) the OPS showed significant time reductions by using more cores.

Computational calculation time for real-world case study of Innsbruck

Figure 11 shows the results from the real-world case study of the sewer system of Innsbruck. The ordered pipelined strategy performs obviously better than the other strategies. On the right hand side of Figure 11 the implementation of the ordered pipelined strategy is given separately.

CONCLUSION AND OUTLOOK

This paper demonstrates the possibility of parallel computing to decrease the run time of urban drainage simulations. Three parallel strategies were tested by means of different benchmark systems. The implementation of the PPS was the one which performed worst. No speedup could be seen in a single test. The randomized characteristic of the PPS was identified to be the weak spot. Getting rid of this randomization resulted in the well performing OPS. The OPS achieved good results throughout all testing systems, from small binary tree to sequential and real-world testing systems. In the sequential testing system the implementation of the OPS had the highest speedup of 3.9 on eight threads. In the tree testing system with ten generations the FPS and OPS were on par with a maximum speedup of around 3.8. In the real-world sewer system of Innsbruck OPS gained a maximum speedup of 4.1.

The sewer testing systems used in the benchmarks included also worst case scenarios to emphasize the weak spot of the parallel strategies, i.e. their worst performance. Therefore it can be concluded that OPS due the good results on all testing systems should also perform good on other sewer systems not investigated in this paper.

OPS should benefit from an optimized lock-free queue implementation as described in Fober *et al.* (2001) and Fober *et al.* (2002) because this strategy has a high usage of queues for communication. Although the FPS performed well on some testing systems, the performance was lower than expected. A parallel computing trace tool revealed high amounts of lock usages implicitly emitted by OpenMP.

REFERENCES

Achleitner, S., Möderl, M. & Rauch, W. 2007 CITY DRAIN© – An open source approach for simulation of integrated urban drainage systems. *Environ. Modell. Softw.* **22**(8), 1184–1195.

Akhter, S. 2006 *Multi-core Programming: Increasing Performance Through Software Multi-threading*. Intel Press, USA.

Butler, D. & Davies, J. 2004 *Urban Drainage*. Spon Press, London.

Butler, D. & Schütze, M. 2005 Integrating simulation models with a view to optimal control of urban wastewater systems. *Environ. Modell. Softw.* **20**(4), 415–426.

Doherty, J., Brebber, L. & Whyte, P. 1994 PEST manual. In: *Watermark Computing, Corinda, Australia*.

Drepper, U. 2007 What every programmer should know about memory. In: *Proceedings of the Red Hat Summit*, Nashville, USA, 21.11.2007.

Feyen, L., Vrugt, J., Nualláin, B., van der Knijff, J. & De Roo, A. 2007 Parameter optimisation and uncertainty assessment for large-scale streamflow simulation with the LISFLOOD model. *J. Hydrol.* **332**(3–4), 276–289.

Fober, D., Letz, S. & Orlarey, Y. 2002 Lock-Free Techniques for Concurrent Access to Shared Objects. In: *Proceedings of the JIM Actes des Journes d'Informatique Musicale*, Marseille, France, pp. 143–150.

Fober, D., Orlarey, Y. & Letz, S. 2001 *Optimised Lock-Free FIFO Queue*. Technical Report-01-01-01 Grame.

Grama, A., Gupta, A. & Karypis, G. 2003 *Introduction to Parallel Computing*, 2nd edition. Addison Wesley Pub Co Inc, Boston, MA.

Kahn, A. B. 1962 Topological sorting of large networks. *Commun. ACM* **5**(11), 558–562.

Kleidorfer, M., Leonhardt, G., Mair, M., McCarthy, D. T., Kinzel, H. & Rauch, W. 2009*a* CALIMERO–A model independent and generalized tool for autocalibration. In: *Proceedings of the 8th International Conference on Urban Drainage Modelling*, Tokyo, Japan, 7. – 11.09.2009.

Kleidorfer, M., Möderl, M., Fach, S. & Rauch, W. 2009*b* Optimization of measurement campaigns for calibration of a conceptual sewer model. *Water Sci. Technol.* **59**(8), 1523–1530.

Kongetira, P., Aingaran, K. & Olukotun, K. 2005 Niagara: a 32-way multithreaded sparc processor. *IEEE Micro* **25**(2), 21–29.

Martins, S. D. L., Ribeiro, C. C., Rodriguez, N. 2001 Parallel Computing Environments. In: *Proceedings of the Handbook of Applied Optimization*.

Olukotun, K., Nayfeh, B. A., Hammond, L., Wilson, K. & Chang, K. 1996 The case for a single-chip multiprocessor. *Proceedings of the IEEE Computer*, pp. 2–11.

OpenMP Architecture Review Board 2008 OpenMP application program interface 3.0. http://www.openmp.org

Rauch, W., Aalderink, H., Krebs, P., Schilling, W. & Vanrolleghem, P. 1998 Requirements for integrated wastewater models–driven by receiving water objectives. *Water Sci. Technol.* **38**(11), 97–104.

Rauch, W., Bertrand-Krajewski, J. L., Krebs, P., Mark, O., Schilling, W., Schütze, M. & Vanrolleghem, P. A. 2002 Deterministic modelling of integrated urban drainage systems. *Water Sci. Technol.* **45**(3), 81–94.