Parallel Computing in Water Network Analysis and Leakage Minimization

José M. Alonso[1], Fernando Alvarruiz[1], David Guerrero[1], Vicente Hernández[2],
Pedro A. Ruiz[1], Antonio M. Vidal[3]
Fernando Martínez[4], Juan Vercher[5], Bogumil Ulanicki[6]

Keywords: High performance computing, hydraulic simulation, water quality simulation, leakage simulation, leakage reduction, EPANET.

Abstract.

In this paper, a parallel computing based software demonstrator for the simulation and leakage minimization of water networks is presented.
This demonstrator, based on EPANET package, tackles three different types of problems making use of parallel computing. First, the solution of the hydraulic problem is treated by

[1] Ph.D. student, Parallel Computing Group, Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia. C/ Camino de Vera s/n, 46022, Valencia. Spain.

[2] Professor, Parallel Computing Group, Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia. C/ Camino de Vera s/n, 46022, Valencia. Spain. Phone: +34 96 387 7356. Fax: +34 96 387 7359. e-mail: vhernand@dsic.upv.es.

[3] Professor, Parallel Computing Group, Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia. C/ Camino de Vera s/n, 46022, Valencia. Spain.

[4] Professor, Departamento de Ingeniería Hidráulica y Medio Ambiente, Universidad Politécnica de Valencia, C/ Camino de Vera s/n, 46022, Valencia. Spain.

[5] Research fellow, Departamento de Ingeniería Hidráulica y Medio Ambiente, Universidad Politécnica de Valencia, C/ Camino de Vera s/n, 46022, Valencia. Spain.

[6] Professor, Water Software Systems, De Monfort University, The Gateway, Leicester, LE1 9BH.

means of the Gradient Method. The key point in the parallelization of the method is the solution of the underlying linear systems, which is carried out by means of a multifrontal Cholesky method. Second, the water quality simulation problem is approached by using the Discrete Volume Element Method. The application of parallel computing is based on dividing the water network in several parts using the Multilevel Recursive Bisection graph partitioning algorithm. Finally, the problem of leakage minimization using Pressure Reducing Valves (PRVs) is approached. This results in the formulation of an optimization problem for each time step, which is solved by means of Sequential Quadratic Programming. Since these sub-problems are independent of each other, they can be solved in parallel.

Introduction.

The *High Performance Computing* (HPC) technology, which is nowadays synonymous with parallel computing (several processors working together to solve the same problem) (Kumar et al. 1994), is a well-established technology. However, its use has been restricted until recently to large enterprises and universities. This was due to the high cost of HPC hardware, which has represented a barrier to its adoption by smaller companies. The advent of low cost multiprocessor systems offers now a powerful choice for industry at large. HPC makes possible, by means of the interconnection of PCs or workstations, to reach a computational power similar to that of the supercomputers at lower costs.

As will be described in this paper, the objective of HIPERWATER (ESPRIT project 24003; http://www. hiperttn.upv.es/hiperwater) has been to build a software demonstrator showing that HPC technology reduces the time spent on water network analysis processes, enabling the acceleration of tasks such as simulation and leakage control. EPANET (Rossman 1993a), a well-known water network simulation package, has been used as the starting point for the demonstrator.

The first technical direction of HIPERWATER was toward simulation processes, specifically hydraulic analysis and water quality modeling. The hydraulic analysis problem involves the solution of a non-linear system of equations using a parallelized version of *Todini's gradient algorithm*. Water quality is simulated using the *Discrete Volume Element Method* (DVEM), which was also modified to a parallel algorithm.

HIPERWATER's second direction moved beyond simulation to a methodology for leakage minimization by finding the optimal pressure reducing valve settings. This optimization problem, formulated as series of non-linear programming problems, is solved by a parallel algorithm based on *Sequential Quadratic Programming*.

Throughout the paper results are shown with performance of the parallel algorithms. These results have been obtained on a cluster of Pentium-Pro PCs linked with a Fast-Ethernet network. Operating system is Windows (95, 98, NT), and MPI (Message Passing Interface) (Dongarra et al. 1996) communication library is used. Each PC has 64 Mb RAM and runs at 200 MHz.

Background.

Although computer performance has steadily increased during the last few decades, the limit imposed by the speed of light indicates that this situation will not last forever. One way to circumvent this difficulty is to use an ensemble of interconnected processors. This is also a cost-effective way of obtaining an increase in raw computational power. As pointed out in (Kumar et al. 1994), the cost-performance ratio for computers can be represented by a curve similar to that shown in Figure 1. For low prices, performance increases significantly with cost until a saturation point is reached. Beyond this point, small increases of performance come at a high cost. Parallel computing is an economic alternative to obtain extra performance at a low cost.

Parallel scientific and engineering computing is becoming of paramount importance in several industrial applications, especially when the solution of large and complex problems must be done in a reduced time. Computational fluid mechanics, structural engineering, computational chemistry, electronic and electro-magnetic circuits, signal and image processing, are some engineering areas where parallel computing has been successfully applied (Palma et al. 1998).

In the context of water networks, the use of Geographical Information Systems (GIS) facilitates constructing detailed realistic network models. The analysis of these large networks however is extremely complex. As a consequence, more powerful computing resources are needed hence the interest in the use of parallel computing.

In addition, it is not unusual for the water quality simulations to require a long simulation period to reach equilibrium due to a lack of understanding of the initial water quality conditions in the system. As the simulation advances, the quality conditions tend to a cyclical pattern that repeats over 24-hour periods. Thus, a water quality analysis may extend over a simulation time (not computing time) of several weeks. In this context the computing time can be important, and the use of parallel computing is justified.

Finally, water industry desires to solve water-network optimization problems that are more complex and computationally-intensive than simulation problems. Optimization problems can be formulated for design and operational purposes. In the latter case, a typical problem faced by water companies is that of finding pump, valve and water production schedules to satisfy user demands and to provide feasible operational conditions at the least cost. In many networks, leakage is significant but can be reduced by proper operational decisions. In any case, water network optimization problems are one of the areas that can benefit from the introduction of parallel computing.

Indeed, parallel computation can help water companies increase their productivity by substantially reducing the time to analyze networks. This technology can also allow more complex problems to be solved that are not technologically possible today using traditional (non-HPC) approaches.

Hydraulic Simulation.

The equations which model the water networks (and piping networks in general) are non-linear and therefore require an iterative solution, i.e. the problem is reduced to the solution of a sequence of systems of linear equations. One of the most effective solution methods is the *Hybrid* or *Gradient Method*, (Hamam and Brameller 1971; Todini 1979; Todini and Pilati 1987). This method consists in the application of the Newton-Raphson technique, both in terms of nodal heads and pipe flows, to the simultaneous solution of the equations of conservation of mass and energy. It is represented by the following coupled equations that need to be solved iteratively:

i) Nodal head update:

$$H^{(k+1)} = -\left(A_{21}G^{-1}A_{12}\right)^{-1}\left(A_{21}G^{-1}\left(A_{11}Q^{(k)} + A_{10}H_0\right) - \left(A_{21}Q^{(k)} - q\right)\right), \tag{1}$$

ii) Pipe flow update:

$$Q^{(k+1)} = \left(I - G^{-1}A_{11}\right)Q^{(k)} - G^{-1}\left(A_{12}H^{(k+1)} + A_{10}H_0\right), \tag{2}$$

where:

k: Gradient Method iteration counter,

H: piezometric head, column vector of *NN*–*NS* elements,

NN: total number of nodes,

NS: number of source nodes (reservoirs or tanks),

H_0: piezometric head of source nodes, column vector of *NS* elements,

Q: flowrate, column vector of *NP* elements,

NP: number of network links (pipes, valves or pumps),

q: known nodal consumption, column vector of *NN–NS* elements,

A_{12}: ($NP \times NH$) edge-to-non-source-node connectivity matrix,

$NH = NN - NS$, number of non-source nodes,

$A_{21} = A_{12}^T$ (transpose of matrix A_{12}),

A_{10}: ($NP \times NS$) edge-to-source-node connectivity matrix,

$A_{11} = \mathrm{diag}(\alpha_i |Q_i^k|^{n-1} + \beta_i / Q_i^k)$, ($NP \times NP$) diagonal matrix,

α_i, β_i: known characteristic parameters of the link *i*,

n: headloss-flow exponent,

$G = N A_{11}'$,

$N = \mathrm{diag}(n)$, ($NP \times NP$) diagonal matrix,

$A_{11}' = \mathrm{diag}(\alpha_i |Q_i^k|^{n-1})$, ($NP \times NP$) diagonal matrix,

I: identity matrix.

Equation (1) represents a linear system of *NH* equations in the unknown piezometric heads, and it can be expressed in the standard format of a linear system:

$$\left(A_{21} G^{-1} A_{12}\right) H^{(k+1)} = -A_{21} G^{-1} \left(A_{11} Q^{(k)} + A_{10} H_0\right) + \left(A_{21} Q^{(k)} - q\right). \tag{3}$$

System (3) is a standard linear problem $A\ x = b$, where the coefficient matrix is sparse, symmetric and positive definite.

In the remainder of this section, a parallel algorithm for hydraulic simulation will be described. This algorithm is based on the EPANET hydraulic simulation method which is outlined in the following subsection.

Hydraulic Simulation Process.

A scheme of the algorithm used in EPANET to perform the hydraulic simulation can be seen in Figure 2.

After an initialization phase, loop *A* simulates the network hydraulic behavior over an extended period of time, iterating for successive time steps for the duration of the simulation period. The most important and computationally expensive task is loop *B*, in which the system of non-linear equations is solved. A linear system of equations is solved by means of the *Cholesky factorization* in each iteration of the loop, until convergence is achieved.

Parallel Hydraulic Simulation.

To reduce computation times, the major effort is to apply an efficient parallel algorithm to solve the sparse symmetric positive definite linear systems of equations (loop *B*).

In addition, the remaining steps of the hydraulic simulation have also been parallelized. One of the processors is in charge of reading the input file and broadcasting most of its contents to the rest of processors. Given the network data, the coefficient matrix structure is generated following the rowwise block-striped partitioning method (Kumar et al. 1994), while the parallel solution of the linear equations is performed by means of a *Multifrontal Cholesky Factorization* (Liu 1992; Heath et al. 1991). The problem consists of solving the system $Ax = b$, where A is a sparse, symmetric and positive definite matrix. Cholesky factorization is composed of four consecutive phases: fill-reducing ordering, symbolic factorization, numerical factorization and solution of triangular systems. During the ordering phase, a permutation matrix P is computed so that the matrix PAP^T will incur a minimal fill-in during the factorization phase. During the symbolic factorization phase the non-zero structure of the triangular Cholesky factor L is determined. The role of the symbolic factorization phase is to increase the performance of the numerical factorization phase. The necessary operations to compute the values in L that satisfy $PAP^T = LL^T$, are performed during the phase of numerical factorization. Finally the solution to $Ax = b$ is computed by solving two triangular systems, $Ly = b'$ followed by $L^Tx' = y$, where $b' = Pb$ and $x' = Px$. The former system is solved by means of a forward elimination algorithm, while the solution of the latter involves a backward substitution process. The final solution, x, is obtained using $x = P^Tx'$.

In the case of hydraulic simulation, the ordering and symbolic factorization stages need to be performed only once as an initialization process prior to the simulation, because the matrix structure does not change in the different time steps.

In essence, the algorithms are driven by the *elimination tree* of matrix A (see Figure 3). A set of consecutive nodes in the *elimination tree*, each with only one child, is called a supernode. Nodes in each supernode are joined to form a *supernodal elimination tree*, which is a binary tree with top *log p* levels, where p is the number of processors involved. In order to factorize the sparse matrix in parallel, portions of the elimination tree are assigned to processors using a *subtree-to-subcube* assignment strategy. The topmost supernode is assigned to all processors. The two subtrees of the root are assigned to subcubes of *p/2* processors each. Each subtree is further partitioned recursively using the same strategy. Thus, the *p* subtrees at a depth of *log p* supernodal levels are each assigned to individual processors. Figure 3 shows the elimination tree of a sparse symmetric matrix, with the subtree-to-subcube mapping onto 8 processors ($\mathbf{P}_0$ to $\mathbf{P}_7$). The non-zero elements in the original coefficient matrix are denoted by "×" and fill-in elements are denoted by the symbol "o".

Fill-Reducing Ordering.

The serial approach of the *Multilevel Nested Dissection* (MND) algorithm is applied to obtain a fill-reducing ordering (Bui and Jones 1993; Heath et al. 1991) of the coefficient matrix.

The ordering scheme is a key factor determining the efficiency of both serial and parallel implementations. In the first case, the objective is to minimize the memory requirements and the number of operations during the factorization of the coefficient matrix. In the parallel case, it also has to be taken into account that the degree of parallelism during the factorization must be high.

As an example, the *Minimum Degree* (MD) algorithm (used in EPANET) has been found to produce very good orderings on serial computers. However, nested-dissection-based orderings lead to more concurrence and better load balance during parallel factorization than MD. Moreover, MND is usually faster than MD for large matrices and, whereas MD is serial in nature, MND can be effectively parallelized.

In fact, experiences have shown that not only is MND preferable to MD in parallel implementations, but also in the sequential case, especially when large matrices are encountered. MND reduces the number of non-zero elements in the matrix factorization, accelerating the phases of factorization and triangular systems solution.

The MND algorithm is based on the multilevel graph bisection partitioning technique (Bui and Jones 1993). Briefly, the multilevel graph bisection involves gradual coarsening of a graph to a few hundred vertices, then partitioning this smaller graph, and finally, projecting the partitions back to the original graph by gradually refining them. Further details of this will be given in the section devoted to the water quality simulation.

Figure 4 (a) shows the non-zero structure of the coefficient matrix *A* corresponding to test network 1 (see Table 1 for the features of the test networks); Figure 4 (b) presents the corresponding structure of the Cholesky lower triangular matrix *L* when the MD ordering method is used; finally, Figure 4 (c) shows the structure of *L* when the MND method is employed. It can be clearly appreciated that MND results in a smaller number of operations during numerical factorization, due to the smaller number of non-zero elements.

Symbolic Factorization.

In this second stage, the non-zero structure of the triangular Cholesky factor *L* is computed. In the parallel algorithm applied, the supernodal elimination tree is distributed among the processors using the *subtree-to-subcube* mapping and the columns of the coefficient matrix of each supernode are distributed using a *bitmask based block-cyclic strategy*. The structure of *L* is obtained in a bottom-up fashion, where the non-zero structure of the leaf nodes is computed first, and sent upwards in the elimination tree to the processors that store the supernode in the upper level. These processors compute the non-zero structure of their supernode and join it with the structure received from their children nodes. Moreover, the structure of two nonoverlapping subtrees can be computed concurrently.

Numerical Factorization.

In the third stage of the direct solution method, a parallel numerical Cholesky factorization is performed (using a *parallel multifrontal algorithm* (Heath et al. 1991, Liu 1992)) and the values in *L* are computed.

Broadly speaking, there are two levels of parallelism in this numerical phase. First, the processing of the frontal/update matrices associated with two non-overlapping subtrees can be treated as completely independent tasks. Second, since the amount of computation involved in the assembly and partial elimination of a frontal matrix can be quite substantial, a further subdivision of the task into smaller subtasks must be considered.

Here, the supernodal elimination tree is distributed among the processors following the same *subtree-to-subcube* strategy mentioned in the symbolic factorisation phase. The frontal matrix of each supernode is assigned to each processor using the same *bitmask based block-cyclic strategy*. With this distribution, the extend-add operations of the algorithm are computed in parallel and each processor exchanges approximately half of its data with its partner from the other subcube. The parallel factor operation at each supernode is based on the dense column Cholesky factorisation.

Triangular Systems Solution.

Finally, two triangular systems (first $Ly = b'$, followed by $L^T x' = y$) must be solved by using parallel approaches of forward elimination and backward substitution respectively, where $b' = Pb$ and $x' = Px$. The final solution x is then given by $x = P^T x'$. The parallel version of the algorithm is again governed by the supernodal elimination tree and it uses the same *subtree-to-subcube* mapping and the same two-dimensional distribution of the matrix L as mentioned in both factorizations. Whereas for the forward elimination the computation proceeds in a bottom-up fashion, the parallel backward substitution proceeds from the top supernode of the tree down to the leaf. In both triangular systems, two dimensional pipelined dense algorithms are used.

Experimental results.

Three water networks (Test 1, 2 and 3) have been used to test the algorithms of hydraulic and water quality simulation. In order to show the computing time reduction by using HPC techniques, large water networks are needed. From a hydraulic point of view, these networks are simple. They are fed by gravity pipes from one or more variable head tanks. Demand modulation is classified into five previously fixed patterns distributed among all the network nodes. Table 1 shows some summary information for the networks.

Figure 5 presents the time spent on the hydraulic simulation for test networks 1, 2 and 3, using both EPANET on a single processor, and HIPERWATER on 1, 2 and 4 processors (HW-1p, HW-2p and HW-4p, respectively). Due to the design of the parallel algorithm, the number of processors must be a power of 2.

It can be seen that HIPERWATER on one processor is slightly faster than EPANET. This is due to the fact that HIPERWATER uses the more efficient MND ordering.

On the other hand, we can appreciate that, while the parallel algorithm shows good performance for test 2, this is not the case with tests 1 and 3. This is due to the fact that the solution of the underlying linear systems does not contain sufficient computational complexity. For test 1, the size of the network (2,500 nodes) accounts for this lack of complexity. In test 3, although the network is large (32,404 nodes), the small number of pipes (34,516) with respect to the number of nodes produces a reduced number of non-zero elements in the coefficient matrix, resulting in a low number of floating-point operations in

the system solution. Thus although test 3 is larger than test 2, the computation time (with EPANET and HIPERWATER-1p) for test 3 is lower than the computation time for test 2.
The difference of complexity of the linear systems corresponding to the test cases can be also seen by estimating the average solution time per set of equations for a sequential algorithm, such as HW-1p. For test 1, a total of 102 linear systems are solved in the process of hydraulic simulation, the total computing time being 9.82 seconds for HW-1p. Thus, each linear system solution takes less than 0.1 seconds. Test 2 involves the solution of 52 linear systems with a total computing time of 50.65 seconds, which yields 0.97 seconds per linear system. Finally, 52 linear systems are solved in test 3 within a total computing time of 21.98 seconds, which gives 0.42 seconds per linear system. When using parallel computation, it is typical to obtain poorer simulation times when very small computational load is assigned to each processor and the number of processors is increased, as is the case with tests 1 and 3. This is due to the fact that the communication overhead rises and the computational benefits are not available.
As a conclusion, we can say that the performance of the parallel hydraulic simulation algorithm, in terms of execution time, highly depends on the complexity of the underlying linear equations to be solved. This is not surprising if we take into account that the parallelization approach for the simulation process is driven mainly by the parallelization of the linear systems solution. In any case, an interesting outcome is the good performance for the developed algorithm on a single processor.

Parallel Water Quality Simulation.

Various methods can be used for water quality simulation. EPANET employs the *Discrete Volume Element Method* (DVEM) (Rossman et al. 1993b). This method is fast and flexible, and can perform three different quality simulations: substance concentrations, water age analysis and percentage of flow from a source.
Each hydraulic time step is divided into smaller DVEM steps. Since the different DVEM steps have to be performed in a sequential manner, the basic idea to parallelize the quality simulation is to divide the water network in several parts, one for each processor in the system. Thus each DVEM step can be performed in parallel. The parallel algorithm for the hydraulic simulation is based on two tasks: the initial network partitioning, and the parallel algorithm for a DVEM step. These two points are discussed in the following subsections.

Network Partitioning.

A network can be considered as a graph where the vertices are given by the nodes and the edges of the graph are the pipes and valves of the network. Graph partitioning algorithms can be used to obtain a partition in which a similar number of elements (nodes) is assigned to each part and the number of edges (pipes) connecting different parts is minimized (as a consequence of this, inter-processor communications will be reduced). This initial network partitioning plays an important role to minimize communications and balance the computational load.

In particular, the approach used is known as *Multilevel Recursive Bisection* technique (Bui and Jones 1993; Hendrickson and Leland 1993). Since the partition of the network is performed only once and it is not a time-consuming task (for the test networks the time involved is less than a quarter of second) a sequential version of this algorithm has been applied.

This algorithm, as other multilevel partitioning algorithms, works in the way shown in Figure 6. First, a *coarsening phase* takes place, where the size of the graph to be partitioned is reduced, by collapsing vertices and edges. This phase consists of successive steps, a graph with a few hundred vertices being eventually obtained. Then, a bisection of this small graph is carried out. This is called the *partitioning phase,* in which two subgraphs are obtained, with a minimum number of edges with nodes situated in different parts, and a similar amount of vertices in each subgraph. Finally, the *uncoarsening phase* takes place, where the objective is to project back the partition to the original graph, by reversing the process of collapsing vertices and edges. For this purpose, the partition is successively refined. In each finer graph there are more degrees of freedom, which are used to improve the quality of the partition by moving elements from one part to the other.

This complete process quickly produces a good partition for the graph. It must be noted that the graph partitioning determines how the nodes are assigned to each processor, but nothing is said about the distribution of the pipes. As one would expect, a pipe will belong to the processor owning their end nodes. If the two end nodes belong to different processors, the pipe will be arbitrarily assigned to one of them. Actually, this means that a frontier between network parts crosses nodes and not pipes, although the associated frontier in the graph crosses edges and not vertices. Whenever a graph frontier crosses an edge, the network frontier is moved to one of the two end nodes of the corresponding pipe. We refer to the nodes situated in a network frontier as *shared nodes*.

Parallel DVEM Iterations.

The parallel algorithm for the basic quality time step is largely given by the sequential one applied in each processor to the corresponding local portion of the network. Of course, communication operations have to be carried out, since the different network portions are not independent of one another. In particular, in order to perform the transport of substance into the shared nodes, each processor has a local instance of these nodes into which the transport is done, obtaining the local values of mass and volume. Then, a communication operation is required in which the local contributions of the shared nodes are combined to obtain the final mass and water volumes, values which are then sent back to the processors sharing the nodes (this communication operation can be implemented by means of a single MPI function). The rest of the steps in the sequential DVEM algorithm are not altered.

On the other hand, the process of computing the water quality time step is done by computing locally the minimum travel time for each network portion, then obtaining the minimum of these values. Note also that since each pipe is completely owned by a single processor, the process of recalculating the concentrations of the pipe segments when a new hydraulic time step is started, is carried out locally in each processor, without communications.

Experimental Results

Time spent on this parallel water quality simulation scheme is shown in Figure 7, where HW-1p, HW-2p, HW-3p and HW-4p correspond to 1, 2, 3 and 4 processors, respectively. The 3 sample networks introduced earlier have been used. The results are compared with those obtained with the sequential EPANET water quality simulation. It can be seen that the use of the parallel algorithm leads to important reductions in computation time. In particular, a speed-up of up to 3.1 is obtained when 4 processors are used in test 1. In parallel computing context, *speed-up* is defined as the ratio of the time taken to solve a problem on a single processor to the time required to solve the same problem on a parallel computer with several identical processors

Overlapping Hydraulic and Quality Simulations.

This section presents a parallel approach complementary to that presented in previous sections. Since hydraulic and water quality simulations are frequently performed together, the idea is to overlap both simulations, so that they take place simultaneously.

In EPANET, hydraulic simulation is completely carried out before water quality simulation starts. This is shown in Figure 8, where hydraulic and water quality simulations are divided in several cells, each corresponding to a time step. After solving a time step of the hydraulic simulation, the results that will be necessary for the water quality simulation (link flows) are stored in a file. Later, these results are read before solving a time step of the water quality simulation. The subdivision of each hydraulic time step in smaller water-quality steps will not be considered in this section. Consequently the term *time step* will be used always with the meaning of *hydraulic time step*.

An alternative parallel computing approach is described in this section, in which one or several processors are in charge of the hydraulic simulation and one or several other processors are in charge of the quality simulation. As soon as the first processor group completes the hydraulic simulation of the first time step, the results (link flows) are sent to the second processor group. Then, while the first processor group proceeds with the hydraulic simulation of the next step, the second group can perform the quality simulation of the first time step (see Figure 9). In this way, both simulations are performed simultaneously, with the exception of the hydraulic simulation of the first time step and the quality simulation of the last time step. The result is an important reduction in the execution time, as can be appreciated by comparing Figure 8 with Figure 9.

The computation time required for the hydraulic simulation of a time step can be considerably different from (either greater than or less than) that required for the quality simulation, thus causing one of the processor groups to be idle most of the time. However, this can be avoided by selecting an adequate number of processors for each simulation, assigning more processors to the simulation which is observed to be more expensive.

A secondary advantage of this approach is that it is not necessary to write a temporary file with the hydraulic results, thus time is saved on input/output operations. Only part of this time is needed for communication between processors.

Figure 10 shows the computing time spent on the complete (hydraulic + water quality) simulation when this overlapping approach is used. In particular, tests have been made with

2, 3 and 4 processors (HW-2p, HW-3p and HW-4p respectively), one being in charge of the hydraulic simulation and the rest performing the water quality simulation. Time for EPANET on a single processor is also included. Speed-up reaches 4.56 (with 4 processors on test 2), showing that this parallel approach is the best choice when both simulations are performed.

Leakage Minimization.

Leakage Model.

Taking into account that permanent leakage is the consequence of defects in the network, leakage characterization may be based on the equation for a discharge through an orifice

$$q = K(p_1 - p_2)^{\beta},$$

where p_1 stands for pressure upstream of the orifice, p_2 is the pressure downstream of the orifice, and β is an exponent taking the value of 0.5 according to both theory and laboratory experiments. *K* is a coefficient that depends on the shape and size of the orifice.

Leaks through defects can be associated with a discharge to atmosphere, so $p_2 = 0$. The coefficient β must be determined from experience. A series of experiments to quantify the effect of pressure on leakage from a water supply network were carried out in (Goodwin, 1980). The experiments were conducted by operating districts with little or no night metered consumption at varying pressures and recording the net night flows. The results show that β is larger than 0.5 and takes values around 1.18. This could be justified by the deformation of defects with pressure.

Consequently, the following expression will be considered for leakage

$$q_j = K_j \, p_j^{\;1.18}, \tag{4}$$

where q_j represents the leakage flow for a network node and p_j is the pressure at the leakage node. The variable K_j represents a leakage coefficient to be determined for each node, and remains constant for long periods of time.

To obtain the values of the K_j, we make use of a method described in (Martínez et al. 1999), which takes into account the total leakage percentage observed in the whole network for district metering areas, under well-defined operating conditions.

Taking into account leakage, the process of network analysis for a given time instant is carried out by the following iterative process:

1. Leakage flows are initially assumed to be zero at every node.
2. Temporary network pressures are determined by means of the standard Gradient Method described earlier.
3. By means of equation (4), an initial approximation of leakage for every node can be obtained.
4. Leakage flows are then added to real demands and new global consumption values are obtained.
5. The iterative process ends if leakage values at any node remain constant within a given desired accuracy, else return to step 2.

Leakage Minimization.

The problem of minimizing leakage over the whole simulation period is approached by minimizing leakage at a series of different instants, according to a predefined time-step. Leakage is minimized by controlling pressures with a number of Pressure Reducing Valves (PRV). We make the assumption that leakage is minimized at each time instant independently. This independence is not strictly true, since PRV settings at an instant can alter the flows into/out of tanks, and therefore the network state at subsequent steps. However, this is a reasonable assumption, taking into account that PRVs usually control sectors with no tanks.

The problem of leakage minimization is formulated as the minimization of a nonlinear function subject to nonlinear constraints, or

$$\begin{aligned} &min\ f(x) = \sum_{j=1}^{NH} q_j(x) \\ &\text{subject to: } p_j(x) \geq pm_j, \qquad j \in I_c \end{aligned} \tag{5}$$

where:

x: PRV settings, a column vector of *NV* elements,

NV: Number of PRV valves that we can act on,

NH: Number of non-source nodes in the network,

$q_j(x)$: leakage flow at node *j*, expressed as a function of the PRV settings, given by $q_j(x) = K_j\, p_j(x)^{1.18}$,

K_j: leakage coefficient for node *j*,

$p_j(x)$: pressure at node *j*, given by $p_j(x) = H_j(x) - E_j$,

$H_j(x)$: piezometric head at node j,

E_j: elevation of node j (constant for each node),

pm_j: Minimum pressure imposed on node j,

I_c: subset of nodes where a minimum pressure is imposed.

An alternative, more standard way to express pressure constraints is:

$$g_j(x) = pm_j - p_j(x) \leq 0, \qquad j \in I_c,$$

where constraint functions $g_j(x)$ are introduced.

Note that the equations corresponding to hydraulic equilibrium do not appear explicitly, but implicitly through the functions $q_i(x)$ and $p_i(x)$. In particular, given a value of x, we can compute the corresponding values $H_j(x)$ and $q_j(x)$ by solving the hydraulic analysis problem (with leakage). Then calculation of $f(x)$ and $g_j(x)$ is straightforward.

The optimization problem stated can be solved by many different methods. In this paper we explore the use of the *Sequential Quadratic Programming* method (SQP), for the solution of the leakage minimization problem.

SQP method requires the availability of derivative information (derivatives of the objective and constraint functions). Thus, the elements that we have to provide for application of the SQP method are:

- The objective function $f(x)$ and the constraint functions $g_j(x)$. As indicated above, this involves the problem of hydraulic equilibrium.
- Gradients of the previous functions with respect to the decision variables x: $\nabla f(x)$ and $\nabla g_j(x)$. These gradients are computed by finite differences.

The Application of Sequential Quadratic Programming.

The stated optimization problem is solved by means of the *Sequential Quadratic Programming* method (SQP), by making use of a general purpose method implemented in CFSQP (*C Code for Feasible Sequential Quadratic Programming*) (Lawrence et al. 1997). Here, we will outline the basic features of CFSQP, although the interested reader should consult (Lawrence et al. 1997) and (Panier and Tits, 1993) for further details.

The CFSQP implements a complex and sophisticated algorithm, being able to cope with a large number of optimization problems, with different kinds of constraints. However, the complexity of the algorithm reduces considerably when dealing with problems like those appearing in the context of leakage minimization, i.e.

$$\text{minimize } f(x)$$
$$\text{subject to } g(x) \le 0, \tag{6}$$

where $x \in \Re^n$, $g{:}\Re^n \to \Re^p$, with $g(x) = [g_1(x), g_2(x), ..., g_p(x)]^T$.

Given a current approximation of the solution x_k, and an approximation H_k of the Hessian of the *Lagrangian* (a quasi-Newton approach with the BFGS formulation (*Broyden Flecher Goldfarb Shanno*) can be used for obtaining H_k), the standard iteration of the SQP method consists of first computing a search direction d^0 by solving the *quadratic program*

$$\underset{d^0}{\text{minimize}} \quad \tfrac{1}{2} d^{0^T} H_k d^0 + d^{0^T} \nabla f(x_k)$$
$$\text{subject to } \ g_j(x_k) + d^{0^T} \nabla g_j(x_k) \le 0, \qquad 1 \le j \le p, \tag{7}$$

and then obtain x_{k+1} by performing a line search along direction d^0, with a particular merit function. This method presents, among others, two main difficulties. First, although the original constraints are consistent, the linear approximations appearing in the quadratic programs can easily become inconsistent. Additionally, selecting an adequate merit function is a difficult issue.

CFSQP circumvents these difficulties by requiring that each approximation x_k be feasible, which eliminates possible inconsistencies in the linearized constraints, and makes the objective function an appropriate choice as a merit function.

However, feasibility of x_k does not guarantee feasibility of the standard SQP direction d^0. Even when d^0 is feasible, the line search may not allow a full step of one to be taken in a neighborhood of a solution, thus preventing superlinear convergence.

In order to overcome these difficulties, d^0 is “tilted”, i.e. replaced by a combination $d = (1-\rho)d^0 + \rho d^1$, of d^0 and an (essentially arbitrary) feasible descent direction d^1. Additionally, search direction is also “bent”, which means that the line search will be carried out along an arc $x + td + t^2\tilde{d}$. The purpose of $\tilde{d}$ is to allow a full step of one to be taken near a solution.

Parallel Algorithm for Leakage Minimization.

To parallelize the optimization process, an alternative approach takes advantage of the time-step subproblems being independent of each other. Thus each time step is assigned to a different processor, leading to a simple yet efficient coarse-grain parallel implementation.

This implies that tank levels should not affect the optimal PRV settings, which happens when the area to be controlled by the valves does not contain tanks, and is connected with the rest of the network only by means of one or more active PRVs.

The approach is very suitable to a master-slave paradigm. The master is a process whose only task is to distribute the time-step subproblems among the different slave processes. The master has a set of subproblems to be solved, (e.g. if the time-step size is 1 hour and the simulation duration is 24 hours, there would be 25 subproblems corresponding to hours 0 to 24). It initially gives one sub-problem to each slave. As soon as a slave process completes the solution of its sub-problem, it sends a message informing the master of the value of the solution. Upon receipt of this message, the master selects a new sub-problem from the set and gives it to the slave process. This scheme of dynamic allocation of sub-problems, enables a good load balance while not requiring a sophisticated implementation.

Sending a subproblem from the master to a slave involves only the communication of an integer number (indicating the time-step of the subproblem). Other information such as network topology, roughness coefficients, demand patterns, control rules, etc. is already known to the slaves, as it has been sent to them in a prior initialization stage. Messages from a slave to the master, returning information about the solution of a sub-problem, are also limited, as they contain only the PRV settings and information concerning the success of the optimisation process.

Experimental Results

The leakage reduction process described above has been applied to two water networks (see Table 2). Test network 5 contains four tanks. The level of 3 tanks is fixed to keep the distribution of the injected flow while the last varies during the simulation. Two interacting PRVs control the pressures and consequently the leakage flow.

Test network 4 is controlled by three PRVs which allow a combined or independent management. Both possibilities have been analyzed, results indicating that an independent management is better from the point of view of leakage reduction, and its results are reported below.

Each of the PRVs in network 4 is placed at the supply point of a corresponding network sector. Initially, we consider the case in which the PRVs are open (thus producing no headloss), and we assume that the percentage of leakage volume with respect to the injected volume in the network is 12%, over a simulation period of 24 hours. This initial leakage percentage is fixed arbitrarily, only to show the possibilities offered by the leakage reduction method. Using the leakage simulation process described earlier, the spatial leakage distribution shown in Figure 11 (at 0:00 hrs) is obtained.

Next, the best settings for the PRVs at each hour are obtained by means of the leakage minimization method. The result is that leakage is reduced to a value of 4.71%. Figure 12 shows the new leakage distribution at 0:00 hrs.

Table 3 shows leakage rates obtained with and without the application of the minimization algorithm, for tests 4 and 5.

On the other hand, time spent on the solution of the leakage minimization problem for test networks 4 and 5 is analyzed in Figure 13, where execution time of the parallel algorithm with different number of processors (from 1 to 5) is shown. A speed-up of 5.0 is obtained with 5 processors for test 4.

Conclusions.

This paper shows the application of HPC techniques in water network hydraulic and water quality simulation and leakage reduction.

The application of parallel computing in the water-network computational processes can take into account the complexity of optimization problems, the growing level of detail of network models related to the use of GIS, or the long simulations required for water-quality problems.

The performance of the parallel hydraulic simulation algorithm, in terms of execution time, is highly dependent on the complexity of the underlying systems of linear equations. An interesting outcome is that the *Multilevel Nested Dissection* ordering leads to reduced computation time even for sequential computations.

Performance of the HIPERWATER approach compared to the EPANET water quality simulation is very satisfactory. This is especially important if we take into account that it is not unusual for the water quality simulations to require a long simulation period to reach an equilibrium situation.

Finally, parallel algorithms developed for hydraulic and quality simulations have been successfully combined, carrying out both simulations simultaneously. This approach provides good results even for smaller networks.

The methodology presented for the determination of optimal pressure reducing valve settings minimizing leakage requires considerable computational power, which can be provided by means of the parallel algorithm described in this paper. Results are presented for a sample water network where leakage losses reduce importantly with the application of the leakage minimization algorithm described in the paper.

From a business and industrial point of view, parallel processing can help water companies increase their productivity by reducing the time to analyse networks. A further benefit is the reduction of water leakage through the optimisation of network operating conditions.

Acknowledgments.

The authors wish to acknowledge the financial support provided by the ESPRIT programme of the European Commission (HIPERWATER, ESPRIT project 24003), by the CICYT TIC96-1062-C03-01 project, and also by research staff training grants from the Spanish Government and the Autonomous Government of the Comunidad Valenciana, in Spain.

APPENDIX. References.

BUI, T., and JONES, C. (1993). “A Heuristic for Reducing Fill in Sparse Matrix Factorisation”, 6th *SIAM Conf. Parallel Processing for Scientific Computing*, 711-718.

DONGARRA, J., SNIR, M., OTTO, S., HUSS-LEDERMAN, S., and WALKER, D. (1996). *MPI: The Complete Reference*, The MIT Press Cambridge.

GOODWIN, S. J., (1980). "The Results of the Experimental Program on Leakage and Leakage Control". Tech. Rep. TR 145, Water Research Centre.

HAMAM, Y.M. and BRAMELLER, A. (1971). “Hybrid Method for the solution of Piping Networks” Proc. IEE, Vol. 118, N. 11, pp. 1607-1612.

HEATH, M. T., NG, E. G.-Y., and PEYTON, B. W. (1991). “Parallel Algorithms for Sparse Linear Systems”, *SIAM Review*, 33: 420-460.

HENDRICKSON B., and LELAND R. (1993). “A Multilevel Algorithm for Partitioning Graphs”. *Technical Report SAND93-1301*, Sandia National Laboratories.

KUMAR, V., GRAMA, A., GUPTA, A., and KARYPIS, G. (1994). *Introduction to Parallel Computing: Design and Analysis of Algorithms*. Benjamin/Cummings, Redwood City, CA.

LAWRENCE, C. T., ZHOU, J. L., and TITS, A. L. (1997). “User's Guide for CFSQP Version 2.5: A C Code for Solving (Large Scale) Constrained Nonlinear (Minimax) Optimization Problems, Generating Iterates Satisfying All Inequality Constraints”, *Technical Report TR-94-16r1*, Institute for Systems Research, University of Maryland.

LIU, J. W. H. (1992). “The Multifrontal Method for Sparse Matrix Solution: Theory and Practice”, *SIAM Review*, vol. 34, no.1.

MARTÍNEZ, F., CONEJOS, P., and VERCHER, J. (1999). “Development of an Integrated Model for Water Distribution Systems Considering Both Distributed Leakage and Pressure-Dependent Demands”. *26th Ann. Water Resources Planning and Management Conf.*, ASCE, Arizona.

PALMA J., DONGARRA J., and HERNÁNDEZ. V. (Eds.) (1998). *Vector and Parallel Processing - VECPAR’98. Third Int. Conference, Porto*. Lecture Notes in Computer Science. Vol 1573. Springer, Berlin.

PANIER, E.R., TITS, A.L. (1993). "On Combining Feasibility, Descent and Superlinear Convergence in Inequality Constrained Optimization", *Math. Programming 59*, pp. 261-276.

ROSSMAN, L. A. (1993a). *EPANET User’s Manual*, US Environmental Protection Agency.

ROSSMAN, L. A., BOULOS, P. F., and ALTMAN, T. (1993b). “Discrete Volume-Element Method for Network Water-Quality Models”, *J. of Water Resources Planning and Management*, ASCE, 119(5), 505-517.

SALGADO, R., TODINI, E., and O’CONNELL P. E. (1987). “Comparison of the Gradient Method with some Traditional Methods for the Analysis of Water Supply Distribution Networks”. *Proc. Int. Conf. Computer Applications for Water Supply Distribution*, Leicester Polytechnic, UK.

SALGADO, R. (1992). “Comparison between Linear Solvers for Sparse Systems in Steady State Pipe Network Analysis with the Gradient Method”. *Numerical Methods in Engineering and Applied Sciences*, CIMNE, 297-306.

TODINI, E. (1979). “Un Metodo del Gradiente per la Verifica delle Reti Idrauliche”, *Bolletino degli Ingegneri della Toscana*, No. 11, 11-14.

TODINI, E., and PILATI, S. (1987). “A Gradient Method for the Solution of Looped Pipe Networks”, *Proc. Int. Conf. Computer Applications for Water Supply Distribution*, Leicester Polytechnic, UK.

Table 1: Water Networks Used for Hydraulic and Water Quality Simulation.

	# pipes	**# nodes**	**# tanks**
Test 1	4,901	2,501	1
Test 2	19,801	10,001	1
Test 3	34,516	32,404	4

Table 2: Water Networks Used for Leakage Minimization.

	# pipes	# nodes	# tanks	# PRVs
Test 4	1,345	1,241	1	3
Test 5	126	66	4	2

Table 3: Leakage Rates of Different Water Networks (Test 4 and 5) Obtained with/without Minimization.

	Without minimization	**With minimization**
Test 4	12%	4.71%
Test 5	12%	8.82%

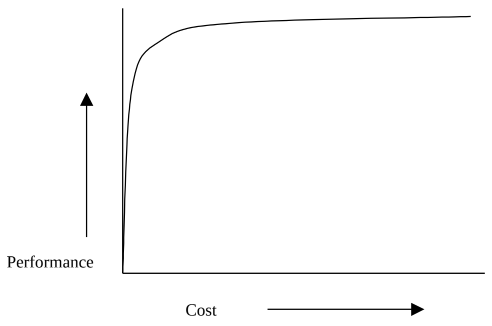

Figure 1: Cost versus performance curve.

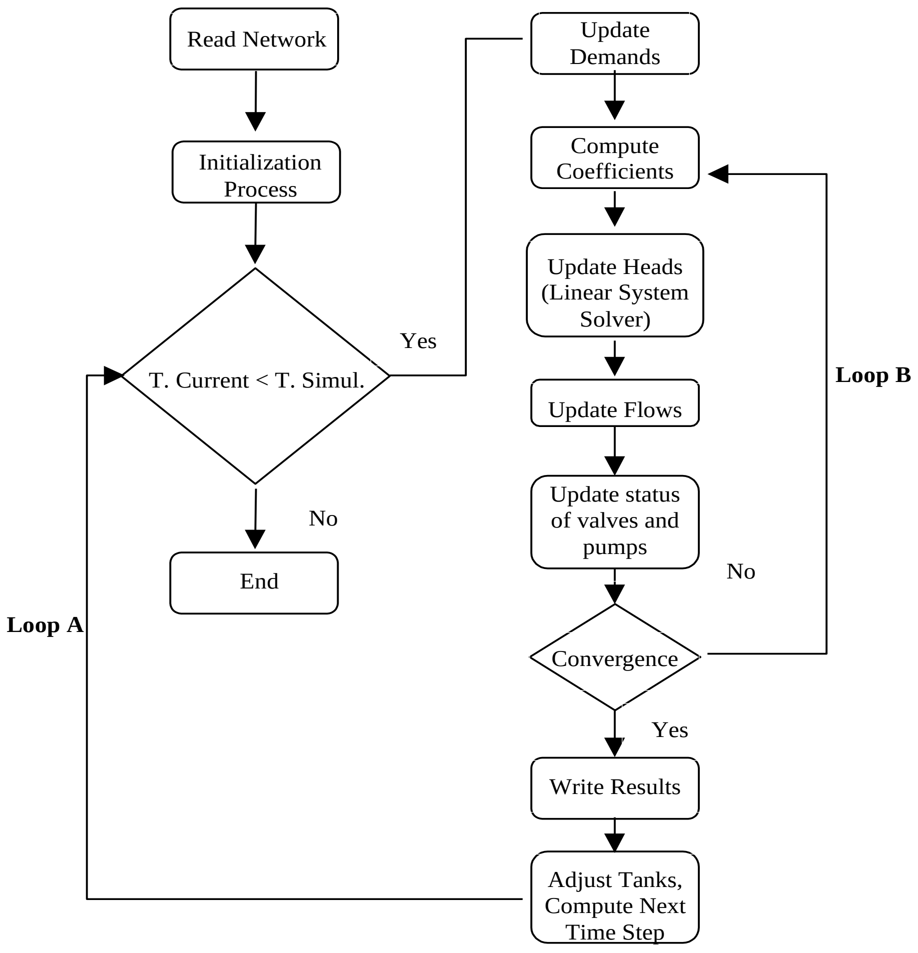

Figure 2: Flow chart for hydraulic simulation.

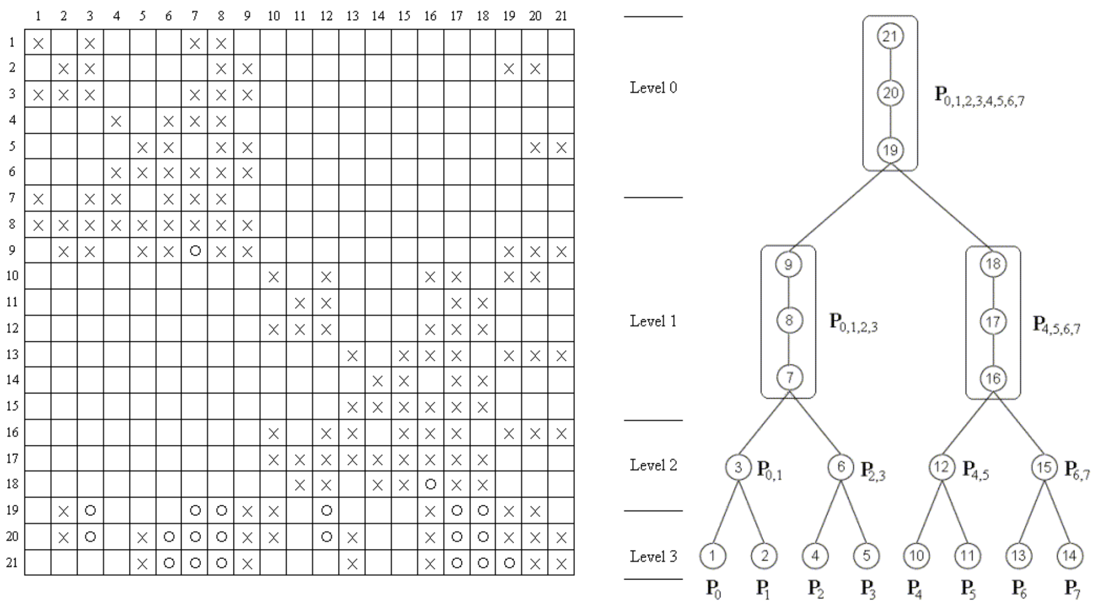

Figure 3: Elimination tree of a sparse symmetric matrix.

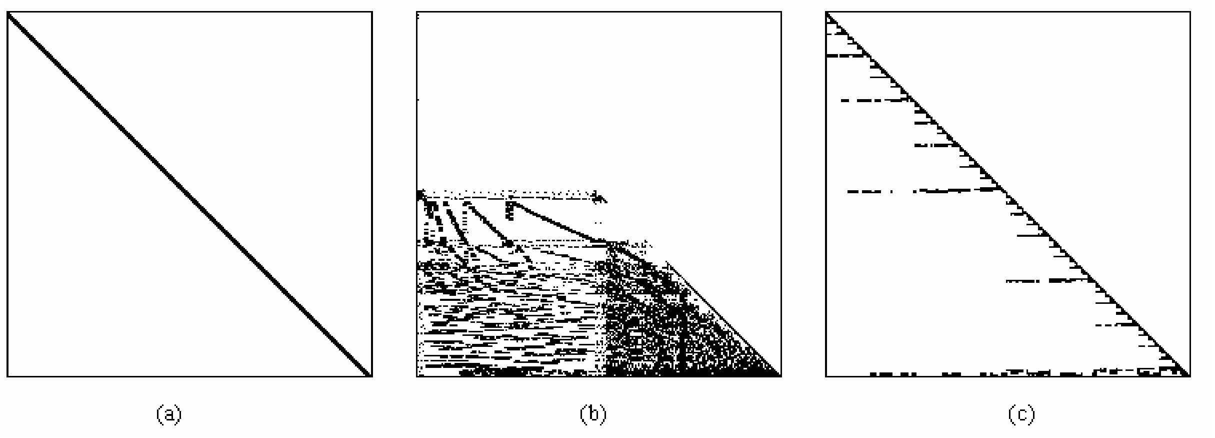

Figure 4: Non-zero structure of a coefficient matrix.

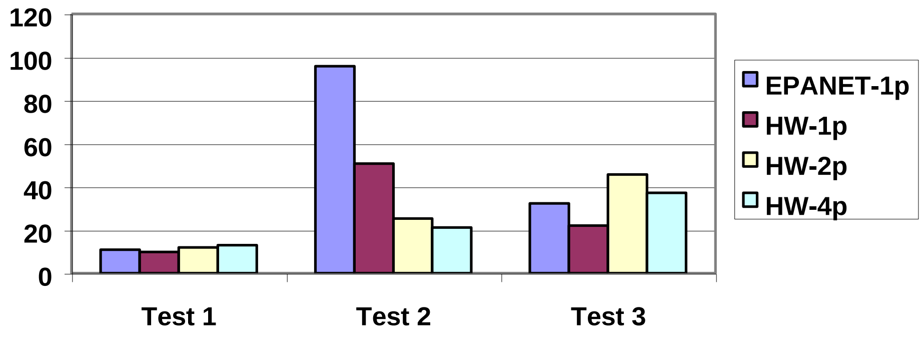

Figure 5: Hydraulic simulation time (in seconds).

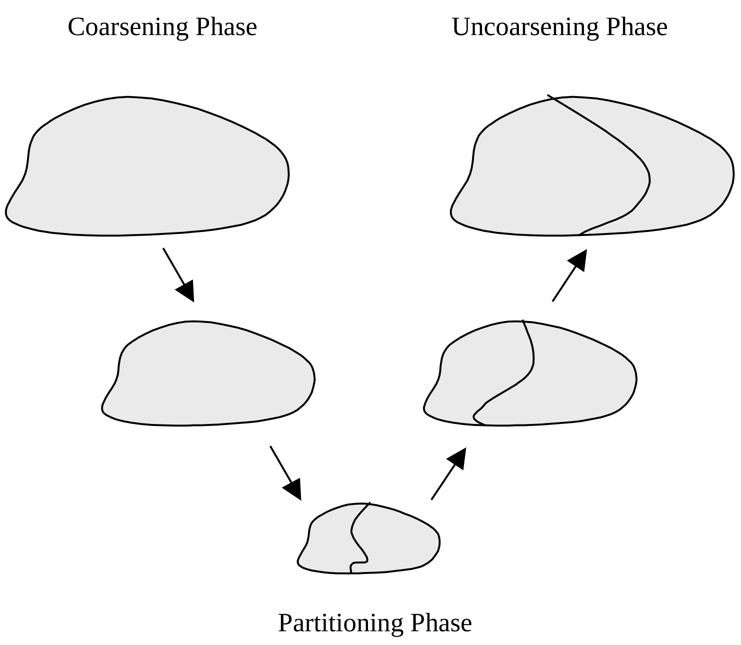

Figure 6: Multilevel partitioning algorithms.

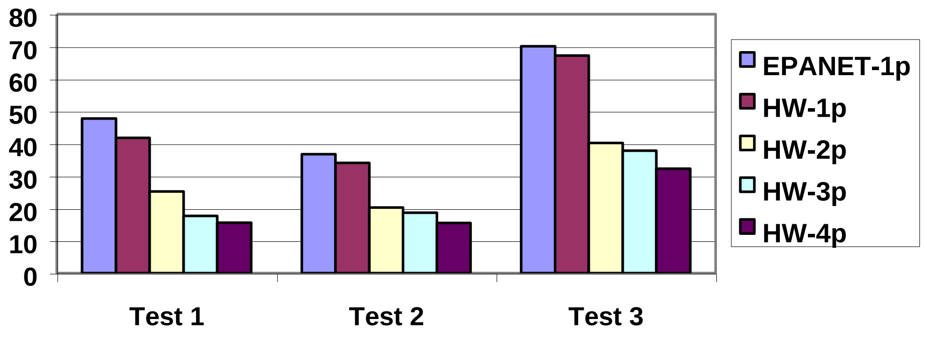

Figure 7: Water quality simulation time (in seconds).

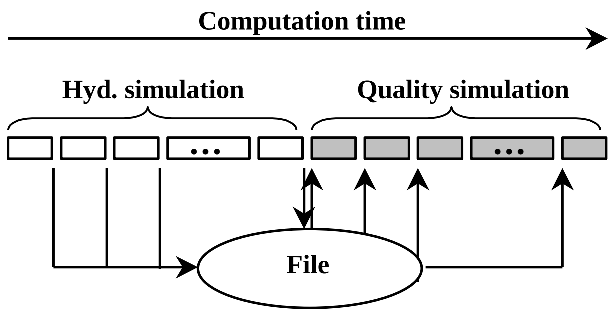

Figure 8: Hydraulic and water quality simulations in EPANET.

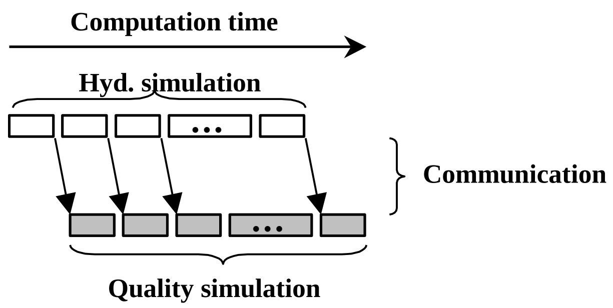

Figure 9: Overlapped hydraulic and water quality simulations in HIPERWATER.

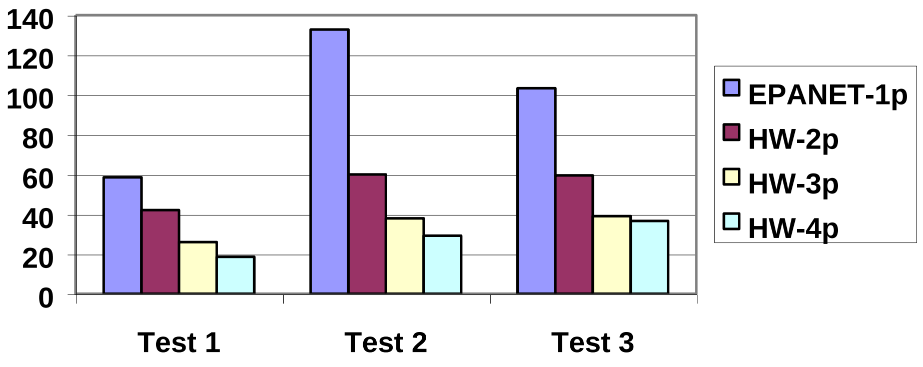

Figure 10: Execution times (in seconds) for overlapped simulations.

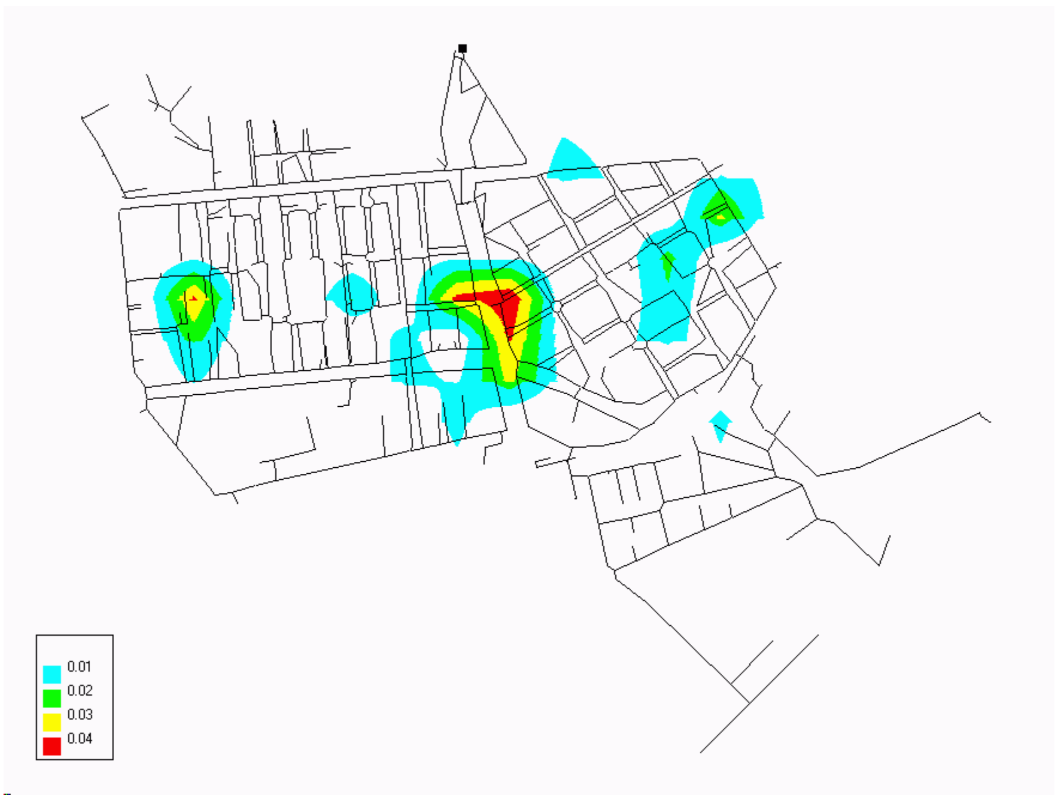

Figure 11: Initial leakage distribution (liters/sec) in test 4, at 0:00 hours.

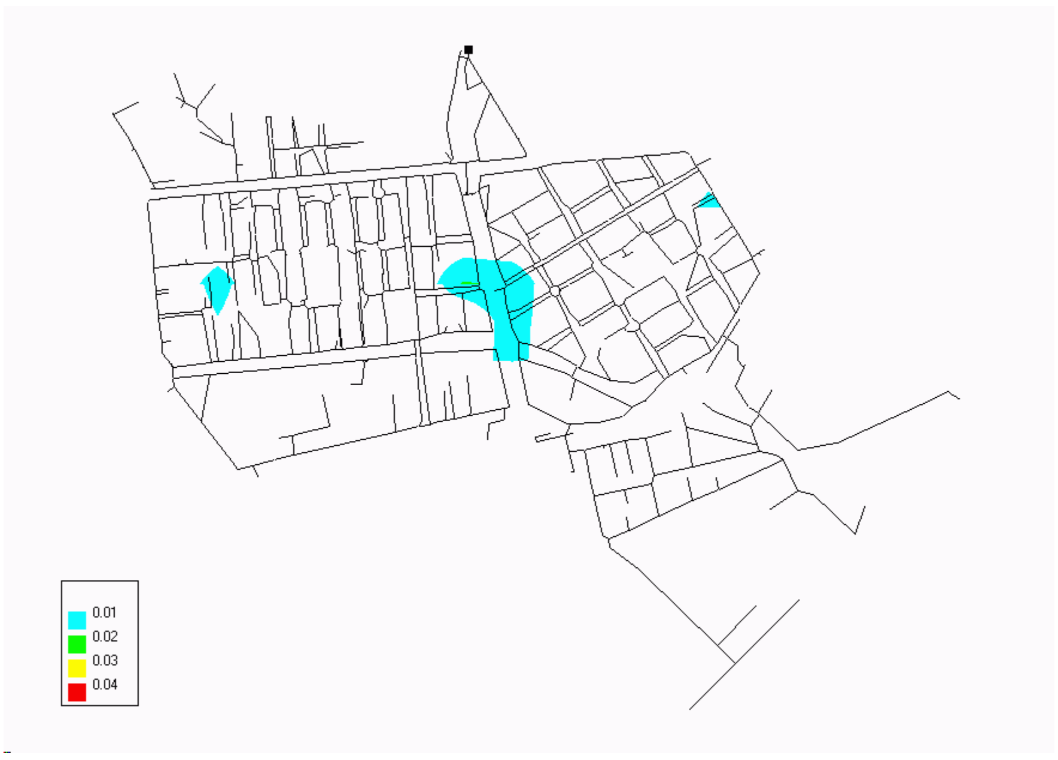

Figure 12: Final leakage distribution (liters/sec) in test 4 (at 0:00 hours).

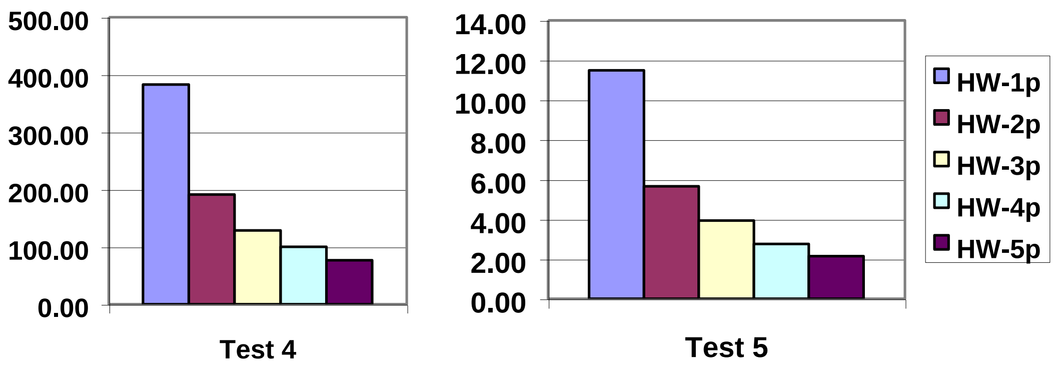

Figure 13: Leakage minimization time (in seconds).