Parallel profiling of water distribution networks using the Clément formula

Parallel profiling of Water Distribution Networks using the Clément formula

Guilherme P. Pezzi[a], Evelyne Vaissié[b], Yann Viala[b], Denis Caromel[c], Philippe Gourbesville[d]

[a] *Università degli Studi di Torino, Corso Svizzera 185, 10149 Torino, Italy*
[b] *Société du Canal de Provence, Le Tholonet, 13182 Aix-en-Provence, France*
[c] *Université de Nice Sophia Antipolis, 2004, Route des Lucioles, 06903 Sophia Antipolis, France*
[d] *Polytech'Nice-Sophia, 930 Route des Colles, 06903 Sophia Antipolis, France*

Abstract

Optimization of water distribution is a crucial issue which has been targeted by many modelling tools. Useful models, implemented several decades ago, need to be updated and implemented in more powerful computing environments. This paper presents the distributed and redesigned version of a legacy hydraulic simulation software written in Fortran (IRMA) that has been used for over 30 years by the Société du Canal de Provence in order to design and to maintain water distribution networks. IRMA was developed aiming mainly at the treatment of irrigation networks – by using the Clément demand model and is now used to manage more than 6.000 km of piped networks. The growing complexity and size of networks requested to redesign the code by using modern tools and language (Java) and also to run distributed simulations by using the ProActive Parallel Suite.

Keywords:

1. Introduction

Optimization of water distribution network for irrigation is one activity where the development of hydroinformatic tool was initiated several decades ago. With the use of 70% of freshwaters, irrigation represents a major challenge for water management, agricultural development and sustainability. Since the 70's, many tools have been developed and implemented in order to manage the irrigation and distribution issues. Late 80's, some tools like EPANET [1] based on the "Gradient method" defined by E. Todini (1987) [2] became real standards for the engineering activities, which are basically divided into design and operation. Several tools are duplicating this approach.

An alternative to this concept was introduced with the "Débit de Clément" by Société du Canal de Provence in 1977. The approach used is based on a demand model – "Débit de Clément" - that was developed to estimate on-demand irrigation consumption behaviour [3]. Société du Canal de Provence (SCP) is French company with more than 50 years experience in designing, building and maintaining water distribution networks (WDN) especially for agricultural purposes. The company uses extensively since more than 35 years numerical simulation models and has produced internally IRMA, one of the main simulation tools, since 1977.

IRMA was written first in FORTRAN 77 and was later on converted into FORTRAN 90. Nowadays, it is very difficult to implement new features or interactions of this legacy code with current technologies deployed at the SCP (such as GIS databases). IRMA had also some performance limitations due to the growth of its networks and maintaining this software became too expensive because of the lack of qualified FORTRAN developers.

The project of redesigning IRMA [4] has the goal of building a new simulation engine using up-to-date technologies, allowing to overcome the performance issues and to deliver new features to the users. The language chosen for this new tool is Java, mainly for its portability, its ability to integrate easily with other systems deployed at SCP and also for enabling distributed Grid execution using the *ProActive Parallel Suite* [5].

This paper presents the parallel pump profiling use case, including the key aspects that enable to efficiently perform distributed simulations using IRMA over the Grid. First an overview of IRMA's simulation model is presented, followed by a description of the main simulation modes supported by IRMA. Then, some of the existing Java libraries for solving linear systems are presented, followed by the adopted solution using a FORTRAN library. Finally, the distributed Pump Profiling algorithm and results are presented in order to validate our approach.

Email addresses: peretti@di.unito.it (Guilherme P. Pezzi), Evelyne.Vaissie@canal-de-provence.com (Evelyne Vaissié), Yann.Viala@canal-de-provence.com (Yann Viala), Denis.Caromel@inria.fr (Denis Caromel), Philippe.Gourbesville@unice.fr (Philippe Gourbesville)

Author copy of: Guilherme Peretti Pezzi, Evelyne Vaissié, Yann Viala, Denis Caromel, Philippe Gourbesville, Parallel profiling of water distribution networks using the Clément formula, Applied Mathematics and Computation, Volume 267, 15 September 2015, Pages 83-95, ISSN 0096-3003, http://dx.doi.org/10.1016/j.amc.2015.05.084.

2. Related work

This section is divided in 3 parts in order to cover all the main aspects of this work, first it is presented some of the existing tools for simulating *WDN*. Since the linear system solving is the most computing intensive task of the simulation, the secont part presents the studied linear algebra libraries.

2.1. WDN simulation tools

Epanet [1] is a software for modeling *WDN* that was developed by the United States Environmental Protection Agency (EPA). Epanet is an open source tool, it first appeared in early 90's and nowadays it has become a standard reference for designing *WDN*. Epanet provides an integrated environment for editing networks, running hydraulic and water quality simulations, and viewing the results.

MIKE URBAN [6] is a commercial software solution developed by DHI for urban water modeling. Besides treating *WDN*, it covers also sewer and water drainage systems. The underlying engine used by MIKE URBAN for simulating *WDN* is Epanet based and being a commercial solution it offers other interesting features, such as support for Geographical Information Systems (GIS), fire flow analysis, demand allocation and distribution. It also performs water quality analysis, for example water age, blending water from different sources, contamination propagation and growth of disinfection products.

Porteau [7] is a freeware software for simulating *WDN* and it is developed by the IRSTEA, which is a public research institute in France focusing on land management issues such as water resources and agricultural technology. The first version of Porteau was written in FORTRAN released in 1977. In 2008 CEMAGREF released the version 3.0, it is written in Java with graphical interfaces for editing and visualizing the networks.

2.2. Linear algebra libraries

The networks currently simulated by IRMA generate unstructured matrices containing up to 15.000 elements and the iterative nature of the simulation requires solving linear systems up to thousands of times for each execution. Therefore, the performance of the linear solving library has a crucial impact on the total simulation runtime. This section presents some background on the existing libraries for solving linear systems: first the java alternatives and then the standard C/FORTRAN packages.

JAMA (JAva MAtrix Package) [8] is a basic linear algebra package for Java and it was developed by the *National Institute of Standards and Technology* (NIST) at the University of Maryland, USA. It provides user-level classes for constructing and manipulating real, dense matrices.

Colt [9] provides a set of Open Source Libraries for High Performance Scientific and Technical Computing in Java and it was developed by the *European organization for Nuclear Research* (CERN). *Colt* uses *JAMA* as underlying library for linear algebra and therefore offers the same five decompositions available in *JAMA*.

MTJ [10] is a library for developing numerical applications, both for small and large scale computations. The library is based on BLAS [11] and LAPACK [12] for its dense and structured sparse computations, and on the Templates project for unstructured sparse operations. *MTJ* is based on the netlib-java project [13], which can be set up to use machine-optimized BLAS libraries for improved performance of dense matrix operations, falling back to a pure Java implementation.

BLAS [14] stands for *Basic Linear Algebra Subprograms*, it was first published in 1979 and is a standard API for libraries publishing basic linear algebra operations. The FORTRAN version is widely used but it has been also translated into other languages (such as C and Java).

LINPACK [15] is a collection of FORTRAN subroutines that analyze and solve linear equations and linear least-squares problems. It uses the *BLAS* libraries for performing basic vector and matrix operations. *LINPACK* offers decomposing methods such as LU, QR and Cholesky.

All of these libraries require either a pre-conditioned matrix (for using the sparse solvers) or a dense matrix storage for implementing a robust solver (which is not efficient for solving IRMA's sparse systems).

Finally, the Yale Sparse Matrix Package [16] implements the *LU Decomposition* with sparse matrix storage and is optimized for iterative system solver (by reusing pivots). This is a FORTRAN library, more details about its usage and integration with Java is given in Sec. 4.

In order to evaluate these solvers, a test case simulation has been performed using some of these libraries. Figure 1 shows the execution times of Jama, Colt and Blas are pure Java solutions, FORTRAN Yale is pure FORTRAN and Java Yale is the mixed FORTRAN/Java solution (which was adopted in this work).

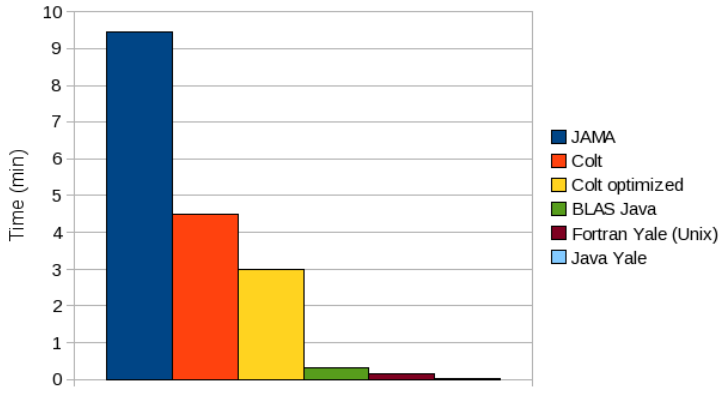

Figure 1: Execution time of a network simulation using different solvers in a Linux machine and the former IRMA FORTRAN version executed in a HP-UX system.

2.3. Parallel/distributed computation tools

There are hundreds of environments that support parallel and distributed computation, it is worth mentioning the MPI interface [17] that is the de facto standard for scientific applications and it is very well suited for applications that require to perform communication with very low latency. There exist more recent frameworks that focus also on multicore, such as OpenMP [18], which is a standard API for exploiting parallel resources on shared-memory architectures, and FastFlow framework [19] that provides C++ patterns for parallel and distributed application.

Since the parallel tasks executed in the pump profiling are coarse grained and IRMA is written in Java, it is more convenient to use a tool that is simple to use with Java. Finally, the chosen framework for this work is *ProActive Parallel Suite* [5], which is an innovative Open Source solution (OW2 consortium) for parallel, distributed, multi-core computing. Developed in Java, it features a concurrent and parallel programming model, offers distributed and asynchronous communications, mobility and a deployment framework. The main reason for this choice is the straightforward integration with IRMA's code and the possibility to easily deploy parallel simulations across an environment with distributed resources.

3. IRMA Simulation model

The main IRMA engine computes pressures, flow and head losses in piped water distribution networks. In top of that, there are other layers for modelling equipment, demands, tank state over time. Most IRMA features are very similar to EPANET [1], which is extensively used for *WDN* simulation, but some features are specifically designed to fit *SCP*'s needs and methodologies, such as the *Clément's Formula* that is used to estimate the demand for rural clients.

In order to describe a meshed network, it is necessary to identify its **nodes**, its **pipes**, its **points with known charge**, its **equipment** and to specify the **discharges** at each point.

A **node** is a junction of two (or more) connected pipes in the network. A **point with known charge** is a node where the head (piezometry) is known at the beginning of the simulation (such as a reservoir). An **equipment** is a device that creates a specific head loss that is added to the pipes' linear head losses (or head gains in case of pumping stations).

From this description, IRMA needs to find some specific paths before starting the computation: all the closed loops and the path between points with known charge need to be identified. These paths are used for building the equation system and also for calculating the probabilistic demand model.

The mathematical model used by IRMA is based on the Kirchhoff's circuit laws that state: *the algebraic sum of currents in a network of conductors meeting at a point is zero.* These laws can be also applied to water networks, they are used in IRMA to write equations that represent the flow conservation at the nodes and also the expected head differences between two nodes.

IRMA's system of equations is divided in 3 groups:

$$\begin{cases} \sum_i Q_i = 0, \text{ KN equations} & (1) \\ \sum_i J(Q_i) = 0, \text{ KL equations} & (2) \\ \sum_i J(Q_i) = \Delta H, \text{ KE-1 equations} & (3) \end{cases}$$

where
Q_i are the incoming and outgoing flows from each node,
$J(Q_i)$ are the head losses associated to each pipe,
KN is the number of nodes in the network,
KL is the number of loops,
KE is the number of nodes with known charge (tanks/reservoirs) and
ΔH is the head difference between the tanks.

- First group of equations (Eq. 1) represents the *flow conservation at the nodes*: sum of the flows in each node must be *zero*. Each node is represented in the linear system by one equation that defines its connections with all other nodes: $'1'$ is used to represent a connection between two nodes and $'0'$ otherwise. These equations represent largest portion of the linear system and since each node is usually connected to a maximum of 3 other nodes the matrix representing the system will be sparse.
- Second group (Eq. 2) represents the *head loss equations in the meshes*: sum of head losses from the first to the last node in each loop must be *zero*.
- Third group (Eq. 3) represents the *head losses in the paths between tanks*: head loss in each path between two tanks must be equal to head difference between first and last tank.

The unknown variables in the equation system are the flows in each node (Q_i). The heads (pressures) in each node are obtained from the flows, by calculating the pipes head losses. The number of equations must respect the following equation:

$$KK = KN + KL + KE - 1$$

where KK is number of unknown flows.

The first group is filled with linear coefficients but second and third groups contain non linear elements and must be linearized in order to be solved using traditional equations solving algorithms.

Iterative mechanism: IRMA determines initial flow values for each pipe and then calculates the head in each node. The resolution is then performed iteratively by applying the Kirchhoff's laws to calculate the resulting flows and then updating the heads based on these new flows. This process is repeated until the sum of all differences between input and result flows are smaller than a determined convergence value or until a maximum number of iterations is performed (without reaching a stable state).

3.1. Head loss equations

The head loss (J) in each pipe is calculated in function of the flow (Q) and is given by the formula:

$$J(Q) = -(a + b.Q + c.Q^2) + \alpha.Q^\beta + D.Q^2 \text{ (positive flowrate values)}$$

where a, b, c represent pump coefficients (if it exists),
α represents friction head loss coefficients,
β is given by the head loss calculation method and
D represents singular head loss coefficient.

The equation needs to be adapted when the flowrates are negatives:

$$J(Q) = -(a + b.Q + c.Q.|Q|) + \alpha.Q.|Q|^{\beta-1} + D.Q.|Q|$$

with positive pump coefficients a, b and c.

This head loss expression is continuous and derivable. Therefore we can write, with $Q = Q_0 + \Delta Q$ and $J'(Q_0) = (\frac{dJ}{dQ})_{Q_0}$:

$$J(Q) = J(Q_0) + J'(Q_0).\Delta Q = (J(Q_0) - Q_0.J'(Q_0)) + J'(Q_0).Q$$

Head loss in a loop: the sum of all head losses in a loop is *null*. In a loop we have $J(Q) = 0$, therefore:

$$\sum J'(Q_0).Q = -\sum(J(Q_0) - Q_0.J'(Q_0))$$
First member . X = Second member

This formula will be used to write one equation for each loop in the network.

Head loss in a path between tanks: in these path equations the sum of all head losses in each path is equal to the head difference between the two tanks. In each path we have $J(Q) = \Delta H$, therefore:

$$\sum J'(Q_0).Q = -\sum(J(Q_0) - Q_0.J'(Q_0)) + \Delta H$$

This formula will be used to write one equation for each 2 tanks in the network. If a network contains KE tanks, there will be $KE - 1$ tank equations in the system.

3.2. Linearizing head losses

Head loss values (J_1 and J_2) are calculated with one of these formulas using two reference flowrate values ($1m^3/s$ and $0.5m^3/s$). Then, a linear head loss coefficient α is determined by the equation:

$$\alpha = (2.\sqrt{J_1} - \sqrt{J_2})^2.(1 - \frac{\sqrt{J_1} - \sqrt{J_2}}{2.\sqrt{J_1}})^2$$

After obtaining the linear coefficient α, the following equation is used for calculating the head loss:

$$J(Q) = \alpha.Q.|Q|$$

Finally, matrix's head loss equations (loops and path between tanks) will be filled using this formula:

$$J'(Q_0) = \frac{J(Q_0+h) - J(Q_0)}{Q_0.h}$$

where h is a constant small value.

3.3. Convergence analysis

IRMA uses several mechanisms for evaluating the convergence of a simulation [20] and for sake of simplicity this section presents the main convergence test, that regards the variation of flow values in each pipe. For each pipe with a significant flow value (> 0.005 l/s), the following sum is calculated:

$$\Delta Q = \frac{\sum_{j=0}^{Npipes} \frac{|Q_i - Q_{i-1}|}{|Q_{i-1}|}}{Npipes}$$

where
ΔQ is the average flow variation between current and previous iteration,
$Npipes$ is the total number of pipes in the network,
Q_i is the current flow value in the pipe and
Q_{i-1} is the previous flow value in the pipe.

If a **peak consumption** simulation is performed, a similar test is performed but instead of summing the flow values, the sum of the number of offtakes is taken:

$$\Delta n = \frac{\sum_{j=0}^{Npipes} \frac{|n_i - n_{i-1}|}{|n_{i-1}|}}{Npipes}$$

where
Δn is the average variation of number of offtakes between current and previous iteration,
$Npipes$ is the total number of pipes in the network,
n_i is the current number of offtakes in the pipe and
n_{i-1} is the previous number of offtakes in the pipe.

Then the following test is executed:

$$\Delta Q \text{ or } \Delta n < TOL$$

where TOL is a constant tolerance value.

IRMA can model different kinds of equipment: Pressure regulators, Singular head loss / Valve, Differential head loss / One-way valve, Flow regulators and Pumps. As an example for the Pump Profiling use case, the following section describes how Pumps are modelled with IRMA.

3.4. Equipments

This section presents the equipments that can be described using IRMA and how they are modeled.

3.4.1. Pumps

Pumps are mechanical devices used for moving the water, they can be used to feed tanks or as a *booster* to increase the network's pressure. Figure 2 shows a network scheme to illustrate the piezometric line behavior of a pump in a sample network. The piezometric head represents the elevation summed with the pressure head at each point of the network. In this case the pump is used to feed tank 2, located in the higher end of the network.

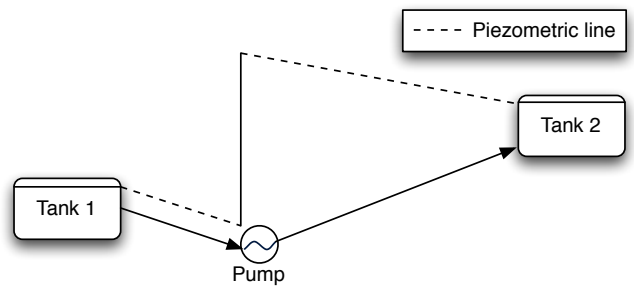

Figure 2: Scheme representing the piezometric level of a network with a pumping station.

IRMA supports several types of pumps, they can have variable or fixed behavior and therefore they are divided in two groups: fixed or variable speed pumps.

Fixed speed pumps have a constant behavior during the iterative simulation and are described by the curves that represent their Total Dynamic Head (TDH). The TDH of a pump is given by the following equation:

$$TDH(Q) = a + b.Q + c.Q^2$$

where a, b, c are the curve coefficients (usually given by the pump manufacturer) and Q is the flowrate. The TDH is obtained in function of the observed flowrate and then it will be applied as head gain to the pipe containing a pump.

Variable speed pumps adapt their behavior depending on an observed parameter. They can have 3 regulation modes:

- *Imposed regulation curve*: in this mode there are different curves associated with a flowrate interval. Whenever the observed flowrate is within an interval, the corresponding curve will be adopted.
- *Imposed head in a determined node*: this mode is used to find out the coefficient k so that $TDH(Q) = k^2.a + k.b.Q + c.Q^2$ can be used to obtain the desired head in a specified node.
- *Imposed flowrate value*: pumps in this mode are used to fill a tank using a chosen flowrate value ($Q_i = cst$). In this case, TDH is calculated automatically at each iteration in order to respect the hydraulic equilibrium.

3.4.2. Pressure regulating devices

Pressure regulating devices are used to control the pressure in a specific part of the network. There are 2 types of regulators: pressure reducing and pressure sustaining valves. The head loss for these equipments is automatically adjusted by IRMA at each iteration in order to achieve the desired head.

Pressure Reducing Valves are used to maintain the downstream pressure **below** a chosen head value (Z_{reg}). After each iteration i, the coefficient DH is adjusted for the iteration $i + 1$ taking into account the difference between the obtained head and the objective head :

$$DH_{i+1} = DH_i + \frac{Z_i - Z_{reg}}{Q^2}$$

where
DH_{i+1} is the adjusted head loss coefficient for the next iteration,
DH_i is the current head loss coefficient,
Z_i is the obtained head,
Z_{reg} is the desired downstream head and
Q is the flowrate.

Figure 3 displays a scheme representing the piezometric line when the pressure reducing valve is active (left) and the case where it is inactive because the downstream head is below the desired head (right).

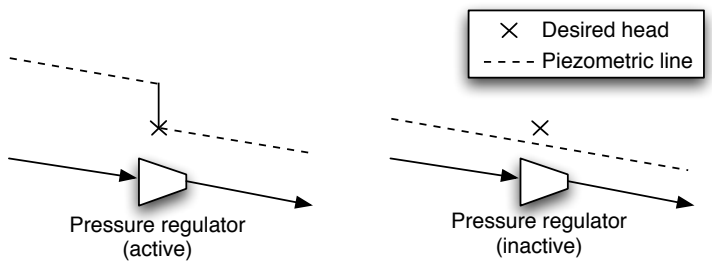

Figure 3: Piezometric schemes for a pressure reducing valve.

Pressure Sustaining Valves are used to maintain the upstream pressure **above** a chosen head value (Z_{reg}). After each iteration i, the coefficient DH is adjusted for the next iteration using the following formula:

$$DH_{i+1} = DH_i + \frac{Z_{reg} - Z_i}{Q^2}$$

where
DH_{i+1} is the adjusted head loss coefficient for the next iteration,
DH_i is the current head loss coefficient,
Z_i is the obtained head,
Z_{reg} is the desired upstream head and
Q is the flowrate.

Figure 4 displays a scheme representing the piezometric line when the pressure sustaining valve is active (left) and the case where it is inactive because the upstream head is above the desired head (right).

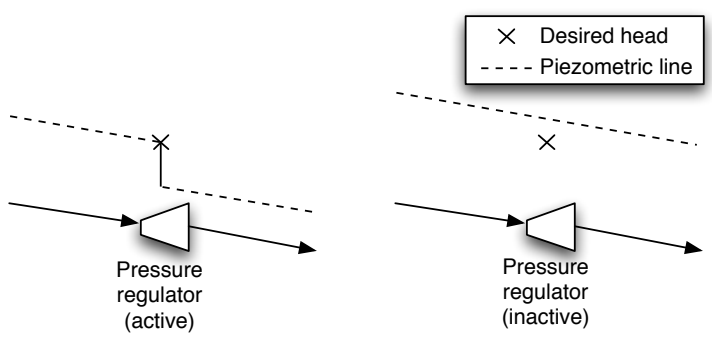

Figure 4: Piezometric schemes for a pressure sustaining valve.

If the water flows towards the direction that is not concerned by the valve (open position), a second head loss coefficient k_{open} can be applied using the following formula:

$$DH = k_{open}.Q^2$$

3.4.3. Singular head loss / Valve

A valve is treated as a singular head loss k that is constantly applied regardless of the flowrate direction.

$$DH = -k.|Q|.Q$$

where Q is the flowrate. If k has a high value, this equipment will be treated as a closed valve and this will represent a discontinuity in the network.

3.4.4. Differential head loss / One-way valve

A differential head loss is used to model valves that have different behaviors depending on the flow direction. In this case, two coefficients k_{pos} and k_{neg} are informed for calculating respectively the head loss in positive (Eq 4) and negative (Eq 5) position.

$$\begin{cases} DH_{pos} = -k_{pos}.|Q|.Q & (4) \\ DH_{neg} = -k_{neg}.|Q|.Q & (5) \end{cases}$$

where Q is the flowrate. If k_{pos} or k_{neg} has a high value, this equipment will be treated as a one-way (security) valve that only allows the water to flow in one direction. This valve is often used, for example, associated to a tank in order to avoid emptying it.

3.4.5. Flow control valves

Flow control valves are used in order to keep flow under a chosen value. In order to achieve the desired flow value (Q_{reg}), the head loss coeffcient DH are automatically adjusted at each iteration using this formula:

$$DH_{i+1} = DH_i + \frac{K_i}{Q_i^2}$$

where
DH_{i+1} is the adjusted head loss coefficient for the next iteration,
DH_i is the current head loss coefficient,
Q_i is the flow and
k_i is an adjustment coefficient.

If DH_{i+1} becomes negative (causing a head gain), the value *zero* is imposed to the coefficient.

4. Linear system solving using the Yale Sparse Matrix Package

This FORTRAN library [16] was written at the Yale University and it implements the *LU Decomposition* with sparse matrix storage. The resolution is divided in two phases:

1. **Pivoting:** consists in organizing the equation matrix assuring that it contains a *non-zero* element in the diagonal. The row permutations that need to be performed are stored in a *permutation matrix* which is given as input for the second phase.
2. **Resolution:** solves the system by performing the *LU Decomposition* using the **permutation matrix**.

The **pivoting** phase has the highest computational cost and this package also allows the reuse of the *permutation matrix*. Since we need to solve several times our systems and the *pivots* rarely change among the iterations, this feature can dramatically speed up our simulation.

In order to use this FORTRAN library from a Java application it is necessary to perform native calls using JNI (Java Native Interface). Since JNI does not allow direct access to FORTRAN code from Java, a C++ interface was written to wrap the FORTRAN library. The main drawbacks of this approach are reducing the portability of the Java code and complicating the compilation process, but this solution has been already successfully deployed in Windows, HP-UX Unix and Linux.

This library takes as arguments the list of system's *pivots* and the matrix stored using the Yale sparse matrix format. Therefore the last requirements for using this library are to search the *pivots* before first iteration (also after a few particular cases, e.g. when a pipe is closed between two iterations) and write the matrix respecting Yale's compressed format for matrix storage.

Yale's compressed matrix format:

The sparse format stores only the *nonzero* entries of the matrix A, row-by-row in the array YA. To identify the individual *nonzeros* entries in each row, it is needed to know which column each entry lies. The column indexes which correspond to the *nonzero* entries of the original matrix are stored in the array JA; i.e, if $YA(K) = A(I, J)$ then $JA(K) = J$.

In addition, it is needed to know where each row starts and how long it is. The index positions in JA and YA where the rows of A begin are stored in the array IA; i.e., if $A(I, J)$ is the first *nonzero* stored entry in the $I - th$ row and $YA(K) = A(I, J)$, then $IA(I) = K$.

Moreover, the index in JA and YA of the first location following the last element in the last row is stored in $IA(N+1)$. Thus, the number of entries in the $I - th$ row is given by $IA(I + 1) - IA(I)$, the *nonzero* entries of the $I - th$ row are stored consecutively in:

$$YA(IA(I)), YA(IA(I) + 1), ..., YA(IA(I + 1) - 1)$$

and the corresponding column indexes are stored consecutively in:

$$JA(IA(I)), JA(IA(I) + 1), ..., JA(IA(I + 1) - 1)$$

For example, the $5x5$ matrix:

$$\begin{pmatrix} 1 & 0 & 2 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 \\ 0 & 4 & 5 & 6 & 0 \\ 0 & 0 & 0 & 7 & 0 \\ 0 & 0 & 0 & 8 & 9 \end{pmatrix}$$

is stored as:

	1	2	3	4	5	6	7	8	9
IA	1	3	4	7	8	10			
JA	1	3	2	2	3	4	4	4	5
YA	1	2	3	4	5	6	7	8	9

5. Simulation modes

This section presents the different simulation modes proposed by IRMA, including the peak demand computation using the *Clément's Formula* that is used to estimate the peak demand in irrigation networks.

5.1. Continuous flows

Continuous flow is the basic simulation mode where all the demands are known and it is also used for performing the other types of simulation that are detailed later in this chapter. IRMA's system of equations is modified in order to take the demand into account and perform this simulation:

$$\begin{cases} \sum_i Q_i = D_n, \text{ KN equations} & (6) \\ \sum_i J(Q_i) = 0, \text{ KL equations} & (7) \\ \sum_i J(Q_i) = \Delta H, \text{ KE-1 equations} & (8) \end{cases}$$

where
Q_i are the incoming and outgoing flows from each node,
D_n is the node's demand,
$J(Q_i)$ are the head losses associated to each pipe,
KN is the number of nodes in the network,
KL is the number of loops,
KE is the number of nodes with known charge (tanks/reservoirs) and
ΔH is the head difference between the tanks.

Second member of equation 6 represents the demand at each node. These equations are then used to iteratively calculate the hydraulic equilibrium for scenarios with known demand values.

5.2. Peak consumption using the Clément demand model

The goal of this simulation mode is to compute the dynamic heads at the moment of peak consumption. Since this consumption is unknown, the flows are estimated at each pipe by using the *Clément's formula* [3]. Basically, this formula takes as input the number of offtakes (among other parameters) at each point and calculates the flow in a given pipe based on the sum of all the estimated downstream discharges. The offtakes are separated in classes according to the nominal flow (established on the contracts) and they are also associated to a probability according to the usage (for example, type of crop).

5.2.1. The general expression

The flow Q generated for a given class of offtake in a pipe is given by the general expression of *Clément's formula*:

$$Q_{continuous} + \sum n_k.p_k.d_k + \epsilon.U\sqrt{|\sum n_k.p_k.(1-p_k).d_k^2|}$$

where
d_k is the nominal discharge of class k,
n_k is the number of served offtakes from class k,
p_k is the usage probability of the offtakes from class k,
U determines the desired quality of service,
ϵ is the sense of the flow (represented by the sign of the term $\sum n_k.p_k.(1-p_k).d_k^2$) and
$Q_{continuous}$ is the pipe's continuous flow (which is known in advance).

An association of *nominal discharge* and *opening probability* defines a class of offtake. Therefore, a new class should be created for each offtake with different flow value or probability.

When the distribution of a given class of offtake k is calculated, the distribution of the classes 1 to $k-1$ and the distribution of continuous flows are also known. The total flow in a pipe (in function of the number of offtakes) is given by the following formula:

$$Q = Q_{continuous} + n_k.p_k.d_k + \sum_{j=1}^{k-1} n_j.p_j.d_j + \epsilon.U\sqrt{|n_k.p_k.(1-p_k).d_k^2 + \sum_{j=1}^{k-1} n_j.p_j.(1-p_j).d_j^2|}$$

5.2.2. Adapting conservation equations to Clément formula

When applying *Clement's* formula, the equations need to be adapted in order to treat the conservation of number of offtakes instead of flow. For a given class of offtake, the flow conservation equations are modified to:

$$\begin{cases} \sum_i n_i = 0, \text{ KN equations} & (9) \\ \sum_i J(n_i) = 0, \text{ KL equations} & (10) \\ \sum_i J(n_i) = \Delta H, \text{ KE-1 equations} & (11) \end{cases}$$

where n_i is the number of offtakes of this class served by the pipe.

Most networks have several classes of offtake and therefore this system needs to be solved successively for each class of offtake:

$$\begin{cases} \sum_i n_{ik} = 0, \text{ KN equations} & (12) \\ \sum_i J(n_{ik}) = 0, \text{ KL equations} & (13) \\ \sum_i J(n_{ik}) = \Delta H, \text{ KE-1 equations} & (14) \end{cases}$$

where n_{ik} is the number of offtakes of class k served by the pipe.

This means that the hydraulic equilibrium needs to be iteratively calculated for each class of offtake and the obtained results are used for calculating *Clément's* values for the next class.

5.2.3. Calculating usage probability for irrigation networks

One crucial aspect when performing simulations is to check if the model corresponds to what is actually observed in the network. IRMA allows fitting the probabilistic parameters based on the observed consumptions in the network.

Given a subset of network nodes, the usage probability of the offtakes of the class i is calculated by using this formula [3]:

$$p = \frac{Si.v}{r.\sum_i (n_i.d_i)}$$

where
Si is the irrigated surface,
v fictive continuous flow,
r is the crop yield (efficiency)
n_i is the number of offtakes of the class i present in the subset of the network and
d_i is the nominal flow of the class i.

6. Pump Profiling Use Case

This section presents how IRMA can help the design of new pumps in a network without tanks. Pump profiling is performed first by defining a fixed amount of flow (demand) objective values, varying from the minimum demand that can be observed (only continuous flows) up to a demand that is equivalent to 25% more than the estimated peak consumption.

Second step is to generate the possible discharge scenarios that will reach each objective flow value. These scenarios could be generated exhaustively in networks containing only a few nodes, however in most networks this is not possible and for that reason they will be randomly generated. The number of scenarios will be calculated in function of the number of offtakes, with a minimum of 250 scenarios in order to guarantee a minimum sample size in case of networks with a small number of offtakes.

Finally, all these scenarios will be simulated and the resulting value for each scenario will be the required head in the node where the pump will be placed in order to provide the guaranteed pressure to all clients. All this data will then be gathered to plot one graph where each line represents the required head to satisfy a percentage of scenarios.

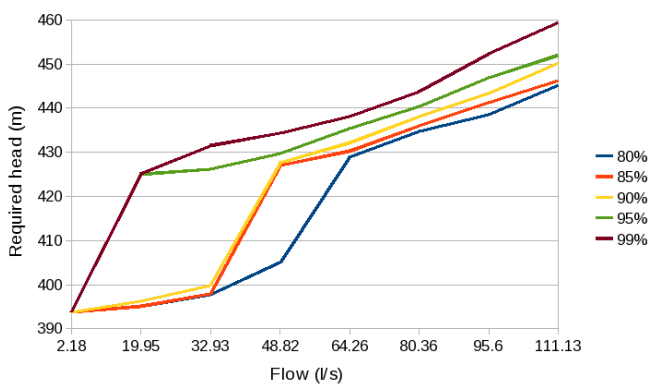

Figure 5: Pump profiling in the "Boutre" network.

Figure 5 displays a sample result of this simulation in an existing network, objective flows are represented in the x axis, required head in the y axis and each line represent a % of scenarios that are satisfied. This simulation mode has two main steps, first the initialization of the network and of the different parameters that will be used for generating all the scenarios (Sec. 6.1) and the simulation of these scenarios (Sec. 6.2).

6.1. Initialization

The **initialization** phase consists in the execution of the following steps:

1. Definition of the **number of configurations** (scenarios) to simulate for each objective flow. This value is calculated as follows:

$$N_{conf} = k.N_{hydrants}$$

where
k is a multiplicative coefficient that can be defined by the user and
$N_{hydrants}$ is the total number of hydrants installed in the network.

A minimum of 250 scenarios is imposed, in order to have a significant sample size for networks containing a small number of hydrants.

2. Definition of the **objective flow intervals** : a ΔQ is calculated in order to divide the flow interval $[0, Qmax]$ in equal parts respecting the limit of 20 objective flows.
Q_{max} is defined as 25% more than the estimated peak discharges and it is obtained as follows:

$$Q_{max} = 1,25.(Q_{peak} + Q_{cont})$$

where
Q_{peak} is the peak demanded flow (using the *Débit de Clément*) and
Q_{cont} is the sum of all known continuous discharges.

3. **Pumping station neutralization:** this step consists in removing the pumping station from the network description and updating the upstream tank head level:

$$Z_{tank} = Z_{ref} + a_{pump}$$

where
Z_{ref} is reference value for the desired head in the upstream tank and
a_{pump} is the TDH coefficient taken from the neutralized pump.

4. **Hydrants initialization:** transformation of the discharges represented by number of offtakes of each class into individual objects representing each hydrant (client) and the associated nominal flow.
5. **Computation of the piezometric line:** calculation of the static piezometric line by performing a hydraulic equilibrium computation considering the network without discharges and after the pump neutralization.

6.2. Simulation steps

The second step is to iterate through all the objective flows and simulate N_{conf} configurations (scenarios) where the demand is equal (or close to) the objective value. This process is described in the Figure 6.

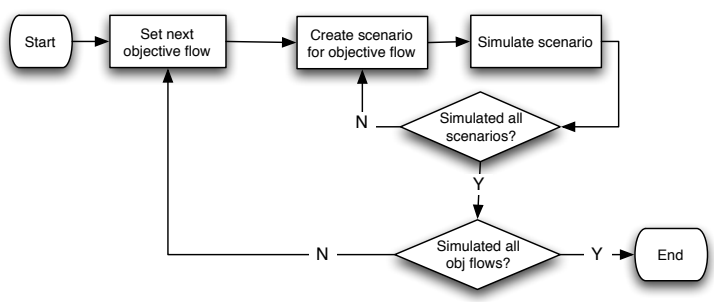

Figure 6: Flow diagram for the pump profiling simulation.

The result of each simulation is the required head where the pump will be placed, in order to obtain a satisfying pressure in all points of the network. This required pressure is obtained based on the observed pressure at the point with minimum exceeding pressure.

At the end of all simulations, a curve representing the required head in order to satisfy a certain percentage of scenarios will be plotted for each objective flow.

6.3. Parallel pump profiling on the Grid

In order to perform the pump profiling several different scenarios need to be simulated. The total number of simulated scenarios is proportional to the number of hydrants present in the network and therefore the execution time can be prohibitive in case of large networks.

This section presents a solution for reducing the execution time by performing a distributed simulation of these scenarios on a Grid, using the *ProActive Parallel Suite*.

6.4. Parallelization algorithm

The initialization phase is executed sequentially (as described in Sec. 6.1), since it is not computing intensive. The simulation phase is executed as described in Fig 7: parallel tasks containing X scenarios to be simulated are created. This parameter X defines the granularity of each task and should be adjusted according to the network size and number of available resources where the tasks can be deployed.

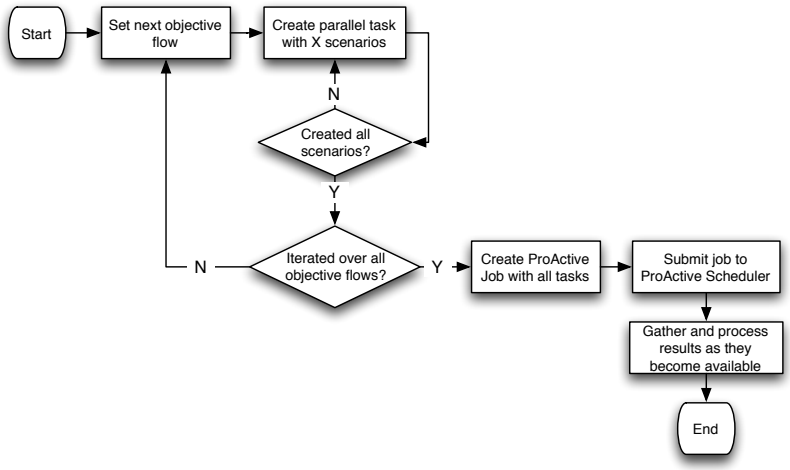

Figure 7: Flow diagram representing the parallelization algorithm for performing the pump profiling.

The implementation with the *ProActive* Scheduler is straightforward: all the parallel tasks are aggregated into a job and then submitted to the *ProActive* Scheduler. The Scheduler is responsible for deploying the tasks on the Grid machines, for retrieving the results from each task and returning them back to the user machine where the simulation was launched.

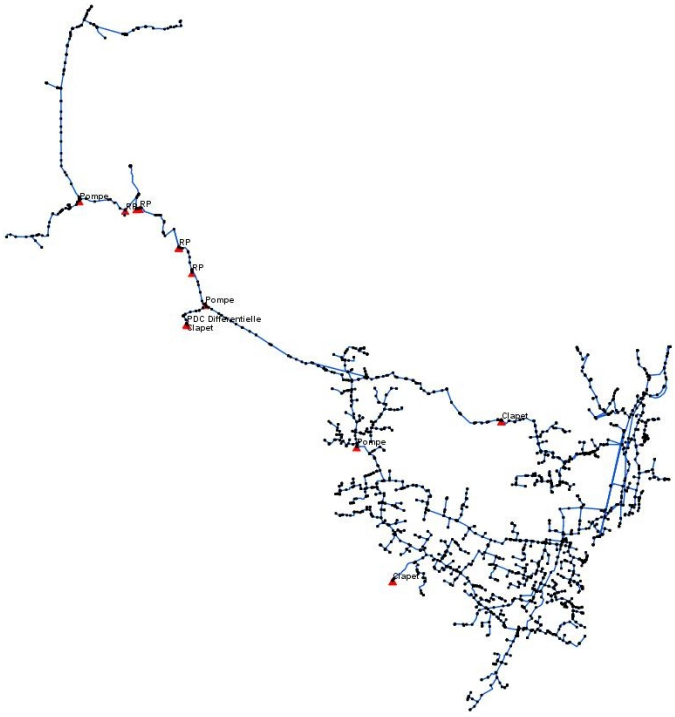

Figure 8: Overview of the downstream networks from *Bouteillère*.

6.5. Parallel results

This section presents the benchmarks performed on PACA Grid for validating the parallel pump profiling. PACA Grid is a set of machines (1 368 cores, 30 TB, 480 CUDA cores) accessible via *ProActive Parallel Suite* tools. The Cloud aggregates dedicated machines, running both Linux and Windows.

The network file used for this profiling models the downstream networks from *Bouteillère* (Fig. 8) and it contains:

- a total pipe length of 190 km.
- 2832 nodes.
- 5 tanks.
- 4 distinct probability zones (total of 28 classes of probability)
- 6 downstream pressure regulators.
- 4 pumping stations.
- 2 security valves.
- 2 differential head losses.

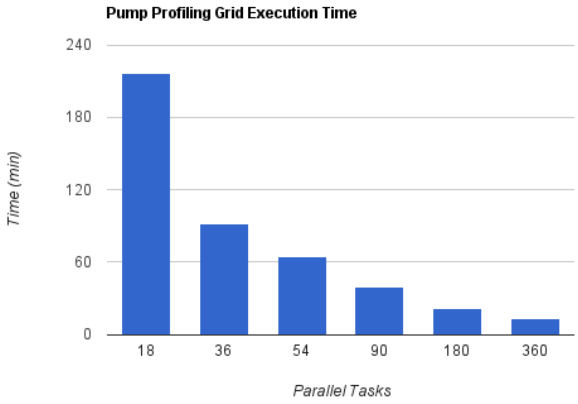

Figure 9: Pump profiling execution time of the downstream networks from *Bouteillère* in function of the number of parallel tasks.

The total sequential execution time of this pump profiling using one machine from the Grid is: **25 hours and 23 minutes**. Figure 9 displays the obtained times by splitting the scenarios into different number of parallel tasks: from 18 up to 360 tasks (the initial number 18 comes from the 18 different objective flows values that are taken for this network and the remaining values are taken as multiples of 18).

The improvement in the performance drastically reduces the execution time: from **25 hours** sequentially to **13 minutes** using the Grid (360 parallel tasks). These test were repeated at least 5 times, the values represent typical execution times obtained when there are enough resources available to simultaneously run all the parallel tasks.

Figure 10 presents the obtained parallel *speedup* compared to the corresponding sequential version.

7. Final considerations

Modern technologies often lack efficient numerical libraries for scientific computing and can also add some overhead (such as garbage collection and dynamic bound checking) that can be crucial to the performance of numerical applications. However some applications are only partially computing intensive and a mixed approach might be the most appropriate: with the use of a high level language for implementing the application and performing calls to low level language for the numerical code.

This paper presents a use case of a high performance scientific application using the Java language and the successful exploitation of a Grid environment in order to reduce the execution runtime. The Java version of IRMA is being currently

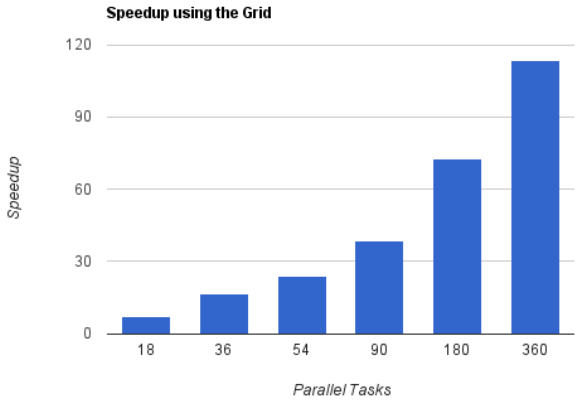

Figure 10: Obtained speedup by number of parallel pump profiling tasks executed in parallel on PACA Grid.

used to design and maintain 6.000 km of *WDN* by the Société du Canal de Provence and, besides outperforming the legacy FORTRAN version for sequential simulations [citation removed], it enables to perform distributed simulations such as the pump profiling described in this paper.

The use of Java enables to easily integrate the application with modern infrastructures (such as graphical/web interfaces, GIS, data bases, . . .) while the FORTRAN backend allows achieving good performance for the numerical resolution. Java also enables a straightforward integration with *ProActive*, that allows easily deploying distributed simulations over the Grid.

[1] L. A. Rossman, EPANET User's manual, U.S. Environmental Protection Agency, Risk Reduction Engineering Laboratory, Cincinnati, OH (2000).

[2] E. Todini, S. Pilati, A gradient algorithm for the analysis of pipe networks, Research Studies Press Ltd., Taunton, UK, 1988, pp. 1–20.

[3] R. Clément, Calcul des débits dans les réseaux d'irrigation fonctionnant à la demande, in: La Houille Blanche, Vol. 5, Société Hydrotechnique de France, 1966, pp. 553–576.

[4] G. P. Pezzi, E. Vaissié, Y. Viala, B. Grawitz, F. Bonnadier, P. Gourbesville, High Performance and Grid Computing Based Hydraulic Simulations Using the Clement Formula, in: Computing and Control for the Water Industry 2011, Centre for Water Systems, Exeter, United Kingdom, 2011.

[5] L. Baduel, F. Baude, D. Caromel, A. Contes, F. Huet, M. Morel, R. Quilici, Grid Computing: Software Environments and Tools, Springer-Verlag, 2006, Ch. Programming, Deploying, Composing, for the Grid.

[6] Mike urban website, http://www.mike-by-dhi.com/ (2014).

[7] Porteau website, http://porteau.irstea.fr/ (2014).

[8] Jama website, <http://math.nist.gov/javanumerics/jama/>. (2014).

[9] Colt project website, <http://acs.lbl.gov/software/colt/>. (2014).

[10] Mtj website, <http://code.google.com/p/matrix-toolkits-java/>. (2014).

[11] C. L. Lawson, R. J. Hanson, D. R. Kincaid, F. T. Krogh, Basic linear algebra subprograms for fortran usage, ACM Trans. Math. Softw. 5 (1979) 308–323. doi:http://doi.acm.org/10.1145/355841.355847.
URL http://doi.acm.org/10.1145/355841.355847

[12] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, D. Sorensen, LAPACK Users' Guide, 3rd Edition, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1999.

[13] Netlib java website, <http://code.google.com/p/netlib-java/>. (2014).

[14] Basic linear algebra subprograms (blas) website at netlib, http://www.netlib.org/blas/ (2014).

[15] J. J. Dongarra, C. B. Moler, J. R. Bunch, G. Stewart, LINPACK Users' Guide, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1979.

[16] S. Eisenstat, M. C. Gursky, M. H. Schultz, A. H. Sherman, Yale sparse matrix package, ii. the nonsymmetric codes, Tech. rep., Department of Computer Science, Yale University (1977).

[17] W. Gropp, E. Lusk, A. Skjellum, Using MPI: Portable Parallel Programming with the Message-Passing Interface, MIT Press, Cambridge, MA, 1994.

[18] I. Park, M. J. Voss, S. W. Kim, R. Eigenmann, Parallel programming environment for OpenMP, Scientific Programming 9 (2001) 143–161.

[19] M. Aldinucci, M. Danelutto, P. Kilpatrick, M. Torquati, Fastflow: high-level and efficient streaming on multi-core, in: in Programming Multi-core and Many-core Computing Systems, ser. Parallel and Distributed Computing, S. Pllana, 2012, p. 13.

[20] G. P. Pezzi, High performance hydraulic simulations on the grid using java and proactive, Ph.D. thesis, Université de Nice-Sophia Antipolis. Faculté des sciences. (2011).