DYNAMIC CAPE-OPEN SIMULATION APPROACH ON CLUSTER ORIENTED ARCHITECTURE

Laurent Pigeon, IFP, Solaize, France
Pascal Roux, IFP, Solaize, France
Bertrand Braunschweig, IFP, Rueil Malmaison, France
Thierry Gautier, INRIA, Grenoble, France

Abstract: *Process simulations are more and more time consuming. One of the methods that allow reducing the computational time of an application is to distribute the workload onto several computers. In this paper, we present some techniques from parallel and distributed computing applied to CAPE-OPEN compliant dynamic process simulations. We have implemented a fully transparent easy-to-use prototype which noticeably reduces the computational time of an offshore production network while running on five computers.*
Keywords: *cluster of PCs, distributed simulation, HPC, CAPE-OPEN, dynamic simulation*

Introduction

Process simulations become increasingly complex in terms of physico-chemical properties that unit operations can simulate. Plugging into process simulator complex unit operations from heterogeneous sources becomes essential. In order to do so, unit operations components have to match required interface specifications so that they can be easily integrated into legacy simulators. Since 1995, the CAPE-OPEN standard, now maintained by CO-LaN (http://www.colan.org), offers business interface specifications to answer the need of interoperability between heterogeneous codes. The CAPE-OPEN standard addresses steady-state as well as transient simulations.

As a matter of fact, computer aided process simulations are more and more time consuming. Thus, process simulations can be regarded as a trade-off between accuracy and execution time. Given the fact that computational power is still a bottleneck to process simulations, accuracy is decreased to reach a whole acceptable computation time. A feasible solution that is well accepted in High Performance Computing is the use of cluster of workstations.

A cluster oriented architecture, also generically called distributed memory architecture, is defined as a collection of interconnected standalone computers (also called node) co-operatively working together as a single computing resource. Nevertheless, the implementation of efficient algorithms that exploit the overall computing power is difficult and generally requires computing knowledge. The need to parallelize codes becomes a reality while considering current personal computers with multi-core CPUs.

In this paper, we present an extension of the INDISS software from RSI that allows distributed execution of process simulations. This extension reduces the total execution time of a simulation and is completely transparent for the final user. Even though methods and concepts can be applied to any CAPE-OPEN compliant simulator, our extension is strongly coupled to the INDISS runtime.

Thanks to the CAPE-OPEN standard, running a simulation onto a cluster of PCs can be now fully automated without user interaction. Given a number of CAPE-OPEN compliant software components are seen as computational atomic entities, our approach aims at providing an optimal placement of such components to reduce the overall completion time, also called *makespan*. Although this approach can be applied to any flowsheet, we test our computational environment on an offshore deepwater production network. This test case is especially instructive due to the complex modules (in terms of UOs and compositional thermodynamics) composing the flowsheet and the efficiency especially requires while evaluating transient simulations.

Previous works have studied and quantified the benefits to use parallel and distributed architectures. [1] reviews the first attempts of using supercomputers and we refer the reader to this paper for a historical view of parallel CAPE simulations. Nowadays, most works study parallel solvers based on frontal algorithms [4, 5]. Such works need an equation oriented approach either for the whole process flowsheet or for unit operation solving approaches. Some coupled works [2, 3] aim at providing matrix reordering to fully exploit a parallel solver. Although these works give good performance, most simulators do not yet implement equation-oriented approach. [6] presents simulation of dynamic processes on distributed memory architecture, addressing similar issues as our own work. The authors physically divide the process at points that naturally separate the overall plant into distinct subprocesses. Our implementation does not consider physical subprocesses but the process. It means that no specific knowledge about the process is needed: only computing issues are addressed. Nevertheless, both approaches are complementary. We may apply our implementation for each subprocess.

This work is the result of a partnership between IFP and INRIA. It aims at proposing a generic transparent easy-to-use framework to run distributed process simulations. **Institut Français du Pétrole** (IFP) is a French institute for research and industrial development, education, and information for the oil, natural gas and automotive industries. It is one of the major actors supporting the CAPE-OPEN standard. **Institut National pour la Recherche en Informatique et Automatisme** (INRIA) is a public scientific and technological establishment. It aims to network skills and talents from the fields of information computer science and technology from the entire French research system. INRIA has strong knowledge in distributed computation. **RSI** takes part in this project by supplying experience and software in the area of process simulation.

The paper is structured as follows. The second section presents the CAPE-OPEN standard interfaces to run transient simulations. Section 3 introduces some of the main High Performance Computing concepts to reach and sustain performance. Then, we show how these concepts are implemented given process simulation issues. Section 4 shows experimental results given the dynamic computation of oilfield productions networks. Finally, conclusions and final remarks end the paper.

Process Simulations using the CAPE-OPEN specification interfaces

Originally designed to manage steady-state simulations, the CAPE-OPEN standard has recently extended its set of interfaces to handle transient simulations. Although this kind of simulation might be solved using an equation-oriented approach, the CAPE-OPEN standard adopts an original method presented in this section. Steady-state simulations based on the sequential modular approach have been presented in [12].

CAPE-OPEN Main Abstractions

In order to specify a process flowsheeting, the CAPE-OPEN standard defines a so-called Process Modelling Component (PMC). PMCs gather all functional business entities that might take place in a simulation run. Two essential PMCs are (i) the Unit Operation component and (ii) the Physical Property Package which aggregates several components with same properties. The Unit Operation (UO) component exposes the *ICapeUnit* interface, which gives access to all functions that UOs perform in a CAPE-OPEN simulation. The Physical Property Package mainly exposes the *ICapeThermoPropertyPackage* which handles thermodynamic calculations. Apart from these two PMCs, the concept of *MaterialObject* has been introduced to represent the material flow in terms of enthalpy, pressure, temperature, mass flow rate, etc. The *ICapeThermoMaterialObject* is the interface representing a container of physical properties and exposes methods allowing thermodynamic calculations on it.

In order to compose and to launch a simulation, these PMCs are plugged into a simulator, called Process Modelling Environment (PME) in the CAPE-OPEN specifications. Our work extends PMEs to allow distributed execution of all functional business units.

Transient Simulations

Transient simulations, also called dynamic simulations, have justified the extension of the *ICapeUnit* interface. According to the Unit Operation Special Interest Group of CO-LaN, the dynamic approach can be implemented using a simultaneous modular approach; such an approach is implemented in the INDISS and D-SPICE commercial software respectively from RSI and Fantoft.

This non-conventional approach can be seen as a three-step loop where each loop represents a time step calculation [7] (see also Figure 1)

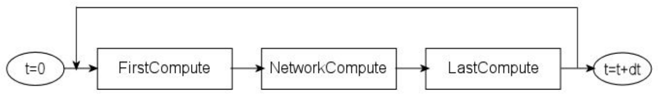

Figure 1 The INDISS solution approach: a sequential 3-steps method.

To solve this approach, the CAPE-OPEN standard defines the *ICapeDynamicUnit* interface. The *StartTimeStep* method informs a unit operation that a time step is being performed. Ordered call on all of the units composing the flowsheet matches the *FirstCompute* step in the INDISS specification. The standard envisages that Boundary Units will update the pressure values on this call. Other units may compute any internal variables that are independent of the flow-pressure calculation. The *ResolveFlowPressure* method, which is the second INDISS step, informs a unit operation that it should resolve its flow-pressure relationships. Given the type of the unit operation, *i.e.* arc or node, the standard defines methods to publish their flow-pressure equations to the network solver. An arc is a unit operation without time lag. It sets an equation that calculates flow given inlet and outlet pressures. A node is a unit operation with a small time lag. It sets an equation that calculates its compressibility given its pressure. To represent this, the CAPE-OPEN standard extends the *ICapeDynamicUnit* interface by the *ICapeArcDynamicUnit* and the *ICapeNodeDynamicUnit* interfaces. To get back the total flow and its first derivative with respect to inlet and outlet pressures, the *ICapeArcDynamicUnit* interface exposes the *GetFlowAndDerivatives* method. On the other side, the *ICapeNodeDynamicUnit* interface provides the *GetCompressibilityAndDerivatives* method which obtains the compressibility function and its first derivative with respect to pressure. These dependences are summurized by the UML class diagram shown in **Figure 2**.

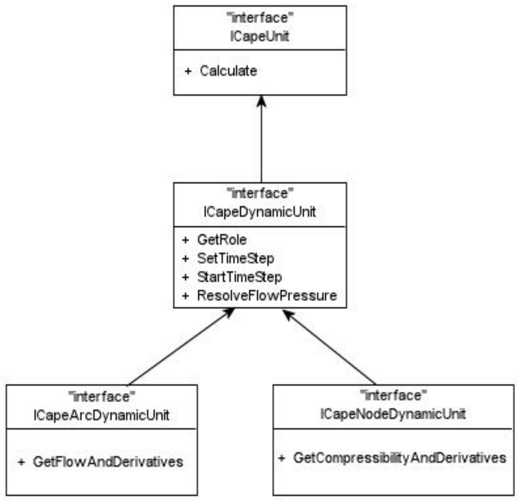

Figure 2 Extension of the ICapeUnit interface, UML Class Diagram [7].

Used in INDISS runtime PME, these methods allow computing the mass balance in an equation oriented fashion. Apart from these new methods, each unit operation always exposes the *Calculate* method. So, the last step (*LastCompute*) which computes the energy balance can be achieved.

High Performance Computing

The key idea behind the High Performance computing is to fully exploit parallel computing architectures. Parallel computing aims at solving a problem by resolving independent sub-problems on different processors. This parallel evaluation allows reducing the computational time compared to a single computer executing the same pieces sequentially. It is important to note that even though several processors are used, a parallel computation might be less efficient compared to a single computer execution due to inherent synchronizations between processors. This section outlines key concepts that have to be implemented in order to fully exploit the power of the underlying computational architecture.

Concept

Cluster oriented architecture generally requires algorithms that exploit coarse-grain parallelism, because of the delay to access distant memory (using the network), several orders of magnitude slower than local memory access. The main methods to exploit processors are: 1/ to overlap communication delay by computations; 2/ to minimize communications by scheduling tasks among the processors. The former is done by implementing multithreading and asynchronous (non-blocking) communication primitives. Within middleware such as CORBA, which is used by CAPE-OPEN standard, the asynchronous method invocation (AMI [8]) allows non-blocking remote invocations. The latter method is based on scheduling techniques which allow computing the mapping of the tasks of an application on distributed memory architectures. Such techniques are based on graph theory and require that the program is represented as a precedence graph which models its execution.

Execution Graph

All scheduling algorithms are based on an abstract representation of the application, seen as a graph. Formally speaking, a precedence graph is defined as a directed graph $G = \{V, E\}$ where V is a finite set of vertices and E is a set of edges representing precedence relations between vertices. The vertex set consists of computational tasks, as seen in the traditional context of task scheduling, and the edge set represents the data dependencies between the tasks.

Given the precedence graph of an application, a direct assumption may be exposed. The width shows the potential parallelism of the application and thereby the applications associated with wide graphs are more suitable to be launched on distributed memory architectures.

Scheduling

The precedence graph is fundamental in mapping computation onto the available resources of a cluster. A *scheduler* is a program that computes the mapping of each task on a processor in accordance with a given objective function to minimize. A classical function is often the *makespan*: given a precedence graph, a scheduling algorithm provides logical partitions which will be mapped onto physical computational nodes. Each partition is a set of tasks for which a start date is given.

In order to calculate an accurate scheduling, the scheduler needs the acyclic precedence graph weighted by cost information (computation cost for each task and communication cost for each edge). This scheduling problem, in its generality, is a *NP-hard problem* [14]. Thus heuristics have to be used to compute approximations to the optimal solution.

Our Approach

This section presents add-ons to INDISS that have been made in order to automate distributed computation on a cluster of workstations, as presented above.

From Flowsheet to Execution Graph

Given a particular flowsheet, a first task is to convert it into a precedence graph allowing its partitioning and its deployment on several PCs. The flowsheet is a business abstraction of the suitable program execution. Depending on the solving approach used (*i.e.*, sequential modular, simultaneous modular), the execution graph will differ. Nevertheless, the CAPE-OPEN standard is based on component techniques which define a software component as a deployment unit. Then, a component has its own context of execution as well as its own internal states. Even though the CAPE-OPEN standard defines the *ICapePropertySetPersist* interface which allows saving and restoring internal states of objects deriving from this interface, too much work has to be done and a too large amount of data has to be sent in order to transfer a unit operation from one computational node to another one. Thus, we define the precedence oriented graph as the flowsheet following the direction of the material flow.

In the sequential modular approach, it is important to note that if design simulation or if flowsheeting simulation problems (where control or convergence loops) are considered, business tearing algorithm are used to come down to an acyclic graph.

Implementation of the HPC concepts

Our software architecture is based on a master-slave approach. INDISS is the master of the simulation. It acts as a CAPE-OPEN compliant PME enhanced with specific functionalities handling the distributed computation. These functionalities are:

1. The discovery of the distant computational nodes;
2. The application scheduling using DSC [10] or ETF [11] schedulers;
3. The flowsheet initialisation on each distant node;

4. In case of dynamic simulation, solving the mass balance using a centralized solver.

Since some unit operations are not CAPE-OPEN compliant and are coupled to the PME, the master also takes part in the simulation by running unit operations from a logical partition plus the native unit operations.

Slaves have to run a set of unit operations defined by the scheduler. Each slave has its own thermodynamic calculation server.

Master and slaves are CORBA components as shown in Figure 3. Slaves may be regarded as unique complex unit operations which exhibit extended CAPE-OPEN methods. While invoking such an extended method, our extension calls it on the whole set of unit operations it stands for. Then message aggregation is defined on control flow (which refers to the order in which methods have to be performed in order to preserve results from a sequential computation) as well as on data flow (which actually is exchanged data). During execution, an extended unit operation might need data from another distant one. Thus, during the simulation execution, each slave maintains a list of "ready" tasks. A "ready" task is a computational task for which all inputs are produced following the precedence constraints. There is no need to keep a centralized manager of the control flow. When two distinct computational nodes share the same material flow, this CAPE-OPEN Material Object is duplicated on both nodes [12]. When the unit operation producer finishes its computation, its slave sends the whole Material Object to the slave which consumes it. Then, the duplicated Material Object is updated and the outlet unit operation may become ready if all of the others inlet Material Objects are up to date. In case of dynamic simulation, while computing the second step, full Material Objects are not required: only flows and conpressibilities with their first derivatives are required to compute the Jacobian matrix. Thus, only these values are sent to the centralized solver. It is important to note that given the duplicated Material Object concepts, flows and pressures are updated while the *ResolveFlowPressure* method is evaluated on the set of arcs composing each partition. These values are used while computing the *ResolveFlowPressure* on nodes.

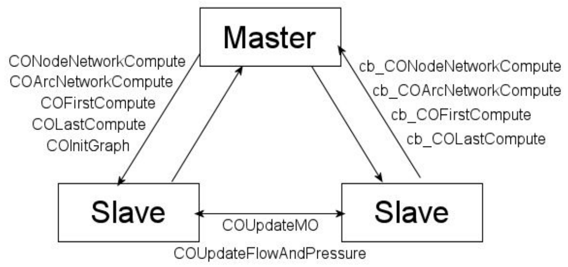

Figure 3 Software architecture: a master-slave approach.

Even though the CORBA standard defines asynchronous message invocation, the omniORB implementation that we use does not yet implement it. To manage the communication-to-computation overlapping, in our implementation each extended CAPE-OPEN method uses the *oneway* attribute which defines non-blocking remote method invocation. The client process has to implement the callback method in compensation. On Figure 3, these methods are symbolized using the "*cb*" prefix.

Experiments

Our extension to INDISS consists in around 5000 lines of source code. Performance results were experimentally determined on a business test case from IFP, IFP and RSI providing the business entities. In this paper we only present the placement resulting from using the DSC algorithm. All experiments were conducted on a small Windows-based cluster composed of five 733 MHz PIII PCs with 256 MB of RAM interconnected with FastEthernet link (100 Mbit / s).

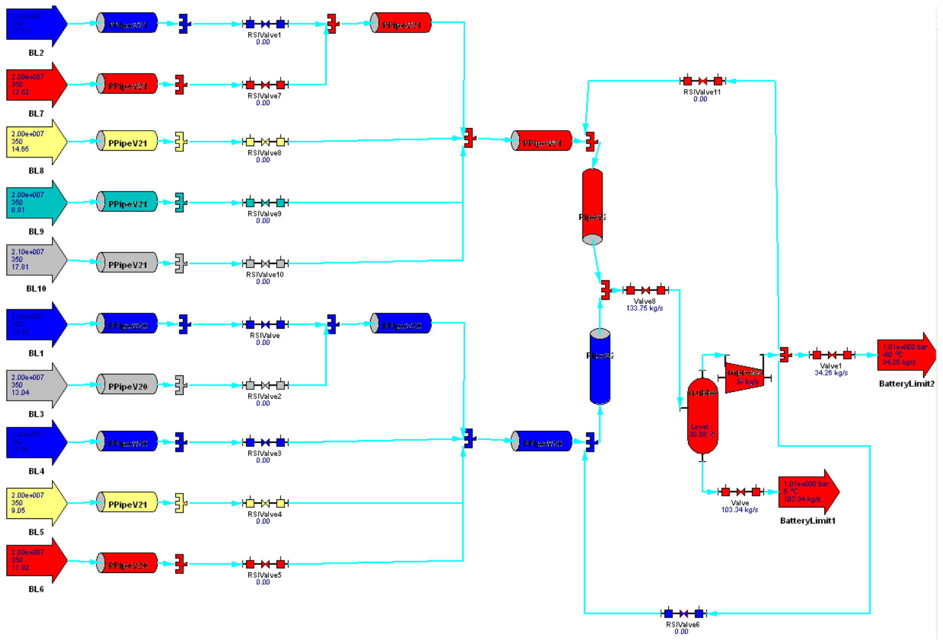

Figure 4 Business Test Cases: a production network.

The business test case simulates the production of a pipe network. In its dynamic feature, the simulator aims at validating operating condition such as shutdown, restart, cool down, etc, and at estimating production instabilities (severe-slugging, terrain-slugging). The process flowsheet is presented in Figure 4.

Running this simulation in a dynamic fashion, the overall computational time may change if we introduce perturbation into the model. The first time step is time-consuming due to the mathematical model of the Pipe unit operation using a compositional thermodynamics and full hydrodynamics. Given the solving approach where the mass balance is computed using the states from the previous time step, the solver gives unstable values and pipe unit times greatly vary. After a few time steps, the network converges and the whole computational time remains stable. This "steady-state" behaviour will continue until a perturbation is introduced. Such a perturbation may simply be a valve opening. In this case, as previously, the pipe model is disturbed and the network needs several time steps to absorb it in order to get back its steady-state rate.

Results

Figure 5 presents a comparison of computational times of 90 time steps between a single computer and a five computers distributed executions running a dynamic simulation. To assess the performance using our computational environment, the figure also presents the lower bound of execution times. This lower bound is computed by summing unit times of each unit operation from each partition, the minimum time is then the lower bound. To quantify the performance, the speedup metric S_p is chosen. It is defined by $S_p = T_{seq} / T_{//}$ where T_{seq} and $T_{//}$ are computational times respectively from a single computer computation and a distributed computation.

To value the distributed behaviour in case of perturbation injection, we reduce a valve opening from 8% to 5% at iteration 21.

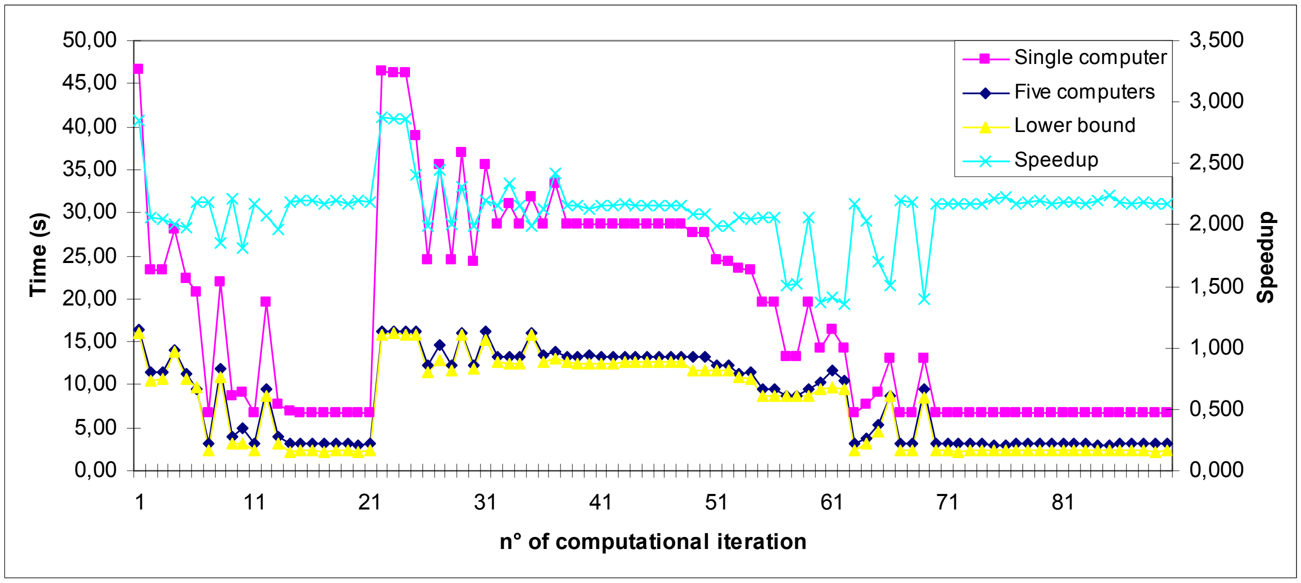

Figure 5 Comparison of overall execution times and of the seepdup given a time step.

Experimental results show that our distributed computational environment induces a small overhead in the range of 0.74 seconds. Considering the steady-state behaviour, the speedup is constant and is of 2.2. During the unstable stage while disturbing the computation, the speedup is at its top and reaches its peak at 2.9. *A contrario*, at the perigee the speedup is about 1.4. This interval comes from the solver instability and the linearized model used inside pipe units. The solver may propose pressures being out of range of previously computed values, and then the pipe has to recalculate the new values for this range of pressures. Nevertheless, this processing does not occur on all pipes composing the flowsheet and most of the work may be processed by only one computational node of the cluster. Thus, the scheduling does not balance well the workload during these unstable phases. The computational time when such a processing happens differs by four orders of magnitude.

Conclusions

Clusters of workstations are no more for academic use. These architectures are now effective and less expensive compared to previously supercomputers such as Cray vector computer. Flowsheeting process engineers need increasing computational power for representing the complex reality with greater accuracy. Nevertheless, process engineers are not computer engineers and do not want to manage software implementation issues on parallel computers.

In this paper, we have shown that HPC concepts can be applied to process simulation without intervention from process engineers. Using our implementation means to use a common simulator, to restore or to design a process flowsheet and to launch the simulation as usual. We have shown, based on experimental results gathered from a business test case with dynamic execution, that some time savings can be achieved.

In previous work [12], we have presented results with a sequential modular approach to solve steady-state simulations. In this paper, we focussed on transient simulations where a novel solving approach is used. Nevertheless, as described in [13], such approaches will not scale well using parallel computation due to the sequential nature of some flowsheets. The overall performance is limited by the structural parallelism that can be obtained with the flowsheet. To reach better performance, parallelism within unit operations computation should be introduced.

References

1. Moe, H. I. and Hertzberg T. (1994), "Advanced computer architectures applied in dynamic process simulation: a review", *Computers & Chemical Engineering*, 18(Suppl.): S375-S384.
2. Hu, Y.F., Maguire, K. C. G. and Blake R. J. (2000), "A multilevel unsymmetric matrix ordering algorithm for parallel process simulation", *Computers & Chemical Engineering*, 23(11-12): 1631-1647.
3. Mallya, J. U., Zitney, S. E., Choudhary, S. and Stadtherr M. A. (1999), "Matrix reordering effects on a parallel frontal solver for large scale process simulation", *Computers & Chemical Engineering*, 23(4-5): 585-593.
4. Zitney, S. E., Mallya, J., Davis, T. A. and Stadtherr M. A. (1996), "Multifrontal vs frontal techniques for chemical process simulation on supercomputers", *Computers & Chemical Engineering*, 20(6-7): 641-646.
5. Scott, J. A (2001), "The design of a portable parallel frontal solver for chemical process engineering", *Computers & Chemical Engineering*, 25(11-12): 1699-1709.
6. Smith, F. G. and Dimenna, R. A. (2004), "Simulation of a batch chemical process using parallel computing with PVM and Speedup", *Computers & Chemical Engineering*, 28(9): 1649-1659.
7. Paen D., Cameron D. and Colantonio M.C. (2005), "Enhancing Process modelling Environments: CAPE-OPEN Dynamic Unit Operations", in *Proceedings of 7th World Congress of Chemical Engineering*.
8. The Object Management Group (2002), "The Common Object Request Broker: Architecture and Specification v. 2.6.1".
9. Graham, R. L. (1969), "Bounds on Multiprocessing Timing Anomalies", in *SIAM Journal of Applied Mathematics*, 17(2): 416–429, 1969.
10. Yang, T. and Gerasoulis, A. (1994), "DSC: Scheduling Parallel Tasks on an Unbounded Number of Processors", in *IEEE Trans. Parallel Distrib. Syst.*, 5(9): 951–967.
11. Hwang J.-J., Chow Y.-C., Anger, F. D. and Lee, C-Y. (1989), "Scheduling precedence graphs in systems with interprocessor communication times", in *SIAM J. Comput.*, 18(2):244-257.
12. Pigeon, L., Testard, L., Gautier, T. and Roux, P. (2005), "Towards a Distributed High Performance Computing Environment for CAPE-OPEN Standard", in *Proceedings of 7th World Congress of Chemical Engineering*.
13. Vegeais, J. A. and Stadtherr, M. A. (1992), "Parallel processing strategies for chemical process flowsheeting". *AIChE Journal*, 38(9): 1339-1407.
14. Leung, J, Kelly, L. and Anderson, J. H. (2004), "Handbook of scheduling: algorithms, models, and performance analysis", Joseph Y-T Leung (ed), Chapman Hall/CRC Press.