18th European Symposium on Computer Aided Process Engineering – ESCAPE 18
Bertrand Braunschweig and Xavier Joulia (Editors)

A framework for analysis of computational load of CAPE tools

Pablo A. Rolandi[a], Alejandro Cano[b]

[a]*Process Systems Enterprise, 107a Hammersmith Bridge Road, London W6 9DA, UK*
[b]*Process Systems Enterprise, 2 Ridgedale Avenue, Cedar Knolls, NJ 07927, USA*

Abstract

This work presents a framework for analysis of computational load centred on three components: i) the model of the system (the model), ii) the model of the mathematical/engineering problem (the problem) and iii) the numerical solution method(s) used for solving the combined model/problem formulation (the solver(s)). The framework classifies computational load as emerging from four mayor elements: i) expression (function/residuals and Jacobian) evaluations encapsulated in the model, ii) solver calls, iii) foreign services external to the core modelling-and-solution engine, and iv) overheads. The framework is used to analyse the computational cost of two simulation case-studies.

1. Introduction

The last decade witnessed the reduction of major commercial software providers of CAPE tools and consolidation of their process modelling packages. The (Global) CAPE-OPEN international and industrial project targeted the development of open software architectures for computer-aided process modelling and simulation. The results of this initiative provided a framework that formalised the notion of a model-server, a software abstraction of the information and mechanisms required by model-clients, enabling the division of models, solution algorithms and model-based applications. Today, this framework promoted by the CAPE-OPEN standards is the backbone of state-of-the-art modelling and solution engines such as gPROMS (PSE Ltd).

As models grow in size, complexity, fidelity and predictive accuracy, they also become more computationally expensive to solve. Unfortunately, it is difficult to predict the impact that changing the characteristics of a model, the nature/configuration of the solution algorithms or the formulation of the corresponding engineering/mathematical problem may cause on the final computational speed. Overall, model developers, model users and application developers recognise the lack of a unifying and systematic study on the factors affecting the computational cost of models of arbitrary complexity implemented in CAPE tools. Fortunately, the dissemination of the CAPE-OPEN standards and adoption of its proposed software architecture at the core of most advanced process modelling technologies provide the starting point for developing a general framework shedding some light into the topic of computational load. In this work we present such a framework.

2. Framework

2.1. Fundamentals

In order to assist model and (model-based) application developers to identify the factors affecting the computational performance of their systems, we propose to view these factors from two different perspectives: i) the fundamentals, which are (abstract)

mathematical, numerical analysis and engineering concepts that ultimately determine key aspects that define, to a large extent, the computational load of model-based activities; ii) the modelling technologies and tools, which, by providing the platform for development and solution of mathematical models and deployment of the corresponding model-based applications, also determine, to some degree, the overall computational performance of model-based applications. Due to space constraints, we are not able to discuss computational load from the perspective of technologies and tools in this work.
In the category of fundamentals we combine elements that are largely independent of technological aspects and, therefore, are more prone to theoretical analysis. Inspired by the framework promoted by the CO standards, it is both natural and convenient to divide the "fundamentals" into: i) the mathematical description of the process model ("the model"); ii) the formulation of the engineering problem ("the problem"); iii) the algorithms that derive a solution of the aforementioned problem ("the solution method"). In the following sections we elaborate these concepts.

2.1.1. Model

The following characteristics of a model are relevant from a computational load viewpoint:

- the class of system of equations: for instance, algebraic, ordinary differential, differential and algebraic, and (integro-) partial-and-ordinary differential and algebraic systems are the most common mathematical formalisms upon which mechanistic models of natural phenomena are expressed.
- the type of system of equations: for example, explicit, semi-implicit and implicit systems, are the most common.
- the degree of nonlinearity of the system.
- the nominal size and degree of sparsity of the system of equations.
- the degree of continuity of a predominately continuous system of equations: since physical systems arise spontaneously in either purely continuous/discrete or hybrid continuous/discrete forms.
- the degree of redundancy of a subsystem of equations.

In general, not only structural characteristics of a given model will "induce" the computational load of a model-based application but also the numerical values of variables and equations. In practice, this is mostly given by: i) the initial guesses of the system of variables (instanciation); ii) the magnitude of the system of variables and equations.

2.1.2. Problem

The concept of "model" presented in the previous section is entirely contained in a larger mathematical entity: the formulation of the engineering problem. In other words, we argue that in practice we do not solve a mathematical model of a physical process/system but rather the mathematical model of a physical problem subject to the mathematical representation of the process/system.
In the field of PSE, most process engineering problems entail the solution of simulation and optimisation activities. In the case of simulation runs, for example, the reconstruction of (input forcing) process trajectories from discrete process-instrumentation/control-system data and characterisation of the system's initial state are two of the most challenging engineering matters to address. In the case of optimisation problems, the form of additional operative constraints, as well as the formulation of infinite-dimensional controls and constraints into finite-dimensional approximations deserve further attention. Since the distinction between the model and the problem may be difficult to grasp, these concepts are illustrated more rigorously in Eq. 1.

Model	Simulation "problem"		Optimisation "problem"		
			$\min_{U} g(x)$		
$F(\dot{x},x,y,u)=0 \ \forall t$	$F(\dot{x},x,y,u)=0$	$\forall t \in [0,t_f]$	s.t. $F(\dot{x},x,y,u)=0$	$\forall t \in [0,t_f]$	(1)
	$I(\dot{x},x,y,u)=0$	$t=0$	$I(\dot{x},x,y,u)=0$	$t=0$	
	$u(t)=U$	$\forall t \in [0,t_f]$	$u(t)=U$	$\forall t \in [0,t_f]$	
			$H(\dot{x},x,y,z) \le 0$	$\forall t \in [0,t_f]$	

2.1.3. Solution method

An algorithm is needed to derive a solution of the overall mathematical system (which from now on we shall call the "model/problem ensemble"). Numerical algorithms exploit one or more properties of the mathematical system to achieve this goal. Hence, a bad choice of solution algorithm may mask the "intrinsic" computational load of a model/problem ensemble by imposing an unusually high computational cost. Trade-offs between robustness, accuracy and efficiency of solution may be fine-tuned by controlling the behaviour of the algorithms by a series of solver parameters.

2.2. Structure and indicators

The notions of model, problem and solution method provide the fundamental grounds over which computational cost and load can be evaluated, examined and understood. In the following sections we discuss specific characteristics of the proposed framework.

2.2.1. Indicators:

It is hard to suggest a metric for computational speed representative of the processes taking place throughout the solution of a model-based activity. While conventional measures of computational cost such as the big-O notation and number of FLOPs are appealing for theoretical analyses, they fail to provide any practical insight into real-world examples. In this work, we adopt a more natural approach and we report computational cost in the form of the so-called "speed factors" denoting a relative measure of computational cost. These statistics are calculated as the ratio between the total execution time and the characteristic time of the run (for instance, the integration horizon in a simulation run or the control window in a moving-horizon application). Because this speed factor (SF) is adimensional, we consider it an "extensive" measure of speed (and denote it by the acronym ESF); the "intensive" counterpart is simply the computed adimensional speed per degree-of-freedom (i.e. per number of unknown variables) (denoted by the acronym ISF). Finally, the relative computational load is reported as the fraction of total time spent in a given computational process (in percentage). The ESF and ISF and loads are reported in Table 1 for the case studies presented in this work.

2.2.2. Task clusters: macro- and micro-structures

In order to construct a framework that is useful for the purpose of computational load evaluation and analysis, it is particularly convenient to present common calculation processes in the form of "task clusters". Due to the complexity and diversity of tasks, the framework discussed here has been divided in two tiers of different granularity. On the one hand, there is the macro-structure of the framework, which characterises the causes of computational load at a very coarse level of detail; four major tasks clusters are proposed in this work (see below). In addition, a framework's micro-structure has

also been considered, although space constraints prevent us from presenting these in detail. However, a few micro-structure indicators will be presented for the solvers to facilitate the discussion of results (e.g. integration-step and corrector-iteration statistics). In this work, we propose the following coarse compartmentalisation of computational processes that contribute to computational load:

1. *expression evaluations* (EE): arising from the model/problem ensemble, they are independent of numerical solution algorithms but depend on the form in which expressions are represented (e.g. modelling/programming language internal data structures). Hence, if a solver changes, the individual cost of expression evaluations is expected to remain constant. Furthermore, we find it attractive to divide the whole set of expression evaluations into those arising from model equations (c.f. residual/function evaluations) (RE) and those from derived expressions such as the Jacobian (c.f. Jacobian evaluations) (JE). Expression evaluations are mostly related to the model definition but not the solution methods as one would expect from the notion of "model-server".
2. *foreign-service calls* (FS): resulting from the model, these are physical property packages (PP) and other computation services foreign to the core modelling-and-solution engine which interact with the modelling-and-solution kernel via well-defined mechanisms. The incentive for grouping these modelling mechanisms in a separate task cluster is straightforward: within the CAPE-OPEN framework, PPs are specialised process-modelling components (PMCs) providing services to higher-level PMCs (e.g. unit operations); because of their wide use, specialisation and domain-specific implementation details, it is convenient to group them separately.
3. *solver calls* (SV): these are the structural/numerical solution algorithms that are expected to be expression-independent. Hence, if the model-server changes (e.g. a different model-server engine is used or simply another version of the same engine), the cost of a given solution path is expected to remain unaffected.
4. *overheads* (OH): inevitably, the technological choices made in the design, development and implementation of any a modelling-and-solution engine will give rise to different "overheads" occurred in the overall computation process but not arising from the aforementioned expression evaluations, solver and foreign-service calls. An example of these is I/O and memory allocation/indirection, among others.

3. Case studies

The case-studies presented in this work are based on models available from the standard gPROMS distribution. The first system is a tubular reactor distributed along axial and radial domains with two components and a nonlinear reaction term giving rise to a IPDAE system. In the base case (CS1), the initial conditions are given by a constant profile, and the forcing inputs are so that the system evolves undisturbed for 5 time units [sec]. The axial coordinate is discretised by a backward finite-difference scheme of order 1 using 50 uniformly distributed elements, and the implicit integrator is based on a BDF scheme. In CS2, an IRK scheme is used as implicit integrator, while in CS3 the finite-difference method is of order 2; finally in CS4 the initial conditions are at steady-state, and again the system evolves undisturbed for 5 time units. The systems is of size 4668 with 736 differential variables and 17658 non-zero elements; its characteristic time is 5 seconds.

The second system is a water/methanol distillation column, given by a hybrid continuous-discrete DEA system. In the base case (CS1) (as well as CS2), the initial conditions are a linear profile, and the forcing inputs represent an undisturbed operation

for 1000 time units [sec] with no feed and total reflux, followed by undisturbed operation for 5000 time units at constant feed and fixed reflux ratio. In CS3 and CS4, the initial conditions are at steady-state (at conditions above), and the inputs show a feed reduction by 50% and undisturbed for 6000 time units. In CS1 and CS3 a BDF integrator is used, while in CS2 and CS4 an IRK integrator is chosen. The system is of size 1938 with 103 differential variables and 7022 non-zero elements; its characteristic time is 5 seconds.

4. Analysis of results

4.1. System 1

CS1 vs CS2: the determining costs in these runs are expression evaluations and solver calls (>90%); CS1 is faster. In the case of CS1, the ratio between solver calls and expression evaluations is approximately 1.9; in CS2, the ratio is 5.7. One would not expect the unit cost of expression evaluations to be different between these two runs; however, the cost of solver calls per step for CS2 is 2.1 times more expensive than that of CS1. The unit cost of the IKR integration scheme is approximately one order of magnitude more expensive than that of BDF integration (0.19 vs 0.022 respectively). Interestingly, the number of steps taken by the IRK method is much smaller than that of the BDF method (23 vs 122, respectively); however, this less than an one-fold reduction is not enough to compensate for the increased unit cost.

Table 1. Computational statistics

		System 1				*System 2*			
		CS1	*CS2*	*CS3*	*CS4*	*CS1*	*CS2*	*CS3*	*CS4*
EE	TRE	0.844	0.531	1.875	0.016	2.703	3.109	0.375	0.250
	TJE	0.047	0.109	0.031	0.000	0.563	0.609	0.109	0.094
	NRE	324	70	601	6	2396	1034	365	83
	NJE	15	23	23	2	337	272	33	33
FS	TPP	0.000	0.000	0.000	0.000	1.047	1.407	0.187	0.109
SV	TSV	2.000	3.703	3.875	1.750	5.250	11.453	1.281	0.875
	NSA	122	23	215	5	1128	302	233	36
	NSF	0	0	1	0	17	22	0	0
	NCI	324	233	601	6	2396	3374	365	282
	NCF	2	0	6	0	52	4	12	3
OH	SYS	0.266	0.031	0.656	0.047	2.625	0.641	0.500	0.063
	CPU	3.156	4.375	6.438	1.813	12.188	17.219	2.453	1.391
	ESF	1.6e0	1.1e0	7.8e-1	2.8e0	4.0e2	3.5e2	2.4e3	4.3e3
	ISF	3.4e-4	2.4e-4	1.6e-4	5.9e-4	2.1e-1	1.8e-1	1.3e0	2.2e0

Time and number of on residual and Jacobian evaluations (TRE, NRE, TJE, NJE), time of physical-property package calculations (TPP), time of solver calls (TSV), number of steps attempted (NSA), number of step failures (NSF), number of corrector iterations (NCI), number of corrector failures (NCF), total system time (overheads) (SYS), total execution time (CPU); extensive- and intensive-speed factor (ESF, ISF), time units in seconds.

CS1 vs CS3: the model under consideration is a distributed parameter system given by a set of IPDAEs; these equations are discretised by the gPROMS kernel according to a user-defined approximation scheme that gives rise to the low-level model representation upon which the solution engine actually operates. Loosely speaking, these case studies have the same problem definition but different "solver configurations". In both cases

expression evaluations and solver calls account for approximately 89% of the overall computational cost. CS3 is approximately 2.0 times slower than CS1; while CS3's costs on expression evaluations are 2.1 times more expensive than CS1, its solver calls are 1.9 times more expensive. The ratio between integration steps taken for CS3 and CS1 is 1.8, while the ratio of corrector iterations is 1.9 (hence, the number of corrector iterations per step is approximately constant for both runs). This explains the increased computational cost of expression evaluations, since one additional function evaluation is needed per corrector iteration. The increased cost of solver calls can be explained as a combination of the cost of factorisations (a ratio of 1.5 in Jacobian evaluations and one additional, more expensive factorisation in the case of RADAU-IIA IKR methods) and the cost of back-substitutions (1.9) at the level of the LA of the corrector computation, plus unaccounted overheads.

CS1 vs CS4: these two simulation studies represent a "plant" at two very distinct operating regimes) and, not surprisingly, they give rise to very different computational loads. This example shows that a "model" does not have an intrinsic computational cost associated to it but only a model/problem ensemble does and, therefore, the trajectories of forcing inputs and values of initial conditions are paramount to a realistic characterisation of computational cost/load.

4.2. System 2

CS1 vs CS2: the cost of expression evaluations, foreign services (PPs) and solver calls only are approximately 34%, 55% & 11% and 23%, 68% & 9% respectively. Overall, this indicates that efficient PP calculations case consume approximately 25% of model-server call costs and 10% of the overall computation load. Note that the number of error test failures increased by one fold in the IRK scheme, which indicates good scope for fine-tuning the step-adaption heuristics.

CS3 vs CS4: the cost of solver calls is marginally larger for IRK methods compared with BDF methods (0.81sec vs 0.78sec), but IRK methods benefit from a reduced number of expression evaluations resulting more computationally attractive in this case.

CS1 vs CS3 and CS2 vs CS4: this couple of simulation studies represent different operating conditions and use BDF/IKR integrators. The cost decrease of BDF-type integration is 6.2 while the cost decrease of IKR-type integration is approximately 12. It follows that the choice of most efficient integration scheme greatly depends on the operating regime; IRK integration seems more favourable for more nonlinear processes.

5. Summary and Future Work

This work is the first step towards the development and validation of a comprehensive framework for evaluation of computational speed of state-of-the-art CAPE tools. The framework aims at shedding some light into the issue of computational load and how choices at the level of engineering/mathematical modelling and numerical solution algorithms can substantially change the computational speed of an application, providing a basis to tackle the analysis and reduction of computational load in a systematic and quantitative way. This contribution is the first of a series of publications addressing the topic of computational cost of model-based activities (including optimisation-based problems) and the impact of process modelling technologies.

6. Acknowledgments

PA Rolandi acknowledges the funding of the Marie Curie Research Training Network under the programme PROMATCH, contract number MRTN-CT-2004-512441.