# System-Level Partitioning with Uncertainty

Jones Albuquerque Claudionor Coelho Jr. Carlos Frederico Cavalcanti
Diógenes Cecílio da Silva Jr. Antônio Otávio Fernandes

Computer Science Department
DCC - ICEx - UFMG. Caixa Postal 702.
30161-970. Belo Horizonte, MG - BRAZIL

**Abstract**

Several models and algorithms have been proposed in the past to generate HW/SW components for system-level designs. However, they were focused on a single designer who had a throughout knowledge of the design. In other words, the decision trade-offs were simplified to a stand-alone developer who did not have to consider individual skills, concurrent development for portions of the design, risk analysis for time-to-market development, nor team load and assignment.

In this paper, we propose a design management approach associated with a partitioning methodology to deal with the concurrent design problems of system-level specifications. This methodology allows one to incorporate the uncertainties related to development at the very early stages of the design, and to follow up during the development of a final product.

## 1 Introduction

System-level design refers to a design methodology targeted to complex systems implemented as a mixture of Hardware/Software (HW/SW) components. In such systems, divide and conquer techniques are usually used to create a conceptual model of the system at very early stages of the design. This conceptual model does not have any information on the actual implementation, but rather estimates on how the behavior will be implemented on its possible technologies (by technology, we do not limit ourselves to single HW or SW modules or components, but on different specialities required to accomplish a design). The ever increasing complexity of such systems and the variety of implementation choices available to the developer required a common methodology where HW and SW tasks were combined at the very early stages of the design in order to accelerate the design process [1].

Large system-level designs are usually implemented by separate teams (maybe accross country boundaries) in order to reduce its time-to-market. Team work adds new complexity, since no single designer has a complete knowledge of the system, and task/team management must be performed before the system is implemented.

In this paper, we propose an approach to analyze and estimate the performance and design time of system-level designs. This approach works before synthesis and performs team analysis during system-level partitioning. The proposed approach is present in Figure 1 and it addresses the following problems:

**Risk analysis and evaluation.** Traditional HW/SW codesign approaches [4, 10, 12] were focused on a single designer, i.e. the decision trade-offs were simplified to a stand-alone developer who did not have to consider individual skills, concurrent development for portions of the design, risk analysis for time-to-market development, nor team load and management. Furthermore, they were not able to use the capability of a development team to estimate and execute a specific task. Traditional approaches presented in the literature do not treat the uncertainties involved in human activity. In other words, most of these models ignore issues such as the accuracy of human estimates and time-to-market.

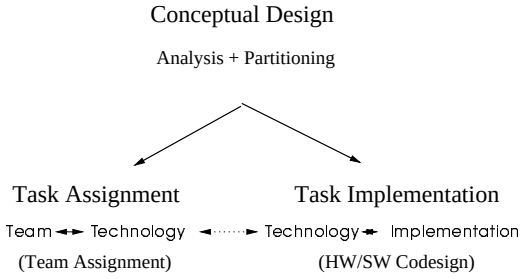


Figure 1: Design views

**System partitioning in presence of uncertainty.** The treatment of uncertainties in design projects are becoming common in system-level design. [11, 5] present partitioning approaches to treat the uncertainties in HW/SW codesign and [8] presents an iterative partitioner using uncertainty. However, they are targeted to a stand-alone developer and not focused on process but rather on implementation uncertainties.

**Task and team management.** Team strategies are common in Software Engineering [13]. In system-level de-

sign we observe that the models that have been presented work on a two axes model, either on Production "Cost x HW/SW Performance", found in codesign systems, where performance is measured by system delay, throughput or power consumption; or on Development "Team x Development Cost", where Team stands for the manpower requirements. These models do not consider the team management for development and task partitioning problems as co-dependent problems, when in fact every team is associated with a specific technology (such as HW or SW) and each technology is associated with an implementation, as presented in Figure 1. As a result, a bad partitioning based solely on performance metrics may produce a design which cannot be implemented with the current resources, and team management based on development metrics may produce infeasible implementations from performance or production cost metrics.

This paper is outlined as follows. Section 2 presents the approach principles, its probabilistic treatment and limitations. Section 3 presents the abstract approach description and how it works. Section 4 provides the mathematical formalism to solve the hardware/software partitioning problem considering the problems mentioned. Section 5 presents the results and section 6 presents the conclusions.

## 2 Probabilistic Characterization of Estimates

The reader should note that our model works at the very early stages of the design, after the system has been partitioned into tasks and given development and implementation estimates. We assume that each team detains the knowledge of only one technology, and that estimates are given based on the team's historical data. We also assume that in zero time no job can be done, and if sufficient time is given, any team can implement the entire system.

Two types of estimates must be treated probabilistically in our work: development estimates and implementation estimates. Development estimates are usually related to the team's ability to execute or implement a specific task. They are usually measured in terms of development time, team load and implementation risks. Implementation estimates are related to production metrics such as execution time or power consumption.

A team estimate for a task can be captured by the 3-tuple $(m, M, c)$, where $m$ is the minimum value, $M$ is the maximum value and $c$ is a *confidence degree*, for example: ("very high", "high", or "low"). These confidence degrees can be changed according to the project and they are based on historical data; if a team has a precise historical data of its tasks then its degree is "very high" and if it has not a historical data its degree is "low", and this confidence degree degrades in absence of historical data or if the team historically has imprecise estimation methods. Historical data can be collected by tracking time, effort and design estimates used in past design activities. Thus we can obtain a complete profile about the teams. [9, 16] have a thorough explanation on how to obtain historical data and how precise they can be (confidence degree).

We treat the estimates in a probabilistic way, since the actual values can only be obtained after a task is implemented. We base our probabilistic treatment on *Normal (or Gaussian) Distribution*, given by the following probability density function [7], which is

$$f(x;\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left[-\frac{(x-\mu)^2}{2\sigma^2}\right],$$
$$-\infty \leq x \leq \infty, -\infty \leq \mu \leq \infty, \sigma \leq 0$$

where $\mu$ represents the estimates average and $\sigma$ represents the standard deviation defined by the confidence degrees.

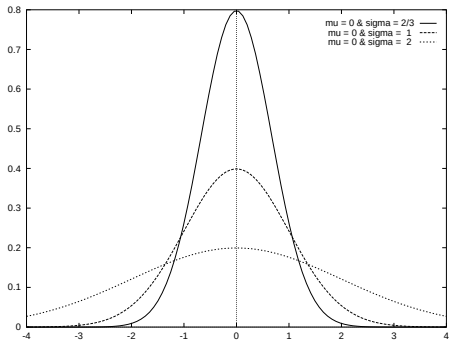


Figure 2: Normal distributions with different values of $\sigma$ (*sigma*) for a single $\mu$ (*mu*).

A normal form is chosen to model constraints in system-level design with teams because it is the best known statistical model for general-purpose cases in engineering [7], and it is a good representation for human estimates [17]. Although [18] showed that Rayleigh and Gamma distributions represent the manpower curve of a project development better than any other curve, they also showed limitations for these distributions as representative manpower model. The choice of the normal curve was also motivated by the approximation strategy presented in the literature and also used in this work, that is *approximation by expected value* [14].

Gaussian Distribution is associated with the estimation 3-tuple $(m, M, c)$, where $\mu = \frac{m+M}{2}$ and $\sigma = \frac{M-m}{2c}$. Table 1 presents confidence degrees for integer valus of c, i.e. $c = 3$ ("Very High"), $c = 2$ ("High") and $c = 1$ ("Low"). The confidence degree represents the possibility of an estimate to fall inside the interval $[m, M]$. In Figure 2, let us assume $m = -2$ e $M = 2$, we can compare the different degrees associated with these minimum and maximum values: $Very\ High$ for the inner curve ($\sigma = 2/3$), $High$ for the middle curve ($\sigma = 1$) and $Low$ for the outter curve ($\sigma = 2$).

| Confidence Degree | Standard Deviation ($\sigma$) | Statistical Semantic |
|---|---|---|
| **Very High (VH)** | $3\sigma = M - \mu$ | 99.7% of the values are in the estimated range. |
| **High (H)** | $2\sigma = M - \mu$ | 95.5% of the values are in the estimated range. |
| **Low(L)** | $\sigma = M - \mu$ | 68.3% of the values are in the estimated range. |

**Table 1. Definition of the $\sigma$ parameter.**

With this approach, we can represent the uncertainties and estimate the capability of development teams.

## 3 A Design Management Approach for System-Level Designs

The basic assumption lies in the fact that for large systems no single designer has the complete knowledge of the system. As a result, team assignment and partitioning must be

performed before implementation. In addition, task/team management affects task implementation, since it may be better to use underutilized teams (technology) to implement tasks to reach time-to-market at the expense of decreasing system performance.

Design at the early stages is assumed to be a refinement on the range of risks for each portion of the design. Initially, the estimates are in the ranges of low accuracy for the undefined portions of the design. These estimates translates into system-level constraints. As each design is refined, the constraint ranges are modified accordingly.

### 3.1 Design Views

Our approach is based on a task graph containing two different views: task development view and task implementation view. The model is briefly described here due to the lack of space.

Because of the co-dependency problem of task assignment and task implementation, the model for design management must consider a conceptual view of the system and a task development view of the system. Since task assignment is associated to task implementation through technology (Figure 1), implementation estimates are given by teams.

The system-level specification can be represented as a task hierarchy graph, which reflects the hierarchy of the project in the developing and structural dependency views. Figure 3 illustrates an Network Controller task graph, for more details refer to Section 5.

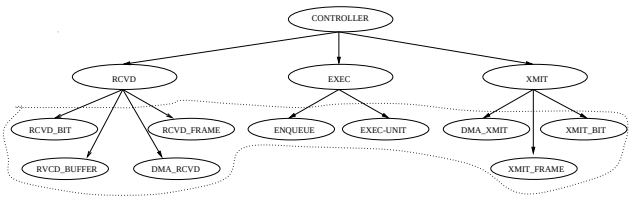


Figure 3: MicroController Task Graph

Each task (node in the task graph) stores probabilistic information which are used by the methodology to determine in which technology option it should be place. The *working set of a project*, marked in Figure 3, is the most well refined tasks of the hierarchical graph. They are the tasks that will be used by the model to start the partitioning algorithm.

### 3.2 System-Level Constraints

We consider the following constraints usually found in the literature [4].

**Production constraints:**

- $(C)$, Monetary cost. It is the total budget available to the project and its inference degree;
- $(D_i)$, Delay time. This is the total delay constraint imposed by the functional restrictions and its inference degree;
- Communication cost. Our approach does not work on actual implementation but rather the conceptual view of the system. As a result, it is difficult to make an accurate estimate on the communication cost. [6] has suggested that 30% overhead (of delay) is a good estimate at this level. We consider that team estimates already consider this overhead.

**Development constraints:**

- $(D_d)$, Development time. This is the time needed to put the product in the market and its inference degree;
- $(Teams, \Gamma)$, Team load. This is the set of development teams allocated for the project and their load, repectively;
- Integration cost. The integration cost of the partitions that were developed in parallel can also be estimates as an overhead, 20% of developed time [18].

## 4 A Stochastic Linear Programming Formulation for Management and Partitioning in System-Level Design

As presented before, we model the uncertainties of the team estimates as probabilistic curves. Thus, each curve represents the team experience and historical data about a specific parameter in previous system design. These curves are used in a stochastic linear approach to improve the partitioning problem solutions.

*Stochastic Integer Linear Programming* investigates linear programming problems in which the variables are randomly distributed: we are interested on the specific problem $\mathbf{min}\{\sum_{j=1}^{n} c_j x_j$ s.t. $Prob(\sum_{j=1}^{n} a_{ij} x_j \leq b_i) \geq 1 - \alpha_i$; $x_j$ *integer*, $a_{ij}$ *random variables*, $i = 1...m$, $0 < \alpha_i < 1\}$ [3].

Computationally, a more quantifiable approach is to solve the original Integer Linear Programming where all the probabilistic data have been replaced with their *expected values*. In this way, the problem can be written as a large deterministic problem (Expected Value Formulation [14]). The resulting *deterministic equivalent* problem can be solved using any general purpose optimization package. This is the approximation used to solve our model for system-level partitioning when the variables are randomly distributed.

### 4.1 System-Level Partitioning Formulation

In this section we present the notation and how constraints and team estimates are modeled in a mathematical ILP to solve the partitioning problem. We can observe that the formulation is a general one, i.e. the cost function and constraints can be customized to the specific partitioning problem. Here, we are interested to show that the stochastic approach can be solved by a general purpose optimization package, and so, computationally feasible [15].

#### 4.1.1 Notation

The general symbols are presented in Table 2 and all decision variables are presented in Table 3.

| Symbol | Descripton |
|---|---|
| $Tasks$ | set of the tasks |
| $C_d - Tasks$ | set of tasks developed concurrently |
| $C_i - Tasks$ | set of tasks executed concurrently |
| $Teams$ | set of development teams available |
| $Weeks$ | set of weeks (or any other time metric) |
| $Paths_d$ | development dependencies |
| $Paths_i$ | execution dependencies |
| $C$ | monetary cost desired for the project |
| $D_i$ | delay time desired for the project |
| $D_d$ | time-to-market desired for the project |

**Table 2. Symbols Used in the ILP Formulation.**

| Variable | Descripton |
|---|---|
| $c_{ij}$ | monetary cost for the task $i$ by team $j$ |
| $d_{ij}$ | estimated execution of task $i$ when implemented by team $j$ |
| $t_{ij}$ | time taken by team $j$ to implement task $i$ |
| $x_{ij}$ | binary variable which assumes value 1 if task $i$ is implemented by team $j$ |
| $\gamma_{ijk}$ | binary variable meaning that at week $k$, team $j$ implementing task $i$ |

**Table 3. Decision Variables Used in the ILP Formulation.**

#### 4.1.2 Formulation

We denote the summations over all tasks as $\sum_i$ (instead of $\sum_{i=0}^{n}$) to preserve the simplicity. Following the notation presented, we write the constraints inequalities and the objective cost function:

1. The *probabilistic parameters of each node* $x_{ij}$ are a 3-tuple $(c_{ij}, d_{ij}, t_{ij})$ with non-negative numbers for each possible technology option (HW or SW). Each parameter is represented as the 3-tuple $(m, M, c)$.

2. Every task $i$ must be implemented by only one $j$ team

$$\forall_{i \in Tasks} \sum_{j \in Teams} x_{ij} = 1$$

3. Path execution time (minimum and maximum)

$$Prob\{\sum_{i \in Paths_i} d_{ij} x_{ij} \leq D_i\} \geq 1 - \alpha_P$$
$$Prob\{\sum_{i \in Paths_i} d_{ij} x_{ij} \geq D_i\} \geq 1 - \alpha_P$$

4. Tasks developed sequentially are implemented in that order

$$Min(\sum_{j,k} k\gamma_{i_2jk}) - Max(\sum_{j,k} k\gamma_{i_1jk}) \geq \sum_j t_{i_1j} x_{i_1j}$$

for each edge $(i_1, i_2)$ ($e_{i_1} \rightarrow e_{i_2}$ is a development dependency in task graph), $i \in Tasks$, $j \in Teams$ and $k \in Weeks$.

5. Maximum load for team $j$

$$\forall_{k \in Weeks} \forall_{j \in Teams} \sum_{i \in tasks} \gamma_{ijk} \leq 1$$

6. Task assignment

$$\forall_{i \in Tasks, j \in Teams} \sum_{k \in Weeks} \gamma_{ijk} = t_{ij} x_{ij}$$

7. Cost of project

$$\sum_{i \in Tasks} \sum_{j \in Teams} c_{ij} x_{ij} \leq C$$

8. Risk of development in time ($i \in Tasks$)

$$Prob\{\sum_{j \in Teams} \sum_{k \in Weeks} t_{ij} \gamma_{ijk} \leq D_d\} \geq 1 - \alpha_D$$

The objective function can be defined as

$$f(x) = \sum_e z_e constraint_e$$

where the $z_e$'s are customized constants representing the priorities for the cost optimization and $constraint_e$'s are the constraint inequalities listed early.

In this way, e.g. if the $z_3$ is zero, then the "Path Execution Time" must not be considered in the cost of the system design.

9. The hardware/software partitioning is the problem of finding a mapping $map : Tasks \rightarrow (HW, SW) \times Teams$ in such a way that all performance and constraints are fulfilled and the design costs (objective function) are minimized.

## 5 Solution and Results

We modeled a Network Controller [2] using our approach and we solve the resulting problem using AMPL package with a C_PLEX ILP solver. This is a small example to illustrate the model use, but complex systems can be modeled in a similar way.

We solve the probabilistic equations by its expected values $E()$ [7, 14], i.e. the probability of a sum is approximated by the sum of probabilities. Then, the equation

$$Prob(\sum_{j=1}^{n} a_{ij} x_j \leq b_i) \geq 1 - \alpha_i$$

can be written in its approximatted form as

$$\sum_{j=1}^{n} Prob(a_{ij} \geq 1 - \alpha_i) x_j \leq b_i$$

where, the uncertainty $(1-\alpha_i)$ is pushed into the stochastic coeficients $(a_{ij})$. This approximation is used in order to be able to solve the set of inequalities by a commercial solver.

The Network Controller specification generates eleven tasks (nodes in task graph): nine system tasks (RCVD_BIT, RCVD_BUFFER, RCVD_FRAME, DMA_RCVD, ENQUEUE, EXEC_UNIT, DMA_XMIT, XMIT_FRAME, XMIT_BIT) and one milestone (CONTROLLER_DONE), as illustrated in Figure 4.

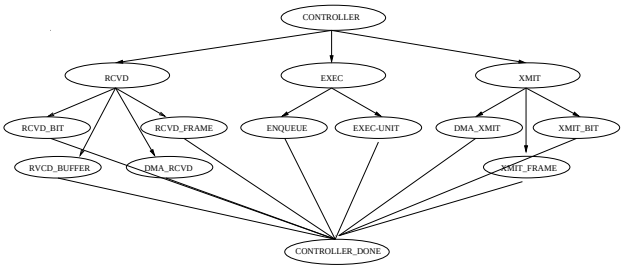


Figure 4: MicroController Task Graph

In our example, we simulated some situations as team losses, bad estimates and strict constraints varying the confidence degrees. Solving this example for different confidence degrees ($\mu$, $\mu \pm \sigma$, $\mu \pm 2\sigma$, $\mu \pm 3\sigma$) and for the same probability $(1 - \alpha)$, we obtain the following results. The development times for different degrees (in weeks with 2 development teams) are listed in Figure 5.

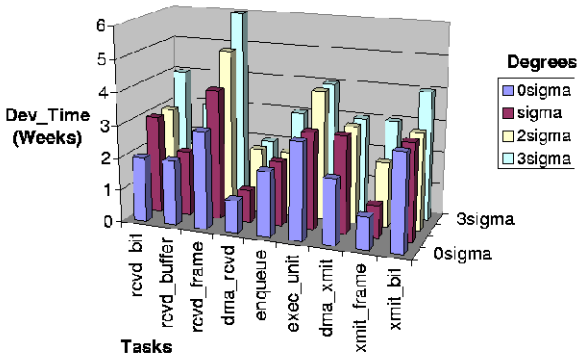


Figure 5: Development Times using Different Confidence Degrees.

The Network Controller development time for the cases presented in Figure 5 (risk probabilities of 0.68, 0.95 and 0.99 are 11, 7 and 5 weeks, respectively.

The development times presented several implementation scenarios that can be used to estimate the team's work in order to reach the market with low risk. Since most of the tasks are implemented in hardware, as the confidence degree lowers, more tasks need to be implemented in software in order to reduce implementation risks.

## 6 Conclusions

This paper presented a stochastic approach for managing and partitioning system-level design. Partitioning was treated as an interactive process that considers system constraints, human depended factors, development teams, and uncertainty to find out the objectives of the project.

The increased time for calculating the ILP solution, in contrast to other approaches, is compensated by an improved quality of the estimations. Other methods for solving probabilistic linear programming problems has appeared in the literature [3], which can improve this work.

The results offered a set of early design scenarios to guide the system manager in order to improve time-to-market and team assignment and codesign tasks.

We used the expected value approximation to solve the set of probabilistic inequalities equations. We are currently investigating other solution methods.

## Acknowledgements

This work has been support by a scholarship from CAPES and by grants from the agencies FAPEMIG, CNPq and PRONEX Finep/CNPq/MCT 76/97/1016/00.

## References

[1] J. Adams and D. Thomas. The design of mixed hardware/software systems. In *ACM* $33^{th}$ *DAC*, Las Vegas, 1996.

[2] J. Albuquerque, C. C. Jr., C. Cavalcanti, and A. Fernandes. A system-level design model for hardware/software codesign. Technical Report DCC 012/98, UFMG, nov 1998.

[3] M. Biswal, N. Biswal, and D. Li. Probabilistic linear programming problems with exponential random variables: A technical note. *European Journal of Operational Research*, (111):589–597, 1998.

[4] R. Camposano and J. Wilberg. Embedded system design. *Design Automation for Embedded Systems An International Journal*, 1(1-2):5–50, January 1996.

[5] V. Catania, M. Malgeri, and M. Russo. Applying fuzzy logic to codesign partitioning. *IEEE Micro*, pages 62–70, May/June 1997.

[6] J. A. Debardelaben, V. K. Madisetti, and A. J. Gadient. Incorporating cost modeling in embedded-system design. In *IEEE Design & Test of Computers*. IEEE, july-september 1997.

[7] G. Hahn and S. Shapino. *Statistical Models in Engineering*. John Wiley & Sons, 1994.

[8] D. Herrmann, J. Henkel, and R. Ernst. An approach to the adaptation of estimated cost parameters in the cosyma system. In *CODES'94*, 1994.

[9] W. Humphrey. *A Discipline For Software Engineering*. Addison Wesley, 1995.

[10] A. Kalavade. *System-level Codesign of Mixed Hardware-Software Systems*. PhD thesis, University of California, CA, September 1995.

[11] I. Karkowski and R. Otten. Uncertainties in hardware-software co-synthesis of embedded systems. In *Workshop on High Level Synthesis Algoritms, Tools and Design (HILES)*, Stanford University, January 1996.

[12] J. Karlsson and K. Ryan. A cost-value approach for prioritizing requirements. *IEEE Software*, 1997.

[13] T. Khoshgoftaar, E. B. Allen, and etc. Using process history to predict sofware quality. *IEEE Computer*, 1998.

[14] G. Nemhauser, A. H. G. R. Kan, and M. J. Todd, editors. *Handbooks in Operations Research and Management Science: Optimization*, volume 1. North-Holland, 1989. Chapter 8.

[15] R. Niemann and P. Marwedel. Hardware/software partitioning using integer programming, 1996. IEEE ED&TC'96.

[16] M. Paulk, B. Curtis, and etc. Capability maturity model for software, version 1.1. Technical Report CMU/SEI-93-TR24, Software Engineering Institute, 1993.

[17] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers, 1988.

[18] K. Pillai and V. S. Nair. A model for software development effort and cost estimation. *IEEE Transactions on Software Engineering*, 23(8):485–497, August 1997.