Synthesis of Distributed Execution Platforms for Cyber-Physical Systems with Applications to High-Performance Buildings

Francesco Leonardi
Dept. of Computer Science
Columbia University
New York, USA
leonardi@cs.columbia.edu

Alessandro Pinto
Systems Department
United Technologies Research Center, Inc
Berkeley, CA, USA
alessandro.pinto@utrc.utc.com

Luca P. Carloni
Dept. of Computer Science
Columbia University
New York, USA
luca@cs.columbia.edu

***Abstract*—We propose a methodology, and its embodiment into a design flow, to realize execution platforms for high-performance building applications. This is an example of a class of cyber-physical systems where a network of sensors, controllers, and actuators must be designed under physical spatial constraints to implement various types of signal processing and control tasks. In our approach, the applications are specified using the dataflow model of computation while the building dictates the physical constraints, including the position of sensors and actuators. We present a rigorous formulation of the design-space exploration problem and we propose to solve it by progressing through a sequence of refinement steps from specification to detailed implementation. Two key steps are the synthesis of the computation platform and the synthesis of the communication network. Combined, they allow us to automatically derive an optimal implementation through the selection and composition of processing and networking elements from given technology libraries. We demonstrate the applicability of our approach by comparing it to the manual design of a given case study: the real-time estimation of building occupancy using a network of video cameras.**

***Index Terms*—Cyber–Physical Systems; CAD; Sythesis;**

I. Introduction

Aircraft, cars, buildings, factories are just a few examples of Cyber-Physical Systems (CPS) [1], [2], [3], [4], [5], [6]. The characterizing feature of these systems is the interaction between physical processes governed by the laws of physics and an execution platform which comprises embedded software, hardware devices (sensors, actuators, processors) and communication networks to interconnect them. Verification and Validation (V&V) of CPSs is a challenging task that is becoming the major roadblock to their cost-effective deployment. It has been argued that to realize the full potential of CPSs it is not enough to improve current design methods for traditional general-purpose computer systems, but instead it is necessary to rebuild the computing and networking abstractions [5]. The main reason is that current programming models and system abstractions cannot express aspects of the behavior, such as time, that are essential to CPSs. For this reason, CPS designers are forced to extensive testing and analysis in order to verify that their "cyber" sub-system meets the application requirements. When CPSs are networked, V&V becomes even more difficult because of the state-explosion problem [7].

To reduce the V&V effort, we are pursuing a correct-by-construction design methodology for CPSs. Our approach will ultimately encompass all phases of the design process from environment modeling and validation, through the specification of the target application (which will be increasingly heterogeneous as it will combine signal processing and control algorithms as well as functional and non-functional requirements), down to the implementation of this application on a distributed execution platform. In this paper we focus on the last problem and we present a design flow that allows us to automatically optimize and synthesize this implementation given the algorithm specification and the physical constraints imposed by the environment.

We start from the assumption that the control design has been already performed, thereby determining the optimal placement of sensors and actuators to guarantee observability and controllability of the physical system. Moreover, the dynamics involved in the physical processes to be controlled determine a minimum execution period for each control task. Hence, besides the functional specification, the input to the execution-platform design process includes a set of timing constraints, the position of the sensors and actuators, as well as other physical constraints imposed by the surrounding environment. We also assume that the algorithms are given in terms of a network of processes (called actors) that read from their input ports, execute some local computation, and write on their output ports. Starting from this high-level specification, we propose synthesis methods capable of exploring different solutions for the computation and communication platforms of a CPS, corresponding to different trade-offs between distributed and centralized architectures, or between cost and performance. The result generated by the automatic design flow is correct, meaning that all the actors are guaranteed to complete their execution within the given timing constraints, and all data are delivered through the network on time for the actors to start the next execution.

The core contribution of this work is a design flow that tackles the complexity of the execution-platform design problem by introducing multiple abstraction layers (Sec. II). The optimization problem is divided into three mains steps: the *clustering and placement* of the actors in the physical space (Sec. III), the *communication platform synthesis* that decides the type and topology of the network (Sec. IV), and the *computation platform synthesis* that decides the number, type

978-0-7695-4361-1/11 $26.00 © 2011 IEEE
DOI 10.1109/ICCPS.2011.23

IEEE computer society

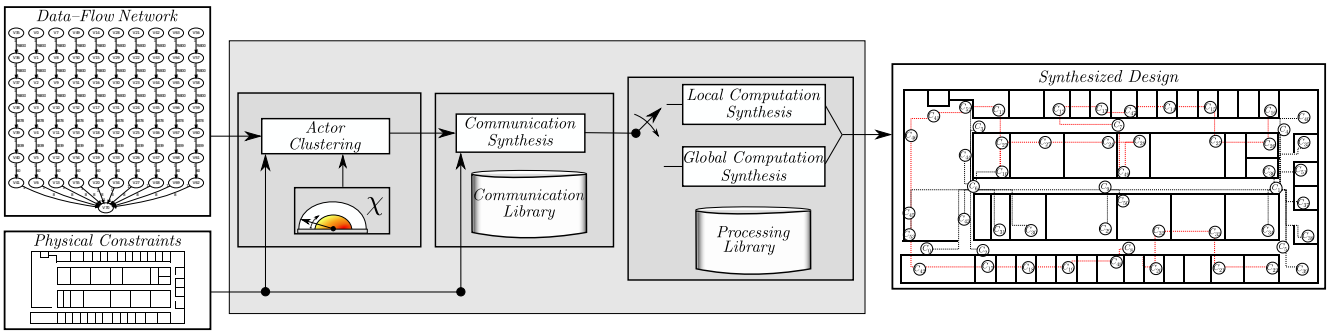


Fig. 1. The proposed synthesis-based design flow to design networked cyber-physical systems for high-performance building applications.

and organization of the processing elements (Sec. V). The reduction in complexity allows us to perform an efficient design-space exploration in reasonable time, as shown by the experimental results for the case study of real-time estimation of building occupancy (Sec. VI).

Related Work. Prakash and Parker formulated the problem of the synthesis of heterogeneous multiprocessor systems as a Mixed Integer Linear Program [8]. Their focus is on the mapping and scheduling of computational tasks across the available processors. However the high-level model of the interconnection technology that they employ prevents an effective communication synthesis. Wolf *et al.* have proposed a synthesis flow to map and schedule the computation and communication processes of the input specification onto a set of distributed processors in order to minimize a given cost function and satisfy given deadline constraints [9], [10], [11]. They note that communication contributes to a significant part of the global cost of a distributed embedded system. This observation leads to the definition of heuristics that generate partitions of the specification task graph which attempt to minimize the communication requirements. Kuchcinski suggested to formulate the synthesis of distributed embedded systems using Constraint Logic Programming (CLP), since CLP solvers may be more efficient than MILP ones [12].

While previous research efforts have focused on the synthesis of the processing architecture and the scheduling of computational and communication tasks, our approach is centered on the impact that the physical constraints imposed by the environment have on the design of the execution platform. Indeed, the environment where a CPS operates plays a critical role in determining the optimal deployment of the processing elements, the optimal assignment of tasks to them, and the optimization of the communication infrastructure.

II. The Synthesis-Based Design Methodology

We capture the specification of the target embedded-software applications using the dataflow model of computation. A dataflow network is a tuple $G(V, E, prod, cons)$, where V is the set of actors, E is the set of edges (i.e. FIFO queues), $prod : E \rightarrow \mathbb{N}$ is a function that associates to each edge the token production rate of the source actor, and $cons : E \rightarrow \mathbb{N}$ is a function that associates to each edge the token consumption rate of the destination actor [13]. The actors in the dataflow graph are partitioned into sensors, actuators, and computing nodes. Sensors are actors with no input edges while actuators are actors with no output edges. Each application is characterized by an execution period T: at the beginning of the period the sensors sample the physical environment and produce new data that are processed by the computing nodes, which execute embedded software tasks to derive new outputs for the actuators. A repetition vector $\mathbf{f}$ defines the number of times each actor $v \in V$ is executed in a given period, denoted $\mathbf{f}(v)$, according to a given schedule [13]. We allow to specify complex systems that comprise multiple applications, each with its own execution period and we define the global period T_G of the whole dataflow network as the least common multiplier among them.

Our design goal is to automatically derive an optimal implementation of the specification G for a given building. The synthesis process consists in selecting and combining instances of building-block components from two given technology libraries to realize a distributed execution platform that implements the computation and communication tasks which are specified by G. The *processing library* contains a set J of processing elements (PEs) that differ by processor type and memory size and are used to execute the embedded software tasks. Typically, multiple PEs are instantiated and deployed in various clusters across the building. The *communication library* contains networking components such as switches, routers, and wires. These components are used to realize the interconnection network that supports the communications among the tasks which are mapped on the various PEs.

The characteristics of the given building in terms of function, structure and dimension affect the design process by introducing a set of *physical constraints*. First, it is reasonable to assume that the positions of all sensors and actuators in the building are specified by means of their coordinates in the Euclidean space. These positions depend on the building geometry and the characteristics of the given applications. Conversely, they constrain the space of possible design choices to derive the execution platform. Other examples of physical constraints imposed by the building include various limitations in the deployment of the interconnection network: e.g. some wiring paths may be too expensive, long, or simply infeasible.

Fig. 1 illustrates the proposed synthesis-based methodology to derive an optimal design of the execution platform. The

inputs to the design process are the functional specification of the applications (the dataflow network), the physical constraints imposed by the building, and the technology libraries. The output is the architecture of the execution platform, which is obtained by combining instances of the library components, together with its physical deployment in terms of the detailed positions of the processing and networking components as well as the detailed mapping of the computation and communication tasks onto the various components.

The design process proceeds through three main steps: (1) *Actor Clustering*, (2) *Communication Synthesis*, and (3) *Computation Synthesis*. In the first step, the actors of G are partitioned into a set C of clusters and each cluster is automatically assigned a position in the building to minimize the aggregate inter-cluster communication requirements. Also, each cluster becomes implicitly characterized with an aggregate set of computation requirements in terms of processor cycles and memory occupation. In the second step, we synthesize an optimal network that connects the actor clusters and satisfies the communication requirements for all the edges of G. In the final step, we complete the synthesis of the distributed execution platform by deriving for each cluster $c \in C$ an optimal combination of PEs that can sustain the execution of all the actors of c. While the optimality criteria may vary for different classes of CPS applications, in this work we focus on optimizing the implementation cost of the execution platform. The computation synthesis uses performance and cost metrics of the PEs in the library. Computation and memory requirements are estimated by first profiling one execution of an actor on each PE, and then using the repetition vector to account for the requirements over one period of the schedule.

III. Actor Clustering and Placement

We start by partitioning V into a set of clusters of actors to minimize the cost of the cyber part of the system. This step consists of two sub-steps:

A. Actor Placement

The goal is to optimize the positions of the actors in the Euclidean space such that the cost of the interconnection network that supports the end-to-end communications requirements expressed by the set of edges E is minimized. This cost depends on the network capacity and the distance between the nodes to be interconnected. The total distance to be spanned by the network depends on the distance between PEs, which in turn depends on the position of the clusters of actors. Hence, we aim at placing actors that exchange large amount of data close to each other. Let $F \subseteq V$ be the set of actors with fixed position (i.e. sensors and actuators). Actor placement can be formulated as the following optimization problem:

$$\texttt{PR1}: \quad \min_{\mathbf{p}_u, \mathbf{p}_v, u,v \in V} \quad \sum_{u,v \in V} \alpha_{uv} \cdot ||\mathbf{p}_u - \mathbf{p}_v||_2^2$$
$$s.t \qquad \mathbf{p}_f = (x_f, y_f, z_f), \ \forall f \in F$$

where $\mathbf{p}_u$ and $\mathbf{p}_v$ are the positions of actor u and v, respectively and $||\mathbf{p}_u - \mathbf{p}_v||_2^2$ is the square of the Euclidean distance between them. The weighting coefficient α_{uv} abstracts the cost of implementing the communication constraint $e(u,v) \in E$:

$$\alpha_{uv} = \left[\frac{b_e \cdot prod(e) \cdot \mathbf{f}(u)}{T_u} \right]^{\chi} \tag{1}$$

where b_e is the *payload* associated with edge e (i.e. the size in bits of the tokens exchanged between u and v in the invocation period T_u), and $\chi \in \mathbb{R}$ is a design parameter that weights the estimated communication cost. A sweep over different χ values can be used to explore the design space (as shown in Section VI). Problem PR1 can be decomposed in three sub-problems ($\texttt{PR1}_x$, $\texttt{PR1}_y$, and $\texttt{PR1}_z$) across the x, y and z axes. We discuss the formulation of $\texttt{PR1}_x$ as the others are similar:

$$\texttt{PR1}_x : \min_{\mathbf{x}_u, \mathbf{x}_v, u,v \in V} \sum_{u,v \in V} \alpha_{uv} \cdot \left(x_u^2 - 2 \cdot x_u \cdot x_v + x_v^2 \right) \tag{2}$$

In general, the summation contains only non-linear terms. However, some terms become linear or constant when either x_u or x_v is given. Also, the constant terms, resulting from a known actor position, are neglected in the minimization problem, since they do not affect optimality.

Without loss of generality we identify each actor with an integer index in the range $[0, |V|-1]$ where the first $|F|$ indices are reserved to actors in F (i.e. with a given position in the Euclidean space). Problem $\texttt{PR1}_x$ can be expressed in a matrix notation as:

$$\min \left(\mathbf{x} P \mathbf{x}^T - 2\mathbf{x} Q \right) \tag{3}$$

where P is a $(|V|-|F|) \times (|V|-|F|)$ matrix whose elements are the coefficients of the non-linear terms of $\texttt{PR1}_x$:

$$p_{ii} = \sum_{m=0}^{m=|V|-1} \alpha_{i+|F|,m} + \sum_{n=0, n \neq i+|F|}^{n=|V|-1} \alpha_{n,i+|F|}$$

$$p_{ij} = \alpha_{(i+|F|),(j+|F|)} + \alpha_{(j+|F|),(i+|F|)}$$

The coefficients of the linear terms in $\texttt{PR1}_x$ are the elements of the Q vector.

$$q_i = \sum_{m=0}^{m=F-1} \alpha_{i+|F|,m} \cdot x_m + \sum_{n=0}^{n=F-1} \alpha_{n,i+|F|} \cdot x_n$$

Notice that x_n and x_m in this equation are known terms, representing the x coordinate of a sensor or actuator. The optimal solution to the problem of Eq. 3 is the solution of the following linear system:

$$P\mathbf{x} - Q = 0 \quad \Rightarrow \quad \mathbf{x} = QP^{-1}$$

B. Actor Clustering

We formulate this step as an optimization problem where the set V needs to be partitioned into a set of clusters $C = \{c_1, ..., c_n\}$ so that the size of the edge cut is minimized. The size of the cut is the sum of the weights of edges in the set $\{e(u,v) \in E | u \in c_i, v \in c_j, c_i \neq c_j\}$. Weight h_e is an approximation of the cost to implement edge $e(u,v)$:

$$h_e = \begin{cases} \frac{b_e}{|\mathbf{p}_u - \mathbf{p}_v|_1} & if \quad |\mathbf{p}_u - \mathbf{p}_v|_1 < 1 \\ b_e \cdot |\mathbf{p}_u - \mathbf{p}_v|_1 & if \quad |\mathbf{p}_u - \mathbf{p}_v|_1 \geq 1 \end{cases} \tag{4}$$

where $|\cdot|_1$ denotes the Manhattan distance (norm 1). We prefer the Manhattan distance over the Euclidean distance because it is more accurate to model the wiring paths in buildings. Equation 4 emphasizes the edges of actors that exchange large amount of data or that are close in the Euclidean space.

We constrain the optimization problem so that each cluster contains either one sensor or one actuator. Therefore, the number of clusters $|C|$ is given by the number of sensors and actuators. This choice is motivated by preliminary experiments: each sensor/actuator has a unique position in the Euclidean space, which, in turn, determines the position of its cluster, while having a larger number of clusters tends to be sub-optimal since the communication cost increases because more devices have to be connected. We use hMetis [14] to solve the clustering problem.

IV. Communication Synthesis

The goal of this step is to generate an optimal interconnection network among actor clusters that satisfies the communication requirements. The synthesis algorithm takes three inputs: 1) the specification of the communication requirements, 2) the physical constraints, and 3) the communication library. The communication requirements are captured by a *communication graph* $G'(C, E')$ where an edge $e(c, d) \in E'$ carries the aggregate bandwidth of all the edges in E that connect actors in cluster c to actors in cluster d. The physical constraints are the position of each cluster $c \in C$ in the Euclidean space and a set of limitations arising from the environment. For instance, the geometry of the building determines the set W of wiring paths available between any two clusters. Each element $w(c, d) \in W$ represents the *minimum-length physical link* that can be instantiated between cluster c and d. Function $\Gamma : W \rightarrow \mathbb{R}_{\geq 0}$ gives the length of the link. This depends on the type of application and implementation technology. For a wired network, it is the length of the wiring path between c and d, accounting for wiring constraints (e.g. presence of walls and areas that cannot be wired.) The communication library contains nodes and links that can be implemented based on a given set I of networking technologies: e.g. Table I reports three technologies that are common for building-automation systems and that we use for the case study of Sec. VI. The communication synthesis problem consists in finding a cost-effective, potentially heterogeneous, combination of components instantiated from the communication library such that the functional requirements imposed by G' and the constraints imposed by the environment (i.e. the building) are satisfied. We formulate it as an Integer Linear Program (ILP).

A. The ILP Formulation

We first partition the communication requirements in E' into a set K of groups. The number of groups $|K|$ is a design parameter and it represents the multiplicity of sub-networks that can be implemented with the same technology. For each group, we map its edges on components of the communication library in order to realize up to $|I|$ sub-networks. Hence, the number of possible sub-networks in the final solution is at most $|K| \cdot |I|$. A cluster belongs to the k-th group if it has an incoming or outgoing edge that belongs to that group. Since a group can be implemented using any of the technologies in set I, it is possible to synthesize a heterogeneous network. We use two indexes k and i for the groups and the technologies, and a combined index ki to denote the k-th group implemented with the i-th technology. Two main decision variables are used in the problem formulation. Binary variable ϵ_w^{ki} is equal to one when the wiring link w is part of the k-th group and uses the i-th interconnection technology, and zero otherwise. Binary variable ζ_e^{ki} is equal to one when edge $e \in E'$ is implemented by the ki-th sub-network. We allow the solver to optimally map each edge onto a sub-network. Hence only a sub-set of the possible sub-networks can be instantiated. The cost function to be minimized is:

$$\Phi_C = \sum_{k \in K} \sum_{i \in I} \phi_{ki}$$

where $\phi_{ki} = \phi_{ki}^l + \phi_{ki}^{dev}$ represents the cost of the ki sub-network, which is given by the sum of the cost of the links ϕ_{ki}^l and the cost of the equipment ϕ_{ki}^{dev}. The cost of the links is the sum of the costs of all physical links instantiated to connect the clusters in the sub-network:

$$\phi_{ki}^l = g_{0i} \cdot \sum_{w \in W} \epsilon_w^{ki} \cdot \Gamma(w)$$

where g_{0i} represents the cost per unit length of the link type used by the i-th interconnection technology. The equipment cost ϕ_{ki}^{dev} can be expressed as follows:

$$\phi_{ki}^{dev} = g_{1i} \cdot \sum_{w \in W} \epsilon_w^{ki} + g_i \cdot \nu_i$$

where g_{1i} is the cost of the interface used by a cluster to communicate on the i-th interconnection technology; g_i is the cost of an intermediate node (i.e. the cost of a router or a switch) for the i-th technology; ν_i is the number of the intermediate nodes installed by the synthesis algorithm for that technology. This number is technology dependent and is determined by a set of constraints in the optimization problem. There are two kinds of constraints: functional constraints, which ensure that the edges in E' are correctly implemented, and structural constraints, which are imposed by the network components (such as limitations on its topology, on the number of devices on a bus, on the length of a link, etc.)

Functional Constraints. The following three constraints guarantee that: each edge $e \in E'$ is mapped to exactly one sub-network; each sub-network sustains the communication traffic; and a cluster is physically connected to the sub-networks where its edges are mapped.

$$\sum_{k \in K} \sum_{i \in I} \zeta_e^{ki} = 1 \quad \forall e \in E' \tag{5}$$

$$\sum_{e \in E'} b_e \cdot \zeta_e^{ki} \leq B_i \cdot T_G \quad \forall k \in K, \forall i \in I \tag{6}$$

$$\sum_{w(c,d) \in W} \epsilon_w^{ki} \geq \delta_{kip} \quad \forall k \in K, \forall i \in I, \forall c \in C \tag{7}$$

i	interconnection technology	bandwidth (B_i)
0	Low-Speed ARCnet	78 Kbps
1	High-Speed ARCnet	2.5 Mbps
2	100 Mbps Ethernet	100 Mbps

TABLE I
THE COMMUNICATION LIBRARY.

Variable b_e represents the payload sent with period T_G over edge $e \in E'$ and is the sum of all the payloads of the subset of edges in E corresponding to e. Variable B_i is the maximum bandwidth supported by the i-th technology. The binary variable δ_{kip} is subjected to the constraint:

$$\delta_{kip} > \frac{1}{|C|} \sum_{\substack{e(c,d)\in E' \\ :d \in C}} \zeta_e^{ki} \quad \forall k \in K, \forall i \in I \tag{8}$$

If no edge $e \in E'$ is mapped on the ki-th sub-network, then δ_{kip} can assume both values. If at least one edge is mapped on the ki-th sub-network, then δ_{kip} must be equal to one, requiring the presence of a physical link to support the logical connections in the ki-th sub-network. It is worth mentioning that Constraint 7 allows a cluster to be connected to a sub-network even if it does not communicate with any actor in this sub-network. In that case the cluster is used as a link repeater.

Structural Constraints. These are necessary to guarantee that the synthesized network complies with the requirements of the technologies available in the communication library. For instance, the topology of a network (i.e. the structure of the graph defined by C and W) may be constrained to be a tree. By including some additional constraints, a tree can also capture topologies such as chains and stars. Other example examples of structural constraints can be found in the literature [15], [16]. In the sequel we present the structural constraints that we defined to model the ARCnet and Ethernet networking technologies, which are used in the case study of Sec. VI. The topologies of ARCnet and Ethernet networks are restricted to be set of chains and star topologies, respectively. Further, the number of routers for an ARCnet sub-network depends on the number of clusters connected by a chain and by the length of the chain itself.

$$\begin{cases} \vartheta^{k0} \geq \frac{1}{32} \sum_w \epsilon_w^{k0} \quad \forall k \\ \vartheta^{k0} \geq \frac{1}{1200} \cdot \sum_w \epsilon_w^{k0} \cdot \Gamma(w) \quad \forall k \\ \vartheta^{k1} \geq \frac{1}{8} \sum_w \epsilon_w^{k1} \quad \forall k \\ \vartheta^{k1} \geq \frac{1}{120} \sum_w \epsilon_w^{k1} \cdot \Gamma(w) \quad \forall k \\ \vartheta^{k2} = 0 \quad \forall k \end{cases} \tag{9}$$

where integer variable ϑ^{ki} indicates the number of ARCnet routers required by each sub-networ. The above constraints can be explained as follows: a Low-Speed ARCnet sub-network can interconnect at most 32 devices on a chain whose length cannot exceed 1200 m; a High-Speed ARCnet sub-network can interconnect at most 8 devices on a chain whose length cannot exceed 120 m. The number of instantiated routers is:

$$\nu_i = \sum_{k\in K, i\in=\{0,1\}} \vartheta^{ki} \tag{10}$$

A similar model can be developed for Ethernet networks. The total number of Ethernet switches is computed as:

$$\nu_2 = \sum_{c\in C, k\in K} \pi_c^{k2} \tag{11}$$

where the binary variable π_c^{k2} indicates whether a switch is mapped on cluster c. A switch is associated with cluster c if the cluster is part of an Ethernet network and has more than 1 link on the Ethernet network, as modeled by the following set of constraints:

$$\begin{cases} \pi_c^{ki} = 0 \quad i = 0,1 \quad \forall k \in K, \forall c \in C \\ \pi_c^{k2} \leq \sum_{w\in p} \epsilon_w^{k2} \quad \forall k \in K, \forall c \in C \\ |C| \cdot \pi_c^{k2} \geq \sum_{w\in p} \epsilon_w^{k2} \quad \forall k \in K, \forall c \in C \end{cases} \tag{12}$$

parameter	definition	value
g_{00}, g_{01}	$[\$/m]$: cost per meter of an ARCnet cable	\$3
g_{02}	$[\$/m]$: cost per meter of an Ethernet cable	\$4.5
g_{sw}	Ethernet switch cost	\$250
g_{ar}	ARCnet router cost	\$560
g_{10}, g_{11}	ARCnet interface cost	\$1
g_{12}	Ethernet interface cost	\$2

TABLE II
COST PARAMETERS OF THE COMMUNICATION MODEL.

Tables I and II summarize the bandwidths (B_i) and the cost parameters for each communication technology in our library.

V. COMPUTATION SYNTHESIS

The set of actors in each cluster is executed by a processing platform. The computation synthesis step optimizes the cost of the distributed processing platform while satisfying computational requirements (i.e. the maximum execution time of actor v must be shorter than its period T_v). The result generated by communication synthesis is taken as constraint, i.e. the inter-cluster network is fixed and the offered bandwidth cannot be changed. We developed two alternative approaches to computation synthesis. *Local Computation Synthesis* assumes that the allocation of actors to clusters is fixed and for each cluster finds the composition of PEs from the Processing Library that executes the actors at minimum cost. *Global Computation Synthesis* performs a joint optimization of the binding between actors and clusters and of the computation architecture for each cluster. In fact, the clustering algorithm presented in Section III uses abstract cost models and can generate a solution which is only close to the optimal one.

A. Local Computation Synthesis

The inputs to this problem are the dataflow $G(V, E, prod, cons)$, the set of actors V_c for each cluster $c \in C$, the Processing Library containing a set J of PE types, and a characterization of each actor v on each PE of type j in terms of execution time t_{vj}, data memory m_{vj}, and program memory r_{vj} requirements. The computation cost for implementing all clusters is:

$$\Phi_P = \sum_{c\in C} \Phi_{Pc} \tag{13}$$

where Φ_{Pc} is the computation cost of cluster c. Since we assume a static binding of actors to clusters, to minimize Φ_P is equivalent to minimize each term Φ_{Pc} independently. We

give the ILP formulation of the synthesis problem for a generic cluster c since the same formulation is used for each cluster. Two variables capture the problem: variable κ_{jc} denotes the number of instantiated PEs of type j for cluster c, while the binary variable ρ_{vjc} is equal to one if actor v is executed by a PE of type j in cluster c. The cost of a cluster is defined as:

$$\Phi_{Pc} = \sum_{j \in J} \phi_j \cdot \kappa_{jc} \tag{14}$$

and ϕ_j is the cost of a PE of type j. PEs of type j are instantiated only if there are actors mapped on them:

$$\sum_{v \in V_c} \rho_{vjc} \geq \kappa_{jc} \qquad \forall j \in J$$

The total execution time must be less than the deadline T_G:

$$\kappa_{jc} \cdot (T_G - T_{os}) \geq \sum_{v \in V} \rho_{vjc} \cdot \mathbf{f}(v) \frac{T_G}{T_v} (T_{cs} + t_{vj}) \quad \forall j \in J, \forall c \in C$$

where T_{os} summarizes the time consumed by the operating system to perform I/O operations; T_{cs} is a time penalty due to context switching. The data memory provided by the PEs of type j must be larger than the one required for actor execution and the implementation of the communication buffers:

$$\kappa_{jc} \cdot M_j \geq \sum_{v \in V} \rho_{vjc} (m_{v,j} + n_v^{out} + n_v^{in}) \quad \forall j \in J$$

where M_j is the data memory available on a PE of type j; n_v^{out} and n_v^{in} are the sizes of the output and input buffer required by actor v, respectively:

$$n_v^{out} = \mathbf{f}(v) \cdot \sum_{e(u,v) \in E} prod(e) \cdot b_e$$

$$n_v^{in} = \sum_{e(u,v) \in E} \mathbf{f}(u) \cdot prod(e) \cdot b_e$$

The program memory of a PE of type j must be able to store the embedded software of the actors that it executes:

$$\kappa_{jc} \cdot R_j \geq \sum_{v \in V} \rho_{vjc} \cdot r_{vj} \quad \forall j \in J$$

where R_j is the program memory available on a PE of type j. Since actors are atomic, a single PE of type j must be able to entirely sustain each actor mapped on it in terms of execution time, data memory, and program memory, respectively:

$$\rho_{vjc} \cdot \mathbf{f}(v) \cdot (T_{cs} + t_{vj}) \leq (T_v - T_{os}) \cdot \rho_{vjc} \quad \forall j \in J, v \in V$$

$$M_j \cdot \rho_{vjc} \geq \rho_{vjc} \cdot (m_{v,j} + n_v^{out} + n_v^{in}) \quad \forall j \in J, v \in V$$

$$\rho_{vjc} \cdot r_{vjc} \leq R_j \cdot \rho_{vjc} \quad \forall j \in J, v \in V$$

Finally, we constrain an actor to be executed only by a PE mapped on the same cluster

$$\begin{cases} \sum_{j \in J} \rho_{vjc} = 1 & \forall v \in V_c \\ \sum_{j \in J} \rho_{vjc} = 0 & \forall v \notin V_c \end{cases} \tag{15}$$

B. Global Computation Synthesis

In addition to the inputs already defined for local computation synthesis, global computation synthesis uses the interconnection infrastructure as a constraint to limit the way in which actors can be repositioned to different clusters. The solution to the optimization problem is a new assignment of actors to clusters and the optimal computation architecture for each cluster. The ILP formulation for the global computation synthesis is based on the one for the local computation synthesis. The objective function is the same (Eq. 13), and the same constraints are included with the exception of Constraint 15. Additional constraints need to be added to support the actor-relocation option. Five more binary variables are introduced: variable ω_{uv} is equal to one when actor u and actor v are mapped on the same cluster; variable ψ_{uvc} indicates whether actors u and v both belong to cluster c; variable η_{uvki} is equal to one when actors u and v communicates on the ki sub-network; variable μ_{vc} tells whether actor v is mapped on cluster c. Each actor is mapped to only one cluster:

$$\sum_{c \in C} \mu_{vc} = 1 \quad \forall v \in V \qquad where \qquad \mu_{vc} = \sum_{j \in J} \rho_{vjc}$$

Each edge is mapped either inside a cluster or to one sub-network:

$$\omega_{uv} + \sum_{k \in, i \in I} \eta_{uvki} = 1 \quad \forall e(u,v) \in E$$

$$\omega_{uv} = \sum_{c \in C} \psi_{uvc}$$

where ψ_{uvc}=1 when actors u and v are mapped on cluster c:

$$\begin{cases} \psi_{uvc} \geq \mu_{uc} + \mu_{vc} - 1 & \forall u, v \in V, c \in C \\ \psi_{uvc} \leq \frac{\mu_{uc} + \mu_{vc}}{2} \end{cases}$$

Additional constraints are needed to guarantee that the aggregate bandwidth can be sustained by the instantiated network. Note that only the edges that are mapped on the ki subnetwork (i.e. when $\eta_{uvki} = 1$) contribute to its bandwidth count:

$$\sum_{e(u,v) \in E} prod(e) \cdot b_e \frac{T_G}{T_u} \cdot (\eta_{uvki}) \leq B_i \cdot T_G \quad \forall k, i$$

Variable η_{uvki} can be greater than zero when both actor u and v are mapped on a cluster connected by the ki subnetwork:

$$\eta_{uvki} \leq \sum_{c \in C_{ki}} \frac{\mu_{uc} + \mu_{vc}}{2} \qquad \forall e(u,v), k, i$$

where C_{ki} is the set of cluster connected by the ki subnetwork. Each actor in the network is binded to only one PE type on only one cluster:

$$\sum_{c \in C} \sum_{j \in J} \rho_{vjc} = 1 \quad \forall v \in V$$

We force the mapping of those actors whose positions are known (i.e. sensors and actuators), by simply setting to one the variable μ_{vc} corresponding to the desired cluster for actor v. We also require that the edges of actors that belong to clusters connected to certain subnetworks cannot be mapped on clusters connected by other subnetworks.

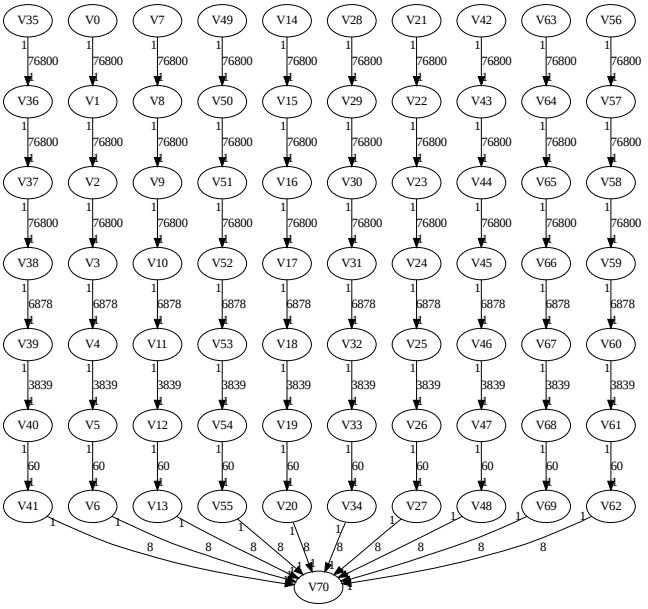


Fig. 2. Static dataflow capturing the specifications of a smart-camera network to estimate the building occupancy. Each actor is executed once in each period, consuming one token at the input and producing one token at the output. However, tokens have generally different sizes (in terms of bytes) thus yielding different bandwidth values for the edges in the graph.

VI. Experimental Results

To demonstrate the applicability of our approach we chose an application that offers an interesting design space while being simple enough to compare a manual design with the synthesized ones. For this application we 1) appraise the quality of the results of our synthesis tool against those obtained manually for two reference buildings that impose different physical constraints (Building A and Building B). 2) show how to automatically perform a cost-performance analysis; and (3) study the scalability of our approach.

Fig. VI illustrate the dataflow of the chosen application: a smart-camera network [17][18] to estimate people occupancy in a high-performance building [19], [4]. The real-time information on building occupancy may enable high-efficiency HVAC ("heating, ventilating, and air conditioning") systems and it may also dramatically improve the quality of emergency-evacuation procedures in large buildings. The dataflow of this application is characterized by a *radial symmetry* in the sense that each video stream traverses the same processing pipeline. There are 8 types of actors:

1) *Video Sensing.* Each of the actors $v_{7 \cdot j}$, where j denotes the index of the corresponding video sensor, transfers a picture from the camera sensor to the main memory.

2) *Motion Detection.* Each of the actors $v_{7 \cdot j+1}$ performs a simple presence/motion detection and triggers the execution of the subsequent tasks by forwarding only those frames that have possible people presence/motion.

3) *Preprocessing.* Each of the actors $v_{7 \cdot j+2}$ subtracts a reference background frame from each raw frame and then applies basic morphological filters to improve the image quality.

4) *Blob Extraction.* Each of the actors $v_{7 \cdot j+3}$ identifies contiguous pixel regions (*blobs*) of a frame that are brighter than the background. The outputs of these actors are the data representation for each blob.

5) *People Detection.* Each of the actors $v_{7 \cdot j+4}$ processes each blob to establish whether it represents a person. The output is the set of blobs recognized as persons.

6) *People Tracking.* Each of the actors $v_{7 \cdot j+5}$ tracks the detected people across frames by maintaining a list of their centers of mass.

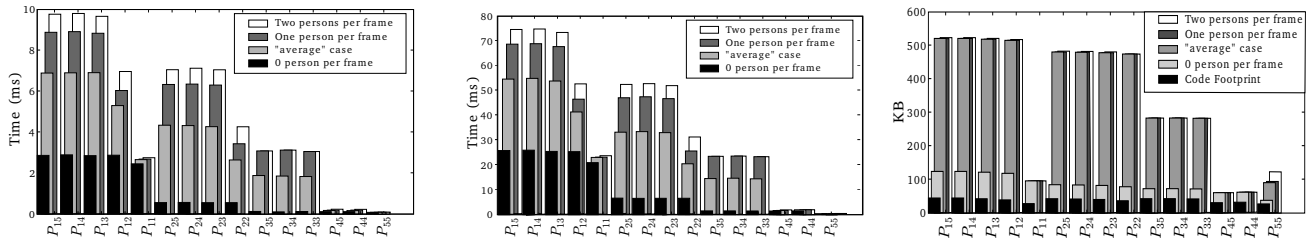
7) *People Counting.* Each of the actors $v_{7 \cdot j+6}$ determines when a tracked person crosses a hypothetical threshold between two building areas and updates one of two counters depending on the person direction (in or out).

8) *Occupancy Estimation.* Actor v_{70} (the collector), gathers the processed information for each video-stream in the network and estimates the occupancy for each building area.

Embedded Platform	Micro-controller	Memory		Cost ($)
		NOR FLASH (Mb)	RAM	
0	LPC2131	2	SRAM 2 Mb	60
1	LPC3180	16	SDRAM 64 Mb	130
2	AT91SAM	16	SDRAM 64 Mb	150
3	i.MX27L	32	SDRAM 64 Mb	170
4	PowerPC750 (@400 MHz)	32	DDR 256 Mb (@400 MHz)	220
5	PowerPC750 (@600 MHz)			225
6	PowerPC750 (@800 MHz)			255

TABLE III
THE PE LIBRARY FOR THE SMART-CAMERA NETWORK APPLICATION.

Manual Design. We profiled the software implementation of the actors on two widely-adopted embedded architectures: a 200 MHz ARM 9, which is used in many low-cost micro-controllers within smart-cameras, and a 350 MHz POWERPC, a typical mid-range CPUs suitable to process multiple video streams. Since the same sequence of actors is executed for the video-stream produced by each camera we characterized only one *branch* of the dataflow (i.e. actors $v_0, v_1, v_2, v_3, v_4, v_5, v_6$). Moreover, for the manual design, we limited the analysis to a subset of possible clusters, called *actor partitions* P_{ij}. These represent ordered sequences of data-dependent actors $\{v_i, \ldots, v_{j-1}, v_j\}$ where $i, j \in [0, 6]$ and $i \leq j$. For each partition of P_{ij}, we measured the time needed to process a frame, the instruction-memory size, the peak data-memory size, and the average output payload per frame (i.e the number of bits that must be transferred to the next actor in G). To take into account the data-dependent behavior of the application, we measured the performance of each partition using input streams composed of 1000 frames for four different scenarios, each causing a different workload: a stream capturing a scene without any person, a stream where a subset of the frames contains people, a stream where a person appears in each frame, and a stream where two persons appear in each frame. The bar diagrams of Fig. 3 report the average values of the execution times of the alternative partitions P_{ij} obtained by running the embedded software on the two processors. Specifically, the four parts of each bar correspond to the average time across 1000 experimental runs that is necessary to complete the execution of the actors belonging to P_{ij} on one



(a) POWERPC (b) ARM 9 (c) Memory Usage

Fig. 3. Embedded software profiling results: average values of frame execution time (a,b) and peak memory usage (c).

video frame for the four possible input scenarios[1]. Fig. 3(c) shows the memory taken by each P_{ij} when running on the ARM 9. Results for the other processor are similar[2].

Based on the knowledge of the dataflow and on the measured data we designed a set of possible PEs that are able to execute alternative mappings of the actors. Table III lists the PE types: each line reports a different architecture including processor, memories, and estimated cost [18]. Then, we considered two buildings (Building A and Building B) of similar size but different layouts and for each of them we distributed 10 cameras to monitor 10 zones. Each camera acquires a new image every 40 ms. By combining the PEs and networking technologies we manually derived two cost-effective implementations of the specification expressed by G: a *distributed design* where the bulk of the processing is executed locally on each camera and a *centralized design* where the cameras send their video to the collector, which is equipped with a fast processor that analyzes all video streams.

Synthesis-Driven Design. Next, we used the synthesis-based design flow of Fig. 1 to explore the design space and derive various synthesized implementations that we compared to the two manual designs. By sweeping the values of design parameter χ (see Eq. 1) we can automatically explore different design alternatives—from more centralized to more distributed ones—and evaluate the tradeoffs of the various solutions. Fig. 4 summarizes graphically the result of the design exploration. We report four quantities for each building: 1) the total cost of the synthesized system; 2) the computation cost: i.e. the cost of the required PEs; 3) the communication cost: i.e. the cost of the interconnection infrastructure onto which the various PEs, sensors, and actuators exchange data; and 4) a metric Δ^2 of the degree of centralization of a solution.

$$\Delta^2 = \frac{1}{|C|^2} \sum_{c=0}^{|C|-1} \left[\sum_{v=0}^{|V|-1} \mu_{vc} - \frac{|V|}{|C|} \right]^2 \tag{16}$$

When actors are uniformly distributed among clusters, variable Δ^2 reaches its lower bound (i.e. 0). Larger values of Δ^2 correspond to cases where some clusters aggregate more actors than others, i.e. a more centralized implementation. It is worth noting that Δ^2 is application dependent and therefore should not be compared across different applications.

For this application, large values of χ correspond to smaller communication costs and larger computation costs as computation tends to be more distributed (small values of Δ^2). In fact, the communication requirements determined by this mapping of actors can be satisfied by a low-cost technology. Conversely, small values of χ tend to generate more centralized designs (larger values of Δ^2) that require a faster and more expensive interconnection network and that can share more processing resources, thus resulting in a more efficient processing platform. Fig. 4(b) shows a sharp transition in the communication cost. For large values of χ, the network is implemented using a low-cost ARCnet-based solution, while for small values of χ a more expensive Ethernet infrastructure is required. This is an expected result since for large values of χ each smart-camera has to transmit a few bits per second; and for small values of χ we need a fast communication network that allows streaming the videos to the collector. This effect depends also on the building topology. In fact, in Fig. 4(a) the same transition is smoother because hybrid interconnecting infrastructures can be viable solutions. The opposite behavior can be observed for the computation cost. The more a solution is distributed the more expensive the supporting processing platform is. We also note that the global computation synthesis produces lower-cost solutions than the local computation synthesis.

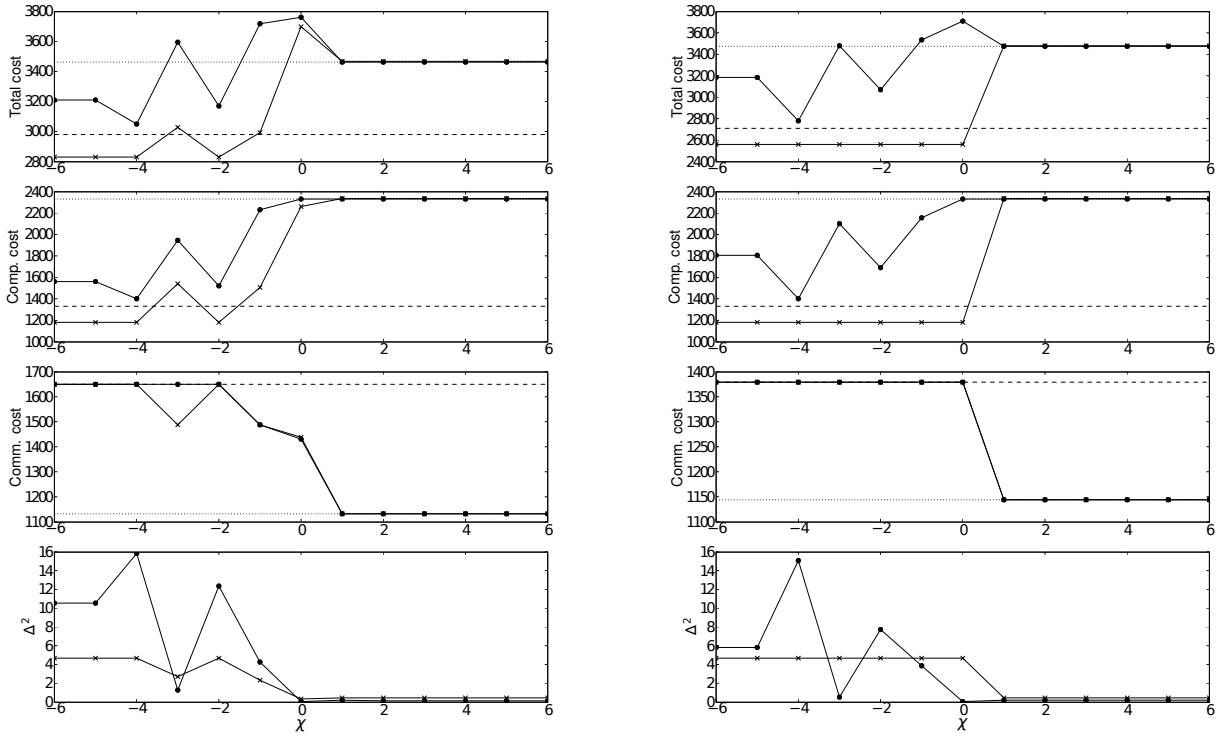
Fig. 5 shows the automatically-generated graphic representations of the results of the synthesis process for two data points of Fig. 4(b) for $\chi = -2$ and for $\chi = 2$. In Fig. 5(a) the actors are mostly mapped on a few clusters, thus imposing the transmission of large quantity of data, that can only be sustained by an Ethernet network. Conversely in Fig. 5(b), the more distributed implementation requires more powerful PEs to execute a larger number of mapped actors but in turns leads to lighter communication requirements that can be sustained by a low-speed ARCnet chain. Due to the discrete nature of the processing library, some PEs may be underloaded, resulting in a sub-optimal processing infrastructure.

When the global computation synthesis is in effect, our synthesis framework can find solutions whose cost outperforms or matches the cost of a manual design.

Multi-Objective Analysis. While we focused on the minimization of the implementation cost of the distributed execution platform for the given application, other metrics may be

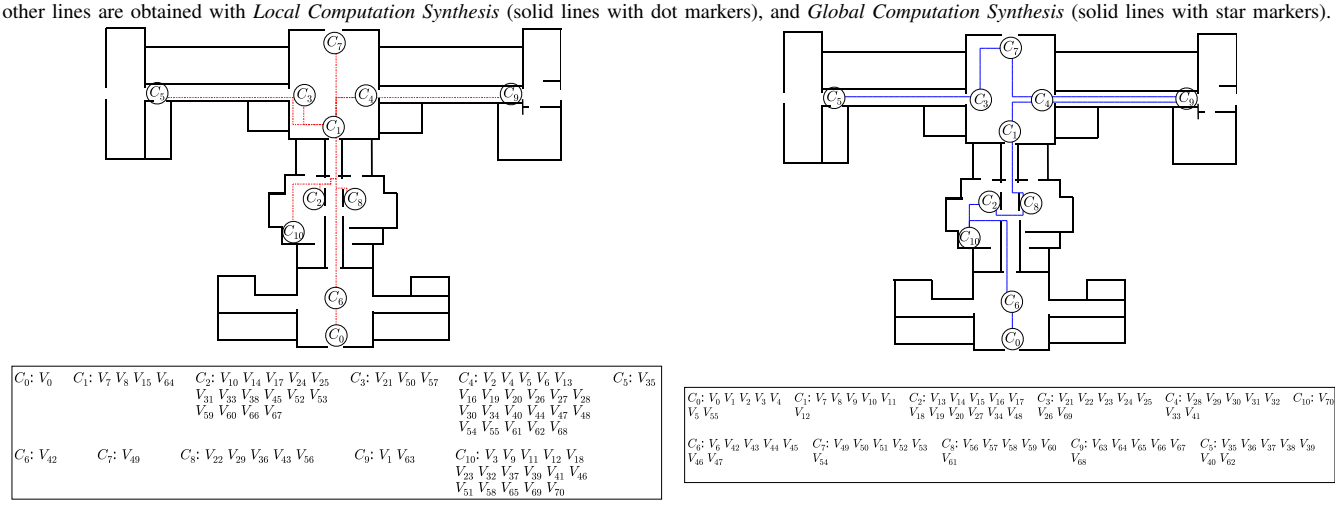
[1]Notice that the software implementation and the choice of the compiler were not optimized for any processor. This explains the similar ratios between two P_{ij} across the two processors.

[2]Data are reported only for the actors with significant execution time and memory usage (i.e. v_1, v_2, v_3, v_4, v_5).



(a) Building A (b) Building B

Fig. 4. Synthesis results for the smart-camera network case study. Four quantities are plotted as functions of χ: total cost, computation cost, communication cost, and centralization degree Δ^2. The horizontal lines correspond to a manual centralized design (dashed) and a manual distributed design (dotted). The two other lines are obtained with *Local Computation Synthesis* (solid lines with dot markers), and *Global Computation Synthesis* (solid lines with star markers).



(a) Building B, $\chi = -2$ (b) Building B, $\chi = 2$

Fig. 5. Representations of two synthesis results for the smart-camera network application in Building B, obtained with the *Global computation synthesis*. Below each building layout we report the list of actors mapped on each cluster. In (a) setting $\chi = -2$ returns a more centralized solution: the clusters are connected with high-speed Ethernet (dashed red lines) and some clusters execute many more actors than the others. In (b) setting $\chi = 2$ produces a distributed solution: clusters exchange small-payload messages on a low-speed ARCnet bus (solid blue lines). The actors are almost uniformly distributed among clusters.

of interest for a CPS architect: e.g. high-performance buildings are designed to last for years and the execution platforms must be flexible enough to accommodate future applications. To this end we can slightly modify the problem formulation to study the solution tradeoffs. For instance, to account for the relation between the computation cost and the *aggregate CPU slack*, i.e. the CPU time that remains unused during each period of the application, we can rewrite Eq. 13 as:

$$\Phi_P = \lambda \cdot \sum_{c \in C} \Phi_{Pc} - (1 - \lambda) \cdot \sum_{c \in C} Slack_c$$

where $Slack_c$ is the sum of the slacks of the PE in cluster c. By sweeping parameter $\lambda \in [0, 1]$, we derive many solutions, each with a different cost-slack tradeoff. Fig. 6 reports the Pareto set for this analysis.

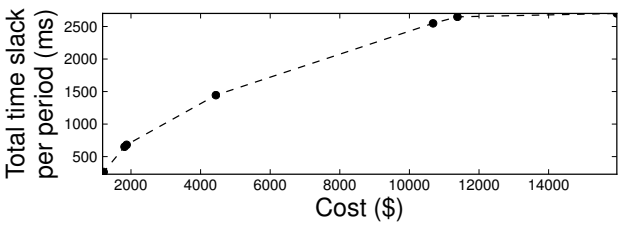


Fig. 6. Pareto frontier showing the tradeoff between the computation cost and the aggregate CPU time slack in the designed system (with $\chi = 6$).

Execution Time. The communication and computation synthesis, which are based on an ILP formulation, are the most computationally-intensive steps of our design flow. Hence, we studied the scalability of our approach as we increase the problem complexity by adding more cameras, and therefore actors, to the design. The synthesis times and solution cost for the Local and Global Computation Synthesis, and for the Communication Synthesis are reported in Table IV. We solve these problems with the Gurobi ILP solver [20] on a dual-processor Intel X5550 machine. The Communication Synthesis handles well the complexity of the given cases. The Local Computation Synthesis is also fast as it decomposes the problem into a number of simpler ones (one per cluster). Its main drawback is the large cost of the solutions compared to those of the Global Computation Synthesis. This takes significantly more time to complete, but, even in the harder case, it reaches better solutions in less than 150 seconds. Worth noticing is that the Global Computation Synthesis takes significantly less time for Building A with 40 and 50 cameras than for Building B. This difference is due to the different synthesized networks: in Building A the optimal network is heterogeneous as it results from the combination of Ethernet and Low-Speed ARCnet; whereas in Building B the clusters are connected only by Ethernet. Hence, in Building A the Global Computation Synthesis has less degree of freedom to re-map actors to clusters (due to the more stringent bandwidth constraints) and the optimal solution can be reached faster.

VII. Concluding Remarks

We presented a methodology, and its embodiment into a design flow, to automatically synthesize the distributed execution platform for a given dataflow specification of a CPS while accounting for the constraints imposed by the environment in which it operates. Our main contribution is a design flow that tackles the complexity of the synthesis problem by introducing abstraction layers and decomposing it in a sequence of steps, among which the most important are computation synthesis and communication synthesis. We restricted our attention to dataflows, a model of computation (MoC) that is suitable for a wide class of applications. To derive computation and communication constraints, however, our methodology leverages properties of dataflows (such as actors, input-output functions, scheduling, data size) that are present in some form in most MoCs. Hence, extensions of our approach to these MoCs are at least foreseeable.

Acknowledgments. This research is sponsored in part by the National Science Foundation (under Awards #: 0644202 and 0931870.)

Synthesis Time for Building A (in seconds)						
Number of Cameras		10	20	30	40	50
Local Computation	time	0.60	2.03	1.53	0.61	2.17
	cost	1560	3275	5315	7190	10295
Global Computation	time	13	150.70	1202[a]	354.81	700
	cost	1180	2595	3930	6390	8405
Communication	time	0.60	2.15	23.38	101.50	243
	cost	1649	2927	3923.5	4696.50	5342.5
Synthesis Time for Building B (in seconds)						
Number of Cameras		10	20	30	40	50
Local Computation	time	0.59	0.79	0.82	1.9	1.41
	cost	1560	3205	5410	6670	9090
Global Computation	time	13	150	1221[a]	4045[a]	35517[a]
	cost	1180	2595	3930	5220	6505
Communication	time	0.68	2.56	21	35.02	240
	cost	1379	2265	3122.5	4200	4881.5

[a] interrupted when the solution cost was 3% above the solution cost of the relaxed problem.

TABLE IV
Synthesis times with different number of cameras for Building A and Building B. Results are obtained with $\chi = -6$.

References

[1] O. Servin, H. Chen, K. Boriboonsomsin, and M. Barth, "An energy and emissions impact evaluation of intelligent speed adaptation," in *Proc. of the IEEE Intelligent Transportation Systems Conference*, Sep. 2006.

[2] M. Amin, "Toward a smart grid: power delivery for the 21st century," *IEEE Power & Energy Magazine*, vol. 3, no. 5, pp. 34–41, 2005.

[3] M. Ilic and U. Khan, "Modeling future cyber-physical energy systems," in *Power and Energy Society General Meeting-Conversion and Delivery of Electrical Energy in the 21st Century*, Jul. 2008, pp. 1–9.

[4] S. Meyn, A. Surana, Y. Lin, S. M. Oggianu, S. Narayanan, and T. A. Frewen, "A sensor-utility-network method for estimation of occupancy distribution in buildings," in *Proc. of CDC*, Dec. 2009, pp. 1494–1500.

[5] E. A. Lee, "Cyber physical systems: Design challenges," in *Proc. of ISORC*, May 2008, pp. 363–369.

[6] J. A. Stankovic, I. Lee, A. Mok, and R. Rajkumar, "Opportunities and obligations for physical computing systems," *Computer*, vol. 38, no. 11, pp. 23–31, Nov. 2005.

[7] L. Sha and J. Meseguer, "Design of complex cyber physical systems with formalized architectural patterns," in *Software-Intensive Systems and New Computing Paradigms*. Springer, Nov. 2008, pp. 92–100.

[8] S. Prakash and A. Parker, "SOS: Synthesis of application-specific heterogeneous multiprocessor systems," *Journal of Parallel and Distributed computing*, vol. 16, no. 4, pp. 338–351, 1992.

[9] J. Hou and W. Wolf, "Process partitioning for distributed embedded systems," in *Proc. of the 4th International Workshop on Hardware/Software Co-Design*, Mar. 1996, pp. 70–76.

[10] T. Yen and W. Wolf, "Communication synthesis for distributed embedded systems," in *Proc. of the International Conference on Computer Aided Design*, Nov. 1995, pp. 288–294.

[11] T. Y. Yen and W. Wolf, "Sensitivity-driven co-synthesis of distributed embedded systems," in *Proc. of the 8th International Symposium on System Synthesis*, Sep. 1995, pp. 4–9.

[12] K. Kuchcinski, "Synthesis of distributed embedded systems," in *Proceedings of the 25th EuroMicro Conference*, Sep. 1999, pp. 1022–1028.

[13] E. A. Lee and T. Parks, "Dataflow process networks," in *Proc. of the IEEE*, 1995, pp. 773–799.

[14] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, "Multilevel hypergraph partitioning: Application in VLSI domain," in *Proc. of the 34th annual Design Automation Conference*, Jun. 1997, pp. 526–529.

[15] M. Gruber and G. Raidl, "A new 0–1 ILP approach for the bounded diameter minimum spanning tree problem," in *Proc. of the 2nd International Network Optimization Conference*, Mar. 2005, pp. 178–185.

[16] A. Pinto, "A platform-based approach to communication synthesis for embedded systems," Ph.D. dissertation, EECS Department, University of California, Berkeley, May 2008.

[17] B. Rinner and W. Wolf, "An introduction to distributed smart cameras," *Proc. of the IEEE*, vol. 96, no. 10, pp. 1565–1575, Oct. 2008.

[18] F. Leonardi, A. Pinto, and L. P. Carloni, "A case study in distributed deployment of embedded software for camera networks," in *Proc. of DATE*, Apr. 2009, pp. 1006–1011.

[19] R. Tomastik, S. Narayanan, A. Banaszuk, and S. Meyn, "Model-based real-time estimation of building occupancy during emergency egress," in *Pedestrian and Evacuation Dynamics. Klingsch et al. (eds.)*. Springer, 2010, pp. 215–224.

[20] http://www.gurobi.com/.