

# COWLS: Hardware-Software Co-Synthesis of Distributed Wireless Low-Power Embedded Client-Server Systems

Robert P. Dick and Niraj K. Jha

Department of Electrical Engineering  
Princeton University  
Princeton, New Jersey 08544  
dickrp/jha@ee.princeton.edu

## Abstract

In this paper, we present a hardware-software co-synthesis algorithm, called COWLS, which targets embedded systems composed of servers and low-power clients which communicate with each other through a channel of limited bandwidth, e.g., a wireless link. Clients may be mobile. COWLS allows both hard and soft real-time constraints. It simultaneously optimizes the price of the client-server system, the power consumption of the client, and the response times of tasks which have only soft deadlines, while meeting all the hard deadlines. It produces numerous solutions which trade off different architectural features, e.g., price, power consumption, and response time, of an embedded client-server system. As far as we know, this is the first co-synthesis algorithm of its kind. We present experimental results of synthesizing a low-power, client-server camera system as well as several randomized examples.

## 1. Introduction

A bandwidth-constrained embedded client-server system is a system in which clients and servers communicate with each other via a channel of limited bandwidth. Clients are frequently consumer products, e.g., portable communication devices, for which price is often particularly important. Server price is also an important factor, although it is usually less important than client price because clients typically outnumber servers. In this work, we assume that servers have access to high-capacity power supplies, e.g., they may be plugged in to wire-delivered power or have access to high-capacity batteries. In order to maintain mobility, clients may be small and battery-powered. Therefore, client power consumption must be minimized to reduce heat production and increase battery life.

The literature abounds with case studies of embedded client-server system design and general descriptions of the client-server problem domain. Some researchers have discussed wireless and cellular systems [1], [2], some have focused on embedded systems in which the server is a satellite [3], [4], and others have studied telerobotics, systems in which a robot is partially or totally controlled via a limited-bandwidth communication channel [5], [6]. The majority of previous research on embedded client-server systems either surveys the problems typically faced by the designer of such systems or provides case studies detailing specific solutions to individual problems.

There is a significant body of work on hardware-software co-design, i.e., concurrent design of the hardware and software portions of an embedded system, and hardware-software co-synthesis, the automatic synthesis of hardware-software embedded systems [7], [8]. Work in hardware-software co-design focuses on providing a designer with tools and guidelines which ease the exploration of the available implementation options. Schulz *et al.* present a co-design approach based on the use of a simulatable, implementation-independent system representation [9]. Passerone *et al.* discuss a rapid abstract hardware-software simulation method [10]. The hardware-software co-synthesis problem is intractable. It is composed of a number of sub-problems, many of which are NP-complete, e.g., allocation/assignment and scheduling [11]. Presently, only non-exhaustive optimization al-

gorithms are capable of solving large problem instances of distributed, embedded systems in a reasonable amount of time. Researchers have tackled variants of the co-synthesis problem with iterative improvement algorithms [12], constructive algorithms [13], simulated annealing algorithms [14], evolutionary algorithms [15], [16], and a rapid, sub-optimal timing constraints solver [17].

Despite the previous work dealing with embedded client-server systems and hardware-software co-synthesis, we know of no previous work which automatically synthesizes such systems. COWLS automatically synthesizes embedded client-server systems, taking into account price, power consumption, bandwidth requirements, as well as task deadlines.

## 2. Embedded client-server system requirements

In this section, we explain task graphs, the means by which a designer specifies the requirements of an embedded client-server system. A task graph is a directed acyclic graph in which each node represents a task and each arc represents a data dependency and a communication event. In Fig. 1, the top node represents the *SCN* task type and that node's outgoing arc represents a communication of 224 kilobytes of data to the *ID* task. There may be more than one task instance of a given type, e.g., *IND* in Fig. 1. The *ID* task's incoming arc indicates that it may not begin execution until the *SCN* task has completed execution and transmitted 224 kilobytes of data to it. Any task may have a hard deadline, signified by a solid bar, or a soft deadline, signified by a dotted bar. Thus, a designer may provide a specification containing only hard real-time deadlines, a specification containing only soft deadlines, or specification containing any combination of the two types of deadlines. In the first case, COWLS will attempt to minimize price and power consumption under hard real-time deadlines. In

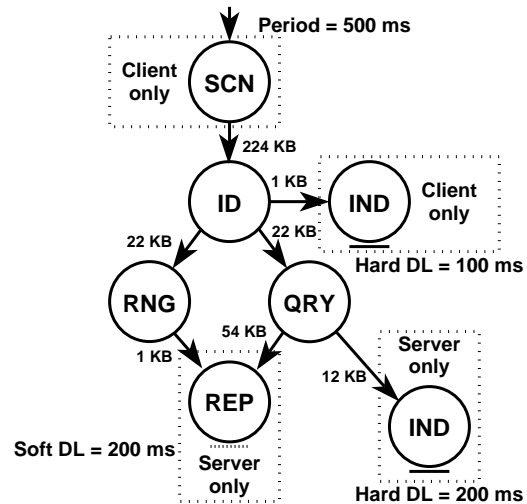


Fig. 1: Task graph

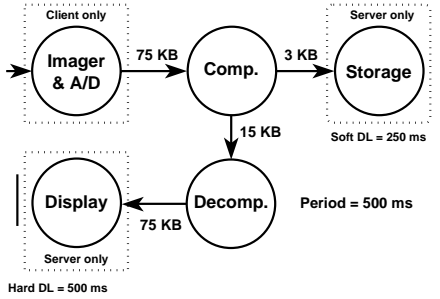


Fig. 2: Camera specification

the second case, it will attempt to simultaneously minimize soft deadline violation, price, and power consumption. If all soft deadlines are set to zero, it will minimize system execution time, price, and power consumption.

In Fig. 1, the upper *IND* task must complete execution within 100 ms of the start of the task graph's execution. This hard deadline must be met for the synthesized architecture to be valid. The *REP* task should complete execution by 200 ms after the start of the task graph's execution. However, the system will still be valid if this soft deadline is not met. In the case of a task with a soft deadline, the aim is to minimize its response time if the deadline cannot be met. The specification may require some tasks to be executed on the client, e.g., *SCN*. Others must be executed on the server, e.g., *REP*. A task graph's period is the amount of time that elapses between its subsequent executions. The task graph's period is 500 ms in Fig. 1. The requirements placed on an embedded client-server system are specified with a set of task graphs, each of which may have a different period. A task graph may have a period which is less than, greater than, or equal to the maximum deadline within it.

### 3. Motivating example

A synthesis system for the client-server problem domain should simultaneously optimize multiple costs. It must also consider the differences in cost between executing a task on a client or a server. It is, therefore, necessary to allow tasks to migrate from one side of the primary communication resource, the link connecting the client and server, to the other. Optimizing only one cost, or considering local improvements instead of system-level improvements is likely to lead to a poor overall solution.

Previous work has discussed a client-server specification for a wireless camera [18]. It gives a brief case study of the design of an embedded client-server system. Generating a high-quality architecture, which meets the specifications given, is not entirely straightforward, even for a human designer. We will show how COWLS explores the design space in a manner which allows it to uncover a similar high-quality design to the one proposed in the case study, and consider other options which trade off different system costs.

Consider a system specification requiring a battery-powered camera to transmit video information to a base station via a limited-bandwidth wireless link. If the designer has decided that the video information should be compressed, but has not yet decided what sort of processor should be used to carry out this operation, or even whether it should be done by the client or the server, COWLS will simultaneously explore the different options.

Fig. 2 shows an example of a task graph for a wireless camera. In this example, imaging and analog-to-digital conversion must be carried out on the client. Data storage and display must be carried out on the server. Storage has a soft deadline of 250 ms and display has a hard deadline of 500 ms.

In one possible partitioning, shown in Fig. 3, Imaging & A/D, as well as data compression, are executed on the client and all other tasks are carried out on the server. This partitioning reduces the load on the wireless communication link and allows an inexpensive communication resource to be used between the client and

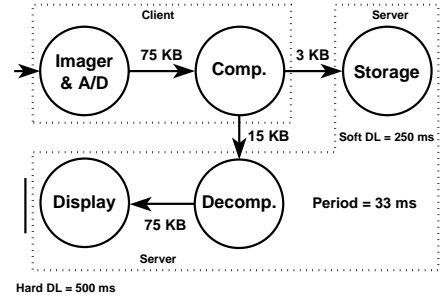


Fig. 3: Camera architecture 1

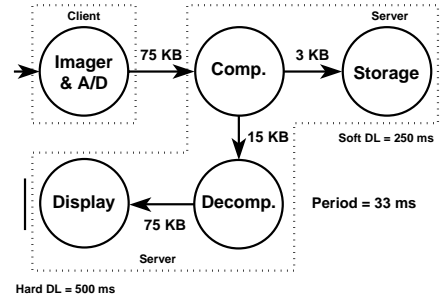


Fig. 4: Camera architecture 2

the server. However, carrying out data compression on the client requires increased client price and power consumption.

In another possible partitioning, shown in Fig. 4, Imaging & A/D is executed alone on the client and all other tasks are executed on the server. In this partitioning, the client executes only essential functions, shifting all other computational burdens to the server. This decreases the client's price and power consumption. However, it increases the demands upon the communication link between the client and the server, increasing its price. Although some of the trade-offs facing the designer of client-server systems are apparent even from this simple example, the problems tackled by COWLS are significantly larger and more complicated.

### 4. Problem formulation

In this section, we present the client-server synthesis problem formulation used for COWLS.

The independent synthesis of a client or server is similar to the distributed, heterogeneous embedded system co-synthesis problem. However, COWLS targets the servers and clients simultaneously.

The manner in which a designer specifies the requirements placed on a client-server system was explained in Section 2. It is also necessary to describe the characteristics of the resources which may be used to meet these requirements. We model three main types of resources: processing elements, communication resources, and memory.

Processing elements (PEs) model general-purpose or special-purpose processors which are capable of executing tasks. There are two types of PEs: client PEs and server PEs. A number of attributes are associated with each type of PE. More than one instance of a single type of PE may eventually be present in a synthesized client-server system. Each PE type has a price, an idle power consumption, as well as a Boolean value indicating whether or not the PE uses buffered communication. In addition, we use a table describing the performance of each task on each PE type. For each task, this table indicates whether or not the task can be executed on each type of PE. For each task which may execute on a given PE type, it holds the task's worst-case execution time, its average power consumption, and the amount of memory it requires. Due to packaging considerations (e.g., weight, size, and cooling), the memory available in the clients may differ from the memory

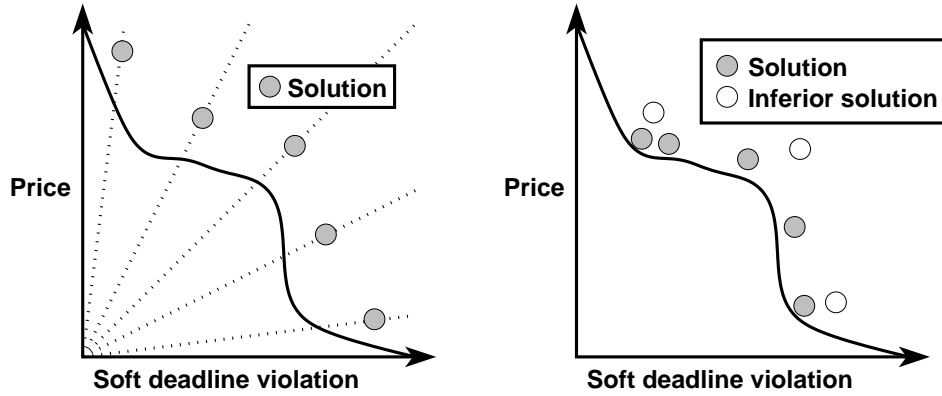


Fig. 5: Multiple runs of conventional algorithm vs. single run of multiobjective algorithm

available in the servers. Therefore, the price of memory depends on whether it is located in a client or server.

COWLS synthesizes embedded systems containing arbitrary-topology busses and point-to-point communication links [13], as well as primary communication resources, which are used to connect clients and servers. There are a number of attributes associated with each type of communication resource. There may be multiple communication resources within the client, and within the server. Numerous alternative primary communication resources may be available. However, only one primary communication resource may be present in a client-server pair, as multiple wireless transmitters and receivers will typically result in unreasonably expensive client-server systems. The use of a unique link allows a trade-off between price and bandwidth when modeling wireless communication links. Each type of communication resource has a price, a packet size (which can be very small to model communication which is not packet-based), a power consumption per packet, an amount of transmission time required per packet, and a maximum number of contacts. A communication resource's number of contacts is the number of different PEs which it may connect together, *i.e.*, a communication resource with two contacts is a point-to-point link. Primary communication resources have two prices, one of which is associated with the client and one of which is associated with the server.

Given the client-server system requirements, in the form of a set of task graphs, as well as the attributes of the PEs, memory, and communication resources available, COWLS attempts to synthesize client-server systems which meet the requirements with minimal price, client power consumption, and soft deadline violation. An architecture's costs are derived from the manner in which resources are used in its construction. Therefore, by attempting to meet real-time constraints, one ensures that high-speed PEs, which are tailored to the tasks required, are used for tasks which lie along critical paths in the task graphs. By attempting to minimize price, one ensures that PEs, which are capable of carrying out the required tasks with minimal price, are used. By attempting to minimize client power consumption, one minimizes the number of power-intensive tasks run on power-hungry PEs located on the client. Of course, some of these goals conflict with each other. For this reason, a single run of COWLS generates multiple solutions which explore the trade-offs among different costs.

## 5. Optimization framework

In this section, we explain the optimization framework COWLS uses to synthesize client-server systems.

In order to simultaneously optimize different costs without any pre-determined numerical specification of the relative importance of those costs, a synthesis system must be capable of producing multiple solutions to each problem. One can achieve this result by running a single-solution, constructive or iterative improvement algorithm multiple times. The left graph in Fig. 5 shows the results of running a conventional single-objective iterative improvement

algorithm multiple times, given different relative importance of two costs, price and soft deadline violation. In this figure, the dotted lines represent the relative importance of the two costs during each of the five runs of the algorithm. The curved solid line represents the Pareto-optimal solution curve, *i.e.*, those solutions which are not inferior to any other solutions in all of their costs [19]. The circles represent the solutions ultimately found by the algorithm. Note that, as the shape of the Pareto-optimal curve is unknown, it is impossible to determine the appropriate relative importance for the two costs before an optimization run. Thus, the algorithm spends nearly as much time during the optimization runs which have no hope of producing a high-quality solution as it does during runs which have the potential to find high-quality solutions. The end result is that computational effort is not focused on the appropriate areas of the solution space.

The right graph in Fig. 5 shows the result of running a true multiobjective algorithm once. In this figure, the gray circles represent solutions ultimately reported to the designer by the multiobjective algorithm. The white circles represent solutions which are strictly inferior to other known solutions. Although they aid in the exploration of the solution space by sharing information with other solutions, they are eliminated before results are reported to the designer.

The multiobjective algorithm focuses its efforts on the areas of the solution space which are most promising by eliminating solutions which are strictly inferior to others during the optimization run. Given equal computational resources, a multiobjective algorithm will, in general, find solutions which are closer to the most promising portions of a multiobjective problem's Pareto-optimal curve than those located by multiple runs of a conventional algorithm.

A given solution,  $A$ , dominates solution  $B$  if all of  $A$ 's costs are lower than or equal to the corresponding costs of  $B$ , and  $A$  is not equivalent to  $B$ . Instead of using a conventional cost function, COWLS computes the *rank* of each solution relative to the other solutions that are currently maintained by its optimization algorithm. A solution's rank is the number of other currently existing solutions which do not dominate it.

In Fig. 6, each oval represents a solution. The position of a solution indicates its costs, *i.e.*, its price and soft deadline violation. The number in each oval indicates the rank of the corresponding solution. Solution  $B$  has a rank of three because it is not dominated by three other solutions:  $C$ ,  $D$ , and  $E$ . Note that even though a two-dimensional solution space has been shown in Fig. 6 in order to simplify the example, in general the solution space has three or more dimensions, *e.g.*, price, soft deadline violation, and power consumption.

COWLS uses a multiobjective, evolutionary optimization framework. This optimization framework was selected because it is well-suited to solving optimization problems in which solutions have multiple costs, it is unlikely to become trapped in local minima, and its effectiveness, when applied to related synthesis

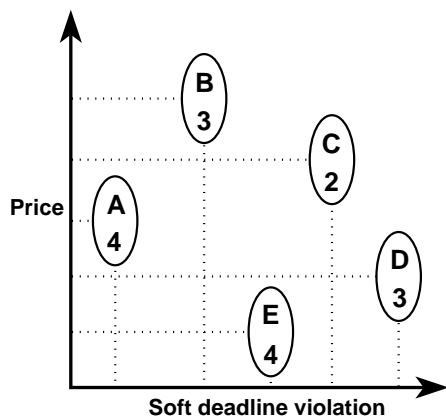


Fig. 6: Solution ranking

problems, has been demonstrated in previous work. COWLS simultaneously maintains multiple solutions which trade off different costs. Instead of using a conventional cost function, it derives solution ranks based on the quality of each solution, relative to other existing solutions. It eliminates poor solutions, those which are inferior to other existing solutions in every way, and reproduces good solutions.

## 6. Optimization formulation

In this section, we explain the manner in which solutions are represented and optimized by COWLS.

Each solution is defined hierarchically. At the top level of the hierarchy, the physical resources available to a solution are specified. Thus, the primary communication resource type, the client and server PE allocations (the numbers and types of PEs present in the client and server), and the client and server communication resource allocations (the numbers and types of communication resources present in the client and server) are specified at this level. Multiple resource allocations simultaneously exist. We refer to a group of solutions which are guaranteed to share the same resource allocations as a *cluster*. Within each cluster, there are multiple solutions. In addition to resource allocation information, assignment information is associated with each solution. Each solution is further defined by its client task assignment (information specifying the PEs upon which each client task is executed) and its server task assignment. A number of command line parameters affect the behavior of the optimization algorithm, *e.g.*, the number of solutions and the number of clusters.

The evolutionary algorithm maintains a geometrically decreasing global temperature, initially set to one, which causes its behavior to change during the course of an optimization run. Fig. 7 gives an overview of the evolutionary optimization framework used in COWLS. Initially, pseudo-random solutions and clusters are generated. The optimization algorithm consists of a set of two nested loops. Within the outer loop, the cluster loop, changes are made to existing clusters via mutation and information trading resulting in the production of new clusters. The solution loop is then entered. For each cluster, changes are made to the solutions via mutation and information trading resulting in the production of new solutions, the new solutions are scheduled, their costs are evaluated, and Boltzmann trials are conducted between the new solutions and the old solutions [20]. Solutions which lose their Boltzmann trials are eliminated until the number of solutions in each cluster is the same as it was at the beginning of execution. This inner solution loop executes multiple times (two to five times works well in practice), after which control returns to the outer cluster loop. At this point, Boltzmann trials are conducted between pairs of clusters, eliminating the losing clusters, until the number of clusters is the same as it was at the beginning of execution. This outer loop continues to execute until a number of iterations have been carried out without an improvement in the quality of the clusters (10 iterations

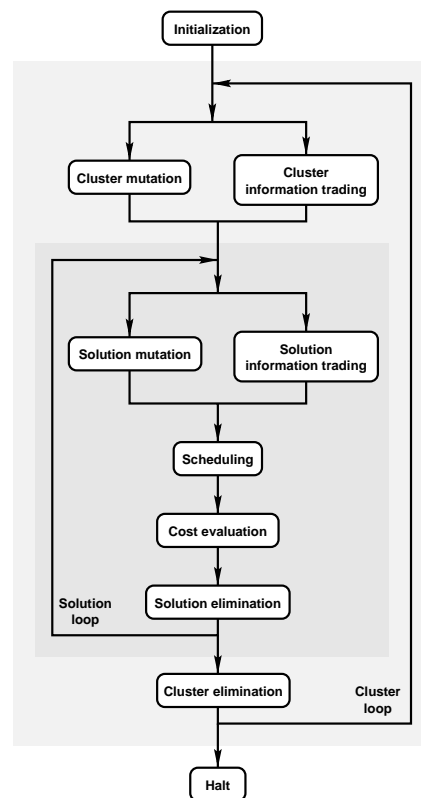


Fig. 7: Optimization overview

works well in practice), at which time the global temperature is lowered, resulting in greedier optimization. When a large number of iterations have passed without an improvement in the quality of the clusters (31 works well in practice), the algorithm is halted and all of the valid solutions encountered during the optimization run, which are not dominated by other solutions, are reported.

During an optimization run, the resource allocation and assignment information associated with solutions undergoes randomized local changes, or mutations. Every time this happens, one newly modified solution is tentatively created. Each mutation may affect communication resources or PEs. Mutation either adds or removes a randomly selected resource, although the client and server are guaranteed to maintain at least one PE each. The probability that a resource will be added is proportional to the global temperature, causing resource allocations to initially be grown, then pruned back later during an optimization run. When the primary communication resource information is mutated, another type of primary communication resource is randomly selected as a replacement.

Randomized changes, similar to those which occur to resource allocations, are made to the information specifying the manner in which local communication resources are connected to PEs as well as the assignments of tasks to PEs. Task assignment mutation is randomized. However, it is not entirely random. Task assignment mutation is guided by a heuristic designed to minimize task execution time and primary communication link bandwidth requirements. After randomly determining which task will be reassigned, this heuristic generates an array of PEs capable of executing the task. Two costs are associated with each PE: loading and distance. Loading is the approximate proportion of a PE's time which has already been occupied by the other tasks assigned to it. Distance is a metric which takes into account the likely impact of a change to a task's assignment upon the bandwidth requirements placed on the primary communication resource. Recall that each PE is located either on the client or the server. A task's neighbors are the other tasks with which it communicates, *i.e.*, the tasks connected to it by

arcs as shown in Fig. 1. Let separation be a function of two PEs. This function takes the value one if one of its arguments is located on the client and the other argument is located on the server. Otherwise, it takes the value zero. Let assigned\_to be a function of one task. Its value is the PE to which its argument is assigned. Given a PE,  $pe$ , a length  $n$  array of neighbors,  $neighbors$ , of the task under consideration, the distance,  $dist$ , of that PE is defined as:

$$dist(pe) = \frac{1}{n} \sum_{i=1}^n separation(pe, assigned\_to(neighbors_i))$$

In other words, distance is the proportion of the PEs to which a task's neighbors are assigned, which are located on the other side of the client-server PE division from the PE whose distance is being calculated. Once loading and distance have been calculated, these costs are used to compute the multiobjective ranks (see Section 5) of the PEs. The array of PEs is then sorted in order of increasing rank. Finally, the task is assigned to a PE which is selected by indexing into this array, from the highest-rank PE, with the square of a uniform random variable which ranges from zero to one, multiplied by the length of the array, *i.e.*, if  $m$  is the length of the array and  $uvv$  is a uniform random variable ranging from zero to one, then the index variable,  $index$ , is defined as:

$$index = m \cdot uvv^2$$

This selection method is designed to allow the PE ranking heuristic to guide the assignment of tasks to PEs in a probabilistic manner.

In addition to mutation, solutions may be changed by trading information with other solutions. Every time information trading occurs, two newly modified solutions are tentatively created. It has been shown that for information trading in genetic algorithms, which are closely related to the type of evolutionary algorithm employed by COWLS, it is important to keep related information together during information trading [21]. Doing so results in a dramatic decrease in optimization time ( $n$  evaluations implicitly evaluate  $n^3$  solutions), or an increase in solution quality given the same amount of optimization time. We have empirically verified this assertion. The relative frequencies of the different types of mutation and information trading are specified by empirically determined parameters.

The allocation information trading algorithm used by COWLS is designed to keep related information together more frequently than disrupting it. Each PE type is characterized by a set of parameters, *e.g.*, price and idle power consumption. Given that there are  $n$  parameters, each parameter is treated as a coordinate of an  $n$ -dimensional vector. There exists one such vector for each PE type. These vectors are then normalized to a unit-diameter  $n$ -dimensional hyper-sphere. The hyper-sphere is then bisected by a randomly positioned and randomly oriented  $n$ -dimensional hyper-plane. PE allocations associated with PE vectors on one side of the hyper-plane are swapped between the two solutions involved in information trading while PE allocations associated with PE vectors on the other side of the hyper-plane remain in their original solutions. Communication resource information trading is carried out with a similar algorithm.

After making changes to solutions, it is necessary to determine whether or not those changes resulted in improved costs. Thus, after modifying a solution, COWLS carries out cost calculation to determine its aggregate price, the client's power consumption, and the degree to which soft deadlines are violated. In addition to these visible costs, there are a number of invisible costs which need never be displayed to the designer. Hard deadline violation is an example of such a cost. All solutions in which the hard deadline violation is non-zero are eliminated before results are presented to the designer. However, during optimization, solutions with hard real-time deadline violations are allowed to exist, for they have the capacity to evolve into high-quality, valid solutions during optimization.

Aggregate price is computed by taking the sum of the prices of the PEs, memory, communication resources, and the primary communication resource associated with the client, multiplying this by the expected number of clients, and adding to this the sum of the prices of the resources used in the server multiplied by the expected number of servers. This gives a total client-server system price. If the designer is interested in optimizing only client price, the expected number of servers may be set to zero for the purpose of price computation.

In order to determine the client power consumption, soft deadline violation, and hard deadline violation, it is necessary to generate a complete schedule for a solution. COWLS uses a rapid multi-rate list scheduler which is capable of handling task graphs with periods which are greater than, equal to, or less than the deadlines in the task graphs [22]. While scheduling, bus contention is explicitly simulated. If there are more clients than servers, the appropriate number of copies of each task and communication event are scheduled on the servers in order to guarantee that they can simultaneously handle multiple clients. The scheduler is deterministic, *i.e.*, given a particular resource allocation and task assignment, it always produces the same complete, static schedule. Therefore, after scheduling, the completion times of each task and communication event are known. This allows straightforward calculation of soft and hard deadline violations. In addition, the scheduler determines the communication resources upon which each communication event occurs. This information allows the calculation of power consumption by the client's communication resources. The power consumption of each task which executes on the client, as well as the power consumed by the client PEs while idle and communicating, is added to the client communication resource power consumption to determine the total client power consumption. Once a schedule is computed for a solution, that solution's client power consumption and soft deadline violation information is stored in a look-up table and used for any equivalent solutions which subsequently arise during optimization.

Once every solution's costs are known, the evolutionary optimization algorithm globally compares every solution with every other solution in order to determine which solutions are inferior to others, as defined in Section 5. A solution's rank is the total number of solutions minus the number of solutions to which it is inferior. Boltzmann trials [20], using the global temperature and the ranks of the two solutions, are conducted between randomly selected pairs of solutions, one of which is a newly modified solution. In a Boltzmann trial, the degree of correlation between the superiority of a solution's rank over its competitor's rank, and its probability of winning the trial, is inversely related to the global temperature. The losers of these trials are deleted, until the number of solutions is the same as it was after initialization.

## 7. Experimental results

In this section, we present the results COWLS produced when run on the camera client-server example shown in Fig. 2, as well as a number of pseudo-random examples. We know of no previous work on automatic synthesis of client-server systems. Therefore, it is not possible to compare the performance of COWLS against prior art. However, the general multiobjective evolutionary optimization framework employed in COWLS has performed equally well or better when compared with other co-synthesis systems for distributed, non-client-server, architectures [16]. For these examples, COWLS was used to explore the trade-offs between different system costs, instead of attempting to minimize a single cost. Given a similar amount of CPU run time, it would be possible to better optimize a single cost by ignoring all other costs. However, this approach would ignore the fundamentally multiobjective nature of embedded system design. Each synthesis run reported in this section took less than ten minutes on a PentiumPro running at 200 MHz with 96 MB of memory.

The pseudo-random examples, as well as the resource database of the camera examples, were produced with the aid of TGFF, a set of algorithms which generates multiple task graphs and resource descriptions based upon a number of parametric defini-

Table 1: Camera experiments

Example number	System price (USD)	Client price (USD)	Server price (USD)	Soft DL violation	Client power (mW)
1	1,762	67	87	0.108	643.9
	1,768	67	93	0.060	2,685.3
	1,785	67	110	0.060	643.9
	1,883	71	108	0.112	641.1
	1,889	71	114	0.064	641.1
	2,534	79	559	0.150	590.2
2	2,596	79	621	0.148	590.2
	1,762	67	87	0.310	729.7
	1,789	67	114	0.263	729.7
	2,534	79	559	0.353	676.0
	2,596	79	621	0.350	676.0
3	3,579	120	579	0.344	676.0
	1,762	67	87	0.513	815.5
4	1,789	67	114	0.465	815.5
	1,789	67	114	0.673	901.2
4	1,810	67	135	0.667	901.2

Table 2: Randomized multiobjective experiments

Example number	System price (USD)	Client price (USD)	Server price (USD)	Soft DL violation	Client power (mW)
1	1,639	76	119	3.235	2,847.2
	1,662	76	142	3.621	2,789.7
2	906	37	166	2.531	3,594.0
	974	40	174	0.699	1,024.6
3	1,392	42	552	3.279	11,932.2
	1,462	56	342	2.765	12,040.5
4	6,660	322	220	2.583	6,752.0
	7,120	345	220	2.431	6,469.1
	7,130	345	230	2.682	2,635.2
	7,142	345	242	2.357	6,440.4
5	3,995	172	555	3.163	1,997.2
6	1,095	49	115	2.730	1,544.9
7	1,197	55	97	2.341	1,977.5
8	1,223	49	243	2.282	1,224.2
	1,234	49	254	2.576	1,160.7
	2,281	100	281	2.270	7,196.4
10	4,888	212	648	5.490	4,943.7
	5,188	227	648	5.039	4,968.7
12	1,295	60	95	2.232	1,360.2
	1,349	63	89	2.346	1,238.5
	1,581	64	301	2.032	1,101.2

tions [23]. Parameters were derived from textbooks dealing with wireless communication [24] as well as trade journals which describe general-purpose and embedded processors [25].

Table 1 shows the results of running COWLS on the camera client-server system task graph shown in Fig. 2. In these examples, there are four types each of the following resources: client PEs, client communication resources, and server communication resources. In addition, there are five types of server PEs and six types of primary communication resources. Separate types of memory are provided for the clients and the servers. The attributes vary randomly from resource to resource and are based on values found in the literature, *e.g.*, in Example 1, the client-side active power consumption of the different client, server, and primary communication resources available varied from 94 mW to 4,078 mW. The same set of COWLS command line parameters

were used for each one of the camera examples.

A solution’s soft deadline violation proportion is equal to the sum of all soft deadline violations within it divided by the solution’s hyperperiod. For each run, a different set of client PEs were provided. For a given example number, *ex*, the average execution time,  $av\_ex\_time$ , of a task on a client PE is defined as:

$$av\_ex\_time = 5 \text{ ms} + (ex - 1) \cdot 50 \text{ ms}$$

The PEs available in the higher-number examples tend to be slower than those available in the lower-number examples. Thus, the soft deadline violation proportions tend to increase with increasing example numbers. As a result of the tight soft deadline constraints, soft deadline violations occur in all four examples. For a number of examples, COWLS produces multiple solutions which trade off soft deadline violations, price, and power consumption, *e.g.*, Ex-

ample 1. In the third and fourth columns of Table 1, the prices for a single client and server are given. However, for the purpose of optimization, COWLS used an aggregate value based on the number of clients and servers in the target system. For these examples, there were twenty five clients and one server. Thus, given a client price of 67 USD and a server price of 87 USD, the aggregate system price would be  $67 \text{ USD} \cdot 25 + 87 \text{ USD} \cdot 1 = 1,762 \text{ USD}$ . As average task execution times become high enough to make solutions which meet their hard real-time constraints rare, the number of solutions produced by COWLS decreases, e.g., Example 4.

In general, COWLS assigns tasks to the clients and server such that only a small amount of data must be transferred via the wireless link. For most of the solutions produced for Example 1, data are sent over the wireless link only after compression. Occasionally, some of the decisions made by COWLS seem counter-intuitive. For example, the second solution produced by the run on Example 1 has a significantly higher power consumption (2,685.3 mW) than the other solutions. In this case, COWLS has assigned the decompression task to the client. This increased the power consumption of the client PE used for decompression, as well as the power consumption of the primary link, which was required to transmit uncompressed data to the server. However, by making this counter-intuitive choice, COWLS found a solution which has a lower system price (1,768 USD), for the given soft deadline violation (0.060), than any of its other solutions. Solutions minimizing power consumption at the expense of price or soft deadline violation were also produced during the optimization run.

Table 2 shows the result of running COWLS on multi-rate, pseudo-random examples, each of which contains four task graphs. Within each task graph, there are between seven and thirty five tasks. The resource databases are similar to those used in the camera examples. The same set of COWLS command line parameters were used for each one of the pseudo-random examples. COWLS did not produce solutions for some of the problems, e.g., Example 9 and Example 11. Note that there is no guarantee that all of the problem instances generated by TGFF are solvable. For a number of examples, multiple non-dominated solutions were produced by each design run, as in the camera experiments.

Although the client-side transmission rates of the primary communication links available vary widely, their average is 470 Kbps, which would be provided by a channel with a bandwidth of 300 KHz and a spectrum efficiency of 1.6 bps/Hz, which is a relatively small portion of the available short-range RF spectrum in most nations [24]. An exhaustive listing of the parameters defining the camera and pseudo-random example task graph sets and resource databases is beyond the scope of this paper. However, the parameters defining these examples, as well as the examples themselves, are available via anonymous FTP [26].

## 8. Conclusions

COWLS automatically synthesizes embedded client-server architectures. It uses a multiobjective evolutionary algorithm to simultaneously produce multiple solutions which trade off different costs. It optimizes price, client power consumption, and soft deadline violations under hard real-time constraints and constrained client-server communication bandwidth. The experimental results show that COWLS frequently makes design decisions which are similar to those which would be intuitive to a human designer. However, it occasionally makes counter-intuitive decisions which are preserved if they assist in the evolution of non-dominated solutions.

## References

- [1] D. Halchin and M. Golio, "Trends for portable wireless applications," *Microwave J.*, vol. 40, pp. 62–78, Jan. 1997.
- [2] S. Komaki and E. Ogawa, "Trends of fiber-optic microcellular radio communication networks," *IEICE Trans. Electronics*, vol. E79-C, pp. 98–103, Jan. 1996.
- [3] G. Comparetto and R. Ramirez, "Trends in mobile satellite technology," *Computer*, vol. 30, pp. 44–52, Feb. 1997.
- [4] F. Ananasso and F. D. Priscoli, "Issues on the evolution towards satellite personal communication networks," in *Proc. IEEE Global Telecommunications Conf.*, pp. 541–545, Nov. 1995.
- [5] R. E. Barry and J. P. Jones, "Rapid world modeling from a mobile platform," in *Proc. IEEE Int. Conf. Robotics and Automation*, pp. 72–78, Apr. 1997.
- [6] D. W. Gage, "Telerobotic requirements for sensing, navigation, and communications," in *Proc. IEEE National Telesystems Conf.*, pp. 145–148, May 1994.
- [7] D. D. Gajski, F. Vahid, S. Narayan, and J. Gong, *Specification and Design of Embedded Systems*. Prentice-Hall, Englewood Cliffs, NJ, 1994.
- [8] G. De Micheli and R. K. Gupta, "Hardware/software co-design," *Proc. IEEE*, vol. 85, pp. 349–365, Mar. 1997.
- [9] S. Schulz *et al.*, "Model-based codesign," *Computer*, pp. 60–67, Aug. 1998.
- [10] C. Passerone *et al.*, "Fast hardware/software co-simulation for virtual prototyping and trade-off analysis," in *Proc. Design Automation Conf.*, pp. 389–394, June 1997.
- [11] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, NY, 1979.
- [12] W. H. Wolf, "An architectural co-synthesis algorithm for distributed, embedded computing systems," *IEEE Trans. VLSI Systems*, vol. 5, pp. 218–229, June 1997.
- [13] B. P. Dave, G. Lakshminarayana, and N. K. Jha, "COSYN: Hardware-software co-synthesis of heterogeneous distributed embedded systems," *IEEE Trans. Very Large Scale Integration Systems*, vol. 7, pp. 92–104, Mar. 1999.
- [14] T. Benner and R. Ernst, "An approach to mixed systems co-synthesis," in *Proc. Int. Workshop Hardware/Software Codesign*, pp. 9–14, Mar. 1997.
- [15] J. Teich, T. Blickle, and L. Thiele, "An evolutionary approach to system-level synthesis," in *Proc. Int. Workshop Hardware/Software Codesign*, pp. 167–171, Mar. 1997.
- [16] R. P. Dick and N. K. Jha, "MOGAC: A multiobjective genetic algorithm for hardware-software co-synthesis of distributed embedded systems," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 17, pp. 920–935, Oct. 1998.
- [17] K. Kuchcinski, "Embedded system synthesis by timing constraints solving," in *Proc. Int. Symp. on System Synthesis*, pp. 50–57, Sept. 1997.
- [18] A. Chandrakasan *et al.*, "Trends in low power digital signal processing," in *Proc. Int. Symp. Circuits & Systems*, pp. 604–607, May 1998.
- [19] C. M. Fonseca and P. J. Fleming, "Multiobjective genetic algorithms made easy: Selection, sharing and mating restrictions," in *Proc. Genetic Algorithms in Engineering Systems: Innovations and Applications*, pp. 45–52, Sept. 1995.
- [20] S. W. Mahfoud and D. E. Goldberg, "Parallel recombinative simulated annealing: A genetic algorithm," *Parallel Computing*, vol. 21, pp. 1–28, Jan. 1995.
- [21] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, MA, 1989.
- [22] R. P. Dick and N. K. Jha, "CORDS: Hardware-software co-synthesis of reconfigurable real-time distributed embedded systems," in *Proc. Int. Conf. Computer-Aided Design*, pp. 62–68, Nov. 1998.
- [23] R. P. Dick, D. L. Rhodes, and W. Wolf, "TGFF: task graphs for free," in *Proc. Int. Workshop Hardware/Software Codesign*, pp. 97–101, Mar. 1998.
- [24] T. S. Rappaport, *Wireless Communications: Principles & Practice*. Prentice Hall, NJ, 1996.
- [25] "Computer design." Product trends sections of vol. 35: n. 2, 6, 8, 9, vol. 36: n. 1, 9, and vol. 37: n. 1–3.
- [26] <ftp://ftp.ee.princeton.edu/pub/dickrp/COWLS>.