平行計算機系統軟硬體共合成方法論

CMAPS: A Cosynthesis Methodology for Application-Oriented General-Purpose Parallel Systems†

熊博安
Pao-Ann Hsiung
Institute of Information Science
Academia Sinica, Taipei, Taiwan.
E-mail: eric@iis.sinica.edu.tw

李宗演, 陳少傑
Trong-Yen Lee, and Sao-Jie Chen
Department of Electrical Engineering
National Taiwan University, Taipei, Taiwan.
E-mail: csj@cc.ee.ntu.edu.tw

摘要

本論文提出一套設計平行計算機系統之軟體及硬體的方法論。

Abstract

Given an application problem, a hardware-software solution is derived such that the synthesized software, a parallel pseudo-program, can be scheduled and executed on the synthesized hardware, a set of system-level parallel computer specifications, with heuristically optimal performance. This is known as system-level cosynthesis of application-oriented general-purpose parallel systems for which a novel methodology called Cosynthesis Methodology for Application-Oriented Parallel Systems, *is presented.*

Keywords: application-oriented general-purpose multiprocessor systems, comodeling, cosynthesis

1 Introduction

A system is often designed from a set of behavioral or architectural specifications, than from the original requirements of a user. This is called *system design*. Before system design, a user's requirements must often be analyzed to derive system specifications. This is called *requirements analysis*. Much research work have been done in developing methods, either technical or formal, to design a system from specifications. A user often has to specify in elaborate detail the behavior or architecture of the designed system. As far as design automation is concerned, it would be certainly desirable if a user's requirements could be directly input to a synthesis tool or methodology and a system designed from the requirements. This paper provides a solution within the hardware-software codesign perspective.

Synthesis is the process of automatically transforming a set of high-level system specifications to a lower-level design schematic that includes more architectural details required for the physical design of the system. Hardware synthesis has helped designers to reduce design time, effort, and cost. Several methodologies and tools have been proposed at each level of synthesis [10, 9, 3, 7]. When software is simultaneously synthesized, it is called *cosynthesis* or *codesign* which requires system partitioning, hardware-software tradeoff refinements, and cosimulation. Embedded digital systems and DSP applications are often targets of hardware-software cosynthesis [21, 13].

Increasing diversity in user requirements for computer applications implies higher budget allocation for several different specialized systems. The overall cost expended by a user needing to run several applications can be lowered at the expense of a slight decrease in performance by using a general-purpose parallel computer system whose subsystems are appropriately configured for executing some given applications. Our target system of cosynthesis is somewhat different from the traditional *application-specific parallel* (ASP) systems. We consider the codesign of *application-oriented general-purpose parallel* (AOGPP) systems, which are defined as general-purpose systems with their subsystems designed for the efficient execution of some software solution to a given problem. The reason for selecting such a target system is intuitive. On one hand, a purely *general-purpose system* is a performance-balanced system which may not give the best performance in solving a specific problem, and on the other hand, an *application-specific system* often cannot be used to solve any problem besides the original application that it was designed for.

Section 2 describes some previous and related

†This research was supported by the National Science Council, Taipei, Taiwan under grant NSC 86-2221-E002-066.

work. Section 3 defines AOGPP system cosynthesis and the three repositories used in the design. Section 4 describes our cosynthesis methodology for AOGPP systems. A conclusion is drawn in Section 5.

2 Previous and Related Work

As far as hardware design is concerned, methodologies for the system-level synthesis of general-purpose multiprocessor systems have been proposed recently. *Performance Synthesis Methodology* (PSM) [10] and *Intelligent Concurrent Object-Oriented Synthesis* (ICOS) methodology [9] are two of the most recently proposed methodologies. Some other successful methodologies for hardware design include the MICON System [3, 7] and the Megallan System [6].

Current codesign researches are all devoted to application-specific systems such as heterogenous multiprocessor systems [16], DSP applications [13], embedded digital systems [8], and distributed embedded computing systems [21, 24, 22, 23]. Application-specific systems typically require system partitioning into hardware and software parts. Therefore, current researches are typically devoted to hardware-software partitioning and tradeoffs exploration [17, 2], which include strategies to move operations from software to hardware [5] and from hardware to software [8], to allocate functions in an 1-CPU/n-ASIC system [20], to use multiple task graphs for heuristic cosynthesis [24], and to derive method dataflow graphs from object-oriented specifications for the construction of distributed hardware-software topologies [22].

The codesign framework proposed by Kumar et al. [14] presented an important concept of iterative system refinements using an integrated hardware-software model. The codesign methodology proposed by Thomas et al. [18] used a mixed hardware-software system model that facilitated cosimulation and cosynthesis. Gupta and De Micheli [8] proposed the cosynthesis of digital systems which used timing constraints to delegate tasks between hardware and software. Yen and Wolf [21] considered the codesign of embedded computing systems. Their target design consisted of a hardware engine made up of several processing elements (PE) which could be either CPUs or ASICs and an application software architecture with allocation and scheduling of processes and communication [24]. The advantages of an object-oriented (OO) specification were explored by Wolf [22], including the two levels of partition granularity inherent in OO specifications, the encapsulation of system objects, and the natural cut points provided by method decomposition.

From the above literatures, we have adapted a few techniques into our methodology such as the iterative refinement of an integrated system, the mixed hardware-software model, and the graph-based software models.

3 Cosynthesis Problem

System-level cosynthesis of application-oriented general-purpose parallel systems is defined as follows.

Definition 1 AOGPP System Cosynthesis*: Given an application problem composed of several elementary subproblems, a complete parallel system including the hardware system architecture description and the software program solution, is to be synthesized such that the given problem can be optimally solved by executing the synthesized software on the synthesized hardware system.*

Optimal execution of software tasks on a parallel system requires *multiprocessor task scheduling* [15] which is a known NP-complete problem [19], hence it is concluded that AOGPP system cosynthesis is at least NP-hard.

Since we work at the system-level of design, scalability in terms of the complexity of the application problem and the upgradability to new technologies are two major issues of any proposed codesign methodology. Scalability is increased in our methodology through the use of modularized problem models. A user can specify a complex application problem by referring to the *elementary problems* in a Problem Base and describing how the selected elementary problems compose into the desired application problem. Upgradability is made easy through the use of *elementary algorithms* which act as off-the-shelf building blocks for software and the use of subsystem architecture models for hardware. Three repositories are used in our methodology, namely *Problem Base* (PB), *Algorithm Base* (AB), and *Model Base* (MB), which represent the modularizations of specification input, of software synthesis, and of hardware synthesis, respectively.

PB is used to store elementary problems and related information such as the unique problem name and pointers to the corresponding elementary algorithms that can be used to solve the specific problem. For example, sorting a sequence, solving a set of linear equations, generating permutations and combinations, and computing the discrete Fourier transform are all elementary problems. A list of elementary problems is shown in Table 1. AB is a collection of elementary parallel algorithms that can be used to solve the problems in PB. Related information, such as the time and space complexities, and the requirement restrictions on the hardware architecture are all stored along with each algorithm. A partial Algorithm Base is shown in Table 1. The algorithms

Table 1: The three repositories: PB, AB, and MB

$p\#$	**Problem Names**	$a\#$	$t(n)$	$p(n)$	CM	ML	MA	CO
p_a	Sorting a sequence	a_{a_1}	O(1)	n^2	SM	NUMA	CRCW	SIMD
		a_{a_2}	O($n$log$n$)	N	SM	NUMA	CREW	SIMD
		a_{a_3}	O($n$logn/N)	N	SM	NUMA	EREW	SIMD
		a_{a_4}	O(n)	n	SM	NUMA	EREW	SIMD
p_b	Solving systems of linear equations	a_{b_1}	O(n)	n^2	SM	NUMA	CREW	SIMD
		a_{b_2}	N/A	N	SM	NUMA	CREW	MIMD
p_c	Finding roots of nonlinear equations	a_{c_1}	O($\log_{N+1} w$)	N	SM	NUMA	CREW	SIMD
		a_{c_2}	N/A	N	SM	NUMA	CRCW	SIMD
p_d	Minimum spanning tree	a_{d_1}	O(n^2/N)	N	SM	NUMA	EREW	SIMD
p_e	Prefix sums	a_{e_1}	O(log$n$)	n	SM	NUMA	EREW	SIMD
		a_{e_2}	O(log$n$)	$2n-1$	MP	Tree	EREW	SIMD
		a_{e_3}	O($n^{1/2}$)	n	MP	Mesh	EREW	SIMD

are from Akl's book on parallel algorithms [1]. MB is a repository of models for hardware subsystems, such as *Communication* models (CM), *Memory Latency* models (ML), *Memory Access* models (MA), and *Control* models (CO).

4 Cosynthesis Methodology

Having gone through the basic concepts, we explain our methodology called *Cosynthesis Methodology for Application-Oriented Parallel Systems* (CMAPS) in this section. As shown in Fig. 1, the design flow is divided into three main phases: (1) *Initialization*, (2) *Modeling and Evaluation*, and (3) *Synthesis and Simulation*.

In brief, designers can input their specifications by constructing a Problem Graph using elementary subproblems from a Problem Base, along with subproblem sizes and other related constraints. First, CMAPS maps this graph into an initial solution. Then, CMAPS transforms the initial solution into hardware models and software models, and coevaluates them while checking which models can be eliminated to decrease the complexity of synthesis. Finally, the hardware and software models are synthesized into hardware system-level specifications and software pseudo-programs, respectively, and a cosimulation of hardware and software is performed after having chosen an appropriate scheduling algorithm.

4.1 Initialization Phase

The designer specifies his or her problem using a *Problem Graph* (PG) which is a directed acyclic graph $G_P(V_P, E_P)$, such that $V_P = \{v_i \mid$

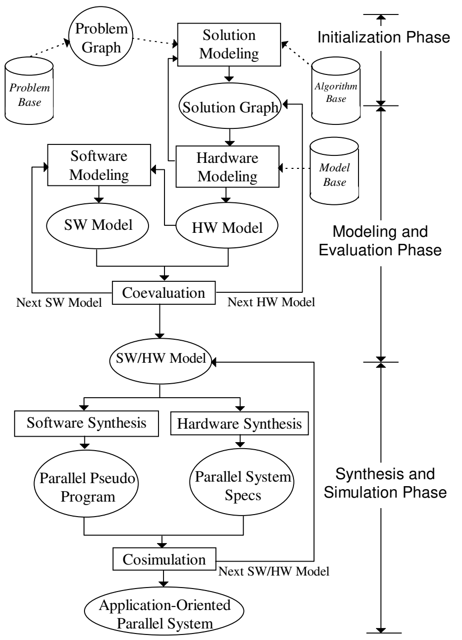

Figure 1: CMAPS Design Flow

v_i represents a problem $p_i \in P\}$ and $E_P = \{(v_r, v_s) \mid v_r$ must be solved before v_s and $v_r, v_s \in V_P\}$, where P is a set of problems in PB. This graph is similar to the traditional task graph specification used in distributed system synthesis [4] and cosynthesis algorithms [16, 24, 22, 23].

The result of this phase is a *Solution Graph* (SG), which is defined to be a directed acyclic graph $G_S(V_S, E_S)$, where each vertex in V_S represents an elementary algorithm from AB and each edge in E_S represents the order of precedence between two algorithms.

A PG input $(G_P(V_P, E_P))$ is transformed into an SG result $(G_S(V_S, E_S))$ through the following solution modeling process:

```
model_solution(G_P, G_S, A)
begin
 for each v_i in V_P do
  select a_i from A such that
  (1)a_i solves p_i
  (2)c(a_i)=MIN{c(a_k)|a_k solves p_i}
     where c(a_i)=time(a_i)*#cpu(a_i)
  V_S = set_union(V_S, {a_i})
 endfor
 for each (v_r, v_s) in E_P do
  E_S = union(E_S, {(a_r, a_s)})
 endfor
end.
```

The resulting SG is most probably not a feasible solution, but it serves as a useful initial solution for the *Modeling and Evaluation* phase. The various phases in this section are illustrated using a small running example given in Fig. 2. The Problem Graph consists of five subproblems, $p_1, p_2, \ldots, p_5$, each being an elementary problem from the Problem Base. Five subalgorithms, $a_1, a_2, \ldots, a_5$, are selected from the Algorithm Base, each being the algorithm that best solves the corresponding problem. These subalgorithms are composed into the initial Solution Graph, as shown in Fig. 2.

4.2 Modeling and Evaluation Phase

Solution Graph (SG) obtained in the *Initialization* phase is made feasible iteratively through an interleaving of hardware and software modeling processes. This phase consists of three subphases: *Hardware Modeling*, *Software Modeling*, and *Coevaluation*. Using SG, a *Hardware Model* (HM) is generated in the *Hardware Modeling* subphase by going through the following steps: *Initialization*, *Model-Space Exploration*, and *Model Configuration* steps. The *Software Modeling* subphase mainly constitutes the transformation of a Solution Graph (SG) into a *Software Model* (SM), the difference is that SG may be non-feasible, but SM has to be feasible, that is, its requirements match those provided by the corresponding HM. The final *Coevaluation* subphase reduces the number of hardware and software models to be considered for synthesis, thus significantly decreasing the complexity of cosynthesis.

4.2.1 Hardware Modeling Subphase

In the following, we assume that a given problem has n subproblems, that is, $|V_S| = n$, where SG = (V_S, E_S) is the Solution Graph of the given problem. We also assume that a *Hardware Model* (HM) has m *features*, where a *feature* is a hardware design characteristic; for example, some features can be the memory organization, the system interconnection network, etc. Further, each hardware feature may have different values assigned to it, we call them *feature options*; for example, `Shared Bus`, `Mesh`, and `Hypercube` are feature options for the system interconnection network feature. As described below, this subphase consists of three steps *Initialization*, *Model-Space Exploration*, and *Model Configuration*.

Step a. *Initialization*: An $n \times m$ *hardware requirement matrix*, $M(m_{ij})$, is constructed as follows such that m_{ij} represents the jth hardware model feature (f_j) of the ith subalgorithm (a_i), $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, m$.

1. Sort the hardware model features in a descending order of the overall degree of effect that a feature has on the system or in a descending order of the degree of importance as stipulated by a system designer.
2. Denote feature options using binary values from the set $\{1, 10, 100, \ldots\}$ such that a larger value indicates a functionally stronger option, e.g., CRCW = 100, CREW = 010, and EREW = 001 in the case of memory access models.
3. Let $\text{bit}(m_{ij}, k)$ return the kth least significant bit of m_{ij} and let b_j be the number of significant bits in the binary representation of the jth feature f_j, for all $k = 1, 2, \ldots, b_j$.

$$\text{bit}(m_{ij}, k) = 1 \text{ if } a_i \text{ requires } k\text{th option of } f_j \tag{1}$$

For instance, the matrix M for the small running example given in Fig. 2 is given in the figure.

Step b. *Model-Space Exploration*: In this step, as given in Equation (2) the kth option of the jth feature is considered for further software modeling (denoted by $t_{jk} = 1$) if the *option demand* (s_{jk}) is at least the *mean demand* (n/b_j), $k = 1, 2, \ldots, b_j$. Here, the *option demand* is defined as the number of subalgorithms which demand the kth option of the jth feature and the *mean demand* is the average weight assigned to each feature, that is, the mean demand for f_j is n/b_j.

$$t_{jk} = \begin{cases} 1 \text{ if } s_{jk} \geq \frac{n}{b_j}, \text{ where } s_{jk} = \sum_{i=1}^{n} \text{bit}(m_{ij}, k) \\ 0 \text{ otherwise} \end{cases} \tag{2}$$

For the running example in Fig. 2, $n = 5$, $m = 4$, and using Equation (1) and Equation (2) t_{jk} are computed.

Step c. *Model Configuration*: In this step, the hardware model configurations are generated from

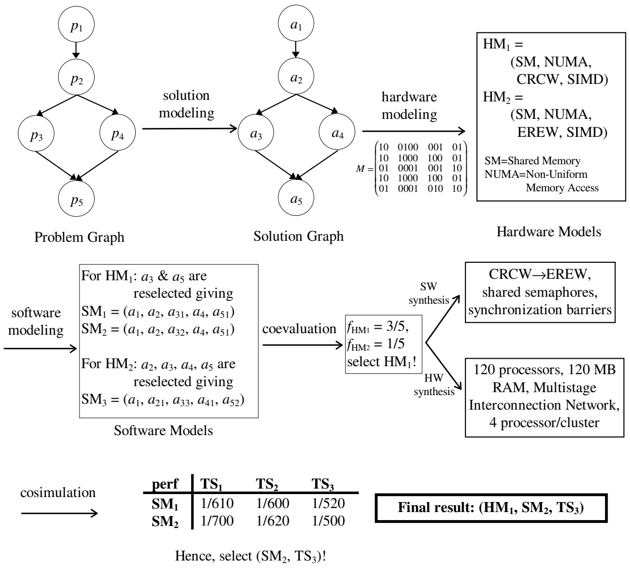

Figure 2: CMAPS Running Example

the *hardware model vector*, $\vec{v}$, which is defined from t_{jk} as $\vec{v} = (t_{11} \ldots t_{1b_1}, t_{21} \ldots t_{2b_2}, \ldots, t_{m1} \ldots t_{mb_m})$. For the running example, $\vec{v} = (10, 1001, 101, 01)$. A designer may specify some hardware requirements which will be represented by a *hardware specification vector* $\vec{u}$. The hardware model configurations are then generated.

The configurations are generated starting from the *functionally strongest* one. Non-feasible hardware models are eliminated. For our running example in Fig. 2, after eliminating non-feasible hardware models, the final feasible configurations generated are $(10, 1000, 100, 01)$ and $(10, 1000, 001, 01)$ corresponding to HM_1 = (SM, NUMA, CRCW, SIMD) and HM_2 = (SM, NUMA, EREW, SIMD).

4.2.2 Software Modeling Subphase

Considering one at a time the feasible hardware models generated in the *Modeling and Evaluation Subphase*, software models are generated by transforming the Solution Graph (SG) into a feasible software solution. This transformation process checks the compatibility of each subalgorithm in SG with the hardware model under consideration. *Compatibility* is defined in terms of the hardware model features, $\{CM, ML, MA, CO\}$, of which *Communication* (CM) model and *Control* (CO) model require an exact match, whereas *Memory Latency* (ML) model and *Memory Access* (MA) model are defined *compatible* when the hardware feature is functionally stronger than the software requirement; for example, CRCW PRAM and CREW algorithm are assumed compatible. Due to space limitations, the software modeling phase is not explained in detail here.

4.3 Synthesis and Simulation Phase

In this phase, the hardware and software models are now individually synthesized into parallel system specifications and parallel pseudo-programs, respectively, and then cosimulated by scheduling the parallel program on the parallel architecture defined by the parallel system specifications. The *Hardware Synthesis* subphase consists of four steps: *System Configuration*, *Processor Clustering*, *System Interconnection Selection*, and *Cluster Design*. The *Software Synthesis* subphase interconnects the final choice of algorithms by *Algorithm Interface Construction*, *Serialization of Memory Accesses*, and *Addition of Communication Constructs*. In the *Cosimulation* subphase, the hardware and software solutions generated in the previous two subphases are now inter-related by scheduling the software on the hardware using:

$$\pi(\mathrm{SM}_i, \mathrm{HM}_j, \mathrm{TS}_k) = \frac{1}{\mathrm{T}(\mathrm{SM}_i, \mathrm{HM}_j, \mathrm{TS}_k) \times \mathrm{C}(\mathrm{HM}_j)} \tag{3}$$

where $\mathrm{T}(\mathrm{SM}_i, \mathrm{HM}_j, \mathrm{TS}_k)$ is the execution time of SM_i on HM_j scheduled using TS_k and $\mathrm{C}(\mathrm{HM}_j)$ is the total hardware cost of the system.

For the running example, the results of hardware and software syntheses are given in Fig. 2.

5 Conclusion and Future Work

A methodology called *Cosynthesis Methodology for Application-Oriented Parallel Systems* (CMAPS) was presented for synthesizing both the software and hardware of AOGPP systems. CMAPS uses an iterative procedure beginning with a solution graph and going through interleaved phases of software and hardware modeling. The software-hardware model combinations are coevaluated in order to decrease the size of the design space to be explored. Hardware and software are then synthesized separately and cosimulated by scheduling the synthesized software on the hardware using multiprocessor task scheduling algorithms. These three repositories: PB, AB, and MB, constructed using OO technology [12], also contribute toward easily upgrading to new technologies such that new hardware components, new algorithms, and new elementary problems can always be integrated into existing repositories.

Future work would be applying OO technology not only to the repositories, but to the codesign process itself [22]. Hardware and software dependence on each other also need further investigation. A formal verification model [11] for the codesign of AOGPP systems would also be an interesting research topic.

References

[1] S. G. Akl. *The Design and Analysis of Parallel Algorithms*. Prentice-Hall International, Englewood Cliffs, New Jersey, 1989.

[2] J.-M. Berge, O. Levia, and J. Rouillard, editors. *Hardware/Software Co-Design and Co-Verfiication*. Kluwer Academic Publishers, the Netherlands, 1997.

[3] W. P. Birmingham, A. P. Gupta, and D. P. Siewiorek. The MICON system for computer design. In *Proc. 26th ACM/IEEE Design Automation Conference*, pages 135–140, 1989.

[4] W. W. Chu and L. Tan. Task allocation and precedence relations for distributed real-time systems. *IEEE Trans. on Computers*, C-36(6):667–679, June 1987.

[5] R. Ernst, J. Henkel, and T. Berner. Hardware-software co-synthesis for microcontrollers. *IEEE Design and Test of Computers*, 10(4):64–75, December 1993.

[6] A. J. Gadient and D. E. Thomas. A dynamic approach to controlling high-level synthesis CAD tools. *IEEE Trans. on VLSI Systems*, 1(3):328–341, September 1993.

[7] A. P. Gupta, W. P. Birmingham, and D. P. Siewiorek. Automating the design of computer systems. *IEEE Trans. on CAD of IC*, 12(4):473–487, April 1993.

[8] R. K. Gupta and G. De Micheli. Hardware-software cosynthesis for digital systems. *IEEE Design and Test of Computers*, 10(3):29–41, September 1993.

[9] P.-A. Hsiung, C.-H. Chen, T.-Y. Lee, and S.-J. Chen. ICOS: An intelligent concurrent object-oriented synthesis methodology for multiprocessor systems. to appear in *ACM Trans. on Design Automation of Electronic Systems*, 3(2), April 1998.

[10] P.-A. Hsiung, S.-J. Chen, T.-C. Hu, and S.-C. Wang. PSM: An object-oriented synthesis approach to multiprocessor system design. *IEEE Trans. on VLSI Systems*, 4(1):83–97, March 1996.

[11] P.-A. Hsiung, T.-Y. Lee, and S.-J. Chen. MOBnet: An extended Petri net model for the concurrent object-oriented system-level synthesis of multiprocessor systems. *IEICE Trans. on Information and Systems*, E80-D(2):232–242, February 1997.

[12] P.-A. Hsiung, T.-Y. Lee, and S.-J. Chen. Object-oriented technology transfer to multiprocessor system-level synthesis. In *Proc. 24th International Conference on Technology of Object-Oriented Languages and Systems*, September 1997.

[13] A. Kalavade and E. A. Lee. A hardware-software codesign methodology for DSP applications. *IEEE Design and Test of Computers*, 10(3):16–28, September 1993.

[14] S. Kumar, J. H. Aylor, B. W. Johnson, and W. A. Wulf. A framework for hardware/software codesign. *IEEE Computer*, pages 39–45, December 1993.

[15] J.-F. Lin and S.-J. Chen. An analysis of multiprocessor tasks scheduling. *Computer Systems Science and Engineering*, 11(2):117–120, March 1996.

[16] S. Prakash and A. C. Parker. SOS: Synthesis of application-specific heterogeneous multiprocessor systems. *Journal of Parallel and Distributed Computing*, 16:338–351, 1992.

[17] J. Rozenblit and K. Buchenrieder, editors. *Codesign: Computer-Aided Software/Hardware Engineering*. IEEE Press, New York, USA, 1995.

[18] D. E. Thomas, J. K. Adams, and H. Schmit. A model and methodology for hardware-software codesign. *IEEE Design and Test of Computers*, pages 6–15, September 1993.

[19] J. D. Ullman. NP-complete scheduling problem. *Journal of Computer and System Sciences*, 10:384–393, 1975.

[20] F. Vahid, J. Gong, and D. D. Gajski. A binary-constraint search algorithm for minimizing hardware during hardware/software partitioning. In *European Design Automation Conference*, pages 214–219, 1994.

[21] W. Wolf. Hardware-software co-design of embedded systems. *Proceedings IEEE*, 82(7):967–989, July 1994.

[22] W. Wolf. Object-oriented cosynthesis of distributed embedded systems. *ACM Trans. on Design Automation of Electronic Systems*, 1(3):301–314, July 1996.

[23] W. Wolf. An architectural co-synthesis for distributed, embedded computing systems. *IEEE Trans. on VLSI Systems*, 5(2):218–229, June 1997.

[24] T.-Y. Yen and W. Wolf. Communication synthesis for distributed embedded systems. In *ICCAD*, pages 288–294, 1995.