Evaluating Cost-Performance Tradeoffs for System Level Applications†

Wei-Liang Ing[1], Cheng-Tsung Hwang[2], and Allen C.-H. Wu[1]

[1]Department of Computer Science, Tsing Hua University
Hsinchu, Taiwan, 300, Republic of China
[2]Department of Computer Science and Information Management,
Providence University, ShaLu, Taiwan, Republic of China

Abstract

Evaluation of design cost and performance is indispensable to system partitioning. In the absence of a system-level estimation and analysis tool, system partitioning is difficult to perform in an efficient and accurate manner because design evaluation can only be done after the final results are achieved. Furthermore, without cost-performance tradeoff information relating to different design alternatives, the designer can not make intelligent design decisions at the early system-level partitioning stages. In this paper, we present a system-level cost/performance evaluation approach which systematically explores the AT (Area-Time) design-space from a system description. This allows the designer to obtain first-hand design tradeoff information before the partitioning process has taken place. We have also developed a system-level interactive design evaluation system on top of the proposed approach. Experiments on a number of examples demonstrate that our approach provides the designer with a comprehensive system-level design evaluation method to effectively explore all possible design alternatives in the early stages of system development.

1 Introduction

Most embedded systems consist of a general-purpose processor along with specialized hardware to perform application-specific tasks. Converting a system-level specification into a target embedded system is usually referred to as a hardware-software codesign problem. A system-level specification can be decomposed into a set of interconnected modules. Each module can then be implemented in a specialized hardware or in software running on a processor. In general, a hardware implementation has better performance whereas a software implementation has lower cost. Thus, the main objective in converting a system-level specification into a target embedded-system is to implement the specification that has the minimum overall cost, including hardware and software, while satisfying the required performance constraints.

†Supported by the National Science Council of R.O.C. under contracts No. NSC-83-0404-E-007-020 and NSC 85-2221-E-007-034

ASP-DAC '97
0-89791-851-7/$5.00 ©1997 IEEE

In the past several years, hardware-software codesign related problems have been extensively studied in the CAD community. A number of frameworks have been proposed for design modeling, simulation, synthesis, and integration [1, 2, 3]. In addition, many partitioning techniques have been proposed [2, 4, 5, 6] to tackle hardware-software codesign problems. As indicated in [2, 4], design evaluation of cost and performance is indispensable in system partitioning. Many estimation techniques have been proposed for area-performance prediction in physical design [7, 8, 9] and high-level synthesis [10, 11, 12]. However, little work has been done at the system-level and efforts that have addressed this area focus mainly on software performance estimation and evaluation [13, 14, 15]. In the absence of a system-level estimation and analysis tool, system partitioning is difficult to perform in an efficient and accurate manner because design evaluation can only be done after the final results are achieved. Furthermore, without cost-performance tradeoff information relating to different design alternatives, the designer can not make intelligent design decisions at the crucial, early system-level partitioning stages. Consequently, a comprehensive system-level estimation and evaluation method is needed to assist the designer in performing system partitioning.

In this paper, we present a system-level cost-performance evaluation approach. We propose a composite evaluation method that systematically explores the AT design-space of a given design. Based on our proposed evaluation approach, we have developed S-LIDE: A System-Level Interactive Design Evaluator. Using this tool, the designer is able to effectively explore all possible design alternatives at the early stages of system development.

The remainder of the paper is organized as follows. Section 2 gives the overview of our proposed approach. Section 3 presents the system-level AT design-space exploration method. Section 4 presents the experimental results. Finally, section 5 gives concluding remarks.

2 Overview of the proposed approach

We propose a composite evaluation method that systematically explores the AT design-space of a given design targeted to an embedded system. Figure 1(a)

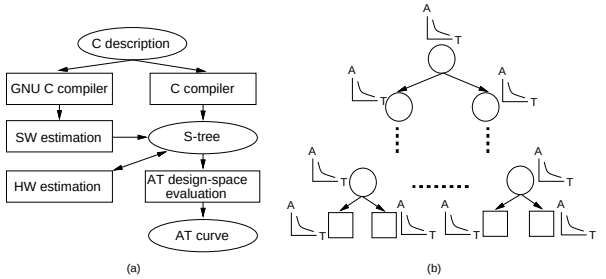


Figure 1: The proposed: (a) cost-performance evaluation approach, (b) a composite approach to AT design-space evaluation.

shows our proposed approach which consists of three parts: software estimation, hardware estimation, and AT design-space evaluation. The input to the evaluator is a system-level description in C. The input description is parsed into an S-tree (structural tree) which represents the structure of the C description in a hierarchical fashion. We use a constructive method to generate an AT-curve for each basic block including a software implementation and a set of hardware implementations with different set of resources. Then we use an analytical method to explore the system AT design-space by adding-up the AT-curves of leaf nodes throughout the hierarchy all the way to the root node, as shown in Figure 1(b).

In the section, we first describe the target system architecture. Then we present the S-tree. Finally, we give an overview of our proposed AT design-space evaluation approach.

2.1 Target architecture

We use a typical embedded system as our target architecture which consists of a re-programmable processor, a memory module, a hardware module (containing one or a number of ASICs), and an I/O interface. The core processor is a general purpose processor, such as 8086 and 68000. The memory module includes the storages used for program and data. For simplicity, we assume that all memory accesses are to a single-level memory. The hardware module is connected to the system address and data buses. Data communications between the processor, the hardware module, and the external world take place via a shared memory. In this paper, we make a simplifying assumption that the target architecture has only one re-programmable processor and a single-port memory module. Under this assumption, only one software task can be executed on the processor at any time. Furthermore, only one module, either a software or a hardware module, is allowed to access data from the memory at any time. We also assume that all functionalities in the hardware module is implemented with a single ASIC chip. Multiple-chip partitioning is not considered in this paper.

2.2 S-tree Construction

An S-tree (Structural tree) represents the structure of a C program in a hierarchical fashion. It consists of two types of nodes: leaf nodes and internal nodes. A leaf node contains a basic block which is defined as a sequence of consecutive statements without any halting or possibility of branching statements. On the

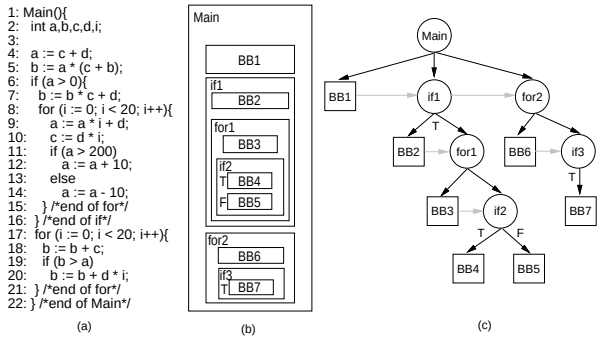


Figure 2: An S-tree example: (a) a C code example, (b) its corresponding hierarchical structure, (c) its S-tree.

other hand, an internal node contains a control statement such as conditional statements (e.g., if, case, for, while) and procedure calls. Each internal node also contains a basic block describing the conditional expression which may be a simple comparison expression or a complex computational expression (e.g., $if\ (100 \leq a + b \div c \times d)\ then$). An internal node represents a hierarchical block which may contain a nested conditional statement and/or a set of basic blocks. In the S-tree, there are two types of edges. The first one represents the hierarchical relationship between two connecting nodes. The other represents the dependency relationship between nodes. Figures 2(a), (b), and (c) show a C code example, its corresponding hierarchical structure, and the corresponding S-tree, respectively.

2.3 A composite AT evaluation method

There are two commonly used estimation methods: analytical and constructive methods. Analytical methods perform design prediction based on abstract analytical models. It usually takes less time to run but suffers from less accurate estimates. By contrast, constructive methods usually produce more accurate estimates but more time consuming compared to that of analytical methods. To make a good compromise of these two methods, Kurdahi and Ramachandran [8] proposed a composite approach mixing constructive and analytical methods to produce accurate estimates in a reasonable amount of time. In our approach, we use a composite method similar to the one proposed by Kurdahi and Ramachandran to perform AT design-space evaluation.

Figure 1(b) shows the composite methodology for AT design-space evaluation. Our method first parses a C description into an S-tree. Since each leaf node and/or internal node in the S-tree contains a basic block, we can transform the C statements in the basic block into a DFG. Subsequently, our method uses a constructive method to generate an AT-curve for each basic block. Each design point on the AT-curve represents a cost-performance pair for a particular implementation of the design, such as a software implementation or a hardware implementation with a specific set of resources. After generating AT-curves for all basic blocks, AT-curves of leaf nodes are then added-up using an analytical method throughout the hierarchy all the way to the root node.

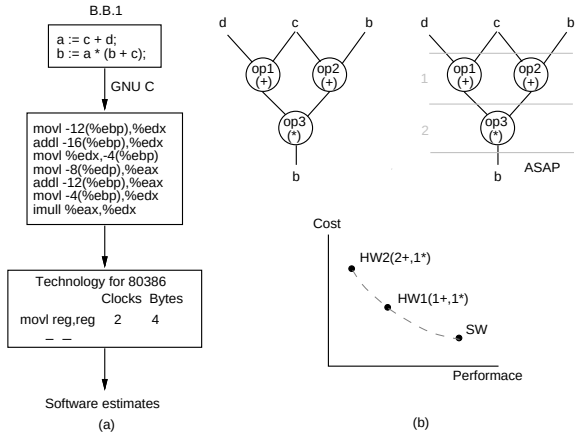


Figure 3: AT-curve generation: (1) software estimation, (2) hardware estimation.

3 Design-space exploration

3.1 AT-curve analysis for basic blocks

In this section, we first describe how to obtain the software estimate of a basic block and then the hardware estimates with a varying set of resources.

In order to obtain the software estimates for basic blocks, we need to compile the code in the basic block into the instruction set of the target processor. For example, if we choose the Intel 80386 as our target processor, it needs to compile the code into the 80386 instruction set. In our implementation, we use the GNU C compiler to translate the code into assembly codes of the target processor. For example, Figure 3(a) depicts the C code of basic-block $BB1$ (Figure 2) and the 80386 assembly code generated by the GNU C. Using the timing and size information of each type of instruction provided by the databook, the estimator can obtain the software metrics, such as execution time and program size, for each basic block.

To obtain an AT-curve of a design needs to explore the entire design space by trading-off hardware cost and performance. We use an iterative approach to perform hardware estimation for each basic block. Each design point (i.e., a cost-performance pair) on the AT-curve is obtained by performing scheduling on the DFG derived from the code in a basic block with a specific set of resources. Our approach first determines the minimum and maximum bounds on execution units. Given a DFG, the minimum bound on execution units is defined as the minimum resources required to execute the DFG. Let $OP = \{op_i \mid i = 1..n\}$ be a set of distinct type of operators in the DFG. For example, in Figure 3(b), $Min(+) = 1$ and $Min(*) = 1$. The maximum bound on execution units of the DFG which is defined as the resources required to achieve the maximum performance (i.e., the maximally-parallel execution) of the design. The maximally-parallel execution of the design can be achieved by performing the as-soon-as-possible (ASAP) scheduling on the DFG by taking into account only the data dependencies but ignoring the resource constraints. Let t be the shortest execution time-step of the DFG and $par_j(op_i)$ be the number of type-i operations executed by type-i execution unit at time-step j. The maximum bound on execution units of the DFG is defined as $Max(op_i) = MAX_{j=1}^{j=t}(par_j(op_i))$ for all $op_i \in OP$. For example, in Figure 3(b), the maximum bounds on adders and multiplier are two and one ($Max(+) = 2$ and $Max(*) = 1$), respectively.

After determining the minimum and maximum bounds on execution units, we use an iterative approach to search the design space. The minimum bound is then served as the initial resource constraint for the design-space search process. By incrementally relaxing the resource constraints (i.e., adding more execution units), our approach produces a set of cost-performance pairs indicating the design-space of the design. Assume that there are n distinct types of operators in the DFG of a basic block. The pseudo code for the AT-curve generation of a basic block is shown as below.

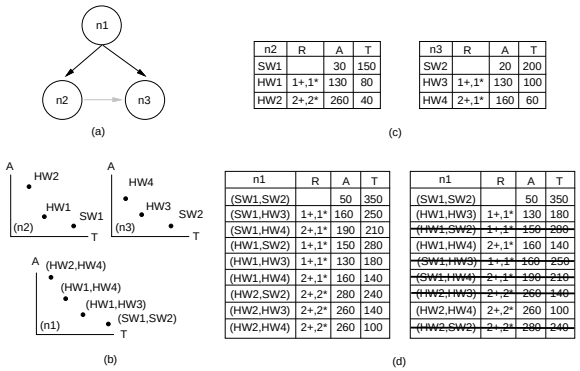
```
Procedure AT_CURVE_GENERATION(DFG)
begin
  {Min(op_i) | i = 1..n} = Min_Bound(DFG);
  {Max(op_i) | i = 1..n} = Max_Bound(DFG);
  AT = (Cost_SW, Delay_SW);
  for (Min(op_1) to Max(op_1))
  begin
    .....
    for (Min(op_n) to Max(op_n))
    begin
      (Cost_R, Delay_R) = LIST_SCHE(DFG,R);
      AT = AT ∪ (Cost_R, Delay_R);
    end_of_for
  end_of_for
  AT = PRUNE(AT);
end_of_Procedure
```

$Min\_Bound(DFG)$ and $Max\_Bound(DFG)$ are two procedures determining the minimum and maximum bounds on execution units, AT is a set of cost-performance pairs, R is the resource constraint. A design point (i.e., a cost-performance pair) is obtained by performing the list scheduling algorithm ($LIST\_SCHE$) on the DFG under the given resource constraint R. We use the minimum bound as the initial resource constraint and then incrementally add more execution units. It takes $\prod_{i=1}^{i=n} Max(op_i)$ iterations to explore the entire design-space of a basic block. For example, in Figure 3(b), $Min(+)$, $Min(*)$, $Max(+)$, and $Max(*)$ are 1, 1, 2, and 1, respectively. Two iterations are needed to generate the AT-curve. When the search of the entire design space is completed, the procedure $PRUNE$ is invoked to prune the inferior design points.

3.2 System-level AT-curve analysis

After generating AT-curves for all basic blocks, we use an analytical method to merge the AT-curves recursively throughout the hierarchy of the S-tree all the way to the root node. We use an approach similar to the one reported in [9] to obtain the AT-curve shape function.

There are two main factors which will directly affect the AT-curve analysis. The first one is the hardware sharing effect in the hardware module. As indicated



n2	R	A	T
SW1		30	150
HW1	1+,1*	130	80
HW2	2+,2*	260	40

n3	R	A	T
SW2		20	200
HW3	1+,1*	130	100
HW4	2+,1*	160	60

n1	R	A	T
(SW1,SW2)		50	350
(SW1,HW3)	1+,1*	160	250
(SW1,HW4)	2+,1*	190	210
(HW1,SW2)	1+,1*	150	280
(HW1,HW3)	1+,1*	130	180
(HW1,HW4)	2+,1*	160	140
(HW2,SW2)	2+,2*	280	240
(HW2,HW3)	2+,2*	260	140
(HW2,HW4)	2+,2*	260	100

n1	R	A	T
(SW1,SW2)		50	350
(HW1,HW3)	1+,1*	130	180
~~(HW1,SW2)~~	~~1+,1*~~	~~150~~	~~280~~
~~(SW1,HW3)~~	~~1+,1*~~	~~160~~	~~250~~
~~(SW1,HW4)~~	~~2+,1*~~	~~190~~	~~210~~
~~(HW2,HW3)~~	~~2+,2*~~	~~260~~	~~140~~
(HW2,HW4)	2+,2*	260	100
~~(HW2,SW2)~~	~~2+,2*~~	~~280~~	~~240~~

Figure 4: AT-curve merging method: (1) merging two consecutive nodes, (2) AT-curves, (c) tabulated design-space of nodes $n2$ and $n3$, (d) tabulated design-space of node $n1$.

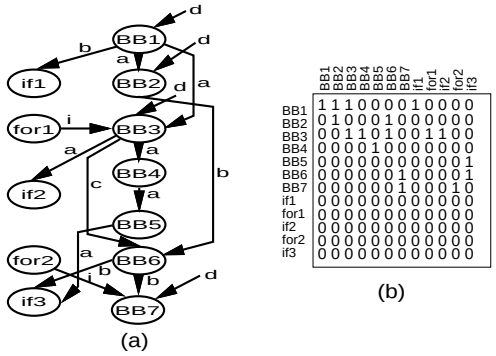


Figure 5: Communication analysis: (a) communication graph, (b) communication matrix.

earlier, we assume that all functionalities in the hardware module is implemented with a single ASIC chip. Hence, in order to obtain more realistic cost analysis, the hardware sharing between functionalities in the hardware module has to be considered. The second one is the communication delays. In order to obtain more realistic delay analysis, the communication overhead between hardware and software components has to be taken into account.

In this section, we first present the AT-curve generation method considering hardware sharing and then describe how to estimate communication delays.

3.2.1 AT-curve generation

There are three basic AT-curve merging operations: (1) merging a sequence of nodes, (2) merging a conditional node, and (3) merging a loop node. Due to the page limitation, in this paper we only show the first merging operation. The detailed description of the other two merging operations can be found in [16].

We use the example illustrated in Figure 4 to demonstrate how to merge two consecutive nodes $n2$ and $n3$ into $n1$. Each node has three design points, $SW1$, $HW1$, and $HW2$ for node $n2$ and $SW2$, $HW3$, and $HW4$ for node $n3$, as shown in Figures 4(b) and (c). Figure 4(d) shows the nine possible design points by merging the AT-curves of $n2$ and $n3$.

Each merged design point represents a new cost-performance pair of an implementation combination applied to $n2$ and $n3$. We first describe the performance estimation and then the cost estimation of a merged node. Since these two nodes are executed in sequential order, the delay of merged node $n1$ is the sum of the delays of nodes $n2$ and $n3$. For example, in Figure 4(d), when nodes $n2$ and $n3$ are executed by implementations of $SW1$ (delay=150) and $SW2$ (delay=200), respectively, the delay by merging $n2$ and $n3$ is 350. We now describe the cost estimation of a merged node. Since these two nodes are executed in sequential order, the hardware used in $n2$ can be shared by $n3$. Hence, the cost estimation of a merged node needs to take into account the hardware-sharing effect. For example, in Figure 4(d), when nodes $n2$ and $n3$ are executed by implementations of $HW1$ (R=$\{1+,1*\}$) and $HW4$ (R=$\{2+,1*\}$), it needs only two adders and one multiplier to implement the merged design, i.e., A(HW1,HW4)=160. (Note that the interconnect cost of a merged design may be increased. However, for simplicity we only consider the cost of execution units.) After generating all nine possible design points, we apply a pruning procedure to eliminate the inferior design points. For example, in Figure 4(d), the cost-performance pair $(A = 150, T = 200)$ of implementation-combination $(HW1, SW2)$ is more expensive and slower than that of implementation-combination $(HW1, HW3)$ $((A = 130, T = 180))$. Hence, implementation-combination $(HW1, SW2)$ is an inferior design which can be pruned. As a result, the final AT design-space of merged node $n1$ contains only four design points, as shown in Figures 4(b) and (d).

3.2.2 Communication delay estimation

Based on the target architecture, there are three types of data communications: (1) data transfers between the processor and the memory, (2) data transfers between the ASIC and the memory, and (3) data transfers between the processor and the ASIC. We assume that the data transfers between the functionalities in the software module are through the internal registers in the processor. Similarly, the data transfers between the functionalities in the hardware module are through the internal registers in the ASIC. Therefore, there is no communication overhead for these types of data transfers. Furthermore, we assume that all the global variables declared in the C description are stored in the memory. A data transfer is required between the memory and the processor or the memory and the ASIC when the functionalities in the software module or the hardware module need to access those global variables.

We use a communication graph to express data-transfer relationships between basic blocks. The communication graph can be easily retrieved from the symbol table during the parsing stage. Figure 5(a) depicts the communication graph of the C description shown in Figure 2. Each node in the communication graph represents a basic block or an internal node. A directed-edge connected two nodes represents that data is produced

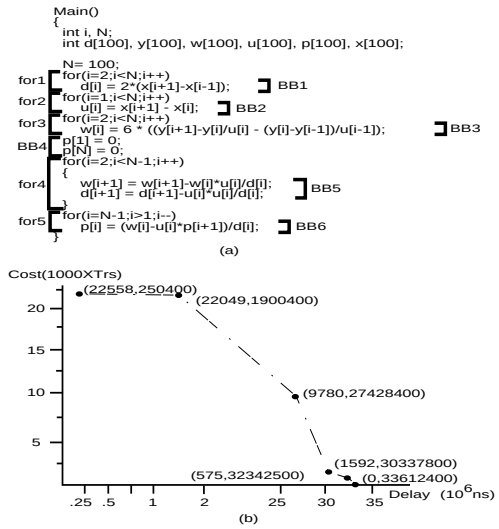


Figure 6: Example 1: (a) the C code, (b) the AT curve.

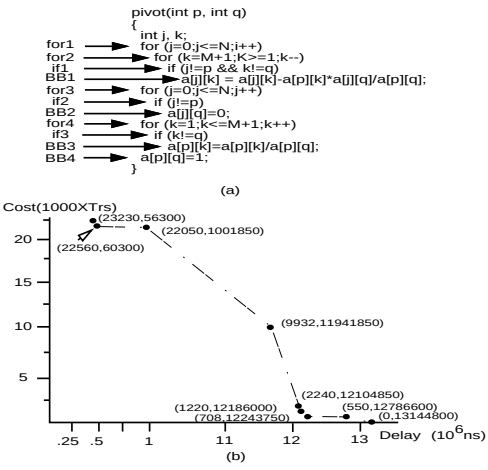


Figure 7: Example 2: (a) the C code, (b) the AT curve.

in one node and consumed by the other. Each edge is associated with a weight indicating the size of data transferred between two nodes. For instance, in Figure 5(a), variable a is produced by $BB1$ and consumed by $BB2$. In addition, variable d is stored in the memory and consumed by $BB1$. After generating a communication graph, we then construct a communication matrix to represent data sizes communicating between nodes. For example, Figure 5(b) shows the communication matrix derived from the communication graph shown in Figure 5(a). Note that the number on the diagonal line represents the number of global variables consumed by each node (e.g., [$BB1$,$BB1$]=1 because one global variable d is consumed by $BB1$).

During the AT-curve merging stage, the communication overhead can be estimated through a table look-up procedure. For example, in Figure 4, when merging two nodes $n2$ and $n3$ communication overhead occurs in two cases. First, $n2$ or/and $n3$ need to access global variables. Second, there exists data transfers between these two nodes, while one of the nodes is implemented in the hardware module and the other in the software module (e.g., $n2$ with $HW2$ and $n3$ with $SW2$). The communication data sizes can be directly obtained from the communication matrix. The estimated communication overhead is then added into cost-performance pair for AT-curve generation.

4 Experiments

Based on the proposed approach, we have developed the SLIDE: A System-Level Interactive Design Evaluator. It provides the user an interactive design evaluation environment. With SLIDE, the user can interactively evaluate any segment of the system description via the GUI.

The SLIDE is implemented in C with Motif graphics library and X-Window system running on SUN or/and HP workstations. We performed two sets of experiments. We first tested the feasibility of our proposed approach on two C examples. Then, we applied our approach to a medical measuring system as a case study. In all experiments, our software estimation was based on an 8086 microprocessor with a 5MHz clock (i.e., 200ns per cycle). We also use a simplified hardware component library including an adder, subtractor, multiplier, divider, and comparator with costs of 512, 512, 8192, 12800, and 512 transistors, respectively. (We assume the bit width of all components is 16). We also assume that the clock rate for the hardware implementation is 20MHz (i.e., 50ns per cycle). In addition, we assume that memory access time takes three clock cycles.

The first example is the C code for computing a cubic spline [17]. This example contains five *for* loops, as shown in Figure 6(a). In our design evaluation, we assume that all of the arrays (e.g., d[100], y[100],...) are stored in the memory module. The result of the software execution-time profile shows that loops $for1$, $for2$, $for3$, $for4$, and $for5$ account for 9.9%, 4.9%, 29%, 37.3%, and 18.9%, respectively, of the total execution time. In addition, there are 6 design alternatives ranged from (cost=22558 transistors,delay=250400 ns) with all hardware implementations to (cost=0 transistors,delay=33612400 ns) with all software implementations, as shown in Figure 6(b).

The second example is the C code of the requisite pivoting procedure [17], as shown in Figure 7(a). We assume that the coefficients M and N are 10 and 6, respectively. The result of the software execution-time profile shows that the nested $for1$ and $for2$ loop accounts for 90% of the total execution time. Finally, Figure 7(b) shows the 9 possible design alternatives ranged from (cost=23230 transistors,delay=56300 ns) with all hardware implementations to (cost=0 transistors,delay=13144800 ns) with all software implementations.

The third example is a medical system for measuring bladder volume. Figures 8(a) and (b) show the system block diagram and the system graph, respectively. In the initialization step, the system loads the number of scanning points. For each scan, the motor-controller first sends out signals to move the transducer

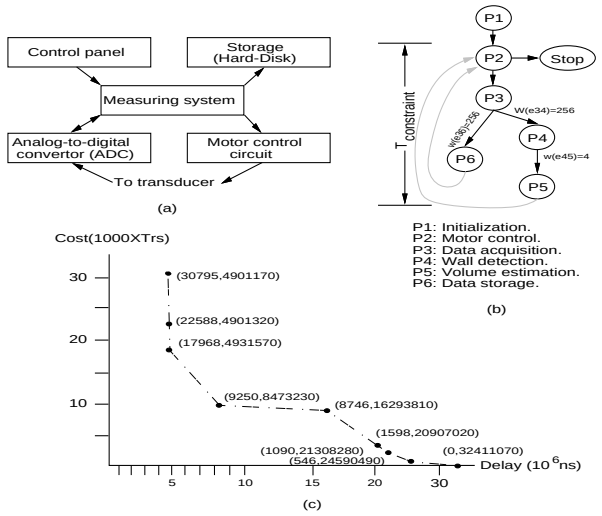


Figure 8: The medical measuring system: (a) the system block diagram, (b) the system graph, (c) the AT curve.

to the designated scanning position. The data acquisition module then converts the ultrasonic echo into digital data and sends the data back to the system. For simplicity, in this paper we assume that each echo takes 256 bytes of data . The wall detector and volume estimator then compute the diameter and the volume of the bladder. Finally, the echo data is stored into the hard-disk for later analysis.

In this experiment, we assume that the ultrasonic echo data (256 bytes for scan) is stored in the memory module. Figure 8(c) shows the AT curve for each scanning iteration. There are nine possible design alternatives ranged from (cost=30795 transistors,delay=4901170 ns) with all hardware implementations to (cost=0 transistor,delay=32411070 ns) with all software implementations. One interesting observation of this example is that the *wall detection* section accounts for 70% of the total execution time in which the main contributor is the data communication delay (up to 50%). From the analysis, we can conclude that the obvious performance bottleneck of the system caused by data-transfers between software or/and hardware modules and the memory.

5 Conclusions

In this paper, we have presented a system-level cost-performance evaluation approach to systematically explore the AT design space of a given design. The approach allows the designer to obtain first-hand design tradeoff information before the partitioning process has taken place. We have developed the system-level interactive design evaluator SLIDE on top of the proposed approach. Using the tool, the designer can interactively perform design evaluation on any segment of the system description. Furthermore, the wide variety of graphical design views provided fully support user analysis in every aspect of design evaluation. Our proposed approach and evaluator provide the designer with a comprehensive system-level design estimation and evaluation method to effectively explore all possible design alternatives in the early stages of system development.

Future work include enhancing the evaluator by adding more accurate estimation models and incorporating a partitioning approach for hardware-software partitioning.

References

[1] R. Ernst, J. Henkel, and T. Benner, "Hardware-Software Cosynthesis for Microcontroller," *IEEE Design & Test of Computers*, pp. 64-75, Dec., 1993.

[2] R. K. Gupta and G. D. Micheli, "System-Level Synthesis Using Re-programmable Components," *in Proc. of EDAC'92*, pp. 2-7, 1992.

[3] M. B. Srivastava and R. W. Brodersen, "Rapid-Prototyping of Hardware and Software in A Unified Framework," *in Proc. of the ICCAD'91*, 1991.

[4] R. Ernst and J. Henkel, "Hardware-Software Codesign of Embedded Controllers Based on Hardware Extraction," *in Proc. of the Internation Workshop on Hardware/Software Codesign*, Sept., 1992.

[5] F. Vahid, D. D. Gajski, and J. Gong, "A Hardware-Software Partitioning Algorithm for Minimizing Hardware," *in Proc. of the Euro-DAC'94*, 1994.

[6] E. Barros, W. Rosentiel, and X. Xiong, "A Method for Partitioning Unity Language in Hardware and Software," *in Proc. of EuroDAC'94*, pp. 220-225, 1994.

[7] F. J. Kurdahi and A. C Parker, "Technique for Area Estimation of VLSI Layouts," *IEEE Trans. on CAD*, vol. 9, no. 9, pp. 938-950, 1990.

[8] F. J. Kurdahi and C. Ramachandran, "Evaluating Layout Area Tradeoffs for High Level Applications," *IEEE Trans. on VLSI Systems,* vol. 1, no. 1, pp. 46-55, March, 1993.

[9] G. Zimmermman, "A New Area and Shape Function Estimation Technique for VLSI Layouts," *in Proc. 25th DAC*, pp. 60-68, 1988.

[10] R. Jain, A. C. Parker, and N. Park, "Predicting System-Level Area and Delay for Pipelined and Non-pipelined Designs," *IEEE Trans. on CAD*, vol. 11, August, 1992.

[11] D. S. Rao and F. J. Kurdahi, "Hierarchical Design Space Exploration for a Class of Digital Systems," *IEEE Trans. on VLSI Systems*, vol. 1, no. 3, pp. 282-294, September, 1993.

[12] J. M. Rabaey and M. Potkonjak, "Estimating Implementation Bounds for Real Time DSP Application Specific Circuits," *IEEE Trans. on CAD*, vol. 13, no. 6, pp. 669-683, June, 1994.

[13] J. Gong, D. D. Gajski, and S. Narayan, "Software Estimation from Executable Specification," *Journal of Computer and Software Engineering*, 1994.

[14] W. Wolf and J. Martinez, "C Program Performance Estimation for Embedded Systems Architecture Sizing," *in Proc. of the Internation Workshop on Hardware/Software Codesign*, 1993.

[15] W. Ye, R. Ernst, T. Benner, and J. Henkel, "Fast Timing Analysis for Hardware-Software Co-Synthesis," *in Proc. of the ICCD'93*, pp. 452-457.

[16] W.-L. Ing, "Evaluating Cost-Performance Tradeoffs for System Level Applications," Master Thesis, CS Dept., Tsing Hua University, 1995.

[17] R. Sedgewick, *Algorithms in C*, Addison-Wesley Publishing Company, Inc., pp. 549 and pp. 617, 1990.