Architectural Exploration for Datapaths with Memory Hierarchy

Nancy D. Holmes, Daniel D. Gajski
Dept. of Information & Computer Science
University of California
Irvine, California 92717

Abstract

In this paper, we present a new design-space exploration algorithm, the architecture explorer (AE), for analyzing performance/cost tradeoffs in memory-intensive applications. AE evaluates FU, bus, and memory *cost for a series of performance constraints to produce a performance/cost tradeoff curve. Unlike previous approaches, AE handles both* **hierarchical** *and non-hierarchical memory architectures with various speeds of memory.*

1 Introduction

High-level synthesis (HLS), the process of automatically producing a register-transfer (RT) level design from a behavioral description, is currently an important area of research in design automation. Many HLS methodologies have been proposed, most of which advocate a three-step approach to the synthesis problem [5]. The first step is *allocation*, in which RT-level components such as functional units (FUs), buses, and memories are selected to implement the design. Next, *scheduling* assigns operations to control steps, and finally, *binding* maps operations to specific RT-level component instances. The output of allocation, scheduling, and binding is an RT-level netlist for the datapath and a control unit specification.

The HLS process can be complicated and time-consuming due to the conflicting goals of synthesis (minimum area, minimum execution time, minimum power, etc ...); therefore, designers using HLS may explore few alternatives before settling on a final design. To alleviate this problem, researchers developed *design-space exploration (DSE)* tools to quickly suggest many different alternatives to the designer and help him/her select an initial design which is "close" to satisfying the requirements. This initial design can then be modified slightly or refined, either manually or automatically, to generate the final design.

In general, DSE tools view the design space as a plane with delay on the x-axis and area on the y-axis [8, 10, 14, 15]. A design is defined as a point in the plane with a delay value and an area value. Tools explore the design space by performing a series of area (allocation) estimations for a fixed delay and/or a series of delay estimations for a fixed area (allocation). The important goal in these approaches

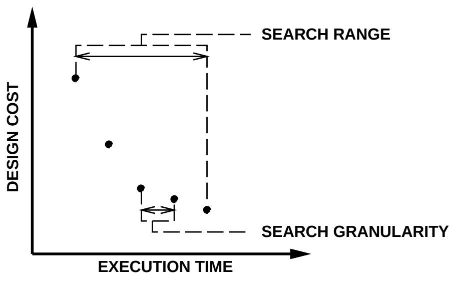

Figure 1: Design Space Search.

is to estimate area by optimizing the number of FUs, registers, and interconnect units. Unfortunately, these tools do not take into account the required *memory* or *memory hierarchy* frequently employed by designers to reduce cost when large, high-speed memories are needed.

We present a new DSE algorithm, **Architecture Explorer (AE)**, for memory-intensive descriptions. Unlike the previous approaches, AE estimates *memory* cost for both *hierarchical and non-hierarchical memory architectures* in addition to FU and bus cost. So, AE can be used to determine the "best" memory hierarchy for the design. AE's main contributions are as follows.

1. **Memory Hierarchy -** AE handles memory hierarchy where direct communication is permitted only between adjacent levels of memory.
2. **Different Speeds of Memory -** In AE, different memories may have different access times and these access times may take any number (one or more) of clock cycles.
3. **Simultaneous Estimation of All Resources -** This allows the designer to see the tradeoffs between FUs, memories, and buses so he/she can determine which one dominates the design cost.
4. **User-Controllable Design-Space Search -** In AE, the designer controls the range and granularity

(and therefore the speed) of the design-space search. (See Figure 1.)

2 Previous Work

Many design-space exploration techniques have been proposed in the literature [8, 10, 14, 15]. These approaches estimate area by optimizing the number of FUs, registers, and interconnect units. Although they have been very successful for small problems, they cannot be applied directly to memory-intensive descriptions, which are characterized by large array variables and complex control flow, since it is inefficient to map all array elements into registers.

Several memory synthesis tools have also been proposed; however, they are not directly applicable to memory-intensive applications [1, 2, 7, 11, 13, 16]. For instance, the techniques in [1, 2, 11] are designed to reduce wiring area by using register files or n-port memories instead of distributed registers; however, only *scalar* variables are permitted in the input descriptions, and memory hierarchy is not allowed. In [13], array variables are permitted, but the memory model is still not hierarchical. The algorithms from [16] synthesize Silage descriptions, where each variable denotes an infinite stream of data. The goal is to optimize the size of storage elements according to the data dependencies in the description. In [7] memory-intensive applications are scheduled onto a fixed target architecture with a datapath, an off-chip memory, and an I/O buffer. The goal in this work is to minimize the size of the I/O buffer.

3 Architectural Model

Figure 2 depicts AE's architectural model which consists of a datapath of functional units, buses, and memories arranged hierarchically into $L+1$ levels. Level 0 contains the FUs, while each level $l, 1 \leq l \leq L$, contains a set of buses and a memory. As the level number increases, the speed of the memory decreases. So, if $l_1 < l_2$, then the memory at level l_1 is faster than the memory at level l_2.

Communication is permitted only between adjacent levels. For instance, data cannot be transferred directly from the level 3 memory to the level 1 memory. Instead, we must move the data from level 3 to level 2 and then from level 2 to level 1.

Based on this model, an *architecture allocation* consists of the number L of memory levels, and (1) the number, type, bitwidth, delay, and number of stages (if pipelined) of FUs, (2) the number, bitwidth, and delay of buses, and (3) the number of words, bitwidth, number of ports, and delay of memory for each level $l, 0 \leq l \leq L$. Note that the type of an FU refers to the functions it can perform. For example, and ALU which performs ADD and SUB operations is one type of FU while an adder which performs ADD operations is another.

The cost of an architecture allocation C_{aa} is defined as $C_{aa} = C_{fu} + C_{mem} + C_{bus}$ where C_{fu} is the functional unit cost, C_{mem} is the total memory for all the levels and C_{bus} is the total bus cost for all the levels. The functional unit cost is given by $C_{fu} = \sum_{u \in UL} n_u * fu\_cost(u)$ where UL

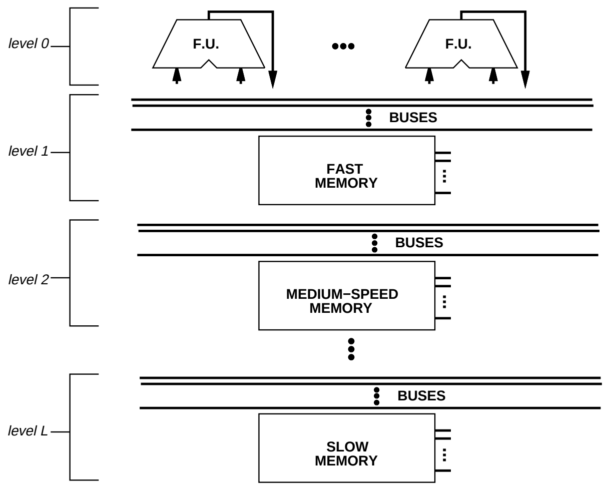

Figure 2: Architectural Model.

is the FU library, n_u is the number of units of type u, and $fu\_cost(u)$ is the cost of unit u. The cost of the memory at each level l given by $C^l_{mem} = p_l * b_l * w_l * mem\_cost(l)$ where p_l is the number of ports on the memory, b_l is the bitwidth of the memory, w_l is the number of words in the memory, and $mem\_cost(l)$ is the cost per cell, specified by the user, for the memory at level l. The bus cost at each level l is given by $C^l_{bus} = n_l * bus\_cost(l)$ where n_l is the number of buses at level l and $bus\_cost(l)$ is the cost, specified by the user, for buses at level l.

4 Algorithm Outline

The goal of AE is to explore the design space from the fastest, most-expensive design to the slowest, minimum-cost design. This is accomplished by performing a series of cost estimations for different delays.

The inputs to AE are as follows.

1. **Behavioral VHDL description -** The description may contain complex control flow such as nested conditionals, case statements, and loops (bounded or unbounded), as well as both array and scalar variables.

2. **Clock period -** The designer must specify the clock period in nanoseconds.

3. **FU library -** AE accepts simple FU libraries where each type of operation can be performed by exactly one FU. Multicycle, pipelined, and multifunctional FUs are allowed. The FU cost and delay must be specified in the library.

4. **Memory configuration -** The memory configuration indicates the number of levels of hierarchy in the design as well as the delay and cost per cell of the memory at each level.

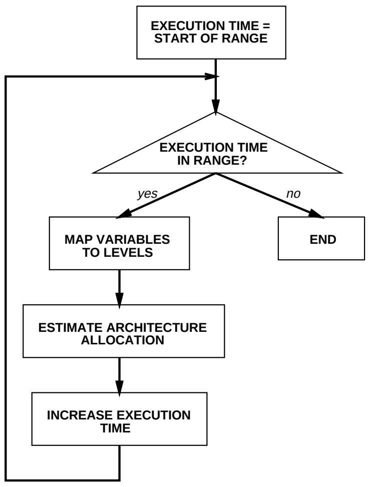

Figure 3: Architecture Explorer Flow Chart.

5. **Bus configuration -** The bus configuration specifies the delay and cost of the buses at each level.
6. **Search parameters -** The designer can control the range and granularity of the design-space search. The *range* of the search gives the minimum and maximum execution times considered, and the *granularity* is the execution time differential as shown in Figure 1.

The output of AE is series of architecture allocations for various execution times.

Figure 3 shows a flow chart of the AE algorithm. The idea is to estimate the minimum cost architecture allocation for each execution time in the range. Due to the hierarchical memory model, estimation requires two steps. The first step, which maps the variables in the description to the "most appropriate" level of memory, is explained in Section 4.1. The second step, which estimates an architecture allocation using the variable-to-level mapping, is explained in Section 4.2.

4.1 Variable-to-Level Mapping

The algorithm pseudo-code for variable-to-level mapping is shown below. The basic idea is to map large, infrequently-accessed variables to the slow levels of memory and small, frequently-accessed variables to the fast levels. The access frequency of a variable is the number of times it is referenced during the execution of the description. The size of a scalar variable is $1\times$ the bitwidth of the variable, while the size of an array variable is the number of elements in the array $\times$ the bitwidth of a single array element.

In line 2 of the pseudo-code, we estimate the access frequency of each variable statically using 50% branching probability for conditionals, $(100/n)\%$ branching probability for n-way case statements, and 90% branching probability for loops. Line 3 computes the *priority* of each variable, which is defined as its access frequency divided by its size. In line 5, we sort the variables in order from minimum to maximum priority, and finally, in lines 6-15 we map each variable (in sorted order) to the slowest level possible such that the execution time constraint is not violated. The complexity of this algorithm is $\mathcal{O}(V(V+E))$ where V is the number of nodes in the CDFG and E is the number of edges. Step 9, testing whether the execution time constraint is satisfied, takes $\mathcal{O}(V+E)$ time, and we must add on a factor of $\mathcal{O}(V)$ since step 9 is in a loop.

```
Variable-to-Level Mapping
Input: Behavioral VHDL description, memory and bus
       configurations, FU library, clock period, search
       parameters, and execution time constraint.
Output: Assignment of variables to memory levels.
Begin Algorithm
(1)    For each variable $v$ do
(2)        Estimate access frequency $acc\_freq(v)$
(3)        Compute $priority(v) = acc\_freq(v)/size(v)$
(4)    End for
(5)    Sort variables on minimum to maximum priority
(6)    For each variable $v$ in sorted order do
(7)        Let $l$ be the slowest level of memory
(8)        Map $v$ to $l$
(9)        If execution time constraint is satisfied do
(10)           Goto (6) (accept new mapping)
(11)       Else
(12)           $l = l - 1$ (reject new mapping and
(13)           Goto (8) try a faster memory)
(14)       End if
(15)   End for
End Algorithm
```

During the level mapping process, all input and output variables are mapped, by default, to the slowest level of memory; however, the user can change this specification if he/she chooses. Note that the variable-to-level mapping completely determines the data-transfer-to-level mapping as well. After variable-to-level mapping, new nodes are added to the CDFG to represent the additional memory accesses and data transfers needed to preserve the memory hierarchy.

4.2 Resource Estimation

The goal of resource estimation is to determine the minimum number of resources needed to implement the design within the given execution time constraint. In AE, resource estimation is done separately for each basic block of the initial behavioral description, and the results for basic blocks are combined into a solution for the entire description. In this paper, we only explain resource estimation for basic blocks since the method of combining basic block solutions appears in [8].

The inputs for resource estimation are listed in Section 4 (behavioral design description, FU library, etc ...). In addition, an execution time constraint E and a variable-to-level mapping are given. Since the variable-to-level mapping is known and the FU library is simple, each data flow node is bound to a specific resource type. For instance a memory read node may be bound to the memory at level 2, a data-transfer node may be bound to a bus at level 1, or a operation node may be bound to an adder/subtractor. However, we still need to determine the required *number* of resources of each type.

The algorithm for estimating the required number of resources is listed below. This algorithm must be executed once for each resource type R. Resource types include memory ports (for the memory at each level), buses (at each level), and FUs such as adders or multipliers. Memory size is not considered as a resource here since it can be estimated immediately from the variable-to-level mapping by adding up the sizes of the variables.

```
Estimate_Resource(l,r)
Input: Minimum (maximum) number of resources $l$ ($r$).
Output: Lower bound estimate of resource cost $best$.
Begin Algorithm
(1)   If ($\sim$ Feasible_Schedule($l$)) & ($\sim$ Feasible_Schedule($r$)) do
(2)        Return
(3)   End If
(4)   If (Feasible_Schedule($l$)) and (Feasible_Schedule($r$)) do
(5)        Return
(6)   End If
(7)   Let $m = \lfloor (l+r)/2 \rfloor$
(8)   If Feasible_Schedule($m$) do
(9)        Let $r = m - 1$
(10)       Let $best = m$
(11)       Estimate_Resource(l,r)
(12)  Else
(12)       Let $l = m + 1$
(13)       Estimate_Resource(l,r)
(15)  End if
End Algorithm
```

```
Feasible_Schedule(m)
Input: Number of resources $m$.
Output: Boolean $b$ which is true if there is a feasible schedule
      using at most $m$ resources and false otherwise.
begin Algorithm
(1)   For time step $t$ = execution time constraint downto 1 do
(2)        Select $k$ nodes (where $k$ is as large as possible but not
           exceeding $m$) such that
           (i) the mobility of each selected node contains $t$, and
           (ii) the selected nodes have maximum ASAP values
                among all nodes satisfying (i)
(3)        Schedule the selected nodes in time step $t$
(4)   End for
(5)   If all nodes with the current resource type $R$ have been
      scheduled do
(6)        Return yes
(7)   Else
(8)        Return no
(9)   End if
end Algorithm
```

The algorithm Estimate_Resource performs a binary search to determine a *lower bound* on the number of resources of type R. We know that the minimum possible number of resources is 1 and the maximum is N, where N is the total number of nodes in the data flow graph of the basic block. We can perform binary search on the sequence of numbers $1 \ldots N$ to determine the required number of resources. In order to do binary search, we need the procedure Feasible_Schedule(m) which tells us whether or not there is a schedule for the data flow graph using at most m resources and E clock cycles, where E is the execution time constraint. Based on the results of Feasible_Schedule(m), we can determine which half of the sequence, $1 \ldots m-1$ or $m+1 \ldots N$, to search for the optimal number of resources.

Table 1: Functional Unit Descriptions

FU Name	Functions	BW	Area $((\mu m)^2)$	Delay (ns)
ADD16	$+$	16	118,272	27.00
ALU8	$+, -, =, <, >, \leq, \geq$	8	153,856	28.00
ALU16	$+, -, =, <, >, \leq, \geq$	16	307,712	32.00
CMP16	$=, <, >$	16	93,184	18.00
MUL16	$*$	16	8,675,744	79.00

Table 2: Memory Descriptions

Memory Name	Delay (ns)	Cost/Cell $((\mu m)^2)$
M_{10}	10.00	11,560
M_{50}	50.00	5,504
M_{100}	100.00	2,400

However, since resource and time constrained scheduling is NP-complete, procedure Feasible_Schedule(m) cannot tell us, for certain in polynomial time, whether a constraint-satisfying schedule exists. So, we relax the problem to see whether there exists a schedule such that every node n in the data flow graph is scheduled within its mobility interval, [ASAP(n),ALAP(n)]. This problem can be solved optimally in polynomial time using algorithm Feasible_Schedule(m) shown above. The proof of correctness appears in [9].

The worst-case time complexity of algorithm Estimate_Resource is $\mathcal{O}(N^2 \log N)$ where N is the number of nodes in the data flow graph. Procedure Feasible_Schedule(m) takes $\mathcal{O}(N^2)$ time since step 2 is $\mathcal{O}(N)$ and iterates at most N times (the execution time constraint cannot exceed the number of nodes). The factor of $\log N$ is added for the binary search. A detailed complexity analysis of Feasible_Schedule(m) is presented in [9].

Note that the problem solved by Feasible_Schedule(m) was originally defined in [15], and solved using a linear programming approach. Our approach is faster since the complexity of linear programming is dependent on many factors such as precision of the solutions, etc .... Finally, the worst-case complexity of our algorithm, Estimate_Resource, is lower than the complexity of any previous DSE algorithms [8, 10, 14, 15] since most of these approaches use ILP solvers which take exponential time or force-directed scheduling which has complexity $\mathcal{O}(N^3)$.

5 Experimental Results

AE has been implemented in C on a SUN SPARC 2 workstation. The following experiments show how AE can determine the best memory hierarchy for memory-intensive descriptions. The examples used in the experiments include (1) the centroid computation (CENTROID) from an industrial fuzzy logic controller design [6], (2) the inverse discrete cosine transform (IDCT) from [4], and (3) the kalman filter (KALMAN) from the high-level synthesis benchmark suite [3]. These examples are memory-intensive since their behavioral VHDL descriptions contain both complex control flow and frequent array-variable accesses.

For each of the examples, we varied the memory hierarchy and memory delay to observe the tradeoffs in cost

Table 3: CENTROID Least-Cost Design

Clock Period = $55ns$		
Execution Time	Memory Cost	Total Design Cost
8-14	M_{10}/M_{50}	M_{10}/M_{50}
15-40	M_{10}/M_{100}	M_{10}/M_{100}
41-52	M_{100}	M_{100}

Table 4: IDCT Least-Cost Design

Clock Period = $60ns$		
Execution Time	Memory Cost	Total Design Cost
33-37	M_{10}/M_{100}	M_{10}/M_{100}
38-42	M_{10}	M_{10}
43-47	M_{10}/M_{100}	M_{10}/M_{100}
48-52	M_{10}/M_{50}	M_{10}/M_{50}
53-57	M_{10}/M_{100}	M_{10}/M_{100}
58-88	M_{10}	M_{10}
89-107	M_{50}	M_{50}
108-154	M_{10}/M_{100}	M_{10}/M_{100}
155-160	M_{100}	M_{100}

Table 5: KALMAN: Least-Cost Design

Clock Period = $55ns$		
Execution Time	Memory Cost	Total Design Cost
38-47	M_{10}/M_{50}	M_{10}/M_{50}
48-67	M_{10}	M_{10}
68-77	M_{10}/M_{50}	M_{10}/M_{50}
78-117	M_{10}/M_{100}	M_{10}/M_{100}
118-127	M_{10}/M_{50}	M_{10}/M_{50}
128-171	M_{10}/M_{100}	M_{10}/M_{100}
172	M_{100}	M_{100}

and performance. A specific memory hierarchy is denoted by $M_{d_1}/\ldots/M_{d_L}$ where L is the number of levels, and $d_i, 1 \leq i \leq L$ is the memory delay. For instance, a 2-level memory hierarchy with memory delay $10ns$ ($50ns$) at the fast (slow) level would be denoted M_{10}/M_{50}. We used AE to determine the least-cost memory hierarchy for our examples out of five possible alternatives: $M_{10}, M_{50}, M_{100}, M_{10}/M_{50}$, and M_{10}/M_{100}.

The AE inputs for the experiments are described below. Table 1 lists areas, delays, and bitwidths (BW) for the different FUs in the $3\mu m$ CMOS technology, as determined by the estimator from [12], and Table 2 gives the estimated cost per cell for various speeds of memory. Bus delays are assumed to be $2ns$ and bus cost is 1. The search range is unrestricted, beginning with the fastest, most expensive design, and ending with the slowest, minimum-cost design, and the search granularity is 1, 5, and 10 for the CENTROID, IDCT, and KALMAN examples, respectively.

Tables 3, 4, and 5 list the minimum-cost memory configurations for the CENTROID, IDCT, and KALMAN examples at different execution times. The first column of each table lists execution time in clock cycles. The second column lists the configuration which minimizes total *memory* cost, while the third column shows the configuration which minimizes *total (FU, memory, and bus) design* cost.

The complete architecture explorations for the CENTROID, IDCT, and KALMAN examples (as opposed to the abstracted results listed above) appear in [9].

6 Conclusions

Since memory cost often dominates total design cost, finding the minimum-cost memory hierarchy for a design is an important problem. In fact, our experimental results show that, the least-cost design usually employs memory hierarchy because most of the variables are not accessed on every clock cycle. Also, determining the "correct" speed of the memory at each level is important, since sometimes different speeds of memory ($50ns$ or $100ns$) are more cost-efficient than others.

In summary, memory hierarchy is a cost-efficient design alternative to large high-speed memories; but there are many tradeoffs to evaluate (number of levels, speeds of memories, etc ...). Therefore, automatic exploration of different memory hierarchies is among the most important problems to study in the CAD field.

References

[1] I. Ahmed and C. Chen, "Post Processor for Data Path Synthesis Using Multiport Memories," *IEEE Transactions on Computer-Aided Design*, Nov. 1991.

[2] M. Balakrishnan, A. Majmudar, D. Banerji, J. Linders, and J. Majithia, "Allocation of Multiport Memories in Data Path Synthesis," *IEEE Transactions on Computer-Aided Design*, Apr. 1988.

[3] N. Dutt and C. Ramachandran, "Benchmarks for the 1992 High-Level Synthesis Workshop," Technical Report #92-107, Department of Information and Computer Science, University of California at Irvine.

[4] P. M. Embree and B. Kimble, C Language Algorithms for Digital Signal Processing, Prentice Hall, Inc., Englewood Cliffs, New Jersey 07632, 1991.

[5] D. D. Gajski, N. Dutt, A. C-H. Wu, and S. Lin, High-Level Synthesis, Kluwer Academic Publishers, Norwell, Massachusetts 02061, 1992.

[6] D. D. Gajski, L. Ramachandran, P. Fung, S. Narayan, and F. Vahid, "100-Hour Design Cycle: A Test Case," *Proceedings of the European Design Automation Conference (EURO-DAC)*, to appear 1994.

[7] P. Gupta and A. C. Parker, "SMASH: A Program for Scheduling Memory-Intensive Application-Specific Hardware," *Proceedings of the High-Level Synthesis Workshop*, 1994.

[8] N. D. Holmes and D. D. Gajski, "An Algorithm for Generation of Behavioral Shape Functions," *Proceedings of the European Design Automation Conference (EDAC)*, 1994.

[9] N. D. Holmes and D. D. Gajski, "Architectural Exploration for Datapaths with Memory Hierarchy," Technical Report #94-37, Department of Information and computer Science, University of California at Irvine.

[10] R. Jain, A. C. Parker, and N. Park, "Predicting System-Level Area and Delay for Pipelined and Non-Pipelined Designs," *IEEE Transactions on Computer-Aided Design*, Aug. 1992.

[11] T. Kim and C. L. Liu, "Utilization of Multiport Memories in Data Path Synthesis," *Proceedings of ACM/IEEE Design Automation Conference*, 1993.

[12] L. L. Larmore, D. D. Gajaki, and A. C-H. Wu, "Layout Placement for Sliced Architecture," *IEEE Transactions on Computer-Aided Design*, Oct. 1991.

[13] L. Ramachandran, D. D. Gajski, and V. Chaiyakul, "An Algorithm for Array Variable Clustering," *Proceedings of the European Design Automation Conference (EDAC)*, 1994.

[14] A. Sharma and R. Jain, "Estimating Architectural Resources and Performance for High-Level Synthesis Applications," *IEEE Transactions on VLSI Systems*, Jun. 1993.

[15] A. H. Timmer, M. J. M. Heijligers, and J. A. G. Jess, "Fast System-Level Area-Deay Curve Prediction," *Proceedings of the Asian-Pacific Conference on Design Automation*, 1994.

[16] J. Vanhoof, K. Van Rompaey, I. Bolsens, G. Goossens, and H. De Man, High-Level Synthesis for Real-Time Digital Signal Processing, Kluwer Academic Publishers, Norwell, Massachusetts 02061, 1993.