

SMASH: A Program for Scheduling Memory-Intensive Application-Specific Hardware*

Pravil Gupta and Alice C. Parker
Electrical Engineering – Systems
University of Southern California
Los Angeles, CA 90089-2562

Abstract

The research described in this paper addresses automatic synthesis of memory-intensive application-specific systems, with emphasis on hierarchical storage architecture design. SMASH is a program which combines storage hierarchy design with data path synthesis. It uses appropriate system parameters in order to coordinate between the synthesis of different sub-architectures of the system and schedules data transfers between them. We synthesized some designs examples using SMASH and have included the results in this paper.

1 Introduction

Our goal is to automate the design of datapaths and storage architectures for memory-intensive systems. The storage architecture is closely connected to the system datapath, and isolating its synthesis from datapath synthesis may not result in an efficient solution. Our tools design application-specific systems, where the memory-access pattern is not only relatively fixed but also known before hand. This mostly-deterministic access characteristic makes it straightforward to automate the memory design process.

SMASH designs a hierarchical storage system concurrently with the datapath and also determines the input/output data-transfer schedule between various hierarchies and the datapath, as the datapath itself is scheduled. The need for such a combined synthesis is illustrated in Figure 1. In Figure 1 three sched-

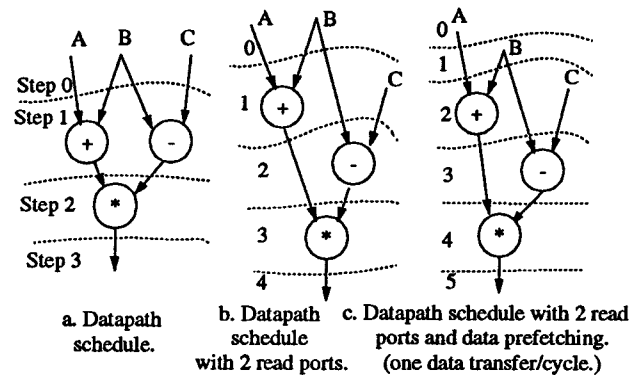


Figure 1: An example showing scheduling with memory-related issues.

ules are shown, first unconstrained, the second with a constraint on memory ports and the third with an additional constraint on data-transfer bandwidth. The required buffer size decreases from 3 to 2 as the number of ports is constrained. In general, the increase in schedule length due to data transfers may not result in execution delay if the transfers overlap the execution of other parts of the CDFG.

2 Problem statement

An overview of SMASH is shown in Figure 3. The program inputs are

- the behavioral VHDL description of a memory-intensive application-specific system, which may contain bounded inner loops and conditional branches. Loop structures, arrays and indexed references are assumed to be transformed and optimized.
- the module library consisting of (i) functional modules (e.g. adders) with each module characterized by its area, delay and bitwidth, and (ii) storage modules (e.g. registers, single-

*This work was supported by the Advanced Research Projects Agency and monitored by the Federal Bureau of Investigation under contract No. JFBI90092. The views and conclusions considered in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Advanced Research Projects Agency or the U.S. Government.

port/multiport register-files, single-port/multiport RAMs) characterized by cost per word, number of ports, access time and storage capacity;

- area-performance constraints;
- the clock cycle, which is the duration of each control step in the datapath;
- external input/output timing constraints; and
- memory bandwidth constraints (the number of words that can be transferred onto the chip in one control step).

Our synthesis system produces a two-chip system with a *datapath* consisting of operators and operation schedule, *size and port configuration* for on-chip foreground memory to store input/output and intermediate variables, *data-transfer schedule* between the datapath and on-chip memory, *size and port configuration* of off-chip background memory for bulk storage, and *data-transfer schedule* between the on-chip and off-chip memory.

3 Related research

The original MIMOLA system was the first system to make tradeoffs in the use of multiport memories [7]. Balakrishnan et al. [2] presented an approach to use multi-port memories to implement single isolated registers. Chen [3] explored the design space for multiport memory synthesis. Ahmad and Chen [1] use 0-1 integer-linear programming to group intermediate variables in the datapath into a minimum number of multiport memories depending on their ports and their access pattern. Stok [11] optimizes register files during the synthesis process. Grant et al. [5] suggested an approach to group the memory requirements of various operators using single-port memory modules such that control and communications may be optimized.

Lippens et al. [6] described automatic memory allocation and address allocation for high speed applications without conditional branches. They synthesize memory after the datapath scheduling and allocation. They model multi-dimensional periodic signals as data streams and then manipulate these streams to form a distributed memory structure. They allow only 1 and 2 port RAMs and distribute the data among parallel memories. They do not consider storage hierarchies.

IMEC's CATHEDRAL-II [12] compiles multi-dimensional data structures into distributed dual-port register files and single-port SRAMs. They use a polyhedral-based model for high-level memory management for linear, piecewise linear and data dependent signal indexing[4]. Although other researchers

have published results on transformations to reduce memory costs, we do not report them here since the objective of this specific software is to perform synthesis after the high-level memory management and transformation steps have been applied.

4 Target architecture

The target system architecture consists of a **datapath** and a **hierarchical storage architecture** (Figure 2) as described below. Although the top-level target architecture is fixed, there are a large number of tradeoffs possible. The memories can degenerate into a simple set of wires if they are not needed. The **on-chip foreground memory** consists

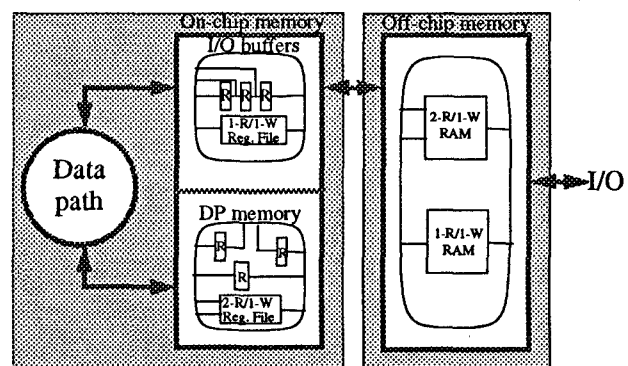


Figure 2: Target architecture.

of I/O buffers which interface to the off-chip memory and datapath memory. The user specifies the bandwidth between the buffers and the off-chip memory (BW_{on-off}), which is the number of inputs/outputs that can be transferred from/to the off-chip memory in one control step. The synthesis software determines (i) total buffer size, which is determined by the maximum number of inputs/outputs stored in the buffers in any given step, and (ii) the number of read/write ports accessible to the datapath R_{buf}/W_{buf} , which is the maximum number of inputs/outputs the datapath accesses in any given step. Datapath memory stores the intermediate variables in the datapath. Other researchers [2, 3, 11] have studied datapath memory tradeoffs and hence such tradeoffs are not described in this paper.

All the I/O data values from/to the external world are stored in the **Off-chip background memory**, which is generally large and inexpensive, just as in a general purpose computer. The user determines the number of read/write ports on the off-chip memory* (BW_{on-off}) and the synthesis software determines the off-chip memory size.

The required bandwidth between the on-chip and the off-chip memory imposes cost constraints on the

overall design because of the pin requirements on the chips, and the expense of having multiport memory for off-chip bulk storage. Furthermore, the access time for the off-chip background memory may be greater because it is off chip, and is bigger in size. As a result, the software schedules the prefetch of the input variables from off-chip memory into the I/O buffer of the on-chip memory before they are required. In each control step the required data is loaded onto the buffer ports and as the datapath reads data from the buffers, the controller moves new data to them from the off-chip memory for further processing. Similarly, the output variables are first stored in the on-chip memory and then are transferred to off-chip memory when they are no longer needed or if they can be refetched easily, overlapped with the datapath execution.

5 Design methodology

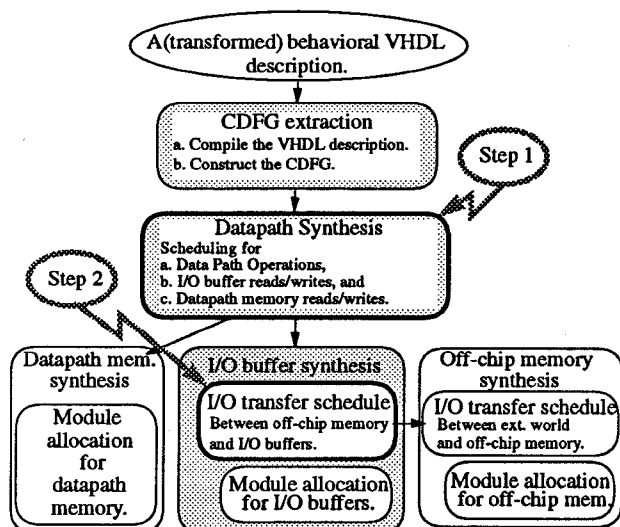


Figure 3: Design steps in SMASH.

The major design steps of our methodology are shown in Figure 3. The SMASH software implements the following two steps (highlighted in the figure): First, **datapath synthesis** with operation scheduling is performed combined with scheduling of on-chip data transfers to/from I/O buffers. As a result of this scheduling, constraints are placed on the memory structure. The second step, the **I/O transfer scheduling**, includes determining the data transfers between the two levels of the memory hierarchy. We ensure that first step of the stepwise construction of the system takes into account the second step by looking ahead so that the second step is not overly constrained. Global design parameters like BW_{on-off} and timing constraints are considered when construct-

ing the partial design in each step.

5.1 Design tradeoffs in storage architecture used in SMASH

Three parameters can vary while making the cost-performance tradeoff, (i) number of ports, which determines BW_{on-off} , (ii) size of the storage architecture, and (iii) execution time. To deal with this 3-way tradeoff, we iterate on BW_{on-off} by repeatedly invoking the synthesis software. For each user-specified BW_{on-off} , the software trades off between the size and the execution time. The software can trade off between (i) more clock cycles (delay the execution) or larger buffer size (prefetch the data before it is required and store it); (ii) more ports (fetch more data with a wider bandwidth) or more clock cycles (fetch more data with extra cycles); and (iii) more ports (retrieve them repeatedly whenever needed with wider BW_{on-off} specified by the user) or larger buffer size (save them for future use which will increase the I/O buffer size) for data values which are used again. This can be done efficiently because the number of ports is usually small.

5.2 Step 1: Combining datapath scheduling with I/O accesses

In step 1, the software determines the number of functional units of each type required for the design and the scheduling of all the operations in the CDFG to appropriate control steps. It also outputs the percentage utilization of each type of module in each step, which helps us determine the quality of the schedule. We combine the scheduling of I/O reads/writes from/to the on-chip buffers with the scheduling of operators by inserting a read/write (R/W) node which corresponds to every I/O access to the buffers into the CDFG. The read node consists of two inputs: (i) array name, and (ii) index and one output: value. The write node has three inputs: (i) array name, (ii) index, and (ii) value and one output: array (modified). During the datapath scheduling step, whenever an operator involving I/O is scheduled, the corresponding R/W node is also scheduled to the same step, implying an I/O buffer access in that step. All memory-related issues are considered during this scheduling by checking if

1. the I/O buffers are able to provide the required number of R/W ports in that step, and
2. the input data, if required, could be prefetched into the buffers from the off-chip memory prior to that step and the output data, if any could be transferred back to the off-chip memory before it was required

elsewhere. Note that the data transfer is overlapped with the datapath execution:

$$\begin{aligned} I(s) &\leq BW_{on-off} \times s \text{ and} \\ O(s) &\leq BW_{on-off} \times (MaxSteps - s) \end{aligned}$$

where $I(s)$ is the number of data values required for all the operations scheduled before or in control step s for all s . $O(s)$ is the output number.

The schedule obtained in the first step guarantees that no bandwidth or I/O timing constraint is violated in the following design step when the complete I/O transfer schedule between the on-chip and off-chip memory is determined. In addition, this process provides the I/O requirement schedule which is the starting point for the second step.

Priority can be given to performance optimization or area optimization while satisfying the area-performance constraints [9]. When performance optimization is a primary objective, whenever the scheduler is unable to schedule a node, it first attempts to add an extra resource and if it fails adds an additional step (*vice versa* for area optimization). The software also takes into account external timing constraints on the input/output data values.

The software assumes two-phase clocking. The datapath reads and processes the data in the first phase and writes it back on or off-chip in the second phase.

We estimate the cost of the I/O buffer using (i) buffer size (i.e. the maximum number of inputs/outputs stored in a step) and (ii) the required number of ports.

The software uses list scheduling with freedom of the nodes as the urgency factor [9]. Distribution graphs of the *read* (*write*) operations and the datapath operations are used to assign them probabilistically to the most suitable control step, as is done in force-directed scheduling [10].

To improve the total execution time and allow operators with varying execution delays, the nodes in the CDFG can be *chained*, or *distributed over multiple cycles* appropriately. A bitmap representing the associated predicates is attached to each node, and is used in determining the mutual exclusion among the nodes efficiently.

For the I/O operations inside the **conditional branches** we attempt to schedule the *test* node as early as possible so that the interval between the test node and operation node (requiring the I/O) is large enough to transfer the data into I/O buffers from off-chip memory. The data transfer is done concurrently with the transfer of other data values being used in other parts of the CDFG. In step two when the ac-

tual data transfer is scheduled, the software may or may not be able to make all the required transfers in time. This gives rise to the following two scenarios: (i) dynamic selection of the data to be transferred, or (ii) worst case transfer of all potentially required values. The scheduler prefers dynamic selection over worst case analysis as the latter is inefficient. Our approach to **array accesses** is very similar. The loops are pipelined in order to achieve higher performance. They are scheduled first and then are treated as one unit for the scheduling of the remaining CDFG.

5.3 Step 2: I/O transfer between on-chip and off-chip memory

The second step is to schedule data accesses from the off-chip memory to the I/O buffers to satisfy the schedule obtained in the previous step minimizing the buffer size while meeting timing constraints. Given BW_{on-off} , the time interval during which the data is available, and the data requirement (as determined in the previous step); we determine a complete data transfer schedule to and from off-chip memory, such that the size of the I/O buffers is minimized, and the buffers provide the data to the datapath as they are required and store the outputs as they are produced.

The software expects as inputs the BW_{on-off} and the data requirement, determined in the datapath synthesis step; and the interval during which each data value is available, determined by the external timing constraints.

Data-transfer scheduling is analogous to the scheduling problem in datapath synthesis. The software uses *list scheduling*, with a redefinition of the objective function. An *urgency factor* associated with each value determines the necessity of the value transfer to be scheduled in a step. This factor depends on: (i) the number of steps remaining to prefetch the data into buffers, and (ii) whether the value is going to be required again, which is detected based on its index (then its urgency is lowered for the second access because it can be stored in the buffer). Updating the list is based on the urgency factor associated with each data value. We fetch the inputs as late as possible, in a greedy fashion, to minimize the buffer size. Similarly, the outputs are transferred back to the off-chip memory as soon as possible.

The output of step two consists of the complete data access schedule for the I/O buffers and the required buffer size (determined from the access pattern.)

6 Experimental results

We applied our technique to three representative benchmark examples: a second-order differential equation solver [10], an AR filter element [8], and an Elliptic filter element. Our experiments were designed to demonstrate the following aspects of our research: (i) the capabilities of the software to combine the datapath scheduling with I/O access from the buffers, (ii) validating datapath schedules obtained when considering memory-related constraints, and (iii) existence of the cost-performance tradeoff in the storage architecture (cost being a function of R/W ports, storage size, and BW_{on-off}).

Module name	Area (μm^2)	Delay (ns)	Module name	Area (μm^2)	Delay (ns)
Multiplier	2398490	55	Comparator	81558	30
Adder	81558	30	Dist./Join	0	0
Subtractor	81558	30	Register	96624	3

Table 1: module library.

Design number	Input constraints		Software output				
	Functional area (μm^2)	BW_{on-off} (words/cycle)	R_{buf}	W_{buf}	Buffer size (words)	Functional resources	Execution time (ns)
2nd-order differential equation							
1	7500000	(4)	(4)	(2)	NA (0)	<, +, -, 3 *	240
2	7500000	2	4	2	2	<, +, -, 3 *	240
3	6000000	(4)	(4)	(1)	NA (0)	<, +, -, 2 *	300
4	6000000	2	3	1	2	<, +, -, 2 *	300
5	6000000	1	3	1	3	<, +, -, 2 *	330
6	3000000	2	2	1	1	<, +, -, *	450
7	3000000	1	2	1	3	<, +, -, *	480
AR filter element							
1	10000000	(6)	(6)	(2)	NA (0)	2 +, 4 *	390
2	10000000	4	4	2	0	2 +, 4 *	390
3	10000000	3	4	2	4	2 +, 4 *	420
4	5000000	2	4	2	2	2 +, 2 *	600
Elliptic wave filter							
1	10000000	(3)	(3)	(4)	(3)	4 +, 4 *	540
2	10000000	3	3	4	3	4 +, 4 *	540
3	7500000	2	2	2	3	3 +, 3 *	600
4	5500000	2	3	3	2	3 +, 2 *	690
5	5000000	2	2	1	1	2 +, 2 *	720

Table 2: Experimental results.

The module library used is shown in Table 1. The clock cycle used is 30ns. To simplify the experiments

we have assumed that the input data is present in an off-chip RAM before processing. Our software does allow the user to impose timing constraints on I/O. The software was executed with priority on performance optimization while meeting the area constraint. We also generated some designs without considering memory-related constraints (BW_{on-off} and R_{buf}/W_{buf}). These implementations are summarized in Table 2. The manually determined storage-architecture-related parameters are shown in parentheses for the designs without memory-related constraints.

In this paper, we analyze the differential equation example in detail. Designs 1 and 3 vs. 2 and 4 respectively clearly demonstrate the effectiveness of our technique in reducing the unnecessary storage cost of ports on off-chip memory (BW_{on-off}). Schedules with memory constraints achieved the same performance with the same functional area while reducing BW_{on-off} and R_{buf}/W_{buf} . This was because in our technique we distributed the data requirement evenly and overlapped the data transfer with the execution.

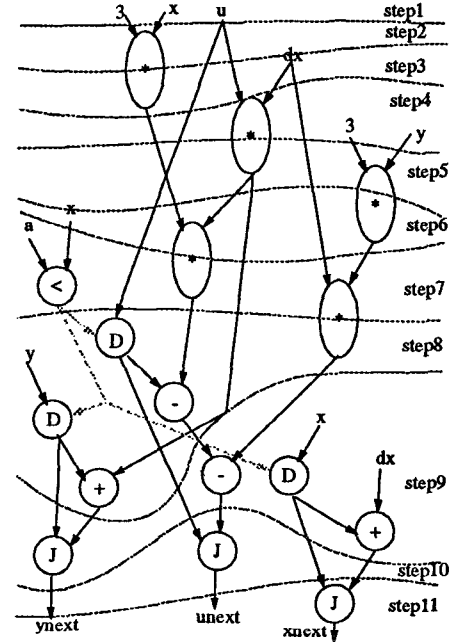


Figure 4: Schedule for 2nd-order differential equation (design 5).

Designs 5, 6 and 7 were generated to demonstrate the tradeoff curve existing in the storage architecture. Designs 4 vs. 5 and 6 vs. 7 show that for the same functional resources, varying storage parameters resulted in different execution delays. Designs 5 and 7 are slower than designs 4 and 6 respectively. Analysis

shows that this was because of the limited bandwidth allowed in 5 and 7. Furthermore, in these designs, as the I/O transfer was the bottleneck, unnecessary transfers were avoided by storing the data value in the buffers for further use, hence the buffer size increased. Note that although these slower designs require bigger buffer sizes, they are still cheaper overall because of the lower bandwidth (pin) requirement.

Figure 4 shows the datapath schedule in detail for design 5, excluding the R/W nodes. The results obtained from the on-chip off-chip data transfer scheduling software are summarized in Table 3. Inputs 'x' and

Step no.	Buffers-Off chip data transfer		Buffer contents	Buffers-Datapath data transfer	
	Write into buffer	Read from buffer		Read from buffer	Write into buffer
1	x	-	x	-	-
2	3	-	3,x	3,x	-
3	dx	-	3,dx	-	-
4	u	-	3,dx	u,dx	-
5	y	-	dx,y	3,y	-
6	x	-	dx,x,y	-	-
7	a	-	dx,x,y	a,dx,x	-
8	u	-	dx,x,y	y,u	-
9	-	ynext	dx,x	dx,x	ynext
10	-	unext	-	-	unext
11	-	xnext	-	-	xnext

Table 3: Data transfer schedule for design 5 (2nd-order differential equation).

'u' are read twice into the buffer because there was an empty 'slot' available for retransfer before they were required again, and storing them in the buffers would have resulted in an increase in the buffer size. On the other hand, 'dx' remained in the buffers for future use because its retransfer was not possible before the given time. The inputs which are transferred from the off-chip memory and required by the datapath in the same step need not be stored. Similarly the outputs which are produced and transferred back to off-chip memory in the same step are not stored.

References

- [1] I. Ahmad and C. Y. Roger Chen. Post-Processor For Data Path Synthesis Using Multiport Memories. In *Proc. of the Int'l Conf. on Computer Aided Design*, pages 276–279, 1991.
- [2] M. Balakrishnan, A.K. Majumdar, D.K. Banerji, J.G. Linders, and J.C. Majithia. Allocation of Multiport Memories in Datapath Synthesis. *IEEE Trans. on Computer Aided Design*, 7(4):536–540, April 1988.
- [3] C. H. Chen. Allocation of Multiport Memory with Ports of Different Types in Register Transfer Level Synthesis. In *Proc. of the Int'l Conf. on Computer Design*, pages 418–421, 1991.
- [4] Frank Franssen, Florin Balasa, M. Swaaij, F. Catthoor, and H. De Man. Modeling multi-dimensional data and control flow. *IEEE Trans. on VLSI Systems*, 1(3):319–327, September 1993.
- [5] D.M. Grant and P.B. Denyer. Memory, Control and Communication Synthesis for Scheduled Algorithms. In *Proc. of the 27th Design Automation Conference*, pages 162–167, June 1990.
- [6] P.E.R. Lippens, J.L. van Meerbergen, A. van der Werf, W.F.J. Verhaegh, and B.T. McSweeney. Memory Synthesis for High Speed DSP Applications. In *Proc. of the IEEE Custom Integrated Circuits Conf.*, pages 11.7.1–11.7.4, May 1991.
- [7] P. Marwedel. The MIMOLA Design System: Detailed Description of the Software System. In *Proc. of the 16th Design Automation Conference*, pages 59–62, 1979.
- [8] A. C. Parker, P. Gupta, and A. Hussain. The Effects of Physical Design Characteristics on the Area - Performance Tradeoff Curve. In *Proc. of the 28th Design Automation Conference*, pages 530–534, June 1991.
- [9] A. C. Parker, J. Pizarro, and M. Mlinar. MAHA: A Program for Datapath Synthesis. In *Proc. of the 23th Design Automation Conference*, pages 461–466. IEEE and ACM, July 1986.
- [10] P.G. Paulin and J.P. Knight. Force-Directed Scheduling for the Behavioral Synthesis of ASICs. *IEEE Trans. on Computer Aided Design*, pages 661–679, June 1989.
- [11] L. Stok. Interconnect Optimization during Datapath Synthesis. In *Fourth International Workshop on High-Level Synthesis*, October 1989.
- [12] J. Vanhoof, I. Bolsens, and H. De Man. Compiling Multi-dimensional Data Streams into Distributed DSP ASIC Memory. In *Proc. of the Int'l Conf. on CAD*, pages 272–275, 1991.