

CONCURRENCY CHARACTERISTICS IN DSP PROGRAMS

Lisa Guerra[†], Miodrag Potkonjak[‡], and Jan Rabaey[†]

[†]Dept. of EECS, University of California, Berkeley

[‡]C&C Research Laboratories, NEC USA, Princeton

ABSTRACT

The exploration of concurrency has emerged as a dominant research problem in the VLSI DSP literature. A great variety of forms of concurrency exploration have been proposed, analyzed and used on a number of hardware platforms. The belief that the DSP domain is amenable to concurrency exploration has been gaining popularity; however, no systematic study has been conducted to confirm or dispel this claim. This paper presents a global view of the concurrency problem, presenting comprehensive statistics on concurrency properties in commonly used DSP programs. Particular emphasis is placed on the potential cost effectiveness of concurrency exploitation.

1. MOTIVATION AND CURRENT STATE-OF-THE-ART

The study of concurrency and its applications is an important research area. Concurrency can be realized in many forms, including pipelining, superpipelining, lookahead, vectorization, superscalar processing, interleaving, systolic and wavefront arrays, time and space sharing, and multithreading [1].

Exploiting concurrency improves not only speed measures such as throughput and latency, but also other performance metrics. For example, it has been shown that one of the most effective ways of reducing power consumption is to exploit concurrency to trade area for lower voltage and thus lower power [2].

There is a strong consensus that finding concurrency and using it efficiently are key tasks in DSP, and in particular in the area of VLSI DSP. A number of popular VLSI DSP hardware platforms, including DSP processors, cores and full custom ASICs, support one or more forms of concurrency.

An important step in several computational areas has been the undertaking of studies on the amount of

available concurrency for the particular domain. Results of those studies, in particular for the domains of general purpose and scientific computing, have greatly influenced architecture and compiler designs.

In the general purpose computing domain, for example, a number of studies have been conducted, where the amount of parallelism in common programs was found to be between 2 and 90 operations, depending on the hardware model, power of optimizing compiler and the set of analyzed examples [3,4,5]. For computation intensive engineering and scientific computing applications, Kumar's study showed that between 500 and 3,500 operations simultaneously execute in each clock cycle [6], giving impetus for the exploration of massive parallelism. Contrary to the initial widespread belief that artificial intelligence (AI) programs cannot achieve significant speed-ups by exploiting concurrency, Kibler and Conery [7] showed more than an order of magnitude improvement, influencing the development of several parallel AI architectures.

This paper presents a comprehensive analysis of concurrency properties found in DSP applications. This study has a number of implications for guiding architecture and compiler design and optimization tool development. In addition, this study of concurrency is one step in distinguishing different classes of examples and can guide in the creation of a taxonomy of DSP applications.

2. EXPERIMENTS ON CONCURRENCY PROPERTIES

This project aims to quantitatively characterize concurrency properties in the DSP domain. Important concurrency properties are identified, and extracted over a large set of DSP examples.

The primary goals are the following:

1. to analyze the amount of parallelism that can be exploited in general purpose DSP and application specific processors, and in particular, to determine the level of efficiency which can be achieved with their exploitation;
2. to analyze intrinsic bounds on available concurrency (e.g. iteration bound) and their importance;
3. to determine which functional units are the most amenable for time sharing (in order to justify or challenge the widespread view that the multiply-accumulate (MAC) unit [8, 9] is a mandatory part of DSP datapaths);
4. to identify which components dominate custom ASIC implementation cost during concurrency exploitation.

A number of concurrency properties have been measured on a set of 59 examples which include FIR and IIR linear, nonlinear and adaptive filters of various structures, one dimensional FFT and DCTs, echo cancellers, convolutions, elementary functions calculations, linear controllers, histogram and several DSP subsystems and systems. Note that this set is by no means complete, and has possibly missed important extremities in the algorithmic space. We believe though that it is representative of a large part of DSP applications.

In order to quantify concurrency on these examples, we have measured and analyzed concurrency properties in the following categories:

1. Amount of parallelism, measured by the maximum sustained parallelism, which represents the number of operations that can be executed simultaneously under the assumption that the execution takes place at the maximum sampling rate, using minimal amount of hardware. For this measurement we have used a polynomial time algorithm for maximum parallelism detection, developed in the Hyper system [10];
2. Upper bound on resource utilization, calculated as the ratio of number of nodes in the computation graph over the product of maximum sustained parallelism and initial maximal sampling rate;
3. Potential effectiveness of pipelining for throughput, composed of two measures: (i) the integer iteration bound¹ (relative to the critical path) [1] and (ii) the percentage of operations in cycles (recursive paths). Note that all operations outside cycles can be func-

tionally pipelined, and can thus be executed simultaneously;

4. Instruction set/template matching, quantified by the percentage of nodes that match a given template, and by the node coverage by sets of templates;
5. Overall foreground memory vs. functional unit area, determined by the percentage of area contributed by memory (memory/(memory + EXU area)) in custom ASIC implementations.

All parameters are measured on the initial set of DSP examples, without applying any concurrency optimizing transformations. Since larger examples have the potential for larger numbers in the categories of maximum parallelism and iteration bound, these values are normalized by the size of the example (number of operations). This enables the detection of trends in the behavior of those parameters, and gives insight into what can be expected as the size of examples increases. Table 1 shows the compiled results for all parameters. The smallest example has 9 operations, while the largest example has 7,376,121 operations.

3. ANALYSIS OF RESULTS

The main conclusions from the study can be summarized as follows:

1. Maximum sustained parallelism was notable, but not exceptionally high (range 3-33). However, after applying optimizing transformations [11, 10] (in particular after functional pipelining, algebraic manipulations and common subexpression replication), the maximum parallelism increased dramatically. In a few examples it was possible to execute several hundred operations simultaneously.
2. Upper bound on resource utilization was in the range of 20-80%. The average value, however, was only 50%, implying that a high utilization (needed in ASIC applications) can rarely be achieved without optimizing transformations.
3. On the subject of cycles influence (percentage of nodes in cycles and iteration bound), examples fell mainly into two distinct classes: those with no nodes in cycles, and those with over 50% of nodes in cycles. Surprisingly, the iteration bound was mainly low, even for examples with a high percentage of nodes in cycles, and frequently, transformations were efficient in reducing it. We also noted that the size of the iteration bound increased at a lower than linear rate with the size of the example. The low ratio between iteration bound and critical

1. the smallest integer greater than the iteration bound

Parameter	min	median	average	max
Maximum Sustained Parallelism	3	8	11.5	33
Normalized Maximum Sustained Parallelism	0.04	0.31	0.22	1.00
Upper bound on Resource Utilization	19%	46%	49.2%	86%
Percentage of nodes in cycles	0%	52%	37%	100%
Critical Path	3	14	100, 395	3,161, 852
Integer Iteration bound	1	2	3.6	17
(Integer Iteration bound) / (Critical Path)	.033	.30	.33	1
Normalized Integer Iteration bound	0.001	0.079	0.027	0.5
Most dominant pattern (multiply-add)	0%	41%	39%	100%
Second most dominant pattern (add-add)	0%	15%	15%	48%
Coverage by pair of dominant patterns	0%	41%	43%	100%
Memory/ (EXU + Memory) Cost	4%	14%	24.5%	90%

Table 1: Various concurrency parameters for a variety of examples from the DSP domain

path indicates the importance of functional pipelining for achieving high throughput.

- In only one example, the 5th order wave digital filter [12], all operations were in cycles. This was also the only example where the critical path started and finished at an algorithmic delay. This example is a de facto standard benchmark of the high level synthesis community, and is often one of the few used to determine the quality of a synthesis tool [13]. However, analysis of concurrency in DSP programs suggests that the peculiar and unique structure of this example makes it ill-suited to act as a predictor of how a particular tool would perform on other examples which have largely different concurrency characteristics.
- Among template patterns (instruction sets), the dominant one was the multiply-add, with the add-add pattern placing a distant second. This confirms the widespread belief that the MAC unit [9] is an important component of DSP datapath architectures. The percentage of nodes covered by these top two patterns was barely higher than that covered by the multiply-add pattern alone, which indicates that adding the add-add unit to the current generation of DSP processors is not likely to improve the efficiency of time sharing. Note that the compiler and high level synthesis optimizing transformations can have a dramatic effect on the concurrency properties. In terms of template patterns, for example, after substituting constant multiplication with shift and adds, the shift-add template emerges as a dominant pattern while the multiply-add becomes less common. Table 2 shows a set of more detailed results. Of interest is the difference in node coverage when either addition or subtraction can be per-

formed and when only addition can be performed. The mult-add/sub has slightly greater coverage than the mult-add, and the shift-add/sub has significantly greater coverage than the shift-add (after transformation).

- The percentage of area contributed by foreground memory relative to EXU area was relatively low. This is mainly a consequence of two facts. First, most of the examples used several multiplier units (whose area grows quadratically with the number of bits) which are much larger than registers. Second, we have measured those two parameters assuming maximum throughput which often results in underutilized execution units. For several examples, we have tried more sequential implementations. The memory cost was only slightly affected by a change of throughput constraints, while the area of execution units was sharply reduced. This difference in sensitivity of two components of implementation cost to throughput, made memory heavily dominant part of the implementation cost in low speed designs (range 60-95%). In general when the number of bits was high and the time sharing factor was low, the designs were EXU-dominated. On the other hand, for low bitwidths and high time sharing ratios, memory clearly dominated the implementation cost.

4. CONCLUSIONS

Several aspects of concurrency in the DSP domain have been analyzed in a comprehensive experimental study. A number of important consequences for DSP general purpose and dedicated architectures and DSP compilers have been derived. First, the maximum

Instruction Set	Before Transformation				After Transformation			
	min	median	average	max	min	median	average	max
mult-add	0	41	39	100	0	0	4	48
add-add	0	15	15	48	0	8	9	25
shift-add	0	0	0	0	0	39	32	59
mult-add, add-add	0	41	43	100	0	8	11	48
shift-add, add-add	0	15	15	48	0	43	37	60
mult-add/sub	0	41	45	100	0	0	5	58
add/sub-add/sub	0	21	23	48	0	13	17	62
shift-add/sub	0	0	0	0	0	64	59	93
mult-add/sub, add/sub-add/sub	0	44	53	100	0	14	19	75
shift-add/sub, add/sub-add/sub	0	21	23	48	0	73	69	97

Table 2: Node coverage (%) by instruction sets before and after constant multiplication expansion

sustained parallelism, upper bound on resource utilization, and iteration bound relative to critical path all showed that although the initial concurrency of DSP programs is relatively modest, application of optimizing transformations in almost all examples significantly increases concurrency. Second, although overall the dominant template pattern was the MAC, we found that after applying transformations, other patterns such as shift-add became notable. Lastly, it is evident that the concurrency properties can be used to distinguish classes of examples. Creation of a taxonomy of DSP examples can lead to the development of new tools which target these classes in more efficient and effective ways than tools designed to be effective on a large variety of computations with a great diversity of concurrency characteristics.

Acknowledgments

This work is supported by ARPA under research grant J-FBI-90-073, and NEC. L. Guerra is supported by ONR and AT&T Fellowships.

5. REFERENCES

- [1] S.Y. Kung: *VLSI Array Processors*, Prentice-Hall, Englewood Cliffs, NJ, 1988.
- [2] A.P. Chandrakasan, M. Potkonjak, J. Rabaey, R. Brodersen: "Hyper-LP: A Design System for Power Minimization using Architectural Transformations," *IEEE/ACM Int'l Conference on Computer-Aided Design*, Santa Clara, CA, pp. 300-303, 1992.
- [3] N.P. Jouppi, D.W. Wall: "Available instruction-level parallelism for superscalar and superpipelined machines," *3rd Int'l Symposium on Architectural Support for Programming Languages and Operating Systems*, pp. 272-282, 1989.
- [4] A. Nicolau, J.A. Fisher: "Measuring the parallelism available for very long instruction word architecture," *IEEE Trans. on Computers*, Vol. 33, No. 11, pp. 968-976, 1984.
- [5] G.S. Tjaden, M.J. Flynn: "Detection and Parallel execution of parallel instructions," *IEEE Trans. on Computers*, Vol. 19, No. 10, pp. 889-895, 1970.
- [6] M. Kumar: "Measuring Parallelism in Computation-Intensive Scientific/Engineering Applications," *IEEE Trans. on Computers*, Vol. 37, No. 9, pp. 1088-1098, 1988.
- [7] D.F. Kibler, J. Conery: "Parallelism in AI Programs," *9th Int'l Joint Conference on Artificial Intelligence*, Los Angeles, CA, pp. 53-56, 1985.
- [8] Analog Devices: *Digital Signal Processing Applications using the ADSP-2100 Family*, Prentice-Hall, Englewood Cliffs, NJ, 1992.
- [9] E.A. Lee: "Programmable DSP Architectures: Part I & II," *IEEE ASSP Magazine*, pp. 4-19, October 1988 and pp. 4-14, January 1989.
- [10] J. Rabaey, et al.: "Synthesis of Datapath Architectures," in *Anatomy of a Silicon Compiler*, ed. by R.W. Brodersen, Kluwer, 1992.
- [11] K.K. Parhi: "Algorithm transformation technique for concurrent processors," *Proc. of the IEEE*, Vol. 77, No. 12, pp. 1879-1895, 1989.
- [12] P. Dewilde, E. Deprettere, R. Nouta: "Parallel and pipelined implementation of signal processing algorithms," in *VLSI and Modern Signal Processing*, ed. by S.Y. Kung, H.J. Whitehouse, T. Kailath, Prentice Hall, Englewood Cliffs, NJ, pp. 258-264, 1985.
- [13] R.A. Walker, R. Camposano: "A Survey of High-Level Synthesis Systems," Kluwer, Boston, MA, 1991.