Cache Performance of Data Flow/Control Flow Architecture

Joseph M. Arul and NanSheng Lin
Fu Jen Catholic University, Hsin Chuang 242, Taipei, Taiwan. R.O.C.
{arul, nat93j} @csie.fju.edu.tw

ABSTRACT

Instruction reference in data flow differs from that of conventional systems. In data flow environments, execution is data driven, hence it depends on data reference. Scheduled data flow is a unique architecture that uses a non-blocking multithread and decoupled access model based on data flow paradigm. The parallel nature found in data flow architectures and the dependence of execution on data cause the dimensional reference patterns more effective and complex than the conventional sequential execution. This research compares the cache performance of such hybrid architecture and the existing conventional architecture. By using non-blocking multithreaded model to divide the program into different code sections, data organizations and layouts provide an essential mechanism to improve the cache locality. Our evaluations, with several small benchmarks, demonstrate that the cache performance using different associativity -in most cases, significantly outperform conventional system.

1: INTRODUCTIONS

Processor performance has been doubling every 18 months (Moore's Law) while the performance of memory is increasing by only about 7% a year. Hence, the hardware technology trends have produced an increasing disparity between processor speeds and the memory access times in the recent computer development history. There have been several techniques proposed by many scholars to alleviate the gap and to tolerate the memory latency.

Many hardware and software techniques such as speculative execution [1, 2] multithreading, memory interleaving, non-blocking (lockup/free) [3, 4] caches, prefetching [3, 5], dynamic scheduling and victim cache [6] -all of them try to tolerate or reduce the memory accesses to improve the performance of overall CPU. Until recently, memory caches have been a promising and very effective method to reduce the disparity seen in this realm. Most processors now include a first-level cache, and some even include multi-level caches. All of them are designed to reduce the average data access time by capturing the most frequently used data items. The data hazards caused on the pipeline could further increase the complexity of multi-level caches leading to several miss cycles in case of a cache miss. Since the recent processors rely heavily on cache memories, more cache misses also would consume extra power [1, 7, 8, 9]. Thus, to reduce the power consumption by keeping cache misses as low as possible would be an important task if it is targeted to the embedded systems.

All these techniques have been successfully implemented and proved to improve the performance of the control flow or von Neumann architecture that has been widely used in the computer industry. Alternative to the conventional architecture is the data flow architecture which has been researched by many [10, 11] does not support locality, since the execution sequence is enforced by the availability of operands. The Scheduled Data Flow (SDF) architecture is a unique architecture that uses a non-blocking multithreaded and decoupled memory access model based on data flow paradigm. SDF tries to bring the data flow closer to control-flow model of execution to take advantages of all the benefits found in data flow as well as control-flow execution. Such hybrid execution models have been studied [12-14]. The detailed explanation of this unique architecture will be presented in section 3.

In this research we try to compare the cache performance of the traditionally used control-flow architecture and SDF. Even though few data flow machines have been built for evaluation and study purpose those are based on pure data flow machines and several hybrid machines [15, 17-19] using multithreaded techniques, rarely ever the comparisons of cache performance of these machines have been reported. Some have done analytical studies and have reported the results. The remainder of this paper is as follows. The second section of this paper will present the related works and background. In the third section we will present the details of this architecture. The fourth section will present the cache performance evaluation as compared to SDF. The final section will present the conclusion and the future work.

2: RELATED WORKS

In conventional architectures, reducing memory latency is achieved through (explicit) programmable registers and (implicit) high speed caches. However, adding caches or register-caches to the data flow framework could better exploit parallelism and hardware utilization [16]. Hurson et. al. have shown that data flow model of cache has shown unequaled effectiveness in improving system performance. Besides, they also present that the hybrid machine in which a

control-driven model executes intraprocess instructions, and a data-driven model executes interprocess communication and synchronization can be applied to data flow environment. They also point out that the cache memories, if designed properly may show great promise in data flow organization. Because, the program can reference an instruction block in more than one context in a data flow environment, a simple process count attached to each block can help effectively select suitable blocks for replacement.

In their article by Kavi et. al., cache memories in data flow architectures present that the data flow paradigm does not imply to the locality. However, data flow programs can be reordered to enhance the locality of instruction references. Further, they state that careful partitioning of a data flow program into vertical layers of data dependent recurrence portions can definitely enhance the locality of instruction references [18].

According to Takesue's article cache memories for data flow machines present that, to make the data flow caches practical, the active state processes (Na) in a Processing Element (PE) must be controlled. Besides, the author designed a load control mechanism which keeps Na near some threshold value with little performance degradation. Thus, a block structured set associative model is presented in this article. The research presents the idea that a data flow/von Neumann hybrid or mixed flow machine should be devised in which intra-process instructions can be executed by a control-driven model and interprocess communication and synchronization are executed by a data-driven model [19].

Thoreson et. al. present that the instruction reference patterns in data flow environments differ from those in conventional systems. In data flow machines, patterns of instruction references depend on data references. They presented the models of instruction reference patterns to illustrate program behavior and to provide insight into the potential usefulness of an instruction cache in data flow environments. Their results showed that locality exists and it is definitely exploitable in some data flow environments [20].

Chilimbi et. al. developed a cache-conscious structure layout to improve the cache locality of pointer-manipulating programs and as a result improve the performance. Their study demonstrated that careful data organization and layout could improve cache performance by increasing such pointer structure's temporal and spatial locality as well as reduce cache-conflicts. The idea of clustering improves spatial and temporal locality and it also provides implicit prefetching too. Coloring helps to map same cache block without incurring conflict misses. [22].

2.1: CACHE MEMORY FOR DATA FLOW ARCHITECTURE

Two principles that drive the cache performance are spatial and temporal locality. Spatial locality refers to virtual space adjacent to the last reference. If data m were referenced at time t, then the reference at time t+1 would be either from the previous (m-k) or the next (m+k). In a data flow environment, code may produce several spatial localities depending on the availability of data, since the data flow is a data driven model. In temporal locality t is a point in time and the set p is referenced during the interval (t-k) or t and it is likely that p will be referenced again at interval (t) or (t+k). Such recurrence use of some instructions is normally found in loops that help produce temporal locality. In data flow architecture, a processor allows a loop to unwind naturally, so that the initiation of successive iterations is constrained only by data dependencies. Thus, the above mentioned characteristics of temporal and spatial locality of data flow architecture present a good candidate for a better performance. The natural parallel paradigm found in data flow architecture and the dependence of execution on data cause the dimensional reference patterns more effective and complex than sequential environments. Hence, we compare the SDF based mainly on data flow paradigm with the sequential execution of programs. In the following section, we will present the Scheduled Data Flow architecture in detail including its pipeline, Synchronization Processor (SP) and Execution Processor (EP).

All the studies mentioned above point to the fact that the data flow architecture would definitely perform better in terms of cache memories due to the patterns that exist and are different from conventional architecture where the references depend on the data. Besides, a hybrid machine with a control flow and data flow would be a good choice to improve the cache performance. Hence, our research aims to show that the cache performance of such a hybrid machine such as SDF is comparable to conventional architectures. Since the cache performance of SDF shows fewer cache misses, it could be used in embedded systems too. Reducing cache misses would lead to less power consumption which would make SDF an ideal choice for embedded systems.

3: SCHEDULED DATA FLOW (SDF) EXECUTION MODEL AND CODE GENERATION

In this section, we describe how SDF executes a program. For the conventional computer, a program must be explicitly partitioned into procedures, modules, methods, etc. In our SDF model, a program must be partitioned into threads. For every code block, each block consists of a number of instructions that can be executed by SP or EP. Synchronization Count (SC) is the number of inputs needed for a thread before it can be scheduled for execution. Since we use non-blocking thread, a thread can be executed by SP or EP without interruption when the data has been loaded into its registers. When a thread is created, a frame memory is also allocated simultaneously. The data needed for the thread is stored into the related frame memory. During the preload phase,

the data is loaded from frame memory to its register context. When a thread completes its execution, if its results are needed for the consecutive thread or threads, they will be stored into the related frame memory. This is done in the post store section of the code block by SP. Once a thread's registers have the related data, the thread is put into EP. The EP executes without blocking or accessing memory. Thus, there is a clean separation of data access and execution, which make a decoupled architecture. In the following section we will present the SP and EP, before we explain the efficient method of storing data.

3.1: EXECUTION PIPELINE

The execution pipeline consists of four pipeline stages; instruction fetch, decode, execute and write back as seen in Figure 1. Instruction fetch unit behaves similar to a traditional fetch unit, which fetches the next instruction. Decode and register fetch units obtain a pair of registers that contain the two source operands for the instruction. Execute unit executes the instruction and sends the results to write-back unit along with the destination register numbers. Write back unit writes two values to the register file. In our SDF architecture, a pair of registers is viewed as source operands for an instruction.

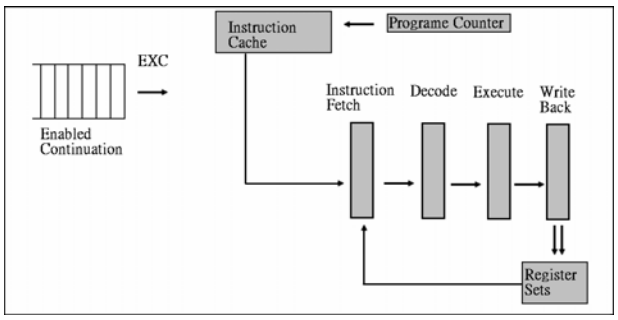

Figure 1: Execution Pipeline

3.2: SYNCHRONIZATION PIPELINE

The synchronization pipeline consists of five stages: instruction fetch, decode, memory access, execute, and write-back as seen in Figure 2. As mentioned earlier, the synchronization pipeline handles pre-load and post-store instructions. The instruction fetch unit retrieves an instruction belonging to the current thread using program counter (PC). The decode unit decodes the instruction and fetches registers. The effective address unit computes addresses for LOAD and STORE instructions. LOAD and STORE instructions only reference the frame memories of threads by using a Frame Pointer (FP) and an offset into the frames; both of which are contained in registers. The memory access unit completes LOAD and STORE instructions. Pursuant to a post-store, the synchronization count of a thread is decremented. The write-back unit completes LOAD (pre-load) and I-FETCH instructions by storing the values in appropriate registers.

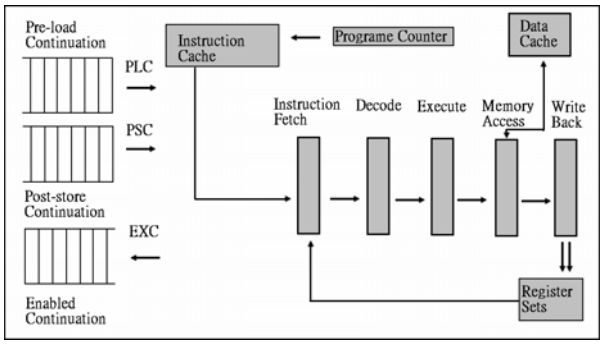

Figure 2: Synchronization Pipeline

3.3: EFFECT OF CACHE MEMORY FOR SDF

As for conventional architecture, increasing memory access instructions leads to increased cache misses, thus increasing the execution time. However, the decoupling of memory access and memory execution permits the multithreaded processors to tolerate the cache miss penalties. Our experimental result reveals that the SDF outperforms conventional architecture. The improvement primarily comes from the "pre-loading" and "post-storing" performed by SDF. The decoupling of memory accesses from instruction execution is more useful if the memory accesses can be grouped together (as done in this scheduled data flow architecture). In SDF, (note that only pre-load and post-store threads access the memory), assuming non-blocking caches, a cache miss does not prevent the memory accesses for other threads. The delays incurred by preloads and post-stores in SDF do not lead to additional context switches since threads are enabled for execution only when the pre-loading is complete, and once they are enabled for execution, they will complete without blocking. In SDF, a thread is a sequence of instructions with pre-load, execute and post-store code. Once the first instruction of the non-blocking thread is executed, the remaining thread executes without interruption. So the thread is the basic unit, which determines the synchronization and execution. Since all the data needed for a thread resides in a frame memory, cache memory should be designed to accommodate frames of active threads. In the following section we will present the results of running several benchmarks on conventional architecture as well as on SDF.

4: EXPERIMENT RESULTS

For the experimental results shown on this section, we used Red Hat Linux 9.0 with kernel 2.4.20-8. The benchmarks used for comparisons were Livermore loop5, merge sort, matrix multiplication, Minimum Spanning Tree (MST), and saddle point. We also used SDF simulator (SDFSIM) written in C++ language and SDF assembler (SDFASM) written in Lex and Yacc. Both cache performance analysis were done using DineroIV [21]. For the cache performance analysis, different

settings (Table 1) were used to present the final outcome of the results.

Cache classification	Replacement policy	Block size	Associate	Cache size
Unified	LRU	32	2/4	1K/2K/4K/8K
Separated				1K/2K/4K/8K for Instruction and Data cache

Table 1: Dinero Iv Setting to Collect Cache Performance

Table 2 presents the results of testing Livermore loop5, which uses three loops with the data size of 1000. From Table 2 we can note that the total fetches (195167) in SDF program are more than those of C program (129282), as the execution is done in von Neumann architectures. However, the miss rate for data flow (0.67%) is significantly lower than that of the control flow (4.06%) architectures using unified cache either associativity 2 or 4. When we use separate caches for instruction and data, we see that the cache miss rate of instruction cache is much less for SDF compared with control flow architecture. However, the data cache of SDF architecture is little bit higher than the normal architectures. SDF uses I-structure which is the characteristic of data flow architecture to store data. The special characteristic of I-structure is - write ones, read multiple times. If the cache size is rather small, the cache miss rate is higher for SDF architecture. If the cache size is larger, then the cache miss for SDF is still less than control flow architectures. When associativity 2 is used, the cache performance of SDF is only about 0.67% whereas the control flow architecture is about 4.06% in case of unified cache memory using 8 Kbytes. In the case of associativity 4, still the SDF cache performance is 0.69% and the control flow architecture is 2.45%. We can note the significant reduction in the cache performance of SDF architecture. Since, SDF architecture uses multithreaded paradigm, it groups well the instructions and data leading to good locality.

Loop5 Cache Miss Rate in Percent							
Cache Type		Unified Cache		Separate Cache			
				Instruction		Data	
		SDF 195167	C Flow 129282	SDF 146138	C Flow 95380	SDF 49029	C Flow 33902
Assoc 2	1 KB	3.2039	12.6793	0.0192	6.0117	11.4993	19.0254
	2 KB	2.9882	9.3068	0.0192	2.9272	11.4993	11.1822
	4 KB	2.9160	5.5870	0.0192	1.1617	11.4891	7.9907
	8 KB	0.6687	4.0609	0.0192	1.0369	1.8948	5.2475
Assoc 4	1 KB	2.9047	11.2684	0.0192	6.1795	11.4993	17.7807
	2 KB	2.9042	9.0562	0.0192	1.6031	11.4993	10.5097
	4 KB	2.9021	4.5923	0.0192	1.1187	11.4912	5.5365
	8 KB	0.6902	2.4381	0.0192	0.9918	2.0580	4.5396

Table 2: Comparison of Livermore Loop 5 Using Different Cache Sizes.

The second program is a merge sort where two arrays of 256 are sorted and merged together to present the final result. Table 3 also presents the similar performance as the previous program. In the case of unified cache with the similar associativity of 2 and 4 we see the similar performance as seen in the previous program. In the case of separate caches for instruction and data, we see that the instruction cache miss is similar to the previous result. When we observe the separate caches for instruction and data, we see that the data cache is worse than that of the instruction cache. The similar results between loop5 and merge sort is due to the I-structure nature present in both programs. It is also clear that the number of fetches in the case of unified cache for SDF architecture are higher than the numbers in control flow architecture. In SDF architecture the total fetches are about 15 million. Whereas in the control flow architecture, the total fetches are not even close to one million. In spite of the high number of fetches, the miss rates are very low. It again verifies that the data flow architecture presents a good way to get a better cache performance as compared with the existing architectures.

Merge Sort Cache Miss Rate in Percent							
Cache Type		Unified Cache		Separate Cache			
				Instruction		Data	
		SDF 15435491	C Flow 131565	SDF 12264623	C Flow 97034	SDF 3170868	C Flow 34531
Assoc 2	1 KB	4.1448	12.0906	0.0004	5.9113	10.4478	17.5697
	2 KB	2.6968	9.0351	0.0004	2.8804	8.4980	10.6397
	4 KB	1.1825	5.3738	0.0004	1.1429	4.4454	7.2254
	8 KB	0.2677	3.3687	0.0004	1.0182	0.2739	4.9868
Assoc 4	1 KB	2.8492	11.1048	0.0004	6.0752	9.0862	17.3294
	2 KB	1.9776	8.8785	0.0004	1.5768	8.5157	9.3539
	4 KB	1.2815	4.6943	0.0004	1.1316	5.1975	5.3778
	8 KB	0.1514	2.4125	0.0004	0.9759	0.1757	4.4076

Table 3. Comparison of Merge Sort.

Table 4 shows the results of running matrix multiplication program of size 100x100 by using I-structure to store the X, Y and Z matrices. For the unified cache, total number of fetches is about 14 million. However, in the case of control flow architecture, the number is far lower than a million. We can note that the cache miss rate, in the case of 8 Kbytes are about 1.11% and 1.34%. However, the control flow architecture, the numbers are 2.76% and 2%. In the case of smaller cache size, the SDF cache miss rate is less than half of control flow architecture. In the case of separate caches for SDF, compared with the control flow, it has similar results as of the previous programs presented.

Matrix Multiplication Cache Miss Rate in Percent							
Cache Type		Unified Cache		Separate Cache			
				Instruction		Data	
		SDF 14705575	C Flow 185459	SDF 11621711	C Flow 140785	SDF 3083864	C Flow 44674
Assoc 2	1 KB	3.8794	8.8251	0.0104	4.0089	13.7451	14.8767
	2 KB	3.1027	6.6645	0.0007	1.9583	11.5663	9.9118
	4 KB	2.5268	3.8278	0.0007	0.7735	10.5094	6.2676
	8 KB	1.1049	2.7596	0.0007	0.6897	3.2776	5.1641
Assoc 4	1 KB	3.6084	8.0460	0.0130	4.1389	14.0215	14.4133
	2 KB	3.1339	6.6198	0.0007	1.0676	11.4473	8.4837
	4 KB	2.3324	3.5210	0.0007	0.7693	10.4141	5.7841
	8 KB	1.3423	2.0042	0.0007	0.6613	4.3849	4.8328

Table 4. Comparison of Matrix Multiplication.

Table 5 presents the cache performance of saddle point program running on SDF and control flow

architecture. The saddle point program uses 100x100 size of matrix arrays. Saddle point is an element in matrix. Which means that its value is the biggest in a row and the smallest in a column? Thus, the program identifies the row and the column that the element belongs to. In table 5 we can note that the total number of fetches in the case of unified cache is about 4 million. With regard to control flow it is only 129,270 fetches. SDF architecture shows high number of fetches but very low cache miss rate as compared to control flow architecture. This confirms again that the SDF architecture with its data flow paradigm would be a good approach to achieve better performance as compared to control flow architecture. The SDF architecture's engine relies on control flow like instruction scheduling. For 2 and 4 way set associativity cache for 8 Kbytes the miss rates of SDF is only about 0.81% and 0.96%. The rate for control flow architectures are about 3.45% and 2.43%. Together with multithreaded, data flow architecture performs very well under different set associativity for various cache sizes.

Saddle Point Cache Miss Rate in Percent							
Cache Type		Unified Cache		Separate Cache			
				Instruction		Data	
		SDF 4013934	C Flow 129270	SDF 2977903	C Flow 95371	SDF 1036031	C Flow 33899
Assoc 2	1 KB	3.8990	12.7717	0.1423	6.0134	11.1176	18.1480
	2 KB	3.1743	9.3084	0.0024	2.9286	10.8453	10.6611
	4 KB	2.7827	5.3593	0.0024	1.1597	10.0981	7.6108
	8 KB	0.8090	3.4540	0.0024	1.0370	1.6694	4.9382
Assoc 4	1 KB	4.2184	11.5820	0.1523	6.1801	10.9845	17.7380
	2 KB	2.9119	9.2721	0.0024	1.6043	10.8809	9.3543
	4 KB	2.7046	4.3351	0.0024	1.1492	10.1136	5.5931
	8 KB	0.9585	2.4329	0.0024	0.9919	1.8541	4.4928

Table 5. Comparison of SDF and Control flow Running Saddle Point Program.

MST Cache Miss Rate in Percent							
Cache Type		Unified Cache		Separate Cache			
				Instruction		Data	
		SDF 7504063	C Flow 128609	SDF 5587074	C Flow 94890	SDF 1916989	C Flow 33719
Assoc 2	1 KB	3.9615	12.4082	0.5549	6.0164	9.4174	18.3072
	2 KB	2.9182	8.9177	0.1704	2.9424	8.8532	11.0205
	4 KB	1.8643	5.8192	0.0406	1.1708	5.4777	8.4759
	8 KB	0.5248	3.4407	0.0037	1.0423	0.5358	4.9883
Assoc 4	1 KB	3.6893	11.1610	0.5831	6.2125	9.2359	17.9068
	2 KB	2.7554	9.0219	0.1807	1.6134	9.0832	9.1491
	4 KB	1.9373	4.7096	0.0300	1.1550	5.5334	5.7119
	8 KB	0.4442	2.5301	0.0037	0.9969	0.4829	4.6413

Table 6. Comparison of SDF and Control Flow Architecture running Minimum Spanning Tree.

Table 6 presents the results of Minimum Spanning Tree (MST) used in several applications. The MST program we used to test the cache performance includes 100 nodes and 318 edges; all of the edges have been sorted by the weight of the edges. As can be seen in table 6, for the unified cache, miss rate of SDF is lower than control flow architecture. Total amount of fetches for SDF is about 7.5 million. The control flow architecture fetches are 128,609. However, the cache miss rate of SDF is 0.53% as compared with control flow's 3.44% miss rate for set associativity of 2 and 8 Kbytes of cache size. Similarly, for 4 set associativity cache and a cache size of 8 Kbytes miss rate is about 0.44% for SDF and 2.53% for control flow architecture.

Table 7 presents the results of cache misses running saddle point and matrix multiplication programs. Both programs use I-Structure memory. The experiments were conducted with separate data cache and I-Structure cache. The table reveals that the I-Structure cache behavior is rather poor. We observe that for the saddle point program, the data cache performs better in the case of 2-way set associativity for all different cache sizes. Similarly, for the matrix multiplication program, we observe the same result. Even though the results of table 7 may not be positive compared with the previous results, we show that the cache behavior of I-structure cache can cause some problem compared to the unified cache.

The parallel nature of data flow processors and the data dependence nature of execution on data cause two dimensional reference patterns rather than one dimensional nature found in the control-flow architecture. Due to the nature of data flow environment in SDF, it produces good spatial locality improving cache misses seen in data flow architecture. In data flow architecture, instructions are reused, which makes it a good choice for using temporal locality.

Cache Miss Rate in Percent					
Cache Type		Saddle Point		Matrix Multiplication	
		Data 986031	I-Structure 50000	Data 2826364	I-Structure 257500
Assoc 2	1 KB	10.2519	28.1900	10.1086	53.6602
	2 KB	10.0916	25.7100	10.0770	27.9138
	4 KB	9.3759	24.3400	9.5642	20.8850
	8 KB	0.5902	22.9520	2.0138	17.1495
Assoc 4	1 KB	10.2741	24.9940	10.1087	56.9689
	2 KB	10.1704	24.8920	10.1082	26.1596
	4 KB	9.4037	24.1140	9.6196	19.1355
	8 KB	0.8017	22.6080	3.2489	16.8536

Table 7. Cache Miss Ratio for Separate Data Cache from I-structure Memory.

5: CONCLUSION

All the benchmark results show that the SDF architecture's unified cache miss rate is certainly lower than that of conventional architecture. It is mainly due to the effective way of partitioning of the data based on data driven model as well as non-blocking multithreaded method. This research presents the application of the instruction, operand and I-structure cache memories within the scope of the non-blocking multithreaded dataflow system. Cache design issues have been discussed and compared based on running real applications. The experiments show that while all cache memories contribute to performance gains in dataflow machines, I-structure cache memories produce not much improvement. When I-structure memory is separated from instruction caches, different mapping could be used in multiprocessor system. The unique nature of I-structures write ones including read multiple times can

cause several misses. An important issue in multithreading is the partitioning of programs into several sequential threads.

In SDF architecture, a non-blocking thread defines the granularity of a computation and thus it is the basic unit for scheduling. When we consider the partitioning of a program into small number of threads, instructions that are grouped into a thread should be the parts of a code where little or no exploitable parallelism exists. This certainly increases the spatial locality and hence the utilization of resources is increased. In the future, we plan to study the cache performance of multithreaded programs using pthread, openMP, and Java threads. A program can be executed by using different number of threads. What impact does it have on cache performance? The SDF architecture presented here could be implemented on embedded systems which give fewer cache misses leading to low power-consumption that is necessary in such systems.

REFERENCES

[1] Musoll, E., "Speculating to reduce unnecessary power consumption," Trans. on Embedded Computing Sys. 2. 4, pp. 509-536, Nov. 2003.

[2] Vijaykumar, T. N., Gopal, S., Smith, J. E., and Sohi, G., "Speculative Versioning Cache," IEEE Trans. Parallel Distrib. Sys. 12. 12, pp. 1305-1317, Dec. 2001.

[3] Chen, T. and Baer, J., "Reducing memory latency via non-blocking and prefetching caches," In Proceedings of the Fifth international Conference on Architectural Support For Programming Languages and Operating Systems, Oct. 1992.

[4] Oner, K. and Dubois, M., "Effects of memory latencies on non-blocking processor/cache architectures," In Proceedings of the 7th international Conference on Supercomputing, July 1993.

[5] Solihin Y., Lee J., and Torrellas J., "Prefetching in an Intelligent Memory Architecture Using a Helper Thread," Proceedings of Workshop on Multithreaded Execution, Architecture and Compilation (MTEAC-5), Dec. 2001.

[6] A. Naz, M. Rezaei, K. Kavi, and P. Sweany, "Improving data cache performance with integrated use of split caches, victim cache and stream buffers," MEDEA 2004, Antibes Juan-Les Pins, France, 2004.

[7] T. D. Givargis, F. Vahid and J. Henkel, "Evaluating power consumption of parameterized cache and bus architectures in system-on-a-chip designs," IEEE Trans. Very Large Scale Integr. Syst. 9. 4, pp. 500-508, Aug. 2001.

[8] Badulescu, A., "Power Efficient Instruction Cache for Wide-issue Processors," In Proceedings of the innovative Architecture For Future Generation High-Performance Processors and Systems, Jan. 2001.

[9] Chen H. C. and Chiang J. S., "Design of An Adjustable-way Cache for Energy Reduction," Journal of The Chinese Institute of Engineers , Vol. 28, pp. 691-700, 2005.

[10] Deepika S., Kot L., "Dataflow Architectures," Digital Image Computing and Applications 97 in New Zealand, 1997.

[11] Veen, A. H., "Dataflow machine architecture," ACM Comput. Surv. 18. 4, pp. 365-396, Dec. 1986.

[12] Arvind and Nikhil R.S., "Can Dataflow subsume von Neumann Computing?," Proc. 16th Annl. Intl. Symp. on Computer Architecture, pp. 262-272, 1989.

[13] D.E. Culler et. al., "TAM - a compiler controlled threaded abstract machine," Journal of Parallel and Distributed Computing, 18 (3), pp. 347-370, 1993

[14] R.A. Ianucci, "Toward a dataflow/von Nuemann Hybrid Architecture," Proc. 15th Annul. Intl. Symp. on COmputer Architecture, pp. 131-140, 1988.

[15] P. Shanmugam, S. Andhare, K. Kavi, B. Shirazi and A.R. Hurson, "Cache memory for an explicit token store dataflow architecture," Proceedings of the 5th IEEE Symposium on Parallel and Distributed Processing, pp. 45-50, 1992.

[16] Hurson, A. R., Kavi, K. M., Shirazi, B., and Lee, B., "Cache Memories for Dataflow Systems", IEEE Parallel Distrib. Technol. 4. 4, pp. 50-64, Dec. 1996.

[17] K.M. Kavi and A.R. Hurson. "Cache Memories in Dataflow Architecture", Proceedings of the 7th IEEE Symposium on Parallel and Distributed Processing, pp. 182-189, Oct. 1995.

[18] Kavi K.M., Hurson A.R., "Design of Cache Memories for Dataflow Architecture," Journal of Systems Architecture, Volume 44, Number 9, pp. 657-674, June 1998.

[19] M. Takesau, "Cache Memories for Dataflow Machines," IEEE. Trans. Computers, Vol. 41,. No. 6, pp. 677-687, June 1992.

[20] Thoreson, S. A. and Oldehoeft, A. E., "Instruction reference patterns in data flow programs," In Proceedings of the ACM 1980 Annual Conference ACM '80. ACM Press, New York, NY, pp. 211-217, 1980.

[21] J. Edler and M. D. Hill, "Dinero IV Trace-Driven Uniprocessor Cache Simulator," 1998.

[22] Chilimbi, T. M., Hill, M. D., and Larus, J. R., "Cache-conscious structure layout," SIGPLAN Not. 34. 5, pp. 1-12, May 1999.Roman. When citing within the text, enclose the citation number in square brackets (for example, [1]).