Scalability of Scheduled Dataflow (SDF) Architecture With Register Context

周　賜 福
Joseph Arul

Department of Computer Science and Information Engineering
資訊工程學系
Fu Jen Catholic University, 輔仁大學
91年**5**月**16**日

Background

Introduction to von Neumann and Dataflow Models.

Multithreading Concepts (Blocking vs Non-Blocking Thread)

Decoupled Architecture.

Unique Features of the Scheduled Dataflow Architecture.

Scheduled Dataflow (SDF)Architecture

Example Graph in MIPS(Conventional) and SDF.

Pre-load and Post-store Code for the thread.

Thread Code portions for SP and EP.

Synchronization Processor (SP) and Execution Processor (EP).

Performance Evaluation and comparison

Comparison of SDF with Superscalar architecture (Simplescalar).

- One unit versus multiple units and different data sizes.

Comparison of SDF with VLIW architectures.

- Trimaran and TMSC6000 –TI.

Summary and Conclusions

Unique Contributions of SDF

Future work.

Control Flow vs. Dataflow Models of Architectures.

Control Flow or von Neumann model:

In this type of architecture the execution of the instructions is controlled by the program counter and executes sequentially unless the sequence is changed by a jump, branch, interrupt or return.

Dataflow Model:

The application program is translated into a dataflow graph (directed graph consisting of named nodes and arcs). Nodes represent instructions and arcs represent data dependencies among instructions.
The execution is data dependent as opposed to control driven.
It eliminates the need for program counter.

Multithreaded architecture.

In a single threaded machine, program execution is defined by the processor state.

Processor state: Memory state (program memory,
data memory, stack),
Activity specifier (program counter and
stack counter) and
Register Context (set of registers).

Context of thread: Activity specifier and
Register context.

(can hide memory latency, long I/O latency or can be used to interleave instructions on a cycle-by-cycle basis.)

Blocking versus Non-Blocking Multithread:

Blocking: A thread can block during its execution. (Memory accesses, cache miss, synchronization)

Drawback: There could be too many context switch leading to smaller performance gain.

Non-Blocking: A non-blocking thread proceeds to evaluation as soon as all the inputs are available. It completes execution without blocking the processor pipeline.
(Basically thread switching is controlled by the compiler with the idea of creating new threads rather than blocking).

Drawback: It could lead to creating of many small threads.

What is decoupled Architecture?

Memory Access times and I/O bandwidth have not improved compared to processor clock rate.
The gap seems to be widening.

Decoupled: To separate the operand accesses from execution.
Both instruction streams are executed on independent processing units

(Accessing processor AP and Execution processor EP)

What is Schedule Dataflow (SDF) Architecture?

- It is a decoupled non-blocking multithreaded scheduled dataflow architecture.

- The instructions are scheduled for execution.

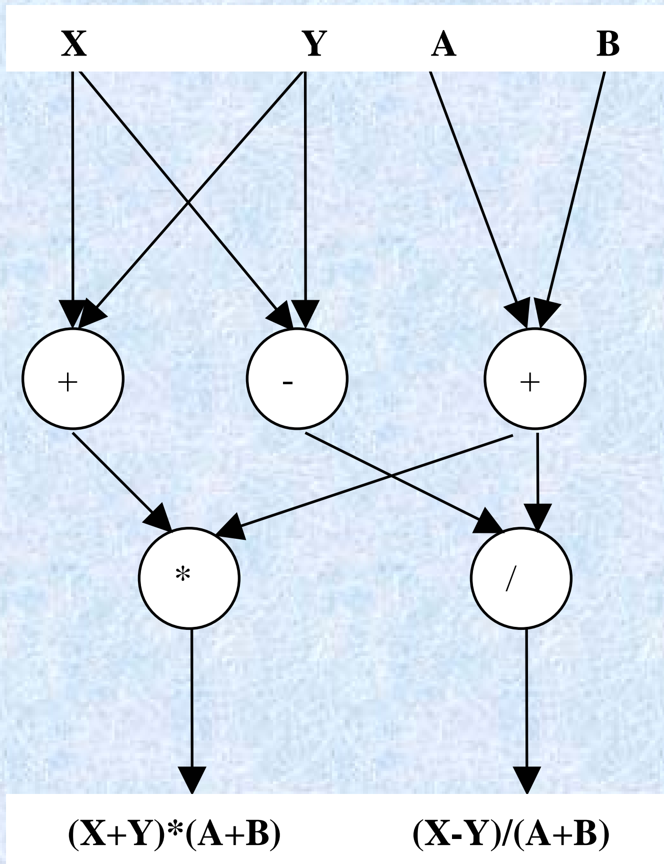
- It is not data driven.

- The compiler partitions the program into non-blocking threads.

Features of Scheduled Dataflow (SDF) Architecture:

1. **Instruction** driven rather than the token driven.
2. Instructions retain **'functional'** properties of dataflow.
3. Instructions are executed **synchronously** like control flow.
4. Memory accesses are completely **decoupled** from execution pipeline.
5. **Non-blocking** multithreaded model is used (A thread is enabled when all inputs are received. Once enabled, thread executes to completion without blocking or context switching).

Representation a dataflow graph in a MIPS like Architecture



MIPS like Instructions:

1.	LOAD	R1, X	/Load X in R1
2.	LOAD	R2, Y	/Load Y in R2
3.	LOAD	R3, A	/Load A in R3
4.	LOAD	R4, B	/Load B in R4
5.	ADD	R10, R3, R4	/R10=A+B
6.	ADD	R11, R1, R2	/R11=X+Y
7.	SUB	R12, R1, R2	/R12=X-Y
8.	MUL	R14, R11,R10	/R14=(X+Y)*(A+B)
9.	DIV	R15, R12,R10	/R15=(X-Y)/(A-B)
10.	STORE	OUT, R14	/First Result
11.	STORE	OUT, R15	/Second Result

Representatin of the dataflow in SDF

```
code main
 LOAD  RFP|2,   R2                        ; load A into R2
 LOAD  RFP|3,   R3                        ; load B into R3
 LOAD  RFP|4,   R4                        ; load X into R4
 LOAD  RFP|5,   R5                        ; load Y into R5
 LOAD  RFP|6,   R6                        ;load frame pointer for returning  1st result
 LOAD  RFP|7,   R7                        ;load frame offset for returning  1st result
 LOAD  RFP|8,   R8                        ;load frame pointer for returning 2nd  result
 LOAD  RFP|9,   R9                        ;load frame offset for returning 2nd result
 PUTR1 submain
 FORKEP R1
 STOP
submain:  ADD    RR2,     R11,    R13     ; Compute A+B, Result in R11 and R13
          ADD    RR4,     R10             ; Compute X+Y, Result in R10
          SUB    RR4,     R12             ; Compute X-Y, Result  in R12
          MULT   RR10,    R14             ; Compute (X+Y)*(A+B), Result in R14
          DIV    RR12,    R15             ; Compute (X-Y)/(A+B), Result in R15
          PUTR1 finemain
          FORKSP R1
          STOP
finemain: STORE R14,    R6|R7             ; Store first result
          STORE R15,    R8|R9             ; Store second result.
          FFREE
          STOP
```

Data for the thread.

Data for the thread can come from Frame Memory or I-structure.
I-structure are special memory to store the data such as array or structures and so on.

When a thread completes its execution, it stores the result in the frame memory and the next thread loads from the Frame memory to the registers.

I-Structure Memory operation in SDF

IALLOC: Opcode to allocate a given number of elements for a new I-structure and the pointer is stored in the destination register. Ex: IALLOC R5, R8

IFREE: Opcode to free the I-structure memory pointed to by the given I-structure pointer. Ex: IFREE R8

IFETCH: Opcode to retrieve the specified content of the element and store in the destination registers.
(If the element is not yet written, it is deferred.)
Ex: IFETCH RR8, R15 or IFETCH R8|15, R15

ISTORE: Opcode to store the value into the specified I-structure element .
(if the element is already present, an error condition is returned.)
Ex: ISTORE R15, R8|R9 or ISTORE R15, R8|20

Pre-Load and Post Store

Group all LOAD instructions together at the beginning of the thread.

- Pre-load thread's data into registers before scheduling for execution.

During execution the thread does not access memory.

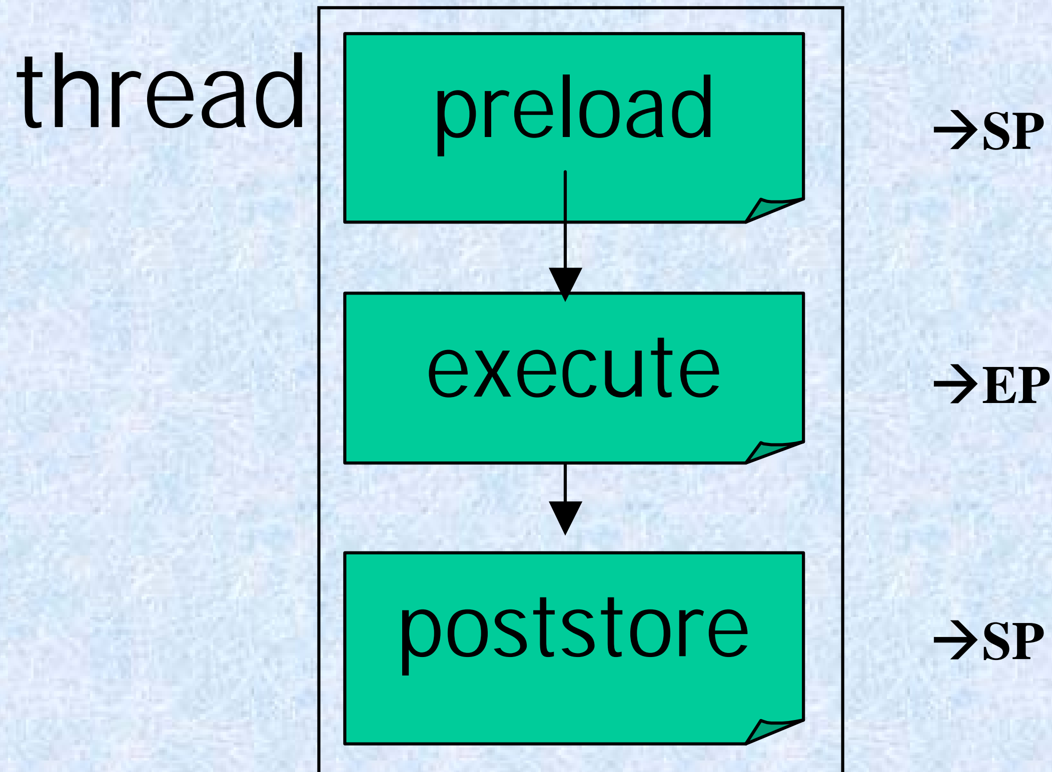
Group all STORE instructions together at the tail of the thread.

- Post-store thread results into memory after thread completes execution.

(Data may be stored into awaiting Frames.)

Non blocking model facilitate clean separation of memory Accesses into Pre-load and Post-store.

Thread code portion of SP and EP

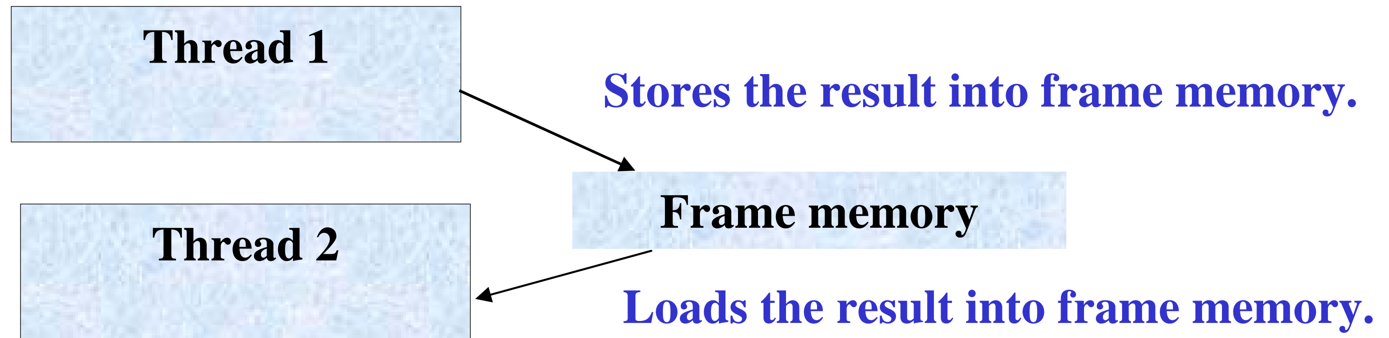


SP:Synchronization Process
EP:Execution Process

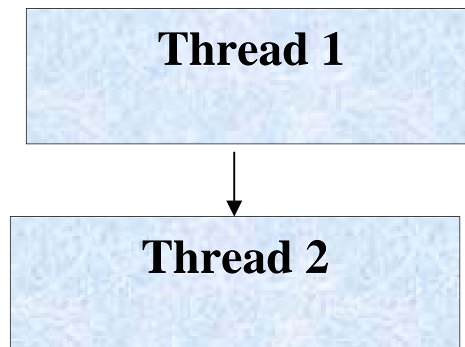
Two different ways of passing of data to a thread.

Method 1.

When a thread completes it's execution, it stores the result into a frame memory. Next thread loads the data from the frame memory.



Method 2.First thread stores the result into the second thread if the register set is available for the next thread.



Avoid the second thread's loading.

Need enough register sets.

Scalability of the Scheduled dataflow depends on the register sets

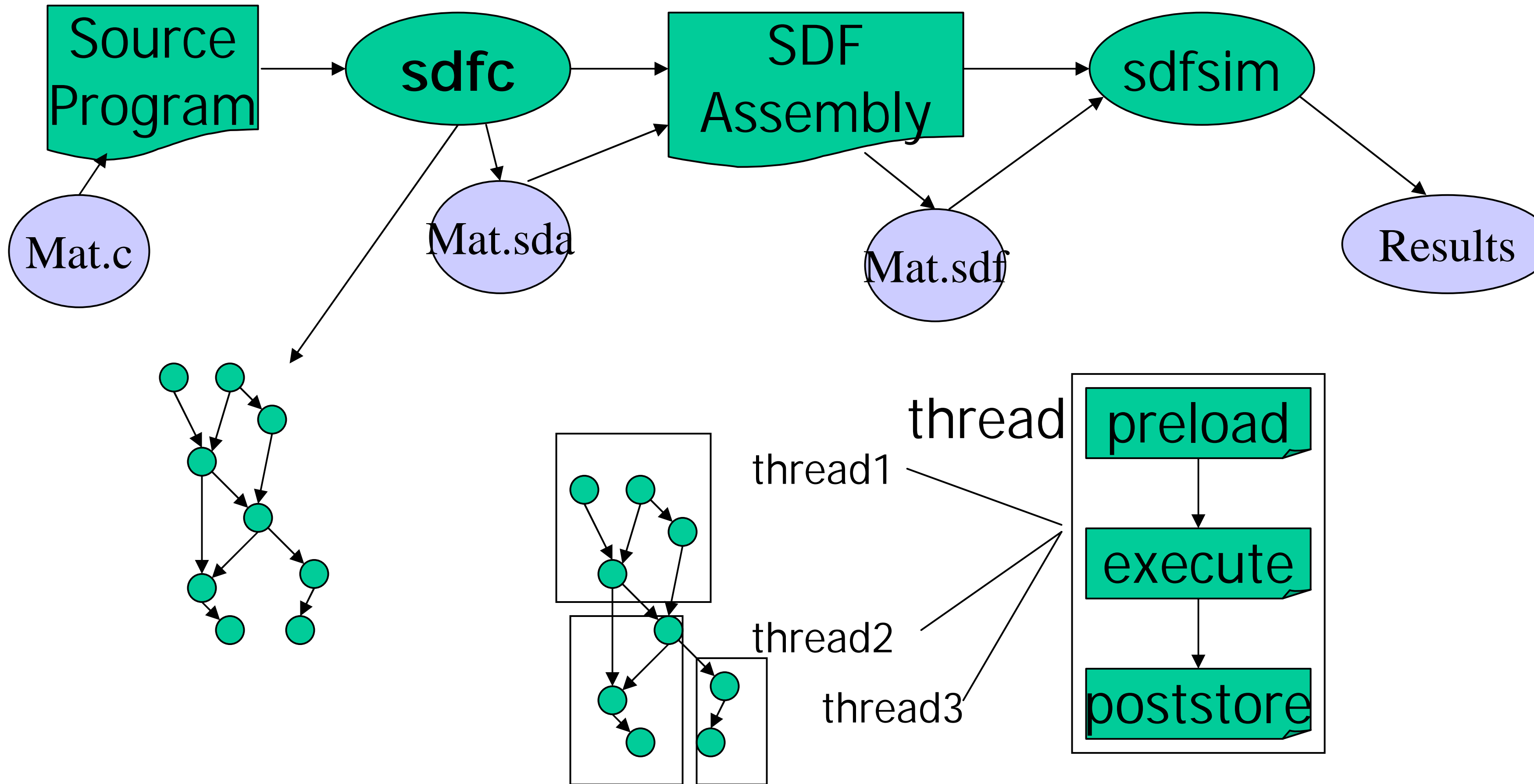
If we have more register sets, then we can have more threads spawned simultaneously. The results can be passed from one thread to the other without going through the frame memory.

When we have more threads simultaneously, the parallelism also improves. Thread granularity also can be improved and so the overall performance can be improved.

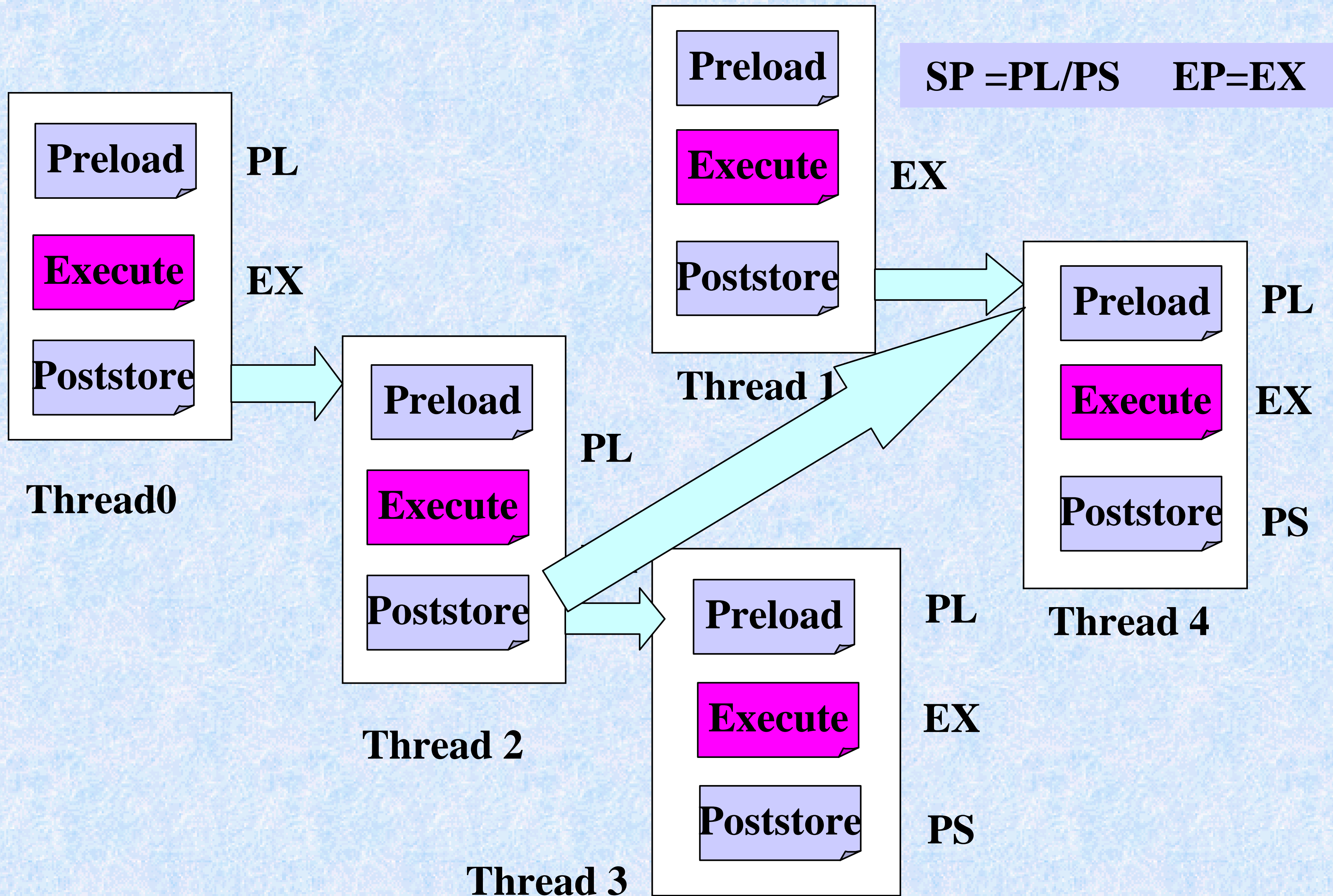
Having more threads simultaneously, improves the cache performance of the architecture since the thread data is handled on the registers avoiding of loading from the memory.

Since the hardware complexity is less, we can use that extra hardware savings to allocate more register sets.

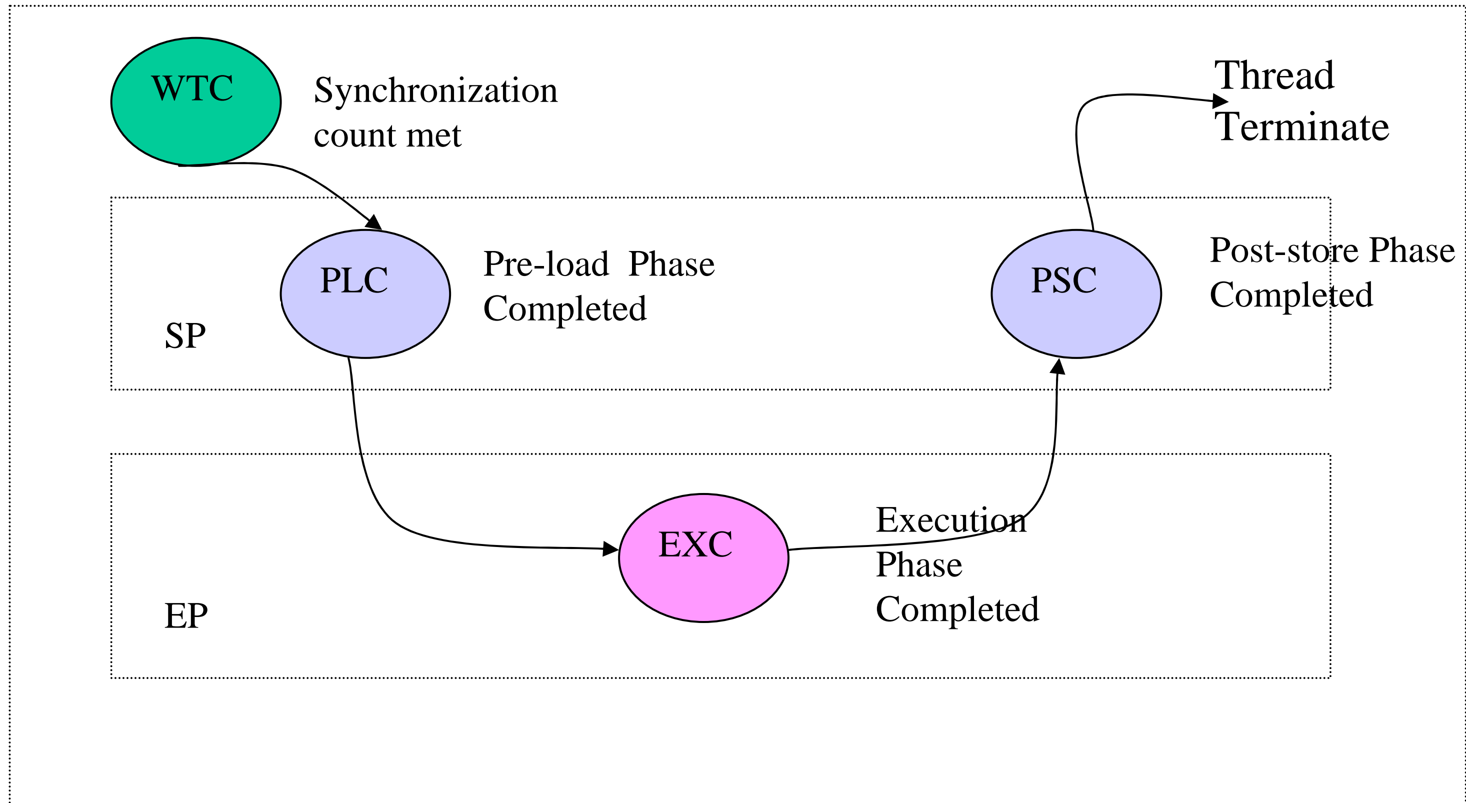
Scheduled Dataflow Procedure



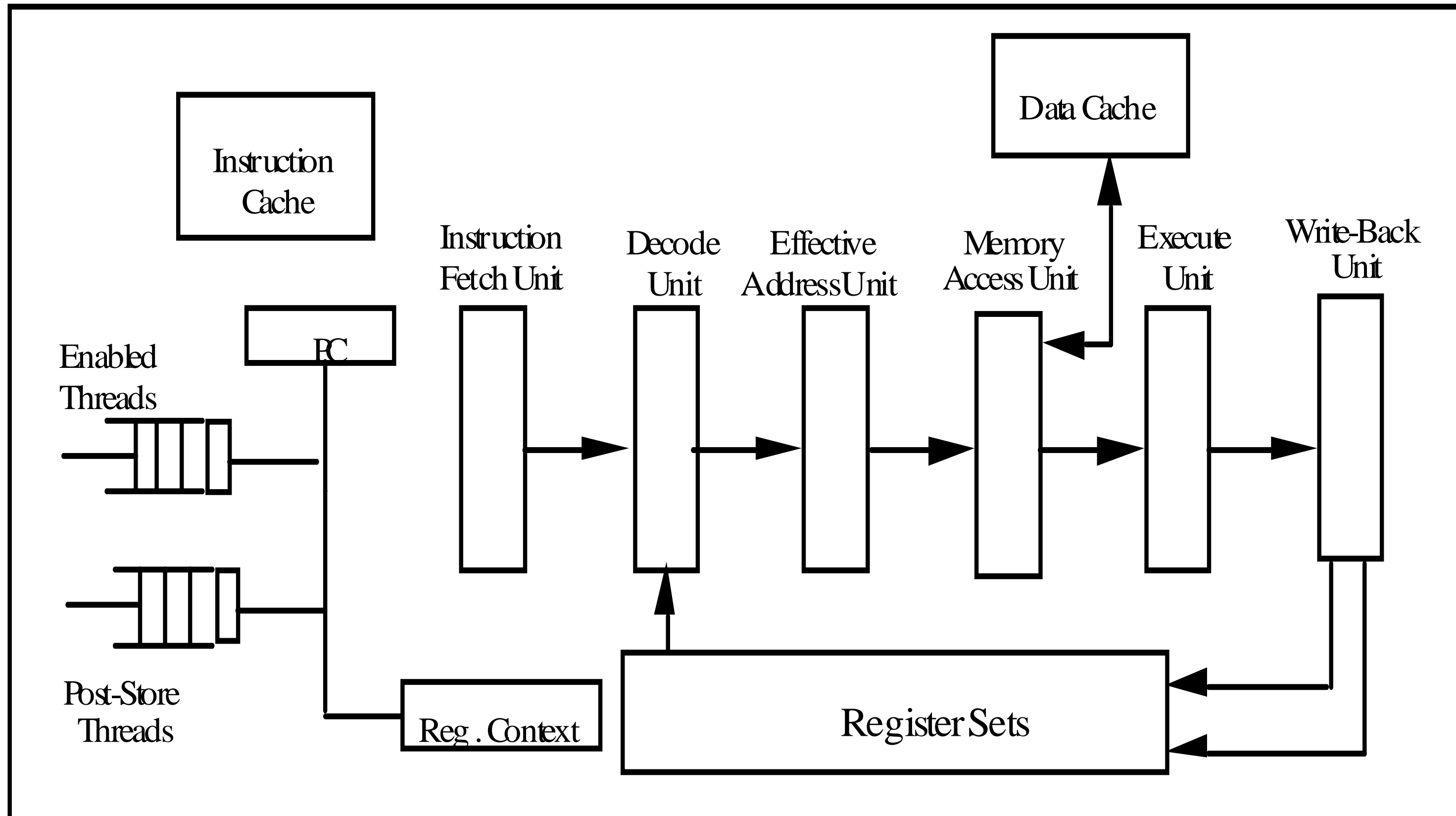
Execution of SDF Programs



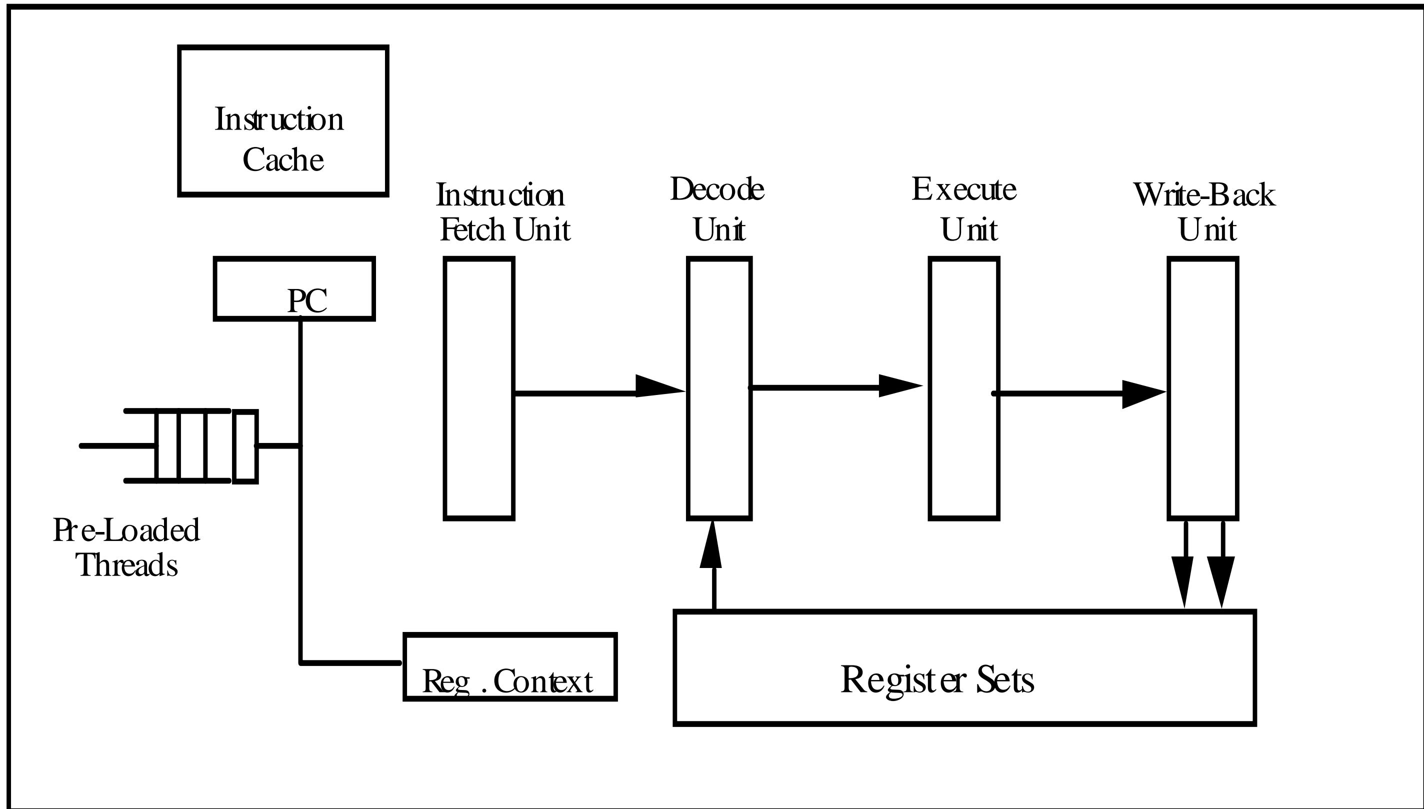
Continuation Transition



Synchronization Processor (SP)



Execution Processor (EP)



Benchmark Programs

Primarily Scientific and DSP oriented programs:

Fibonacci Program (Recursive in nature).

Matrix Multiplication

FFT

Picture zooming Program

Livermore Loop 5

Most of the data presented here would be based on the first method of allocating threads.

- Compared SDF with MIPS like single threaded architecture.
- Compared SDF with Superscalar (Simplescalar).
- Compared SDF with Trimaran and TMSC6000.

SDF Versus MIPS

SDF: Used hand coded programs, sdfasm (assembler) and sdfsim(instructin set simualtor) to obtain cycle count.

MIPS: Used dlxcc (compiler) and dlxsim (simulator) to obtain cycle count.

Compared the results for varying:

- **-problem sizes.**
- **-thread granularity (by increasing thread size by unrolling of loops).**
- **-thread parallelism (by increasing the number of simultaneously forked threads).**
- **-Utilization of SP and EP on SDF.**

SDF versus MIPS

Matrix Multply

N	DLX Cycles	SDF Cycles	Speed UP
25*25	966090	306702	3.150
50*50	7273390	2159780	3.368
75*75	24464440	6976908	3.506
100*100	57891740	16175586	3.579

Livermore 5

Loop=N	DLX Cycles	SDF Cycles	Speed Up
50	87359	56859	1.536
100	354659	215579	1.645
150	801959	476299	1.684
200	1429259	839019	1.703
250	2236559	1303739	1.715
300	3223859	1870459	1.724
350	4391159	2911789	1.508
400	5738459	3309899	1.734
450	7265759	4182619	1.737

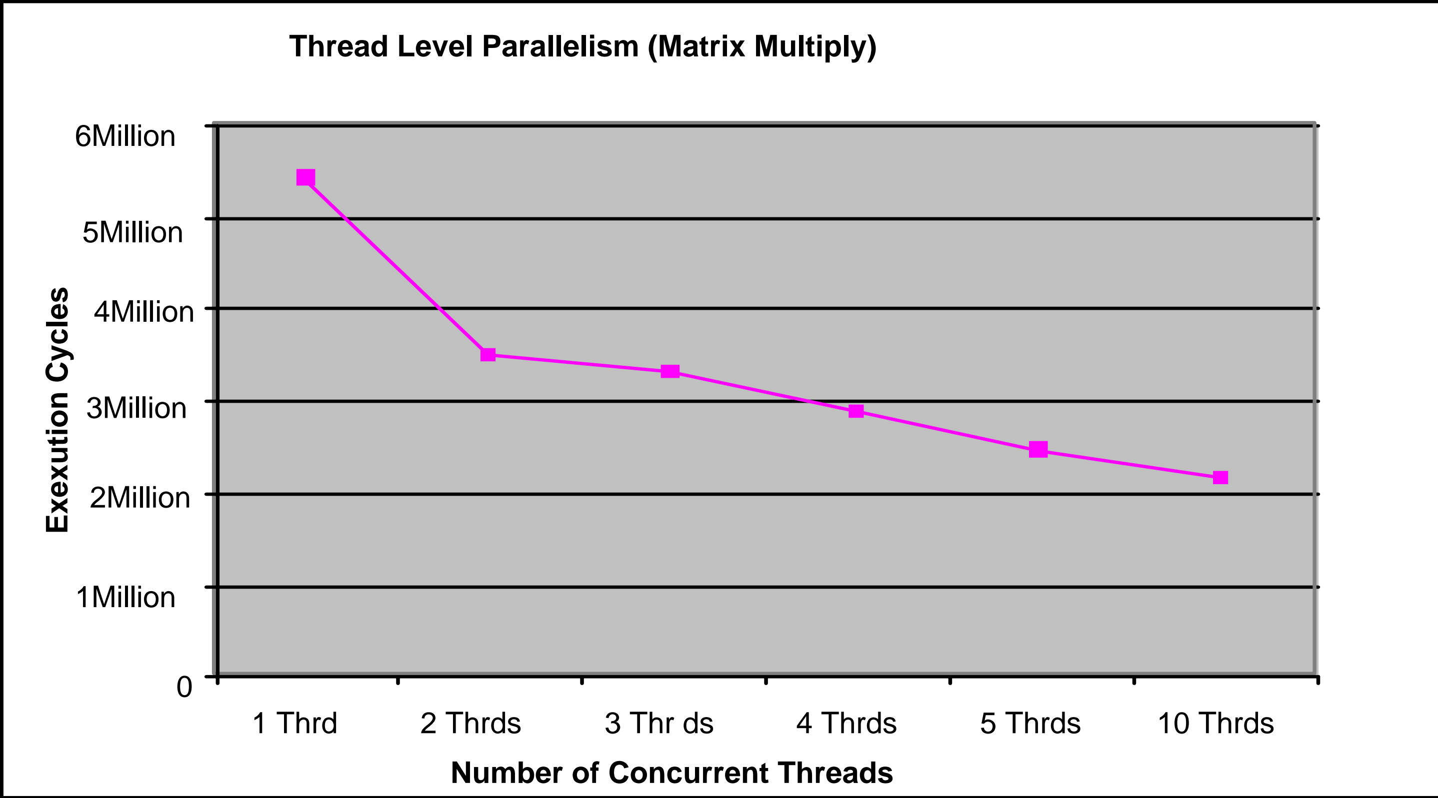
Fibonacci

N	DLX Cycles	SDF Cycles	Speed UP
5	615	842	0.7304
10	7014	10035	0.699
15	77956	111909	0.6966
20	864717	1241716	0.6964
25	9590030	13771467	0.6964
30	1.06E+08	1.53E+08	0.6964

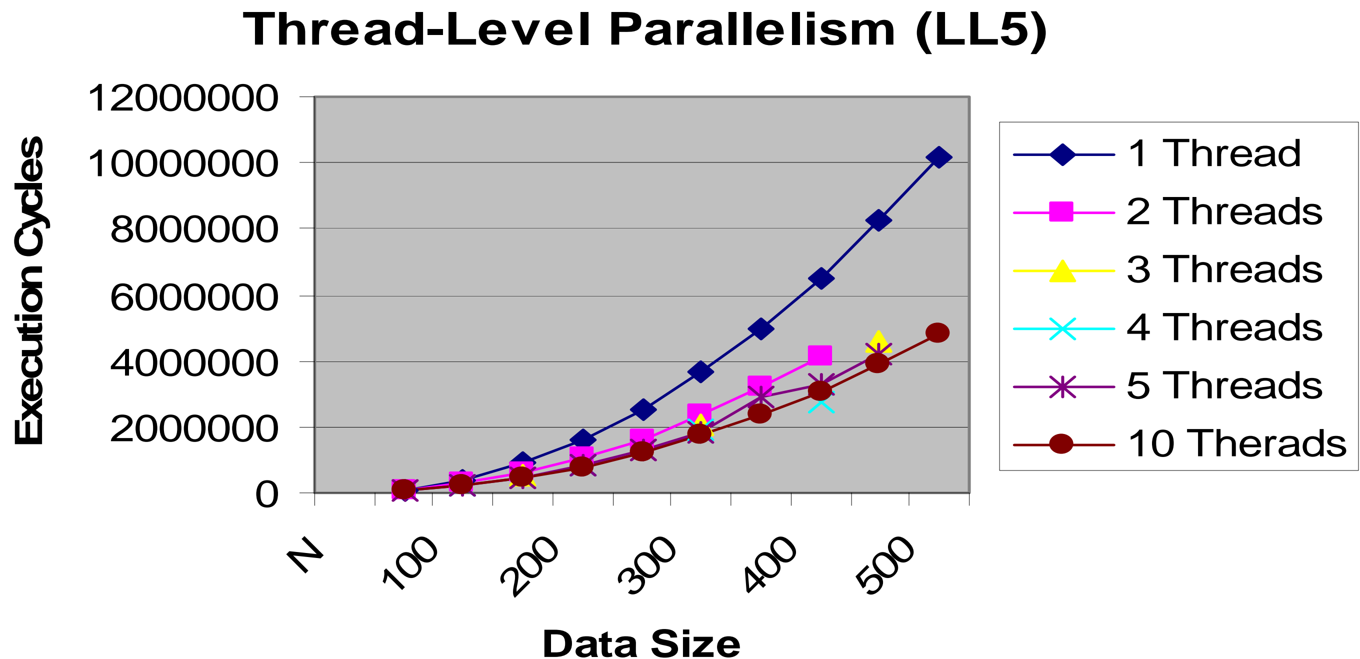
Zoom

N	DLX Cycles	SDF Cycles	Speed UP
5,5,4	10175	9661	1.0532
10,10,4	40510	37421	1.0825
15,15,4	97945	83331	1.1754
20,20,4	161580	147391	1.0963
25,25,4	271175	229601	1.1811
30,30,4	391150	329961	1.1854
35,35,4	532285	448471	1.1869
40,40,4	645520	585131	1.1032

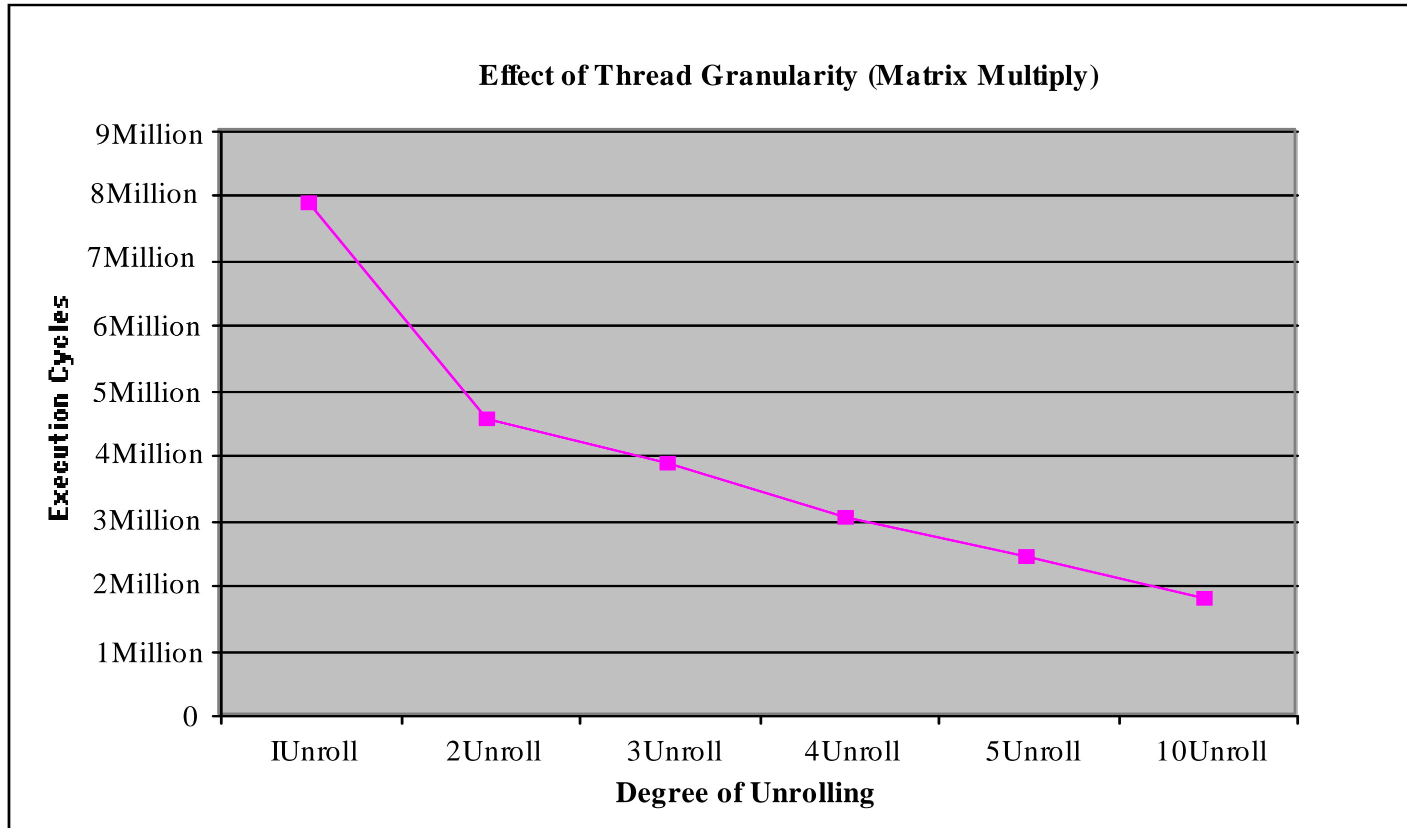
Effect of thread level parallelism(Matrix Multiply)



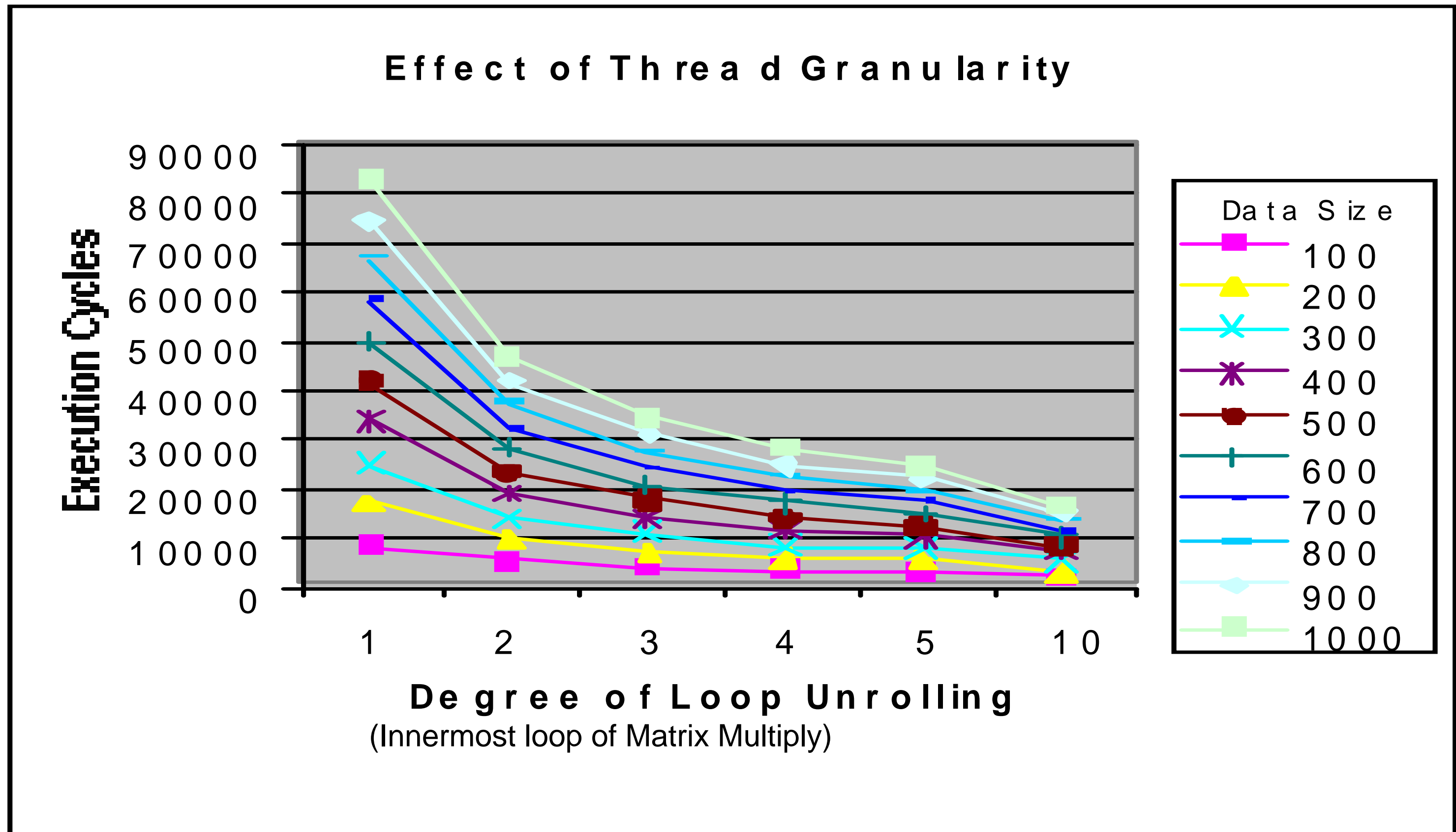
Effect of thread level parallelism(LL5) (different data sizes)



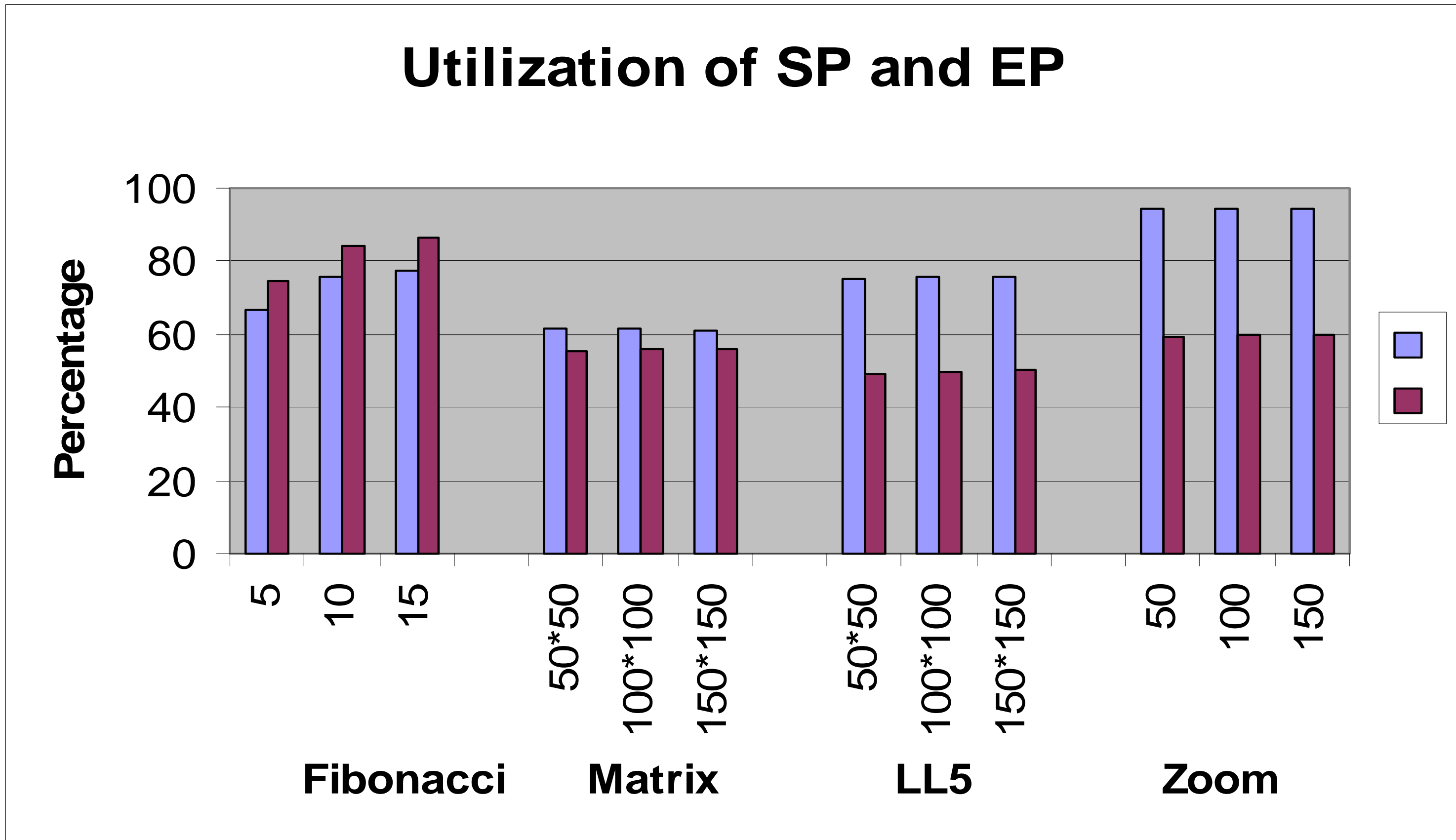
Effect of thread granularity (Matrix Multiply)



Effect of thread granularity (Matrix Multiply) (different data sizes)



Utilization of SP and EP



SDF vs Superscalar for Matrix Multiplication

Data Size	SDF (cycles)	SS-IO (cycles)	SS-OO (cycles)	SDF vs. SS-IO Speedup	SDF vs. SS-OO Speedup
50*50	1,720,885	1,968,235	1,174,250	0.8743	1.4655
100*100	13,318,705	15,434,835	9,170,850	0.8628	1.4522
150*150	44,453,530	51,811,474	30,747,479	0.8579	1.4457

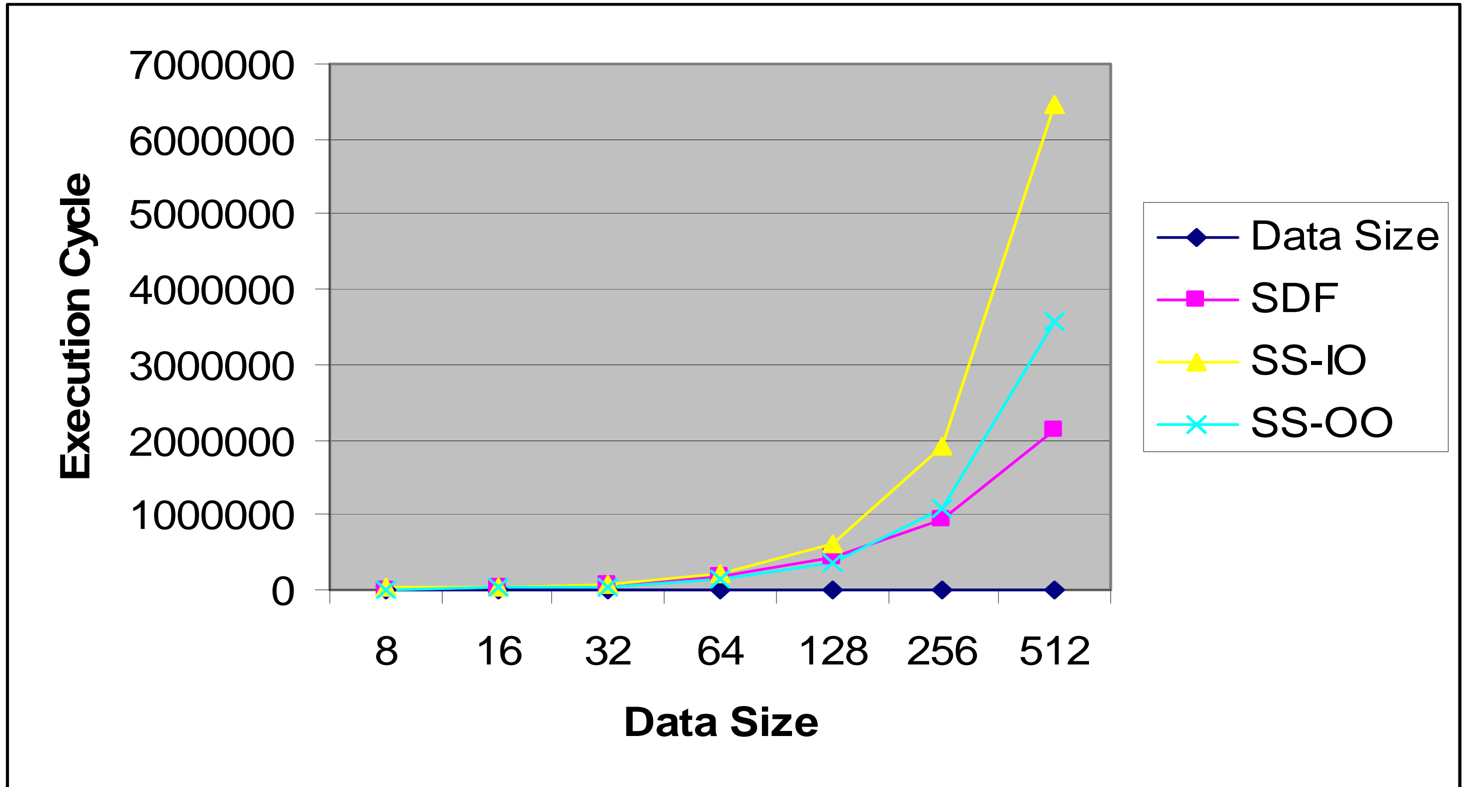
SS-SI = Superscalar Inorder
SS-OO=Superscalar Outorder

SDF vs Superscalar for FFT

Data Size	SDF (cycles)	SS-IO (cycles)	SS-OO (cycles)	SDF vs. SS-IO speed-up	SDF vs. SS-OO speed-up
8	13,791	21,423	14,294	0.6437	0.9648
16	33,635	37,917	25,608	0.8870	1.3134
32	79,895	83,024	53,595	0.9623	1.4907
64	185,765	212,301	132,479	0.8750	1.4022
128	424,411	604,674	364,203	0.7018	1.1653
256	**955,721**	1,906,115	**1,095,955**	0.5013	0.8720
512	**2,126,583**	6,543,376	**3,576,399**	0.3295	0.5946

As the data size increases we improve the performance.

Comparing SDF with Superscalar for FFT



SDF vs Superscalar for Fibonacci program

Data Size	SDF (cycle)	SS-IO (cycles)	SS-OO (cycles)	SDF vs. SS-IO Speedup	SDF vs. SS-OO Speedup
5	799	11,914	7,853	0.0670	0.1017
10	8,092	15,676	10,697	0.5162	0.7564
15	87,835	57,422	42,226	1.5296	2.0801
19	597,382	326,032	245,107	1.8322	2.4372

Since fibonacci is a recursive program, the smaller data size the performance is better and then it drops. No thread level parallelism and so it does not do better.

SDF vs Superscalar for zoom program

Data Size	SDF (cycles)	SS-IO (cycles)	SS-OO (cycles)	SDF vs. SS-IO Speedup	SDF vs. SS-OO Speedup
50*50*4	442,940	353,969	250,508	1.2514	1.7682
100*100*4	1,768,070	1,400,923	998,666	1.2621	1.7704
150*150*4	3,975,450	3,231,841	2,326,862	1.2301	1.7085
200*200*4	7,065,080	5,584,818	3,970,448	1.2651	1.7794

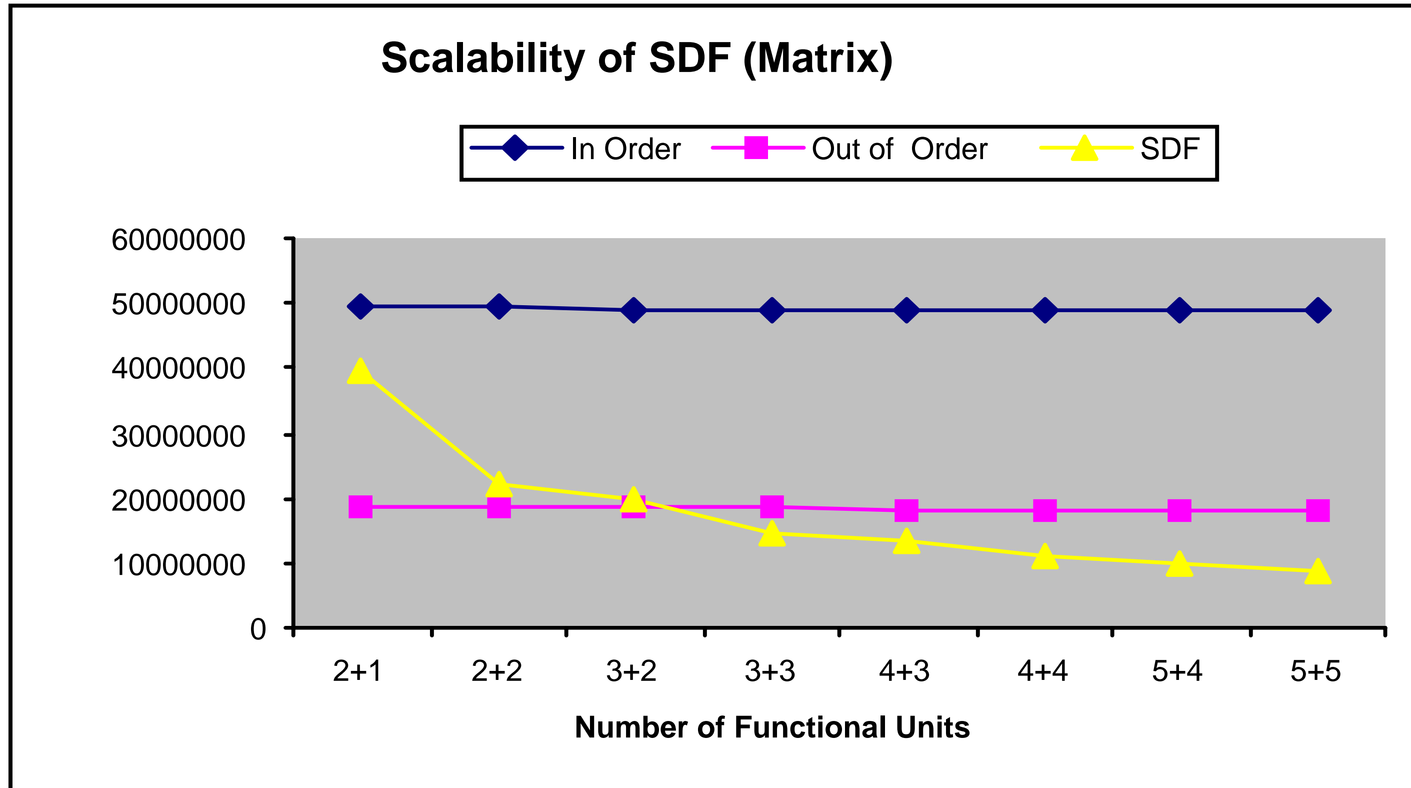
In zoom program, since there is a loop dependency, it does not improve the performance of the program.

<u>SDF with multiple SPs and Eps</u>
Matrix Multiply of different data sizes.

Data size		Superscalar (cycles) 2INT ALU 1FP ALU	SDF (cycles) 2SP 1EP	Superscalar (cycles) 2INT ALU 2FP ALU	SDF (cycles) 2SP 2EP	Supserscalar (cycles) 3INT ALU 2FP ALU	SDF (cycles) 3SP 2EP	Superscalar (cycles) 3INT ALU 3FP ALU	SDF (cycles) 3SP 3EP
50*50	IO	1,890,104	1,504,297	1,890,104	860,782	1,867,200	756,707	1,867,200	**574242**
	OO	712,396		712,396		706,877		706,877	
100*100	IO	14,824,104	11,843,442	14,824,104	6,660,012	14,633,700	5,941,602	14,633,700	**4,402,772**
	OO	5,532,202		5,532,202		5,511,587		5,511,587	
150*150	IO	49,763,150	39,762,487	49,763,150	22,227,742	49,110,246	19,924,912	49,110,246	**14,819,482**
	OO	18,514.510		18,514,510		18,468,811		18,468,409	

Data size		Superscalar (cycles) 4INT ALU 3FP ALU	SDF (cycles) 4SP 3EP	Superscalar (cycles) 4INT ALU 4FP ALU	SDF (cycles) 4SP 4EP	Supserscalar (cycles) 5INT ALU 4FP ALU	SDF (cycles) 5SP 4EP	Superscalar (cycles) 5INT ALU 5FP ALU	SDF (cycles) 5SP 5EP
50*50	IO	1,867,200	**507,197**	1,867,200	**430,957**	1,867,200	**381,247**	1,867,200	**345,027**
	OO	680,321		680,321		680,321		680,321	
100*100	IO	14,633,700	**3,970,682**	14,633,700	**3,330,992**	14,633,700	**2,982,702**	14,633,700	**2,665,472**
	OO	5,306,381		5,306,381		5,306,380		5,306,380	
150*150	IO	49,110,246	**13,308,457**	49,110,246	**11,115,592**	49,110,246	**9,990,607**	49,110,246	**8,894,002**
	OO	17,782.453		17,782,453		17,782,453		17,782,453	

Scalability of SDF for Matrix Multiply



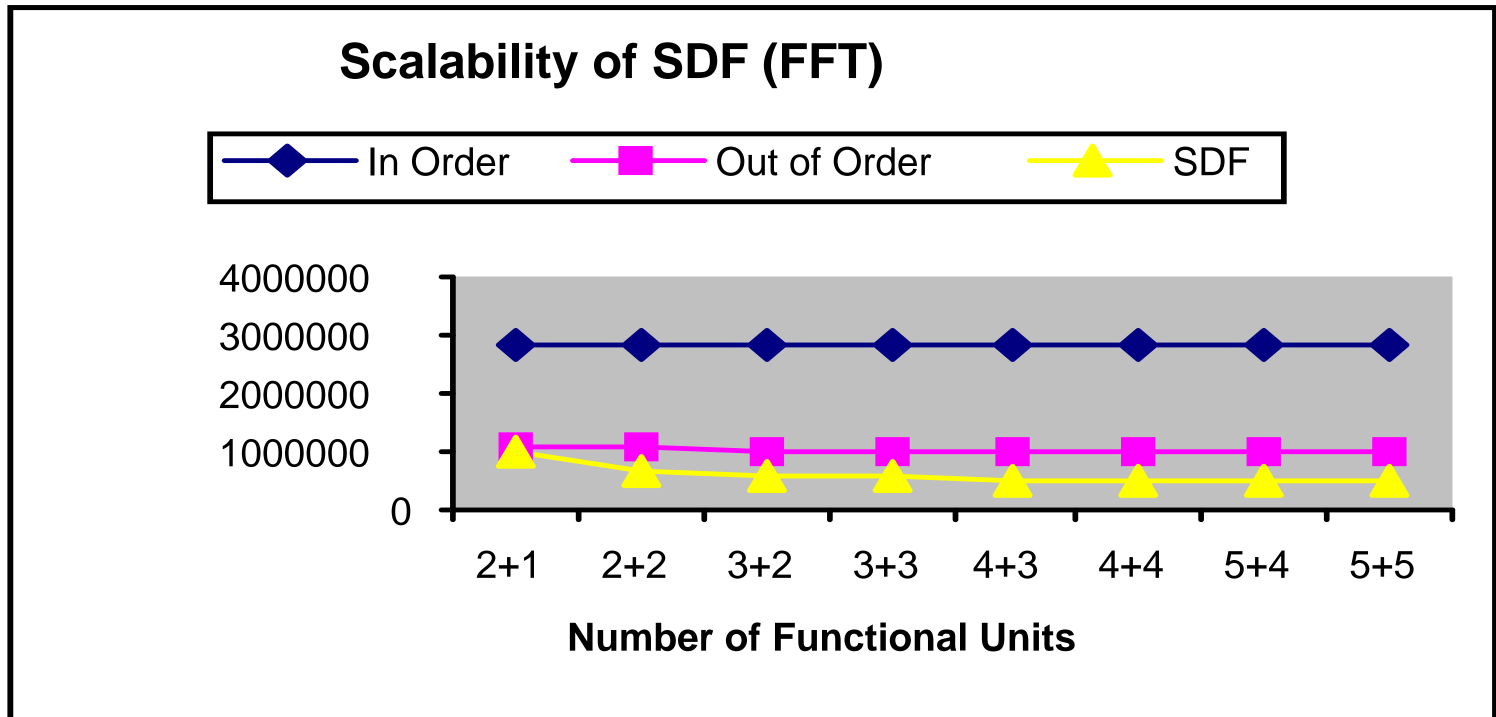
Data size 150*150

SDF with multiple SPs and Eps
FFT program of different data sizes.

Data size		Superscalar (cycles) 2INT ALU 1FP ALU	SDF (cycles) 2SP 1EP	Superscalar (cycles) 2INT ALU 2FP ALU	SDF (cycles) 2SP 2EP	Supserscalar (cycles) 3INT ALU 2FP ALU	SDF (cycles) 3SP 2EP	Superscalar (cycles) 3INT ALU 3FP ALU	SDF (cycles) 3SP 3EP
256	IO	2,874,748	**992,129**	2,874,748	**638,705**	2,843,118	**559,665**	2,843,118	**558,353**
	OO	1,056,799		1,055,514		1,005,519		1,005,519	
512	IO	8,672,232	**1,516,660**	8,672,232	**1,418,356**	14,633,700	**1,240,948**	8,571,665	**1,238,580**
	OO	2,977,573		2,974,495		2,808,108		2,808,108	

Data size		Superscalar (cycles) 4INT ALU 3FP ALU	SDF (cycles) 4SP 3EP	Superscalar (cycles) 4INT ALU 4FP ALU	SDF (cycles) 4SP 4EP	Supserscalar (cycles) 5INT ALU 4FP ALU	SDF (cycles) 5SP 4EP	Superscalar (cycles) 5INT ALU 5FP ALU	SDF (cycles) 5SP 5EP
256	IO	2,871,928	**525,873**	2,871,928	**525,457**	2,871,928	**508,047**	2,871,928	**507,633**
	OO	1,013,417		1,013,417		1,012,031		1,012,031	
512	IO	8,636,150	**1,163,956**	8,639,150	**1,163,956**	8,639,150		8,639,150	
	OO	2,828,025		2,828,025		2,823,356		2,823,356	

Scalability of SDF for FFT



Data size N=256

SDF with multiple SPs and Eps

Fibonacci program of different data sizes.

Data size		Superscalar (cycles) 2INT ALU 1FP ALU	SDF (cycles) 2SP 1EP	Superscalar (cycles) 2INT ALU 2FP ALU	SDF (cycles) 2SP 2EP	Supserscalar (cycles) 3INT ALU 2FP ALU	SDF (cycles) 3SP 2EP	Superscalar (cycles) 3INT ALU 3FP ALU	SDF (cycles) 3SP 3EP
10	IO	14,677	7,408	14,677	**4,162**	14,208	**3,929**	14,208	**2,874**
	OO	7,064		7,064		6,471		5,974	
15	IO	51,580	80,802	51,580	44,033	50,566	41,481	50,566	29,463
	OO	27,967		27,967		24,439		24,439	

Data size		Superscalar (cycles) 4INT ALU 3FP ALU	SDF (cycles) 4SP 3EP	Superscalar (cycles) 4INT ALU 4FP ALU	SDF (cycles) 4SP 4EP	Supserscalar (cycles) 5INT ALU 4FP ALU	SDF (cycles) 5SP 4EP	Superscalar (cycles) 5INT ALU 5FP ALU	SDF (cycles) 5SP 5EP
10	IO	14,174	**2,764**	14,174	**2,263**	14,174	**2,193**	14,174	**1,888**
	OO	5,964		5,964		5,960		5,960	
15	IO	50,189	28,151	50,189	**22,206**	50,189	**21,415**	50,189	**17,841**
	OO	24,534		24,534		24,530		24,530	

Scalability of SDF for Fibonacci program

Data size N=15

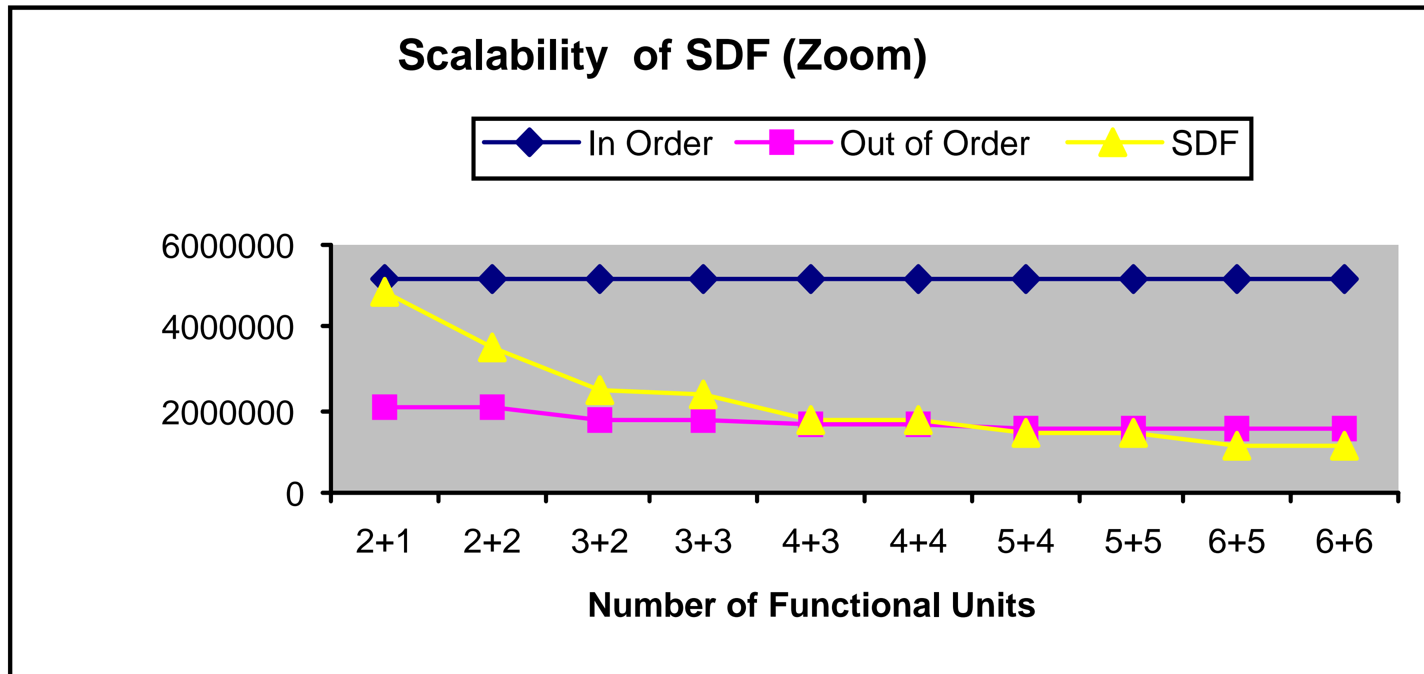
SDF with multiple SPs and Eps

zoom program of different data sizes.

Data Size		Superscalar (Cycles) 2INT ALU 1FP ALU	SDF (Cycles) 2SP 1EP	Superscalar (Cycles) 2INT ALU 2FP ALU	SDF (cycles) 2SP 2EP	Superscalar (cycles) 3INT ALU 2FP ALU	SDF (cycles) 3SP 2EP	Superscalar (Cycles) 3INT ALU 3FP ALU	SDF (cycles) 3SP 3EP	Superscalar (cycles) 4INT ALU 3FP ALU	SDF (Cycles) 4SP 3EP
100*100*4	IO	1,310,785	1,217,847	1,310,785	881,837	1,310,379	626,967	1,310,379	586,417	1,310,377	441,847
	OO	518,518		518,518		447,182		447,182		408,164	
200*200*4	IO	5,224,086	4,868,037	5,224,086	3,522,217	5,223,680	2,522,577	5,223,210	2,340,497	5,223,208	1,760,257
	OO	2,061,519		2,061,519		1,777,283		1,776,955		1,622,538	

Data Size		Superscalar (Cycles) 4INT ALU 4FP ALU	SDF (Cycles) 4SP 4EP	Superscalar (Cycles) 5INT ALU 4FP ALU	SDF (cycles) 5SP 4EP	Superscalar (cycles) 5INT ALU 5FP ALU	SDF (cycles) 5SP 5EP	Superscalar (Cycles) 6INT ALU 5FP ALU	SDF (cycles) 6SP 5EP	Superscalar (cycles) 6INT ALU 6FP ALU	SDF (Cycles) 6SP 6EP
100*100*4	IO	1,310,377	440,417	1,310,377	**353,037**	1,310,379	**353,057**	1,310,379	**295,357**	1,310,377	**295,497**
	OO	408,164		398,454		398,454		398,128		398,128	
200*200*4	IO	5,223,208	1,756,957	5,223,208	**1,406,537**	5,223,680	**1,406,717**	5,223,210	**1,177,057**	5,223,208	**1,175,137**
	OO	1,622,538		1,585,528		1,585,528		1,585,528		1,585,528	

Scalability of SDF for zoom program



Data size N=200*200*4

SDF versus VLIW Architectures

Two different VLIW simulators are used

- **TI's DSP TMS320C6000**
  - **8-wide VLIW**
- **Trimaran Simulation Environment**
  - **9 functional unit**
  - **32-loop unrolling.**

Compared with SDF using 4 Eps and 4 SPs.

SDF versus VLIW Architecture(Matrix Multiply)

Matrix Multiplication					
Data Size	SDF	Trimaran	TMS 'C6000	SDF/Trimaran	SDF/TMX 'C6000
50*50	430957	331910	1033698	1.29841523	0.416908033
100*100	3330992	2323760	16199926	1.43344924	0.205617729
150*150	11115592	4959204	86942144	2.24140648	0.127850447

Trimaran uses a well optimized compiler as well as loop unrolling of program many times and so it certainly improves performance.

Impact of Thread level parallelism onSDF versus VLIW Architecture (Matrix Multiply)

Matrix Multiplication							
Data Size	TMS 'C6000 VLIW 8-wide Optimized	SDF 4SPs,4Eps 10 Threads	SDF 4SPs,4EPs 5 Threads	SDF 4SPs,4Eps 2 Threads	SDF-10 vs VLIW	SDF-5 vs VLIW	SDF-2 vs VLIW
50*50	1033698	**430957**	1525560	2956616	0.4169	1.4758	2.8602
100*100	16199926	**3330992**	**11096312**	22297111	0.2056	0.6850	1.3764
150*150	86942144	**11115592**	**36242464**	**73773061**	0.1279	0.4169	0.8485

As we add more units to SDF, the performance increases.

SDF versus VLIW Architecture(FFT)

FFT					
Data Size	SDF	Trimaran Optimized	TMS C6000 Optimized	SDF/Trimaran	SDF/TMS
8	8148	4622	26717	1.762873215	0.304974361
16	19323	12391	73456	1.559438302	0.263055435
32	45028	31665	213933	1.422011685	0.210477112
64	103491	81375	619241	1.271778802	0.167125562
128	234766	214685	2040729	1.093537043	0.115040263
256	**525457**	595211	6943638	0.882807945	0.075674596
512	**1163956**	1768441		0.658181981	

As we saw with the superscalar architecture, the performance of SDF increases with the larger data size and more thread level parallelism.

SDF versus VLIW Architecture(zoom)

ZOOM					
Data Size	SDF	Trimaran Optimized	TMS Optimized	SDF/Trimaran	SDF/TMS
50*50*4	**115452**	157770	144201	0.7317741	0.800632451
100*100*4	**459667**	630520	641625	0.72902842	0.716410676
150*150*4	**1032567**	1418270	1480525	0.72804685	0.697433005
200*200*4	**1833337**	2521020	2959430	0.72722033	0.619489902
250*250*4	**2862857**	3938770	4729593	0.72684036	0.605307264

Zoom program loop dependency affects the other two model of architecture such as TMS and Trimaran.

Summary and conclusion of the study

- SDF architecture separates memory accesses and execution neatly
  - -because of the non-blocking multithreaded model
- The non-blocking model allows the pipes to execute independently pre-load/post-store/Execute portions of a thread
  - - fewer context switches(possibly more threads)
- SDF architecture outperforms MIPS, superscalar and VLIW .
  - - because of multithreaded model
- SDF scales better with more functional units
  - - because of multithreaded model
- The SDF architecture is simpler since it does not need complex hardware for detecting data hazards, register renaming or out-of-order instruction execution
  - - because of the functional nature of dataflow instructions

Future Work

- **Implement optimization techniques.**
  (Efficient use of Registers)

- **Direct storing of results into the next thread instead of frame memory. (Already investigated this.)**
  (Performance can be improved)

- **Present more benchmarks such as SPEC2000.**

- **Build an optimized compiler with a suitable branch prediction techniques. (Being developed)**

Publications:

1. **J.Arul** and K.M. Kavi, "Scalability of Scheduled Dataflow Architecture," to be presented in the 5th *International conference on Algorithms and Architecture for Parallel Processing,* Beijing, Oct 23-25th 2002.

2. K.M. Kavi,R.Giorgi and **J. Arul** "Scheduled Dataflow: Execution Paradigm, Architecture, and Performance Evaluation" *IEEE Trans.Computers,Vol.50, No.8,* August 2001, pp:834-846

3. **J. Arul,** K.M. Kavi and S. Hanief. " Cache Performance of Scheduled dataflow Architecture", *Proc.of the 4th International Conference on Algorithms for Parallel Processing (ICA3PP),* 2000, PP 110-123.

4. **J.Arul**. "Execution Performance of the Scheduled Dataflow Architecture(SDF)", *International Conference on Parallel Architecture and Compilation Techniques:* Workshop Oct 13-15th 2000.

5. K.M. Kavi,**J.Arul** and R.Giorgi."Execution and cache performance of the Scheduled Dataflow Architecture", *Journal of Universal Computer Science, Special Issue on Multithreaded and Chip Multiprocessors*, Vol. 6, No. 10, Oct. 2000, pp 948-967,.

6. K.M. Kavi, H.-S.Kim, **J. Arul** and A.R. Hurson, "A decoupled scheduled dataflow multithreaded architecture", *Proceedings of the 1999 International Symposium on Parallel Architectures*, Algorithms and Networks(I-SPAN99), June 23-25, 1999.