Scheduled Dataflow Architecture

Krishna Kavi
Department of Computer Science
The University of North Texas
kavi@cs.unt.edu
http://crash.eb.uah.edu/~kavi
Overview Of Different Research Projects

- Computer Systems
  - Multithreaded Architecture and Cache Memories for IRAM devices and storage management for object-oriented systems
- Intelligent Agent Based Systems
  - Software architectures for agent based systems
  - Extensions to UML to model agents
Overview of SDF

- Based on our past work with Dataflow and Functional Architectures
  - Instruction set retains functional nature, but it is not data-driven

- Non-Blocking Multithreaded Architecture
  - Contains multiple functional units like superscalar and other multithreaded systems
  - Contains multiple register contexts like other multithreaded systems

- Decoupled Access - Execute Architecture
  - Completely separates memory accesses from execution pipeline
Presentation Outline

• Dataflow and Instruction Driven Model of Execution
  – Schedule instructions like “Control Flow” Architecture

• Multithreaded Architectures

• Decoupling Memory Accesses

• SDF

• Evaluation of SDF
Presentation Outline

- Dataflow and Instruction Driven Model of Execution
  - Schedule instructions like “Control Flow” Architecture
- Multithreaded Architectures
- Decoupling Memory Accesses
- SDF
- Evaluation of SDF
Dataflow Architecture

Pure Dataflow Instructions

1: LOAD  3L  / load A, send to Instruction 3
2: LOAD  3R  / load B, send to Instruction 3
3: ADD   8R, 9R  / A+B, send to Instructions 8 and 9
4: LOAD  6L, 7L  / load X, send to Instructions 6 and 7
5: LOAD  6R, 7R  / load Y, send to Instructions 6 and 7
6: ADD   8L  / X+Y, send to Instructions 8
7: SUB   9L  / X-Y, send to Instruction 9
8: MULT  10L  / (X+Y)*(A+B), send to Instruction 10
9: DIV   11L  / (X-Y)/(A+B), send to Instruction 11
10: STORE ....,  R14  / store first result
11: STORE ....,  R15  / store second result

MIPS like instructions

1. LOAD    R2, A  / load A into R2
2. LOAD    R3, B  / load B into R3
3. ADD      R11, R2, R3  / R11 = A+B
4. LOAD    R4, X  / load X into R4
5. LOAD    R5, Y  / load Y into R5
6. ADD      R10, R4, R5  / R10 = X+Y
7. SUB      R12, R4, R5  / R12 = X-Y
8. MULT    R14, R10, R11  / R14 = (X+Y)*(A+B)
9. DIV      R15, R12, R11  / R15 = (X-Y)/(A+B)
10. STORE   .....,  R14  / store first result
11. STORE   .....,  R15  / store second result
<table>
<thead>
<tr>
<th>Features Of Dataflow</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data Driven</strong> ---- Instructions are enabled for execution Only when operands are made available by preceding instructions (We are changing this as explained later)</td>
</tr>
<tr>
<td><strong>Implied Synchronization</strong></td>
</tr>
<tr>
<td>Data availability is the only synchronization</td>
</tr>
<tr>
<td>No explicit instructions for synchronization are needed</td>
</tr>
<tr>
<td><strong>No Variables -- only Data</strong> -- Results are sent directly to instructions</td>
</tr>
<tr>
<td><strong>Freedom From Side-Effects</strong></td>
</tr>
<tr>
<td><strong>Free Context Switch</strong></td>
</tr>
<tr>
<td>Each instruction is an independent context</td>
</tr>
<tr>
<td>Each instruction can be viewed as a thread in multi-threaded systems</td>
</tr>
<tr>
<td>Can be used to hide remote-memory latency</td>
</tr>
</tbody>
</table>
Data Driven Model

Data driven model

Each generated data is “tagged” to identify the destination instruction and a context (or loop iteration) for the instruction.

The tokens (data + tag) is stored in a separate memory -- separate from instructions

Matching operands of an instruction.
If two tokens have the same tag, they are destined to the same instruction -- a matched pair

Matching in original dynamic dataflow (Tagged Token model) used associative memory
Tagged Token Architecture

Token Format

<table>
<thead>
<tr>
<th>PE#</th>
<th>Context (iteration #)</th>
<th>Instruction Id</th>
<th>port</th>
<th>Data Value</th>
</tr>
</thead>
</table>

Instruction Format

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Dest-1 andport</th>
<th>Dest-2 andport</th>
</tr>
</thead>
</table>

Data Tokens From Other Processors

A PE

Match Operands

Assemble Instruction

Packets

ALU

Form Tags for Results

Data Tokens To Other Processors

Data Tokens To Be Used Locally

Data Tokens To Other Processors

University of Pisa, Sept 28, 2004
## Limitations Of Dataflow Implementations

- Memory Hierarchies cannot be used
  - No variables -- only data

- Too fine-grained
  - Each instruction is a continuation (or thread)
  - Could incur too much overhead

- Localities are difficult to synthesize
  - Both data and instruction localities are difficult
  - Can’t benefit from cache memories

- Asynchronous execution
  - Data driven model implies asynchronous
  - Difficulties in hardware design and complexities
The Explicit Token Store Architecture (ETS)
A possible Solution!

Each instruction specifies a data memory location (Frame location) where its input operands will be matched.

ETS Instructions

1: LOAD 1, 3L / load A, send to Instruction 3
2: LOAD 2, 3R / load B, send to Instruction 3
3: ADD 3, 8R, 9R / A+B, send to Instructions 8 and 9
4: LOAD 4, 6L, 7L / load X, send to Instructions 6 and 7
5: LOAD 5, 6R, 7R / load Y, send to Instructions 6 and 7
6: ADD 6, 8L / X+Y, send to Instructions 8
7: SUB 7, 9L / X-Y, send to Instruction 9
8: MULT 8, 10L / (X+Y)*(A+B), send to Instruction 10
9: DIV 9, 11L / (X-Y)/(A+B), send to Instruction 11
10: STORE 10, / store first result
11: STORE 11, / store second result

Token Format

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Offset (R)</th>
<th>Dest-1 and Port</th>
<th>Dest-2 and Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>Opcode</td>
<td>Offset (R)</td>
<td>Dest-1 and Port</td>
<td>Dest-2 and Port</td>
</tr>
<tr>
<td>Opcode</td>
<td>Offset (R)</td>
<td>Dest-1 and Port</td>
<td>Dest-2 and Port</td>
</tr>
<tr>
<td>Opcode</td>
<td>Offset (R)</td>
<td>Dest-1 and Port</td>
<td>Dest-2 and Port</td>
</tr>
<tr>
<td>Opcode</td>
<td>Offset (R)</td>
<td>Dest-1 and Port</td>
<td>Dest-2 and Port</td>
</tr>
<tr>
<td>Opcode</td>
<td>Offset (R)</td>
<td>Dest-1 and Port</td>
<td>Dest-2 and Port</td>
</tr>
</tbody>
</table>
ETS Example

Code-Block Activation

Instruction Memory

<table>
<thead>
<tr>
<th>Opcode</th>
<th>r</th>
<th>dests</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>2</td>
<td>+1,+2L</td>
</tr>
<tr>
<td>NEG</td>
<td>-</td>
<td>+6</td>
</tr>
<tr>
<td>SUB</td>
<td>3</td>
<td>+1</td>
</tr>
</tbody>
</table>

Frame Memory

Presence Bits

4.24
An Implementation of ETS
ETS requires two cycles per instruction -- one cycle per token
   First token is stored in frame memory awaiting its partner
   The instruction is executed only when the second token arrives
   -- All data driven implementations require 2 cycles

Instead

Associate a pair of memory locations (or a pair of registers)
   with each instruction
   ETS associates one location per instruction

Operands for an instruction are saved in the pair of locations until the
   instruction is Scheduled for execution

No longer Data Driven execution -- but Instruction Driven
A Programming Example

LOAD "X" R2  /* Load X into R2
LOAD "Y" R3  /* Load Y into R3
LOAD "A" R4  /* Load A into R4
LOAD "B" R5  /* Load B into R5

ADD RR2, R6  /* Add R2, R3, (X+Y) Result in R6
SUB RR2, R8  /* Sub R2, R3, (X-Y) Result in R8
ADD RR4, R7, R9 /* Add R4, R5, (A+B) Result in R7 and R9
DIV RR8, R12 /* Div R8, R9 [(X-Y)/(A+B)], Result in R12
MUL RR6, R11 /* Multiply R6, R7, [(X+Y)*(A+B)], Result in R11

STORE R11 “OUT’’
STORE RR8, “OUT”
• Instructions retain “functional” properties of dataflow
• Instructions are executed synchronously like control-flow
• Can utilize registers, caches and other memory hierarchies

• Non-blocking multithreading model (like Cilk)
  A thread is enabled when all inputs are received
  Once enabled thread executes to completion without blocking or context switching

• Permits for decoupling memory accesses from execution pipeline
Presentation Outline

• Dataflow and Instruction Driven Model of Execution
  – Schedule instructions like “Control Flow” Architecture

• Multithreaded Architectures

• Decoupling Memory Accesses

• SDF

• Evaluation of SDF
Multithreaded Architectures

An architecture that switches between multiple threads of execution

*With multiple register contexts, switching is fast*

Thread switch may occur
- on cache misses
- on remote memory accesses
- or on other types of long latency operations

Switching between threads can take place on every instruction --- *Interleaving execution of threads*

Tera and SMT
Blocking vs Non-Blocking Thread Models

Traditional multithreaded systems use blocking models

- A thread is blocked (or preempted)
- A blocked thread is switched out
  and execution resumes in future

- In some cases, the resources of a blocked thread
  (including register context) may be assigned to other
  awaiting threads.
- Blocking models require more context switches

In a non-blocking model, once a thread begins execution, it will not be stopped (or preempted) before it completes execution
Non-Blocking Threads

Most functional and dataflow systems use non-blocking threads

A thread/code block is enabled when all its inputs are available.
A scheduled thread will run to completion.

Similar to Cilk Programming model

Note that recent versions of Cilk (Clik-5) permits thread blocking and preemptions
thread fib (cont int k, int n)
    {   if (n<2)
            send_argument (k, n)
        else{
            cont int x, y;
            spawn_next sum (k, ?x, ?y); /* create a successor thread
            spawn fib (x, n-1); /* fork a child thread
            spawn fib (y, n-2); /* fork a child thread
        }
    }
thread sum (cont int k, int x, int y)
    {send_argument (k, x+y);} /* return results to parent’s
    /*successor
Cilk Runtime Structures

SDF will use Frames Similar to that of Cilk to save Thread inputs
Presentation Outline

• Dataflow and Instruction Driven Model of Execution
  – Schedule instructions like “Control Flow” Architecture
• Multithreaded Architectures
• Decoupling Memory Accesses
• SDF
• Evaluation of SDF
Separate Processor to handle all memory accesses

The earliest suggestion by J.E. Smith -- DAE architecture
Limitations of DAE Architecture

- Designed for STRETCH system with no pipelines
- Single instruction stream
- Instructions for Execute processor must be coordinated with the data accesses performed by Access processor
- Very tight synchronization needed
- Coordinating conditional branches complicates the design
- Generation of coordinated instruction streams for Execute and Access my prevent traditional compiler optimizations
A multithreaded processor
Separate Memory and Execution Pipelines
A thread is handed off to Memory processor when a Memory Access Instruction is decoded
A thread is handed off to Execute processor when a non-memory access instruction is decoded

Other context switches may be needed
Switch on Use -- data dependencies
Synchronization

Recent implementations

Rhamma Processor
(Univ. Karlsruhe)
## Limitations Of Rhamma

- Blocking Thread Model -- Requires More context switches
- Checking for data dependencies requires complex hardware
- Bubbles in pipelines are unavoidable on context switches and cache misses
- Does not group memory access instructions
  - A thread may shuttle between the two processors
Pre-Load and Post-Store

Group all LOAD instructions together at the head of a 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 in awaiting Frames

Our non-blocking model facilitates clean separation of memory accesses into Pre-load and Post-store
Features Of Our Decoupled System

• No pipeline bubbles due to cache misses
• Overlapped execution of threads
• Opportunities for better data placement and prefetching
• Fine-grained threads -- A limitation?
• Multiple hardware contexts add to hardware complexity

If 35% of instructions are memory access instructions, PL/PS can achieve 35% increase in performance with sufficient thread parallelism and completely mask memory access delays!
• Dataflow and Instruction Driven Model of Execution
  – Schedule instructions like “Control Flow” Architecture
• Multithreaded Architectures
• Decoupling Memory Accesses
• Architectural Details of SDF
• Evaluation of SDF
A Programming Example using SDF Instructions

Pre-Load
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  / frame pointer for returning first result
LOAD  RFP|7,    R7  / frame offset for returning first result
LOAD  RFP|8,    R8  / frame pointer for returning second result
LOAD  RFP|9,    R9  / frame offset for returning second result

Execute
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

Post-Store
STORE   R14, R6|R7  / store first result
STORE   R15, R8|R9  / store second result
Conditional Statements in SDF

Pre-Load
LOAD RFP| 2, R2 / load X into R2
LOAD RFP| 3, R3 / load Y into R3
/ frame pointers for returning results
/ frame offsets for returning results

Execute

<table>
<thead>
<tr>
<th>Operation</th>
<th>Then_T</th>
<th>Else_T</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQ</td>
<td>RR2, R4</td>
<td></td>
</tr>
<tr>
<td>NOT</td>
<td>R4, R5</td>
<td></td>
</tr>
<tr>
<td>FALLOC</td>
<td>“Then_Thread”</td>
<td></td>
</tr>
<tr>
<td>FALLOC</td>
<td>“Else_Thread”</td>
<td></td>
</tr>
<tr>
<td>FORKSP</td>
<td>R4, “Then_Store”</td>
<td></td>
</tr>
<tr>
<td>FORKSP</td>
<td>R5, “Else_Store”</td>
<td></td>
</tr>
</tbody>
</table>

/ compare R2 and R3, Result in R4
/ Complement of R4 in R5
/ Create Then Thread (Allocate Frame memory, Set Synch-Count,
/ Create Else Thread (Allocate Frame memory, Set Synch-Count,
/If X=Y, get ready post-store “Then_Thread”
/Else, get ready pre-store “Else_Thread”

In Then_Thread, We de-allocate (FFREE) the Else_Thread and vice-versa
SDFC (SDF Compiler)
Execution of SDF Programs

Thread 0
- Preload
- Execute
- Poststore

Thread 1
- Preload
- Execute
- Poststore

Thread 2
- Preload
- Execute
- Poststore

Thread 3
- Preload
- Execute
- Poststore

Thread 4
- Preload
- Execute
- Poststore

SP = PL/PS
EP = EX

University of Pisa, Sept 28, 2004
What is a Continuation?

- Continuation = <FP,IP,RS,SC>
  - FP = Frame Pointer
  - IP = Instruction Pointer
  - RS = Register Set (or Register Context)
  - SC = Synchronization Count

- Type of continuations:
  - Post-Store Continuation  <--,IP,RS,-->  PSC
  - Pre-Loaded Continuation  <--,IP,RS,-->  PLC
  - Enabled Continuation     <FP,IP,RS,-->  EXC
  - Waiting Continuation     <FP,IP,--,SC>  WTC

- At a certain instant a continuation can be of a certain type, depending on which fields are in valid state (i.e. Assigned)
Continuation Transitions

- WTC: SYNCHRONIZATION COUNT MET
- SP → PLC: PRELOAD PHASE COMPLETED
- PLC → EXC: EXECUTION PHASE COMPLETED
- EXC → PSC: THREAD TERMINATED
- PSC: EXECUTION PHASE COMPLETED
Memory Access Pipeline (SP)

- **ENABLED CONTINUATION**
  - PLC
  - PSC
- **POST-STORE CONTINUATION**
- **PRE-LOADED CONTINUATION**
  - EXC
- **INSTRUCTION CACHE**
  - EFFECTIVE ADDRESS
  - MEMORY ACCESS
- **PROGRAM COUNTER**
- **DATA CACHE**
- **EXECUTE**
- **WRITE-BACK**
- **REGISTER SETS**
Execution Pipeline (EP)

- FETCH
- DECODE
- EXECUTE
- WRITE-BACK
- REGISTER SET

PRE-LOADED CONTINUATION

EXC

INSTRUCTION CACHE

PROGRAM COUNTER
Execution Pipeline (EP-new)

Use superscalar like execution unit with multiple functional units
still no memory operations

Interleave instructions from multiple “pre-loaded” threads
similar to SMT

Instructions can be executed out of order -- but less complex
instruction scheduling hardware
no need for register renaming

May even permit speculative execution

Now collecting data - but shows about 15% improvement over
previous SDF
Execution Pipeline (EP-new)

pre-loaded Th1 | pre-loaded Th2 | pre-loaded Th3 | pre-loaded Th4

Fetch-Issue Unit

FU-1 | FU-2 | FU-n

Instruction Retire

post-store Th1 | post-store Th2 | post-store Th3 | post-store Th4
SDF Architecture

Execute Processor (EP)

Memory Access Pipeline

Synchronization Processor (SP)
Presentation Outline

- Dataflow and Instruction Driven Model of Execution
  - Schedule instructions like “Control Flow” Architecture
- Multithreaded Architectures
- Decoupling Memory Accesses
- SDF
- Evaluation of SDF
Performance Evaluation (Real Programs)
**Benchmark Programs**

<table>
<thead>
<tr>
<th>Scientific and DSP oriented</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix Multiplication</td>
</tr>
<tr>
<td>FFT</td>
</tr>
<tr>
<td>Picture Zooming</td>
</tr>
<tr>
<td>Livermore Loops</td>
</tr>
</tbody>
</table>

- Code is partially generated using a Sisal back-end and augmented hand-coding
- Compared SDF with MIPS-like single threaded system
  - Superscalars
  - VLIW
MIPS vs. SDF

- SDF: used sdfc compiler (and hand-coding) and sdfsim simulator to obtain cycle count
- MIPS: used dlxcc and dlxsim to obtain cycle count

- Compared the results varying:
  - problem size
  - thread granularity (increasing thread size by unrolling loops)
  - thread parallelism (increasing the number of simultaneously forked threads)
## MIPS vs. SDF

<table>
<thead>
<tr>
<th>Matrix Multiply (size N x N)</th>
<th>Zoom (PX x PY x C)</th>
<th>Livermore 5</th>
<th>Fibonacci</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>DLX Cycles</td>
<td>SDF Cycles</td>
<td>Speed</td>
</tr>
<tr>
<td>25</td>
<td>966090</td>
<td>336153</td>
<td>2.87</td>
</tr>
<tr>
<td>50</td>
<td>727390</td>
<td>243475</td>
<td>2.99</td>
</tr>
<tr>
<td>75</td>
<td>2.4E+07</td>
<td>793853</td>
<td>3.02</td>
</tr>
<tr>
<td>100</td>
<td>5.8E+07</td>
<td>18489453</td>
<td>3.14</td>
</tr>
</tbody>
</table>

| 25  | 966090    | 336153      | 2.87     | 25*25*4 | 271775  | 229601    | 1.18     | 250 | E+06    | 1E+06      | 1.72     | 25   | 959030     | E+07       | 0.696 |
| 50  | 727390    | 243475      | 2.99     | 30*30*4 | 391150  | 329961    | 1.19     | 300 | E+06    | 2E+06      | 1.72     | 30   | 1.1E+08    | 2E+08      | 0.696 |
| 75  | 2.4E+07   | 793853      | 3.02     | 35*35*4 | 532285  | 448471    | 1.19     | 350 | E+06    | 3E+06      | 1.72     |       |            |            |       |
| 100 | 5.8E+07   | 18489453    | 3.14     | 40*40*4 | 645520  | 585131    | 1.10     | 400 | E+06    | 4E+06      | 1.74     |       |            |            |       |
SDF: Thread Granularity Effect

![Graph showing the effect of thread granularity on execution cycles for matrix multiply. The x-axis represents the degree of unrolling (1Unroll to 10Unroll), and the y-axis represents the execution cycles (1 Million to 9 Million). The graph shows a decreasing trend in execution cycles as the degree of unrolling increases.]
SDF: Thread Parallelism

Thread Level Parallelism (Matrix Multiply)

Number of Concurrent Threads

6 Million
5 Million
4 Million
3 Million
2 Million
1 Million

Execution Cycles
### SDF vs Superscalar Architectures

<table>
<thead>
<tr>
<th>Superscalar Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Functional Units</td>
<td>Varied</td>
</tr>
<tr>
<td>Instruction Issue Width</td>
<td>64</td>
</tr>
<tr>
<td>Instruction Decode Width</td>
<td>64</td>
</tr>
<tr>
<td>RUU</td>
<td>64</td>
</tr>
<tr>
<td>LSQ</td>
<td>64</td>
</tr>
<tr>
<td>Branch Prediction</td>
<td>Bimodal with 2048 entries</td>
</tr>
</tbody>
</table>
## SDF vs Superscalar (Matrix Multiplication)

<table>
<thead>
<tr>
<th>Data Size</th>
<th>SDF</th>
<th>Superscalar</th>
<th>SDF</th>
<th>Superscalar</th>
<th>SDF</th>
<th>Superscalar</th>
<th>SDF</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1FP ALU</td>
<td>2FP ALU</td>
<td>2EP</td>
<td>2FP ALU</td>
<td>2EP</td>
<td>3FP ALU</td>
</tr>
<tr>
<td>50*50</td>
<td>I-O</td>
<td>1890104</td>
<td>1890104</td>
<td>1867200</td>
<td>1867200</td>
<td>574242</td>
<td></td>
</tr>
<tr>
<td></td>
<td>O-O</td>
<td>712396</td>
<td>5532202</td>
<td>5532202</td>
<td>5511587</td>
<td>4440772</td>
<td></td>
</tr>
<tr>
<td>100*100</td>
<td>I-O</td>
<td>14824104</td>
<td>5532202</td>
<td>5511587</td>
<td>4440772</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>O-O</td>
<td>49763150</td>
<td>49763150</td>
<td>49110246</td>
<td>49110246</td>
<td></td>
<td></td>
</tr>
<tr>
<td>150*150</td>
<td>I-O</td>
<td>18514510</td>
<td>49110246</td>
<td>49110246</td>
<td>14819482</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>O-O</td>
<td>39762487</td>
<td>18514510</td>
<td>18468409</td>
<td>14819482</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data Size</th>
<th>SDF</th>
<th>Superscalar</th>
<th>SDF</th>
<th>Superscalar</th>
<th>SDF</th>
<th>Superscalar</th>
<th>SDF</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1FP ALU</td>
<td>3FP ALU</td>
<td>3EP</td>
<td>4FP ALU</td>
<td>4EP</td>
<td>5FP ALU</td>
</tr>
<tr>
<td>50*50</td>
<td>I-O</td>
<td>1867200</td>
<td>1867200</td>
<td>1867200</td>
<td>1867200</td>
<td>345027</td>
<td></td>
</tr>
<tr>
<td></td>
<td>O-O</td>
<td>680321</td>
<td>507197</td>
<td>680321</td>
<td>381247</td>
<td>680321</td>
<td></td>
</tr>
<tr>
<td>100*100</td>
<td>I-O</td>
<td>14633700</td>
<td>14633700</td>
<td>14633700</td>
<td>14633700</td>
<td>2665472</td>
<td></td>
</tr>
<tr>
<td></td>
<td>O-O</td>
<td>5306381</td>
<td>3970682</td>
<td>5306381</td>
<td>2982702</td>
<td>5306380</td>
<td></td>
</tr>
<tr>
<td>150*150</td>
<td>I-O</td>
<td>49110246</td>
<td>49110246</td>
<td>49110246</td>
<td>49110246</td>
<td>8894002</td>
<td></td>
</tr>
<tr>
<td></td>
<td>O-O</td>
<td>17782453</td>
<td>17782453</td>
<td>17782453</td>
<td>17782453</td>
<td>8894002</td>
<td></td>
</tr>
</tbody>
</table>
Scalability of SDF (Matrix Multiplication)

Data size 150*150
### SDF vs Superscalar (FFT)

<table>
<thead>
<tr>
<th>Data Size</th>
<th>FFT</th>
<th>SDF</th>
<th>FFT</th>
<th>SDF</th>
<th>FFT</th>
<th>SDF</th>
<th>FFT</th>
<th>SDF</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Superscalar</td>
<td>SDF</td>
<td>Superscalar</td>
<td>SDF</td>
<td>Superscalar</td>
<td>SDF</td>
<td>Superscalar</td>
<td>SDF</td>
</tr>
<tr>
<td></td>
<td>2INT Alu</td>
<td>2SP</td>
<td>2INT Alu</td>
<td>2SP</td>
<td>3INT Alu</td>
<td>3SP</td>
<td>3INT Alu</td>
<td>3SP</td>
</tr>
<tr>
<td></td>
<td>1FP Alu</td>
<td>1EP</td>
<td>2FP Alu</td>
<td>2EP</td>
<td>2FP Alu</td>
<td>2EP</td>
<td>3FP Alu</td>
<td>3EP</td>
</tr>
<tr>
<td>8</td>
<td>I-O</td>
<td>20418</td>
<td>20418</td>
<td>19933</td>
<td>19933</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>O-O</td>
<td>9377</td>
<td>10045</td>
<td>9377</td>
<td>9526</td>
<td>8645</td>
<td>8556</td>
<td>8202</td>
</tr>
<tr>
<td>16</td>
<td>I-O</td>
<td>36038</td>
<td>36038</td>
<td>35394</td>
<td>35394</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>O-O</td>
<td>15737</td>
<td>24303</td>
<td>15737</td>
<td>22927</td>
<td>14550</td>
<td>20385</td>
<td>13997</td>
</tr>
<tr>
<td>32</td>
<td>I-O</td>
<td>78794</td>
<td>78794</td>
<td>77695</td>
<td>77695</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>O-O</td>
<td>32902</td>
<td>57444</td>
<td>32902</td>
<td>54012</td>
<td>30515</td>
<td>47740</td>
<td>29621</td>
</tr>
<tr>
<td>64</td>
<td>I-O</td>
<td>201547</td>
<td>201547</td>
<td>199069</td>
<td>199069</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>O-O</td>
<td>81519</td>
<td>133139</td>
<td>81519</td>
<td>124915</td>
<td>75952</td>
<td>110003</td>
<td>73937</td>
</tr>
<tr>
<td>128</td>
<td>I-O</td>
<td>577851</td>
<td>577851</td>
<td>570906</td>
<td>5770906</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>O-O</td>
<td>228095</td>
<td>303518</td>
<td>228095</td>
<td>284382</td>
<td>214191</td>
<td>249774</td>
<td>208285</td>
</tr>
<tr>
<td>256</td>
<td>I-O</td>
<td>1816758</td>
<td>1816758</td>
<td>1794386</td>
<td>1794386</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>O-O</td>
<td>703548</td>
<td>682417</td>
<td>703548</td>
<td>638705</td>
<td>664954</td>
<td>559665</td>
<td>644899</td>
</tr>
<tr>
<td>512</td>
<td>I-O</td>
<td>6165028</td>
<td>6165028</td>
<td>6086525</td>
<td>6086525</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>O-O</td>
<td>2350656</td>
<td>1516660</td>
<td>2350656</td>
<td>1418356</td>
<td>2235095</td>
<td>12409448</td>
<td>2161717</td>
</tr>
</tbody>
</table>
Scalability of SDF (FFT)

Data size 256
# SDF vs Superscalar (Zoom)

<table>
<thead>
<tr>
<th></th>
<th>SS</th>
<th>SDF</th>
<th>SS</th>
<th>SDF</th>
<th>SS</th>
<th>SDF</th>
<th>SS</th>
<th>SDF</th>
<th>SS</th>
<th>SDF</th>
</tr>
</thead>
<tbody>
<tr>
<td>1INT</td>
<td>528100</td>
<td>499976</td>
<td>499976</td>
<td>499573</td>
<td>499573</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1SP</td>
<td>416625</td>
<td>464765</td>
<td>221253</td>
<td>314032</td>
<td>221253</td>
<td>230072</td>
<td>170235</td>
<td>163907</td>
<td>170235</td>
<td>153542</td>
</tr>
<tr>
<td>2INT</td>
<td>2094969</td>
<td>1994236</td>
<td>1994236</td>
<td>1993829</td>
<td>1993829</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2SP</td>
<td>1660478</td>
<td>1855370</td>
<td>877150</td>
<td>1254357</td>
<td>877150</td>
<td>915057</td>
<td>696002</td>
<td>655907</td>
<td>696002</td>
<td>611707</td>
</tr>
<tr>
<td>3INT</td>
<td>4989542</td>
<td>7486216</td>
<td>7486216</td>
<td>4785812</td>
<td>4785812</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3SP</td>
<td>3994462</td>
<td>4171875</td>
<td>2108470</td>
<td>2821032</td>
<td>2108470</td>
<td>2061057</td>
<td>1693042</td>
<td>1476057</td>
<td>1693042</td>
<td>1374402</td>
</tr>
<tr>
<td>4INT</td>
<td>8387641</td>
<td>7986709</td>
<td>7986709</td>
<td>7986302</td>
<td>7986302</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4SP</td>
<td>6613286</td>
<td>7414280</td>
<td>3503558</td>
<td>5014057</td>
<td>3503558</td>
<td>3661977</td>
<td>2779131</td>
<td>230072</td>
<td>2441917</td>
<td></td>
</tr>
<tr>
<td>5INT</td>
<td>1993827</td>
<td>460317</td>
<td>459667</td>
<td>646252</td>
<td>368417</td>
<td>646252</td>
<td>368147</td>
<td>646252</td>
<td>308777</td>
<td></td>
</tr>
<tr>
<td>5SP</td>
<td>656328</td>
<td>656328</td>
<td>459667</td>
<td>646252</td>
<td>368417</td>
<td>646252</td>
<td>368147</td>
<td>646252</td>
<td>308777</td>
<td></td>
</tr>
<tr>
<td>6INT</td>
<td>4785811</td>
<td>4785811</td>
<td>4785811</td>
<td>4785811</td>
<td>4785811</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6SP</td>
<td>163811</td>
<td>163811</td>
<td>163811</td>
<td>1615054</td>
<td>827232</td>
<td>1615054</td>
<td>826827</td>
<td>1615054</td>
<td>693567</td>
<td></td>
</tr>
<tr>
<td>7INT</td>
<td>7986300</td>
<td>7986300</td>
<td>7986300</td>
<td>7986300</td>
<td>7986300</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7SP</td>
<td>2624712</td>
<td>1834797</td>
<td>2624712</td>
<td>1833337</td>
<td>2587702</td>
<td>1468757</td>
<td>2587702</td>
<td>1468057</td>
<td>2587702</td>
<td>1229557</td>
</tr>
</tbody>
</table>

University of Pisa, Sept 28, 2004
Scalability of SDF (Zoom)

Data size 200x200x4

Scalability of SDF (Zoom)

Execution Cycles

In Order  Out of Order  SDF

Number of Functional Units

2+1  2+2  3+2  3+3  4+3  4+4  5+4  5+5  6+5  6+6

University of Pisa, Sept 28, 2004
We used two different VLIW simulators

TI’s DSP TMS320C64X
  8-wide VLIW
Trimaran Simulation Environment
  9 functional units
  32-loop unrolling

Compared with SDF using 4EPs and 4SP’s
SDF vs VLIW Architectures (Matrix Mult)

<table>
<thead>
<tr>
<th>Data Size</th>
<th>SDF</th>
<th>Trimaran</th>
<th>TMS 'C6000 optimized</th>
<th>SDF/Trimaran</th>
<th>SDF/TMS 'C6000</th>
</tr>
</thead>
<tbody>
<tr>
<td>50*50</td>
<td>430957</td>
<td>331910</td>
<td>1033698</td>
<td>1.29841523</td>
<td>0.416908C</td>
</tr>
<tr>
<td>100*100</td>
<td>3330992</td>
<td>2323760</td>
<td>16199926</td>
<td>1.43344924</td>
<td>0.2056177</td>
</tr>
<tr>
<td>150*150</td>
<td>11115592</td>
<td>4959204</td>
<td>86942144</td>
<td>2.24140648</td>
<td>0.1278504</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data Size</th>
<th>SDF</th>
<th>SDF 10/VLIW</th>
<th>SDF 5/VLIW</th>
<th>SDF 2/VLIW</th>
</tr>
</thead>
<tbody>
<tr>
<td>50*50</td>
<td>430,957</td>
<td>1,525,560</td>
<td>2,956,616</td>
<td>0.4169</td>
</tr>
<tr>
<td>100*100</td>
<td>3,330,992</td>
<td>11,096,312</td>
<td>22,297,111</td>
<td>0.2056</td>
</tr>
<tr>
<td>150*150</td>
<td>11,115,592</td>
<td>36,242,464</td>
<td>73,773,061</td>
<td>0.1279</td>
</tr>
</tbody>
</table>
## SDF vs VLIW Architectures (FFT)

<table>
<thead>
<tr>
<th>Size</th>
<th>Data</th>
<th>SDF</th>
<th>Trimaran</th>
<th>TMS 'C6000</th>
<th>SDF/Trimaran</th>
<th>SDF/TMS 'C6000</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>8148</td>
<td>4622</td>
<td>26717</td>
<td>1.762873215</td>
<td>0.304974361</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>19323</td>
<td>12391</td>
<td>73456</td>
<td>1.559438302</td>
<td>0.263055435</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>45028</td>
<td>31665</td>
<td>213933</td>
<td>1.422011685</td>
<td>0.210477112</td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>103491</td>
<td>81375</td>
<td>619241</td>
<td>1.271778802</td>
<td>0.167125562</td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>234766</td>
<td>214685</td>
<td>2040729</td>
<td>1.093537043</td>
<td>0.115040263</td>
<td></td>
</tr>
<tr>
<td><strong>256</strong></td>
<td><strong>525457</strong></td>
<td><strong>595211</strong></td>
<td><strong>6943638</strong></td>
<td><strong>0.882807945</strong></td>
<td><strong>0.075674596</strong></td>
<td></td>
</tr>
<tr>
<td><strong>512</strong></td>
<td><strong>1163956</strong></td>
<td><strong>1768441</strong></td>
<td></td>
<td><strong>0.658181981</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# SDF vs VLIW Architectures (Zoom)

<table>
<thead>
<tr>
<th>Size</th>
<th>Data</th>
<th>SDF</th>
<th>Trimaran Optimized</th>
<th>TMS C'6000</th>
<th>SDF/Trimaran 'C6000</th>
</tr>
</thead>
<tbody>
<tr>
<td>50<em>50</em>4</td>
<td>115452</td>
<td>157770</td>
<td>144201</td>
<td>0.7317741</td>
<td>0.8006324</td>
</tr>
<tr>
<td>100<em>100</em>4</td>
<td>459667</td>
<td>630520</td>
<td>641625</td>
<td>0.72902842</td>
<td>0.7164106</td>
</tr>
<tr>
<td>150<em>150</em>4</td>
<td>1032567</td>
<td>1418270</td>
<td>1480525</td>
<td>0.72804685</td>
<td>0.6974330</td>
</tr>
<tr>
<td>200<em>200</em>4</td>
<td>1833337</td>
<td>2521020</td>
<td>2959430</td>
<td>0.72722033</td>
<td>0.6194899</td>
</tr>
<tr>
<td>250<em>250</em>4</td>
<td>2862857</td>
<td>3938770</td>
<td>4729593</td>
<td>0.72684036</td>
<td>0.6053072</td>
</tr>
</tbody>
</table>

University of Pisa, Sept 28, 2004
# SDF Scalability With Register Sets

<table>
<thead>
<tr>
<th>Number of Register Sets</th>
<th>50x50</th>
<th>100x100</th>
<th>150x150</th>
</tr>
</thead>
<tbody>
<tr>
<td>1SP-1EP</td>
<td>4985911</td>
<td>38929961</td>
<td>130334934</td>
</tr>
<tr>
<td>2SP-2EP</td>
<td>4985911</td>
<td>38929961</td>
<td>130334934</td>
</tr>
<tr>
<td>3SP-3EP</td>
<td>4985911</td>
<td>38929961</td>
<td>130334934</td>
</tr>
<tr>
<td>4SP-4EP</td>
<td>4985911</td>
<td>38929961</td>
<td>130334934</td>
</tr>
</tbody>
</table>

## Matrix Multiplication
## Cache Memories in SDF

<table>
<thead>
<tr>
<th>N</th>
<th>DLX Refs</th>
<th>DLX Misses</th>
<th>DLX Miss Rate</th>
<th>SDF Refs</th>
<th>SDF Misses</th>
<th>SDF Miss Rate</th>
<th>SDF Refs</th>
<th>SDF Misses</th>
<th>SDF Miss Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>286714</td>
<td>61</td>
<td>0.0002</td>
<td>156470</td>
<td>382</td>
<td>0.0024</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>1627928</td>
<td>237</td>
<td>0.00015</td>
<td>1094360</td>
<td>614</td>
<td>0.0006</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>75</td>
<td>5462503</td>
<td>530</td>
<td>0.0001</td>
<td>3526250</td>
<td>1010</td>
<td>0.0003</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>12910828</td>
<td>940</td>
<td>0.00007</td>
<td>8164640</td>
<td>1558</td>
<td>0.0002</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Matrix Multiply

<table>
<thead>
<tr>
<th>N</th>
<th>SDF Refs</th>
<th>Unified Refs</th>
<th>Unified Misses</th>
<th>Unified Miss Rate</th>
<th>I-Struct Refs</th>
<th>I-Struct Misses</th>
<th>I-Struct Miss Rate</th>
<th>Frame Refs</th>
<th>Frame Misses</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>156470</td>
<td>382</td>
<td>0.0024</td>
<td>81</td>
<td>0.0005</td>
<td>11</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>1094360</td>
<td>614</td>
<td>0.0006</td>
<td>319</td>
<td>0.0003</td>
<td>11</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>75</td>
<td>3526250</td>
<td>1010</td>
<td>0.0003</td>
<td>3132</td>
<td>0.0009</td>
<td>11</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>8164640</td>
<td>1558</td>
<td>0.0002</td>
<td>6215</td>
<td>0.0008</td>
<td>11</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Cache Memories in SDF

<table>
<thead>
<tr>
<th>Loop=N</th>
<th>DLX Refs</th>
<th>DLX Misses</th>
<th>DLX Miss Rate</th>
<th>SDF Refs</th>
<th>SDF Misses</th>
<th>SDF Miss Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>22429</td>
<td>8</td>
<td>0.00036</td>
<td>24177</td>
<td>22</td>
<td>0.00091</td>
</tr>
<tr>
<td>100</td>
<td>88829</td>
<td>13</td>
<td>0.000146</td>
<td>91913</td>
<td>31</td>
<td>0.000337</td>
</tr>
<tr>
<td>150</td>
<td>199229</td>
<td>17</td>
<td>8.53E-05</td>
<td>203249</td>
<td>40</td>
<td>0.000197</td>
</tr>
<tr>
<td>200</td>
<td>353629</td>
<td>22</td>
<td>6.22E-05</td>
<td>358185</td>
<td>49</td>
<td>0.000137</td>
</tr>
<tr>
<td>250</td>
<td>550458</td>
<td>27</td>
<td>4.90E-05</td>
<td>556721</td>
<td>58</td>
<td>0.000104</td>
</tr>
<tr>
<td>300</td>
<td>794429</td>
<td>32</td>
<td>4.03E-05</td>
<td>798857</td>
<td>67</td>
<td>8.39E-05</td>
</tr>
<tr>
<td>350</td>
<td>1080829</td>
<td>36</td>
<td>3.33E-05</td>
<td>1084593</td>
<td>76</td>
<td>7.01E-05</td>
</tr>
<tr>
<td>400</td>
<td>1411229</td>
<td>41</td>
<td>2.91E-05</td>
<td>1413929</td>
<td>88</td>
<td>6.22E-05</td>
</tr>
<tr>
<td>450</td>
<td>1785629</td>
<td>46</td>
<td>2.58E-05</td>
<td>1786865</td>
<td>97</td>
<td>5.43E-05</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>N</th>
<th>SDF Refs</th>
<th>SDF Misses</th>
<th>SDF Miss Rate</th>
<th>Unified Refs</th>
<th>Unified Misses</th>
<th>Unified Miss Rate</th>
<th>I-Struct Refs</th>
<th>I-Struct Misses</th>
<th>I-Struct Miss Rate</th>
<th>Frame Misses</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>24177</td>
<td>22</td>
<td>0.00091</td>
<td>12</td>
<td>0.0005</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>91913</td>
<td>31</td>
<td>0.000337</td>
<td>21</td>
<td>0.0002</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>150</td>
<td>203249</td>
<td>40</td>
<td>0.00097</td>
<td>30</td>
<td>0.0001</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>200</td>
<td>358185</td>
<td>49</td>
<td>0.000137</td>
<td>39</td>
<td>0.0001</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>250</td>
<td>556721</td>
<td>58</td>
<td>0.000104</td>
<td>48</td>
<td>8.62E-05</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>300</td>
<td>798857</td>
<td>67</td>
<td>8.39E-05</td>
<td>57</td>
<td>7.14E-05</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>350</td>
<td>1084593</td>
<td>76</td>
<td>7.01E-03</td>
<td>66</td>
<td>6.09E-05</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>400</td>
<td>1413929</td>
<td>88</td>
<td>6.22E-05</td>
<td>78</td>
<td>5.52E-05</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>450</td>
<td>1786865</td>
<td>97</td>
<td>5.43E-05</td>
<td>87</td>
<td>4.87E-05</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Summary And Conclusions

• 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
  Preload/Poststore/Execute portions of a thread
  – fewer context switches (but possibly more threads)
• SDF architecture OUTPERFORMS MIPS, and Superscalar architectures
  – 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
Current And Future Work

- Optimize current compiler
  - Including speculative thread generation
  - Pointer and alias analysis
- Extend I-structure semantics to handle alias issues
  - Regular load/store
  - I-fetch, I-store
  - L/J structures
  - Speculative load and store
- Superscalar EP implementation
- Explore Hardware complexity and MCD implementations
  - Develop VHDL simulations
Current And Future Work

• Globally Asynchronous and Locally Synchronous systems
  SPs and EPs can be run at different frequencies and voltages

• Clustered implementation of SDF
  • Each cluster contains a small number of SPs and EPs
  • Register sets are local to each cluster
  • Intra-cluster communication through shared Frames
    And I-structure memories
References


