A Decoupled Scheduled Dataflow Multithreaded Architecture

Krishna Kavi, H.S. Kim and J. Arul
The University of Alabama in Huntsville
{kavi, hskim, arulj} @ece.uah.edu

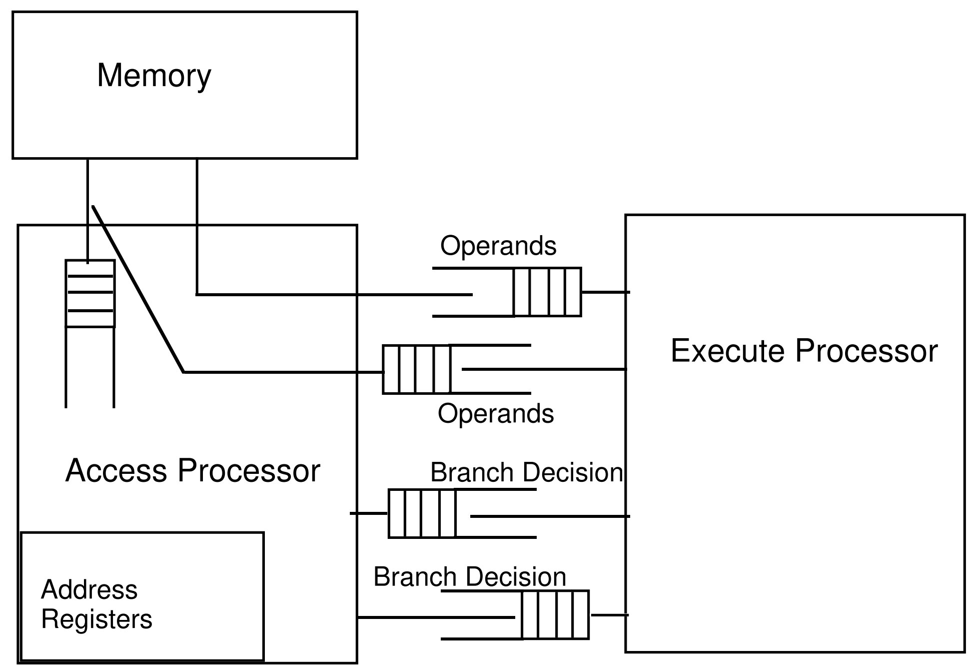
A. R. Hurson
Penn State University
hurson@cse.psu.edu

Jump to first page

Decoupled Memory Access

Separate Processor to handle all memory accesses

The earliest suggestion by J.E. Smith -- DAE architecture



Limitations of Smith's DAE processor

- Designed for STRETCH system with no pipelines

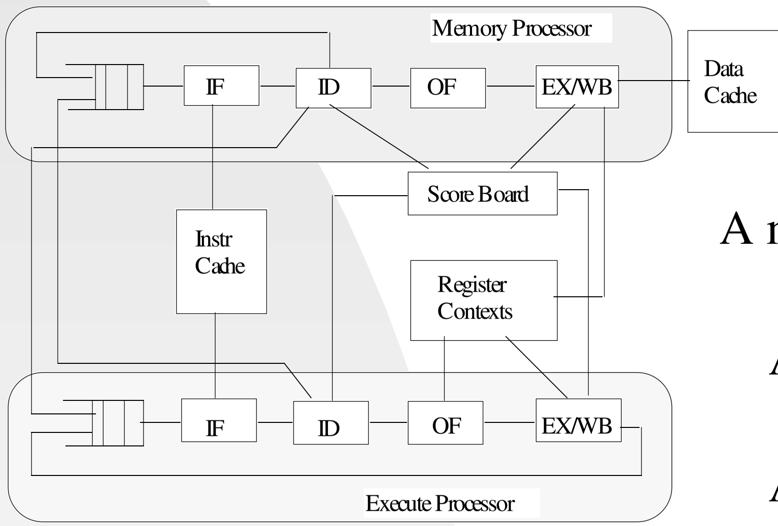
    Single instruction stream

- Instructions for Execute processor must be coordiated 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

More Recent implementations



Rhamma Processor

(Univ. Karlsruhe)

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

Limitations of Rhamma Processor

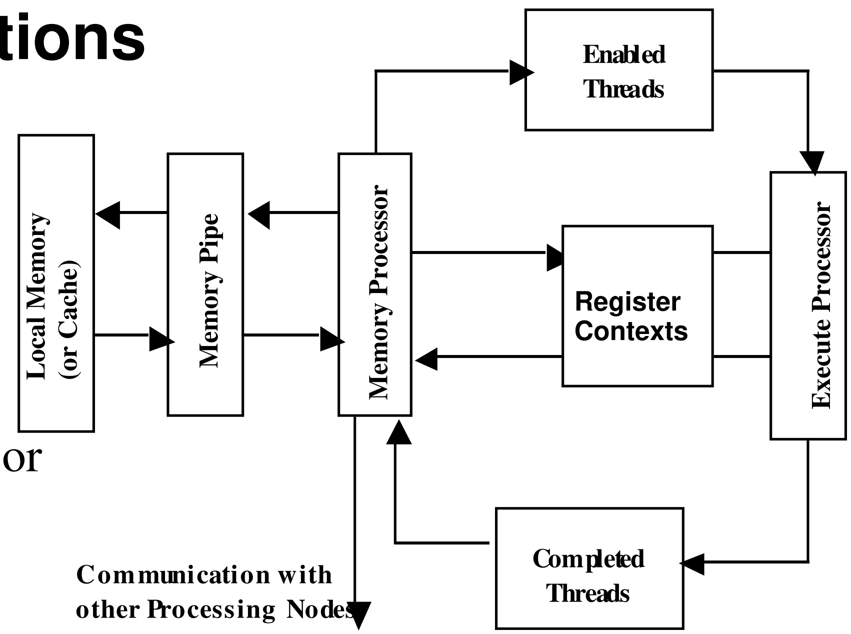
- Blocking Thread Model

  Requires More context switches

- Checking for data dependencies requires complex hardware
- Bubbles in pipelines are unavoidable on context switches and cache misses

More Recent implementations

Pre-Load/Post-Store Processor



- A non-blocking multithreaded processor
- Separate Memory and Execution Pipelines
- A thread is enabled for execution only after all data is loaded into registers
- Storing of data is delayed until the thread completes execution
- Branch instructions cause new threads

A Simple Example

```
LD      F0, 0(R1)
LD      F6, -8(R1)
MULTD   F0, F0, F2
MULTD   F6, F6, F2
LD      F4, 0(R2)
LD      F8, -8(R2)
ADDD    F0, F0, F4
ADDD    F6, F6, F8
SUBI    R2, R2, 16
SUBI    R1, R1, 16
SD      8(R2), F0
BNEZ    R1, LOOP
SD      0(R2), F6
```

Conventional

```
LD      F0, 0(R1)
LD      F6, -8(R1)
LD      F4, 0(R2
LD      F8, -8(R2)
MULTD   F0, F0, F2
MULTD   F6, F6, F2
SUBI    R2, R2, 16
SUBI    R1, R1, 16
ADDD    F0, F0, F4
ADDD    F6, F6, F8
SD      8(R2), F0
SD      0(R2), F6
```

New Architecture

Features of PL/PS

- Multiple hardware contexts
- 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!

Scheduled Datalow

- Brings dataflow closer to conventional RISC architecture
- Utilizes Decoupled processors to eliminate pipeline bubbles on cache misses -- combines Preload/post-store with dataflow
- Eliminates WAR and WAW dependencies in pipelines
  - The result of using dataflow execution
- Uses Non-blocking Multithreaded model

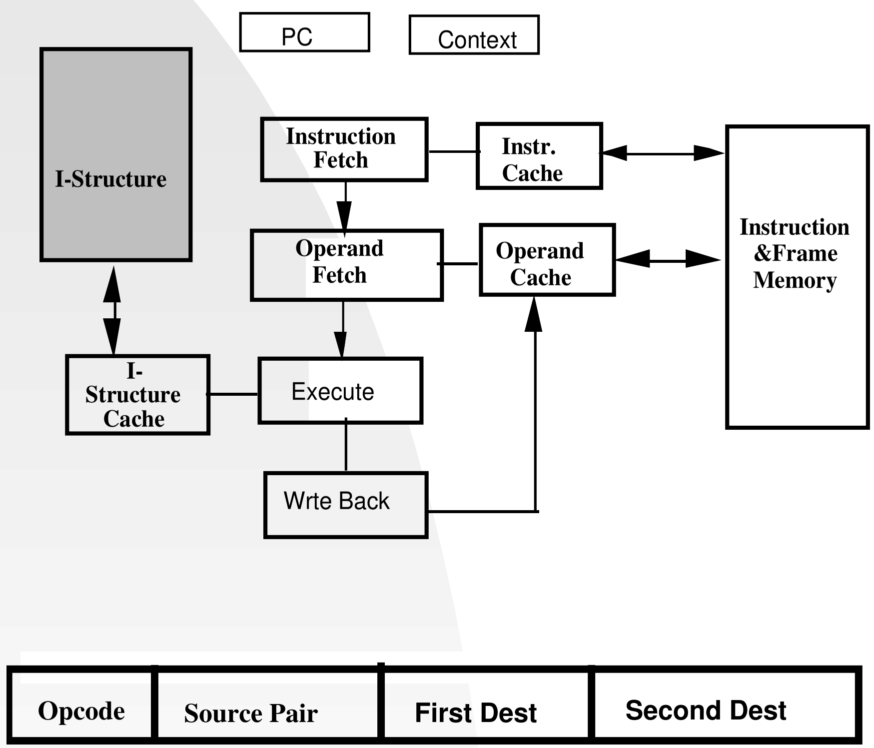
Limitations of Previous Dataflow Architectures

- Memory Hierarchies cannot be used
- Too fine-grained
- Localities are difficult to synthesize
- Asynchronous execution

The first 3 limitations have been addressed by other researchers

Scheduled dataflow addresses the last limitation

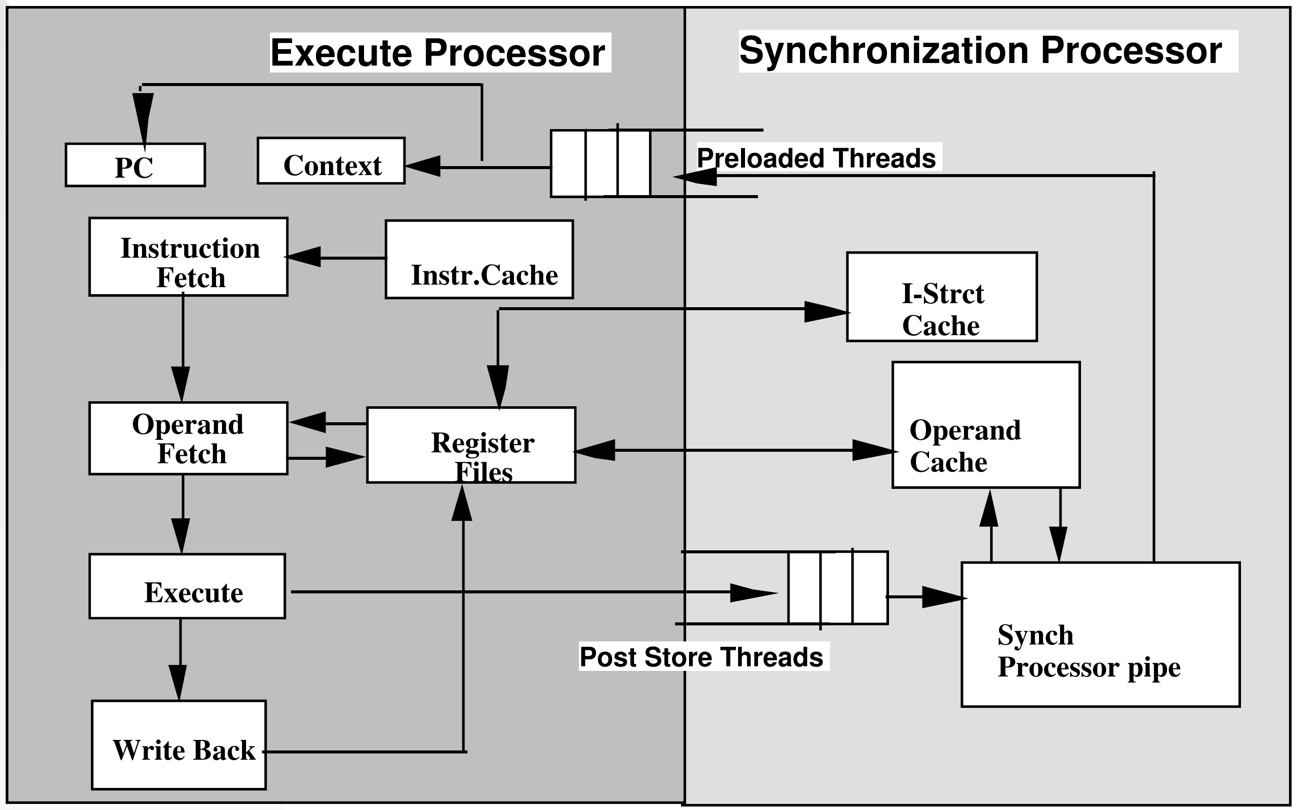
Scheduled Datalfow Architecture



- Each instruction is associated with a pair of “source registers”.
- Predecessor instructions store their results in these registers
- An instruction is not enabled immediately when the two source registers are loaded.
- Instructions are scheduled similar to conventional processors.

However, instructions retain functional properties

Decoupled Processors For Scheduled Datafllow



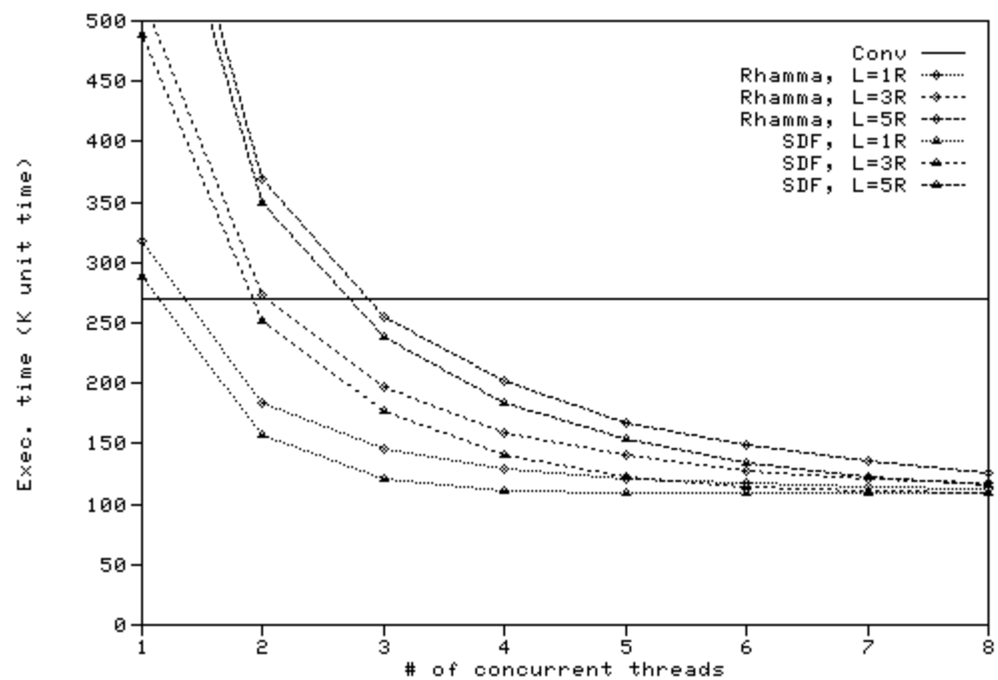
Preliminary Performance Comparisons

- Monte Carlo simulations using simple models for Rhamma, Scheduled Dataflow and conventional RISC processors
- Some of the parameters are based on published data (% load/stores, avg memory latency, cache miss rates).
- Some parameters are based on simple programs coded in our architecture
- Some parameters are based on guesswork

Performance Results

Effect Of Thread Level Parallelism

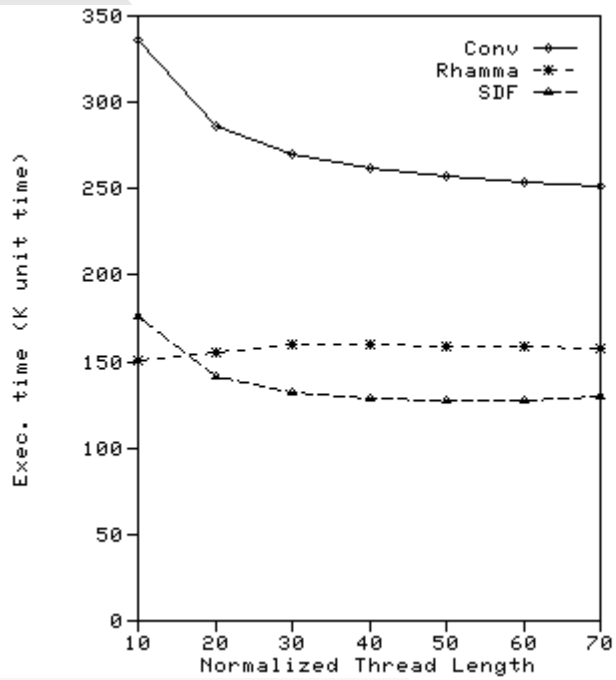
- Multithreaded architectures (Rhamma and SDF) perform poorly for small degrees of parallelism
- Conventional architecture is assumed to be single threaded
- SDF is non-blocking and incurs no context switches during exection



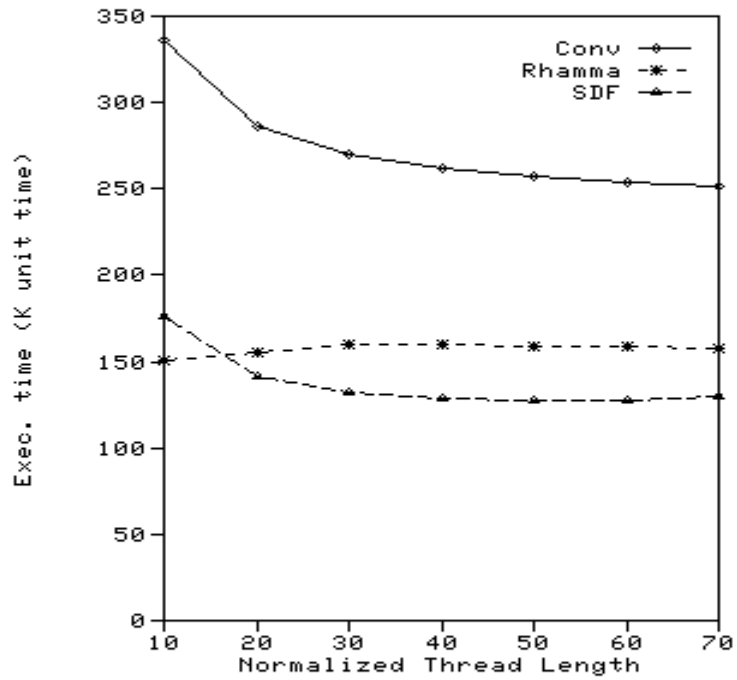
L is Latency and it is set to 1, 3, and 5 times the Thread run lengths

Jump to first page

Effect Of Thread Granularity

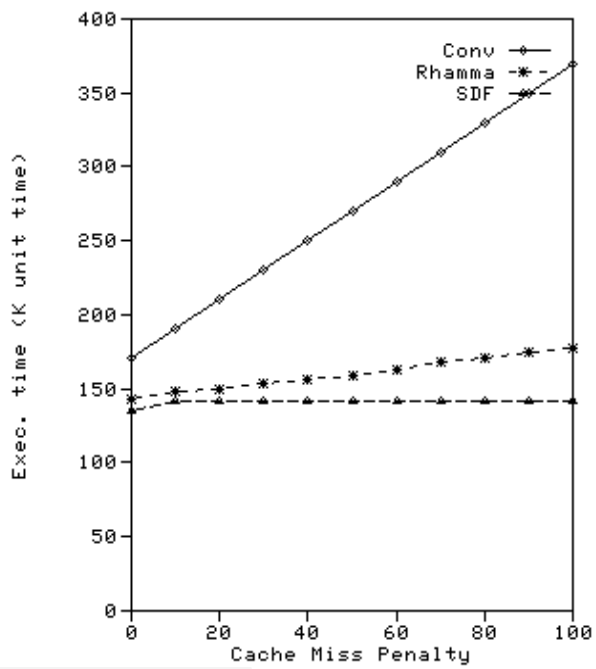


Effect Of Load/Store Instructions

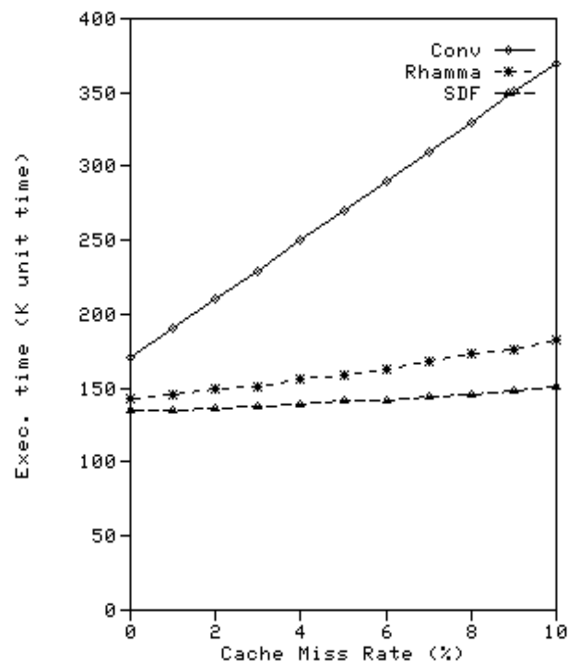


- SDF is finer-grained. But modest thread run-lengths of 20 instruction are sufficient to outperform Rhamma
- Decoupling is the main reason for the lack of performance losses even when load/store instructions dominate

Effect Of Cache Misses and Miss Penalties



(a) Impact of miss rates



(b) Impact of miss penalties

- SDF permits for data alignment and prefetching leading to lower cache misses
- Preload/Post store eliminates unnecessary context switches during thread execution

Conclusions

- Combined Dataflow Architecture With Conventional control-flow like scheduling and Decoupled memory accesses
- The performance gains are primarily due to
  - Overlapped Memory/Execute processing
  - Non-Blocking and fine grained threads
    - One difference between Rhamma and SDF
  - Pre-load/Post-Store Decoupling
    - Another difference between Rhamma and SDF
    - Permits for data placement and prefetching
- Eliminates Complex Instruction Scheduling hardware
  - For register renaming, detecting WAR/WAW dependencies, Branch prediction
    - A third difference between Rhamma and SDF

Current Status And Future Research

- A detailed instruction simulator is being designed
- Converting Compiler backends to generate code for SDF
- Should be able to evaluate the architecture more thoroughly using large benchmarks
    - Not just SPEC, but special purpose and embedded applications
- Investigate compiler optimizations
    - Data placement/prefetch
    - Predictive preloading
- Estimate hardware savings