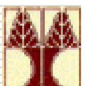
The Case for Chip Multiprocessors based on the Data-Driven Multithreading Model: Scale the memory and power walls

Skevos Evripidou

Department of Computer Science
University of Cyprus

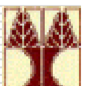
Outline

- Motivation & Challenges in Computer Architecture
- Overview of Data-flow
- Data-Driven Multithreading
  - Thread Synchronization Unit (TSU)
  - Cache-flow
- DDM Chip Multiprocessor

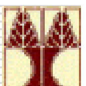


Motivation

- Challenges:
  - Utilize Chips with Billions of transistors
  - Neutralize the effect of the Memory Wall
  - Control the Power consumption and the heat generation (Power Wall)
- Current trend in building high-end microprocessors based on the von-Neumann model:
  - Exploit ILP through increased complexity and larger caches
  - Deconstruct the Sequential program with hardware assisted implicit synchronization.



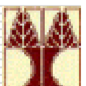
- While this approach has worked well in the past, it is currently only resulting in diminishing returns.
  - The optimizations employed to exploit ILP from sequential code do not scale well
  - Huge caches take too much real-estate and consume lots of Power
- Chip Multiprocessors (CMP) offers an attractive alternative.
- CMP based on the von Neumann model
  - will inherit the models bottlenecks: synchronization and memory latencies and the complex design.
  - In order to overcome the Multiprocessor synchronization and latencies will require even more comlpex designs
- Alternative models?



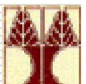
Conventional Wisdom (CW) in Computer Architecture

- Old CW: Multiplies are slow, Memory access is fast
- New CW: "Memory wall" Memory slow, multiplies fast
  (200 clocks to DRAM memory, 4 clocks for multiply)
- Old CW: Power is free, Transistors expensive
- New CW: "Power wall" Power expensive, Xtors free
  (Can put more on chip than can afford to turn on)
- Old CW: Uniprocessor performance 2X / 1.5 yrs
- New CW: Power Wall + Memory Wall = Brick Wall
  - Uniprocessor performance only 2X / 5 yrs
- Sea change in chip design: multiple "cores"
  (2X processors per chip / ~ 2 years)
  - More simpler processors are more power efficient

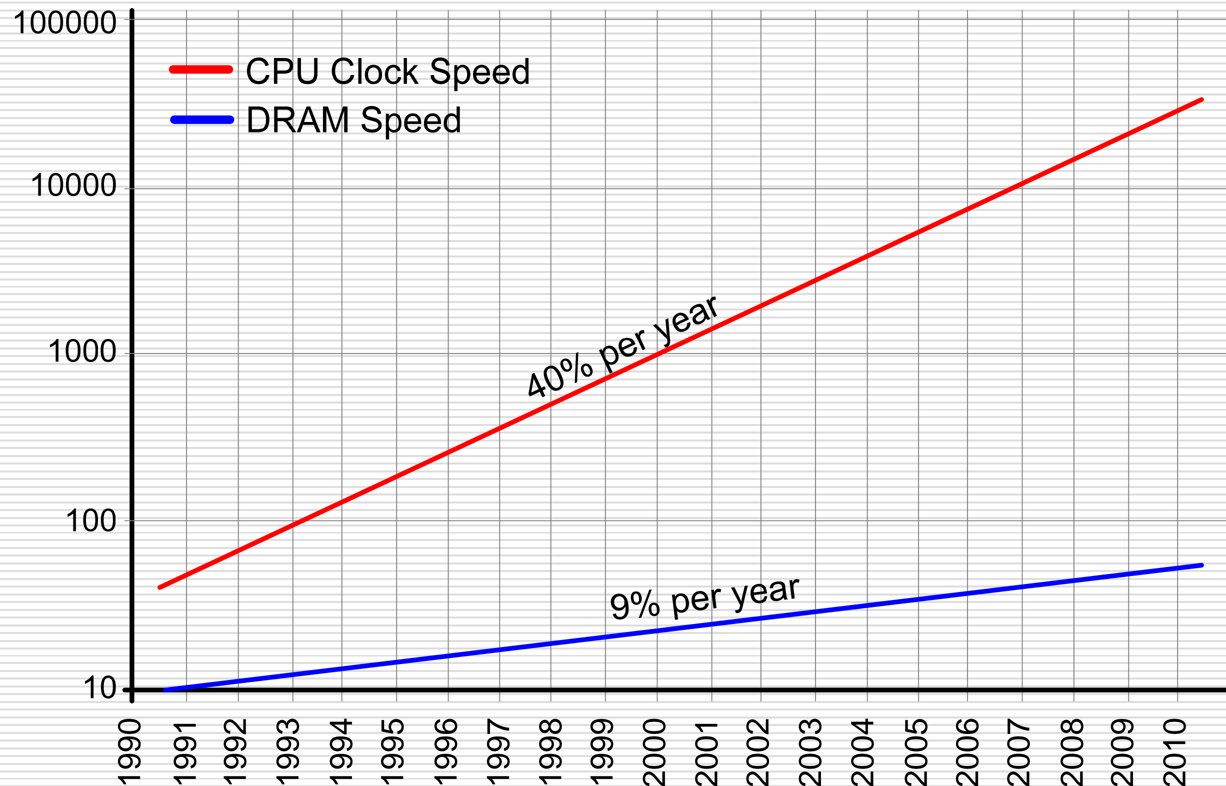
Courtesy: RAMPS project -- Dave Patterson



The Memory Wall

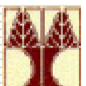


Fact 1: CPU-Memory Speed

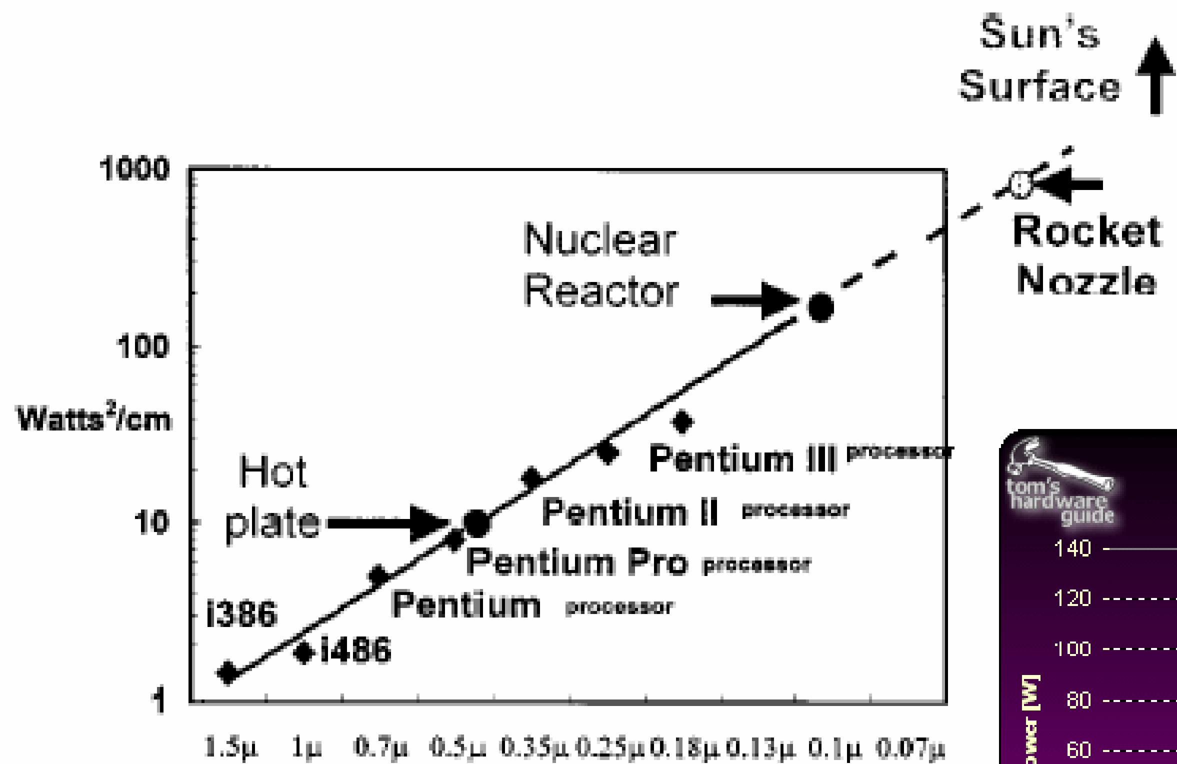


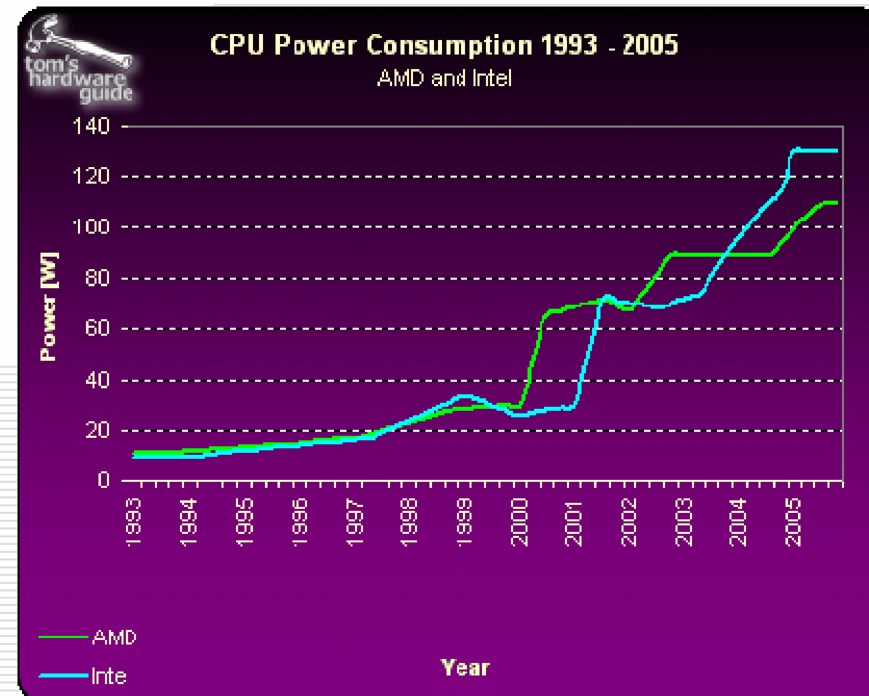
Source: J. Patterson, “Modern Microprocessors”, www.pattosoft.com.au/Articles/ModernProcessors

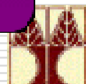
Overcoming the memory wall:
- 8KB L1 (Intel 486, 1989)
- On-board L2 (Pentium Pro, 1995)
- On-package L2 (Pentium II, 1997)
- On-die L2 (Pentium III, 1999)



Power Consumption- *Power Wall*







Courtesy: RAMPS project -- Dave Patterson

Uniprocessor Performance (SPECint)

Performance (vs. VAX-11/780)

10000
1000
100
10
1

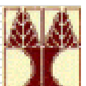
1978 1980 1982 1984 1986 1988 1990 1992 1994 1996 1998 2000 2002 2004 2006

25%/year

52%/year

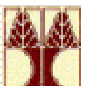
20%/year

- **VAX : 25%/year 1978 to 1986**
- **RISC + x86: 52%/year 1986 to 2002**
- **RISC + x86: 20%/year 2002 to present**



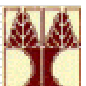
Data-flow Architectures

- Proposed in the 70s (Most people credit Jack Dennis of MIT as the "father" of Data-Flow)
  - Asynchrony: Execution is driven by data availability.
  - Functional: No side effects.
- Implementation: Provide "Context-switch" support at the instruction level
- Data-flow programs are represented as graphs:
  - The nodes (actors) are the instructions of the program
  - The arcs carry data from producer to consumer actor
- Enabling rule: an instruction is enabled (i.e. executable) if all operands are available.
- An instruction can be fired (i.e. executed) only after it becomes enabled.



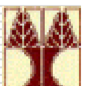
Static Data-Flow

- *A data-flow graph* is represented as a collection of *activity templates*:
  - the opcode of the represented instruction,
  - operand slots for holding operand values,
  - and destination address fields,
- Each *token* consists only of a value and a destination address.
  - Static dataflow approach allows at most one token on any one arc.
- *Extending the basic firing rule* : **An enabled node is fired if there is no token on any of its output arcs.**
- Implementation of the restriction by *acknowledge signals* (additional tokens), traveling along additional arcs from consuming to producing nodes.



MIT Static Dataflow Machine

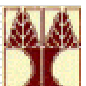
- Consecutive iterations of a loop can only be pipelined.
- Due to acknowledgment tokens, the token traffic is increased (1.5x-2x)
- Advantage: simple model



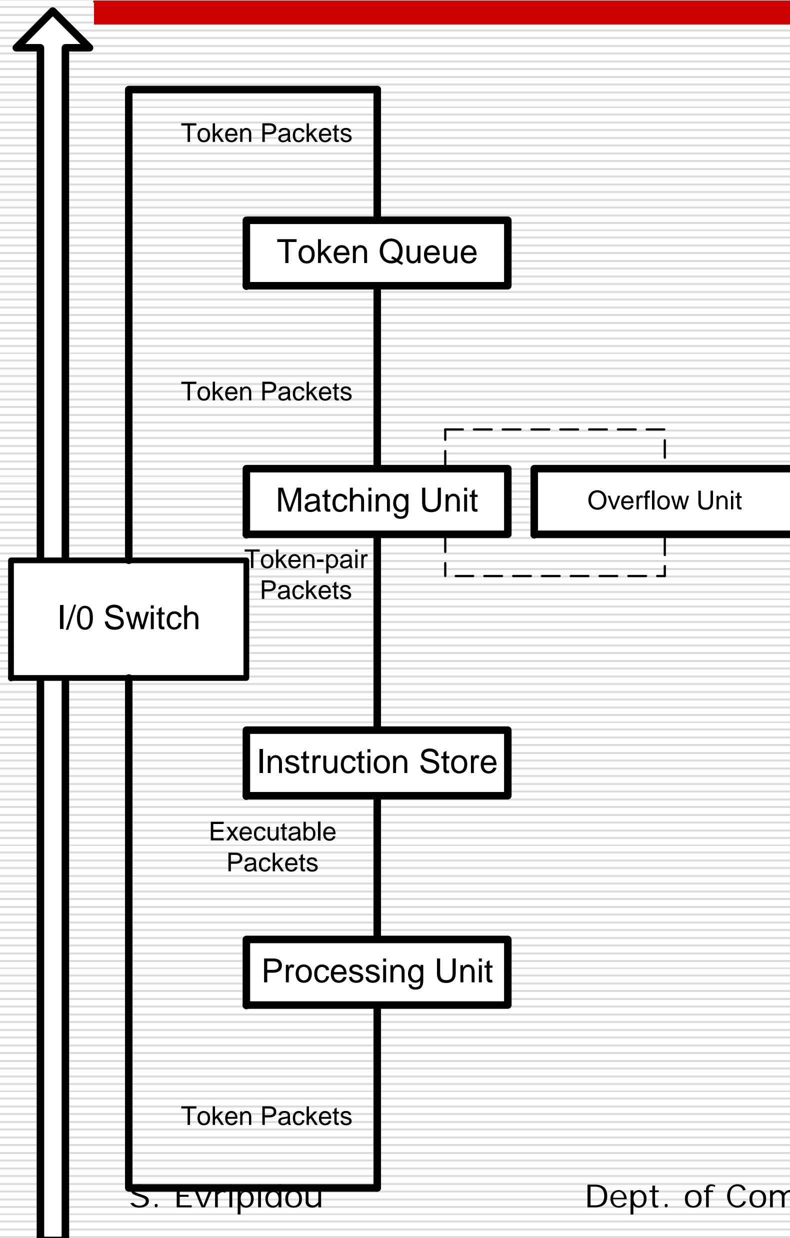
Dynamic Data-Flow

- Each loop iteration or subprogram invocation should be able to execute in parallel as a separate instance of a reentrant subgraph.
- Each token has a tag:
  - address of the instruction for which the particular data value is destined and context information
  - $V_{[c.s.i]p}$ C: context, s: instruction pointer and i: Iteration identifier, p: port number
  - Tagged Token Data-flow (TTDF)
- Each arc can be viewed as a bag that may contain an arbitrary number of tokens with different tags.
- The enabling and firing rule is now:

  A node is enabled and fired as soon as tokens with identical tags are present on all input arcs.



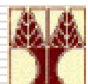
Manchester Dataflow Machine [Gurd & Watson 1979]



- Operational 1981
- Performance of 1.2 MIPS
- Matching Unit 1M tokens
- Parallel Hashing: mapping of incoming tag to a set of 8 slots
- Associative matching at the Slot

Program	Vax 11/780	Dataflow 1 FU	Dataflow 12 FUs
RSIM/1	0.04	1.36	0.16
RSIM/1	0.10	8.26	0.89
RSIM/1	0.08	6.12	0.68
RSIM/1	0.28	8.67	0.88

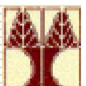
[Gurd Kirkham & Watson 1985]



Limitations of TTDF machines

1. Implementation of Waiting-Matching Store.
   - Associate memory is ideal but unfeasible
   - Hashing techniques are not fast enough to be a single pipeline stage.
   - Amount of parallelism is unpredictable, might fill up the WM-store and cause deadlock.
     - Overflow is possible but complicated.
2. Unbounded size of the activity names.
3. Different types of Stores (Matching store, Program store,Token Queue) made it difficult for memory management.
4. Poor performance with sequential code

[Arvind, Bic, Ungerer 1991]

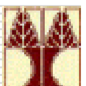


DDF Summary

- Elegant solution: parallel processing with implicit synchronization
- It can exploit the ultimate amount of parallelism.
  - Loop throttling to limit the amount of parallelism!
- Immunity to high communication and memory latencies

- Token matching and processing is expensive.
  - Associative matching is needed for token matching
  - It is very difficult to build a hierarchy of Tokens stores/matching Units.
- A DDF machine executes a lot more instructions than control-flow.

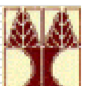
- The innovative prototypes showed very good relative performance
  - In absolute performance they did not fare well when compared to commercial offerings of the same era.
- Difficult to benefit directly from efficient constructs and building blocks of the von Neumann model



Two Fundamental issues in Multiprocessing

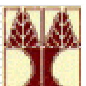
- Arvind and Iannucci wrote a series of papers in the 1980s about the two fundamental issues in Multiprocessing
  - Synchronization
  - Communication

- The Faster/Larger the machine the more serious the problem gets.



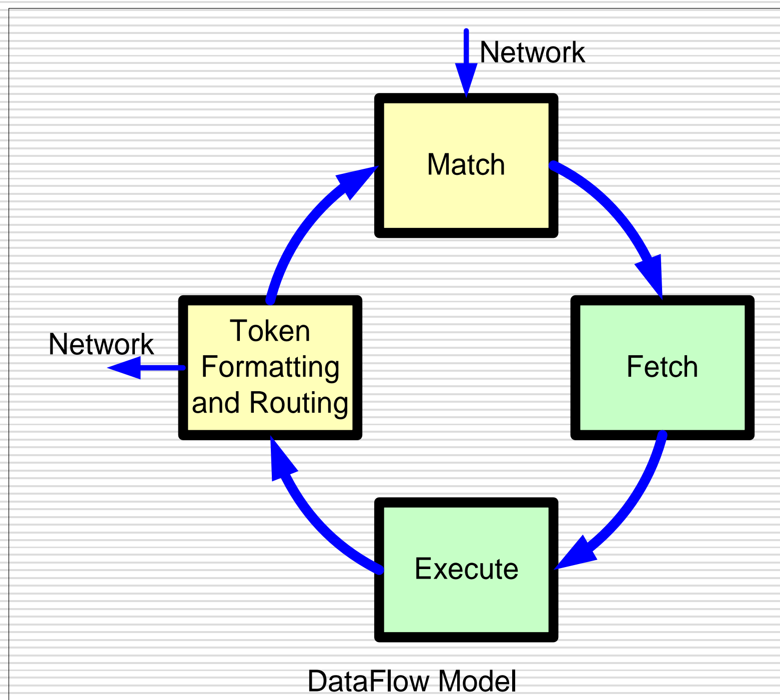
Hybrid Data-Flow/Control-flow systems

- Use the data-flow principles at higher resolution than the instruction level:
  - Macro Actors: a collection of operations
  - Fat Tokens: tokens with multiple operands (vectors etc)
- JUMBO Newcastle Data-Control Flow computer [1982]
- PDF Piecewise Dataflow Architecture [LLNL 1983]
- Muse machine - an architecture for structure data-flow control flow [Uni. Of Nottingham 1985]
- Parallel Data-Driven Graph Reduction [Univ. Of Manchester 1986, Watson et al]
- Toward a dataflow/von Neumann hybrid architecture [MIT 1988 Iannucci]
- USC Decoupled Multilevel Data-Flow execution Model [1989]

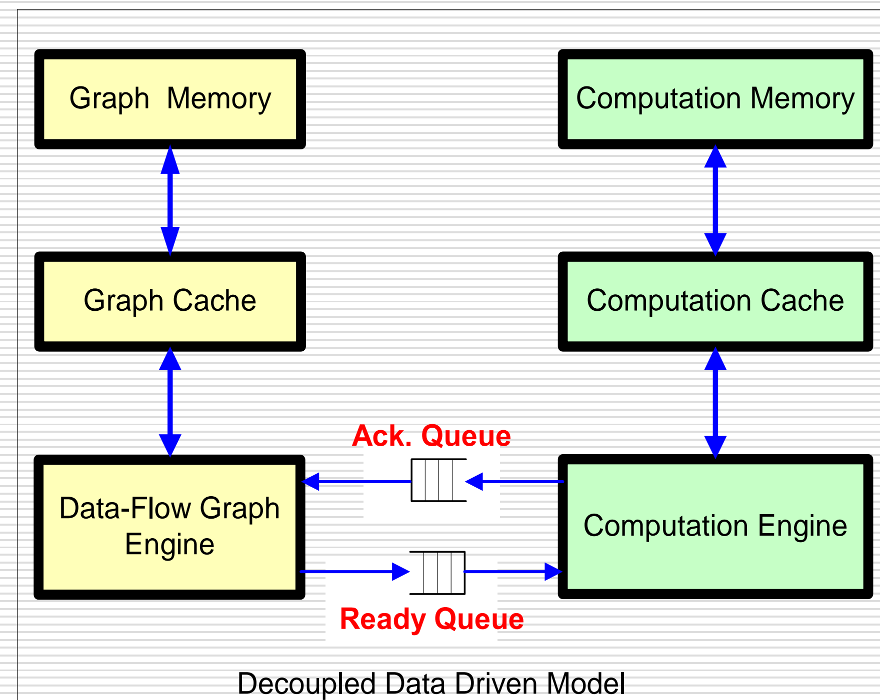


Decoupled Data-Driven Machine (D^3-machine)

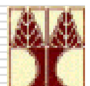
- Basic structure of a DDF machine is a cyclic pipeline
- Decoupled Data-Driven model separates the DDF synchronization from the computation.
  - Computation can be handled by a conventional Computer



DataFlow Model

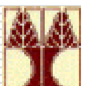


Decoupled Data Driven Model



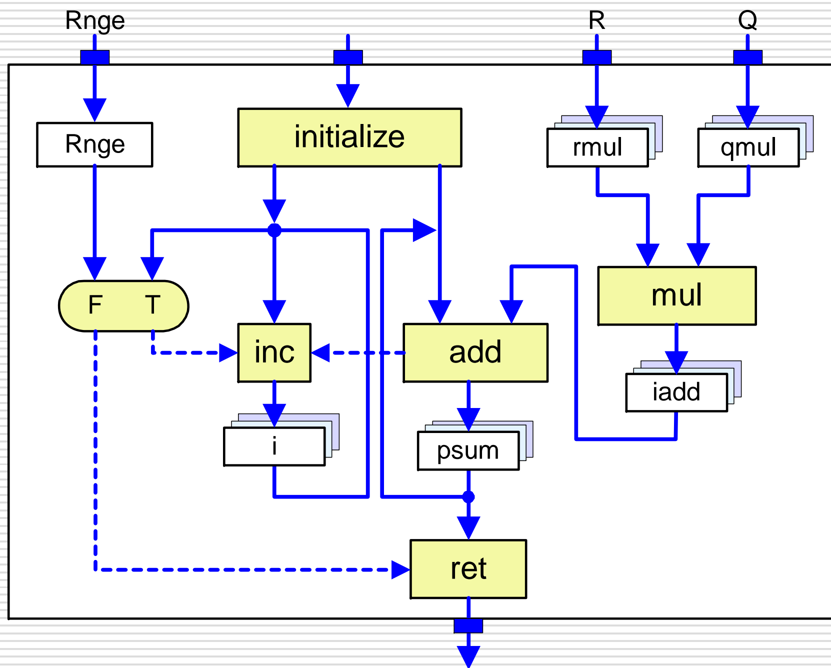
D^3-machine

- Data-Flow Graph Engine (DFGE): performs all graph operations
- Computation Engine (CE): performs all computation operation
- Asynchronous operation of two engines
  - Communication via the Ready Queue (RQ) and the Acknowledgment Queue (AQ)
- Cacheflow: Data-driven cache management
  - RQ gives the near-future execution patterns.

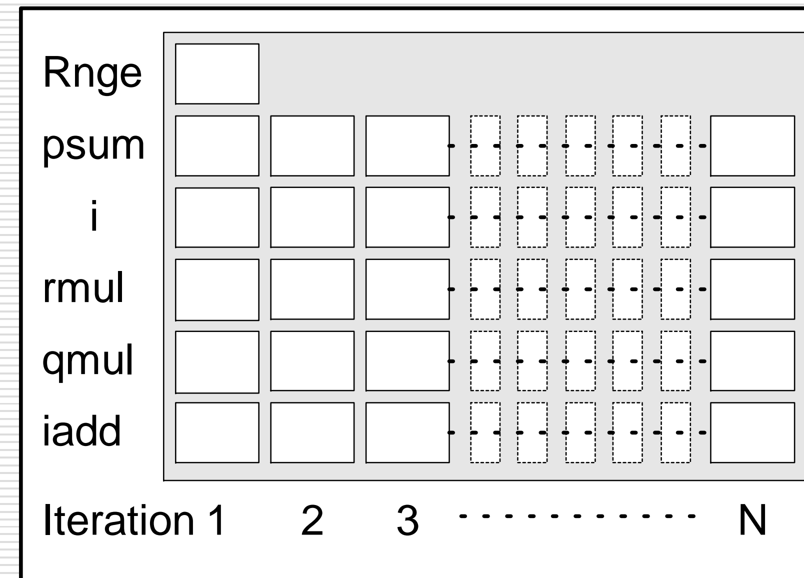


Token Mapping to Memory

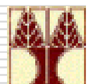
- Tokens are mapped in the Virtual memory
- f_{map} : <actor id, context> → virtual address
- No need for associative matching
- Normal Memory hierarchies can be used.



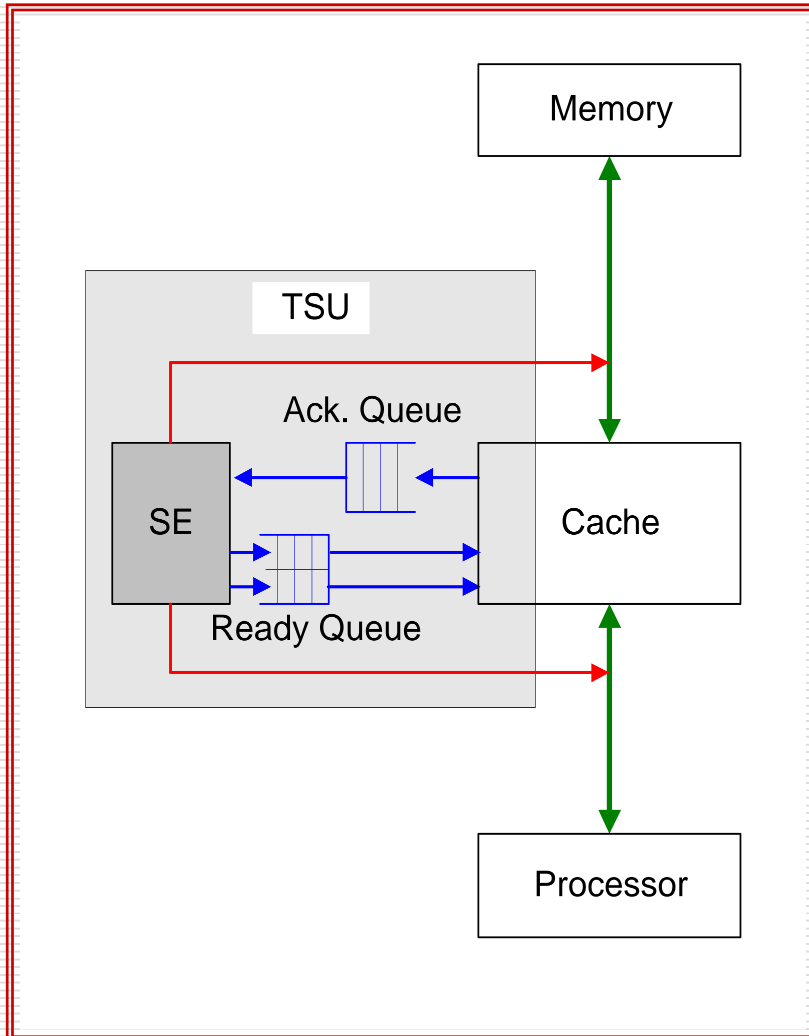
D³-graph for Inner product



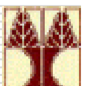
Computational memory



Thread Synchronization Unit (TSU)



- Use a commercial microprocessor for the computational Engine
- Implement the functionality of the Synchronization Engine and the two queues in an extra module.
- The TSU integrates the function of the SE, the RQ and AQ.
- The processor does not have any knowledge about the presence of the TSU.
- Five addresses are reserved for the communication of the processor and the TSU.
- The TSU uses snooping to intercept these addresses and process them accordingly.

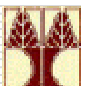


Data-Driven Multithreading

- Non-blocking …
  - Threads are scheduled based on data availability
  - Threads are created at compile time.

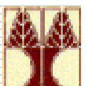
Multithreading Taxonomy

- Explicit
  - Non-Blocking Multithreading
  - Blocked Multithreading
  - Interleaving Multithreading
  - Simultaneous Multithreading
- Implicit
  - Threads created dynamically at runtime by the hardware
    - Minor/No compiler support
    - Thread level speculation on sequential (single-threaded) programs



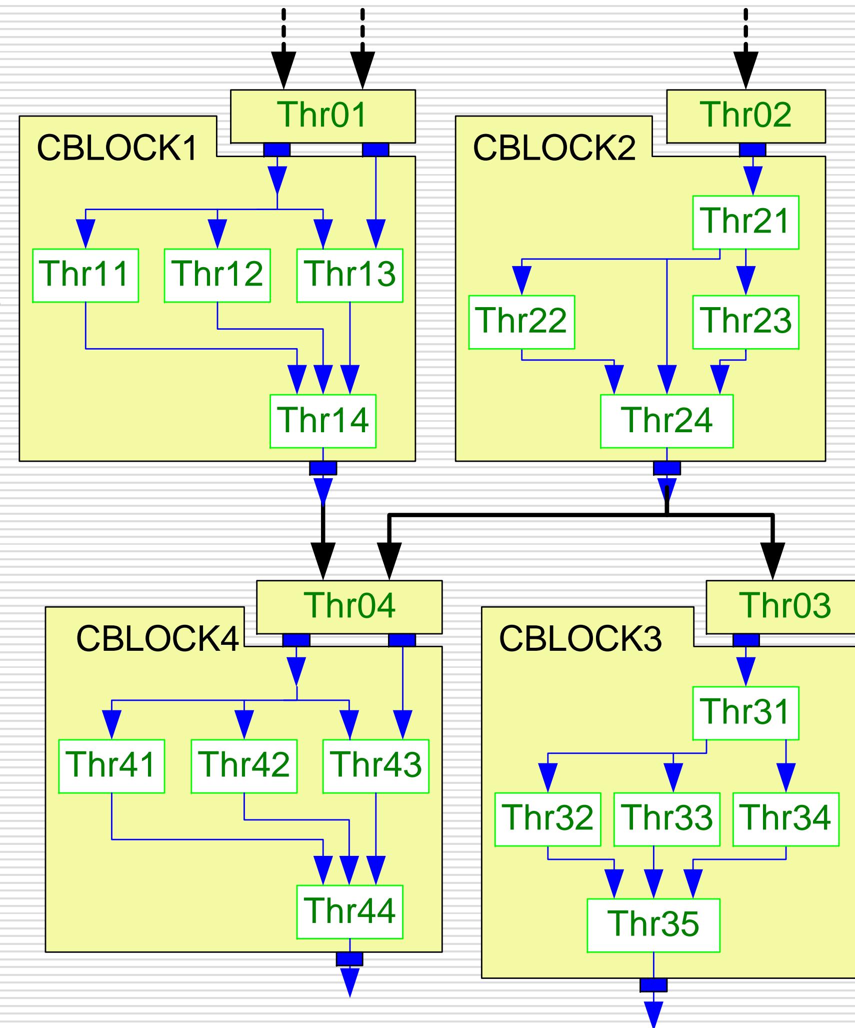
Non-Blocking Multithreading

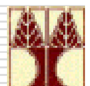
- Non-Blocking Threads
  - a sequentially ordered block of instructions
  - that do **not** stall the processor due to
    - remote memory accesses
    - synchronization waits
    - cache misses (not always)
- Thread scheduling based on data availability
  - Threads executed to completion
- Also known as
  - Threaded Dataflow (Monsoon) or
  - Large-Grain Dataflow [StarT, TAM, EARTH]



DDM code blocks & Thread scheduling

- Code Block: several threads equivalent to a function or loop body.
- Inlet thread: Initializes the code block.
- Outlet: Garbage collection
- Code block scheduling is done dynamically by the TSU.
- Inlet and Outlet threads provide synchronization among code-blocks





The TSU/Processor Interface

TSU

Ack. Queue

0004	0000	
0000	0000	
003C	0004	0005

AqStatus
AqIndex
AqTNum

Consumers

Determine executbility

IPtr
DFPtr

0030	301A	0005
0000	0000	0000
0130	0F00	0150
40A0	4000	40A0

RqTNum
RqIndex
RqIPtr
RqDFPtr

Ready Queue

Snooping Unit

PC Motherboard

Processor L1 Cache

L2 Cache

Memory

Graph

a b c

5

$d=b^2-4ac$

>0 <=0

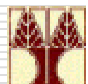
(6) (7)

Template

0005
0150
0003
0000
0006
0007

```
0150   mov   ebp,RqDFPtr
       mov   eax,[ebp+4]
       mul   eax
       mov   ebx,eax
       mov   eax,[ebp+8]
       mov   ecx,[ebp+0c]
       mul   ecx
       mov   ecx,4
       mul   ecx
       sub   ebx,ecx
       mov   [ebp+10],ebx
       mov   eax,04
       cmp   ebx,0
       jl    ngtv
       mov   eax,0c
ngtv:  mov   AqStatus,eax
       mov   eax,RqIPtr
       jmp   eax
```

The TSU communicates with the CPU through five non-cachable memory addresses. These can also be I/O addresses.



The TSU/Processor Interface

TSU

Ack. Queue

0004	0000	
0000	0000	
003C	0004	0005

AqStatus
AqIndex
AqTNum

Consumers

Determine executbility

IPtr
DFPtr

0030	301A	0005
0000	0000	0000
0130	0F00	0150
40A0	4000	40A0

RqTNum
RqIndex
RqIPtr
RqDFPtr

Ready Queue

Snooping Unit

PC Motherboard

Processor L1 Cache

L2 Cache

Memory

Graph

a b c

5

$d=b^2-4ac$

>0 <=0

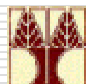
(6) (7)

Template

0005
0150
0003
0000
0006
0007

```
0150   mov   ebp,RqDFPtr
       mov   eax,[ebp+4]
       mul   eax
       mov   ebx,eax
       mov   eax,[ebp+8]
       mov   ecx,[ebp+0c]
       mul   ecx
       mov   ecx,4
       mul   ecx
       sub   ebx,ecx
       mov   [ebp+10],ebx
       mov   eax,04
       cmp   ebx,0
       jl    ngtv
       mov   eax,0c
ngtv:  mov   AqStatus,eax
       mov   eax,RqIPtr
       jmp   eax
```

The **RqIPtr** used to provide to the CPU the address of the next thread to be executed.



The TSU/Processor Interface

TSU

Ack. Queue

0004	0000	
0000	0000	
003C	0004	0005

AqStatus
AqIndex
AqTNum

Consumers

Determine executbility

IPtr
DFPtr

0030	301A	0005
0000	0000	0000
0130	0F00	0150
40A0	4000	40A0

Ready Queue

RqTNum
RqIndex
RqIPtr
RqDFPtr

Snooping Unit

PC Motherboard

Processor L1 Cache

L2 Cache

Memory

Graph

a b c

5

$d=b^2-4ac$

>0 <=0

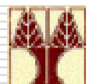
(6) (7)

Template

0005
0150
0003
0000
0006
0007

```
0150   mov   ebp,RqDFPtr
       mov   eax,[ebp+4]
       mul   eax
       mov   ebx,eax
       mov   eax,[ebp+8]
       mov   ecx,[ebp+0c]
       mul   ecx
       mov   ecx,4
       mul   ecx
       sub   ebx,ecx
       mov   [ebp+10],ebx
       mov   eax,04
       cmp   ebx,0
       jl    ngtv
       mov   eax,0c
ngtv:  mov   AqStatus,eax
       mov   eax,RqIPtr
       jmp   eax
```

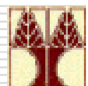
The RqDFPtr used to provide to the CPU the address of the data frame of the thread.



The TSU/Processor Interface


```
0150   mov  ebp,RqDFPtr
       mov  eax,[ebp+4]
       mul  eax
       mov  ebx,eax
       mov  eax,[ebp+8]
       mov  ecx,[ebp+0c]
       mul  ecx
       mov  ecx,4
       mul  ecx
       sub  ebx,ecx
       mov  [ebp+10],ebx
       mov  eax,04
       cmp  ebx,0
       jl   ngtv
       mov  eax,0c
ngtv:  mov  AqStatus,eax
       mov  eax,RqIPtr
       jmp  eax
```

The AqStatus used by the CPU to provide to the TSU Information about the status of the completed thread.



The TSU/Processor Interface

TSU

Ack. Queue

0004	0000	
0000	0000	
003C	0004	0005

AqStatus
AqIndex
AqTNum

Consumers

Determine executbility

IPtr
DFPtr

0030	301A	0005
0000	0000	0000
0130	0F00	0150
40A0	4000	40A0

Ready Queue

RqTNum
RqIndex
RqIPtr
RqDFPtr

Snooping Unit

PC Motherboard

Processor L1 Cache

L2 Cache

Memory

Graph

a b c

5

$d=b^2-4ac$

>0 <=0

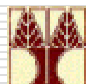
(6) (7)

Template

0005
0150
0003
0000
0006
0007

```
0150   mov   ebp,RqDFPtr
       mov   eax,[ebp+4]
       mul   eax
       mov   ebx,eax
       mov   eax,[ebp+8]
       mov   ecx,[ebp+0c]
       mul   ecx
       mov   ecx,4
       mul   ecx
       sub   ebx,ecx
       mov   [ebp+10],ebx
       mov   eax,04
       cmp   ebx,0
       jl    ngtv
       mov   eax,0c
ngtv:  mov   AqStatus,eax
       mov   eax,RqIPtr
       jmp   eax
```

The AqIndex/RqIndex used by the TSU/CPU to provide the iteration index of the thread.



The TSU/Processor Interface

TSU

Ack. Queue

Consumers

0004	0000	
0000	0000	
003C	0004	0005

AqStatus
AqIndex
AqTNum

Determine executbility

IPtr
DFPtr

0030	301A	0005
0000	0000	0000
0130	0F00	0150
40A0	4000	40A0

Ready Queue

RqTNum
RqIndex
RqIPtr
RqDFPtr

Snooping Unit

PC Motherboard

Processor L1 Cache

L2 Cache

Memory

Graph

a b c

5

$d=b^2-4ac$

>0 <=0

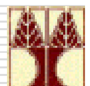
(6) (7)

Template

0005
0150
0003
0000
0006
0007

```
0150   mov   ebp,RqDFPtr
       mov   eax,[ebp+4]
       mul   eax
       mov   ebx,eax
       mov   eax,[ebp+8]
       mov   ecx,[ebp+0c]
       mul   ecx
       mov   ecx,4
       mul   ecx
       sub   ebx,ecx
       mov   [ebp+10],ebx
       mov   eax,04
       cmp   ebx,0
       jl    ngtv
       mov   eax,0c
ngtv:  mov   AqStatus,eax
       mov   eax,RqIPtr
       jmp   eax
```

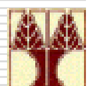
These addresses are intercepted by the Snooping Unit and forwarded to TSU for further processing.



Data-Driven Multithreading Execution

Thread					
0031	1				
0032	1	1	1		1
0033	3	3	3		3
0034	2	2	2		2

Thread	Con1	Con2	IFP	DFP
0031	0033	0034	0100	3A00
0032	0033	0036	0108	3A00
0033	0034	0000	011C	3A00
0034	0032	0033	0122	3A00



Data-Driven Multithreading Execution

Thread Synchronization Unit (TSU)

Synchronization Memory (SM)

Graph Memory (GM)

Ack. Queue (AQ)

0?

Ready Queue (RQ)

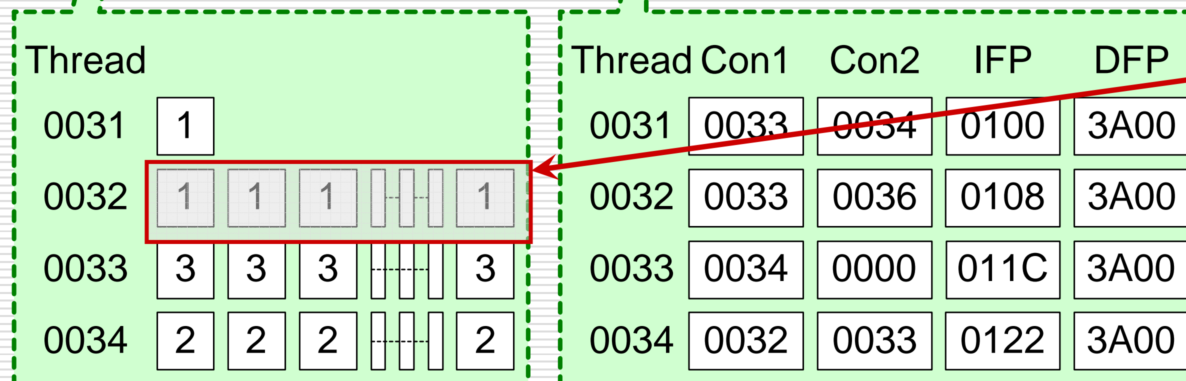
Snooping Unit

Workstation

Processor L1 Cache

L2 Cache

Memory



Thread					
0031	1				
0032	1	1	1	...	1
0033	3	3	3	...	3
0034	2	2	2	...	2

Thread	Con1	Con2	IFP	DFP
0031	0033	0034	0100	3A00
0032	0033	0036	0108	3A00
0033	0034	0000	011C	3A00
0034	0032	0033	0122	3A00

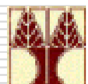
The SM contains the Ready Counts. One value for each loop iteration.

Data-Driven Multithreading Execution

Thread					
0031	1				
0032	1	1	1		1
0033	3	3	3		3
0034	2	2	2		2

Thread	Con1	Con2	IFP	DFP
0031	0033	0034	0100	3A00
0032	0033	0036	0108	3A00
0033	0034	0000	011C	3A00
0034	0032	0033	0122	3A00

The GM contains the IFP, DFP and the two consumers (Con1 and Con2).



Data-Driven Multithreading Execution

Thread Synchronization Unit (TSU)

Synchronization Memory (SM)

Graph Memory (GM)

Ack. Queue (AQ)

Ready Queue (RQ)

0?

Snooping Unit

Workstation

Processor L1 Cache

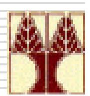
L2 Cache

Memory

Thread					
0031	0				
0032	1	1	1		1
0033	3	3	3		3
0034	2	2	2		2

Thread	Con1	Con2	IFP	DFP
0031	0033	0034	0100	3A00
0032	0033	0036	0108	3A00
0033	0034	0000	011C	3A00
0034	0032	0033	0122	3A00

The processor reads from the RQ pointers (IFP, DFP and index) of ready threads and executes them

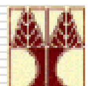


Data-Driven Multithreading Execution

Thread					
0031	1				
0032	1	1	1		1
0033	3	3	3		3
0034	2	2	2		2

Thread	Con1	Con2	IFP	DFP
0031	0033	0034	0100	3A00
0032	0033	0036	0108	3A00
0033	0034	0000	011C	3A00
0034	0032	0033	0122	3A00

After executing a thread, the processor stores in the AQ information (Thread#, index and status) of the executed thread.

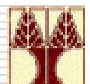


Data-Driven Multithreading Execution

Thread					
0031	1				
0032	1	1	1		1
0033	3	3	3		3
0034	2	2	2		2

Thread	Con1	Con2	IFP	DFP
0031	0033	0034	0100	3A00
0032	0033	0036	0108	3A00
0033	0034	0000	011C	3A00
0034	0032	0033	0122	3A00

The TSU determines the consumers of executed threads from the GM.



Data-Driven Multithreading Execution

Thread Synchronization Unit (TSU)

Synchronization Memory (SM)

Graph Memory (GM)

Ack. Queue (AQ)

Ready Queue (RQ)

0?

Snooping Unit

Workstation

Processor L1 Cache

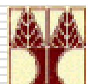
L2 Cache

Memory

Thread							
0031	1						
0032	1	0	1				1
0033	3	2	3				3
0034	2	2	2				2

Thread	Con1	Con2	IFP	DFP
0031	0033	0034	0100	3A00
0032	0033	0036	0108	3A00
0033	0034	0000	011C	3A00
0034	0032	0033	0122	3A00

Update SM and check if any of the consumers is ready (Ready Count = 0)

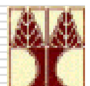


Data-Driven Multithreading Execution

Thread							
0031	1						
0032	1	0	1				1
0033	3	2	3				3
0034	2	2	2				2

Thread	Con1	Con2	IFP	DFP
0031	0033	0034	0100	3A00
0032	0033	0036	0108	3A00
0033	0034	0000	011C	3A00
0034	0032	0033	0122	3A00

The TSU loads in the RQ the pointers (IFP, DFP and index) of ready thread from the GM.



Thread Synchronization Unit (TSU)

Thread Synchronization Unit (TSU)
Interconnection Network
Network Interface Unit (NIU)
Post Processing Unit (PPU)
Thread Issue Unit (TIU)
RxQ
TxQ
NetQ
Synchronization Memory
Graph Memory
PPU Control Unit
TIU Control Unit
Ack. Queue
Waiting Queue
Firing Queue
Prefetch Unit
Main Memory
Processor
Cache
DMA

- Three Units
  - Thread Issue Unit
  - Post Processing Unit
  - Network Interface Unit
- Units are Decoupled/asychronous
- Communicate via Queues

D²NOW with Pentium Workstations

Commodity Workstation

Pentium Processor

TSU attached to COAST slot

Interconnection Network

Workstation 1

L2 Cache

TSU

NIU PPU TIU

Add-on Card

COAST Slot

Main Memory

Processor

Motherboard

Workstation N

L2 Cache

TSU

NIU PPU TIU

Add-on Card

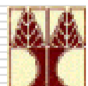
COAST Slot

Main Memory

Processor

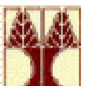
Motherboard

- TSU could go on the COAST slot
  - TSU and Cache with dual ported Tag bits
- TSU in the Co-processor slot, Use MESI instructions for cacheflow

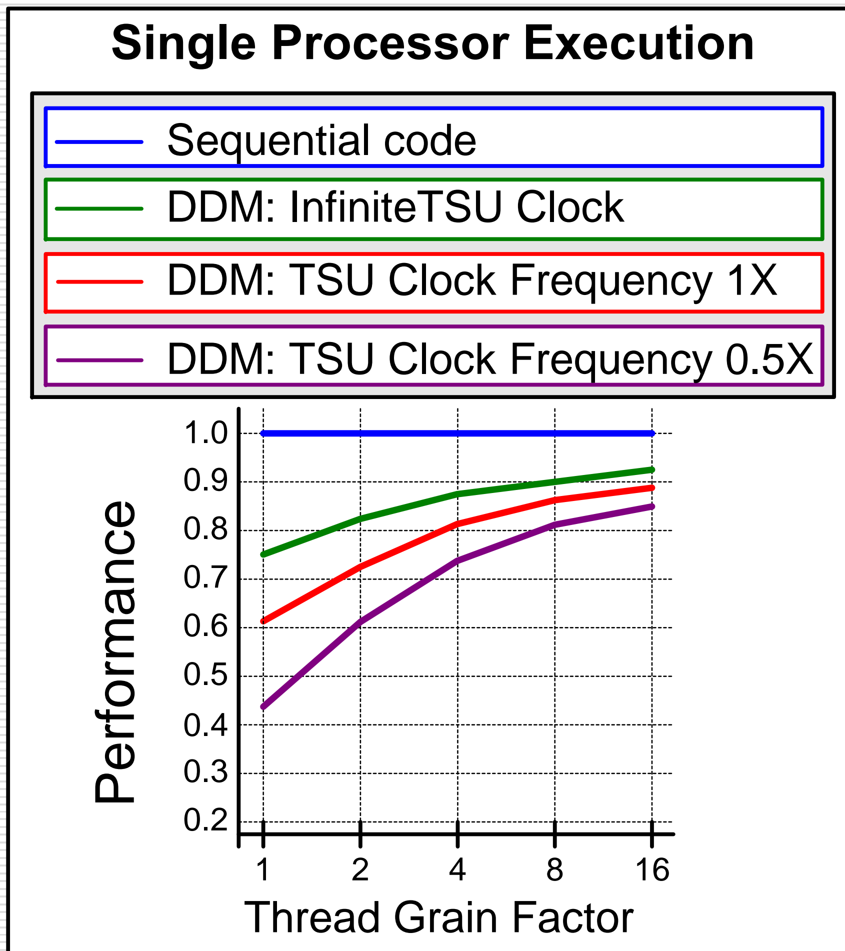


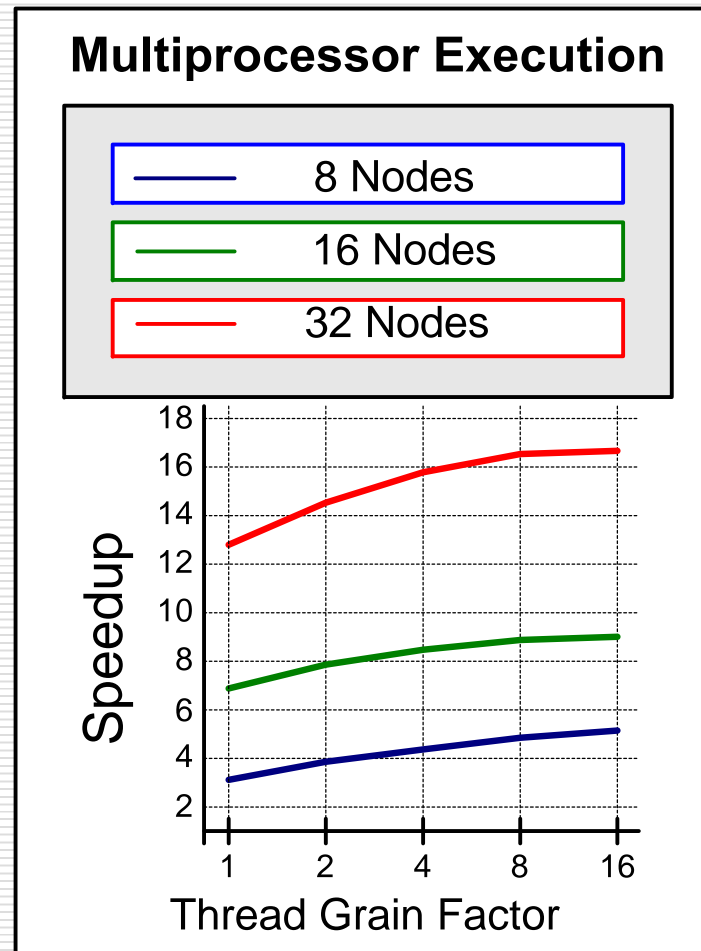
Performance Analysis

- DDM-SIM: Simulated D^2NOW
  - Execution Driven Simulator
- Application Suite:
  - Seven Splash-2 applications: ***FFT, LU***, ***Radix***, ***Barnes***, ***FMM***, ***Cholesky***, & ***Radiosity***
  - Two scientific micro-kernels: ***Mult*** & ***Trapez***
  - ***For all speedup comparisons we compare DDM threaded code vs h C code compiled with the MS optimized compiler and executed on the hardware.***

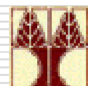


Effect of Thread Granularity on Performance



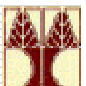


Increasing thread granularity from 1 to 8
Average speedup improvement: 27%
(from 12.8 to 16.8 on 32-node system)



Effect of **Communication Mechanism** on Performance

- Three communication mechanisms employed:
- *Fine-grain*: TSU through interconnection network
  - Single value packets
- *Block*: TSU initiates DMA through inter. network
  - Packets with up to 512 bytes (17% speedup improvement compared to only *fine-grain* communication)
- *Ethernet*: CPU through an Ethernet LAN
  - Packets with more than 512 bytes (22% speedup improvement compared to only *fine-grain* communication)
- Speedup reduction when the interconnection switch clock cycle is increased 500%:
  - Minimum: 2.8%
    - Average: 13.4%
  - Maximum: 22.8%

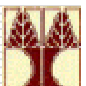


CacheFlow:

A Cache Management Policy for Data-Driven Multithreading

CacheFlow Policy

- Motivation
  - The Firing Queue of a DDM machine determines the order in which the threads are executed
  - Each Thread has a pointer to its Data
  - Future memory accesses are known!
- CacheFlow reduces cache misses by Prefetching blocks to be used in the near future (basic implementation)
  - Not scheduling threads that could cause false cache conflicts (Optimization 1: False Conflict avoidance)
    - False cache conflicts: prefetching displaces data prefetch for other threads waiting in the FQ
  - Reordering the sequence of execution of ready threads to exploit locality (Optimization 2: Thread Reordering)



Basic CacheFlow Implementation

Post Processing Unit
Ack. Queue
Ready Threads
Thread Issue Unit
Waiting Queue (WQ)
Index
Thread#
IFP
DFP 1
DFP 2
Graph Memory (GM)
DFP Select Unit
Prefetch Unit
Firing Queue (FQ)
Thread#
Index
IFP
RqTNum
RqIndx
RqIptr
Processor
L1 Cache
L2 Cache

Thread IFP and DFP placed in GM

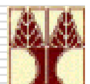
Determine data addresses

Hardware prefetching

Shift thread pointers into the FQ

Two shifting policies:

- ***Policy 1***: Shift thread into FQ as soon prefetching is initiated
- ***Policy 2***: Shift thread into FQ after prefetching is completed



Optimization 2: *Cache-Flow with Thread Reordering*

Threads are always in the correct order in the WQ.

New thread

Thread#	Index
20	46

Waiting Queue before reordering

Thread#	Index
25	89
18	79
18	76
23	134
20	48
20	45

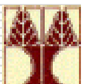
Waiting Queue after reordering

Thread#	Index
25	89
18	79
18	76
23	134
20	48
20	46
20	45

New thread inserted according to its number . . .

. . . and according to its index

Reordering is done in parallel with other TSU operations



Effect of CacheFlow on Cache Miss Rate and Speedup

Sequential | DDM - No CacheFlow | DDM - Basic Prefetch | DDM - Conflict Avoidance | DDM - Thread Reordering

Cache miss rate (%)

0 2 4 6 8 10 12

Small Data Size: 6.9, 9.8, 3.1, 2, 1.4

Large Data Size: 16%, 23%, 7%, 5%, 8%

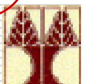
DDM - No CacheFlow | DDM - Conflict Avoidance | DDM - Basic Prefetch | DDM - Thread Reordering

Speedup

0 4 8 12 16 20 24 28

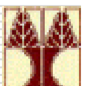
Average among all applications: 19.7, 22.6, 24.4, 26.0

- DDM increases miss rate
- CacheFlow reduces miss rate (lower than the sequential)
- Optimizations reduce further miss rate
- Average speedup increases from 19.7 to 26.0



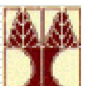
Experimental Results Summary

- Thread granularity: impact on
  - DDM overheads, locality, TSU latency, pipeline performance
  - Increasing from 1 to 8 increases performance by 20%
    - **12.8 to 16.8 (on 32-node system)**
- Communication assist optimizations: impact on
  - CPU communication overheads
  - Lead to 22% increase in speeedup (19.7 speedup 32-nodes)
  - Increase in communication latency by 500%
    - **in 13.4% (2.8-23%) average speedup reduction**
- But DDM could destroy locality and increase cache misses
- CacheFlow: hardware prefetching to
  - Completely eliminates extra misses due to DDM
    - **Serial: 6.9** **DDM: 9.8** **DDM w Cacheflow: 1.4**
      - further reduces cache misses
  - Overall speedup increased from **19.7 to 26.0 (32-nodes)**



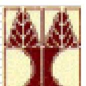
DDM Conclusions & Future work

- DDM is a technique proposed for parallel processing systems employing off-the-shelf processors
  - Exploit the parallelism of the Dataflow model
  - Effectively tolerate long latency and improves cache misses
    - Improves Locality
- Migrate TSU on Chip
- Prototype implementation platform: Xilinx VirtexPro II
- DDM chip multiprocessor
  - Preliminary Estimates:
    - 4 DDM cores (PIII + TSU) Can fit in P4 die
      - Speedup range 2.7-7.6
    - 8 DDM cores (Basic RISC pipeline no OoO execution etc)
      - Speedup range 5.3-14.9



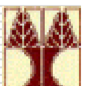
DDM CMP

- A **chip multiprocessor** that is able to support the **Data-Driven Multithreading** model of execution
- Does **not** require any:
  - Modifications to the CPU
  - Modifications to the ISA
  - Modifications to the Operating System
- Can execute both DDM and non-DDM applications
- Has shown both power and performance benefits

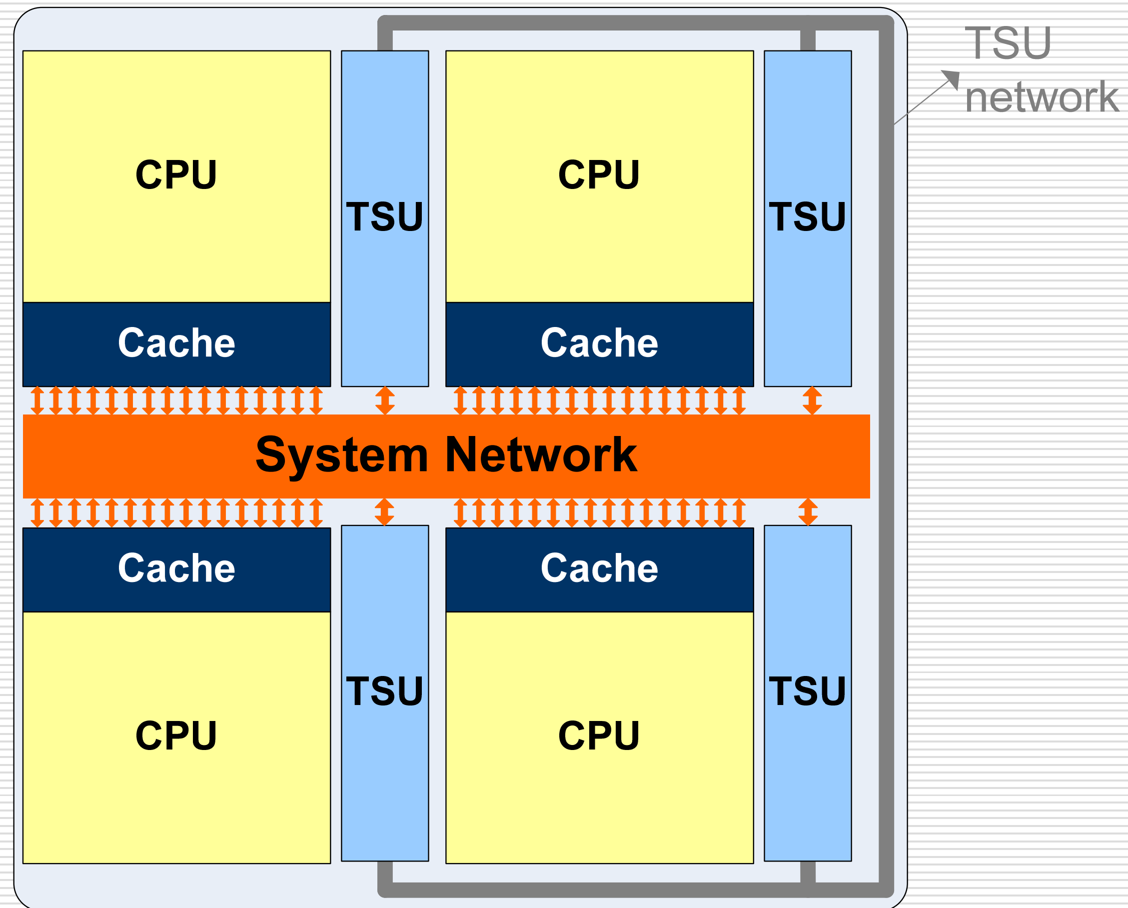


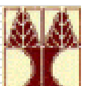
DDM CMP - Kernel

- **DDM-CMP Kernel** is a simple, lightweight user-level process that allows the DDM-CMP system to operate on unmodified OS
- The DDM-CMP Kernel
  - Guarantees a **common virtual address space** for all on-chip execution cores
  - Dynamically **loads the TSU**



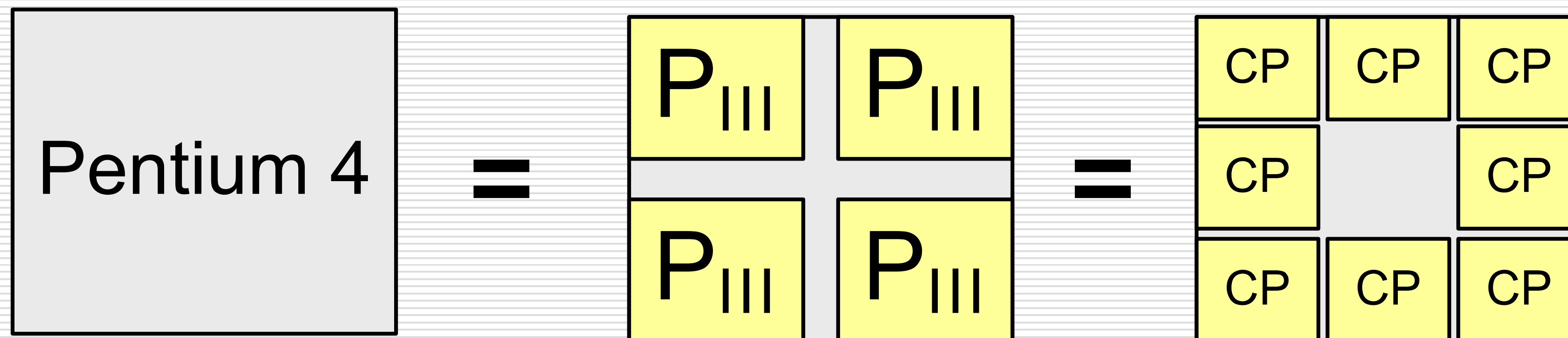
The DDM CMP Chip



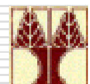


Pentium based DDM CMP

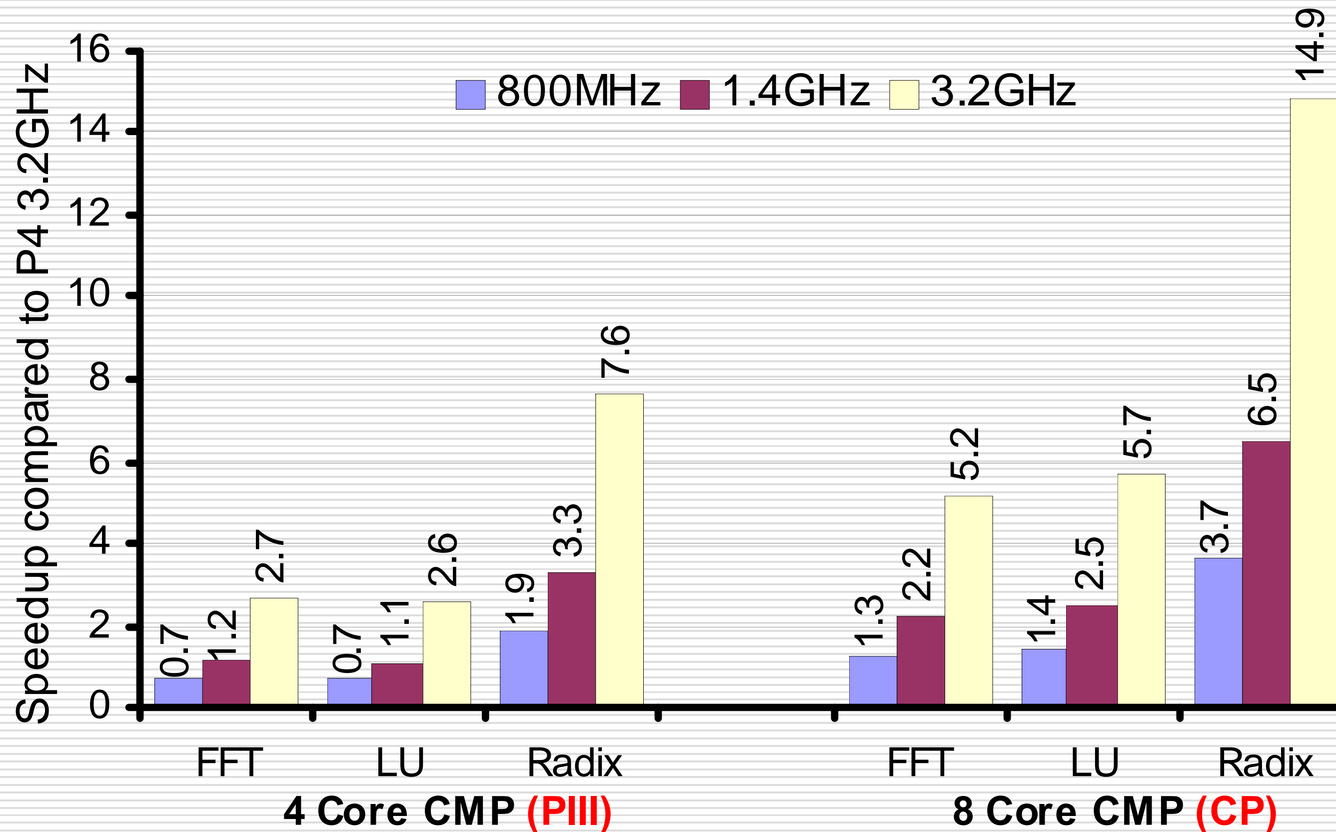
- The number of **transistors** of **Pentium 4** is enough for
  - **5.7 Pentium 3** processors
  - ~**10 "modified Pentium 3"** processors.



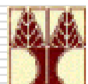
	Processor	Freq.	Tech.	L2 cache	L1 cache
P4	Pentium 4 - HT	3.2GHz	90nm	1MB	
PIII	Pentium 3	800MHz	180nm	256KB	32KB
CP	**Modified** Pentium 3	800MHz	180nm	32KB	16KB



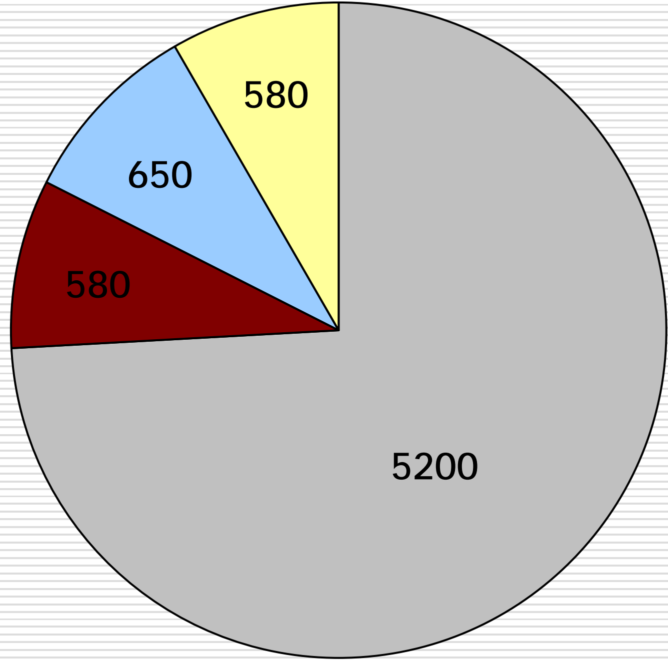
Preliminary Experimental Results

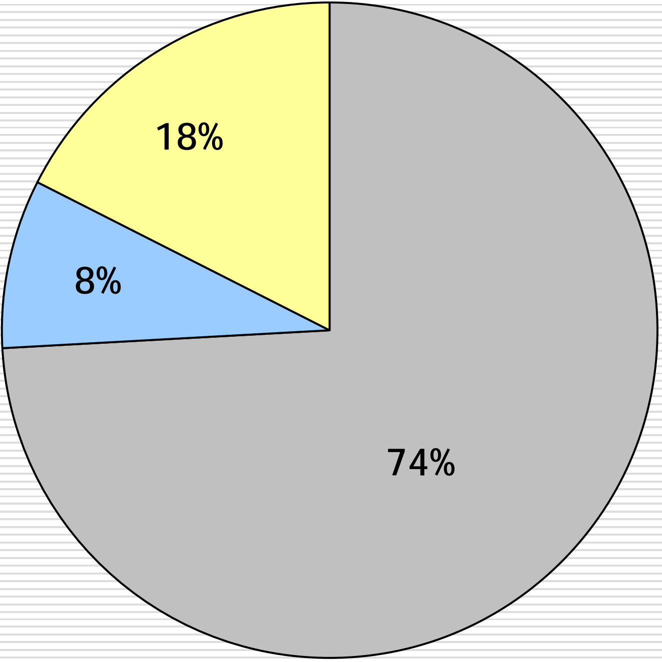


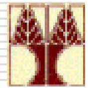
	Processor	Freq.	Tech.	L2 cache	L1 cache
P4	Pentium 4 - HT	3.2GHz	90nm	1MB	
PIII	Pentium 3	800MHz	180nm	256KB	32KB
CP	**Modified** Pentium 3	800MHz	180nm	32KB	16KB



DDM CMP Chip - Hardware Budget







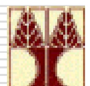
Speedup for equal power consumption

Speedup

Speedup Compared to P4 3.2GHz

25.0
20.0
15.0
10.0
5.0
0.0

	FFT	LU	Radix	Mmult	Trapez
DDM-CMP4 (4009MHz)	3.4	3.3	9.5	2.3	11.7
DDM-CMP8 (3901MHz)	6.4	7.0	18.1	2.3	22.6

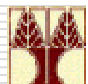


Power consumption for equal speedup

Power Consumption

Power Consumption Compared to P4 3.2GHz

	FFT	LU	Radix	Mmult	Trapez
DDM-CMP4	18%	19%	1%	35%	1%
DDM-CMP8	3%	2%	1%	37%	1%



CMP-DDM Conclusions

- DDM is a technique proposed for parallel processing systems employing off-the-shelf processors
  - Exploits the parallelism of the Dataflow model
  - Effectively tolerate long latencies and improves cache misses
    - Cacheflow Improves Locality!
- CMP-DDM: Chip Multiprocessors based Data-Driven Multithreading
  - Elegant and formal way to handle Multiprocessor Synchronization
  - Latency Tolerance overcomes the Memory wall.
  - Power efficient through Reduced processor complexity and smaller caches