CS184c:
Computer Architecture
[Parallel and Multithreaded]

Day 14: May 24, 2001

SCORE

Previously

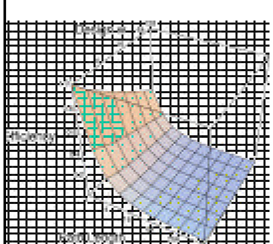
- Interfacing Array logic with Processors
- Single thread, single-cycle operations
- Scaling
  - models weak on allowing more active hardware
- Can imagine a more general, heterogeneous, concurrent, multithreaded compute model….

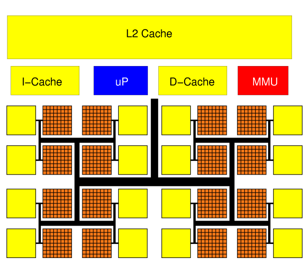
Today

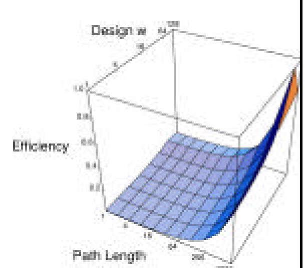
- SCORE
  - scalable compute model
  - architecture to support
  - mapping and runtime issues

UCB BRASS RISC+HSRA

- Integrate:
  - processor
  - reconfig. Array
  - memory
- Key Idea:
  - best of both worlds temporal/spatial

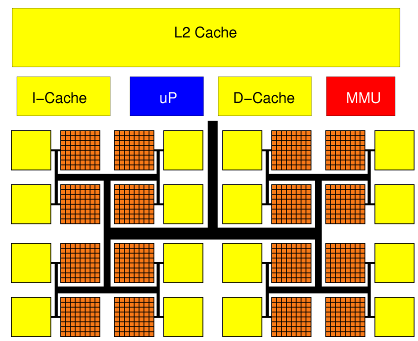






Bottom Up

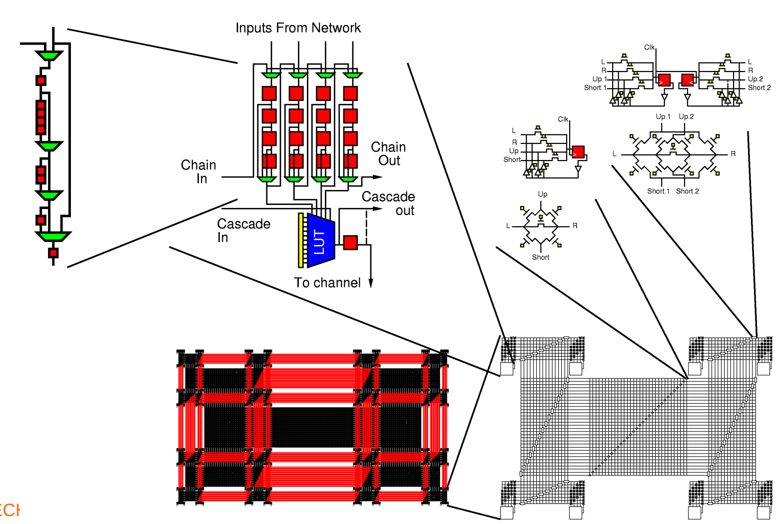
- GARP
  - Interface
  - streaming
- HSRA
  - clocked array block
  - scalable network
- Embedded DRAM
  - high density/bw
  - array integration



Good handle on:
    raw building blocks
    tradeoffs

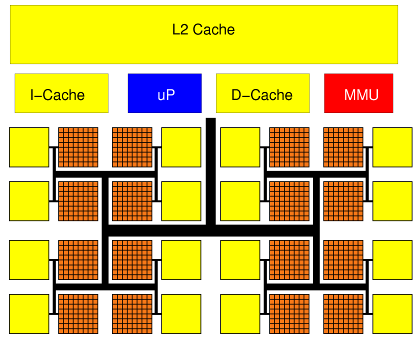
CS184a: Day16

HSRA Architecture



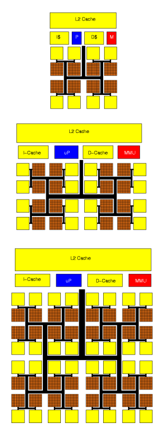
Top Down

- Question remained
  - How do we control this?
  - Allow hardware to scale?
- What is the higher level model
  - capture computation?
  - allows scaling?



SCORE

- An attempt at defining a computational model for reconfigurable systems
  - abstract out
    - physical hardware details
    - especially size / # of resources
    - timing
- Goal
  - achieve device independence
  - approach density/efficiency of raw hardware
  - allow application performance to scale based on system resources (w/out human intervention)



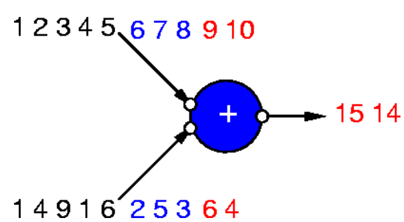
CALTECH cs184c Spring2001 -- DeHon

SCORE Basics

- Abstract computation is a dataflow graph
  - stream links between operators
  - dynamic dataflow rates
- Allow instantiation/modification /destruction of dataflow during execution
  - separate dataflow construction from usage
- Break up computation into compute pages
  - unit of scheduling and virtualization
  - stream links between pages
- Runtime management of resources

Stream Links

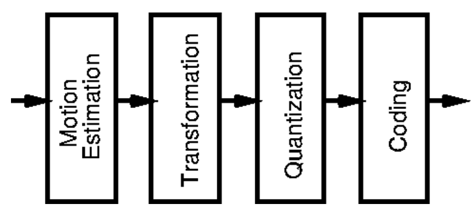
- Sequence of data flowing between operators
  - e.g. vector, list, image
- Same
  - source
  - destination
  - processing



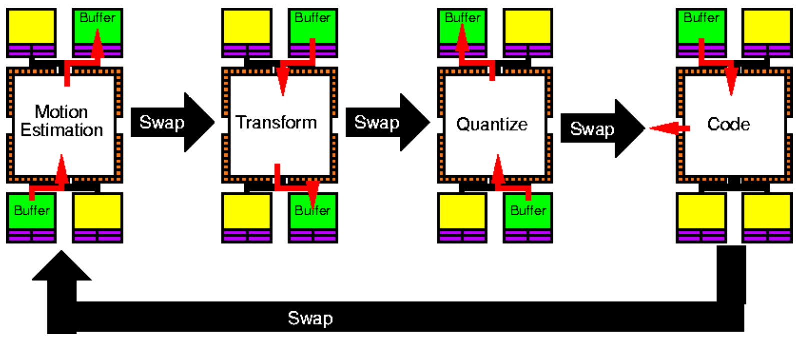
Virtual Hardware Model

- Dataflow graph is arbitrarily large
- Hardware has finite resources
  - resources vary from implementation to implementation
- Dataflow graph must be scheduled on the hardware
- Must happen automatically (software)
  - physical resources are abstracted in compute model

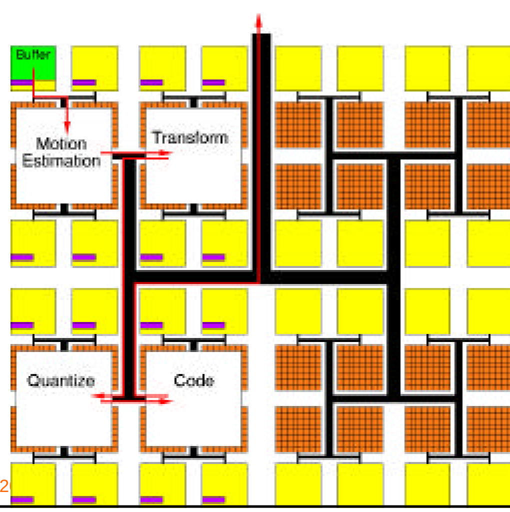
Example



Ex: Serial Implementation

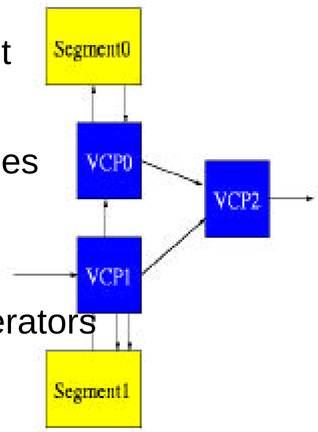


Ex: Spatial Implementation



Compute Model Primitives

- SFSM
  – FA with Stream Inputs
  – each state: required input set
- STM
  – may create any of these nodes
- SFIFO
  – unbounded
  – abstracts delay between operators
- SMEM
  – single owner (user)



SFSM

- Model view for an operator or compute page
  – FIR, FFT, Huffman Encoder, DownSample
- Less powerful than an arbitrary software process
  – bounded physical resources (no dynamic allocation)
  – only interface to state through streams
- More powerful than an SDF operator
  – dynamic input and output rates
  – dynamic flow rates

SFSM

Operators are FSMs not just Dataflow graphs

- Variable Rate Inputs
  - FSM state indicates set of inputs require to fire
- Lesson from hybrid dataflow
  - control flow cheaper when succ. known
- DF Graph of operators gives task-level parallelism
  - GARP and C models are all just one big TM
- Gives programmer convenience of writing familiar code for operator
  - use well-known techniques in translation to extract ILP within an operator

STM

- Abstraction of a process running on the sequential processor
- Interfaced to graph like SFSM
- More restricted/stylized than threads
  - cannot side-effect shared state arbitrarily
  - stream discipline for data transfer
  - single-owner memory discipline

STM

- Adds power to allocate memory
  - can give to SFSM graphs
- Adds power to create and modify SCORE graph
  - abstraction for allowing the *logical* computation to evolve and reconfigure
  - Note different from physical reconfiguration of hardware
    - that happens below the model of computation
    - invisible to the programmer, since hardware dependent

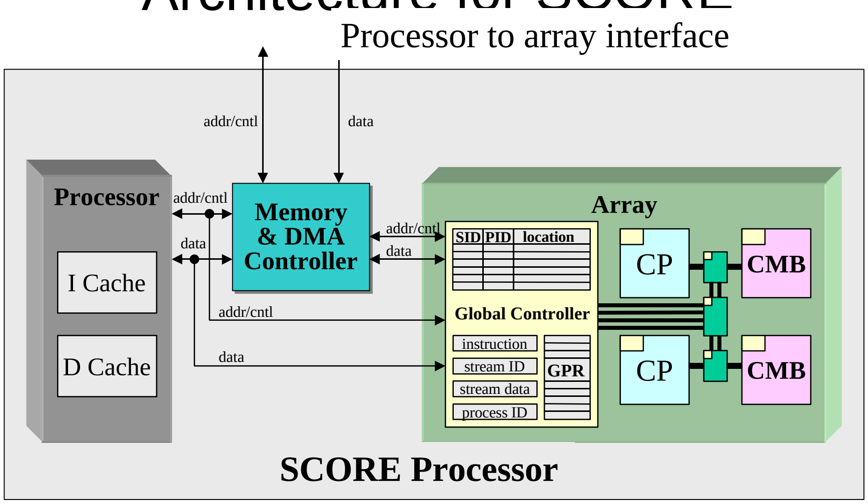
Model consistent across levels

- Abstract computational model
  - think about at high level
- Programming Model
  - what programmer thinks about
  - no visible size limits
  - concretized in language: *e.g.* TDF
- Execution Model
  - what the hardware runs
  - adds **fixed-size** hardware pages
  - primitive/kernel operations (*e.g.* ISA)

Architecture

Lead: Randy Huang

Architecture for SCORE



Processor ISA Level Operation

- User operations
  - Stream write STRMWR Rstrm, Rdata
  - Stream read STRMRD Rstrm, Rdata
- Kernel operation (not visible to users)
  - {Start,stop} {CP,CMB,IPSB}
  - {Load,store} {CP,CMB,IPSB} {config,state,FIFO}
  - Transfer {to,from} main memory
  - Get {array processor, compute page} status

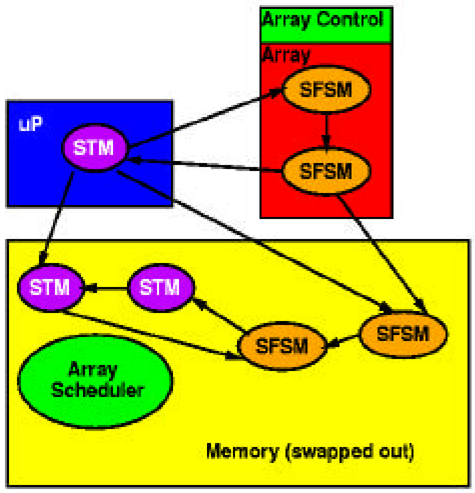
Communication Overhead

Note

- single cycle to send/receive data
- no packet/communication overhead
  - once a connection is setup and resident
- contrast with MP machines and NI we saw earlier

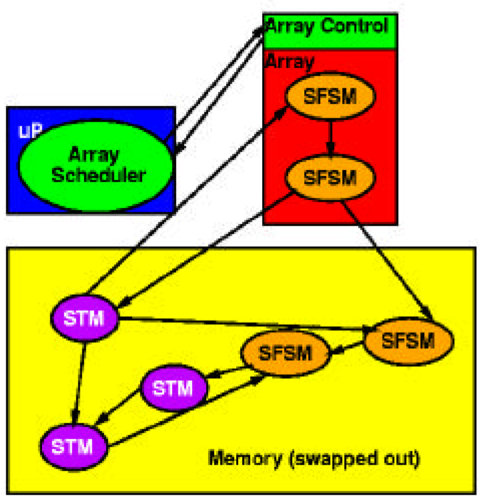
SCORE Graph on Hardware

- One master application graph
- Operators run on processor and array
- Communicate directly amongst



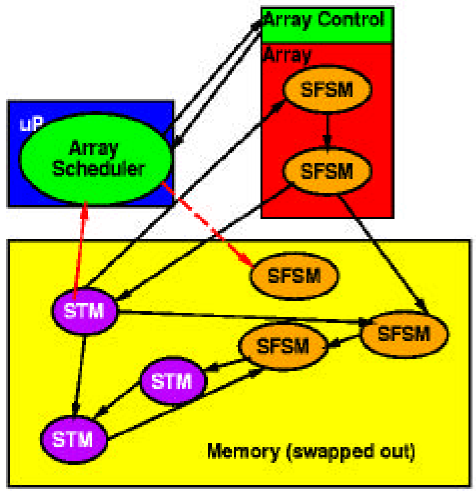
SCORE OS: Reconfiguration

- Array managed by OS
- Only OS can manipulate array configuration

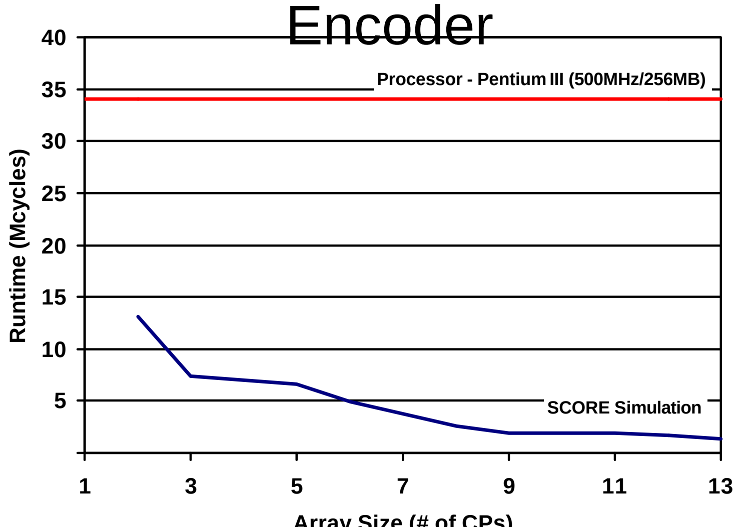


SCORE OS: Allocation

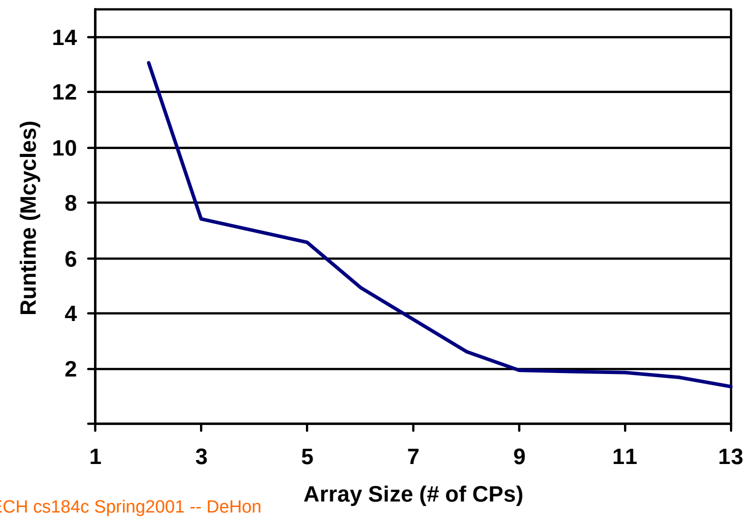
- Allocation goes through OS
- Similar to sbrk in conventional API



Performance Scaling: JPEG Encoder



Performance Scaling: JPEG Encoder



Page Generation (work in progress)

Eylon Caspi, Laura Pozzi

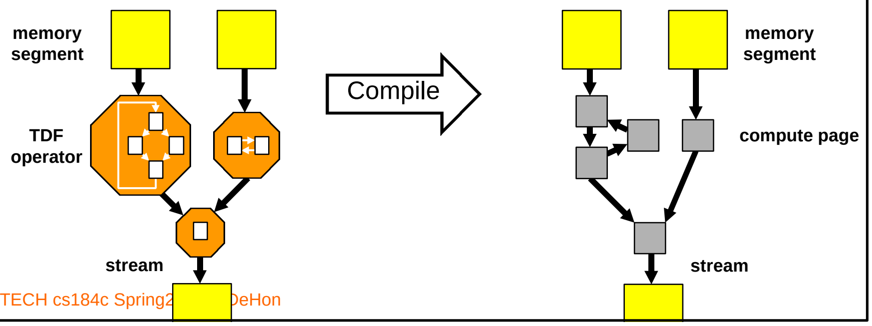
SCORE Compilation in a Nutshell

Programming Model

- Graph of TDF FSMD operators
  - unlimited size, # IOs
  - no timing constraints

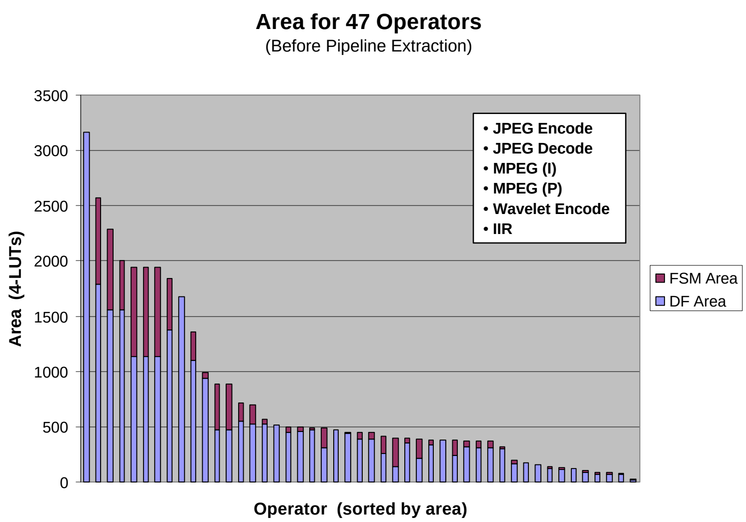
Execution Model

- Graph of page configs
  - fixed size, # IOs
  - timed, single-cycle firing



CALTECH cs184c Spring2 ... DeHon

How Big is an Operator?



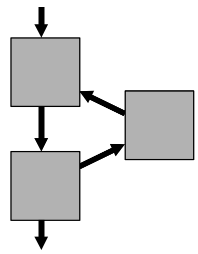
CALTI

Unique Synthesis / Partitioning Problem

- Inter-page stream delay not known by compiler:
  - HW implementation
  - Page placement
  - Virtualization
  - Data-dependent token emission rates
- Partitioning must retain stream abstraction
  - also gives us freedom in timing
- Synchronous array hardware

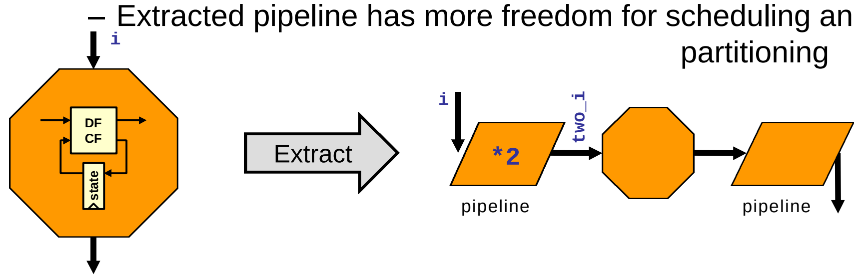
Clustering is Critical

- Inter-page comm. *latency* may be long
- Inter-page *feedback loops* are slow
- Cluster to:
  - Fit feedback loops within page
  - Fit feedback loops on device



Pipeline Extraction

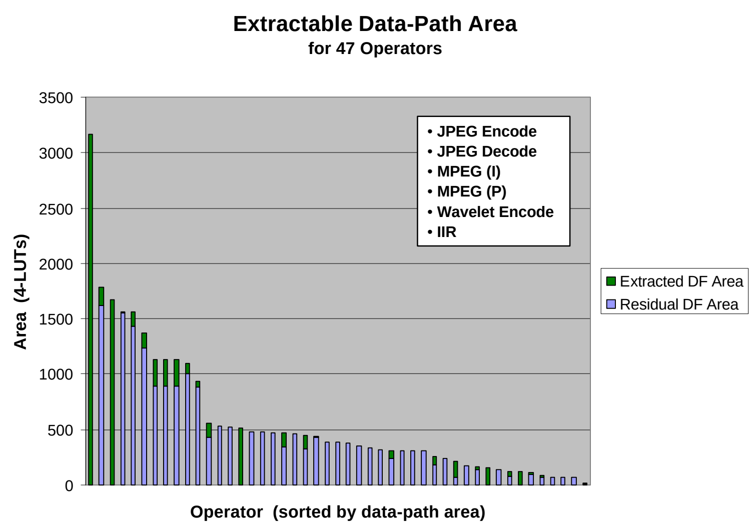
- Hoist uncontrolled FF data-flow out of FSMD
- Benefits:
  - Shrink FSM cyclic core
  - Extracted pipeline has more freedom for scheduling and partitioning




```
state foo(i):
  acc=acc+2*i
```

```
state foo(two_i):
  acc=acc+two_i
```


Pipeline Extraction –

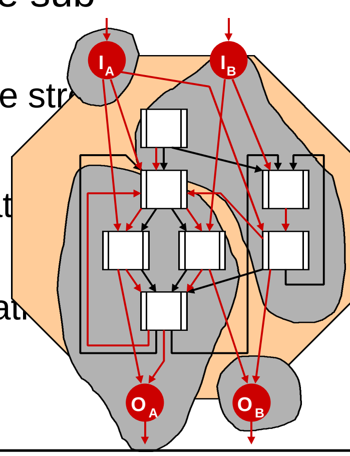


Page Generation

- Pipeline extraction
  - removes dataflow can freely extract from FSMD control
- Still have to partition potentially large FSMs
  - approach: turn into a clustering problem

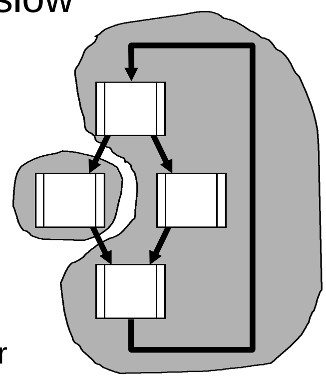
State Clustering

- Start: consider each state to be a unit
- Cluster states into page-size sub-FSMDs
  - Inter-page transitions become str
- Possible clustering goals:
  - Minimize delay (inter-page lat
  - Minimize IO (inter-page BW)
  - Minimize area (fragmentat



State Clustering to Minimize Inter-Page State Transfer

- Inter-page state transfer is slow
- Cluster to:
  - Contain feedback loops
  - Minimize frequency of inter-page state transfer
- Previously used in:
  - VLIW trace scheduling *[Fisher '81]*
  - FSM decomposition for low power *[Benini/DeMicheli ISCAS '98]*
  - VM/cache code placement
  - GarpCC code selection *[Callahan '00]*



Scheduling (work in progress)

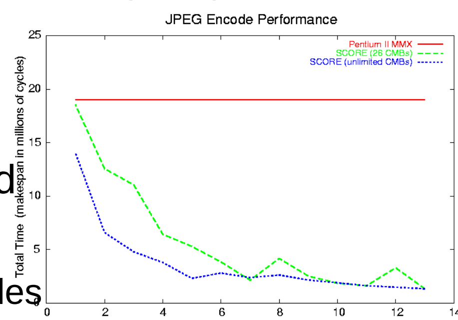
Lead: Yury Markovskiy

Scheduling

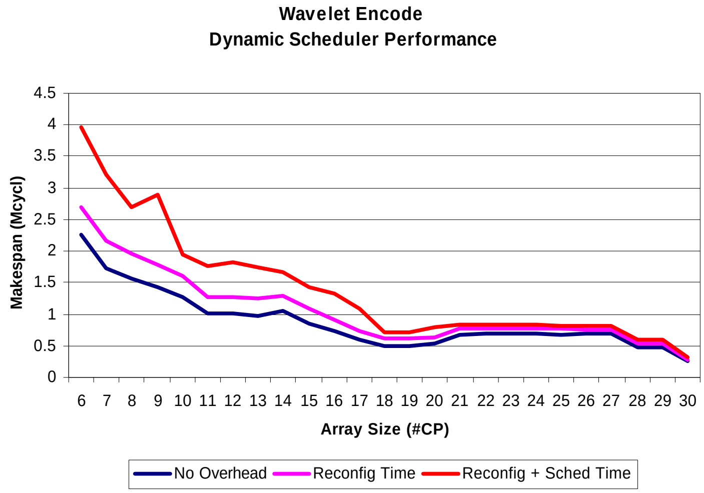
- Time-multiplex the operators onto the hardware
- To exploit scaling:
  - page capacity is a late-bound parameter
  - cannot do scheduling at compile time
- To exploit dynamic data
  - want to look at application, data characteristics

Scheduling: First Try Dynamic

- Fully Dynamic
- Time sliced
- List-scheduling based
- Very expensive:
  - 100,000-200,000 cycles
  - scheduling 30 virtual pages
  - onto 10 physical

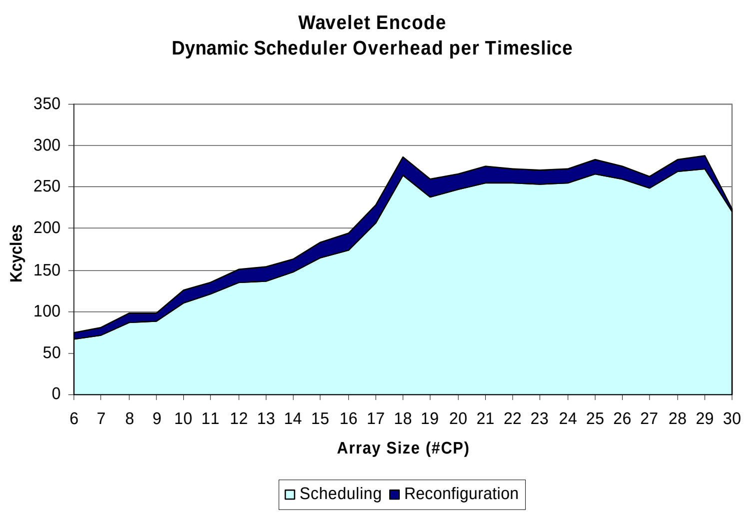


Overhead Effects



CALTECH cs184c Spring2001 -- DeHon

Overhead Costs



CALTECH cs184c Spring2001 -- DeHon

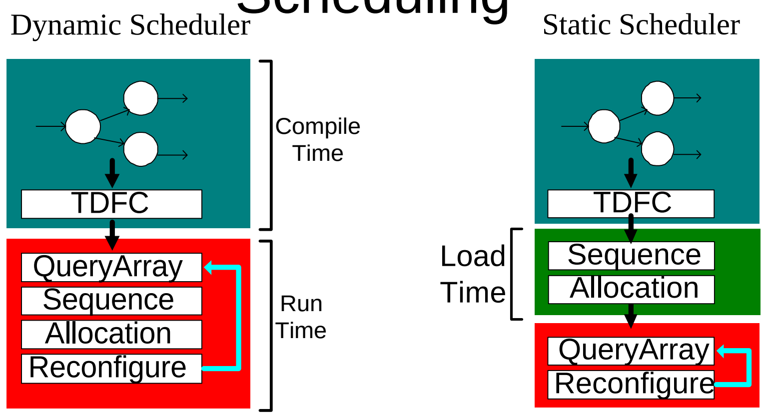
Scheduling: Why Different, Challenging

- Distributed Memory vs. Uniform Memory
  - placement/shuffling matters
- Multiple memory ports
  - increase bandwidth
  - fixed limit on number of ports available
- Schedule subgraphs
  - reduce latency and memory

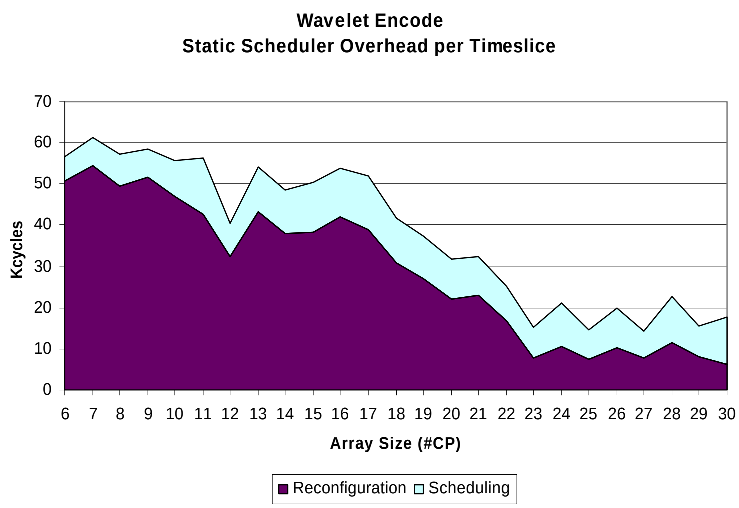
Scheduling: Taxonomy (How Dynamic?)

- Static/Dynamic Boundary?

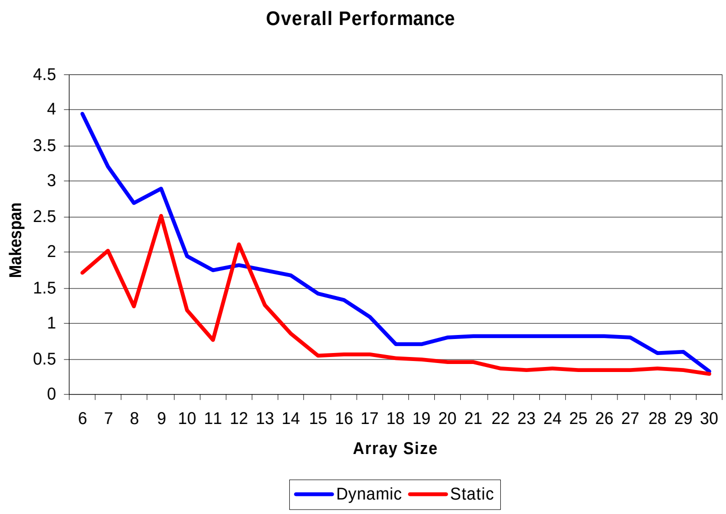
Dynamic→Load Time Scheduling



Static Scheduler Overhead



Static Scheduler Performance

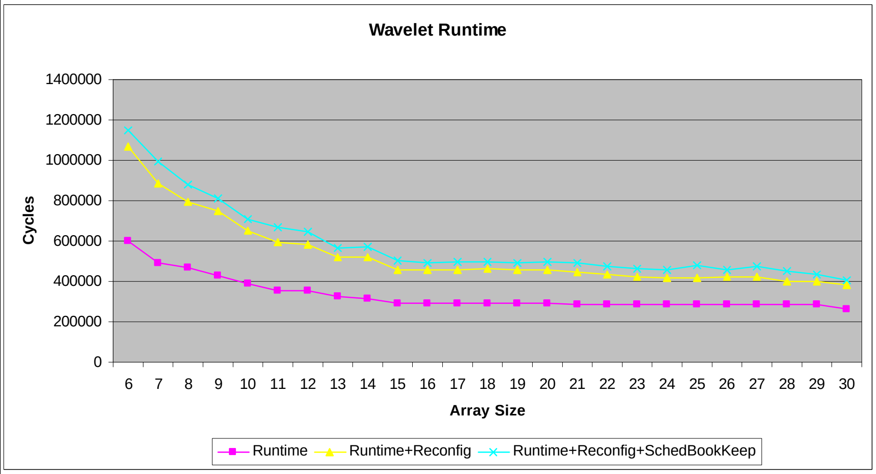


Anomalies and How Dynamic?

- Anomalies on previous graph
  - early stall on stream data
  - from assuming fixed timeslice model
- Solve by
  - dynamic epoch termination
  - detect when appropriate to advance schedule

Placement Sequence Rate Timing

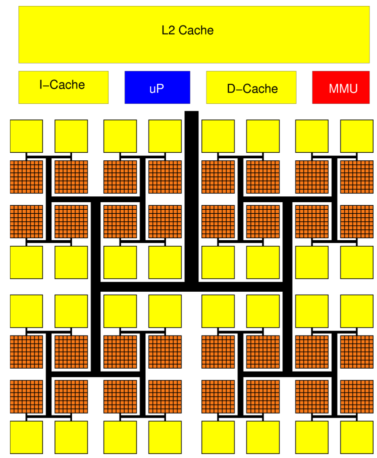
Static Scheduler w/ Early Stall Detection



More Heterogeneous Programmable SoC

Broader Programmable SOC Applicability

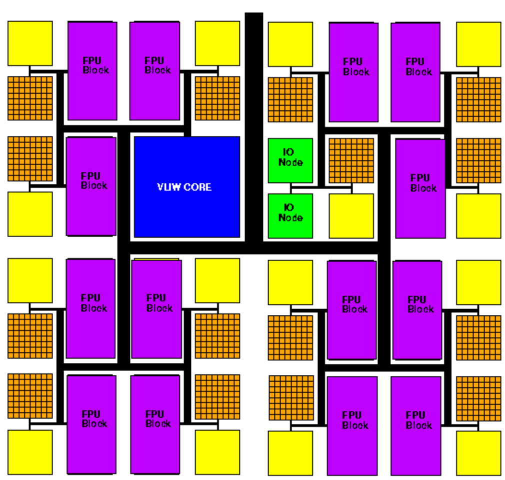
- Model potentially valuable beyond homogenous array
- Already introduced idea of different page types



Heterogeneous Pages

Small conceptual step to generalize

- Memory (CMB)
- Processor
- FPGA
  - vary granularity
  - vary depth
- IO
- Custom (e.g. FPU)



Summary

- Advantage and value for programmable spatial computing components
- Need a compute model
  – to permit device scaling
  – while preserving human effort
- SCORE model captures parallelism and freedom in these applications
- Believe it can be efficient
- Starting to get a handle on hardware/compiler/runtime support

Additional Information

- SCORE:
  – http://brass.cs.berkeley.edu/SCORE
  – especially see “Introduction and Tutorial”
- CALTECH:
  – http://www.cs.caltech.edu/research/ic/

Big Ideas

- Model
  - basis for virtualization
  - basis for scaling
  - allows common-case optimizations
  - supports kind of computations which exploit this architecture
    - spatial composition of computing blocks

Big Ideas

- Expose parallelism
  - hidden by sequential control flow in ISA-based models
- Communication to operator
  - not to resource (ala. GARP)
- Support spatial composition
  - contrast sequential composition in ISA
- Data presence [self timed!]
  - tolerant to timing and resource variations

Big Ideas

- Persistent Dataflow
  - separate creation and use
  - use many times (amortize cost of creation)
- Persistent Communication
  - separate setup/allocation form use
  - amortize out cost of routing/negotiation/setup