

Empirical Performance Modeling Using Synthetically Built Assemble Directives (SynBAD)

Kirk W. Cameron

Parallel Architecture Team

Computer, Information, & Communications Division

Los Alamos National Laboratory

What is SynBAD?

- Allows creation of synthetic instruction streams to assist in performance modeling
- User has control over instruction types and dependence relations introduced
- Executable meeting user specifications is created and executed
- Underlying performance counters are used to collect run-time information
- Tool designed to be used across different platforms for comparison of results

Why use SynBAD?

- Instruction-level model development
 - *became necessary to isolate stall contributors*
 - *need for expansion of existing models*
 - *desire to obtain closer estimates of cpi0*
- Processor benchmarking
 - *desire to empirically infer architectural limitations across architectures*
 - *need to empirically understand performance utilization*

Categorizing Data Dependences

- Necessary to determine an expressive, intuitive, method for describing data dependences in streams
- We identify 4 characteristics of dependences at the instruction level
 - *inter-queue influence*
 - *intra-queue influence*
 - *link-length*
 - *link-width*

Dependence Characteristics

- Inter-queue influence
 - *dependence involving two instructions that reside within a single on-chip instruction queue*
- Intra-queue influence
 - *dependence involving two instructions that reside in different on-chip instruction queues*
- link-length
 - *# of consecutive linked dependences between two instructions*
- link-width
 - *# instructions between two dependent instructions in program order*

SynBAD supported instructions

- Floating point add (fa)
- Floating point multiply (fm)
- Floating point store (fs)
- Floating point load (fl)
- Jumps*
- Integer add (ia)
- Integer multiply (im)
- Integer store (is)
- Integer load (il)

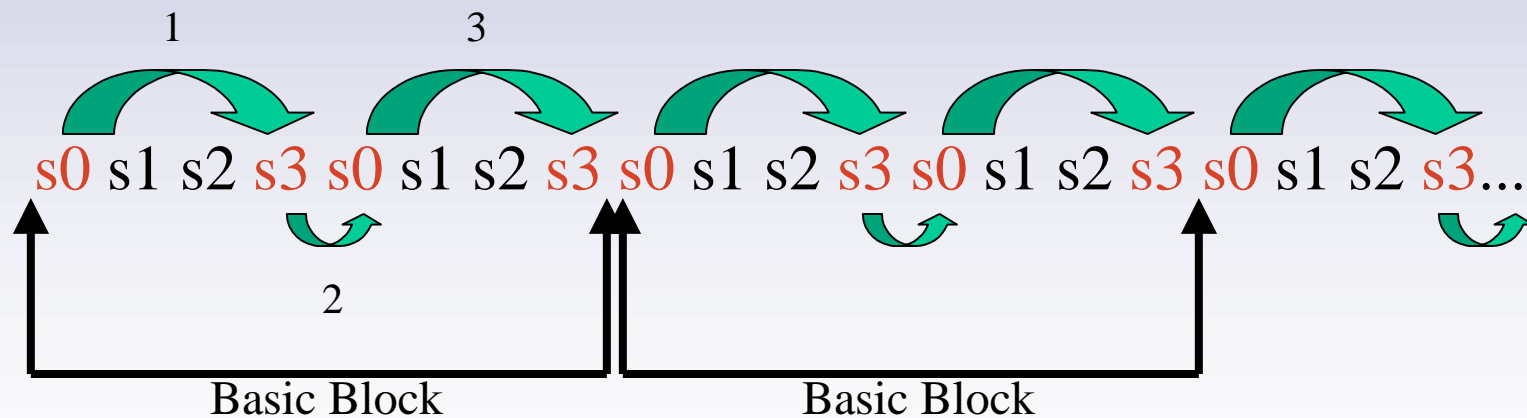
*version including jumping is work in progress

How SynBAD works

- User creates configuration file describing instruction stream and dependence relationship
- SynBAD creates basic block with user-specified qualities
- Only allowable dependences created (errors caught)
- SynBAD creates executable code using basic blocks
- Executable code contains negligible cache misses, icache misses, branches
- Spread-sheet ready output from counters stored in specified files

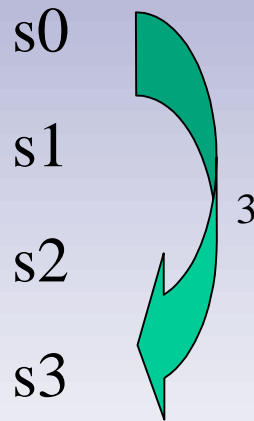
SynBAD Example: english description

- Create a stream with the following characteristics
 - s_0, s_1, s_2, s_3 compose basic instruction set
 - s_3 is data dependent on s_0 (s_3 needs s_0 's data to complete)
 - this dependence is a linked dependence with length 3 (i.e. dependence is $s_0 \rightarrow s_3 \rightarrow s_0 \rightarrow s_3$)



SynBAD Example: Input version

How SynBAD thinks:



User configuration file:

```

4           ← # instructions
s0
s1         ← Specify each instruction
s2
s3
U X X 3   ← Dep from s0->s3,length=3
U U X X   ← No dep this row
U U U X
U U U U
    
```

Varying Configurations

- Using configuration file, we can vary three parameters
- link-length: varied by creating lengths of any value as well as an infinite link
- link-width: varied by creating larger basic instruction sets, making for larger basic blocks
- instruction types: varied by inserting instructions of acceptable types in different mixes, numbers, etc.
 - *this allows varying the inter-queue and intra-queue influence*

The “ideal” experiments

- Use only instruction streams that can achieve ideal cpi (approx. .25 on MIPS R10K)
- use basic instruction sets of length 4
- use all combinations (256) with a single dependence
- isolate for combinations that allow link-length > 1 dependence from first to fourth instruction (s0- \rightarrow s3)
- 16 combinations found
- vary link-length and link-width
- analyze results

“ideal” experiment results

- Collected data for all 256 combinations using SynBAD
- Varied basic instruction set: 4 (4x4), to 8 (8x8), ...36
 - *4x4 means 4 instr in basic instruction set*
 - *8x8 means 8 instr in basic instruction set*
 - *fix location of dependence (from s0->s3) regardless of basic instruction set size*
- Analyzed 16 fitting criteria
- Dependences found and analyzed:
 - *fa->fm, fm->fa*
 - *ia->ia*
 - *fs->fm, fs->fa, is->ia*

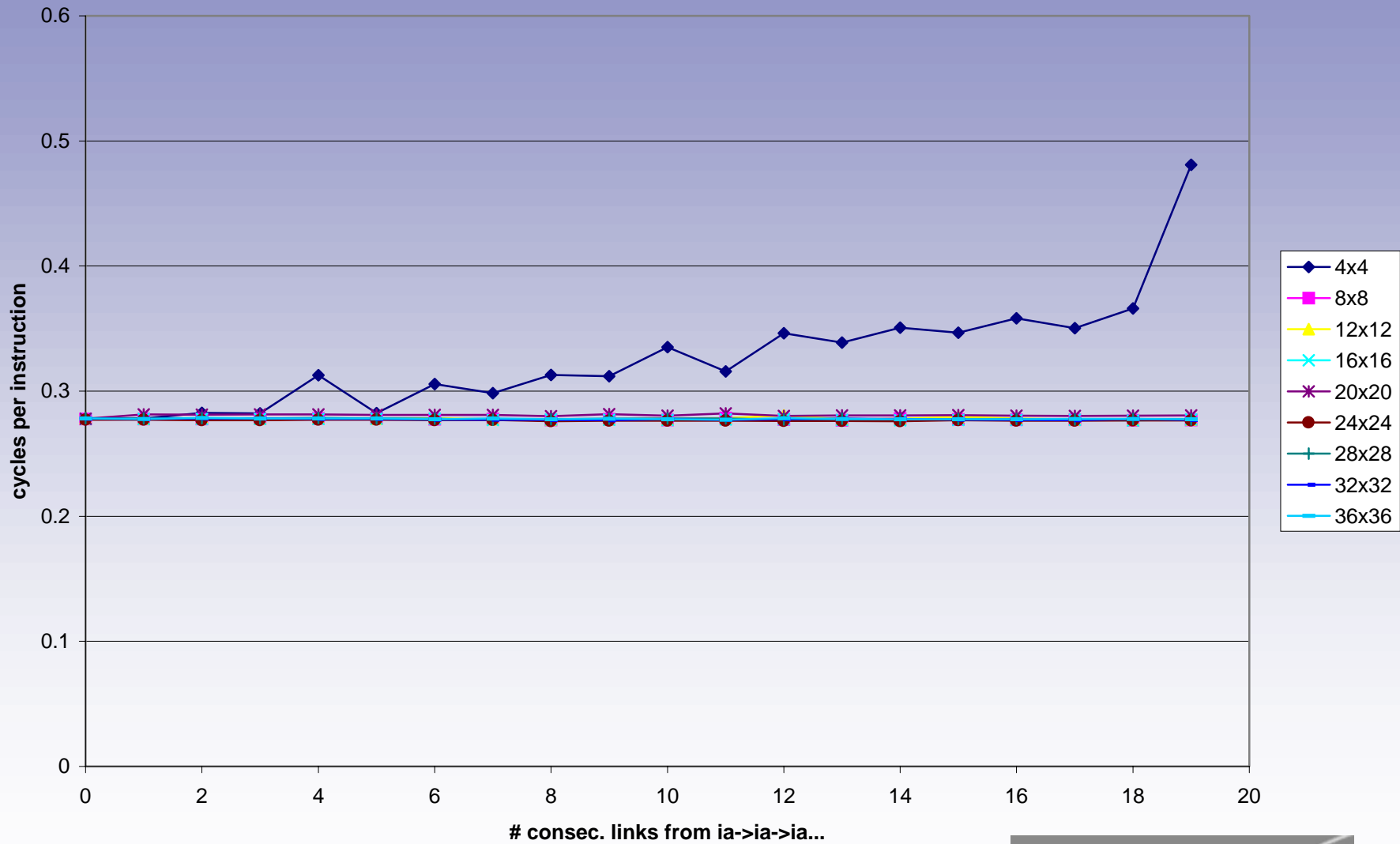
“ideal” experiment conclusions

- fa->fm, fm->fa
 - *increasing trends for 4x4, 8x8, 12x12 showing degradation of performance as link length increases*
 - *no link-length influence for 16x16 and above*
 - *performance increases with link-width to a plateau above 16x16*
- ia->ia
 - *increasing trends for 4x4 show degradation of performance as link length increases*
 - *no link-length influence for 8x8 and above*
 - *performance increases with link-width almost immediately above 8x8*

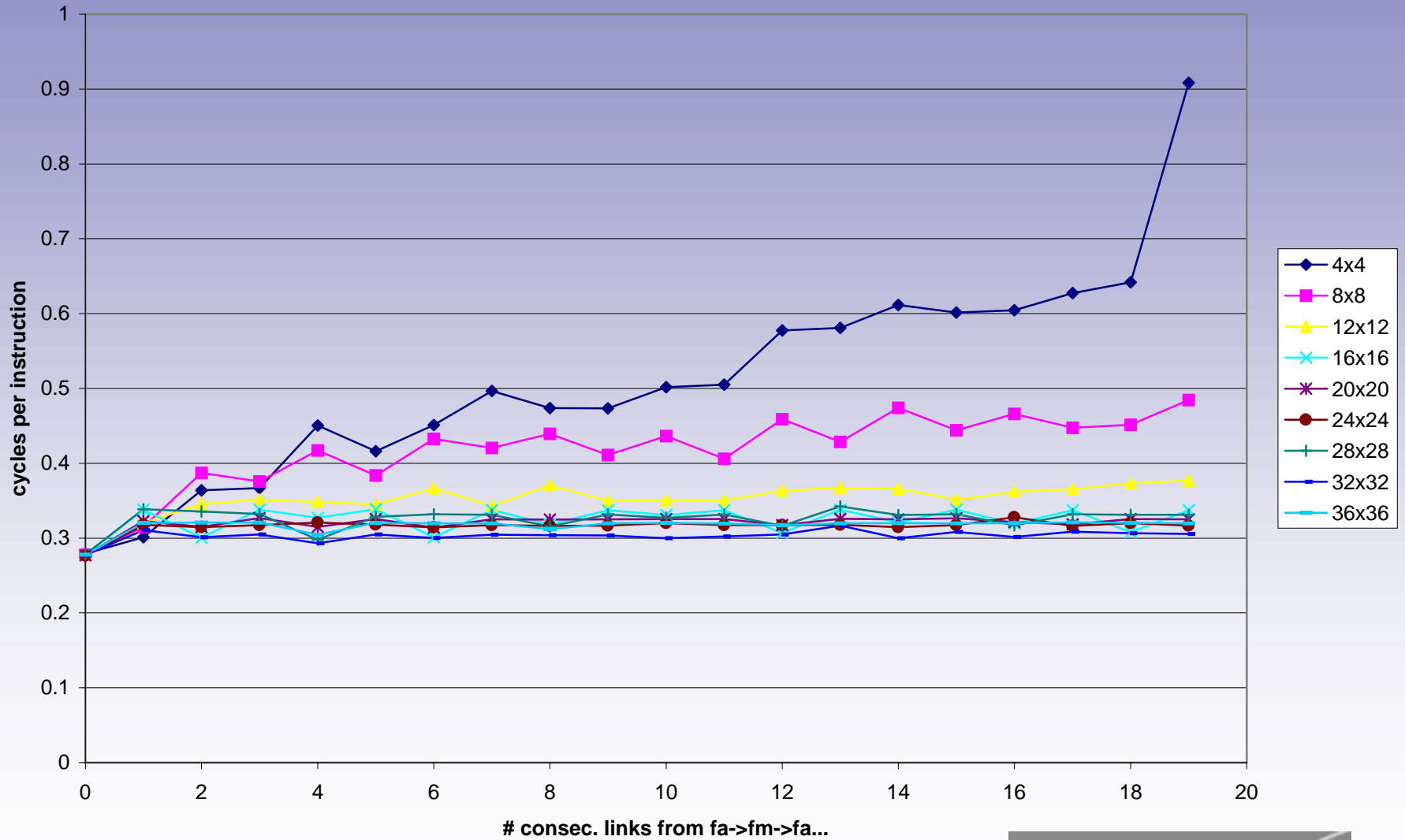
“ideal” experiment conclusions

- fs->fm, fs-fa, is->ia
 - *showed no performance degradation as link-width and link-length were varied*
 - *no sensitivity to this type of dependence*

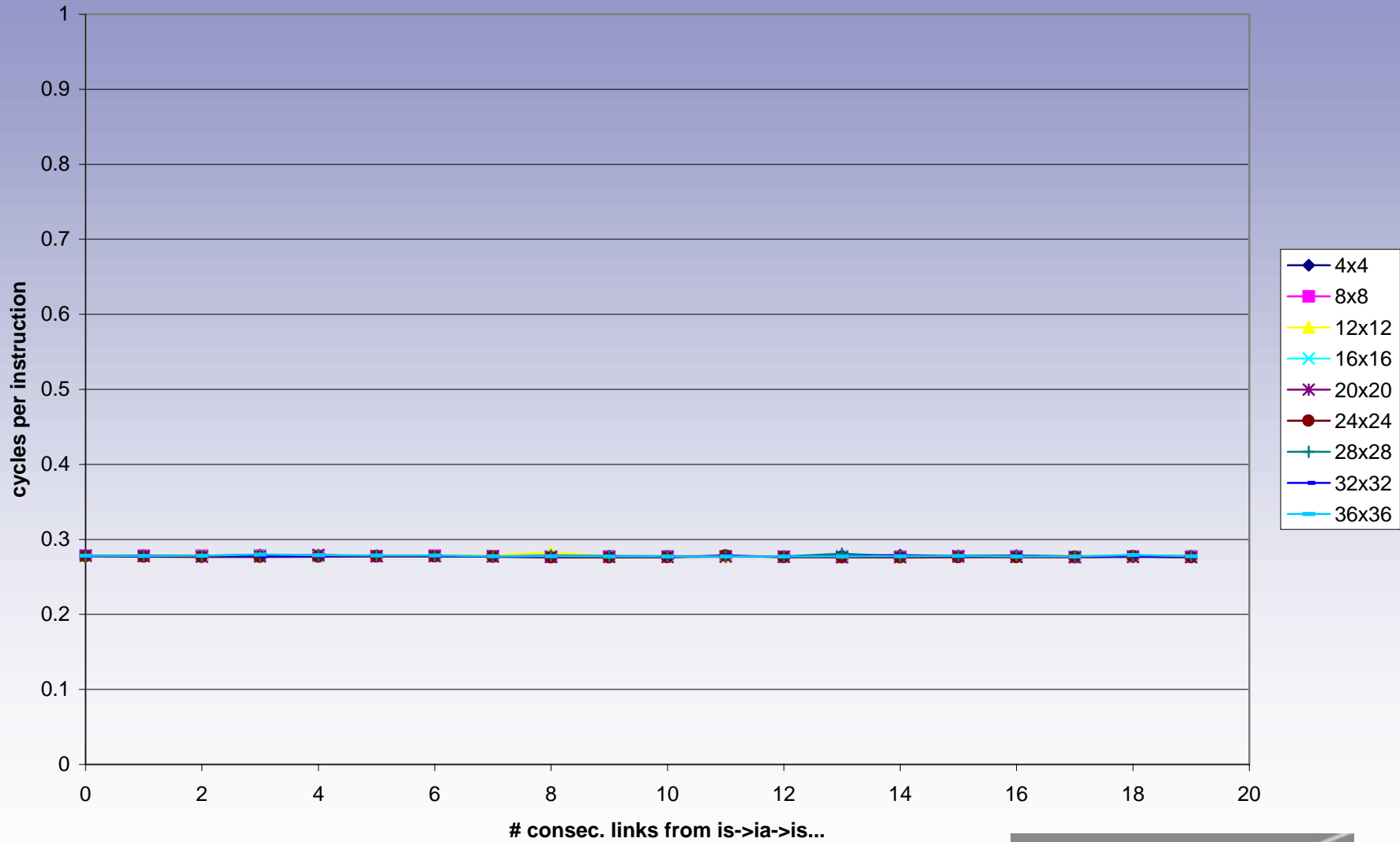
iafmfaia (cpi vs. dep. link length)



faiaiafm (cpi vs. dep. link length)



isfafia (cpi vs. dep. link length)



Future Work

- Incorporate jump and other instructions into SynBAD
- Perform more exhaustive analyses for other stream and dependence combinations (including multiple dep)
- Expand to CISC-based processor to prove portability

Modeling Motivations

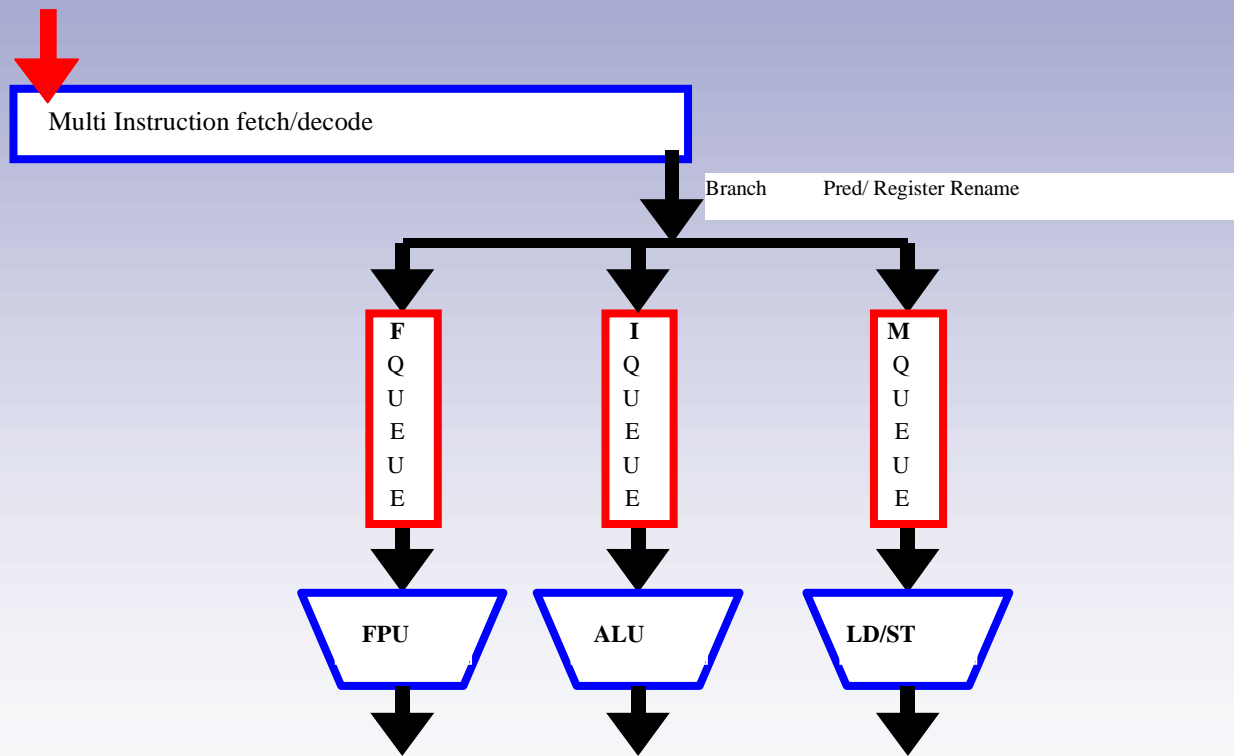
- Processor Utilization Efficiency
- Utilization of architectural features
- Separation and quantification of other effects (data dependency etc)

Modeling Methodology

- Counter-based average abstract parameters related to architectural pipeline queues
- Analysis Assumptions:
 - *no data dependency, uniform instruction distribution, branches and lcache misses negligible*
- Saturation based bottleneck analysis

General Pipeline Model (CPU Only)

Incoming Instruction Stream



Queueing-based Parameters

C_n denotes n th customer to arrive

τ is the associated arrival for C_n (equal to n)

$t_n = \tau_n - \tau_r = \text{inter - arrival distance (} r \text{ is last occurrence of this type)}$

$x_n = \text{service time in cycles for } C_n$

$\bar{t} = \text{avg inter - arrival distance (no unit)}$

$\bar{x} = \text{avg service time in cycles for this type}$

Architectural Parameters

β = max num instructions fetched per cycle (IPC)

m = number of servers for type

Instruction-level Formulae

- Number of instructions introduced every cycle

$$\lambda = \beta / \bar{t}$$

- Utilization Factor: expected fraction of system capacity in use

$$\rho = (\lambda \times \bar{x}) / m$$

- Capacity: max rate system can perform work

$$C = m$$

- Work rate: avg rate of work demanded from system

$$R = \lambda \times \bar{x}$$

Instruction-level Analysis

- $\rho > 1$ Indicates saturation
- if $R < C$ then system capacity is sufficient
- else if $R > C$ then system capacity is insufficient
- Instructions such that $R/C > 1$ limits IPC by causing architecturally limited stalls
- For such cases we can calculate IPC_0

$$\text{ideal } ipc_0 = (\bar{t} \times m) / \bar{x}$$

Scientific Examples

- Test bed: MIPS R10000
- Benchmarks: ASCI application benchmark codes
- General Characteristics:
 - *Branches $\leq 10\%$, miss-pred_branches $\leq 1\%$*
 - *Icache misses $\leq 0.1\%$*
 - *All t's converge to constants (steady state)*

Architectural Constraints (MIPS R10000)

- Queue Lengths: 16 entries for all three queues
- Max. Instructions in Flight: 32
- Graduation Rates: 2/cycle for FP, 2/cycle for INT,
1/cycle for Mem
- Outstanding Misses: 4 on Origin2000, ≤ 2 on
PowerChallenge

Example Growth Rates (ASCI Benchmarks)

$$\rho = (\lambda \times \bar{x}) / m$$

	Sweep			Dsweep			Heat			Hydro			Hydro-t		
	ρ_m	ρ_i	ρ_f	ρ_m	ρ_i	ρ_f	ρ_m	ρ_i	ρ_f	ρ_m	ρ_i	ρ_f	ρ_m	ρ_i	ρ_f
PowerChallenge	1.41	0.63	0.66	1.84	0.68	0.40	1.43	0.84	0.45	1.08	1.05	0.40	1.08	1.06	0.40
Origin 2000	1.42	0.62	0.67	1.89	0.65	0.40	1.42	0.84	0.45	1.09	1.05	0.40	1.08	1.06	0.40