

Reconciling Performance and Programmability in Networking Systems

Jayaram Mudigonda
Hewlett-Packard Labs, Palo Alto / University of Texas, Austin

Prof. Harrick M. Vin and Prof. Stephen W. Keckler
University of Texas at Austin

1

Twin Goals: Performance & Programmability

- n Trend: Richer services, applications and architectures
 - q Stateful firewalls and virus scanners
 - q DDoS detection and mitigation systems
 - q Load balancing switches
 - q XML and SOA routers
 - q Internet Indirection Infrastructure

 - n Twin goals: High performance and Ease of programming
 - q Effective experimentation on realistic scales
 - q Rapid development & Incremental deployment
-

2

Trends in Platform Design

- n Achieving the twin goals has proved to be difficult
 - q General-purpose machines
 - J Easier to program
 - L Substantially lower throughput
 - q ASIC-based platforms
 - L Limited/no programmability
 - J Much higher throughput
- n Latest platforms are based on Network Processors
 - q NPs: multicore multithreaded processors
- n Current NPs are still not easy to program

3

The Memory Bottleneck

- n A major challenge: The memory bottleneck
 - q Worsening with time
 - q Primary limitation for high throughput
- n Prior work
 - q Algorithmic techniques
 - n Route and classification schemes
 - q Special-purpose hardware
 - n Special route- and classification- caches
 - n Specialized buffer memories for packet payload

4

This Paper

- n Focuses on processor support
 - q Generic - complements most of the prior work
 - q Radical shift in processor architectures à opportune time

Two Questions:

1. What architectural mechanisms should a processor support?
 2. How to exercise these mechanisms automatically?
-

5

State of The Art

- n Two primary mechanisms
 - q Caching: exploits locality to reduce the average latency
 - q HW multithreading: exploits parallelism to hide the latency
 - n Need same resource: fast memory close to the pipeline
 - q Caching: Cache memory
 - q Threading: Separate register file for each thread
 - n Current practice: allocate chip-area at chip-design time
 - q Fixed cache capacity and # of threads
-

6

Contributions

- n Case for a malleable processor
 - q Dynamic tradeoff between threads and cache
 - n Realizing the malleable processor
 - q Novel predictive prefetching of registers
 - n Automatically exploiting the malleability
 - q Run-time adaptation with low overhead (< 3%)
 - n Comparison to an IXP2800-like processor
 - q Outperforms in about 88% of deployments
 - q In 26% of deployments throughput quadruples
-

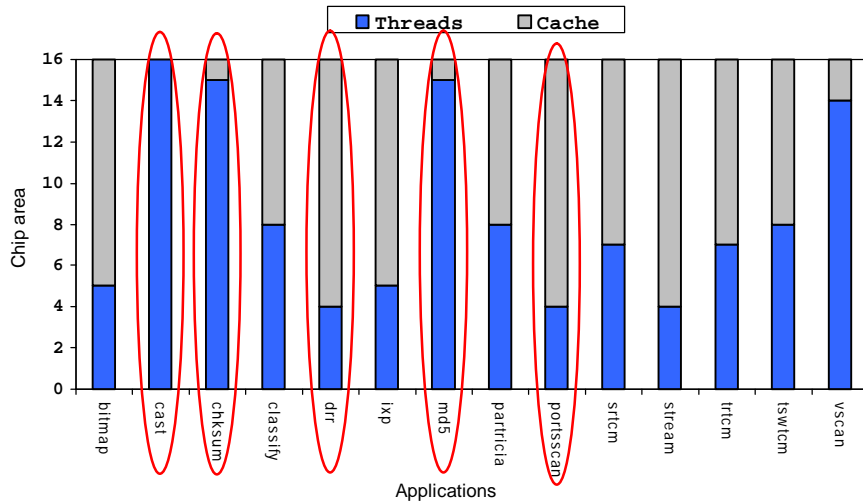
7

Case for Malleability

- n Studied 2000+ "deployments"
 - q Deployment = <App, PktTrace, MemLatency, MemBW>
 - q Applications - typical packet processing steps
 - q Packet traces from NLANR and UNC
 - q SimpleScalar - cycle accurate CPU, cache, memroy simulator
 - n Optimal thread-cache balance varies across deployments
 - q Several simultaneous factors with significant influence
-

8

Thread-Cache Balance: Effect of Application



9

Case for Malleability

- n Several simultaneous factors influence the balance
 - q Application : Data locality, critical sections
 - q Traffic: Number and burstiness of flows
 - q System: Off-chip memory latency and bandwidth
- n Variance in thread-cache balance + fixed provisioning à
 - q Programmers struggle with ill-suited hardware
- n Malleability (dynamic thread-cache tradeoff)
 - q Achieves impressive throughput gains
 - q Plausible:
 - n Replace cache and register files with malleable storage

10

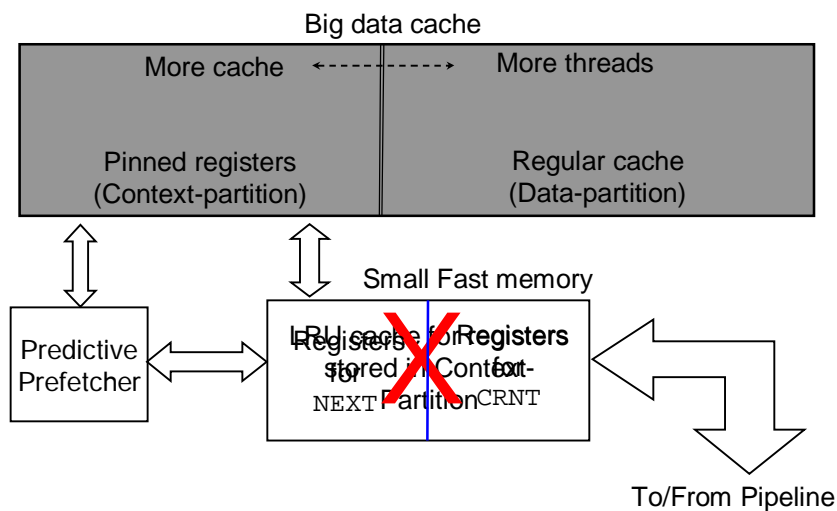
Realizing Malleability - The Challenge

	Registers	Data
Size	Small (0.25 KB)	Large (10's of KB)
Bandwidth	High (3 refs/cycle)	Low (~0.33 refs/cycle)
Latency	Low (1 cycle)	Moderate (2-4 cycles)
Addressing	Index	Associative

- n Register files: sacrifice **capacity** for **speed and bandwidth**
- n Caches: sacrifice **speed and bandwidth** for **capacity**
- n Malleable storage needs all three: capacity, speed, and bandwidth
 - q No feasible circuit-level techniques to achieve all three at once
- n Need novel architectural techniques

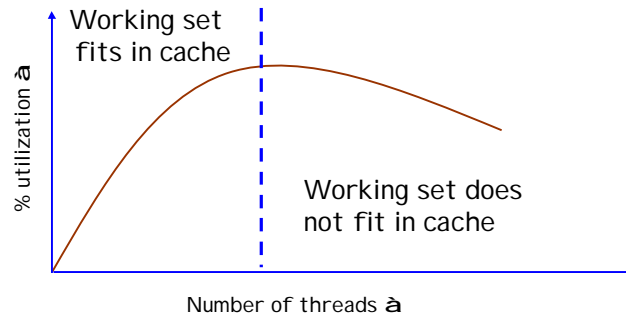
11

Our Architecture



12

Exploiting Malleability - Key Intuition



- n Key intuition: (Holds in all deployments)
With increasing number of threads, utilization, always first increases, and then decreases

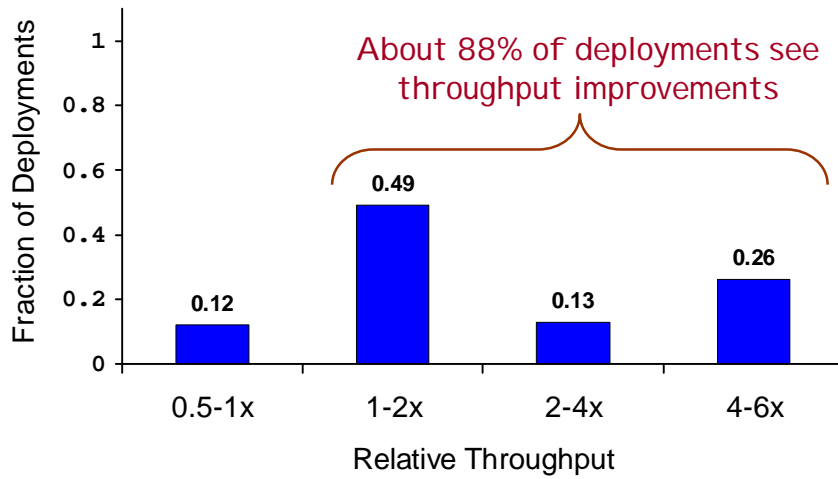
13

Exploiting Malleability - Run-time Adaptation

- n Simple linear search works well
- n Low overhead \bar{a} can be done in data path
 - q Few 10s of instructions and about 6 memory references
- n Malleable processor + run-time adaptation \bar{a}
 - q No ad hoc performance tuning headaches
 - q Networking researchers can focus on networking

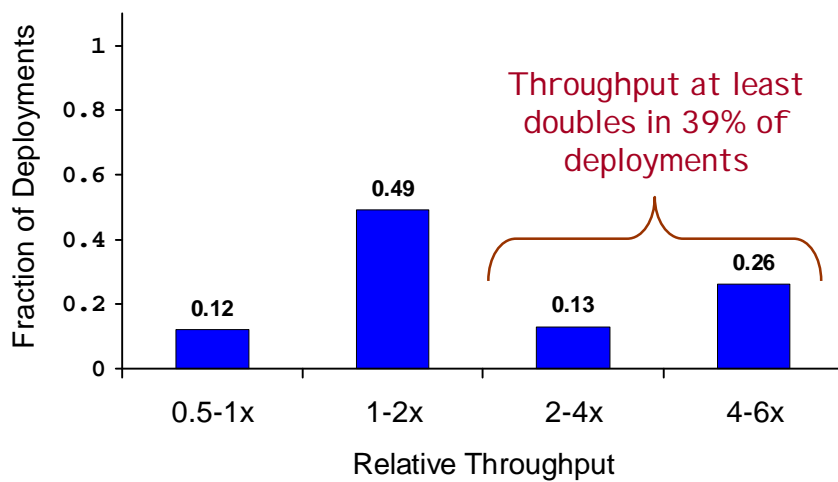
14

Our Platform vs. IXP2800-like



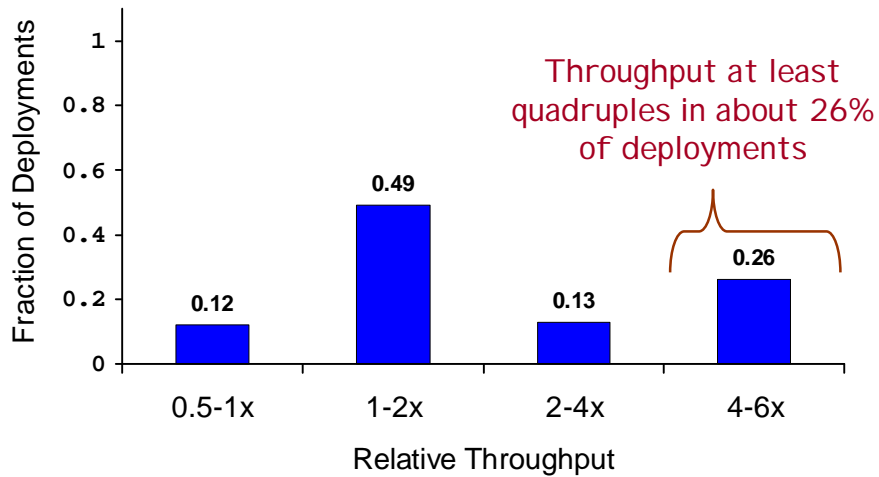
15

Our Platform vs. IXP2800-like



16

Our Platform vs. IXP2800-like



17

Conclusions

- n Richer services and applications need
 - q High performance and Ease-of-programming
- n Malleability (dynamic thread-cache tradeoff) is needed
- n Novel malleable processor architecture
- n Algorithm for automatic exploitation
- n Our platform:
 - q Achieves impressive throughput gains
 - q Automatically handles the memory bottleneck
- n Broader applicability to request processing systems

18