

Capriccio: Scalable Threads for Internet Services

Lee Chew

OVERVIEW

- 1) Problem
- 2) Approach
- 3) Outcome
- 4) Related Work
- 5) Future Work

“THE DIGG EFFECT”

- need scalable servers
- need an easy programming model

HOW TO SOLVE?

- Eric Brewer, circa 2001 – “Threads suck – lets use events !”
- Eric Brewere, circa 2003 – “Events suck – lets use threads !”

LETS TRY THREADS THIS TIME

But with improvements:

1. User-level Threads
2. Linked Stacks
3. Resource-aware Scheduler

USER-LEVEL THREADS

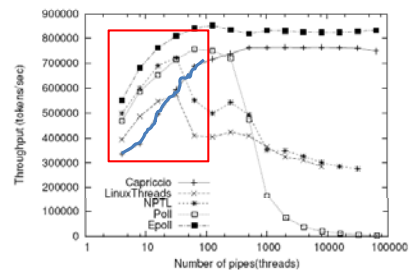
Benefits

- cooperative threading
- use less kernel address space

Drawbacks

- blocking → non-blocking conversion overhead

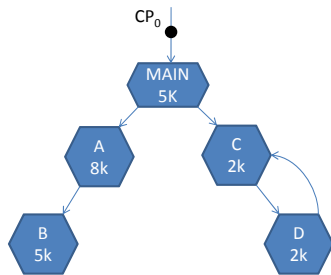
USER-LEVEL THREADS



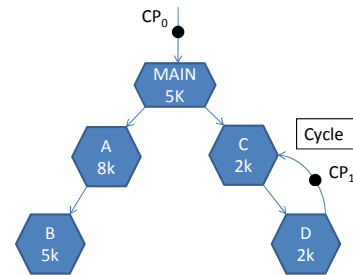
LINKED STACKS

- stop wasting memory!
 - static vs. dynamic stack allocation
- create *weighted call graph*
 - checkpoints to prevent stack overflow
- with MaxPath = 2kB & MinChunk = 4kB, overall slowdown of 3-4% in Apache

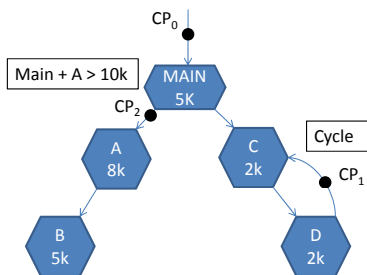
LINKED STACKS: EXAMPLE



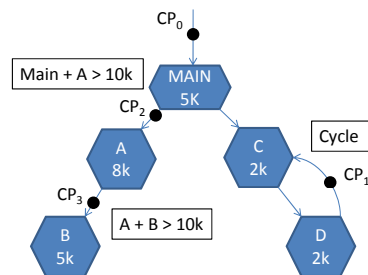
LINKED STACKS: EXAMPLE

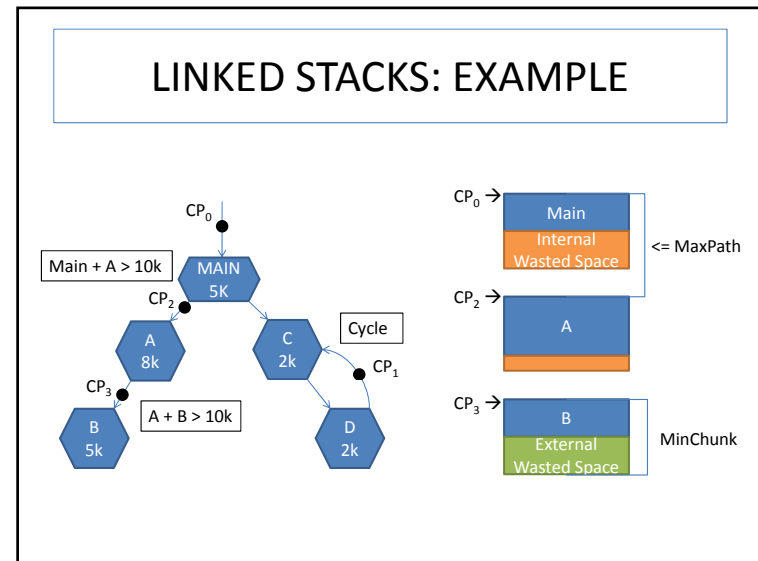
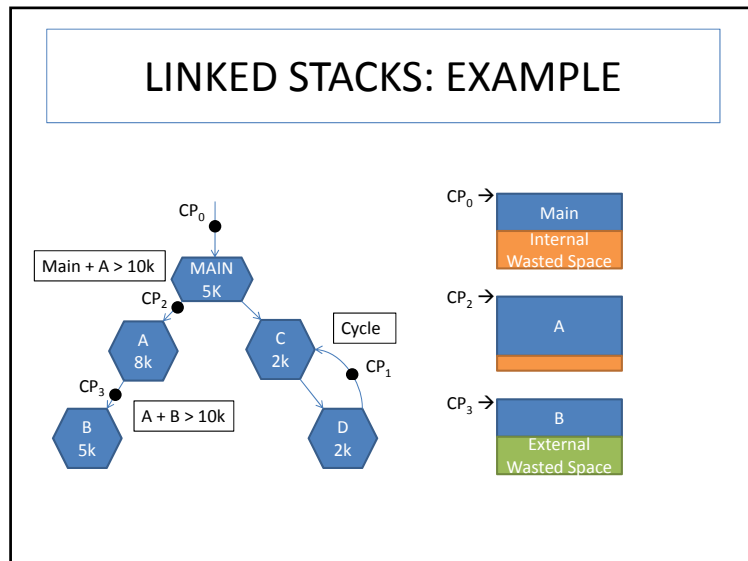
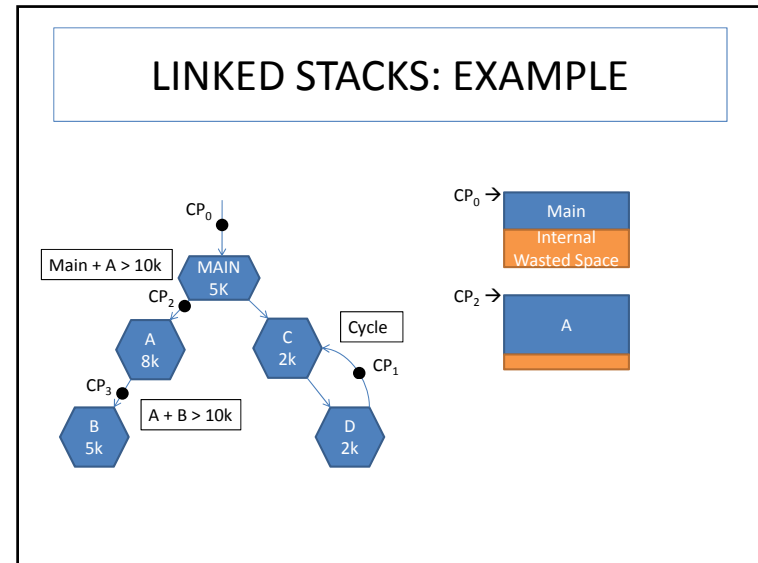
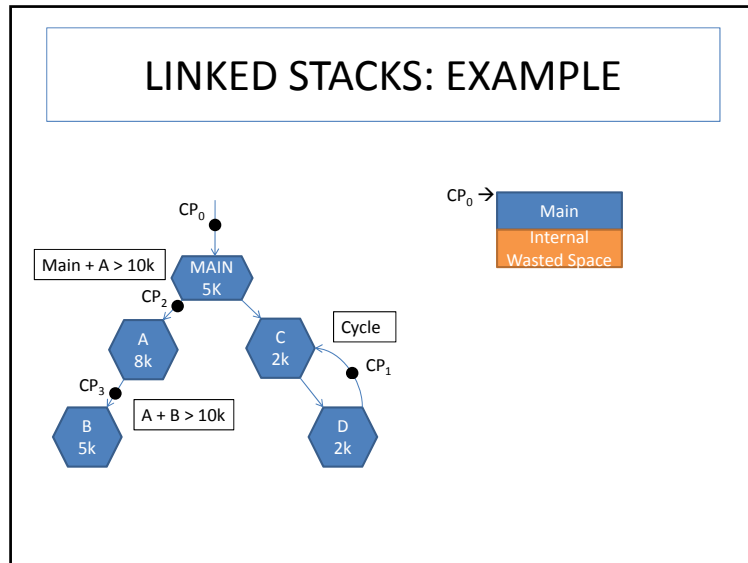


LINKED STACKS: EXAMPLE



LINKED STACKS: EXAMPLE





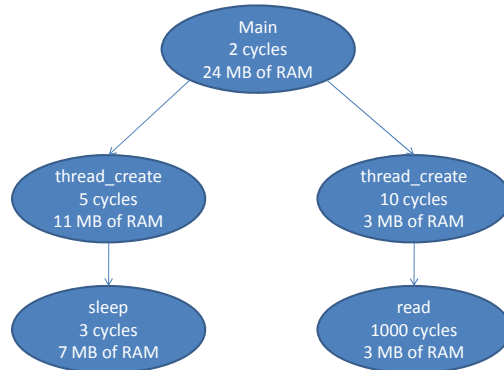
RESOURCE-AWARE SCHEDULING

- each time we schedule a new thread:
 - if resources are high → pick threads that are going to use lots of resources
 - if resources are low → pick threads that are going to release lots of resources

RESOURCE-AWARE SCHEDULING

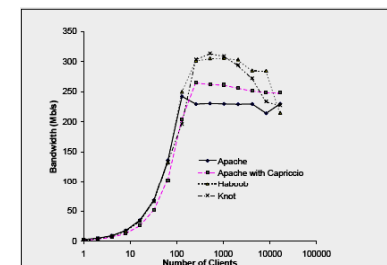
1. How do we know system resource levels?
 - track CPU, memory and file descriptors
 - watch for thrashing, thread creation/deletion rate, file open/close rate
 - invisible resources? invisible changes?
2. How do we know whether a thread will use/release resources?
 - use a *Blocking Graph* to record past behaviour

RESOURCE-AWARE SCHEDULING: BLOCKING GRAPH



RESULTS

Web Server Bandwidth



4x500 MHz Pentium Server, 2GB Memory, Gigabit Ethernet
Linux 2.4.20

RESULTS

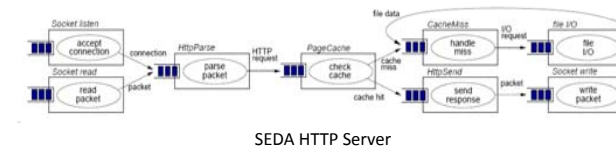
Average Cycle Count Stats

Item	Cycles	Enabled
Stack Trace	2447	Always for dynamic BG
Edge Statistics	673	During sampling periods

Application	Cycles	Stack Trace	Edge Statistics
Apache	32697	7.5%	2%
Knot	6868	35.6%	9.8%

RELATED WORK

- event-driven systems (e.g., SEDA)
 - pipeline of stages
 - stages are connected by queues
 - stages can be individually tuned



RELATED WORK

- stack management:
 - heap-allocated activation frames
 - “spaghetti stacks”
 - Lazy Threads project
- application-specific optimizations:
 - exokernels
 - microkernels

FUTURE WORK

Authors

- improve existing algorithms (e.g. thrashing detection)
- add multi-core support, profiling tools, fairness, compile-time blocking graph generation

Myself

- emphasize usability
- look at performance degradation

CONCLUSION

- improved thread package:
 - scales to 100,000's of threads
 - adapts to applications
- comparable to event-driven systems