

Evaluation of KPN implementation techniques

Željko Vrba^{1,2} Pål Halvorsen^{1,2} Carsten Griwodz^{1,2}
{zvrba,paalh,griff}@ifi.uio.no

¹Institute for informatics, University of Oslo

²Simula research laboratory

2009-03-16

- 1 Introduction
- 2 Kahn process networks
- 3 Implementation
- 4 Performance evaluation
- 5 Summary

Trends

- beginnings: Intel's hyperthreading
- AMD, Intel – true multi-core, small scale
- Sun – Niagara T2, 8×8 hardware threads
- More total MIPS/FLOPS, but slower sequential

Parallel programming challenges

- shared memory
- thread-based concurrency
- low-level synchronization (mutexes, CVs)
- \Rightarrow non-determinism

Why do we need another programming model

Existing approaches...

- futures, polyphonic C#, Erlang
- software transactional memory
- MapReduce, Dryad
- other suggestions?

...and their problems

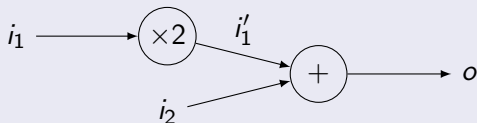
- determinism?
- efficiently implementable for multi-core?
- modeling, generality?

Kahn process networks: simple, intuitive, generic!

Kahn process networks

Definition

Example



Building blocks

- sequential, asynchronous, isolated processes [nothing-shared]
- 1:1 channels, inputs, outputs
- non-blocking send, blocking receive
- infinite channel capacity
- \Rightarrow deterministic [debuggability!]

Kahn process networks

Practical considerations

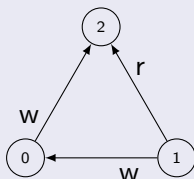
Theoretical assumptions

- fair scheduling – OK (otherwise: correct, but possibly incomplete output)
- infinite capacities, arbitrary process behavior – NOT OK

Practical solutions

- *assume* effective networks; cannot be enforced
- finite capacities, redefine send to block on full channels
- computing sufficient capacities – undecidable
- \Rightarrow need deadlock detection and resolution

Deadlock example



- 1:1 channels \Rightarrow simple DD implementation [no back-tracking]
- both read and write blockings have to be considered
- enlarge the full channel in the cycle with least capacity

Desirable properties

- support arbitrary process graphs
- be resilient to “wrong” patterns
- do as much as possible automatically
- and do it all *efficiently!*

Consequences for implementation

- blocking & arbitrary graphs \Rightarrow must support overdecomposition
- varying amounts of work per message

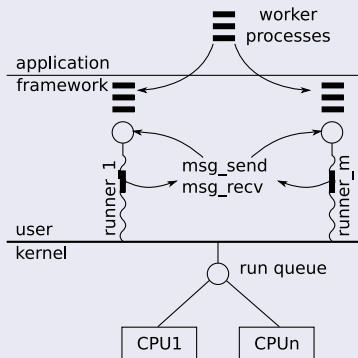
“Naive” implementation

- 1:1 preemptive scheduling (1 thread / KP)
- native message queues for transport (non-blocking for DD)
- condition variables for sleeping

Optimizations

- m:n cooperative scheduling (user-mode ctx switch)
- own message transport (resizeable circular queue)
- direct call into scheduler to sleep (no need to support multiple sleepers)

m:n scheduler

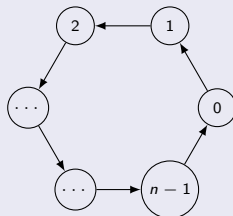


- m OS threads (runners) time-multiplex n Kahn processes
- cooperative scheduling; *static assignment*
- yield points: send/receive
- DD integrated w/scheduler

Performance evaluation

Ring

Topology

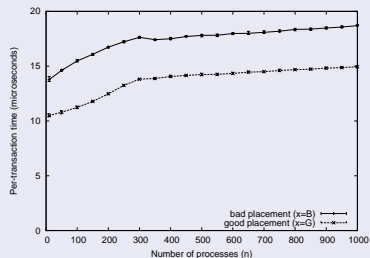


- little (no) processing time per message
- number of round-trips: $\sim 10^6$ transactions
- transaction: [send \rightarrow ctx switch \rightarrow receive]

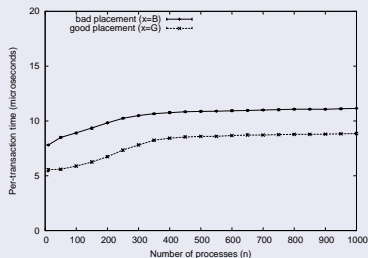
Performance evaluation

Ring: message transport implementation

pthread / POSIX MQ



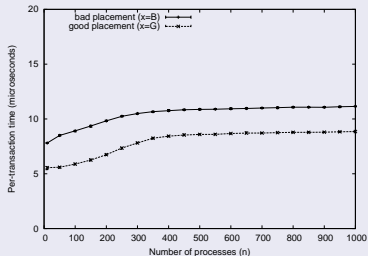
pthread / optimized



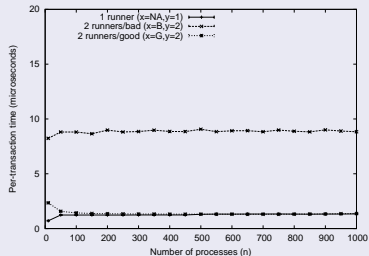
Performance evaluation

Ring: scheduler implementation

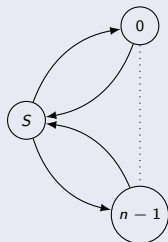
pthread / optimized



m:n / optimized



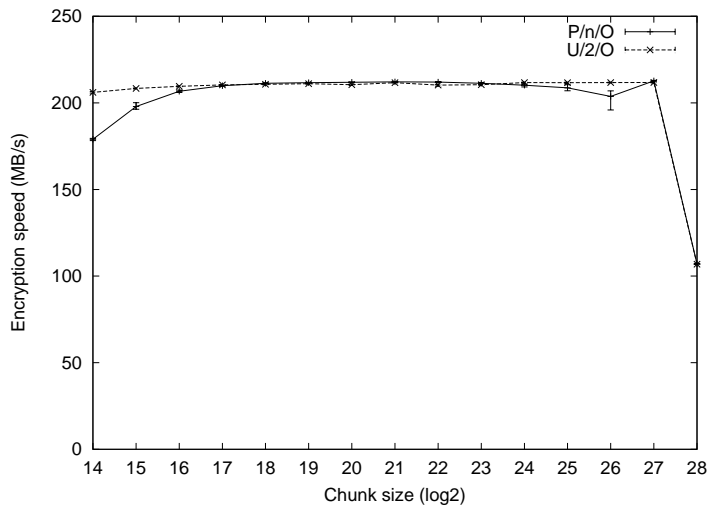
Topology



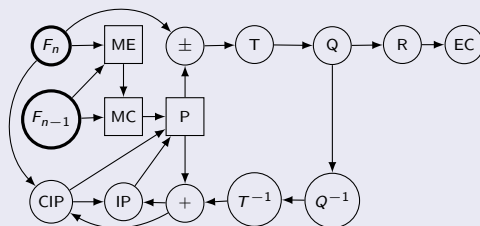
- a lot of processing time per message
- $w = 2^{28-c} \in \{1, 2, 4, \dots, 2^{14}\}$
- 2 messages and 8 repetitions per chunk
- \Rightarrow number of transactions: $2 * 8 * w \in \{16, 32, \dots, 2^{18}\}$

Performance evaluation

AES



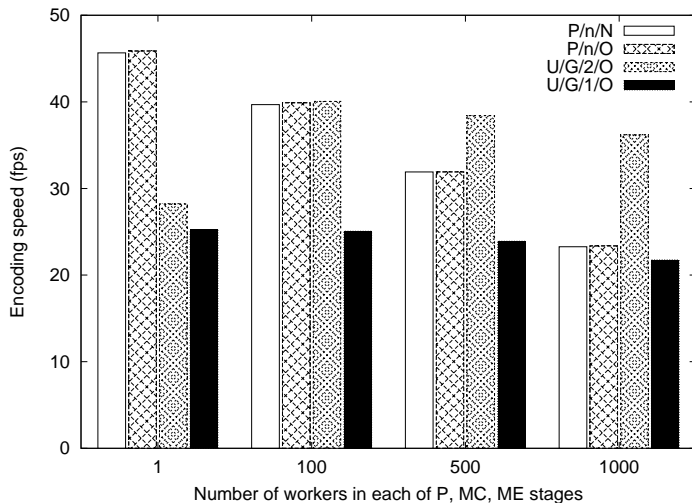
Topology



- few messages exchanged
- lot of processing time per message

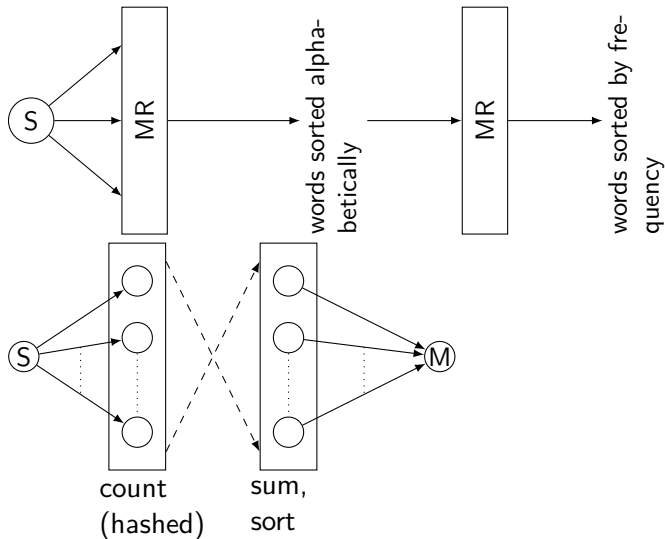
Performance evaluation

H.264



Performance evaluation

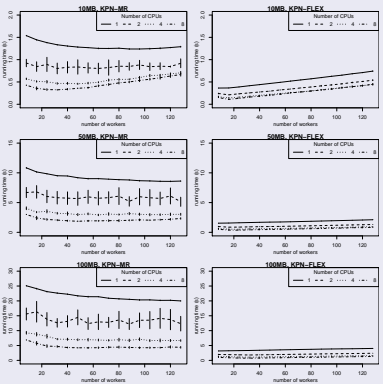
Word count



Performance evaluation

Word count

MR vs. specialized topology



Summary

- KPNs are a viable and simple parallel programming model
- it is efficiently implementable – esp. with work stealing
- <http://simula.no/research/networks/software> (look for “Nornir”)
- achieved resiliency to “wrong” patterns

What next?

- scheduling policies: (modified) work stealing, graph partitioning
- performance optimizations (lock-free, D4R)
- distributed version
- simpler network description language