An Online Computation of Critical Path Profiling

Jeff Hollingsworth

© Copyright 1996, Jeffrey K. Hollingsworth, All Rights Reserved.

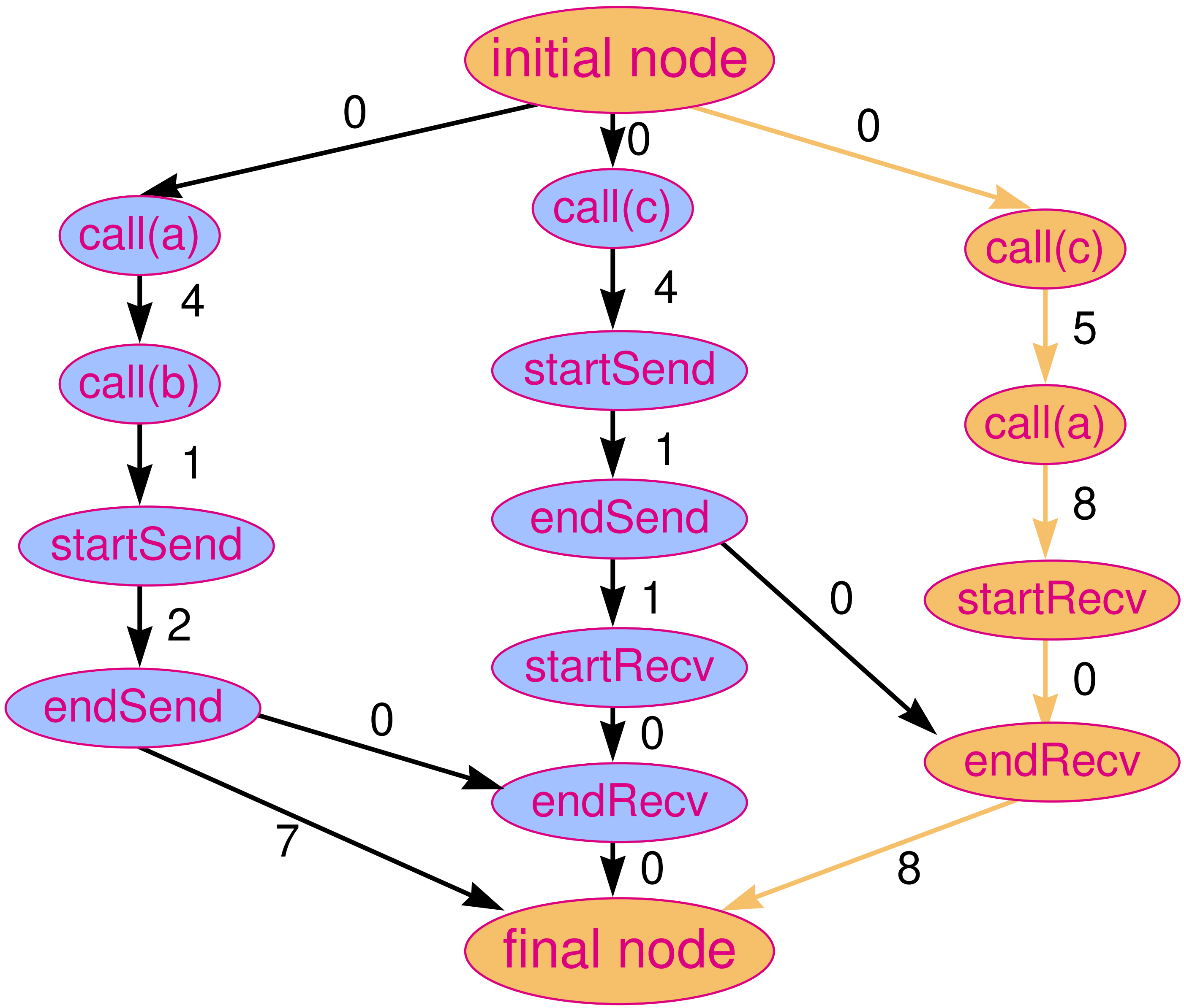
The Need for Parallel Metrics

- “Local” Profiling
  - Compute share of CPU time for each procedure
  - Add values from all processes
  - Sort by decreasing share of overall CPU time
- Limitations of local profiling
  - Largest CPU time does not imply more important
  - Need to understand relationships between processes

Critical Path Profiling

- Critical Path
  - Longest path through a parallel program
  - To speedup program, must reduce path
- Critical Path Profile
  - Time each procedure is on the critical path
- Useful, but requires
  - logging all events (**big**)
  - building a graph of events (**slow**)

Program Activity Graph



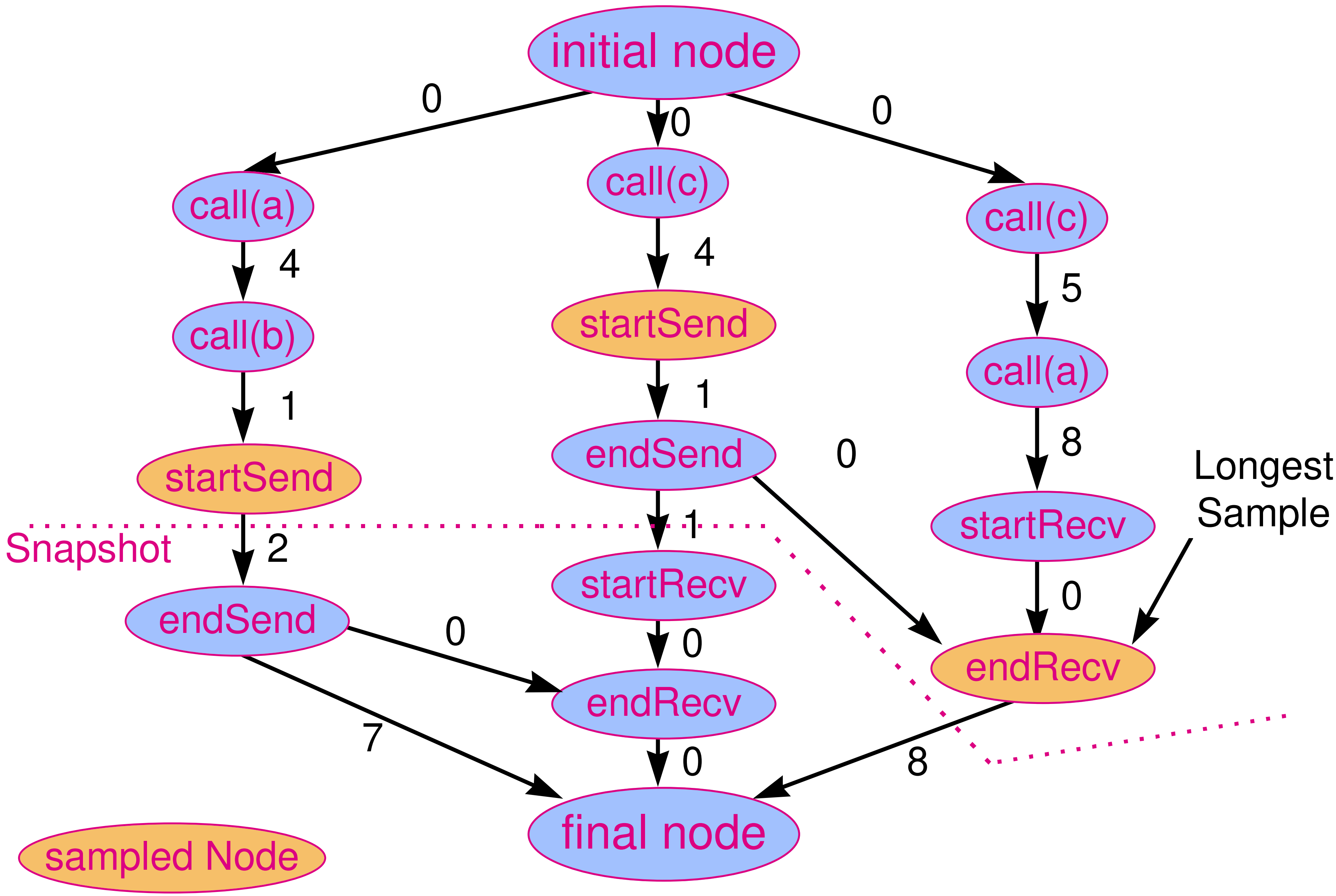
Online Approach to Critical Path

- Key Observations
  - PAG is implicitly walked by program execution.
  - Select a procedure to compute its share of the CP.
- Algorithm:
  - Maintain a <CP, share> tuple per process.
  - Computation (local events)
    - update tuple based on time.
  - Send
    - attach tuple onto message.
  - Receive
    - compare CP of received tuple to local one.
    - keep the tuple with the largest CP value.

Sampling Intermediate Values

- Compute CP for intermediate points to:
  - Study program phases.
  - Select different procedures during one execution.
- Idea:
  - Sample values from each process and combine.
- Problems:
  - Need a consistent snapshot of the CP.
    - define a cut set of the PAG.
  - Want the combined sample to be *timely.*
    - no earlier than the last sample from each process.

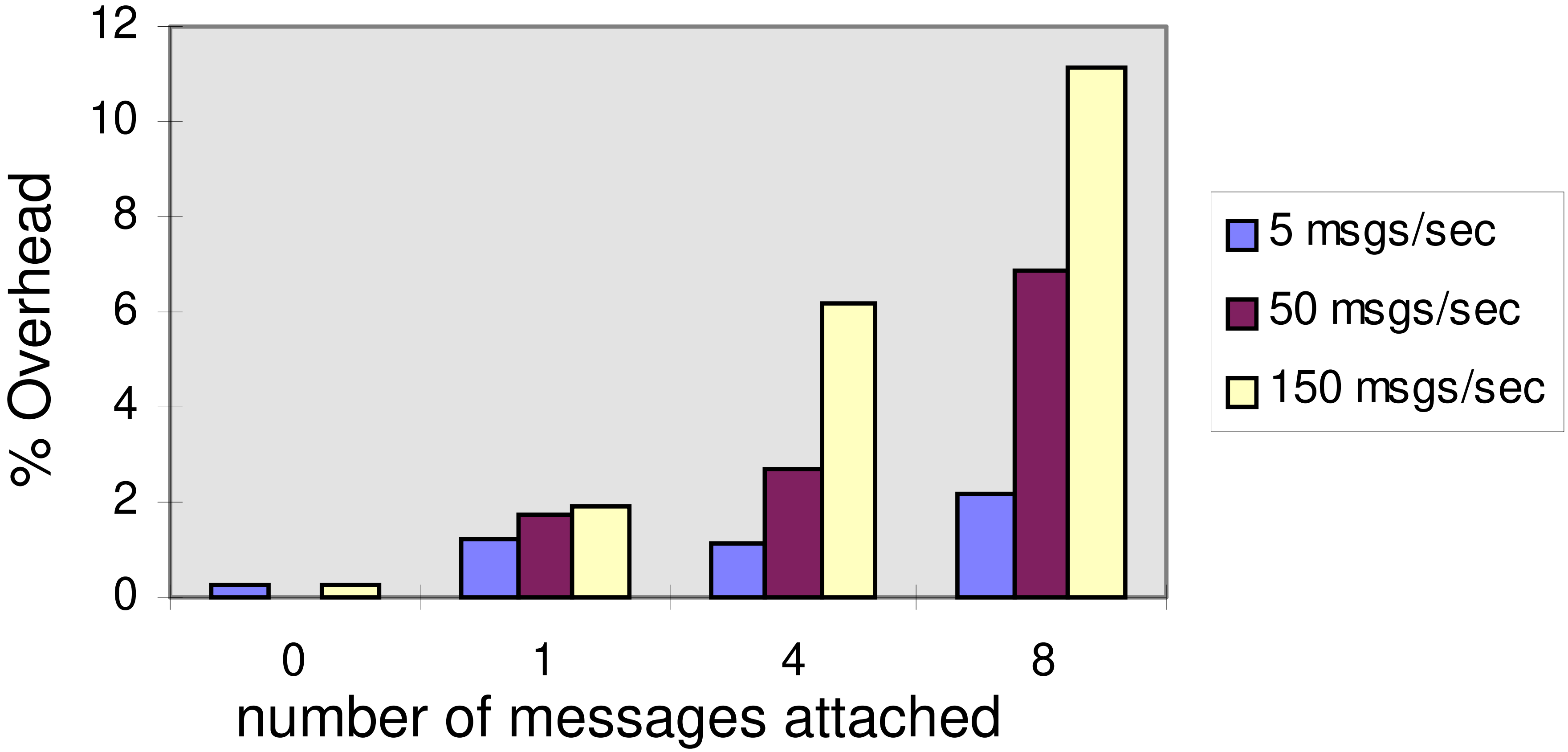
Program Activity Graph



Starting CP During Execution

- Need to start CP at a consistent point.
  - similar to Chandy & Lamport's snapshot algorithm.
  - tell each process to start computing the CP
- Issue: sender/receiver miss match
  - sender already computing CP, receiver not
    - receiver starts on receipt of message
  - receiver already computing CP, sender not
    - do nothing, send was before "snapshot"

Piggy-back Overhead



- Simple ping-pong style program
- Zero messages shows overhead of mechanism

NAS IS Application

Procedure	CP	% CP	CPU	% CPU
nas_is_ben	**12.4**	**56.4**	**54.8**	**74.1**
create_seq	**9.2**	**42.0**	**9.2**	**12.4**
do_rank	**0.4**	**1.6**	**9.2**	**12.5**

- create_seq is more important than CPU time indicates.
- do_rank is ranked higher than create_seq by CPU time.

Critical Path Zeroing

- CP is useful but
  - sub-critical paths can limit improvement possible.
  - selected procedure can occur on other paths.
- Solution: CP Zeroing
  - compute the CP as if the a procedure's time is 0.
  - use a variation of online CP algorithm
    - CP_{net} = CP - Share
    - at receive, keep tuple with largest CP_{net}

Ocean Application

Component	CP Zero	% CP Zero	CP	%CP	CPU	% CPU
s_recv	**7.3**	**18.4**	**9.7**	**24.5**	**36.6**	**29.8**
step	**5.5**	**13.9**	**8.5**	**21.3**	**23.9**	**19.4**
s_send	**4.2**	**10.6**	**7.1**	**17.9**	**21.4**	**17.4**
tracer	**1.7**	**4.3**	**1.9**	**4.8**	**5.3**	**4.3**
clinc	**1.5**	**3.9**	**1.5**	**3.9**	**5.8**	**4.7**

– s_recv is less important than CP or CPU predicted

Conclusion

- Online Critical Path is useful
  - metric provides helpful information
  - overhead of computation is reasonable
  - can compute intermediate values during computation
- Critical path Zeroing is effective
  - provides "what-if" style computation
  - one online pass vs. three off-line passes in previous algorithms