Parallel Metropolis-Hastings

Boyan Bejanov
bbejanov@bankofcanada.ca
Data and Statistics Office, Bank of Canada

COMP5704
School of Computer Science, Carleton University

April 2014

Motivation

- Economists at BoC use a large DSGE model to simulate the Canadian economy
  - 350+ equations, 300+ variables, 150+ parameters
  - The solution is in the form

$$X_t = T(\theta)X_{t-1} + R(\theta)\varepsilon_t$$

$$Y_t = d(\theta) + ZX_t + v_t$$

- The likelihood, $P(Y_t|\theta)$, is evaluated by Kalman filter

Motivation

- Point estimates for θ are obtained by maximum likelihood

- For confidence intervals we use Bayesian inference:

$$P(\theta|Y_t) \propto P(Y_t|\theta)P(\theta)$$

- We need a sample from $P(\theta|Y_t)$

- Sampling in this context is done by Metropolis-Hastings algorithm

Outline

- Brief introduction to Markov Chains
- Metropolis-Hastings Algorithm
- Overview of parallelization approaches
- Prefetching algorithm
- Analysis
- Implementation details
- Computational experiments
- Conclusions

Markov Chains - definition

- Markov chain is an infinite sequence of random variables: $X_0, X_1, X_2, \ldots$
- Interpretation: X_t is the state of the system at time t.

- ***Markovian property***: the transition rule depends only on the current state

$$P(X_{t+1}|X_t, \ldots, X_0) = P(X_{t+1}|X_t)$$

- The function $p(x, y) = P(X_{t+1} = y|X_t = x)$ is called ***transition kernel***.

Markov Chains - properties

- ***Stationary distribution***

$$\pi(x) = \int p(y,x)\pi(y)dy$$

- The time it takes to converge is called ***burn-in***
  - depends on both $p(x,y)$ and the initial state, X_0

Metropolis-Hastings Algorithm

- Markov Chain Monte Carlo (MCMC)
  - the limiting distribution, $\pi(x)$, is known

    here we call $\pi(x)$ the ***target density***
  - find a transition kernel, $p(x, y)$

- Metropolis-Hastings Algorithm is a recipe for building a transition kernel for given target density.

Metropolis-Hastings Algorithm

- Proposal kernel, $q(x, y)$
  - when the chain is at state $X_t = x$, the proposal generates a candidate $Y \sim q(x, \cdot)$

- The candidate is accepted with probability

$$\alpha = \min\left\{\frac{\pi(y)q(y, x)}{\pi(x)q(x, y)}, 1\right\}$$

Metropolis-Hastings Algorithm

Input: $\pi(x)$, X_0, n, $q(x, y)$

for $k = 0$ **to** $n - 1$

generate $Y \sim q(X_k, \cdot)$

compute $\alpha = \frac{\pi(Y) q(Y, X_k)}{\pi(X_k) q(X_k, Y)}$

generate $U \sim uniform(0,1)$

if $U < \alpha$ **then** set $X_{k+1} = Y$ (accept)

else set $X_{k+1} = X_k$ (reject)

next k

Assumptions and Goals

- The target distribution is high-dimensional (30+)
  - Long burn-in time

- Computation of $\pi(x)$ is the most expensive operation

- We want a general algorithm, not one specific to a narrow class of target distributions.

Parallel M-H approaches

- An ***independence*** proposal does not depend on the current state, i.e. $q(x, y) = q(y)$
  - Generate n proposals, $Y_1, \dots, Y_n$, and pre-compute $\pi(Y_1), \dots, \pi(Y_n)$ on p processors in n/p time.
  - Gather these values on processor 0 and run the chain sequentially.

- A good $q(y)$ is problem specific, or
- $q(y)$ is estimated from the chain – ***adaptive M-H***

Parallel M-H approaches

- ***Regeneration time*** is a time when the state is independent of the history of the chain up to that time.
  - ***Independent tours*** between regeneration times
  - Run p tours in parallel, then stitch them together to form a single long chain

- Identification of regeneration times is very difficult
- The expected length of a tour increases dramatically with the dimension of $\pi(x)$

Parallel M-H approaches

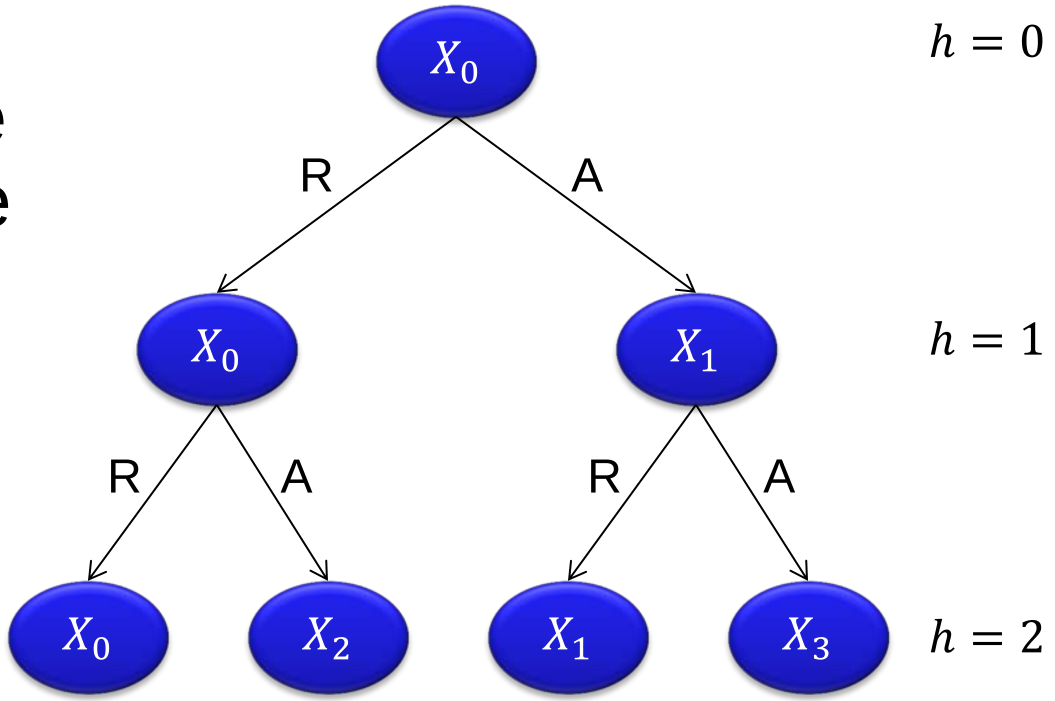
- ***Prefetching***
  - Generate proposals for all possible paths h steps into the future
  - Compute the target densities in parallel
  - Run h steps of M-H sequentially

- A.E.Brockwell (2006)
- J.Byrd *et.al.* (2008), Strid (2010)

Prefetching

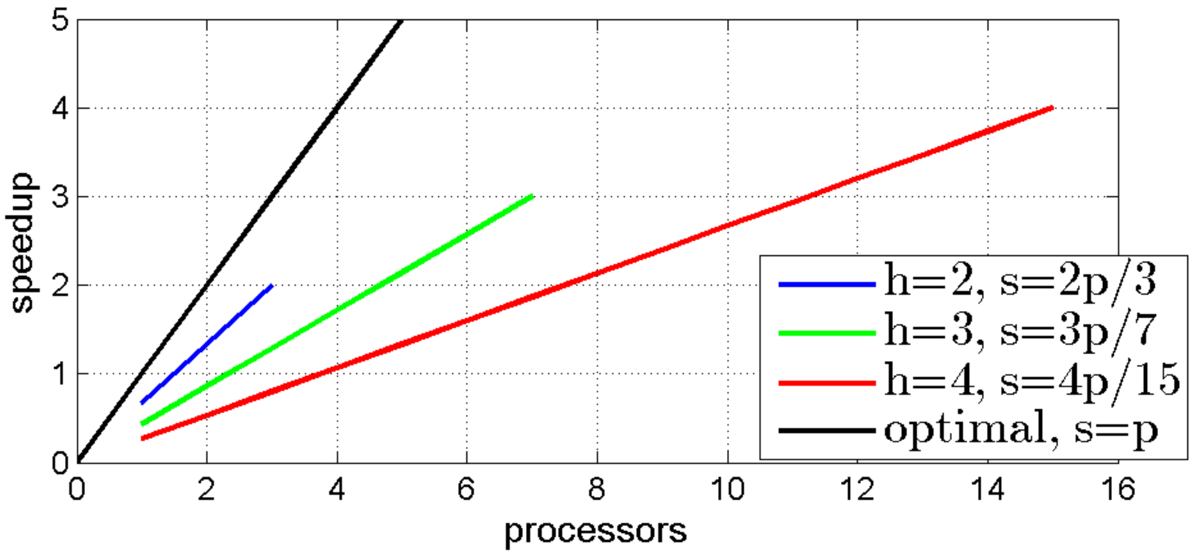
- At each of the h steps we either accept or reject the proposed candidate

- There are 2^h possible paths from the root to a leaf



Complexity Math

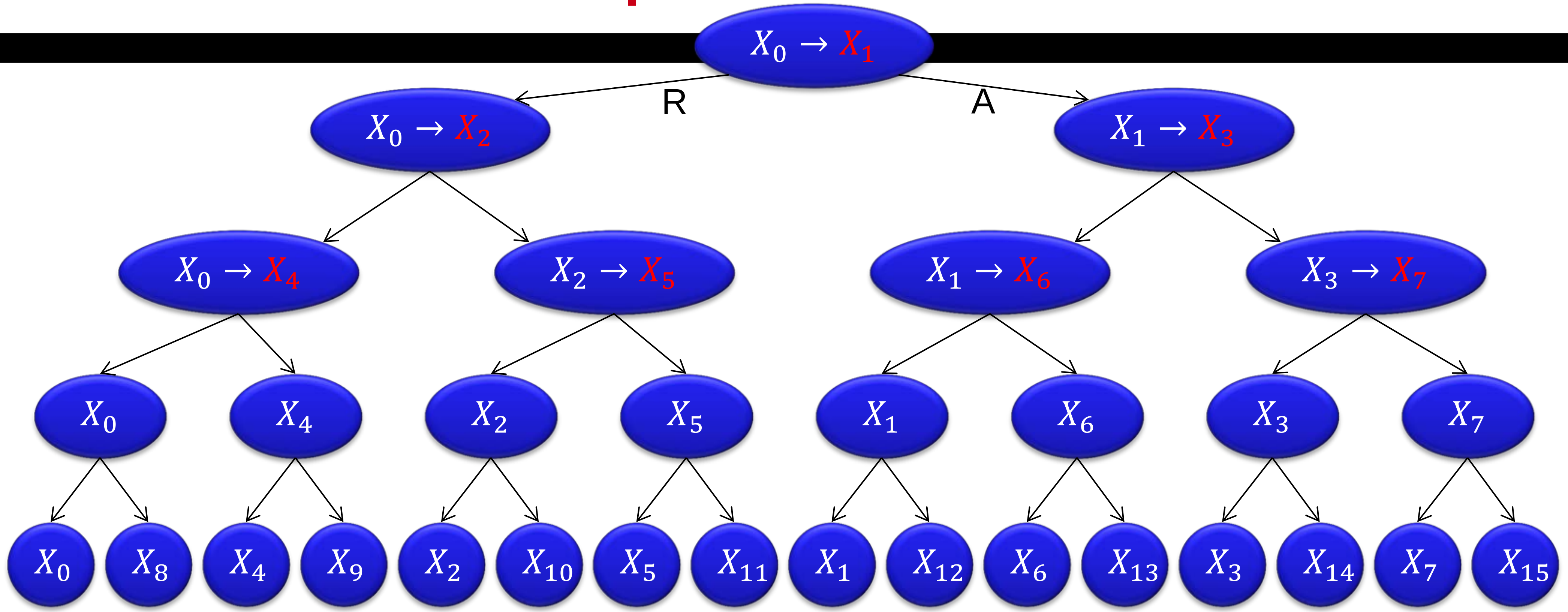
- $n = h$ (the problem size)
- $T_s = n$
- $p = 2^n - 1$: $T(p) = 1$, $s(p) = \frac{T_s}{T(p)} = n, e(p) = \frac{s(p)}{p} = O(n2^{-n})$
  - not optimal, not even efficient, not NC
- For a general p: $T(n,p) = \frac{2^n - 1}{p}, s(n,p) = \frac{np}{2^n - 1}$



P-Completeness

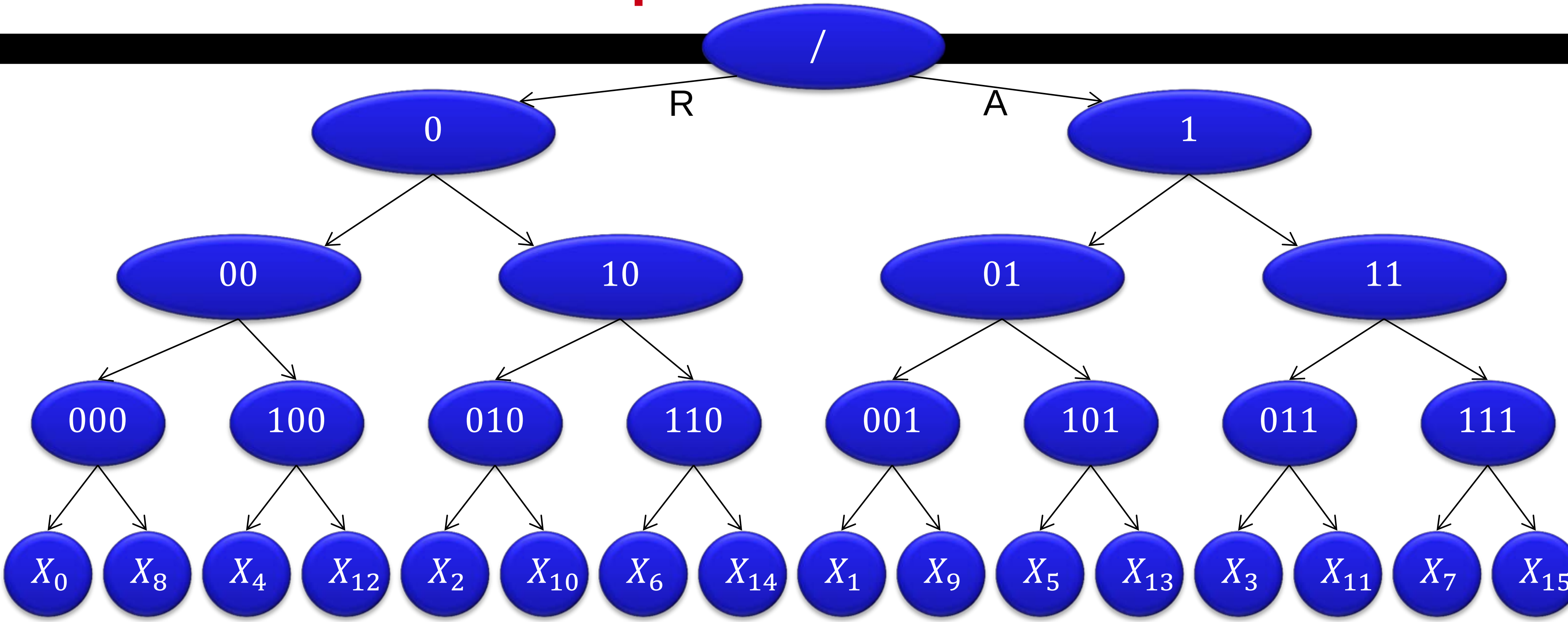
- Given an instance of generability problem
  - have a set X, with $|X| = n$, a subset $T \subseteq X$, an element $x \in X$ and a binary operation $*$
  - Is x in the closure of T under $*$?
- Construct a Markov chain that solves it
  - WLOG assume $X = \{0, 1, \ldots, n-1\}$
  - state space: $2^{\mathrm{X}} = \{0, \ldots, 2^n - 1\}$
  - initial state: $t = \sum_{i \in T} 2^i$
  - transition rule: when current state is $a = \sum_{i \in A} 2^i$
    - propose candidate b by selecting at random(?) one of the 0 bits in a and setting it to 1, i.e. $b = a + 2^k$ for some k.
    - if $k = i * j$ for some $i, j \in A$, then accept b, otherwise reject it

Implementation Remarks



- Each leaf uniquely determines the path. From the index, we should be able to determine:
  - At which step was X_k first proposed?
  - What was the current point when X_k was proposed?
  - What other points are needed in order to reach X_k at the bottom?

Implementation Remarks



- Step s sets bit in position s, counted from right to left
- Rejection sets the bit to 0, acceptance sets it to 1
- Step where X_k was proposed: $s(k) = 1 + \lfloor \log k \rfloor$, $k > 0$, $s(0) = 0$
- X_k was proposed from: $c(k) = k - 2^{s(k)-1}$

Implementation Remarks

k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$s(k)$	/	1	2		3				4							
$c(k)$		0	0	1	0	1	2	3	0	1	2	3	4	5	6	7

- To build the tree:

```
for(c=0; c<$2^{h-1}$; ++c) {
   for(s=s(c); s<h; ++s) {
      p = c+$2^{s}$
      generate $X_p \sim q(X_c | \cdot)$
   }
}
```

- The M-H algorithm

```
c=0;
for(s=1; s<h; ++s) {
   p = c + $2^{s-1}$;
   if (accept $X_p$) c = p;
   State[s] = $X_c$;
}
```

Implementation Remarks

- To compute the target density in parallel:
- OpenMP

```
#pragma omp parallel for
```

for$(k = 1;\ k < 2^h;\ ++k)$ *compute* $\pi(X_k)$

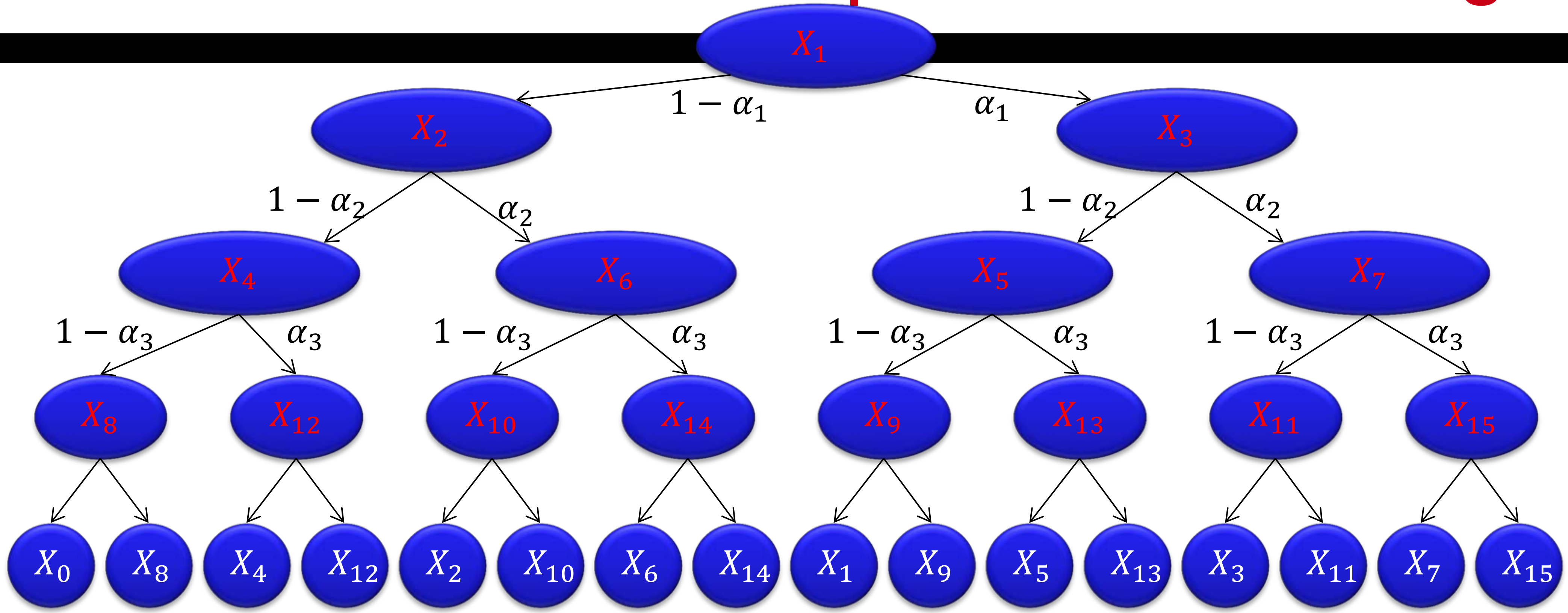
- Cilk

```
cilk void prefetch_target(int c, int s) {
    if(s==h) {
        if (c>0) compute π(X_c);
    } else {
        spawn prefetch_target(c, s+1);
        spawn prefetch_target(c+2^s, s+1);
        sync;
}   }
```

Incomplete Prefetching

- Prefetch only the most likely paths
- There is no guarantee that the chain will make h steps.
- Maximize the expected depth for p evaluations of $\pi(x)$.

Incomplete Prefetching

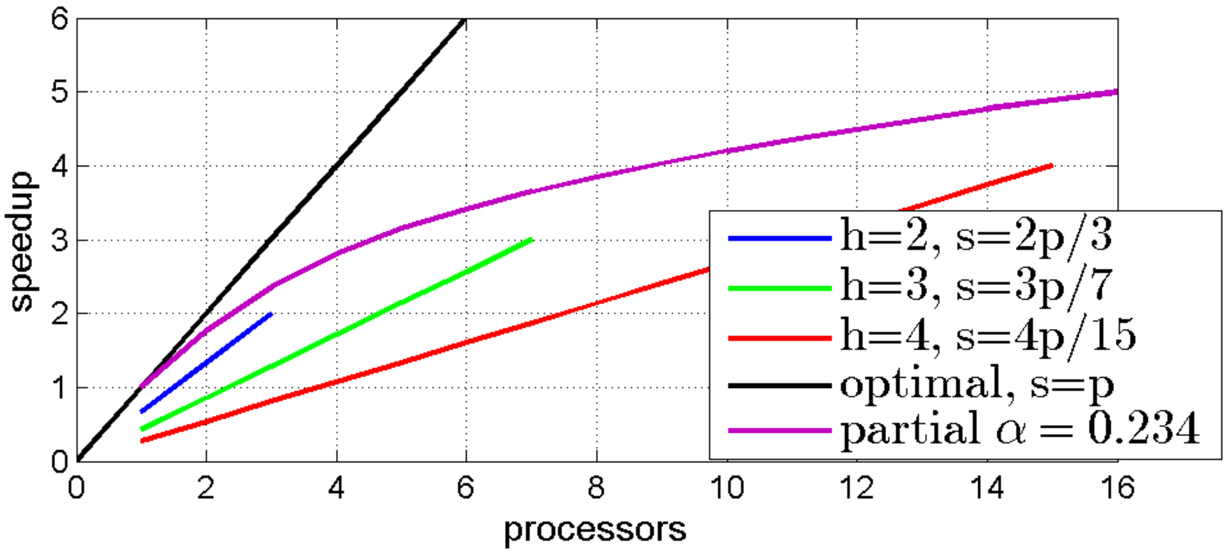


- The contribution of given node is the probability it would be reached
- The probability to reach a node is product of all weights from the root to the node where it is first proposed
- All proposals along the path must be prefetched
- The two children of p are $a = p + 2^{s(p)}$ and $r = a - 2^{s(p)-1}$

Incomplete Prefetching

p	$D(p)$
1	1
2	1.77
3	2.35
4	2.80
5	3.15
6	3.41
7	3.64
8	3.84
9	4.03
10	4.20
11	4.36
12	4.49
13	4.63
14	4.77
15	4.88

- Complexity: example with $\alpha = 0.234$
- $T(p) = n / D(p)$, $s(p) = D(p)$



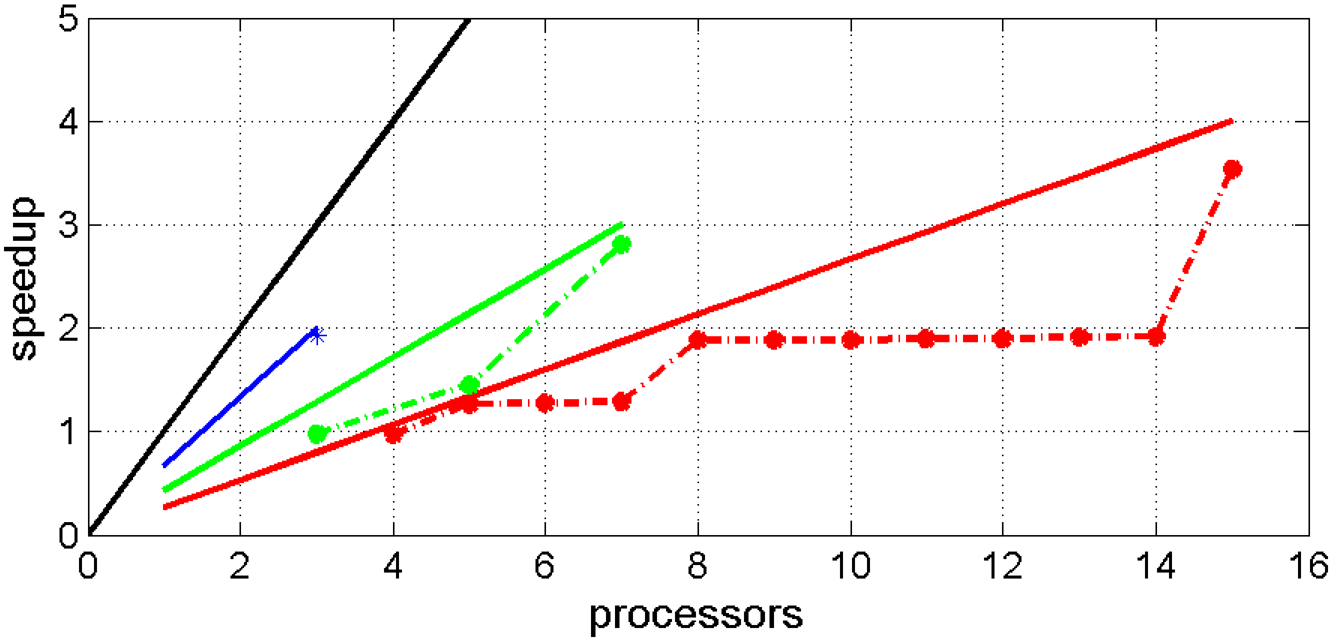
- these are expected rates, your mileage may vary

Run Times Full Prefetching

- Intel Xeon E5-2690 8 cores @ 2.90 GHz, dual socket (16 core total)
- $n = 1200$, $\pi(x)$ is 15 dimensional with added work for delay

h	p	#threads	parallel method	total time	average time per step
	baseline		seq	120.06	0.1017
2	3	3	omp	61.65	0.0514
2	3	3	cilk	62.06	0.0517
3	3	3	cilk	122.66	0.1022
3	5	5	cilk	82.81	0.0690
3	7	7	cilk	42.70	0.0356
3	7	7	omp	42.72	0.0356
4	15	15	omp	34.55	0.0288
4	15	15	cilk	33.93	0.0283
5	16	31	cilk	73.72	0.0614
5	16	16	cilk	67.08	0.0559

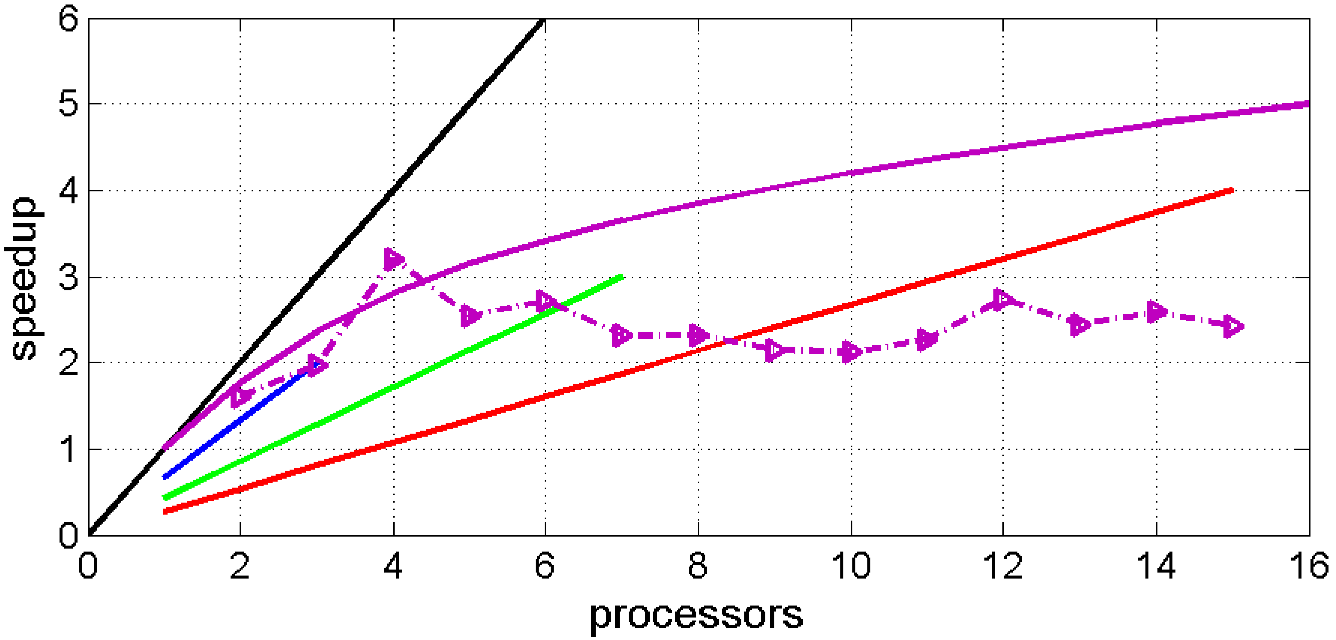
Run Times Full Prefetching



Run Times Partial Prefetching

p	D	h	d	total time cilk	total time omp	tree
2	1.766	2	1.66297	74.5942	74.7524	1 2
3	2.35276	3	2.06724	60.8497	60.6849	1 2 4
4	2.80221	4	3.36798	37.5026	37.1227	1 2 4 8
5	3.14649	5	2.71655	47.3608	46.9558	1 2 4 8 16
6	3.41021	6	2.90511	44.1376	43.7954	1 2 4 8 16 32
7	3.64421	6	3.54142	51.9093	58.3732	1 2 3 4 8 16 32
8	3.84622	7	3.45533	51.7107	60.1928	1 2 3 4 8 16 32 64
9	4.02547	7	3.44669	55.8634	59.8548	1 2 3 4 5 8 16 32 64
10	4.20471	7	3.82428	56.6887	54.4586	1 2 3 4 5 6 8 16 32 64
11	4.35945	8	3.85209	52.9827	54.1078	1 2 3 4 5 6 8 16 32 64 128
12	4.49675	8	4.50000	43.9406	46.5858	1 2 3 4 5 6 8 12 16 32 64 128
13	4.63405	8	4.13495	49.1822	50.4384	1 2 3 4 5 6 8 9 12 16 32 64 128
14	4.77135	8	4.37226	46.3788	47.8808	1 2 3 4 5 6 8 9 10 12 16 32 64 128
15	4.88988	9	4.16725	49.4775	49.9524	

Run Times Partial Prefetching



Conclusion

- Prefetching is a viable parallel algorithm for multicore and small h
- Partial prefetching seems promising, although the static acceptance probabilities don't seem to work well
- Next steps
  - integrate with Matlab and R
  - combine with adaptive IMH

Random Number Generators