Scalability of the Parallel Pollard-Rho Algorithm

Faisal Nawab
nawab@cs.ucsb.edu

The discrete logarithm problem

- The Pollard-Rho algorithm [1] is one of the best generic algorithms for the discrete logarithm problem
- Variants of the algorithm apply to the integer factorization problem
- Various cryptography techniques rely on the difficulty of the discrete logarithm and factorization problems
- Studying the efficiency of algorithms to solve them is important to establish the integrity of these cryptography techniques

The Pollard-Rho method

- The discrete logarithm problem is the problem of finding the value x that satisfies the equation

$$y = g^x (mod\ p)$$

  where y, g, and p are given
- The Pollard-Rho method has $\sqrt{\frac{\pi . p}{2}}$ computational complexity [1]
- The algorithm searches for a cycle in a sequence in the group (**search factor**)
- The cycle is deterministic: each step in the sequence depends on its predecessor (**determinism factor**)

Challenges of the Pollard-Rho method

- The **search factor** means that we need to search for occurrences of elements to find repetitions
- The search might imply the need for storing elements and continuously searching in the stored elements
- With large group sizes this might lead to degraded performance as the number of stored elements becomes high
- The **determinism factor** means that we cannot compute multiple elements of the same sequence in parallel
- This makes the algorithm serial in nature, which makes taking advantage of parallelism challenging
- This project studies variants of the Pollard-Rho method and compares their **space efficience** and **scalability**

Pollard-Rho algorithm background

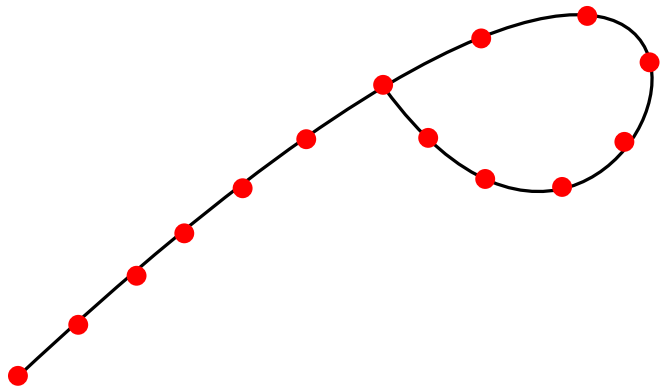
- A sequence is defined, where an element a_{i+a} is equal to:

$$\begin{array}{c} y \ . \ a_i \quad for \ a_i \in S_0 \\ a_i^2 \quad for \ a_i \in S_1 \\ g \ . \ a_i \quad for \ a_i \in S_2 \end{array}$$

  Where S_i's are disjoint partitions
- A random α is chosen for the initial element so that $a_0 = g^{\alpha}$
- Equality of elements in the sequence implies equality of the exponents mod (p-1)
- This equality can be used to solve for x

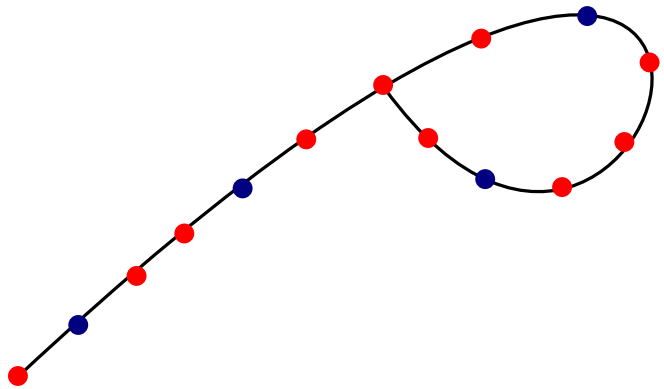
A straight-forward implementation (ALL)

- A straight-forward implementation of the Pollard-Rho algorithm is to compute elements in the sequence and stop when the first re-occurrence is detected
- To find re-occurrences, all computed elements are stored
- Sorting or indexing can be used to detect re-occurrences efficiently
- Space needed by the algorithm is in the order of $\sqrt{\frac{\pi . p}{2}}$
- We call this variant ALL, because it considers all computed elements
- *Is there a trade-off between efficiency and space?*



The distinguished points optimization (DIST)

- Manage the trade-off between efficiency and space
- Main idea: only store and search for elements with a certain distinguishing property [2]
- For example, only store elements with values that are multiples of a number D
- In this way, space required becomes in the order of $\frac{\sqrt{\frac{\pi . p}{2}}}{D}$
- We call this variant DIST(D'), where D' is the percentage of distinguished points to all points
- We can control the trade-off of efficiency and space by controlling D'
- *How does the trade-off look like?*

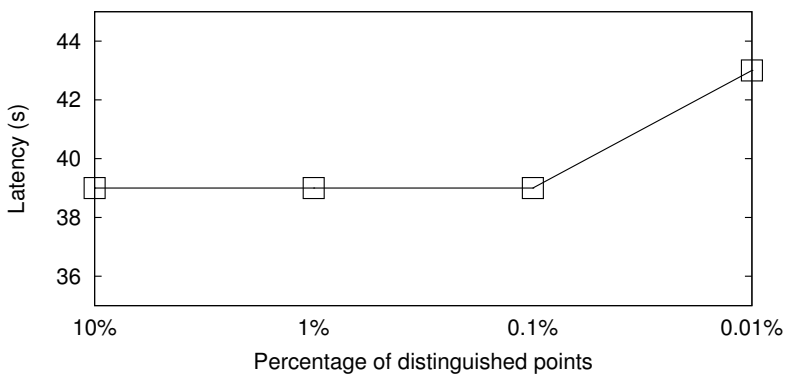


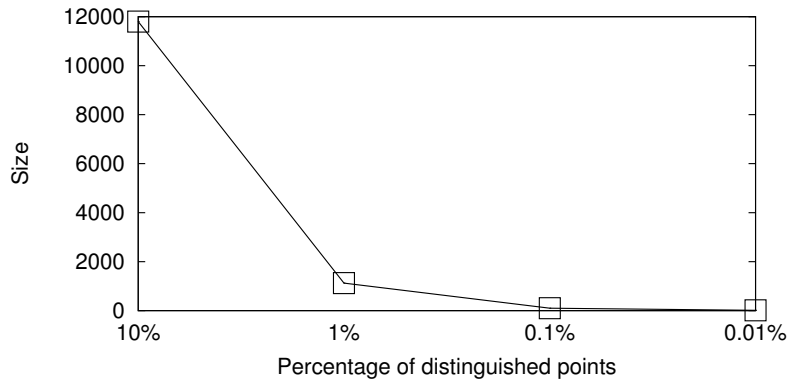
Distinguished points space-efficiency trade-off

- The trade-off is shown in the expected number of computations needed to find a cycle:

$$\sqrt{\frac{\pi.p}{2}} + \frac{1}{\theta}$$

  where θ is the proportion of points satisfying the distinguishing property [2]
- The trade-off is quantified for a group size 5915587277
- We measure the efficiency as the time it takes to find x (latency)
- We measure space as the number of elements in storage



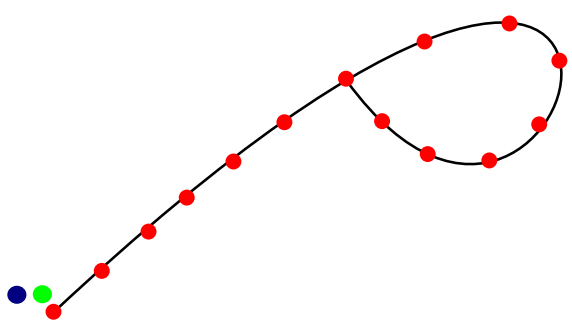


Caveats in DIST performance results

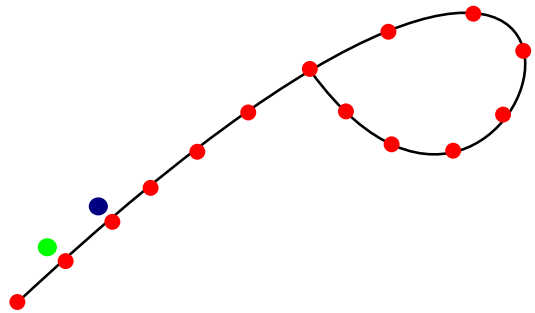
- The group size used in the previous experiments is much smaller than what is used in cryptography
- Although in our case 0.01% of points leads to a very small storage (less than 10 objects), with realistic group sizes, this percentage must be much smaller to have a significant effect on space saving
- It can be speculated that the trade-off trend we observe is going to be similar to "real" larger group sizes
- Thus, if the theoretical bound continuous to hold, the cost of saving space with distinguished points will be proportional to $\frac{1}{\theta}$ [2]
- However, what is concluded from the previous experiments is a byproduct of both the group size and size of distinguished points – further experiments with larger sizes are needed to confirm the effect of distinguished points

The cycle variant (CYCLE)

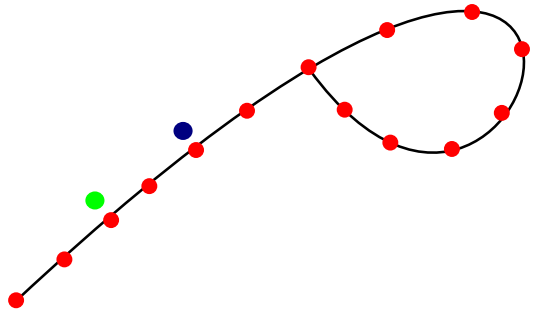
- *Is there a way to detect re-occurrences without maintaining any storage?*
- Re-occurrences can be detected by maintaining two iterators moving at a different speed.
- This is called the Floyd's cycle-finding problem [3]
- Call this variant CYCLE



step 1



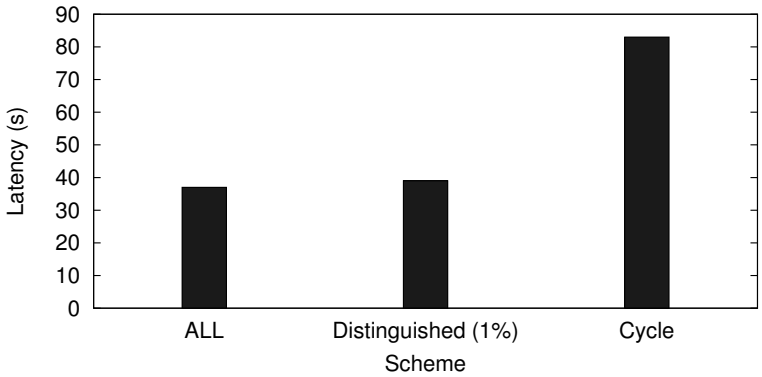
step 2

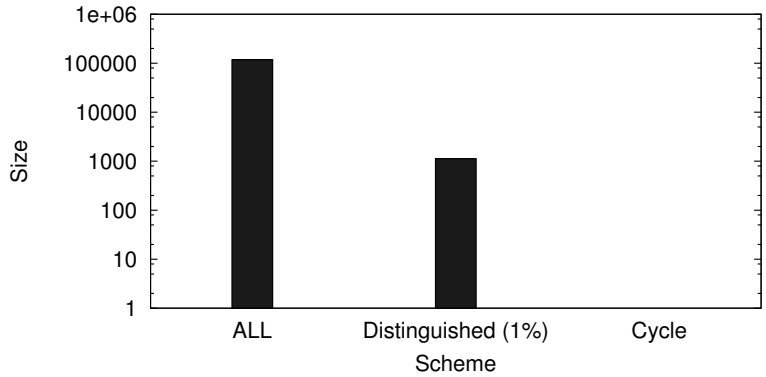


step 3

Performance of Pollard-Rho variants

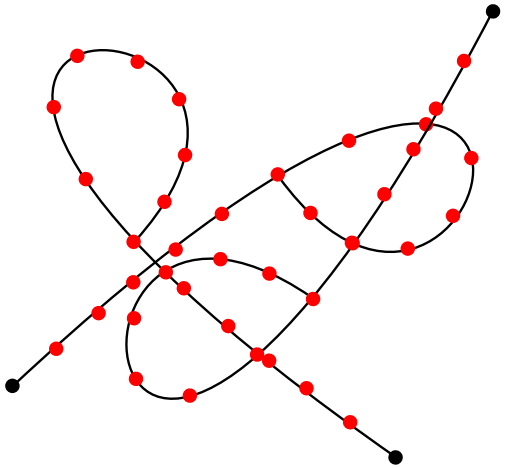
- Figures below measure latency and size of the Pollard-Rho different variants
- DIST(1%) performs closely to ALL with good space savings
- CYCLE takes more than 2x the time to find a cycle than DIST(1%) and ALL

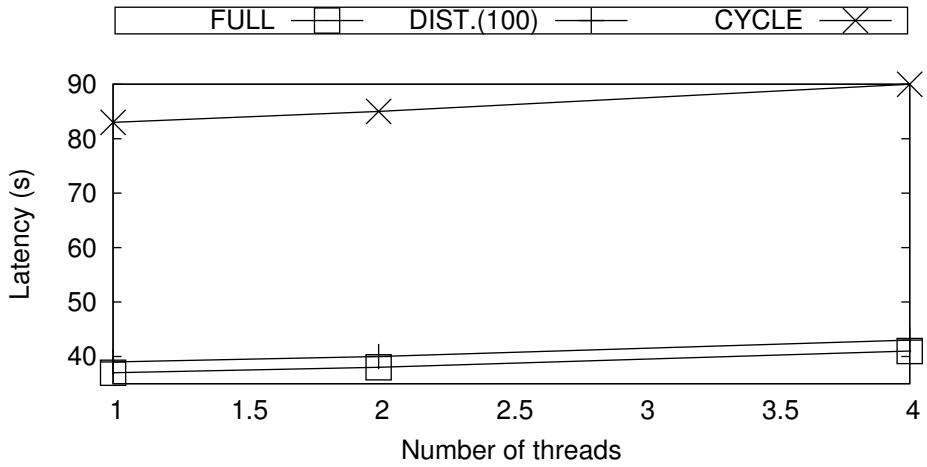




Parallelizing the Pollard-Rho algorithm

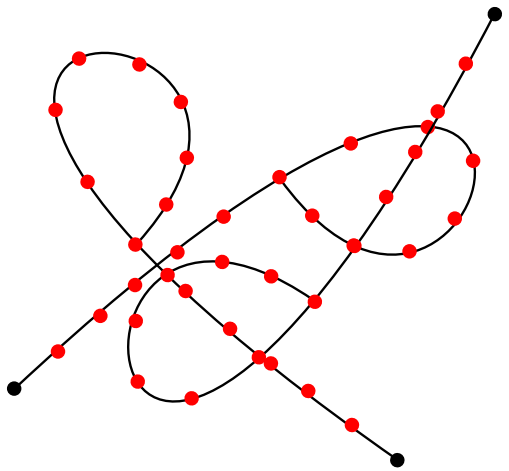
- A straight-forward way of parallelizing could be to run different instances of each algorithm concurrently with different initializations
- Unfortunately, this does not lead to better performance [4]
- The reason is that the expected time to find a cycle is the same from any point





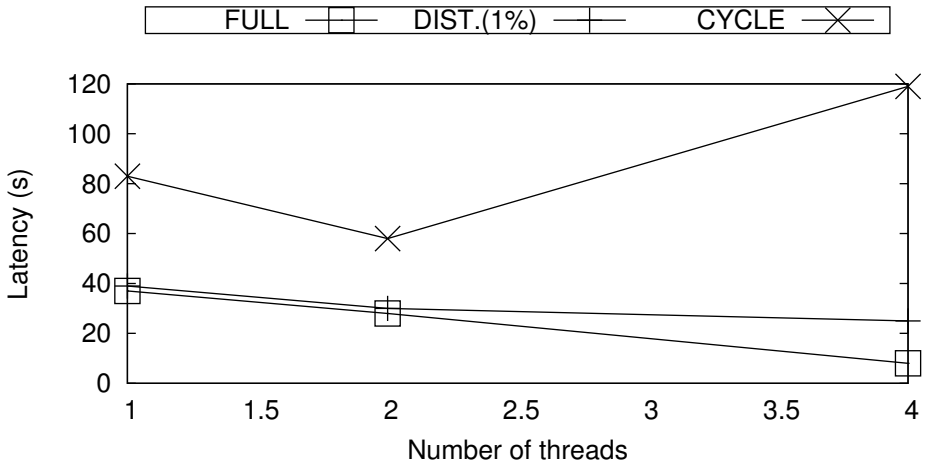
Parallelizing the Pollard-Rho algorithm

- To leverage parallelizm, different instances need to cooperate
- Each instance runs with a different sequence and initialization
- All instances share one storage and look for re-occurences amongst all sequences (in ALL and DIST)
- CYCLE makes each instance responsible for one speed of iteration and checks in each step if there is a re-occurrence



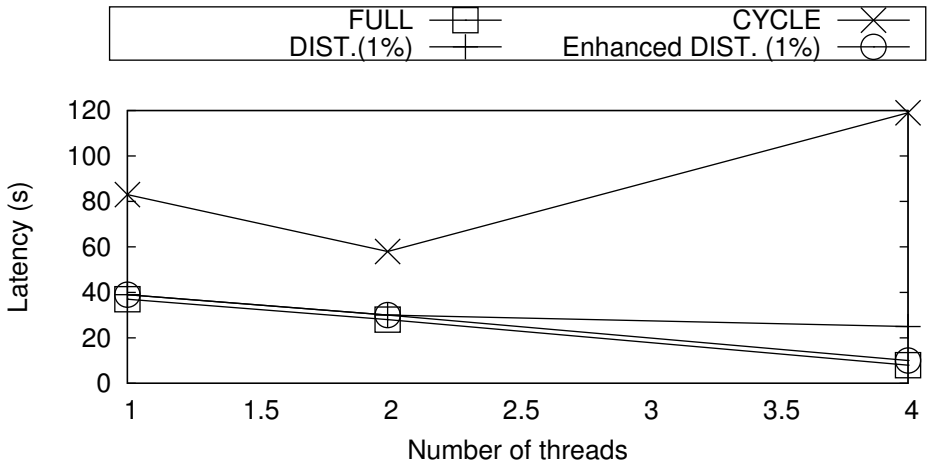
Parallelizing the Pollard-Rho algorithm

- Speedup of ALL is close to a linear speedup
- Speedup of DIST(1%) is close to $\sqrt{m}$, where m is the number of processors [2]
- CYCLE does not experience any speedup – with larger numbers of threads blocking is affected by the weakest link



Enhancing the parallel DIST

- A proposed enhancement of parallel DIST to improve its performance (called Enhanced DIST) [2]
- Main idea: when a distinguished point is found, change to a new sequence
- Enhanced DIST scales linearly similarly to ALL



On scalability and observed latency

- The theoretical studies about scalability [2] are based on the number of iterations needed to find a re-occurrence
- For example, a linear scale-up means that with n agents, we expect to find a re-occurrence in $\frac{k}{n}$ iterations, where k is the number of iterations expected to find a re-occurrence with one agent
- Reminder: The observed latency is the time until a re-occurrence is found
- Agents collectively process iterations, thus the number of already scaled iterations is divided among participating agents
- Thus, latency to finding a re-occurrence scales with a factor n of the number of iterations, where n is the number of agents

Conclusions

- DIST offers good space savings without significantly affecting efficiency and provide the ability to control the trade-off between space and efficiency
- CYCLE's blocking nature limits scalability – asynchronous implementations might overcome this limit
- ALL has good scalability, but as the storage gets larger, scalability might be affected
- DIST can be enhanced to reach similar scalability to ALL without the same storage cost by the optimization of re-initialization when hitting a distinguished point


Pollard, John M. "Monte Carlo methods for index computation (mod\ p)." Mathematics of computation 32.143 (1978): 918-924.


Van Oorschot, Paul C., and Michael J. Wiener. "Parallel collision search with crypt-analytic applications." Journal of cryptology 12.1 (1999): 1-28.


Knuth, Donald E. (1969), The Art of Computer Programming, vol. II: Seminumerical Algorithms, Addison-Wesley, p. 7, exercises 6 and 7


Brent, Richard P. "Parallel algorithms for integer factorisation." Number theory and cryptography 154 (1990): 26-37.