MPI Implementation of AKS Algorithm

Jayasimha T
Supercomputer Education and Research Centre
Indian Institute of Science, Bangalore, India
sercjayasimha@ssl.serc.iisc.in

***Abstract*—AKS algorithm is the first deterministic polynomial time algorithm for primality proving. This project uses MPI (Message Passing Interface) along with GNU MP (Multi Precision) library to implement a variant of AKS algorithm. Two different parallelization strategies have been implemented and we compare these strategies making relevant observations and providing speedups. The method of fast polynomial exponentiation with precomputation has been used to achieve parallelization and improve run time. Further we show experimentally that master slave model of parallelization gives very good results.**

I. Introduction

Prime numbers are of fundamental importance to mathematics in general and number theory in particular. Many cryptographic protocols use large prime numbers for encryption and decryption. Hence it is very important to quickly determine whether a number is prime or not.

Efficient randomized algorithms exist to determine whether a number is prime or not with a certain probability. In cases where certainty of primality is required randomized algorithms cannot be used. In spite of the seemingly simple problem definition, algorithms before AKS relied on some unproven results (extended Riemann hypothesis) or conjectures in number theory to deterministically prove primality in polynomial time. Three Indian researchers Manindra Agrawal, Neeraj Kayal and Nitin Saxena constructed a "polynomial time deterministic primality test", a much sought-after but elusive goal of researchers in the algorithmic number theory world. Most shocking was the simplicity and originality of their test, whereas the "experts" had made complicated modifications on existing tests to gain improvements.

In this project we give a parallel implementation of a variant of AKS algorithm using MPI and GMP (GNU MP) library.AKS algorithm is highly amenable to parallelization. We give two parallelization approaches Viz. Master slave model and parallelization based on precomputation.We compare these two algorithms based on their run times and speedups. The parallelization provides very good speedups and the speedups scale with number of processors, hence there is a great improvement in run time by using the parallel version of AKS. We show experimentally that our implementation performs better than the existing parallel implementation.

This implementation is unique in the sense that it

- Uses precomputed values to speedup exponentiation.
- Uses precomputed values to achieve parallelization.
- Uses faster variant of AKS algorithm.

The remainder of this report is structured as follows ,Section II introduces the work related to this project, Section III deals various algorithms specifically III-C deals with the parallelization strategies and III-D deals with the various implementation issues,Section IV deals with results that we have obtained in this project.We conclude the report in Section V .

II. Related Work

The AKS algorithm was proposed in the paper by Manindra Agrawal et al. [1], this was first deterministic polynomial time primality proving test. The paper by DJ Bernstein [2] improved the original AKS algorithm by reducing the number of iterations required to test the primality. The paper by R. Crandall and J. Papadopoulos [3] discusses various issues regarding the serial implementation of AKS algorithm and optimizations that can be carried out, like using FFT for quickly multiplying large integers etc, these optimizations have been included in the GMP library, this library was used for the serial implementation of AKS by Chris Rotella [4] .A simple parallel implementation with master slave model was done as a part of the master thesis by JS Bronder [5].The paper by Gordon et al. [6] surveys various techniques available for fast integer exponentiation, these techniques achieve speedup at the cost increased memory.

III. Methodology

A. AKS Algorithm

The AKS algorithm[Algorithm 1]relies fundamentally on the following theorem

Theorem 1. *For given integer $n \geq 2$, let r be a positive integer $< n$, for which n has order $> (logn)^2 modulo(r)$. Then n is prime if and only if*

- *n is not a perfect power,*
- *n does not have any prime factor $\leq r$,*
- *$(x+a)^m \equiv x^m + a$ mod $(n, x^r - 1)$ for each integer $a, 1 \leq a \leq \sqrt{\phi(r)} log(n)$.*

where the symbols have usual meanings.

The complexity of AKS algorithm is $o(log(n))^{\frac{15}{2}}$ [1].

DJ Bernstein further improved the algorithm [Algorithm2] by defining a integer $s < \sqrt{\phi(r)} log(n)$ as the looping parameter thereby reducing the number of iterations of the painful loop.

B. Fast Exponentiation

The exponentiation problem is, given fixed number g and a variable exponent n, compute g^n.The goal is to compute g^n with small number of multiplications. Speeding up of such computations is possible by storing precomputed values of powers of g. One such precomputation method is to store the set

$$S = \{g^{2^i} | i = 1, 2, ..., log(n) - 1\}.$$

Let $v(n)$ be the number of non zeros in the binary representation of n. Then g^n may be computed in $v(n) - 1$ multiplications by multiplying together powers corresponding to nonzero digits in the binary representation of n. This reduces to the work of $log(n) - 1$ multiplications in the worst case and $\frac{log(n)}{2} - 1$ on the average case at a cost of storing $log(n)$ powers [7].

This Idea can be extended for the AKS algorithm by storing the coefficients of the polynomials for various values of a and r.

C. Parallel AKS Algorithm

1) Master Slave Approach: AKS algorithm can be parallelized easily using the master slave approach. The master process is responsible for checking the loop termination condition. The slave processes are responsible for exponentiating the polynomial to appropriate modulo and checking for congruency. Instead of synchronizing with the master processor for every iteration, slave processors can evaluate a fixed number of terms of the polynomial and then synchronize with the master.

To speedup polynomial exponentiation each slave processor owns a local copy of the precomputed values and fast exponentiation algorithm is used for exponentiation.

2) Parallelization based on precomputation: In this approach parallelization is achieved at the cost increased memory and precomputation.Let $n = a_b a_{b-1} ... a_0$ be the binary representation of n. These bits are divided among the processors as shown

$$\underbrace{a_b a_{b-1} .. a_{\frac{(p-1)b}{p}}}_{p}, \underbrace{a_{\frac{(p-1)}{p}-1} a_{b-1} .. a_{\frac{(p-2)b}{p}}}_{p-1}, ..., \underbrace{a_{\frac{b}{p}-1} a_1 ... a_0}_{1}$$

Each processor computes its exponent using precomputed values. These are combined repeatedly by multiplying them together in pairs to from g^n.This method is referred to as binary fan-in multiplication [7].By storing polynomial coefficients this method can be extended to AKS algorithm.

At each iteration we use binary fan-in multiplication algortihm. Polynomial are exponentiated across processors and congruence is checked on a single processor. Communication of the termination condition has to be done at each iteration.

D. Implementation

The AKS-Bernstein variant of the algorithm and parallelization by binary fan-in multiplication was implemented in MPI and GMP using parallelization techniques discussed in III-C-1. The time spent in the steps before the polynomial exponentiation, is negligible compared to the time spent in loop, hence these steps are done serially. GMP library provides an API for doing Big integer arithmetic [8], hence this implementation can handle arbitrarily large numbers subjected to the memory limitation of the machine. While communicating GMP variables using MPI, the communicating processes have to negotiate the size of the data that has to be transferred before the actual transfer of the GMP variables can take place, hence this introduces additional overhead for communication.

Algorithm 1 AKS Algorithm

Input n> 1
If($n = a^b$ for $a \epsilon \mathbb{N}$ and $b > 1$),output COMPOSITE
Find the smallest r such that $o_r(n) > log^2 n$.
If $1 < (a, n) < n$ for some $a \leq r$,output COMPOSITE
If $n \leq r$,output PRIME
For $a = 1$ to $\sqrt{\phi(r)} logn$ do
    If($(x + a)^n \neq x^n + a(mod(x^r - 1, n))$)
        output COMPOSITE;
Output PRIME

Algorithm 2 Bernstein's improvement

Input : n >1
$r = 3$
while $r < n$
    if $(gcd(n, r) \neq 1)$Output COMPOSITE
    if(r is prime)
        let q be the largest prime factor of r
        if($\binom{2q-1}{q} \geq 2^{2\sqrt{r} log(n)}$ and $n^{\frac{r-1}{q}} \leq 1$) exit loop.
    $r = r + 1$
set $s = 2\sqrt{r} log(n)$

IV. Experiments and Results

A. Experiment Setup

The speedups and run-times are compared for different number of polynomial terms evaluated at the slave processors. See Figure 1 on page 3,Figure 2 on page 3.

The comparison of run-times is made between the reference parallel implementation [5] and developed Master slave implementation for different number of processors. See Figure 3 on page 3.

The run-times of reference implementation, developed Master slave implementation and binary fan-in multiplication implementation are compared for different primes. See Figure 4 on page 3.

B. Results

With the increase in number of polynomial terms evaluated at the slave processor, the speedup obtained decreases and the run time increases, this is due additional polynomial evaluations that has to be carried out before communicating the result to the master.

The Master slave approach (AKS-Bernstein) with precomputation implemented in this project is faster than both the reference (AKS) parallel implementation and binary fan-in multiplication with precomputation .

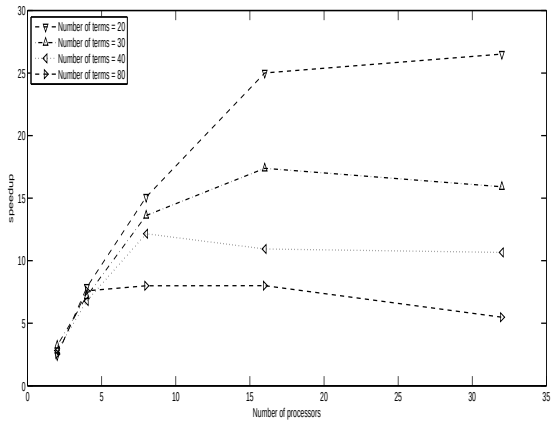

Figure 1. Comparison of speedups obtained for different number of polynomial terms evaluated at slave processor.

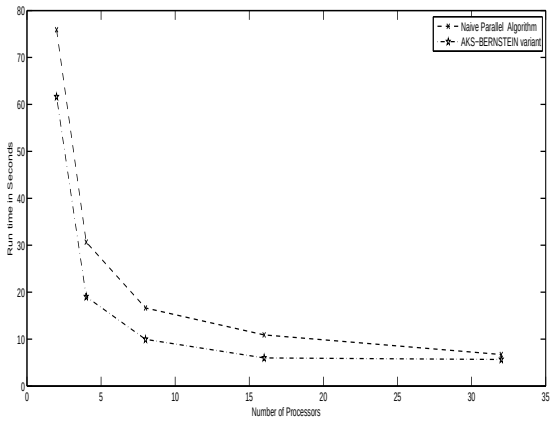

Figure 3. Comparison of run times for the reference implementation and developed implementation.

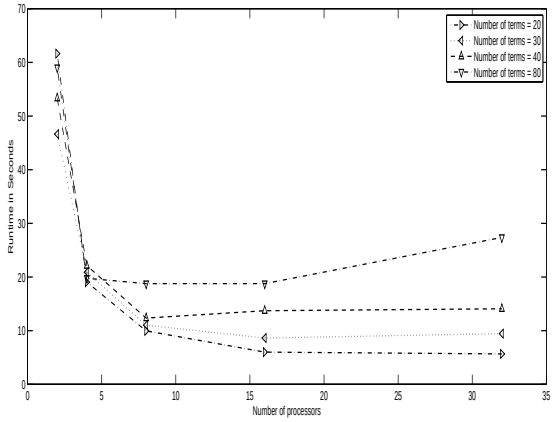

Figure 2. Comparison of run time obtained for different number of polynomial terms evaluated at slave processor.

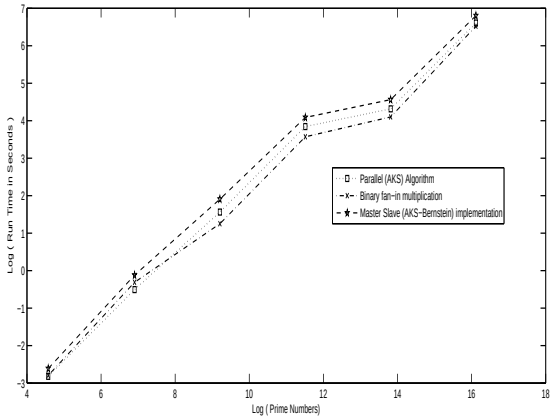

Figure 4. Comparison of run time for various prime numbers for three different implementations.

binary fan-in multiplication with precomputation performs very bad because of the communication overhead involved in communicating GMP variables.Further there will be load imbalance in this version as the binary representation of the exponent contains variable number of ones across each processor and therefore number of exponents that have computed in each processor will be different leading to load imbalance.

V. Conclusions and Future work

The speedups obtained by parallelizing the algorithm scales with number of processors, hence this method can be used in large distributed environments. Precomputed values bring down the run time of the algorithm significantly. In spite of these advantages the algorithm in its current form may not be practical as other algorithms with slightly better asymptotic run time have been developed .

The future work of this project could be to implement other faster variants of AKS algorithm. Memory efficient precomputation can be used so that more polynomial coefficients can be stored in the same amount of space. Develop an application similar GIMPS for proving prime numbers over distributed network of computers.

References

[1] M. Agrawal, N. Kayal, and N. Saxena, "Primes is in p," *Ann. of Math*, vol. 2, pp. 781–793, 2002.

[2] D. J. Bernstein, "Proving primality after agrawal-kayal-saxena," *Department of Mathematics, Statistics, and Computer Science, University of Illinois. Available from the World Wide Web:< http://cr. yp. to/papers/aks. pdf*, 2003.

[3] R. E. Crandall and J. S. Papadopoulos, "On the implementation of aks-class primality tests," *Advanced Computation Group. Apple Computer e University of Maryland College Park*, 2003.

[4] C. Rotella, "An efficient implementation of the aks polynomial-time primality proving algorithm," *School of Computer Science-Carnegie Mellon University. Pittsburgh-Pennsylvania-USA*, 2005.

[5] J. S. Bronder, "The aks class of primality tests: A proof of correctness and parallel implementation," *Electronic Theses and Dissertations,University of Maine*, 2006.

[6] D. M. Gordon, "A survey of fast exponentiation methods," *Journal of Algorithms*, vol. 27, pp. 129–146, 1998.

[7] E. F. Brickell, D. M. Gordon, K. S. McCurley, and D. B. Wilson, "Fast exponentiation with precomputation: algorithms and lower bounds (1995)," *URL: http://research. microsoft. com/˜ dbwilson/bgmw*.
[8] T. Granlund, "Gnu mp," *The GNU Multiple Precision Arithmetic Library*, vol. 2, no. 2, 1996.