Statement of Requirements

Project Lima

25 January 2003

1 Overview

A software program that will factorise an arbitrary integer utilising probabilistic algorithms and distributed computing methods. The factorisation will be optimised for integers of length less than one hundred decimal digits.

2 Background

A large number of techniques exist for testing large integers for primality [2] and (when composite) attempting to find their factors [3]. Some of these methods are probabilistic in the sense that they use pseudo-random sequences to guide them, and others are probabilistic in the sense that they indicate primality/compositeness with measurable degrees of probability [2, 3].

The time-complexities of such factorisation algorithms are generally exponential with respect to the length of a given composite's prime factors [3]. Although using techniques of parallelism cannot change the complexities of these algorithms, they can bring problems that are on the edge of tractability into the realm of possibility by dividing the heavy parts of some algorithms which lend themselves to parallelism [5].

This project shall implement an application software system that will apply some of these methods, optimised for integers having up to one hundred decimal digits.

3 Statement

Explicitly, the software program will meet the following requirements:

3.1 Factorisation

- The software shall take as an input some integer and produce a list of that integer's factors as output, with a configurably high probability that the factors produced are prime [2 p895].
- The software shall either produce the required output or terminate gracefully upon server administrator request.
- The software shall accept integers of any size, limited only by the computing resources available. However suitable warnings, indicating that failure

may occur shall be provided to the user when integers longer than 100 decimal digits are supplied as input.

- Various algorithms shall be employed to perform factorisation [3]. The software should be capable of choosing the most appropriate algorithms to factorise any given integer. The overriding concern should be to maximise time-efficiency.

3.2 Probabilism

3.2.1 Primality Checking

- Fully deterministic tests for primality are generally too slow to achieve satisfactory performance [2 p888]. However, there exist algorithms that indicate compositeness with absolute certainty and primality with a configurably high certainty [2 pp889-896]. Since these methods cannot indicate primality with absolute certainty, they are classified as probabilistic, but the uncertainty is measurable and can be reduced so that the likelihood of any composite being mistakenly identified as prime is negligible. Moreover, these probabilistic methods considerably outperform the available fully deterministic methods[2], and hence shall be used where appropriate in ascertaining the primality of factors obtained.

3.2.2 Prime Factorisation

- Probabilistic techniques exist for extracting factors from composite integers. These techniques have better time-complexity for extracting larger factors from integers than deterministic methods such as trial division [3]. Hence, where performance of factorisation can be improved by such methods, they shall be used. However, deterministic methods may also be used if appropriate, such as may be the case for extracting smaller factors [2 p888].

3.2.3 Distribution

- Prime factorisation is a hard problem, possibly NP-complete, although this has yet to be proven. Existing factorisation algorithms exhibit time-complexities that are bounded by

$$O(N\left(N^{\epsilon(N)}\right))$$
$$\text{where } \epsilon(N) = c\sqrt{\ln\ln N/\ln N}$$

such that c is an constant greater than $\sqrt{8}$ [1 p386]. Although probabilistic techniques provide some increase in performance, there are and may always be fundamental limiting factors imposed by the speed of currently available hardware. Once the limit of an individual processor's speed has been reached the only recourse to further accelerate computation is to parallelise part of the task, possibly on multiple processors, and possibly across multiple machines. Such distribution techniques cannot change the fundamental complexity of the algorithms in use, however ideally they are able to linearly scale the time required to perform computation on some arbitrary input [5], thus bringing a slightly larger range of composites into

the scope of such algorithms. Some factorisation algorithms lend themselves well to parallelism, since they operate on tasks that may be easily divided into subtasks, in a scalable fashion [5]. Hence, where appropriate, the processing of such algorithms shall be distributed across several "machines".

- The distribution of tasks across several machines should be a means to achieving the stated goals, and should not be the goal in itself. Distribution need not be used when the advantages of this method are outweighed by the associated overheads.

- Reliable network communication shall be assumed, that is that any data received by either client will be assumed to be the same as that which was sent. Hence, beyond use of standard protocols, no additional error checking or redundancy need be implemented. Futhermore, network latency shall be assumed to be relatively low such that server and client can communicate without concern as to whether the failure to reply is a system failure or due to network latency.

3.3 Arithmetic

- Experience with large integer packages has shown that they are prone to small bugs [unknown - someone please provide a reference]. However dependability on accuracy for big integer arithmetic is essential. Failure to compute correct values for any given operation could result in catastrophic errors in the result. Hence the libraries or functions used for arithmetic shall be tested thoroughly to produce evidence that the arithmetic is accurate "beyond all reasonable doubt". Further multiplication shall be performed on the factors in order to verify the final result.

- An assumption shall also be made as to the accuracy of the arithmetic libraries and functions, in particular as to their self-consistency and accurate provision of results within reasonable expectations.

3.4 Interface

- A user interface shall be provided for both server and clients offering simple interaction for the user and in particular in the case of the server, the provision of abortion for the current operation. The server user interface shall display the current progress of the operation to the server administrator, both in plain English and with technical details.

4 Conclusion

This system shall be implemented by Wednesday, the 26th Feburary, 2003 by the following group members:

David Cornish

Janet Wang

Jonathan Knowles

Matt Painter

Phil Wise

Raghav Kapoor

Tara Symeonides