1. Introduction

This paper presents a survey of research in **NC** computable functions. The class **NC** is "the class of functions computable very rapidly (in time polynomial in log n) by a parallel computer with a feasible (i.e. polynomial) number of processors [Cook]." Many important functions are in **NC**. These include integer arithmetic, polynomial arithmetic, Discrete Fourier Transforms, matrix determinants, inverses and matrix arithmetic.

In a sense, the class **NC** summarizes the class of functions which can be successfully sped up through parallelization. As the story came to me, this class received the name "Nick's Class" in deference to Nick Pippenger's paper *On Simultaneous Resource Bounds*[79]. Prior to this, models of parallel computation explored machines of unbounded space resource growth, for example *On Parallelism in Turing Machines* by Dexter Kozen[76] and *Relating Time and Space to Size and Depth* by Allan Borodin[77]. This paper also solidified many of the difficult notions involved when comparing machine and circuit capabilities.

The class **NC** can be further divided by considering the degree of the time bounding polynomial. Integer addition, subtraction and multiplication are demonstrable in NC^1. Integer division resists inclusion in NC^1. It can be included in NC^2. The results of [Beame,Cook,Hoover84] however do give a polynomial sized $\log^1$ depth circuit for integer division, it fails to demonstrate division in NC^1 by being too "difficult" to construct. Matrix multiplication is in NC^1. This is the theoretical equivalent to the discussion in [Mead and Conway 80].

The subclass NC^2 includes matrix inversion, matrix determinants and characteristic polynomials of matrices. This result is given by [Csanky 76] and [Berkowitz 84].

For polynomials, the results of [Reif83] show that multiplication and division is in NC^1, and [Borodin, von zur Gathen and Hopcroft82] show GCD in NC^2. The outstanding problems in **NC** theory are :

1. Is Integer GCD in **NC**.
2. Is Integer Division in NC^1.
3. Is $a^b \pmod{2^n}$ in **NC**.

The effort undertaken in this thesis is to survey the field of **NC** theory, to present a detailed description of results for arithmetic computation, to present a detailed description of turing machine to circuit reductions, and to comment on the state of the art in solving the outstanding problem number 1 above, that of fast computation of integer greatest common divisor by poly-bound parallel processors. My goal is to retain much detail on a smaller number of topics in order to ascertain by example the crucial basic concepts of the theory of **NC** computability. The contents of this thesis fall into two classes, literature survey and mathematical proof. The material presented as mathematical proof is, in general, original.

2. Models of Parallel Computation

There are several realizable models for parallel computation. It would be desirable to reduce these models to a single Turing Machine model without destroying the power of these models. All realizable models use actual electronic circuits and memory cells. The use of memory cells, conceptually, is a matter of economy, one would like to use the electronic circuits repeatedly. Conceptually, then, memory cells are avoided by repetitions of the electronic circuits stacked up one level per computation step. If the starting point was a realizable model for parallel computation, a circuit model could replace it without loss of major theoretical features.

Not restricting the original model to be defined in terms of hardware, we could call upon Church's thesis in order to assure us that it could be defined in terms of hardware, that is, simulate the model on a computer simulating a turing machine, the turing machine itself exhausts all possible models of computation.

The point being, the circuit model is an extremely fast model due to its highly distributed nature and the ability to "hand optimize" its operation, yet it is also completely general. Additionally, natural examples of boolean circuits occur, for instance in VLSI or programmable logic arrays, and have great importance for current computer technology.

For these reasons, the following exploration of parallel models is most specifically how the circuit model and the turing machine model relate. Introduced is a new parallel turing machine whose run time and operation is more like that of the circuit model.

2.1. Results in the Literature

Three papers studying boolean circuit complexity are *Parity, Circuits, and the Polynomial-Time Hierarchy* by Furst,Saxe and Sipser [81], *A Complexity Theory for Unbounded Fin-in Parallelism* by Chandra, Stockmeyer and Vishkin [82], and *On Simultaneous Resource Bounds* by Pippenger [79]. Pippenger's paper makes clear the intended nature of the statement "This Turing Machine and that Boolean Circuit are the same." Many results for a circuit-depth hierarchy of functions are given in [CSV82] and the reductions between the circuit model and another popular model, the WRAM, are given. For this paper, the major result of [FSS81] is the establishment of an upper bound to the power of fixed depth, polynomial size circuits. Specifically, parity, majority, multiplication and transitive closure were shown to be too hard to be computed by any polynomial sized circuit of constant depth.

Two frequently referenced works are *Parallel Prefix Computation* by Ladner and Fischer [80] and *On Relating Time and Space to Size and Depth* by Allan Borodin [77]. [LF80] give an excellent procedure for constructing tree structures computing $x_1 \circ x_2 \circ \ldots \circ x_k$ for $1 \leq k \leq n$ where $\circ$ is any associative operator. Such a structure is needed in addition, for example, and [LF80] gives a shallow depth circuit for addition using this structure. Later in this paper I give a larger but faster circuit for addition. The paper by Borodin uses transitive closure to establish the relationship — circuit size is equivalent to turing machine space.

The question whether the genealogy of "best" circuits contained significant branches, that is, whether there exists a family of circuit topologies, each topology serving a certain computation best, was answered in *A Depth Universal Circuit* by Cook and Hoover [85]. The result was a universal circuit which could simulate any circuit of depth d in depth $O(d)$.

Circuits are treated thoroughly in the text **The Complexity of Computing** by John Savage.

Other models for realizable parallel computation include the WRAM, the Galil-Paul tree-machine, and the Cube-Connected-Cycle machine. The WRAM is described in [CSV 82] as P RAM machines running in parallel, communicating using a common memory, but each processor has its own memory as well. It advantage is that is gives a familiar and powerful format for the presentation of algorithms. The Galil-Paul machine is introduced In *An Efficient General Purpose Parallel Computer* [81] by Galil and Paul, a machine is described which can efficiently simulate any parallel computation model. It is a parallel processor which addresses the problem of a universal computing topology. This paper also approaches the question of "good" models for parallel processing theoretically, and gives important results showing that parallelism does always help. The CCC, described in *The Cube-Connected-Cycles : A Versatile Network for Parallel Computation* by Preparata and Vuillemin, is a VLSI efficient implementation of a hypercube structured machine. This model is both technologically feasible and supports optimal time implementation of FFT, matrix multiplication, and efficiently implements sorting.

An even stronger extension of the turing machine is given by the alternating turing machine. The development of this model is given in three papers *On Parallelism in Turing Machines* by Dexter Kozen [76], *Alternation* by Chandra and Stockmeyer [76] and *Alternation* by Chandra, Kozen and Stockmeyer [81]. That this machine can explore all of PSPACE in polynomial bound time invalidates it as a realistic model of parallel computation.

2.2. Boolean Circuits of Unbounded Size

Definition. Let G be a directed graph with node set V and link set E. Define $\pi(v)$ as all nodes u such that there is a directed link from u to v. Then $|\pi(v)|$ is called the in-degree of v and every node in $\pi(v)$ can be called a π-node. Likewise, $\sigma(v)$ is the set of all nodes u such that there is a directed link from v to u. Call $|\sigma(v)|$ the out-degree of v and σ- node any node in $\sigma(v)$.

Definition. A computation net is a directed, acyclic graph with a map $f_v : \pi(v) \to v \quad f_v \in F \quad \forall v \in V$. The computation net is then said to be defined over F.

Definition. A boolean circuit computes a function $g : \{0,1\}^n \to \{0,1\}^m$ if it is a computation net with n inputs and m outputs defined over a set of boolean functions . Inputs are the n nodes of in-degree 0 identified with the domain of g. Outputs are the m nodes identified with the range of g. Two measures are given to circuits. The number of links in the circuit $S(n,m)$ is the size of the circuit. The length of the longest path from any input node to any output node is the depth $D(n,m)$ of the circuit.

Definition. Given the set F, if $\forall g \ \exists$ a boolean circuit over F which computes g, then the set F is a complete basis.

Lemma. *The set of boolean functions $\{\wedge, \vee\}$ is not complete.*

Proof. The function $\neg$ cannot be simulated. This is true by inspection for circuits of depth 1. If a depth n circuit simulated $\neg$, there would be a depth $n-1$ circuit also simulating $\neg$. Assume the output node at depth n is an $\wedge$ node. On input 0, the output is 1, therefore all π-nodes to the depth n node are 1. When the input is 1, the output is 0, therefore some π-node to the depth n node is 0. This node computes $\neg$ and is at depth $n-1$. The case for $\vee$ is similar.

Lemma. *The parity function can not be computed by the incomplete set $\{\wedge, \vee\}$.*

Proof. If H computed the parity function on n bits, by setting $n-1$ bits appropriately, H computes $\neg$.

Theorem. *The set*

$$F = \{1, \neg x_1, \neg(x_1 \wedge x_2), \ldots, \neg \wedge_{i=1}^{n} x_i, \ldots\}$$

is a complete basis.

Proof. Given by the construction in the following theorem.

Theorem. *Any boolean function can be calculated by a net over F of depth 3 and size :*

$$S(n,m) = mn2^n + m2^{(n+1)} + n + 1$$

Proof. Generate the complement of the input using n nodes of in-degree 1. Form a m by 2^n array of nodes to encode the output. The inputs to each node of the array are a selection of n lines from the input and its complement to address that node, and a single constant input to encode the output bit. For example, if bit m is 1 for input 1011, then the node in the m^{th} row and 11^{th} column will be :

$$f_{m,11} = \wedge\{x_1, \neg x_2, x_3, x_4, 1\}$$

The in- degree of these nodes are all $n + 1$. For each of m output bits, it is necessary to merge 2^n lines into 1. This is done using m nodes of in-degree 2^n. To generate the constants 1 and 0 use two nodes, one of in-degree 0, the other of in-degree 1. Sum the in-degree of each node over all nodes to complete the proof.

Theorem. *The depth given by the previous theorem is minimal.*

Proof. It suffices to show a contradiction for a depth 2 boolean circuit over F computing $g : \{0,1\}^n \to \{0,1\}$. There are at most 2^n distinct nodes in the first level. There is at most 1 node in the second level. This node can have in-degree at most $2^n + n$. Applying De Morgan's theorem :

$$C = \big(\bigvee_{i=1}^{2^n} \wedge_i(x_1, x_2, \ldots, x_n)\big) \vee \neg x_{i_1} \vee \neg x_{i_2} \vee \ldots \vee \neg x_{i_k}$$

Where $\wedge_i$ is the computation of the i^{th} AND node of level 1, and $x_{i_1}, x_{i_2}, \ldots, x_{i_k}$ are the inputs with edges to the level 2 node. Call this set X. In any $\wedge_i$ that uses an input $x_j \in X$, either x_j is 1 or C is necessarily one. All $x_j \in X$ can therefore be removed from all $\wedge_i$. If this depth 2 circuit is complete, then let it compute the parity function. For the parity function $|X| = 0$, that is, there are not bits in the input such that by being 0 the parity is necessarily 1. So the parity function is being computed over the incomplete set $\{\wedge, \vee\}$, in contradiction with a previous lemma.

2.3. Boolean Circuits and Turing Machines are Polynomial Equivalent.

What the preceding section shows is that if a function is effectively specified over a finite domain, then it can be pre-computed for all values of the domain and the results placed in a table. Any such function can then be "computed" by table look-up in constant time (as a function of n).

Suppose one were presented with the finite description of a turing machine M computing f on input N of length n. Suppose this turing machine was time bound by $T(n)$ and space bound by $S(n)$. What the preceding section shows is that there is a turing machine which simulates M on the 2^n values $0, 1, \ldots,$ recording the result into a circuit description. So by [P 79] this description is uniform (time $O(T2^n)$). A more useful result is uniformity (space $O(\log n)$). This is due to the large utility of log space reductions in theoretical computer science and due to the short run times of log space bound turing machines.

In a sense, the previous result, uniformity (time $O(T2^n)$) can be compared to the "nonuniform" turing machine which computes f modulo g, where $g \equiv f$.

What this section develops are the following results :

> Given the description of a turing machine, this description being n symbols long, a parameter k, and explicit time and space bounds S and T for the turing machine, there is a LOGSPACE transducer (logarithmic in n) producing from this description the description of an equivalent boolean circuit for inputs of length k. The circuit is of size $O(ST)$ and of depth $O(T)$.
>
> Given the description of a circuit with size S and depth T, there is a LOGSPACE transducer producing from this description the description of an equivalent turing machine. The turing machine runs in time $O(S^2T)$ and space $O(S)$.

Definition. A Turing Machine is the quintuple

$$\{Q, \Sigma, F, q_s, \Delta\}.$$

Where Q is the set of states, $F \subseteq Q$ the set of halting states, $q_s \in Q$, a single start state. Σ is an alphabet, $\Sigma = \{0, 1, ƀ, \vdash, \dashv\}$. Δ is the state transition table, $\Delta : Q \times \Sigma \to Q \times \Sigma \times \{L, R\}$.

Definition. Let Δ be the string encoding the transition table Δ. $\Delta \in \Sigma^*$ is of the form :

$$\vdash \bar{q}\, ƀ\, \bar{\sigma}\, ƀ\, \bar{q}'\, ƀ\, \bar{\sigma}'\, ƀ \underset{L}{\overset{R}{}} ƀ \ldots \dashv$$

where $\bar{q}$ and $\bar{\sigma}$ are encodings in $\{0,1\}^*$of the elements in Q and Σ, respectively.

Definition. A circuit description is a string of the form :

$$\vdash \bar{n}\, ƀ\, \bar{s}\, ƀ\, \bar{n}_1\, ƀ\, \bar{n}_2\, ƀ\ \ldots\ ƀ\, \bar{n}_k\, ƀ\, ƀ\ \ldots \dashv$$

where the $\bar{n}$ are node numbers encoded in $\{0,1\}^*$, $s \in \{\text{input, output}, \neg, \wedge, \vee, 0, 1\}$, $\bar{s}$ is an encoding of s in $\{0,1\}^*$. The double blank space separates the string into repeating groups. Each group encodes node n of type s with inputs $n_1, n_2, \ldots, n_k$.

Lemma. *Given the description Δ of a state transition table, a boolean circuit $\widehat{\Delta}$ simulating Δ is efficiently constructable. That is, there is a LOGSPACE transducer which inputs Δ and outputs $\widehat{\Delta}$.*

Proof. The essential concept here is that for each entry in Δ, the transducer outputs a universal circuit which says, if the input to $\widehat{\Delta}$ matches the left-hand side of this transition table entry, output the right-hand side of this transition table entry, then copies that entry of the table into the circuit as a constant. The final OR layer will be possible without referencing the past layers if they were numbered in some organized fashion. For example, if $|\bar{q}| = n,\ |\bar{\sigma}| = m,$ where $|x|$ is the length of x, the universal circuit requires $5(n+m)+2$ nodes. The domain is coded in $n+m$ constant nodes, and is compared with the input using $n+m$ exclusive or gates. The results are summed by a binary tree of OR gates using $n+m$ nodes, the last node being an inverter. This "select" signal than feeds a bank of $n+m+1$ AND gates which assert the output, stored in $n+m+1$ constant nodes. The first $n+m$ nodes are reserved for input, the next $n+m+1$ are reserved for output. The remaining nodes are numbered beginning with $5(n+m)+2$ and ending with $(5(n+m)+2)i-1$. Of this group, all nodes congruent to $n+m+1$ modulo $5(n+m)+2$ are summed in the final large output tree. A constant number of counters of length $O(\log|\Delta|)$ are need to keep track of all this.

Theorem. *Given Δ, the explicit tape bound S, and the explicit time bound T, of a TM M, a circuit $\widehat{M}$ simulating M is efficiently constructable. $\widehat{M}$ has size $O(ST)$ and depth $O(T)$.*

Proof. Simulate the S tape cells in one step with mS wires, where m is the size of the encoding for Σ. The value of m is available by pre-scanning Δ. Another S wires simulates the head. The TM loops for $l = 1 \ldots T$. For each l it churns out S universal circuits which input the previous tape contents, which output these symbols too, with the exception of the cell upon which the head rests. This cell selects the output of $\widehat{\Delta}$ to pass to the next level of universal tape simulators. Also, these universal circuits input the previous head position and output the next head position, either moved one right or moved one left, according to an input again supplied by $\widehat{\Delta}$. $\widehat{\Delta}$, of which T are needed, are efficiently constructed once for each level. There numerous is a technical details for numbering nodes such that the transducer need never look back. However, the regular structure gives rise to simple a organization scheme based on congruences.

Theorem. *Given a circuit $\widehat{\Delta}$, a TM description M simulating $\widehat{\Delta}$ is efficiently constructable. M runs in time $O(S^2T)$ and space $O(S)$ where S is $\widehat{\Delta}$ size and T is $\widehat{\Delta}$ depth.*

Proof. The transducer outputs a universal circuit simulation Turing Machine and then proceeds to code the given circuit description into a "query box". The universal circuit simulator (UCS) assumes a special state i that when entered with the head of the work tape pointing to an integer, the query box will copy the full node description of the i^{th} entry in $\widehat{\Delta}$ onto the work tape then enter state o. If there is no i^{th} entry, the return is to state f. Assume the node description returns the following:

$$\bar{n}\ \text{ƀ}\ \bar{\sigma}\ \text{ƀ}\ \bar{n}_1\ \text{ƀ}\ \bar{n}_2\ \text{ƀ}\ \bar{n}_3\ \text{ƀ}\ \text{ƀ}$$

Then the UCS does the following :

1. Go to input tape cell n. If it is blank continue, else go to 4.
2. Go to tape cells n_1, n_2, n_3. If any are blank go to 4, else write down on the work tape the contents of each of these cells.
3. Use σ and calculate n from n_1, n_2, n_3. Write the result in cell n. Erase the work tape, mark it with a one and exit.
4. Erase the work tape and exit.

The outer loop of the UCS is as follows:

5. Set a special halt detection cell to zero, a counter to 1
6. Copy to counter to the work tape and goto 1.
7. If return to state f go to 8. Else OR the contents of the work tape with the halt detection cell, increment the counter and go to 5.
8. Check halt detection cell for zero. If zero HALT. Else go to 5.

The creation of the query box is considered next. For each entry in $\widehat{\Delta}$ output a chain of states which decrement the work tape counter, check it for zero, and if zero write the node description string onto the work tape. If the node is not zero, skip ahead to the state which starts an identical chain for the next $\widehat{\Delta}$ entry. If there are no more $\widehat{\Delta}$ entries, go to state f. In implementation, steps 1–3 are handled with one sweep of the tape. Therefore the run time of this algorithm is $O(SNT)$, where N is the number of nodes, T is the depth of the circuit and S is the circuit's size. Since $N \leq S < N^2$, the time bound is expressed as $O(S^2T)$.

2.4. Parallel Models

We would like a universal model of parallel computation. Ultimately, the model should be of the clarity, simplicity and elegance of the Turing Machine model for sequential calculation. We have seen that simulation of a circuit by a TM introduces significant slow-down. Suppose, however, that one tried to capture the parallelism by creating multiple TM's, perhaps with multiple heads, perhaps sharing multiple tapes. Note, however, the following :

Theorem. *Suppose any parallel-TM model. Restrict the model so that the number of heads is bounded by a polynomial on input size. Simulating arbitrary depth $T(n)$ circuits in time bounded by a polynomial in $T(n)$ would imply P=NP.*

Proof. From [Borodin77], any $S(n) \geq \log n$ space bounded TM can be simulated by a depth $\leq O(S(n)^2)$ circuit. Take any PSPACE problem, simulate it by a circuit, and the circuit by our proposed fast TM. The problem is then in PTIME for our parallel- TM model. By direct simulation of the multiple TM's in the parallel-TM model, the problem is also in PTIME for the deterministic TM model. This collapses the PTIME<NPTIME<PSPACE hierarchy.

It is not too disappointing, then, that a naive parallel-TM is possible, but its running time is $O(S+T)$ to simulate size S, depth T circuits. That is to say, the construction in [Borodin77] results in exponential size circuits, hence the simulation is not "fast".

Definition. A parallel-TM is a collection of Turing Machines and a collection of tapes. The individual TM's may have multiple heads, and these heads may be positioned on any of the tapes. In addition to space and time, the additional measure $P(n)$, the number of processors (TM's), is specified.

Theorem. *Given a circuit $\widehat{M}$, of size S and depth T, there is a p-TM simulating this circuit, and it runs in space $O(S)$, time $O(S+T)$, using $O(S)$ processors.*

Proof. For each node in the description of $\widehat{M}$ output a TM. This TM is like the UCS outlined above except that the query box consists of only a single node, the particular node under consideration when this UCS was emanated. The TM is specified to have as many heads as the node has inputs. That is, each individual TM places its heads on the tape cells corresponding to that node and to the input nodes of that node, and then begins simulating the gate. S units of time are needed for positioning, T units are needed for execution.

3. Circuit Complexity of Arithmetic.

The basic arithmetic operations on integers, addition, subtraction and multiplication are in NC^1 by circuits uniform in (space $O(\log n)$). By the same uniformity constraint, integer division is in NC^2. By allowing circuits constructable in polynomial time, integer division is also in NC^1. Demonstrating these results is the purpose of this chapter.

With the exception of the constant depth adder, all circuits are in-degree bound by a constant.

3.1. Addition, Part I.

Addition, typically performed, is $O(n)$. Proceeding from right to left, one digit at a time, a sum is formed, an perhaps a carry. As interest moves from one digit to the next, the carry is brought leftwards also. The essential difficulty in speeding up this method of addition is the slow propagation of the carry from radix place to radix place. This difficulty can be surmounted by a large tree structure called the look-ahead carry.

The look-ahead carry uses a tree to propagate forward the carries out of individual bit positions. Each bit position reports whether that position, by itself, will generate a carry. Each bit position also reports whether it will support transmission of a carry through it (carry propagate). Simple recursive logic formulas combine these bit-wise results into bit-string results where the strings are of lengths powers of 2.A parallel prefix circuit combines these results into a simultaneous image of the carry values that will occur during the addition. The addition is then computed in one step using this image.

Definitions. A "Binary Integer" is a string 0,1 of any length. Further more, integers differing only in the number of zeros on the left are thought of as indistinguishable. In general, I will follow convention and let "number" and "representation of number" be confused. The "Selection" function is defined as :

$\mathrm{Sel}(m, i, j) =$ *the binary integer resulting from the substring operation on binary integer m, the substring being the i^{th} through j^{th} characters extracted.*

The characters ("bits") are numbered 1 through n starting on the right. The "Generate Carry" proposition for binary addition is defined as :

$\mathrm{Gen}(n, a, b) = 1$ *iff the result of $a + b$ generates a carry out of the n^{th} bit position.*

The "Propagate Carry" proposition for binary addition is defined as :

$\mathrm{Prop}\ (n, a, b) = 1$ *iff the result of $a + b + 1$ generates a carry out of the n^{th} bit position.*

Lemma. Gen *and* Prop *are defined recursively for any* $0 < m < n$ *:*

$$\begin{aligned}\mathrm{Gen}(n, a, b) = {} & \mathrm{Gen}(n - m, \mathrm{Sel}(a, m + 1, n), \mathrm{Sel}(b, m + 1, n)) \\ & \vee (\mathrm{Gen}(m, \mathrm{Sel}(a, 1, m), \mathrm{Sel}(b, 1, m)) \\ & \qquad \wedge \mathrm{Prop}(n - m, \mathrm{Sel}(a, m + 1, n), \mathrm{Sel}(b, m + 1, n))) \\ \mathrm{Prop}(n, a, b) = {} & \mathrm{Prop}(n - m, \mathrm{Sel}(a, m + 1, n), \mathrm{Sel}(b, m + 1, n)) \\ & \wedge \mathrm{Prop}(m, \mathrm{Sel}(a, m, n), \mathrm{Sel}(b, m, n))\end{aligned}$$

$$\text{Gen}(1,a,b) = \text{Sel}(a,1,1) \wedge \text{Sel}(b,1,1) \qquad \text{Prop}(1,a,b) = \text{Sel}(a,1,1) \otimes \text{Sel}(b,1,1)$$

Lemma. *Addition of a and b is equivalent to the calculation of Gen(n,a,b) and Prop(n,a,b) for all n.*

Proof. $\text{Sel}(a+b,i,i) = \text{Gen}(i-1,a,b) \otimes \text{Prop}(i,a,b)$.

This calculation can be performed in $O(\log n)$ time. Using the recursive definitions above, choosing $m = n/2$ at each level of recursion. This quickly gets Gen and Prop for all substrings of a and b of the form :

$$a_{(m+1)i}a_{(m+1)i-1}\cdots a_{mi+1} \qquad \forall m = 0,1,\ldots \text{ and } \forall i = 2^k,\ k = 0,1,\ldots \tag{3.1}$$

Theorem. *Not more than $O(\log n)$ of these results need be combined to calculate* Gen *or* Prop *for any n.*

Proof. An algorithm is given that computes Gen(k,a,b) for any k desired. The run time is analyzed and shown $O(\log n)$. Essentially what this algorithm does is build up Gen using pieces of the form of (3.1). It needs to select one piece for every 1 bit in the binary representation of k. Prop can calculated in a similar manner. Since $k \leq n, k$ is representable in $\log n$ bits, hence only $\log n$ applications of the recursive Gen formula are needed.

```
let the binary representation of $k$ be $r$
let $place := 0$; $pointer := \log n$
let $P := 1,\ G := 0$
until $pointer = 0$ do
          if (Sel($r$,$pointer$,$pointer$) = 1) then
                    let $m := 2^{pointer-1}$
                    let $a' := \text{Sel}(a, place+m, place+1)$
                    let $b' := \text{Sel}(b, place+m, place+1)$
                    $P := P \wedge \text{Prop}(m,a',b')$
                    $G := G \wedge \text{Prop}(m,a',b')$
                              $\vee\text{Gen}(m,a',b')$
                    $place := place + m$
          fi
          decrement pointer
od
```

The exact size and depth of the carry-propagate adder are calculated using some general assumptions. Let the input size be a power of 2. We want this analysis to correctly measure size and depth with regard to the circuit topology and major conceptual approach. We wish to ignore measures that depend on implementation technology, availability of certain types of gates compared to other types of gates — for example. Therefore, we allow that each node to have a compound boolean expression attached to it, when that appears convenient and realistic.

Result. *The carry-propagate addition circuit has depth and size :*

$$D(n) = 2\log n \qquad S(n) = 12(n-1) - 4\log n$$

Proof. The circuit has four levels :

1. First level. The single bit results for Gen and Prop are calculated by n nodes of in-degree 2.
2. Carry Propagate. This refers to the Gen and Prop values of equation (3.1). A binary tree ending in $n/2$ nodes is formed. Each node carries the recursive form for Gen and Prop. The depth of the tree is $\log n$, and it contains $1 + 2 + 4 + \ldots + n/2 = n - 1$ nodes of in-degree 4. The root of the tree can be discarded in this analysis, making both depth and node size one smaller.
3. Carry Disperse. This refers to the algorithm given in detail above, implemented "in hardware". A more complicated tree structure is used here, (see diagram). What actually is implemented is a forest of trees on exponentially growing leaf number (2, 4, 8, etc.). The maximum depth of this graded forest is $\log n - 1$, and it has $n/2 - 1 + n/4 - 1 + \ldots + 1 = n - \log n - 1$ nodes of in-degree 4.
4. Final Level. The sum is computed given all Gen and Prop. This requires n nodes of in-degree 2.

Summing these equations, the result is obtained.

3.2. Addition, Part II.

The problem of slow carry propagation can be resolved in a different manner. A circuit of simpler description results, and one whose depth is half that of carry look ahead. The gist of the method is to precalculate partial results of the addition under two assumptions, there is a carry in at the right and there is not such a carry. A tree coordinates and resolves the assumptions, continually creating exponentially larger pairs of partial results.

Definition. Let a and b be n bit numbers, and $m < n$, then

$$\begin{aligned} \mathrm{Proj}(i,(a,b)) &\equiv \begin{cases} a & \text{if } i = 0 \\ b & \text{else} \end{cases} \\ \mathrm{Carry}(n,a) &\equiv \mathrm{Sel}(a,n,n) \\ +(n,a,b) &\equiv (a+b, a+b+1) \end{aligned}$$

The result of $+$ as an operator is a pair of $n+1$ bit numbers.

Definition. $a \mid b$ is the concatenation of binary numbers a and b. $a \mid^* b$ is the concatenation of a with b after the MSB of b has been removed.

Lemma. *For any m, $1 < m < n$, a, b being n bit numbers, and a', a'', b', b'' are numbers such that*

1. $a = a'' \mid a'$ *and* $b = b'' \mid b'$,
2. a' *and* b' *are* m *bits long.*

then

$$\begin{aligned} +(n,a,b) = \big(\ &\mathrm{Proj}\big(\ \mathrm{Carry}(\ \mathrm{Proj}(0, +(m,a',b'))), +(n-m,a'',b'')\big) \\ &\quad |^{*}\ \mathrm{Proj}\big(0, +(m,a',b')\big), \\ &\mathrm{Proj}\big(\ \mathrm{Carry}(\ \mathrm{Proj}(1, +(m,a',b'))), +(n-m,a'',b'')\big) \\ &\quad |^{*}\ \mathrm{Proj}\big(1, +(m,a',b')\big)\big) \end{aligned}$$

Proof. .

$$a + b = (a''2^n + a' + b''2^n + b') = (a'' + b'')2^n + (a' + b')$$

By using the above recursion setting each m to $n/2$, a binary tree of $O(\log n)$ depth results. Each level of the tree requires only concatenation, projection and selection operations, and is therefore of constant depth, showing the above algorithm to be of $O(\log n)$ depth complexity. The exact analysis follows.

Result. *The addition circuit given has depth and size :*

$$D(n) = \log n + 1 \qquad S(n) = 2n + 2n\log n + 4n\sum_{k=1}^{\log n} 2^{-k} = 6n + 2n\log n - 4$$

Proof. At level k, $1 \le k \le \log n + 1$, the output is a set of pairs. There are $2^{\log n - k + 1}$ pairs, each element of the pair being a $2^{k-1} + 1$ bit number. The input to the first level is two n bit numbers. The input of every other level is the output of the previous level. This is sufficient information to calculate the stated bounds.

3.3. Constant Depth Circuit for Addition

The above circuits have nodes of constant in-degree. For this reason they are of immediate practical importance. However, NMOS technologies make possible gates with variable in-degree. The assumption of mild in-degree growth is therefore not without practical interest. For this reason a very fast addition circuit using gates of poly-bounded in-degree is outlined.

Definition. Given a and b, n bit binary integers, define the one- step carry function for $1 \le k \le n$ as :

$$G(k,a,b) = \bigvee_{i=1}^{k-1} \mathrm{Gen}(i,a,b) \wedge \bigwedge_{j=i+1}^{k-1} \mathrm{Prop}(j,a,b)$$

Lemma. *$G(k,a,b)$ is the carry into bit position k when a and b are added.*

Lemma. *Given* $\mathrm{Gen}(i,a,b)$, $\mathrm{Prop}(i,a,b)$, $\forall\, i < k$, *then if* $k > 2$, $G(k,a,b)$ *can be computed by a depth 2, size* $\binom{k}{2} + k - 2$ *circuit of nodes with in-degree bounded by* $k-1$. *If* $k \le 2$, $G(k,a,b)$ *is available directly, that is, circuit size and depth is 0.*

Proof. By direct implementation of the formula using $k-2$ AND gates of in-degree from 2 to $k-1$ and 1 OR gate of in-degree $k-1$.

Lemma. *Given* $\mathrm{Gen}(k,a,b)$,$\mathrm{Prop}(k,a,b)$, $\forall\, k < n$, *then* $G(k,a,b)$, $\forall\, k \leq n$ *can be computed by a depth* 2*, size* $O(n^3)$ *circuit of nodes with in-degree bounded by* $n-1$.

Proof.

$$\sum_{k=3}^{n} \binom{k}{2} + k - 2 = \binom{n+1}{3} - 1 + \sum_{k=3}^{n} (k-2) = n\left(\frac{n^2+3n-10}{6}\right)$$

Theorem. *Addition by circuits with linear-bounded in- degree is constant depth, size* $O(n^3)$.

3.4 Subtraction

The $O(\log n)$ depth complexity of subtraction follows immediately from addition. The subtraction is performed by selecting a large enough modulus and performing the complementary addition over the resulting commutative and finite group. Besides that this is possible, what needs be shown is that the corresponding addition can be found quickly and easily. Formally, what is shown is an NC^1 reduction of subtraction to addition.

Theorem. *For any* $b, n \in \mathbf{Z}$*, the integers* mod b^n *are an Abelian group over addition.*

Proof. This follows from the underlying commutative group, addition over the integers.

Definition. Expressing $x \in \mathbf{Z}_{b^n}$ in base b results in a string $\{0, 1, \ldots, b-1\}^n$. Define $a^* \in \mathbf{Z}_{b^n}$ to be the $(b-1)$*-complement* of $a \in \mathbf{Z}_{b^n}$ iff $a + a^* = b^n - 1 \pmod{b^n}$.

Since $b^n - 1$ base b is the string $\{b-1\}^n$, obvious parallel algorithm exists for generating the $(b-1)$-*complement*. A table look-up, for instance, uses constant depth and size linear in n. In the case $n = 2$, the *1-complement* is generated by a n wide inverter bank.

Theorem. *Let* $a \in \mathbf{Z}_{b^n}$*, then* $-a = a^* + 1 \pmod{b^n}$.

Proof. Since $\mathbf{Z}_{b^n}$ is a group, the additive inverse is unique.

Theorem. *Subtraction is* NC^1.

Proof. It is reduced to $(b-1)$-*complementation* followed by two additions. In case of binary arithmetic, the complementation is obviously bit-wise inversion. In practice, the double binary addition can be reduced to a single binary addition by small modifications to the addition circuitry.

3.5. Multiplication

Multiplication of integers can be thought of as a short form of iterated addition. That is :

$$mn \equiv \overbrace{n + n + \ldots + n}^{m}.$$

This yields, first of all, a very restricted notion of multiplication. The familiar algorithm uses a table of all one digit products to reduce computation steps to quadratic in the size of the arguments.

The familiar algorithm employs a table and two instances of addition. Implicitly, addition collects a string of table look- ups into a single number called a partial product. Explicitly, the partial products are summed. Consider a multiplication in base b of two n place numbers. The table will be size $O(b^2)$. Since the largest entry in the table will be less than b^2, all entries will be represent with at most two places. Since each argument will be less than b^n, their product will be less than b^{2n}, so it can be represented with at most $2n$ places. Using $2n^2$ copies of the table, preform in parallel all table look-ups and place the results into an n by $2n$ grid. Build an addition tree over the grid to collect the partial products. The first addition in the tree simulates the implicit addition, the remaining additions simulate the explicit additions. This tree will be $O(\log n)$ deep, with each node itself $O(\log n)$ deep. The result is a multiplication circuit of polynomial size and $O(\log^2 n)$ depth.

We now describe a circuit of $O(\log n)$ depth for multiplication.

Definition. Let an m-sum representation of a number $a \in \mathbf{Z}$ be the set $A \in \mathbf{Z}^m$ $A = \{a_1, a_2, ..., a_m\}$ iff $a = \sum_{i=1}^{m} a_i$.

Definition. Let an m-sum implementation of addition be any function $f : \mathbf{Z}^m \times \mathbf{Z}^m \to \mathbf{Z}^m$ s.t. $f(A, B) = C$ iff $\sum_{i=1}^{m} a_i + b_i = \sum_{i=1}^{m} c_i$.

Theorem. *For any base $b > 1$, if $m > 2$ there exists an m-sum implementation of addition of constant depth and linear size.*

Proof. The function f given $A, B \in \mathbf{Z}^m,\ m > 2$ does the following. Place-wise add

$$(a_1 + a_2 + \ldots + a_m + b_1 + b_2 + \ldots + b_m)_{i^{th}\text{-place}}$$

Since each digit is equal to or less than $b - 1$, the sum will be equal to or less than $2m(b - 1)$. I wish to prove that this is strictly smaller than b^m. The proof is by induction on b.

Basis $b = 2$.

$$\frac{b^m}{b-1} = 2^m > 2m \text{ for } m \geq 3$$

Induction. If $b^m > 2m(b - 1)$, then $(b + 1)^m > 2mb$. Proof :

$$(b + 1)^m > b^m + mb > 2m(b - 1) + mb > 2m(b - 1) + 2m = 2mb$$

Therefore the sum can be represented in $C \in \mathbf{Z}^m$ by spreading the m places of the sum across the m elements of C. How exactly the place-wise addition is performed is inconsequential, its depth is bounded as a function of m, a constant.

Lemma. *An $(m-1)$-sum representation of a number is derivable from an m-sum representation of that number using one m-sum addition.*

Proof. . Let the m-sum representation be A. Let Z be the m- sum $\{0, 0, \ldots, 0\}$. Then $A + Z$ yields the m-sum B where $b_m = 0$. Delete this element to form an $(m-1)$-sum number. What needs be shown is that $b^{m-1} > m(b-1)$. This is proved by induction on b.

Basis $b = 2$.

$$2^{m-1} > m \text{ for } m \geq 3$$

Induction. If $b^{m-1} > m(b-1)$, then $(b+1)^{m-1} > mb$. Proof :

$$(b+1)^{m-1} > b^{m-1} + (m-1)b + 1 > m(b-1) + (m-1)b + 1 > m(b-1) + (m-1) + 1 = mb$$

Theorem. *Multiplication is* NC^1.

Proof. Using an n by $2n$ grid and $2n^2$ multiplication tables preform the table look-ups in parallel, leaving the partial products in the form of 3-sum numbers $\{a_1, a_2, 0\}$. Using a tree $O(\log n)$ deep of 3-sum addition nodes sum the partial products. Sum the result with the 3-sum number $\{0, 0, 0\}$ to form a 2-sum number. Lastly, use a $O(\log n)$ addition circuit to transform the 2- sum number into a canonical number.

3.6 Division.

Division has not been shown in NC^1. Faster circuits than NC^2 are possible, for example [Reif83] gives a circuit of depth $O(\log n(\log\log n)^2)$. However, we begin with the following simple approach.

Lemma.

$$\sum_{i=0}^{\infty} x^i = \frac{1}{1-x} \qquad \text{iff} \qquad |x| < 1$$

Proof. See [Rudin].

Lemma. *Let $S_n = \sum_{i=1}^{n} x^i$. If $0 < x < 1/2$, then $S_\infty - S_n < 2^{-n}$.*

Proof.

$$S_\infty - S_n = (S_n + \sum_{i=n+1}^{\infty} x^i) - S_n = x^{n+1} S_\infty = x^{n+1}\frac{1}{1-x} < 2^{-n}$$

Lemma. *Given $x, k \in \mathbf{Z}$, such that $x \neq 0$ and $2^k \geq x > 2^{k-1}$. Let $u = 1 - x\,2^{-k}$. Then*

$$0 \leq u < \frac{1}{2} \qquad \textit{and} \qquad \frac{1}{1-u} = \frac{2^k}{x}.$$

Theorem. *Given an n bit integer x, $1/x$ can be calculated with accuracy $o(2^{-n})$ with a circuit of depth $O(\log^2 n)$.*

Proof. Using the previous Lemma, we need to calculate all powers u_i where $i = 0, \ldots, n$. To find u, the value k is needed. In parallel compare x to $2, 2^2, 2^3, \ldots, 2^n$. This requires polynomially many circuits of $O(\log n)$ depth. Generate powers of u quickly using a binary tree. Such a tree is described in [Ladner]. The nodes of the tree use $O(\log n)$ multiplications, and the tree is $O(\log n)$ nodes deep. The depth of the tree is then $O(\log^2 n)$. The final summation requires a $O(\log^2 n)$ fan-in tree of adders, or a $O(\log n)$ fan-in tree of m-sum adders followed by a single $O(\log n)$ adder.

Corollary. *Integer Division is in* NC^2.

4. Greatest Common Divisor of Two Integers.

Given two integers, find their greatest common divisor. There has been little success in showing this problem in **NC** or proving this problem not in **NC**. The best known sequential algorithm is the Euclidean Algorithm, named for its inventor, Euclid, who lived 300 B.C. Briefly stated, this method creates a column of remainders, entry a_i being the remainder when a_{i-2} is divided by a_{i-1}. The algorithm commences by making the larger of the two input numbers a_0, and the smaller a_1. The algorithm terminates at the first step k for which $a_k = 0$. The GCD is then a_{k-1}.

That this algorithm is correct, and that it is $O(n)$, where n is the length of the bigger integer, is verified in theorems that follow this discussion. Our concern is in fact elsewhere. We would like to parallelize this algorithm and have it run faster. At this moment, no one has succeeded much. The algorithm presented below is by [Chor and Goldreich 85] and is the best currently known. It runs in time $O(n/\log n)$.

Theorem. *The Euclidean Algorithm is correct.*

Proof. Given $a, b \in \mathbf{Z},\ d = (a,b)$, that is d is the GCD of a and b, then $d|a$ and $d|b$ (d divides a and d divides b). Let $c = a \bmod b$, then

$$d|a,b \qquad \longrightarrow \qquad a - mb = de - mdf = d(e - mf) = c \qquad \longrightarrow \qquad d|c.$$

$$d|c,b \qquad \longrightarrow \qquad c + mb = de + mdf = d(e + mf) = a \qquad \longrightarrow \qquad d|a.$$

Specifically $(a,b)|c$ and $(b,c)|a$.

$$(b,c)|a \text{ and } (b,c)|b \qquad \longrightarrow \qquad (a,b) \geq (b,c)$$

$$(a,b)|b \text{ and } (a,b)|c \qquad \longrightarrow \qquad (b,c) \geq (a,b)$$

Therefore $(a,b) = (b,c)$. The proof follows simply.

Theorem. *The Euclidean Algorithm runs in time $O(n)$ for inputs of n bits.*

Proof. Suppose $a_i = a_{i-2} \bmod a_{i-1} > \frac{a_{i-1}}{2}$. It is always true that $a_i < a_{i-1}$ so

$$a_{i+1} = a_{i-1} \bmod a_i = a_{i-1} - a_i < a_{i-1} - \frac{a_{i-1}}{2} = \frac{a_{i-1}}{2}$$

If, however, $a_i \leq \frac{a_{i-1}}{2}$ then $a_{i+1} < \frac{a_{i-1}}{2}$. So $a_{i+1} < \frac{a_{i-1}}{2}$ is always true. This bounds the magnitude of a_i as follows :

$$a_i < da_o 2^{-\frac{i}{2}} \text{ for some fixed } d.$$

Since $a_i \in \mathbf{Z}^+$, the run time can be approximated by the value of i s.t.

$$da_o 2^{-\frac{i}{2}} = 1$$

$$i = 2\log_2 a_o + D$$

Q.E.D.

Since we would like a faster algorithm than the Euclidean algorithm, this bound is not enough. It must be shown that for some inputs $O(n)$ time is necessary. The proof is given by the construction of a "hard" example of GCD (for the Euclidean algorithm). Integers whose ratio closely approximate the golden mean are hard examples of GCD. If the ratio of integers were exactly the golden mean, then each iteration would produce two numbers whose ratio was again the golden mean. In this way the magnitude of the a_i goes as τ^{-i}, where τ is the golden mean. It is possible to bound this decay from below by an exponential, and it therefore represents a limiting case for the construction of the previous theorem.

Can such good approximations be found? It will be shown that given an original error ϵ in the approximation, the error will grow as $F_i\,\epsilon$ where F_i is the Fibonacci sequence. Although this means that the error grows exponentially large, the algorithm's run is logarithmically short, therefore only a reasonable approximation is required. One can create, in fact, extremely good approximations to τ using the theory of continued fractions.

Lemma. *Let a_o, a_1 be integers such that $a_1 = \tau^{-1}a_o - \epsilon$ for some $\epsilon \in \mathbf{R}^+$, $\quad \epsilon \ll 1$. Then each a_i in the Euclidean algorithm will be $a_o\tau^{-i} + (-1)^i F_i\,\epsilon$.*

Proof. Suppose that a_o, a_1 are taken over the reals, then for what α does $\alpha = a_i/a_{i+1} = a_{i+1}/a_{i+2}$? If $2 > \alpha > 1$ then $a_i \bmod a_{i+1} = a_i - a_{i+1} = a_{i+2}$. Plugging this into the formula for α, cross multiplying, a quadratic results with solutions :

$$\alpha_1, \alpha_2 = \left(\frac{1 \pm \sqrt{5}}{2}\right)^{-1} = \tau^{-1}, \tau^{-*}$$

Define $\epsilon_i = a_i - a_o\tau^{-i}$. We must prove :

1. $\epsilon_i = (-1)^i F_i\,\epsilon$.
2. $a_i = a_o\tau^{-i} + \epsilon_i$.
3. $a_i \in \mathbf{Z}$. The proof is by induction :

Basis

$i = 1, \qquad a_1 = \dfrac{a_o}{\tau} - \epsilon \qquad a_1 \in \mathbf{Z}$ (by definition)

$i = 2, \qquad a_2 = a_o - a_1 = a_o(1 - \dfrac{1}{\tau}) + \epsilon = \dfrac{a_o}{\tau^2} + \epsilon \qquad a_2 \in \mathbf{Z}$

Induction. For $k = i-2, i-1$, $a_k = a_o\tau^{-k} + (-1)^k F_k\,\epsilon$. Then :

$$\begin{aligned} a_i = a_{i-2} - a_{i-1} &= a_o\tau^{2-i} - a_o\tau^{1-i} + (-1)^{i-2}F_{i-2}\,\epsilon - (-1)^{i-1}F_{i-1}\,\epsilon \\ &= a_o\tau^{2-i}(1 - \tau^{-1}) + (-1)^{i-2}(F_{i-2} + F_{i-i})\,\epsilon \\ &= a_o\tau^{-i} + (-1)^i F_i\,\epsilon \qquad a_i \in \mathbf{Z} \end{aligned}$$

Lemma. *If*

$$|\epsilon_i| < a_{i-1}\left(\frac{1}{\tau} - \frac{1}{2}\right)$$

then $a_{i+1} \neq 0$.

Proof. If $a_{i-1} > a_i > a_{i-1}/2$ then $a_{i-1} \bmod a_i = a_{i-1} - a_i \neq 0$. Since $a_i = a_{i-1}\tau^{-1} + \epsilon_i$ then

$$a_{i-1} - \frac{a_{i-1}}{\tau} - \epsilon_i > 0 \text{ and } \frac{a_{i-1}}{\tau} + \epsilon_i - \frac{a_{i-1}}{2} > 0$$

which implies

$$a_{i-1}\left(\frac{1}{2} - \frac{1}{\tau}\right) < \epsilon_i < \frac{a_{i-1}}{\tau^2}$$

Noting that $1/2 - 1\tau < 0$, and that $-(1/2 - 1/\tau) = 1/2 - 1/\tau^2 < 1/\tau^2$, the result follows.

Lemma. *The run time of the Euclidean Algorithm is logarithmic in the ratio* a_o/ϵ.

Proof. What is required is the smallest i for which :

$$|\epsilon| \geq a_{i-1}\left(\frac{1}{\tau} - \frac{1}{2}\right)$$

From above

$$a_{i-1} = a_o\tau^{-(i-1)} + (-1)^{i-1}F_{i-1}\,\epsilon$$

$$\epsilon_i = (-1)^i F_i\,\epsilon$$

So that the smallest i s.t. :

$$|\,(-1)^i F_i\,\epsilon\,| \geq \left(a_o\tau^{-(i-1)} + (-1)^{i-1}F_{i-1}\,\epsilon\right)\left(\frac{1}{\tau} - \frac{1}{2}\right)$$

is what is desired. Note that the initial conditions $F_1 = F_2 = 1$ give the following solution for the Fibonacci sequence :

$$F_i = A\tau^i + B\tau^{*i} \qquad \text{where } A = \frac{\tau}{\sqrt{5}},\ B = \frac{\tau^*}{\sqrt{5}}$$

w.l.o.g. assume i to be even (if it is not, add the appropriate "a_{-1}" to make it so).

$$F_i\,\epsilon \geq \left(a_o\tau^{-(i-1)} - F_{i-1}\,\epsilon\right)\left(\frac{1}{\tau} - \frac{1}{2}\right)$$

$$(TF_i + F_{i-1})\,\epsilon \geq a_o\tau^{-(i-1)} \qquad \text{where } T = \left(\frac{1}{\tau} - \frac{1}{2}\right)^{-1}$$

$$\tau^{i-1}(TA\tau + A) + \tau^{*(i-1)}(TB\tau^* + B) \geq \frac{a_o}{\epsilon}\tau^{-(i-1)}$$

Since i is even and τ^* is negative, $\tau^{*(i-1)} < 0$. Directly calculating, $TB\tau^* + B > 0$, therefore the second term above is negative. Therefore, the least i such that this holds is greater than or equal to the least k s.t. :

$$\tau^{k-1}(TA\tau + A) \geq \frac{a_o}{\epsilon}\tau^{-(k-1)}$$

$$\tau^{2k-2} \geq \frac{a_o}{\epsilon A}\,\frac{1}{T\tau + 1}$$

$$k \geq \frac{1}{2}\log_\tau\left(\frac{a_o}{\epsilon A}\,\frac{1}{T\tau + 1}\right) + 1$$

$$\geq C\log_\tau\frac{a_o}{\epsilon} + 1$$

Theorem. *Given a set* $\{a_{oi} \,|\, a_{oi} \in \mathbf{Z}\}$, *let* $\{a_{1i} \,|\, a_{1i} \in \mathbf{Z},\ \frac{a_{oi}-1}{a_{1i}} < \tau \leq \frac{a_{oi}}{a_{1i}}\}$. *Then the sequence of GCD's* $\{(a_{oi}, a_{1i})\}$ *requires time* $O(\log a_o)$ *for the Euclidean algorithm.*

Proof. The selection of a_{1i} gives the best choice for approximating τ by a_{oi}/a_{1i}. By the choice of equalities, a_{oi}/a_{1i} underestimates τ. Therefore $0 > \epsilon > -1$. So ϵ can be considered fixed in the previous Lemma, and $C \log_{\mathcal{T}}(a_o/\epsilon) = C' \log_{\mathcal{T}} a_o$.

4.1. Alternative Proof of the Existence of $O(n)$ Hard Examples of GCD for the Euclidean Algorithm.

The Fibonacci series :

$$1, 1, 2, 3, 5, 8, 13, \ldots$$

has terms whose ratio $a_{i+1}/a_i \to \tau$ as $i \to \infty$, where $\tau = \frac{1+\sqrt{5}}{2}$, $\quad \tau^* = \frac{1-\sqrt{5}}{2}$.

And in fact a_{i+1}/a_i is the best approximation of τ by integers of the same order of magnitude of a_i or a_{i+1}. For example, $987/610 = 1.618032787$ are two consecutive terms of the series, and are accurate to -1.2×10^{-6}. Since $\frac{1}{610} = 1.6 \times 10^{-3}$, this approximation to τ is much better than it has a right to be.

Except that the F_1 term is 1, the series run in reverse is the Euclidean Algorithm. The run time of the Euclidean algorithm can be thereby lower bounded for input (F_{i+1}, F_i).

$$F_i = \frac{\tau^{i+1}}{\sqrt{5}} - \frac{\tau^{*i+1}}{\sqrt{5}} = a_o$$

so

$$\log a_o \sqrt{5} = \log(\tau^{i+1} - \tau^{*i+1}) < \log 2\tau^{i+1}$$

This follows from the following observation : $\tau^{*i+1} = (\tau^{i+1})^*$, so the difference looks like as in figure 6. Therefore,

$$\log_\tau \left(\frac{a_o + \sqrt{5}}{2} \right) - 1 < i$$

$$i = \Omega(\log a_o)$$

Aho, A. V., John Hopcroft, Jeffrey Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, Mass., 1974.

Beame, Paul W., Stephen A. Cook, H. James Hoover, Log depth circuits for division and related problems, *FOCS 1984*, pp. 1–6.

Berkowitz, Stuart J., On computing the determinant in small parallel time using a small number of processors, *Information Processing Letters 18*, (1984), pp. 147–150.

Boolos, George S., Richard C. Jeffery,*Computability and Logic*, Cambridge University Press, Cambridge, 1980.

Borodin, A. B., On relating time and space to size and depth, *SIAM J. Comp.,* Dec 1977, pp. 733–744.

Borodin, Allan, Joachim von zur Gathen, John Hopcroft, Fast parallel matrix and GCD computations, *Proc. 23ird IEEE Symp. on Foundations of Computer Science*, Chicago, Ill., Nov. 1982, pp. 65– 71.

Brent, R., On the addition of binary numbers, *IEEE Tran. Comput. C-19*,8 (1970), pp. 758–759.

Chandra, Ashok K., Dexter C. Kozen, Larry J. Stockmeyer, Alternation, *Journal of the ACM*, Vol. 28, No. 1, January 1981, pp. 114-133.

Chandra, A. K., Larry J. Stockmeyer, Alternation,*Proc. 17th IEEE Symp. on Foundations of Computer Science*, Houston, Texas, 1976, pp. 98–100.

Chandra, Ashok K., Larry J. Stockmeyer, Uzi Vishkin, A complexity theory for unbounded fan-in parallelism, *Proc. 23ird IEEE Symp. on Foundations of Computer Science*, Chicago, Ill., Nov. 1982(82), pp. 1–13.

Chor, Benny, Oded Goldreich, An improved parallel algorithm for integer GCD, *MIT Laboratory for Computer Science*, April 29, 1985.

Cook, Stephen A., The classification of problems which have fast parallel algorithms.

Cook, Stephen A., H. James Hoover, A depth-universal circuit, *SIAM Journal on Computing*, Vol. 14, No. 4, November 1985, pp. 833–839.

Csanky, L., Fast parallel matrix inversion algorithms, *SIAM J. Computing 5*, 1976, pp. 618–623.

Furst, Merrick, James B. Saxe, Michael Sipser, Parity, circuits, and the polynomial time hierarchy,*Proc. 22nd IEEE Symp. on Foundations of Computer Science*, 1981, pp. 260–270.

Galil, Zvi, Wolfgang J. Paul, An efficient general purpose parallel computer, *STOC 1981*, pp. 247–262.

Garey, M.R., D. S. Johnson,*Computers and Intractability, A Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco, 1979.

Halmos, Paul R.,*Naive Set Theory*, Springer-Verlag, New York, 1974.

Hopcroft, John E., Jeffery D. Ullman,*Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley Publishing Company, Reading Mass., 1979.

Kannan, Ravindran, Gary Miller, Larry Rudolph, Sublinear parallel algorithm for computing the greatest common divisor of two integers, *IEEE FOCS 1984*, pp. 7–11.

Kozen, Dexter, On Parallelism in Turing Machines,*Proc. 17th IEEE Symp. on Foundations of Computer Science*, Houston, Texas, 1976, pp. 89–97.

Krapchenko, V.M., Asymptotic estimation of addition time of a parallel adder, *Syst. Theory Res. 19*, 1970, pp. 105–122 [*Probl. Kibern. 19*, pp. 107–122 (Russ.)].

Ladner, Richard E., Michael J. Fischer, Parallel Prefix Computation, *Journal ACM*, Vol. 27, No. 4, Oct 1980, pp. 831–838.

Mead, Carver, Lynn Conway, *Introduction to VLSI Systems*, Addison-Wesley, Reading, Mass, 1980.

Ofman, Yu, On the algorithmic complexity of discrete functions, *Sov. Phys. Dokl.* 7(1963), pp. 589–591.

Pippenger, Nicholas, On Simultaneous Resource Bounds, *Proc. 20th IEEE Symp. on Foundations of Computer Science*, Puerto Rico, October 1979, pp. 307–311.

Preparata, Franco P., Jean Vuillemin, The cube-connected-cycles: A versatile network for parallel computation,*Proc. 20th IEEE Symp. on Foundations of Computer Science*, Puerto Rico, October 1979, pp.140- -147.

Reif, John, Logarithmic depth circuits for algebraic functions, *24th IEEE FOCS*, 1983, pp. 138–145.

Rudin, Walter, *Principles of Mathematical Analysis*, McGraw- Hill Book Company, New York, 1976.

Savage, J. E., *The Complexity of Computing*, Wiley, New York, 1976.

Stark, Harold M., *An Introduction to Number Theory*, The MIT Press, Cambridge, Mass., 1984.

Weil, André, *Number Theory for Beginners*, Springer- Verlag, New York, 1979.