PARALLEL ROOTFINDING ALGORITHMS

Dissertation for the Degree of Ph. D.
MICHIGAN STATE UNIVERSITY
GEORGE FREDERICK CORLISS
1974

This is to certify that the

thesis entitled

PARALLEL ROOTFINDING ALGORITHMS

presented by

George Frederick Corliss

has been accepted towards fulfillment
of the requirements for

Ph.D. degree in Mathematics

Gerald D. Taylor
Major professor

Date August 8, 1974

O-7639

f ar

type

The a

Lagra

conta

contr

Lagra

alway

conve

k ca

requ

this

up p

Lagra

algo

ABSTRACT

PARALLEL ROOTFINDING ALGORITHMS

By

George Frederick Corliss

Algorithms to find a simple root α of a continuous function f are considered which are suitable for implementation on an array type of parallel processing computer using N processing elements. The algorithms are compared on the basis of their efficiency

$$EFF = \frac{\log_2 \text{ order of convergence}}{\text{number of parallel arithmetic operations per iteration}} .$$

The first class of algorithms to be considered uses inverse Lagrange interpolation on k points. If $f \in C^k$ on an interval containing α, these methods have order of convergence k. In contrast, the order of convergence for sequential methods using Lagrange interpolation on several previous estimates for α is always less than 2. The parallel methods achieve a higher order of convergence because they use only the most recent estimates for α. k can be chosen, depending on the number of arithmetic operations required to evaluate f, to maximize the efficiency of methods in this class. If the cost of evaluating f is very high, the speed-up possible by using parallel rootfinding algorithms based on Lagrange interpolation over efficient sequential rootfinding algorithms is proportional to $\log_2 N$.

A

example

implement

cessing e

evaluatio

it is mor

more than

The

verse Her

order of

If r is

effective

processing

$\log_2 N$ fo

less costl

a method

efficient

points.

The

estimated

Hermite in

$f,f',\ldots,f$

Hermite in

rare examp

efficient t

based.

A parallel secant rootfinding algorithm is considered as an example of a method using Lagrange interpolation. If it is implemented on a parallel processing system with three or more processing elements, it achieves order of convergence 2 with one parallel evaluation of f and five additional arithmetic operations. Hence it is more efficient than the usual Newton-Raphson method whenever more than three arithmetic operations are required to evaluate f'.

The second class of parallel algorithms considered uses inverse Hermite interpolation of $f,f',\ldots,f^{(r)}$ at k points. The order of convergence is equal to the amount of data used, $k(r+1)$. If r is fixed, the fastest algorithm in this class which can be effectively implemented on a parallel processing system with N processing elements achieves a speed-up ratio proportional to $\log_2 N$ for functions which are very costly to evaluate. If it is less costly to evaluate f than to evaluate any derivative of f, a method using Lagrange interpolation ($r = 0$) on k points is more efficient than any other method using Hermite interpolation on k points.

The third class of parallel algorithms contains derivative-estimated methods. If the values of $f^{(r)}$ used in a parallel Hermite interpolation method are estimated using values of $f,f',\ldots,f^{(r-1)}$ at enough points, the order of convergence of the Hermite interpolation method is not reduced. However, except in rare examples, a parallel derivative-estimated method is less efficient than the Hermite interpolation method on which it is based.

as pos

algori

invers

depend

the ta

speed-

rootfi

the pa

If the answer to a rootfinding problem is desired as quickly as possible, as in real-time computer applications, these parallel algorithms should be used. In general, a parallel method using inverse Lagrange interpolation on k points is fastest, where k depends on the function. If the root is to be computed at low cost, the task should be executed on a sequential machine because the speed-up ratio proportional to $\log_2 N$ achieved by these parallel rootfinding algorithms shows that they do not make efficient use of the parallel capabilities of the system.

PARALLEL ROOTFINDING ALGORITHMS

By

George Frederick Corliss

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Department of Mathematics

1974

Gerald D

tions an

ACKNOWLEDGMENTS

I wish to express my sincere appreciation to Professor Gerald D. Taylor for his leadership and guidance. His suggestions and assistance have been invaluable.

Chapter

I

II

TABLE OF CONTENTS

Chapte

III

IV

Table

2-1

LIST OF TABLES

Figure

1-1

1-2

1-3

1-4

1-5

1-6

1-7

1-8

1-9

2-1

2-2

2-3

3-1

LIST OF FIGURES

INTR

Th

ware conf

reliabili

users. P

considere

processir

minutes,

way to d

construc

effectiv

paper wi

which us

1.1 Mot

systems

4-7). S

detail a

CHAPTER I

INTRODUCTION TO PARALLEL PROCESSING AND ROOTFINDING

The computer industry is currently searching for new hardware configurations which will attain greater speeds with a higher reliability and more flexibility to meet the varying needs of many users. Parallel processing is one very promising approach being considered. The intuitive motivation for research into parallel processing is that if one machine can do a certain job in ten minutes, then perhaps ten machines could be coupled together in some way to do the same job in one minute. In addition to the design and construction of such machines, algorithms must be developed which effectively utilize the parallel capabilities of the machine. This paper will be devoted to the study of algorithms for rootfinding which use the parallel machine structure.

1.1 Motivation for Parallel Processor Design

There are several reasons why parallel processor computer systems may have a place in future computer designs (see [14], pp. 4-7). Some areas of possible benefit which we will discuss in more detail are:

1) High speed computation

2) Very large problems

3) Inherent parallelism in existing problems

4)

5)

6)

We

the use of

1.1.1 Hig

C

that very

expensive

which ele

micro-min

must trav

small tim

being ex

computat

N-fold r

computat

head tim

N diffe

among th

machine

structi

is cons

structi

increas

4) Enormous quantities of data requiring real-time processing

5) Multiuser systems

6) Reliability and graceful degradation.

We now discuss each of these areas of possible benefit from the use of parallel processing computers.

1.1.1 High Speed Computation

Conventional computer hardware design has reached the point that very small improvements in the speed of the machine are very expensive. The speed of the machine is limited by the speed at which electrical pulses travel through the machine. Advances in micro-miniature circuitry have cut down the distances the pulses must travel, and hence the computation times, but the cost of even small time savings is prohibitive, so other avenues of design are being explored. Parallel processing is one such design alternative.

In a parallel processing system, N different parts of a computation may proceed simultaneously, so it is possible that an N-fold reduction in the time needed to perform certain classes of computations could be realized. One would expect the program overhead time to be greater on a parallel processing machine because N different processors must be coordinated and the work shared among them. At the same time, it may be possible to construct a machine with N processing elements at a cost below that of constructing N separate machines. One design in which this is done is considered in Section 1.2. If the savings possible in the construction of the parallel processing machine is larger than the increased cost of the overhead in program execution, then the N-

fold speed-

on existing

promise tha

cost than i

time curren

1.1.2 Very

So

so much co

only at gr

a global w

dynamics p

and other

of data po

achieve a

feasible t

1.1.3 Inh

A

configurat

be perfor

on a para

example,

although

ferent fo

vector op

even thou

fold speed-up can be achieved at a lower total cost than execution on existing machines. That is, parallel processing designs hold the promise that it may be possible to execute some problems at a lower cost than is currently possible and to execute them in 1/N of the time currently required.

1.1.2 Very Large Problems

Some problems of current interest require so much memory or so much computation that they can be solved on conventional machines only at great expense. Some examples are the problem of constructing a global weather model, some nuclear physics problems, large hydrodynamics problems, phased array radar air defense systems (see [1]), and other similar problems requiring computations on a large number of data points. If a parallel processing system can be built to achieve a high speed-up at a relatively low cost, it will become feasible to solve problems too large for existing computer systems.

1.1.3 Inherent Parallelism in Existing Problems

A large class of problems are well suited to a parallel configuration by their structure. If the same computations are to be performed on many independent sets of data, the job may be executed on a parallel processing system in a highly efficient manner. For example, a payroll job must do the same computations for each employee, although the rate of pay, hours worked, withholding, etc., are different for each employee. Tasks of this sort which are essentially vector operations may be economical to run on a parallel machine even though they do not require a very high speed operation.

1.1

whi

dec

con

ila

wo

an

ar

fe

pr

us

de

wi

ti

wh

ta

fr

ir

a

ar

th

ca

th

se

ce

1.1.4 Enormous Quantities of Data Requiring Real-Time Processing

Historically, each time a new computer has been introduced which was significantly faster or had more memory than its predecessors, it became feasible to solve problems which had been too complex or too large for smaller, slower machines to handle. Similarly, the design and construction of a parallel processing system would enable problems which require vast amounts of data to be read and analyzed fast enough to respond in real time. Two such problems are the control of a large phased array radar system for urban defense (see [1], [12], and [8]), or speech and visual perception problems in artificial intelligence (see [31]).

A system to control a large air defense radar network must use the information from several different radar installations to determine the nature, position, and motion of each airborn object within the range of the network. Then appropriate radar installations are directed to conduct a further study of each "target", while continuing a general search by the entire network for other targets. Targets whose identity is known must be distinguished from incoming enemy aircraft and from decoys deployed by the attacking enemy to fool the air defense system. Then the appropriate air defense mechanisms must be alerted. All of this data must be analyzed and action taken before the attacking aircraft arrive at their targets. In [1], Knapp, Ackins, and Thomas describe a system capable of tracking 10^9 targets simultaneously. This requires the controlling computer system to issue about 10^7 commands per second to the radar units. Currently existing computers cannot process that much data fast enough to respond in the time required.

In the second example, human speech and vision require the processing of a large amount of data at a very high rate. For example, when a person is speaking, his listener must detect the sound of his voice, interpret his speech, and formulate a response, all during the time the speaker is talking. Then the listener is able to respond so that a conversation is possible. If a robot is to recognize human speech patterns, it must be able to process one second of speech in one second. Reddy [31] estimates that a robot must be able to process about 2×10^5 bits per second of information to detect, analyze, and recognize speech patterns. If a robot is mobile, it must be able to "see" obstacles in its path in time to avoid them. Reddy estimated that the robot must be able to process approximately 10^8 bits of data per second for this visual recognition task.

Conventional computers are not capable of processing data at the rate required by either of the two examples given above, while a large parallel processing system could be designed to meet these demands for speed and capacity.

1.1.5 Multiuser Systems

It is becoming increasingly common for one computer system to serve many users. Various time-sharing schemes allow work to proceed on different programs at once. A parallel processing computer may be able to perform a similar function by allocating each user to an idle processor without the expensive overhead of the current operating systems.

1.1.6 Rel

A

of the ma

Reliabilit

ponent or

go gracef

roughly i

In a conv

forces th

of the sy

failures

cessing e

assume th

cessing e

only slig

The use o

icant dr

That is,

gradual w

reliabili

conventio

systems p

process c

1.2 Para

T

processin

In 1966,

1.1.6 Reliability and Graceful Degradation

A parallel processor system may be able to withstand failures of the machine's components without a major interruption of service . Reliability is a measure of the probability of the failure of a component or of the entire machine. A computer system is said to undergo graceful degradation if the operation of the machine deteriorates roughly in proportion to the percentage of malfunctioning components. In a conventional computer system, the failure of a major component forces the entire system out of operation. If the major functions of the system are decentralized in a parallel configuration, some failures need not be fatal. For example, if one of the many processing elements failed, the remaining processors could automatically assume the tasks of the disabled one. Hence the failure of a processing element which would be fatal to a conventional machine would only slightly lower the efficiency of a parallel processing system. The use of several processing elements could be lost before a significant drop in the overall performance of the machine would occur. That is, the performance of a parallel system would deteriorate in a gradual way. This graceful degradation would make the overall reliability of such a system much better than the reliability of conventional machines. The high reliability makes parallel processor systems particularly important in such real-time applications as process control or military surveillance.

1.2 Parallel Machine Organization

There are many different alternative designs for parallel processing computers to fulfill the needs discussed in Section 1.1. In 1966, Flynn [11] and [10] described four classes of machine

organizat

different

four clas

sider the

stream-si

ized by

Each inst

carried

data str

proceed

impracti

essary t

instruct

in this

is carri

designs

discusse

stream-m

are bank

work on

gram, al

Carnegie

system,

organization. Parallel processing systems were assigned to two different classes. We will give a brief description of each of the four classes of machine organization considered by Flynn, and consider the basic design of two important parallel processing systems.

Flynn classified conventional computers as single instruction stream-single data stream (SISD) machines since they are characterized by one sequence of instructions acting on one set of data. Each instruction is executed in sequence and the instruction is carried out on only one set of data.

Some early computers were multiple instruction stream-single data stream (MISD) machines. Several sets of instructions could proceed simultaneously on one set of data. MISD machines were impractical because of the hardware and programming overhead necessary to avoid memory conflicts.

A large class of parallel processing machines are single instruction stream-multiple data stream (SIMD) machines. Machines in this class execute instructions in sequence, but each instruction is carried out on several different sets of data. Two important designs for SIMD machines, the ILLIAC IV and the CDC STAR, will be discussed later in this section.

Flynn's fourth class of machine is a multiple instruction stream-multiple data stream (MIMD) machine. These multi-processors are banks of autonomous machines. Each machine can be directed to work on a different program, or on different parts of the same program, all under the overall control of a central control unit. Carnegie-Mellon University is designing a multi-mini-processor system, C.mmp [42]. A MIMD machine is well suited to multi-user

systems,

blocks wh

T

for use c

such mach

designed

for the N

Moffet Fi

I

instructi

cessing e

unit with

memory or

proceeds

executes

Data may

between I

to act or

is an exa

suited t

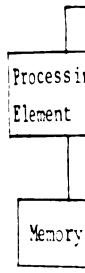
Danenberg

of ILLIAC

systems, or to large problems where the program can be split up into blocks which do not depend on data from the other blocks.

The algorithms for rootfinding in this paper are intended for use on a SIMD machine, so we will consider the design of two such machines. The first is ILLIAC IV, an array processing system designed by the University of Illinois and Burroughs Corporation for the National Aeronautics and Space Administration Laboratory at Moffet Field, California.

ILLIAC IV is composed of one control unit (CU) which generates instruction signals to control the operation of 64 identical processing elements (PE). Each PE consists of an arithmetic and logic unit with its own memory. Each PE processes the data in its own memory or data which is passed to it through the CU. Computation proceeds according to the commands generated by the CU, so each PE executes the same instruction stream unless it has been turned off. Data may be passed between the CU and the PE's, or it may be passed between PE's. This configuration allows one sequence of instructions to act on up to 64 different sets of data simultaneously, so ILLIAC IV is an example of a SIMD machine. This machine organization is well suited to a large class of vector or array type problems. In [9], Danenberg gives a detailed description of the design and organization of ILLIAC IV (see also [22] or [31]).



Fig

A

ciently o

or equal

points, f

64 proces

used. Fo

store and

PE's stor

cessing 6

cessed f

the compu

such as

the comp

so that

is a pipe

systems

can somet

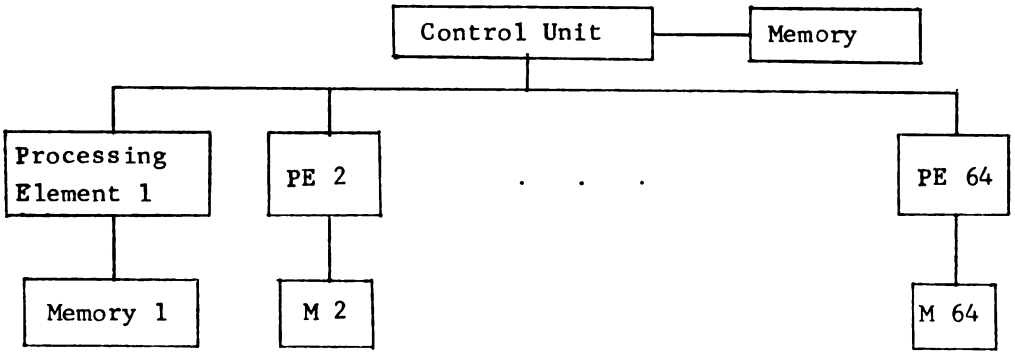


Figure 1-1. Functional Block Diagram for ILLIAC IV

An array processing system like ILLIAC IV operates efficiently only when the size of the data vector is slightly less than or equal to the size of the machine. 63, 64, 190, or 192 data points, for instance, are well suited to processing on ILLIAC IV's 64 processing elements since all, or nearly all, of the PE's are used. For example, if a 190-dimensional vector is used, 62 PE's store and operate on 3 components each, while the two remaining PE's store and operate on 2 components each. By contrast, processing 65-dimensional vectors requires that 64 components be processed first. Then all but one of the PE's must be disabled while the computations proceed on the "extra" component. In many problems such as the numerical solution to partial differential equations, the computations can be structured to use a multiple of 64 points so that ILLIAC IV's 64 PE's can be used efficiently.

The second example of a SIMD machine which we will consider is a pipeline processor, the CDC STAR. While array processing systems like ILLIAC IV are based on the notion that N machines can sometimes do the work of one machine in 1/N of the time,

pipeline computers like STAR gain speed by performing different parts of the same operation simultaneously (see [9] or [13]). For example, suppose the operation of addition is broken into these six sub-operations: 1) fetch operands from memory, 2) adjust exponents to align the decimal points, 3) add mantissas, 4) normalize, 5) round off the result, and 6) store the sum in memory. These six sub-operations are illustrated by a simple example in Figure 1-2. If this addition operation is implemented on a pipeline computer, the sum of two numbers is being stored in memory at the same time that the sum of the next two numbers is being rounded off, etc. Figure 1-3 shows how addition of two N-dimensional vectors would proceed in such a machine.

Sub-operations	
1) Fetch operands	$A = 0.234 \times 10^3$ $B = 0.567 \times 10^1$
2) Adjust exponents	$A = 23.4 \times 10^1$ $B = 0.567 \times 10^1$
3) Add mantissas	$A + B = 23.967 \times 10^1$
4) Normalize	$A + B = 0.23967 \times 10^3$
5) Round off	$A + B = 0.240 \times 10^3$
6) Store the sum	$A + B = 0.240 \times 10^3$

Figure 1-2. Addition with Six Sub-operations

1

2

.

.

.

6

.

Step .

.

N

N+1

.

.

.

N+5

Figure 1

W

pipeline

pipeline

which are

illustrat

same sequ

data.

A

length.

are separ

pipeline

needed to

Because of

Step	Sub-operation 1	2	. . .	6
1	(A_1, B_1)			
2	(A_2, B_2)	(A_1, B_1)		
.				
.				
.				
6	(A_6, B_6)	(A_5, B_5)	. . .	$A_1 + B_1$
.				
.				
.				
N	(A_N, B_N)	(A_{N-1}, B_{N-1})	. . .	$A_{N-5} + B_{N-5}$
N+1		(A_N, B_N)	. . .	$A_{N-4} + B_{N-4}$
.				
.				
.				
N+5				$A_N + B_N$

Figure 1-3. Pipeline Addition of N-dimensional Vectors A + B

We have seen that a single sequence of instructions in a pipeline machine acts on several different sets of data. Hence the pipeline processor is an example of a SIMD machine. Computations which are much more involved than the operation of addition illustrated above can be implemented on a pipeline computer if the same sequence of computations must be repeated on different sets of data.

A pipeline computer can operate on a vector of arbitrary length. Figure 1-3 shows that if the computations being programmed are separated into six parts, five steps are required to fill the pipeline when starting the computations. Similarly, five steps are needed to complete processing of the last component of the vector. Because of the time spent filling the pipeline at the beginning and

emptying it at the conclusion of an operation, the efficiency of the machine decreases if it is necessary to empty and refill the pipeline often. In actual use, the start-up time is typically the time needed to generate 30 - 60 results once the pipeline is full [28], so longer vectors use the machine more efficiently. The vector length is limited to 64,000 by the temporary memory available on the CDC STAR, and 1000-dimensional vectors are typical of efficient machine utilization [28].

There is some overlap among the classes of machines discussed in this section. For example, the array processing system ILLIAC IV uses pipelined instructions and memory access queues to speed execution [9]. As originally designed, the ILLIAC IV system was to be a MIMD machine composed of four autonomous SIMD machines. Each of four control units was to drive 64 processing elements, but only one quadrant of the proposed machine was built.

1.3 Problems Encountered when Implementing Parallel Algorithms

After a parallel processing system has been designed, there are at least four major problem areas which must be studied before the system can be used efficiently. The problems discussed in [37] by Stone are:

1. The arrangement of data in memory is essential for efficient parallel processing, although it is only a minor concern for the conventional sequential machines now in use.
2. An efficient sequential algorithm does not necessarily lead to an efficient parallel algorithm. Hence research is necessary to develop efficient parallel algorithms.

3. M

seque

4. T

etc.,

the c

a roo

influ

roofi

1.4 E

In [15

for fi

of the

Stone

doubli

proble

transf

into a

a surv

value,

velope

efficie

Probabl

3. Many algorithms are inherently sequential, but in some cases the sequential dependence can be relaxed.

4. The behavior such as the order of convergence, round-off error, etc., of a parallel algorithm may be quite different from that of the corresponding sequential algorithm.

In this paper we will consider algorithms designed to find a root of a function. We will see how each of these problem areas influences the construction, the behavior, and the choice of efficient rootfinding algorithms.

1.4 Examples of Parallel Processing Algorithms

There are many algorithms known which are parallel in nature. In [15], Karp and Miranker describe a parallel searching algorithm for finding roots of a function. Pease [29] gives a modification of the fast Fourier transform which is suited to parallel processing. Stone [35] and Kogge and Stone [16] derive a method called recursive doubling which allows them to solve a large class of recurrence problems using parallel processing. The method of recursive doubling transforms a problem which seems to be inherently sequential in nature into a problem which can be solved in parallel. Miranker [24] gives a survey of many other parallel algorithms.

The pull-back interpolation technique for solving initial value, boundary value, and time delay differential equations developed by Rogers [34] incorporates several features which could be efficiently implemented on a parallel processing computer.

We now consider some parallel algorithms in more detail. Probably the simplest use of parallel processing is the addition

of two N-dimensional vectors, A + B. In a pipeline processor like the CDC STAR, addition is broken down into several sub-operations. Figure 1-3 illustrates pipeline addition with six sub-operations. The summands move through the pipeline of sub-operations at the speed of the slowest sub-operation.

In an array processing system like ILLIAC IV, the i th processing element, PE i, performs $A_i + B_i$. Figure 1-4 illustrates the addition of two 64-dimensional vectors. If $N < 64$ some of the PE's are turned off while the rest execute the addition. If $N > 64$, some PE performs the addition of more than one component.

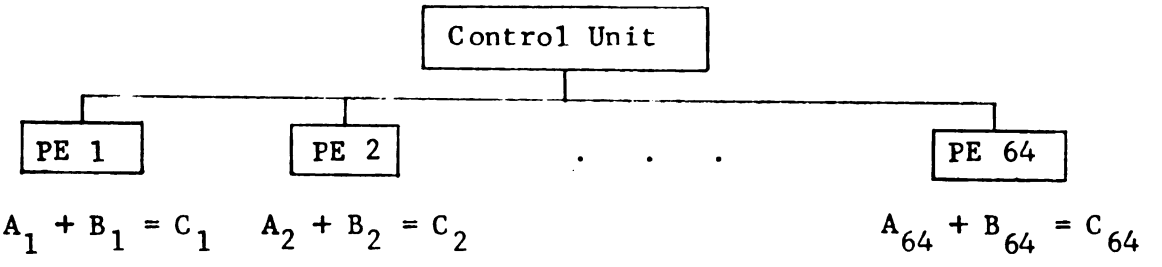


Figure 1-4. Addition of Two 64-dimensional Vectors on ILLIAC IV

An interesting example of the savings in time possible with ILLIAC IV is the computation of $1 + 2 + \ldots + 64$ together with all of the partial sums. This problem requires 63 additions on a sequential machine, but Figure 1-5 shows the sequence of six parallel additions which yields the desired sum and each intermediate partial sum. The instructions indicate the routing of data at each step. If the instructions are changed to allow the data from PE 64 to be passed directly around to PE 1, each PE contains the sum $1 + 2 + \ldots + 64$ after six parallel additions.

PE 1	PE 2	. . .	PE 64
1	2		64
	Route each PE's contents one to the right and add,		
1	1+2	. . .	63+64
	Route contents two right and add,		
1	1+2 1+2+3 1+2+3+4	. . .	61+...+64
	Route contents 4 right and add,		
1	1+2 . . . 1+...+8	. . .	57+...+64
	Route contents 8 right and add,		
1	1+2 . . . 1+...+16	. . .	49+...+64
	Route contents 16 right and add,		
1	1+2 . . . 1+...+32	. . .	33+...+64
	Route contents 32 right and add,		
1	1+2 1+2+3		1+2+...+64

Figure 1-5. 1+2+...+64 on ILLIAC IV

Another type of problem which is well suited to a parallel processing system is the numerical solution of partial differential equations. For example, the point Jacobi iteration to solve the Laplace equation by finite differences with a uniform mesh on a square is given by

$$u_{n+1}(i,j) = \frac{u_n(i,j-1) + u_n(i-1,j) + u_n(i,j+1) + u_n(i+1,j)}{4} .$$

Here the average of the previously computed approximation to the true solution at the four nearest points is used as the new approximation. Since no u_{n+1} uses previously computed values of u_{n+1} at

other points, it is possible to compute u_{n+1} at each point simultaneously on a parallel machine.

In contrast to the point Jacobi method, the Gauss-Seidel iteration uses new approximations for u_{n+1} which have already been computed whenever they are available, so the Gauss-Seidel iteration is inherently sequential. Carrol and Wetherford [6] suggest a variation of the usual Gauss-Seidel method which can be implemented on a parallel machine with N processing elements. Assume $\sqrt{N}$ is an integer. If a $2\sqrt{N} \times 2\sqrt{N}$ grid is used, the solution must be computed on $4N$ points. Hence each processor must compute the approximation at 4 points. Figure 1-6 shows that each processor computes the value at the first point using old data; the values for the second and third points are computed using some new and some old data; the value at the fourth point is computed using only new data. Another variation of the Gauss-Seidel method is due to Stone [37]. He suggests that an $N \times N$ grid be used. Then diagonals of length N are processed simultaneously in the manner shown in Figure 1-7.

As a final example of parallel algorithms we consider a parallel rootfinding algorithm. This algorithm is a special case of the simultaneous $n-2$ degree method due to Rice [32] with $n = 3$. A summary of Rice's results and a more careful discussion of this method will be given in Chapter II.

This algorithm will be called the parallel secant method because it is derived from the usual secant method given by many authors (see [30, pp. 323-328], or [27, Chapter 3]). The secant method is given by

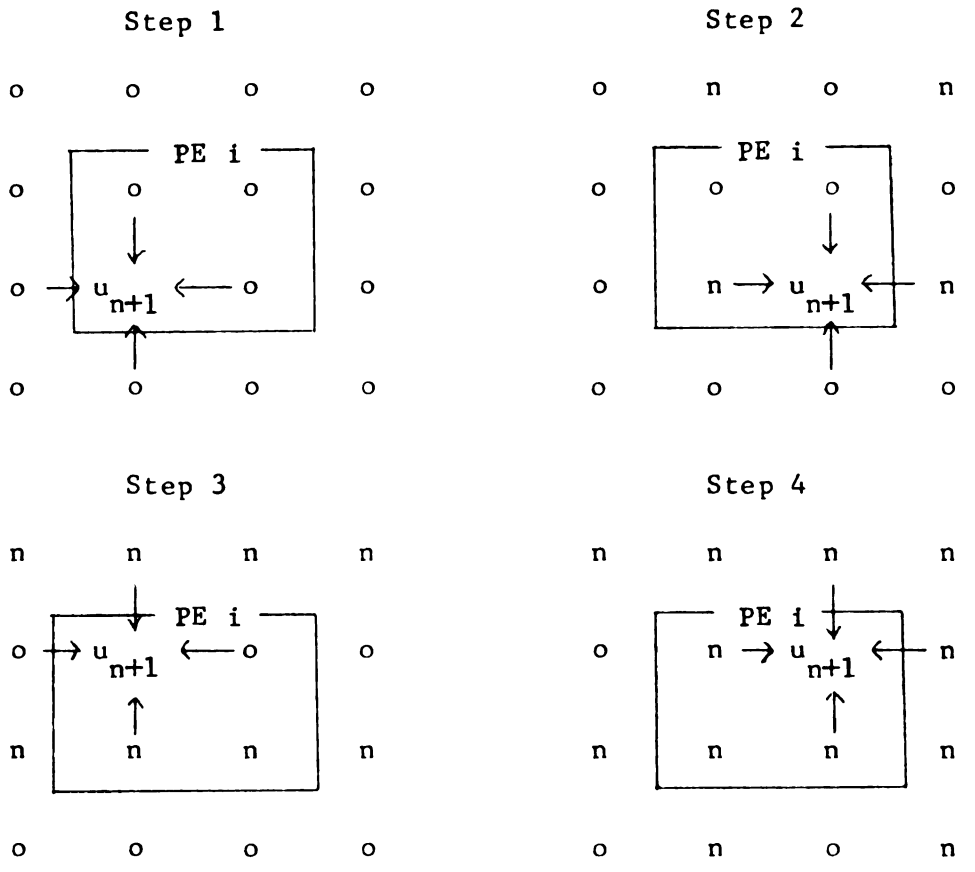


Figure 1-6. Carrol and Wetherford's Variation of Gauss-Seidel

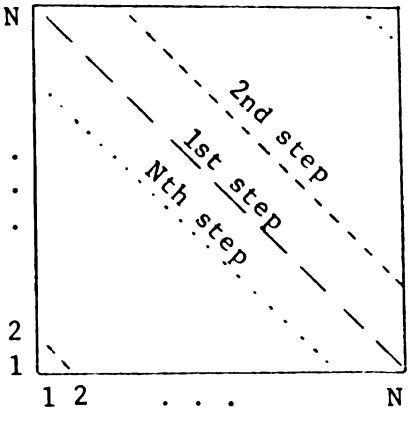


Figure 1-7. Stone's Variation of Gauss-Seidel

$$x_{i+1} = \frac{f(x_i)}{f(x_i)-f(x_{i-1})} x_{i-1} + \frac{f(x_{i-1})}{f(x_{i-1})-f(x_i)} x_i$$

$$= x_i - \frac{x_i - x_{i-1}}{f(x_i)-f(x_{i-1})} f(x_i) \ . \qquad (1\text{-}1)$$

Assume that $f \in C^2$ on some interval containing a simple root α of f. Let x_0 and x_1 be chosen near α. Then the error in equation (1-1) satisfies

$$\alpha - x_{i+1} = \frac{-f''(\xi)f'(\xi_i)f'(\xi_{i-1})}{2[f'(\xi)]^3} (\alpha - x_i)(\alpha - x_{i-1}) \ , \qquad (1\text{-}2)$$

where ξ, ξ_i, and ξ_{i-1} lie on the interval spanned by α, x_i, and x_{i-1}. Then there exists a constant K such that

$$|\alpha - x_{i+1}| \leq K|\alpha - x_i| \ |\alpha - x_{i-1}| \ . \qquad (1\text{-}3)$$

We now generalize the secant method given by (1-1) to a parallel method. Let $x_{0,1}$, $x_{0,2}$, and $x_{0,3}$ be chosen near α. Let $y_{i,j} = f(x_{i,j})$, for $j = 1,2,3$; and $i = 0,1,\ldots$. The parallel secant method defines a sequence of 3-dimensional vectors $\{X_i = (x_{i,1}, x_{i,2}, x_{i,3})\}$ by

$$x_{i+1,1} = x_{i,1} - \frac{x_{i,1} - x_{i,2}}{y_{i,1} - y_{i,2}} y_{i,1}$$

$$x_{i+1,2} = x_{i,2} - \frac{x_{i,2} - x_{i,3}}{y_{i,2} - y_{i,3}} y_{i,2} \qquad (1\text{-}4)$$

$$x_{i+1,3} = x_{i,3} - \frac{x_{i,3} - x_{i,1}}{y_{i,3} - y_{i,1}} y_{i,3} \ .$$

Figure

parall

vious

approx

the er

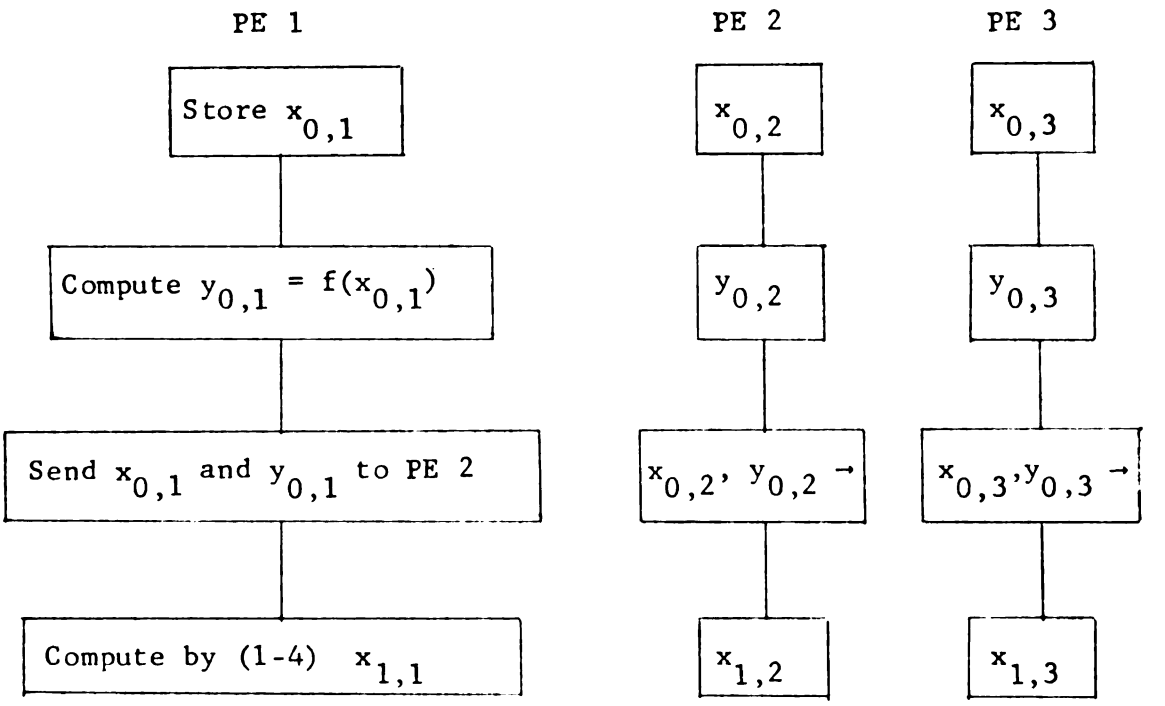


Figure 1-8. Flow Chart for One Iteration of the Parallel Secant Method

As the flow chart in Figure 1-8 shows, each of the three parallel processors uses the approximations generated at the previous step by itself and by another processor to compute a new approximation to α according to the secant method. From (1-3), the errors in (1-4) satisfy

$$\begin{aligned} |\alpha - x_{i+1,1}| &\le K|\alpha - x_{i,1}|\,|\alpha - x_{i,2}| \\ |\alpha - x_{i+1,2}| &\le K|\alpha - x_{i,2}|\,|\alpha - x_{i,3}| \\ |\alpha - x_{i+1,3}| &\le K|\alpha - x_{i,3}|\,|\alpha - x_{i,1}| \quad . \end{aligned} \tag{1-5}$$

If we let $\delta_i = \max\{|\alpha - x_{i,j}|,\ j = 1,2,3\}$, (1-5) becomes

$$\delta_{i+1} \le K(\delta_i)^2 \quad . \tag{1-6}$$

It will

improves

from $\frac{1}{}$

1.5 Spe

elements

N minut

realized

discusse

of the p

impossib

define a

Definitio

N

is being

achieved

time shou

finding a

A

tion in a

Figure 1.

on the se

suppose o

ILLIAC IV

It will be shown in Chapter II that the use of parallel processing improves the order of convergence of the conventional secant method from $\frac{1+\sqrt{5}}{2} \approx 1.6$ to 2.

1.5 Speed-Up Ratio

Ideally, a parallel processing system with N processing elements should be able to execute a job in one minute which requires N minutes to execute on a sequential machine. This ideal is realized for the addition of two 64-dimensional vectors on ILLIAC IV discussed in Section 1.4, but the program overhead and the nature of the problem or algorithm under consideration usually make it imposssible to achieve the ideal savings of time. This leads us to define a measure of the time actually saved by a parallel machine.

Definition 1-1 (see [37, p. 3]). Define the speed-up ratio to be

$$S = \frac{\text{computation time on a sequential computer}}{\text{computation time on a parallel computer}} .$$

Notice that it is necessary to state which sequential algorithm is being used for comparison. When studying the speed-up ratios achieved by different parallel rootfinding algorithms, their execution time should be compared with the execution time of sequential rootfinding algorithms which are optimal in some sense.

As the proportion of commands requiring sequential computation in a program increases, the speed-up ratio falls very quickly. Figure 1-9, due to Amdahl [2], shows how the speed-up ratio depends on the sequential processing inherent in a program. For example, suppose only 1/64 of a job requires sequential computation on ILLIAC IV, while the remaining 63/64 of the job can be executed in

parallel.

execution

sequential

Spee

Rat

Th

processing

computatio

tasks whi

alternativ

automatica

parallel m

ities in a

We

which are a

As

the ideal sp

many problems

a speed-up ra

parallel. The machine's efficiency drops to 1/2 since half of the execution time will be taken by one processor to perform the sequential part of the program while the other 63 PE's are idle.

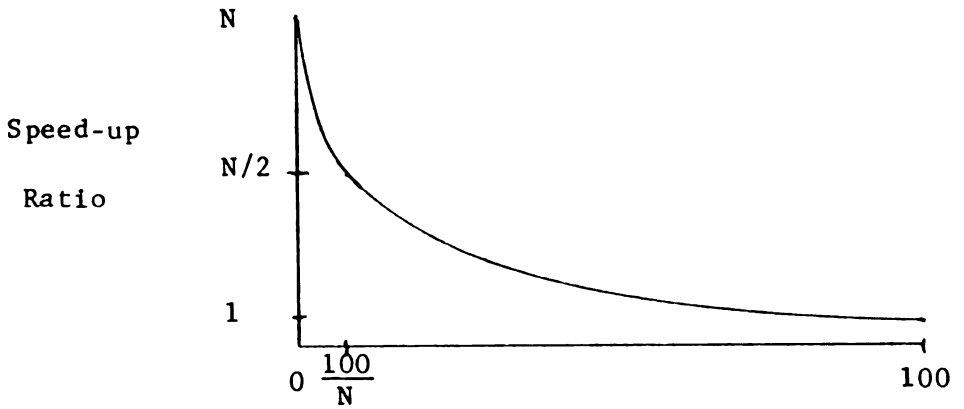


Figure 1-9. Speed-Up Ratio

This example illustrates the importance of using the parallel processing machine only for parallel computations. To avoid sequential computations, the control unit can perform indexing and other overhead tasks while the PE's are executing the body of the program. Another alternative approach is to have a sequential computer available which automatically assumes the execution of sequential tasks while the parallel machine performs tasks which utilize its parallel capabilities in an efficient manner.

We now consider some examples to show the speed-up ratios which are achieved by several different classes of problems.

As the remarks at the beginning of this section point out, the ideal speed-up ratio of N is very rarely attained. However, many problems which have a natural vector or array structure achieve a speed-up ratio which is linear in the number of processors used.

A speed-up r
are very wel
problems inc
numerical so
calculations

Probl
relations (K
and Paterson
speed-up rat
as a linear
parallel pro
by doubling

The
ratio. Alg
[15] achiev
of algorith
little incr
cessors. I
class, they
to compute
large class

The
that the use
These proble
so they shou
processing sy
up ratio.

A speed-up ratio of aN, with $a \leq 1$, characterizes problems which are very well suited to parallel processing. Examples of such problems include matrix computations and problems such as the numerical solution of partial differential equations which use calculations at grid points (see [17]).

Problems such as sorting (Stone in [36]), linear recurrence relations (Kogge and Stone in [16]), and polynomial evaluation (Munro and Paterson in [25]) have been handled on SIMD machines with a speed-up ratio of $aN/\log_2 N$. Although this speed-up is not as good as a linear speed-up, these problems are still well suited to parallel processing since the speed of computation is nearly doubled by doubling the number of processing elements.

The next class of algorithms have an even poorer speed-up ratio. Algorithms such as Karp and Miranker's searching algorithm [15] achieve a speed-up ratio proportional to $\log_2 N$. This class of algorithms is poorly suited to parallel processing since very little increase in speed is achieved by doubling the number of processors. In spite of the poor speed-up ratio of algorithms in this class, they are of use in real-time applications where it is important to compute quickly, regardless of cost. This paper will show that a large class of parallel rootfinding algorithms fall into this class.

The final class of problems are inherently sequential, so that the use of parallel processing yields no increase in speed. These problems have a speed-up ratio which is independent of N, so they should be executed on sequential machines to leave the parallel processing system free to execute algorithms with a favorable speed-up ratio.

1.6 Introduc

This

root α of a

statement of

Let

$f(\alpha) = 0$. A

root of f.

The

general iter

f need only

Other condi

necessary f

The

and has bee

to the fixe

cludes part

boundary va

1.7 Defini

Let

the SIMD ma

α. We cons

N-tuples {

an estimate

be distinct

must have x

form

1.6 Introduction to Iterative Rootfinding

This paper will be concerned with the problem of finding a root α of a real-valued function f. We begin with a careful statement of the rootfinding problem.

Let $f \in C^{\infty}(a,b)$. Find $\alpha \in (a,b)$, if it exists, such that $f(\alpha) = 0$. Assume further that $f'(\alpha) \neq 0$, so that α is a simple root of f.

The requirement that $f \in C^{\infty}(a,b)$ will allow us to consider general iterative methods. For each specific example we will consider, f need only have some finite number of continuous derivatives. Other conditions will be placed on the function f when they are necessary for the discussion.

The rootfinding problem arises frequently in applications and has been the subject of extensive research. It is equivalent to the fixed point problem $x = g(x)$ and its general setting includes partial differential equations, integral equations, and boundary value problems (see Collatz [7]).

1.7 Definition of Iterative Methods

Let N denote the number of processing elements used by the SIMD machine. Assume that the given function f has a root α. We consider iterative methods which compute a sequence of N-tuples $\{X_i = (x_{i,1}, x_{i,2}, \ldots, x_{i,N})\}$. Each component of X_i is an estimate for α. In general, the components of X_i need not be distinct, but for all the methods considered in this paper we must have $x_{i,j} \neq x_{i,k}$ if $j \neq k$. We consider iterations of the form

In general,

considered i

component to

such as ILLL

The a

sufficient f

following ri

of the defin

Kung and Tra

Definition I

on an open

zero α_f o

functions

satisfies:

1)

2)

3)

subinterva

$$x_{i+1,j} = \psi(X_i, X_{i-1},\ldots,X_{i-m}) .$$

In general, ψ may depend on the component j, but all of the methods considered in this paper use the same iteration function for each component to that the methods can be implemented on a SIMD machine such as ILLIAC IV.

The above definition of a parallel iterative method is sufficient for our discussion, but for completeness, we give the following rigorous definition. This definition is a generalization of the definition of a sequential iterative method (N = 1) given by Kung and Traub in [21].

Definition 1-2. Let $D = \{f | f$ is a real analytic function defined on an open interval depending on f, $I_f \subset R$ which contains a simple zero α_f of f, and $f' \neq 0$ on $I_f\}$. Let Ω denote the set of functions $\{\varphi\}$ which map every $f \in D$ to a function $\varphi(f)$ which satisfies:

1) $\varphi(f):R^{N(m+1)} \to R^N$;

2) $\varphi(f)(\alpha_f,\ldots,\alpha_f) = (\alpha_f,\ldots,\alpha_f)$;
$\quad\quad N(m+1)$ times $\quad$ N times

3) For each $\varphi \in \Omega$, $f \in D$, there exists $I_{\varphi,f}$, an open subinterval contained in I_f such that

i) $\alpha_f \in I_{\varphi,f}$,

ii) $\alpha(f)[I_{\varphi,f}^{N(m+1)}] \subset I_{\varphi,f}^{N}$, and

iii) if $X_0,X_1,\ldots,X_m \in I_{\varphi,f}^{N}$ and X_{i+1} is given by

$$X_{i+1} = \varphi(f)(X_i,X_{i-1},\ldots,X_{i-m}) ,$$

then

$$\lim_{i\to\infty} x_{i,j} = \alpha_f, \text{ for } j = 1,2,\ldots,N .$$

Then $\varphi \in \Omega$ is called a parallel rootfinding iteration with memory m. If $m = 0$, φ is called a parallel rootfinding iteration without memory. If $N = 1$, φ is a sequential rootfinding iteration as defined by Kung and Traub.

In general $\varphi(f)$ depends on the values of f and some of its derivatives. If $\varphi(f)$ requires the evaluation of f and its derivatives only at the points $x_{i,1},x_{i,2},\ldots,x_{i,N}$ (values at $x_{i-1,1},\ldots,x_{i-1,N},\ldots,x_{i-m,N}$ may be stored and used in the computation of X_{i+1}), φ is called a one point iteration. If $\varphi(f)$ requires the evaluation of f or its derivatives at points which are not components of X_i, φ is called a multipoint iteration.

All of the iterative methods satisfying this definition are called stationary iterative methods since the iteration function φ does not change as the iterations proceed.

Definition 1-3. The information usage of the iterative method φ is the number of parallel functional evaluations necessary to compute X_{i+1}. If $v_k(\varphi)$ denotes the number of parallel evaluations of $f^{(k)}$ necessary to compute X_{i+1}, then

$$v(\varphi) = \sum_{k\geq 0} v_k(\varphi) \tag{1-7}$$

is the total information usage of φ.

Recall that we are considering only iteration functions φ which are suitable for implementation of a SIMD machine. Hence each component of X_{i+1} is computed by the same formula, so if f must be evaluated at one component of X_i, it must be evaluated at

all N components. These N evaluations of f may be executed in parallel with PE j evaluating $f(x_{i,j})$. Thus one parallel functional evaluation yields N functional values.

1.8 Order of Convergence

An iterative method as defined in the preceding section generates a sequence of N-tuples $\{X_i\}$. The order of convergence of an iterative method is a quantitative way of measuring the rate at which this sequence converges to the N-tuple $(\alpha_f,\ldots,\alpha_f)$. In what follows, we shall discuss this idea for sequential rootfinding algorithms, and then generalize it to include parallel rootfinding algorithms.

As Brent points out [4, p. 21], there are many possible definitions of "order of convergence". We will use two of them. The sequence $\{x_i\}$ with $\lim_{i\to\infty} x_i = \alpha$ and $\epsilon_i = \alpha - x_i$ is said to have order of convergence λ if (1-8) or (1-9) hold.

$$\lim_{i\to\infty} \frac{|\epsilon_{i+1}|}{|\epsilon_i|^{\lambda}} = c > 0 \ , \ \lambda > 1 \ . \tag{1-8}$$

$$\lim_{i\to\infty} (-\ln |\epsilon_i|)^{1/i} = \lambda \ . \tag{1-9}$$

Definition 1-4. We shall say that the sequence $\{x_i\}$ has strong order of convergence λ if (1-8) holds, and weak order of convergence λ if (1-9) holds.

Note that condition (1-8) implies condition (1-9). Indeed, if we assume that (1-8) holds for the sequence $\{x_i\}$ with $x_i \to \alpha$ and let

$$c_i = \frac{|\epsilon_{i+1}|}{|\epsilon_i|^{\lambda}} \ ,$$

then there exists an integer I such that $i \geq I$ implies $c_i \geq c/2$. Without loss of generality, we may assume that $I = 0$ and $|\epsilon_i| < 1$ for all i. It is easily seen that

$$|\epsilon_i| = c_{i-1}c_{i-2}^{\lambda} \cdots c_0^{\lambda^{i-1}} |\epsilon_0|^{\lambda^i} . \tag{1-10}$$

Using (1-10), one has that

$$\ln |\epsilon_i| = \sum_{k=0}^{i-1} \lambda^k \ln c_{i-1-k} + \lambda^i \ln|\epsilon_0| ,$$

so that

$$\begin{aligned}(-\ln |\epsilon_i|)^{1/i} &= \left[\sum_{k=0}^{i-1} \lambda^k \ln \frac{1}{c_{i-1-k}} + \lambda^i \ln \frac{1}{|\epsilon_0|}\right]^{1/i} \\ &= \lambda\left[\sum_{k=0}^{i-1} \lambda^{k-i} \ln \frac{1}{c_{i-1-k}} + \ln \frac{1}{|\epsilon_0|}\right]^{1/i} . \end{aligned} \tag{1-11}$$

The assumption that $|\epsilon_i| < 1$ implies that $-\ln|\epsilon_i| > 0$. Hence the term in brackets in equation (1-11) is positive. Since $\lim_{i\to\infty} a^{1/i} = 1$ if $a > 0$, we need only show that the term in the brackets is bounded as $i \to \infty$. Now

$$\begin{aligned}&\left[\sum_{k=0}^{i-k} \lambda^{k-i} \ln \frac{1}{c_{i-1-k}} + \ln \frac{1}{|\epsilon_0|}\right] \\ &\qquad \leq \frac{1}{\lambda^i}\left[\sum_{k=0}^{i-1} \lambda^k \ln \frac{2}{c}\right] + \ln \frac{1}{|\epsilon_0|} \\ &\qquad \leq \frac{\ln (2/c)}{\lambda^i}\left[\frac{1 - \lambda^i}{1 - \lambda}\right] + \ln \frac{1}{|\epsilon_0|} \\ &\qquad \leq \frac{\ln (2/c)}{\lambda - 1} + \ln \frac{1}{|\epsilon_0|} . \end{aligned}$$

∎

However, condition (1-9) does not imply condition (1-8). This can easily be seen by letting

$$\epsilon_i = \begin{cases} e^{-2^i} & i \text{ even} \\ e^{-2^{i+1}} & i \text{ odd} \end{cases} .$$

Ortega and Rheinboldt [26] discuss these and several other possible definitions of order of convergence. They point out that condition (1-8) usually requires involved proofs when λ is not an integer, while the proofs required when condition (1-9) is used as the definition are the same whether λ is an integer or not.

We shall now generalize Definition 1-4 for parallel root-finding algorithms. While a sequential rootfinding method generates a sequence $\{x_i\}$ of estimates for the root α, a parallel algorithm generates a sequence of N-tuples $\{X_i = (x_{i,1}, x_{i,2}, \ldots, x_{i,N})\}$. Setting $\epsilon_{i,j} = \alpha - x_{i,j}$ and $\delta_i = \max\{|\epsilon_{i,j}|, j = 1,2,\ldots,N\}$, we define the order of convergence in terms of the worst error at each step.

Definition 1-5. We shall say that a sequence of N-tuples $\{X_i\}$ which converges component-wise to the N-tuple $(\alpha,\alpha,\ldots,\alpha)$ has strong order of convergence λ if

$$\lim_{i\to\infty} \frac{\delta_{i+1}}{(\delta_i)^\lambda} = C > 0 , \tag{1-12}$$

and weak order of convergence λ if

$$\lim_{i\to\infty} (-\ln \delta_i)^{1/i} = \lambda . \tag{1-13}$$

For each of the parallel rootfinding algorithms considered in this paper, we show that

$$\lim_{i\to\infty} (-\ln|e_{i,j}|)^{1/i} = \lambda, \text{ for } j = 1,\ldots,N ,$$

so (1-13) holds for the error in each component, not only for the largest error. However, the strong order of convergence of these methods remains an open question. It is our conjecture that for each of the methods considered in this paper, the strong order of convergence is the same as the weak order of convergence. In [32], Rice cites technical problems with using the strong order of convergence in the parallel case as his reason for using the weak order of convergence.

1.9 Examples of Rootfinding Iterations

In this section we shall consider five sequential rootfinding algorithms and the parallel secant method introduced in Section 1.4 to illustrate the definitions of iterative methods and order of convergence.

1.9.1 Newton-Raphson Method

The familiar Newton-Raphson method is given by

$$\varphi(f)(x) = x - \frac{f(x)}{f'(x)} .$$

This is a one-point sequential iteration without memory. It has both strong and weak orders of convergence 2. It is characterized by $m = 0$, $N = 1$, $v(\varphi) = 2$, and $\lambda = 2$.

1.9.2

g.

Thi

N

is

ti

wi

1.

Thi

m =

of

ach

ev

1.9

1.9.2 Generalized Newton-Raphson Method (see [40, pp. 78-87])

Let f^{-1} exist in a neighborhood of α and be denoted by g. Define

$$\gamma_1 = x$$

$$\gamma_2 = \gamma_1 - \frac{f(x)}{f'(x)}$$

$$\varphi(f)(x) = \gamma_3 = \gamma_2 - \frac{f''(x)}{2f'(x)}\left[\frac{f(x)}{f'(x)}\right]^2 .$$

This is a sequential one-point iteration without memory with $m = 0$, $N = 1$, $v_0(\varphi) = v_1(\varphi) = v_2(\varphi) = 1$, $v(\varphi) = 3$, and $\lambda = 3$. This method is a special case of a general class of sequential one-point iterations without memory which achieve a strong order of convergence λ with one evaluation each of $f, f', \ldots, f^{(\lambda-1)}$.

1.9.3 Multipoint Method One (see [21])

Let $\beta \neq 0$ be a constant. Define

$$\psi_0 = x$$

$$\psi_1 = \psi_0 + \beta f(\psi_0)$$

$$\varphi(f)(x) = \psi_2 = \psi_1 - \frac{\beta f(\psi_0) f(\psi_1)}{f(\psi_1) - f(\psi_0)} .$$

This is a sequential multipoint iteration without memory with $m = 0$, $N = 1$, $v_0(\varphi) = v(\varphi) = 2$, and $\lambda = 2$. It is a special case of a general class of sequential multipoint iteration methods which achieve a strong order of convergence $\lambda = 2^{n-1}$ with just n evaluations of f.

1.9.4 Multipoint Method Two (see [21])

Let

$$z_0 = x$$

$$z_1 = z_0 - \frac{f(z_0)}{f'(z_0)}$$

$$\varphi(f)(x) = z_2 = z_1 - \frac{f(z_1)f(z_0)}{[f(z_1) - f(z_0)]^2} \cdot \frac{f(z_0)}{f'(z_0)} \quad .$$

This is a sequential two-point iteration since functional values at z_0 and z_1 are needed to compute $\varphi(f)(x)$. Here $m = 0$, $N = 1$, $v_0(\varphi) = 2$, $v_1(\varphi) = 1$, $v(\varphi) = 3$, and $\lambda = 4$. This method is a special case of a general class of multipoint sequential root-finding algorithms without memory which achieve a strong order of convergence $\lambda = 2^{n-1}$ with $n-1$ evaluations of f and one evaluation of f'.

1.9.5 Lagrange Interpolation Methods (see [30, pp. 336-339], or [39, pp. 60-75]

Let $x_0, x_1, \ldots, x_m$ be given estimates close to α. Let $h(x)$ be the Lagrange interpolation polynomial of degree m which agrees with f at $x_i, x_{i-1}, \ldots, x_{i-m}$. Let x_{i+1} be a real zero of h chosen according to a specified policy. Define $\varphi(f)(x_i, \ldots, x_{i-m}) = x_{i+1}$. This is called direct Lagrange interpolation. Alternately, let $H(y)$ be the Lagrange interpolation polynomial of degree m which satisfies $H(y_j) = x_j$, for $j = i, i-1, \ldots, i-m$, where $y_j = f(x_j)$. Then H interpolates f^{-1}, so this is called inverse Lagrange interpolation. Define $x_{i+1} = H(0)$. Both of these methods are sequential one-point iteration methods with memory m, $N = 1$, and $v_0(\varphi) = v(\varphi) = 1$. The strong order of convergence λ is the unique positive root of the equation

$$t^{m+1} - t^m - \ldots - t - 1 = 0 \quad .$$

λ also satisfies

$$1 \leq \lambda < 2$$

(see [40, pp. 62-74]).

Both direct and inverse Lagrange interpolation use $f(x_j)$, for $j = i, i-1, \ldots, i-m$, and both have the same strong order of convergence. If H is the inverse Lagrange interpolation polynomial, $H(0)$ is easily computed, while it is more difficult to find the root by direct interpolation using h. Although it can be shown that h has a real root x_{i+1} which makes the method converge, that root need not be unique. If more than one root of h have the desired properties, additional criteria are used to assure that the choice of x_{i+1} is well-defined. $h(x)$ is a polynomial of degree m, so the problem of actually finding its root or roots may require a separate rootfinding algorithm. Thus it may be quite difficult to compute x_{i+1} as a root of h, while the computation of x_{i+1} as $H(0)$, where H is the inverse interpolation polynomial, is straightforward. For this reason, rootfinding algorithms using inverse Lagrange interpolation are preferred over those using direct Lagrange interpolation.

Chapter II will consider parallel iterations which use Lagrange interpolation.

1.9.6 Parallel Secant Method

The parallel secant method introduced in Section 1.4 is a parallel one-point iterative method without memory with $m = 0$, $N = 3$, $v_0(\varphi) = v(\varphi) = 1$. That is, only one parallel evaluation of f is necessary to evaluate f at three different points.

In Chapter II, it will be shown that the parallel secant method has weak order of convergence 2.

We note that the parallel secant method is a generalization of a sequential one-point iteration with memory $m = 1$. Most of the parallel one-point methods considered in this paper are generalizations of sequential methods with memory.

1.10 Results on Optimal Order

Since the publication of Traub's book [40] in 1964, considerable progress has been made in the search for sequential iterative rootfinding methods of optimal order. This section will survey some of those results. The orders of convergence discussed in this section are all strong orders of convergence. An extensive list of references dealing with results on optimal orders can be found in [39].

In the case of sequential one-point methods without memory, it is known [40, p. 98] that the highest order which can be achieved using only $f(x_i), f'(x_i), \ldots, f^{(r-1)}(x_i)$ is r for $r \geq 2$. Further, any such method of order r must depend explicitly on $f(x_i), f'(x_i), \ldots, f^{(r-1)}(x_i)$. In fact, this bound is achieved by the generalized Newton-Raphson method introduced in Section 1.9.2.

In the case of sequential one-point methods with memory, consider the method φ which computes $f(x_i), f'(x_i), \ldots, f^{(r-1)}(x_i)$ and uses the values of $f, f', \ldots, f^{(r-1)}$ evaluated at $x_{i-1}, x_{i-2}, \ldots, x_{i-m}$ which were computed and stored during previous iterations. It is shown in [40, p. 106] that the order of convergence λ of φ satisfies

$$r < \lambda < r+1, \quad \text{for} \quad m = 1,2,\ldots . \tag{1-14}$$

Hence no amount of memory will improve the order by as much as one evaluation of $f^{(r)}$.

In [5], Brent and Winograd consider another class of sequential one-point methods with memory. They show that the highest order of convergence which can be achieved by direct or inverse Hermite interpolation of $f^{(k)}(x_j)$, for $j = 0,1,\ldots,i$; $k = 0,1,\ldots,r-1$, is $r+1$. This bound is achieved by using <u>all</u> of the old functional evaluations in the Hermite interpolation to compute x_{i+1}. They suggest that the upper bound in (1-14) is approached as m grows large.

Rissanen [33] proves that the usual secant method has optimal order among sequential one-point iterations using only $f(x_i)$ and $f(x_{i-1})$, if a smoothness condition on admissible algorithms is assumed.

In the case of sequential multipoint methods without memory, Kung and Traub [21] construct two classes of algorithms of order 2^{n-1} using n functional evaluations. The first method evaluates f at n points (see Section 1.9.3), while the second method evaluates f' at one point and f at $n-1$ points (see Section 1.9.4). Kung and Traub go on to show that the order of any sequential multipoint method without memory using n functional evaluations is $\leq 2^n$. It is their conjecture that the optimal order is actually $\leq 2^{n-1}$. In another paper [20], Kung and Traub prove the truth of their conjecture in the case $n = 2$.

Very little work has been done in the area of sequential multipoint iterations with memory. The question of optimal order for such methods is still open.

1.11 Definitions of Computational Efficiency

If we wish to compare two different iterative rootfinding algorithms, we must have some measure of efficiency. Perhaps the ultimate efficiency measure is the time required to compute the desired root to within a specified accuracy. However, this is an impractical measure since it depends heavily on the function, the initial guess, and the machine used for the computations. For comparison of general methods, we want a measure which does not depend on the computer used to implement the algorithm.

The simplest measure is the order of convergence of the method, since the error of a high order method asymptotically approaches 0 more rapidly than the error of a low order method. If two methods have the same order of convergence, the one with the smaller asymptotic error constant converges more quickly. However, the results cited in Section 1.10 on optimal orders showed that high order iterative methods require more functional evaluations than lower order methods. In some cases, the desired root may be found in less time by a low order method requiring fewer functional evaluations than by a high order method.

This suggests that an efficiency measure be defined which reflects both the order of convergence of the method and its information usage (see Definition 1-3). The following efficiency measure generalizes a measure used by Traub [40] for sequential algorithms to include parallel algorithms.

Definition 1-6. Let φ be an iterative rootfinding method with order of convergence λ. Let the information usage of φ, the total number of parallel functional evaluations required by one step of

the method, be denoted by $v(\varphi)$. Define the information efficiency of the method to be

$$IE(\varphi) = \frac{\log_2 \lambda}{v(\varphi)} \quad . \tag{1-15}$$

This measure of efficiency is invariant under the composition of a method with itself. That is, if $\psi = \varphi \circ \varphi$, one step of ψ is the same as two steps of φ. If φ has order λ and information usage v, ψ has order λ^2 and information usage $2v$. Since that same work is being done, one would expect the information efficiency to be the same. Indeed,

$$IE(\varphi) = \frac{2 \log_2 \lambda}{2 v} = IE(\psi) \quad .$$

In [19], Kung and Traub show that (1-15) is essentially the unique way to define an efficiency measure, since any efficiency measure which is invariant under self composition is of this form or is a strictly increasing function of this form. Traub [40] shows that this measure of efficiency is inversely proportional to the total time required to compute the root α to a specified accuracy.

Other estimates of the time required to find the root α to a specified accuracy may replace the information usage used in (1-15). In [18], Kung shows that if the time is assumed to be proportional to the number of non-constant multiplications or divisions, M, then

$$\frac{\log_2 \lambda}{M} \leq 1 \quad .$$

It is for this reason that $\log_2$ is used in (1-15). Unless otherwise noted, all logs in this paper are taken in base 2.

To see that the information efficiency defined by (1-15) does not reflect the entire cost of computation, consider the sequential methods

$$x_{i+1} = \varphi_N(f)(x_i) = x_i - \frac{f(x_i)}{f'(x_i)} \quad , \tag{1-16}$$

and

$$x_{i+1} = \varphi_T(f)(x_i) = x_i - \frac{[f(x_i)]^2}{f(x_i + f(x_i)) - f(x_i)} \quad . \tag{1-17}$$

φ_N is the Newton-Raphson method, while φ_T is a two-point method due to Kung and Traub [20]. Both methods have strong order of convergence $\lambda = 2$, and both have information usage $v = 2$. Hence

$$IE(\varphi_N) = IE(\varphi_T) = 1/2 \quad .$$

Although both methods have the same information efficiency, the time required by each of them to compute α to a specified accuracy is different. φ_T has the advantage of not requiring f'. This is important if f' is not available or if it is costly to evaluate. However, once the functional evaluations have been made, φ_N requires only one addition and one division, while φ_T requires three additions, one multiplication, and one division. Hence if f and f' are equally costly to evaluate, φ_N will converge in less time. In fact, φ_N is faster whenever f' can be evaluated in less time than is required for the evaluation of f and the execution of two additions and one multiplication.

This suggests the following definition of efficiency which is due to Kung and Traub [19].

Definition 1-7. Let φ be an iterative rootfinding method with order of convergence λ. Let $c(f^{(j)})$ denote the number of arithmetic operations needed to evaluate $f^{(j)}$. Let $a(\varphi)$ denote the number of parallel arithmetic operations needed to combine the functional values used to compute X_{i+1}. Then we shall call the efficiency of φ

$$EFF(\varphi,f) = \frac{\log_2 \lambda}{\sum_{j\geq 0} v_j(\varphi)c(f^{(j)}) + a(\varphi)} , \tag{1-18}$$

where $v_j(\varphi)$ denotes the number of parallel evaluations of $f^{(j)}$ which are necessary at each step.

It is easily seen that the denominator of (1-18) is the total number of parallel arithmetic operations required at each step of the method. We will be concerned with finding upper and lower bounds for the efficiencies of various classes of algorithms by finding lower and upper bounds, respectively, for the denominator of (1-18). If we let $c_f = \min\{c(f^{(j)}), j \geq 0\}$, we immediately have the upper bound

$$EFF(\varphi,f) \leq \frac{\log \lambda}{v(\varphi)c_f + a(\varphi)} , \tag{1-19}$$

where $v(\varphi) = \sum_{j\geq 0} v_j(\varphi)$ is the number of parallel functional evaluations required at each step.

1.12 Results on Optimal Efficiency

We now survey some of the results which are known about algorithms whose efficiency (see Definition 1-7) is optimal. This

is an active area of interest and many questions are still open.

In [20], Kung and Traub consider the methods φ_N and φ_T given by equations (1-16) and (1-17), respectively. They show that these methods have optimal efficiency in the sense of Definition 1-7 among all sequential methods without memory using only two functional evaluations. Their respective efficiencies are given by

$$EFF(\varphi_N, f) = 1/(c(f) + c(f') + 2) \ , \tag{1-20}$$

and

$$EFF(\varphi_T, f) = 1/(2c(f) + 5) \ , \tag{1-21}$$

where $c(f)$ and $c(f')$ denote the number of arithmetic operations needed to evaluate f and f', respectively. From equations (1-20) and (1-21), it follows that the Newton-Raphson method φ_N is more efficient than the two-point method φ_T if and only if $c(f') < c(f) + 3$.

Next we compute the efficiency of the parallel secant method introduced in Section 1.4. Recall that the first component of X_{i+1} is given by

$$x_{i+1,1} = x_{i,1} - \frac{x_{i,1} - x_{i,2}}{f(x_{i,1}) - f(x_{i,2})} f(x_{i,1}) \ . \tag{1-22}$$

The parallel secant method requires one parallel evaluation of f, and has weak order of convergence 2. Equation (1-22) requires five arithmetic operations. Hence the efficiency of the parallel secant method, φ_S, is given by

$$EFF(\varphi_S, f) = 1/(c(f) + 5) \quad . \tag{1-23}$$

Comparing equation (1-23) with equations (1-20) and (1-21), we see that the parallel secant method is always more efficient than Traub's two-point method, and that it is more efficient than the Newton-Raphson method whenever $c(f') > 3$.

For iterative rootfinding methods φ belonging to a given class Φ of rootfinding methods, Kung and Traub [19] and Traub [40] define

$$E_n(\Phi, f) = \sup\{EFF(\varphi, f) \mid \varphi \in \Phi,\ v(\varphi) \leq n\}$$

and

$$E(\Phi, f) = \sup\{E_n(\Phi, f),\ n = 1, 2, \ldots\} \ ,$$

where $v(\varphi)$ denotes the information usage of φ. They then give some lower and upper bounds for E_n and E for various classes Φ of algorithms. Their results are summarized by the following theorems.

Theorem 1-1 [19]. Let Φ be the class of sequential one-point iterations without memory. Then there exists some constant $\rho > 0$ such that

$$\frac{\log n}{\sum_{i=0}^{n} c(f^{(i)}) + \rho\, n^2 \log n} \leq E_n(\Phi, f) \leq \frac{\log n}{n\, c_f + n - 1} \ , \tag{1-24}$$

for all n, where $c_f = \min\{c(f^{(i)}),\ i \geq 0\}$, and if $c_f > 4$,

$$\frac{\log 3}{c(f) + c(f') + c(f'') + 7} \leq E(\Phi, f) \leq \frac{\log 3}{3c_f + 2} \ . \tag{1-25}$$

Theorem 1-2 [19]. Let Γ be the class of sequential multipoint iterations without memory using values of f only. Then there exist constants r_0, r_1, and r_2 with $r_2 > 0$ such that

$$\frac{n - 1}{nc(f) + r_2 n^2 + r_1 n + r_0} \leq E_n(\Gamma, f) \leq \frac{n}{nc(f) + n - 1}, \tag{1-26}$$

for all n, and

$$\frac{1}{c(f)}\left(1 + \frac{\zeta}{\sqrt{c(f)}}\right) \leq E(\Gamma, f) \leq \frac{1}{c(f)}, \tag{1-27}$$

for some constant $\zeta < 0$.

It is further conjectured in [19] that for the class of all sequential one-point or multipoint iterations without memory,

$$E_n(\Phi, f) \leq \frac{n - 1}{nc_f + n - 1},$$

and that

$$E(\Phi, f) \leq \frac{1}{c_f + 1}.$$

The results in Theorems 1-1 and 1-2 will be used in the comparison of the efficiences of the parallel rootfinding algorithms developed in this paper to the efficiencies of optimal sequential algorithms.

1.13 Parallel Rootfinding

There has been very little research done in the area of parallel rootfinding algorithms.

In 1968, Karp and Miranker [15] gave a parallel searching technique for a maximum which is optimal in the minimax sense. These techniques can be used to solve the rootfinding problem since finding

a maximum of $-|f(x)|$ is the same as finding a root of f. Their algorithm makes one parallel functional evaluation to yield f evaluated at N points. It then searches for the largest value. Since searching cannot be done efficiently in parallel, this algorithm does not effectively utilize a parallel machine, but it does achieve a speed-up ratio proportional to $\log N$.

In [23], Miranker gives a "parallel" rootfinding algorithm, but the i th processing element PE i must have available the results computed by PE 1, PE 2,..., PE i-1 before it can proceed. Thus this is not a truly parallel algorithm. Miranker claims a speed-up ratio of $\frac{1}{2} \log_{1.618} N$.

In [32], Rice gives several parallel rootfinding algorithms based on direct or inverse polynomial interpolation. Some of the details of Rice's results will be presented in Section 2.1. Roughly, he computes the order of convergence of each of his methods from a matrix representation of the data used by each method. Some of Rice's methods require different computations to be performed by each processing element, so they are only suitable for implementation on a MIMD machine. However, his simultaneous (N-2) degree method is suitable for use on an array processing system like ILLIAC IV. This class of algorithms will be studied in detail in Chapter II.

CHAPTER II

ROOTFINDING BY LAGRANGE INTERPOLATION

One class of rootfinding algorithms whose structure is well suited to an array processing system like ILLIAC IV is based on direct or inverse Lagrange interpolation. In this chapter, we will study such methods. In particular, we will give the order of convergence of such methods, consider their efficiency, and show that their speed-up ratio is proportional to $\log N$, where N is the number of processing elements being used. We conclude that because of their poor speed-up ratio, these methods are not well suited for parallel processing, even though their structure makes efficient use of the parallel capabilities of a SIMD computer.

Definition 2-1. Let G_r denote the class of all stationary one-point parallel iterative rootfinding methods without memory which require that $f, f', \ldots, f^{(r)}$ be evaluated at the components of X_i.

In this chapter we shall consider G_0, the class of methods which use only values of f. The case of the general class G_r will be considered in Chapter III.

In what follows, we shall require that the real valued function f with a simple root α is fixed.

2.1 Rice's Results

In [32], Rice gives the algorithms we will discuss in this chapter. A general description of the algorithms of [32] follows.

A sequence of N-dimensional vectors $\{X_i = (x_{i,1}, x_{i,2}, \ldots, x_{i,N})\}$ is generated, where each $x_{i,j}$ is an approximation to α. An $N \times N$ matrix T is formed by

$$T_{kj} = \begin{cases} 1 & \text{if } x_{i,j} \text{ is used to compute } x_{i+1,k} \\ 0 & \text{otherwise .} \end{cases}$$

Consider the set $S_{i,k} = \{x_{i,j} | T_{kj} = 1\}$ for a fixed k. Let H be the Lagrange interpolation polynomial which satisfies

$$H(y_{i,j}) = x_{i,j}, \quad \text{for } x_{i,j} \in S_{i,k},$$

where $y_{i,j} = f(x_{i,j})$. Then let

$$x_{i+1,k} = H(0) .$$

That is, Rice performs classical inverse Lagrange interpolation on f. Rice's main result is the following.

Theorem 2-1 [32]. Assume that $f \in C^{N+1}$ in a neighborhood of the simple root α. Let ρ be the spectral radius of the matrix representation T. If $\rho > 1$, then the iterative method converges and has weak order of convergence ρ.

We wish now to focus our attention on the type of method which Rice calls the simultaneous (N-2) degree method. This method is of interest because it is the only method proposed by Rice which is suitable for implementation on a SIMD machine. His other methods require a MIMD machine because all of the processing elements do not perform the same computations. For example, one method requires three processors to perform interpolation at three points, while a fourth processor is performing interpolation on four points.

In the simultaneous (N-2) degree method, there are N distinct approximations to α stored at each step. Each of these N points is stored in a different processing element. To compute the new approximations, each processing element uses N-1 of these N points to compute a Lagrange interpolation polynomial of degree at most N-2. Of the many ways the N-1 points could be chosen by each processor, Rice has each processing element use the points stored in the other N-1 processors. Thus the matrix representation T for the simultaneous (N-2) degree method is given by

$$T_{kj} = 1 - \delta_{kj} ,$$

where δ_{kj} is Kronecker's delta. Thus for example, the matrix representation T for N = 4 is

$$T = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} . \qquad (2\text{-}1)$$

The weak order of convergence of the simultaneous (N-2) degree method is N-1. A simplified flow chart for the operations of one of the N processing elements is shown in Figure 2-1.

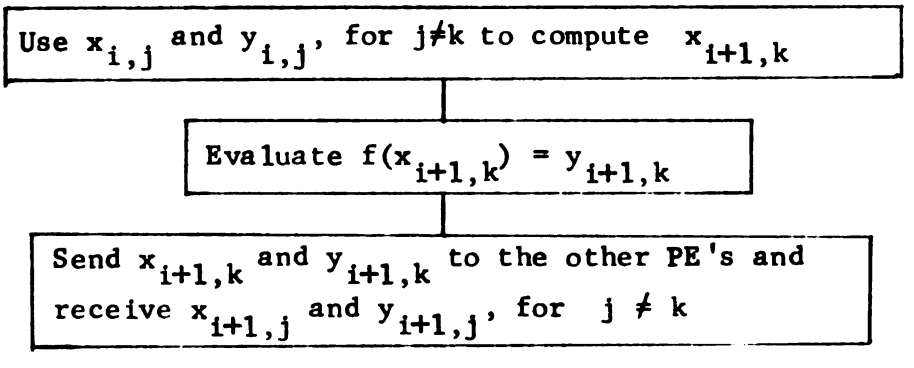


Figure 2-1. Flow Chart for PE k

2.2 Parallel Secant Method

The parallel secant method introduced in Section 1.4 is a slight modification of Rice's simultaneous first degree method (N = 3). Recall that if $y_{i,j} = f(x_{i,j})$, then the parallel secant method is given by

$$x_{i+1,1} = x_{i,1} - \frac{x_{i,1} - x_{i,2}}{y_{i,1} - y_{i,2}} y_{i,1}$$

$$x_{i+1,2} = x_{i,2} - \frac{x_{i,2} - x_{i,3}}{y_{i,2} - y_{i,3}} y_{i,2} \qquad (2\text{-}2)$$

$$x_{i+1,3} = x_{i,3} - \frac{x_{i,3} - x_{i,1}}{y_{i,3} - y_{i,1}} y_{i,3} \quad .$$

If we compare the flow chart given in Figure 1-8 for the parallel secant method with the flow chart given in Figure 2-1 for the simultaneous (N-2) degree method in the case N = 3, we see that the parallel secant method requires less data to be transferred among the processing elements. For example, in Rice's method, PE 1 must receive $x_{i+1,2}$, $y_{i+1,2}$, $x_{i+1,3}$, and $y_{i+1,3}$, while in the parallel secant method, PE 1 need only receive $x_{i+1,2}$ and $y_{i+1,2}$. Hence the parallel secant method requires less communication among the processing elements than Rice's method. Note that this modification does not change the spectral radius of the matrix representation of the method, so both methods have weak order of convergence 2.

Theorem 2-5 will give a different proof that the weak order of the parallel secant method is 2. This method requires one parallel functional evaluation and five arithmetic operations per step, so the information efficiency of this method is

$$IE = \frac{\log 2}{1} = 1 \quad ,$$

and the efficiency is

$$EFF = \frac{\log 2}{c(f) + 5} = \frac{1}{c(f) + 5} \quad ,$$

where $c(f)$ is the number of arithmetic operations needed to evaluate f, and log is understood to be $\log_2$.

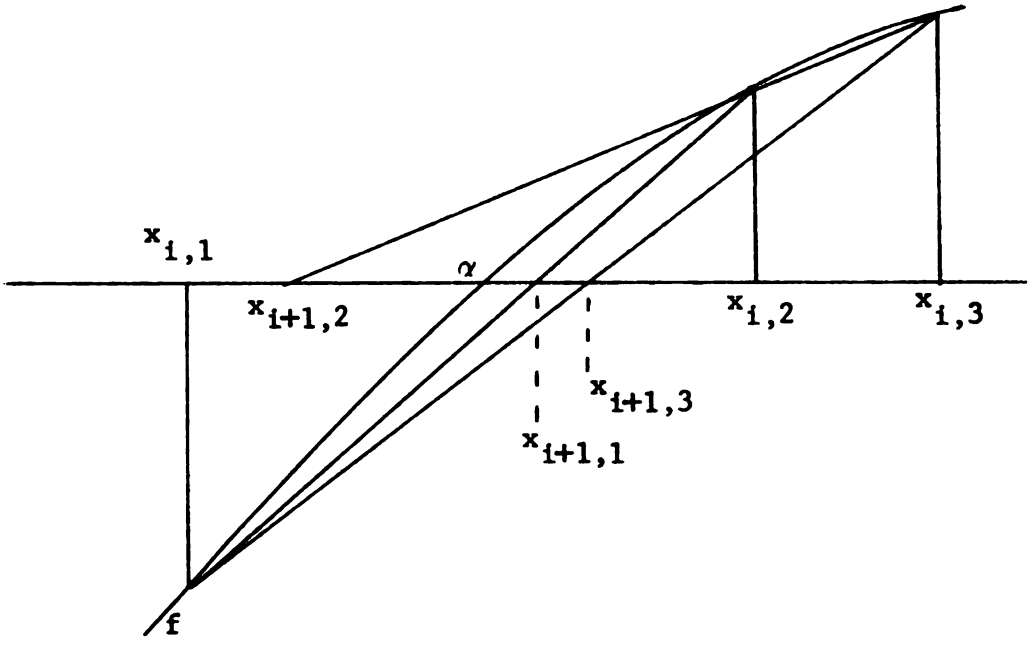


Figure 2-2. Parallel Secant Method

In Section 1.12 we showed that the parallel secant method is more efficient than the Newton-Raphson method whenever $c(f') > 3$.

2.3 Order of Convergence

In this section we will show that the weak order of convergence of parallel rootfinding methods based on Lagrange interpolation is equal to the number of points used for the interpolation. Before stating the exact result, we state the following well

known result on the error of Hermite interpolation (see [40, p. 244]). In this chapter we will only use this result for Lagrange interpolation ($r_j = 0$), but the more general result will be used in Chapter III.

Lemma 2-2. Let $\beta = k + \sum_{j=1}^{k} r_j$, $r_j \geq 0$, $j = 1,2,\ldots,k$. Let $f \in C^{\beta}$ on an interval containing $x_1,x_2,\ldots,x_k$. Let $h(x)$ be the unique Hermite interpolation polynomial of degree $\leq \beta - 1$ satisfying

$$h^{(i)}(x_j) = f^{(i)}(x_j), \text{ for } i = 0,\ldots,r_j;\ j = 1,\ldots,k .$$

Then the error is given by

$$E(x) = f(x) - h(x) = \frac{f^{(\beta)}(\xi)}{\beta!} \prod_{j=1}^{k} (x - x_j)^{r_j+1} ,$$

where ξ is on the interval spanned by $x,x_1,x_2,\ldots,x_k$.

Let us give the notation needed to construct parallel rootfinding algorithms based on inverse Lagrange interpolation. The same notation will be used in Chapter III for methods based on inverse Hermite interpolation. Fix k satisfying $2 \leq k \leq N-1$. Choose N distinct subsets A_j, for $j = 1,2,\ldots,N$, of the set $\{1,2,\ldots,N\}$ such that $j \in A_j$ and A_j contains k elements.

We will use the following lemma and its corollary to find the orders of convergence for most of the methods considered in this paper.

Lemma 2-3. Let φ be a parallel rootfinding algorithm which generates a sequence $\{X_i\}$ whose errors satisfy

$$e_{i+1,j} = K_{i,j} \prod_{t \in A_j} (e_{i,t})^{s} , \quad j = 1,2,\ldots,N , \tag{2-3}$$

for some sequence of constants $K_{i,j}$, where $e_{i,j} = \alpha - x_{i,j}$. Let

$$\delta_i = \max_j |\epsilon_{i,j}|$$

$$\xi_i = \min_j |\epsilon_{i,j}|$$

$$\bar{K}_i = \max_j |K_{i,j}|$$

$$\underline{K}_i = \min_j |K_{i,j}| \quad .$$

Assume further that $\epsilon_{i,j} \neq 0$ for all i and j, and that

$$\lim_{i\to\infty} \bar{K}_i = \bar{K} \neq 0 \ ,$$

$$\lim_{i\to\infty} \underline{K}_i = \underline{K} \neq 0 \ .$$

Then there exist positive sequences c_i and d_i, both with strong order of convergence ks, satisfying

$$c_i \leq \xi_i \leq |\epsilon_{i,j}| \leq \delta_i \leq d_i \ ,$$

for $j = 1,2,\ldots,N$, and $i = 0,1,\ldots$.

Proof. From equation (2-3),

$$|\epsilon_{i+1,j}| = |K_{i,j}| \prod_{t\in A_j} |\epsilon_{i,t}|^s \quad , \tag{2-4}$$

so that

$$|\epsilon_{i+1,j}| \leq \bar{K}_i \prod_{t\in A_j} (\delta_i)^s = \bar{K}_i (\delta_i)^{ks} \quad .$$

Then

$$\delta_{i+1} \leq \bar{K}_i (\delta_i)^{ks} \quad .$$

Let $d_0 = \delta_0$, and $d_{i+1} = \bar{K}_i (d_i)^{ks}$. Then

$$\lim_{i\to\infty} \frac{d_{i+1}}{(d_i)^{ks}} = \bar{K} \quad ,$$

so the sequence $\{d_i\}$ has strong order of convergence ks. By induction, $\delta_i \le d_i$ since

$$\delta_{i+1} \le \bar{K}_i(\delta_i)^{ks} \le \bar{K}_i(d_i)^{ks} = d_{i+1} \quad .$$

Similarly from equation (2-4),

$$|e_{i+1,j}| \ge \underline{K}_i \prod_{t\in A_j} (\xi_i)^s = \underline{K}_i(\xi_i)^{ks} \quad .$$

Then if $c_0 = \xi_0$, and $c_{i+1} = \underline{K}_i(c_i)^{ks}$,

$$\lim_{i\to\infty} \frac{c_{i+1}}{(c_i)^{ks}} = \underline{K} \quad ,$$

so $\{c_i\}$ has strong order of convergence ks, and

$$\xi_{i+1} \ge \underline{K}_i(\xi_i)^{ks} \ge \underline{K}_i(c_i)^{ks} = c_{i+1} \quad . \blacksquare$$

From this lemma it follows that

Corollary 2-4. Under the conditions of Lemma 2-3, the rootfinding method φ has weak order of convergence ks.

Proof. According to the lemma, there exist sequences $\{c_i\}$ and $\{d_i\}$, both with strong orders of convergence ks, which satisfy

$$c_i \le |e_{i,j}| \le d_i \quad , \text{ for } \quad j = 1,\ldots,N, \text{ and } \quad i = 0,1,\ldots \quad .$$

Then since strong order implies weak order,

$$ks = \lim_{i\to\infty} (-\ln d_i)^{1/i} \le \lim_{i\to\infty} (-\ln|e_{i,j}|)^{1/i} \le \lim_{i\to\infty} (-\ln c_i)^{1/i} = ks \quad ,$$

which implies that

$$\lim_{i\to\infty} (-\ln |e_{i,j}|)^{1/i} = ks \quad , \text{ for } \quad j = 1,2,\ldots,N \quad . \qquad \blacksquare$$

In order to find the order of convergence for parallel root-finding methods based on Lagrange interpolation, we need some additional restrictions on f.

Let $f \in C^k(I)$, where I is an open interval containing the simple root α of f, and on which $f' \neq 0$. Let f^{-1} be denoted by g. Then $g \in C^k$ on some open interval containing 0. Assume that $g^{(k)} \neq 0$ in a neighborhood of 0. Let $y_{i,j} = f(x_{i,j})$.

We are now ready to state and prove the main result of this section.

Theorem 2-5. With the notation and assumptions given above, for each $j = 1,2,\ldots,N$, let $H_j(y)$ denote the Lagrange interpolation polynomial of degree at most $k-1$ which satisfies

$$H_j(y_{i,s}) = x_{i,s} \quad , \text{ for } \quad s \in A_j \quad .$$

Let $\varphi_k \in G_0$ be the method which defines

$$x_{i+1,j} = H_j(0) \quad .$$

If distinct points $x_{0,1}, x_{0,2}, \ldots, x_{0,N}$ are chosen sufficiently close to α, then either $x_{i,j} = \alpha$ for some i and j, or

$$\lim_{i\to\infty} x_{i,j} = \alpha \quad , \text{ for } \quad j = 1,2,\ldots,N \quad .$$

Thus the method φ_k converges. Moreover, φ_k has weak order of convergence k.

The proof is a generalization of the proof given in [30] for the sequential case. When no ambiguity will result in the proof, we suppress the subscript i to simplify the notation.

Proof. If $y_t = 0$ for any $t = 1,2,\ldots,N$, then $x_{i,t} = \alpha$, and we are done. Hence we may assume that $y_t \neq 0$. We may also assume that $y_s \neq y_t$, for $s \neq t$, for if $x_{i,s} = x_{i,t}$, we may replace $x_{i,s}$ by any x from the interval spanned by $x_{i,1}, x_{i,2}, \ldots, x_{i,N}$. Since the resulting $\epsilon_{i,s}$ still lies between the smallest and the largest errors, the proof remains valid. If for some i, all of the $x_{i,s}$ are identical, they span no interval and the method fails.

H is the Lagrange interpolation polynomial for g, so that it must have the form

$$g(y) \approx H(y) = \sum_{s \in A_j} \ell_s(y) g(y_s) = \sum_{s \in A_j} \ell_s(y) x_{i,s} , \tag{2-5}$$

where

$$\ell_s(y) = \prod_{\substack{t \in A_j \\ t \neq s}} \frac{y - y_t}{y_s - y_t} .$$

Let $Z = \prod_{s \in A_j} (-y_s)$. Inverse Lagrange interpolation estimates $\alpha = g(0)$ with $H_j(0)$, so in our method we have

$$x_{i+1,j} = H_j(0) = \sum_{s \in A_j} \ell_s(0) x_{i,s} = -Z \sum_{s \in A_j} \frac{x_{i,s}}{y_s \prod_{\substack{t \in A_j \\ t \neq s}} (y_s - y_t)} . \tag{2-6}$$

According to Lemma 2-2, the error in (2-6) is given by

$$\epsilon_{i+1,j} = \alpha - x_{i+1,j} = \frac{g^{(k)}(\eta)}{k!} Z , \tag{2-7}$$

where η lies in the interval spanned by 0 and y_s, $s \in A_j$. Now

$$y_s = f(x_{i,s}) = f(x_{i,s}) - f(\alpha)$$

$$= f'(\xi_s)(x_{i,s} - \alpha) = -f'(\xi_s)\epsilon_{i,s}$$

$$= -\frac{\epsilon_{i,s}}{g'(\eta_s)} \quad ,$$

where ξ_s lies between α and $x_{i,s}$, and η_s lies between 0 and y_s. Then equation (2-7) becomes

$$\epsilon_{i+1,j} = \frac{g^{(k)}(\eta)}{k!} \prod_{s \in A_j} \frac{\epsilon_{i,s}}{g'(\eta_s)} \quad . \tag{2-8}$$

If $x_{0,1}, x_{0,2}, \ldots, x_{0,N}$ are chosen sufficiently close to α, equation (2-8) implies that

$$\lim_{i \to \infty} \epsilon_{i,j} = 0 \quad , \text{ for } \quad j = 1,2,\ldots,N,$$

so the method converges. We have assumed $y_t \neq 0$, so that $\epsilon_{i,t} \neq 0$, for $t = 1,2,\ldots,N$. Since $g^{(k)} \neq 0$ near 0 by assumption, equation (2-8) implies that $\epsilon_{i+1,j} \neq 0$. Let

$$K_{i,j} = \frac{g^{(k)}(\eta)}{k!} \prod_{s \in A_j} \frac{1}{g'(\eta_s)} \quad ,$$

so that

$$\lim_{i \to \infty} K_{i,j} = \frac{g^{(k)}(0)}{k![g'(0)]^k} \neq 0 \quad .$$

Then all of the hypotheses of Lemma 2-3 are satisfied with $s = 1$, so it follows from Corollary 2-4 that φ_k has weak order of convergence k. ■

For each k, there are $\binom{N}{k}$ different subsets of the set $\{1,2,\ldots,N\}$. Hence there are $\binom{\binom{N}{k}}{N}$ different methods of order k

based on inverse Lagrange interpolation, but these methods differ only in the indexing of $x_{i,1},x_{i,2},\ldots,x_{i,N}$. These methods arise from different choices for the subsets A_j.

The condition $2 \le k \le N-1$ arises from the need to choose the N distinct subsets A_j of $\{1,2,\ldots,N\}$, each containing k distinct elements. The subsets must be distinct to assure that $\{x_{i+1,1},x_{i+1,2},\ldots,x_{i+1,N}\}$ are all distinct, so k cannot equal N. This condition gives the following corollary to Theorem 2-5.

Corollary 2-6. Under the conditions of Theorem 2-5, the highest weak order of convergence which can be achieved on a parallel processing system with N processing elements by rootfinding algorithms based on inverse Lagrange interpolation is $N-1$.

This corollary shows that the optimal order for this class of parallel rootfinding algorithms grows linearly with the machine size.

We now compare the orders of convergence of parallel rootfinding algorithms based on inverse Lagrange interpolation with the corresponding orders of convergence of the sequential one-point iterations with memory which also use inverse Lagrange interpolation. We have already seen, for example, that the parallel secant method ($N = 3$, and $k = 2$) increases the order of convergence of the usual secant method from $\frac{1+\sqrt{5}}{2} \approx 1.618$ to 2. As noted in Section 1.10, [40, p. 106] shows that the order of convergence of sequential one-point iterations with memory $m = k-1$ is < 2. Hence the use of parallel processing has increased the order of convergence from something less than two to k.

It can be shown that Theorem 2-5 remains true if direct Lagrange interpolation is used instead of inverse Lagrange interpolation. However, as we saw in Section 1.9.5, the methods using inverse Lagrange interpolation are always preferred over those using direct Lagrange interpolation. For this reason the corresponding theorem for direct interpolation is omitted.

2.4 Efficiency

It was shown in Section 2.3 that a weak order of convergence of $N-1$ can be achieved on a parallel processing computer using N processors. This requires each processor to perform Lagrange interpolation on $N-1$ points. Although this interpolation is very costly for large N, we will show that if evaluations of f are expensive, the optimal efficiency for methods based on Lagrange interpolation is achieved by taking $k = N-1$.

Definition 2-2. Let φ_k be a parallel rootfinding algorithm based on inverse Lagrange interpolation as described in Theorem 2-5. Define

$$P_k(f) = EFF(\varphi_k, f) = \frac{\log_2 k}{\sum_{s \geq 0} v_s(\varphi_k) c(f^{(s)}) + a(\varphi_k)} ,$$

where $v_s(\varphi_k)$ is the number of parallel evaluations of $f^{(s)}$ needed, $c(f^{(s)})$ is the number of arithmetic operations required by one evaluation of $f^{(s)}$, and $a(\varphi_k)$ is the number of arithmetic operations needed to combine the functional values used to compute X_{i+1}. Define also

$$P_*(f) = \sup_{2 \leq k} P_k(f) .$$

Although there are $\binom{N}{k}$ possible choices for φ_k, $P_k(f)$ is well-defined since the possible choices differ only in the indexing of the points used. $P_*(f)$ is the optimal efficiency of all parallel rootfinding algorithms based on Lagrange interpolation on any number of points. That is, we delete the requirement that $k \leq N-1$. Later we will ask whether there are enough processing elements available to perform interpolation on enough points to achieve the optimal efficiency.

Recall that the speed-up ratio was defined to be

$$S = \frac{\text{Computation time on a sequential computer}}{\text{Computation time on a parallel computer}} .$$

Since the efficiency $EFF(\varphi)$ is inversely proportional to the total time required to compute α to a specified accuracy using the method φ, the speed-up ratio for a class of parallel methods satisfies

$$S = \beta \frac{\text{optimal EFF for the class of parallel methods}}{\text{optimal EFF for sequential computation}} , \qquad (2\text{-}9)$$

for some constant $\beta > 0$. We will use the results of [19] summarized in Theorems 1-1 and 1-2 to estimate the optimal efficiency for sequential computation.

Theorem 2-7. Assume that f is analytic in a neighborhood of a simple root α. Let φ_k be an inverse Lagrange interpolation method described in Theorem 2-5. Note that k is not restricted to be $\leq N-1$. Then

i) $IE(\varphi_k) = \log_2 k$.

ii) $P_k(f) = EFF(\varphi_k, f) = \dfrac{\log k}{c(f) + 2k^2 + k - 1}$. $\qquad (2\text{-}10)$

iii) Let γ denote the unique root which is ≥ 1 of

$$u(k) = (4k^2 + k)\ln k - (c + 2k^2 + k - 1) = 0 . \quad (2\text{-}11)$$

Then

$$P_*(f) \approx \frac{1}{(4\gamma^2 + \gamma)\ln 2} . \quad (2\text{-}12)$$

iv) Assume that $2 \leq k \leq N\text{-}1$ and that $c = c(f^{(i)})$ for all i. For large values of c, the speed-up ratio of φ_k compared to the optimal sequential one-point methods without memory is asymptotic to

$$\frac{3}{\ln 3} \ln (N\text{-}1) - \frac{3}{2}$$

as $N \to \infty$.

v) Assume that $2 \leq k \leq N\text{-}1$. For large values of $c = c(f)$, the speed-up ratio of φ_k compared to the optimal sequential multipoint iterations without memory using values of f only is asymptotic to

$$\log (N\text{-}1) - \frac{1}{\ln 4}$$

as $N \to \infty$.

Proof. i) Each iteration of φ_k requires one parallel evaluation of f and yields weak order of convergence k.

ii) Count the arithmetic operations in equation (2-6).

iii) To show that γ is unique, let

$$u(t) = (4t^2 + t)\ln t - (c + 2t^2 + t - 1) .$$

Then

$$u'(t) = (8t + 1)\ln t + (4t + 1) - (4t + 1)$$

$$= (8t + 1)\ln t$$

$$> 0, \text{ for } t > 1 \ .$$

Now $u(1) = -(c + 2) < 0$, while $\lim_{t\to\infty} u(t) = +\infty$. Hence $u(t) = 0$ has a unique solution on the interval $(1, +\infty)$.

To find the value of k for which

$$P_*(f) = \sup_{2\le k} P_k(f)$$

is attained, we observe that $P_1(f) = 0$ and that

$$\lim_{k\to\infty} P_k(f) = 0 \ .$$

Thus $P_*(f)$ is attained for some finite $k > 1$. To find the optimal k, differentiate both sides of equation (2-10) and set equal to zero to get

$$\frac{d\,P_k(f)}{d\,k} = \frac{(c + 2k^2 + k\text{-}1)\,\frac{1}{k\ln 2} - (4k + 1)\log k}{(c + 2k^2 + k - 1)^2} = 0 \ .$$

This implies that

$$(4k^2 + k)\ln k - (c + 2k^2 + k - 1) = 0 \ .$$

γ is assumed to satisfy equation (2-11), so substituting (2-11) into (2-10) gives (2-12).

Although k must be an integer, γ need not be. Hence $P_*(f)$ is actually attained by $P_k(f)$, for some k within one unit of γ.

Before we proceed with the proof of the rest of the theorem, we digress to consider the behavior of γ and $P_k(f)$. Figure 2-3

shows how $P_k(f)$ depends on k and the cost of functional evaluation, c. Table 2-1 shows typical values of γ and the resulting $P_*(f)$ for given values of c.

Table 2-1 $P_*(f)$

c(f)	γ	k_{opt}	$P_*(f)$
1	1.755	2	.100
2	1.864	2	.091
5	2.120	2	.071
20	2.893	3	.040
100	4.764	5	.015
10^4	29.360	29	.00041
10^6	225.36	225	$7. \times 10^{-6}$

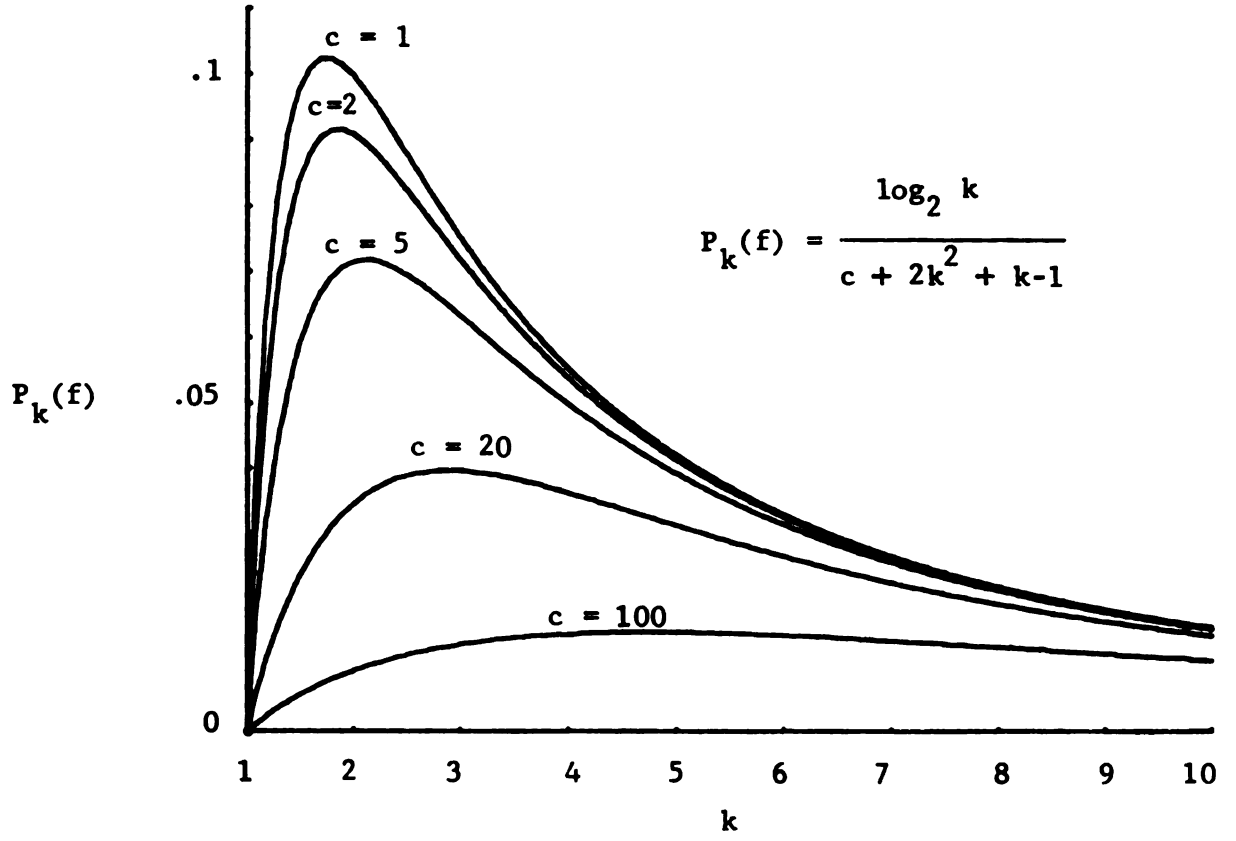


Figure 2-3 $P_k(f)$

Note that $\lim_{i\to\infty} \gamma = \infty$, for if we let

$$c = 1 + e^n(n - 1 + e^n(4n - 2)) ,$$

then $\gamma = e^n$.

Note also that $P_*(f)$ is independent of the machine size N, so it cannot be attained if $\gamma \geq N$. In practice, k should be chosen to satisfy

$$k = \begin{cases} N - 1 & \text{if } c > 3N - 2N^2 + (4N^2 - 7N + 3)\ln (N-1) \\ \text{integer within one of } \gamma & \text{which maximizes } P_k(f) . \end{cases}$$

For ILLIAC IV, N = 64, so that if $c > 58{,}038$, γ cannot be attained. That is, unless 58,038 or more arithmetic operations are required for each evaluation of f (or its rational approximation), the optimal efficiency for the class of parallel rootfinding algorithms based on inverse Lagrange interpolation can be realized on ILLIAC IV. If $c = 58{,}038$, $P_*(f) \approx 9.05 \times 10^{-5}$, $E_1(f) \approx 0.91 \times 10^{-5}$, and $E_2(f) \approx 1.72 \times 10^{-5}$, so all of these methods would require a long time to compute α to a specified accuracy, but the parallel method is slightly faster.

Let us now return to the proof of the theorem.

iv) Using equation (2-12) and the bounds for $E_1(f)$ given in Theorem 1-1,

$$\frac{3c + 2}{\log 3\ (4\gamma^2 + \gamma)\ln 2} \leq \frac{P_*(f)}{E_1(f)}$$

$$\leq \frac{3c + 7}{\log 3\ (4\gamma^2 + \gamma)\ln 2} . \qquad (2\text{-}13)$$

From equation (2-11),

$$c = (4\gamma^2 + \gamma)\ln \gamma - (2\gamma^2 + \gamma - 1) \quad . \tag{2-14}$$

Substituting equation (2-14) into (2-13), we get,

$$\frac{3(4\gamma^2 + \gamma)\ln \gamma - 6\gamma^2 - 3\gamma + 5}{\ln 3\ (4\gamma^2 + \gamma)} \leq \frac{P_*(f)}{E_1(f)}$$

$$\leq \frac{3(4\gamma^2 + \gamma)\ln \gamma - 6\gamma^2 - 3\gamma + 10}{\ln 3\ (4\gamma^2 + \gamma)} \quad ,$$

which implies that

$$\frac{3}{\ln 3}\ln \gamma - \frac{6\gamma^2 + 3\gamma - 5}{(4\gamma^2 + \gamma)\ln 3} \leq \frac{P_*(f)}{E_1(f)}$$

$$\leq \frac{3}{\ln 3}\ln \gamma - \frac{6\gamma^2 + 3\gamma - 10}{(4\gamma^2 + \gamma)\ln 3} \quad . \tag{2-15}$$

We have seen that $\lim_{c\to\infty} \gamma = \infty$, so that if c is large enough, the most efficient Lagrange interpolation method which can be implemented on a machine with N PE's uses $k = N-1$ points. As larger machines are used, the optimal efficiency of $P_*(f)$ is approached. Hence (2-15) shows that $\frac{P_*(f)}{E_1(f)}$ is asymptotic to

$$\frac{3}{\ln 3}\ln (N-1) - \frac{3}{2} \text{ , as } N \to \infty \quad .$$

v) Similarly, using equation (2-12) and the bounds for $E_2(f)$ given in Theorem 1-2,

$$\frac{c}{(4\gamma^2 + \gamma)\ln 2} \leq \frac{P_*(f)}{E_2(f)} \leq \frac{c}{(1 + \frac{\xi}{\sqrt{c}})(4\gamma^2 + \gamma)\ln 2} \quad ,$$

where ξ is a negative constant. This implies that

$$\log \gamma - \frac{2\gamma^2 + \gamma - 1}{(4\gamma^2 + \gamma)\ln 2} \leq \frac{P_*(f)}{E_2(f)} \leq \frac{\log \gamma}{1 + \frac{\xi}{\sqrt{c}}} - \frac{2\gamma^2 + \gamma - 1}{(4\gamma^2 + \gamma)\ln 2} ,$$

so that $\frac{P_*(f)}{E_2(f)}$ is asymptotic to

$$\log (N-1) - \frac{1}{2 \ln 4} , \text{ as } N \to \infty . \quad \blacksquare$$

The important conclusions from Theorem 2-7 are parts iv) and v). They show that if the cost of functional evaluation is so high that $\gamma \geq N$, then the speed-up of parallel rootfinding algorithms based on Lagrange interpolation is proportional to $\log_2$ of the machine size. If it is important to find the root α as fast as possible, as in real-time applications, these methods are useful because they are faster. If, however, one wishes to compute α at a low cost, the speed-up ratio of $\log N$ shows that these methods are not well suited for implementation on a parallel processing system.

CHAPTER III

ROOTFINDING BY HERMITE INTERPOLATION

In this chapter we generalize the results of Chapter II on Lagrange interpolation to include inverse Hermite interpolation. As in the Lagrange case, the order of convergence is equal to the amount of information used. That is, while Lagrange interpolation methods use values of the function f evaluated at k points to achieve order k, Hermite interpolation methods use $f,f',\ldots,f^{(r)}$, each evaluated at k points to achieve weak order of convergence $k(r+1)$. If a parallel processing system has N processing elements, we will also show that the speed-up ratio possible by using Hermite interpolation is proportional to $\log N$. That implies that parallel rootfinding algorithms based on Hermite interpolation are not well suited to parallel processing.

The reader is referred to [30] or [40] for a discussion of sequential rootfinding algorithms using Hermite interpolation.

3.1 Order of Convergence

The main result of this section is a generalization of Theorem 2-5 to show that the weak order of convergence of parallel rootfinding algorithms based on inverse Hermite interpolation of $f,f',\ldots,f^{(r)}$ at k points is $k(r+1)$.

Recall (see Definition 2-1) that G_r is the class of all one-point parallel iterative rootfinding methods without memory

which require that $f,f',\ldots,f^{(r)}$ be evaluated at the components of X_i.

Let us also recall the notation needed to construct parallel rootfinding algorithms based on inverse Hermite interpolation. Fix k satisfying $2 \le k \le N-1$. Choose N distinct subsets A_j, $j = 1,2,\ldots,N$, of the set $\{1,2,\ldots,N\}$ such that $j \in A_j$, and A_j contains k elements.

Theorem 3-1. Let $\gamma = k(r+1)$. Assume that $f \in C^{\gamma}$ on some open interval containing the simple root α, and on which $f' \neq 0$. Let f^{-1} be denoted by g. Then $g \in C^{\gamma}$ on some open interval containing 0. Assume that $g^{(\gamma)} \neq 0$ in a neighborhood of 0. For each $j = 1,2,\ldots,N$, let $H_j(y)$ denote the Hermite interpolation polynomial of degree at most $\gamma-1$ which satisfies

$$H_j^{(t)}(y_{i,s}) = g^{(t)}(y_{i,s}), \text{ for } t = 0,\ldots,r, \text{ and } s \in A_j \ .$$

Let $\psi_k \in G_r$ be the method which defines

$$x_{i+1,j} = H_j(0) \ .$$

If distinct points $x_{0,1},x_{0,2},\ldots,x_{0,N}$ are chosen sufficiently close to α, then either $x_{i,j} = \alpha$ for some i and j, or

$$\lim_{i\to\infty} x_{i,j} = \alpha \ , \text{ for } j = 1,2,\ldots,N \ .$$

Thus the method ψ_k converges. Moreover, ψ_k has weak order of convergence γ.

The proof of this theorem follows closely the proof of Theorem 2-5. As before, we will suppress the subscript i to simplify the notation.

and we

hood o

all di

otherw

the in

where

Now

where

y_s. T

ficien

Proof. We may assume $y_t \neq 0$, for if $y_t = 0$, $x_{i,t} = \alpha$, and we are done. We have assumed that f^{-1} exists in a neighborhood of α, so $y_1,\ldots,y_N$ are all distinct if $x_{i,1},\ldots,x_{i,N}$ are all distinct. If $x_{i,1},\ldots,x_{i,N}$ are identical, the method fails, otherwise, if $x_{i,s} = x_{i,t}$, we may replace $x_{i,s}$ by any point on the interval spanned by the x's.

H_j interpolates g, so according to Lemma 2-2,

$$g(y) - H_j(y) = \frac{g^{(\gamma)}(\eta)}{\gamma!} \prod_{s\in A_j} (y - y_s)^{r+1} \quad , \tag{3-1}$$

where η lies in the interval spanned by y, y_s, for $s \in A_j$. Now

$$\begin{aligned} y_s &= f(x_{i,s}) - f(\alpha) \\ &= (x_{i,s} - \alpha) f'(\xi_s) \\ &= -(\alpha - x_{i,s})/g'(\eta_s) \\ &= -e_{i,s}/g'(\eta_s) \quad , \end{aligned}$$

where ξ_s lies between $x_{i,s}$ and α, and η_s lies between 0 and y_s. Then, substituting zero into equation (3-1),

$$\begin{aligned} e_{i+1,j} &= \alpha - x_{i+1,j} \\ &= \frac{g^{(\gamma)}(\eta)}{\gamma!} \prod_{s\in A_j} (-y_s)^{r+1} \\ &= \frac{g^{(\gamma)}(\eta)}{\gamma! \prod_{s\in A_j} [g'(\eta_s)]^{r+1}} \prod_{s\in A_j} (e_{i,s})^{r+1} \quad . \end{aligned} \tag{3-2}$$

If N distinct points $x_{0,1},x_{0,2},\ldots,x_{0,N}$ are chosen sufficiently close to α, equation (3-2) implies that

so

for

Ther

so t

foll

pola

the

Coro

weak

inte

N PE

memor

of f

r+1, s

orders

spondi

tion.

tion m

$$\lim_{i\to\infty} \varepsilon_{i,s} = 0, \text{ for } s = 1,2,\ldots,N ,$$

so ψ_k converges. Equation (3-2) also implies that $\varepsilon_{i,s} \neq 0$ for all i and s since we assumed that $g^{(\gamma)} \neq 0$ near 0. Let

$$K_{i,j} = \frac{g^{(\gamma)}(\eta)}{\gamma! \prod_{s\in A_j} [g'(\eta_s)]^{r+1}} .$$

Then

$$\lim_{i\to\infty} K_{i,j} = \frac{g^{(\gamma)}(0)}{\gamma! [g'(0)]^{\gamma}} \neq 0 ,$$

so the hypotheses of Lemma 2-3 are satisfied with $s = r+1$. It follows from Corollary 2-4 that ψ_k has weak order of convergence γ. ∎

Since k, the number of points in A_j on which the interpolation is performed, is constrained to lie between 2 and $N-1$, the following corollary follows easily from Theorem 3-1.

Corollary 3-2. Under the conditions of Theorem 3-1, the highest weak order of convergence which can be achieved by an inverse Hermite interpolation method in G_r on a parallel processing computer with N PE's is $(N-1)(r+1)$. This order is achieved by ψ_{N-1}.

We remark that the sequential one-point iterations with memory $m = k-1$ which use inverse Hermite interpolation and values of $f, f', \ldots, f^{(r)}$ have strong order of convergence between r and $r+1$, so the use of parallel processing has greatly improved the orders of convergence of these methods.

As in the case for Lagrange interpolation, a theorem corresponding to Theorem 3-1 can be proven for direct Hermite interpolation. As before, that theorem is omitted because direct interpolation methods are of little practical importance.

3.2 E

we cons

element

are ide

the sub

$x_{i,2}$ =

$y_2 = f($

inverse

data is

$H_2(y)$

Then

x_{i-}

Fi

method of

3.2 Example of an Hermite Interpolation Method

Consider the case for $N = 3$, $r = k = 2$. For simplicity, we consider only the computations done by the second processing element PE 2. The computations performed by the other two PE's are identical, except that they use different points. We suppress the subscript i to simplify the notation.

Let $A_2 = \{1,2\}$ so that PE 2 uses $x_{i,1} = x_1$ and $x_{i,2} = x_2$ to compute $x_{i+1,2}$. The values $y_1 = f(x_1)$, $y_1' = f'(x_1)$, $y_2 = f(x_2)$, and $y_2' = f'(x_2)$ have been previously computed. The inverse interpolation polynomial of degree 3 which agrees with this data is given by

$$H_2(y) = \left[2\frac{y - y_1}{y_2 - y_1} + 1\right]\left[\frac{y - y_2}{y_1 - y_2}\right]^2 x_1 + [y - y_1]\left[\frac{y - y_2}{y_1 - y_2}\right]^2 \frac{1}{y_1'}$$

$$+ \left[2\frac{y - y_2}{y_1 - y_2} + 1\right]\left[\frac{y - y_1}{y_2 - y_1}\right]^2 x_2 + [y - y_2]\left[\frac{y - y_1}{y_2 - y_1}\right]^2 \frac{1}{y_2'} \quad .(3\text{-}3)$$

Then

$$x_{i+1,2} = H_2(0)$$

$$= \frac{1}{(y_1 - y_2)^3}\left[(3y_1 - y_2)x_1(y_2)^2 + (y_1 - 3y_2)x_2(y_1)^2\right]$$

$$- \frac{y_1 y_2}{(y_1 - y_2)^2}\left[\frac{y_2}{y_1'} + \frac{y_1}{y_2'}\right] \quad . \qquad (3\text{-}4)$$

Figure 3-1 illustrates the graphical interpretation of the method of equation (3-4).

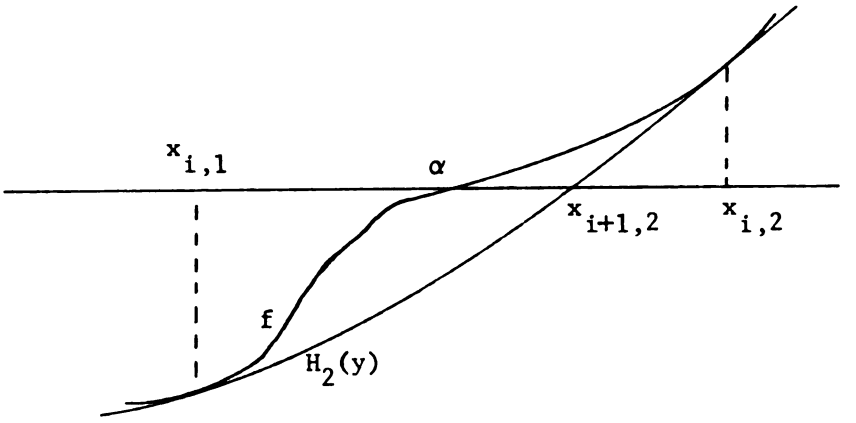


Figure 3-1. Inverse Hermite Interpolation by One PE

According to Theorem 3-1, this method has weak order of convergence 4. Note that it requires one evaluation of f and one evaluation of f' by each processing element so the information efficiency is

$$IE = \frac{\log 4}{2} = 1 \quad .$$

Equation (3-4) requires 22 arithmetic operations (if $(y_1 - y_2)^2$ is stored during the computation of $(y_1 - y_2)^3$), so the efficiency is

$$EFF = \frac{2}{c(f) + c(f') + 22} \quad .$$

The sequential method which this method generalizes has order of convergence $1 + \sqrt{3} \approx 2.732$ [40, p. 66], information efficiency $IE \approx 0.726$, and efficiency

$$EFF \approx \frac{0.726}{c(f) + c(f') + 22} \quad .$$

Thus the use of parallel processing has improved each of these measures. In Section 3.3, it will be shown that the speed-up ratio which can be achieved by methods like this one which are based on Hermite interpolation is proportional to $\log N$, so that they are not well suited to implementation on a parallel processing computer.

3.3 Efficiency

It was shown in Corollary 3-2 that the highest weak order of convergence which can be achieved on a parallel processing computer using N processing elements by inverse Hermite interpolation methods is $(N-1)(r+1)$. This order is achieved by methods which use values of $f, f', \ldots, f^{(r)}$ at $N-1$ components of the N-dimensional vector X_i. It is very costly to perform Hermite interpolation of f and its first r derivatives at $N-1$ points, so one might expect these methods to be inefficient. For example, the simple example of such methods constructed in Section 3.2 required 22 arithmetic operations. Although the interpolation becomes more and more expensive as the number of points used increases, we will show that if a very large number of arithmetic operations are required for each evaluation of $f, f', \ldots, f^{(r)}$, then the most efficient Hermite interpolation method which can be implemented on a given machine is a method which performs interpolation at $N-1$ points. That result will be used to show that the best speed-up ratio which can be achieved by parallel rootfinding methods using Hermite interpolation is proportional to $\log N$. Hence these methods are poorly suited for implementation on parallel processing computers.

Consider the inverse Hermite interpolation method ψ_k for parallel rootfinding described in Theorem 3-1. Assume that f is

analytic so that N, and hence k, may be chosen as large as we wish.

Definition 3-1. Define

$$P_{r,k}(f) = EFF(\psi_k,f) = \frac{\log_2 k(r+1)}{\sum_{i=0}^{r} c(f^{(i)}) + a} ,$$

where $c(f^{(i)})$ denotes the number of arithmetic operations needed to compute $f^{(i)}$, and a denotes the number of parallel arithmetic operations necessary to compute X_{i+1}. Also define

$$P_{r,*}(f) = \sup_{2\le k} P_{r,k}(f) .$$

If $r = 0$, so Lagrange interpolation is being used, $P_{0,k}(f) = P_k(f)$, and $P_{0,*}(f) = P_*(f)$ which were defined for Lagrange interpolation in Definition 2-2.

There are $\binom{\binom{N}{k}}{N}$ possible choices of N subsets of $\{1,2,\ldots,N\}$, so there are the same number of possible choices for ψ_k. In spite of this, $P_{r,k}(f)$ is well-defined since the possible choices for ψ_k differ only in their labeling of the points. Hence their order and computational complexities are the same.

$P_{r,*}(f)$ is the optimal efficiency of all inverse Hermite interpolation methods for parallel rootfinding. We will later ask whether N is large enough to allow $P_{r,*}(f)$ to be achieved.

As in Theorem 2-7, we will compare the efficiencies of parallel inverse Hermite interpolation methods with the sequential one-point and multipoint methods given by Traub in [19]. As in Section 2.4, let $E_1(f)$ denote the optimal efficiency of the class of sequential one-point iterations without memory. Let $E_2(f)$

denote the optimal efficiency of the class of sequential multipoint iterations without memory which use values of f only. We restate the following bounds from Theorem 1-1 and 1-2:

$$\frac{\log 3}{c(f) + c(f') + c(f'') + 7} \le E_1(f) \le \frac{\log 3}{3c_1 + 2} \tag{3-5}$$

$$\frac{1}{c(f)}\left(1 + \frac{\xi}{\sqrt{c(f)}}\right) \le E_2(f) \le \frac{1}{c(f)} \quad . \tag{3-6}$$

Here $c(f^{(i)})$ is the number of arithmetic operations needed to evaluate $f^{(i)}$, $c_1 = \min_{i\ge 0} c(f^{(i)})$, and ξ is some negative constant.

Theorem 3-3. Assume that f is analytic in a neighborhood of a simple root α. Let ψ_k be an inverse Hermite interpolation method given in Theorem 3-1. Let $r \ge 0$ be fixed. Let

$$c_1 = \min_{0\le i\le r} c(f^{(i)}), \text{ and } c_2 = \max_{0\le i\le r} c(f^{(i)}) \quad .$$

Then

i) $IE(\psi_k) = \dfrac{\log k(r+1)}{r+1}$.

ii) There exist positive constants ρ_1 and ρ_2 such that

$$\frac{\log k(r+1)}{(r+1)c_2 + \rho_2 k^2} \le P_{r,k}(f) \le \frac{\log k(r+1)}{(r+1)c_1 + \rho_1 k \ln^2 k} \quad , \tag{3-7}$$

for k sufficiently large.

iii) Let k_1 and k_2 denote the unique solutions to

$$u_1(k) = \rho_1 k \ln k\,(2 + \ln^2 k)\ln k(r+1) - (r+1)c_1 - \rho_1 k \ln^2 k = 0 \quad , \tag{3-8}$$

and

$$u_2(k) = 2\rho_2 (k_2)^2 \ln k(r+1) - (r+1)c_2 - \rho_2 (k_2)^2 = 0 \quad , \tag{3-9}$$

which

$2\rho_2$

iv) A

the s

point

v) A

speed

poin

k(r+

f,f'

tion

imp

to

com

which satisfy $k_1 \geq 1$, and $k_2 \geq 1/(r+1)$, respectively. Then

$$\frac{1}{2\rho_2(k_2)^2 \ln 2} \leq P_{r,*}(f) \leq \frac{1}{\rho_1(2 + \ln k_1)k_1 \ln k_1 \ln 2} \quad . \tag{3-10}$$

iv) Assume that $2 \leq k \leq N-1$. If c_1 is large and $c_1 = c_2$, then the speed-up ratio of ψ_k compared to the optimal sequential one-point methods without memory approaches

$$\frac{3}{(r+1)\ln 3} \ln (N-1), \text{ as } N \to \infty \quad . \tag{3-11}$$

v) Assume that $2 \leq k \leq N-1$. For large values of $c(f) = c$, the speed-up ratio of ψ_k compared to the optimal sequential multi-point iteration without memory using values of f only approaches

$$\frac{\ln (N-1)}{(r+1)\ln 2} \text{ , as } N \to \infty \quad . \tag{3-12}$$

Proof. i) By Theorem 3-1, ψ_k has order of convergence $k(r+1)$. One iteration of ψ_k requires the evaluation of $f,f',\ldots,f^{(r)}$ at each of the components of X_i. Hence $r+1$ functional evaluations are necessary.

ii) By definition,

$$c_1 \leq c(f^{(i)}) \leq c_2 \text{ , for } 0 \leq i \leq r \text{ ,}$$

implies that

$$(r+1)c_1 \leq \sum_{i=0}^{r} c(f^{(i)}) \leq (r+1)c_2 \quad . \tag{3-13}$$

Let a denote the number of arithmetic operations necessary to compute $x_{i+1,j}$, once $f,f',\ldots,f^{(r)}$ have been evaluated at the components of X_i. In [38], Strassen shows that interpolation on

k points requires at least $O(k \ln^2 k)$ operations. Several algorithms have been devised which perform interpolation at k points in $O(k^2)$ operations. Hence there exist positive constants ρ_1 and ρ_2 such that for k sufficiently large,

$$\rho_1 k \ln^2 k \leq a \leq \rho_2 k^2 \quad . \tag{3-14}$$

Applying the lower and upper bounds given by equations (3-13) and (3-14) to the definition of $P_{r,k}(f)$ gives (3-7).

iii) We first show that $u_1(k)$ has a unique solution ≥ 1. Let

$$\begin{aligned} u_1(t) &= \rho_1 t \ln t \,(2 + \ln^2 t) \ln t(r+1) \\ &\qquad - (r+1)c_1 - \rho_1 t \ln^2 t \\ &= 2\rho_1 t \ln t \ln t(r+1) - (r+1)c_1 \\ &\qquad + \rho_1 t \ln^2 t \,(\ln t(r+1) - 1) \quad . \end{aligned}$$

Then

$$u_1(1) = -(r+1)c_1 < 0, \text{ and } \lim_{t\to\infty} u_1(t) = +\infty \quad ,$$

so there exists a solution $k_1 \geq 1$ to equation (3-8). To see that k_1 is unique, notice that

$$\begin{aligned} u_1'(t) &= \rho_1(2 + 4 \ln t + \ln^2 t) \ln t(r+1) \\ &> 0 \text{ , for } t > 1 \quad . \end{aligned}$$

Similarly, to show that $u_2(t) = 0$ has a unique solution, we observe that

and

Henc

so

the

occu

Subs

(3-p

to e

Subs

and

$$u_2(1/(r+1)) = -(r+1)c_2 - \frac{\rho_2}{(r+1)^2} < 0 \quad ,$$

and that

$$\lim_{t\to\infty} u_2(t) = +\infty \quad .$$

Hence equation (3-9) has a solution $k_2 > 1/(r+1)$. But

$$u_2'(t) = 4\rho_2 t \ln t(r+1)$$

$$> 0 \text{ , for } t > 1/(r+1) \text{ ,}$$

so k_2 is unique.

To show (3-10), we take suprema of (3-7). The maxima of the expressions

$$\frac{\log k(r+1)}{(r+1)c_2 + \rho_2 k^2} \text{ , and } \frac{\log k(r+1)}{(r+1)c_1 + \rho_1 k \ln^2 k}$$

occur at k_2 and k_1, respectively. Hence

$$\frac{\log k_2(r+1)}{(r+1)c_2 + \rho_2 (k_2)^2} \leq P_{r,*}(f) \leq \frac{\log k_1(r+1)}{(r+1)c_1 + \rho_1 k_1 \ln^2 k_1} \quad . \quad (3\text{-}15)$$

Substituting equations (3-8) and (3-9) into the denominators of (3-15) gives (3-10).

iv) If we assume that all derivatives are equally costly to evaluate, then $c_1 = c_2 = c$. Dividing (3-10) by (3-5) gives

$$\frac{3c + 2}{2\rho_2 (k_2)^2 \ln 3} \leq \frac{P_{r,*}(f)}{E_1(f)} \leq \frac{3c + 7}{\rho_1 (2 + \ln k_1) k_1 \ln k_1 \ln 3} \quad .$$

Substitution for c in terms of k_1 and k_2 from equations (3-8) and (3-9) gives

$$\frac{3[2\rho_2(k_2)^2 \ln k_2(r+1) - \rho_2(k_2)^2] + 2(r+1)}{(r+1)2\rho_2(k_2)^2 \ln 3} \le \frac{P_{r,*}(f)}{E_1(f)}$$

$$\le \frac{3[\rho_1(2 + \ln k_1)k_1 \ln k_1 \ln k_1(r+1) - \rho_1 k_1 \ln^2 k_1] + 7(r+1)}{(r+1)\rho_1(2 + \ln k_1)k_1 \ln k_1 \ln 3} ,$$

so that

$$\frac{3 \ln k_2}{(r+1)\ln 3} + \frac{6 \ln(r+1) - 3}{2(r+1)\ln 3} + \frac{1}{\rho_2(k_2)^2 \ln 3}$$

$$\le \frac{P_{r,*}(f)}{E_1(f)} \le \frac{3 \ln k_1}{(r+1)\ln 3} - \frac{3}{(r+1)(\frac{2}{\ln k_1} + 1)\ln 3}$$

$$+ \frac{3 \ln(r+1)}{(r+1)\ln 3} + \frac{7}{\rho_1(2 + \ln k_1)k_1 \ln k_1 \ln 3} . \qquad (3\text{-}16)$$

From equations (3-8) and (3-9),

$$\lim_{c_1 \to \infty} k_1 = \lim_{c_2 \to \infty} k_2 = \infty ,$$

so for large values of c, (3-16) shows that the speed-up ratio is approximately bounded below by

$$\frac{3 \ln k_2}{(r+1)\ln 3} ,$$

and above by

$$\frac{3 \ln k_1}{(r+1)\ln 3} .$$

For a parallel processing computer with N PE's, if c is so large that $k_2 \ge N-1$, then the most efficient Hermite interpolation method which can be implemented uses $k = N-1$ points. As larger machines are used, the optimal efficiency $P_{r,*}(f)$ is approached.

Hence as $N \to \infty$, $\frac{P_{r,*}(f)}{E_1(f)}$ is approximately bounded above and below by

$$\frac{3 \ln (N-1)}{(r+1) \ln 3} .$$

v) The proof of v) is similar to the proof of iv) given above. Dividing (3-10) by (3-6) gives

$$\frac{c}{2\rho_2 (k_2)^2 \ln 2} \leq \frac{P_{r,*}(f)}{E_2(f)} \leq \frac{c}{(1 + \frac{\xi}{\sqrt{c}})\, \rho_1 (2 + \ln k_1) k_1 \ln k_1 \ln 2} .$$

Substituting for c in terms of k_1 and k_2 from equations (3-8) and (3-9) and simplifying, we get

$$\frac{\ln k_2}{(r+1)\ln 2} + \frac{2\ln(r+1) - 1}{2(r+1)\ln 2} \leq \frac{P_{r,*}(f)}{E_2(f)}$$

$$\leq \frac{\ln k_1}{(1 + \frac{\xi}{\sqrt{c}})(r+1)\ln 2} + \frac{\ln(r+1)}{(1 + \frac{\xi}{\sqrt{c}})(r+1)\ln 2}$$

$$- \frac{1}{(1 + \frac{\xi}{\sqrt{c}})(r+1)(\frac{2}{\ln k_1} + 1)\ln 2} . \qquad (3\text{-}17)$$

If c is large, k_1 and k_2 are also large, so the speed-up ratio is approximately bounded below and above by

$$\frac{\ln k_2}{(r+1) \ln 2} , \text{ and } \frac{\ln k_1}{(r+1) \ln 2} ,$$

respectively. Hence for large values of c, the speed-up ratio behaves like

$$\frac{\ln (N-1)}{(r+1) \ln 2} , \text{ as } N \to \infty . \quad \blacksquare$$

The important conclusions from Theorem 3-3 are contained in parts iv) and v). Taken together they show that if parallel root-finding methods based on Hermite interpolation are compared to either optimal sequential one-point iterations without memory or to optimal sequential multipoint iterations without memory which use values of f only, the speed-up ratio is proportional to the $\log_2$ of the machine size. Hence these methods are not well suited for implementation on a parallel processing system.

3.4 Comparison of Hermite and Lagrange Methods

In Section 3.3, we studied the efficiency of inverse Hermite interpolation methods for parallel rootfinding which use $f, f', \ldots, f^{(r)}$ at k points to compute $x_{i+1,j}$. In that section, we considered the effect of increasing k to find the speed-up ratios of the methods. In this section, we suppose that k is fixed and let r vary to determine the optimal number of derivatives to be used for the Hermite interpolation.

Let φ_k denote the parallel rootfinding method based on inverse Lagrange interpolation described in Theorem 2-5. Let $\psi_{k,r}$ denote the method based on inverse Hermite interpolation described in Theorem 3-1. Note that φ_k is $\psi_{k,0}$. We will show that φ_k is more efficient than $\psi_{k,r}$.

Theorem 3-4. Assume that f is analytic in a neighborhood of a simple root α. Then

$$IE(\psi_{k,r}) \leq IE(\varphi_k), \text{ for all } r \geq 0 .$$

If $c(f) \leq c(f^{(i)})$, for $0 \leq i \leq r$, then for $r \geq 0$,

$$\mathrm{EFF}(\psi_{k,r}) \leq \mathrm{EFF}(\varphi_k) \quad .$$

Proof. Recall that

$$\mathrm{IE}(\varphi) = \frac{\log_2 (\text{order of convergence of } \varphi)}{\text{number of parallel functional evaluations}} \quad .$$

According to Theorem 3-3,

$$\mathrm{IE}(\psi_{k,r}) = \frac{\log\, k(r+1)}{r+1} \quad .$$

To find the highest possible IE, we differentiate with respect to r to get

$$\frac{d\ \mathrm{IE}}{dr} = \frac{\frac{r+1}{r+1} - \ln k(r+1)}{(r+1)^2 \ln 2}$$

$$= \frac{1 - \ln k(r+1)}{(r+1)^2 \ln 2} \quad .$$

Hence $\frac{d\ \mathrm{IE}}{dr} < 0$ if and only if $k(r+1) > e$. Recall that $2 \leq k \leq N-1$ and $r \geq 0$, so $\frac{d\ \mathrm{IE}}{dr} < 0$ unless $k = 2$ and $r = 0$. (Notice that $k = 2$ and $r = 0$ characterizes the parallel secant method.) In the case $k = 2$, $\mathrm{IE} = 1$ for both $r = 0$ and $r = 1$. For all other values of k and r, IE is a decreasing function of r, so for any fixed $k \geq 2$, the largest IE is always attained by φ_k. Hence

$$\mathrm{IE}(\psi_{k,r}) = \frac{\log\, k(r+1)}{r+1} \leq \log k = \mathrm{IE}(\varphi_k) \quad . \tag{3-18}$$

That is, Lagrange interpolation on k points has a higher IE than any other Hermite interpolation method on the same k points.

We now turn to the efficiences of the two methods. By assumption $c(f) \leq c(f^{(i)})$, for $0 \leq i \leq r$, so from (3-18),

$$\frac{\log k}{c(f)} \geq \frac{\log k(r+1)}{(r+1)c(f)}$$

$$\geq \frac{\log k(r+1)}{\sum_{t=0}^{r} c(f^{(i)})} \quad .$$

Let $a(\varphi)$ denote the number of parallel arithmetic operations used to combine the functional values to compute X_{i+1}. Clearly $a(\varphi_k) \leq a(\psi_{k,r})$, so that

$$EFF(\psi_{k,r}) = \frac{\log k(r+1)}{\sum_{t=0}^{r} c(f^{(i)}) + a(\psi_{k,r})}$$

$$\leq \frac{\log k(r+1)}{c(f) + a(\varphi_k)} = EFF(\varphi_k) \quad . \quad \blacksquare$$

We have shown that if we wish to use a parallel rootfinding algorithm based on interpolation at k points, the fastest method to use is based on Lagrange interpolation.

CHAPTER IV

ROOTFINDING BY DERIVATIVE-ESTIMATED METHODS

We now consider methods for finding the simple root α of the function f which use estimates for the values of one of the derivatives of f.

The Hermite interpolation methods studied in Chapter III use values of $f,f',\ldots,f^{(r)}$. In practice, it is often the case that some derivative of f, say $f^{(r)}$, is not available or is much more costly to evaluate than any of the other derivatives. If one can estimate the values of $f^{(r)}$ using the values of lower, more easily computed, derivatives, we may be able to retain the same order of convergence, while avoiding the need to evaluate $f^{(r)}$.

The parallel rootfinding methods considered in Chapter III were based on inverse Hermite interpolation of $f,f',\ldots,f^{(r)}$ at k points. Theorem 3-1 showed that these methods have weak order of convergence $k(r+1)$. If $f^{(r)}$ is either unavailable or too costly to evaluate, we can use an estimate for the values of $f^{(r)}$ at k points. The methods used to find the orders of convergence in Chapters II and III fail in the case of the derivative-estimated methods because of the form of the difference equations satisfied by the errors. Hence we will only be able to prove lower bounds for the orders of convergence of these methods. It is our conjecture that the lower bounds are exactly the orders. We will show

that if Hermite interpolation of $f, f', \ldots, f^{(r-1)}$ at $\lceil k \frac{r+1}{r} \rceil$ or more points is used to estimate $f^{(r)}$, the order of convergence of the method is not less than $k(r+1)$. ($\lceil t \rceil$ is the smallest integer which is $\geq t$.) We will then consider the conditions under which the derivative-estimated method is more efficient than the Hermite interpolation method on which it is based.

4.1 An Example of a Derivative-Estimated Method

Before studying the general derivative-estimated methods, we shall consider an example to illustrate the derivation of the general methods. We first derive the sequential method and then modify it to form a parallel derivative-estimated method.

This example is based on an exercise in Ralston [30, p. 388] which suggests that we consider the following sequential method:

$$x_{i+1} = x_i - \frac{f(x_i)}{\bar{f}'(x_i)} , \tag{4-1}$$

where

$$\bar{f}'(x_i) = \frac{(x_i - x_{i-1})f(x_{i-2})}{(x_{i-2} - x_{i-1})(x_{i-2} - x_i)} + \frac{(x_i - x_{i-2})f(x_{i-1})}{(x_{i-1} - x_{i-2})(x_{i-1} - x_i)}$$

$$+ \frac{f(x_i)}{x_i - x_{i-2}} + \frac{f(x_i)}{x_i - x_{i-1}} . \tag{4-2}$$

Equation (4-2) is obtained by differentiating the three point Lagrange interpolation polynomial, $G(x)$, to f at x_{i-2}, x_{i-1}, and x_i. According to Lemma 2-2,

$$E(x) = f(x) - G(x) = \frac{1}{6}(x - x_{i-2})(x - x_{i-1})(x - x_i)f'''(\xi(x)), \tag{4-3}$$

where $\xi(x)$ lies on the interval spanned by x, x_{i-2}, x_{i-1}, and

x_i. If $f \in C^{iv}$ in a neighborhood of α, differentiating both sides of equation (4-3) and substituting x_i into the resulting equation gives the well-known estimate

$$E'(x_i) = \frac{1}{6}(x_i - x_{i-2})(x_i - x_{i-1})f'''(\xi_i) \quad , \tag{4-4}$$

where $\xi_i = \xi(x_i)$.

Let

$$F(x_i) = x_i - f(x_i)/f'(x_i) \quad ,$$

and

$$H(x_i) = x_{i+1} = x_i - f(x_i)/\bar{f}'(x_i) \quad ,$$

where $\bar{f}'(x_i)$ is given by equation (4-2). Now F is the usual Newton-Raphson method, so it has error given by

$$\alpha - F(x_i) = -\tfrac{1}{2}f''(\zeta)(\alpha - x_i)^2/f'(x_i) \quad , \tag{4-5}$$

where ζ lies between x_i and α (see [30, p. 332]). Notice that F and H are the same except that in H, $\bar{f}'(x_i)$ from equation (4-2) is used as an estimate for $f'(x_i)$ which is used by F. Notice also that H requires only values of f.

To compute the order of convergence of H, let

$$\begin{aligned}
\epsilon_{i+1} &= \alpha - x_{i+1} = \alpha - H(x_i) \\
&= \alpha - F(x_i) + F(x_i) - H(x_i) \\
&= -\frac{f''(\zeta)}{2f'(x_i)}(\epsilon_i)^2 - f(x_i)\left[\frac{1}{f'(x_i)} - \frac{1}{\bar{f}'(x_i)}\right] \\
&= -\frac{f''(\zeta)}{2f'(x_i)}(\epsilon_i)^2 + \frac{f(x_i)E'(x_i)}{f'(x_i)\bar{f}'(x_i)}
\end{aligned}$$

$$= - \frac{f''(\zeta)}{2f'(x_i)} (e_i)^2 + \frac{f(x_i) f'''(\xi_i)}{6f'(x_i)\bar{f}'(x_i)} (x_i - x_{i-2})(x_i - x_{i-1})$$

$$= - \frac{f''(\zeta)}{2f'(x_i)} (e_i)^2 + \frac{f(x_i) f'''(\xi_i)}{6f'(x_i)\bar{f}'(x_i)} (e_{i-2} - e_i)(e_{i-1} - e_i), \tag{4-6}$$

since $x_i - x_{i-1} = e_{i-1} - e_i$ and $x_i - x_{i-2} = e_{i-2} - e_i$. Now $f(x_i) = f(x_i) - f(\alpha) = (x_i - \alpha) f'(\eta)$, where η lies between α and x_i, so

$$e_{i+1} = - \frac{f''(\zeta)}{2f'(x_i)} (e_i)^2 - \frac{f'(\eta) f'''(\xi_i)}{6f'(x_i)\bar{f}'(x_i)} e_i (e_{i-2} - e_i)(e_{i-1} - e_i) \ . \tag{4-7}$$

Observe that equation (4-7) implies that whenever x_0, x_1, and x_2 are chosen sufficiently close to α,

$$\lim_{i \to \infty} x_i = \alpha \ ,$$

so that the method converges.

From equation (4-7), it can be shown that this method has order of convergence 1.84 (see [40, p. 53]). The method requires one new functional evaluation and 16 arithmetic operations at each iteration. Hence the information efficiency is

$$IE = \frac{\log 1.84}{1} \approx 0.879 \ .$$

and the efficiency is

$$EFF \approx \frac{0.879}{c(f) + 16} \ .$$

We now show how this sequential derivative-estimated method is modified to construct a parallel derivative-estimated method for a parallel processing computer with $N = 3$ processing elements.

Let

$$x_{i+1,1} = x_{i,1} - f(x_{i,1})/\bar{f}'(x_{i,1}) \quad , \tag{4-8}$$

where

$$\bar{f}'(x_{i,1}) = \frac{(x_{i,1} - x_{i,3})f(x_{i,2})}{(x_{i,2} - x_{i,3})(x_{i,2} - x_{i,1})} + \frac{f(x_{i,1})}{x_{i,1} - x_{i,2}}$$

$$+ \frac{(x_{i,1} - x_{i,2})f(x_{i,3})}{(x_{i,3} - x_{i,2})(x_{i,3} - x_{i,1})} + \frac{f(x_{i,1})}{x_{i,1} - x_{i,3}} \, .$$

Let $x_{i+1,2}$ and $x_{i+1,3}$ be defined similarly.

To compute the order of convergence of this parallel method, let $\epsilon_{i,j} = \alpha - x_{i,j}$. Then equation (4-7) becomes

$$\epsilon_{i+1,1} = -\frac{f''(\zeta_1)}{2f'(x_{i,1})} (\epsilon_{i,1})^2$$

$$- \frac{f'(\eta_1)f'''(\xi_{i,1})}{6f'(x_{i,1})\bar{f}'(x_{i,1})} \epsilon_{i,1}(\epsilon_{i,2} - \epsilon_{i,1})(\epsilon_{i,3} - \epsilon_{i,1}), \tag{4-9}$$

with similar expressions holding for $\epsilon_{i+1,2}$ and $\epsilon_{i+1,3}$. This implies that if $x_{0,1}$, $x_{0,2}$, and $x_{0,3}$ are chosen sufficiently close to α, the method converges.

We will show in Theorem 4-1 that this method has order of convergence 2. One parallel functional evaluation and 16 arithmetic operations are necessary at each iteration, so the information efficiency is

$$IE = \frac{\log 2}{1} = 1 \, ,$$

and the efficiency is

$$EFF = \frac{1}{c(f) + 16} \quad .$$

Note that the use of parallel processing has improved the order, and hence the efficiency and the information efficiency, of the sequential method given by equation (4-1).

Although using an approximation for the value of $f'(x_i)$ needed for the usual Newton-Raphson method, F, lowered the order of convergence from 2 to 1.84 in the sequential case, using a similar approximation did not affect the order of convergence of the Newton-Raphson method in the parallel case. We will show that in many cases, an estimation may be used for the values of a derivative without lowering the order of the method.

One of the many variations possible on this example is of interest. Instead of using the data at three points to compute the approximation $\bar{f}'(x_{i,1})$ to $f'(x_{i,1})$, we could use just $f(x_{i,1})$ and $f(x_{i,2})$. Then

$$\bar{f}'(x_{i,1}) = \frac{f(x_{i,2}) - f(x_{i,1})}{x_{i,2} - x_{i,1}} \quad .$$

This is just the parallel secant method introduced in Section 1.4. Hence the parallel secant method may be viewed either as an example of a parallel method based on Lagrange interpolation, or as a derivative-estimated method based on the Newton-Raphson method.

4.2 Order of Convergence

In this section we will indicate that if enough points are used to estimate values of the highest derivative of f needed by certain rootfinding methods, the order of convergence is unchanged.

As in Chapters II and III, we assume that the analytic function f has an inverse, denoted by g, in a neighborhood of the simple root α.

Consider the following parallel generalization of the Newton-Raphson method. Let $x_{0,1}, x_{0,2}, \ldots, x_{0,N}$ be initial approximations chosen sufficiently close to α. Assume that $g \in C^{r+1}$ near 0. For each $j = 1,2,\ldots,N$, let $F_j(y)$ be the Hermite interpolation polynomial of degree at most r which satisfies

$$F_j^{(s)}(y_{i,j}) = g^{(s)}(y_{i,j}), \text{ for } s = 0,\ldots,r \ .$$

Define

$$x_{i+1,j} = F_j(0) \ .$$

Although this is a parallel method, each processor is acting independently of the others since $x_{i+1,j}$ depends only on $x_{i,j}$. Thus each processing element is performing a generalized Newton-Raphson iteration which is well known to have order of convergence $r+1$ (see [40, Chapter 3]).

Let us set the notation which is needed to construct a derivative-estimated method based on this parallel generalization of the Newton-Raphson method. Let $u \geq 2$ be fixed. Choose N subsets, not necessarily distinct, of $\{1,2,\ldots,N\}$, each containing u elements. Denote the j^{th} subset by B_j, and assume $j \in B_j$. Let $G_j(y)$ be the Hermite interpolation polynomial of degree at most $ur-1$ which interpolates

$$G_j^{(s)}(y_{i,t}) = g^{(s)}(y_{i,t}), \text{ for } s = 0,\ldots,r-1, \ t \in B_j \ .$$

We will use $G_j^{(r)}(y_{i,j})$ to approximate the value of $g^{(r)}(y_{i,j})$ which is used in F_j.

Theorem 4-1. Let $f \in C^{ur}$ in a neighborhood of the simple root α. Let F_j and G_j be as defined above. Let $H_j(y)$ be the Hermite interpolation polynomial of degree at most r which satisfies

$$H_j^{(s)}(y_{i,j}) = g^{(s)}(y_{i,j}) \quad , \text{ for } \quad s = 0,\ldots,r-1 \ ,$$

$$H_j^{(r)}(y_{i,j}) = G_j^{(r)}(y_{i,j}) \quad .$$

Define

$$x_{i+1,j} = H_j(0) \quad .$$

If the N components of X_0 are chosen sufficiently close to α, this method converges and has strong order of convergence of at least $r+1$.

The proof of this theorem is a generalization of the corresponding sequential case given in [40] by Traub. The first of two lemmas needed for this proof is a well-known generalization of Rolle's Theorem [40, p. 252].

Lemma 4-2. Let $q = \sum_{j=0}^{k} \gamma_j$. Let $W \in C^q(I)$, where I is an interval which contains the points x_j which are zeros of W of multiplicity γ_j, respectively, for $j = 0,1,\ldots,k$. Then $W^{(i)}$ has at least $q-i$ zeros counting multiplicity, for $i = 0,1,\ldots,q-1$, on I. In particular, there exists $\xi \in I$ such that $W^{(q-1)}(\xi) = 0$.

The second lemma deals with the error in estimating derivatives by Hermite interpolation. We include the proof for completeness.

Lemma 4-3 [40, p. 252]. Let $s = ru$, and let $g \in C^{s}(I)$, where I is an interval containing the distinct points $y_1, y_2, \ldots, y_u$. Let $G(y)$ be the Hermite interpolation polynomial of degree at most $s-1$ satisfying

$$G^{(i)}(y_j) = g^{(i)}(y_j) \quad , \text{ for } j = 1,\ldots,u, \text{ and } i = 0,\ldots,r-1 \ .$$

Then

$$g^{(r)}(y_1) - G^{(r)}(y_1) = \frac{r!}{s!} g^{(s)}(\xi) \prod_{j=2}^{u} (y_1 - y_j)^r \quad ,$$

where $\xi \in I$.

Proof. Define

$$P(t) = \prod_{j=1}^{u} (t - y_j)^r \ .$$

The r th derivative of P,

$$P^{(r)}(y_1) = r! \prod_{j=2}^{u} (y_1 - y_j)^r \neq 0 \quad ,$$

so we may define

$$\lambda = \frac{g^{(r)}(y_1) - G^{(r)}(y_1)}{P^{(r)}(y_1)} \ .$$

Let $W(t) = g(t) - G(t) - \lambda P(t)$. Then $W(t)$ has a zero of multiplicity $r+1$ at y_1, and a zero of multiplicity r at $y_2, y_3, \ldots, y_u$. Hence W has at least $s+1$ zeros, so by Lemma 4-2, there exists $\xi \in I$ such that $W^{(s)}(\xi) = 0$. Since $G(t)$ is a polynomial of degree at most $s-1$,

$$0 = W^{(s)}(\xi) = g^{(s)}(\xi) - \lambda s! \ .$$

Then

$$g^{(r)}(y_1) - G^{(r)}(y_1) = \lambda P^{(r)}(y_1)$$

$$= \frac{1}{s!} g^{(s)}(\xi) P^{(r)}(y_1)$$

$$= \frac{r!}{s!} g^{(s)}(\xi) \prod_{j=2}^{u} (y_1 - y_j)^r \quad . \blacksquare$$

We are now prepared to prove Theorem 4-1. The subscript i will be suppressed to simplify the notation.

Proof. $F_j(y)$ interpolates $g, g', \ldots, g^{(r)}$ at $y_j = y_{i,j}$, so it must have the form

$$F_j(y) = \sum_{s=0}^{r} \frac{g^{(s)}(y_j)}{s!} (y - y_j)^s \quad ,$$

with error given by

$$g(0) - F_j(0) = \frac{g^{(r+1)}(\theta)}{(r+1)!} (-y_j)^{r+1} \quad ,$$

where θ lies between 0 and y_j. Similarly, $H_j(y)$ is given by

$$H_j(y) = \sum_{s=0}^{r-1} \frac{g^{(s)}(y_j)}{s!} (y - y_j)^s + \frac{G_j^{(r)}(y_j)}{r!} (y - y_j)^r \quad .$$

From Lemma 4-3 we have

$$g^{(r)}(y_j) - G_j^{(r)}(y_j) = \frac{r!}{(ur)!} g^{(ur)}(\theta_j) \prod_{\substack{t \in B_j \\ t \neq j}} (y_j - y_t)^r \quad , \quad (4\text{-}10)$$

where θ_j lies on the interval spanned by $\{y_t \mid t \in B_j\}$. Now

$$e_{i+1,j} = \alpha - x_{i+1,j} = g(0) - H_j(0)$$

$$= g(0) - F_j(0) + F_j(0) - H_j(0)$$

$$= \frac{g^{(r+1)}(\theta)}{(r+1)!}(-y_j)^{r+1} + \frac{(-y_j)^r}{r!}\left[g^{(r)}(y_j) - G_j^{(r)}(y_j)\right]$$

$$= \frac{g^{(r+1)}(\theta)}{(r+1)!}(-y_j)^{r+1} + \frac{(-y_j)^r}{r!}\frac{r!}{(ur)!}g^{(ur)}(\theta_j)\prod_{\substack{t\in B_j \\ t\neq j}}(y_j - y_t)^r$$

$$= \frac{g^{(r+1)}(\theta)}{(r+1)!}\frac{(\epsilon_{i,j})^{r+1}}{[g'(\xi_j)]^{r+1}} + \frac{(\epsilon_{i,j})^r}{(ur)!}\frac{g^{(ur)}(\theta_j)}{[g'(\xi_j)]^r}\prod_{\substack{t\in B_j \\ t\neq j}}[f'(\lambda_{j,t})(x_j - x_t)^r , \tag{4-11}$$

since $-y_j = 0 - y_j = f(\alpha) - f(x_j) = f'(\eta_j)(\alpha - x_j) = \frac{\epsilon_{i,j}}{g'(\xi_j)}$,
where ξ_j is between 0 and y_j, and $y_j - y_t = f(x_j) - f(x_t)$
$= f'(\lambda_{j,t})(x_j - x_t)$, where $\lambda_{j,t}$ lies between x_j and x_t.

Let

$$M_{1,j} = \frac{g^{(r+1)}(\theta)}{(r+1)![g'(\xi_j)]^{r+1}} ,$$

and

$$M_{2,j} = \frac{g^{(ur)}(\theta_j)2^{r(u-1)}}{(ur)![g'(\xi_j)]^{r+1}} \cdot \prod_{\substack{t\in B_j \\ t\neq j}}[f'(\lambda_{j,t})]^r .$$

Then from equation (4-11),

$$\epsilon_{i+1,j} = M_{1,j}(\epsilon_{i,j})^{r+1} + \frac{M_{2,j}(\epsilon_{i,j})^r}{2^{r(u-1)}}\prod_{\substack{t\in B_j \\ t\neq j}}(\epsilon_{i,t} - \epsilon_{i,j})^r . \tag{4-12}$$

Let

$$M_1 = \max_j |M_{1,j}|$$

$$M_2 = \max_j |M_{2,j}| \quad .$$

Then if $\delta_i = \max_j |\epsilon_{i,j}|$,

$$|\epsilon_{i+1,j}| \leq M_1(\delta_i)^{r+1} + M_2(\delta_i)^r \prod_{\substack{t\in B_j \\ t\neq j}} (\delta_i)^r$$

$$\leq M_1(\delta_i)^{r+1} + M_2(\delta_i)^{ur} \quad .$$

Now

$$\lim_{i\to\infty} M_1 = \frac{g^{(r+1)}(0)}{(r+1)![g'(0)]^{r+1}} \quad , \quad \text{and}$$

$$\lim_{i\to\infty} M_2 = \frac{g^{(ur)}(0)\ 2^{r(u-1)}}{(ur)![g'(0)]^r} \quad .$$

Hence M_1 and M_2 are bounded by $\bar{M}_1$ and $\bar{M}$, respectively, for all i. If the N components of X_0 are chosen sufficiently close to α, then

$$\delta_{i+1} \leq \bar{M}_1(\delta_i)^{r+1} + \bar{M}(\delta_i)^{ur} \quad ,$$

so

$$\lim_{i\to\infty} \delta_i = 0$$

and the method converges. Since $2 \leq u$, the inequality

$$\frac{\delta_{i+1}}{(\delta_i)^{r+1}} \leq \bar{M}_1 + \bar{M}(\delta_i)^{ur-r-1}$$

implies that

$$\limsup_{i\to\infty} \frac{\delta_{i+1}}{(\delta_i)^{r+1}} \le \bar{M}_1 \quad ,$$

showing that the method has strong order of convergence at least $r+1$. ∎

We are unable to prove the exact order of convergence because of the form of equation (4-12). However, it is our conjecture that:

Conjecture. The parallel derivative-estimated method given in Theorem 4-1 which defines $x_{i+1,j} = H_j(0)$ has order of convergence exactly $r+1$.

We have shown that the order of convergence of this method is at least $r+1$. Recall that the method from which it is derived uses $g^{(r)}$ and has order of convergence $r+1$. It seems highly unlikely that using an approximation for $g^{(r)}$ instead of $g^{(r)}$ itself could make the method converge more rapidly for a general choice of the function f.

How many points u should be used for the Hermite interpolation from which the derivative is estimated? Theorem 4-1 showed that the order of convergence of the sequential generalized Newton-Raphson method is not lowered if $g^{(r)}$ is approximated by $G_j^{(r)}$, where G_j is the Hermite interpolation polynomial which agrees with $g, g', \ldots, g^{(r-1)}$ at $u \ge 2$ points. Notice that u does not effect the order of convergence, so we should use just two points in order to minimize the effort needed to compute G_j.

It does not appear that the stepping down from r to $r-1$ derivatives in Theorem 4-1 can be repeated since that would require an analogue for Lemma 4-3 giving an estimate for $g^{(r+1)}(y_1) - G^{(r+1)}(y_1)$. We can, however, extend Theorem 4-1 by replacing the Newton-Raphson method, F_j, with the inverse Hermite

interpolation methods discussed in Chapter III which interpolate $g, g', \ldots, g^{(r)}$ at k points.

In setting up the notation for the general parallel derivative-estimated methods, we must be very careful to distinguish between three different Hermite interpolation polynomials, F_j, G_j, and H_j, being used.

Choose N distinct subsets of k elements from the set $\{1,2,\ldots,N\}$, for a fixed k, $2 \leq k \leq N-1$. Denote the j th subset by A_j, and assume that $j \in A_j$. The Hermite interpolation polynomials F_j and H_j will interpolate different data on the points $y_{i,t}$, for $t \in A_j$. Let $r \geq 1$ be fixed. Let F_j be the polynomial of degree at most $k(r+1) - 1$ which interpolates

$$F_j^{(s)}(y_{i,t}) = g^{(s)}(y_{i,t}), \text{ for } s = 0,\ldots,r, \; t \in A_j \; .$$

This is the same polynomial which was used in the construction of the method ψ_k in Theorem 3-1.

As in the method considered in Theorem 4-1, we use the r th derivative of the Hermite interpolation polynomial G_j to approximate the values of $g^{(r)}$. To construct G_j, choose N subsets, not necessarily distinct, of u elements from the set $\{1,2,\ldots,N\}$, for a fixed u, $2 \leq u \leq N$. Denote the j th subset by B_j, and assume that $j \in B_j$. We remark that k may not equal N, while u may, because the sets A_j must be distinct, while some or all of the sets B_j may be the same. Let $G_j(y)$ be the Hermite interpolation polynomial of degree at most $ur-1$ which interpolates

$$G_j^{(s)}(y_{i,t}) = g^{(s)}(y_{i,t}), \text{ for } s = 0,\ldots,r-1, \; t \in B_j \; .$$

Now we use $G_j^{(r)}$ as an approximation for $g^{(r)}$ in F_j, so that $H_j(y)$ is the Hermite interpolation polynomial of degree at most $k(r+1) - 1$ which interpolates

$$\left.\begin{aligned} H_j^{(s)}(y_{i,t}) &= g^{(s)}(y_{i,t}), \text{ for } s = 0,\ldots,r-1 \\ H_j^{(r)}(y_{i,t}) &= G_j^{(r)}(y_{i,t}) \end{aligned}\right\} \; t \in A_j \quad .$$

Theorem 4-4. Let f be analytic in a neighborhood of a simple root α. Let $x_{0,1}, x_{0,2}, \ldots, x_{0,N}$ be chosen sufficiently close to α. For each $j = 1,2,\ldots,N$, let F_j, G_j, and H_j be the Hermite interpolation polynomials defined above. Define

$$x_{i+1,j} = H_j(0) \quad .$$

Then this method has strong order of convergence at least $\gamma = \min\{k(r+1), ur\}$.

Definition 4-1. Call the Hermite interpolation method considered in Chapter III which defines $x_{i+1,j} = F_j(0)$ the parent method. Call the parallel derivative-estimated method which defines $x_{i+1,j} = H_j(0)$ the derived method.

Proof. As before, the subscript i is omitted to simplify the notation.

Now F_j and H_j are given by the linear combinations

$$F_j(0) = \sum_{s=0}^{r} \sum_{m \in A_j} a_{s,m} g^{(s)}(y_m) \quad ,$$

and

$$H_j(0) = \sum_{s=0}^{r-1} \sum_{m \in A_j} a_{s,m} g^{(s)}(y_m) + \sum_{m \in A_j} a_{r,m} G_j^{(r)}(y_m) \quad .$$

From the proof of Theorem 3-1, there exist a sequence of constants, $K_{i,j}$, such that

$$|K_{i,j}| \leq K \text{ , for all } i \text{ and } j \text{ ,}$$

and

$$g(0) - F_j(0) = K_{i,j} \prod_{m \in A_j} (\epsilon_{i,m})^{r+1} \quad .$$

By Lemma 4-3,

$$g^{(r)}(y_j) - G_j^{(r)}(y_j) = \frac{r!}{(ur)!} g^{(ur)}(\theta_j) \prod_{\substack{t \in B_j \\ t \neq j}} (y_j - y_t)^r \text{ ,}$$

where θ_j lies on the interval spanned by $\{y_t | t \in B_j\}$. As in the proof of Theorem 4-1,

$$\epsilon_{i+1,j} = \alpha - x_{i+1,j} = g(0) - H_j(0)$$

$$= K_{i,j} \prod_{m \in A_j} (\epsilon_{i,m})^{r+1} + M_{i,j}(\epsilon_{i,j})^r \prod_{\substack{t \in B_j \\ t \neq j}} (\epsilon_{i,t} - \epsilon_{i,j})^r \text{ ,}$$

where

$$M_{i,j} = \frac{g^{(ur)}(\theta_j)\, 2^{r(u-1)}}{(ur)![g'(\xi_j)]^{r+1}} \prod_{\substack{t \in B_j \\ t \neq j}} \left(\frac{1}{g'(\lambda_{j,t})}\right)^r \quad .$$

ξ_j lies between 0 and y_j; $\lambda_{j,t}$ lies between y_j and y_t. In the proof of Theorem 4-1, it was shown that $|M_{i,j}| \leq \bar{M}$, so that

$$|\epsilon_{i+1,j}| \leq K \prod_{m \in A_j} |\epsilon_{i,m}|^{r+1} + \bar{M} |\epsilon_{i,j}|^r \prod_{\substack{t \in B_j \\ t \neq j}} |\epsilon_{i,t} - \epsilon_{i,j}|^r$$

$$\leq K(\delta_i)^{k(r+1)} + \bar{M}(\delta_i)^{ur} \quad .$$

Hence

$$\delta_{i+1} \leq K(\delta_i)^{k(r+1)} + \bar{M}(\delta_i)^{ur} \quad . \tag{4-13}$$

If $x_{0,1}, x_{0,2}, \ldots, x_{0,N}$ are chosen sufficiently close to α, then (4-13) implies that

$$\lim_{i \to \infty} \delta_i = 0 \quad .$$

Thus the method converges. Moreover, since $\gamma = \min\{k(r+1), ur\}$, the inequality

$$\frac{\delta_{i+1}}{(\delta_i)^{\gamma}} \leq K(\delta_i)^{k(r+1)-\gamma} + \bar{M}(\delta_i)^{ur-\gamma}$$

implies that

$$\limsup_{i \to \infty} \frac{\delta_{i+1}}{(\delta_i)^{\gamma}} < \infty \quad .$$

Thus the method has strong order of convergence at least γ. ∎

We conjecture that if $g^{(ur)}(0) \neq 0$ and $g^{(kr)}(0) \neq 0$, then these methods have strong order of convergence exactly γ. In that case, the order of convergence of a derived method may equal, but may not exceed, the order of convergence of its parent method. Since

$$\gamma = \min\{k(r+1), ur\} = \begin{cases} k(r+1), & \text{if } u \geq k(r+1)/r \\ ur, & \text{if } u \leq k(r+1)/r \end{cases} ,$$

the orders of the derived and the parent methods are equal whenever $\lceil k(r+1)/r \rceil$ or more points are used to estimate the values of $g^{(r)}$. For the method considered in Theorem 4-1, $k = 1$, $r \geq 1$,

so $(r+1)/r \leq 2$. Hence in that case, the orders of the derived and parent methods are equal if two or more points are used in the derivative estimation.

In Theorem 4-4, $2 \leq k \leq N-1$, and $2 \leq u \leq N$, so that we have this result.

Corollary 4-5. If the strong order of convergence of the derivative-estimated method for parallel rootfinding described in Theorem 4-4 is exactly γ, then the highest order of convergence which can be achieved on a parallel machine using N processing elements is either

$$\begin{cases} (N-1)(r+1), & \text{if } N \leq r+1, \text{ or} \\ Nr, & \text{if } N \geq r+1. \end{cases}$$

This shows that unless the number of derivatives used in the parent method is as large as the machine size N, use of a derivative-estimated method actually reduces the order of convergence.

4.3 Efficiency

In this section, the efficiency of the parallel derivative-estimated methods is studied by comparing the computational effort required by the derived method with that required by the parent method. We will assume that the derived methods have strong order of convergence exactly $\gamma = \min\{k(r+1), ur\}$.

If an order of convergence of $k(r+1)$ is desired, it can be achieved by the parent method using Hermite interpolation of $g, g', \ldots, g^{(r)}$ at the k points $\{y_{i,t} | t \in A_j\}$. A derived method

with $u \geq \lceil k(r+1)/r \rceil$ retains the same order of convergence, but it does not require the computation and evaluation of $g^{(r)}$. This is an important consideration if $g^{(r)}$ is much more costly to evaluate than $g, g', \ldots, g^{(r-1)}$. However, the cost of avoiding the evaluation of $g^{(r)}$ is high. At least $\lceil k/r \rceil$ additional processing elements are needed to perform the computations required at u points. In addition, each processing element must compute an Hermite interpolation polynomial G_j of degree at most $ur-1$ and evaluate $G_j^{(r)}$ at k points. Hence the parent method is more efficient than the derived method unless at least $\lceil k/r \rceil$ additional PE's are available and the cost of one parallel evaluation of $g^{(r)}$ is greater than the cost of constructing and evaluating G_j. This would be expected to occur only very rarely.

If instead of fixing the desired order, we wish to use the data from $g, g', \ldots, g^{(r)}$ each evaluated at k points, we may proceed in two ways. First, the Hermite interpolation method using this data has order of convergence $k(r+1)$. Alternatively, we can use the data in a derivative-estimated method to approximate $g^{(r+1)}$ at m points, where $m \leq k$. In that case, the order of convergence is $\min\{m(r+2), k(r+1)\}$. If $k(r+1)/(r+2) \leq m \leq k$, both methods have order of convergence $k(r+1)$. Now both of these two methods have the same order, both require the construction of one Hermite interpolation polynomial of degree at most $k(r+1) - 1$, and both require the same functional evaluations. The Hermite method requires one evaluation of its polynomial, while the derived method requires m evaluations of the $(r+1)$st derivative of its polynomial. In addition, the derived method requires the construction and one

evaluation of an Hermite interpolation polynomial of degree at most $k(r+2) - 1$. Hence the derived method requires more effort to achieve the same order of convergence as the Hermite method, so it is less efficient.

If $m < k(r+1)/(r+2)$, the Hermite method shows an even stronger advantage over the derived method. The computations needed by the derived method are the same as those outlined in the preceding paragraph, but the order of convergence is $m(r+2)$ which is less than the $k(r+1)$ order of the Hermite method. That is, the derived method yields a lower order of convergence, despite a higher computational effort.

Thus we have shown that in general, the parent Hermite interpolation methods are more efficient than the derivative-estimated methods because more computational effort is required by the derived methods to achieve the same order of convergence.

CHAPTER V

WHICH METHOD TO USE

In this paper we have developed several different classes of parallel rootfinding algorithms and considered their relative efficiencies. Which of these methods should be used to find a simple root α of a given function? We will show that the choice depends on the function f and on how fast the answer is needed.

5.1 Real-Time Problems

If the rootfinding problem is part of a real-time problem, the root α must be found as quickly as possible, even if the cost is high. For each of the parallel methods considered in this paper, it has been shown that if the cost of functional evaluation is very high, then the speed-up achieved is proportional to $\log_2 N$. That means that the parallel methods can solve some problems much more quickly than currently used sequential methods. To see which of the parallel methods to use, let us consider each class of parallel rootfinding algorithms studied in this paper.

In Chapter IV, it was shown that parallel derivative-estimated methods are less efficient than parallel inverse Hermite interpolation methods for nearly all choices of f. Hence, except in the rare case where one derivative of f is much more expensive to evaluate than all of the lower derivatives of f, the methods using Hermite interpolation are preferred over parallel derivative-estimated methods.

Thus, one should use inverse Hermite interpolation rather than derivative-estimated methods. It only remains to choose the number of points, k, and the number of derivatives, r, on which to perform interpolation. In Section 3.4, we showed that if k is fixed and the cost of evaluating each derivative is at least as high as the cost of evaluating f, then the efficiency decreases as the number of derivatives used increases. In this case, parallel rootfinding methods based on Lagrange interpolation on k points are more efficient than any other Hermite interpolation method on k points. If, however, some derivatives are much easier to evaluate than f, then Hermite interpolation methods may be faster than methods using only Lagrange interpolation.

Assuming that f is such that inverse Lagrange interpolation is faster than inverse Hermite interpolation, we may apply the results of Chapter II to show how many points should be used for interpolation. Let $\beta \geq 1$ denote the unique root of

$$u(k) = (4k^2 + k) \ln k - (c + 2k^2 + k - 1) = 0 \quad .$$

β need not be an integer, so we consider either $k_2 = \lceil \beta \rceil$ or $k_1 = \lceil \beta \rceil - 1$, where $\lceil \beta \rceil$ is the smallest integer $\geq \beta$. Recall that the efficiency of these methods based on inverse Lagrange interpolation is given by

$$P_k(f) = \frac{\log_2 k}{c(f) + 2k^2 + k - 1} \quad .$$

Then the number of points which should be used for the interpolation on a parallel machine using N processing elements is either

$$\begin{cases} N-1, \text{ if } \beta \geq N-1, \text{ or} \\ \text{whichever of } k_1 \text{ or } k_2 \text{ maximizes } P_k(f) \end{cases} .$$

5.2 Problems to be Solved at Low Cost

For other than real-time applications, the root α should be found with as little cost as possible. All of the parallel root-finding algorithms studied in this paper have been shown to achieve a speed-up ratio proportional to $\log_2$ of the number of processors used. This means that if a sequential machine can compute α to a specified accuracy in 10 seconds, a parallel machine can do the same in

$$\sigma \frac{10}{\log_2 N}$$

seconds, so that doubling the number of processing elements used has little effect on the time required. Thus, because of the high cost of a parallel processing system, rootfinding problems should be solved on a sequential machine, leaving the parallel system free to execute other tasks which use its parallel structure in a more efficient manner.

BIBLIOGRAPHY

BIBLIOGRAPHY

1. Ackins, Gary M., Morris A. Knapp, and John Thomas, "Applications of ILLIAC IV to Urban Defense Radar Problem," in Parallel Processor Systems, Technologies, and Applications, edited by Hobbs, L.C., D.J. Theis, Joel Trimble, Harold Titus, and Ivar Highberg, Spartan Books: New York (1970) pp. 23-70.

2. Amdahl, G.M., "Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities," in 1967 Spring Joint Comput. Conf., AFIPS Conf. Proc., 30, Thompson: Washington, D.C. (1967) p. 483.

3. Barnes, George H., Richard M. Brown, Maso Kato, David J. Luck, Daniel L. Slotnick, and Richard A. Stokes, "The ILLIAC IV Computer," IEEE Transactions on Computers, C-17, 8 (August, 1968) pp. 746-757.

4. Brent, Richard P., Algorithms for Minimization Without Derivatives, Prentice-Hall: Englewood Cliffs, New Jersey (1973).

5. Brent, Richard P., Shmuel Winograd, and Philip Wolfe, "Optimal Iterative Processes for Root-Finding," Numerische Mathematic, 20 (April, 1973) pp. 327-341.

6. Carrol, A.B., and R.T. Wetherford, "Applications of Parallel Processing to Numerical Weather Forecasting," Assoc. Comput. Mach. J. (1967) pp. 591-614.

7. Collatz, L., Functional Analysis and Numerical Mathematics, Academic Press: New York (1964).

8. Courtney, J.E., and H.M. Halpern, "Parallel Processing for Phased-Array Radars," in Parallel Processor Systems, Technologies, and Applications, edited by Hobbs, et al., pp. 87-106.

9. Danenberg, S.A., "An Introductory Description of the ILLIAC IV System," ILLIAC Document no. 225, University of Illinois (1970).

10. Flynn, M.J., "Some Computer Organizations and Their Effectiveness," IEEE Transactions on Computers, C-21, 9 (September, 1972) pp. 948-960.

11. Flynn, M.J., "Very High Speed Computing Systems," Proceedings IEEE, 54, 12 (December, 1966) pp. 1901-1909.

12. Githens, J.A., "An Associative, Highly-Parallel Computer for Radar Data Processing," in Parallel Processor Systems, Technologies, and Applications, edited by Hobbs, et al., pp. 71-86.

13. Grahm, William R., "The Parallel and the Pipeline Computers," Datamation (April, 1970) pp. 68-71.

14. Hobbs, L.C., and D.J. Theis, "Survey of Parallel Processor Approaches and Techniques," in Parallel Processor Systems, Technologies, and Applications, edited by Hobbs, et al., pp. 3-20.

15. Karp, Richard M., and Willard L. Miranker, "Parallel Minimax Search for a Maximum," J. Comb. Th., 4 (1968) pp. 19-35.

16. Kogge, Peter M., and Harold Stone, "A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations," IEEE Transactions on Computers, C-22, 8 (August, 1973) pp. 786-793.

17. Kuck, D.J., "ILLIAC IV Software and Applications Programming," IEEE Transactions on Computers, C-17, 8 (August, 1968) pp. 758-770.

18. Kung, H.T., "A Bound on the Multiplication Efficiency of Iteration," Proceedings of the Fourth Annual ACM Symposium on Theory of Computing (1972).

19. Kung, H.T., and J.F. Traub, "Computational Complexity of One-Point and Multipoint Iteration," Department of Computer Science, Carnegie-Mellon University, Pittsburg (April, 1973).

20. Kung, H.T., and J.F. Traub, "Optimal Order and Efficiency for Iterations with Two Evaluations," Department of Computer Science, Carnegie-Mellon University, Pittsburg (November, 1973).

21. Kung, H.T., and J.F. Traub, "Optimal Order of One-Point and Multipoint Iteration," Department of Computer Science, Carnegie-Mellon University, Pittsburg (February, 1973).

22. McIntyre, David E., "An Introduction to the ILLIAC IV Computer," Datamation (April, 1970) pp. 60-67.

23. Miranker, W.L., "Parallel Methods for Approximating the Root of a Function," IBM Journal of Research and Development, 13 (1969) pp. 297-301.

24. Miranker, Willard L., "A Survey of Parallelism in Numerical Analysis," SIAM Review, 13 (1971) pp. 525-547.

25. Munro, Ian, and Michael Paterson, "Optimal Algorithms for Parallel Polynomial Evaluation," Conference Record, 12 th Annual Symposium on Switching and Automata Theory, Michigan State University, IEEE Publication 71 C 45-C (October, 1971).

26. Ortega, J.M., and W.C. Rheinboldt, Iterative Solution of Nonlinear Equations in Several Variables, Academic Press: New York, 1970.

27. Ostrowski, A.M., Solution of Equations and Systems of Equations, Second Edition, Academic Press: New York (1966).

28. Owens, Jerry L., "The Influence of Machine Organization on Algorithms," in Complexity of Sequential and Parallel Numerical Algorithms, edited by Traub, J.F., Academic Press: New York (1973) pp. 111-130.

29. Pease, Marshall, "An Adaptation of Fast Fourier Transform for Parallel Processing," JACM, 15 (April, 1968) pp. 252-264.

30. Ralston, Anthony, A First Course in Numerical Analysis, McGraw-Hill: New York (1965).

31. Reddy, D. Raj, "Some Numerical Problems in Artificial Intelligence: Implications for Complexity and Machine Architecture," in Complexity of Sequential and Parallel Numerical Analysis, edited by Traub, pp. 131-148.

32. Rice, John R., "Matrix Representations of Nonlinear Equation Iterations - Applications to Parallel Computation," Mathematics of Computation, 25, 116 (October, 1971) pp. 639-647.

33. Rissanen, J., "On Optimal Root-Finding Algorithms," IBM Report RJ726 (No. 13830).

34. Rogers, Richard, Uniformly Accurate Numerical Solutions to Differential Equations Using Extrapolation and Interpolation, Ph.D. Thesis, Michigan State University (1974).

35. Stone, Harold S., "An Efficient Parallel Algorithm for the Solution of a Tridiagonal Linear System of Equations," JACM, 20, 1 (January, 1973) pp. 27-38.

36. Stone, Harold S., "Parallel Processing with the Perfect Shuffle," IEEE Transactions on Computers, C-20, 2 (February, 1971) pp. 153-161.

37. Stone, Harold S., "Problems of Parallel Computation," in Complexity of Sequential and Parallel Numerical Analysis, edited by Traub, pp. 1-16.

38. Strassen, V., "Gaussian Elimination is Not Optimal," Numerische Mathematic, 13 (1969) pp. 354-356.

39. Traub, J.F., "Computational Complexity of Iterative Processes," Department of Computer Science, Carnegie-Mellon University, Pittsburg (October, 1971).

40. Traub, J.F., Iterative Methods for the Solution of Equations, Prentice-Hall: Englewood Cliffs, New Jersey (1964).

41. Traub, J.F., "Theory of Optimal Algorithms," Department of Computer Science, Carnegie-Mellon University, Pittsburg (June, 1973).

42. Wulf, William A., "C.mmp: A Multi-Mini-Processor," in Computer Science Research Review, Carnegie-Mellon University, Pittsburg (1972) pp. 37-73.

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* Parallel Rootfinding Algorithms		5. TYPE OF REPORT & PERIOD COVERED 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR*(s)* George Frederick Corliss		8. CONTRACT OR GRANT NUMBER*(s)* AFOSR-72-2271
9. PERFORMING ORGANIZATION NAME AND ADDRESS Michigan State University Department of Mathematics East Lansing, MI 48824		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Air Force Office of Scientific Research 1400 Wilson Blvd. Arlington, VA 22209		12. REPORT DATE August, 1974 13. NUMBER OF PAGES 115
14. MONITORING AGENCY NAME & ADDRESS*(if different from Controlling Office)*		15. SECURITY CLASS. *(of this report)* 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

parallel processing, rootfinding, iterative methods, order of convergence, computational efficiency, Hermite interpolation

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

Algorithms to find a simple root α of a continuous function f are considered which are suitable for implementation on an array type of parallel processing computer using N processing elements. The algorithms are compared on the basis of their efficiency

$$\text{EFF} = \frac{\log_2 \text{order of convergence}}{\text{number of parallel arithmetic operations per iteration}}$$

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

The first class of algorithms uses inverse Lagrange interpolation on k points. If $f \in C^{(k)}$ on an interval containing α, these methods have order of convergence k. k can be chosen, depending on f, to maximize the efficiency of methods in this class. If the cost of evaluating f is very high, the speed-up possible by using parallel root-finding algorithms based on Lagrange interpolation over efficient sequential rootfinding algorithms is proportional to $\log_2 N$.

The second class of parallel algorithms considered uses inverse Hermite interpolation of $f, f', \ldots, f^{(r)}$ at k points. The order of convergence is $k(r+1)$. If r is fixed, the fastest algorithm in this class achieves a speed-up ratio proportional to $\log_2 N$ for functions which are very costly to evaluate f than to evaluate any derivative of f, a method using Lagrange interpolation $(r = 0)$ on k points is more efficient than any other method using Hermite interpolation on k points.

The third class of parallel algorithms are derivative-estimated methods. If the values of $f^{(r)}$ used in a parallel Hermite interpolation method are estimated using values of $f, f', \ldots, f^{(r-1)}$ at enough points. the order of convergence of the Hermite interpolation method is not reduced. However, except in rare examples. a parallel derivative-estimated method is less efficient than the Hermite interpolation method on which it is based.

For real-time computer applications, these parallel algorithms should be used. In general, a parallel method using inverse Lagrange interpolation on k points is fastest, where k depends on the function.

For other applications the rootfinding task should be executed on a sequential machine.