1973

Parallel Algorithms for Adaptive Quadrature - Convergence

John R. Rice
Purdue University, jrr@cs.purdue.edu

Report Number:
73-104

Parallel Algorithms for Adaptive Quadrature-Convergence

John R. Rice

September, 1973

CSD TR 104

PARALLEL ALGORITHMS FOR ADAPTIVE QUADRATURE--CONVERGENCE

John R. Rice

Purdue University

West Lafayette, Indiana

1. INTRODUCTION. A rather detailed analysis of the structure of algorithms for adaptive quadrature is given in [3]. The concept of metalgorithm is introduced and a metalgorithm for adaptive quadrature is illustrated by the block diagram in figure 1.

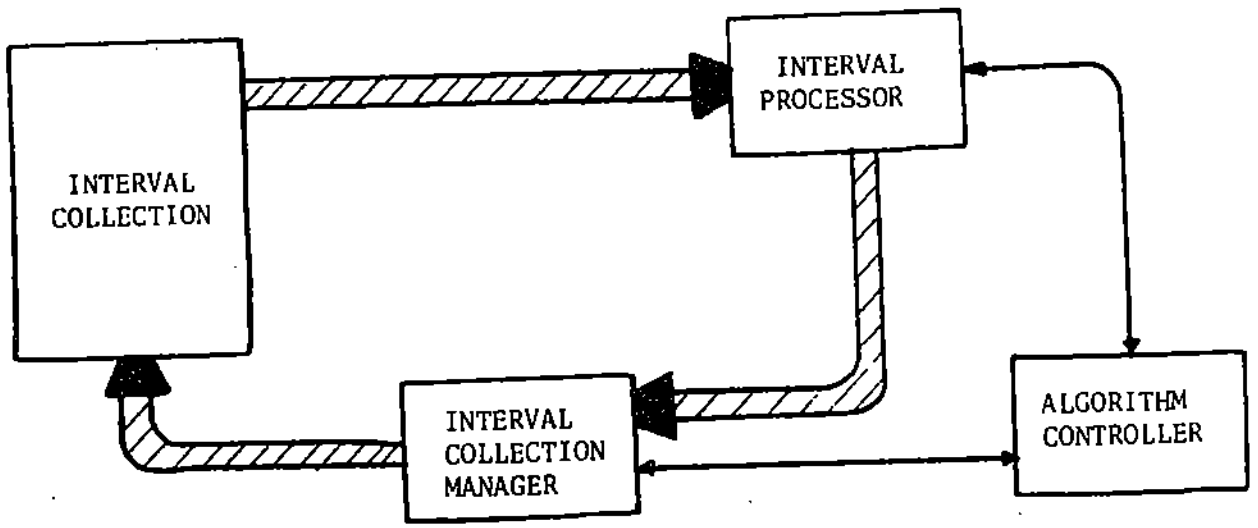

Figure 1. Block diagram of a metalgorithm for adaptive quadrature. The heavy line shows the flow of intervals and the light line the flow of control and other information.

The analysis in [3] shows that there are at least 1 to 10 million potentially interesting adaptive quadrature algorithms. That paper also establishes a range of convergence results and examines three concrete realizations of the metalgorithm.

The purpose of this paper is to use the metalgorithm framework to discuss parallel algorithms for adaptive quadrature. Space precludes the level of detail given in [3] so we refer the reader to that paper for further clarification of some of the concepts presented. The use of parallel computers has been very fruitful in some areas of numerical computation (especially vector and matrix computations) and unfruitful in others (e.g. solving nonlinear equations [2], [4]). It is plausible that quadrature is an area where parallel hardware may be effectively used and this is, in fact, the case. The idea is to have multiple copies of the subalgorithm for processing intervals (i.e. for making estimates of areas and errors on various subintervals of the original one). This subalgorithm is then in execution on each of a number of independent general purpose computers (or CPUs). The interval collection management subalgorithm is in execution on another CPU and it has the task of distributing intervals to the interval processors and collecting results and intervals back from them. The algorithm controller is in execution on yet another CPU and it initiates and monitors the entire computation.

In summary then we have a number of independent CPUs with access to a single large memory. There are three distinct programs involved (for control, collection management and interval processing), one of which is used by many CPUs. Thus we have what is called a "multiple-instruction stream, multiple-data stream, asynchronous, parallel computation".

There are several aspects to these algorithms besides convergence behavior and we organize the material so as to avoid consideration of these other aspects and yet to allow the convergence results to be

to be applicable in a larger context. This is done by stating in the next section a list of assumptions about the integrand, the area and error bound formulas used by the interval processor, the data structure used by the collection manager, the timing and protection of critical data by the collection manager and various other components of the algorithm. These assumptions become hypothesis of the theorem established and thus its domain of applicability is fairly well delineated even though it applies, literally, to millions of potentially interesting algorithms. Note that it is our intention to arrange things so that this theorem is applicable to real algorithms (i.e. Algol or Fortran programs) rather than to have them merely be "mathematically relevant".

The convergence results are stated in terms of the accuracy achieved as a function of the number of evaluations of the integrand. Thus the problem is to evaluate

$$If = \int_0^1 f(x)dx$$

and the algorithm produces an estimate $Q_N f$ after N evaluations of $f(x)$. The theorems then state things like $| If - Q_N f | \leq KN^{-p}$ (where K and p are some constants of the algorithm) which is essentially the same results as established in [3] for sequential algorithms. One expects in general that with NCPU + 2 CPUs (NCPU doing interval processing) that N evaluations may be made in N/NCPU times the time required for 1 evaluation. This would imply that maximum advantage is made of the parallelism. This expectation is approximately fulfilled, but certain

special situations arise (which are not analyzed here) such as the initial stages of a computation where NCPU is very large (the computation may terminate before an appreciable fraction of the CPUs is used). The general question of speed-up due to parallelism is briefly discussed, but not analyzed in depth, in the last section.

2. HYPOTHESES AND ALGORITHM DEFINITION. The general form of the algorithms has been indicated above, we now introduce some definitions and precise hypotheses to be used in the convergence theorem. Our first assumption involves the integrand $f(x)$ and it indicates the domain of efficient applicability of adaptive algorithms.

ASSUMPTION 1 (Integrand). Assume $f(x)$ has singularities

$$S = \{s_i \mid i=1,2,\ldots,R < \infty\}$$

and set $w(x) = \prod_{i=1}^{R} (x-s_i)$

(i) If $x_0 \notin S$ then $f^{(p)}(x)$ is continuous in a neighborhood of x_0.

(ii) There are constants $p \geq 2$, K and α is that

$$| f^{(p)}(x) | \leq K | w(x) |^{\alpha-p}$$

As each interval is processed the algorithm computes an approximate area and an estimate of the error in this approximation. The quadrature rule for the area plays no role in this analysis but the error estimate and the nature of bounds on it play a central role. For simplicity we assume that the interval processor divides an interval into two equal parts

and thus every interval is of the form $[x, x+2^{-k}]$. The algorithm's estimate of the quadrature error on $[x, x+2^{-k}]$ is denoted by ERROR(x,k). Every such algorithm must relate the local error estimates to a global one as discussed in [1]. A fixed error distribution is where the global error is simply the sum of the local ones. We assume this distribution here, but the analysis and proofs may be extended to the more commonly used proportional error distribution as is done in [3].

ASSUMPTION 2 (Error Estimates). There are constants p, K and α (the same as in Assumption 1) so that:

(i) if $[x, x+2^{-k}]$ contains no singularity of f(x)

$$\text{ERROR}(x,k) \leq K \mid f^{(p)}(x) \mid 2^{-k(p+1)}$$

(ii) if $[x, x+2^{-k}]$ contains a singularity of f(x)

$$\text{ERROR}(x,k) \leq K\, 2^{-k(1+\alpha)}$$

The model of a parallel computer used here is that of a number of general purpose processors, essentially identical, that share a common memory. These CPUs operate asynchronously and NCPU of them are assigned to process intervals so that NCPU+2 CPUs are used by the algorithm. We ignore any operating system features and assume that the algorithms correctly initializes memory and the CPUs. The processing time is the time required by a CPU to compute the area and bound estimates and to make auxiliary computations. The return time is the time (delay) from the completion of the processing of an interval to the acquisition of the results by the collection manager and algorithm controller.

ASSUMPTION 3 (Interval Processing). The processing of an interval requires at most q evaluations of f(x) and the processing time is less that a constant C_0. The return time is less than C_0+C_1*NCPU where C_1 is a constant.

The merits of various data structures for the interval collection are discussed in [3], but for the sake of brevity we assume that the collection is divided into two boxes according to whether ERROR(x,k) is larger or smaller than an a priori specified value ε. One may think of these boxes containing "active" and "discarded" intervals and the collection manager merely chooses (by any means whatsoever) an active interval and delivers it to an interval processor. Upon the return of the resulting two intervals it places them in the appropriate boxes. The time required for the collection manager to locate and deliver an interval to an interval processor CPU is the delivery time. It is important to note that this time includes detecting the existence of an idle CPU and an interval in the active box. The time required for the manager to insert returned intervals into the data structure is called the insertion time. We assume that the collection manager preserves the integrity of the interval collection in this concurrent operating environment and that no interlocks may occur.

ASSUMPTION 4 (Interval Collection Management) There are constants C_0 and C_1 so that the delivery time and the insertion time are each less than C_0+C_1*NCPU.

This data structure model may seem overly simplified but, in applications, it is seen that more realistics algorithms may be interpreted in this way and Assumption 4 is satisfied. The value of C_1 here and in Assumption 3 plays a key role in determining the effectiveness of the parallelism. That is to say, the speed-up achieved depends on the behavior of each of the times (processing, return, delivery, insertion) involved and thus an algorithm which is truly efficient must have $C_1=0$ (or replace the term C_1*NCPU by something like C_1*log(NCPU)). The governing time is seen to be the cycle time T_c defined as the total elapsed time from the moment the delivery process is initiated until the insertion of the two halves is completed. It follows from Assumptions 3 and 4 that

$$T_c \leq 4C_o+3C_1*NCPU$$

One obvious data structure is an ordered list and it appears to be difficult to devise algorithms where the insertion time has a smaller bound that C_o+C_1*NCPU+C_2*NLIST where NLIST is the number of intervals in the collection.

In order to clarify the algorithm's processing of intervals we present figure 2.

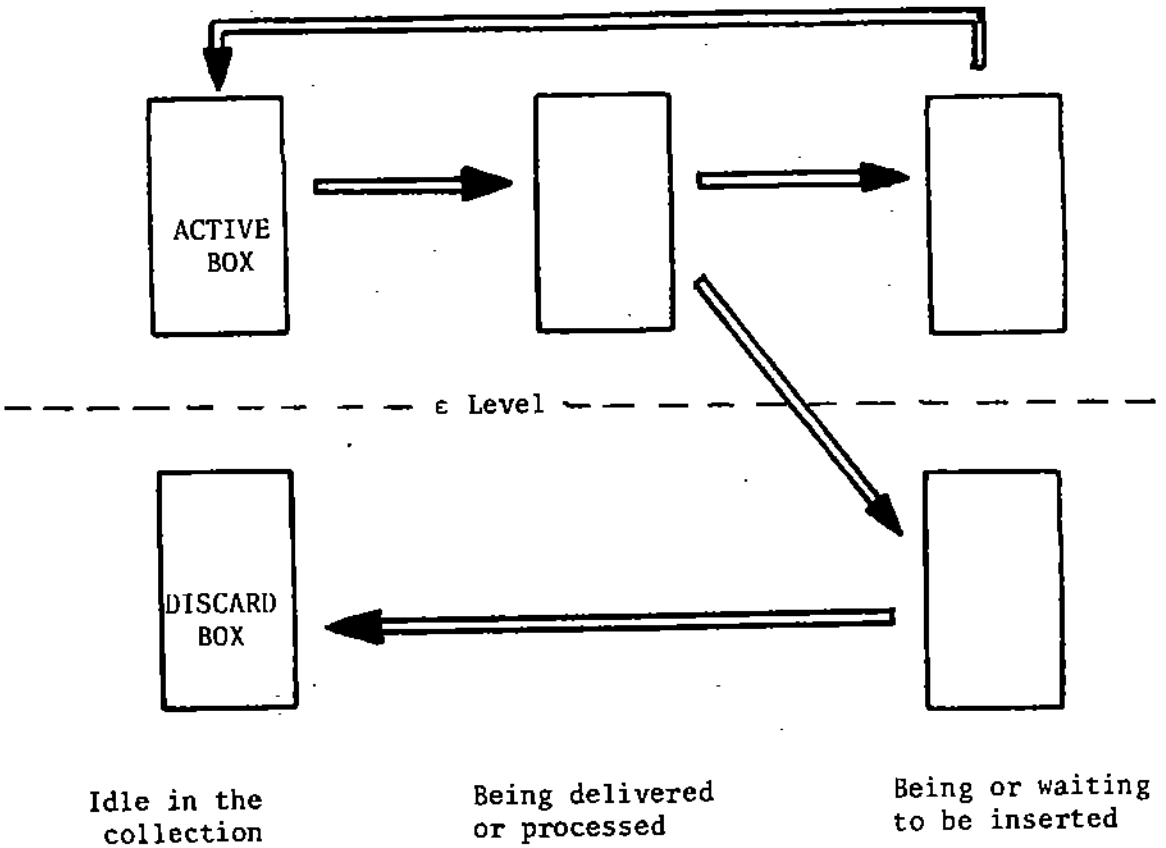

Figure 2. A snapshot of the total interval collection's status. The arrows indicate the possible status transitions that an interval may make.

We see that the algorithm terminates when the discard box contains the total interval collection. The algorithm is initiated by placing the interval [0,1] in the active box.

3. THE CONVERGENCE RESULTS. We begin with a consideration a simple case where $f(x)$ has a single singularity at $x=0$. The quadrature estimate obtained by the algorithm after N evaluations of $f(x)$ is denoted by $Q_N f$ and the time to compute $Q_N f$ is denoted by $T_N f$. The unit of time is that required to evaluate $f(x)$ once.

LEMMA 1. Let a parallel, 2-box algorithm satisfy Assumptions 2, 3, and 4. Let Assumption 1 be satisfied with $S=\{0\}$. Then, as $N \to \infty$, we have

$$|If-Q_N f| \leq \mathcal{O}(\frac{1}{N^p})$$

and for NCPU $< \sqrt[4]{N}$ there is a constant K_4 so that

$$T_N f \leq K_4 \frac{N*T_c}{NCPU}$$

Proof. We consider separately intervals of the form $[0,2^{-t}]$ and note from Assumption 2 that

$$ERROR(0,t) \leq K2^{-t(1+\alpha)}$$

Let t_o satisfy

$$t_o \geq \frac{1}{1+\alpha} \log_2 \varepsilon/K > t_o-1$$

and then we know that $[0,2^{-t_o}]$ is placed in the discard box.

All other intervals are of the form $[2^{-t},2^{-t+1}]$ or descendents of such intervals. We have

$$ERROR(2^{-t},2^{-t}) \leq ERROR(0,1)2^{-t(1+\alpha)}=K2^{-t(1+\alpha)}$$

Let d_t denote the number of times that $[2^{-t},2^{-t+1}]$ must be halved in order

to be certain that all its descendents are discarded. Then d_t is the smallest integer so that

$$K2^{-t(1+\alpha)} \quad 2^{-(p+1)d_t} \leq \varepsilon$$

or

$$2^{-d_t} \leq \left[\frac{\varepsilon 2^{t(1+\alpha)}}{K}\right]^{\frac{1}{p+1}} < 2^{-d_t+1}$$

We may now bound the total number M of distinct intervals that appear in the active box by

$$M \leq t_o + \sum_{t=1}^{t_o} 2^{d_t+1} \leq t_o + 4 \sum_{t=1}^{t_o} \left[\frac{K2^{-t(1+\alpha)}}{\varepsilon}\right]^{\frac{1}{p+1}}$$

$$\leq t_o + 4\left[\frac{K}{\varepsilon}\right]^{\frac{1}{p+1}} \sum_{t=1}^{\infty} 2^{\frac{-t(1+\alpha)}{p+1}}$$

$$\leq 1 + \frac{1}{1+\alpha} \log_2 \varepsilon/K + 4\left[\frac{K}{\varepsilon}\right]^{\frac{1}{p+1}} \Big/ (1-2^{-\frac{1+\alpha}{p+1}}) \leq K_1 \varepsilon^{-\frac{1}{p+1}}$$

where K_1 is a constant independent of ε. It follows from Assumption 3 that $N \leq qM$ and thus

$$N \leq qK_1 \varepsilon^{-\frac{1}{p+1}}$$

and it is clear that

$$|If - Q_N f| \leq 2M\varepsilon \leq 2N\varepsilon \leq 2[qK_1]^{p+1} N^{-p}$$

This establishes the first conclusion of the lemma.

Consider the state of the algorithm at times 0, T_c, $2T_c$, ..., up to termination time T_s. An interval is said to be active if its associated

ERROR value is larger than ϵ, it might not be in the active box. Let L_1 be the set of times that there are NCPU or more active intervals and let L_2 be the remainder. The assertion below follows from the assumptions on the algorithm and the definition of T_c.

Assertion: (i) If there are fewer than NCPU active intervals at time t, then by time $t+T_c$ (1 cycle later) at least this many intervals have been through the interval processor.

(ii) If there are NCPU or more active intervals at time t, then by time $t+T_c$ at least NCPU intervals have been returned to the interval collection (either in the active or discard boxes).

It follows from this assertion that if a time kT_c is one of the L_1 times, then at least NCPU intervals are processed in the period $[kT_c, (k+1)T_c]$. We may bound the size ℓ_1 of L_1 by noting that at most M intervals are processed and thus we have

$$\ell_1 \leq M/NCPU$$

In order to bound the size ℓ_2 of L_2 we let P_k, A_k, and R_k denote, respectively, the number of intervals initially in, added to and removed from the active box during the cycle starting at time kT_c. We have $P_{k+1} = P_k + A_k - R_k$ and R_k is the number of intervals whose processing is initiated during this cycle. Now Let I_j be the number

of cycles in L_2 that exactly j intervals in the active box are not processed. We see that I_o is bounded by the length of the longest chain of active descendents of [0,1] before the last descendent is discarded. Thus we have

$$I_o \leq t_o \leq 1 + \frac{1}{1+\alpha} \log_2 \varepsilon/K$$

Likewise I_1 is bounded by the length of the second longest chain and I_j by the length of the jth longest chain. Thus we have $I_j \leq t_o$. Let IA_j be the number of times that $A_k=j$. We have $R_k \geq P_k$ from the assertion and if $A_k=j$ then the kth cycle has at most j intervals whose processing is not initiated. Thus we have $IA_o \leq I_o$, $IA_1 \leq I_o+I_1$ and, in general

$$\ell_2' = \sum_{j=o}^{NCPU-1} IA_j \leq \sum_{j=o}^{NCPU-1} \sum_{m=o}^{j} I_m \leq \frac{NCPU(NCPU+1)}{2} t_o$$

where ℓ_2' denotes the number of cycles with $A_k < NCPU$. For the remaining $\ell_2-\ell_2'$ cycles we have $A_k \geq NCPU$ and thus

$$\ell_2-\ell_2' \leq M/NCPU$$

Recall that $M \leq K_1 \varepsilon^{-\frac{1}{p+1}}$ so there are constants K_2 and K_3 such that when $NCPU < \sqrt[4]{M}$

$$\ell_2' \leq \frac{NCPU(NCPU+1)}{2} (1+K_2 \log_2 M) \leq K_3 M/NCPU$$

We may combine these estimates to obtain

$$T_N f \leq (\ell_1+\ell_2) T_c \leq (2+K_3)M^*T_c /NCPU \leq q(2+K_3)N^*T_c/NCPU$$

which establishes the second conclusion of the lemma and completes the proof.

The analysis of the behavior of the cycle time is deferred to another paper, but the following corollary indicates what one might hope for the speed-up from a parallel algorithm.

COROLLARY If the cycle time is a constant (C_1=0) in Lemma 1 then for $\sqrt[4]{N}$ > NCPU there is a constant K_4' so that

$$T_N f \leq K_4' \frac{N}{NCPU}$$

This lemma and analysis is now used to establish the general convergence result:

THEOREM 1. Let a parallel, 2-box algorithm satisfy Assumptions 1, 2, 3 and 4. Then, as $N \to \infty$, we have

$$|If - Q_N f| \leq \mathcal{O}(\frac{1}{N^p})$$

and for NCPU < $\sqrt[4]{N}$ there is a K_4 so that

$$T_N f \leq K_4 \frac{N*T_c}{NCPU}$$

Proof. Suppose that the theorem is true for the intervals [a,b] and [b,c] separately replacing [0,1]. It is not difficult to show that Lemma 1 implies that the following assertion is true: If the algorithm is initiated with the two intervals [a,b] and [b,c] in the active box then the convergence behavior is as stated in the conclusions of the theorem. Mathematical induction may be used to extend this assertion to an arbitrary finite sequence of intervals.

We next show that the algorithm generates a sequence $[d_i,d_{i+1}]$ of intervals whose union is $[0,1]$ such that each contains at most one singularity of $f(x)$. If the algorithm never generates an interval with end point d_i satisfying $s_i < d_i < s_{i+1}$ then the interval $[s_i,s_{i+1}]$ would never be subdivided and hence any interval $[x,x+2^{-k}]$ containing $[s_i,s_{i+1}]$ would have

$$\text{ERROR}(x,k) \geq \text{ERROR}(s_i,-\log_2(s_{i+1}-s_i)) = e_i$$

When $\varepsilon<e_i$ the algorithm would never terminate as this interval would never be discarded. This contradicts the easily established fact that the algorithm terminates for every value of $\varepsilon>0$. Thus the subdivision of $[0,1]$ into intervals $[d_i,d_{i+1}]$, $i=1,2,\ldots,R$ does occur in the algorithm. We take $d_o=0$ and $d_R=1$. Note that $[d_i,d_{i+1}]$ is probably not a single interval considered by the algorithm, but rather the union of such intervals.

We next adapt the analysis of Lemma 1 to establish the convergence result for $[d_i,d_{i+1}]$. Let $[a_t,b_t]$ denote the active interval which currently contains s_i. If at any point $a_t=s_i$ or $b_t=s_i$ then we may redefine d_i or d_{i+1} to be s_i and omit the following analysis for $[d_i,d_{i+1}]$. Thus we have

$$a_t < s_i < b_t = a_t + 2^{-t}$$

and, as in the proof of Lemma 1, there is a value t_o where we know that $ERROR(a_t,t) \leq \varepsilon$ and $[a_t,b_t]$ is discarded. All other intervals derived from $[d_i,d_{i+1}]$ are split off the left or right end of $[a_t,b_t]$ or are the descendents of such intervals. An interval that is first split off $[a_t,b_t]$ has

$$ERROR(x) \leq ERROR(0,1)\ 2^{-t(1+\alpha)} = K2^{-t(1+\alpha)}$$

Let N_i denote the number of $f(x)$ evaluations for processing $[d_i,d_{i+1}]$. We may repeat the analysis of Lemma 1 to conclude that

$$N_i \leq qK_1\ \varepsilon^{-\frac{1}{p+1}}$$

and

$$\left| \int_{d_i}^{d_{i+1}} f(x) - Q_{N_i} f \right| \leq 2N_i\varepsilon \leq 2[qK_1]^{p+1}\ N_i^{-p}$$

We now patch the intervals $[d_i,d_{i+1}]$ together and apply the earlier assertion to establish that

$$| If - Q_N f | \leq 2[RqK_1]^{p+1}\ N^{-p}$$

and that the time $T_N f \leq K_4\ N*T_c/NCPU$. This concludes the proof.

We also have

COROLLARY If the cycle time T_c is a constant ($C_1=0$) in Theorem 1 then for $\sqrt[4]{N} > NCPU$ there is a constant K_4' such that

$$T_N f = K_4' \frac{N}{NCPU}$$

4. FURTHER ALGORITHM CONSIDERATIONS. Four specific data structures (stack, queue, ordered list and boxes) for organizing the contents of the active box are described in [3]. It is shown there that all four of these lead to algorithm classes with the convergence properties given in Theorem 1. Timing is a critical consideration in parallel computation and the choice of data structure directly influences the cycle time T_c (and hence the speed-up obtained). An ordered list algorithm, for example, is likely to have an insertion of the order of C_o+C_1*NCPU+C_2*NLIST where NLIST is the list length. This makes it impossible to obtain any speed-up and hence this class of algorithms is unsuitable for parallel computation. The other three data structures allow quick insertions (with $C_1=0$) and thus do not prevent maximum speed-up.

If both the insertion and processing times are constant, then the speed-up possible is governed by the delivery time and return time. A little thought shows that a crucial factor in both these times is how the collection management processor becomes aware of the status of the interval processors. A simple and common approach is to have the interval processors set flags (or semaphores) and then have the collection management processor poll the interval processors to determine their status. This, of course, makes the delivery and return times proportional to NCPU and thus prevents speed-up in the theoretical sense. Before going on it is important to note that very significant speed-up can occur in the practical sense even when there is none theoretically. One must

examine actual algorithms in order to obtain a realistic evaluation of the speed-up obtained by parallel computation.

The polling approach to communication between the collection manager and interval processors is inherently slow (and inefficient) unless the relative times of the computations and the number NCPU are such that the collection management CPU does little besides polling. Once the flow of intervals through the collection manager becomes significant then communication via interrupts is much more efficient. That is, an interval processor indicates its status by interrupting (in some sense) the collection manager. Interrupts can be constructed by software so that hardware interrupts are not required, but hardware can facilitate the tasks. Once NCPU becomes very large even the interrupt approach fails to eliminate the communication bottle neck entirely and then more elaborate mechanisms are required including assigning more than one CPU to manage the interval collection. An analysis of actual algorithms and of mechanisms to minimize the delivery and return times must be deferred to another paper as it is more complex than the traditional convergence analysis. It may well be that algorithms cannot be found where these times are less than $C_0+C_1\log(NCPU)$ asymptotically as $NCPU \to \infty$. However, for a reasonable value like NCPU=50, it is the author's belief that algorithms involving say 52 or 53 processors exist which give a speed-up in time of a factor of about 50.

REFERENCES

[1] Carl W. deBoor, On writing an automatic integration algorithm, in Mathematical Software (J. R. Rice ed.), Academic Press, New York, 1971, 201-209.

[2] John R. Rice, Matrix representations of nonlinear equation iterations-Application to parallel computation, Math. Comp., vol. 25, 1971, 639-647.

[3] John R. Rice, A metalgorithm for adaptive quadrature, report CSD-TR89, Purdue University, March 1973, 1-43.

[4] Shmuel Winograd, Parallel iteration methods, in Complexity of Computer Computations (R. E. Miller and J. W. Thatcher eds.), Plenum Press, New York, 1972, 53-60.