Acta Informatica 5, 273—285 (1975)

Parallel Algorithms for Adaptive Quadrature II Metalgorithm Correctness

John R. Rice

Received February 3, 1975

Summary. This paper introduces a rather specific metalgorithm (or meta-program) for a class of algorithms for adaptive quadrature on parallel (MIMD) computers. This class includes all the current approaches to adaptive quadrature. The main result is that any member of this metalgorithm satisfies the conditions of a traditional numerical analysis convergence theorem from [2]. The algorithm structure in this metalgorithm is specified in some detail and 32 Attributes are assumed. These Attributes and structure serve to guide the design of particular algorithms. They also facilitate establishing algorithm correctness by providing a detailed set of algorithm properties (most of which are like "assertions" in program proving) that are sufficient for correctness.

1. Introduction

The concept of metalgorithm was introduced in [1] and applied to the study of algorithms for adaptive quadrature. Then in [2] the study was extended to consider algorithms for parallel computers and a general convergence theorem was established. Recall that a metalgorithm represents a class of algorithms and the convergence result may be paraphrased as follows: "If all the algorithms represented by the metalgorithm satisfy certain assumptions, then a certain rate of convergence takes place." The assumptions made are fairly simple in nature and the conclusion is a typical mathematical theorem which indicates the exceptional power of adaptive algorithms.

Our ultimate goal is to establish convergence results for actual computer programs and this paper represents the second level of analysis. We introduce a much more specific metalgorithm than in [2], and show that it is contained in the more general metalgorithm. Thus we may conclude that the convergence result is valid for our more specific metalgorithm. The third level of proof is to exhibit actual programs and prove that they are contained in this more specific metalgorithm. This is done in a subsequent paper [3].

The next section presents the general structure of the metalgorithm, associated technical definitions and the identification of certain critical variables. The third section contains a systematic list of the specific assumptions about the metalgorithm. These assumptions are called *attributes* of the programs comprising the metalgorithm. The next section then presents a series of lemmas and theorems which show that these attributes imply that the assumptions of the general metalgorithm of [2] are satisfied and the final section contains a convergence theorem

applicable to any algorithm (computer program) represented by this metalgorithm.

The nature and style of these proofs are those of normal mathematics, a style which is quite different from that of the foundations of mathematics or Euclidean Geometry as taught in high school. This style may be summarized by saying that the author and reader agree that certain questions are obvious or trivial to check and others are not. The trivial questions are ignored and the nontrivial ones are resolved by the author to the satisfaction of the reader. This approach sacrifices the iron-clad guarantees sought in the foundations of mathematics approach, but it has served mathematics well and is probably the only viable approach to correctness proofs for medium or large size programs.

2. The Parallel Metalgorithm

A) *Problem Definition.* The problem is to estimate within ε

$$If = \int_a^b f(x)\,dx$$

given $f(x)$, a, b, ε and a characteristic length CHARF of $f(x)$. The value obtained by an algorithm based on N evaluations of $f(x)$ is called $Q_N f$ and thus we require

$$|If - Q_N f| < \varepsilon.$$

The evaluation of $f(x)$ is used to scale time in the computation and $T_N f$ is the time required to compute $Q_N f$ measured in units of 1 evaluation time of $f(x)$. We measure (or estimate) the execution time of all programs and we assume that their execution times are known relative to that of evaluating $f(x)$. The function $f(x)$ is not considered directly in this analysis and it is assumed that a program for evaluating $f(x)$ is made available to any processor at any time it needs a value.

B) *The Metalgorithm Structure.* There are many instances where we should use the phrase "an algorithm represented by the metalgorithm" but for brevity we often replace this by "the metalgorithm" or "the algorithm". Thus we may attribute properties to a set (the metalgorithm) which only members (an algorithm) may possess. The metalgorithm involves two distinguished central processing units or CPUs called CPU1 and CPU2 and an array (CPUR(IP), for IP=1 to NCPU), of CPUs which are used to process intervals. The programs associated with these CPUs are illustrated in the schema of Fig. 1.

The naming of programs and the description of this metalgorithm suggest that the interval collection is maintained as a queue. This implication is *not* a formal assumption and a perusal of the attributes shows that other data structures might be used. We note that one of the key parts of the proofs in [2] would be drastically simplified with the assumption that the interval collection is a quene.

C) *Global and Critical Variables.* Communication between these CPUs is assumed to take place primarily by common access to certain variables in the algorithm. We identify five distinct groups of variables, name the associated memory areas and certain selected specific variables.

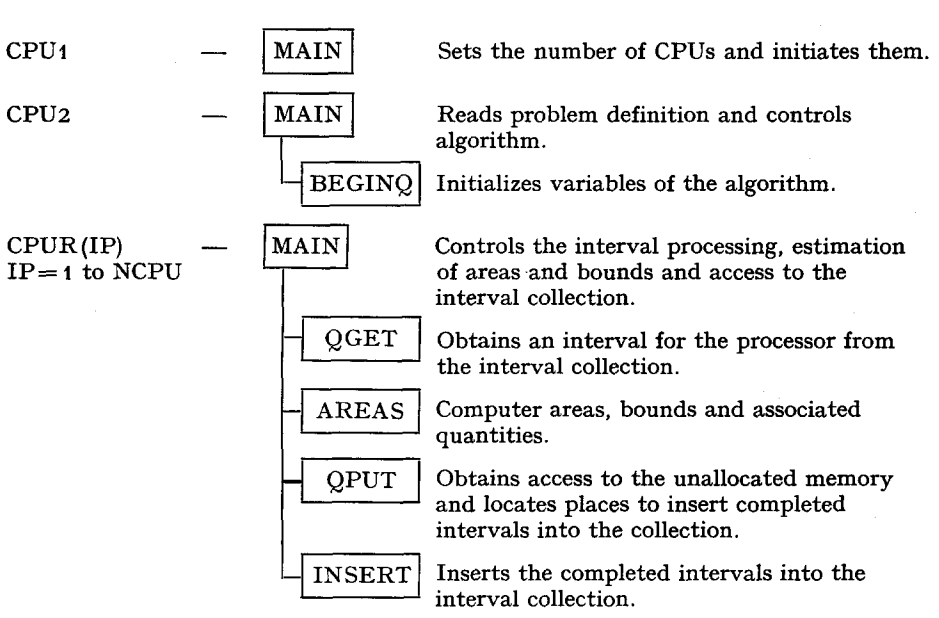

Fig. 1. A schematic diagram of the structure of the metalgorithm for parallel adaptive quadrature

Definition 1. (Global and critical variables.) *The five memory areas for global variables are:*

a) *OPSYS:* *variables defining the computer context.*
NCPU = number of CPUs less two made available to the algorithm.

b) *PROBLEM:* *variables defining the problem.*
A, B = interval of integration,
EPS = required accuracy in the result,
CHARF = characteristic length of $f(x)$,
$F(x)$ = the integrand function.

c) *CONTROL:* *variables used to control the computation.*
AREA = current estimate of If,
BOUNDA = current bound of the error in the estimate of If.

d) *QUEUE:* *the collection of intervals plus variables used to define the structure and status of the collection.*

e) *PROCESSORS:* *variables associated with the NCPU processors that are used by several subprograms of the main program for CPUR(IP). Variables here are in arrays indexed by the associated CPU.*

One of the key requirements in the proof of the correctness of a metalgorithm is to show that the integrity of these global variables is maintained. For some (such as in OPSYS and PROBLEM) this is trivial as they are assigned values once and for all at some point in the metalgorithm. For others (e.g. AREA, BOUNDA and those in QUEUE) this is not trivial as their values are frequently modified and yet their values are critical to the algorithm. Thus a consideralbe part of the design of a concrete algorithm is concerned with preserving the integrity of these critical variables during simultaneous or concurrent processing by several CPUs. A real algorithm may also have variables whose values are frequently modified and yet which are not critical to the algorithm. One may visualize, for example, variables used to aid the efficiency of the computation rather than required for producing correct results.

D) *Program Timing.* A schematic diagram of the flow of intervals in this algorithm is given [2]. Various times of processes are introduced there and their definition in the current context is needed. Note that the parallel nature of the algorithm implies that conflicts may arise between different programs attempting to process the same information. These conflicts are resolved by having some programs wait and these waiting times must be accounted for as well as execution times.

Definition 2 (Timing)

a) *The processing time is the sum of the following times:*
   1. The time to execute AREAS. One would expect this to be nearly constant.
   2. The time to execute MAIN of CPUR(IP) excluding the execution time of its subprograms. This is "fixed overhead".

b) *The delivery time is the time of execution of QGET.* It has two parts: the waiting required to gain sole access to the interval collection (such as the head of the queue) and the time required to assign an interval once access is achieved.

c) *The return time is the time of execution of QPUT.* Like the delivery time, it consists of a waiting to gain sole access to the unallocated memory (such as the tail of the queue) plus a time required to assign places for the return of intervals.

d) *The insertion time is the time of execution of INSERT.* This time varies considerably depending on how many (0, 1 or 2) intervals are inserted.

e) *The cycle time T_c is the time required to select an interval from the collection, process it and insert the resulting intervals, if any, back into the collection* It is the sum of the preceding four times.

3. The Program Attributes

This section contains all the specific attributes assumed for the programs in this metalgorithm.

A) *Attributes of MAIN-CPU1.*

1. Assigns the value of NCPU.
2. Enables the other CPUs.
3. Initializes all control variables to be false and all numerical variables to be zero.

B) *Attributes of MAIN-CPU2.*

1. Obtains the variables that define the problem.
2. Initially invokes BEGINQ.
3. Monitors BOUNDA and terminates the algorithm (with output) when BOUNDA $<$ EPS, when there is a memory overflow or when there are no more active intervals.

C) *Attributes of BEGINQ.*

1. Places the interval [A, B] into the interval collection, computes all associated values and initializes the collection properly.
2. Initializes variables for control of access to the interval collection.
3. Its final statement enables the other CPUs to proceed by designating the interval [A, B] as "free".

D) *Attributes of MAIN-CPUR(IP).* Once this CPU is activated it executes the following sequence of actions:

Invoke QGET
Invoke AREAS
Invoke QPUT
Invoke INSERT
Return to the top of this list

E) *Attributes of AREAS.*

1. Computes changes in AREA and BOUNDA. The resulting values of AREA and BOUNDA satisfy certain requirements (e.g. Assumptions 1 of [2]) provided $F(x)$ satisfies certain requirements (e.g. Assumptions 2 of [2]).
2. Uses a proportional error distribution for BOUNDA and implements the restriction that the interval length be less than CHARF before BOUNDA is allowed to be less than EPS.
3. Determines how many, if any, intervals are to be discarded and identifies them.
4. Computes the variety of information about the two intervals that are obtained. This information, along with the other information generated, is temporarily placed in the memory PROCESSORS and associated with this CPU.
5. There are no unbounded computations in AREAS and its maximum execution time is bounded by a constant. It is the only program of CPUR(IP) that evaluates $F(x)$ and it does this at most q times.

F) *Attributes of INSERT.*

1. Once places have been assigned in QUEUE by QPUT, it places all the relevant information about the new intervals into these places in QUEUE.

2. Prevents an interval from being assigned to another CPU before its insertion into the collection is complete.
3. There are no unbounded computations in INSERT and the maximum execution time is bounded by a constant.

G) *Attributes of QGET.*

1. This program gains sole access to an interval in the collection that is free to be assigned to a CPU. If the interval to be assigned is not free, then QGET waits in an idle loop.
2. Once access is gained to an interval, it is assigned to CPUR(IP) and so identified, and not assigned again. A new interval is designated as next to be assigned.
3. At most NCPU-1 CPUs gain access to the interval collection between the time a particular one tries for and the time it achieves access to the interval collection.
4. There is no conflict between QGET and QPUT.
5. Does not affect information about the interval itself, only about the interval's status in the algorithm.
6. No interlock occurs when more than one CPU is executing QGET and, in such a case, one of them gains access to the interval collection within a fixed time.

H) *Attributes of QPUT.*

1. This program gains sole access to the unallocated or available memory in QUEUE. It waits in an idle loop until this access is achieved.
2. Obtains places in the available memory of QUEUE for the new intervals to be returned and assigns these places to the interval collection. It updates the information about the available memory in QUEUE.
3. At most NCPU-1 CPUs gain access to the available memory between the time a particular one first tries and the time it achieves access to the available memory.
4. While it has access to the available memory it updates the values of AREA and BOUNDA. Thus access to the available memory is required and made even if both new intervals are discarded.
5. If the interval collection is empty when this CPU is obtaining places for the return of intervals to the collection, then QPUT designates one of the returned intervals as the next one to be assigned.
6. There is no conflict between QGET and QPUT.
7. Does not affect information about the interval itself, only about the intervals' status in the algorithm.
8. No interlock occurs when more than one CPU is executing QPUT and, in such a case, one of them gains access to the available memory within a fixed time.

4. Assumptions and Preliminary Results

A) *Previous Results.* The objective is to show that the attributes listed in Section 3 imply that the assumptions of the convergence results in [2] are satisfied. We list here those assumptions and the main result in a slightly rephrased form

to reflect the present context. In order to simplify the notation *we assume that the interval* $[a, b]$ *of integration is* $[0, 1]$ throughout the remainder of this paper.

Assumption 1. *(Integrand)* $f(x)$ *has singularities*

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

Let $w(x)=\prod_{i=1}^{R}(x-s_i)$ *then*

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

(ii) *there are constants* K *and* $\alpha>0$ *so that*

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

We recall the notation that ERROR(x, k) is the algorithm's bound on the quadrature error for the interval $[x,\ x+2^{-k}]$.

Assumption 2. *(Error estimates). There are constants* p, K *and* α *(the same as in Assumption 1) so that when* $2^{-k}<CHARF$ *we have*

(i) *If* $[x, x+2^{-k}] \cap S$ *is empty then the bound* $ERROR(x, k)$ *computed by AREAS satisfies*

$$\text{ERROR}(x, k) \leqq K f^{(p)}(x)\, 2^{-kp}$$

(ii) *If* $[x, x+2^{-k}] \cap S$ *is not empty then*

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

There is an essential change in this assumption from that of [2], namely the inclusion of the condition that the interval length 2^{-k} be less than CHARF. This change is what allows us to change the convergence result from the mathematical sense to the algorithmic sense as discussed in [1, Section 8]. This is logically equivalent to assuming that the constant K is known a priori.

The following combines Assumptions 3 and 4 from [2].

Assumption 3. *(Timing) AREAS evaluates* $F(x)$ *at most* q *times and there are constants* C_0 *and* C_1 *so that:*

(i) *The processing and insertion times are bounded by* C_0.

(ii) *The return and delivery times are bounded by* C_0+C_1*NCPU.

The following assumptions were made in [2] but not explicitly numbered.

Assumption 4. *(Miscellaneous from [2]).*

(i) *AREAS divides intervals into two equal parts.*

(ii) *A proportional error distribution is used (See [1] for terminology).*

(iii) *No interlocks occur and the integrity of the interval collection is maintained.*

(iv) *The algorithm is a parallel 2-box algorithm.*

The result established in [2] is:

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

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

and for $\sqrt{N} > \text{NCPU}$ *there is a constant* K_1 *so that*

$$T_N f \leq K_1 \frac{N^* T_c}{\text{NCPU}}.$$

We now establish a sequence of lemmas and theorems which lead to the main result of this paper, namely that the above theorem applies to any algorithm contained in the parallel metalgorithm defined in Sections 2 and 3. Note that we use the words "the algorithm" in this development interchangeably with the phrase "an algorithm represented by the metalgorithm".

B) *On the Initilization, Integrity and Termination of the Algorithm.*

Lemma 1. *The algorithm is initialized properly with the interval* $[a, b]$ *in the interval collection.*

Proof. Attributes 1 and 2 of the program MAIN of CPU1 imply that the operating system assigns a value to NCPU and enables all the other CPUs. The program MAIN of CPUR(K), $K=1$ to NCPU immediately invokes QGET. By Attribute 1 of QGET it tests to see if the interval to be assigned is free and it is not by Attribute 3 of the program MAIN of CPU1. Thus QGET for each of these CPUs enters an idle loop until an interval in the collection is set free. CPU2 reads the problem information (Attribute 1 of the program MAIN of CPU2) and then invokes BEGINQ. Attributes 1 and 2 of BEGINQ state that $[a, b]$ is placed in the interval collection, all associated values are computed and the interval collection is properly initialized. The final act (see Attribute 3) of BEGINQ is to set the interval $[a, b]$ free and unblock access to the interval collection. At this point the algorithm is initialized correctly and ready to proceed.

Lemma 2. *The integrity of the critical values AREA and BOUNDA is preserved.*

Proof. It follows from Attributes 1 and 4 of AREAS that changes in these values are computed in AREAS and placed in memory associated with the CPU executing AREAS. Thus the changes in AREA and BOUNDA are protected from concurrent modification.

Attribute 4 of QPUT implies that the changes in AREA and BOUNDA computed by other programs are actually used only when QPUT has sole access to the available memory of the interval collection. Thus there can be no simultaneous modification of these two variables and their integrity is preserved. This concludes the proof.

The previous lemma assures the integrity of two critical variables (AREA and BOUNDA) and the next one assures us of the integrity of the other critical information in the algorithm, the interval collection. The interval collection is affected by a number of the programs and this lemma involves the most complex set of possible conflicts due to the parallel nature of the algorithm.

Lemma 3. *The integrity of the interval collection is preserved.*

Proof. The program for CPU1 does not involve the interval collection and CPU2 only initializes it by invoking BEGINQ. Thus we need only consider the programs executing on CPUR(K) which may affect the interval collection and there are 16 possible conflicts. These 16 possibilities are displayed below and the entries in the table refer to the relevant discussion in this proof.

	QGET	AREAS	QPUT	INSERT
QGET	*A*	*B**	*A*	*B**
AREAS	*B**	*C**	*B**	*C**
QPUT	*A*	*B**	*A*	*B**
INSERT	*B**	*C**	*B**	*C**

Twelve of the possibilities are marked by an asterisk * and we show that no conflict arises here because simultaneous execution for one interval cannot occur in these cases.

The four possibilities marked "A" do not cause any conflict by the design of the program QPUT and QGET. This follows directly from Attributes 1, 4 and 6 of QGET and Attributes 1, 6 and 8 of QPUT.

The twelve possibilities marked "B" and "C" require the following:

Assertion. QGET does not assign an interval to more than one CPU and QPUT does not assign a place in memory to more than one interval. If the interval collection is not empty then there is always an interval designated as the next to be assigned.

Attribute 2 of QGET implies that an interval cannot be assigned to more than one CPU. Likewise, Attribute 2 of QPUT states that the information about the available memory in QUEUE is updated after each assignment of places for returned intervals. This establishes the first statement of the assertion. For the second statement, assume now that the interval collection is not empty. When a program QGET gains access to the collection, it follows from Attribute 2 of QGET that another interval is designated as next to be assigned. When a program QPUT gains access to the available memory, it follows from Attribute 2 of QPUT that the places obtained are attached to the interval collection and, after INSERT executes, the returned intervals appear in collection (see Attribute 1 of INSERT). If the interval collection becomes empty and then intervals are returned to it, it follows from Attribute 5 of QPUT that one of the returned intervals is designated as available for assignment. This establishes the second statement and the assertion.

Thus four of the possible conflicts marked "B" cannot occur because QGET cannot access an interval already assigned to a CPU (and hence possibly having AREAS or INSERT executing). It follows from Attribute 2 of INSERT that an interval returned to the collection cannot be assigned until INSERT has finished. The other four conflicts cannot occur because QPUT cannot process an interval other than the one assigned to the CPU executing QPUT and this precludes the execution of AREAS or INSERT for this interval.

The four possibilities marked "C" cannot occur because the programs AREAS and INSERT can execute for an interval only on the unique CPU to which that interval is assigned. Further, the execution of AREAS and INSERT do not overlap for any one CPU. The cases marked "B" and "C" thus never give rise to a possible conflict because no simultaneous execution occurs in these cases for any particular interval.

This concludes the systematic examination of all the possible interactions and conflict arising from the simultaneous execution of different programs for a

particular interval. In each case simultaneous execution cannot occur or it does not affect the validity of the results obtained. This concludes the proof.

The results of these four lemmas may be gathered together in the following theorem.

Theorem 2. *The algorithm is properly initiated and the integrity of the critical values and information is preserved during its execution.*

Proof. This follows directly from Lemmas 1, 2 and 3 and the assertion established in the proof of Lemma 3.

Lemma 4. *The discard procedure is effective and the algorithm terminates.*

Proof. It follows from Attribute 3 of AREAS that intervals to be discarded are identified and from Attribute 2 of QPUT that no place is obtained in the interval collection for them. Thus the discard is effective.

It follows from the proof of the convergence results in [1] and [2] that the total number of active intervals is bounded. It is a consequence of Lemma 3 that the interval collection is maintained correctly for those intervals not discarded. The number of intervals in the collection might exceed the space allocated to the algorithm, otherwise the computation stops when there are no more active intervals or the condition BOUNDA $<$ EPS is satisfied (see Attribute 3 of the program MAIN of CPU2). Later hypotheses will rule out the possibility that the set of active intervals becomes empty before BOUNDA $<$ EPS, but in any case the algorithm terminates.

Corollary. *The algorithm is a 2-box algorithm as defined in [2].*

Theorem 3. *This metalgorithm consists of 2-box, parallel algorithms.*

Proof. It follows from Theorem 2 that an algorithm represented by this metalgorithm is properly initialized and unambigous (i.e. the integrity of the variables is preserved). Lemma 4 implies that the algorithm terminates and its corollary states that the algorithm is a 2-box algorithm. The algorithm is obviously parallel and this concludes the proof.

C) *Timing Aspects of the Algorithm.* We now establish four lemmas concerning the algorithm timing. These lemmas involve certain constants which are all denoted by the symbols C_0 or C_1. It is clear that we may assume that C_0 and C_1 are the same for all the lemmas and for later use.

Lemma 5. *There is a constant C_0 independent of NCPU and the problem so that the processing time is less than C_0.*

Proof. Recall from Definition 2 that the processing time consists of the sum of two times. Attribute 5 of AREAS implies that the first of these is bounded by a constant and the other is the overhead for MAIN of CPUR(K) which is clearly fixed.

Lemma 6. *There are constants C_0 and C_1 so that the delivery time is less than C_0+C_1*NCPU.*

Proof. Let t_1 be the time for QGET to assign an interval once sole access to the interval collection is attained and let t_2 be the time for one attempt to gain sole access. If some CPU is executing QGET at time t_0 then one CPU (perhaps a

different one) will, according to Attribute 6 of QGET, have completed QGET at time $t_0+t_1+t_2$. Note that t_2 includes the possibility of waiting in an idle loop because the interval to be assigned is not free. This occurs at the beginning of execution (before BEGINQ terminates) or when INSERT has designated an interval not free. The execution time of INSERT is bounded (see Attribute 3) and hence so is this waiting time. For most data structures this waiting occurs infrequently, only when the interval collection is nearly empty. It then follows from Attribute 3 of QGET that the maximum delay in QGET for any CPU is (NCPU-1)*(t_1+t_2) and hence the delivery time is bounded by NCPU*(t_1+t_2).

Lemma 7. *There are constants C_0 and C_1 so that the return time is less than C_0+C_1*NCPU.*

Proof. The proof is exactly parallel to that of Lemma 6 except that QPUT replaces QGET and Attributes 3 and 8 are used.

Theorem 4. *The algorithm satisfies Assumption 3 concerning timing.*

Proof. This follows directly from the preceding three lemmas plus Attribute 5 of AREAS and Attribute 3 of INSERT.

5. The Metalgorithm Correctness Theorem

The preliminary results of the preceding section essentially establish the correctness of the algorithm as concerns control, data structures, conflicts resulting from concurrent execution of programs and related items. That is, Assumptions 3 and 4 have been shown to be satisfied. The metalgorithm structure says very little about the detailed numerical behavior relevant to Assumptions 1 and 2. One of the main difficulties of applying these results to an actual computer program is to establish Attribute 1 of AREAS which concerns these numerical analysis assumptions. The main result of this paper is

Theorem 5. *Suppose that $f(x)$ satisfies Assumption 1, the computer memory is unbounded and the computer arithmetic is exact; then when an algorithm is given $f(x)$, A, B, CHARF and EPS>0, it terminates with an estimate $Q_N f$ requiring N evaluations of $f(x)$ so that*

$$|If-Q_N f| \leqq \text{EPS}$$

with

$$N=\mathcal{O}\left(\text{EPS}^{\frac{-1}{p}}\right)$$

or, equivalently,

$$|If-Q_N f| \leqq \mathcal{O}\left(\frac{1}{N^p}\right).$$

If $\sqrt{N} > NCPU$ then the total computation time $T_N f$ satisfies

$$T_N f \leqq K_1 \frac{N^* T_c}{\text{NCPU}} \leqq K_1 \frac{N^*(4C_0+2C_1*\text{NCPU})}{\text{NCPU}}.$$

Proof. We, of course, intend to apply Theorem 1 established in [2]. We have already noted that the algorithm satisfies Assumptions 3 and 4 of that theorem by Theorems 2, 3 and 4. The only references in the metalgorithm to the other assumptions occur in Attributes 1 and 2 of AREAS. Both of these attributes

imply that if $f(x)$ satisfies Assumption 1 then the BOUNDA value computed satisfies Assumption 2. Thus we conclude that the algorithm satisfies the hypothesis of the Theorem 1.

The conclusions of this theorem are somewhat stronger and we now show that the algorithm terminates with $|If-Q_N f| \leqq \text{EPS}$. Note that BOUNDA is the sum of the error bounds for all the intervals, both active and discarded. Let x_i denote the left end points of these intervals.

Then we have $x_{i+1}=x_i+2^{-k_i}$,

$$|If-Q_N f| = \sum_{i=1}^{M} (x_{i+1}-x_i)\,\text{ERROR}(x_i, k_i).$$

In the proof of Lemma 5 it was pointed out the algorithm may terminate due to any one of three conditions. The possibility of memory overflow is excluded by the assumption that the memory is unbounded. Recall from [1] and [2] that the proportional error distribution results in an interval being discarded whenever $2^{-k}\text{ERROR}(x, k)$ is less than EPS. If there are no active intervals, then

$$2^{-k_i}\,\text{ERROR}(x_i, k_i) < \text{EPS}$$

for every interval that has been discarded and thus we have

$$|If-Q_N f| \leqq \text{EPS} \sum_{i=1}^{M} (x_{i-1}-x_i)\,2^{-k_i} = \text{EPS}.$$

Note that the quantity CHARF has played a hidden, but essential, role in this argument. The theorem would still be true, but it would then be impossible to obtain actual algorithms that satisfied Assumption 2 for all the $f(x)$ admitted by Assumption 1. Thus we would have had a true but vacuous theorem.

We have now established the first conclusion of the theorem and that Theorem 1 applies here. The remaining conclusions then follow directly from Theorem 1 and the fact that $T_c \leqq 4C_0+2C_1$*NCPU as seen from Lemmas 5, 6 and 7. This concludes the proof.

It is proved in [1] that the use of the fixed instead of proportional error distribution considerably enlarges the domain of functions for which the algorithm is effective. In fact, it is then effective for almost all integrands which are integrable. It is mentioned in [2] that the analysis can be carried through for parallel algorithms and a fixed error distribution. The difficulty with a fixed error distribution is that one cannot actually discard any of the intervals placed in the discard box. Thus a real algorithm using this method must be prepared to redefine some of the discarded intervals as active or have a data structure where the distinction between active and discarded is relevant only for proofs. The ordered list and boxes data structures discussed in [1] do this in a natural way. We now indicate precisely what modifications of the present metalgorithm must be made in order to carry through proofs similar to those of this paper and we state the resulting theorem (but without actually presenting the proof).

Assumption 5. *The metalgorithm and previous assumptions are modified as follows*:

(i) *In Assumption 1 (ii) the condition* $\alpha > 0$ *is replaced by* $\alpha > -1$.

(ii) *The words "proportional error distribution" are replaced by "fixed error distribution" in Assumption 4 (iii) and Attribute 2 of AREAS.*

(iii) *Attribute 3 of MAIN of CPU2 is modified to eliminate termination when there are no more active intervals. It is assumed instead that a mechanism exists which establishes a new, smaller tolerance to replace EPS in the determination of intervals to be discarded.*

(iv) *Attribute 2 of QPUT is modified so that places are obtained in memory for all new intervals, whether discarded or not, and these places are provided to INSERT.*

Theorem 6. *Let the hypothesis of Theorem 5 be modified by Assumption 5. Then the conclusions of Theorem 5 are valid for the resulting metalgorithm.*

Theorem 5 has been applied to prove the correctness of an actual program PAFAQ written in a modified Fortran for a hypothetical computer. PAFAQ has also been run and tested using the simulation language ASPOL available on CDC 6000 series computers. See [3] for details.

References

1. Rice, J. R.: A metalgorithm for adaptive quadrature. J. ACM. **22**, 61–82 (1975)
2. Rice, J. R.: Parallel algorithms for adaptive quadrature—Convergence. Proc. IFIP Congress 74, Stockholm. Amsterdam: North-Holland 1974, p. 600–604
3. Rice, J. R.: Parallel algorithms for adaptive quadrature III—Program Correctness, ACM Trans. Math. Software, 2 (1976) to appear

John R. Rice
Division of Mathematical Sciences
Purdue University
West Lafayette, Indiana, U.S.A.