SPEEDUP IN PARALLEL ALGORITHMS FOR ADAPTIVE QUADRATURE

James M. Lemme and John R. Rice
Mathematical Sciences
Purdue University
CSD-TR 192
May 15, 1976

ABSTRACT

We describe a fast metalgorithm for adaptive quadrature on a MIMD (Multiple Instruction, Multiple Data) parallel computer and show that its speed up is the order of \( \log \frac{M}{M} \) using a total of \( M \) processors.
SPEEDUP IN PARALLEL ALGORITHMS FOR ADAPTIVE QUADRATURE

James M. Lemme and John R. Rice

1. INTRODUCTION

The quadrature problem for $f(x)$ is to estimate

$$I_f = \int_0^1 f(x) \, dx$$

with formulas of the form

$$Q_N f = \sum_{i=1}^{N} w_i f(x_i).$$

Adaptive algorithms determine the coefficients $w_i$ and abscissae $x_i$ dynamically (see [4]) and the use of such adaptive formulas for hypothetical parallel computers is analyzed in [2], [3] and [5]. Further developments and algorithms for both hypothetical and real (Texas Instruments ASC and ILLIAC IV) parallel computers are given in [1]. This paper considers speedup of parallel algorithms for hypothetical computers which have a large number of independent asynchronous processors and an unbounded memory. Let $T_N^f$ be the time required to compute $Q_N^f$. The metalgorithm of [2], [3] and [5] using $P+2$ processors only has

$$T_N^f \leq C_1 (N/P) + C_2 N$$
and even though $C_2$ is small compared to $C_1$, this is no speedup at all. A little thought shows that this metalgorithm is too general to obtain any speedup; there are algorithms represented by this metalgorithm which, as seen in [5], have $T_N f = O(N)$. Faster algorithms are developed in [1] for such MIMD computers and various speedup results established. This paper presents a version of one of the results of [1] which is a fast metalgorithm where the speedup using $M$ processors is $(\log_2 M)/M$.

Adaptive quadrature, of course, involves certain numerical analysis considerations, but they do not play a role here except to assure that the set of algorithms under consideration is not vacuous. For this reason we refer the reader to [2], [3] and [5] for a discussion of this aspect of the problem. For completeness we do state the two assumptions used in that analysis.

**ASSUMPTION 1.** The integrand $f(x)$ has singularities

$S = \{s_j | j = 1, 2, \ldots, J; J < \infty\}$

and set $w(x) = \prod_{j=1}^{J} (x-s_j)$.

(i) There are constants $r > 2$, $K$ and $\alpha > 0$ so that

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

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


Adaptive algorithms use certain bounds on the error in an interval $[x, x+2^{-k}]$ which are denoted by $\text{ERROR}(x,k)$.

**ASSUMPTION 2.** With the constants of Assumption 1 we have

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

$$\text{ERROR}(x,k) < K \left( \max_{x_0 \in [x, x+2^{-k}]} |f(x_0)| 2^{-kr} \right)$$

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

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

2. **THE FAST METALGORITHM**

The metaalgorithm for the adaptive quadrature computation is shown in Figure 1. The components and their functions are as follows:

**Interval Processors:** These $P$ processors take an interval, split it into two new intervals, compute area estimates and error bounds for each of them and test whether each new interval should be discarded (depending on the size of the corresponding error bounds). In addition, each of these processors manages its own queue of intervals and perhaps passes some of its new intervals to other processor queues. In Figure 1, we indicate that intervals passed from processor $i$ are put in queue $Q(i)$ or $Q(i+1)$, but other configurations are possible. Further, these processors pass certain information to the update processors for transmittal to the algorithm controller. This information consists of
(a) the area estimate from the two intervals just created. Technically, the change in the area estimate due to the splitting and new computation is passed.

(b) the error bound from the two intervals just created (or actually, the change),

(c) the length of its queue.

We assume that $P$ is a power of 2 for simplicity of exposition, but the proofs may be easily modified to treat an arbitrary value of $P$.

Update Processors: These processors combine the information from the interval processors or other update processors and transmit it toward the algorithm controller (along the dotted lines):

(a) the total change in the area estimate,

(b) the total change in the error bound,

(c) the index and lengths of the longest and shortest queues under its purview.
Queues \(Q(i)\). The collection of intervals being processed (active) is divided into \(P\) queues, one per interval processor. Each queue may receive intervals from a fixed set of interval processors whose number is bounded by \(Q_{MAX}\) independent of \(P\) (two sources are shown in Figure 1 for a natural pattern of distributing intervals). The flow of intervals is indicated by the light lines.

Queue Balancing Processors. These \(D\) processors may, from time to time, be used to move blocks of \(D\) intervals from long queues to short queues. The number \(D\) is related to \(P\) and this mechanism is described in more detail later.

We now make certain definitions and list assumptions about them. Some of these assumptions are obviously realistic, we present some explanation for others. The processing time is the time for an interval processor to compute areas, error bounds and to make auxiliary computations for numerical quadrature for one interval. The delivery time is the time for an interval processor to locate and obtain an interval from its queue (if one is available) to be ready for quadrature.

**ASSUMPTION 3.** The processing of an interval requires \(q\) evaluations of \(f(x)\), \(1 < q < q'\) and the processing time is less than a constant \(C_0\). The delivery time is less than a constant \(C_0\).
The **insertion time** is the time required for an interval processor to insert any resulting intervals into the appropriate queues. Note that conflicts may arise in the access to the tails of the queues due to simultaneous attempts to add intervals. Since there is an absolute bound $Q_{\text{MAX}}$ in the number of processors which want to access any one queue, the following assumption is reasonable.

**ASSUMPTION 4.** The insertion time is less than a constant $C_{0}$.

See [5] for a queue access control mechanism which satisfies this assumption.

The **return time** is the time required for information from an interval processor to return to the algorithm controller via the tree of update processors.

**ASSUMPTION 5.** There are constants $C_{0}$ and $C_{1}$ so that the return time is less than $C_{0} + C_{1} \log_{2} P$.

It is clear that there are at most $\log_{2} P$ levels in the tree of update processors and thus one needs only assume that the time for each update processor to receive, process and retransmit information is constant. The processing is simple arithmetic and comparison operations independent of $P$. While conflicts in access may arise due to the parallel nature of the computations, these may be easily handled in a fixed time since at most two processors ever want access to an update processor. Thus this assumption is reasonable.
The queue balancing time is the time required to move D intervals from the tail of one queue over to another queue. This involves locating the original and final positions of all queue elements and making the actual transfer of information within the memory. It also includes the time to resolve access conflicts for the queues involved. The information as to which queues to balance is available from the algorithm controller and the criterion for balancing is:

Whenever the difference between the longest and shortest queue is 2D or more, then move D intervals from the tail of the longest queue to the tail of the shortest queue.

This feature of the metalgorithm keeps all the interval processors busy unless there is only a small number of intervals to be processed. We show later that D may be chosen proportional to $P \log_2 P$ so that a fast algorithm results.

ASSUMPTION 6. The queue balancing time is less than a constant $C_0$.

This assumption requires certain care about the organization of memory in order to be reasonable. For example, if the queues were actually maintained as linked lists in memory, then this assumption would be violated because one must
trace down the list (of length proportional to \( P \)) in order to locate all the intervals to be moved. If the queues are maintained in sequential arrays with contiguous locations, then this difficulty does not arise. This, of course, implies a two-dimensional memory, but the same effect can be obtained for a linear memory by inter-lacing the queues with \( Q(i) \) having addresses equal to \( i \) modulo \( P \). The actual movement of the information and the access conflicts with the queues cause no difficulties for this assumption.

We may summarize any algorithm from this metalgorithm as follows:

A. Initialize by placing \([0,1]\) in \( Q(1) \).

B. Process intervals by the local quadrature rule where Assumptions 1 and 2 are satisfied and determine whether to retain or discard the new intervals generated.

C. Balance the queues whenever the longest is (at least) \( 2D \) longer than the shortest.

D. Terminate the computation when the error bound is less than the specified accuracy requirement.

A basic quantity in the analysis of this metalgorithm is the cycle time \( T_c \), which is the maximum total time elapsed
from the moment delivery is initiated to the completion of the insertion of any new intervals generated, the receptions by the algorithm controller of resulting information and the completion of the queue balancing if that information triggers this action. It is clear that

\[ T_c \leq 5C_0 + C_1 \log_2 P \]

where the right side is the sum of the bounds on the processing, delivery, insertion, return and queue balancing times. In actual practice the overlap of these actions would lead to a smaller cycle time.

3. **THE SPEED-UP THEOREM.**

The key to obtaining the appropriate speed-up is to keep all the interval processors busy most of the time. The queue balancing processors achieve this, but the situation is complicated by the dynamic nature of the computation. In order to analyze this we introduce growth rates for the queues:

- \( G_1 \) = maximum number of intervals which can be added to a queue in one cycle time
- \( G_2 \) = maximum number of intervals which can be removed from a queue in one cycle time
- \( G = G_1 + G_2 \)
The time of the computation is divided into units of one cycle $T_C$ and thus time $k$ means $k*T_C$ since the computation was initiated. We use the notation that $l(i)$ is the length of $Q(i)$, $W_k(i)$ is the $(P-i+1)$st longest queue at time $k$ (e.g. $W_k(P)$ is the longest, $W_k(1)$ is the shortest) and finally we set

$$d_k(i) = I(W_k(P-i+1)) - I(W_k(i)) \quad i = 1, 2, \ldots, P/2$$

**Lemma 1.**

$$I(W_{k+1}(i)) \geq I(W_k(i)) - G_2$$

*Proof.* Suppose $W_k(i) = j$ and $W_{k+1}(n) = j$ so that $Q(j)$ is in position $i$ at time $k$ and position $n$ at time $k+1$. The proof has three cases.

**Case 1:** $n = i$. $Q(j)$ stays in the same position and by definition it cannot decrease in length by more that $G_2$.

$$I(W_{k+1}(i)) = I(W_{k+1}(n)) \geq I(W_k(i)) - G_2$$

**Case 2:** $n < i$ ($Q(j)$ moves down in rank). By definition $I(W_{k+1}(i)) \geq I(W_{k+1}(n))$ and since $I(j)$ cannot decrease by more than $G_2$ we have

$$I(W_{k+1}(i)) \geq I(W_{k+1}(n)) \geq I(W_k(i)) - G_2$$
Case 3: \( n > i \) (\( Q(j) \) moves up in rank). There must be queues with ranks \( s \) and \( t \) with \( t < i < s \) so that \( W_k(s) = r \) and \( W_{k+1}(t) = r \) since some queue of rank higher than \( i \) must be pushed down to rank \( i \). Clearly we have

\[
I(W_k(s)) > I(W_k(i)) \geq I(W_{k+1}(i)) \\
I(W_{k+1}(i)) > I(W_{k+1}(t)) \geq I(W_k(s)) - G_2 > I(W_k(i)) - G_2
\]

This concludes the proof.

**Lemma 2.**

\[
I(W_{k+1}(i)) \leq I(W_k(i)) + G_1
\]

**Proof:** The proof consists of three cases as in Lemma 1 and follows the same lines of reasoning. We omit the details.

These lemmas allow us to bound the change in the spread between the lengths of queues as follows.

**Corollary 1.** \( d_{k+1}(i) \leq d_k(i) + G, \ i = 1, 2, \ldots, P/2 \)

**Proof:**

\[
d_{k+1}(i) = I(W_{k+1}(P-i+1)) - I(W_{k+1}(i)) \\
\leq I(W_k(P-i+1)) + G_1 - I(W_k(i)) + G_2 \\
= d_k(i) + G.
\]
We now turn to the question of how long a situation with $d_k(i) > 2D$ can last, i.e. how many cycles in succession can queue balancing be required.

**Lemma 3.** Suppose that at time $k$, $d_k(i) < 2D$ for all $i$ and at times $k+1, k+2, \ldots, k+m-1$ there is at least one $d_k(i) > 2D$. Then at times $k+n+1, n = 1, 2, \ldots, m-1$ there must be at least $n$ differences $d_{k+n+1}(i)$ less than $(n+1)G-1$. Further, for all $i$ we have

$$d_{k+n}(i) \leq 2D+nG-1, \quad n = 1, 2, \ldots, m-1$$

**Proof:** It is obvious that $d_{k+n}(i) \leq 2D+nG-1$ from Corollary 1. The rest of the proof is by induction on $n$. For $n = 1$ we see that $d_k(i) \leq 2D-1$ so from Corollary 1 we have $d_{k+1}(i) \leq 2D+G-1$ for all $i$. Since queue balancing is performed in cycle $k+1$, we have at time $k+2$ one queue is reduced in length by $D$ and another increased in length by $D$. Hence at least one difference is bounded by $2D + 2G - 2D - 1 = 2G - 1$.

Assume the lemma is true for $n$, then the queue balancing initiated at time $k+n+1$ reduces one difference which was greater than $2D$ (but less than $2D + (n+1)G-1$) by $2D$. The growth in the queues may add $G$ to this difference and all others which are bounded by $(n+1)G-1$ at time $k+n+1$. Hence there are at least $n+2$ differences less than $(n+2)G-1$ at time $k+n+2$ which concludes the induction step and the proof.
LEMMA 4. There are constants $C_2$ and $C_3$ so that

$$G \leq C_2 + C_3 \log_2 P.$$  

Proof: The minimum time to process an interval from

a queue is the time $E_f$ for one function evaluation of

$f(x)$. This assumes zero delivery and return time and

minimal computation in the quadrature. Thus at most

$T_c/E_f$ intervals can be removed from any $Q(i)$ is one

cycle, i.e.

$$G_2 \leq (5C_0 + C_1 \log_2 P)/E_f$$

Thus the maximum number of new intervals produced by

any one processor in one cycle is $2G_2$. So at most

$2Q_{MAX}G_2$ are available to add to any one queue within

one cycle, which implies

$$G_1 \leq 2(5C_0 + C_1 \log_2 P)/Q_{MAX}/E_f$$

These bounds may be combined to establish the lemma.

LEMMA 5. If

$$D = (P+1)((C_2 + C_3 \log_2 P)/4$$

then, for all $i$ and all $l$, we have

$$d_k(i) < 4D$$

Proof: Suppose $d_k(1) < 2D$ and $d_{k+1}(1) \geq 2D$,

then by at least time $k+m+1, m = P/2$ it follows from

Lemma 3 that there must be at least $P/2$ differences less
than \((P/2+1)G-1\). Since there are only \(P/2\) differences we see that by time \(k+P/2+1\) the maximum difference is

\[(P/2+1)[C_2 + C_3 \log_2 P] - 1 < (P+1)[C_2 + C_3 \log_2 P]/2 = 2D\]

Thus the queues can remain unbalanced for at most \(P/2\) cycles and from Lemma 3 we see the maximum difference within this time period is

\[2D + P/2G-1 < 4D\]

which concludes the proof.

The main result of this paper is

**THEOREM.** With Assumptions 1 through 6 for this metaalgorithm and \(D = (P+1)[C_2 + C_3 \log_2 P]/4\) queue balancing processors we have for all \(N > (P \log_2 P)^2\)

\[T_N^f \leq C_4 (N/P) T_C\]

where \(C_4\) is a constant independent of \(P\) and \(N\).

Before completing the proof of this theorem we rephrase it in a way to make the speed up obtained more obvious

**COROLLARY 2.** With the assumptions of the Theorem and \(M\) total processors we have, for \(N > (M^2 \log_2 M)^2\)

\[T_N^f \leq C_6 \log_M(N)\left(1 + C_7 \log_2 M\right) \leq C_8 \log_M\left(\frac{N}{M}\right)\]

where \(C_6, C_7\) and \(C_8\) are constants independent of \(N\) and \(M\).
Proof. Let $A_k$ be the number of active processors (processors with $I(i) > 0$ or which are processing an interval) at time $k$ and we divide time into two parts:

$$Y_1 = \{ k | A_k = P \}$$

$$Y_2 = \{ k | A_k < P \}$$

of sizes $y_1$ and $y_2$, respectively. Let $L$ be the total number of intervals processed, then $y_1 \leq L/P$.

For the times in $Y_2$ we have at least one empty queue and thus by Lemma 5 the maximum queue length for these times is $4D$. All intervals in these queues are processed within $4D$ cycles and hence within $4D$ cycles there must be at least one step made down the longest path in the binary tree of intervals (see [2] for more details on the tree) generated by the metalgorithm. Let $d$ be the length of this longest path and thus we have $y_2 \leq 4dD$.

It is shown in [2] that

$$d \leq (r/a) \log_2 [N/(2q' C K^{1/T})]$$

where $r$, $a$ and $K$ are constants from Assumption 2 and $C$ is an absolute constant. We have then that, with $C$ a generic constant whose value changes from line to line,

$$T_N^f \leq (y_1 + y_2)T_C \leq (L/P + 4dD)T_C$$

$$\leq [(N/P)q' + 4D r \log_2 N - C]T_C$$

$$\leq [(N/P)q' + (P+1)(C_2 + C_3 \log_2 P)r \log_2 N]T_C$$

$$\leq C(N/P)[1 + P(D+1)(C_2 + C_3 \log_2 P)(r \log_2 N)/(Nq')]T_C$$
From the assumption that \( N > (P \log_2 P)^2 \) and the fact that \( \log_2 \log_2 P / \log_2 P \) assumes its maximum value at \( P = 7 \) which is less than 0.5305 it follows that

\[
P(P+1)(C_2 + C_3 \log_2 P)(r \log_2 N)/(Nq') < C
\]

and hence that

\[
T_N f \leq C(N/P)T_c
\]

which concludes the proof.

REFERENCES


