ADAPTIVE QUADRATURE ALGORITHMS FOR ILLIAC IV

James M. Lemme

Department of Computer Science
Purdue University
Lafayette, Indiana 47907

CSD TR 189

May 1976

ABSTRACT

Two classes of adaptive quadrature algorithms for use on ILLIAC IV are described, one designed for high processor utilization, the other for high processing speed. The results of simulation tests comparing the two classes are summarized and commented on. These indicate that a more truly adaptive algorithm without consistently high processor utilization is faster than one designed specifically for high utilization.

ADAPTIVE QUADRATURE ALGORITHMS FOR ILLIAC IV

James M. Lemme

1. INTRODUCTION

The ILLIAC IV uses the single-input stream, multiple-data stream (SIMD) concept of parallel processing. It is necessary that new algorithms be designed to take advantage of the particular characteristics of ILLIAC IV. In this paper we present two possible algorithms for performing automatic numerical integration on ILLIAC IV. A similar less detailed study for Texas Instruments' ASC is reported in [7]. It is seen that quadrature algorithms for a pipeline machine resemble sequential algorithms much more closely than do those for ILLIAC IV. This makes pipeline quadrature algorithms easier to design than completely parallel ones.

In ILLIAC IV a single control unit (CU) decodes an instruction and issues it simultaneously to 64 processing elements (PE). Each PE then executes the instruction, operating on elements of its private memory (PEM) or on constants broadcast by the CU. The CU is also a processor in its own right, keeping track of loop counters and performing other housekeeping tasks.

Even though the CU issues commands to all PE's, it is possible for a PE to disable itself on the basis of a local test. This is done by disabling the writing of information into registers and memory by that PE, and it is thus possible to select particular PE's to have an effect on the calculation.

The only memory elements which may be accessed directly by a PE are those located in its private memory, consisting of 2048 64-bit

words. It is, however, possible for PE's to transfer data from one to another via routing instructions. Direct routing connections exist between PE's one and eight units apart, so it is possible to transfer information from any PE to any other by means of a sufficient number of routing instructions.

The memory element at a given address in a PEM is a single 64-bit word. A collection of memory elements at a given address, one per PEM, is called a super word (or sword) [6]. Just as a conventional computer operates on a single word with a single instruction, ILLIAC IV is able to operate on a super word with a single instruction, since all 64 PE's simultaneously execute the instruction issued by the CU. For a more detailed description of the ILLIAC IV configuration and hardware see [1,2,3,4].

2. SWORD-ADAPTIVE ALGORITHMS

The above description provides enough information to understand the nature of the special integration algorithms for ILLIAC IV. We use an adaptive approach in an attempt to reduce the number of function evaluations required to accurately approximate a given integral. See [8] for the appropriate background in adaptive quadrature.

In order to obtain consistently good utilization of the PE's we perform as many operations as possible in terms of 64-interval units, i.e., swords of intervals. In particular, the intervals in a sword are discarded as a unit, when the sum of the error estimates associated with the intervals becomes less than the total length of the intervals in the sword times the required global error tolerance, EPS. This type of algorithm is termed sword-adaptive since it is adaptive in

the sense that it reacts to local error estimate size, but does so only in terms of swords.

Along with discarding, the other important feature of adaptive quadrature is the splitting of intervals with error estimates which are not yet small enough and the subsequent processing of the resulting interval halves. This phase is also carried out on swords as a unit, resulting in a sword of left halves and one of right halves.

The collection of intervals waiting for processing is made up of a vector of interval swords. Each sword of intervals is actually made up of several swords, one containing the left endpoints, one the right endpoints, etc. Such a sword of intervals is split by an appropriate reassignment of the endpoint and midpoint swords (and the corresponding function value swords), resulting in the sword of left half-intervals occupying the same position in the collection that the old sword did and the sword of right half-intervals forming a new "level" (vector index position) in the collection of interval swords.

The basic form of a sword-adaptive algorithm is as follows.

Initialization. The interval of integration is broken up into 64 equal intervals, and each is put into a different PEM. The endpoints are saved, and the midpoint is calculated for each interval, then the function values for these three points are calculated and saved.

Body. One pass is made through all levels, forming area and error estimates for each sword which resulted from splitting another in the previous iteration (this may include all swords in the collection or just part of them, depending on when discarding is done). The area

estimate for a sword of intervals is a sword containing the estimate for each interval in the sword, while the error estimate for a sword is a scalar, the sum of the error estimates for the intervals in the sword.

During a given iteration all active intervals in the collection have the same length, so each active sword of intervals covers the same length. The initial sword would be discarded if its error estimate was less than EPS, each of its immediate successors would be discarded if they had an error estimate less than EPS/2, and so on. Thus for a particular iteration there is a fixed discard criterion for all active swords. When the new error estimate is formed for a sword, it is compared to this criterion, and a flag is set indicating whether or not the error estimate is satisfactory. If it is, the sword error estimate SWERR is added to OLDERR, which accumulates the sum of the errors in discarded swords, otherwise SWERR is added to ERROR, which contains the sum of the errors in active swords.

When all swords in the collection have current area and error estimates, the global error, given by ERROR + OLDERR, is compared to EPS. If the error is less than EPS the algorithm can be terminated normally. Otherwise unsatisfactory swords must be split and other manipulations performed on the interval collection, depending on the particular algorithm.

Discarding. When a sword is discarded its area estimate sword is added to ASUM, a sword containing the accumulated area estimates for discarded swords. The error estimate for that sword has already been saved in OLDERR, and the information regarding the sword is simply written over when the space is needed.

Termination. If at any time the collection space overflows, a flag is set and control transfers here. Otherwise, this point is reached when the global error estimate is smaller than the required tolerance. In either case the area estimate sword for each level remaining in the collection must be added to ASUM, then the overall area estimate is the sum of the elements of ASUM. This value is returned and the algorithm terminates.

Super Word Sum. In forming values such as the error estimate associated with an entire sword, it is necessary to add up the elements. It is essential that this be done in as efficient a manner as possible. Kuck [5] describes such an algorithm which exhibits a good degree of parallelism. A total of fourteen routes is required to transfer partial sums distances of 1, 2, 4, 8, 16, and 32 units.

Interval Collection Management Schemes. Three different schemes are considered for managing the interval collection, differing only in when satisfactory swords are discarded. The first is called "Discard All Satisfactory Swords (DASS)." Here, after making the pass through the collection to form new area and error estimates, a second pass is made to discard satisfactory swords and compact the remaining collection. A third pass splits the remaining swords.

The second scheme is called "Discard On Overflow (DOO)," where no discards are done until the splitting of a sword causes the overflow of the interval collection. At this point a sword whose error estimate is small enough is discarded, making room for the right half of the sword being split. The splitting of swords then continues, each splitting requiring the location of a satisfactory sword to discard.

The third scheme is similar to DASS, only it performs the second and third passes through the collection in a single pass. This scheme is called "Single Pass (SIP)," and requires more complicated logic. A single pass is made through the sword collection where satisfactory swords are discarded and unsatisfactory ones are split. When a sword is split another is found to discard to create an opening for the new sword. When a sword is discarded another is found to split to fill the space left open. The result is a compacted collection, ready for the next area and error estimate formation.

3. AN ADAPTIVE ALGORITHM

The sword-adaptive algorithms described above have one major disadvantage. Only one interval with a large error estimate, perhaps due to a singularity, may cause the error estimate for the sword to be larger than the discard criterion, resulting in the division of all 64 intervals simply because one interval contains a bad point. By introducing more routing instructions we arrive at an algorithm which behaves more like a standard adaptive one: only intervals with unsatisfactory error estimates are retained and split. The reduced number of intervals to process may compensate for the time spent in doing routing.

In order to arrive at such an algorithm we keep a queue of intervals in each PEM. The interval at the head of each queue can then be simultaneously fetched and processed by the corresponding PE. A PE whose queue is empty is disabled during this processing period.

When processing is complete some PE's will have interval halves which have not been discarded. Any PE which has at least one retained interval routes it to the next higher-numbered PE, where it is placed at the tail of the queue. Any PE which retains both interval halves places the second at the tail of its own queue.

The entire adaptive algorithm for ILLIAC IV is as follows.

Initialization. The original interval of integration is broken up into 64 intervals, and one is placed in each of the 64 queues. Area and error estimates are formed for each of the intervals, and are entered as the original estimates for the corresponding PE. Each PE maintains a part of the global area and error estimates in its PEM. The part in each PE is modified as local estimates change due to processing done by that PE. The PE error estimates are summed to get a global error estimate, and if this is less than the required tolerance the algorithm can be terminated without further work.

Body. Remove the interval from the head of each (nonempty) queue. Split it into two intervals, left and right, and form area and error estimates for each of the halves. Update the PE parts of the global estimates by adding in the changes that result. If the global error estimate is smaller than the required tolerance, terminate the algorithm. If the global error estimate is not yet satisfactory, the algorithm proceeds.

Each left half whose error estimate is not yet satisfactory is routed to the next higher-numbered PE, where it is placed at the tail of the queue. Next, each PE which did not pass the left half (having discarded it instead), but which has a right half with an unsatisfactory

error estimate, routes that right half to the next higher-numbered PE. Finally, any PE which retains both halves places the right half at the tail of its own queue.

All intervals formed in the previous iteration have now been disposed of, either by discarding or placing them at the tail of some queue. Return to the beginning of the algorithm body to again remove the interval from the head of each queue.

Termination. If at any time a queue overflows, a flag is set and control transfers here. Otherwise, this point is reached when the global error estimate is smaller than the required tolerance. In either case the PE parts of the global area estimate are summed and this value is returned, terminating the algorithm.

4. ALGORITHM COMPARISONS

In order to compare the performance of these algorithms we simulated the operation of each algorithm for ILLIAC IV, using a FORTRAN simulation which accumulates the times required to perform each step on ILLIAC IV. Timing information and values were obtained from [3].

For experimental purposes we have chosen to use Simpson's rule as the area estimation algorithm because of its familiarity, relative simplicity, and because it makes good use of previously calculated function values. Any integration rule which does not depend on the local behavior of the integrand would be satisfactory, including any combination of Newton-Cotes rules, and Gauss or Gauss-Kronrod rules. Schemes such as Romberg quadrature, where the time to generate area

and error estimates may vary greatly from interval to interval, may be used, but much of the paralleism of the overall algorithm is lost.

The sword-adaptive algorithms and the adaptive algorithm were tested on fourteen integrals with error tolerances 10^{-3} and 10^{-6}. The test integrals are given in Table 1. The three sword-adaptive algorithms differed only slightly in their performance, and in no consistent way. Overall the DASS management scheme proved slightly faster, so we shall use those times as the basis for comparison.

Table 2 contains a summary of the test results for DASS and the Adaptive ILLIAC IV Algorithm (AIA). In the table "Func" is the function number, "Eps" is the required error tolerance, "Total Time" is the simulated time in microseconds required to solve the problem, "Eff" is the efficiency described below, "Eval" is the number of parallel function evaluations required, "Errest" is the error estimate given by the algorithm, "Error" is the actual error, each for DASS/AIA. The three values associated with the error are given as a two-digit mantissa followed by the base 10 exponent.

The efficiency is a measure of how effectively parallelism is utilized in the algorithms. This is measured differently for DASS and AIA. In DASS it is the time spent doing PE operations divided by Total Time. For AIA it is calculated by dividing the sum of micro-seconds times active processors by the total time multiplied by the number of processors (64).

DASS was designed for consistently good PE utilization, and that aim seems to have been realized. The efficiency for DASS ranges from .605 to .703, while a range from .332 to .754 occurs for AIA. However,

Table 1

Test Integrals $\int_a^b f(x)dx$

Func	f(x)	a	b	Exact value
1	F=X**(1./16.)	0	1	.941176470588
2	F=ABS(X-.3654782)**0.7	0	1	.377733929561
3	F=SIN(314.159265359*X)	0	1	0.
4	F=IFIX(10.*X)	0	1	4.5
5	F=0. IF (X .GT. 0.) F=ALOG(X)	0	1	-1.
6	F=1./(1.+.5*SIN(31.4159*X))	0	1	1.154700669
7	F=0. IF (X .NE. 0.) F=X/(EXP(X)-1.)	0	1	.7775046341
8	F=1./(1.+(230.*X-30.)**2)	0	1	.013492485650
9	F=1./(X**4+X*X+0.9)	-1	1	1.582232964
10	F=.46*(EXP(X)+EXP(-X))-COS(X)	-1	1	.4794282267
11	F=50./(2500.*X*X+1.)/3.14159	0	10	.4993638029
12	F=50. IF (X .GT. 0.) F=(SIN(50.*3.14159*X))**2 /((3.14159*X)**2*50.)	.01	1	.11213956963
13	F=SIGN(1.+X*X,SIN(X))	-10	10	0.
14	F=T_{20}(X) (Chebyshev polynomial of degree 20)	-1	1	-.00501241332

Table 2

Simulation results for DASS and AIA. Values are given first for DASS then for AIA with a slash separation. Floating point numbers are in the format 2-digit mantissa, exponent sign, exponent.

Func	Eps	Total Time	Eff	Eval	Errest	Error
1	1.0-3	249/242	.667/.688	5/5	8.5-4/8.5-4	7.9-5/7.9-5
	1.0-6	2584/2516	.655/.582	45/45	8.1-7/5.6-7	5.2-7/5.0-7
2	1.0-3	255/243	.675/.695	5/5	3.4-5/3.4-5	1.0-5/1.0-5
	1.0-6	1217/919	.671/.635	21/17	9.4-7/8.1-7	4.0-7/2.6-7
3	1.0-3	595/570	.622/.641	17/17	4.6-4/4.7-4	3.1-15/3.1-15
	1.0-6	2312*/3488	.638/.647	65/93	1.8-6/9.1-7	9.6-16/2.5-15
4	1.0-3	1434/470	.605/.351	73/25	7.3-4/7.3-4	8.1-5/8.1-5
	1.0-6	5525/1354	.620/.332	273/65	7.2-7/7.2-7	8.0-8/8.0-8
5	1.0-3	844/794	.636/.566	21/21	7.8-4/7.8-4	8.8-4/8.8-4
	1.0-6	5250/2798	.668/.525	129/69	8.7-7/7.1-7	3.4-7/7.3-7
6	1.0-3	169/162	.645/.675	5/5	4.5-5/4.5-5	0./0.
	1.0-6	652/627	.655/.628	17/17	1.8-7/3.3-7	0./5.0-9
7	1.0-3	331/291	.644/.690	9/9	6.5-4/6.5-4	6.5-4/6.5-4
	1.0-6	1994/1951	.625/.519	49/49	6.4-7/6.4-7	6.4-7/6.4-7
8	1.0-3	90/83	.611/.661	5/5	5.0-4/5.0-4	7.3-5/7.3-5
	1.0-6	1049/670	.654/.460	45/29	2.0-7/7.7-7	4.8-10/2.0-8
9	1.0-3	90/83	.611/.662	5/5	1.0-8/1.0-8	0./0.
	1.0-6	90/83	.611/.662	5/5	1.0-8/1.0-8	0./0.
10	1.0-3	241/234	.656/.674	5/5	1.7-10/1.7-10	0./0.
	1.0-6	241/234	.656/.674	5/5	1.7-10/1.7-10	0./0.
11	1.0-3	483/352	.617/.504	21/17	8.6-4/8.5-4	2.5-5/2.0-5
	1.0-6	2693/1684	.670/.410	117/69	5.2-7/3.6-7	1.8-8/1.7-8
12	1.0-3	356/314	.663/.710	9/9	5.5-4/5.5-4	9.0-6/9.0-6
	1.0-6	2656/1904	.692/.629	65/45	1.4-7/8.7-7	2.1-9/8.4-9
13	1.0-3	11860/2327	.667/.502	317/61	9.0-4/9.0-4	3.2-6/3.2-6
	1.0-6	23434/4065	.669/.492	625/105	5.1-7/5.1-7	1.6-9/1.2-9
14	1.0-3	227/186	.674/.754	9/9	1.9-4/1.9-4	1.1-5/1.1-5
	1.0-6	881/1023	.703/.600	33/37	7.4-7/8.9-7	7.6-8/8.2-8

*Interval collection overflowed.

in every case but one AIA is faster than DASS, in some instances much faster.

The speed advantage of AIA is usually due to a reduced number of function evaluations. The sword-adaptive algorithms do many unnecessary function evaluations in splitting an entire sword when it contains only a few bad intervals. The extra work done by AIA in disabling processors and routing intervals is much less than the work involved in doing extra function evaluations. Function 13 is the best example of this, where the sword-adaptive algorithms require 317 function evaluations at a tolerance of 10^{-3} and 625 evaluations at 10^{-6}, while AIA requires 61 and 105 function evaluations, respectively.

Another advantage of AIA is exhibited in function 3 at 10^{-6}, where DASS overflows the 16 level sword collection while AIA does not. This is due to the fact that DASS keeps an entire sword of intervals when a single interval contains a bad point, while AIA discards all but the bad interval. Besides being faster, AIA usually uses less storage space than DASS.

Function 14 with a tolerance of 10^{-6} is the only instance where DASS is faster than AIA. In this case AIA requires 37 function evaluations while DASS needs only 33. This could be caused by one of two things. The more likely cause is that the nature of the integrand caused one queue to become longer than the rest, enough longer so that one additional interval processing is required. The other possibility is that the oscillatory nature of the integrand caused more than one difficult interval to fall in a single sword in DASS, so when the sword is repeatedly split several bad intervals

are simultaneously divided, resulting in a sudden decrease in the global error.

5. CONCLUSIONS

We see that algorithms can be designed which make use of the parallel nature of ILLIAC IV in performing adaptive quadrature. However, the particular algorithm chosen to guarantee consistent PE utilization turns out to be much slower than an algorithm which disables PE's as required. This speed difference is mainly due to the reduced number of function evaluations required by a truly adaptive algorithm.

Several instances of the speedup resulting from ILLIAC IV's multiple processors were investigated. One test involved restricting DASS to one PE and comparing that time to the time for 64 PE's. The remaining instances include comparing the adaptive ILLIAC IV algorithm to a comparable sequential algorithm and varying the number of available PE's. See [7] for details of this study.

The results of the speedup tests are somewhat encouraging. AIA exhibits a significant speed advantage over a comparable algorithm simulated for a sequential machine with the speed of ILLIAC IV, being an average of 15.5 times faster with a range from 1.1 to 46.9 (one function has a completely accidental ratio of .04). However, this is not close to the factor of 64 we might hope for. Also, when the number of PE's is increased the additional speed gain lessens as the number of PE's becomes larger.

Many variations on the particular forms of these algorithms are possible. The scheme according to which the intervals are passed

in AIA could be modified. A smaller number of initial intervals may prove more effective. In any case, the testing done so far simply gives a feel for the type of quadrature algorithms which can be employed effectively on ILLIAC IV. It remains to do actual testing on ILLIAC IV itself. We can see that in order to get fast algorithms it is probably not advisable to aim for high PE utilization.

Acknowledgement. I thank John R. Rice for his many helpful suggestions in preparing this manuscript and in performing the comparison tests.

References

1. Barnes, G. H., The ILLIAC IV computer, IEEE Trans. Comp., 17(1968), 746-757.

2. Bouknight, W. J., et. al., The ILLIAC IV system, Proc. IEEE, 60(1972), 369-388.

3. Burroughs Corp., ILLIAC IV Systems Characteristics and Programming Manual, Defense, Space and Special Systems Group, IL4-PM1, revised May 16, 1972.

4. Davis, R. L., The ILLIAC IV processing element, IEEE Trans. Comp., 18(1969), 800-816.

5. Kuck, D. J., ILLIAC IV software and application programming, IEEE Trans. Comp., 17(1968), 758-770.

6. Lawrie, D. H., et. al., GLYPNIR: A programming language for ILLIAC IV, Comm. Assoc. Comp. Mach., 18(1975), 157-164.

7. Lemme, J. M., Speedup in parallel algorithms for adaptive quadrature, Ph. D. thesis, Purdue University, 1976.

8. Rice, J. R., A metalgorithm for adaptive quadrature, J. Assoc. Comp. Mach., 22(1975), 61-82.