Adaptive integration algorithm using MPI

1 Sequential Algorithm

A basic or local cubature or integration rule is a weighted sum of function values to approximate the integral of a function $f(\vec{x})$ over a specific domain $\mathcal{D}$, so that

$$\int_D f(\vec{x})\ d\vec{x}\ \approx\ Q = \sum_{j=0}^{N-1} f(\vec{x}_j)\ w_j; \tag{1}$$

thus the sum can be considered as a scalar product

$$\vec{v} \cdot \vec{w} \text{ with } v_j = f(\vec{x}_j),\ 0 \leq j < N.$$

For example, the 1D integral is of the form

$$\mathcal{I} = \int_a^b f(x)\ dx \tag{2}$$

and it is approximated by a 1D integration or quadrature rule of the form

$$Q = \sum_{j=0}^{N-1} f(x_j)\, w_j, \tag{3}$$

We will concentrate on the 2D case where the region $\mathcal{D}$ is a rectangle

$$D = [a, b] \times [c, d];$$

then the integral is

$$\mathcal{I} = \int_a^b dx \int_c^d dy\ f(x, y). \tag{4}$$

For the 2D integration rule we will use a "product rule" of a 1D rule with points x_i and weights u_i, with another (or the same) rule with points y_j and weights v_j,

$$Q = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} f(x_i, y_j)\ u_i i\, v_j. \tag{5}$$

Note that an integral over $[a, b]$ can be transformed to the interval $[-1, 1]$ by a transformation or substitution, $x = \frac{a+b}{2} + \frac{b-a}{2}\, t$, so that

$$\int_a^b g(x)\ dx = \frac{b-a}{2} \int_{-1}^1 g(\frac{a+b}{2} + \frac{b-a}{2}\, t)\ dt. \tag{6}$$

This is important because we are going to include the data for the rules with respect to the standard interval $[-1, 1]$, and an integral over general limits $[a, b]$ will be transformed to $[-1, 1]$ so that the standard form of the rule can be used.

Let us now define the *polynomial degree of accuracy* of a rule. A rule is said to be of *polynomial degree (of accuracy) d* if it gives the exact result for every polynomial of degree $\leq d$, but there are polynomials of degree $d + 1$ that are not integrated exactly. For example, the *Gauss-Legendre* rule with N points is of polynomial degree $2N - 1$, which is the maximum possible for a rule with N points (therefore, these rules are called *optimal*).

We will consider two examples, the Gauss rule ($Q = G_5$) with $N = 5$ points and the Gauss rule ($Q = G_7$) with $N = 7$ points. These are of degrees 9 and 13, respectively.

The Gauss rules are *symmetric*, that means the points are symmetric with respect to the mid-point of the interval, and "mirror" points have equal weights. In the interval $[-1, 1]$ the 7-point Gauss rule, G_7, has the following points in the half $[0, 1]$ and symmetric points in the half $[-1, 0]$:

```
0.949107912342758524526189684047851
0.741531185599394439863864773280788
0.405845151377397166906606412076961
0.000000000000000000000000000000000
```

and the withgoing weights are:

```
0.129484966188869693270611432679082
0.279705391489276667901467771423780
0.381830050505118944950369775488975
0.417959183673469387755102040816327
```

The points and weights that determine the 5-point rule in [-1,1] are given by:

```
0.906179845938664
0.538469310105683
0.000000000000000
```

and the withgoing weights are:

```
0.236926885056189
0.478628670499366
0.568888888888889
```

As a 2D rule we will use the product rule $Q_7 = G_7 \times G_7$. The 2D product rule $Q_5 = G_5 \times G_5$ is structured similarly. With the combination of two product rules we will produce an error estimate.

We will start with the original rectangular region or domain specified as part of the input, and the adaptive integration algorithm will work by subdividing the given domain into sub-rectangles and apply the local rule over each sub-rectangle.

The algorithm takes as inputs: a, b, c, d specifying the original rectangle $\mathcal{D}$ in the x, y-plane; the function definition of the integrand function $f(x, y)$ over $\mathcal{D}$; an absolute error tolerance t_a and relative tolerance t_r determining the requested accuracy; and a maximum number of subdivisions, *limit*.

The outputs af the algorithm include: an approximation $R \approx \mathcal{I}$ over the domain $\mathcal{D}$; an estimate E of the absolute error $|R - \mathcal{I}|$; an error flag indicating if the requested accuracy could not be achieved because the maximum number of subdivisions was reached; and possibly other things such as the total number of evaluations of the integrand function needed throughout the integration, or the total number of subdivisions used.

A representation of the adaptive algorithm is given by

```
Evaluate initial region and update results
Initialize priority queue with initial region
while (evaluation limit not reached and
       estimated error too large)
    Retrieve region from priority queue
    Split region into subregions
    Evaluate new subregions and update results
    Insert new subregions into priority queue
```

The priority queue can be represented by a max heap, where the nodes are the subregions, and the heap is keyed with the error estimates over the subregions. Code the algorithm in C.

2 Parallel Algorithm

The given domain is initially partitioned (statically) among the worker processes. Each worker then performs a sequence of subdivision steps consisting of the selection of the next region to subdivide (based on the local region error estimate), its subdivision and the evaluation of the subregions, and intermittent update messages to the controller to report changes in the worker's integral and error estimate.

Most integrands will feature varying function behavior across the integration domain (e.g., singularities or ridges/peaks). Workers who initially receive a difficult portion of the integration domain will have to perform more work to complete their part of the problem than workers whose initial region is easy. The latter workers may become *idle*. Consequently we need dynamic load balancing (LB) to maintain efficient use of the processors. You are asked to implement the algorithm with a load balancing method (e.g., as presented by Kumar et al.). Test the parallel performance of your program. You may try more than one load balancing method and compare the performance (extra credit).

3 Sample test function

$$\int_0^1 dx \int_0^1 dy \, \frac{1}{[xy(x+y)]^\gamma} + \frac{1}{[(1-x)(1-y)(2-x-y)]^\gamma} + \frac{1}{[(1-x)y(1-x+y)]^\gamma} + \frac{1}{[x(1-y)(x+1-y)]^\gamma} \tag{7}$$

for $\gamma = 0.5$ or 0.2. Set the integrand function to 0 where the denominator is zero. Set $t_a = 0, t_r =$1.0 e-15, max. # function evaluations = 40000000 (so the requested accuracy cannot be reached, and the execution takes a long time). The code in the body of the function could be something like:

```
double d1 = pow((x[0]+x[1])*x[0]*x[1],0.5);
double t1 = (d1 != 0.0 ? 1.0 / d1 : 0.0);
double d2 = pow((2.0-x[0]-x[1])*(1.0-x[0])*(1.0-x[1]),0.5);
double t2 = (d2 != 0.0 ? 1.0 / d2 : 0.0);
double d3 = pow((1.0-x[0]+x[1])*(1.0-x[0])*(x[1]),0.5);
double t3 = (d3 != 0.0 ? 1.0 / d3 : 0.0);
double d4 = pow((1.0+x[0]-x[1])*(1.0-x[1])*(x[0]),0.5);
double t4 = (d4 != 0.0 ? 1.0 / d4 : 0.0);
double f = t1+t2+t3+t4;
return f;
```

For $\gamma = 0.2$, the result is approximately 6.670519792.

Note*: This assignment is a very simplified version of the code in the* ParInt *project, which deals with N-dimensional integration over hyper-rectangular and simplex regions, using adaptive, Monte Carlo and "Quasi-Monte Carlo (lattice based)" approaches.* ParInt *was developed to run over MPI. Later on we developed methods for hybrid parallel architectures, i.e., with distributed, multi-core nodes and with accelerators including GPUs and Intel Xeon Phi co-processors.*
We also developed integration techniques based on combinations of the one-dimensional codes from the Quadpack package.

The current load balancing technique for the adaptive algorithms in ParInt *is* scheduler based (sb)*: the controller maintains a list of idle workers. Updates from workers contain information on the worker's load status. The controller/scheduler organizes the transfer of work from non-idle to idle workers.*

4 Further suggestions

- Implement more than one load balancing method and compare the parallel performance.
- Consider the case where your integrand funtion f is itself a 1D integral.
  - Create a 1D integration procedure F using the same algorithm as above. The integration rules are the 5-point and 7-point Gauss rules.
    Your algorithm will implement an approximation to
    $$F = \int_\alpha^\beta g(z)\ dz.$$
    $g(z)$ can be a function of some other parameters, say x and y. Then F will be a function of those parameters as well, and we can write:
    $$F(x, y) = \int_\alpha^\beta g(x, y, z)\ dz. \tag{8}$$

– F is a worthwile candidate to serve as an integrand function in your 2D integration procedure (which you constructed for this assignment). So, plugging (8) into (4) we have

$$\begin{aligned} \mathcal{I} &= \int_a^b dx \int_c^d dy \; F(x,y) \\ &= \int_a^b dx \int_c^d dy \int_\alpha^\beta dz \; g(x,y,z), \end{aligned} \tag{9}$$

i.e., a 3D integral. The computation of each F value gives a reasonable parallel granularity, which helps the parallel performance of your 2D parallel integration for (9).