PPces

Case Study: Adaptive Integration

Dieter an Mey
Center for Computing and Communication
Aachen University (RWTH)

RWTH AACHEN UNIVERSITY Center for Computing and Communication

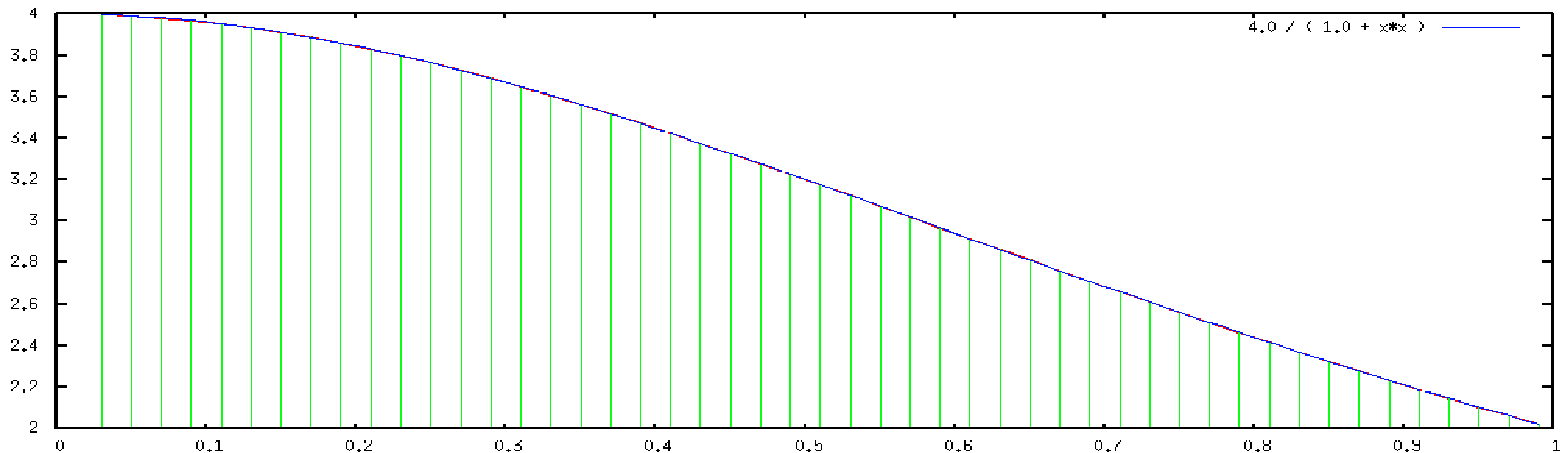
Numerical Integration

π can be calculated by integrating:

$$\pi = \int_0^1 f(x)dx \text{ , with } f(x) = 4/(1 + x^2)$$

This integral can be approximated numerically with a quadrature formula. Here we use the composite midpoint rule:

$$\pi \approx 1/n \sum_{i=1}^{n} f(x_i) \text{, with } x_i = (i-½)/n \text{ for } i=1,...,n$$



Serial Fortran90-Program

```
! statement function
    f(a) = 4.d0 / (1.d0+a*a)
!
    read *, n
    h = 1.0d0 / n
    sum = 0.0d0
    do i = 1, n
      x = h * ( i - 0.5 )
      sum = sum + func(x)
    end do
    pi = h * sum
```

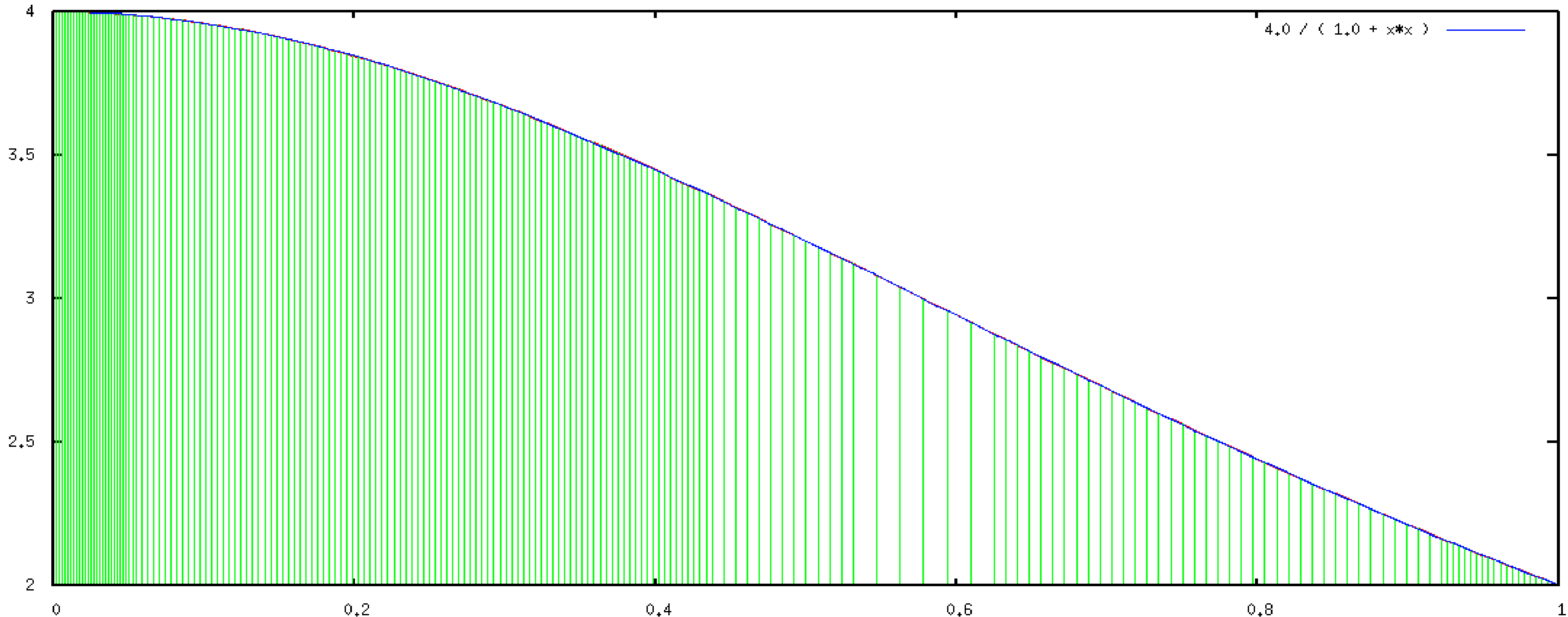
We assume that evaluating the function is expensive

Parallelization using OpenMP is straight forward

```
!  statement function
     f (a) = 4.d0 / (1.d0+a*a)
!
     read *, n
     h = 1.0d0 / n
     sum = 0.0d0
!$omp parallel do private(i,x) reduction(+:sum)
     do i = 1,n
       x = h * ( i - 0.5 )
       sum = sum + f(x)
     end do
!$omp end parallel do
     pi = h * sum
```

More advanced: Adaptive Numerical Integration

Adaptive Integration evaluates the function more frequently where it is less smooth



RWTH AACHEN UNIVERSITY

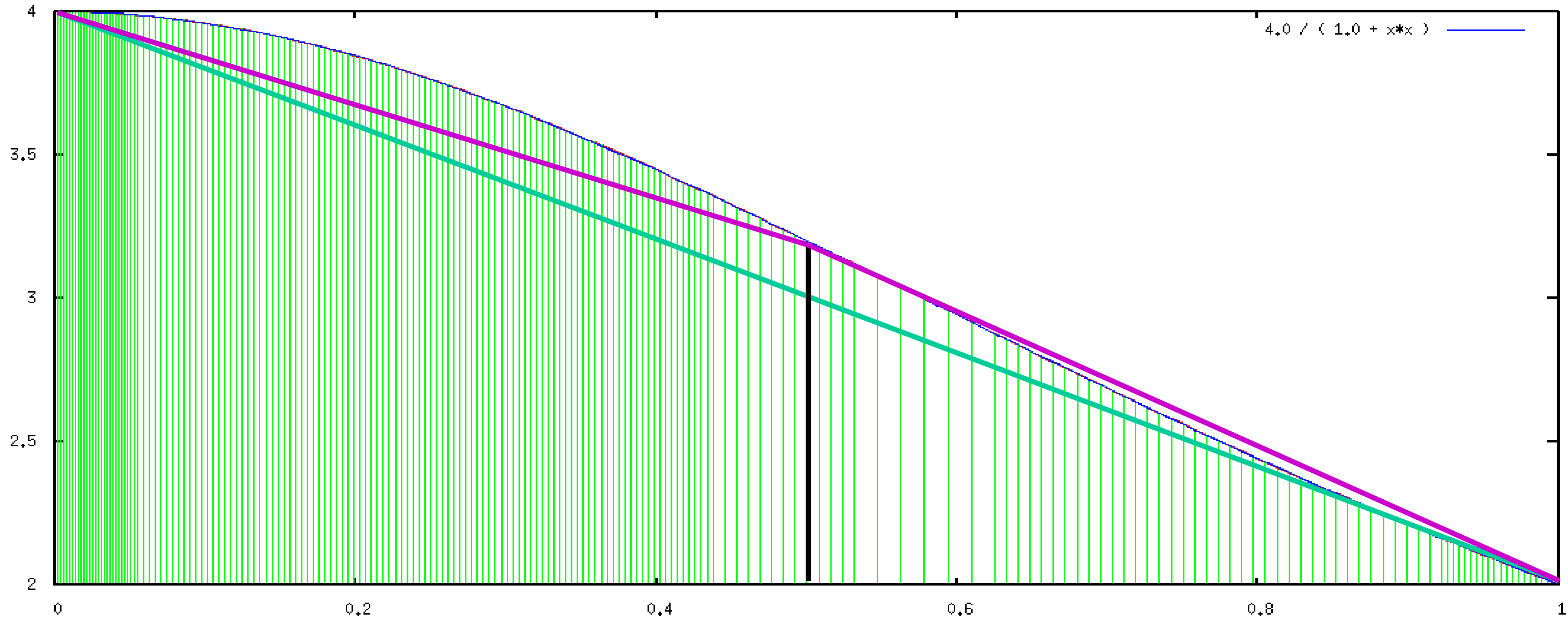
Center for Computing and Communication

Adaptive Integration
Serial Version – Recursive Function

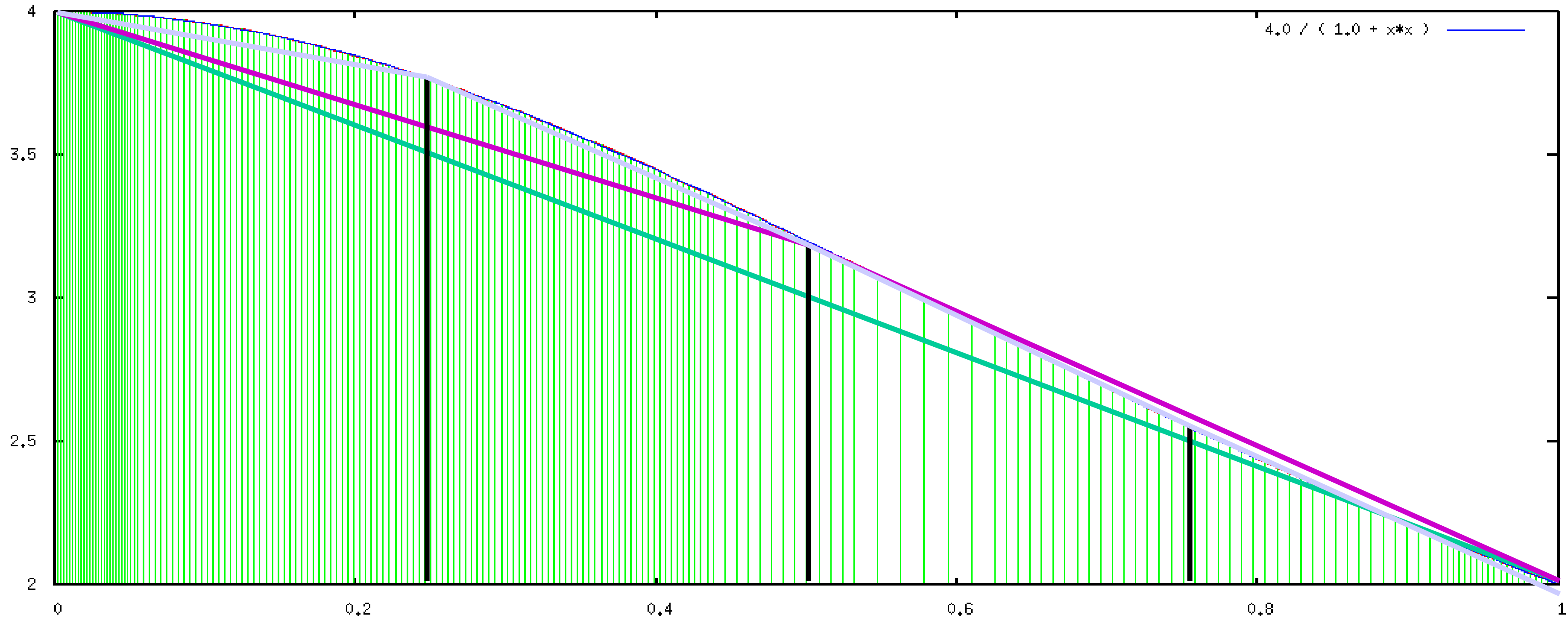
```
recursive function integral (f, a, b, tolerance) &
    result (integral_result)
   interface
   function f (x) result (f_result)
      real, intent (in) :: x
      real :: f_result
   end function f
   end interface
   ..
   h = b - a
   mid = (a + b) /2
   one_trapezoid_area = h * ( f(a) + f(b) )/2.0
   two_trapezoid_area = h/2 * ( f(a) + f(mid) )/2.0 + &
                        h/2 * ( f(mid) + f(b) )/2.0
   if(abs(one_trapezoid_area-two_trapezoid_area)< 3.0*tolerance) &
    then
      integral_result = two_trapezoid_area
   else
      left_area = integral (f, a, mid, tolerance / 2)
      right_area = integral (f, mid, b, tolerance / 2)
      integral_result = left_area + right_area
   end if
end function integral
```

ftp://ftp.swcp.com/pub/walt/F/

More advanced: Adaptive Numerical Integration



More advanced: Adaptive Numerical Integration



Serial Tuning First: Save function calls

```
recursive function integral (f, a, fa, b, fb, tolerance) &
  result (integral_result)
   ...
   h = b - a
   mid = (a + b) / 2
   fmid = f(mid)
   one_trapezoid_area = h * (fa + fb) / 2
   two_trapezoid_area = h/2 * (fa + fmid) / 2 + &
                        h/2 * (fmid + fb) / 2
   if (abs(one_trapezoid_area - two_trapezoid_area) < &
                                             3 * tolerance) then
      integral_result = two_trapezoid_area
   else
      left_area = integral (f, a, fa, mid, fmid, tolerance / 2)
      right_area = integral (f, mid, fmid, b, fb, tolerance / 2)
      integral_result = left_area + right_area
   end if
end function integral
```

The Recursive Algorithm can easily be parallelized by recursively Nesting OpenMP

```
recursive function integral (f, a, fa, b, fb, tolerance)  &
      result (integral_result)
   ...
   h = b - a
   mid = (a + b) /2
   fmid = f(mid)
   one_trapezoid_area = h * (fa + fb) / 2
   two_trapezoid_area = h/2 * (fa + fmid) / 2 + &
                        h/2 * (fmid + fb) / 2
   if (abs(one_trapezoid_area - two_trapezoid_area)  &
         < 3 * tolerance) then
      integral_result = two_trapezoid_area
   else
!$omp parallel sections
!$omp section
      left_area = integral (f, a, fa, mid, fmid, tolerance / 2)
!$omp section
      right_area = integral (f, mid, fmid, b, fb, tolerance / 2)
!$omp end parallel sections
      integral_result = left_area + right_area
   end if
end function integral
```

Preparing the Parallel Version: Serial Stack Mechanism

```
function integral (f, ah, bh, tolerance) result (result)
 ...
   type (stack_t) :: stack
   call new_stack ( stack )
   call push ( stack, ah, bh, tolerance )
   integral_result = 0.0
   do
      if ( empty_stack ( stack ) ) exit
      call pop ( stack, a, b, tolerance )
      h = b - a
      mid = (a + b) /2
      one_trapezoid_area = h * (f(a) + f(b)) / 2.0
      two_trapezoid_area = h/2 * (f(a) + f(mid)) / 2.0 + &
                           h/2 * (f(mid) + f(b)) / 2.0
      if (abs(one_trapezoid_area - two_trapezoid_area)   &
            < 3.0 * tolerance) then
         integral_result = integral_result +
   two_trapezoid_area
      else
         call push ( stack, a, mid, tolerance / 2 )
         call push ( stack, mid, b, tolerance / 2 )
      end if
    end do
 end function integral
```

MPI version: Master - Worker

```
program main
   use mpi
   call MPI_INIT( ierror )
   call MPI_COMM_RANK( MPI_COMM_WORLD, myid, ierror )
   if ( myid == 0 ) then
      call master
   else
      call slave
   end if
   call MPI_FINALIZE (ierror)
end program main

subroutine master
   ...
   x_min = 0.0
   x_max = 1.0
   answer = integral (f, x_min, x_max, 0.00001)
   print *, "result ",  answer, " error ",  abs(answer-pi),
end subroutine master
```

MPI version: Master - Worker

```
subroutine slave
   ...
   private_result = 0.0
   do
      ! wait for work
      call MPI_Send (v,0,MPI_DOUBLE_PRECISION,master,  &
             waitforworktag, MPI_COMM_WORLD,ierror)
      call MPI_Recv (v,5,MPI_DOUBLE_PRECISION,master,  &
             MPI_ANY_TAG,MPI_COMM_WORLD, status, ierror)
      select case (status(MPI_TAG))
         case(readytag)
            exit ! done
         case(poptag)
            a = v(1); fa=v(2); b=v(3); fb=v(4); tolerance=v(5)
         case default
            write (*,*) "error: illegal message tag"
            call MPI_Abort (MPI_COMM_WORLD,ierror)
      end select
      ...
end do

   call MPI_Reduce  (private_result,integral_result,1,MPI_DOUBLE_PRECISION, &
        MPI_SUM,root,MPI_COMM_WORLD,ierror)
end subroutine slave
```

MPI version: Master - Worker

```
subroutine slave
   ...
   private_result =
   do
      ! wait for wo
      call MPI_Send
            waitfo
      call MPI_Recv
            MPI_AN
      select case (
         case(ready
            exit !
         case(popta
            a = v(1
         case defau
            write (
            call MP
      end select
      ...
end do

   call MPI_Reduce (private_result,integral_result,1,MPI_DOUBLE_PRECISION, &
        MPI_SUM,root,MPI_COMM_WORLD,ierror)
end subroutine slave
```

```
       ...
       h = b - a
       mid = (a + b) /2
       fmid = f(mid)
       one_trapezoid_area = h * (fa + fb) / 2.0
       two_trapezoid_area = h/2 * (fa + fmid) / 2.0 + &
                            h/2 * (fmid + fb) / 2.0
       if (abs(one_trapezoid_area - two_trapezoid_area)  &
             < 3.0 * tolerance) then
          private_result = private_result + two_trapezoid_area
       else
          v(1) = a; v(2) = fa; v(3) = mid;
          v(4) = fmid; v(5) = tolerance/2
          call MPI_Send(v,5,MPI_DOUBLE_PRECISION,master,  &
                pushtag, MPI_COMM_WORLD,ierror)
          v(1) = mid; v(2) = fmid; v(3) = b;
          v(4) = fb; v(5) = tolerance/2
          call MPI_Send(v,5,MPI_DOUBLE_PRECISION,master,pushtag,
&                    MPI_COMM_WORLD,ierror)
       end if
       ...
```

```
function integral (f, ah, bh, tolh) result (integral_result)  ! master
   call new_stack ( stack )
   call push ( stack, ah, f(ah), bh, f(bh), tolh )
   integral_result = 0.0; private_result = 0.0
   call MPI_COMM_SIZE( MPI_COMM_WORLD, numprocs, ierror ); busycount = numprocs-1;
   allocate ( slave_status(numprocs-1) ); slave_status(0:numprocs-1) = idle
   do
      call MPI_Recv(v,5,MPI_DOUBLE_PRECISION,MPI_ANY_SOURCE, MPI_ANY_TAG,...)
      select case (status(MPI_TAG) )
       case(waitforworktag)
         busycount = busycount - 1
         if ( empty_stack ( stack ) ) then
            slave_status(status(MPI_SOURCE)) = idle; if ( busycount .eq. 0 ) exit
         else
            call pop ( stack, a, fa, b, fb, tolerance )
            v(1) = a; v(2) = fa; v(3) = b; v(4) = fb; v(5) = tolerance
            call MPI_Send(v,5,MPI_DOUBLE_PRECISION, status(MPI_SOURCE),poptag,...)
            slave_status(status(MPI_SOURCE)) = busy; busycount = busycount + 1
            cycle
         end if
       case(pushtag)
         if ( busycount < numprocs-1 ) then
            do iproc = 1, numprocs-1
               if (slave_status(iproc) == idle ) then
                 call MPI_Send(v,5,MPI_DOUBLE_PRECISION,iproc,poptag, ...)
                 busycount = busycount + 1; slave_status(iproc) = busy; exit
               end if
            end do
         else
            a = v(1); fa=v(2); b=v(3); fb=v(4); tolerance=v(5)
            call push ( stack, a, fa, b, fb, tolerance )
         end if
         cycle
      case default
     end select
   end do
   do iproc=1,numprocs-1; call MPI_Send(v,0,MPI_DOUBLE_..,iproc,readytag,...); end
    do
   call MPI_Reduce(private_result,integral_result,1,MPI_DOUBLE...,MPI_SUM,root,...)
end function integral
```

Preparing a Hybrid Verson
Stack Mechanism - OpenMP Version (1of 2)

```
   call new_stack ( stack )
   call push ( stack, ah, f(ah), bh, f(bh), tolh )
   integral_result = 0.0
!$omp parallel private(a,b,tolerance,h,mid,one_trapezoid_area, two_trapezoid_area,
   ready)
   ready = .false.
   do
!$omp critical (stack)
 !     if ( empty_stack ( stack ) ) then;  ready = .t
       else; call pop ( stack, a, fa, b, fb, toleranc
       end if
!$omp end critical (stack)
       if ( ready ) exit
       h = b - a
       mid = (a + b) /2; fmid = f(mid)
       one_trapezoid_area = h * (fa + fb) / 2.0
       two_trapezoid_area = h/2 * (fa + fmid) / 2.0 + h/2 * (fmid + fb) / 2.0
       if (abs(one_trapezoid_area - two_trapezoid_area) < 3.0 * tolerance) then
!$omp critical (result)
           integral_result = integral_result + two_trapezoid_area
!$omp end critical (result)
       else
!$omp critical (stack)
           call push ( stack, a, fa, mid, fmid, tolerance / 2 )
           call push ( stack, mid, fmid, b, fb, tolerance / 2 )
!$omp end critical (stack)
       end if
   end do
!$omp end parallel
```

The naive approach:
„If stack empty, exit“
does not work, because all
but one threads will stop
immideately

Preparing a Hybrid Verson

Stack Mechanism - OpenMP Version (2 of 2)

```
..
   busy =
!$omp para
   two_trapezoid_area,idle,ready)
   ready = .false.
   idle = .true.
   do
!$omp critical (stack)
        if ( empty_stack ( stack ) ) then
          if ( .not. idle ) then
            idle=.true.; busy = busy - 1
          end if
          if ( busy .eq. 0 ) ready = .true.
        else
           call pop ( stack, a, b, tolerance )
           if ( idle ) then
             idle = .false.; busy = busy + 1
           end if
        end if
!$omp end critical (stack)
        if ( idle ) then
          if ( ready ) exit
          cycle ! try again (delay?)
        end if
         ......
!$omp end parallel end function integral
```

Stop if stack is empty and no thread is busy any more.

Hybrid Version
Master executes OpenMP Stack Mechanism

```
!$omp parallel private(a,b,tolerance,h,mid,one_trapezoid_area, two_trapezoid_area,
   ready)
!$omp&          private(myworker)
   ready = .false.
   myworker = omp_get_thread_num() + 1
   do
!$omp critical (stack)
       ...
!$omp end critical (stack)
       if ( ready ) then
          call MPI_Send(mid,0,MPI_DOUBLE_PRECISION,myworker,readytag,..);  exit
       end if
       h = b - a;  mid = (a + b) /2  ! Worker evaluates: fmid = f(mid)
       call MPI_Send(mid,1,MPI_DOUBLE_PRECISION,myworker,argtag,..)
       call MPI_Recv(fmid,1,MPI_DOUBLE_PRECISION,myworker,functag,MPI_COMM_WORLD,.)
       one_trapezoid_area = h * (fa + fb) / 2.0
       two_trapezoid_area = h/2 * (fa + fmid) / 2.0 + h/2 * (fmid + fb) / 2.0
       if (abs(one_trapezoid_area - two_trapezoid_area) < 3.0 * tolerance) then
!$omp critical (result)
          integral_result = integral_result + two_trapezoid_area
!$omp end critical (result)
       else
!$omp critical (stack)
          call push ( stack, a, fa, mid, fmid, tolerance / 2 )
          call push ( stack, mid, fmid, b, fb, tolerance / 2 )
!$omp end critical (stack)
       end if
   end do
!$omp end parallel
```

Summary

- Adaptive integration is a nice toy problem which is handy to experiment with differnet parallelization approaches.
- Any bisectioning algorithm can employ similar strategies.
- There are many more interesting strategies – like distributed queues, with workstealing algorithms …

#Threads/Procs	1	2	4	8	12	16	20	24
SER_REC	20.3002	-	-	-	-	-	-	-
SER_STACK	20.3020	-	-	-	-	-	-	-
OMP_STACK	20.3026	10.3050	5.4054	3.0073	2.2742	1.8418	1.6429	1.6102
OMP_TASKQ	20.3051	10.3057	5.4060	3.1075	2.4097	1.9188	1.7179	1.6207
OMP_NESTED	20.3012	13.6030	6.9043	3.8045	2.7041	2.4056	2.0044	1.8045
MPI_FARM	-	20.3082	7.1046	3.4040	2.4045	1.9044	1.8044	1.5054
HYB_FARM	-	20.3072	10.310	3.8487	2.7140	2.1853	2.0452	1.8743
MPI_RMA	21.0296	11.5552	8.3343	16.251	271.03	-	-	-
HYB_TASKQ	-	20.3103	7.1090	3.4092	2.4108	2.2893	2.4326	2.4867
HYB_NESTED	-	21.9001	8.5151	4.3135	3.3308	2.8120	2.5487	2.4026

RWTH AACHEN UNIVERSITY Center for Computing and Communication