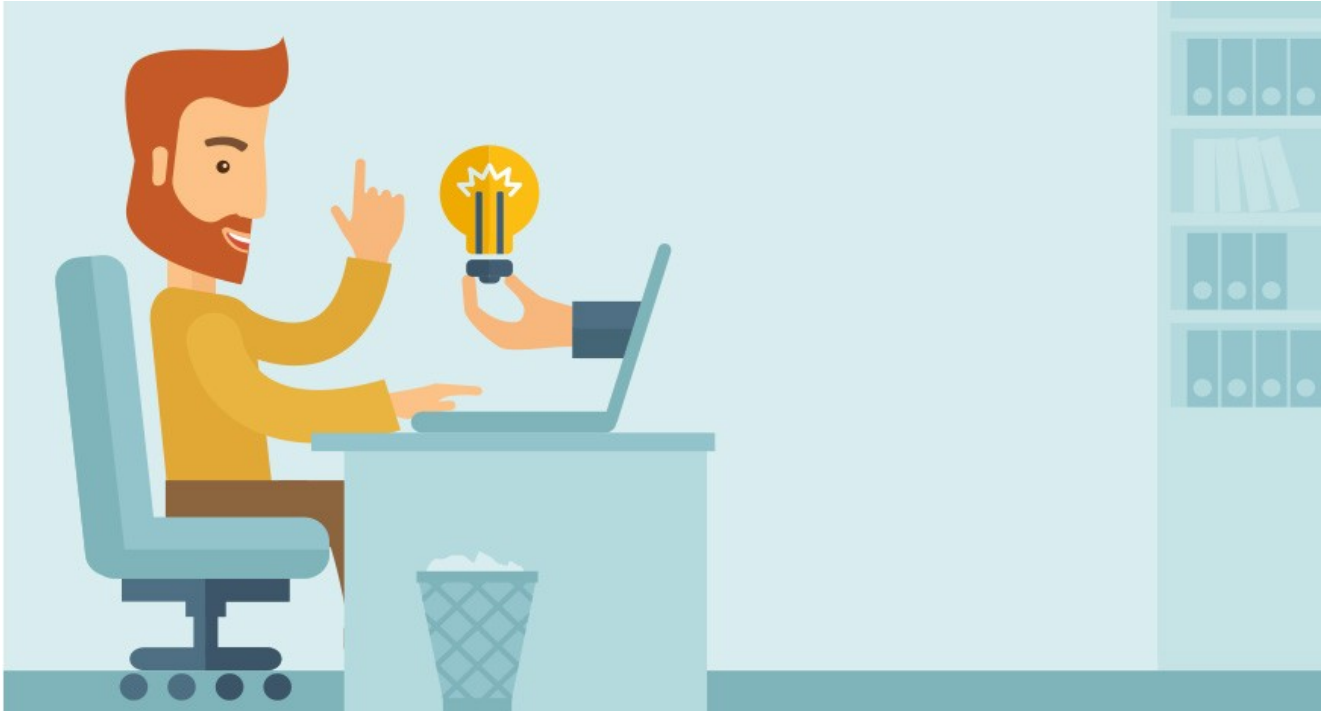
Software Tutorial

Simon Scheidegger

ZICE 17, University of Zürich, Jan 30th



Outline

1. OpenMP exercises

2. MPI exercises

3. Sparse Grid Software

→ All guided exercises, so ...

1. OpenMP exercises

Exercise 1:”hello world from thread”

1. go to /zice17/scheidegger/intro_to_hpc/openmp:

> cd /zice17/scheidegger/intro_to_hpc/openmp

2. Have a look at the code

> vi 1.hello_world.f90 / vi 1a.hello_world.cpp

3. compile by typing:

> make

4. Experiment with different numbers of threads

> export OMP_NUM_THREADS=1 (play around with #threads)
> ./hello_world.exec (FORTRAN)
> ./1a.hello_world.exec (CPP)

Exercise 1:”hello world from thread”

5. experiment with **slurm** (the settings)
   → submit jobs with various #threads
   → change name of output file

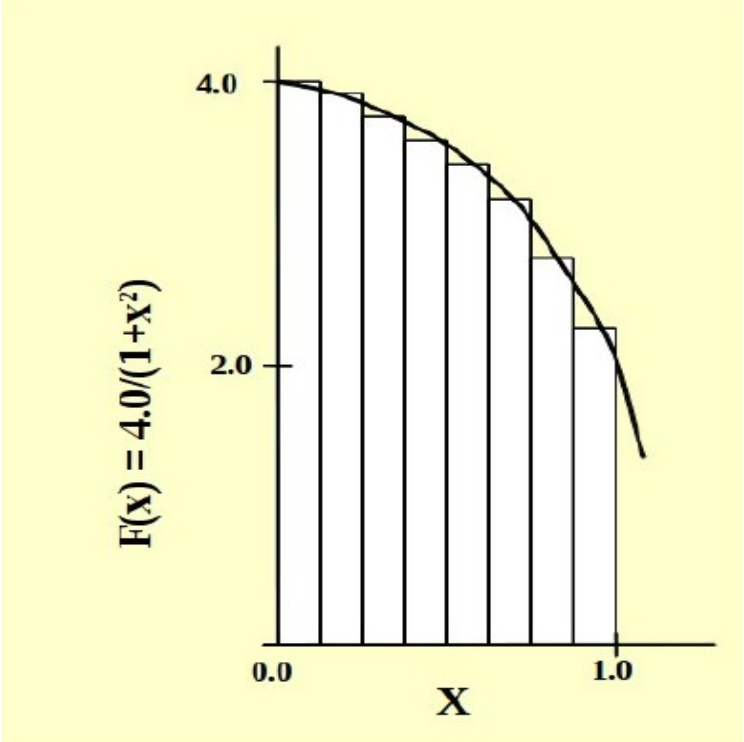
> cd /zice17/scheidegger/intro_to_hpc/openmp
> vi submit_openmp.sh

```
#!/bin/bash -l

#SBATCH --ntasks=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=8

#SBATCH --time=00:01:00


#SBATCH --job-name=test_submission
#SBATCH --output=openmp_test.out
#SBATCH --error=openmp_test.err

export OMP_NUM_THREADS=8


### openmp executable
./1.hello_world.exec
```

6. see **1.hello_world.f90** line 25: !$omp parallel private(tid)
   → what happens if we remove **private(tid)** → try out!

Exercise 2: Computing Pi

Fig. from O. Schenk



$$\int_0^1 \frac{4.0}{(1+x^2)}\,dx = \pi$$

$$\sum_{i=0}^{N} F(x_i)\Delta x \approx \pi$$

Exercise 2: Computing Pi (II)

>/zice17/scheidegger/intro_to_hpc/openmp/4a.integration_pi_reduction.cpp
>/zice17/scheidegger/intro_to_hpc/openmp/4b.integration_pi_reduction.f90

```
!=====================================================================
!
!     OpenMP example: program to approximation pi by a REDUCTION
!
!=====================================================================
      program integration_pi_reduction

      use omp_lib

      implicit none

      real(8) :: pi,w,x, time
      integer(8) :: i, N = 500000000

!---------------------------------------------------------------------
      pi = 0.d0
      w = 1.d0/N

!.....start timer
      time = -omp_get_wtime()

!---------------------------------------------------------------------
!$OMP PARALLEL PRIVATE(x)
!$OMP DO REDUCTION(+:pi)
      do i = 1 , n
        x = w*(i-0.5d0)
        pi = pi + 4.d0/(1.d0+x*x)
      enddo
!$OMP END DO
!$OMP END PARALLEL

!.....end timer
      time = time + omp_get_wtime()

!---------------------------------------------------------------------
      write(*,*) 'The approximation of Pi =', pi*w
      write(*,*) 'took ', time, ' seconds'

!---------------------------------------------------------------------
      end program integration_pi_reduction
!---------------------------------------------------------------------
```

1. Play with the number of threads and check the running times.
2. Play with the size of N=100, 200, 1000,... and the number of threads
3. What do you observe?

2. MPI exercises

Exercise 1: Ping - Pong

1. go to scheidegger/intro_to_hpc/MPI:

> cd scheidegger/intro_to_hpc/MPI

2. Have a look at the code

>vi 2.ping_poing.f90

3. compile by typing:

> make

4. run the executable by typing

>mpiexec -np 2 ./2.ping_pong.exec

5. run the executable with 3 processes.

   What happens?
   Why?

Exercise 1: Ping – Pong (II)

```
      integer, parameter :: process_a = 0
      integer, parameter :: process_b = 1

      integer, parameter :: ping = 17 !message tag
      integer, parameter :: pong = 23 !message tag

      integer, parameter :: length = 1
      integer :: status(MPI_STATUS_SIZE)
      real :: buffer(length)
      integer :: i

      integer :: ierror, my_rank, size

!-----------------------------------------------------------------------
      call MPI_INIT(ierror)

      call MPI_COMM_RANK(MPI_COMM_WORLD, my_rank, ierror)
      call MPI_Comm_size(MPI_COMM_WORLD, size, ierror)


!.....the example should only be run with 2 procs, else abort
      if (size .ne. 2) then
        write(*,*) 'please run this with 2 processors'
        call MPI_Finalize(ierror)
        stop
      end if

!-----------------------------------------------------------------------
!.....write a loop of number_of_messages iterations. Within the loop, process A sends a message
!.....(ping) to process B. After receiving the message, process B sends a message (pong) to process A)
      if (my_rank .eq. process_a) then
        call MPI_SEND(buffer, length, MPI_REAL, process_b, PING, MPI_COMM_WORLD, ierror)
        call MPI_RECV(buffer, length, MPI_REAL, process_b, PONG, MPI_COMM_WORLD, status, ierror)
      else if (my_rank .eq. process_b) then
        call MPI_RECV(buffer, length, MPI_REAL, process_a, PING, MPI_COMM_WORLD, status, ierror)
        call MPI_SEND(buffer, length, MPI_REAL, process_a, PONG, MPI_COMM_WORLD, ierror)
      end if

      write(*,*) 'Ping-pong on process complete - no deadlock on process', my_rank

      call MPI_FINALIZE(ierror)
!-----------------------------------------------------------------------
      end program ping_pong
!-----------------------------------------------------------------------
```

Exercise 2: Reduction

```
      program reduce

      use mpi

      implicit none

      integer :: rank, input, result, ierror

!-----------------------------------------------------------------------
      call MPI_Init(ierror)
      call MPI_Comm_rank(MPI_COMM_WORLD, rank, ierror)

!-----------------------------------------------------------------------
      input = rank + 1

!-----------------------------------------------------------------------
!.....reduce the values of the different ranks in input to result of rank 0
!     with the operation sum (max, logical and)
      call MPI_Reduce(input, result, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD, ierror)

      if (rank .eq. 0) then
        write (*,*) 'result', result
      end if

      call MPI_Finalize(ierror)
!-----------------------------------------------------------------------
      end program reduce
!-----------------------------------------------------------------------
```

Exercise 2: Reduction (II)

1. go to zice17/scheidegger/intro_to_hpc/MPI:

> cd zice17/scheidegger/intro_to_hpc/MPI:

2. Have a look at the code

>vi 3.MPI_reduce.f90

3. compile by typing:

> make

4. run the code

>mpiexec -np 2 ./3.MPI_reduce.exec (experimpent with # processes)

5. change the “root” (line 27) from 0 to 1. What happens?

Exercise 3: MPI & Python

Go to **zice17/scheidegger/intro_to_hpc/alphacruncher/MPI4PY/alphacruncher**

Run with

> mpiexec -np 4 python helloworld.py

Make MPI available

```
#hello.py
from mpi4py import MPI
comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = MPI.COMM_WORLD.Get_size()
print "hello world from process ", rank, " from total ", size , "processes"
```

Exercise 4: MPI Broadcast in Python

> **zice17/scheidegger/intro_to_hpc/MPI4PY/alphacruncher/bcast.py**

```
import numpy
from mpi4py import MPI
comm = MPI.COMM_WORLD
rank = comm.Get_rank()

#intialize
rand_num = numpy.zeros(1)

if rank == 0:
        rand_num[0] = numpy.random.uniform(0)

comm.Bcast(rand_num, root = 0)
print "Process", rank, "has the number", rand_num
```

1. Run with

> mpiexec -np 4 python bcast.py

2. What happens if you change root = 0 to root = 2 ?

3. What would you need to do to fix the bug from (2) ?

TASMANIAN – open source ASG

Tasmanian

tasmanian.ornl.gov

Apps | Combining Pyth | nbviewer.ipytho | simplex | CourseWork : F1 | Investment Man | Optimal Moneta | Wealth Managen | errors still prett

TASMANIAN

Toolkit for Adaptive Stochastic Modeling and Non-Intrusive ApproximatioN
ORNL Laboratory Directed Research and Development
DoE: Office for Advanced Scientific Computing Research

Home | About Tasmanian | Development Team | Downloads | Manuals | Contact Us

Development Team

Miroslav Stoyanov
Lead Developer
Developer of the Sparse Grids Module
Full Time Staff
Department of Applied Mathematics
Oak Ridge National Laboratory

Guannan Zhang
Developer
Developer of the Dream Module
Full Time Staff
Department of Applied Mathematics
Oak Ridge National Laboratory

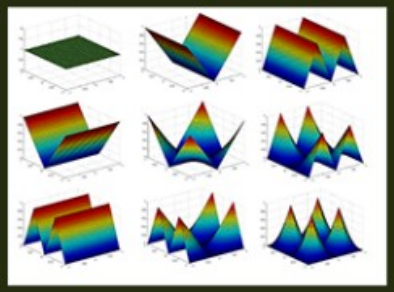
John Burkardt
Developer
Developer of the Dream Module
Researcher
Department of Scientific Computing
Florida State University

Clayton Webster
Principle Investogator
Full Time Staff
Department of Applied Mathematics
Oak Ridge National Laboratory

Drayton Munster

ABOUT Tasmanian

The Toolkit for Adaptive Stochastic Modeling and Non-Intrusive ApproximatioN is a collection of robust libraries for high dimensional integration and interpolation as well as parameter calibration. The code consists of several modules that can be used individually or conjointly. The project is sponsored by Oak Ridge National Laboratory Directed Research and Development as well as the Department of Energy Office for Advanced Scientific Computing Research.



Sparse Grids

Sparse Grids is a family of algorithms for constructing multidimensional quadrature and interpolation rules from tensor products of one dimensional such rules. Tasmanian Sparse Grids Module implements a wide variety of one dimensional rules based on global and local function basis. Refer to the Manual for a complete list of the capabilities.

Software tutorial in the afternoon!!

The Toolkit for Adaptive Stochastic Modeling and Non-Intrusive ApproximatioN

http://tasmanian.ornl.gov/

TASMANIAN Sparse Grids v3.1 (February 2016).

Very recent open source library written in CPP:

→ Contains “ordinary and adaptive” sparse grids.

→ Many more basis functions (global polynomials, wavelets,…).

→ Interfaces to Python and Matlab.

→ Moderately parallelized (with OpenMP).

Compile & run TASMANIAN*

!!! READ THE * MANUAL (RTFM) – p.50ff !!!**

1. go to TASMANIAN:
> cd zice17/scheidegger/sparse_grids/TasmanianSparseGrids

2. compile:
> make

3. go to simple example:
> cd zice17/scheidegger/sparse_grids/TasmanianSparseGrids/ZICE17_Matlab

4. let's have a look at the example:
> tsg_example_ZICE17.m

5. launch matlab & run example:
> matlab -nojvm (**no gui**)
> tsg_example_ZICE17()

6. NOTE: Tasmanian [-1,1]^d instead of [0,1]^d

If you are interested in CPP code examples, TASMANIAN provides examples here:
> cd Tzice17/scheidegger/sparse_grids/TasmanianSparseGrids/Example/example.cpp
> make

*more examples provided from TASMANIAN in InterfaceMATLAB/tsgExample.m → beyond scope of lecture.

Exercises

Create sparse grids based on different analytical test functions, e.g. Genz (1984).

→ different test functions can be obtained by varying $c = (c_1, \ldots, c_d)$ (c> 0) and $w = (w_1, \ldots, w_d)$

→ difficulty of functions is monotonically increasing with c.

→ randomly generate 1,000 test points and compute error(s): $e = \max_{i=1,\ldots,1000} |f(\vec{x_i}) - u(\vec{x_i})|.$

→ **play with adaptive/non-adaptive sparse grids/refinement level and criterion.**

→ generate convergence plots (number of points versus error – as done above).

Genz (1984) test functions

1. OSCILLATORY: $f_1(x) = \cos\left(2\pi w_1 + \sum_{i=1}^{d} c_i x_i\right),$
2. PRODUCT PEAK: $f_2(x) = \prod_{i=1}^{d} \left(c_i^{-2} + (x_i - w_i)^2\right)^{-1},$
3. CORNER PEAK: $f_3(x) = \left(1 + \sum_{i=1}^{d} c_i x_i\right)^{-(d+1)},$
4. GAUSSIAN: $f_4(x) = \exp\left(-\sum_{i=1}^{d} c_i^2 t (x_i - w_i)^2\right),$
5. CONTINUOUS: $f_5(x) = \exp\left(-\sum_{i=1}^{d} c_i |x_i - w_i|\right),$
6. DISCONTINUOUS: $f_6(x) = \begin{cases} 0, & \text{if } x_1 > w_1 \text{ or } x_2 > w_2, \\ \exp\left(\sum_{i=1}^{d} c_i x_i\right), & \text{otherwise.} \end{cases}$