CS601: Software Development for Scientific Computing

Autumn 2021

Week7:

- Unstructured Grids, Tools (GDB)

Last Week..

- Elliptic PDEs - theory and program representation
  - 4-point stencil and system of equations
  - Solving system of equations – Jacobi and Gauss-Seidel iteration
  - Program representation: classes Domain, GridFn, and Solution

GDB

- GNU Debugger – A tool for inspecting your C/C++ programs
  - How to begin inspecting a program using gdb?
  - How to control the execution?
  - How to display, interpret, and alter memory contents of a program using gdb?
  - Misc – displaying stack frames, visualizing assembler code.

GDB

– Compile your programs with –g option

```
hegden$gcc gdbdemo.c -o gdbdemo -g
hegden$
```

```
1 #include<stdio.h>
2 int foo(int a, int b)
3 {
4     int x = a + 1;
5     int y = b + 2;
6     int sum = x + y;
7
8     return x * y + sum;
9 }
10
11 int main()
12 {
13    int ret = foo(10, 20);
14    printf("value returned from foo: %d\n",ret);
15    return 0;
16 }
```

GDB – Start Debug

- Start debug mode (`gdb gdbdemo`)
  - Note the executable on first line (not .c files)
  - Note the last line before `(gdb)` prompt:
    - if –g option is not used while compiling, you will see "`(no debugging symbols found)`"

```
[ecegrid-thin4:~/ECE264] hegden$gdb gdbdemo
GNU gdb (GDB) Red Hat Enterprise Linux (7.2-92.el6)
Copyright (C) 2010 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/min/a/hegden/ECE264/gdbdemo...done.
(gdb)
```

GDB – Set breakpoints

- Set breakpoints (b)
  - At line 14
  - Beginning of foo

```
1 #include<stdio.h>
2 int foo(int a, int b)
3 {
4     int x = a + 1;
5     int y = b + 2;
6     int sum = x + y;
7
8     return x * y + sum;
9 }
10
11 int main()
12 {
13    int ret = foo(10, 20);
14    printf("value returned from foo: %d\n",ret);
15    return 0;
16 }
```

```
(gdb) b gdbdemo.c:14
Breakpoint 1 at 0x400512: file gdbdemo.c, line 14.
(gdb) b foo
Breakpoint 2 at 0x4004ce: file gdbdemo.c, line 4.
(gdb)
```

GDB – Start execution

- Start execution (`r <command-line arguments>`)
  - Execution stops at the first breakpoint encountered

```
(gdb) r
Starting program: /home/min/a/hegden/ECE264/gdbdemo

Breakpoint 3, main () at gdbdemo.c:13
13          int ret = foo(10, 20);
```

  - Continue execution (`c`)

```
(gdb) c
Continuing.

Program exited normally.
```

GDB – Printing

– Printing variable values (`p <variable_name>`)

```
Breakpoint 2, foo (a=10, b=20) at gdbdemo.c:4
4           int x = a + 1;
(gdb) n
5           int y = b + 2;
(gdb) p x
$3 = 11
```

– Printing addresses (`p &<variable_name>`)

```
(gdb) p &x
$5 = (int *) 0x7fffffffc4f4
```

GDB – Manage breakpoints

- Display all breakpoints set (`info b`)

```
(gdb) info b
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x0000000000400512 in main at gdbdemo.c:14
2       breakpoint     keep y   0x00000000004004ce in foo at gdbdemo.c:4
(gdb)
```

- Delete a breakpoint (`d <breakpoint num>`)

```
(gdb) d 1
(gdb) info b
Num     Type           Disp Enb Address            What
2       breakpoint     keep y   0x00000000004004ce in foo at gdbdemo.c:4
(gdb)
```

- Disable a breakpoint (`disable <breakpoint num>`)

```
(gdb) disable 2
(gdb) info b
Num     Type           Disp Enb Address            What
2       breakpoint     keep n   0x00000000004004ce in foo at gdbdemo.c:4
(gdb)
```

- Enable breakpoint (`enable <breakpoint num>`)

```
(gdb) enable 2
(gdb) info b
Num     Type           Disp Enb Address            What
2       breakpoint     keep y   0x00000000004004ce in foo at gdbdemo.c:4
```

GDB – Step in

– Steps inside a function call (s)

```
Breakpoint 3, main () at gdbdemo.c:13
13          int ret = foo(10, 20);
(gdb) s
foo (a=10, b=20) at gdbdemo.c:4
4       _     int x = a + 1;
```

GDB – Step out

– Jump to return address (`finish`)

```
(gdb) finish
Run till exit from #0  foo (a=10, b=20) at gdbdemo.c:4
0x000000000040050f in main () at gdbdemo.c:13
13          int ret = foo(10, 20);
Value returned is $2 = 275
```

GDB – Memory dump

– Printing memory content (`x/nfu <address>`)

- n = repetition (number of bytes to display)
- f = format (‘x’ – hexadecimal, ‘d’-decimal, etc.)
- u = unit (‘b’ – byte, ‘h’ – halfword/2 bytes, ‘w’ – word/4 bytes, ‘g’ – giga word/8 bytes)
- `E.g. x/16xb 0x7fffffffc500` (display the values of 16 bytes stored from starting address

```
(gdb) x/16xb 0x7fffffffc500
0x7fffffffc500: 0x20    0xc5    0xff    0xff    0xff    0x7f    0x00    0x00
0x7fffffffc508: 0x0f    0x05    0x40    0x00    0x00    0x00    0x00    0x00
```

GDB – Printing addresses

- `Registers ($rsp, $rbp)`
  - Note that we use the ‘x’ command and not the ‘p’ command.

```
(gdb) x $rsp
0x7fffffffc500: 0x20
(gdb) x $rbp
0x7fffffffc500: 0x20
```

GDB – Altering memory content

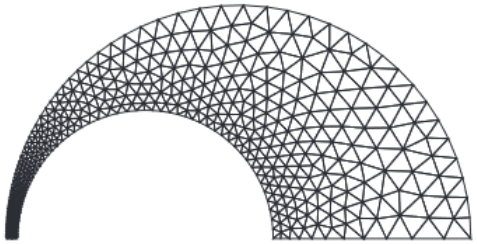
- Set command (`set variable <name> = value`)

```
(gdb) n
6           int sum = x + y;
(gdb) p x
$7 = 11
(gdb) p y
$8 = 22
(gdb) set variable y = 0
(gdb) n
8           return x * y + sum;
(gdb) p sum
$9 = 11
```

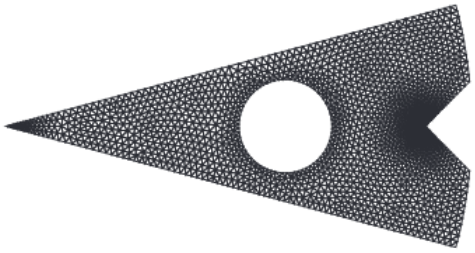
- Set command (`set *(<type *>addr) = value`)

Unstructured Grids

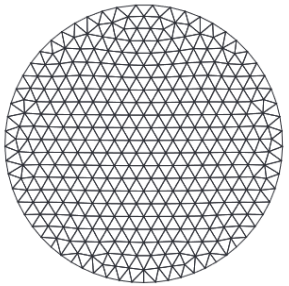
- Motivation:
  - E.g. reduce the noise because of air flowing through a duct, Maintain uniformity in flow (no pressure drop)



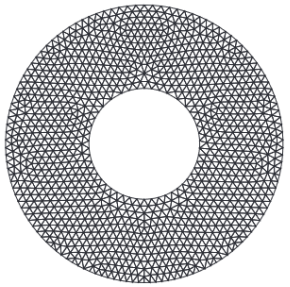
(e) Geometric Adaptivity



(f) Pie with hole



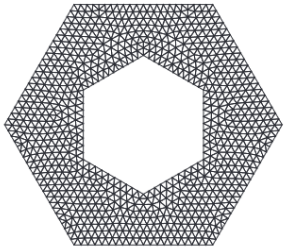
(a) Unit circle



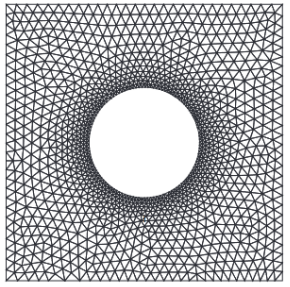
(b) Unit circle with hole

https://www.math.uci.edu/~chenlong/Papers/Chen.L%3BHolst.M2010.pdf

- Handle complex geometries
- Refine at region of interest



(c) Square with hole (uniform)



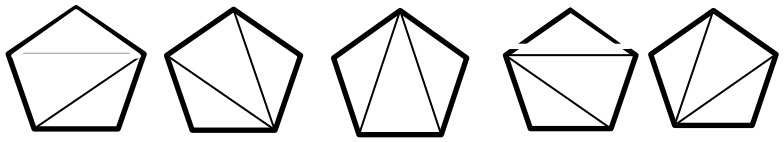
(d) Square with hole (refined at hole)

Unstructured Grids - Problem

- Discretize the domain into an optimal set of triangles (or tetrahedra / simplex)
- Areas: Mesh Generation and Mesh Optimization
  1. Fix the location of vertices and optimize
  2. Fix the vertices' connectivity and optimize (determine optimal placement of vertices)
  3. Iterate

(Optimize based on:

- Sum of edge weights
- Harmonic energy, Distortion energy)



Unstructured Grids – Program Representation (partial)

- Option 1:

```
Point points[n]; // represents coordinates
int triangles[m][3]; // represents triangles
```

- Option 2:

```
double x[n], y[n]; // represents coordinates (2D)
int triangles[m][3]; // represents triangles
```

- Option 3: $G(V, E)$

```
double x[n], y[n]; // represents vertices (2D)
int edges[m][2]; // represents edges
```

Unstructured Grids - Challenges

- Placing the points
  - What constitutes inside and what is outside the domain? *Place points only in the interior.*
  - What should be the distance between points?
- Connecting the points
  - What is the best way to create tiles once the points are placed?

Unstructured Grids - Approaches

- Delaunay Triangulation is the most commonly used unstructured triangulation method
  - Advantage: can automatically give a ‘better’ triangulation (e.g. w.r.t aspect ratio)
  - Disadvantage: suitable for convex domain
- Advancing Front Method is another method
  - Advantage: suitable for concave domain
  - Disadvantage: No prioritization of triangulation