CSE 6230: A tutorial on C and using the PACE cluster

Kaan Sancak, Abhishek Kalokhe

How to use the PACE Cluster

Accessing the cluster

Note: You need to be connected to Georgia Tech’s VPN (GlobalProtect Client).

1. Web-based User Interface - Open OnDemand

    Access: https://ondemand-ice.pace.gatech.edu (Login to GT SSO)

    - **Accessing the login node (Head Nodes)**

      Select **>_ICE Shell Access** from Clusters option in the navigation bar.

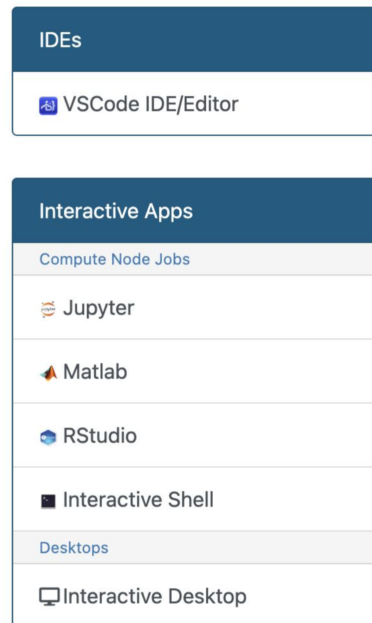
    - **Accessing the compute nodes**

      Choose **Interactive Shell** from the Interactive Apps option in the navigation bar.

      Choose the configuration of the required compute nodes.

      Click on **Launch** and wait. You will receive an email when the resources are allocated.

      You can find the session under **My Interactive Sessions** and then click on **Connect**.



Accessing the cluster

Note: You need to be connected to Georgia Tech’s VPN (GlobalProtect Client).

2. Command Line

- You will need an SSH Client (a.k.a. terminal).
  - Windows: Powershell (built-in on Windows 10) or Windows Subsystem for Linux (WSL)
  - MacOS: Terminal (built-in)
  - Linux: System-default terminal (gnome/KDE)

- SSH access to PACE clusters:

```
ssh <GT_username>@login-ice.pace.gatech.edu
```

This will give you the access to the login node.

Requesting compute nodes using the command line

1. Interactive Job

    Use the **salloc** command to request an interactive job.
    Please check `salloc --help` for more info.

    Example: `salloc -N1 --ntasks-per-node=4 -t1:00:00`

2. Batch Job

    You will need a bash script which contains:

    - Required configuration for the compute nodes.
    - Commands to run your program on the nodes once allocated.

Sample Bash Script

job_script.batch

```
#!/bin/bash

#SBATCH --job-name=testjob
#SBATCH -N2 --ntasks-per-node=2
#SBATCH --mem-per-cpu=6G
#SBATCH -t3:00:00
#SBATCH --output=testjob.%j.out
#SBATCH --mail-type=BEGIN,END,FAIL
#SBATCH --mail-user=gburdell3@gatech.edu

#### Load the modules required ####
module load valgrind

#### Code Execution ####

echo "Hello"
sleep 50
echo "Bye"
```

Shell

A name for this job, can be anything
2 nodes, 2 cores in each
6GB memory per core (24GB total)
3 hours max, after which job is stopped!
name output file (includes STDOUT and STDERR)
when to receive email notifications
email address for notifications

Submitting a job:

```
sbatch job_script.batch
```

Available Resources (ICE Nodes)

- 60 Intel CPU nodes: 24 processors
- 4 AMD CPU nodes: 64 processors
- 98 GPUs
  - 19 GPU nodes with Nvidia Tesla V100 (1-4 GPUs/node)
  - 10 GPU nodes with Nvidia RTX6000 (4 GPUs/node)
  - 4 GPU nodes with Nvidia A100 (2 GPUs/node)
  - 2 GPU nodes with Nvidia A40 (2 GPUs/node)
  - 2 GPU nodes with AMD MI210 (2 GPUs/node)

Slurm Info Commands

To check job status	squeue –u <GT-username>
To cancel a job	scancel <job id>
Info on completed jobs	sacct –j <job id>
Review completed jobs	pace-job-summary <job-id>
Additional Commands	
View partition and node info	sinfo
Run a parallel job on the cluster	srun

Using the Debugger (GDB)

What is GDB?

- A debugger for several languages, including C and C++.
- It allows you to inspect what the program is doing at certain point during execution.
- Errors like segmentation faults may be easier to find using gdb.
- See https://sourceware.org/gdb/current/onlinedocs/gdb for details.

Compiling for GDB

- Compile your program with -g option to enable debugging support.

Original: gcc [options] <source files> -o <output file>

Debugging enabled: gcc [options] -g <source files> -o <output file>

Example: gcc -g test.c -o test.out

```
1  #include <stdio.h>
2
3  int main() {
4    int x;
5    int a = x;
6    int b = x;
7    int c = a + b;
8    printf("%d\n", c);
9    return 0;
10 }
```

GDB Basics - Loading your file

- Run your executable with gdb: `gdb <filename>`
- **Example:** `gdb test.out`
  - You should get something like this:

```
GNU gdb (GDB) Red Hat Enterprise Linux 7.6.1-120.el7
Copyright (C) 2013 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 /storage/ice1/8/5/ksancak3/test...done.
(gdb)
```

GDB Basics - Help

- Use help command to learn more information at any point: `help [command]`

```
(gdb) help
List of classes of commands:

aliases -- Aliases of other commands
breakpoints -- Making program stop at certain points
data -- Examining data
files -- Specifying and examining files
internals -- Maintenance commands
obscure -- Obscure features
running -- Running the program
stack -- Examining the stack
status -- Status inquiries
support -- Support facilities
tracepoints -- Tracing of program execution without stopping the program
user-defined -- User-defined commands

Type "help" followed by a class name for a list of commands in that class.
Type "help all" for the list of all commands.
Type "help" followed by command name for full documentation.
Type "apropos word" to search for commands related to "word".
Command name abbreviations are allowed if unambiguous.
```

GDB Basics - Running your program

- After loading your program, just use: `run`

```
(gdb) run
Starting program: /storage/ice1/8/5/ksancak3/test
0
[Inferior 1 (process 126817) exited normally]
```

- If it has no serious problems, the program should run fine here too.
- If the program did have issues, then you should get some useful information like the line number where it crashed, and parameters to the function that caused the error.

GDB Basics - Some helpful commands

- `list` or `l`

```
(gdb) list
warning: Source file is more recent than executable.
1       #include <stdio.h>
2
3       int main() {
4         int x;
5         int a = x;
6         int b = x;
7         int c = a + b;
8         printf("%d\n", c);
9         return 0;
10      }
```

GDB Basics - Setting breakpoints

- Breakpoints can be used to stop the program run in the middle, at a designated point.
  - The simplest way is the command `break` or `b` – Sets a breakpoint at a specific line.
- **Usage:** `b <line number>`

```
(gdb) b 3
Breakpoint 1 at 0x40057a: file test.c, line 3.
(gdb) info b
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x000000000040057a in main at test.c:3
```

- `info b` summarizes the breakpoints
  - You can also use `disable/enable` to enable/disable all or specific breakpoint
- `run` the program again

```
Starting program: /storage/ice1/8/5/ksancak3/./test

Breakpoint 1, main () at test.c:5
5           int a = x;
```

GDB Basics - Printing variables

- `print` or `p`

```
(gdb) r
Starting program: /storage/ice1/8/5/ksancak3/./test

Breakpoint 1, main () at test.c:5
5           int a = x;
Missing separate debuginfos, use: debuginfo-install glibc-2.17-326.el7_9.x86_64
(gdb) print a
$1 = 0
(gdb) p a
$2 = 0
```

GDB Basics - Printing variables

- You can also put a breakpoint to specific function: break <function_name>
- continue to proceed to next breakpoint
- step to proceed to next line of code
  - The next command single-steps as well, except this one doesn’t execute each line of a sub-routine, it just treats it as one instruction.
- backtrace or bt to produce a stack trace of the function calls that lead to a segfault

GDB - Example 2

```
1 #include <stdio.h>
2
3 int main() {
4     int *ptr;
5     *ptr = 10;
6
7     return 0;
8 }
```

```
[ksancak3@atl1-1-02-003-19-1 scratch]$ ./segfault
Segmentation fault (core dumped)
```

GDB - Example 2

```
[ksancak3@atl1-1-02-003-19-1 scratch]$ gdb segfault
GNU gdb (GDB) Red Hat Enterprise Linux 7.6.1-120.el7
Copyright (C) 2013 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 /storage/ice1/8/5/ksancak3/segfault...done.
```

```
(gdb) run
Starting program: /storage/ice1/8/5/ksancak3/segfault

Program received signal SIGSEGV, Segmentation fault.
0x000000000040052a in main () at segfault.c:5
5           *ptr = 10;
Missing separate debuginfos, use: debuginfo-install glibc-2.17-326.el7_9.x86_64
```

GDB - Example 2

- `run` and `backtrace`

```
(gdb) run
Starting program: /storage/ice1/8/5/ksancak3/segfault

Program received signal SIGSEGV, Segmentation fault.
0x000000000040052a in main () at segfault.c:5
5           *ptr = 10;
Missing separate debuginfos, use: debuginfo-install glibc-2.17-326.el7_9.x86_64
(gdb) bt
#0  0x000000000040052a in main () at segfault.c:5
```

- `Segfault at line 5`

GDB - Example 2

- `print` value of ptr

```
(gdb) p ptr
$1 = (int *) 0x0
```

- Suggest that the value of ptr is not initialized properly.

```
5     int *ptr = (int*)malloc(sizeof(int));
6
7     if (ptr != NULL) {
8         *ptr = 10;
9
10        free(ptr);
11    } else {
12        printf("Memory allocation failed\n");
13    }
```

Introduction to C

Structure of a C program

It basically consists of:

- Preprocessor Commands
- Functions
- Variables
- Statements & Expressions
- Comments

A simple C program

```
#include<stdio.h>
int main()
{
    printf(“Hello, World! \n”);
    Return 0;
}
```

Compiling a C program

```
$ gcc hello.c
$ ./a.out
```

Hello, World!

Data types in C

- Basic Types
  - `int,`
  - `double,`
  - `float,`
  - `char, etc`
- Void type
- Derived Types
  - Pointer types
  - Array types
  - Structure types, etc

Operators

- Arithmetic Operators
  - `+` , `-` , `*` , `/` , `%` , `++` , `--`
- Relational Operators
  - `==` , `!=` , `>` , `<` , `>=` , `<=`
- Logical Operators
  - `&&` , `||` , `!`
- Assignment Operators
  - `=` , `+=` , `-=` , `*=` , `/=` , `%=` , etc…
- Misc. Operators
  - `sizeof` , `&` (Address) , `*` (pointer) , `?:` (Conditional Expression)

Declaring Variables

- You have to declare the data type of each variable explicitly.
- Basic data types:
    - `int`
    - `double`
    - `char`
    - Others,...
- Declaring and Initializing variables:
    - `int a;`
    - `int a, b;`
    - `int a = 1, b = 2;`
- Variable names are case sensitive
    - `int hat;`
    - `int Hat;`

Control Structures

- Conditional
  - `if`
  - `if-else`
  - `Switch`
- Iterative loops
  - `while`
  - `for`
- Nested Loops
- Loop control statements
  - `break;`
  - `continue;`

Functions

A function is a group of statements that together perform a task.

Defining a function:

```
return_type function_name( parameters )
{
     body of the function
}
```

Parts of a function:

1. Return Type - data type of the value the function returns.
2. Function Name - name of the function.
3. Parameters - information is passed to function as parameters.
4. Function Body - collection of statements which defines what the function does.

Scope

It is a region in the program where the specific variable(s) can be accessed, and beyond that it cannot be accessed.

- **Local Variables:** Variables that are declared inside a function or block are called local variables.

- **Global Variables:** Global variables are defined outside of a function, which can be accessed anywhere in the program.

Arrays

- C programming language provides a data structure called the array, which can store a fixed-size sequential collection of elements of the same type.
- All arrays consist of contiguous memory locations.
- Declaring Arrays:
  - `type arrayName [ arraySize ];`
  - `arraySize` must be defined as a positive integer constant which fixes the size of the array while declaring.
- Defining or accessing array data:
  - The indices of an array go from `0` to `arraySize-1`.
  - Particular data entries in the arrays can be defined or accessed using the respective indices. `arrayName[0] = 1` will make the first entry of the array equal to 1.

Pointers

- A pointer is a variable, whose value is the address of another variable, i.e., direct address of the memory location.
- Like any variable or constant, you must declare a pointer before you can use it to store any variable address.
    - `int *ip; /* pointer to an integer */`
    - `double *dp; /* pointer to a double */`
    - `float *fp; /* pointer to a float */`
    - `char *ch /* pointer to a character */`

- Difference between pointers of different data types is the data type of the variable or constant that the pointer points to.

Using the pointers

```
#include <stdio.h>
int main ()
{
     int var = 20;  /* actual variable declaration */
     int *ip;       /* pointer variable declaration */
     ip = &var;     /* store address of var in pointer variable*/

     printf("Address of var variable: %x\n", &var );

     /* address stored in pointer variable */
     printf("Address stored in ip variable: %x\n", ip );

     /* access the value using the pointer */
     printf("Value of *ip variable: %d\n", *ip );

     return 0;
}
```

Output:

```
Address of var variable: bffd8b3c
Address stored in ip variable: bffd8b3c
Value of *ip variable: 20
```

Strings

- The string in C programming language is actually a one-dimensional array of characters which is terminated by a null character '\0'.

- Initialization:
    - `char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};`
    - `char greeting[] = "Hello";` Do not have to place \0 at the end. C compiler does it.

- Functions that manipulate null-terminated strings:
    - `strcpy s1, s2;` Copies string s2 into string s1.
    - `strcat s1, s2;` Concatenates string s2 onto the end of string s1.
    - `strlen s1;` Returns the length of string s1.
    - `strcmp s1, s2;` Returns 0 if s1 and s2 are the same; less than 0 if s1<s2; greater than 0 if s1>s2.
    - `strchr s1, ch;` Returns a pointer to the first occurrence of character ch in string s1.
    - `strstr s1, s2;` Returns a pointer to the first occurrence of string s2 in string s1.

Structures

- Structure is another user defined data type available in C programming, which allows you to combine data items of different kinds.
- Defining a Structure:

```
struct Books
{
char title[50];
char author[50];
char subject[100];
int book_id;
};
```

- Accessing the struct members:

```
struct Books Book1;

Book1.title
Book1.author
```

Memory Management

void *calloc `int num, int size;`

This function allocates an array of num elements each of whose size in bytes will be size.

void free `void *address;`

This function release a block of memory block specified by address.

void *malloc `int num;`

This function allocates an array of num bytes and leave them initialized.

void *realloc `void *address, int newsize;`

This function re-allocates memory extending it upto newsize.

Thank you!

References: https://www.tutorialspoint.com/cprogramming/pdf/c_quick_guide.pdf
https://www.cs.umd.edu/~srhuang/teaching/cmsc212/gdb-tutorial-handout.pdf