Preliminaries II

Dmitri Loguinov
Texas A&M University

January 21, 2016
Preliminaries: Agenda

- Pointers
- Homework setup
- Cave lights
- Cave search
- Pipes
**Pointers**

- **What is a C/C++ pointer?**
  - 4-byte number on x86, 8-byte on x64

```c
// example assumes x86
char str [] = "hello";
short *p = (short*) str;
printf ("%X %X %X\n", p, &p, *p);
printf ("%X %X %X\n", str, &str, *str);
char **p2 = (char**) &p;
printf ("%X %X %X %X\n", p2, &p2, *p2, **p2);
```

- **What is a static array?**
  - Immutable pointer hidden in compiler space
  - &arr same as arr (compiler hack)
Preliminaries: Agenda

- Pointers
- Homework setup
- Cave lights
- Cave search
- Pipes
Preliminaries: Homework Setup

• Implement four parallel search algorithms on a weighted graph under random edge-traversal delay

• Now the simple (but lengthy) explanation
  - Assume you have a space rover stuck in some cave on a remote planet
  - The cave is dark and its topology is unknown
  - As the rover is slow, it cannot directly search for the exit
Preliminaries: Homework Setup

• However, it has a number of flybots
  – These can travel all over the cave much quicker and search for the exit

• Main problem is flybots are dumb
  – Cannot remember which rooms they have been to
  – Cannot decide which next room to explore
  – Cannot talk to each other
• Your job is to write software that can control the flybots from the rover to find the exit in the shortest time

• Communication from your process goes through the Command Center (CC) block on the rover
  – Commands: MOVE to a given room R
  – Responses: list of R’s neighboring rooms
Preliminaries: Homework Setup

- Response delays are random
  - Based on distance traveled and power state of flybot antenna
  - Report will determine the average delay
- Target cave size 10M rooms
  - Single robot requires over 2 months
  - Obviously there is a need to massively parallelize the search
Preliminaries: Agenda

- Pointers
- Homework setup
- Cave lights
- Cave search
- Pipes
Preliminaries: Cave Lights

• So far, the problem is solvable by the most basic parallel BFS
  – Final element is to make the graph weighted

• Assume the cave is pitch black, except certain rooms where light penetrates from the outside
  – Presence of light could indicate there is an exit
  – Or there might be a ceiling hole through which the rover cannot escape

• Light propagation
  – Given a light source of intensity \( L \geq 1 \), all neighboring rooms get their light boosted by \( L/2 \), which repeats recursively
  – Exponential decay of light until it drops below 1 unit
Preliminaries: Cave Lights

- Note that some initially dark rooms may become brighter than some of the rooms with light sources.

Before: 0, 5, 20

After: 12.5, 5, 20
Preliminaries: Agenda

- Pointers
- Homework setup
- Cave lights
- Cave search
- Pipes
Preliminaries: Cave Search

• What would be a good search technique for this problem?
  - **Key observation:** light hopefully monotonically increases as we move towards the exit

• Assume we maintain two structures:
  - Set of unexplored nodes $U$
  - Set of discovered nodes $D$

• **Note:** each room in $D$ has been inserted into $U$, but not necessarily visited by a robot yet

• The main difference between the four studied algorithms is how to select the next node from $U$
Preliminaries: Cave Search

• BFS and DFS are classic, already covered in 221
• **Best First Search (bFS)**
  - Largest intensity of light among U
  - Finds sub-optimal paths when distracted by a very bright, but lengthy path
• **A** is a combination of BFS and bFS
  - Heuristically weighs both distance and amount of light
  - For each neighboring node i, compute its quality
    \[ Q_i = L_i + \frac{w}{(d_i + 1)} \]
    where \(L_i\) is amount of light in the room, \(d_i\) is its distance from the rover, and \(w\) is some weight
• What do we get with \(w = 0\) and \(w = \infty\)?
• How to implement bFS and A* efficiently?
Preliminaries: Agenda

- Pointers
- Homework setup
- Cave lights
- Cave search
- Pipes
Preliminaries: Pipes

- Pipes are communication channels between processes
  - Lossless
  - Implemented as FIFO queues through the kernel

- Anonymous pipes
  - Can be passed only to a child process (using pipe handle)
  - One-way, byte-based queue
  - Requires 2 pipes for duplex communication
  - Mostly used to redirect `stdin/stdout` of the child

- Named pipes
  - Globally unique names
  - Duplex (bi-directional)
  - Can be both byte-based and message-based
  - Homework uses the latter type
Preliminaries: Pipes

- Robot responses consist of a header, followed by an array of tuples (node, intensity)
  - Node is an 8-byte hash of a neighboring room
  - Intensity is a float value (amount of light)
Preliminaries: Pipes

- By default, CC pipes are blocking and synchronous
  - Only one message at a time can be in the pipe
  - However, its size is unknown a-priori

- **Idea**: receive as much of the message as buffer allows, then peek at the pipe, receive the rest
  - Here is pseudo-code (needs more work to be functional)

```c
#define BUF_SIZE 128   // small initial size to prevent over-allocation
char *buf = new char [BUF_SIZE];
ReadFile (pipe, buf, ..., &read, ...);
if (read == BUF_SIZE) // buffer filled to the max?
{
    PeekNamedPipe (pipe, ..., &remainder, ...);
    // need (BUF_SIZE + remainder) total space
    // allocate a new buffer, memcpy buf to it, delete the old one
    ReadFile (pipe, ...);
    buf = newbuf;
}
```
Preliminaries: Pipes

• Optimization
  - Per-message allocation/deletion of buf should be avoided
  - Retain newbuf until some future message overflows it
  - For monster caves, keep the buffer only if smaller than 5 KB

• Pipe names
  - Case insensitive:
  - Dot \cdot represents the same host

• Pipe names must be globally unique
  - If users run multiple copies of CC.exe on the same host, the pipe name must specify which of them to use
  - This homework uses \server\pipe\pipename, where X is the process ID of the CC in hex
Wrap-up

• Reminder: hw1-part1 is due in a week
  - Error checking for all function calls, proper disconnect
  - Wait for CC.exe to quit
  - Print initial room and all CC/robot text responses
  - Read robot response in one call to ReadFile

• See the sample grade sheet on page 15 of handout

• Example: allocate a buffer with 100 bytes and fill in three NodeTuple64 classes starting from byte 37

```c
char buf [100];
NodeTuple64 *nt = (NodeTuple64 *) (buf + 37);
for (int i = 0; i < 3; i++) {
    nt[i].node = i;
    nt[i].intensity = 1.0 / i;
}
```