Project tutoring

Alessandro Di Federico

ale@clearmind.me

Algorithms and Parallel Computing
Mathematical Engineering
Politecnico di Milano

December 17, 2015

Index

How to divide your program

- Identify the core components:
  - a set of classes/functions closely related
  - keep them together if they interact frequently
  - split them if they have a tiny interface
- Create a .cpp file and .h file for each component

What to put in .h files

- In general:

  .cpp files contain definitions
  .h files contain declarations

- Suppose a .cpp file needs a function defined elsewhere
- It must include a .h file containing its declaration

Example

```
// example.h
int factorial(int i);

// example.cpp
#include "example.h"

int factorial(int i) {
  if (i > 1)
    return i * factorial(i - 1);
  else
    return i;
}
```

Example

```
// user.cpp
#include "example.h"

int main() {
  factorial(4);
  return 0;
}
```

Static functions

- Suppose a function is used only in a single .cpp
- Define it there, no declaration in .h files
- Mark it `static`

Static function example

```
// user.cpp
#include "example.h"

int main() {
  factorial(4);
  return 0;
}
```

Inline functions

- In general .h files should only contain declarations
- Suppose you put a definition in an .h file
- Suppose the .h file is included by multiple .cpp files
- The compiler will find multiple definitions and will complain
- If you really want to put a definition in an .h file, use `inline`

Bad

```
// example.h
int meaning() { return 42; }

// example.cpp
#include "example.h"

int main() {
  meaning();
}
```

OK

```
// example.h
inline int meaning() { return 42; }

// example.cpp
#include "example.h"

int main() {
  meaning();
}
```

Classes

- If a class is used by a single .cpp file it can stay there
- If it’s used by multiple .cpp files put it in a .h file

Method definitions

- Two options:
  - define it inside the class (*inline* definition)
  - define it outside the class (*outline* definition)
- It's OK to put inline definitions in .h files
- They are implicitly inline functions

Method definition examples

```
// example.h
class MyClass {
  int methodInline() { return 42; }
  int methodOutline();
};

// example.cpp
#include "example.h"

int MyClass::methodOutline() {
  return 44;
}
```

And with CMake?

Just add to your executable all your .cpp files

```
add_executable(hello hello.cpp
                     example.cpp
                     car.cpp)
```

Index

Dense matrix

- What's the best way to represent a dense matrix?
- A sequential memory area of $M \times N$ elements

Size known statically

```
const int row_count = 1234;
const int col_count = 4321;

int denseMatrix[row_count][col_count];
```

Size not known statically

```
int *denseMatrix = new int[row_count * col_count];
// A std::vector is fine too

// To access element (x, y)
denseMatrix[x * col_count + y];
```

Dense matrix: pros & cons

- Suppose we have T non-zero elements:
  - Memory taken: $M \times N$ elements
  - Access time: constant
  - Enumeration time: $M \times N$

Sparse matrix

- Assume the full matrix won't fit in memory
- We can represent only the non-zero elements
- There are various options
- e.g. an unordered vector of $< x, y, value >$ triples

Dense matrix: pros & cons

- Suppose we have T non-zero elements:
  - Memory taken: T elements
  - Access time: T
  - Enumeration time: T

In general

- Think about your data structure in terms of:
  - memory taken
  - access time
  - enumeration time
- Evaluate it in terms of corner cases:
  - How does it perform with a very sparse matrix?
  - How does it perform with an average matrix?
  - How does it perform with a very dense matrix?
- Try to get good metrics in all cases

Don’t use both

- Using a sparse and dense representation is a bad idea
- You get all the pros and cons
  In particular you’re using a sparse representation but if the input is too large you’re program won’t work anyway
- So don’t use both (at the same time)
- You can choose at run-time which representation to use!

Allocations

- Allocations and memory releases take time
- How frequently do you allocate/release memory?
- Our problem is pretty static, after initialization:
  - grid size is constant
  - total amount of cars is constant

Index

Optimizations

- To test the performance of your code, enable optimizations!
- Pass the “-O2” flag to the compiler
- Or simply build in release mode with CMake[1]

[1] See CMake slides

Compiler warnings

- Compiler warnings might seem boring
- 99% of the times it’s a sign of something wrong
- Take the time to fix them all
- You can get additional warnings with the “-Wall” flag
- Don’t send me code which triggers warning

No magic numbers

- Don’t put wild numbers in your code
- The same number might have different meanings
- Always define a global constant

using

- `using` it's similar to `typedef`
- Useful to define shorter alias of an existing type

```
using IntVect = std::vector<int>;
IntVect MyVector;
```

using namespace

```
using namespace something;
```

- Makes `something::` implicit

```
using namespace std;
vector<int> MyVector;
cout << "Hi!" << endl;
```

- **Never** put `using namespace` in a .h file

Inheritance or templates?

- Should `BlueCar` inherit a base class `Car`?
- Should it specialize a template class `Car`?

Inheritance

```
class Car {
  virtual void advance() = 0;
  // ...
};

class BlueCar : Car {
  void advance() {
    // BlueCar-specific logic
  }
  // ...
};
```

Template specialization

```
const int BlueColor = 1;

template<int Color>
class Car {
  void advance();
  // ...
};

template<>
class Car<BlueColor> {
  void advance() {
    // BlueCar-specific logic
  }
  // ...
};
```

Use inheritance if...

you want to use `BlueCar` and `RedCar` interchangeably through their base class (`Car`)

```
int myFunction(Car *MyCar) {
  MyCar->advance();
}
```

Note: virtual methods incur in a small extra cost upon call.

Use templates if...

you always know if the object you're dealing with it's a BlueCar or RedCar

```
int myFunction() {
  std::vector<Car<BlueColor>> BlueCars;
  // ...
  for (Car<BlueColor>& MyCar : BlueCars) {
    // ...
  }

  std::vector<Car<RedColor>> RedCars;
  // ...
}
```

If your laptop sucks

- Some laptops don’t support VT-x
- So, the VM will see only a single core
- No real speedups from OpenMP/MPI!

Possible solutions

- Buy a decent laptop
- Install Linux side-by-side on your laptop
- Get in touch with the “Servizio prestito PC”[2]
- Ask us for access to a server

[2] Istituto per il diritto allo Studio Universitario (ISU) - Servizio Prestito Personal Computer, Centro Servizi Via Golgi, 20

License

This work is licensed under the Creative Commons Attribution-ShareAlike 3.0 Unported License. To view a copy of this license, visit http://creativecommons.org/licenses/by-sa/3.0/ or send a letter to Creative Commons, 444 Castro Street, Suite 900, Mountain View, California, 94041, USA.