Graph Coloring based Register Allocation

EE 677 – VLSI CAD

Course Project

Members –

Ritvik Mathur – 14D070018

Swapnil Bembde – 14D070034

Shantanu Choudhary – 14D070044

Under Guidance of –

Prof. Sachin Patkar

Electrical Engineering Department

Indian Institute of Technology, Bombay

Mumbai – 400076

Introduction

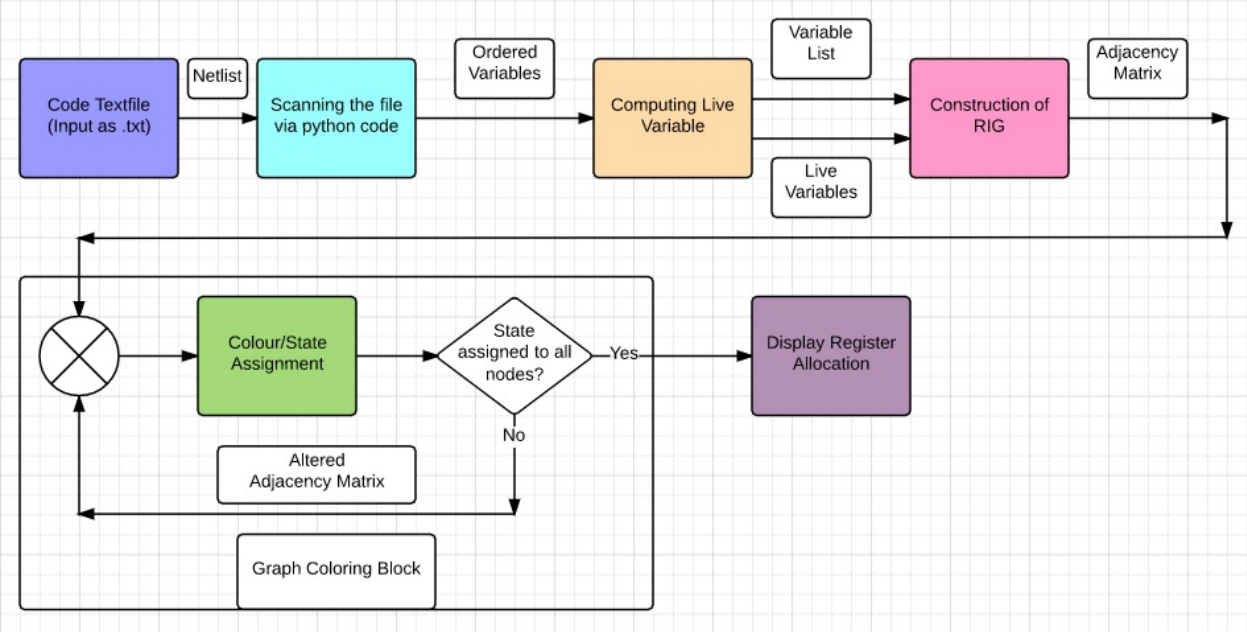
Programs are written as if there are only two kinds of memory: main memory and disk. Programmer is responsible for moving data from disk to memory (e.g., file I/O). Hardware is responsible for moving data between memory and caches. Compiler is responsible for moving data between memory and registers.

Cache and register sizes are growing slowly. Processor speed improves faster than memory speed and disk speed. So, the cost of a cache miss is growing. The widening gap is bridged with more caches. It is very important to manage registers and caches properly. Compilers are good at managing registers, and here we choose to implement a method to improve it further, to manage the registers more efficiently and quickly.

An intermediate code uses as many temporaries as necessary. This complicates final translation to assembly, but simplifies code generation and optimization. A typical intermediate code uses too many temporaries.

Register allocation problem requires us to rewrite the intermediate code to use fewer temporaries than there are machine registers. So, we need to assign multiple temporaries to a register. However, we need to ensure that the program behavior does not change.

Project Architecture



1. **Code Text file input**
   We are using a .txt format file as an input code. The language to be executed is written line by line in the text file, and consists of mainly assignment commands, with while loops, for loops, and if-else conditional assignments

2. **Scanning the file**
   We are using python as the programming language to perform the High-Level Synthesis. So, we scan the text file using python commands, and using string library commands to eliminate and analyze all assignments, we make a list of sets of variables for each line of the program, where the first element of each of the tuples correspond to the variable to which assignment is made, and rest are variables used in assignment. This ordered list of tuples for variables is given as the output.

3. **Computing Live Variable**
   We iteratively analyze the live variables for a particular line of the code. With respect to the current line, we check the variables in the ordered set for the line, and keep that variable live which is to be used in assignment in some of the further code, if there is no assignment made to this variable prior to this usage.

4. **Construction of RIG**
   Based on the set of tuples of live variables, we first compute the list of variables used in the netlist. Then we are constructing a graph, given as an Adjacency matrix, for

which edges are set between nodes i and j, if the variables corresponding to node i and j are present together in a tuple in live variable list.

5. **Graph Colouring Block**
   This block takes the Adjacency Matrix as an input. We have a set restriction for the number of registers, so we cannot assign more than that number of colours. We first initialize the colours of all the variables as 0. Then we start assigning a colour to a node in order, and check if it violates the colouring criteria. If not, the colour is assigned to the node. If yes, we check for the next available colour. If no possible colour satisfies the condition, we let the colour remain as 0.

   Now, if we have obtained a proper colouring sequence for all the nodes, we give this to output. Otherwise, we first compute the node with maximum degree, and allot this variable to directly store and load from the memory. Then we modify our adjacency matrix and recompute the colouring algorithm. This is performed recursively to remove all the nodes, without which we can provide a satisfactory colour sequence for the graph. This is known as spilling.

6. **Display Register Allocation**
   The colour sequence corresponding to each variable, obtained after previous block, is our output. However, we have displayed this coloured graph in a graphical image form. This is our final output, which gives us the registers allocated corresponding to the given vaariables.

Configuration Details

To configure and run this project, the following softwares and libraries must be installed

1. This project is based on Python 2.7, and requires installation for running the program
2. The program uses networkX libraries for the final block implementation
3. We have used numpy, matplotlib, and string packages, which must be installed prior to the execution

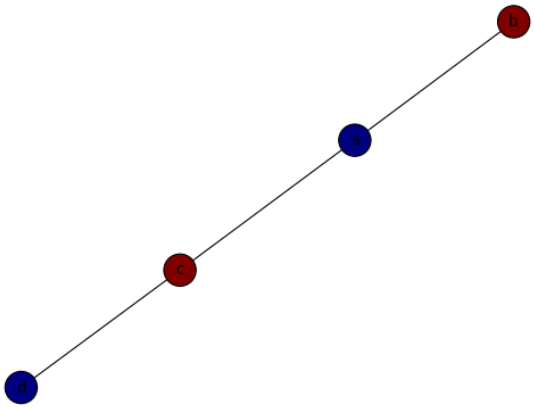
Testing

We have used the following netlists for testing purposes and obtained the following results

1. Test Case 1
   The netlist used is as follows

```
a = 1
b = 2
c = a + b
d = c + a
d = d + 1
b = d*2
d = c - d
return d
```

The output graph is

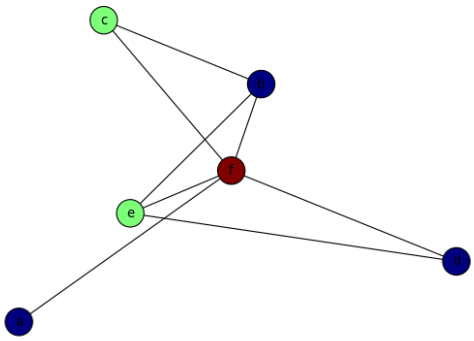


2. Test Case 2

Netlist used is as follows

```
b=1
c=2
f=3
a = b + c
d = -a
e = d + f
b = d + e
e = e - 1
b = f + e
return b
```

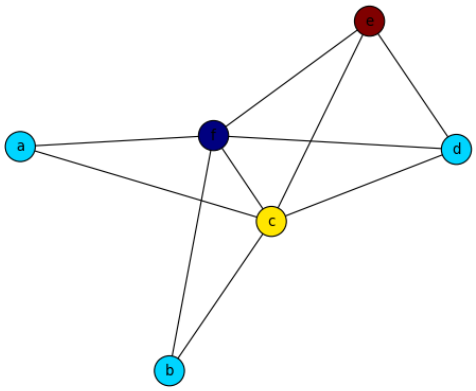
The output graph is



3. Test Case 3
   The following netlist is used

```
f = 0
b = 1
c = 1
while
a = b + c
d = - a
e = d + f
f = 2 * e
b = d + e
e = e - 1
b = f + c
end while
return b
```

   The output graph is



Use of existing softwares/projects

We have used Python 2.7 platform to program, along with the following packages

1. networkX
2. Numpy
3. Matplotlib
4. String

Assumptions and Limitations

We have used the following assumptions in our program with regard to the input netlist.

1. The variables can't be Alpha-Numeric
2. The assignments performed in the arguments of the while, for-loops, and if-else conditions are taken care of by the compilers, and are hence not considered by our program
3. The code is written in a specific format, as given in examples
4. The code does not consist of compilation errors
5. All the temporaries are well initialized
6. No assignment is made without being utilized anywhere. In case it is still done, the variable will be live throughout the program without being used anywhere, reducing efficiency. But it may be useful for the case where we want to keep a flag, to use in another netlist linking to this one.

Limitations of this project

1. The graph colouring algorithm used is of high efficiency, but better colouring techniques do exist. However, for most of the cases with lesser number of variables, the graph colouring obtained is minimized.
2. We have considered that there is no nested if-else condition in a code. Nested if-else conditions can be considered as future work, although we did try to implement it.

Individual Contribution

The following individual contribution was made by each of the team members

1. Ritvik Mathur – 14D070018
   Performed programming for scanning of Netlist and giving ordered list of variable, and construction of RIG from given list of live variables in Adjacency Matrix format

2. Swapnil Bembde – 14D070034
   Performed programming of Computation of live variables, and for Display of Coloured Graph using Networkx

3. Shantanu Choudhary – 14D070044
   Performed coding for Graph colouring for a given Adjacency matrix, and for the recursive case of Spilling

Although we divided our work equally, we also consulted each other during individual distributed work, as well as during program integration.x