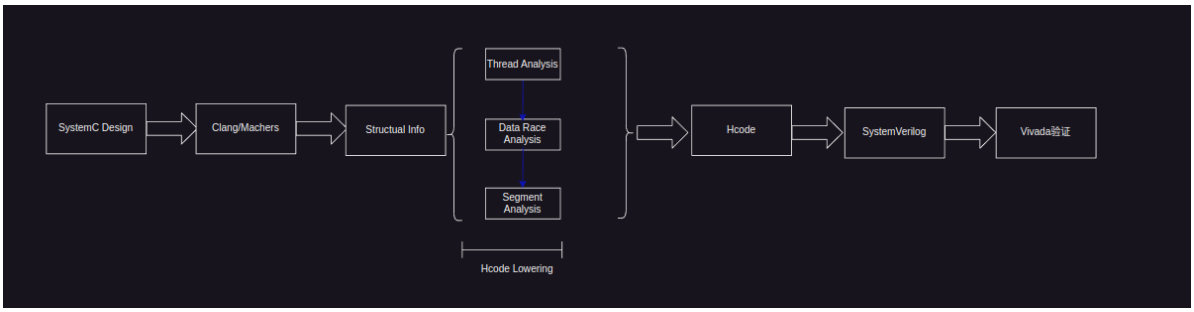
论文的主要想法



示例代码：

```
#include <stdio.h>
int x = 0, y;
behavior B1(event e)
{
    void main() {
        int i = 0;
        do {
            waitfor 1;
            wait e;
            x = i;
        }while (i ++ < 2);
        waitfor 3;
        x = 27; }
};
behaviror B2(event e)
{
    void main() {
        int i = 0;
        do {
            waitfor 2;
            y = i;
            notify e;
        }while (i++ < 2);
        waitfor 4;
        x = 42; }
}
behaviror Main()
{
    event e;
    B1 b1(e);
    B2 b2(e);
    par{
        b1.main();
        b2.main();
    }
}
```

Thread Analysis

Thread Analysis 负责 分析线程中的那些函数会成为system进程；上面的代码经过分析得出代码中共有两个线程。

可能发生数据竞争的行号

全局变量有三个 event e，x, y,

数据竞争发生的行号：9, 10， 13， 21，22，25

Segment Analysis

主要的任务是对进程函数进行分段，分段的原则

1. 对全局变量不能存在读写访问的最大代码段设为一个段；
2. 对全局变量存在读写访问的最大代码段设为一个段；

分段优化

```
1  global int x = 0, y;
2
3  void threadFunc() {
4      int i = 0;
5      x = 2;
6      int j = 0;
7      y = 1;
8  }
```

如果按照上面的原则， 那么会存在四个段， 即： 4， 5， 6， 7 行各为一段；

经过分析上面的代码可以等效成下面的代码：

```
1  global int x = 0, y;
2
3  void threadFunc() {
4      int i = 0;
5      int j = 0;
6      x = 2;
7      y = 1;
8  }
```

这样， 程序只会形成两个段；

代码中的段

经过上面的分段后， 示例代码可以变成如下的段；

```
1: #include <stdio.h>
2: int x = 0, y;
3: behavior B1(event e)        behavior B2(event e)
4: {                           {
5:   void main(){                 void main(){
6:     int i = 0;                    int i = 0;
7:     do {                          do {
8:       waitfor 1;                    waitfor 2;
9:       wait e;                       y = i;
10:      x = i;                        notify e;
11:    }while( i ++ < 2);            } while( i ++ < 2);
12:    waitfor 3;                    waitfor 4;
13:    x = 27;  }                    x = 42;   }
14: };                         };
15: behavior Main()
16: {
17:   event e; B1 b1(e); B2 b2(e);
18:   int main(){
19:     par{
20:       b1.main();
21:       b2.main();
22:   }
23:   printf("x = %d\n", x);}
24: };
```

竞争数据竞争分析

Data Conflict Table

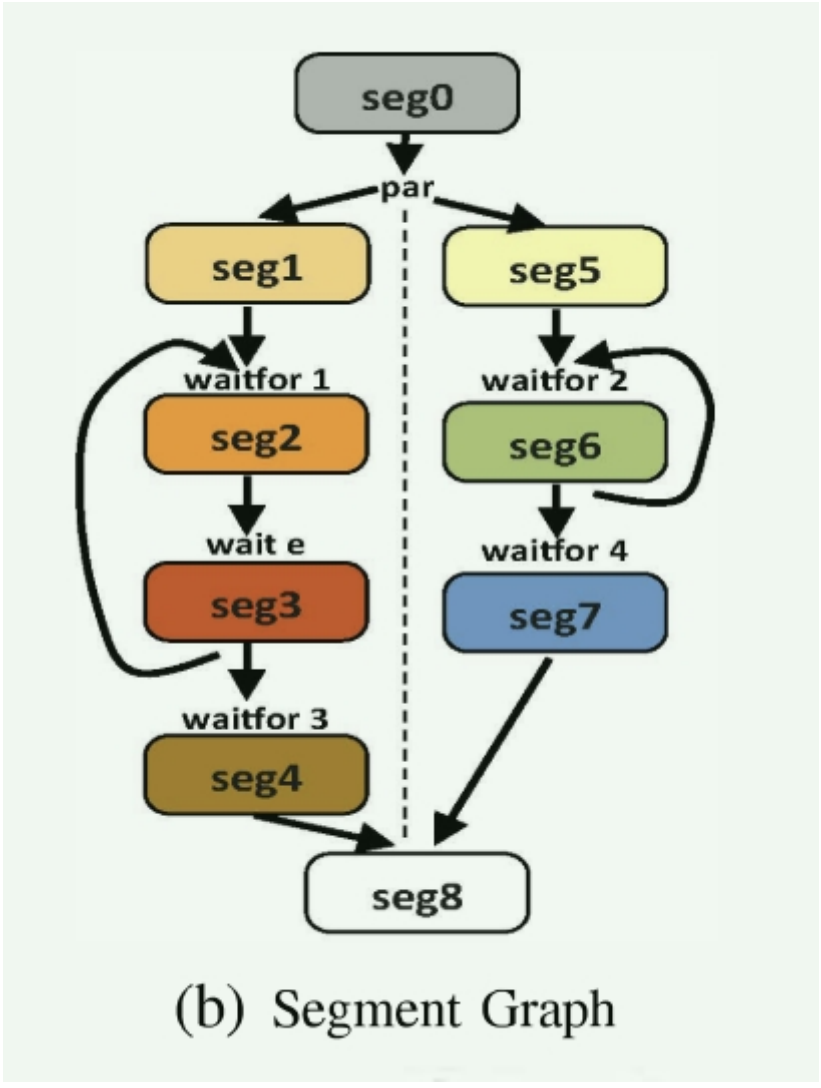
seg	VAlist
0	
1	
2	
3	x(W)
4	x(W)
5	
6	y(W)
7	x(W)
8	x(R)

seg	0	1	2	3	4	5	6	7	8
0	F	F	F	F	F	F	F	F	F
1	F	F	F	F	F	F	F	F	F
2	F	F	F	F	F	F	F	F	F
3	F	F	F	T	T	F	F	T	T
4	F	F	F	T	T	F	F	T	T
5	F	F	F	F	F	F	F	F	F
6	F	F	F	F	F	F	T	F	F
7	F	F	F	T	T	F	F	T	T
8	F	F	F	T	T	F	F	T	F

Event Notification Table

seg	0	1	2	3	4	5	6	7	8
0	F	F	F	F	F	F	F	F	F
1	F	F	F	F	F	F	F	F	F
2	F	F	F	F	F	F	F	F	F
3	F	F	F	F	F	F	F	F	F
4	F	F	F	F	F	F	F	F	F
5	F	F	F	F	F	F	F	F	F
6	F	F	F	T	F	F	F	F	F
7	F	F	F	F	F	F	F	F	F
8	F	F	F	F	F	F	F	F	F

执行流图



(b) Segment Graph

```
`timescale 1ps/1ps
module Main;
initial begin
        seg0;
        fork
            seg1;
            seg5;
            end
        fork
            seg6;
            seg2;
            seg3
            end
        seg4;
        seg7;
        seg8;
end
endmodule
```