Due on 2021-09-29, 23:59 IST.

1 point

1 point

1 point

1 point

1 point

```
NPTEL » C-Based VLSI Design
Course outline
How does an NPTEL online
course work?
Prerequisite: Week 0
Week1: Introduction to C-
based VLSI Design
Week2: C-Based VLSI
Design: Basic Scheduling
Week3: C-Based VLSI
Design: List Based
Scheduling
Week 4: C-Based VLSI
Design: Advanced
Scheduling
Week 5: C-Based VLSI
Design: Allocation and
Binding
Week 6: C-Based VLSI
Design: Allocation, Binding,
Data-path and Controller
Generation
Week 7: C-Based VLSI
Design: Efficient Synthesis of
C Code
Week 8: C-Based VLSI
Design: Hardware Efficient C
Coding
Week 9: C-Based VLSI
Design: Impact of Compiler
Optimizations in Hardware

    Lec1: Frontend Optimizations

   in C

    Lecture Note for Lec1

 Lec 2: HLS Optimizations:
   Case Study 1

    Lecture Note for Lec2

 Lec 3: HLS Optimizations:
   Case Study 2

    Lecture Note for Lec3

  Quiz: Week 9: Assignment 9
 Week 9: Feedback Form

    Solution: Assignment 9

Week 10: Verification of High-
level Synthesis
Week 11: Securing Design
with High-level Synthesis
Week 12: Introduction to EDA
and Recent Advances in C-
Based VLSI Design
```

Download

Live Sessions

```
Week 9: Assignment 9
The due date for submitting this assignment has passed.
As per our records you have not submitted this assignment.

    Which transformation has been applied in the below code snippet Optimized code?
```

```
Source code:
 do i = 1, n
       a[i] = a[i] + sqrt(x);
end do
Optimized Code:
if(n > 0) C = sqrt(x);
 do i = 1, n
       a[i] = a[i] + C;
 end do

    Constant Folding

    Code Motion

    Tree Height Reduction

    Copy Propagation

No, the answer is incorrect.
Score: 0
```

```
Accepted Answers:
Code Motion
2) Consider the code-snippets below and the statements that follow:
```

S1: C2 is more efficient than C1 because it increases locality of reference of array accesses. S2: C2 may not be more efficient than C1 because it changes the iteration order for (j = 0; j < N; j++)

```
C1. for (i = 0; i < N; i++)
                y[i] += A[i][j] * x[j];
  C2. for (ii = 0; ii < N; ii += B)
         for (jj = 0; jj < N; jj += B)
                for (i = ii; i < ii+B; i++)
                       for (j = jj; j < jj+B; j++)
                             y[i] += A[i][j] * x[j];
   Both S1 and S2 are true.
   Only S1 is true.
   Only S2 is true.

    Neither S1 or S2 is true.

  No, the answer is incorrect.
  Score: 0
  Accepted Answers:
  Only S1 is true.
 3) Consider the code snippets given below and select the correct option related to which transformations are applied on the Source Code to
                                                                                                                            0 points
obtain the Transformed Code.
```

```
Source Code:
 for(i = 0; i < 2; i++)
       for(j = 0; j < 2; j++)
              a[i][j] = a[i][j] / 8;
 Transformed Code:
 for(j = 0; i < 2; j++)
        for(i = 0; j <2; i++)
              a[i][j] = a[i][j] >> 3;
 Only Loop Interchange.

    Only operator strength reduction.

    Both Loop Interchange and Operator Strength Reduction

 None of the above
No, the answer is incorrect.
Score: 0
Accepted Answers:
Both Loop Interchange and Operator Strength Reduction
4) Will the below code transformation result in a performance improvement in terms of latency?
                                                                                                             1 point
 Source Code:
  for (i = 0; i < 99; i++)
        for (j = 0; j < 99; j++)
                x[i][j] = x[i - 1][j+1];
 Transformed Code:
  for (j = 0; j < 99; j++)
        for (i = 0; i < 99; i++)
```

```
x[i][j] = x[i - 1][j+1];

    No, since the transformation is incorrect.

    No, since the transformation has no impact

  Yes, since the transformation results in performance improvement.
  None of the above.
No, the answer is incorrect.
Score: 0
Accepted Answers:
No, since the transformation is incorrect.
Following code-snippet is an example of:
m = a + b
                              m = a + b
n = m - c
                               n = m - c
```

m = d - e

O = m + p

the:

int i, j, k;

res[i][j] = 0;

for (i = 0; i < N; i++) { for (j = 0; j < N; j++) {

```
    Variable Propagation

    Copy Propagation

    Variable Renaming

    Code Motion

No, the answer is incorrect.
Score: 0
Accepted Answers:
Variable Renaming
```

m1 = d - e

O = m1 + p

```
for (k = 0; k < N; k++) {
        res[i][j] += mat1[i][k] * mat2[k][j];
    }}} }
   Loop i
   O Loop j
   Loop k
    None of the above
  No, the answer is incorrect.
  Score: 0
  Accepted Answers:
  Loop i

    Consider the matrix multiplication code snippet again with N=10. If we apply Loop Pipelining Optimization on the inner most loop, which of the 2 points

following statements is true compared to the iterative execution of the loop:
S1: Design will run in faster clock if loop pipeline is applied as compared to its iterative execution.
S2: Area overhead is significant for loop pipeline implementation as compared to its iterative implementation.
```

void multiply(int mat1[N][N], int mat2[N][N], int res[N][N]) {

res[i][j] += mat1[i][k] * mat2[k][j];

for (k = 0; k < N; k++) {

}}} }

16

24

37

O 48

Score: 0

Yes

○ No

Score: 0

No

No, the answer is incorrect.

Accepted Answers:

37

No, the answer is incorrect.

Accepted Answers:

void multiply(int mat1[N][N], int mat2[N][N], int res[N][N]) {

int i, j, k; for (i = 0; i < N; i++) { for (j = 0; j < N; j++) { res[i][j] = 0;

6) Consider the matrix multiplication code snippet below. Assume N = 10. The maximum area overhead occurs when loop unrolling is applied on 1 point

```
Only S1 is TRUE.
 Only S2 is TRUE.
 Both S1 and S2 are TRUE.
  Both S1 and S2 are FALSE.
No, the answer is incorrect.
Score: 0
Accepted Answers:
Only S1 is TRUE.
8) Consider the following function. Assume that the inputs a, b, c and d are 12 bit width integers. What would be the output data width?
Return_type fn(int12 a, int12 b, int12 c, int12 d)
    Intx t1, t2, t3;
    t1 = a + b;
    t2 = c * d;
         t3 = t1 * t2;
    return t3;
```

```
if (I < r) {
  // Same as (I+r)/2, but avoids overflow for
  // large I and h
  int m = I + (r - I) / 2;
  // Sort first and second halves
  mergeSort(arr, I, m);
  mergeSort(arr, m + 1, r);
  merge(arr, I, m, r);
```

Consider the following Merge Sort Code. Is it synthesizable by HLS tool?

void mergeSort(int arr[], int I, int r)