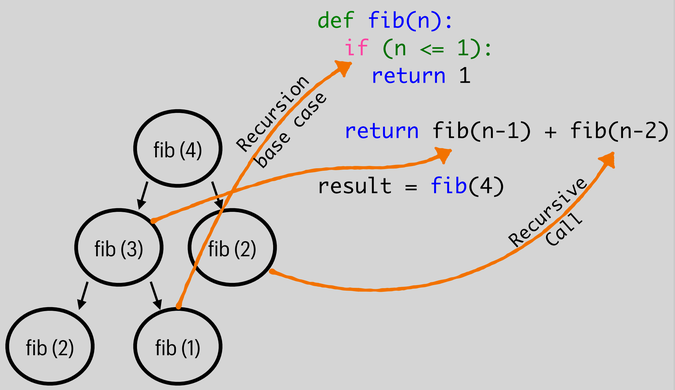
Biscotti: An Approach to Parallel Scheduling for Vectorized Encrypted Arithmetic Circuits

Vedant Paranjape, Aman Gupta, Sreevickrant S., Dulani W., Raghav Malik, Milind Kulkarni

PURDUE UNIVERSITY

Motivation

- Fully Homomorphic Encryption lets us evaluate arbitrary circuits with encrypted inputs.
- Encrypted computation is very slow, but we can use FHE vectors to recover some performance.



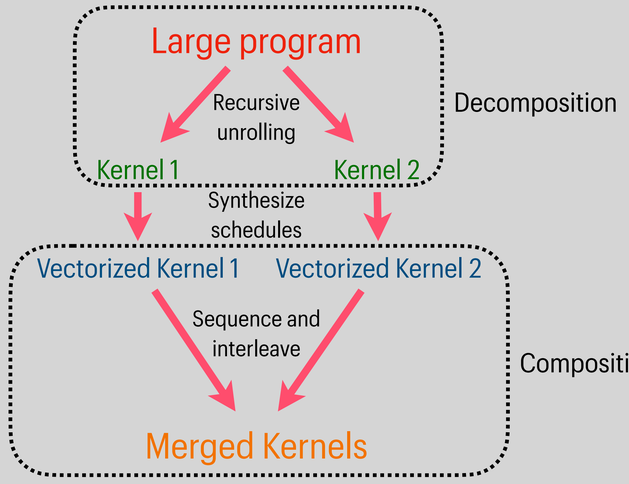
- Divide-and-conquer style problems can parallelized by recursively splitting a problem into subproblems.
- But current techniques cannot map these divide-and-conquer style programs to FHE vectors efficiently.
- Biscotti uses a novel way to recompose recursive problem, while respecting the limitations of FHE vectors.

Our Approach

Intuition: Divide-and-conquer style programs can be broken into smaller subprograms which map to FHE vectors!

Divide: Decompose a large program into multiple kernels that can be scheduled on FHE vectors!

Conquer: Compose the kernels into a unified schedule, combining the results of each kernel!



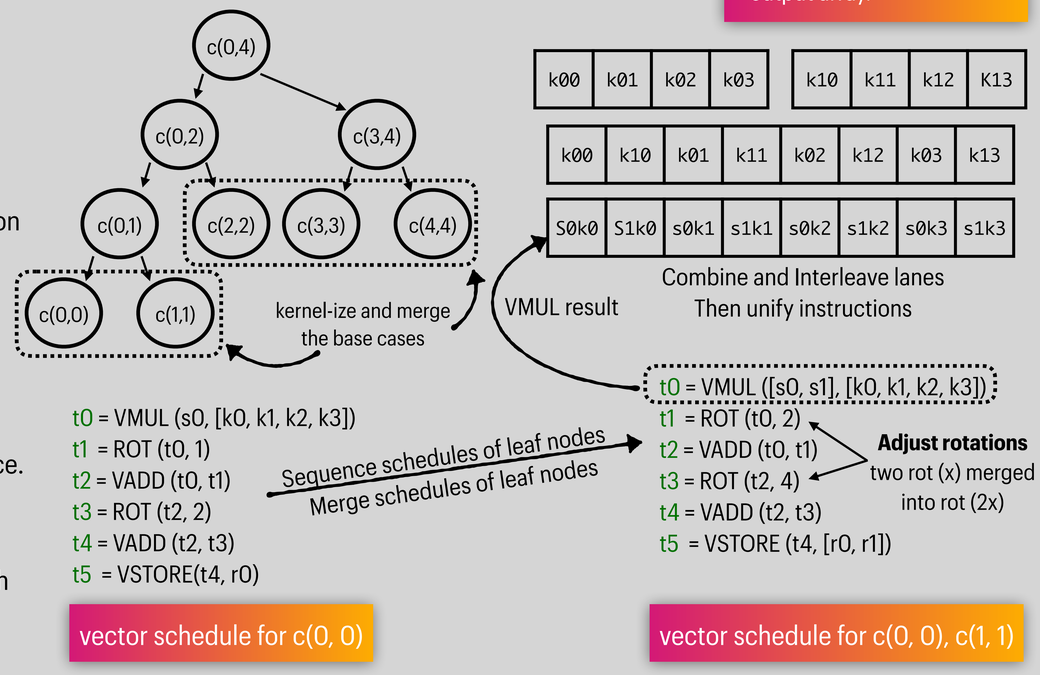
Biscotti targets recursive programs of the form:

- **Base case:** Performs a fixed, constant-size kernel
- **Recursive:** Makes recursive calls on a reduced problem space.
- **Combine:** Store or merge outputs of the children.

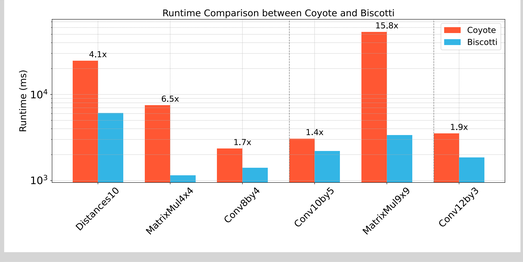
Unrolling the recursion produces a **tree of subproblems**. Each leaf node corresponds to a constant-sized kernel that can be vectorized.

```
def conv8_recursive(signal, kernel, start, end, output):
   if start == end:
      sum = 0
      for j in range(4):
         sum += signal[start] * kernel[j]
      output[start] = sum
      return

   mid = (start + end) / 2
   conv8_recursive(signal, kernel, start, mid, output)
   conv8_recursive(signal, kernel, mid+1, end, output)

def conv8(signal, kernel):
   output = [0] * 5
   conv8_recursive(signal, kernel, 0, 4, output)
   return output
```

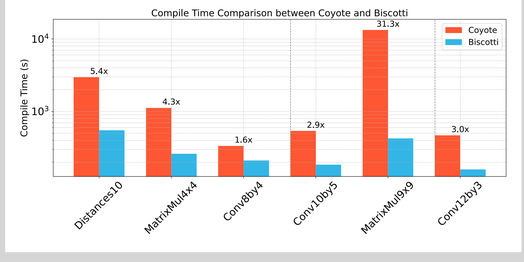
- Tiling is a special case of recursive function unrolling!
- We can merge leaf nodes to map to FHE vectors, enabling parallelism.
- Compose the smaller kernels to store results in output array.



Results



- We compile several benchmarks with Coyote and Biscotti and compare the compile time and runtime of each.
- We see compile-time speedups of up to 31.3x! Mainly due to efficient decomposition of programs into subprograms.
- We see run-time speedups of up to 15.8x! Generating subprograms reduces search space for Coyote - efficient vector schedules!



Conclusion

- Biscotti decomposes programs into smaller subprograms, synthesizes efficient schedules for each and merges them into a single vector schedule.
- It finds a more efficient schedule, and finds it faster as compared to existing works that use synthesis-based techniques.