StreamBit: Sketching high-performance implementations of bitstream programs

Armando Solar-Lezama, Rastislav Bodik
UC Berkeley

Bitstream Programs

- Bitstream programs: A Growing Domain
  - Crypto: DES, Serpent, Rijndael, …
  - coding in general
  - NSA/BitTwiddle
- Bitstream programs operate under strict constraints
  - Performance is very important
    - Resource constrained environments like smartcards
  - Correctness is crucial
    - Subtle bug in Blowfish implementation allowed over half the keys to be cracked in less than 10 minutes
- Efficient bitstream programs are hard to write

Current State of the art

The 64 bits of the input block to be enciphered are first subjected to the following permutation, called the initial permutation **IP**:

IP

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

That is the permuted input has bit 58 of the input as its first bit, bit 50 as its second bit, and so on with bit 7 as its last bit.

Description from NIST document

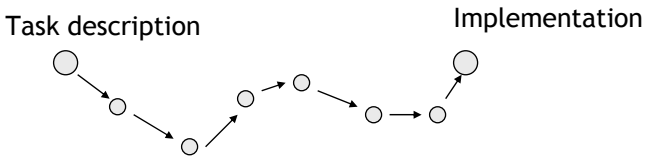
```
#define PERM_OP(a,b,t,n,m) ((t)=((((a)>>(n))^(b))&(m)),\
        (b)^=(t),\
        (a)^=((t)<<(n)))

#define IP(l,r) \
     { \
        register DES_LONG tt; \
        PERM_OP(r,l,tt, 4,0x0f0f0f0fL); \
        PERM_OP(l,r,tt,16,0x0000ffffL); \
        PERM_OP(r,l,tt, 2,0x33333333L); \
        PERM_OP(l,r,tt, 8,0x00ff00ffL); \
        PERM_OP(r,l,tt, 1,0x55555555L); \
     }
```

Code from LibDES

- Lots of hand compilation
  - Tedious
  - Error prone
  - Not portable: optimizing for different word-size non-trivial
- Difficult to automate
  - Exponentially many choices
  - Greedy is Bad

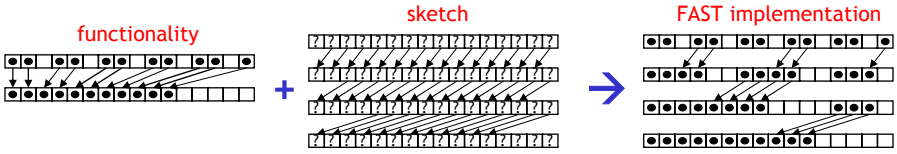
We can do better

- Separate specification from implementation
  - Specify the task without regard for performance
  - Specify Implementation without regard for bug
- How?
  - High level program encodes Task description
  - Sequence of transformations from Task description into equivalent Low Level program encodes an implementation
  - Transformations can be Sketched!



Example

- "Drop every third bit in the bit stream."
- Exhibits many features of complicated permutations
  - Exponentially many choices
  - Greedy choice is suboptimal
- Fast implementation can be sketched

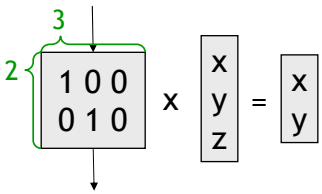


What you gain

- Drop Third Benchmark:
  - Speedups over naïve code with a 14 line sketch:
    - 32 bit on a Pentium IV: 83.8%
    - 64 bit on an Itanium II: 233%
- DES Benchmark:
  - 32 bit on a Pentium IV with 30 line sketch:
    - 634% speedup over naïve
    - Within ¾ of hand optimized libDES

The DSL: StreamIt

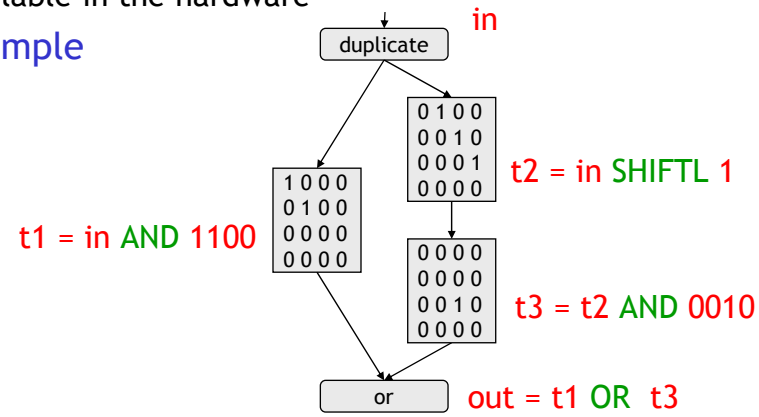
- High-level bitstream *algorithms* written a language derived from *StreamIt*
  - Synchronous Dataflow language
  - Filters represented internally as matrices



consumes a 3-bit chunk of input;
produces a 2-bit of output.

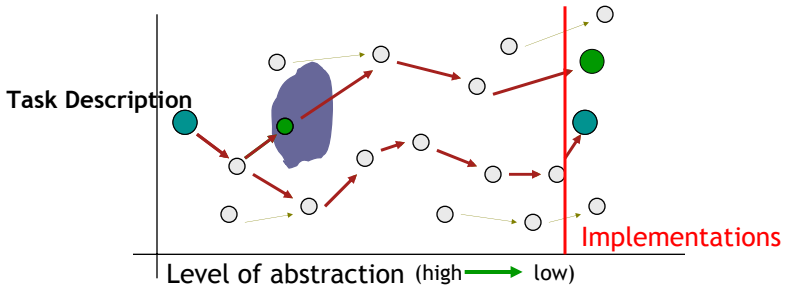
Implementation in StreamIt

- Make each filter correspond to one basic operation available in the hardware
- Example



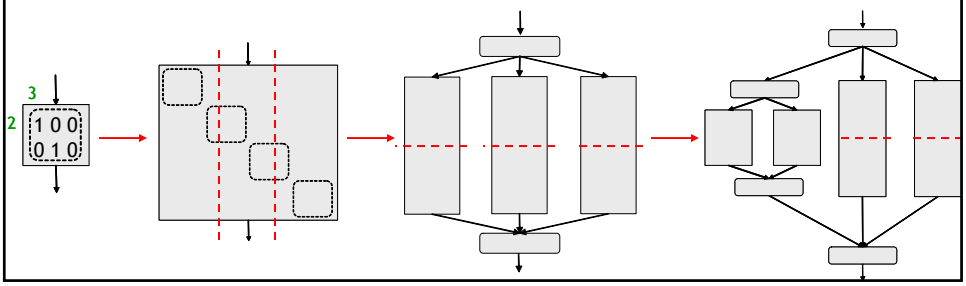
The Full Picture

- For every task there is a space of programs that describe it
- Greedy Transformation algorithm defines a path from any program to an implementation
- Halfway Transformation can be used to get a different implementation
- A Sketch can simplify the process of writing a transformation



Greedy Transformation

- Defines a default implementation for any program
- Leveraged by the user for custom implementations
- Example: Drop Third Bit
  - Unroll filter
  - decompose into filters operating on W=4 bits of input.
  - decompose into filters producing W=4 bits of output

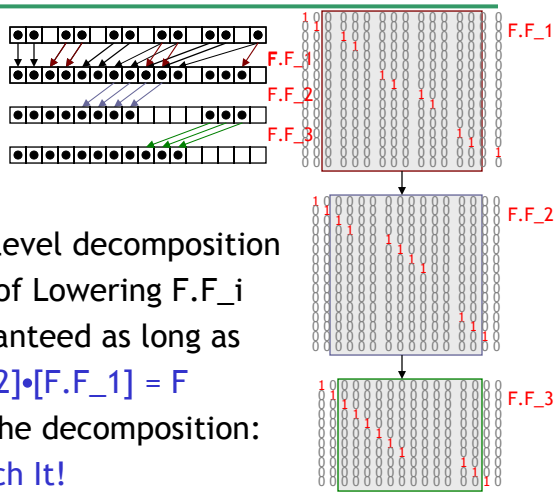


Half way transformations

- Allows the user to guide lowering
  - Impart structure to the filter
  - Bring it closer to the desired implementation
- Add structure by decomposing into a pipeline of steps
  - Equivalent to specifying a matrix decomposition.
  - filter = filter1 • filter2 guarantees correctness
- This can be done hierarchically,
  - Detail is only added where necessary.

Half way transformations-example

Half way transformation to specify FAST bit shifting algorithm:



- User provides high level decomposition
- System Takes care of Lowering F.F_i
- Correctness is guaranteed as long as

  [F.F_3]•[F.F_2]•[F.F_1] = F
- Avoid spelling out the decomposition:

  Sketch It!

Sketching: How it works

- Start with a sketch
- Define $x_{i,j}$ as the amount bit i will move on step j
- Semantic equivalence imposes linear constraints on the $x_{i,j}$
- Many of the constraints in the sketch also impose linear constraints on $x_{i,j}$
- Solving the linear constraints produces a space of possible solutions
- Map the nonlinear constraints to this solution space
- Search

```
SketchDecomp[
      [shift(1:32 by 0 || 1)],
      [shift(1:32 by 0 || 2)],
      [shift(1:32 by 0 || 4)],
      [shift(1:32 by 0 || 8)]
]( Filter );
```

Programming with StreamBit

The 64 bits of the input block to be enciphered are first subjected to the following permutation, called the initial permutation **IP**:

IP

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

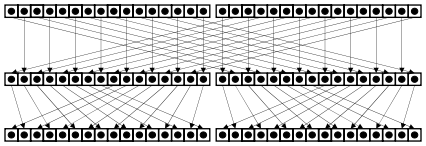
That is the permuted input has bit 58 of the input as its first bit, bit 50 as its second bit, and so on with bit 7 as its last bit.

Description from NIST document

```
filter DES_IP  {
    Work push 2 pop 3  {
        int p[] = { 58, 50, 42, 34, 26, 18, 10, 2,
                    60, 52, 44, 36, 28, 20, 12, 4,
                    62, 54, 46, 38, 30, 22, 14, 6,
                    64, 56, 48, 40, 32, 24, 16, 8,
                    57, 49, 41, 33, 25, 17, 9, 1,
                    59, 51, 43, 35, 27, 19, 11, 3,
                    61, 53, 45, 37, 29, 21, 13, 5,
                    63, 55, 47, 39, 31, 23, 15, 7    };
        for (int i = 0; i < 64; ++i){
            x = peek(p[i]);
            push(x);
            pop();
        }
    }
}
```

StreamIt Code

- StreamBit program close to original task description
- Clever algebraic transformation expressed as a sketch



```
SketchDecomp[
      [ shift(1:2:31 by -33), shift(34:2:64 by 33),
        shift(1:64 by 0 || 33 || -33) ],
      [ ]
] (DES Encoding.IP);
```

Sketch

Productivity

- User still has to come up with clever idea
  - People are good at clever ideas
  - Machines are good at tedious details
- Sketching allows user to experiment with different implementations
  - 6 different implementations in one afternoon
  - Don't have to worry about bugs

Concluding Remarks

- StreamBit allows for
  - Task specification oblivious to performance
  - Implementation specification without bugs
- Same idea may apply in other domains
  - If people currently resort to very low level coding
  - If some algebraic structure can be imposed on the task
  - It may be amiable to Sketching.