



Towards Semantics Lifting for Scientific Computing: A Case Study on FFT

Naifeng Zhang, Sanil Rao, Mike Franusich, Franz Franchetti

ORNL AI4Science Workshop



Does this code implement the fast Fourier transform?

1	<pre>#include <math.h></math.h></pre>	
2	<pre>#include <stdlib.h></stdlib.h></pre>	
3	#define M_PI 3.14159265358979323846	
4	<pre>void fft_recursive(double* data, int n) {</pre>	
5	<pre>if (n <= 1) return;</pre>	
6	<pre>// Allocate temporary storage for half-size FFTs.</pre>	
7	<pre>double* even = (double*)malloc(n * sizeof(double));</pre>	
8	<pre>double* odd = (double*)malloc(n * sizeof(double));</pre>	
9	for (int $i = 0; i < n / 2; ++i$) {	
10	even[2 * i] = data[4 * i];	Complex numbers
11	even[2 * i + 1] = data[4 * i + 1];	
12	odd[2 * i] = data[4 * i + 2];	
13	odd[2 * i + 1] = data[4 * i + 3];	
14	}	
15	<pre>// Recursively compute FFTs.</pre>	
16	fft_recursive(even, n / 2);	Docursion
17	<pre>fft_recursive(odd, n / 2);</pre>	
18	for (int i = 0; i < n / 2; ++i) {	
19	<pre>double theta = -2.0 * M_PI * i / n;</pre>	
20	<pre>double wr = cos(theta);</pre>	Sing & cosing
21	<pre>double wi = sin(theta);</pre>	
22	<pre>// Twiddle factor multiplication.</pre>	
23	double real = odd[2 * i] * wr - odd[2 * i + 1] * wi;	
24	double imag = odd[2 * i] * wi + odd[2 * i + 1] * wr;	
25	data[2 * i] = even[2 * i] + real;	
26	data[2 * i + 1] = even[2 * i + 1] + imag;	
27	data[2 * (i + n / 2)] = even[2 * i] – real;	
28	data[2 * (i + n / 2) + 1] = even[2 * i + 1] – imag;	
29	}	
30	// Cleanup.	
31	free(even);	
32	<pre>free(odd);</pre>	
33	}	

Hey GPT, can you provide a C implementation of the fast Fourier transform (FFT)?



Does this code implement the fast Fourier transform?

1	<pre>#include <math.h> #include <math.h></math.h></math.h></pre>
2	#Include \Statib.n>
4	void fft recursive(double* data_int_n) {
5	if (n <= 1) return
6	// Allocate temporary storage for half-size FFTs.
7	<pre>double* even = (double*)malloc(n * sizeof(double)):</pre>
8	<pre>double* odd = (double*)malloc(n * sizeof(double));</pre>
9	for (int $i = 0$; $i < n / 2$; ++i) {
10	even[2 * i] = data[4 * i];
11	even[2 * i + 1] = data[4 * i + 1]; Complex numpers
12	odd[2 * i] = data[4 * i + 2];
13	odd[2 * i + 1] = data[4 * i + 3];
How to ve	rify the semantic correctness for LLM-
10	fit equivier (add in 7.2); Recursion
generated	
20	double wr = cos(theta);
21	double wi = sin(theta);
22	// Twiddle factor multiplication.
23	<pre>double real = odd[2 * i] * wr - odd[2 * i + 1] * wi;</pre>
24	<pre>double imag = odd[2 * i] * wi + odd[2 * i + 1] * wr;</pre>
25	data[2 * i] = even[2 * i] + real;
26	data[2 * i + 1] = even[2 * i + 1] + imag;
27	data[2 * (1 + n / 2)] = even[2 * 1] - real;
28	data[2 * (1 + n / 2) + 1] = even[2 * 1 + 1] - 1mag;
29	} (/ 0)
30	// Leanup.
31	rree(even);
32	inee(odd);
33	3

Hey GPT, can you provide a C implementation of the fast Fourier transform (FFT)?

We can solve for small sizes using symbolic execution

```
1 #include <math.h>
 2 #include <stdlib.h>
 3 #define M PI 3.14159265358979323846
 4 void fft_recursive(double* data, int n) {
       if (n <= 1) return;</pre>
 5
       // Allocate temporary storage for half-size FFTs.
 6
        double* even = (double*)malloc(n * sizeof(double));
 7
 8
        double* odd = (double*)malloc(n * sizeof(double));
        for (int i = 0; i < n / 2; ++i) {</pre>
 9
            even[2 * i] = data[4 * i];
10
           even[2 * i + 1] = data[4 * i + 1];
11
12
           odd[2 * i] = data[4 * i + 2];
           odd[2 * i + 1] = data[4 * i + 3];
13
14
        }
       // Recursively compute FFTs.
15
16
        fft recursive(even, n / 2);
        fft_recursive(odd, n / 2);
17
        for (int i = 0; i < n / 2; ++i) {</pre>
18
            double theta = -2.0 \times M_PI \times i / n;
19
            double wr = cos(theta);
20
            double wi = sin(theta);
21
22
           // Twiddle factor multiplication.
23
            double real = odd[2 * i] * wr - odd[2 * i + 1] * wi;
            double imag = odd[2 * i] * wi + odd[2 * i + 1] * wr;
24
25
            data[2 * i] = even[2 * i] + real;
            data[2 * i + 1] = even[2 * i + 1] + imag;
26
27
            data[2 * (i + n / 2)] = even[2 * i] - real;
            data[2 * (i + n / 2) + 1] = even[2 * i + 1] - imag;
28
29
        }
30
       // Cleanup.
        free(even);
31
32
        free(odd);
33 }
```

FFT is a linear transform: $y = T \cdot x$

```
When n = 4,
```

 $T = egin{bmatrix} 1 & 1 & 1 & 1 \ 1 & -i & -1 & i \ 1 & -1 & 1 & -1 \ 1 & i & -1 & -i \end{bmatrix}$

We can derive this matrix via symbolic execution!

What about larger sizes?



Proposed solution: stepwise semantics lifting







Software/Hardware Generation for Performance



CMU



Lifting GPT-generated C FFT

LLM-generated C code

1	<pre>#include <math.h></math.h></pre>		
2	<pre>#include <stdlib.h></stdlib.h></pre>		
3	#define M_PI 3.14159265358979323846		
4	<pre>void fft_recursive(double* data, int n) {</pre>		
5	if (n <= 1) return;		
6	<pre>// Allocate temporary storage for half-size FFTs.</pre>		
7	<pre>double* even = (double*)malloc(n * sizeof(double));</pre>		
8	<pre>double* odd = (double*)malloc(n * sizeof(double));</pre>		
9	<pre>for (int i = 0; i < n / 2; ++i) {</pre>		
10	even[2 * i] = data[4 * i];		
11	even[2 * i + 1] = data[4 * i + 1];		
12	odd[2 * i] = data[4 * i + 2];		
13	odd[2 * i + 1] = data[4 * i + 3];		
14	}		
15	// Recursively compute FFTs.		
16	fft_recursive(even, n / 2);		
17	<pre>fft_recursive(odd, n / 2);</pre>		
18	<pre>for (int i = 0; i < n / 2; ++i) {</pre>		
19	double theta = -2.0 * M_PI * i / n;		
20	<pre>double wr = cos(theta);</pre>		
21	<pre>double wi = sin(theta);</pre>		
22	<pre>// Twiddle factor multiplication.</pre>		
23	<pre>double real = odd[2 * i] * wr - odd[2 * i + 1] * wi;</pre>		
24	<pre>double imag = odd[2 * i] * wi + odd[2 * i + 1] * wr;</pre>		
25	data[2 * i] = even[2 * i] + real;		
26	<pre>data[2 * i + 1] = even[2 * i + 1] + imag;</pre>		
27	data[2 * (i + n / 2)] = even[2 * i] - real;		
28	data[2 * (i + n / 2) + 1] = even[2 * i + 1] - imag;		
29	}		
30	// Cleanup.		
31	free(even);		
32	<pre>free(odd);</pre>		
33	}		
-			

Symbolic expression

$$\mathcal{A}_{n} = \overline{(\mathrm{DFT}_{2} \otimes \mathrm{I}_{n/2}) \operatorname{T}_{n/2}^{n}} (\mathrm{I}_{2} \otimes M_{n/2}) \overline{\mathrm{L}_{2}^{n}}$$

Symbolic execution on small n & Induction

High-level semantics

$$\overline{\text{OFT}_n} = \overline{(\text{DFT}_2 \otimes \text{I}_{n/2}) \text{T}_{n/2}^n (\text{I}_2 \otimes \text{DFT}_{n/2})} \text{L}_2^n$$

Equivalence checking 🔽



FFT specification in SPIRAL

 $DFT_n = (DFT_m \otimes I_k) T_k^n (I_m \otimes DFT_k) L_m^n, n = mk$



Lifting GPT-generated C FFT

LLM-generated C code

- 1 #include <math.h</pre>
- 2 #include <stdlib.h>
- 3 #define M_PI 3.14159265358979323846
- 4 void fft_recursive(double* data, int n) {
- if (n <= 1) return
- // Allocate temporary storage for half-size FFTs.
- double* even = (double*)malloc(n * sizeof(double));
- double* odd = (double*)malloc(n * sizeof(double));

Symbolic expression

 $M_n = \overline{(\mathrm{DFT}_2 \otimes \mathrm{I}_{n/2}) \operatorname{T}_{n/2}^n} (\mathrm{I}_2 \otimes M_{n/2}) \overline{\mathrm{L}_2^n}$

Symbolic execution on small n 8

Safeguarding LLMs: Neural-symbolic AI in code generation

21		<pre>double wi = sin(theta);</pre>
		<pre>double real = odd[2 * i] * wr - odd[2 * i + 1] * wi;</pre>
		<pre>double imag = odd[2 * i] * wi + odd[2 * i + 1] * wr;</pre>
30		// Cleanup.
31		free(even);
32		free(odd):
33	3	
	5	

 $DFT_n = (DFT_2 \otimes I_{n/2}) T_{n/2}^n (I_2 \otimes DFT_{n/2}) L_2^n$

Equivalence checking 🗸

FFT specification in SPIRAL

 $DFT_n = (DFT_m \otimes I_k) T_k^n (I_m \otimes DFT_k) L_m^n, n = mk$

naifengz@cmu.edu