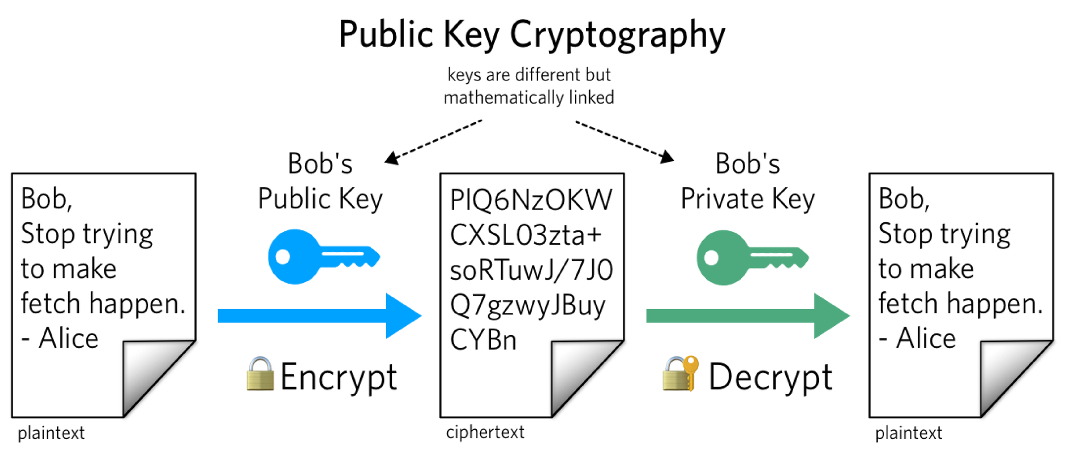
Interval Arithmetic FFT for Large Integer Multiplication

Zibo Gong*, Nathan Zhu*, Matthew Ngaw*, Joao Rivera†,
Larry Tang*, Eric Tang*, Het Mankad*, Franz Franchetti*
*Carnegie Mellon University †ETH Zurich

Motivation

Large integer multiplication has applications in:
- Modular arithmetic routines in cryptographic systems
- Number theoretic tasks in kernels
- Computer algebra systems

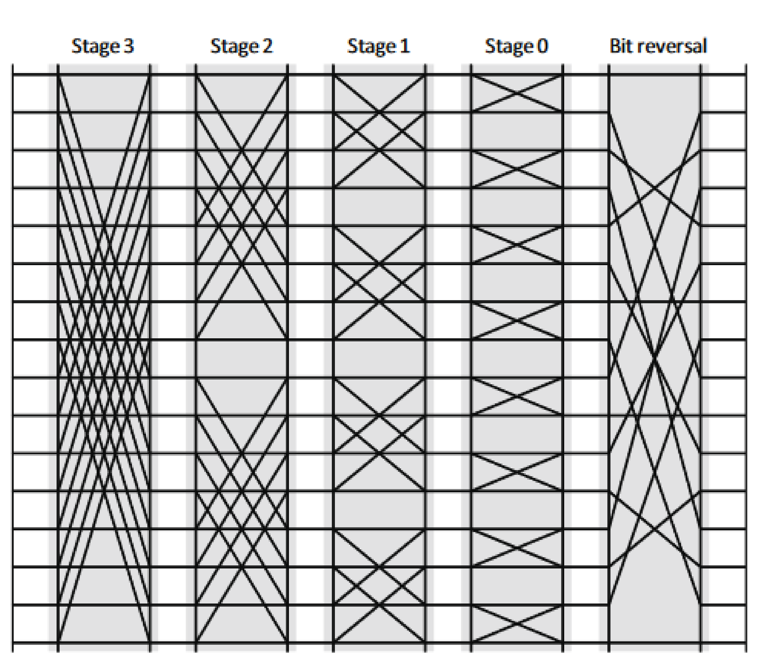


Computation complexity of naive multiplication: $O(N^2)$
Too slow and infeasible for these applications

Our Approach

Fast Fourier Transform (FFT) + Interval Arithmetic

- We perform our multiplication pipeline with FFT → computation complexity O(NlogN)



$$y[n] = \sum_{m=0}^{N-1} x_1[m]x_2[(n-m)_N] \overset{\text{DFT/IDFT}}{\Longleftrightarrow} Y[k] = X_1[k] \circ X_2[k]$$

$$12 \times 21 = [1,2] * [2,1] = [2,5,2]$$

$$12 \times 21 = \mathcal{F}^{-1}(\mathcal{F}[1,2] \odot \mathcal{F}[2,1]) = [2,5,2]$$

- **Pitfalls** ⚠: Complex FFTs lead to **round-off errors** associated with **floating point arithmetic**

- Interval Arithmetic to handle floating point round-off error ☑

$$[x_1, x_2] + [y_1, y_2] = [x_1 + y_1, x_2 + y_2]$$

$$[x_1, x_2] \cdot [y_1, y_2] = [\min\{x_1y_1, x_1y_2, x_2y_1, x_2y_2\}, \max\{x_1y_1, x_1y_2, x_2y_1, x_2y_2\}]$$

Algorithm Interval-arithmetic FFT-based Integer Multiplication

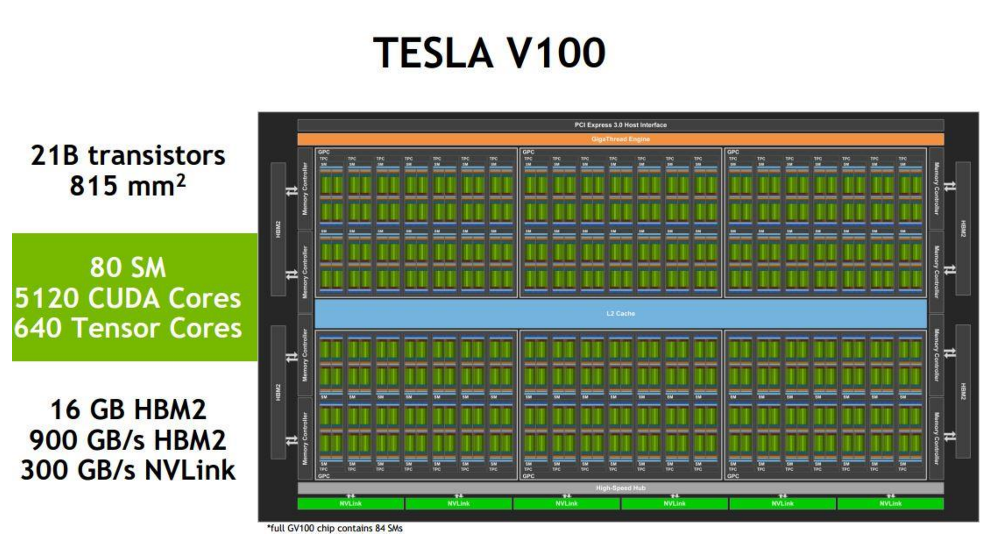
Input: vector x and y of size N
Output: vector z of size 2N
$i1 \leftarrow$ **ZeroPad**(x)
$i2 \leftarrow$ **ZeroPad**(y) ▷ zero padding to size of 2N
$f1 \leftarrow$ **IntervalFFT**$(i1)$
$f2 \leftarrow$ **IntervalFFT**$(i2)$
$prod \leftarrow$ **Mul**$(f1, f2)$ ▷ point-wise multiplication
$raw\_retv \leftarrow$ **IntervalIFFT**$(prod)$
$z \leftarrow$ **Carry**(raw_retv) ▷ Propagate carries

Our Optimization in algorithm-level:

- **IntervalFFT** uses Decimation-in-frequency (DIF) and **IntervalIFFT** uses Decimation-in-time (DIT) → avoid bit reversal operation
- Used Real FFT to help decrease the memory usage

Results

- We tested our algorithms on two platforms:
- GPU (NVIDIA Tesla V100)
- CPU (Intel Xeon E7-4850 v3)



OpenMP

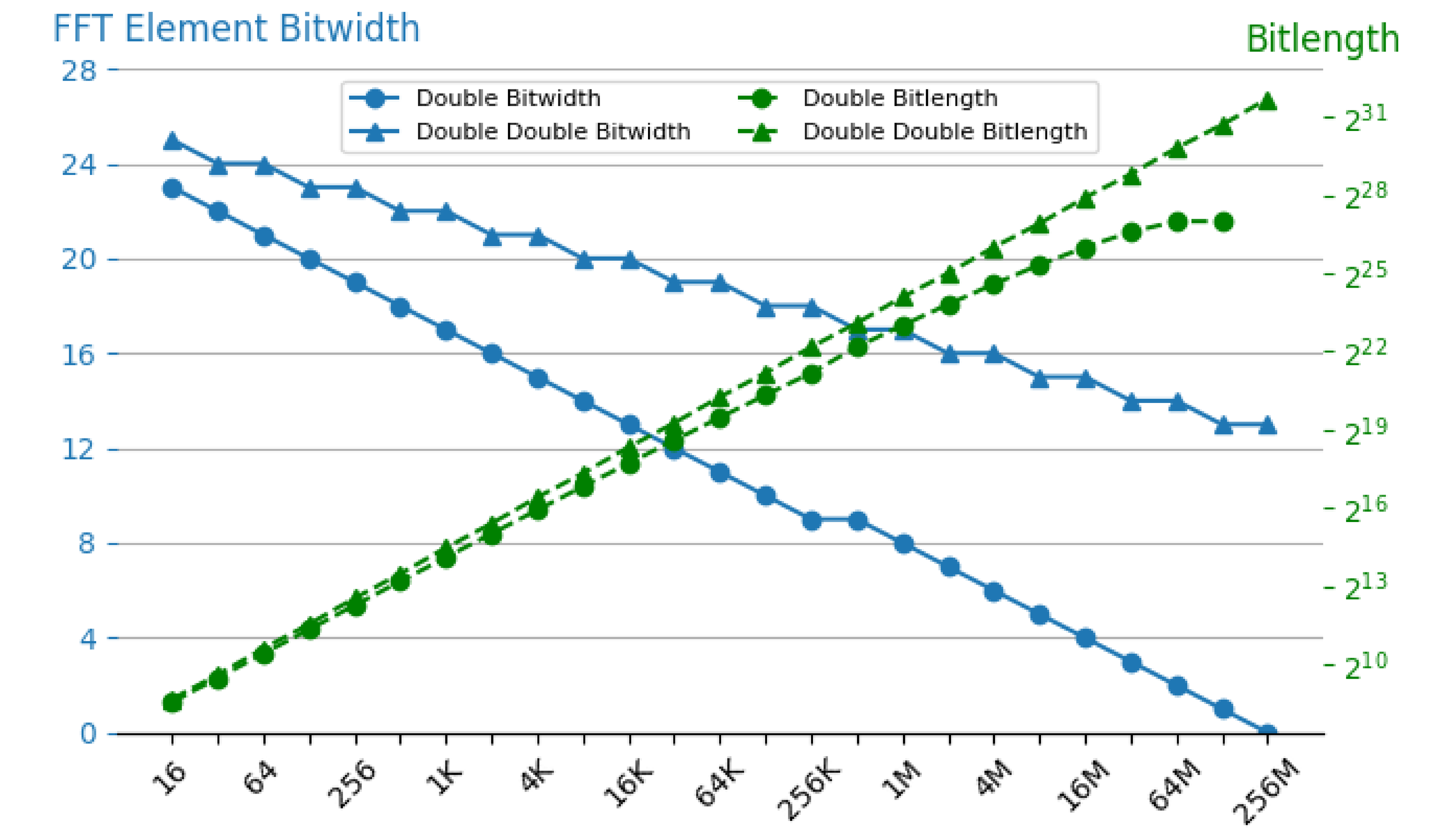
GMP
«Arithmetic without limitations»

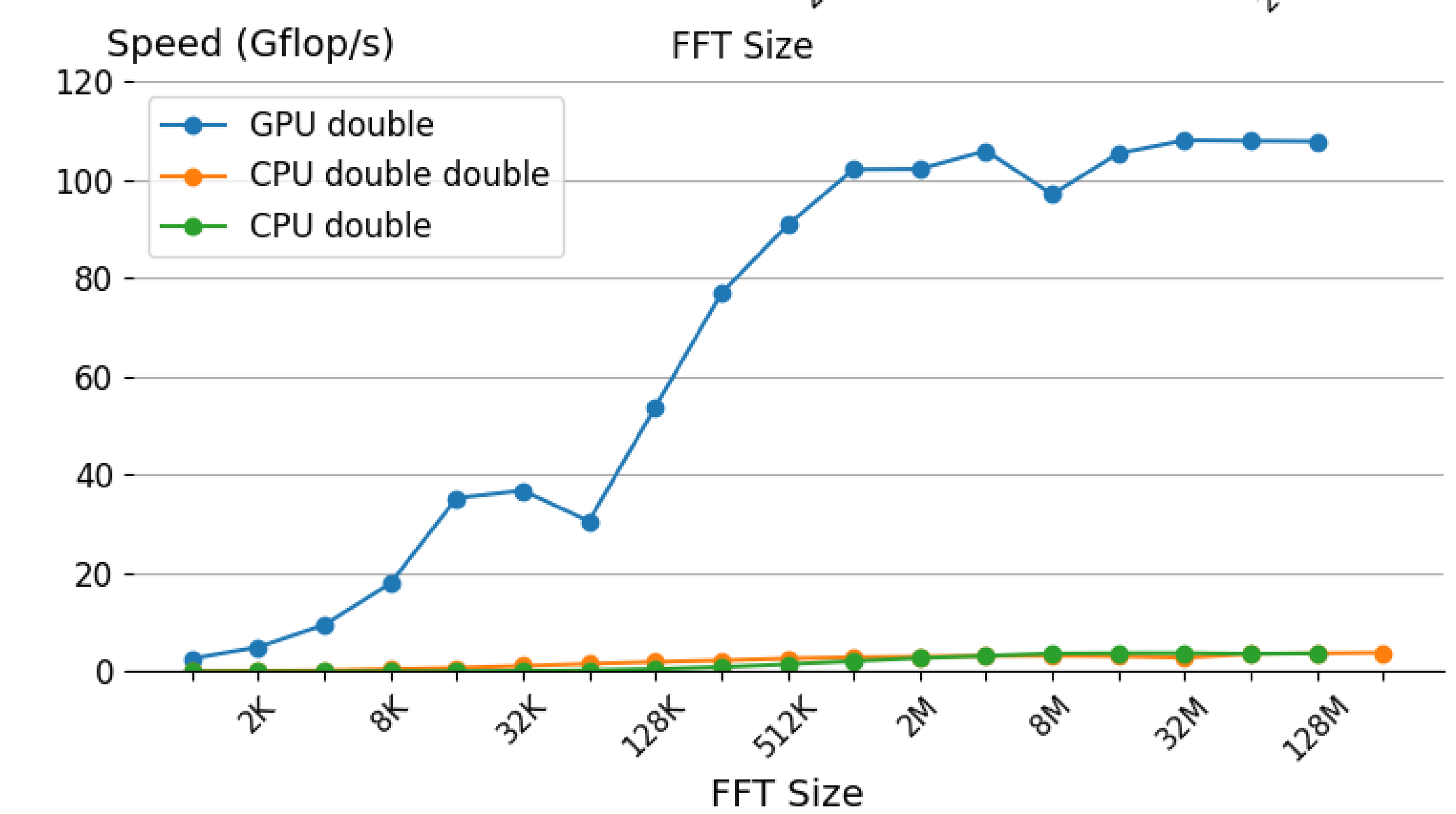
Our Optimization in software-level:

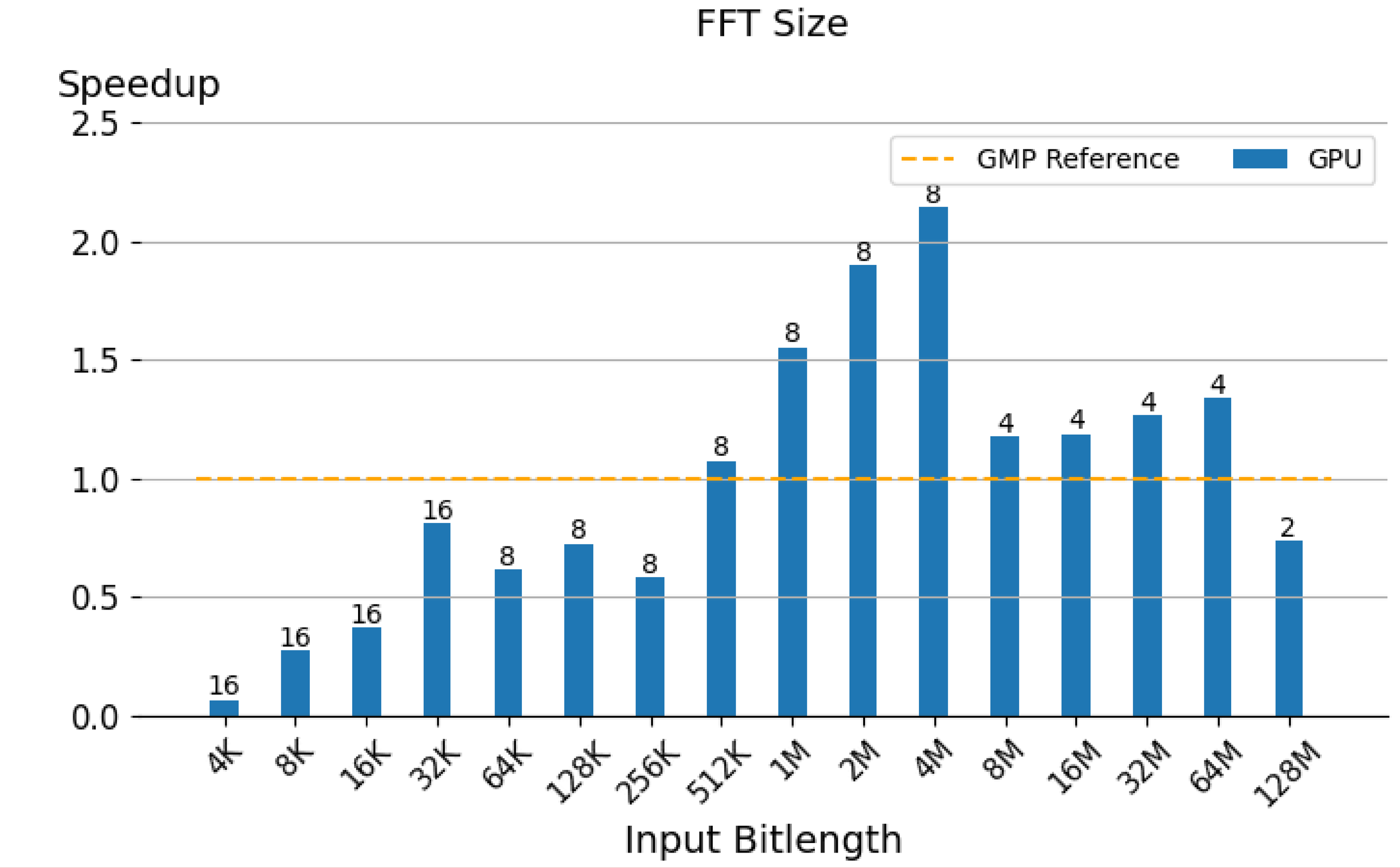
- Used double-double (128 bits floating point data type) to provide higher precision
- Packed up more bits for an element to lower the FFT size
- Used shared memory for GPU to decrease global memory accesses
- Used OpenMP for CPU to parallelize

Our Goal:

- Support multiplication of billions of digits input
- Achieve comparable or better performance in contrast with GMP







Future work

- We plan to use pruned FFTs to further decrease our memory usage and speed up our algorithm
- We plan to combine our algorithm with Karatsuba to help round-off errors
- We plan to use higher radix FFT in our algorithm