Fast Matrix Multiplications for Multicore Multiprocessor Systems

Paolo D'Alberto
Yahoo!

Marco Bodrato and Alex Nicolau

What We Present

- **FastMM: A library of fast algorithms for MM and its performance, for different machines, types and sizes**
  - Fast Algorithms: 3M, Strassen, Winograd
  - Types: single, double, single complex, and double complex
  - Problem size: 2,000 – 12,000
- **The algorithms are hand crafted**
  - The development and engineering is automatic

Our Main Message

- **Performance**
  - Algorithm design + development + system based optimizations
  - There is no dominant algorithm
- **We *show* that :**
  - Our new algorithms translate to simple code
  - Algorithm design, development and care for system optimizations can be done naturally using recursive algorithms

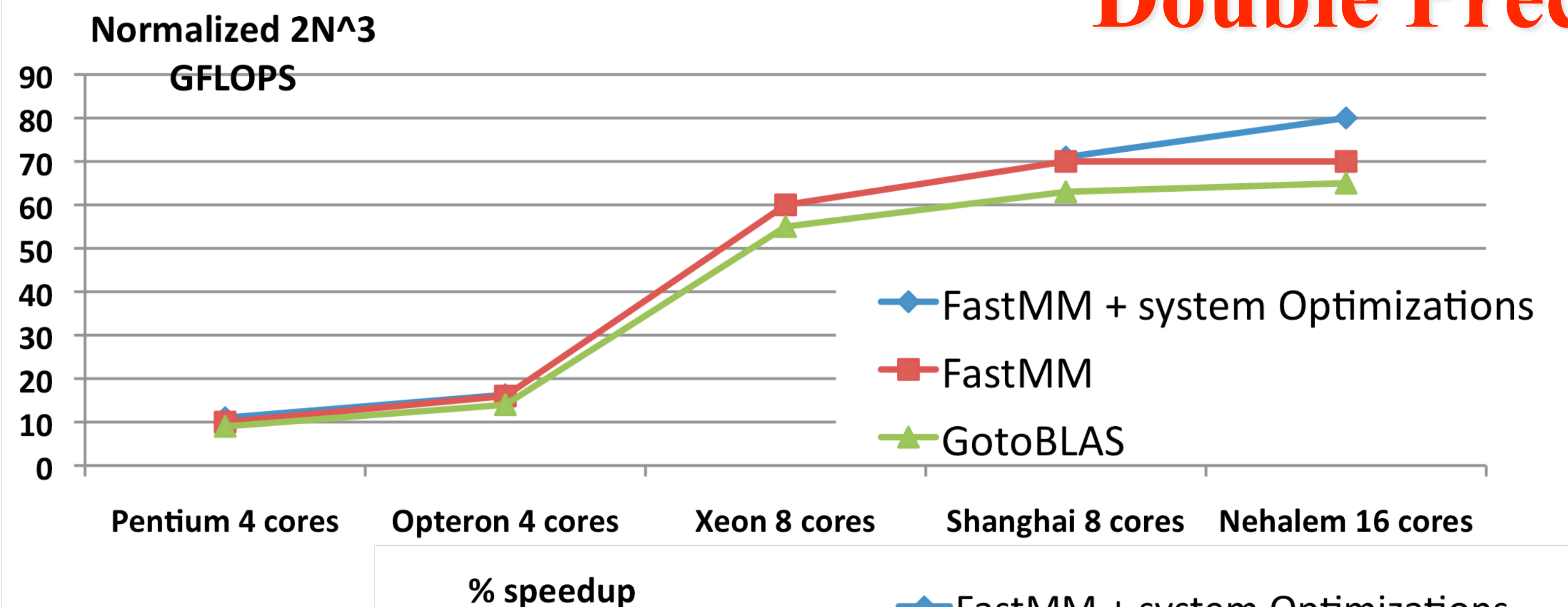
Disclaimer: no dominant algorithm

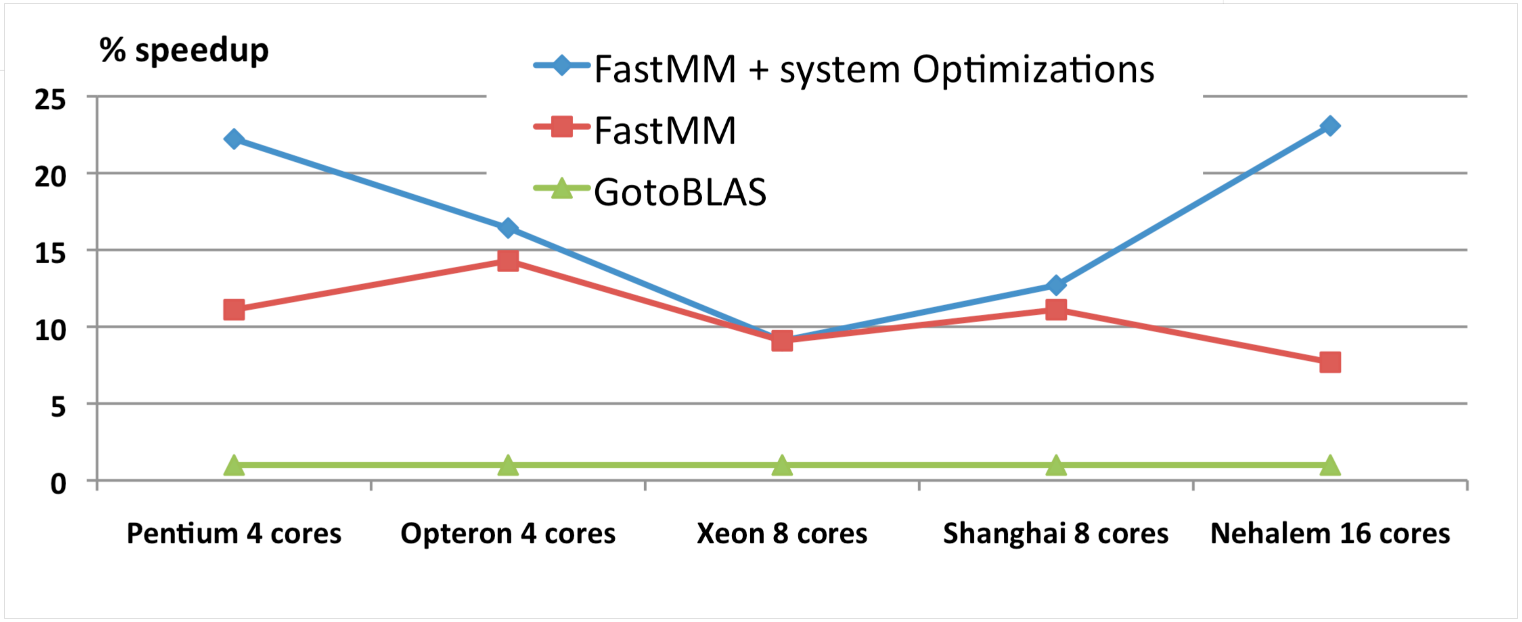
- **There is NOT a single algorithm that is always better**
  - You may say that there is no good solution because there is not a single solution
  - Why bother ?
- **If you don't: you may miss the Gestalt's effect of algorithm design and algorithm optimization**
  - You may lose a 30% speed-up
- **I am not here to preach for any specific algorithm**

State-of-the-art

- **Take any BLAS library: MKL, ATLAS, GotoBLAS**
  - E.g., GotoBLAS
  - 90-95% of peak performance
    - *Nehalem 2 processor system (16 cores), 150 GFLOPS for single precision matrices*
    - *Performance equivalent to a Cell processor*
  - Further improvements are very hard
- **We have the perfect computational work horse**
  - We can build complex applications on it
  - We can build fast MM
- **We do not compete with BLAS, we extend BLAS**

Algorithms, Systems, and Optimizations
Double Precision





Algorithm + Optimizations

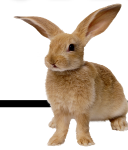


TABLE IV

WINOGRAD'S MM (IMPROVED IMPLEMENTATION C=AB)

Sequential		Parallel/Pipelining		
$S = A_3 - A_2$	1:	$S = A_3 - A_2$		
$T = B_3 - B_2$	2:	$T = B_3 - B_2$		
$C_3 = ST$	3:	$C_3 = ST$		
$U \mathrel{+}= A_1B_2$	4:	$U + = A_1B_2$		
$C_0 = A_0B_0$	5:	$C_0 = A_0B_0$	$S = S + A_1$	$T = T + B_1$
$C_0 \mathrel{+}= U$		$C_0 + = U$		
$S = S + A_1$				
$T = T + B_1$				
$U \mathrel{+}= ST$		$U + = ST$		
$C_1 = U - C_3$	8:	$C_1 = U - C_3$		
$S = A_0 - S$	9:	$S = A_0 - S$		
$C_1 \mathrel{+}= SB_1$	10:	$C_1 + = SB_1$	$T = B_0 - T$	
$T = B_0 - T$				
$C_2 \mathrel{+}= A_2T$	11:	$C_2 + = A_2T$	$S = A_3 - A_1$	$T = B_3 - B_1$
$S = A_3 - A_1$				
$T = B_3 - B_1$				
$U \mathrel{-}= ST$	12:	$U - = ST$		
$C_3 \mathrel{-}= U$	13:	$C_3 - = U$		
$C_2 \mathrel{-}= U$	14:	$C_2 - = U$		

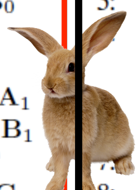
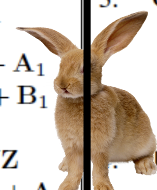


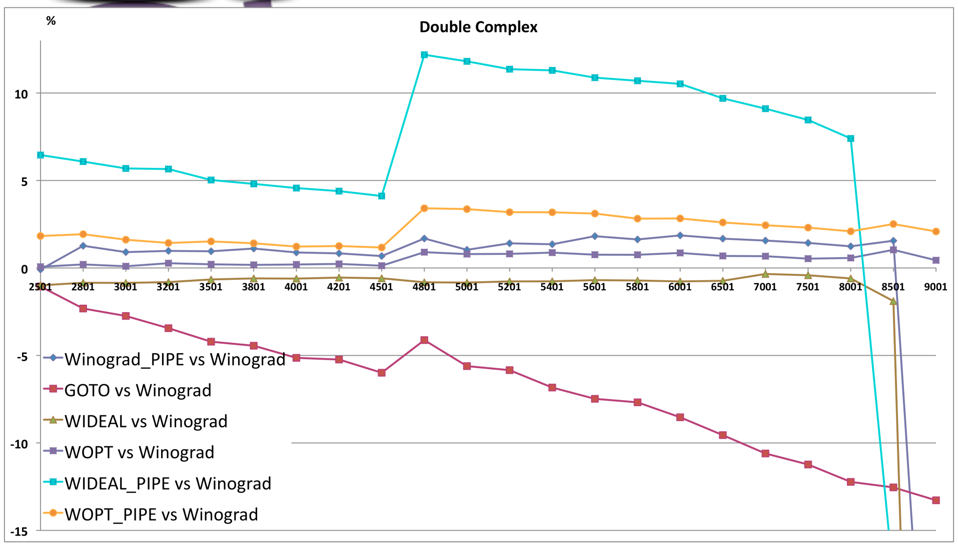
TABLE VI

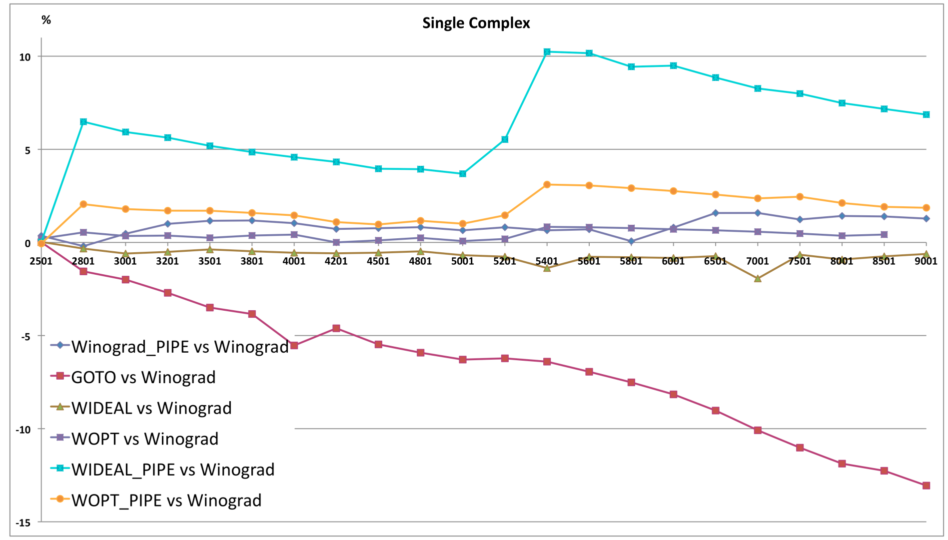
WINOGRAD'S MM (NO MAS IN THE CRITICAL PATH OF C=AB)

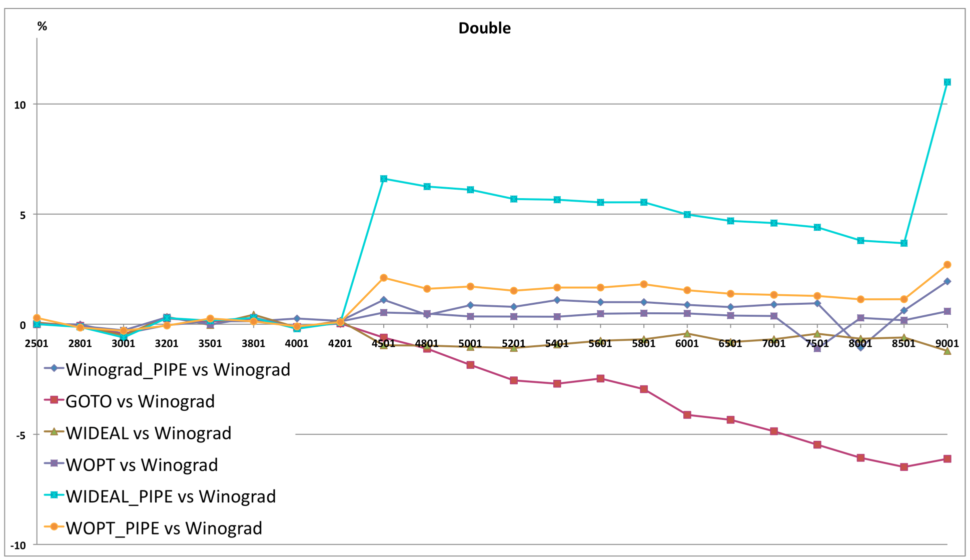
Sequential		Parallel/Pipelining			
$U = A_1B_2$	1:	$U = A_1B_2$			
$C_0 = A_0B_0$	2:	$C_0 = A_0B_0$	$S = A_3 - A_2$	$T = B_3 - B_2$	
$S = A_3 - A_2$					
$T = B_3 - B_2$					
$C_3 = ST$	3:	$C_3 = ST$	$C_0 \mathrel{+}= U$	$V = S + A_1$	$Z = T + B_1$
$C_0 \mathrel{+}= U$					
$V = S + A_1$					
$Z = T + B_1$					
$U \mathrel{+}= VZ$		$U \mathrel{+}= VZ$	$S = A_3 + A_1$	$T = B_0 - Z$	
$S = A_3 + A_1$					
$T = B_0 - Z$					
$C_2 = A_2T$	5:	$C_2 = A_2T$	$Z = B_3 + B_1$	$C_1 = U - C_3$	
$Z = B_3 + B_1$					
$C_1 = U - C_3$					
$U \mathrel{-}= SZ$	6:	$U \mathrel{-}= SZ$	$V = A_0 - V$		
$V = A_0 - V$					
$C_1 \mathrel{+}= VB_1$	7:	$C_1 \mathrel{+}= VB_1$	$C_3 \mathrel{-}= U$	$C_2 \mathrel{-}= U$	
$C_3 \mathrel{-}= U$					
$C_2 \mathrel{-}= U$					

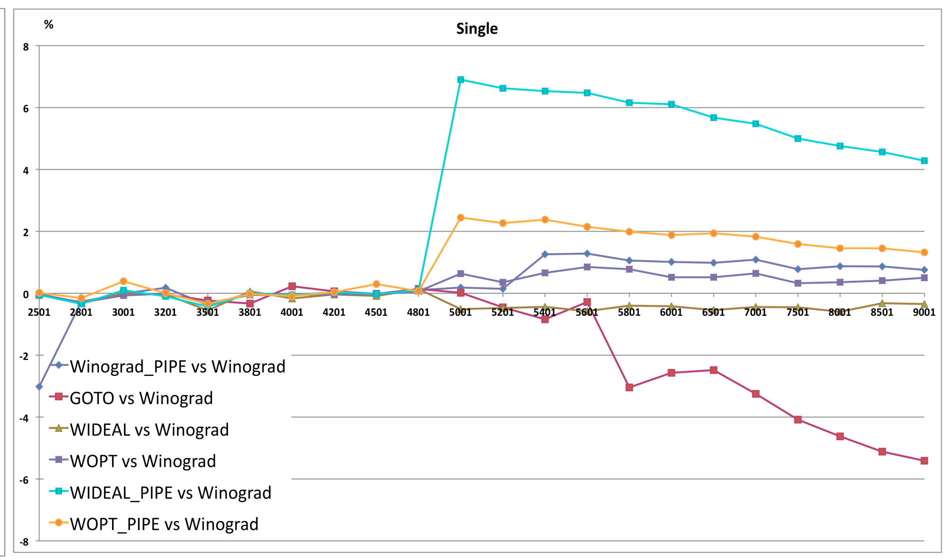


Algorithm + Performance Nehalem

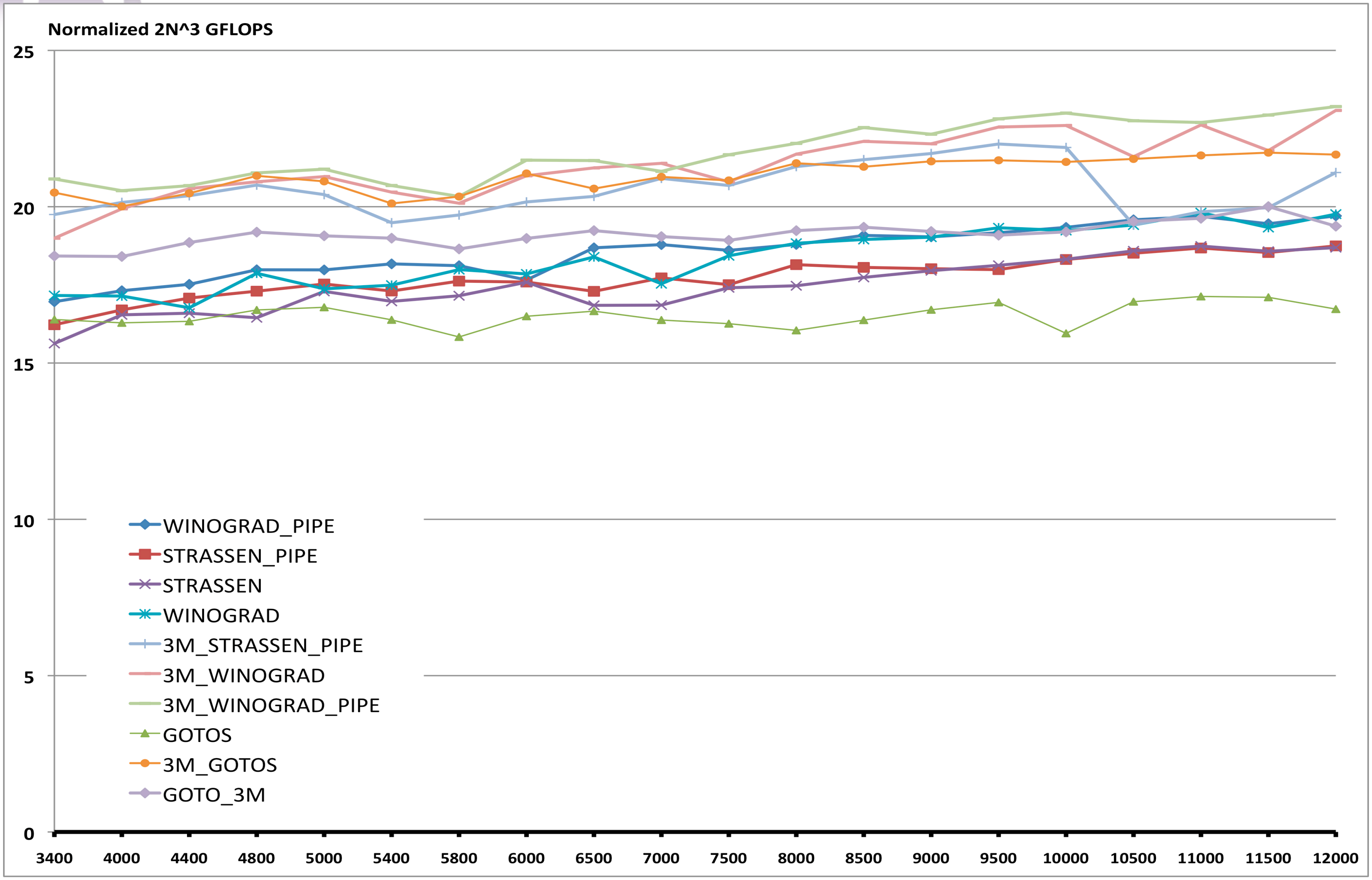




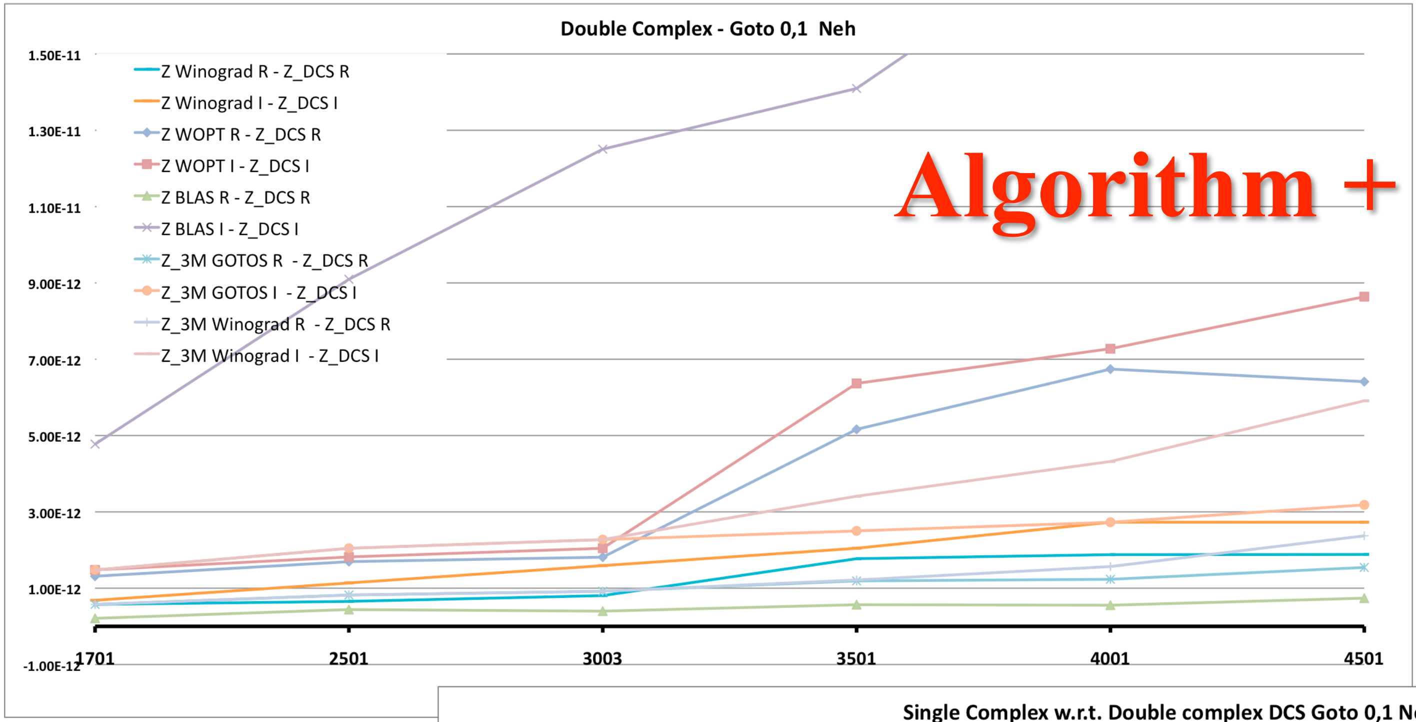


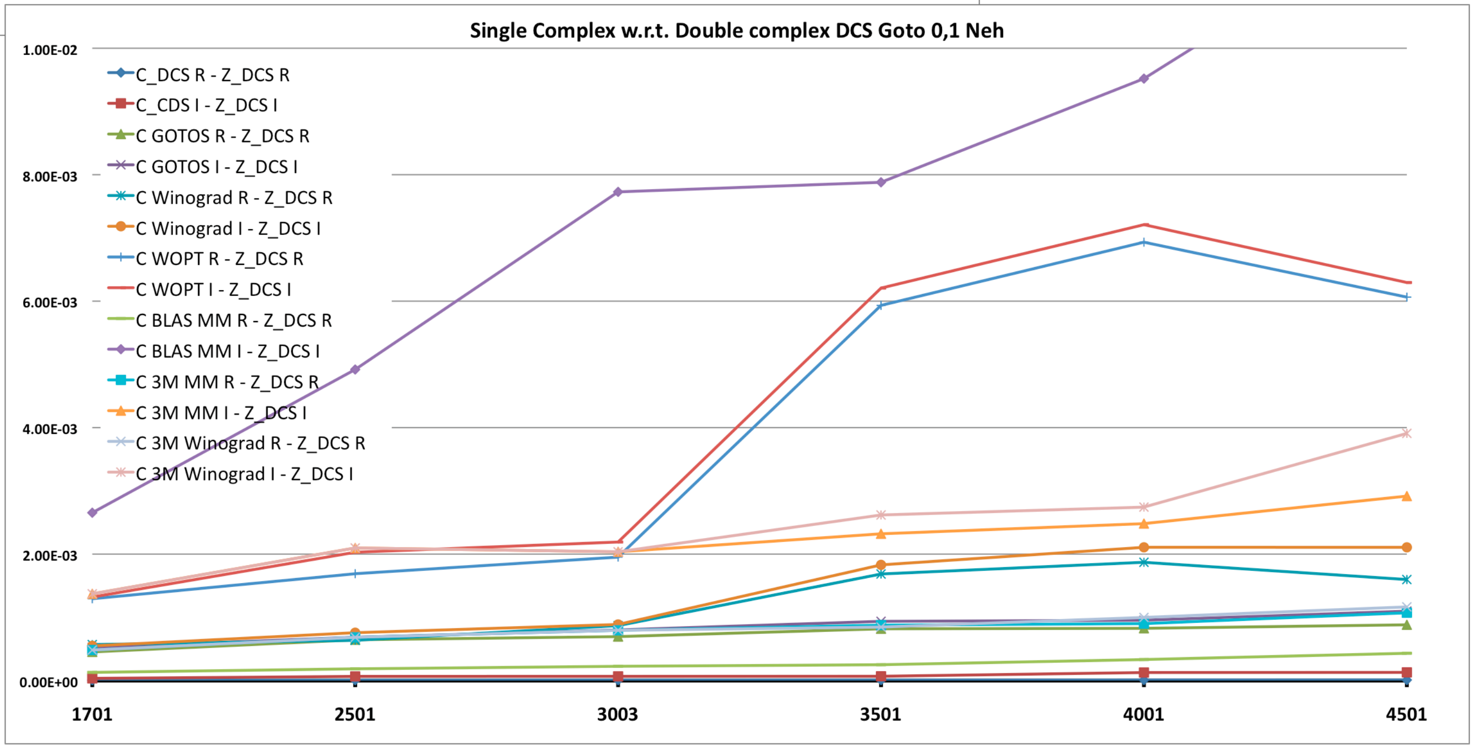


Algorithm + Performance Shanghai



Algorithm + Accuracy





Conclusions

- **Though there is no dominant algorithm**

1. We have an arsenal of algorithms
   - *We can fit to the occasion*
2. We have algorithm optimizations
   - *We can fit to the system*
3. Neglecting these, we may lose up to 30% performance
   - *On average, the accuracy is not too bad*

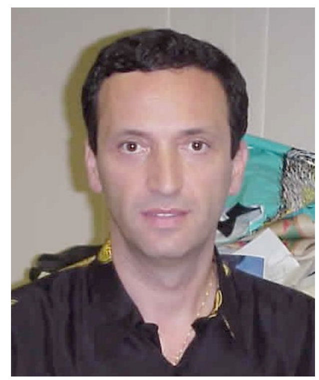
Future Works and Collaboration

- **Algorithm implementation and choice done automatically**
  - Expand the set of fast algorithms
  - Similar to what has been done for FFT
  - Automate the process and development of hybrids methods
- **Numerical correction**
  - Discover, develop, and deploy techniques for error reduction

Thank you

Marco Bodrato

Paolo D'Alberto



Alex Nicolau