

# An Optimizing Compiler for Parallel Chemistry Simulations



J. Cao, A. Goyal,

S. Midkiff and J. M. Caruthers

*{caruther,smidkiff}@purdue.edu*

Schools of ECE and Chemical Engineering

Purdue University

# Why develop a chemistry compiler?



---

- An important goal in chemistry research is the development of kinetic reaction models.
  - These models are systems of ODEs with  $\sim 250,000$  terms
  - Manual development of smaller models takes months
  - Large size and long development times lead to many errors
  - Models are often too large to be compiled with commercially available compilers
- Computationally solving a model requires weeks of CPU time



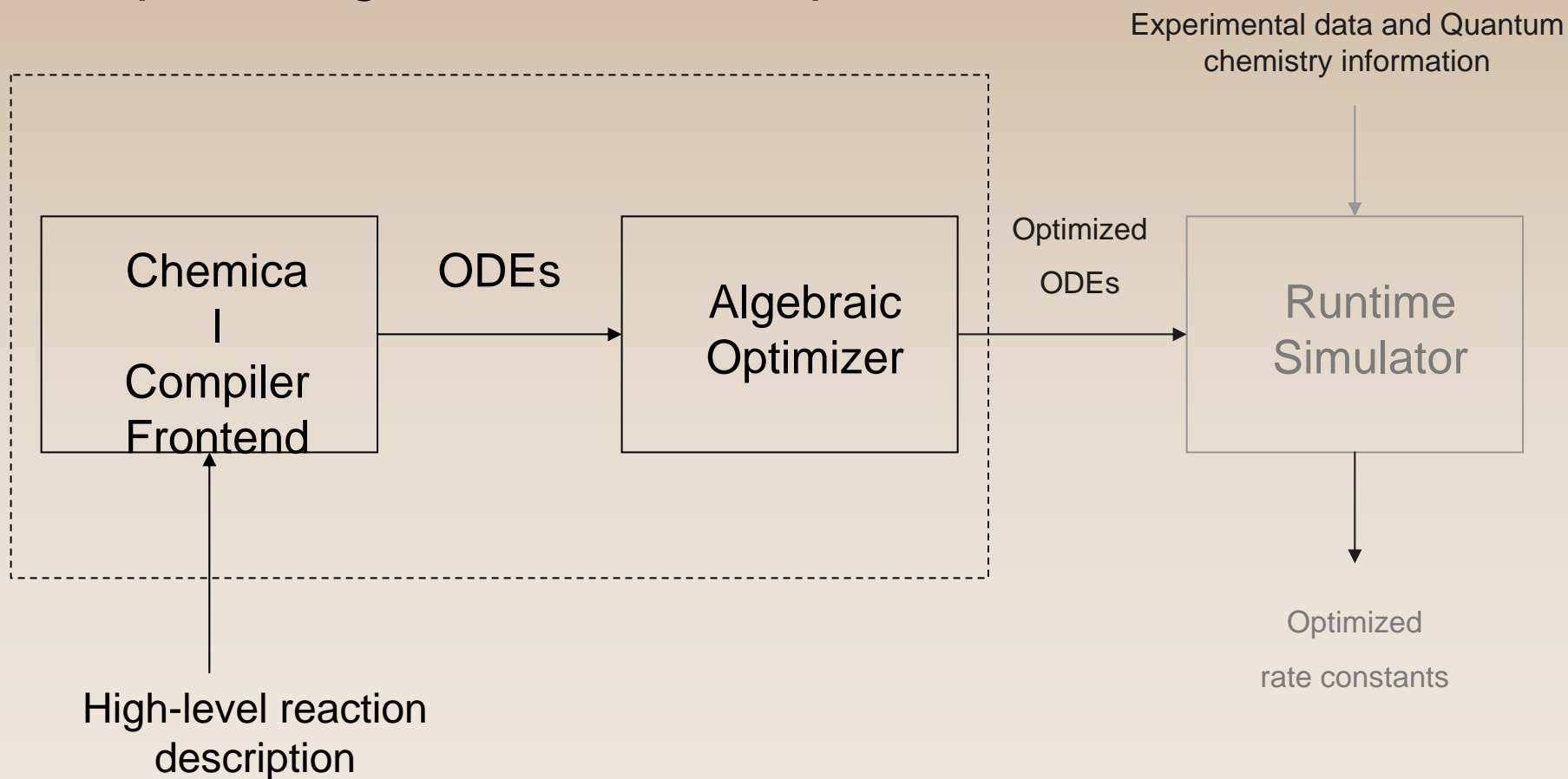
# Solution

---

- An optimizing chemical compiler to make describing a model easy
  - Approximately 30 lines of input to describe reaction
  - Hours to formulate rather than days
  - Output is system of ODEs and driver code to optimize them - typically 300 ODEs of ~1000 terms each
- Algebraic optimizations to simplify the ODEs that represent the model
  - Enables compilation to binary code
  - Reduces the execution time for optimizing the model

# The optimizing chemical compiler

## Optimizing Chemical Compiler



# Algebraic optimizations targeting the ODEs

- Regular, predictable structure of the ODEs allow them to be targeted with fast, efficient, specialized optimizations
  - Single assignment of all variables
  - A, B, ... representing molecules defined once before the ODE solved
- Specialized optimizations perform better than general purpose optimizations provided by commercial compilers for our problem
- Memory limitations prevent commercial compilers from working on basic blocks of 100s of thousands of terms

*Initialization of A, B, ..., ki*

$$dA/dt = k1*A*B+k2*D*E+...$$

$$dB/dt = -k1*A*B+k2*D*E+...$$

$$dD/dt = -k2*D*E+...$$

$$dE/dt = -k2*D*E+...$$

...

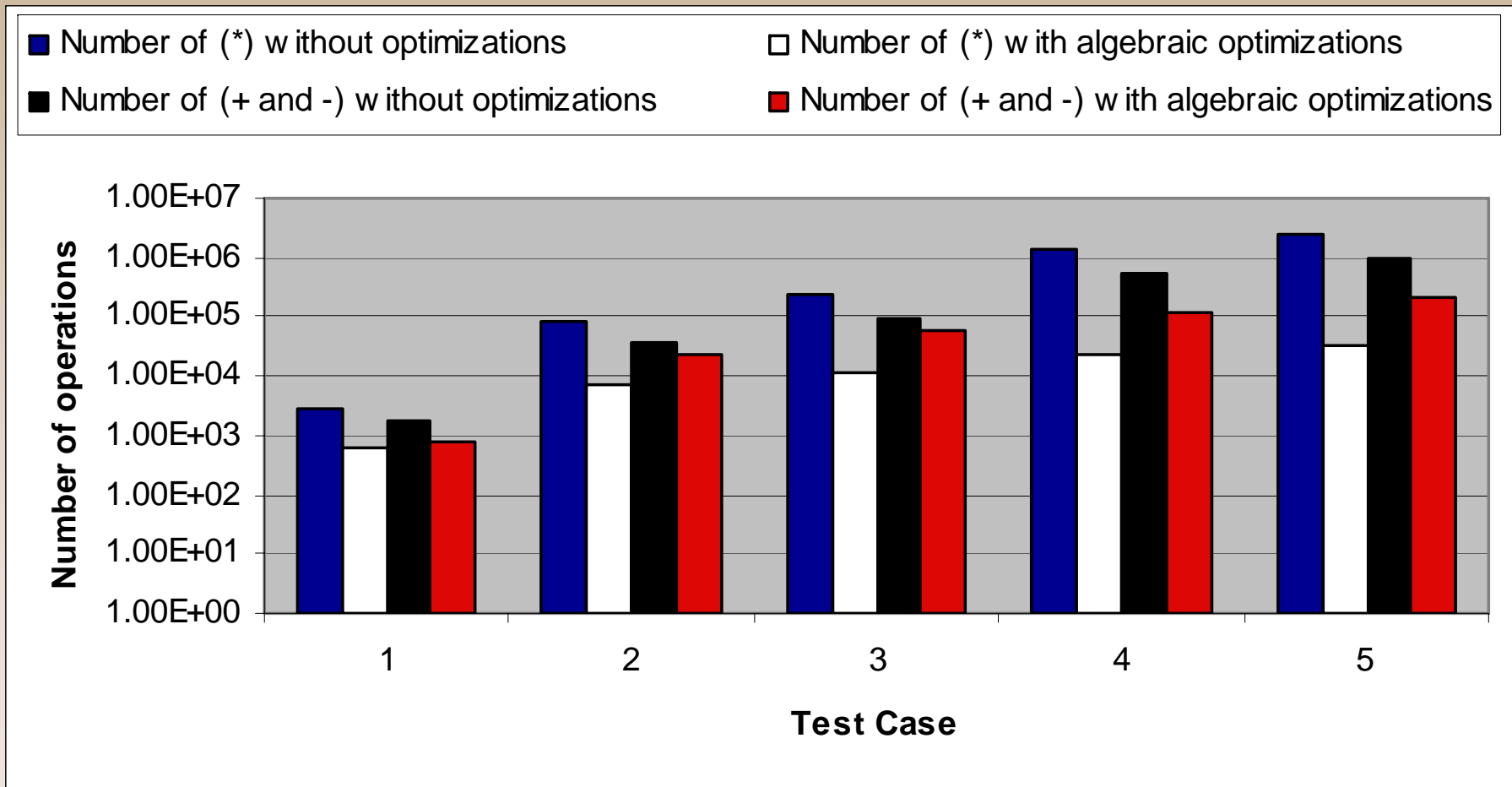
***Example of produced code***

# Constant Folding, Distributive Optimization and CSE

- *Constant folding*
  - **Input:**  $2*k*B*C + 3*k*B*C$
  - **Output:**  $5*k*B*C$
- *Distributive optimization*
  - **Input:**  $+k*B*C + k*B*D + k*E*F +$
  - **Intermediate result:**  $+k*(B*C + B*D + E*F) +$
  - **Final output:**  $... + k*(B*(C + D) + E*F) +$

- *CSE:*
  - **Input:**  
 $... + (A + B + C + D) * k1 * E + ...$   
 $... + (A + B + C + D) * k2 * F + ...$   
 $... + (A + B + C) * k3 * G + ...$   
 $... + (A + B + C) * k4 * H + ...$
  - **Output:**  
 $temp[0] = A + B + C;$   
 $temp[1] = temp[0] + D;$   
 $... + temp[1] * k1 * E + ...$   
 $... + temp[1] * k2 * F + ...$   
 $... + temp[0] * k3 * G + ...$   
 $... + temp[0] * k4 * H + ...$

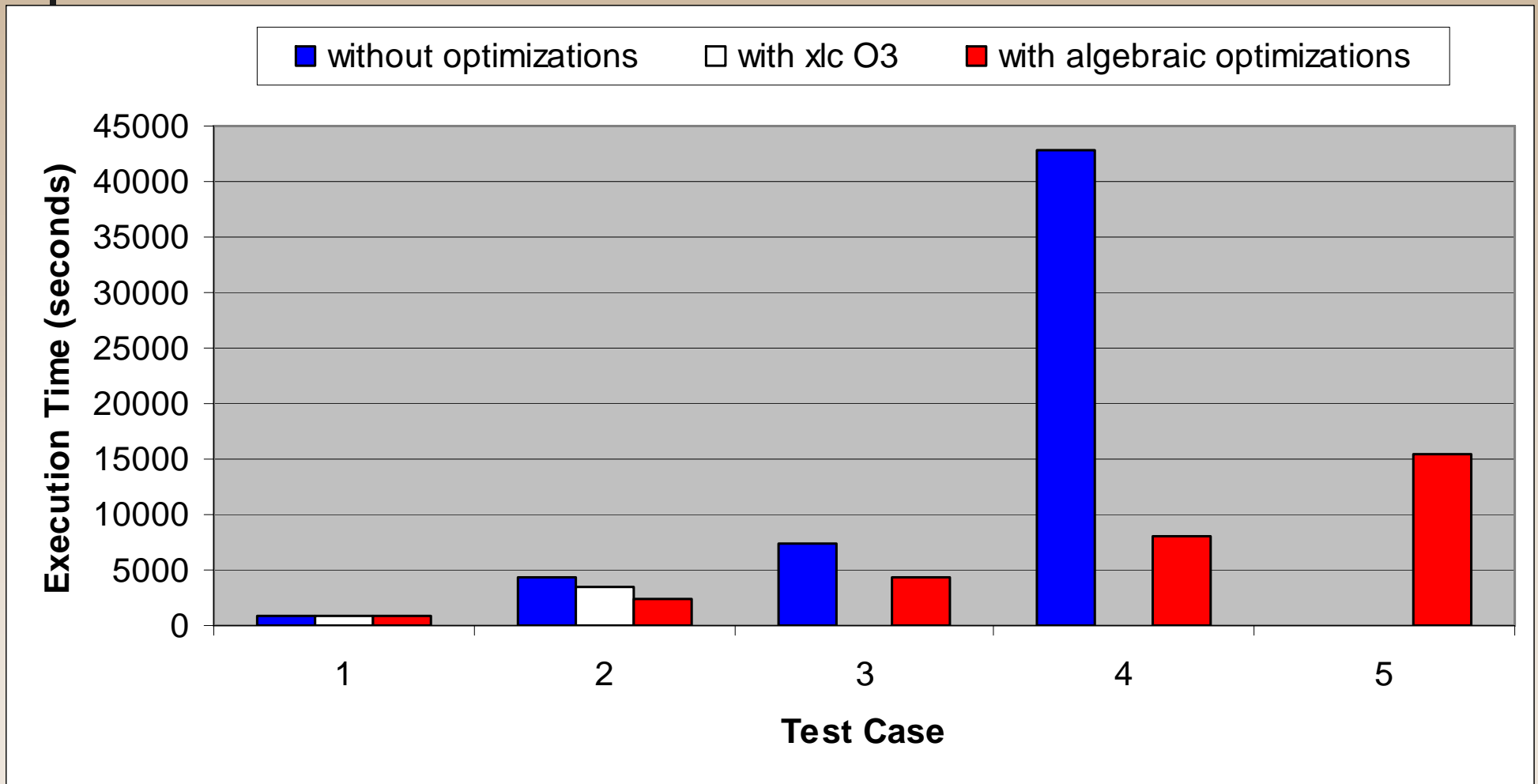
# Effectiveness of Optimizations in removing operations



Lower is better, note log scale on the y axis

# Benefit of Algebraic Optimizations

*IBM SP with Winterhawk II nodes*





# Conclusions

---

- The time to create a kinetic model is reduced from weeks and months to a few days
- Chemistry compiler generates code creates large basic blocks with hundreds of thousands of terms
  - ODEs simplification via algebraic optimizations necessary
  - Simplification improves performance, allows compilation with off-the-shelf compilers
  - Simple optimizations enabled by domain-specific structure of computations
- ***Currently porting and performance tuning on Linux/Itanium system at Purdue ECE***