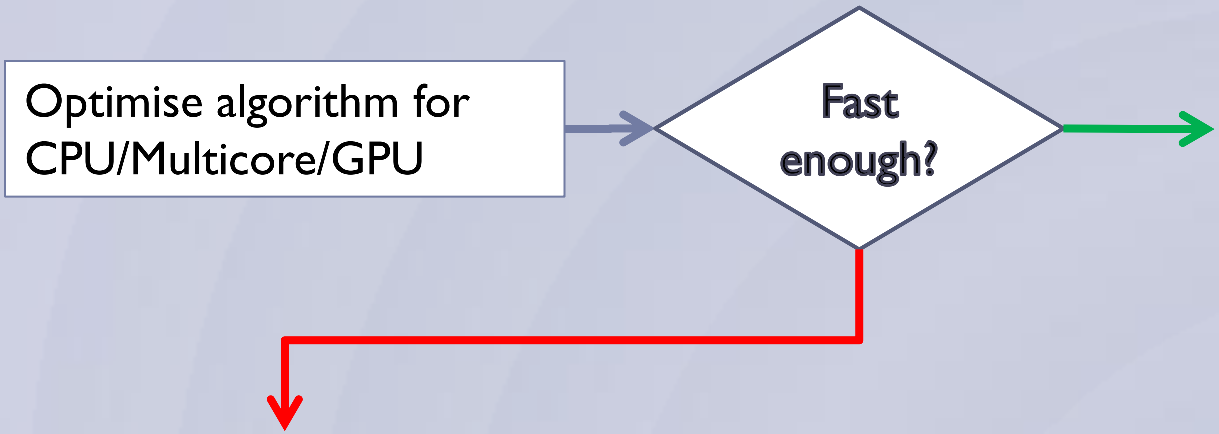
A Scalable Approach for Automated Precision Analysis

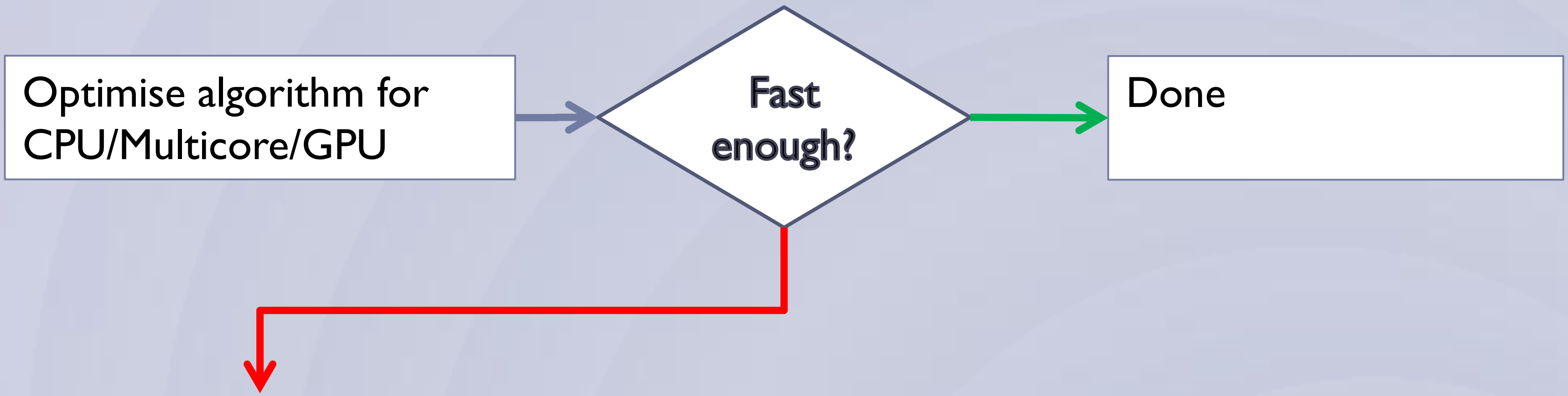
David Boland
and George A. Constantinides

Accelerating an application

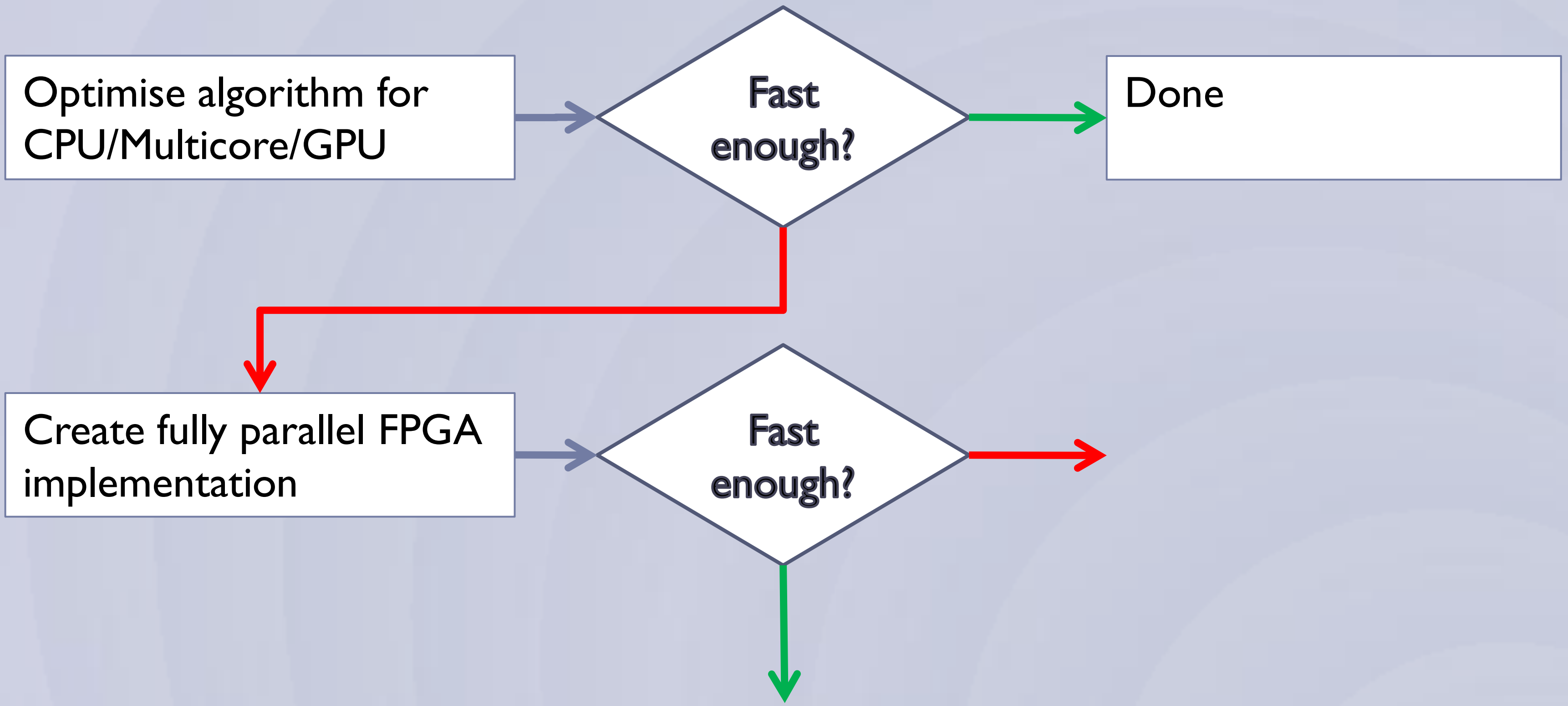
Accelerating an application



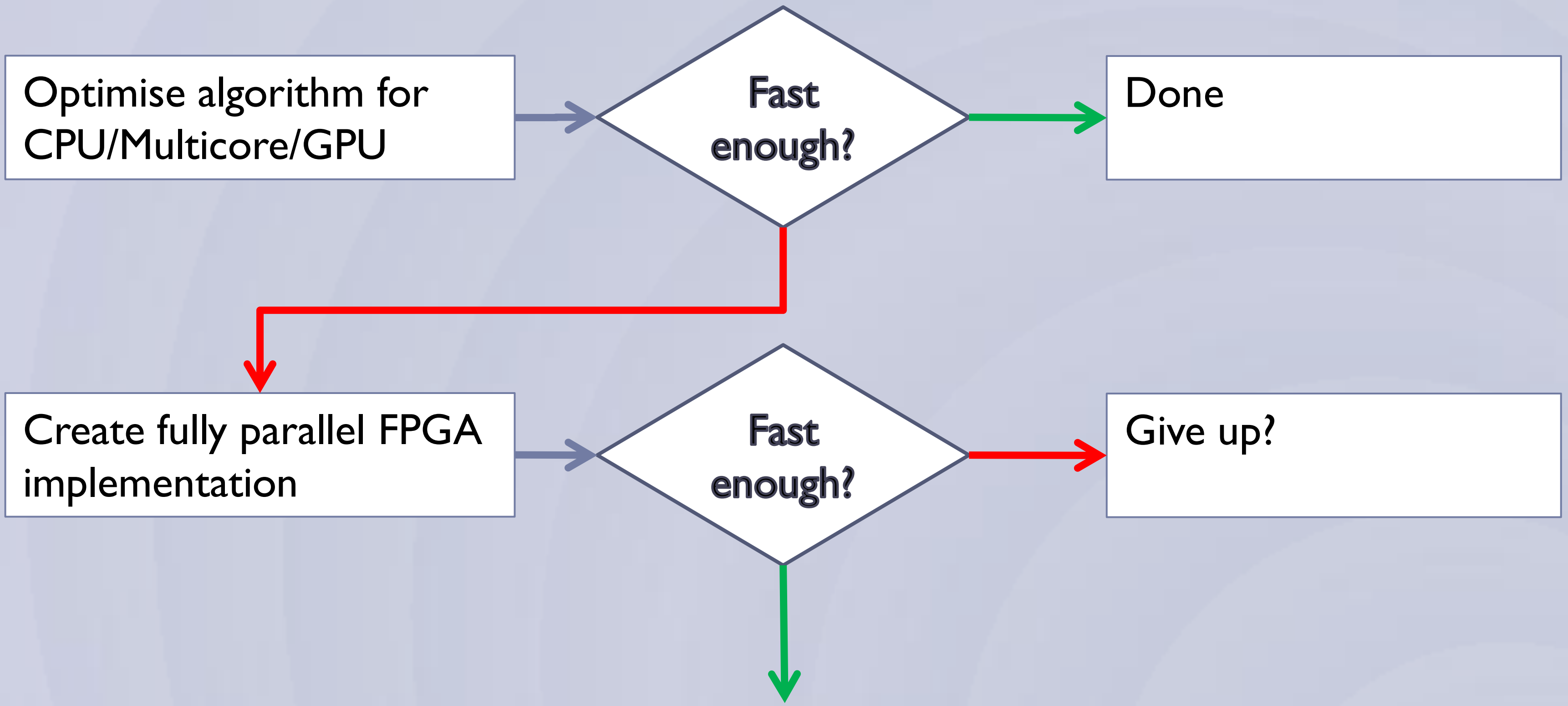
Accelerating an application



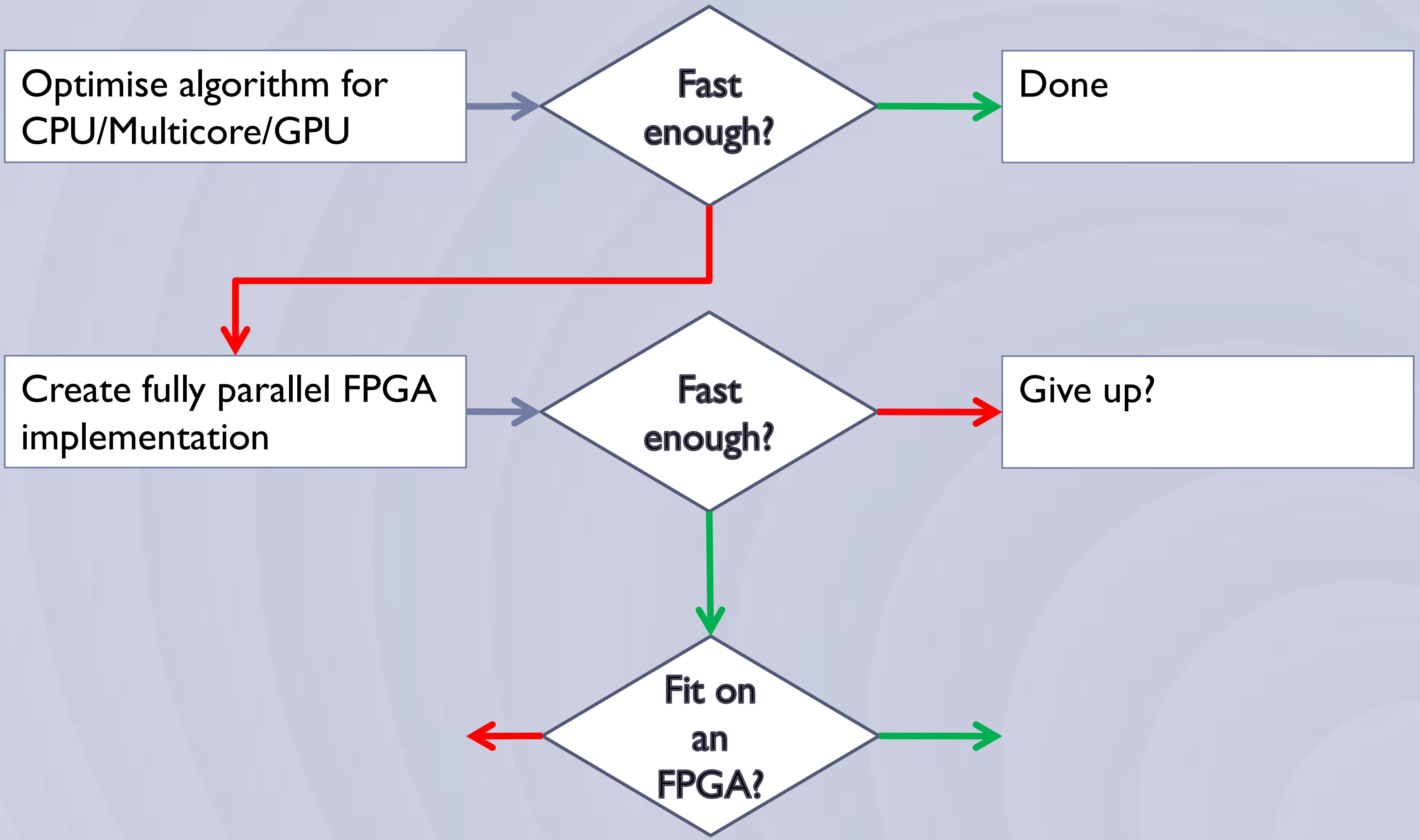
Accelerating an application



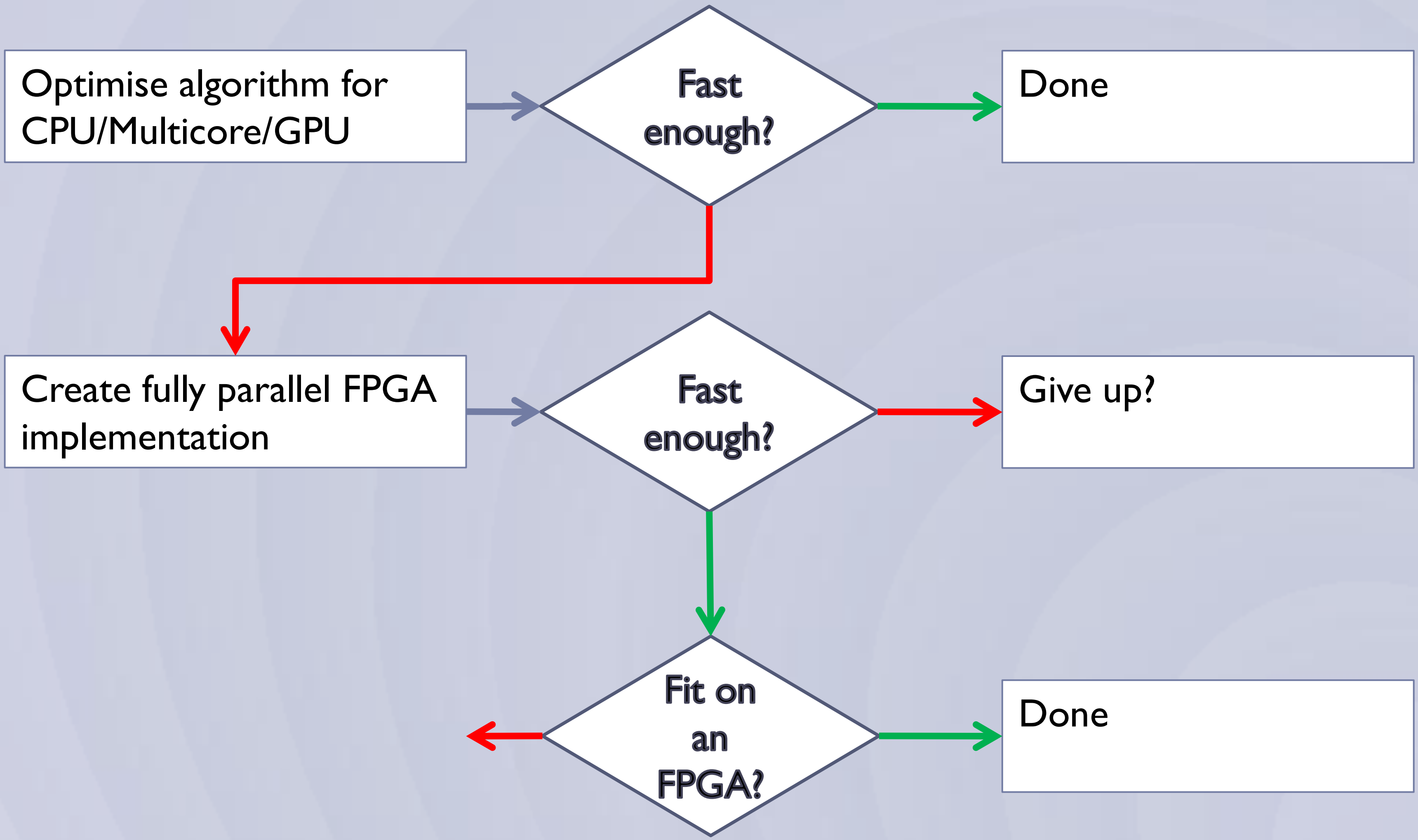
Accelerating an application



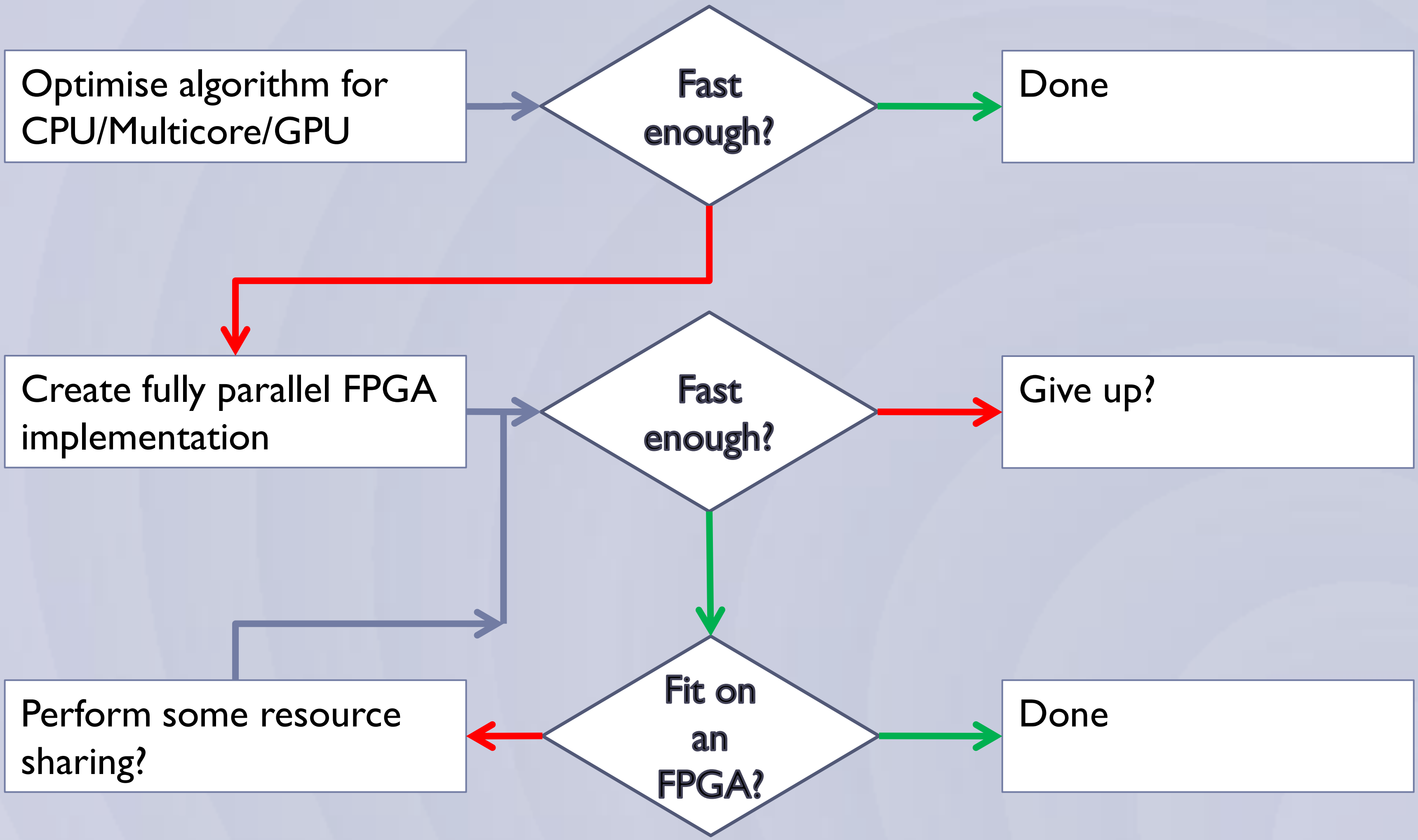
Accelerating an application



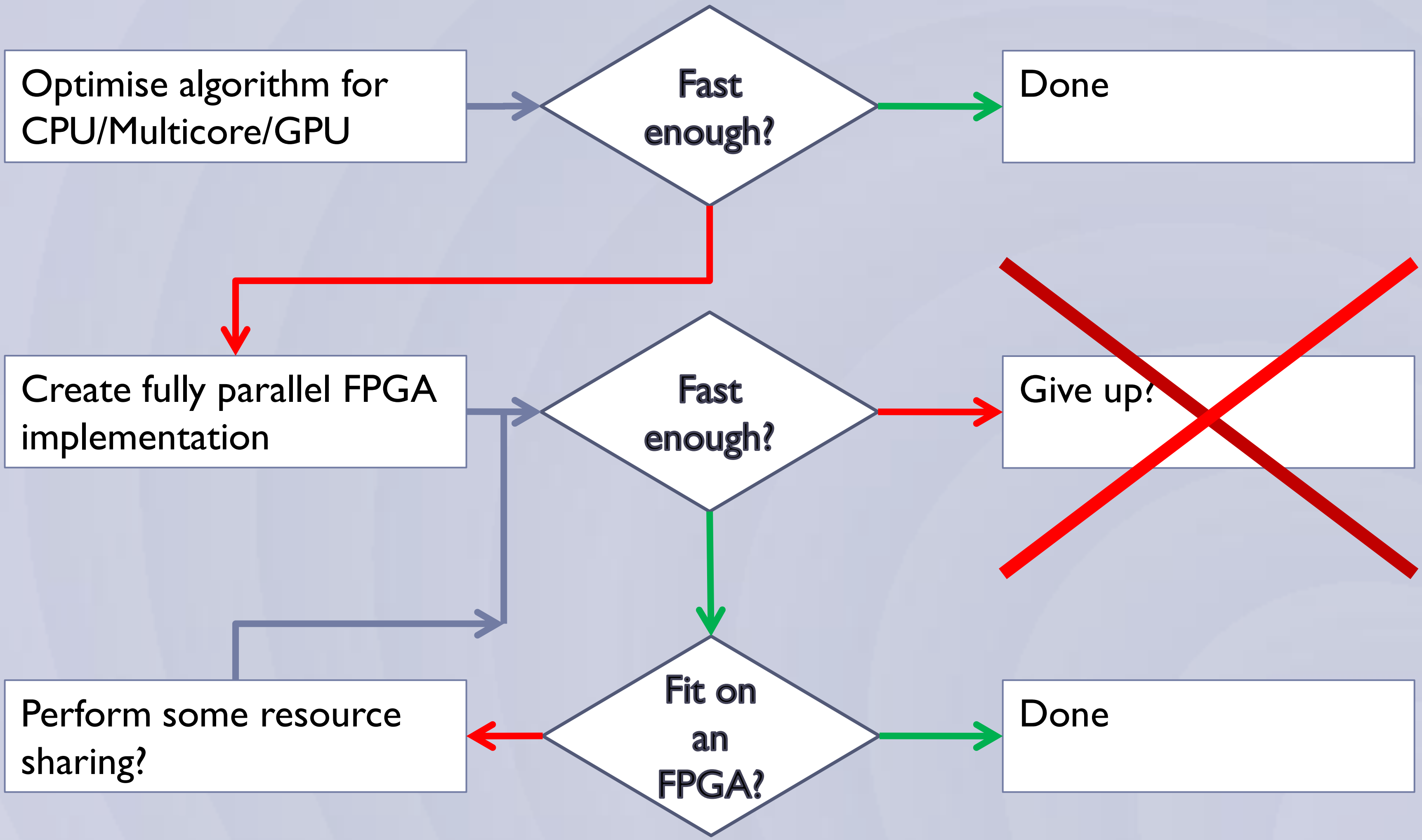
Accelerating an application



Accelerating an application



Accelerating an application



Word-length optimisation can be the game changer…

- Performance gain by moving from IEEE 754 double precision to single precision:
  - 2x for a CPU
  - 2-9x for a GPU
- FPGAs have much greater flexibility
  - Can implement any custom precision
    - Large performance trade-offs
    - Many factors affected
      - Silicon area
      - Clock speed
      - Latency
      - Memory use
      - Data transfer overhead

So why don’t we perform word-length optimisation?

- Reducing word-length can cause errors
  - Overflow error
  - Accumulation of individual round-off errors

So why don't we perform word-length optimisation?

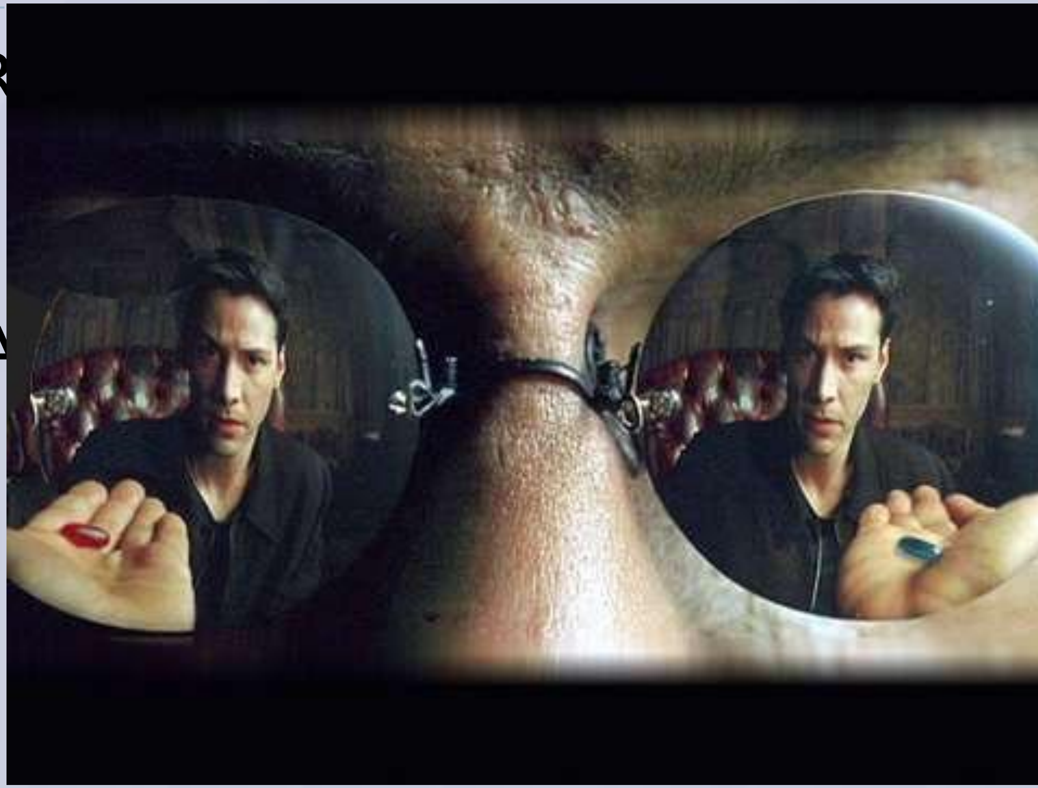
- Reducing word-length can cause errors
  - Overflow error
  - Accumulation of individual round-off errors
- Allows 'fair' comparison versus software

So why don't we perform word-length optimisation?

- Reducing word-length can cause errors
  - Overflow error
  - Accumulation of individual round-off errors
- Allows 'fair' comparison versus software
  - Lazy (& incorrect?) comparison
    - $(a+b)+c \neq a+(b+c)$

So why don't we perform word-length optimisation?

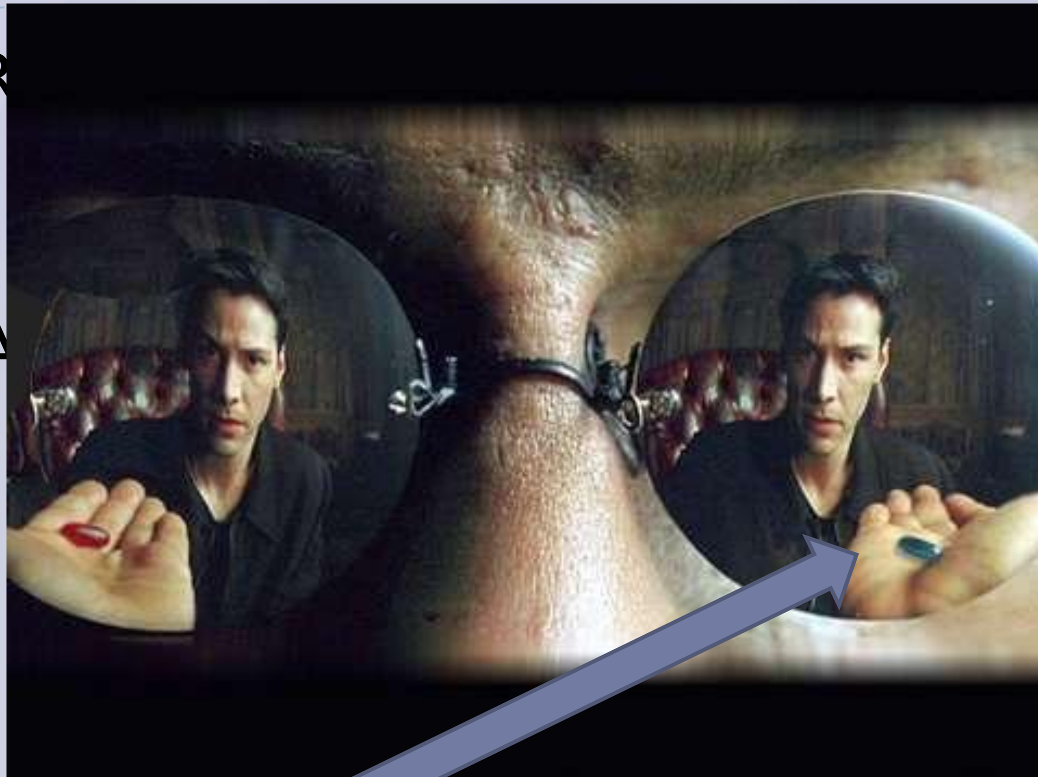
- R … s
- A … re



Do you want to care about numerical issues?

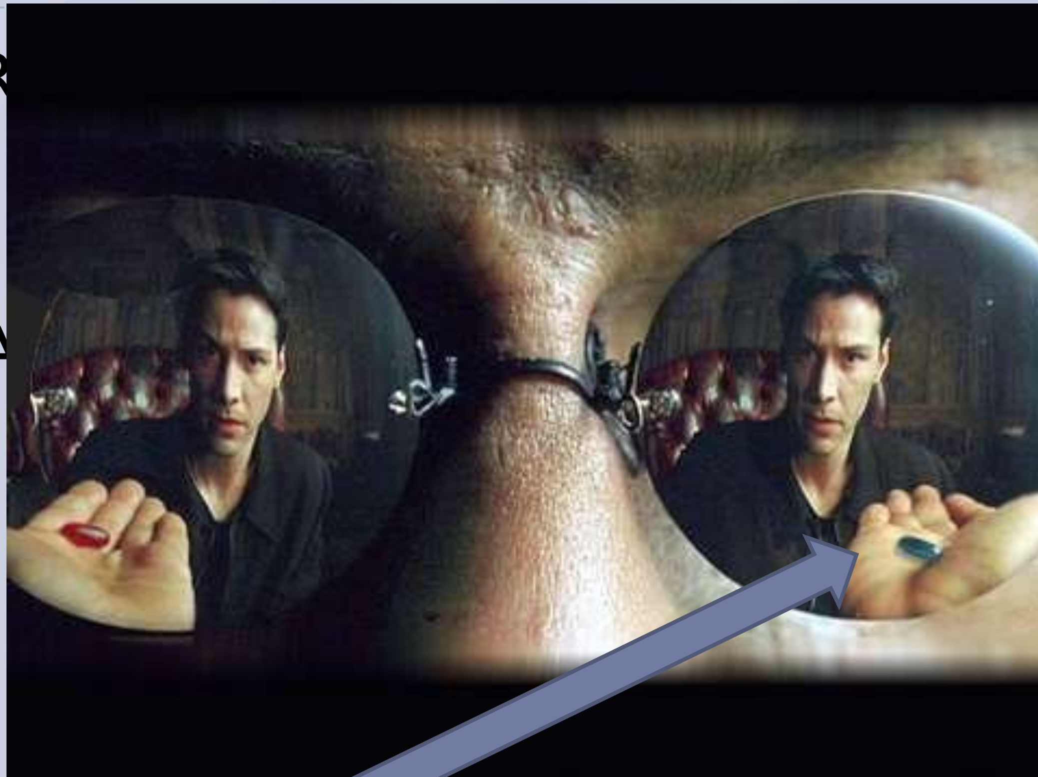
So why don't we perform word-length optimisation?

- R s
- A re



Do you want to care about numerical issues?

So why don't we perform word-length optimisation?



Do you want to care about numerical issues?

Report greater speed up factors by using IEEE single precision.

Ideal word-length optimisation

Algorithm being accelerated:

- Code

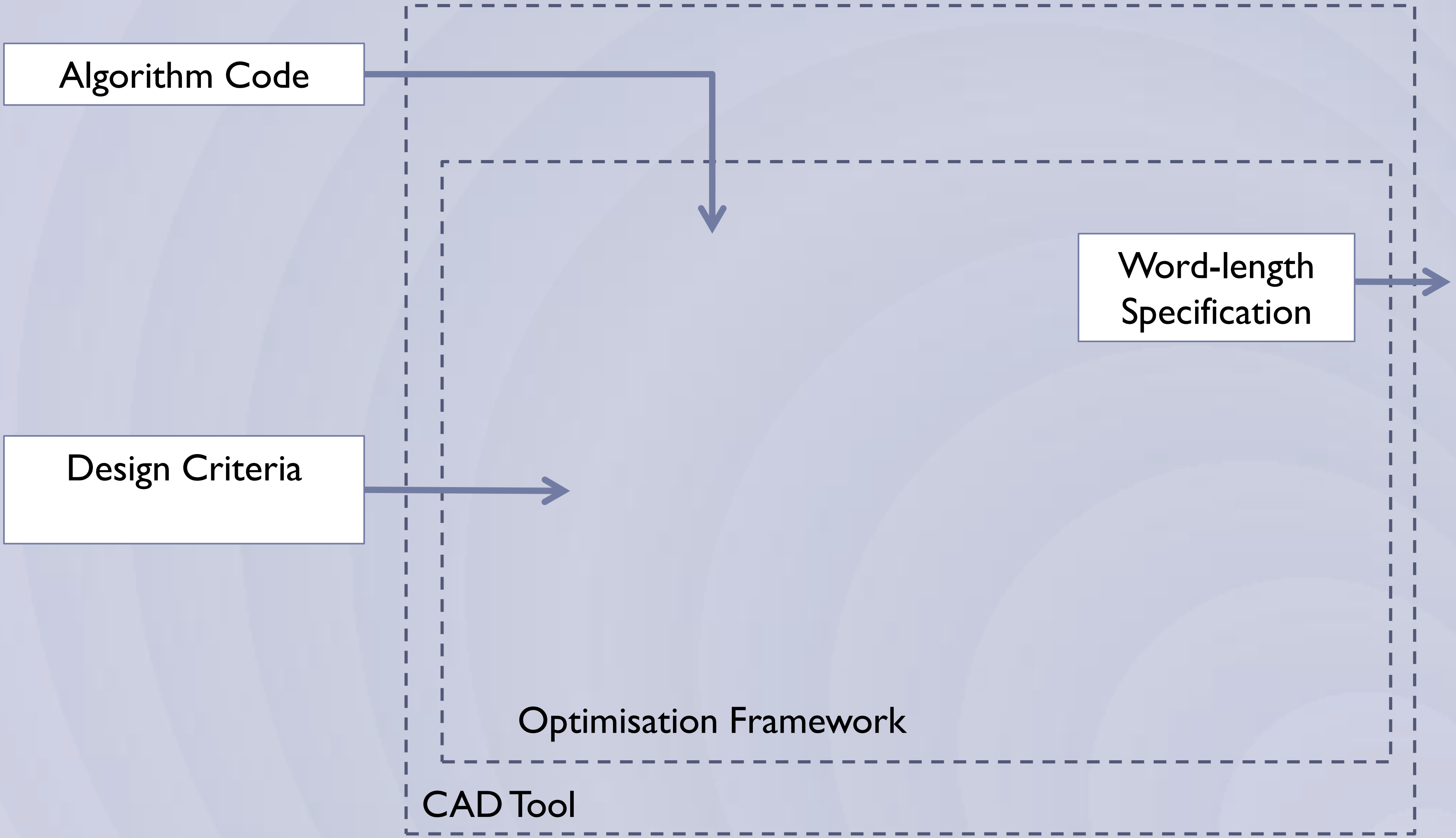
Design Criteria:

- Bound on Error
- Bound on Range

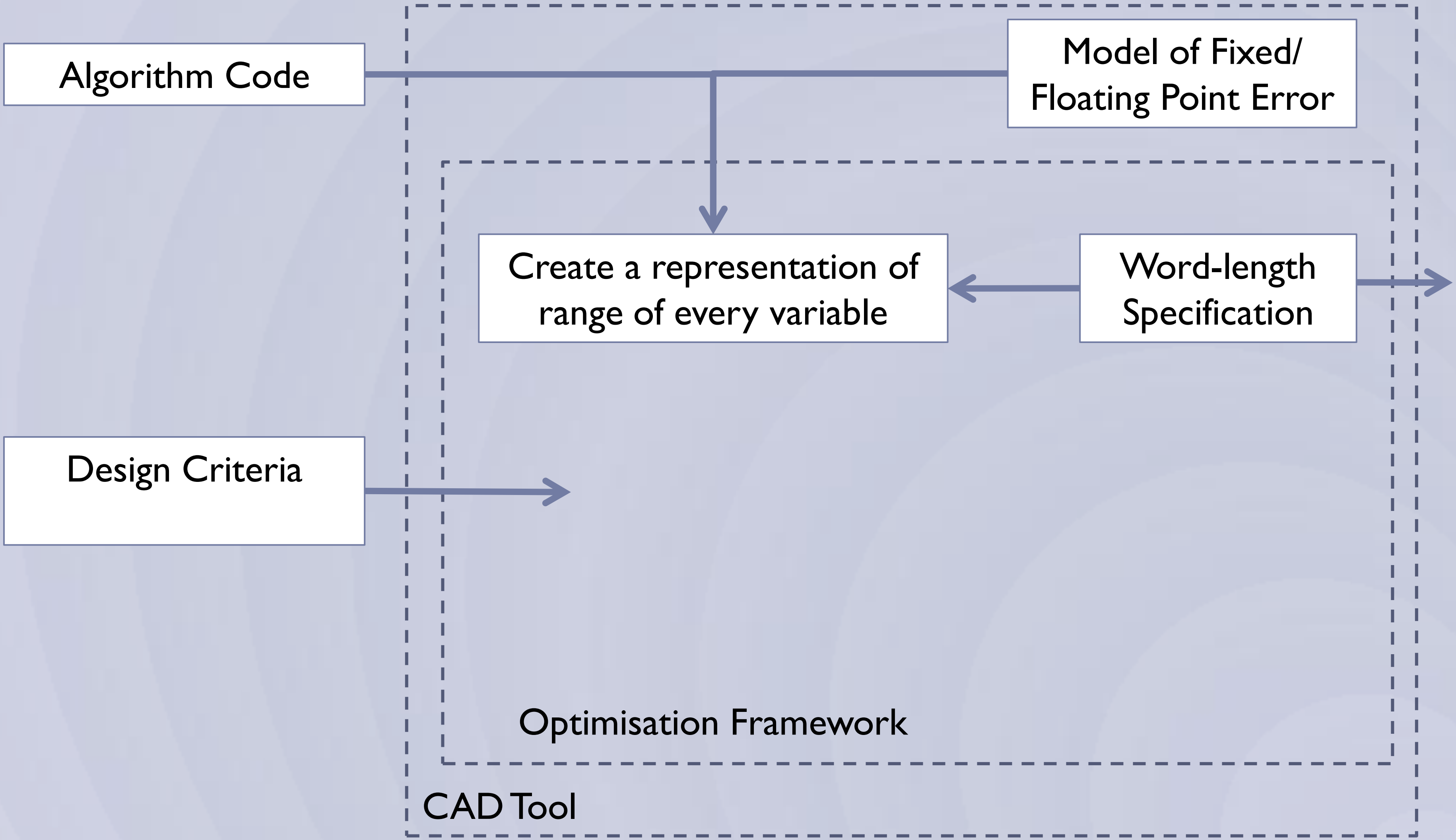
CAD Tool

Minimum word-length specification for every arithmetic operator in the hardware accelerator **guaranteed** to meet design criteria

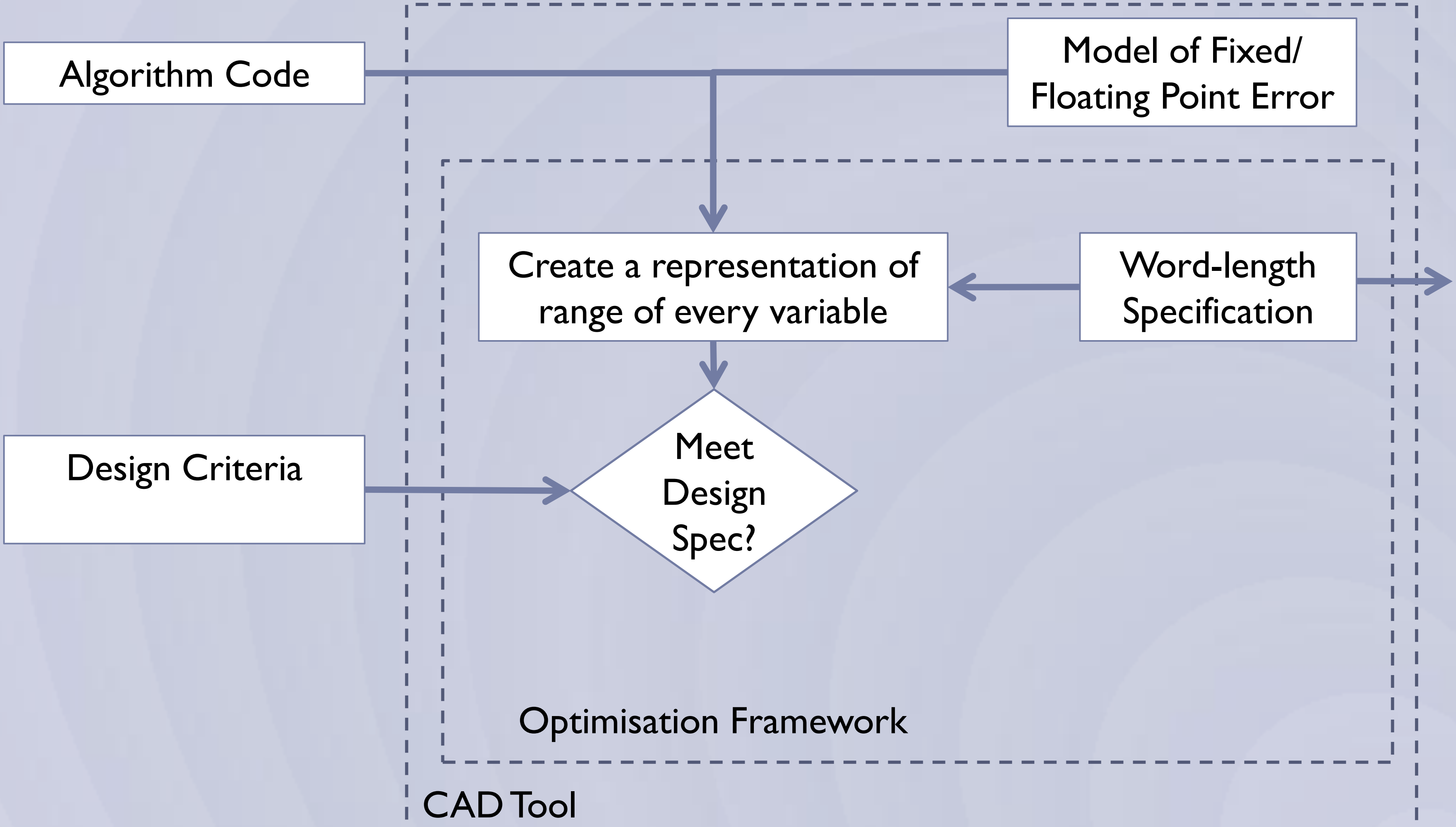
What are the main parts of this tool?



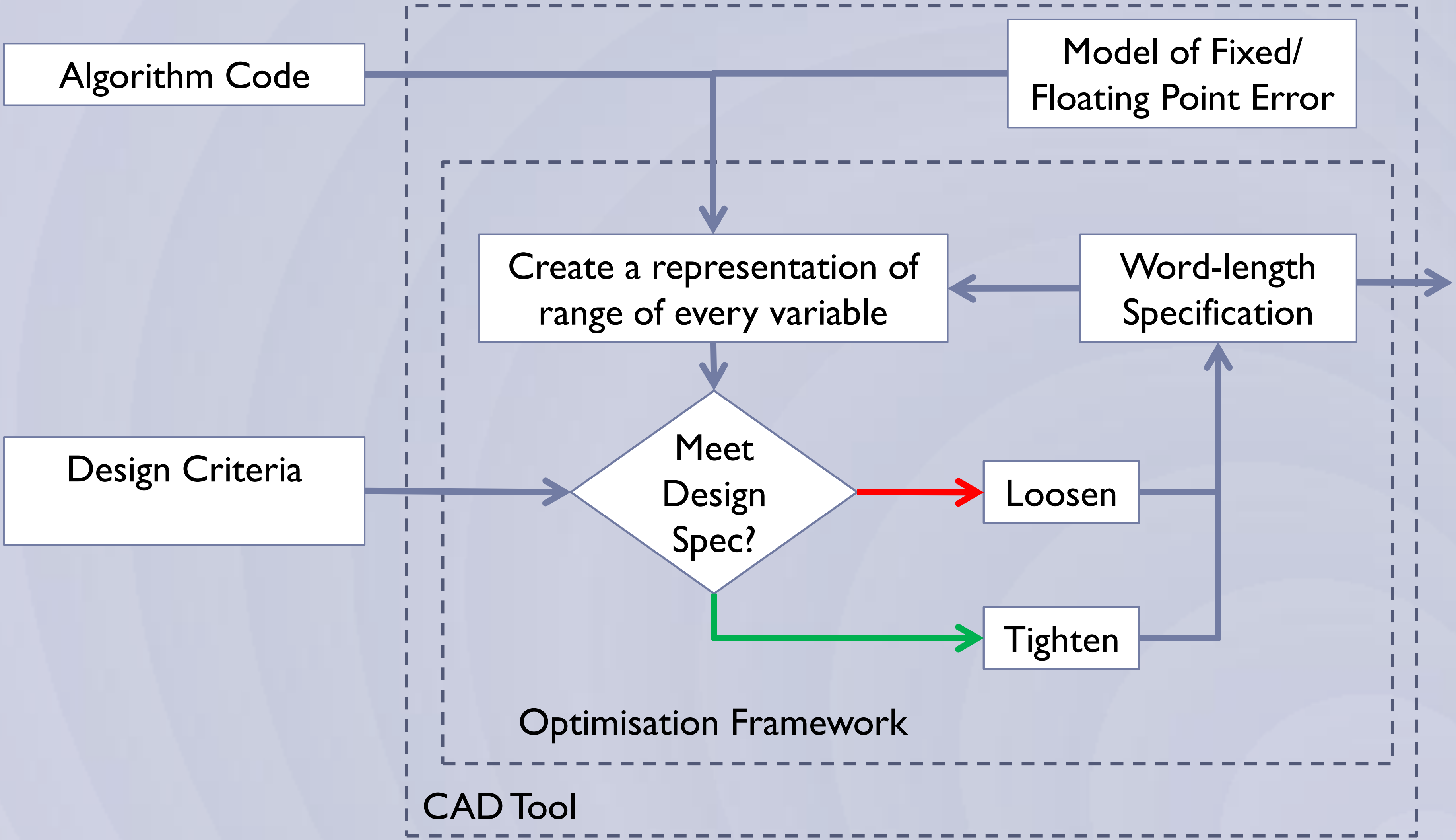
What are the main parts of this tool?



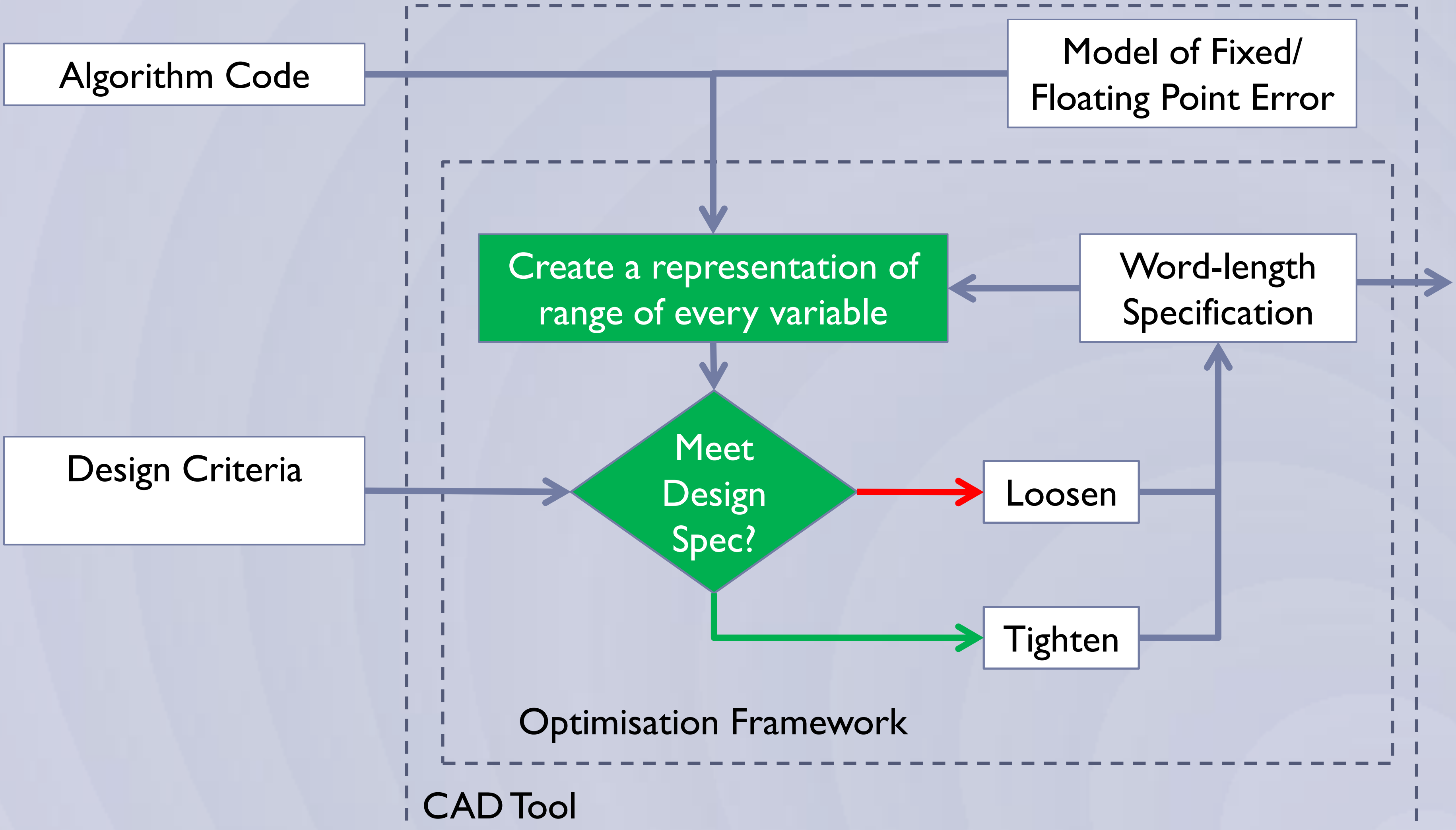
What are the main parts of this tool?



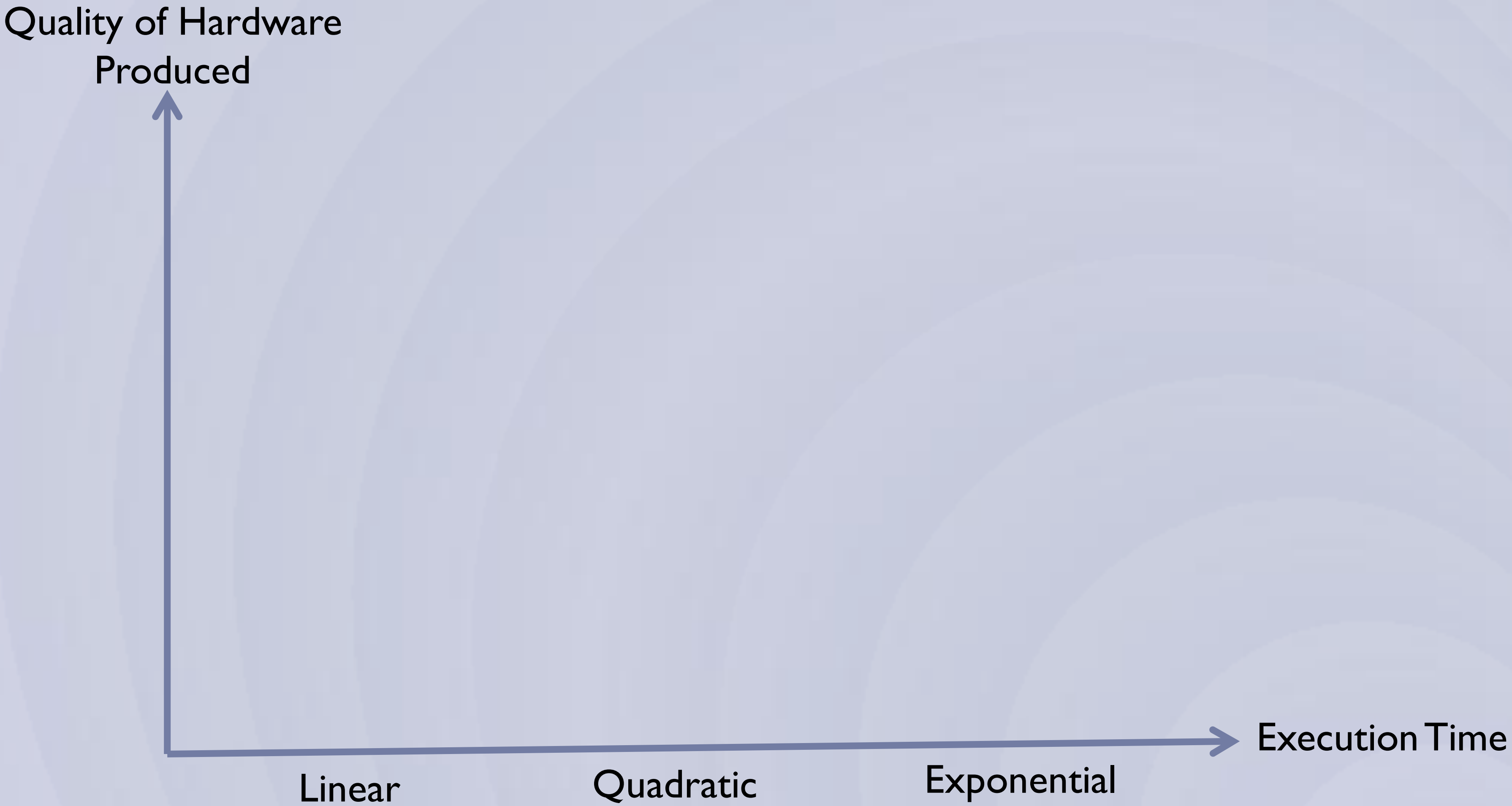
What are the main parts of this tool?



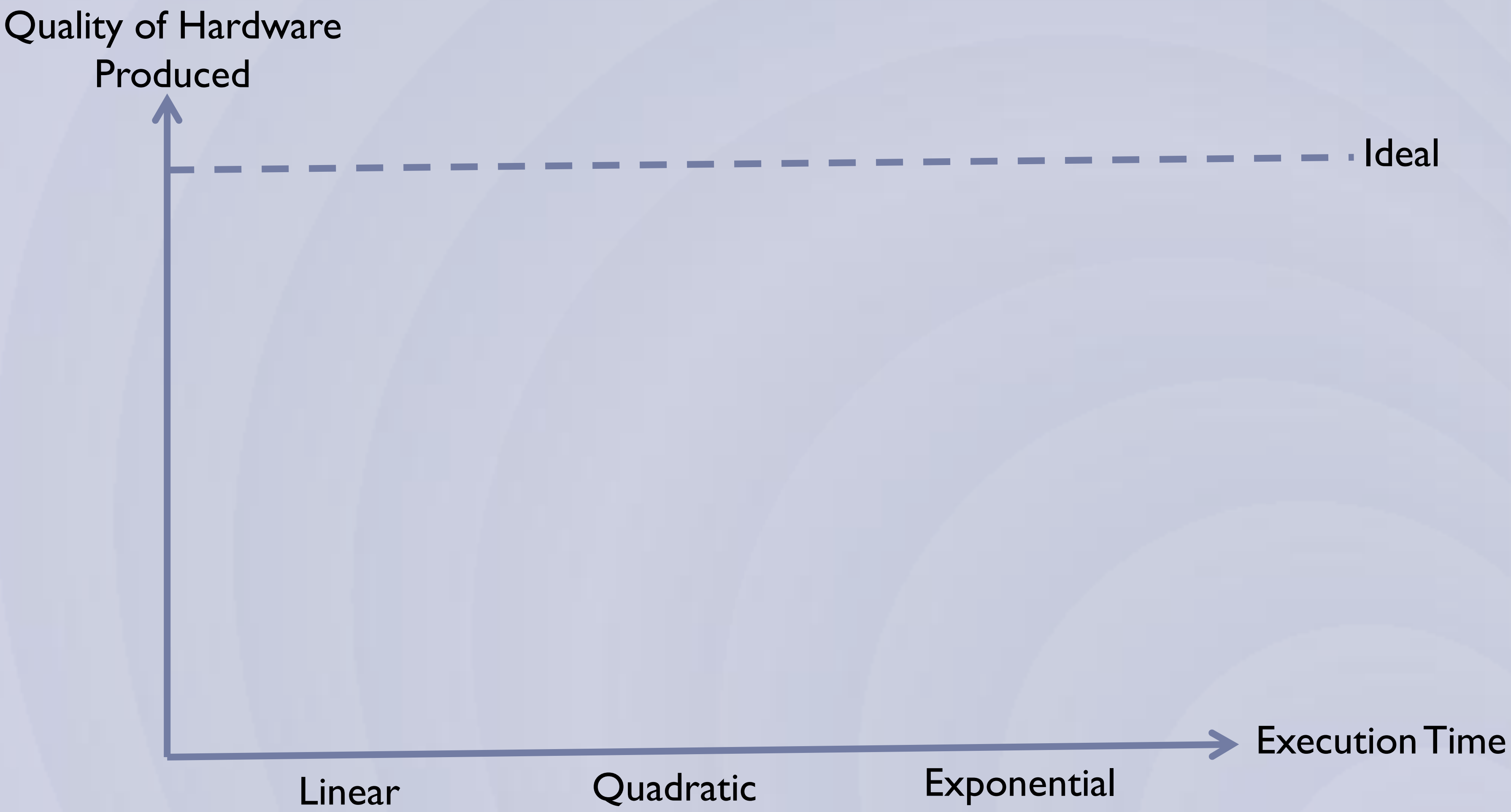
What are the main parts of this tool?



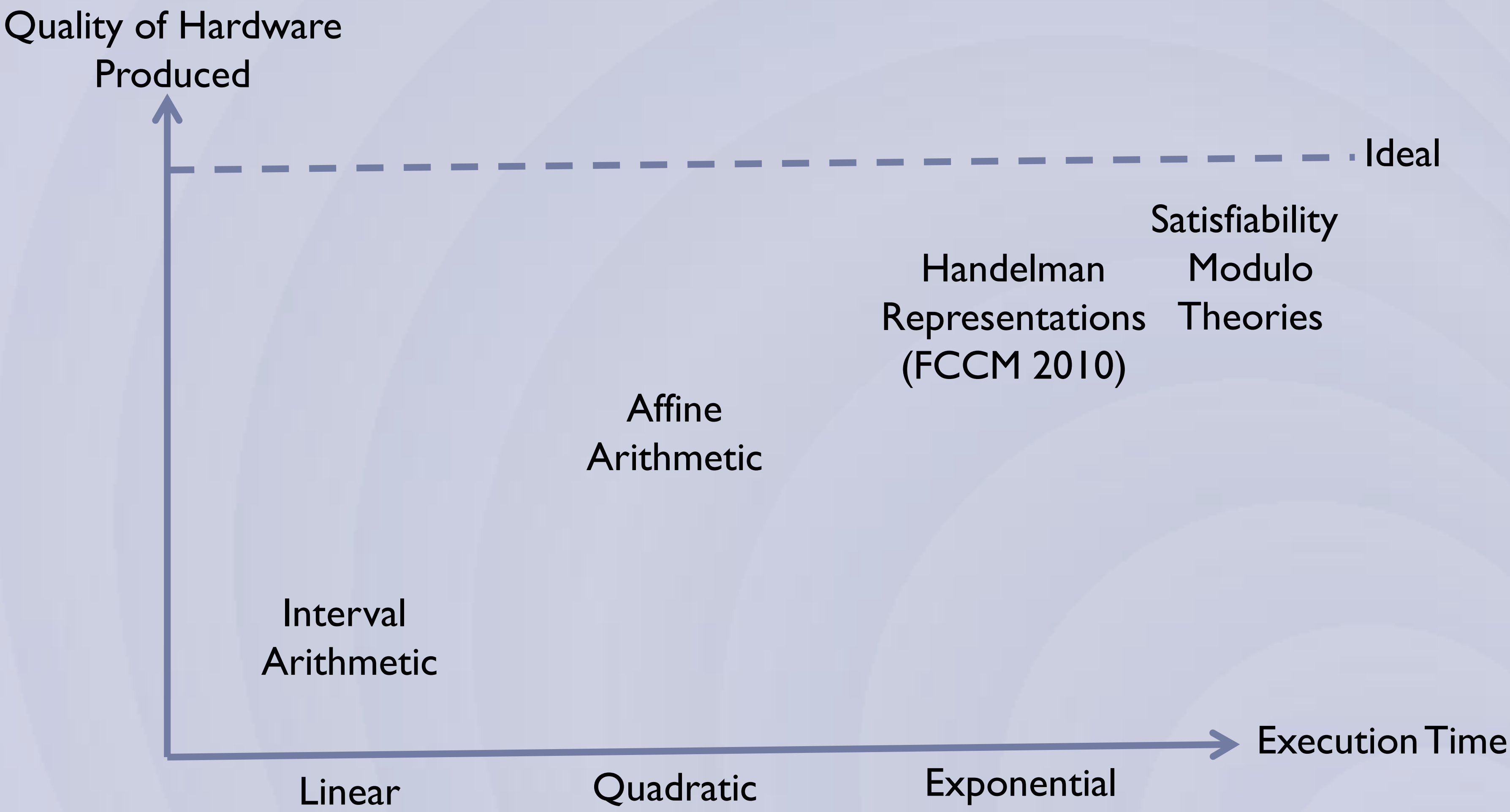
The battlefield



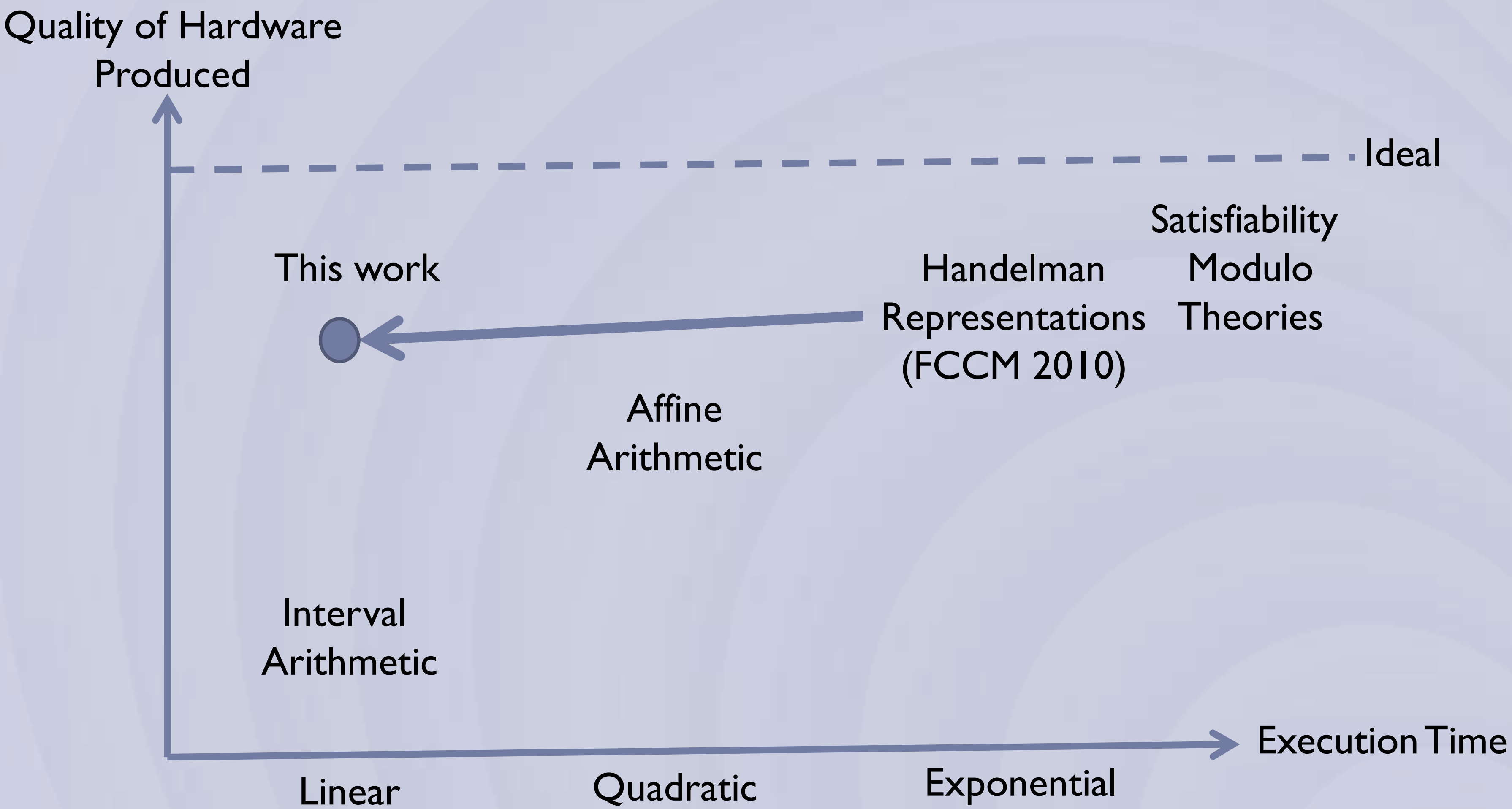
The battlefield



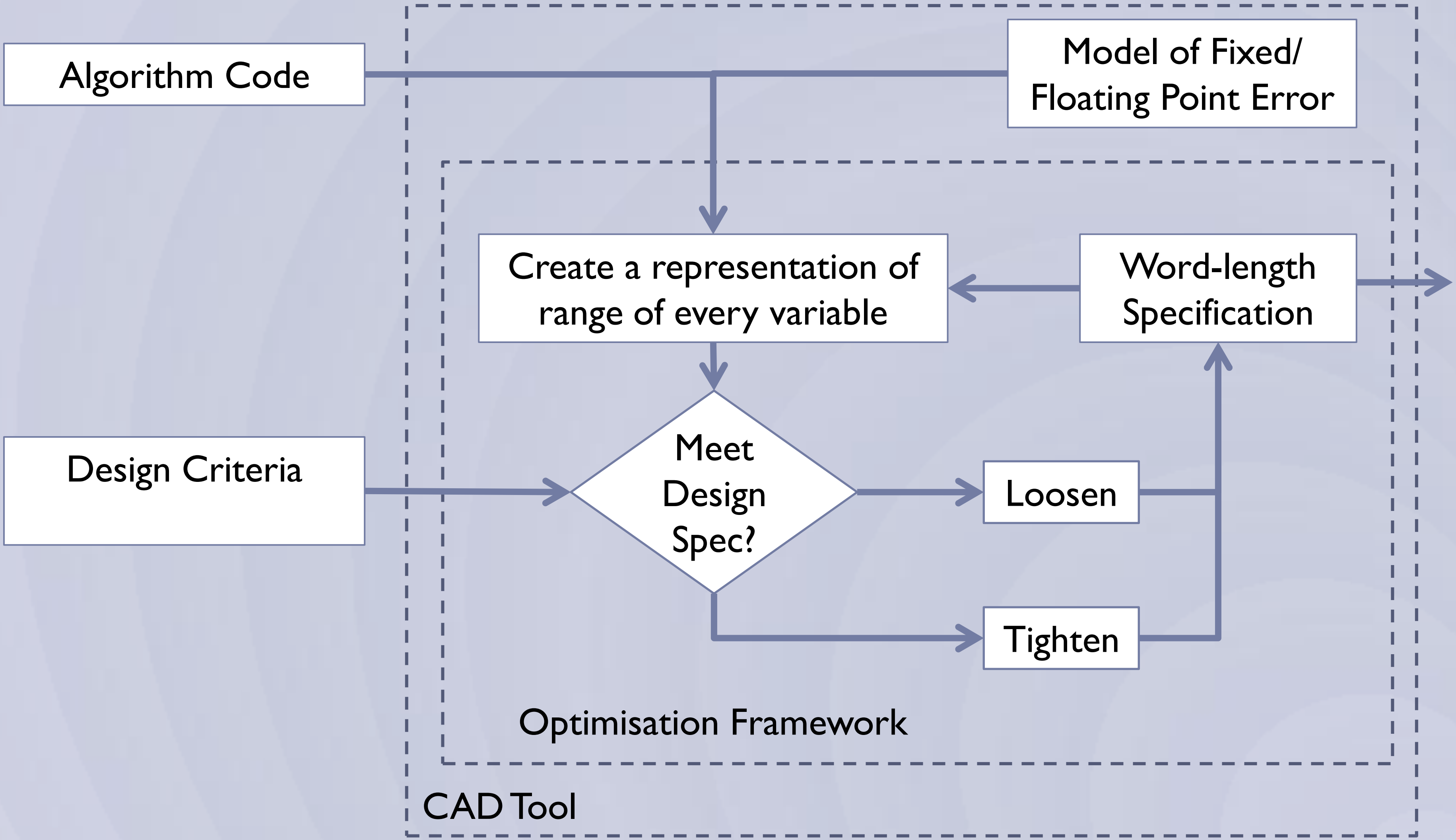
The battlefield



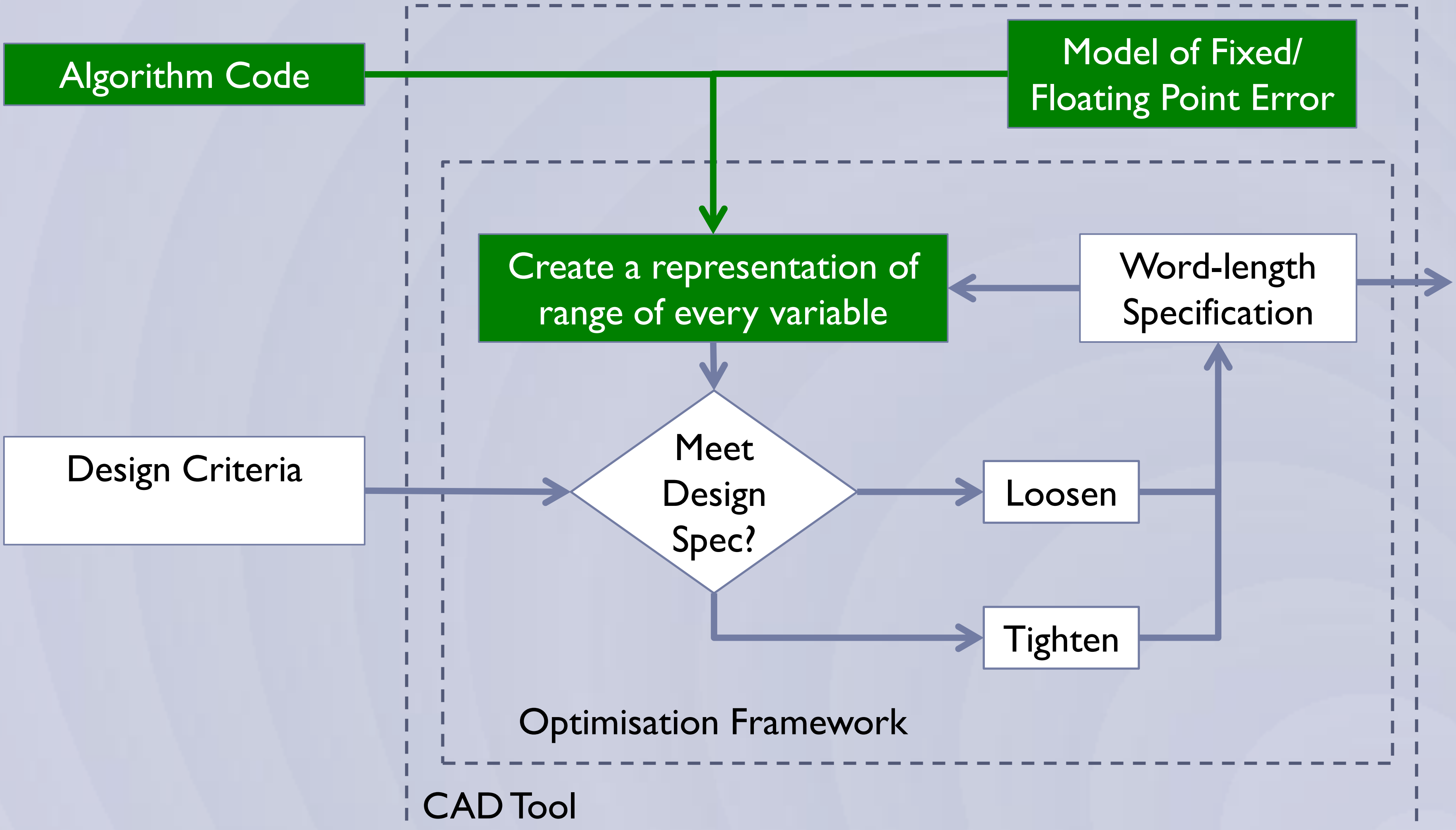
The battlefield



Why is scalability an issue?



Why is scalability an issue?



Why is scalability an issue?

- Modelling Floating Point Error
  - The closest floating point approximation $\hat{x}$ of x can be expressed as:

$$\hat{x} = x(1 + \delta_1) \qquad |\delta_1| \leq 2^{-m}$$

(m = # of mantissa bits)

  - The floating point result of any scalar operation $\odot$, where $\odot \in \{+, -, \times, \div\}$ can be bounded as:

$$\widehat{x \odot y} = (x \odot y)\,(1 + \delta_1)$$

Why is scalability an issue?

- Simple example:
  - Code: $a = x * y; b = a * z;$
  - Where: $x \in [0.8, 1.2], y \in [0.9, 1.1], z \in [9.9, 10.1]$

Why is scalability an issue?

- Simple example:
  - Code: $a = x * y; b = a * z;$
  - Where: $x \in [0.8, 1.2], y \in [0.9, 1.1], z \in [9.9, 10.1]$
  - If we denote: $|x_1| \leq 0.2, |y_1| \leq 0.1, |z_1| \leq 0.1 \; |\delta_i| \leq 2^{-12}$
  - Then: $x = (1 + x_1), y = (1 + y_1), z = (10 + z_1)$

Why is scalability an issue?

- Simple example:
  - Code: $a = x * y; b = a * z;$
  - Where: $x \in [0.8, 1.2], y \in [0.9, 1.1], z \in [9.9, 10.1]$
  - If we denote: $|x_1| \leq 0.2, |y_1| \leq 0.1, |z_1| \leq 0.1 \; |\delta_i| \leq 2^{-12}$
  - Then: $x = (1 + x_1), y = (1 + y_1), z = (10 + z_1)$
  - Create polynomials:

$$a = (1 + x_1)(1 + y_1)(1 + \delta_1)$$

$$a = 1 + x_1 + y_1 + x_1 y_1 + \delta_1 + x_1\delta_1 + y_1\delta_1 + x_1 y_1 \delta_1$$

Why is scalability an issue?

- Simple example:
  - Code: $a = x * y; b = a * z;$
  - Where: $x \in [0.8, 1.2], y \in [0.9, 1.1], z \in [9.9, 10.1]$
  - If we denote: $|x_1| \leq 0.2, |y_1| \leq 0.1, |z_1| \leq 0.1\ |\delta_i| \leq 2^{-12}$
  - Then: $x = (1 + x_1), y = (1 + y_1), z = (10 + z_1)$
  - Create polynomials:

$$a = (1 + x_1)(1 + y_1)(1 + \delta_1)$$

$$a = 1 + x_1 + y_1 + x_1 y_1 + \delta_1 + x_1\delta_1 + y_1\delta_1 \\ + x_1 y_1 \delta_1$$

$$b = (1 + x_1 + y_1 + x_1 y_1 + \delta_1 + x_1\delta_1 + y_1\delta_1 \\ + x_1 y_1 \delta_1)(10 + z_1)(1 + \delta_2)$$

Why is scalability an issue?

$$
\begin{aligned}
b &= 10 + 10x_1 + 10y_1 + 10x_1y_1 \\
&\quad +10\delta_1 + 10x_1\delta_1 + 10y_1\delta_1 + 10x_1y_1\delta_1 \\
&\quad + z_1 + x_1z_1 + y_1z_1 + x_1y_1z_1 \\
&\quad +\delta_1z_1 + x_1\delta_1z_1 + y_1\delta_1z_1 + x_1y_1\delta_1z_1 \\
&\quad + 10\delta_2 + 10x_1\delta_2 + 10y_1\delta_2 + 10x_1y_1\delta_2 \\
&\quad +10\delta_1\delta_2 + 10x_1\delta_1\delta_2 + 10y_1\delta_1\delta_2 + 10x_1y_1\delta_1\delta_2 \\
&\quad + z_1\delta_2 + x_1z_1\delta_2 + y_1z_1\delta_2 + x_1y_1z_1\delta_2 \\
&\quad +\delta_1z_1\delta_2 + x_1\delta_1z_1\delta_2 + y_1\delta_1z_1\delta_2 + x_1y_1\delta_1z_1\delta_2
\end{aligned}
$$

Why is scalability an issue?

$$
\begin{aligned}
b
&= 10 + 10x_1 + 10y_1 + 10x_1y_1 \\
&\quad +10\delta_1 + 10x_1\delta_1 + 10y_1\delta_1 + 10x_1y_1\delta_1 \\
&\quad + z_1 + x_1z_1 + y_1z_1 + x_1y_1z_1 \\
&\quad +\delta_1z_1 + x_1\delta_1z_1 + y_1\delta_1z_1 + x_1y_1\delta_1z_1 \\
&\quad + 10\delta_2 + 10x_1\delta_2 + 10y_1\delta_2 + 10x_1y_1\delta_2 \\
&\quad +10\delta_1\delta_2 + 10x_1\delta_1\delta_2 + 10y_1\delta_1\delta_2 + 10x_1y_1\delta_1\delta_2 \\
&\quad + z_1\delta_2 + x_1z_1\delta_2 + y_1z_1\delta_2 + x_1y_1z_1\delta_2 \\
&\quad +\delta_1z_1\delta_2 + x_1\delta_1z_1\delta_2 + y_1\delta_1z_1\delta_2 + x_1y_1\delta_1z_1\delta_2
\end{aligned}
$$

What are the bounds on the range and relative error of b?

Why is scalability an issue?

$$
\begin{aligned}
b \\
= \; & 10 + 10x_1 + 10y_1 + 10x_1y_1 \\
& +10\delta_1 + 10x_1\delta_1 + 10y_1\delta_1 + 10x_1y_1\delta_1 \\
& + z_1 + x_1z_1 + y_1z_1 + x_1y_1z_1 \\
& +\delta_1z_1 + x_1\delta_1z_1 + y_1\delta_1z_1 + x_1y_1\delta_1z_1 \\
& + 10\delta_2 + 10x_1\delta_2 + 10y_1\delta_2 + 10x_1y_1\delta_2 \\
& +10\delta_1\delta_2 + 10x_1\delta_1\delta_2 + 10y_1\delta_1\delta_2 + 10x_1y_1\delta_1\delta_2 \\
& + z_1\delta_2 + x_1z_1\delta_2 + y_1z_1\delta_2 + x_1y_1z_1\delta_2 \\
& +\delta_1z_1\delta_2 + x_1\delta_1z_1\delta_2 + y_1\delta_1z_1\delta_2 + x_1y_1\delta_1z_1\delta_2
\end{aligned}
$$

What are the bounds on the range and relative error of b?

This is computing x × y × z!!

Why is scalability an issue?

$$
\begin{aligned}
b &= 10 + 10x_1 + 10y_1 + 10x_1y_1 \\
&\quad +10\delta_1 + 10x_1\delta_1 + 10y_1\delta_1 + 10x_1y_1\delta_1 \\
&\quad + z_1 + x_1z_1 + y_1z_1 + x_1y_1z_1 \\
&\quad +\delta_1z_1 + x_1\delta_1z_1 + y_1\delta_1z_1 + x_1y_1\delta_1z_1 \\
&\quad + 10\delta_2 + 10x_1\delta_2 + 10y_1\delta_2 + 10x_1y_1\delta_2 \\
&\quad +10\delta_1\delta_2 + 10x_1\delta_1\delta_2 + 10y_1\delta_1\delta_2 + 10x_1y_1\delta_1\delta_2 \\
&\quad + z_1\delta_2 + x_1z_1\delta_2 + y_1z_1\delta_2 + x_1y_1z_1\delta_2 \\
&\quad +\delta_1z_1\delta_2 + x_1\delta_1z_1\delta_2 + y_1\delta_1z_1\delta_2 + x_1y_1\delta_1z_1\delta_2
\end{aligned}
$$

Why is scalability an issue?

$$
\begin{aligned}
& b \\
& = 10 + 10x_1 + 10y_1 + 10x_1y_1 \\
& \quad +10\delta_1 + 10x_1\delta_1 + 10y_1\delta_1 + 10x_1y_1\delta_1 \\
& \quad + z_1 + x_1z_1 + y_1z_1 + x_1y_1z_1 \\
& \quad +\delta_1z_1 + x_1\delta_1z_1 + y_1\delta_1z_1 + x_1y_1\delta_1z_1 \\
& \quad + 10\delta_2 + 10x_1\delta_2 + 10y_1\delta_2 + 10x_1y_1\delta_2 \\
& \quad +10\delta_1\delta_2 + 10x_1\delta_1\delta_2 + 10y_1\delta_1\delta_2 + 10x_1y_1\delta_1\delta_2 \\
& \quad + z_1\delta_2 + x_1z_1\delta_2 + y_1z_1\delta_2 + x_1y_1z_1\delta_2 \\
& \quad +\delta_1z_1\delta_2 + x_1\delta_1z_1\delta_2 + y_1\delta_1z_1\delta_2 + x_1y_1\delta_1z_1\delta_2
\end{aligned}
$$

Why is scalability an issue?

$$
\begin{aligned}
b \\
= \; & 10 + 10x_1 + 10y_1 + 10x_1y_1 \\
& +10\delta_1 + 10x_1\delta_1 + 10y_1\delta_1 + 10x_1y_1\delta_1 \\
& + z_1 + x_1z_1 + y_1z_1 + x_1y_1z_1 \\
& +\delta_1z_1 + x_1\delta_1z_1 + y_1\delta_1z_1 + x_1y_1\delta_1z_1 \\
& + 10\delta_2 + 10x_1\delta_2 + 10y_1\delta_2 + 10x_1y_1\delta_2 \\
& +10\delta_1\delta_2 + 10x_1\delta_1\delta_2 + 10y_1\delta_1\delta_2 + 10x_1y_1\delta_1\delta_2 \\
& + z_1\delta_2 + x_1z_1\delta_2 + y_1z_1\delta_2 + x_1y_1z_1\delta_2 \\
& +\delta_1z_1\delta_2 + x_1\delta_1z_1\delta_2 + y_1\delta_1z_1\delta_2 + x_1y_1\delta_1z_1\delta_2
\end{aligned}
$$

Can contribute $\pm 1.1920929 \times 10^{-10}$ to final range of b

Why is scalability an issue?

Can contribute ±2 to final range of b

$$
\begin{aligned}
b &= 10 + 10x_1 + 10y_1 + 10x_1y_1 \\
&\quad +10\delta_1 + 10x_1\delta_1 + 10y_1\delta_1 + 10x_1y_1\delta_1 \\
&\quad + z_1 + x_1z_1 + y_1z_1 + x_1y_1z_1 \\
&\quad +\delta_1z_1 + x_1\delta_1z_1 + y_1\delta_1z_1 + x_1y_1\delta_1z_1 \\
&\quad + 10\delta_2 + 10x_1\delta_2 + 10y_1\delta_2 + 10x_1y_1\delta_2 \\
&\quad +10\delta_1\delta_2 + 10x_1\delta_1\delta_2 + 10y_1\delta_1\delta_2 + 10x_1y_1\delta_1\delta_2 \\
&\quad + z_1\delta_2 + x_1z_1\delta_2 + y_1z_1\delta_2 + x_1y_1z_1\delta_2 \\
&\quad +\delta_1z_1\delta_2 + x_1\delta_1z_1\delta_2 + y_1\delta_1z_1\delta_2 + x_1y_1\delta_1z_1\delta_2
\end{aligned}
$$

Can contribute ±1.1920929 × 10^{-10} to final range of b

Why is scalability an issue?

$$
\begin{aligned}
b &\\
= \; & 10 + 10x_1 + 10y_1 + 10x_1y_1 \\
& +10\delta_1 \cancel{+ 10x_1\delta_1 + 10y_1\delta_1 + 10x_1y_1\delta_1} \\
& + z_1 + x_1z_1 + y_1z_1 + x_1y_1z_1 \\
& \cancel{+ \delta_1z_1 + x_1\delta_1z_1 + y_1\delta_1z_1 + x_1y_1\delta_1z_1} \\
& + 10\delta_2 \cancel{+ 10x_1\delta_2 + 10y_1\delta_2 + 10x_1y_1\delta_2} \\
& \cancel{+ 10\delta_1\delta_2 + 10x_1\delta_1\delta_2 + 10y_1\delta_1\delta_2 + 10x_1y_1\delta_1\delta_2} \\
& \cancel{+ z_1\delta_2 + x_1z_1\delta_2 + y_1z_1\delta_2 + x_1y_1z_1\delta_2} \\
& \cancel{+ \delta_1z_1\delta_2 + x_1\delta_1z_1\delta_2 + y_1\delta_1z_1\delta_2 + x_1y_1\delta_1z_1\delta_2}
\end{aligned}
$$

$$
\begin{aligned}
b = 10 + 10x_1 + 10y_1 + 10x_1y_1 + 10\,\delta_1 + z_1 \\
+ x_1z_1 + y_1z_1 + x_1y_1z_1 + 10\delta_2
\end{aligned}
$$

Why is scalability an issue?

$$
\begin{aligned}
b &= 10 + 10x_1 + 10y_1 + 10x_1y_1 \\
&\quad +10\delta_1 \cancel{+ 10x_1\delta_1 + 10y_1\delta_1 + 10x_1y_1\delta_1} \\
&\quad + z_1 + x_1z_1 + y_1z_1 + x_1y_1z_1 \\
&\quad \cancel{+ \delta_1z_1 + x_1\delta_1z_1 + y_1\delta_1z_1 + x_1y_1\delta_1z_1} \\
&\quad + 10\delta_2 \cancel{+ 10x_1\delta_2 + 10y_1\delta_2 + 10x_1y_1\delta_2} \\
&\quad \cancel{+ 10\delta_1\delta_2 + 10x_1\delta_1\delta_2 + 10y_1\delta_1\delta_2 + 10x_1y_1\delta_1\delta_2} \\
&\quad \cancel{+ z_1\delta_2 + x_1z_1\delta_2 + y_1z_1\delta_2 + x_1y_1z_1\delta_2} \\
&\quad \cancel{+ \delta_1z_1\delta_2 + x_1\delta_1z_1\delta_2 + y_1\delta_1z_1\delta_2 + x_1y_1\delta_1z_1\delta_2}
\end{aligned}
$$

$$
\begin{aligned}
b = 10 + 10x_1 + 10y_1 + 10x_1y_1 + 10\,\delta_1 + z_1 \\
+ x_1z_1 + y_1z_1 + x_1y_1z_1 + 10\delta_2 + \zeta_1
\end{aligned}
\qquad |\zeta_1| \le 0.0015
$$

An added bonus

- Can use methods from *approximation theory* to make our technique applicable to algorithms including any elementary functions (e.g. Sine/Cosine/Sqrt)
  - These methods approximate an elementary function using a polynomial and an extra term bounding the error of the approximation

Why is scalability an issue?

$$
\begin{aligned}
b &= 10 + 10x_1 + 10y_1 + 10x_1y_1 \\
&\quad +10\delta_1 \cancel{+ 10x_1\delta_1 + 10y_1\delta_1 + 10x_1y_1\delta_1} \\
&\quad + z_1 + x_1z_1 + y_1z_1 + x_1y_1z_1 \\
&\quad \cancel{+ \delta_1z_1 + x_1\delta_1z_1 + y_1\delta_1z_1 + x_1y_1\delta_1z_1} \\
&\quad + 10\delta_2 \cancel{+ 10x_1\delta_2 + 10y_1\delta_2 + 10x_1y_1\delta_2} \\
&\quad \cancel{+ 10\delta_1\delta_2 + 10x_1\delta_1\delta_2 + 10y_1\delta_1\delta_2 + 10x_1y_1\delta_1\delta_2} \\
&\quad \cancel{+ z_1\delta_2 + x_1z_1\delta_2 + y_1z_1\delta_2 + x_1y_1z_1\delta_2} \\
&\quad \cancel{+ \delta_1z_1\delta_2 + x_1\delta_1z_1\delta_2 + y_1\delta_1z_1\delta_2 + x_1y_1\delta_1z_1\delta_2}
\end{aligned}
$$

$$
\begin{aligned}
b = 10 + 10x_1 + 10y_1 + 10x_1y_1 + 10\,\delta_1 + z_1 \\
+ x_1z_1 + y_1z_1 + x_1y_1z_1 + 10\delta_2 + \zeta_1 \qquad |\zeta_1| \le 0.0015
\end{aligned}
$$

An added bonus

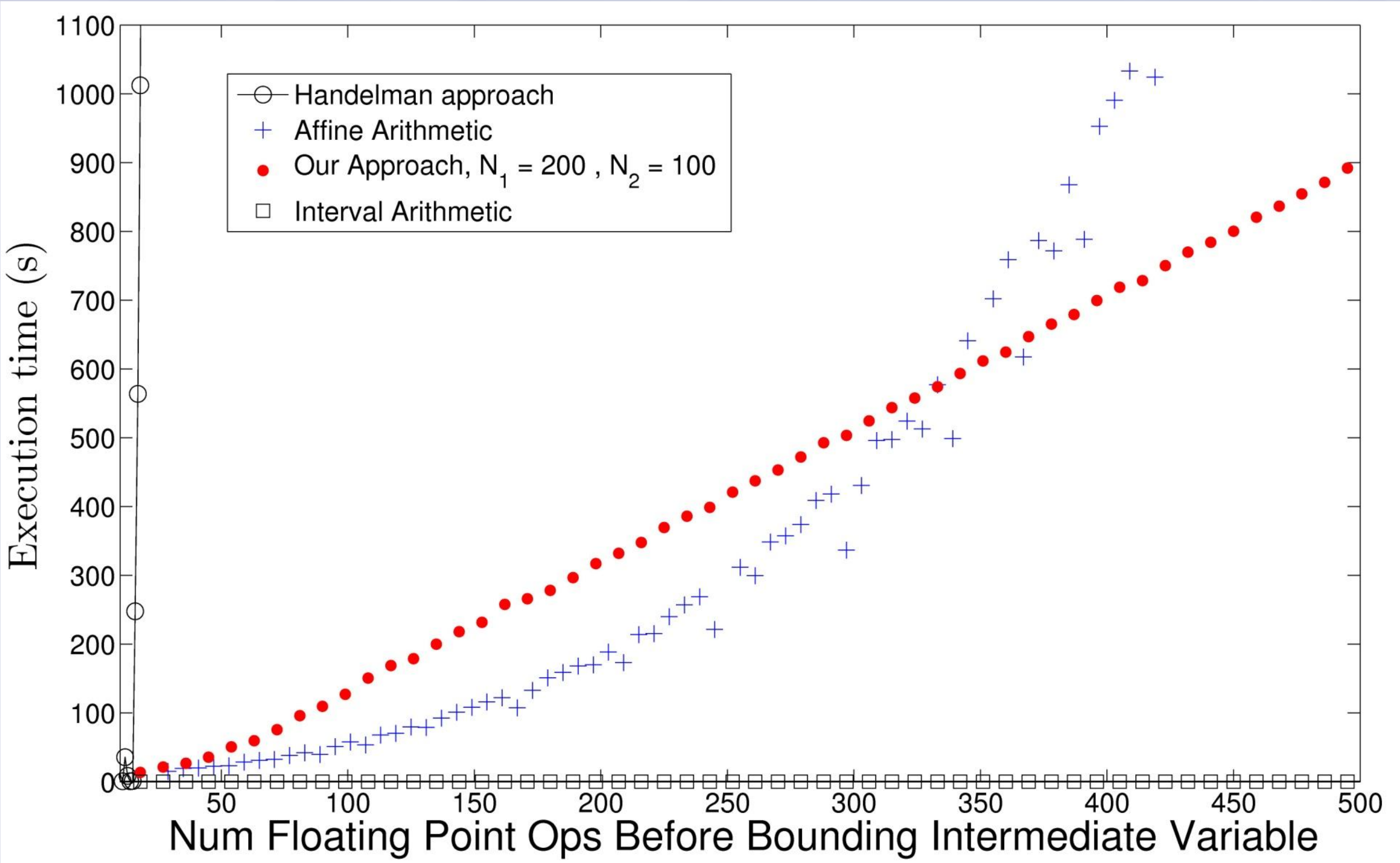
- Can use methods from *approximation theory* to make our technique applicable to algorithms including any elementary functions (e.g. Sine/Cosine/Sqrt)
  - Methods from approximation theory approximate an elementary function using a polynomial and an extra term bounding the error of the approximation
    - We get this for free!!

Tests

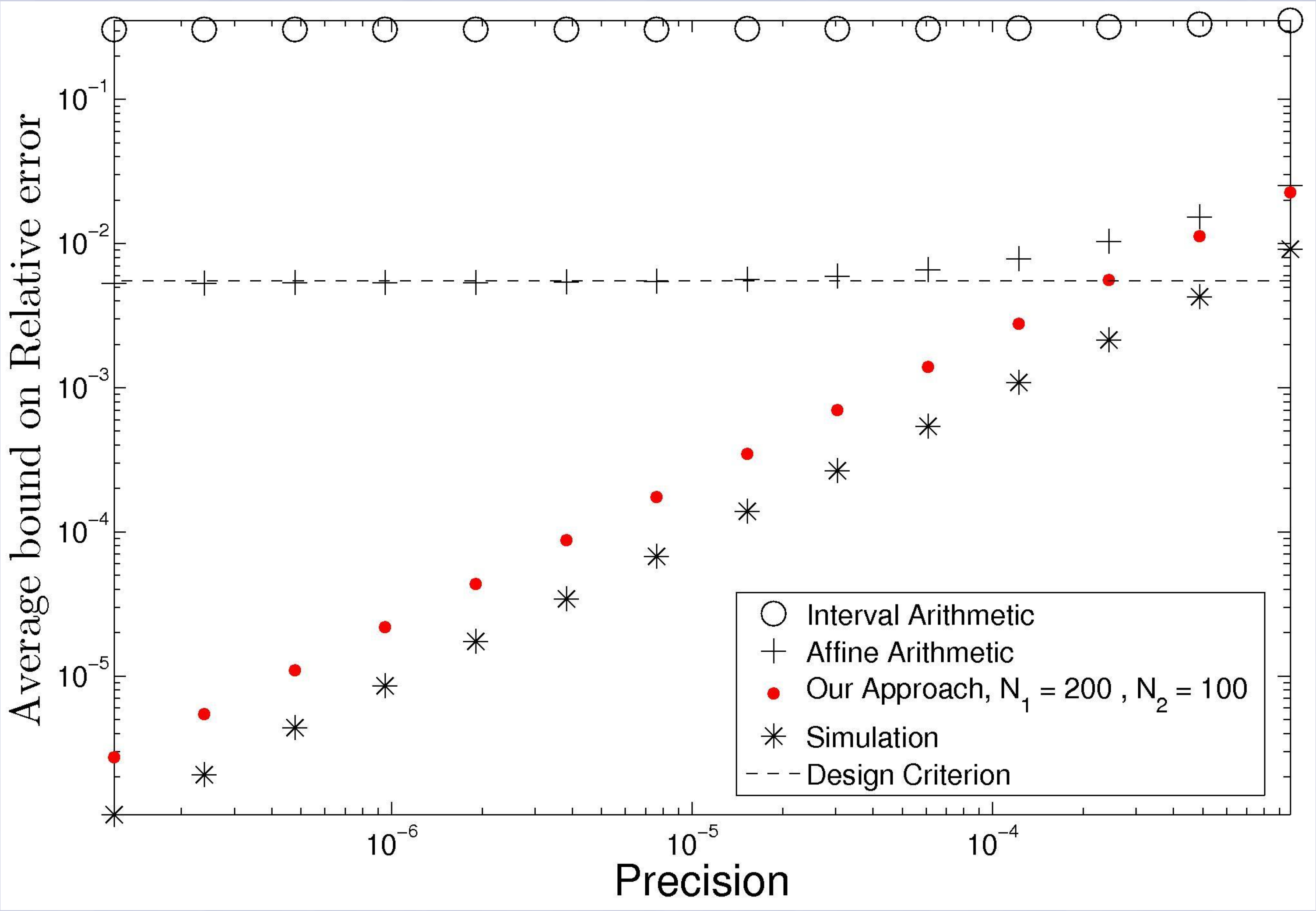
- 5x5 Successive over relaxation
  - Real algorithm to find the solution to a system of linear equations of form $Ax = b$

$$
\begin{array}{l}
\textbf{for } k = 1; k \leq 8; k++ \textbf{ do} \\
\quad \textbf{for } j = 1; j \leq 5; j++ \textbf{ do} \\
\quad\quad x^j = (1 - w)x^j + \frac{w}{A(j)^j}\left(b_j - \sum_{i=1, i \neq j}^{5} A(j)^i x^i\right) \\
\quad \textbf{end for} \\
\textbf{end for}
\end{array}
$$

Scalability: Execution time vs #operations



Quality of bounds: Relative error vs precision



Hardware use

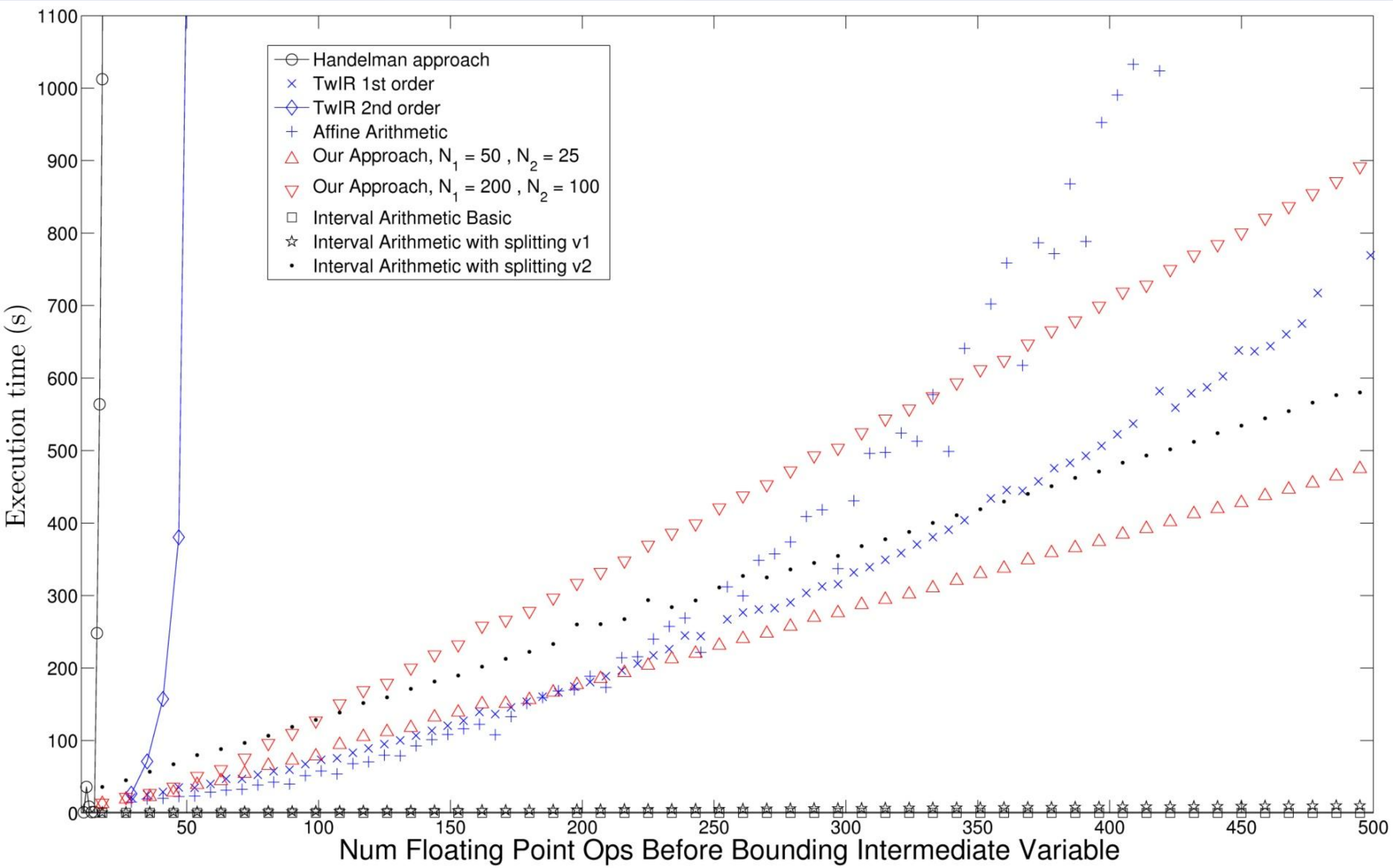
Method	Exponent (# bits)	Mantissa (# bits)	Slice Regs	Slice LUTs	Frequency (MHz)
Simulation	8	11	3562	3012	330
Our Approach	8	13	4261	3647	330
Affine Arithmetic	8	18	6606	5368	300
IA	∞	∞	∞	∞	N/A
IEEE Single Precision	8	24	8407	6815	280
IEEE Double Precision	11	53	27200	22066	251

Summary

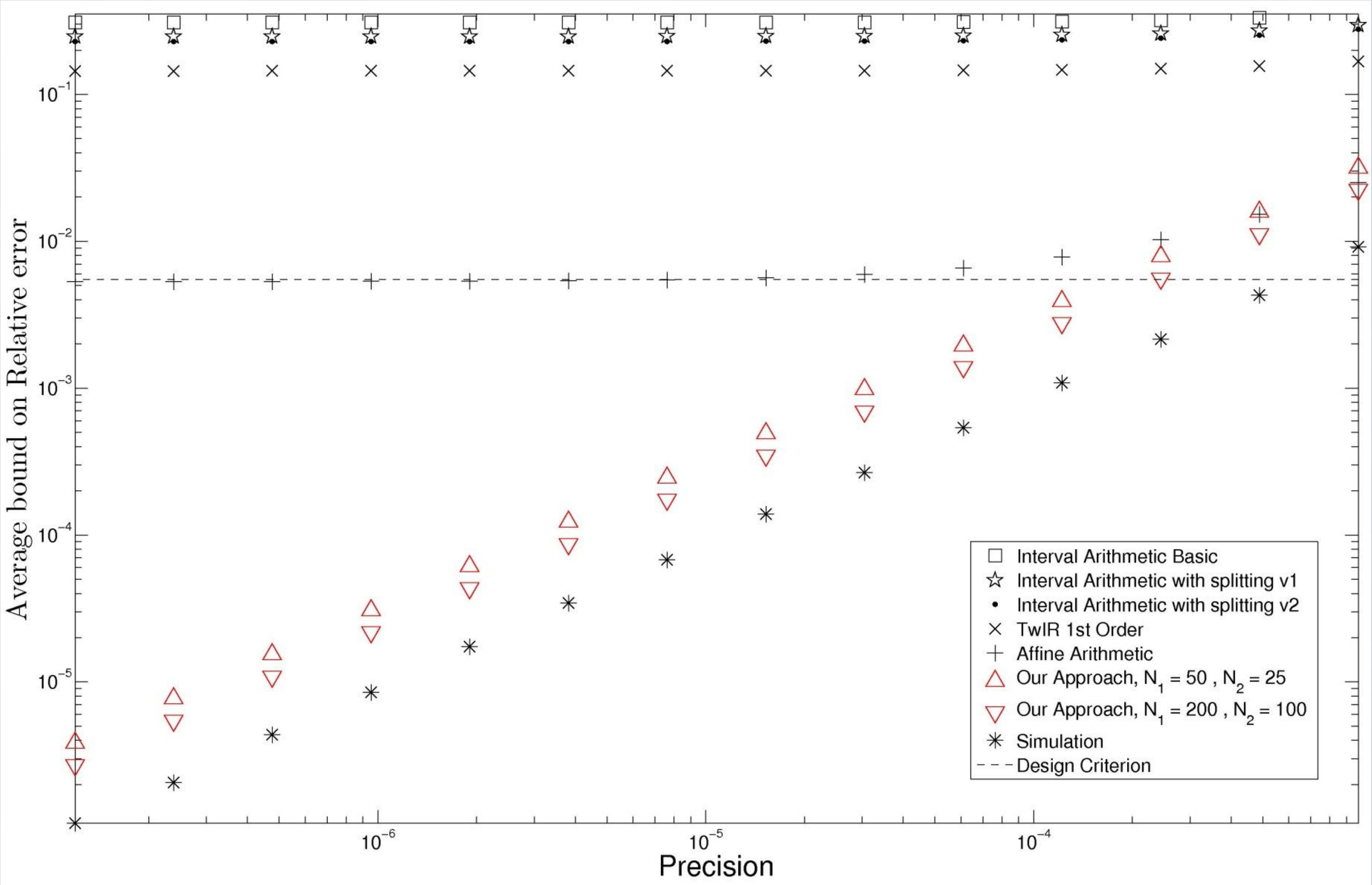
- Word-length optimisation can significantly improve hardware
- Need scalable analysis techniques to apply word-length optimisation on larger, more complex algorithms
- Our paper describes a simple set of algorithms to obtain tight bounds within a scalable execution time
  - Can use >80% fewer slice registers than IEEE double precision arithmetic
  - Can use >30% fewer slice registers than competing methods.
  - Can create hardware that is guaranteed to meet design criteria that is not possible using alternative methods

Thank you for listening

Scalability: Execution time vs #operations



Quality of bounds: Relative error vs precision



Quality of bounds vs execution time