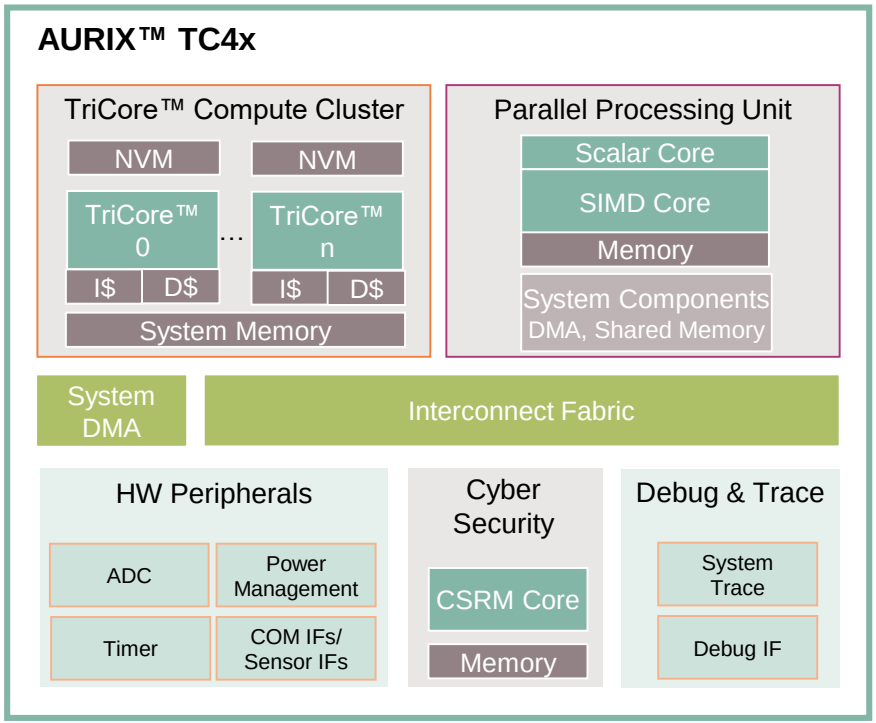
Model-Based Design for Next Generation AURIX™ Automotive Microcontroller

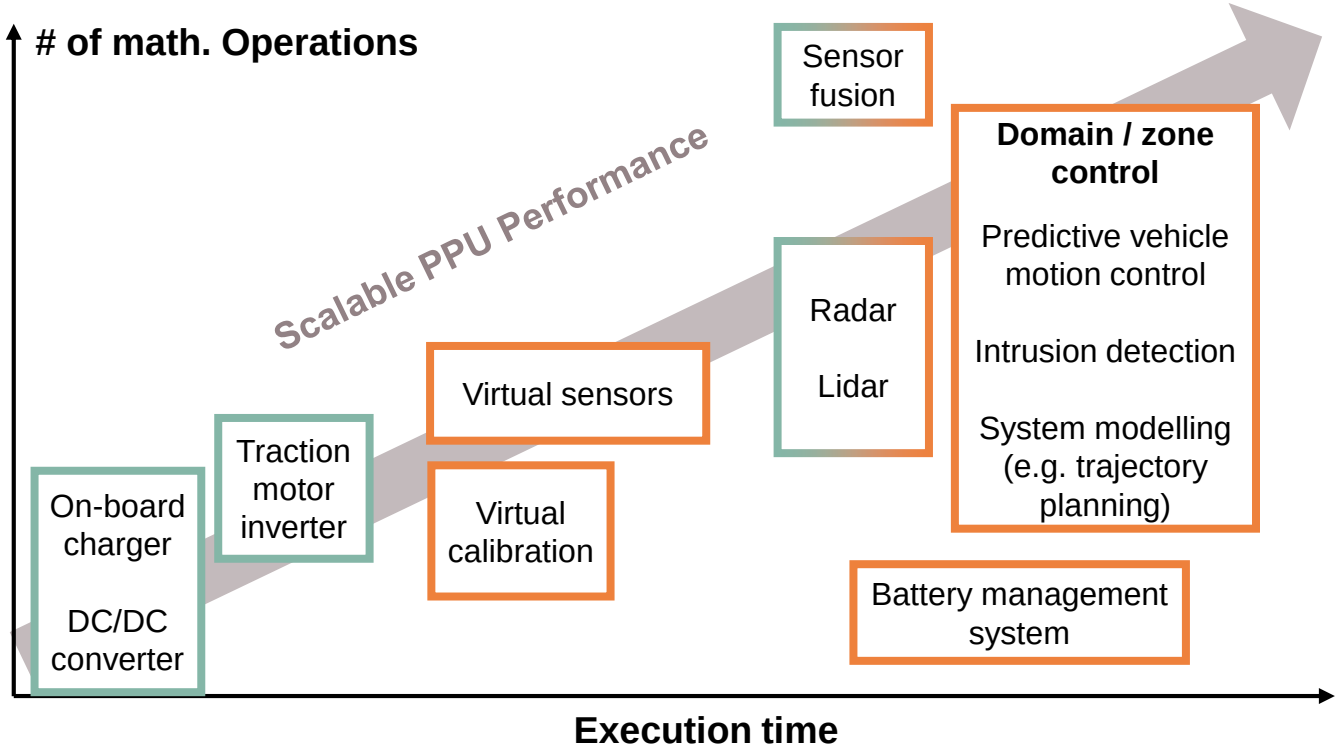
Kajetan Nürnberger
MATLAB Expo 2021

The next generation of AURIX™ automotive microcontroller



- In automotive, the demand on processing power is constantly increasing
- Offering the requested processing capabilities is a challenge under circumstances like:
  - Ambient temperature
  - Power consumption
- AURIX™ TC4x fulfills these requirements by providing a heterogenous architecture

PPU is a flexible architecture to address applications with fast execution times and/or large data processing requirements

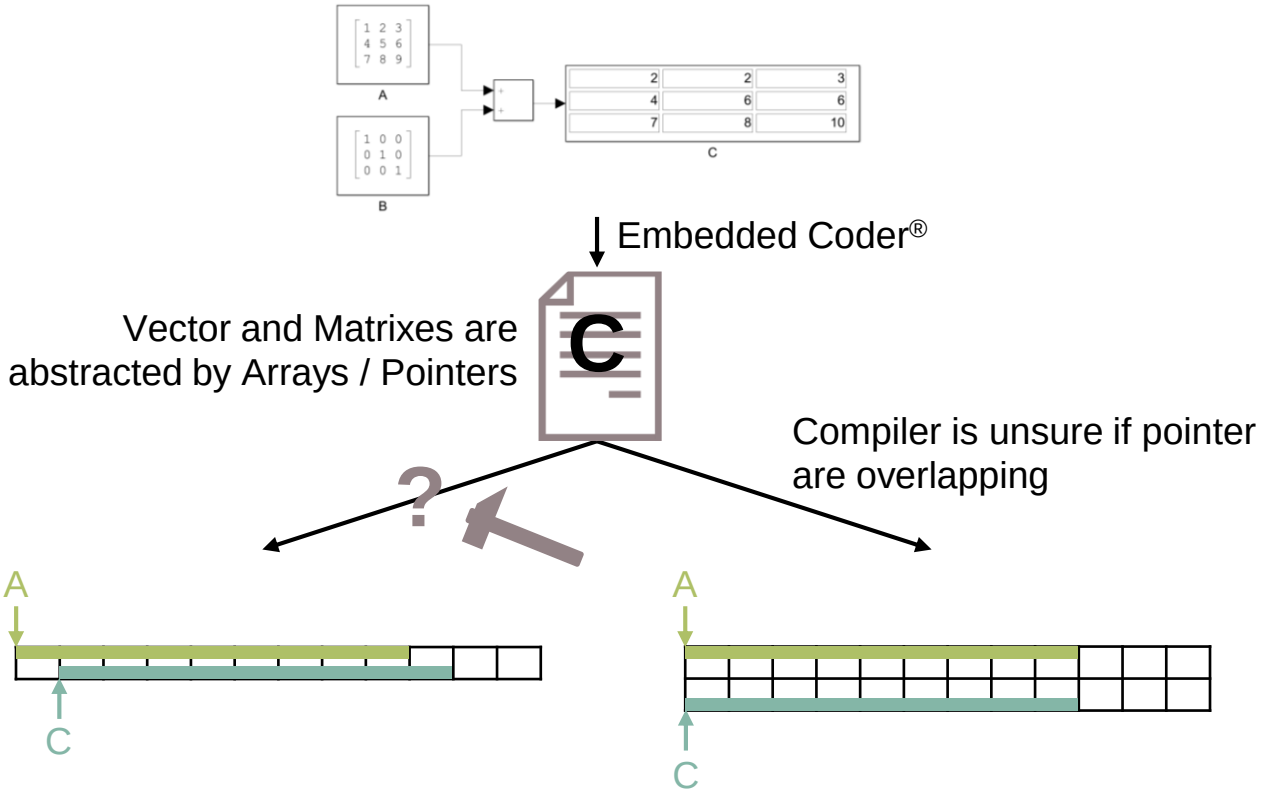


Implemented tasks per application:

Complex data processing and observer based controlling of sensor actuator systems

Artificial neural network (MLP, RBF, RNN, CNN) based system modelling and object classification

Challenges with heterogenous architectures
Programming model



Parallel Programming Models

CUDA®

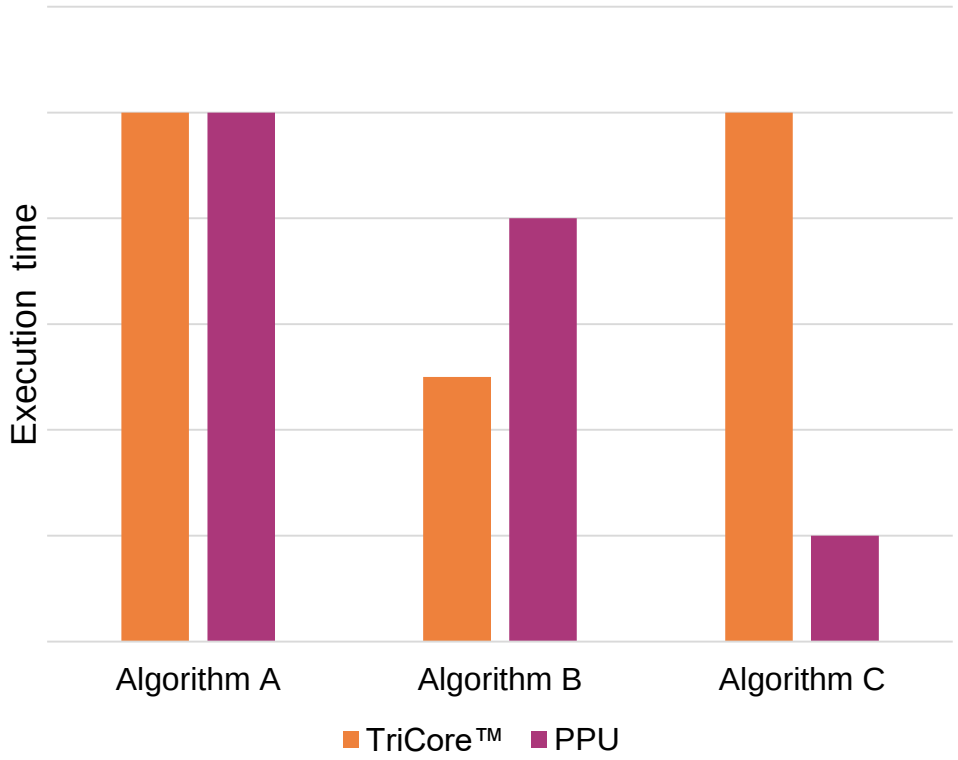
OpenCL™

OpenMP®

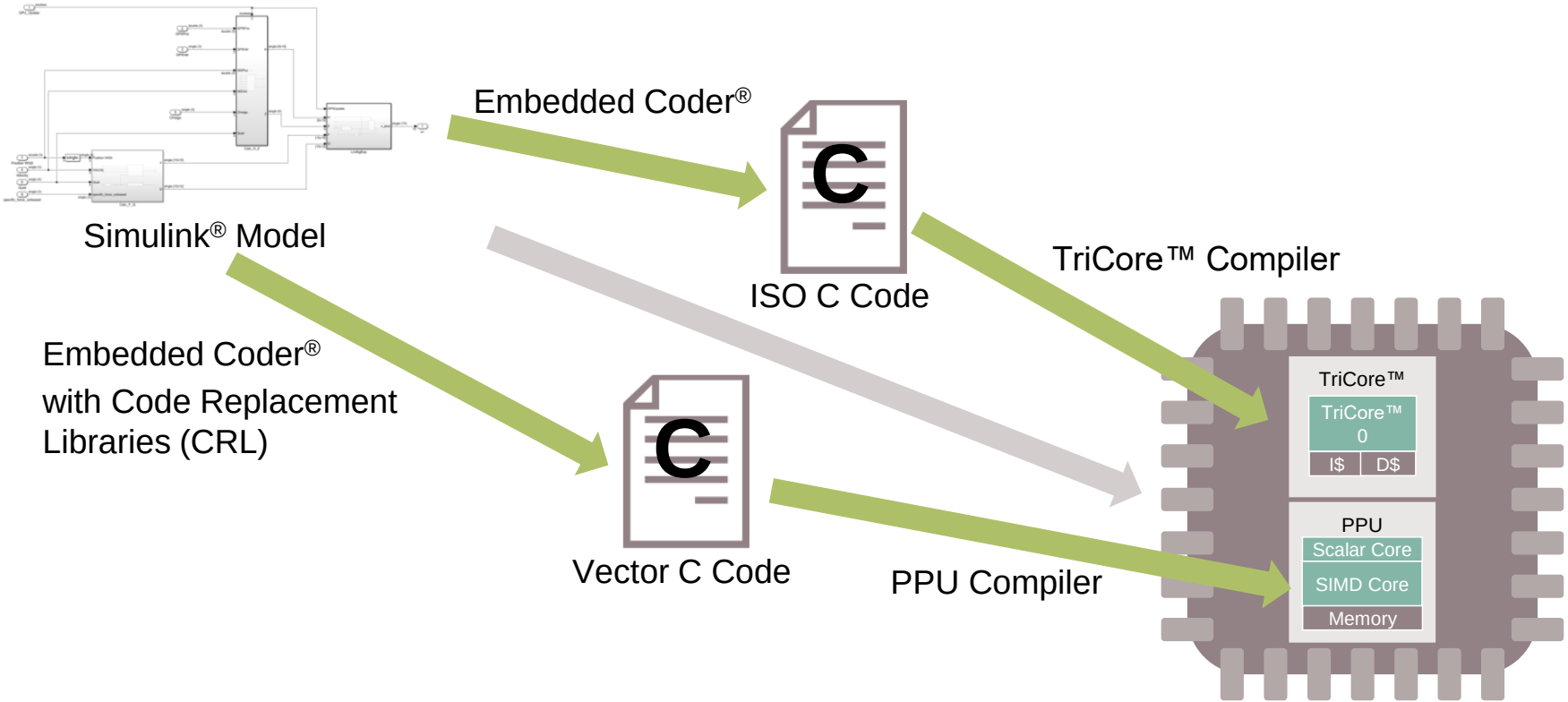
HW Intrinsics

Challenges with heterogenous architectures
Execution time dilemma

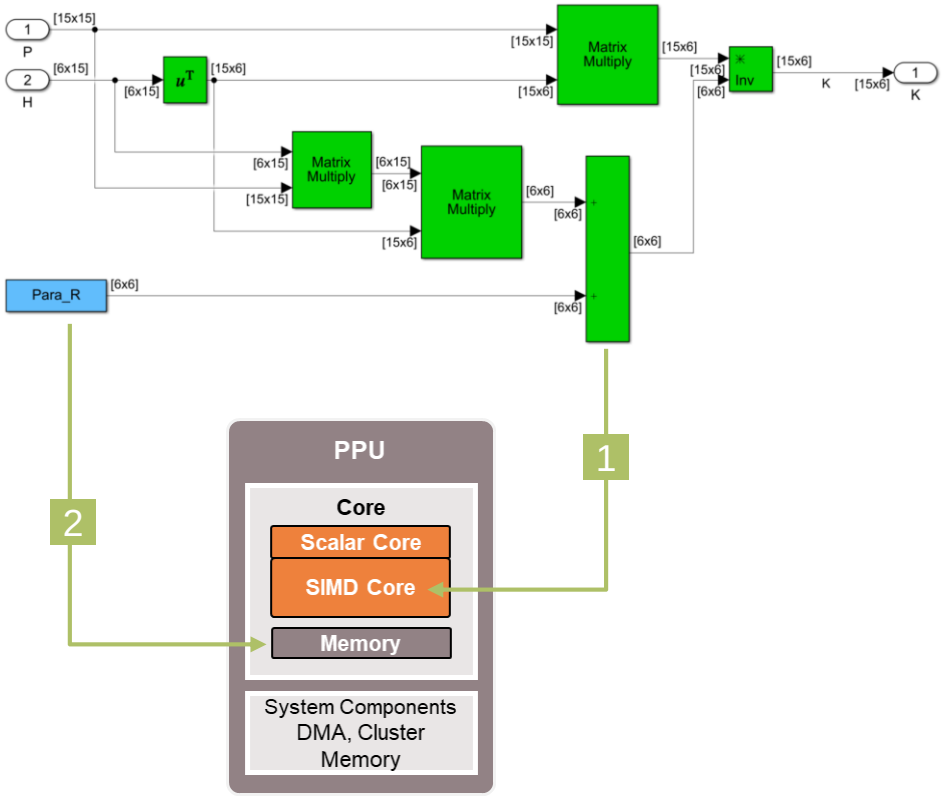
- Execution time of modern complex architectures cannot be easily predicted
- Execution time might completely vary on heterogenous compute units
- SoC interconnects might influence the overall response time



Simulink® & Embedded Coder® enable a smooth transition between different computing architectures



Dedicated tool flow enables implementation from model level to target device code



1 Code Replacement Library

Name	Implementation	NumIn	In1Type	In2Type	OutType
RTW_OP_ADD	ifx_mm_add_f32	2	single[5 5; Inf Inf]	single[5 5; Inf Inf]	single[5 5; Inf Inf]
RTW_OP_ELEM_MUL	ifx_ms_scale_f32	2	single[5 5; Inf Inf]	single	single[5 5; Inf Inf]
RTW_OP_ELEM_MUL	ifx_ms_scale_f32	2	single	single[5 5; Inf Inf]	single[5 5; Inf Inf]
RTW_OP_MINUS	ifx_mm_minus_f32	2	single[5 5; Inf Inf]	single[5 5; Inf Inf]	single[5 5; Inf Inf]
RTW_OP_MUL	ifx_mm_mul_f32	2	single[5 5; Inf Inf]	single[5 5; Inf Inf]	single[5 5; Inf Inf]
RTW_OP_RDIV	ifx_mm_rdiv_f32	2	single[5 5; Inf Inf]	single[5 5; 16 16]	single[5 5; Inf Inf]
RTW_OP_TRANS	ifx_m_transpose_f32	1	single[5 5; Inf Inf]		single[5 5; Inf Inf]

2 Custom Storage Class

Definition:

```
__vccm
/* CSC definition comment generated by default */
const DATATYPE DATANAME[DIMENSION] [= {...}];
```

What CRL implementations look like

- Special keywords to map static and global data to vector memory
  `__vccm`
- Special vector data types e.g.
  `vNfloat_t`
- Special intrinsic functions to address special vector instructions

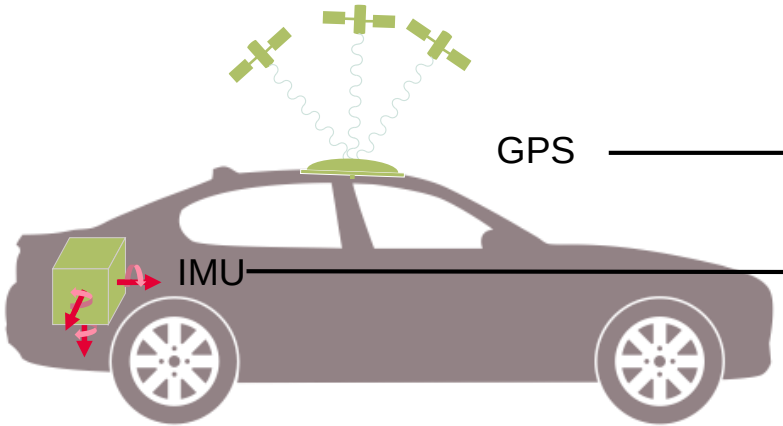
```
void ifx_mm_add_f32(const float __vccm* restrict A,
                    const int Row,
                    const int Col,
                    const float __vccm* restrict B,
                    float __vccm* restrict C)
{
  vNfloat_t ev_a, ev_b;
  vNint_t offsets = {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15};
  __attribute__((address_space(4))) float * c_ptr;
  int size = Row * Col;
  int i = 0;
  int k = 0;

  for (i = 16; i <= size ; i+=16)
  {
    k = i - 16;
    ev_a = vloadN(A + k);
    ev_b = vloadN(B + k);
    ev_b = ev_a + ev_b;
    c_ptr = C + k;
    vstoreN( ev_b,c_ptr);
  }

  k = i - 16;
  offsets = offsets;
  pvN predicate = to_pvN(offsets < (size-k));
  ev_a = vloadN(A + k);
  ev_b = vloadN(B + k);
  ev_b = ev_a + ev_b;
  c_ptr = C + k;
  vscatter(ev_b, c_ptr, offsets, predicate);
}
```

Process completely populated vectors

Process partially populated vectors

Navigation example

Data Acquistion | **Strapdown Navigation** | **Sensor Fusion**



Position,
Speed

Applying
Kalman Filter

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k \tilde{y}_k$$

Acceleration,
Turn Rate

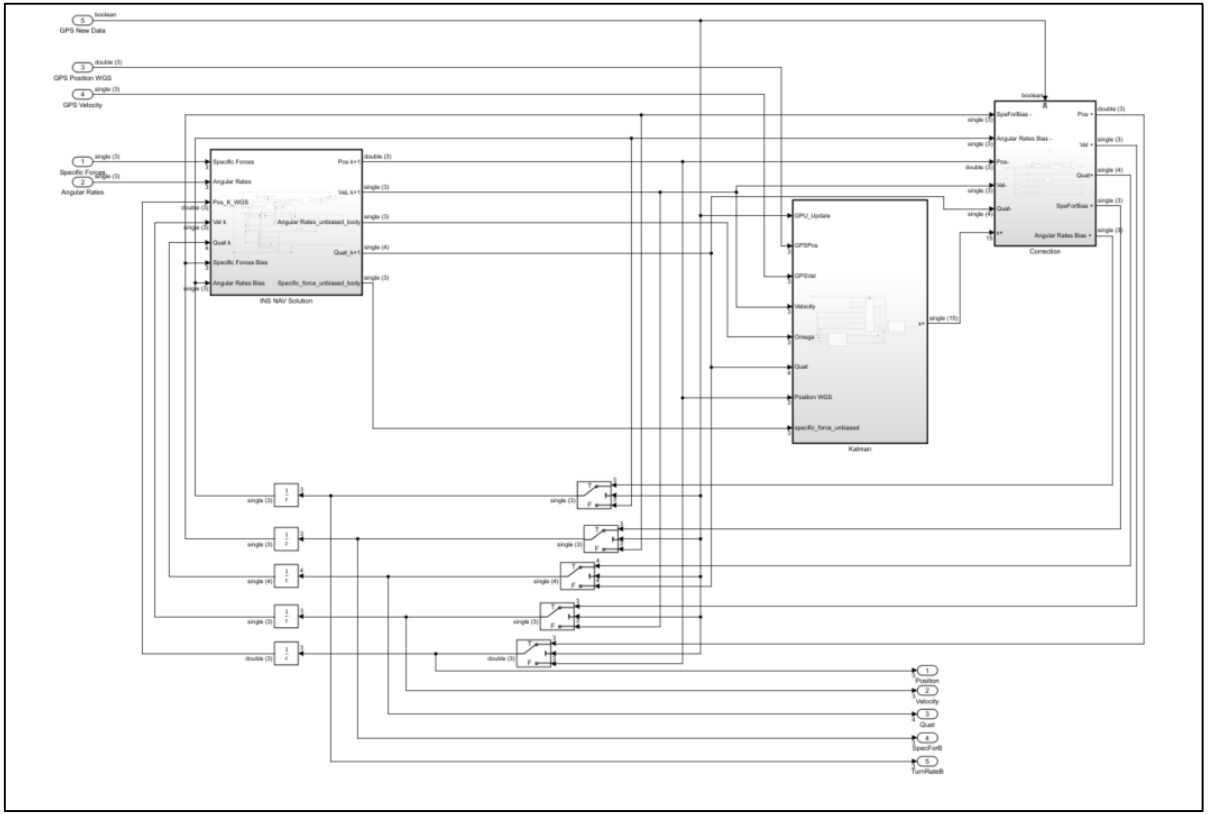
$\int$

Position,
Speed,
Orientation

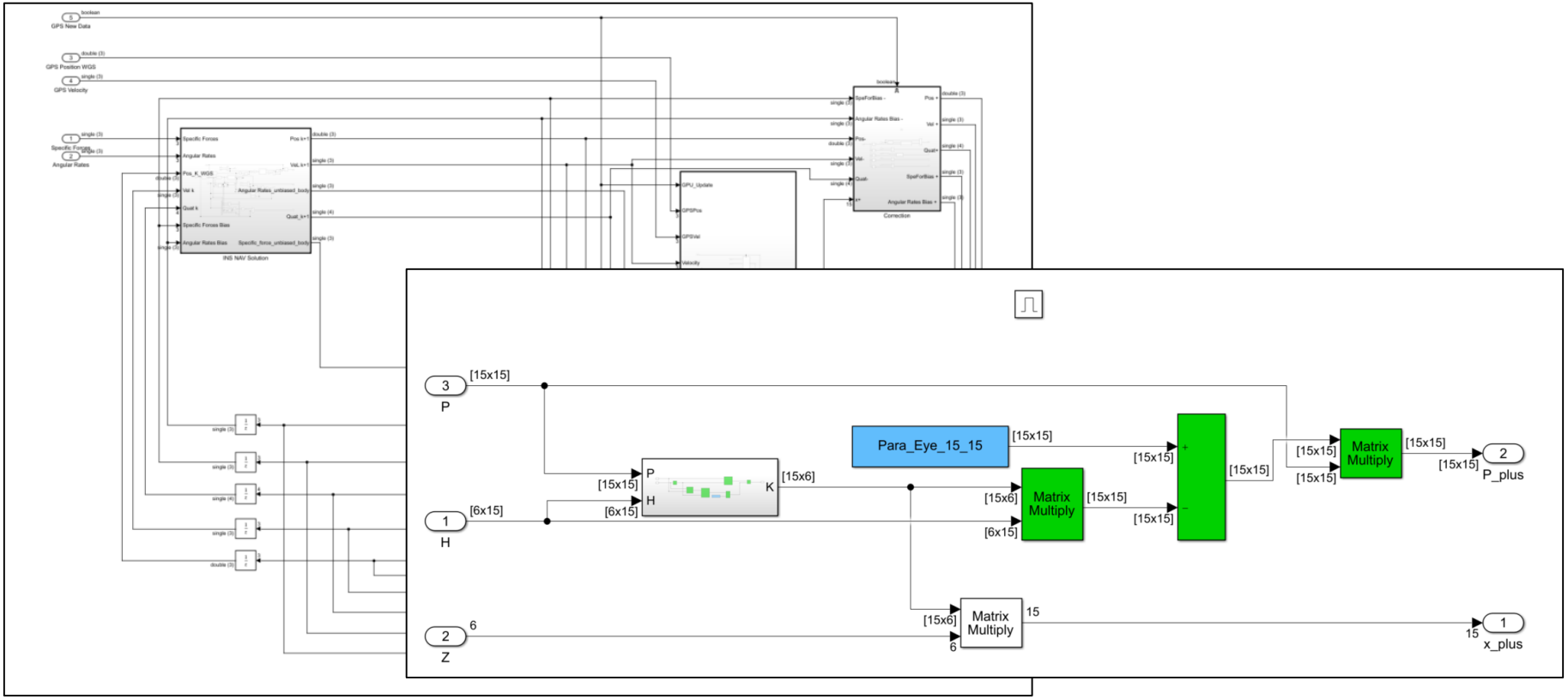
$$P_{k|k} = (I - K_k H_k)\, P_{k|k-1}$$

- Enhanced ADAS functions rely on sensor fusion
- Sensor fusion algorithms have a high demand of matrix and vector operations
- Keeping the time budget within a hard real-time system is challenging with big matrix operations
- Algorithm developers like to work in HW independently

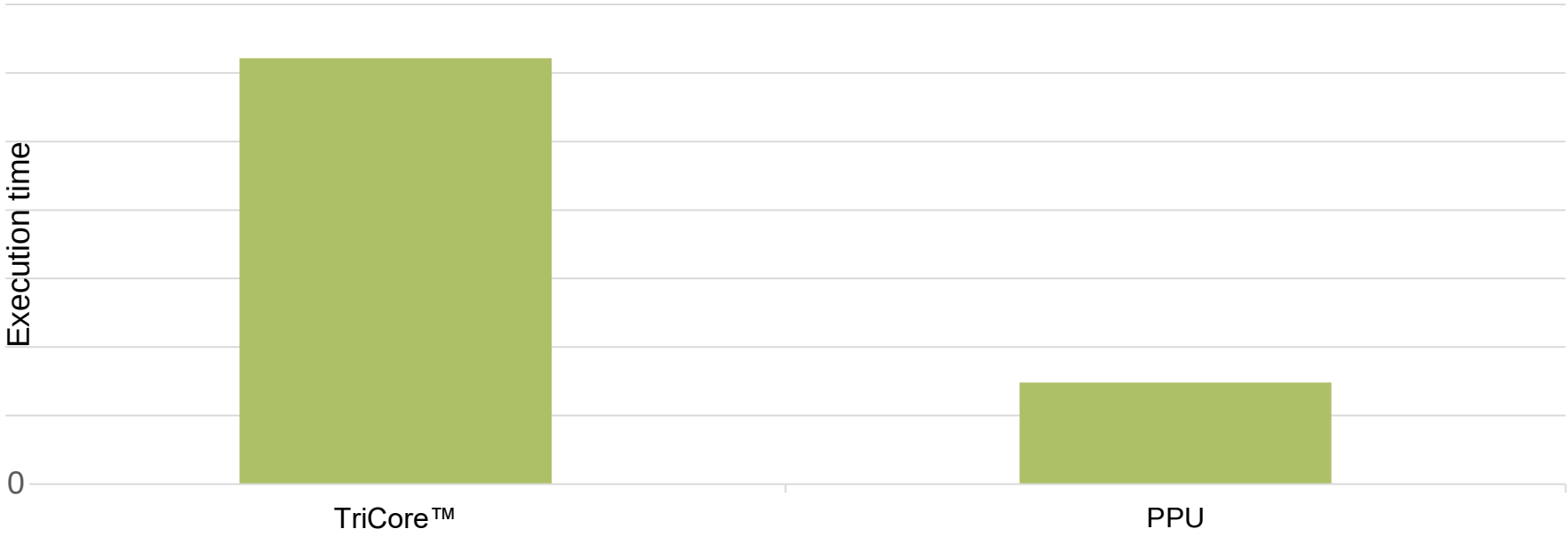
Designing the algorithm using Simulink



Designing the algorithm using Simulink

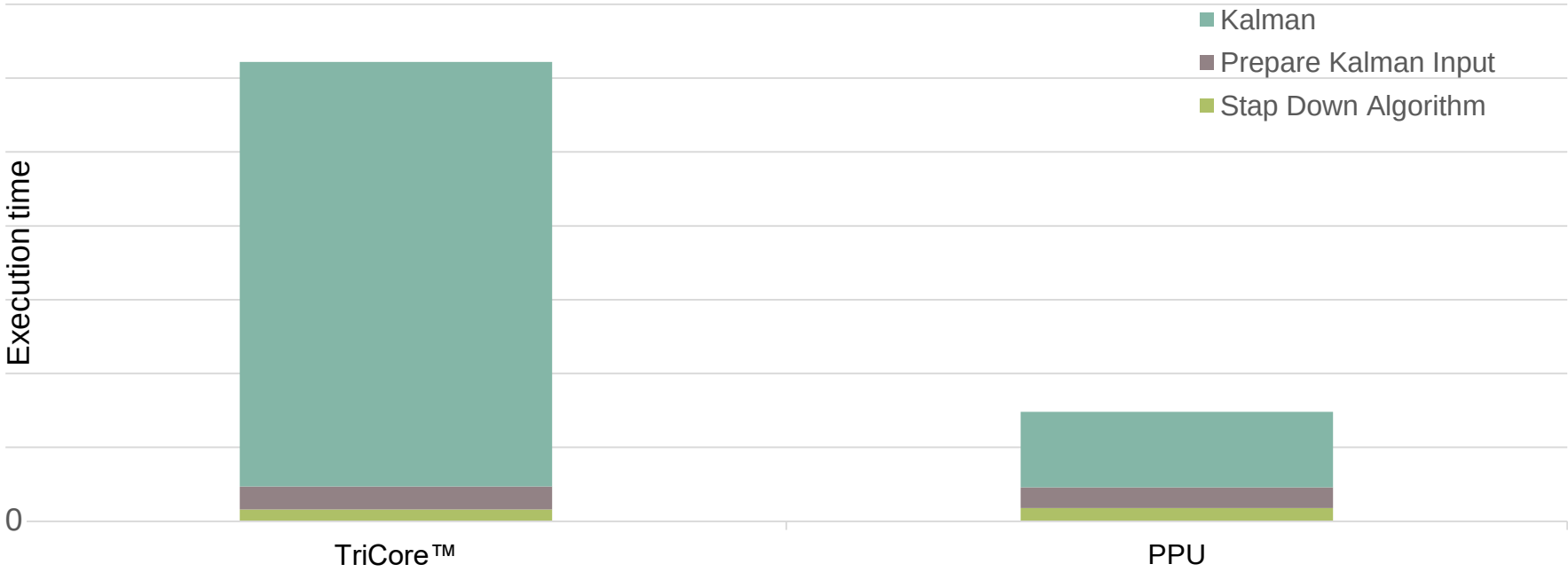


Comparison of the execution time of the complete algorithm



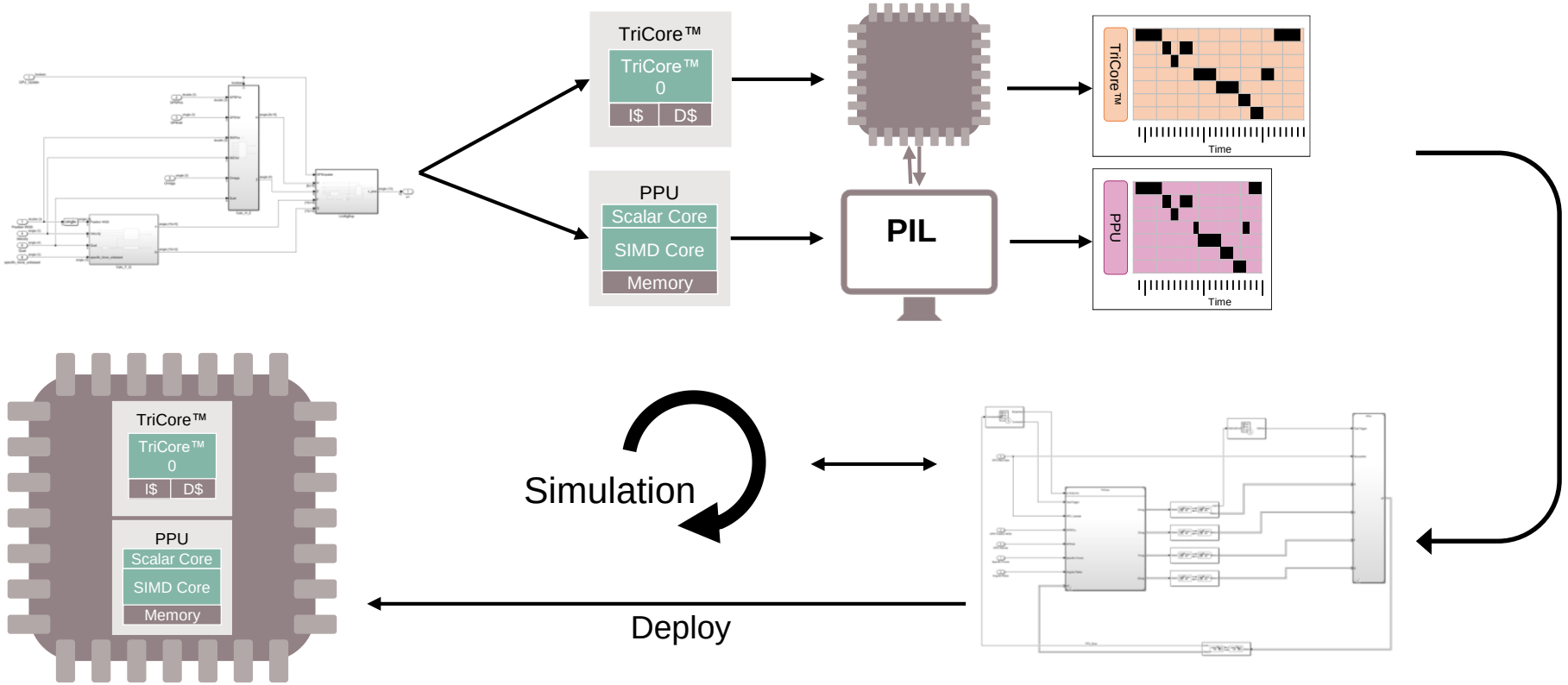
› Significant decrease of execution time by utilization of SIMD capabilities

Detailed look at execution times



› Splitting the algorithm might bring benefit as PPU time can be used otherwise

Solving the computing resource allocation dilemma



Conclusion

- The enhanced matrix / vector calculation capabilities of the next generation AURIX™ TC4x can be utilized using the Embedded Coder
- The use of Model-Based Design enables an efficient tailoring of an algorithm to a heterogenous HW architecture
- The higher level representation of a model can be easily ported between different HW architectures
- The SoC Blockset™ enables simulation of the integrated SOC HW + SW directly within the MathWorks® tool chain

Part of your life. Part of tomorrow.