

# **Multi-Core Execution of Parallelised Hard Real-Time Applications Supporting Analysability parMERASA**

Parallelizing avionics applications

João Fernandes  
Honeywell International s.r.o.

- Improvements in SWaP
  - Space
  - Weight
  - Power
- Increased overall safety and efficiency
  - Ability to provide new features
- Challenges (specific to the avionics domains):
  - (Very) Conservative domain
  - (Re-)Certification
  - Huge legacy code bases (tens of **millions** LoC!)

- 3 applications

### 3D Path Planning



### Stereo Navigation



### GNSS Receiver



- **Task parallelism**

- 3D Path Planning
  - Compartments in different iterations (pipelining)
- Stereo Navigation
  - 3 stages pipeline
  - Feature extraction – convolutions
- GNSS Receiver
  - Decoding procedures differ for different types of signals

- **Data parallelism**

- – 3D Path Planning
  - Compartments in the same iteration
- Stereo Navigation
  - Feature extraction – tiles
  - Pose estimation
- GNSS Receiver
  - Decoding procedures do not differ for the same type of signal
  - PVT solvers

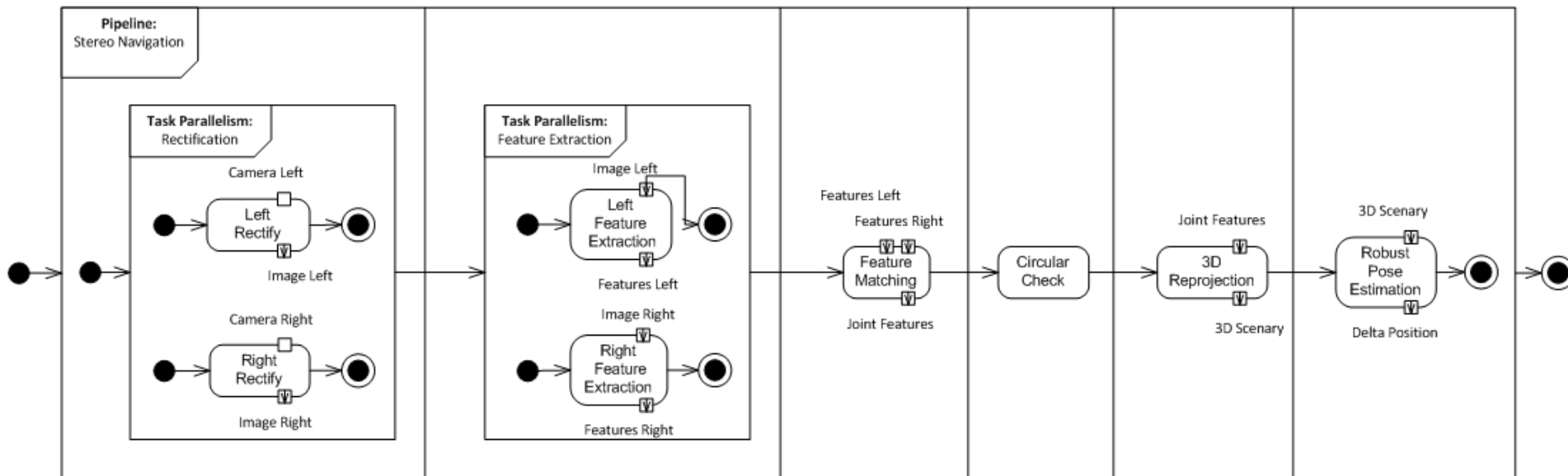
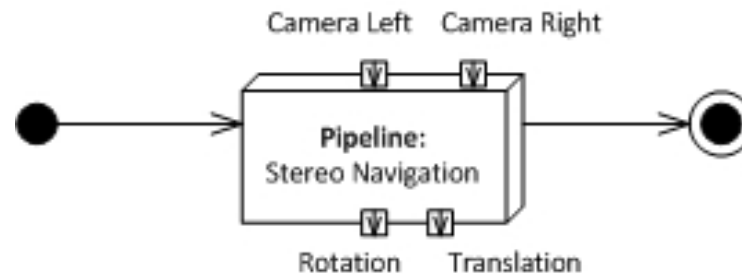
- Challenge:
  - Parallel but analyzable
  - High WCET speedup
  - Low development effort
- parMERASA Pattern-supported Parallelization Approach
  - Defined structures for analysability and dev. Effort
    - Parallel Design Patterns (see approach by Mattson et al.)
  - Methodical (see PCAM by Foster) → Two phases
  - Respecting the requirements of WCET analysis
  - Model based → Extended UML2 Activity Diagram
- Detailed description of approach without hard real-time:

[Jahr, R.; Gerdes, M. & Ungerer, T.; **A Pattern-supported Parallelization Approach**; Proceedings of the 2013 International Workshop on Programming Models and Applications for Multicores and Manycores (PMAM 2013); ACM digital library; to appear in February 2013]

- Starting point: sequential program (problem description)
- Final result: **predictable** parallel program
- Step 1: Targeting Maximum Parallelism
  - Create model to reveal parallelism
  - **Activity and Pattern Diagram (APD)** consists of sequential **Activities** and **Parallel Design Patterns**
  - Platform independent
- Step 2: Targeting Optimal Parallelism
  - Agglomeration of its nodes
  - Creation of threads
  - Mapping onto target architecture
  - Platform dependent



## ■ Stereo Navigation



- Minimum update rate: **1 Hz**
- Desirable update rate: **10 Hz**



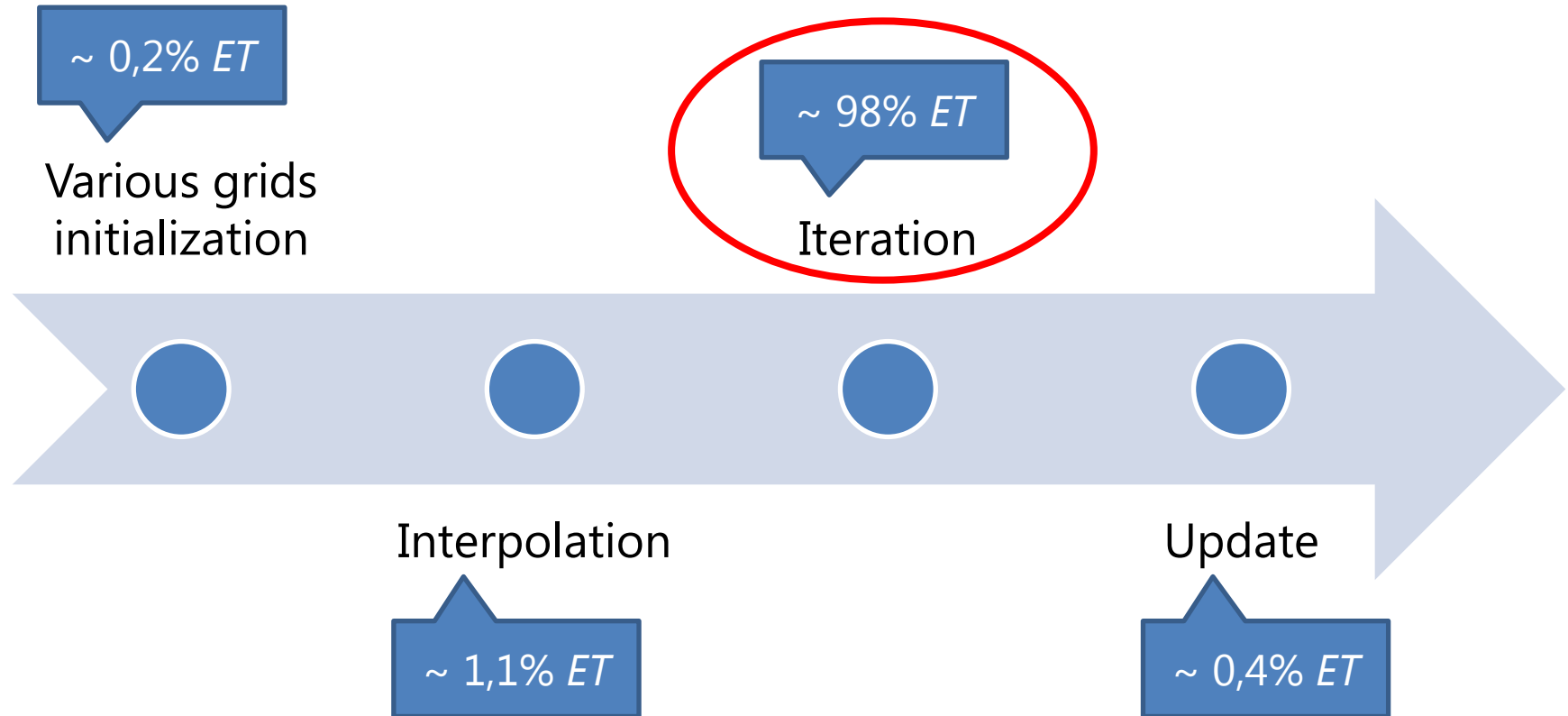
- **Goal:** autonomously flying vehicle



- Using Laplace's equation

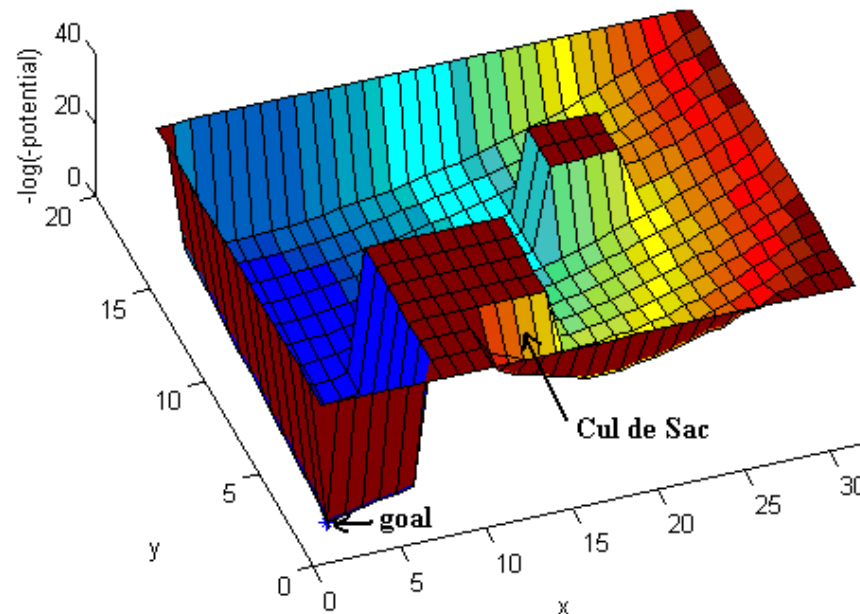
$$\Delta f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} + \frac{\partial^2 f}{\partial z^2} = 0.$$

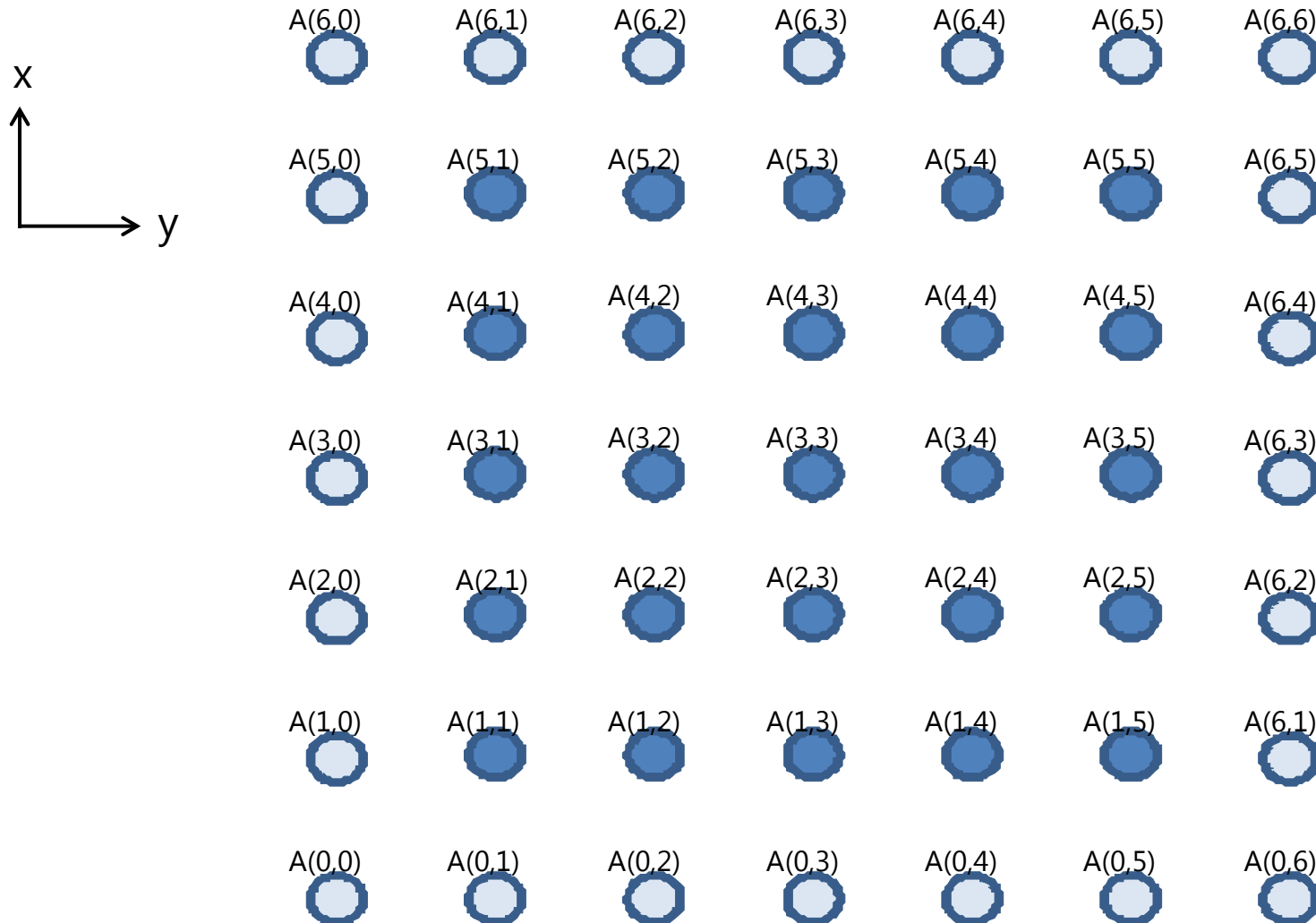
- Always find paths when path exists
- Generates smooth paths
- Generates non-optimal paths (length), but “good enough”



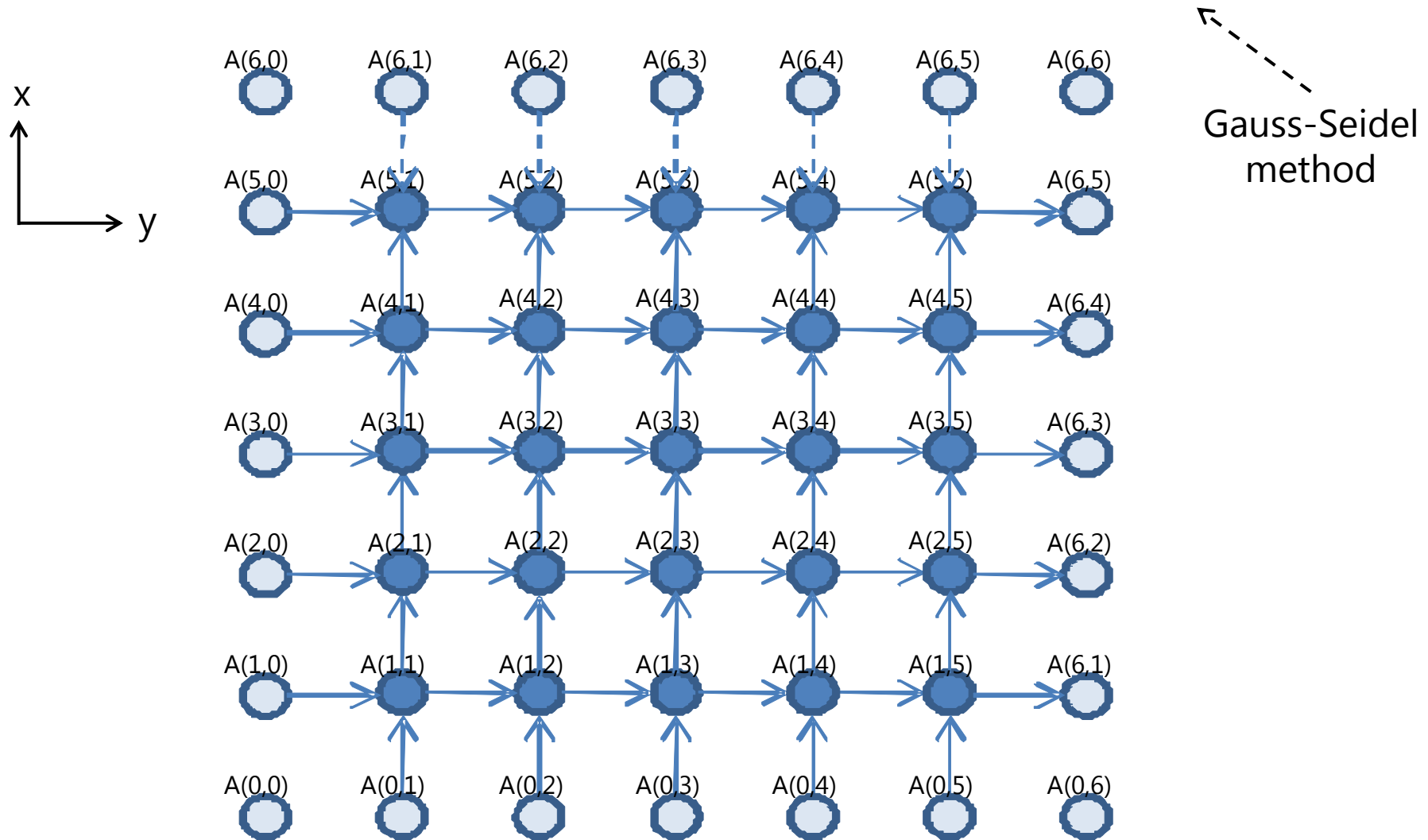
*Results obtained with a PowerPC 440 CPU*

- **Input:** obstacle map
- **Output:** potential grid
- **Steps:**
  1. If node  $(x,y)$  contains an obstacle, then  $u(x,y) = 0$
  2. If node  $(x,y)$  contains the goal point, then  $u(x,y) = -1$
  3. If node  $(x,y)$  is in interior, then iterate:  $u(x,y) = \text{average of neighbors}$



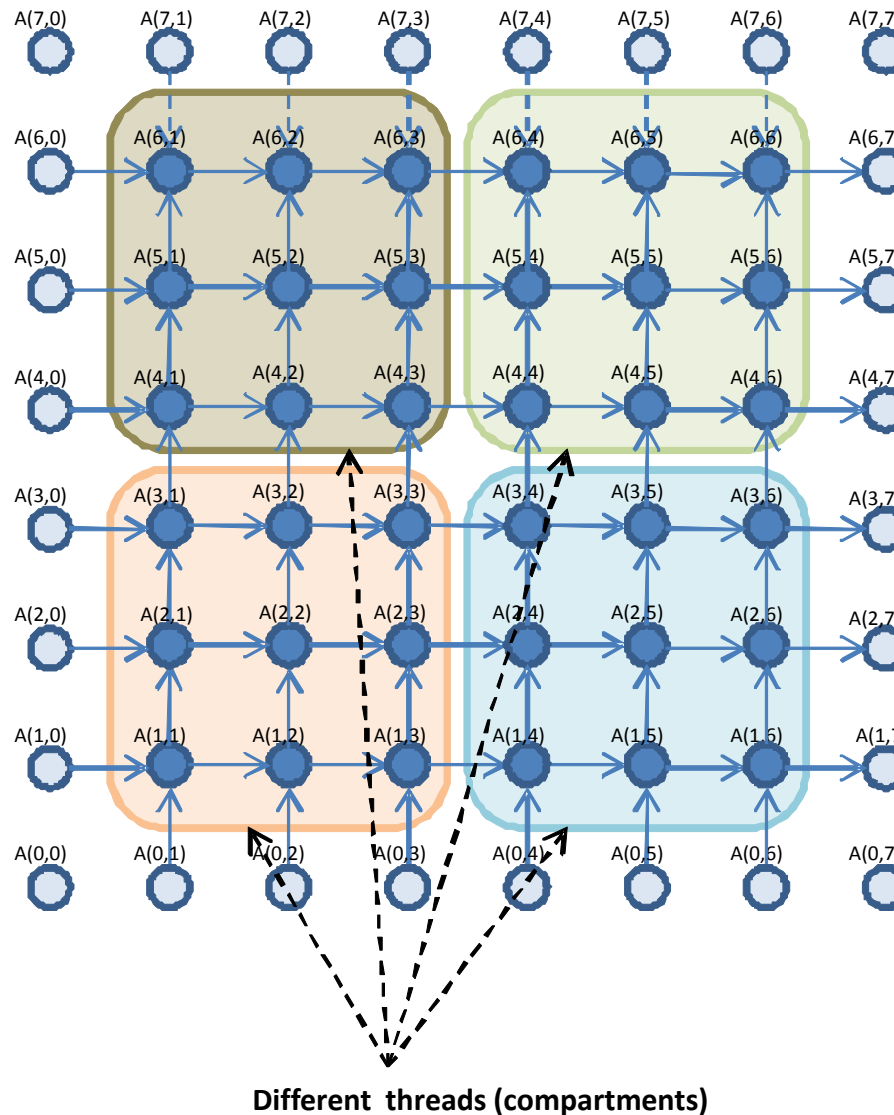


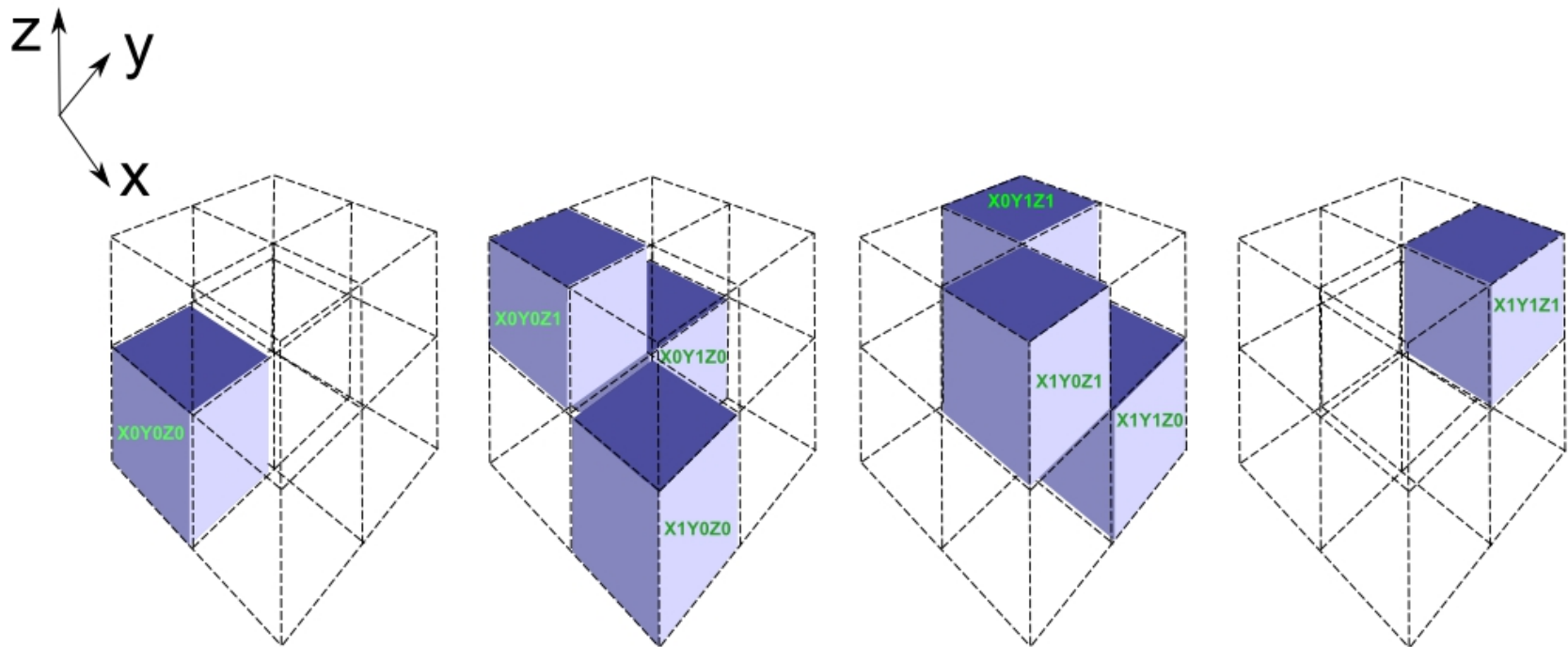
$$u^{n+1}(x_i, y_i) = \frac{1}{4} (u^{n+1}(x_{i-1}, y_i) + u^{n+1}(x_i, y_{i-1}) + u^n(x_{i+1}, y_i) + u^n(x_i, y_{i+1}))$$



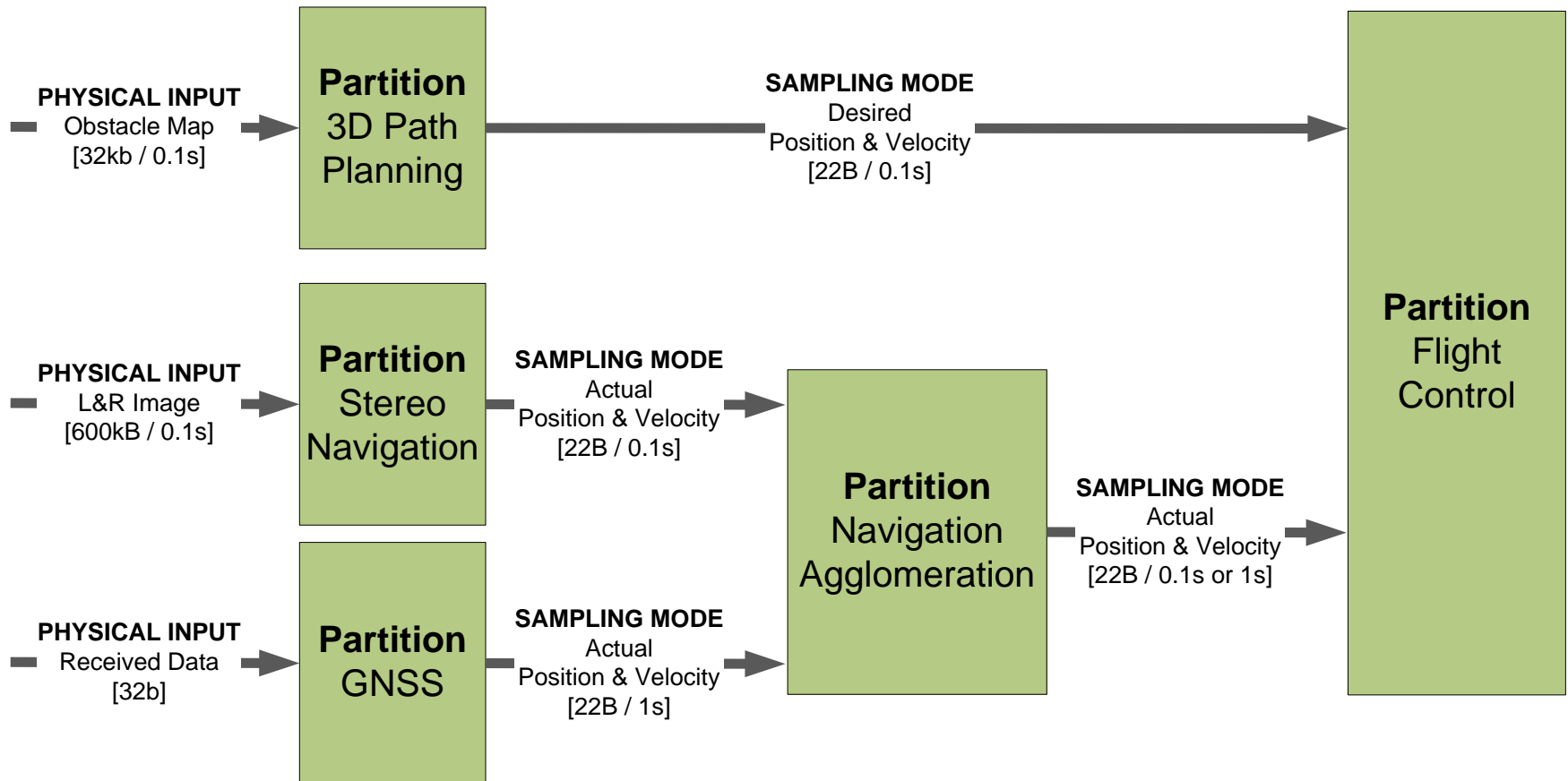
- 2D pipelined Laplacian matrix computation











- Past work
  - Applications were chosen and analysed
    - Prevalent parallel patterns were identified and detailed
  - Parallelisation approaches were specified
  - Initial modelling effort performed
  - Requirements were defined
- Current / Future work
  - Parallelisation efforts will continue
  - System-wide integration and mapping will be performed
  - WCET speedup will be assessed

# Parallelization of automotive software

Dipl.-Inf. Sebastian Kehr  
s.kehr@denso-auto.de



- › Goals for the project
- › Automotive Electronic Control Units (ECUs)
- › Diesel Engine Management System (EMS)
- › AUTOSAR reconstruction from legacy source code
- › Parallelization approach
- › Summary

1. Optimized multi-core use for short latencies
2. Keep timing predictable properties of single-core implementation
3. Support freedom from interference between Software-Components
4. Compliance with AUTOSAR standard + extensions



- Modern road vehicles carry up to 70 ECUs
  - Application Specific Integrated Circuits with multiple I/O devices
  - Robustness: temperature range, low error rate, long lasting (20 years)
  - Price is the limiting factor for hardware selection
  - › **Leads to limited computation and memory resources**

Runnable Entity

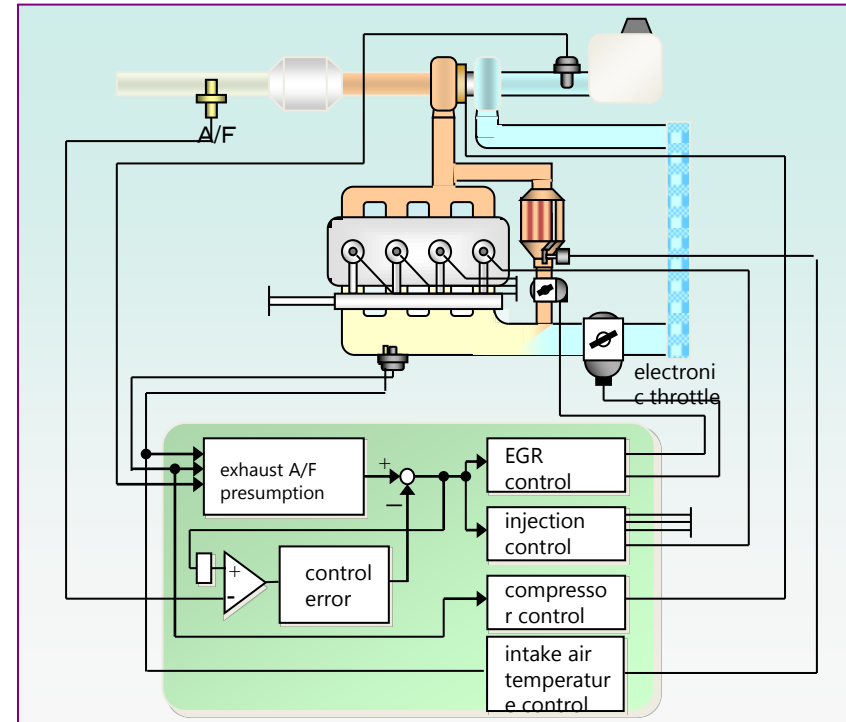


Task 128ms
Read_Temperature()
Check_Cooling_Required()
...

- Automotive software has evolved over years
  - One of the first industry standard was OSEK/VDX<sup>1</sup> OS
  - AUTOSAR established industry wide uniform understanding
  - › **Application has modular structure**
  - › **Data exchange over shared memory**
  - › **Static software configuration (tasks) – limited dynamic behaviour**



- 4-cylinder diesel engine
- About 30 sensors (temperature, ...)
- About 25 actuators (injection, ...)
- CAN interface
- Fuel injection controls engine revolutions
- Exhaust Gas Recirculation (EGR) to compress and heat up intake air



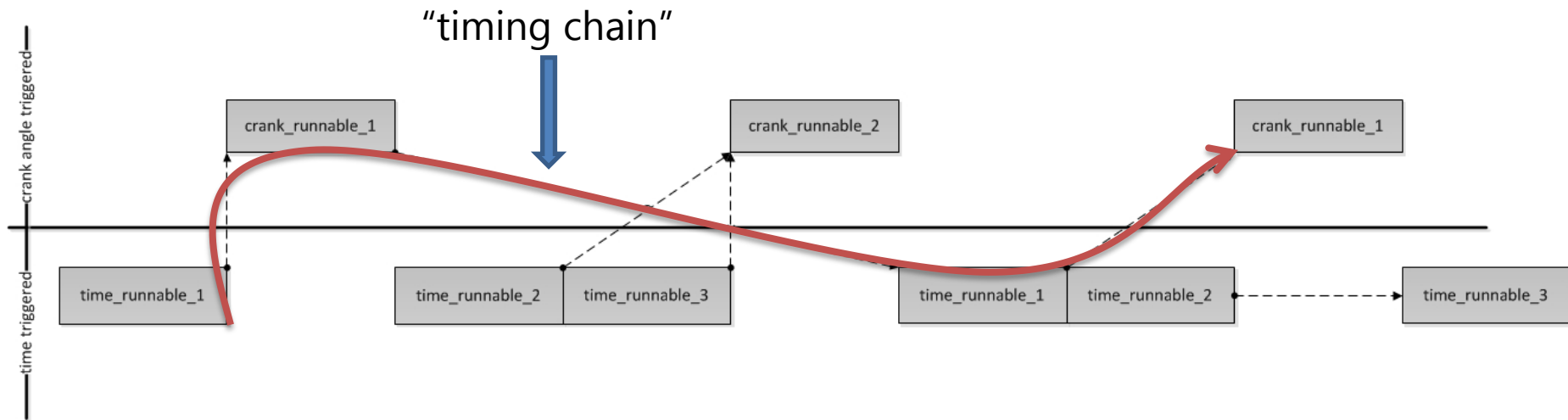
- › **Typical automotive application**
- › **Complex enough to have benefits from parallelization**
- › **Two time scales drift against each other: real-time vs. crank-angle**

- Limiting factor for parallelization are dependences between instructions
  - In legacy software global variables are usually only written by the modules that define them
  - Access from other modules through a function call
- The main emphasis is put on communication
  - AUTOSAR provides facilities to describe software
  - **Software-Components (SWCs)** communicate through ports with each other
- Description of structure and communication dependencies
  - SWCs are reconstructed from the file-hierarchy and aggregated in **Compositions**
  - Runnables and communication are found by analysis of the call trees



## Constraint for distribution of SWCs and scheduling

- Data flow analysis
  - Measurement traces using RapiTime
  - Reconstruct directed graph of data flow from sensors to actuators (depends on RPM)
- Statically calculated WCET of each Runnable using OTAWA



## ■ Prerequisites

1. Runnables are either triggered by time (cyclic) or after an event (crank-angle)
2. Data flow is constraint for parallelization
3. No updates of data during execution of a Runnables
  - Read before start and write after termination

## ■ Principles for parallelization

- Communication is used to de-couple SWCs
- Compositions are distributed over different clusters
- Communication between clusters via message passing
- Parallelization of a Runnable's call tree using techniques from HPC

- Automotive ECUs use application specific devices with limited resources
- Case study for parallelization is a diesel engine management system
  - Typical automotive and complex enough to gain benefit
- AUTOSAR reconstruction from legacy source code
  - Provides useful facilities to describe static software structure
  - Emphasis is put on communication as main source of dependence
- Parallelization approach
  - Data flow and timing analysis used as constraint for parallelization
  - Distribution of Composition to a set of clusters

## Outlook

- Data flow analysis of the complete system (RapiTime from Rapita)
- Implementation as parallel application based on time-triggered communication (Mapping tool in collaboration with BSC)
- Assessment of pattern based parallelization approach (UAU)



Thank you

