

From Single-Core to Multi-Core Platforms

Systematic Migration of Hard Real-Time Software in AUTOSAR

Embedded World 2011, Nuremberg

Peter Gliwa (Gliwa GmbH)

Jens Harnisch, Ursula Kelling (Infineon Technologies AG)

Christoph Ficek (SYMTAVISION GmbH)



Never stop thinking

Outline

- Motivation, assumptions and challenges
- Overall flow from single to multi-core software
- Metrics
- Iterative methodology
- Conclusions and outlook

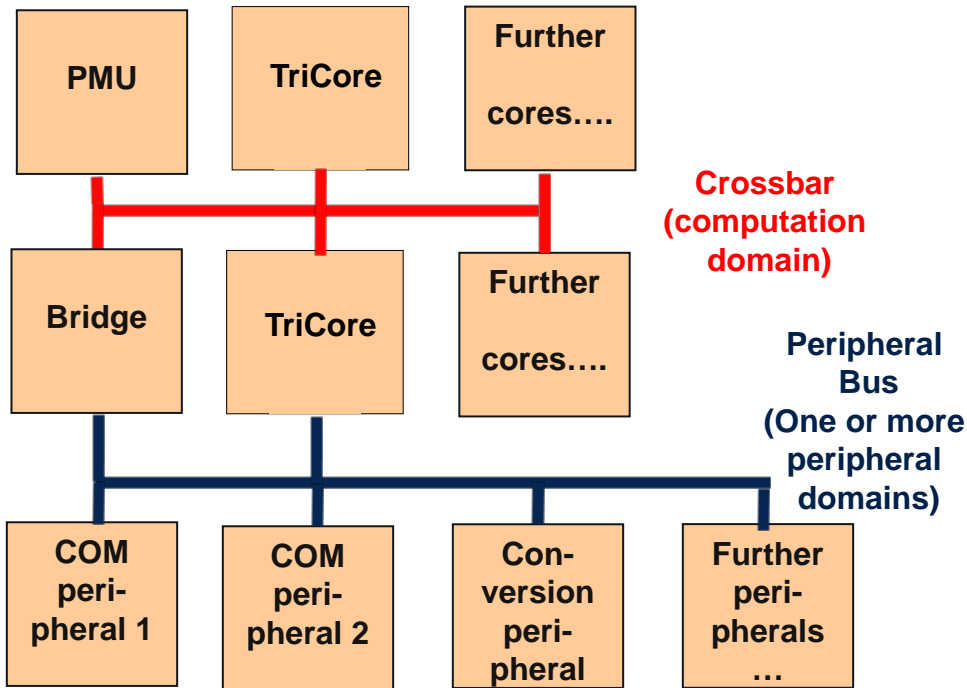
Motivation



- How to get multi-core on the road reliably and affordable?

- Several approaches for supporting multi-core hardware with parallel software available; however, still open questions for hard real time applications
- Even the metric for a successful porting of software from single core to multi-core is still unclear
- This presentation suggests
 - One possible approach for mapping runnables and data to cores and memories
 - Considering explicitly timing constraints and performance
 - Based upon a model for efficient exploration
 - With explicit support of AUTOSAR
 - Metrics to compare different software mappings

Some assumptions for an automotive multi-core system



- Please note: This is only a very abstract view. For details please get in touch with your Infineon contact.

- Heterogeneous systems can be better suited with regards to power consumption and hardware costs than homogenous systems
- Scheduling analysis on chip will become more complicated, with possibly different execution times on cores and communication overhead.
- **Infineon focus is not the 'biggest' multi-core system, but the system best suited for hard real time automotive applications including sophisticated multi-core debugging and tracing**

Practical challenges when moving to multi-core

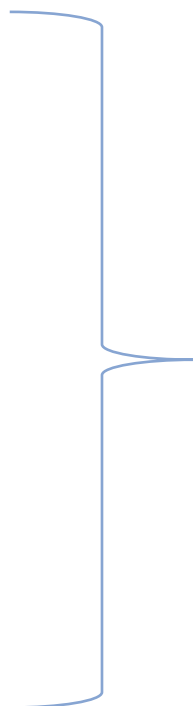


- Implicit assumptions in legacy software (e.g. non-preemptive tasks, dependencies through priorities)
- New hardware requires new compiler and new OS version
- Architectures are not always directly comparable to predecessors due to different clock rates, connectivity and memory characteristics
- Software mapping to cores uses new communication primitives
- **Many changes at once! The practice: Why is the performance not as expected?**
- Solution for investigation:
 - Use a model based approach with explicit consideration of timing
 - Use communication cost catalogue

Overall flow from single to multi-core software

- Start at high level: decomposition into runnables / tasks based on timing, application characteristics or other characteristics. Do not start with parallelization within functions!

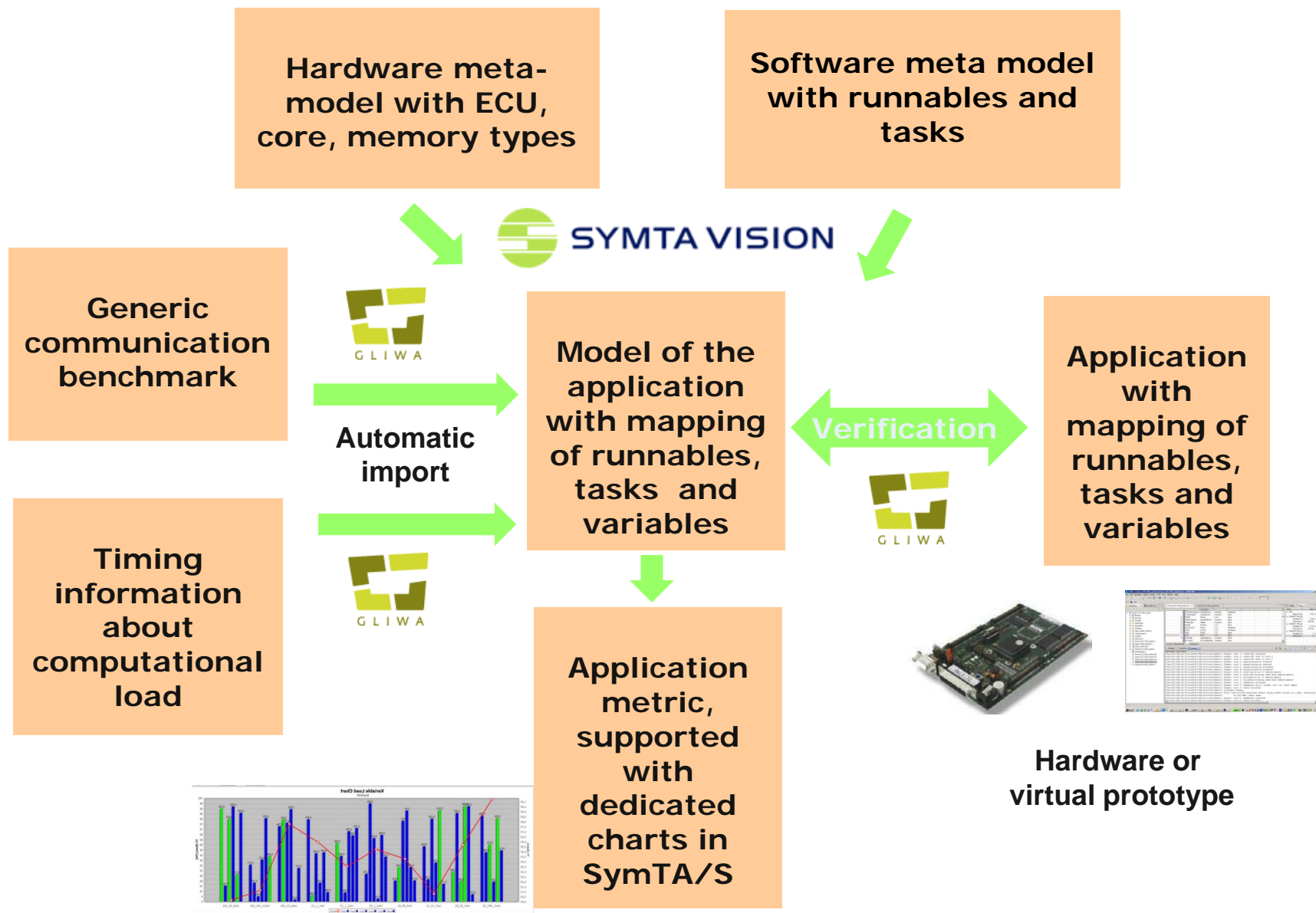
- Use the orchestration and mapping design space
 - Synchronization:
 - No explicit synchronization
 - Blocking synchronization with suspending or spinning
 - Task triggering
 - Priorities and Preemption:
 - SPNP, SPP
 - Data mapping and exchange
 - Consider local/ remote access times
 - Consider hardware arbitration at slaves
 - Mapping of tasks and runnables to cores



This should be considered as one large software design space

Tooling is mandatory for efficient design space exploration

Development flow for multi-core software development



Communication Benchmark

- Idea: Easy to handle benchmark with guidance for measurement
- Results of benchmark can be automatically processes

- Low level mechanisms (without OS involvement)
 - Read and write different data types (1, 2 and 4 byte) and
 - Different data sizes (multiples of the basic data types)
 - From/ to different memories (local and global) from the different cores

- AUTOSAR mechanisms (communication benchmark not fully implemented yet)
 - Read and write functions of the AUTOSAR RTE, again for different data sizes
 - IOC mechanisms (spin-locks, events, task activation, message passing)

Communication Benchmark

- Limitation of the benchmark: Idealized scenarios without conflicts at slaves and consideration of cache scenarios
- On the other hand: The hardware has several duplicated resources (memories) and makes it possible for the programmer to avoid conflicts

Result example:

```
<OverheadMeasurement CallDirection="write" Call="pure" MaxValue="5991"
MinValue="5991">
  <DataMemory Core = "0" Memory="DMI" Width="8bit" />
  <InstructionMemory Core = "0" Memory="PMI" />
  <!-- T1SwId="0" -->
</OverheadMeasurement>
```

Examples for timing related cost metrics on different levels



Platform level

- Average and min/max costs numbers from controller level numbers from controller level
- Efforts to guarantee schedulability on the different controller types of one platform

Core level

- IPC
- Task Deadline Margin
- Interrupt Handling Time
- Interrupt Latency Time

Controller level

- Averaged and min/max cost numbers from core level
- **Meet all deadlines**
- **Data rate and communication cost overhead**
- **Speedup against single core**
- Times when all cores are idle at the same time
- Application sensitiveness against jitter of different factors

Metrics on controller level

- ❑ Data rate metric
 - ❑ Calculate the exchanged data rates between runnables/tasks for a certain application
 - ❑ Independent from HW architecture

- ❑ Com-overheads with costs metric
 - ❑ Analyse the overhead times between runnables/tasks for a certain application
 - ❑ Consider HW architecture (memories, cores,..)

- ❑ Full blown scheduling analysis with overheads
 - ❑ Consider overheads in scheduling analysis

Data rate metric

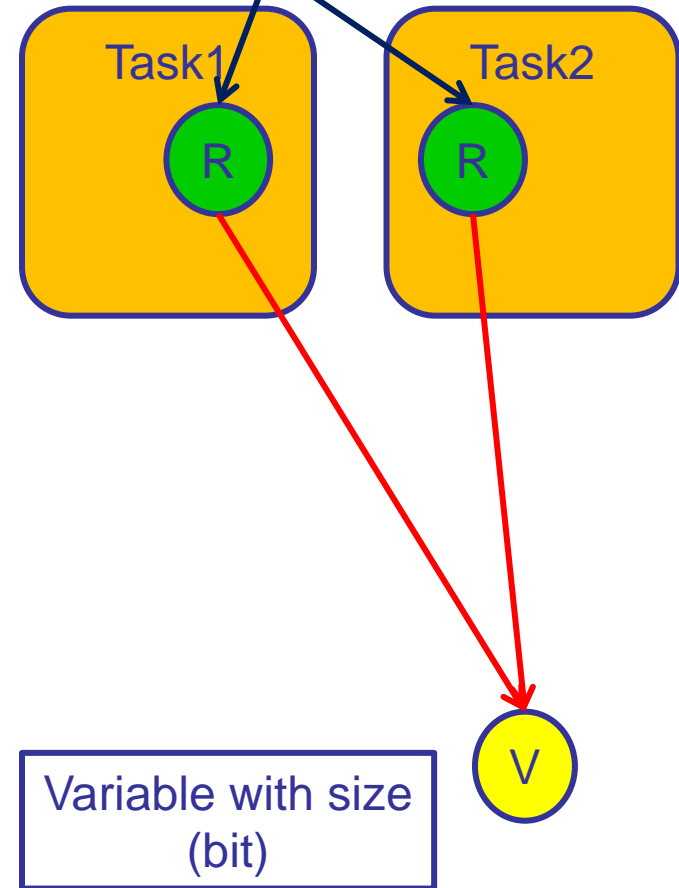
Activation frequency f (e.g. $1/\text{Period}$)

- ❑ The average rate of communication data transferred by runnables in bit/s
- ❑ Calculated by:
 - ❑ f : average activation frequency [1/s]
 - ❑ c : access count during execution
 - ❑ vs : variable/data size [bit]

→ $f * c * vs = \text{data rate [bit/s]}$

- ❑ Example:
 - ❑ $f = 1/20\text{ms} = 50/\text{s}$
 - ❑ $c = 2$ accesses of the data
 - ❑ $vs = 32\text{bit}$

→ $50/\text{s} * 2 * 32\text{bit} = 3200\text{bit/s}$



Data rate metric

- ❑ Already helps to quantify the data rates exchanged in the system

- ❑ Helps to make first mapping decisions
 - ❑ Assign runnables/tasks to specific cores
 - ❑ Assign data to specific memories

- Get runnables and data needed by them close together
 - ❑ E.g. local memory of the core executing the runnable
- Place runnables communicating a lot with each other on the same core
 - ❑ Constraints has to be considered

Com-overheads with costs metric

- ❑ Data rate metric extended by the cost catalogue:
 - ❑ Costs per access as overhead time
 - ❑ Outcome of the benchmark
 - ❑ Depending on HW, Compiler and OS

Needs to be measured individually

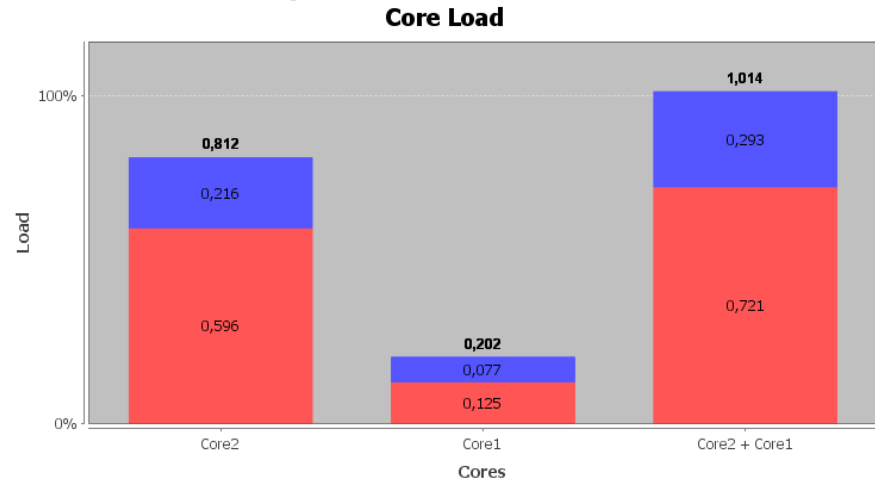
Call Type	Access Type	Instruction Core	Data Memory	Duration	Size [bit]
RTE	READ	Core1	DMI_Core1	X;Y [ns]	8
RTE	READ	Core2	DMI_Core1		16
RTE	WRITE	Core2	DMI_Core2		32
RTE	READ	Core1	EBU		64
IOC	WRITE	Core1	DMI_Core1		8
IOC	READ	Core1	LMU		16
PURE	WRITE	Core1	DMI_Core2		32
PURE	READ	Core1	DMI_Core1		64
...					

Has to cover all combinations

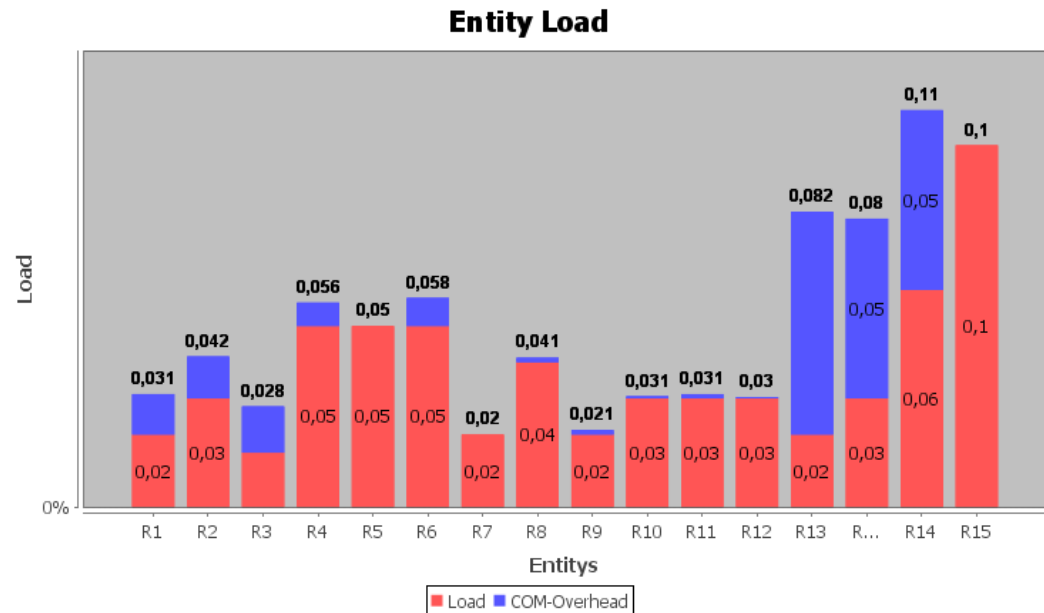
Com-overheads with costs metric

- Used to determine the additional **overhead time** per runnable/task
- And the **load** per runnable/task and (summarized) per core

Name	Buffer Access Overhead
R1	[76500 ns;112200 ns]
R2	[89675 ns;116025 ns]
R3	[70975 ns;127925 ns]
R4	[112625 ns;130000 ns]
R5	[48450 ns;61625 ns]
R6	[121200 ns;156000 ns]
R7	[42075 ns;49725 ns]



Processing load ; COM-Overhead



Com-overheads with costs metric

- Used to determine the load on each core per variable

Buffer	Memory	Mapped on Core	Accessing Core	Load
Var1_1_10	DMI_Core1	Core1	Core2	0.08466
Var2_1_10	DMI_Core1	Core1	Core2	0.08076
Var4_20_E0	DMI_Core1	Core1	Core1	0.00299625
Var4_20_E0	DMI_Core1	Core1	Core2	0.00378
Var5_E0_E1	DMI_Core1	Core1	Core1	0.0060775
Var6_10_100	DMI_Core1	Core1	Core2	0.007182

Com-overheads with costs metric

- Used to determine the load on each core per variable

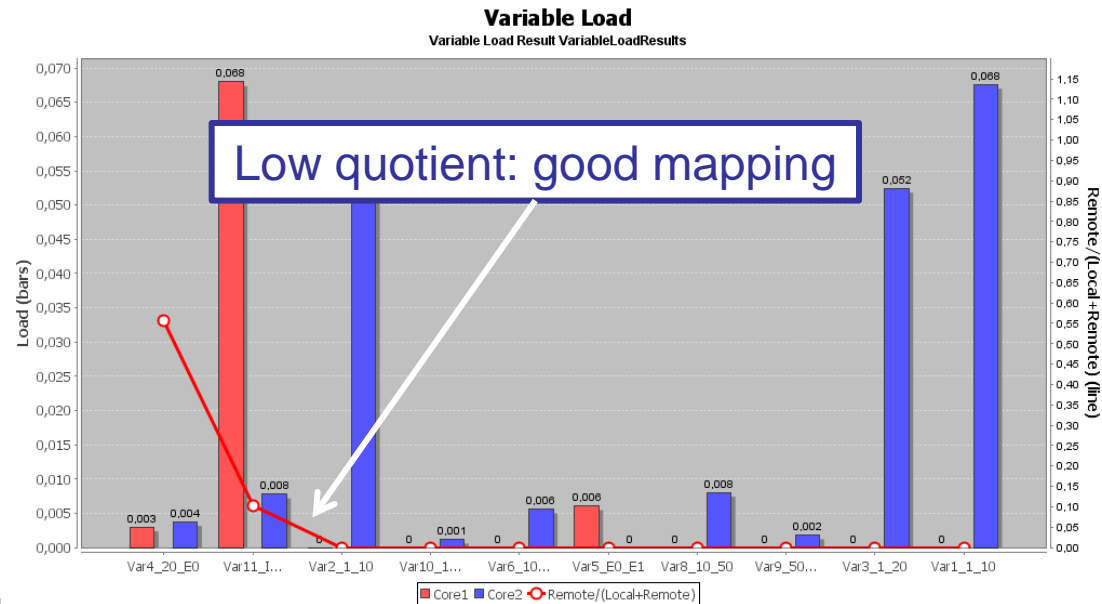
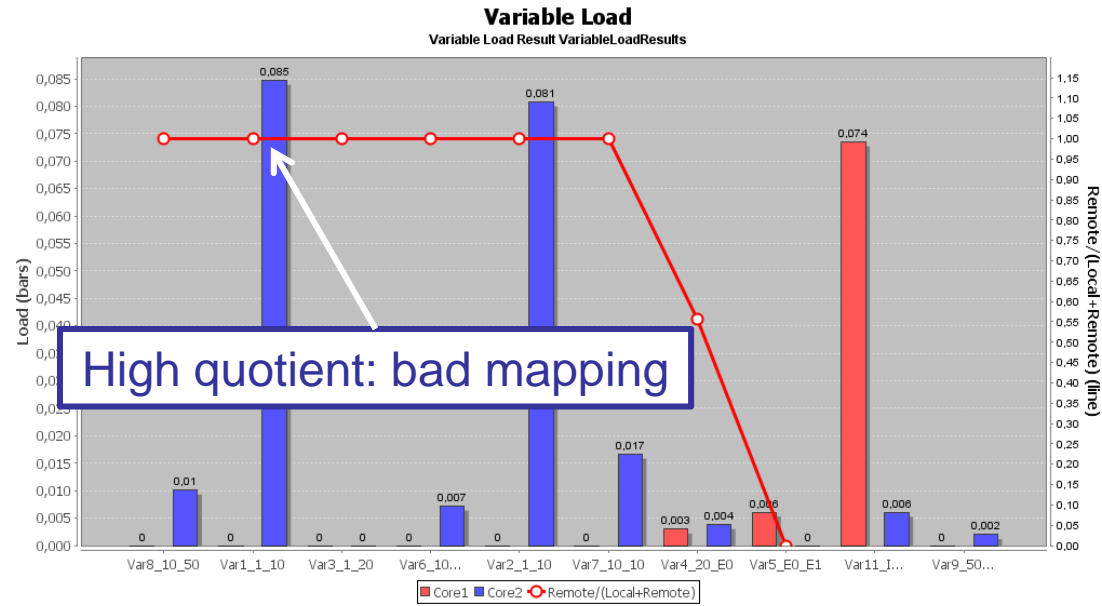
- As extended chart

- Quotient Q_i :

$$Q_i = \frac{\text{remote load}}{\text{remote load} + \text{local load}} \in [0..1]$$

- Makes mappings comparable

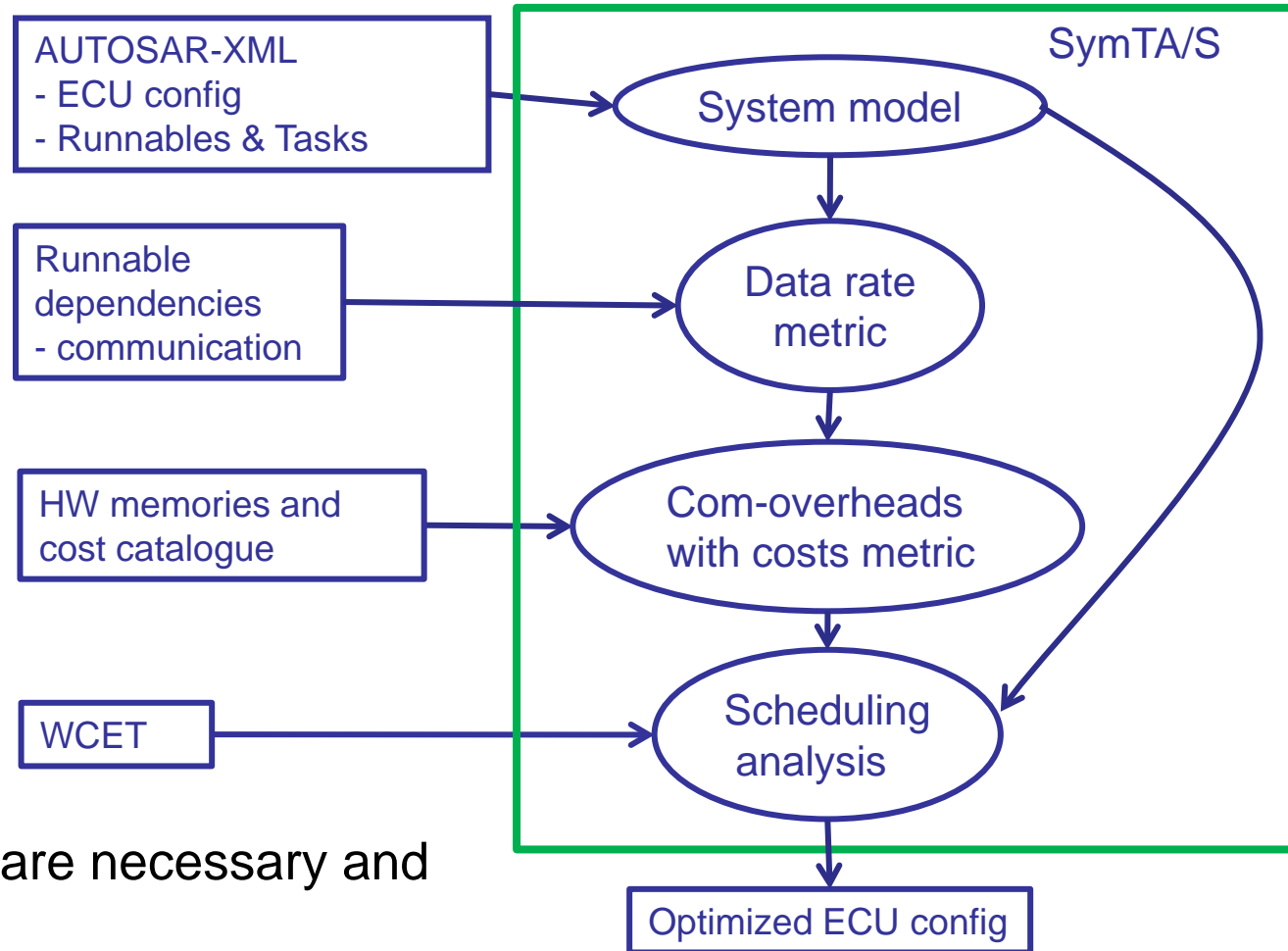
- indicates candidates for remapping



Scheduling analysis

- ❑ Distinguish between processing execution time (formally known as WCET) and communication overhead time
 - ❑ Consider both in the scheduling analysis
 - ❑ Check if all timing requirements are met
- A system balanced to load/overhead does not necessarily met all runnable/task deadlines
- Scheduling analysis is a must

Methodology



- ❑ Probably iterations are necessary and useful
- ❑ The model based approach allows fast answering of “what-if” questions
- ❑ Different mapping scenarios can be proven

Conclusion & Outlook

- ❑ New methodology for mapping applications to multi-core environments developed
 - ❑ Implementation in SymTA/S available for upcoming Infineon multi-core architectures
- ❑ The approach facilitates:
 - ❑ Fast exploration of design alternatives
 - ❑ Explicit visibility of communication overheads in system timing behaviour
 - ❑ Easy application in customer environment with specific operating system, compiler and HW-target
- ❑ Future work: Heuristic for automatic (re-)mapping of runnables/tasks to cores and variables to memories

Acknowledgements

- Ingo Houben (SYMTAVISION GmbH)
- Nico Kelling (Infineon Technologies AG)
- Dr. Albrecht Mayer (Infineon Technologies AG)
- Dr. Nick Merriam (Gliwa GmbH)
- Gerd Punsmann (Infineon Technologies AG)
- Dr. Kai Richter (SYMTAVISION GmbH)
- Klaus Scheibert (Infineon Technologies AG)
- Harry Siebert (Infineon Technologies AG)
- Arndt Voigtländer (Infineon Technologies AG)

Thank You!

ficek@symtavision.com
www.symtavision.com

Hall 10 - Booth 412
Real-Time Experts