Abstract

The Performance Design Methods described in this article are based on a multi-view approach. The needs are covered by a requirements view. The system design consists of a HW block diagram, a SW decomposition, a functional design and other models dependent on the type of system. The system design is used to create a performance model. Measurements provide a way to get a quantified characterization of the system. Different measurement methods and levels are required to obtain a usable characterized system. The performance model and the characterizations are used for the performance design. The system design decisions with great performance impact are: granularity, synchronization, prioritization, allocation and resource management. Performance and resource budgets are used as tool.
Positioning in CAFCR

<table>
<thead>
<tr>
<th>Customer What</th>
<th>Customer How</th>
<th>Product What</th>
<th>Product How</th>
</tr>
</thead>
<tbody>
<tr>
<td>C (Customer objectives)</td>
<td>A (Application)</td>
<td>F (Functional)</td>
<td>C (Conceptual)</td>
</tr>
</tbody>
</table>

- **Customer needs**
  - diverse
  - complex
  - fuzzy
  - performance
  - expectations

- **Product What**
  - SMART
    - timing requirements
    - external interfaces

- **Product How**
  - execution architecture design
    - threads
    - allocation
    - interrupts
    - scheduling
    - timers
    - synchronization
    - queues
    - decoupling

- **Models**
  - analysis
  - simulations
  - measurements

- **Additional Models**
  - analysis
  - simulations
  - measurements
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1A Collect most critical performance and timing requirements</strong></td>
<td></td>
</tr>
<tr>
<td><strong>1B Find system level diagrams</strong></td>
<td>HW block diagram, SW diagram, functional model(s) concurrency model, resource model, time-line</td>
</tr>
<tr>
<td><strong>2A Measure performance at 3 levels</strong></td>
<td>application, functions and micro benchmarks</td>
</tr>
<tr>
<td><strong>2B Create Performance Model</strong></td>
<td></td>
</tr>
<tr>
<td><strong>3 Evaluate performance, identify potential problems</strong></td>
<td></td>
</tr>
<tr>
<td><strong>4 Performance analysis and design</strong></td>
<td>granularity, synchronization, prioritization, allocation, resource management</td>
</tr>
<tr>
<td><strong>Re-iterate all steps</strong></td>
<td>are the right requirements addressed, refine diagrams, measurements, models, and improve design</td>
</tr>
</tbody>
</table>

Performance Method Fundamentals
Gerrit Muller

version: 0.2
June 5, 2018
PMFtopLevel
Incremental Approach

- measure
- evaluate
- analyse

- simulate
- build proto

- determine most important and critical requirements
- model
- analyse constraints and design options
Decomposition of System TR in HW and SW

most and hardest TR handled by HW

new control TRs

system TR

Hardware TR

software TR

Performance Method Fundamentals
5 Gerrit Muller

version: 0.2
June 5, 2018
EAAhwswRequirements
Quantification Steps

- Order of magnitude
- Guestimates
- Calibrated estimates

Feasibility measure,
analyze,
simulate

Back of the envelope,
benchmark,
spreadsheet calculation

Cycle accurate

99.999 - 100.001
Iteration

- zoom in on detail
- aggregate to end-to-end performance
- from coarse guestimate to reliable prediction
- from typical case to boundaries of requirement space
- from static understanding to dynamic understanding
- from steady state to initialization, state change and shut down

- discover unforeseen critical requirements
- improve diagrams and designs
- from old system to prototype to actual implementation
Construction Decomposition

- Applications:
  - View
  - PIP
  - Adjust
  - View TXT

- Services:
  - Viewport
  - Menu
  - Audio
  - Video
  - TXT
  - ETC
  - Browse
  - Networking
  - File system

- Toolboxes:
  - Drivers
  - Scheduler
  - Tuner
  - Frame buffer
  - MPEG
  - DSP

- Hardware:
  - Signal processing subsystem
  - Control subsystem

- Domain Specific

- Generic
Functional Decomposition

- Acquisition
- Acquisition processing
- Compress
- Encoding
- Storage
- Decoding
- Display processing
- Display
- Decompress
An example of a process decomposition of a MRI scanner.

Diagram showing the process decomposition with the following components:

- **Scan Control**
  - Scan UI
  - Scan control
  - Acq control
  - Recon control
  - xDAS
  - Recon
  - Disk

- **Image Handling**
  - Image handling UI
  - Archiving control
  - Import export
  - Export control
  - Display control
  - Media
  - Network
  - Display

Legend:
- UI process
- Server process
- Device hardware
Combine views in Execution Architecture

- **dead lines**
- **timing**, **throughput**
- **requirements**

**functional model**
- receive
- demux
- display
- store
- process

**hardware**
- CPU
- DSP
- RAM
- tuner
- drive

**repository structure**
- Applications
- play
- zap
- list
- UI toolkit
- menu
- processing
- DCT
- foundation classes
- queue
- list
- hardware abstraction
- tuner
- DVD drive

**other architecture views**
- process
- task
- thread
- interrupt handlers

**execution architecture issues:**
- concurrency
- scheduling
- synchronisation
- mutual exclusion
- priorities
- granularity
Layered Benchmarking Approach

typical values
interference
variation
boundaries

end-to-end function

duration services
interrupts
task switches
OS services
CPU time
footprint
cache

applications

network transfer
database access
database query services/functions

interrupt
task switch
OS services
duration
footprint

CPU

cache

memory

bus

.. (computing) hardware

latency

bandwidth

efficiency

operating system

locality
density
efficiency
overhead

tools

Performance Method Fundamentals

version: 0.2
June 5, 2018
EBMibenchmarkStack
## Micro Benchmarks

<table>
<thead>
<tr>
<th>Category</th>
<th>Infrequent Operations, Often Time-Intensive</th>
<th>Often Repeated Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>database</strong></td>
<td>start session</td>
<td>perform transaction query</td>
</tr>
<tr>
<td></td>
<td>finish session</td>
<td></td>
</tr>
<tr>
<td><strong>network, I/O</strong></td>
<td>open connection</td>
<td>transfer data</td>
</tr>
<tr>
<td></td>
<td>close connection</td>
<td></td>
</tr>
<tr>
<td><strong>high level construction</strong></td>
<td>component creation</td>
<td>method invocation</td>
</tr>
<tr>
<td></td>
<td>component destruction</td>
<td>same scope, other context</td>
</tr>
<tr>
<td><strong>low level construction</strong></td>
<td>object creation</td>
<td>method invocation</td>
</tr>
<tr>
<td></td>
<td>object destruction</td>
<td></td>
</tr>
<tr>
<td><strong>basic programming</strong></td>
<td>memory allocation</td>
<td>function call</td>
</tr>
<tr>
<td></td>
<td>memory free</td>
<td>loop overhead</td>
</tr>
<tr>
<td></td>
<td></td>
<td>basic operations (add, mul, load, store)</td>
</tr>
<tr>
<td><strong>OS</strong></td>
<td>task, thread creation</td>
<td>task switch</td>
</tr>
<tr>
<td></td>
<td></td>
<td>interrupt response</td>
</tr>
<tr>
<td><strong>HW</strong></td>
<td>power up, power down</td>
<td>cache flush</td>
</tr>
<tr>
<td></td>
<td>boot</td>
<td>low level data transfer</td>
</tr>
</tbody>
</table>
The ASP™ course is partially derived from the EXARCH course developed at *Philips CTT* by *Ton Kostelijk* and *Gerrit Muller*.

Extensions and additional slides have been developed at *ESI* by *Teun Hendriks*, *Roland Mathijssen* and *Gerrit Muller*. 