Performance Method Fundamentals

by Gerrit Muller      Buskerud University College
    e-mail: gaudisite@gmail.com
    www.gaudisite.nl

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.

The complete course ASP™ is owned by Buskerud University College. To teach this course a license from Buskerud University College is required. This material is preliminary course material. The final material and course information can be found at: www.esi.nl/cursus.

March 6, 2013
status: draft
version: 0.2
Positioning in CAFCR

What does Customer need in Product and Why?

Customer
What
Customer objectives

Customer
How
Application

Product
What
Functional

Product
How
Conceptual

Realization

SMART
+ timing
requirements
+ external
interfaces

execution architecture
design
threads
interrupts
timers
queues
allocation
scheduling
synchronization
decoupling

models
analysis

simulations
measurements

diverse
complex
fuzzy
performance
expectations
needs

Performance Method Fundamentals
©2006, Embedded Systems Institute
version: 0.2
March 6, 2013
EAAandCAFCR

©2006, Gerrit Muller
# Toplevel Performance Design Method

1A Collect most critical performance and timing requirements

1B Find system level diagrams
   - HW block diagram, SW diagram, functional model(s)
   - concurrency model, resource model, time-line

2A Measure performance at 3 levels
   - application, functions and micro benchmarks

2B Create Performance Model

3 Evaluate performance, identify potential problems

4 Performance analysis and design
   - granularity, synchronization, prioritization,
     allocation, resource management

Re-iterate all steps
   - are the right requirements addressed,
     refine diagrams, measurements, models, and improve design
Incremental Approach

- determine most important and critical requirements
- model
- analyse constraints and design options
- simulate
- build proto
- measure
- evaluate
- analyse

Performance Method Fundamentals
©2006, Embedded Systems Institute
©2006, Gerrit Muller
Decomposition of System TR in HW and SW

most and hardest TR handled by HW

new control TRs

system TR

hardware TR

software TR

ns
us
ms
s

©2006, Embedded Systems Institute

version: 0.2
March 6, 2013
EAAhwswRequirements
Quantification Steps

- back of the envelope
- benchmark, spreadsheet calculation
- measure, analyze, simulate
- cycle accurate

- order of magnitude
- guestimates
- calibrated estimates
- feasibility
- measure, analyze, simulate

- back of the envelope
- order of magnitude
- guestimates
- calibrated estimates
- feasibility
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
Functional Decomposition

acquisition -> acquisition processing -> compress -> encoding

encoding -> storage

storage -> decoding

display -> display processing -> decompress -> decoding
An example of a process decomposition of a MRI scanner.
Combine views in Execution Architecture

Issues:
- Concurrency
- Scheduling
- Synchronisation
- Mutual exclusion
- Priorities
- Granularity

Functional Model
- Receive
- Demux
- Display
- Store
- Process

Execution Architecture
- Process
- Task
- Thread
- Interrupt handlers

Hardware
- CPU
- DSP
- RAM
- Tuner
- Drive

Repository Structure
- Applications
- UI toolkit
- Processing
- Foundation classes
- Hardware abstraction
- Decoder
- Tuner
- DVD drive

Deadlines
- Timing
- Throughput
- Requirements

Other Architecture Views

Performance Method Fundamentals
©2006, Embedded Systems Institute
version: 0.2
March 6, 2013
CVexecutionArchitecture
Layered Benchmarking Approach

typical values
interference
variation
boundaries

CPU
cache
memory
bus
..
(computing) hardware
typical values
interference
variation
boundaries

network transfer
database access
database query
services/functions

end-to-end
function
duration
services
interrupts
task switches
OS services
CPU time
footprint
cache

applications

interrupt
task switch
OS services

duration
footprint
cache

services

operating system

(locality
density
efficiency
overhead)

tools

latency
bandwidth
efficiency

(duration
footprint
.cache

OS services

interrupts
task switches
OS services

EBMbenchmarkStack

Performance Method Fundamentals
©2006, Embedded Systems Institute
## Micro Benchmarks

<table>
<thead>
<tr>
<th>Category</th>
<th>Function</th>
<th>Frequency/Intensity</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>infrequent operations,</strong></td>
<td><strong>often time-intensive</strong></td>
<td></td>
</tr>
<tr>
<td>database</td>
<td>start session</td>
<td>perform transaction</td>
</tr>
<tr>
<td></td>
<td>finish session</td>
<td>query</td>
</tr>
<tr>
<td>network, I/O</td>
<td>open connection</td>
<td>transfer data</td>
</tr>
<tr>
<td></td>
<td>close connection</td>
<td></td>
</tr>
<tr>
<td>high level construction</td>
<td>component creation</td>
<td>method invocation</td>
</tr>
<tr>
<td></td>
<td>component destruction</td>
<td>same scope</td>
</tr>
<tr>
<td></td>
<td></td>
<td>other context</td>
</tr>
<tr>
<td>low level construction</td>
<td>object creation</td>
<td>method invocation</td>
</tr>
<tr>
<td></td>
<td>object destruction</td>
<td></td>
</tr>
<tr>
<td>basic programming</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>OS</td>
<td>task, thread creation</td>
<td>task switch</td>
</tr>
<tr>
<td></td>
<td></td>
<td>interrupt response</td>
</tr>
<tr>
<td>HW</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>

- **infrequent operations,** **often time-intensive**
- **often repeated operations**
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.