Abstract
The course Architecting System Performance provides an approach to design performance for software intensive systems. Core to the approach is the combination of measuring and modeling. Models are used for reasoning and analysis of performance, scalability, sensitivity and robustness. The course emphasis is on practice, not on theory. For example patterns and pitfalls from practice are provided.
Abstract
What is System Performance? Why should a software engineer have knowledge of the other parts of the system, such as the Hardware, the Operating System and the Middleware? The applications that he/she writes are self-contained, so how can other parts have any influence? This introduction sketches the problem and shows that at least a high level understanding of the system is very useful in order to get optimal performance.
content of this presentation

Example of problem

Problem statements
Image Retrieval Performance

Sample application code:

for x = 1 to 3 {
    for y = 1 to 3 {
        retrieve_image(x,y)
    }
}

alternative application code:

event 3*3 -> show screen 3*3

<screen 3*3>
    <row 1>
        <col 1><image 1,1></col 1>
        <col 2><image 1,2></col 2>
        <col 3><image 1,3></col 3>
    </row 1>
</screen 3*3>

application need:

at event 3*3 show 3*3 images
instantaneous

or

Sample application code:
What If....

Sample application code:

```
for x = 1 to 3 {
    for y = 1 to 3 {
        retrieve_image(x,y)
    }
}
```

UI process

store

screen
More Process Communication

What If....

Sample application code:
for x = 1 to 3 {
  for y = 1 to 3 {
    retrieve_image(x,y)
  }
}

UI process

screen server

9 *
update

screen

database

9 *
retrieve
Meta Information Realization Overhead

What If....

Sample application code:

```java
for x = 1 to 3 {
    for y = 1 to 3 {
        retrieve_image(x,y)
    }
}
```

Attribute = 1 COM object
100 attributes / image
9 images = 900 COM objects
1 COM object = 80µs
9 images = 72 ms
What If....

Sample application code:

```c
for x = 1 to 3 {
  for y = 1 to 3 {
    retrieve_image(x,y)
  }
}
```

- I/O on line basis (512^2 image)

\[ 9 \times 512 \times t_{I/O} \]

\[ t_{I/O} \sim= 1 \text{ms} \]
Sample application code:

```cpp
for x = 1 to 3 {
    for y = 1 to 3 {
        retrieve_image(x,y)
    }
}
```

can be:
- fast, but very local
- slow, but very generic
- slow, but very robust
- fast and robust
- ...

The emerging properties (behavior, performance) cannot be seen from the code itself!

Underlying platform and neighbouring functions determine emerging properties mostly.
Function in System Context

Performance and behavior of a function depend on realizations of used layers, functions in the same context, and the usage context.

Middleware
Operating systems
Hardware
<table>
<thead>
<tr>
<th>F &amp; S</th>
<th>F &amp; S</th>
<th>F &amp; S</th>
<th>F &amp; S</th>
<th>F &amp; S</th>
<th>F &amp; S</th>
<th>F &amp; S</th>
</tr>
</thead>
<tbody>
<tr>
<td>MW</td>
<td>MW</td>
<td>MW</td>
<td>MW</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OS</td>
<td>OS</td>
<td>OS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HW</td>
<td>HW</td>
<td>HW</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Functions & Services

Middleware

Operating systems

Hardware

Performance = Function (F&S, other F&S, MW, OS, HW)

MW, OS, HW >> 100 Manyear : very complex

Challenge: How to understand MW, OS, HW with only a few parameters
Summary of Introduction to Problem

Resulting System Characteristics cannot be deduced from local code.

Underlying platform, neighboring applications and user context:

have a big impact on system characteristics

are big and complex

Models require decomposition, relations and representations to analyse.
From Synchronous to Asynchronous Design

by Gerrit Muller    Buskerud University College

e-mail: gaudisite@gmail.com

www.gaudisite.nl

Abstract
The most simple real time programming paradigm is a synchronous loop. This is an effective approach for simple systems, but at a certain level of concurrent activities an asynchronous design, based on scheduling tasks, becomes more effective. We will use a conventional television as case to show real time design strategies, starting with a straightforward analog television based on a synchronous design and incrementally extending the television to become a full-fledged digital TV with many concurrent functions.
Hard Real Time Design

- **Hard Real Time**
  - Disastrous failure
  - Human safety
  - Device safety
  - Loss of information

- **Soft Real Time**
  - Dissatisfaction
  - Irritation
  - Limited throughput
  - Waiting time
  - Loss of eye hand coordination
  - Loss of functionality or (image) quality

From Synchronous to Asynchronous Design

©2006, Embedded Systems Institute
C2006, Gerrit Muller

Version: 0
July 31, 2014
PHRTpositioning
Simple Analog TV

Multiple views on system
Fundamentals of  *periodic* or *streaming* Hard Real-Time applications
System performance characterisation: Performance model
Synchronous design concept
Functional Flow Simple Analog Television

- Tuner
demux

- Video signal de-mux

- Audio processing

- Line demux: ~ 60µs

- Bit detection ~ 150 ns

- Teletext processing

- Picture processing

- User Interface ~100 ms

- User i/f graphics generation

- Teletext overlay generation

- Video signal mux

- Audio / video sync ~ 20ms

- Control

- User Interface ~100 ms
  ~1.8ms / bit
From Synchronous to Asynchronous Design
©2006, Embedded Systems Institute
©2006, Gerrit Muller

Version: 0
July 31, 2014
PHRT television SW construction
Video Timing

For PAL-625:

- Line Frequency: 15.625 kHz
- Scanning Lines: 625
- Field Frequency: 50 Hz

Hidden lines (can contain data)
Audio-Video Synchronization Requirement

Images:
Discrete in time

Sound:
Continuous in time

Latency
Sound and vision must be lip-sync or better
Maximum latency ~ +/- 100 msec

0 ms  40 ms  80 ms
Time

From Synchronous to Asynchronous Design
©2006, Embedded Systems Institute
©2006, Gerrit Muller
Synchronous Control Software

Synchronous design

Frame interrupt

Capture teletext
Initiate video proc.
Initiate audio proc.
Check user input
Do User Interface
Display teletext (when active)
Check status (HW)

Frame interrupt

20 msec
Synchronous design questions

Estimate processing time on a 100 MHz ARM core
Assuming that all processing and acquisition is done in HW
Graphics rendering (user interface + teletext display) is done in SW

Where do you expect variation?

How feasible and how reliable is this design?
Low Priority Work in the Background

Design with multiple parallel tasks

- Do User Interface
- Display teletext (when active)
- Check status (HW)
- Do User Interface
- Display teletext (when active)
- Check status (HW)

Frame interrupt

image processing
Parallel / background tasks

20 msec

20 msec
Synchronous or Asynchronous?

**Synchronous**

=> Map on Highest frequency

Constraints:
- Processing frequency must be a whole (integer) multiple of the lower frequencies
- Each process must be completed within the period of the highest frequency, together with the high-frequency process

**A-Synchronous**

=> Concurrent processes
### Multiple Periods in a Simple TV

<table>
<thead>
<tr>
<th>Category</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input signal</td>
<td>50 Hz</td>
</tr>
<tr>
<td>Processing</td>
<td>100 Hz</td>
</tr>
<tr>
<td>User Interface</td>
<td>20 Hz</td>
</tr>
<tr>
<td>Power and Housekeeping</td>
<td>0.5 Hz</td>
</tr>
<tr>
<td>Output</td>
<td>50, 100 Hz</td>
</tr>
</tbody>
</table>
Simple Analog TV

Performance model requires:
- identification of processing steps
- their relation
- critical parameters and values

Synchronous design sufficient for periodic applications with one dominant frequency

Multiple views on system:
- HW diagram
- SW construction diagram
- Functional flow
- Time-line
Case Digital Television

From Analog TV to Digital TV

Adding more input formats and output devices

Multiple heterogenous periods: asynchronous design with concurrent tasks.
## Digital Television

<table>
<thead>
<tr>
<th>Input</th>
<th>Many frequencies</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Video &amp; Audio variable timing</td>
</tr>
<tr>
<td>Output</td>
<td>Many frequencies</td>
</tr>
<tr>
<td>Processing</td>
<td>Variable</td>
</tr>
</tbody>
</table>

Many video variants (see table)
Many audio variants (quality, number of speakers, ...)

---

©2006, Embedded Systems Institute
©2006, Gerrit Muller
In modern television the format of the image can change (e.g. widescreen)
The user can set the refresh rate to higher values (e.g. 100Hz anti-flicker)
Different displays (CRT, LCD, Plasma) can be attached
that need the image in different formats
(interlaced, non-interlaced, different refresh rates)
Non interlaced images need special filtering of the image
to prevent ragged images
<table>
<thead>
<tr>
<th>spec</th>
<th>Horizontal pixels</th>
<th>Vertical pixels</th>
<th>Aspect ratio</th>
<th>Monitor interface</th>
<th>Format name</th>
<th>Frames per sec</th>
<th>Fields per sec</th>
<th>Transmitted interlaced</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATSC</td>
<td>1920</td>
<td>1080</td>
<td>16:09</td>
<td>1080i</td>
<td>1080i60</td>
<td>30</td>
<td>60</td>
<td>yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1080p30</td>
<td>30</td>
<td>30</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1080p24</td>
<td>24</td>
<td>24</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>1280</td>
<td>720</td>
<td>16:09</td>
<td>720p</td>
<td>720p60</td>
<td>60</td>
<td>60</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>720p30</td>
<td>30</td>
<td>30</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>720p24</td>
<td>24</td>
<td>24</td>
<td>no</td>
</tr>
<tr>
<td>NTSC</td>
<td>720</td>
<td>480</td>
<td>16:09</td>
<td>480p</td>
<td>480p60</td>
<td>60</td>
<td>60</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>480</td>
<td>480</td>
<td>16:09</td>
<td>480p</td>
<td>480i60</td>
<td>30</td>
<td>60</td>
<td>yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>480p30</td>
<td>30</td>
<td>30</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>480p24</td>
<td>24</td>
<td>24</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>640</td>
<td>480</td>
<td>04:03</td>
<td>480p</td>
<td>480i60</td>
<td>30</td>
<td>60</td>
<td>yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>480p30</td>
<td>30</td>
<td>30</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>480p24</td>
<td>24</td>
<td>24</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>480</td>
<td>480</td>
<td>04:03</td>
<td>480p</td>
<td>480i60</td>
<td>30</td>
<td>60</td>
<td>yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>480p30</td>
<td>30</td>
<td>30</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>480p24</td>
<td>24</td>
<td>24</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>480</td>
<td>480</td>
<td>04:03</td>
<td>480p</td>
<td>480i60</td>
<td>30</td>
<td>60</td>
<td>yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>480p30</td>
<td>30</td>
<td>30</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>480p24</td>
<td>24</td>
<td>24</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>640</td>
<td>483</td>
<td>04:03</td>
<td>Note 1</td>
<td>Note 1</td>
<td>30</td>
<td>60</td>
<td>yes</td>
</tr>
</tbody>
</table>

Note 1: Some people refer to NTSC as 480i.

Source: [http://www.hdtvprimer.com/ISSUES/what_is_ATSC.html](http://www.hdtvprimer.com/ISSUES/what_is_ATSC.html)
### Data Packets in Digital TV

<table>
<thead>
<tr>
<th>Data</th>
<th>Compr. Audio</th>
<th>Compressed Video</th>
<th>Data</th>
<th>Compr. Audio</th>
<th>Compr. Video</th>
<th>Data</th>
</tr>
</thead>
</table>

Packet

Reference Frame

△ Frame

△ Frame

△ Frame

△ Frame

△ Frame

Reference Frame

Bi-directional
Dependence

Prediction
From Analog TV to Digital TV

Real-life applications rapidly introduce all kinds of variations
Concurrent tasks cope with different periods
Abstract
A simple measurement exercise is described. Purpose of this exercise is to build up experience in measuring and its many pitfalls. The programming language Python is used as platform, because of its availability and low threshold for use.

Distribution
This article or presentation is written as part of the Gaudí project. The Gaudí project philosophy is to improve by obtaining frequent feedback. Frequent feedback is pursued by an open creation process. This document is published as intermediate or nearly mature version to get feedback. Further distribution is allowed as long as the document remains complete and unchanged.
Select a programming environment, where loop overhead and file open can be measured in 30 minutes.

If this environment is not available, then use Python.
Active State Python (Freeware distribution, runs directly)
http://www.activestate.com/Products/ActivePython/

Python Language Website
http://www.python.org/

Python Reference Card
http://admin.oreillynet.com/python/excerpt/PythonPocketRef/examples/python.pdf
import time

for n in (1,10,100,1000,10000,100000,1000000):
    a = 0
    tstart = time.time()
    for i in xrange(n):
        a = a+1
    tend = time.time()

    print n, tend-tstart, (tend-tstart)/n

def example_filehandling():
    f = open("c:\temp\test.txt")
    for line in f.readlines():
        print line
    f.close()

tstart = time.time()
example_filehandling()
tend = time.time()
print "file open, read & print, close: ", tend-tstart,"s"
• Perform the following measurements
  1. loop overhead
  2. file open

• Determine for every measurement:
  What is the expected result?
  What is the measurement error?
  What is the result?
  What is the credibility of the result?
  Explain the result.
  (optional) What is the variation? Explain the variation.
Reflection on Exercise

+ measuring is easy
+ measuring provides data and understanding

~ result and expectation often don't match

- sensible measuring is more difficult
Abstract
This presentation addresses the fundamentals of measuring: What and how to measure, impact of context and experiment on measurement, measurement errors, validation of the result against expectations, and analysis of variation and credibility.

Distribution
This article or presentation is written as part of the Gaudí project. The Gaudí project philosophy is to improve by obtaining frequent feedback. Frequent feedback is pursued by an open creation process. This document is published as intermediate or nearly mature version to get feedback. Further distribution is allowed as long as the document remains complete and unchanged.
content

What and How to measure

Impact of experiment and context on measurement

Validation of results, a.o. by comparing with expectation

Consolidation of measurement data

Analysis of variation and analysis of credibility
### Measuring Approach: What and How

**what**

<table>
<thead>
<tr>
<th>1. What do we need to know?</th>
</tr>
</thead>
<tbody>
<tr>
<td>2. Define quantity to be measured.</td>
</tr>
<tr>
<td>3. Define required accuracy</td>
</tr>
<tr>
<td>4A. Define the measurement circumstances</td>
</tr>
<tr>
<td>4B. Determine expectation</td>
</tr>
<tr>
<td>4C. Define measurement set-up</td>
</tr>
<tr>
<td>5. Determine actual accuracy</td>
</tr>
<tr>
<td>6. Start measuring</td>
</tr>
<tr>
<td>7. Perform sanity check</td>
</tr>
</tbody>
</table>

**how**

- initial model
- purpose
- fe.g. by use cases
- historic data or estimation
- uncertainties, measurement error
- expectation versus actual outcome

**iterate**
1. What do We Need? Example Context Switching

**What:**
context switch time of VxWorks running on ARM9

guidance of concurrency design and task granularity

estimation of total lost CPU time due to context switching

test program

VxWorks
operating system

ARM 9
200 MHz CPU
100 MHz bus

(computing) hardware
2. Define Quantity by Initial Model

What (original):
context switch time of VxWorks running on ARM9

What (more explicit):
The amount of lost CPU time, due to context switching on VxWorks running on ARM9 on a heavy loaded CPU

\[ t_{\text{context switch}} = t_{\text{scheduler}} + t_{p1, \text{loss}} \]

Legend:
- Scheduler
- Process 1
- Process 2

\[ t_{p1, \text{no switching}} \]

\[ t_{p1, \text{before}} \quad t_{\text{scheduler}} \quad t_{p2, \text{loss}} \quad t_{p2} \quad t_{\text{scheduler}} \quad t_{p1, \text{loss}} \quad t_{p1, \text{after}} \]

\( p2 \text{ pre-empts } p1 \quad p1 \text{ resumes} \]

\( = \text{lost CPU time} \)
3. Define Required Accuracy

~10%

Guidance of concurrency design and task granularity

Estimation of total lost CPU time due to context switching

Number of context switches depends on application

Cost of context switch depends on OS and HW

Purpose drives required accuracy
Intermezzo: How to Measure CPU Time?

Low resolution (~ μs - ms)
Easy access
Lot of instrumentation

High resolution (~ 10 ns)
requires
HW instrumentation

Low resolution (~ μs - ms)
Easy access
Lot of instrumentation

High resolution (~ 10 ns)
Requires Timer Access

Cope with limitations:
- Duration (16 / 32 bit counter)
- Requires Timer Access
4A. Define the Measurement Set-up

*Mimick relevant real world characteristics*

**real world**
many concurrent processes, with
# instructions >> l-cache
# data >> D-cache

**experimental set-up**

- P1
- P2
- pre-empts
- cache flush
- no other CPU activities

$t_{p1, before}$ $t_{scheduler}$ $t_{p2, loss}$ $t_{p2}$ $t_{scheduler}$ $t_{p1, loss}$ $t_{p1, after}$

$p_2$ pre-empts $p_1$

$p_1$ resumes = lost CPU time
4B. Case: ARM9 Hardware Block Diagram

- CPU
- on-chip bus
- Instruction cache
- Data cache
- cache line size: 8 32-bit words
- memory bus
- memory
- 200 MHz
- 100 MHz

chip

PCB

©2006, Gerrit Muller
Key Hardware Performance Aspect

memory request

22 cycles

memory response

word 1
word 2
word 3
word 4
word 5
word 6
word 7
word 8

data

38 cycles

memory access time in case of a cache miss
200 Mhz, 5 ns cycle: 190 ns
OS Process Scheduling Concepts

- New
- Running
- Waiting
- Ready
- Terminated
- Scheduler
- dispatch
- create
- exit
- IO or event completion
- interrupt
- Wait
- (I/O / event)

Modeling and Analysis: Measuring
©2006, Embedded Systems Institute
version: 1.2
July 31, 2014
PSRTprocessConcepts
Determine Expectation

simple SW model of context switch:
- save state P1
- determine next runnable task
- update scheduler administration
- load state P2
- run P2

input data HW:
- $t_{\text{ARM instruction}} = 5 \text{ ns}$
- $t_{\text{memory access}} = 190 \text{ ns}$

Estimate how many
instructions and memory accesses
are needed per context switch

Calculate the estimated time
needed per context switch
Determine Expectation Quantified

- **Input Data HW:**
  - ARM instruction: \( t_{\text{ARM instruction}} = 5 \text{ ns} \)
  - Memory access: \( t_{\text{memory access}} = 190 \text{ ns} \)

- **Simple SW Model of Context Switch:**
  1. Save state P1
  2. Determine next runnable task
  3. Update scheduler administration
  4. Load state P2
  5. Run P2

- **Estimate:** How many instructions and memory accesses are needed per context switch

- **Calculate:** The estimated time needed per context switch

- **Round up (as margin) gives expected:**
  - Context switch time: \( t_{\text{context switch}} = 2 \mu\text{s} \)
4C. Code to Measure Context Switch

Task 1
- Time Stamp End
- Cache Flush
- Time Stamp Begin
- Context Switch

Task 2
- Time Stamp End
- Cache Flush
- Time Stamp Begin
- Context Switch
- Time Stamp End
- Cache Flush
- Time Stamp Begin
- Context Switch
- Time Stamp End
- Cache Flush
- Time Stamp Begin
- Context Switch
Measuring Task Switch Time
Understanding: Impact of Context Switch

![Diagram showing the impact of context switch on clock cycles per instruction (CPI)]

Based on figure diagram by Ton Kostelijk

Clock cycles Per Instruction (CPI)

Task 1

Task 2

Scheduler

Process 1

Process 2

Time
5. Accuracy: Measurement Error

Measurements have stochastic variations and systematic deviations resulting in a range rather than a single value.
Accuracy 2: Be Aware of Error Propagation

\[ t_{\text{duration}} = t_{\text{end}} - t_{\text{start}} \]

\[ t_{\text{start}} = 10 \pm 2 \mu s \]

\[ t_{\text{end}} = 14 \pm 2 \mu s \]

\[ t_{\text{duration}} = 4 \pm \? \mu s \]

systematic errors: add linear

stochastic errors: add quadratic
Measurements have

stochastic variations and systematic deviations

resulting in a range rather than a single value.

The inputs of modeling,
"facts", assumptions, and measurement results,
also have stochastic variations and systematic deviations.

Stochastic variations and systematic deviations

propagate (add, amplify or cancel) through the model

resulting in an output range.
### 6. Actual ARM Figures

**ARM9  200 MHz**

$t_{\text{context switch}}$ as function of cache use

<table>
<thead>
<tr>
<th>cache setting</th>
<th>$t_{\text{context switch}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>From cache</td>
<td>2 µs</td>
</tr>
<tr>
<td>After cache flush</td>
<td>10 µs</td>
</tr>
<tr>
<td>Cache disabled</td>
<td>50 µs</td>
</tr>
</tbody>
</table>
7. Expectation versus Measurement

expected: $t_{\text{context switch}} = 2 \mu s$

measured: $t_{\text{context switch}} = 10 \mu s$

**How to explain?**

**potentially missing in expectation:**
- memory accesses due to instructions
  - $\sim 10$ instruction memory accesses $\sim= 2 \mu s$
- memory management (MMU context)
- complex process model (parents, permissions)
- bookkeeping, e.g. performance data
- layering (function calls, stack handling)
- the combination of above issues

**However, measurement seems to make sense**

**input data HW:**
- ARM instruction: $t_{\text{ARM instruction}} = 5 \text{ ns}$
- memory access: $t_{\text{memory access}} = 190 \text{ ns}$
- context switch: $t_{\text{context switch}} = 1140 \text{ ns}$
## Conclusion Context Switch Overhead

The overhead time for context switches can be calculated using the formula:

\[ t_{\text{overhead}} = n_{\text{context switch}} \times t_{\text{context switch}} \]

where \( t_{\text{context switch}} \) is the time for a single context switch.

### Table

<table>
<thead>
<tr>
<th>( n_{\text{context switch}} ) (s(^{-1}))</th>
<th>( t_{\text{context switch}} ) = 10µs</th>
<th>( t_{\text{context switch}} ) = 2µs</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_{\text{overhead}} )</td>
<td>( t_{\text{overhead}} )</td>
<td>( t_{\text{overhead}} )</td>
</tr>
<tr>
<td>500</td>
<td>5ms</td>
<td>0.5%</td>
</tr>
<tr>
<td>5000</td>
<td>50ms</td>
<td>5%</td>
</tr>
<tr>
<td>50000</td>
<td>500ms</td>
<td>50%</td>
</tr>
</tbody>
</table>
**Summary Context Switching on ARM9**

<table>
<thead>
<tr>
<th>goal of measurement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Guidance of concurrency design and task granularity</td>
</tr>
<tr>
<td>Estimation of context switching overhead</td>
</tr>
<tr>
<td>Cost of context switch on given platform</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>examples of measurement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Needed: context switch overhead ~10% accurate</td>
</tr>
<tr>
<td>Measurement instrumentation: HW pin and small SW test program</td>
</tr>
<tr>
<td>Simple models of HW and SW layers</td>
</tr>
<tr>
<td>Measurement results for context switching on ARM9</td>
</tr>
</tbody>
</table>
Conclusions

Measurements are an important source of factual data.

A measurement requires a well-designed experiment.

Measurement error, validation of the result determine the credibility.

Lots of consolidated data must be reduced to essential understanding.

Techniques, Models, Heuristics of this module

experimentation

error analysis

estimating expectations
This work is derived from the EXARCH course at CTT developed by Ton Kostelijk (Philips) and Gerrit Muller.

The Boderc project contributed to the measurement approach. Especially the work of Peter van den Bosch (Océ), Oana Florescu (TU/e), and Marcel Verhoef (Chess) has been valuable.
Introductory discussion
Abstract
Performance models are mostly simple mathematical formulas. The challenge is to model the performance at an appropriate level. In this presentation we introduce several levels of modeling, labeled zeroth order, second order, et cetera. As illustration we use the performance of MRI reconstruction.
<table>
<thead>
<tr>
<th>Order</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0&lt;sup&gt;th&lt;/sup&gt;</td>
<td>Main function parameters</td>
</tr>
<tr>
<td>1&lt;sup&gt;st&lt;/sup&gt;</td>
<td>Add overhead secondary function(s)</td>
</tr>
<tr>
<td>2&lt;sup&gt;nd&lt;/sup&gt;</td>
<td>Interference effects circumstances</td>
</tr>
</tbody>
</table>
CPU Time Formula Zero Order

\[ t_{\text{cpu total}} = t_{\text{cpu processing}} + t_{\text{UI}} \]

\[ t_{\text{cpu processing}} = n_x \times n_y \times t_{\text{pixel}} \]
\[ t_{\text{cpu total}} = t_{\text{cpu processing}} + t_{\text{Ul}} + t_{\text{context switch}} + t_{\text{overhead}} \]
CPU Time Formula Second Order

\[ t_{\text{cpu total}} = t_{\text{cpu processing}} + t_{UI} + t_{\text{context switch overhead}} \]

\[ t_{\text{stall time due to}} + t_{\text{stall time due to}} \]

signal processing: high efficiency

control processing: low/medium efficiency
MRI reconstruction

"Test" of performance model on another case

Scope of performance and significance of impact
MR Reconstruction Context
MR Reconstruction Performance Zero Order

\[ t_{\text{recon}} = n_{\text{raw-x}} \times t_{\text{fft}}(n_{\text{raw-y}}) + n_y \times t_{\text{fft}}(n_{\text{raw-x}}) \]

\[ t_{\text{fft}}(n) = c_{\text{fft}} \times n \times \log(n) \]
Zero Order Quantitative Example

Typical FFT, 1k points ~ 5 msec
( scales with 2 * n * log (n) )

using:

\[ n_{raw-x} = 512 \]
\[ n_{raw-y} = 256 \]
\[ n_x = 256 \]
\[ n_y = 256 \]

\[ t_{recon} = n_{raw-x} \times t_{fft}(n_{raw-y}) + n_y \times t_{fft}(n_{raw-x}) + 512 * 1.2 + 256 * 2.4 \]

\[ \approx 1.2 \text{ s} \]
MR Reconstruction Performance First Order

\[ t_{\text{recon}} = t_{\text{filter}}(n_{\text{raw-x}}, n_{\text{raw-y}}) + n_{\text{raw-x}} \cdot t_{\text{fft}}(n_{\text{raw-y}}) + n_{y} \cdot t_{\text{fft}}(n_{\text{raw-x}}) + t_{\text{corrections}}(n_{x}, n_{y}) \]

\[ t_{\text{fft}}(n) = c_{\text{fft}} \cdot n \cdot \log(n) \]
Typical FFT, 1k points ~ 5 msec  
   ( scales with $2 \times n \times \log(n)$ )

Filter 1k points ~ 2 msec  
   ( scales linearly with n )

Correction ~ 2 msec  
   ( scales linearly with n )
\[ t_{\text{recon}} = t_{\text{filter}}(n_{\text{raw-x}}, n_{\text{raw-y}}) + n_{\text{raw-x}} \cdot (t_{\text{fft}}(n_{\text{raw-y}}) + n_{\text{raw-x}} \cdot (t_{\text{fft}}(n_{\text{raw-x}}) + n_y \cdot (t_{\text{fft}}(n_{\text{raw-x}}) + t_{\text{corrections}}(n_x, n_y) + t_{\text{control-overhead}}) + t_{\text{col-overhead}})) + t_{\text{row-overhead}}) + t_{\text{row-overhead}} + t_{\text{col-overhead}} + t_{\text{corrections}}(n_x, n_y) + t_{\text{control-overhead}}\]

\[ t_{\text{fft}}(n) = c_{\text{fft}} \cdot n \cdot \log(n) \]
Second Order Quantitative Example

Typical FFT, 1k points ~ 5 msec
( scales with $2 \cdot n \cdot \log(n)$ )

Filter 1k points ~ 2 msec
( scales linearly with $n$ )

Correction ~ 2 msec
( scales linearly with $n$ )

Control overhead = $n \cdot t_{row \, overhead}$

10 .. 100 µs
MR Reconstruction Performance Third Order

\[
T_{\text{recon}} = T_{\text{filter}}(n_{\text{raw-x}}, n_{\text{raw-y}}) + n_{\text{raw-x}} \cdot (T_{\text{fft}}(n_{\text{raw-y}}) + T_{\text{col-overhead}}) + n_y \cdot (T_{\text{fft}}(n_{\text{raw-x}}) + T_{\text{row-overhead}}) + T_{\text{corrections}}(n_x, n_y) + T_{\text{read I/O}} + T_{\text{transpose}} + T_{\text{write I/O}} + T_{\text{control-overhead}}
\]

Focus on overhead reduction is more important than faster algorithms. This is not an excuse for sloppy algorithms.
MRI reconstruction

System performance may be determined by other than standard facts
E.g. more by overhead I/O rather than optimized core processing

==> Identify & measure what is performance-critical in application
Abstract
Soft Real Time design addresses the performance aspects of the system design, under the assumption that the hard real time design is already well-covered. Core decisions in soft real time design are:

• granularity
• synchronization
• prioritization
• allocation
• resource management
Soft Real Time Design

- **hard real time**
  - disastrous failure
  - human safety
  - device safety
  - loss of information

- **soft real time**
  - dissatisfaction irritation
  - limited throughput
  - waiting time
  - loss of functionality or (image) quality
  - loss of eye hand coordination

- **soft real time**
Case 1

TV zapping

Problem introduction
Approach for solving response time problems
Revised functional model
Measuring and modelling
Zap timing: What is the Requirement?

- **P+**
- **P-**
- Remote control
- **new channel**
- **zap**
- **total response time**
- **zap repetition**
- **visual feedback**
- **open for next response**
- **new channel**
- **visual feedback time**
- **time**
## Approach

1) Measure the end-to-end time

2) Decompose the processes based on expected outcome

3) Measure the individual components
   use previous decomposition (2)

4) Clarify the unknown parts and make them explicit

5) Further divide the major posts

6) Aggregate the smaller posts
### Expected values:

<table>
<thead>
<tr>
<th>Action</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mute</td>
<td>50 ms</td>
</tr>
<tr>
<td>Blank</td>
<td>40 ms</td>
</tr>
<tr>
<td>Flush AV pipeline</td>
<td>160 ms</td>
</tr>
<tr>
<td>Set tuner</td>
<td>200 ms</td>
</tr>
<tr>
<td>Fill AV pipeline</td>
<td>160 ms</td>
</tr>
<tr>
<td>Unmute</td>
<td>50 ms</td>
</tr>
<tr>
<td>Unblank</td>
<td>40 ms</td>
</tr>
</tbody>
</table>

### Measured values:

<table>
<thead>
<tr>
<th>Action</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mute</td>
<td>60 ms</td>
</tr>
<tr>
<td>Blank</td>
<td>120 ms</td>
</tr>
<tr>
<td>Flush AV pipeline</td>
<td>0 ms</td>
</tr>
<tr>
<td>Set tuner</td>
<td>180 ms</td>
</tr>
<tr>
<td>Fill AV pipeline</td>
<td>40 ms</td>
</tr>
<tr>
<td>Format detection</td>
<td>200 ms</td>
</tr>
<tr>
<td>Unmute</td>
<td>60 ms</td>
</tr>
<tr>
<td>Unblank</td>
<td>120 ms</td>
</tr>
<tr>
<td>Summing</td>
<td>~ 900 ms</td>
</tr>
</tbody>
</table>

**Total time measured:** 2000 ms
Analysis and Improvements

Zapping Problem step 4

Somewhere 1000 ms are missing
Detection of frame size takes a long time!
+ Lots of software overhead
Analyze frame size detection and SW overhead

Zapping Problem step 5

Subdivide / analyze format detection (200 ms)

Zapping Problem step 6

Ignore pipeline effects
Simple Concurrency Model (with waits)

Zapping tasks sequential

- Set Tuner
- Detect Framesize
- Video present
- No video
- Video present
- Blank Video
- Flush AV pipeline
- Fill AV pipeline
- Mute Audio
- Blink LED
- Zap
- OSD
- Zap finished
Zapping tasks parallel

- Set Tuner
- Detect Framesize
- Video (No video to Video present)
- Blank Video
- Flush AV pipeline
- Fill AV pipeline
- Mute Audio
- Blink LED
- OSD
- Zap
- Zap finished
TV zapping

Understanding of the problem is crucial

Iterate over modelling and measuring to build balanced performance model
EasyVision: Resource Management

Introduction to application

SW design

Memory and performance

Memory design

CPU load and Performance
**Easyvision**

Medical Imaging Workstation

serving 3 X-ray examination rooms

providing interactive viewing and printing on high resolution film

Challenge: interoperability and WYSIWYG over different products
Easyvision Serving Three URF Examination Rooms

URF-systems

EasyVision: Medical Imaging Workstation

typical clinical image (intestines)
Image Quality Expectation WYSIWYG

what you see at one work-spot is what you get at another work-spot

X-ray system

image generation

presentation

Easyvision

application processing

presentation

3rd party workstation

monitor

film

network, storage

monitor

film

network, storage

Soft Real Time Design
©2006, Embedded Systems Institute
94 ©2006, Gerrit Muller
Presentation Pipeline for X-ray Images

- Image from database
- Spatial enhancement
  - bi-linear
  - bi-cubic
- Interpolate
  - Bi-linear
  - Bi-cubic
- Look up table
  - Invert
  - Contrast / brightness
- Graphics merge
- Colour LUT
- Monitor

Legend:
- SW
- HW
Quadruple View-port Screen Layout

UI icons, text

view-port 1
view-port 2
view-port 3
view-port 4
view-port 5

960 pixels
ca. 460 pixels
ca. 200 pixels
1152 pixels
Rendered Images at Different Destinations

**Screen:**
- low resolution
- fast response

**Film:**
- high resolution
- high throughput

**Network:**
- medium resolution
- high throughput
Easyvision SW design

Concurrency design

SW layers
Concurrency via Software Processes

- remote systems and users
- communication
- user interface
- user
- data base
- export
- network
- disk drive
- optical drive
- optical storage
- print
- printer
- UI devices

Legend:
- client
  - user control
  - control and data flow
  - operational process
  - system monitor
  - Unix daemons
  - server process
  - associated hardware

- client process

Soft Real Time Design
©2006, Embedded Systems Institute
version: 0.2
July 31, 2014
Criteria for Process Decomposition

- management of concurrency
- management of shared devices
- unit of memory budget (easy measurement)
- enables distribution over multiple processors
- unit of exception handling: fault containment and watchdog monitor

Processes are a facility provided by the Operating System (OS) to manage concurrency, resources and exceptions.
### Simplified Layering of the SW (Construction Decomposition)

#### Medical Imaging R/F

<table>
<thead>
<tr>
<th>Print</th>
<th>Store</th>
<th>View</th>
<th>Cluster</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spool</td>
<td>HCU</td>
<td>Image</td>
<td>Gfx</td>
</tr>
<tr>
<td></td>
<td>Store</td>
<td>UI</td>
<td>DB</td>
</tr>
<tr>
<td>RC driver</td>
<td>HC driver</td>
<td>DOR driver</td>
<td>NIX</td>
</tr>
<tr>
<td>SunOS</td>
<td></td>
<td></td>
<td>PMS-net in</td>
</tr>
</tbody>
</table>

#### Desk, cabinets, cables, etc.

- **Standard IPX workstation**
- **Desktop, cabinets, cables, etc.**

#### SW infrastructure

- **Devs. tools**
- **Service**
- **SW keys**
- **Config**
- **Install**
- **Start up**

#### Connected system

- **RC**
- **RC interf**
- **3M**
- **DSI**

#### Connected system

- **User interface**
- **Application functions**
- **Toolbox**
- **Operating system**
- **Hardware**

**Legend:**
- **SW infrastructure**
- **Connected system**

---

**Soft Real Time Design**
©2006, Embedded Systems Institute

**version: 0.2**
July 31, 2014
MICVswLayers1992

©2006, Gerrit Muller
Easyvision Memory and Performance

Performance problems

Analysis of memory use

Memory budget
Performance as a Function of Memory Use

![Graph showing performance as a function of memory usage. The x-axis represents memory usage in MB, ranging from 0 to 200 MB, and the y-axis represents performance. The graph is divided into two regions: Good and Bad. The Good region is from 0 to 64 MB, with a line indicating a decrease in performance. The Bad region is from 64 MB to 200 MB, with a line indicating a further decrease in performance. The graph also highlights the transition from physical memory to paging to disk.]

- **Good**: Memory usage from 0 to 64 MB
- **Bad**: Memory usage from 64 MB to 200 MB

- **Physical Memory**
- **Paging to Disk**

---

**Soft Real Time Design**

©2006, Embedded Systems Institute

©2006, Gerrit Muller

.version: 0.2
July 31, 2014
EASRTperformanceVsMemory
Problem: Unlimited Memory Consumption (1992)

The diagram illustrates the total measured memory usage, categorized into different components:

- **Code**
- **Data**
- **Bulk Data**
- **Fragmentation**

The memory usage is measured in MB, ranging from 0 to 200 MB. The chart shows the performance degradation as the memory usage increases, with a significant drop in performance around 64 MB. The physical memory is limited, and paging to disk is required beyond this point.

---

Soft Real Time Design
©2006, Embedded Systems Institute
104 ©2006, Gerrit Muller

version: 0.2
July 31, 2014
MSmemoryZeroMeasurement
Measurement Per Process

- **data**: measured (left column), budget per process (right column)
- **MByte**: 0, 10, 20, 30
- **code**: 0, 10, 20

- **UNIX**: 10 MByte
- **shared libraries**: 20
- **UI**: 30
- **communication**: measured
- **server**: budget per process
- **storage server**: measured
- **print server**: budget per process
- **other**: measured

Soft Real Time Design
©2006, Embedded Systems Institute
105 ©2006, Gerrit Muller

version: 0.2
July 31, 2014
MSmemoryBudget
Solution: Measure and Iterative Redesign

- measured code
- OS data
- bulk data
- fragmentation

- budget
- anti-fragmenting
  - budget based
  - awareness, measurement
    - tuning
    - DLLs

200 MB
74 MB

Soft Real Time Design
©2006, Gerrit Muller
Method: Budget per Process

Budget:

+ measurable

+ fine enough to provide direction

+ coarse enough to be maintainable
Example of a Memory Budget

<table>
<thead>
<tr>
<th>memory budget in Mbytes</th>
<th>code</th>
<th>obj data</th>
<th>bulk data</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>shared code</td>
<td>11.0</td>
<td></td>
<td></td>
<td>11.0</td>
</tr>
<tr>
<td>User Interface process</td>
<td>0.3</td>
<td>3.0</td>
<td>12.0</td>
<td>15.3</td>
</tr>
<tr>
<td>database server</td>
<td>0.3</td>
<td>3.2</td>
<td>3.0</td>
<td>6.5</td>
</tr>
<tr>
<td>print server</td>
<td>0.3</td>
<td>1.2</td>
<td>9.0</td>
<td>10.5</td>
</tr>
<tr>
<td>optical storage server</td>
<td>0.3</td>
<td>2.0</td>
<td>1.0</td>
<td>3.3</td>
</tr>
<tr>
<td>communication server</td>
<td>0.3</td>
<td>2.0</td>
<td>4.0</td>
<td>6.3</td>
</tr>
<tr>
<td>UNIX commands</td>
<td>0.3</td>
<td>0.2</td>
<td>0.0</td>
<td>0.5</td>
</tr>
<tr>
<td>compute server</td>
<td>0.3</td>
<td>0.5</td>
<td>6.0</td>
<td>6.8</td>
</tr>
<tr>
<td>system monitor</td>
<td>0.3</td>
<td>0.5</td>
<td>0.0</td>
<td>0.8</td>
</tr>
<tr>
<td>application SW total</td>
<td>13.4</td>
<td>12.6</td>
<td>35.0</td>
<td>61.0</td>
</tr>
</tbody>
</table>

| UNIX Solaris 2.x                         |      |          |           | 10.0  |
| file cache                               |      |          |           | 3.0   |
| total                                    |      |          |           | 74.0  |
Exercise: Bulk Data Capacity

Memory block

12MByte

How many blocks of 1024 x 1024 8-bits data can be stored?

How many blocks of 1024 x 1024 16-bits data can be stored?
Exercise: Object Data Capacity

<table>
<thead>
<tr>
<th>Frequency</th>
<th>Description</th>
<th>Typical size</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Large objects (e.g. dictionary)</td>
<td>20 kB</td>
</tr>
<tr>
<td>20</td>
<td>Medium object, e.g. UI data</td>
<td>200 Bytes</td>
</tr>
<tr>
<td>1000</td>
<td>Small object, e.g. image attributes</td>
<td>20 Bytes</td>
</tr>
</tbody>
</table>

Total

How many objects with this distribution fit in the 3MByte Object data store?
# Memory Budget of Easyvision RF R1 and R2

<table>
<thead>
<tr>
<th>memory budget in Mbytes</th>
<th>code</th>
<th>object data</th>
<th>bulk data</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>R1</td>
<td>R2</td>
<td>R1</td>
<td>R2</td>
</tr>
<tr>
<td>shared code</td>
<td>6.0</td>
<td>11.0</td>
<td>6.0</td>
<td>11.0</td>
</tr>
<tr>
<td>UI process</td>
<td>0.2</td>
<td>0.3</td>
<td>3.0</td>
<td>12.0</td>
</tr>
<tr>
<td>database server</td>
<td>0.2</td>
<td>0.3</td>
<td>3.2</td>
<td>3.0</td>
</tr>
<tr>
<td>print server</td>
<td>0.4</td>
<td>0.3</td>
<td>1.2</td>
<td>7.0</td>
</tr>
<tr>
<td>DOR server</td>
<td>0.4</td>
<td>0.3</td>
<td>2.0</td>
<td>2.0</td>
</tr>
<tr>
<td>communication server</td>
<td>1.2</td>
<td>0.3</td>
<td>2.0</td>
<td>10.0</td>
</tr>
<tr>
<td>UNIX commands</td>
<td>0.2</td>
<td>0.3</td>
<td>0.5</td>
<td>0.2</td>
</tr>
<tr>
<td>compute server</td>
<td>0.3</td>
<td>0.5</td>
<td>6.0</td>
<td>6.8</td>
</tr>
<tr>
<td>system monitor</td>
<td>0.3</td>
<td>0.5</td>
<td></td>
<td>0.8</td>
</tr>
<tr>
<td>application total</td>
<td>8.6</td>
<td>13.4</td>
<td>12.6</td>
<td>35.0</td>
</tr>
<tr>
<td>UNIX</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>file cache</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>total</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Memory block

12MByte

How many blocks of 1024 x 1024 8-bits data can be stored?

12

How many blocks of 1024 x 1024 16-bits data can be stored?

6

* Assuming that 8-bit data is stored as 8-bit (char)
Assuming that 16-bit data is stored as 16-bit (short int)
# Object Data Capacity

<table>
<thead>
<tr>
<th>Frequency</th>
<th>Description</th>
<th>Typical size</th>
<th>Size * Freq</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Large objects (e.g. dictionary)</td>
<td>20 kB</td>
<td>20 kB</td>
</tr>
<tr>
<td>20</td>
<td>Medium object, e.g. UI data</td>
<td>200 Bytes</td>
<td>4kB</td>
</tr>
<tr>
<td>1000</td>
<td>Small object, e.g. image attributes</td>
<td>20 Bytes</td>
<td>20kB</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td><strong>44kB</strong></td>
<td></td>
</tr>
</tbody>
</table>

44kB fits approximately 68 times in 3MByte
Expect to store at most 68 large objects
(1360 Medium sized objects, 68000 small objects)
Easyvision Memory Design

Fragmentation and consequences

Application caches

Memory design applied
Memory Fragmentation

1. replace image 3 by image 4

2. add image 5

3. replace image 1 by image 6
Memory Fragmentation Increase

- used address space
- gross used
- nett used

MBytes vs. time

version: 0.2
July 31, 2014
<table>
<thead>
<tr>
<th>Medical imaging R/F cache sizes</th>
</tr>
</thead>
<tbody>
<tr>
<td>cluster PixMap cache</td>
</tr>
<tr>
<td>print PixMap cache</td>
</tr>
<tr>
<td>view PixMap cache</td>
</tr>
<tr>
<td>allocator, chunk</td>
</tr>
<tr>
<td>heap memory, malloc() free()</td>
</tr>
<tr>
<td>virtual memory</td>
</tr>
<tr>
<td>memory management unit</td>
</tr>
<tr>
<td>instruction and data cache</td>
</tr>
<tr>
<td>physical memory</td>
</tr>
<tr>
<td>disk storage</td>
</tr>
</tbody>
</table>

**Legend**

- **User Interface**
- **Application Functions**
- **Toolbox**
- **Operating System**
- **Hardware**
Bulk Data Memory Management Memory Allocators

chunk size: 3MB
for large images
from 225 kB (480*480*8)
to 3 MB (1536*1024*16)
block size: 256kB

chunk size: 1MB
for stamp images
96*96*8 (9kB)
block size: 9kB

chunk size: 2MB
for small (screen) images
from 8kB
to 225 kB
block size: 8 kB

chunk size: 1MB
for small (screen) images
from 8kB
to 225 kB
block size: 8 kB
Cached Intermediate Processing Results

---

raw image

enhanced image

resized image

lookup

merge

viewport

display

retrieve

enhance

interpolate

---

text
gfx

version: 0.2
July 31, 2014
MICVprocessingCachedPixmaps
Example of Allocator and Cache Use

- 1024² 8 bit image requires 4 256kB blocks
- 8 1024² images require 48 256kB blocks
  12 blocks shortage

- block size: 256kB
- block size: 9kB
- block size: 8 kB

- 460² 8 bit requires 27 8kB blocks
- 200² images require 5 8kb blocks
- all screen-size images require 334 8kB blocks, 78 blocks shortage

- raw image
- enhanced image
- resized image
- grey-value image
- viewports

- 4 * 1024²
  1 byte / pixel
- 4 * 1024²
  2 byte / pixel
- 4 * 460²
  2 byte / pixel
- 4 * 460²
  1 byte / pixel
- 4 * 460²
  1 byte / pixel

- 96²
- 96²
- 200²
- 200²
- 200²

Soft Real Time Design
©2006, Embedded Systems Institute
120 ©2006, Gerrit Muller
Print Server is Based on Banding

Original images

1024 pixels

1024 pixels

128 pixels

4k pixels
Easyvision Memory CPU load and performance

CPU load analysis
response time
throughput
measurement tools
CPU Processing Times and Viewing Responsiveness

![Diagram showing CPU processing times and viewing responsiveness.]

- **retrieve** 0.3s
- **enhance** 0.5s
- **interpolate** 0.2s
- **resized image** 0.075s
- **lookup (LUT)**
- **merge**
- **view-port**
- **display** 0.05s

- **update rate for common user actions**
  - next 0.9s\(^{-1}\)
  - zoom 3 s\(^{-1}\)
  - C/B 7 s\(^{-1}\)

**Pipeline timing proportional**

- 1024\(^2\)
- 1024\(^2\)
- 920\(^2\)
- 920\(^2\)
- 920\(^2\)

**Accumulated processing time in seconds**

- 1.1
- 1.0
- 0.9
- 0.8
- 0.7
- 0.6
- 0.5
- 0.4
- 0.3
- 0.2
- 0.1
- 0
Server CPU Load

remote systems and users
communication
data base
print
printer

serving one examination room

CPU time available for interactive viewing

serving 3 examination rooms

import

print

2.5 CPU second per Mbyte input
2.5 min / exam

3.5 CPU second per Mpixel output
50 s/exam

30%

90%

30%

2 min

210 s/exam

50 s/exam
Resource Measurement Tools

- $t_{n-2}$: preamble to remove start-up effects
- $t_{n-1}$: use case
- $t_n$: time

<table>
<thead>
<tr>
<th>oit</th>
<th>$\triangle$ object instantations heap memory usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>ps</td>
<td>kernel CPU time</td>
</tr>
<tr>
<td>vmstat</td>
<td>user CPU time</td>
</tr>
<tr>
<td>kernel resource stats</td>
<td>code memory</td>
</tr>
<tr>
<td></td>
<td>virtual memory</td>
</tr>
<tr>
<td></td>
<td>paging</td>
</tr>
</tbody>
</table>

heapviewer (visualise fragmentation)
<table>
<thead>
<tr>
<th>class name</th>
<th>current nr of objects</th>
<th>deleted since $t_{n-1}$</th>
<th>created since $t_{n-1}$</th>
<th>heap memory usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>AsynchronousIO</td>
<td>0</td>
<td>-3</td>
<td>+3</td>
<td></td>
</tr>
<tr>
<td>AttributeEntry</td>
<td>237</td>
<td>-1</td>
<td>+5</td>
<td></td>
</tr>
<tr>
<td>BitMap</td>
<td>21</td>
<td>-4</td>
<td>+8</td>
<td></td>
</tr>
<tr>
<td>BoundedFloatingPoint</td>
<td>1034</td>
<td>-3</td>
<td>+22</td>
<td>[819200]</td>
</tr>
<tr>
<td>BoundedInteger</td>
<td>684</td>
<td>-1</td>
<td>+9</td>
<td>[8388608]</td>
</tr>
<tr>
<td>BtreeNode1</td>
<td>200</td>
<td>-3</td>
<td>+3</td>
<td></td>
</tr>
<tr>
<td>BulkData</td>
<td>25</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>ButtonGadget</td>
<td>34</td>
<td>0</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>ButtonStack</td>
<td>12</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>ByteArray</td>
<td>156</td>
<td>-4</td>
<td>+12</td>
<td>[13252]</td>
</tr>
</tbody>
</table>
### Overview of Benchmarks and Other Measurement Tools

<table>
<thead>
<tr>
<th>test / benchmark</th>
<th>what, why</th>
<th>accuracy</th>
<th>when</th>
</tr>
</thead>
<tbody>
<tr>
<td>SpecInt (by suppliers)</td>
<td>CPU integer</td>
<td>coarse</td>
<td>new hardware</td>
</tr>
<tr>
<td>Byte benchmark</td>
<td>computer platform performance</td>
<td>coarse</td>
<td>new hardware</td>
</tr>
<tr>
<td></td>
<td>OS, shell, file I/O</td>
<td></td>
<td>new OS release</td>
</tr>
<tr>
<td>file I/O</td>
<td>file I/O throughput</td>
<td>medium</td>
<td>new hardware</td>
</tr>
<tr>
<td>image processing</td>
<td>CPU, cache, memory as function of image, pixel size</td>
<td>accurate</td>
<td>new hardware</td>
</tr>
<tr>
<td>Objective-C overhead</td>
<td>method call overhead</td>
<td>accurate</td>
<td>initial</td>
</tr>
<tr>
<td></td>
<td>memory overhead</td>
<td></td>
<td></td>
</tr>
<tr>
<td>socket, network</td>
<td>throughput</td>
<td>accurate</td>
<td>ad hoc</td>
</tr>
<tr>
<td></td>
<td>CPU overhead</td>
<td></td>
<td></td>
</tr>
<tr>
<td>data base</td>
<td>transaction overhead</td>
<td>accurate</td>
<td>ad hoc</td>
</tr>
<tr>
<td></td>
<td>query behaviour</td>
<td></td>
<td></td>
</tr>
<tr>
<td>load test</td>
<td>throughput, CPU, memory</td>
<td>accurate</td>
<td>regression</td>
</tr>
</tbody>
</table>

- **Public**
- **Self made**
MRI Volume Reconstruction and Viewing

Usage patterns as impact on performance

Resource model and requirements identification for usage patterns
Data in bytes =

\[ 2 \times 512 \times 512 \times 256 \times 2 = \]

Volumes
\( x \times y \times z \)

bytes per pixel

256 MBytes

in 2 * 2 minutes = 240 seconds
Performance Requirements

- George arrives at radiology department
- Nurse explains the procedure
- George is waiting in the dressing room
- Prepare George for the examination (a.o. RF coils)
- Position
- Imaging
- View away
- View away
- George leaves exam room

15 minute time slot
Examination of previous patient

14:00
14:15
14:30
Resource Model

Acquisition → Reconstruction → Viewing

Intermediate data:
256 MByte

Storage
2 Volumes
256 MByte

View away in ca 10 sec.
full screen
25 images per second
Critical Resources

Buffer architecture

Acquisition → Reconstruction

Intermediate data: 256 MByte

Viewing

Pipeline & caching

View away in ca 10 sec.

full screen 25 images per second

Attribute access

Storage

2 Volumes 256 MByte

Soft Real Time Design
©2006, Embedded Systems Institute
132 ©2006, Gerrit Muller

version: 0.2
July 31, 2014
MRneuroResourceCriticalities
MRI Volume Reconstruction and Viewing

Operational usage pattern drives (implicit/explicit) system performance requirements

Resource / cost trade-off must support operational usage patterns
Mobile Display Appliances

Modelling external environment

End-to-end performance

Allocation choices
Mobile Display Appliances

Mobile Display Appliance

Mediascreen

Original pictures from Nokia
User Access Point to a Long Foodchain

User

Appliance

Home Server

Network Providers

Service Providers

Content Providers
The "SMART" World of the Design

Standard Interactive System

Data transport  Security  Virtual Machine  Display and UI

Applications

free after Nick Thorne, Philips Semiconductors, Systems Laboratory Southampton UK, as presented at PSAVAT April 2001
Specifiable Characteristics

Standard Interactive System

Data transport

Security

Virtual Machine

Applications

Functionality

Performance

Power, Footprint

Display size

Color depth

Rendering

Performance

IQ

UI modi

Throughput

Latency

Distance

Power

Security Level

Performance

Encryption

Authentication

Servers & Networks

Display and UI

Functionality

Performance

Power, Footprint

Version: 0.2
July 31, 2014
## Response Time: Latency Budget

<table>
<thead>
<tr>
<th></th>
<th>Message Latency</th>
<th>Response Time</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Appliance</strong></td>
<td>40</td>
<td>100</td>
</tr>
<tr>
<td>Data transport</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>Security</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>Virtual Machine</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>Application</td>
<td>10</td>
<td>30</td>
</tr>
<tr>
<td>Graphics and UI</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td><strong>Home Network</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Home Server</td>
<td>10</td>
<td>30</td>
</tr>
<tr>
<td>Network contention</td>
<td></td>
<td>20</td>
</tr>
<tr>
<td><strong>Provider Infrastructure</strong></td>
<td>50</td>
<td>160</td>
</tr>
<tr>
<td>Last-Mile network</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>Backbone network</td>
<td>20</td>
<td>40</td>
</tr>
<tr>
<td>Service server</td>
<td>10</td>
<td>50</td>
</tr>
<tr>
<td>Content server</td>
<td>10</td>
<td>50</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>110</td>
<td>310</td>
</tr>
<tr>
<td><strong>User need</strong></td>
<td></td>
<td>200</td>
</tr>
</tbody>
</table>

*All numbers are imaginary and for illustration purposes only*
Interaction or Irritation?

Response Time (ms)

310
150
100

Interactive Experience

Irritating Experience

User

Appliance

Home Server
Network Providers
Service Providers
Content Providers
Mobile Display Appliances

Modelling external environment: make assumptions

End-to-end performance:

  large part of performance budget is not controlled

User perceived performance determines function allocation
Exercise

Explore “Fast Browser” product specification, design options and performance issues
Abstract
The choice of scheduling technique and it’s parametrization impacts the performance of systems. This is an area where quite some theoretical work has been done. In this presentation we address Earliest Deadline First and Rate Monotonic Scheduling (RMS). We provide how-to information for RMS, based on Rate Monotonic Analysis (RMA).
Theory Hard Real Time Scheduling

Earliest Deadline First (EDF)

Rate Monotonic Scheduling (RMS)
Real Time Scheduling

Scheduler

Priorities
Ready
Queue
Wait
Queue
Scheduler admin

Run

Process / tasks instances

Proc. 1
Prio. High
State ready

Proc. 2
Prio. Med.
State ready

Proc. 3
Prio. High
State ready

...
Earliest Deadline First

<table>
<thead>
<tr>
<th>Constraints</th>
<th>in Absolute time (CPU cycles or msec, etc.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Determine deadlines</td>
<td>Process that has the earliest deadline gets the highest priority (no need to look at other processes)</td>
</tr>
<tr>
<td>Assign priorities</td>
<td>Smart mechanism needed for Real-Time determination of deadlines Pre-emptive scheduling needed</td>
</tr>
</tbody>
</table>

EDF = Earliest Deadline First

Earliest Deadline based scheduling for (a-)periodic Processing

The theoretical limit for any number of processes is 100% and so the system is schedulable.
Exercise Earliest Deadline First (EDF)

Calculate loads and determine thread activity (EDF)

<table>
<thead>
<tr>
<th>Thread</th>
<th>Period = deadline</th>
<th>Processing</th>
<th>Load</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thread 1</td>
<td>9</td>
<td>3</td>
<td>33.3%</td>
</tr>
<tr>
<td>Thread 2</td>
<td>15</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>Thread 3</td>
<td>23</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

Suppose at t=0, all threads are ready to process the arrived trigger.

Source: Ton Kostelijk - EXARCH course
Rate Monotonic Scheduling

- Determine deadlines (period) in terms of Frequency or Period (1/F)
- Assign priorities
  - Highest frequency (shortest period) => Highest priority
- Constraints
  - Independent activities
  - Periodic
  - Constant CPU cycle consumption
  - Assumes Pre-emptive scheduling

RMS = Rate Monotonic Scheduling

Priority based scheduling for Periodic Processing of tasks with a guaranteed CPU - load
Calculate loads and determine thread activity (RMS)

<table>
<thead>
<tr>
<th>Thread</th>
<th>Period = deadline</th>
<th>Processing</th>
<th>Load</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thread 1</td>
<td>9</td>
<td>3</td>
<td>33.3%</td>
</tr>
<tr>
<td>Thread 2</td>
<td>15</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>Thread 3</td>
<td>23</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

Suppose at t=0, all threads are ready to process the arrived trigger.

<table>
<thead>
<tr>
<th>0</th>
<th>9</th>
<th>15</th>
<th>18</th>
<th>23</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thread 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Thread 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Thread 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Source: Ton Kostelijk - EXARCH course
Real-time scheduling theory, utilization bound

- Set of tasks with periods $T_i$, and process time $P_i$: load $u_i = P_i / T_i$

- Schedule is at least possible when tasks are independent and:

$$Load = \sum_i U_i \leq n \left( \frac{1}{2^n} - 1 \right)$$

- $1.00, 0.83, 0.78, 0.76, \ldots \log(2) = 0.69$

Source: Ton Kostelijk - EXARCH course
• RMS cannot utilize 100% (1.0) of CPU, but for 1, 2, 3, 4, ... processes:
  1.00, 0.83, 0.78, 0.76, ...
  \( \log(2) = 0.69 \)

• RMS guarantees that all processes will always meet their deadlines, for any interleaving of processes.

• With fixed priorities, context switch overhead is limited

Source: Ton Kostelijk - EXARCH course
• For specific cases the utilization bound can be higher:
  up to 0.88 load for large n

• A processor running only hard-real-time processes is rare.
  For soft-RT less of a problem

• A lot of additional theory exists.
  Meeting deadlines in hard-real-time systems
  (L.P. Briand & D.M. Roy)

Source: Ton Kostelijk - EXARCH course
## Answers: loads and thread activity (EDF)

<table>
<thead>
<tr>
<th>Thread</th>
<th>Period = deadline</th>
<th>Processing</th>
<th>Load</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thread 1</td>
<td>9</td>
<td>3</td>
<td>33.3%</td>
</tr>
<tr>
<td>Thread 2</td>
<td>15</td>
<td>5</td>
<td>33.3%</td>
</tr>
<tr>
<td>Thread 3</td>
<td>23</td>
<td>5</td>
<td>21.7%</td>
</tr>
</tbody>
</table>

\[
\text{Total Load} = 88.3\%
\]

Source: Ton Kostelijk - EXARCH course
Answers: loads and thread activity (RMS)

<table>
<thead>
<tr>
<th>Thread</th>
<th>Period = deadline</th>
<th>Processing</th>
<th>Load</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thread 1</td>
<td>9</td>
<td>3</td>
<td>33.3%</td>
</tr>
<tr>
<td>Thread 2</td>
<td>15</td>
<td>5</td>
<td>33.3%</td>
</tr>
<tr>
<td>Thread 3</td>
<td>23</td>
<td>5</td>
<td>21.7%</td>
</tr>
</tbody>
</table>

88.3%

Source: Ton Kostelijk - EXARCH course
Extensions of the Application of RMS

\[
\begin{align*}
\text{if } \text{deadline} &\neq 1/\text{period} \\
\text{then } &\text{use period } = 1/\text{deadline}
\end{align*}
\]

\[
\begin{align*}
\text{if } \text{CPU consumption varies} \\
\text{then } &\text{use worst case CPU consumption}
\end{align*}
\]

More advanced techniques are available, for instance in case of "nice" frequencies.
Theory Hard Real Time Scheduling

Earliest Deadline First (EDF):

- optimal according theory, but practical not applicable due to overhead

Rate Monotonic Scheduling (RMS):

- provides recipe to assign priorities to tasks
- results in predictable real time behavior
- works well, even outside theoretical constraints
Exercise

Measurement of file transfers with different HTTP, FTP, Windows filesystem, on fast and slow networks
Navigation Case to be inserted here
Assignment for next block
to be inserted here
Exploring an existing code base: measurements and instrumentation

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

Abstract
Many architects struggle with a given large code-base, where a lot of knowledge about the code is in the head of people or worse where the knowledge has disappeared. One of the means to recover insight from a code base is by measuring and instrumenting the code-base. This presentation addresses measurements of the static aspects of the code, as well as instrumentation to obtain insight in the dynamic aspects of the code.

Distribution
This article or presentation is written as part of the Gaudí project. The Gaudí project philosophy is to improve by obtaining frequent feedback. Frequent feedback is pursued by an open creation process. This document is published as intermediate or nearly mature version to get feedback. Further distribution is allowed as long as the document remains complete and unchanged.

July 31, 2014
status: draft
version: 0.4
wanted:  
new functions and interfaces, higher performance levels, improvements, et cetera

given:  
document repository  
> 100 klines  
> 1k docs

code repository  
> 1Mloc  
> 1k files

complex system

created by  
>100 people

>100 people left

version: 0.4  
July 31, 2014  
EBMIProblem
## Overview of Approach and Presentation Agenda

<table>
<thead>
<tr>
<th>1 collect overviews</th>
<th>software system</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 study static structure</td>
<td></td>
</tr>
<tr>
<td>2A macroscopic fact finding</td>
<td></td>
</tr>
<tr>
<td>2B microscopic sampling</td>
<td></td>
</tr>
<tr>
<td>2C construct medium level diagrams</td>
<td></td>
</tr>
<tr>
<td>3 study dynamic behavior</td>
<td></td>
</tr>
<tr>
<td>3A measurements</td>
<td></td>
</tr>
<tr>
<td>3B construct simple models</td>
<td></td>
</tr>
</tbody>
</table>

- Iterate

---

Exploring an existing code base: measurements and instrumentation

©2006, Embedded Systems Institute

version: 0.4
July 31, 2014
EBMImethod
**SW Overview(s)**

- **Registry**
- **NameSpace server**
- **server**
- **Monitor**
- **Broker**
- **Event manager**
- **Transparant Communication**
- **Persistent Storage**
- **Abstraction Layer**
- **Device independent format**
- **Plug-in framework**
- **Resource scheduler**
- **Configurable pipeline**
- **Spool server**
- **Queue manager**
- **Session manager**
- **Property editor**
- **Monitor**
- **Compliance profile**
- **Device independent format**
- **Plug & play**
- **Application**

**delivery centric**

- **applications**
- **middleware services**
- **hardware abstraction layer**

**mechanism centric**

**(over)simplistic**

Exploring an existing code base: measurements and instrumentation

©2006, Embedded Systems Institute

version: 0.4
July 31, 2014
EBMlinputs
System Overviews

**subsystems**

- laser
- light source
- illuminator
- beam shaping
- reticle stage positioning
- reticle handler input/output
- lens
- projection
- C&T contamination, temperature
- wafer stage positioning
- wafer handler input/output

**control hierarchy**

- system control coordination
- ethernet
- laser
- illuminator
- lens
- measurement
- C&T
- reticle stage
- reticle handler
- wafer stage
- wafer handler

**physics/optics**

- dynamic exposure through slit
- reticle
- slit
- wafer
- 250 mm/s

**kinematic**

- expose
- step
- expose
- t

Exploring an existing code base: measurements and instrumentation
©2006, Embedded Systems Institute
166  ©2006, Gerrit Muller

version: 0.4
July 31, 2014
EBMIsystemDiagrams
Case 1: EasyVision (1992)

EasyVision: Medical Imaging Workstation

URF-systems

typical clinical image (intestines)
Examples of Macroscopic Fact Finding

```
> wc -l *.m
72 Acquisition.m
13 AcquisitionFacility.m
330 ActiveDataCollection.m
132 ActiveDataObject.m
304 Activity.m
281 ActivityList.m
551 AnnotateParser.m
1106 AnnotateTool.m
624 AnyOfList.m
466 AsyncBulkDataIO.m
264 AsyncDeviceIO.m
261 AsyncLocalDbIO.m
334 AsyncRemoteDbIO.m
205 AsyncSocketIO.m
```

version control information:
#new files
#deleted files
#changes per file since ...

package information:
#files

metrics:
QAC type information
#methods
#globals
Histogram of File Sizes EV R1.0

largest file: 4473 lines
DatabaseTool.m

legend
- size OK, sample few
- slightly suspect, sample some
- suspect, have a look

Exploring an existing code base: measurements and instrumentation
©2006, Embedded Systems Institute
169 ©2006, Gerrit Muller

version: 0.4
July 31, 2014
EBMihistogram
Example of small classes due to database design;
These classes are only supporting constructs

13 IndexBtree.m
12 IndexInteriorNode.m
13 IndexLeafNode.m
13 ObjectStoreBtree.m
12 ObjectStoreInteriorNode.m
13 ObjectStoreLeafNode.m

Example of large classes due to inherent complexity;
some of these classes are really suspect

1541 GenericRegion.m
1415 GfxArea.m
1697 GfxFreeContour.m
4095 GfxObject.m
1714 GfxText.m
1374 CVOBJECT.m
1080 ChartStack.m
1127 Collection.m
1651 Composite.m
1725 CompositeProjectionImage.m
1373 Connection1.m
1181 Database1.m
3707 DatabaseClient.m
3240 Image.m
1861 ImageSet.m

Example of large classes due to large amount of UI details

4473 DatabaseTool.m
1291 EnhancementTool.m
1106 AnnotateTool.m
1291 EnhancementTool.m
3471 GreyLevelTool.m
1639 HCConfigurationTool.m
1007 HCQueueViewingTool.m
1590 HardcopyTool.m
Changes Over Time

- redesign by mature designer
- partial redesigns
- failed in retrospect

ever changing files e.g.:
- systemConstants.h
- ShakyImplementation.m

hot spots
The real layering diagram did have >15 layers
Quantification helps to *calibrate* the *intuition* of the architect

Macroscopic numbers related to *code level* understanding provides insight

+ relative complexity
+ relative effort
+ hot spots
+ (static) dependencies and relations
Exploring an existing code base: measurements and instrumentation

Dynamics ≫ Static

- user interface
- running system
  - data
  - resources (CPU, cache, memory, bus BW, network, ...)
- behavior functionality
  - emerging properties
  - performance reliability

- system context
- code context
- design context
- images
  - patient info
  - configuration

version: 0.4
July 31, 2014
EBMIdynamics
Layered Benchmarking

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

operating system

OS services

duration
interrupts
task switches

services

(locality
density
efficiency
overhead)

tools

(latency
bandwidth
efficiency)

(duration
footprint
)

(EBMibenchmarkStack version: 0.4)

version: 0.4
July 31, 2014
EBMibenchmarkStack
Example: Processing HW and Service Performance

- Example: Processing HW and Service Performance
- spatial enhancement
- interpolate
- Look up table
- invert
- contrast / brightness
- graphics
- merge
- colour
- LUT
- HW
- SW
- monitor
- image from database
- bi-linear
- bi-cubic
- brightness
- contrast
- output
- input
- legend
- SW
- HW

Exploring an existing code base: measurements and instrumentation
©2006, Embedded Systems Institute
176
©2006, Gerrit Muller

version: 0.4
July 31, 2014
MICVpresentationPipeline
Exploring an existing code base: measurements and instrumentation

©2006, Embedded Systems Institute
©2006, Gerrit Muller

version: 0.4
July 31, 2014
MICVprocessingTimes
Resource Measurement Tools

Exploring an existing code base: measurements and instrumentation
©2006, Embedded Systems Institute
178 ©2006, Gerrit Muller

$t_{n-2}$, preamble to remove start-up effects, $t_{n-1}$, use case, $t_n$, time

<table>
<thead>
<tr>
<th>oit</th>
<th>$\Delta$ object instantations heap memory usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>ps</td>
<td>kernel CPU time</td>
</tr>
<tr>
<td>vmstat</td>
<td>user CPU time</td>
</tr>
<tr>
<td>kernel resource stats</td>
<td>code memory</td>
</tr>
<tr>
<td></td>
<td>virtual memory</td>
</tr>
<tr>
<td></td>
<td>paging</td>
</tr>
</tbody>
</table>

heapviewer (visualise fragmentation)
## Object Instantiation Tracing

<table>
<thead>
<tr>
<th>class name</th>
<th>current nr of objects</th>
<th>deleted since $t_{n-1}$</th>
<th>created since $t_{n-1}$</th>
<th>heap memory usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>AsynchronousIO</td>
<td>0</td>
<td>-3</td>
<td>+3</td>
<td>[819200]</td>
</tr>
<tr>
<td>AttributeEntry</td>
<td>237</td>
<td>-1</td>
<td>+5</td>
<td>[8388608]</td>
</tr>
<tr>
<td>BitMap</td>
<td>21</td>
<td>-4</td>
<td>+8</td>
<td></td>
</tr>
<tr>
<td>BoundedFloatingPoint</td>
<td>1034</td>
<td>-3</td>
<td>+22</td>
<td></td>
</tr>
<tr>
<td>BoundedInteger</td>
<td>684</td>
<td>-1</td>
<td>+9</td>
<td></td>
</tr>
<tr>
<td>BtreeNode1</td>
<td>200</td>
<td>-3</td>
<td>+3</td>
<td></td>
</tr>
<tr>
<td>BulkData</td>
<td>25</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>ButtonGadget</td>
<td>34</td>
<td>0</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>ButtonStack</td>
<td>12</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>ByteArray</td>
<td>156</td>
<td>-4</td>
<td>+12</td>
<td>[13252]</td>
</tr>
</tbody>
</table>
Memory Instrumentation

200 MB

bulk data

data

code

unaccounted
big lump

unaccounted
leftover

accountable
by OS services
and OIT

manually
instrumented

budget
## Overview of Benchmarks and Other Measurement Tools

<table>
<thead>
<tr>
<th>test / benchmark</th>
<th>what, why</th>
<th>accuracy</th>
<th>when</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SpecInt (by suppliers)</td>
<td>CPU integer</td>
<td>coarse</td>
<td>new hardware</td>
</tr>
<tr>
<td>Byte benchmark</td>
<td>computer platform performance</td>
<td>coarse</td>
<td>new hardware</td>
</tr>
<tr>
<td></td>
<td>OS, shell, file I/O</td>
<td></td>
<td>new OS release</td>
</tr>
<tr>
<td>file I/O</td>
<td>file I/O throughput</td>
<td>medium</td>
<td>new hardware</td>
</tr>
<tr>
<td>image processing</td>
<td>CPU, cache, memory</td>
<td>accurate</td>
<td>new hardware</td>
</tr>
<tr>
<td></td>
<td>as function of image, pixel size</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Objective-C overhead</td>
<td>method call overhead</td>
<td>accurate</td>
<td>initial</td>
</tr>
<tr>
<td></td>
<td>memory overhead</td>
<td></td>
<td></td>
</tr>
<tr>
<td>self made</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>socket, network</td>
<td>throughput</td>
<td>accurate</td>
<td>ad hoc</td>
</tr>
<tr>
<td></td>
<td>CPU overhead</td>
<td></td>
<td></td>
</tr>
<tr>
<td>data base</td>
<td>transaction overhead</td>
<td>accurate</td>
<td>ad hoc</td>
</tr>
<tr>
<td></td>
<td>query behaviour</td>
<td></td>
<td></td>
</tr>
<tr>
<td>load test</td>
<td>throughput, CPU, memory</td>
<td>accurate</td>
<td>regression</td>
</tr>
</tbody>
</table>
**Tools and Instruments Positioned in the Stack**

**typical small test program**

create steady state
\[ t_s = \text{timestamp}() \]
for(i=0;i<1M;i++) do something
\[ t_e = \text{timestamp}() \]
duration = \[ t_s - t_e \]

---

**Exploring an existing code base: measurements and instrumentation**

\[ \odot 2006, \text{Embedded Systems Institute} \]

version: 0.4
July 31, 2014
EBMbenchmarkPositions
Case 2: ARM9 Cache Performance

- CPU
- Instruction cache
- Data cache
- Memory cache
  -cache line size: 8 32-bit words

200 MHz

PCB

chip

100 MHz
Example Hardware Performance

memory request

22 cycles

memory response

word 1
word 2
word 3
word 4
word 5
word 6
word 7
word 8

38 cycles

memory access time in case of a cache miss

200 Mhz, 5 ns cycle: 190 ns
**Actual ARM Figures**

**ARM9 200 MHz**  
$t_{\text{context switch}}$ as function of cache use

<table>
<thead>
<tr>
<th>cache setting</th>
<th>$t_{\text{context switch}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>From cache</td>
<td>2 µs</td>
</tr>
<tr>
<td>After cache flush</td>
<td>10 µs</td>
</tr>
<tr>
<td>Cache disabled</td>
<td>50 µs</td>
</tr>
</tbody>
</table>
## Context Switch Overhead

The context switch overhead can be calculated using the following formula:

\[ t_{\text{overhead}} = n_{\text{context switch}} \times t_{\text{context switch}} \]

<table>
<thead>
<tr>
<th>( n_{\text{context switch}} ) (s(^{-1}))</th>
<th>( t_{\text{context switch}} )</th>
<th>( t_{\text{overhead}} )</th>
<th>CPU load overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>10µs</td>
<td>5ms</td>
<td>0.5%</td>
</tr>
<tr>
<td>5000</td>
<td>100ms</td>
<td>5%</td>
<td>1%</td>
</tr>
<tr>
<td>50000</td>
<td>1000ms</td>
<td>50%</td>
<td>10%</td>
</tr>
</tbody>
</table>

Where:
- \( t_{\text{context switch}} \) = 10µs
- \( t_{\text{context switch}} \) = 2µs

The CPU load overhead decreases as the number of context switches increases.

### Exploring an existing code base: measurements and instrumentation

©2006, Gerrit Muller

---

version: 0.4
July 31, 2014
PSRTcontextSwitchOverhead
system performance = f(
, 
, 
, 
, 
)
system performance = f(

- applications: hit-rate, miss-rate, #transactions, interrupt-rate, task switch rate, CPU-load
- services: transaction overhead: 25 ms
- operating system: interrupt latency: 10 us, task-switch: 10 us (with cache flush)
- hardware: cache miss: 190ns
- tools
)

Exploring an existing code base: measurements and instrumentation

version: 0.4
July 31, 2014
EBMiperformanceExample
Keep iterating!

- Keep iterating!
  - Zoom in on suspect parts
  - Code reading
  - Problematic dynamic properties
- New measurements and experiments
- Create (recover) insight in complex system

Exploring an existing code base: measurements and instrumentation

©2006, Embedded Systems Institute
189 ©2006, Gerrit Muller

version: 0.4
July 31, 2014
EBMiteration
Discussion propositions

0. many design teams have lost the overview of the system

1. a good (sw) architect has a quantified understanding of system context, system and software

2. a good design facilitates measurements of critical aspects for a small realization effort
Abstract
Performance Design is based on the application on many performance oriented patterns. Patterns are a way are to consolidate experience: what solution fits to what problem in what situation? Pitfalls are also a way to consolidate experience: what are common design mistakes?
Common Platforms and Bloating

Generic nature of platforms

Most SW implementations are way too big

Performance suffers from oversize and generic provisions
Exploring Bloating: Main Causes

>90% of all Software statements are not needed, but caused by:
  over-specification
  bad design
  too generic
  dogmatic rules
  legacy remains

core function
less than 10%

legend
overhead
value
### Necessary Functionality ➞ Intended Regular Function

<table>
<thead>
<tr>
<th>Testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>boundary behavior:</td>
</tr>
<tr>
<td>exceptional cases</td>
</tr>
<tr>
<td>error handling</td>
</tr>
<tr>
<td>regular functionality</td>
</tr>
<tr>
<td>instrumentation</td>
</tr>
<tr>
<td>diagnostics</td>
</tr>
<tr>
<td>tracing</td>
</tr>
<tr>
<td>asserts</td>
</tr>
</tbody>
</table>
The Danger of Being Generic: Bloating

client side

lots of config over-rides

lots of config over-rides

lots of config over-rides

toolbox side

lots of if-then-else

lots of configuration options

lots of stubs

lots of best guess defaults

over-generic class

generic design from scratch

in retrospect common (duplicated) code

specific implementations without a priori re-use

after refactoring

"Real-life" example: redesigned Tool super-class and descendants, ca 1994
Problem Propagation via Copy & Paste

needed code

bad code

copy paste modify

needed code

code not relevant for new function

repair code

bad code

new needed code

new bad code
Example of Problem Propagation

Class Old:
    capacity = startCapacity
    values = int(capacity)
    size = 0
    def insert(val):
        values[size]=val
        size+=1
        if size>capacity:
            capacity*=2
            relocate(values, capacity)

Class New:
    capacity = 1
    values = int(capacity)
    size = 0
    def insert(val):
        values[size]=val
        size+=1
        capacity+=1
        relocate(values, capacity)

Class DoubleNew:
    capacity = 1
    values = int(capacity)
    size = 0
    def insert(val):
        values[size]=val
        size+=1
        capacity+=1
        relocate(values, capacity)
        def insertBlock(v,len):
            for i=1 to len:
                insert(v[i])
Overhead Penalty of Modularity

- **modular**
  - fine grain

- **value**
  - medium grain

- **monolithic**
  - coarse grain

<table>
<thead>
<tr>
<th>81%</th>
<th>63%</th>
<th>44%</th>
</tr>
</thead>
<tbody>
<tr>
<td>overhead</td>
<td>overhead</td>
<td>overhead</td>
</tr>
<tr>
<td>value</td>
<td>value</td>
<td>value</td>
</tr>
</tbody>
</table>

---

Performance Patterns, Pitfalls, and Approach
©2006, Embedded Systems Institute
198 ©2006, Gerrit Muller

version: 0.1
July 31, 2014
EASRTcallTree
Function Call Overhead

- do something useful
- prepare parameters
- save state
- jump
- access parameters
- do something useful
- return
- restore state
- do something useful

Load and depth dependent (hidden) side effects
- pipeline flush
- I-cache disturbance
- D-cache disturbance

Legend:
- overhead
- value
Exercise Call Tree Overhead

Suppose:

Call Overhead = 10µs
Call graph branching factor = 2
Depth = 12

What is the Call overhead when all branches are followed?
Suppose:

Function call = 10µs
Call layer depth = 20
1024 calls per image

What is the maximum frame rate possible assuming that the complete CPU time is available for function calls?
Case 6

Common Platforms and Bloating

Platforms are overprovisioned and very generic

Are benefits > disadvantages?

Performance loss is significant and can be measured and modelled
Multi-Dimensional Viewing of many Images: Greedy and Lazy Design Patterns
Greedy versus Lazy

Greedy and Lazy systems

Greedy: pre-fetched lots of data:
System tries to have data available for the requesting system

Lazy: hardly of no pre-fetching of data:
System tries to set data available for the requesting system
only when asked for
### Example Greedy / Lazy (1)

<table>
<thead>
<tr>
<th>META DATA</th>
<th>META DATA</th>
<th>META DATA</th>
<th>META DATA</th>
<th>META DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>META DATA</td>
<td>META DATA</td>
<td>META DATA</td>
<td>META DATA</td>
<td>META DATA</td>
</tr>
<tr>
<td>META DATA</td>
<td>META DATA</td>
<td>META DATA</td>
<td>META DATA</td>
<td>META DATA</td>
</tr>
<tr>
<td>META DATA</td>
<td>META DATA</td>
<td>META DATA</td>
<td>META DATA</td>
<td>META DATA</td>
</tr>
</tbody>
</table>

**META DATA**

- Patient name
- Slice nr. / position
- Annotation
- Explanation
- Date / time

---

**Performance Patterns, Pitfalls, and Approach**

©2006, Embedded Systems Institute

©2006, Gerrit Muller

Version: 0.1

July 31, 2014

PSRTgreedyLazyImaging
Example Greedy / Lazy (2)

Lazy: Fetch only the requested image

Greedy: Fetch all the images in the set

In between options:
- Fetch requested image + surrounding images
- Fetch requested image + only meta information of images
Example Greedy / Lazy (3)

Lazy:
- low load on system
- long waiting time for next image

Greedy:
- high load on system
- possible long initial wait
- short response time insteady state

In between options:
- medium system load
- fast response for initialization and common image fetches
Initialization, Steady State and Finalization
Start-up, Steady State, Shut Down Scheme

Performance Patterns, Pitfalls, and Approach
©2006, Embedded Systems Institute
210 ©2006, Gerrit Muller

version: 0.1
July 31, 2014
CVstartUp
Trade-off:

Optimize on steady state may result in poor performance for initialization and process finish.

Optimize on Initialization and/or finish may result in poor steady state performance.
Common Performance Pitfalls

- Overhead
- Data bloating
- Cache thrashing
- Layering
- Process communication
- Conversions
- Serialization
- Backfiring optimalisations
- Hidden loads (bus, DMA etc)
- Poor algorithms
- Wrong dimensioning
Performance Design of Streaming Systems

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

Abstract
Video and audio content is a continuous stream of data. Video and audio systems have to be designed in such a way that these streams are processed and delivered continuously. We discuss the pipelining of multiple functions and the impact on bus bandwidth, memory use and CPU overhead.
Video Streaming

Hard real-time performance for distributed system with memory-bus

Trade-off of between latency, memory and overhead

Performance consideration in increasing detail
Case Video Streaming: Performance Design

Fixed HW diagram

Fixed algorithmic flow

Design process

- Latency
- CPU Overhead
- Memory use
- Bus load

- Granularity
- Synchronisation strategy
Video Streaming: Latency

Start P1
Start P2
Start P3
Start P4

Latency

Frame Available
T1
T2
T3
T4

P1
P2
P3
P4

P3_{Proc} \lesssim \frac{1}{2} T_{frame}

Latency \approx 2 T_{frame}

Legenda
- Task switch
- Process 1
- Process 2
- Process 3
- Process 4
Video Streaming: Resources

Overhead = \( (T_1 + T_2 + T_3 + T_4) \times \text{Frame rate} \)

Memory usage = \( 3 \times 2 \times \text{Frame size} \)

Bus load = \( \frac{3 \times 2 \times \text{Frame size} \times \text{Frame rate}}{\text{Bus capacity}} \)%

T1 .. T4 = Overhead to start P1 .. P4
Latency Calculation

Latency = Nr. of Proc. blocks \times \text{processing time per block} \times \text{frame fragment}

Memory = (Nr. of Proc. blocks - 1) \times 2 \times \text{pixels per frame} \times \text{frame fragment}

Overhead = Nr. of Proc. blocks \times \text{task switch time}

Overhead (%) = \frac{\text{Overhead}}{\text{Latency}}

Busload = \text{Memory usage} \times \text{frame fragment} \times (\text{frames/s}) / \text{BusCapacity}

(mind the units, ms vs. µs and kB vs MB!)

<table>
<thead>
<tr>
<th>lines</th>
<th>576</th>
<th>pixels per frame</th>
<th>414720</th>
</tr>
</thead>
<tbody>
<tr>
<td>pixels per line</td>
<td>720</td>
<td>Memory in kB</td>
<td>405</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Memory in MB</td>
<td>0.40</td>
</tr>
<tr>
<td>frame time</td>
<td>0.04</td>
<td>frame time in µs</td>
<td>40000</td>
</tr>
<tr>
<td>task switch time (µs)</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Processing per block</td>
<td>0.01</td>
<td>Processing in µs</td>
<td>10000</td>
</tr>
<tr>
<td>Bus capacity (MB/s)</td>
<td>500</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Line time (µs)</td>
<td>69</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Frame fragment</td>
<td>Full frame : 1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Exercise

Calculate:
- Processing time
- Overhead
- Memory Use
- Latency

for buffer size = 1/4 frame size
and for
buffer size = 1 video line
## Exercise Worksheet

<table>
<thead>
<tr>
<th>Nr of Processing Blocks</th>
<th>4</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block size</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Latency (ms)</td>
<td>40</td>
<td>200</td>
</tr>
<tr>
<td>Frame Memory (kB)</td>
<td>2430</td>
<td>15390</td>
</tr>
<tr>
<td>Overhead (µs)</td>
<td>40</td>
<td>200</td>
</tr>
<tr>
<td>Overhead (%)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Busload (%)</td>
<td>12.15</td>
<td>76.95</td>
</tr>
<tr>
<td>½ Frame Memory (kB)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Latency (ms)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 Memory (kB)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Overhead (µs)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Overhead (%)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Busload (%)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Line Memory (kB)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Latency (µs)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>576 Overhead (µs)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Overhead (%)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Busload (%)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| lines | 576 |
| pixels per line | 720 |
| pixels per frame | 414720 |
| Memory in kB | 405 |
| Memory in MB | 0.395508 |
| frame time | 0.04 |
| frame time in µs | 40000 |
| task switch time (µs) | 10 |
| Processing per block | 0.01 |
| Processing in µs | 10000 |
| Bus capacity (MB/s) | 500 |
| Line time (µs) | 69 |
### Changing the Buffer Size

<table>
<thead>
<tr>
<th>Buffer Size</th>
<th>Processing Time</th>
<th>Latency</th>
<th>Overhead</th>
<th>Memory Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{buffersize} = \frac{1}{4} \text{ frame} )</td>
<td>( \frac{1}{4} ) * original (per fragment)</td>
<td>( \frac{1}{4} ) * original</td>
<td>4 * original</td>
<td>( \frac{1}{4} ) original</td>
</tr>
<tr>
<td>( \text{buffersize} = 1 \text{ line} )</td>
<td>( \frac{1}{576} ) * original (per fragment)</td>
<td>( \frac{1}{576} ) * original + overhead</td>
<td>576 * original</td>
<td>( \frac{1}{576} ) original</td>
</tr>
</tbody>
</table>
Video Streaming

Properly designing distributed HRT systems requires trade-off between latency, overhead, and memory needs.

Performance model detailing dependent on significance of impact factors.
Measure functions or platform characteristics needed for “Fast Browser”. Select most critical characteristics
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 TNO-ESI. 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.
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

models analysis
- simulations measurements

diverse complex fuzzy performance expectations needs
# Toplevel Performance Design Method

1A Collect most critical performance and timing requirements

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

- measure
- evaluate
- analyse

- determine most important and critical requirements

- simulate
- build proto

- model
- analyse constraints and design options
Decomposition of System TR in HW and SW

- most and hardest TR handled by HW
- new control TRs

Hardware TR

System TR

Software TR

Original by Ton Kostelijk
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
  - Browse

- Toolboxes:
  - Audio
  - Video
  - TXT
  - Etc.

- Driver:
  - Drivers
  - Scheduler
  - Networking
  - OS

- Hardware:
  - Tuner
  - Frame-buffer
  - MPEG
  - DSP
  - CPU
  - RAM
  - Etc.
  - Signal Processing Subsystem
  - Control Subsystem

- Domain Specific
- Generic
Functional Decomposition
An example of a process decomposition of a MRI scanner.

The diagram illustrates the process decomposition of a MRI scanner. It is divided into two main categories: **scan control** and **image handling**. Each category further breaks down into various sub-processes and components:

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

- **Image Handling**:
  - Image handling UI
  - Database control
  - Archiving control
  - Import export
  - Display control
  - Disk
  - Media
  - Network
  - Display

The diagram also includes a legend indicating:

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

**Other architecture views**

**Execution architecture issues:**
- concurrency
- scheduling
- synchronisation
- mutual exclusion
- priorities
- granularity

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

**Hardware**
- CPU
- DSP
- RAM
  - tuner
  - drive

**Repository structure**
- Applications
  - play
  - zap
  - list
- UI toolkit
  - menu
- Processing
  - DCT
- Foundation classes
  - database
  - list
- Hardware abstraction
  - tuner
  - DVD drive

**Execution architecture**
- process
- task
- thread
- interrupt handlers

**Map**

**Dead lines**
- timing, throughput requirements
Layered Benchmarking Approach

typical values
interference
variation
boundaries

end-to-end function

network transfer
database access
database query
services/functions

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

duration
services
interrupts
task switches
OS services
CPU time
footprint
cache
applications

interrupt
task switch
OS services

duration
CPU time
footprint
cache

operating system

latency
bandwidth
efficiency

(locality
density
efficiency
overhead)

services

(system call
overhead)

(duration
footprint
interrupts
task switches
OS services)

(tools
version: 0.2
July 31, 2014
EBMbenchmarkStack

EBMIbenchmarkStack
©2006, Gerrit Muller

Performance Method Fundamentals
©2006, Embedded Systems Institute
239

CPU
cache
memory
bus
..
### Micro Benchmarks

<table>
<thead>
<tr>
<th>Category</th>
<th>In frequent operations, often time-intensive</th>
<th>Often repeated operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>database</td>
<td>start session, finish session</td>
<td>perform transaction, query</td>
</tr>
<tr>
<td>network, I/O</td>
<td>open connection, close connection</td>
<td>transfer data</td>
</tr>
<tr>
<td>high level construction</td>
<td>component creation, component destruction</td>
<td>method invocation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>same scope, other context</td>
</tr>
<tr>
<td>low level construction</td>
<td>object creation, object destruction</td>
<td>method invocation</td>
</tr>
<tr>
<td>basic programming</td>
<td>memory allocation, memory free</td>
<td>function call, loop overhead, basic operations (add, mul, load, store)</td>
</tr>
<tr>
<td>OS</td>
<td>task, thread creation</td>
<td>task switch, interrupt response</td>
</tr>
<tr>
<td>HW</td>
<td>power up, power down, boot</td>
<td>cache flush, low level data transfer</td>
</tr>
</tbody>
</table>
Home work reporting
To be indented here
Create “fast Browser” performance model. Finish measurements where needed