Benchmarking Real-World In-Vehicle Applications

NVIDIA GTC 2015-03-18
Agenda

- High-End embedded Processor System Design Challenge
- Theoretical Benchmarking Approach
- Practical Approach for critical Use Case Benchmarking
- Application specific Performance Optimization with Tegra
- Future System Integration Challenges
- Q&A
High-End embedded System Design Challenge
Background

- **mycable** defines and develops new Technologies for automotive and industrial Applications since 2001
  - Based on high performance, highly integrated processor systems

- Application specific performance is crucial for each processor subsystem

<table>
<thead>
<tr>
<th>Automotive</th>
<th>Industrial</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instrument Cluster</td>
<td>Digital Radar</td>
</tr>
<tr>
<td>Infotainment</td>
<td>3D Measurement</td>
</tr>
<tr>
<td>EV Connectivity</td>
<td>Optical Inspection</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Design Challenges

- **Available Performance Figures does not help**
  - CPU: Frequency, DMIPS, MFLOPs, etc.
  - GPU: GFLOPs, Mpoly/s, Mpixel/s, etc.
  - Even standard or application benchmarks often do not help
    - SPECint, EEMBC, etc.

- **Application specific Performance is very specific and depends on many Resources**
  - Of course on CPU and GPU performance, but also on
    - Volatile and non volatile memory size, performance and latency
    - Internal/external bus and interface bandwidth/arbitration including full path to memory
    - Cache sizes and architecture
    - Further acceleration units like hardware crypto acceleration, DSP, etc.
Further Design Challenges

- A Proof of Concept is usually not possible because
  - The system-on-chip you intend to use is not yet available
  - Evaluation system does have the interfaces of your application
  - Your application software is not yet ready

- Nevertheless Time is always critical in your project!

- A Model-based Approach can help you to analyze your application specific Performance on a not-yet-existing Processor Subsystem
  - By building a system resource model
  - By building an application specific software benchmark
Theoretical Benchmarking Approach
System Example

- A High-End Infotainment System is selected as an Example for Building a System Resource Model
  - An infotainment system is a good “worst case” example due to its combination of many tasks with different system resource demands
  - System use cases combining performance critical tasks with excessive usage of system resources

- Methodology can be transferred to any other system type
Theoretical Benchmarking Approach

1. Define your elementary high Level Tasks
   - Top level tasks running on your system combined in use cases

2. Create a System Resource Model from your System Architecture
   - Simplified architecture with focus on performance relevant resources

3. Quantify the particular Resource Demand for elementary Tasks
   - By calculation, estimation, simulation or extrapolation

4. Benchmark your System by combining elementary High Level Tasks to System Use Cases
   - Detect mismatches for critical resource usage like CPU performance, memory performance, bus bandwidths, and others
Define your elementary high Level Tasks

<table>
<thead>
<tr>
<th>System</th>
<th>3D Navigation</th>
<th>HMI</th>
</tr>
</thead>
<tbody>
<tr>
<td>• scheduler</td>
<td>• positioning</td>
<td>• 3D HMI rendering</td>
</tr>
<tr>
<td>• drivers</td>
<td>• dynamic route calculation</td>
<td>• HUD rendering</td>
</tr>
<tr>
<td>• services</td>
<td>• 3D map rendering</td>
<td>• speech recognition</td>
</tr>
<tr>
<td>• display output</td>
<td>• traffic information proc.</td>
<td>• text to speech</td>
</tr>
<tr>
<td>• file system,</td>
<td>• database access</td>
<td>• list handling</td>
</tr>
<tr>
<td>de/encryption</td>
<td>• ...</td>
<td>• ...</td>
</tr>
<tr>
<td>• ...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Entertainment</th>
<th>Connectivity</th>
<th>Driver Assistance</th>
</tr>
</thead>
<tbody>
<tr>
<td>• H.264 video decoding</td>
<td>• hands free telephony</td>
<td>• camera input</td>
</tr>
<tr>
<td>• audio decoding</td>
<td>• 3G/4G connectivity</td>
<td>• image pre-processing</td>
</tr>
<tr>
<td>• DRM handling</td>
<td>• WiFi AP</td>
<td>• object recognition</td>
</tr>
<tr>
<td>• audio processing</td>
<td>• gateway functionality</td>
<td>• camera view compositing</td>
</tr>
<tr>
<td>• video transcoding</td>
<td>• data de-/encryption</td>
<td>• warning generation</td>
</tr>
<tr>
<td>• ...</td>
<td>• ...</td>
<td>• ...</td>
</tr>
</tbody>
</table>

All trademarks herein before mentioned are the property of their respective owners.
Creating a simplified System Resource Model

= performance relevant resources

Tegra SoC

CPU

GPU

VIC ISP

Disp

bus system arbiter

DRAM I/F

audio DSP

HW crypto

video codec

periph. I/F

eMMC/ SATA

eMMC or SSD

DRAM

Tuner

Wifi Bluetooth

3G/4G

Amplifier

CAN

Ethernet

GPS

Gyro accel.
...

Disp

Wifi

Bluetooth

Amplifier

3G/4G

GPS

Gyro accel.
...
### Quantify particular Resource Demand for Tasks

<table>
<thead>
<tr>
<th></th>
<th>CPU MIPS</th>
<th>GPU GFLOPS</th>
<th>DRAM MB</th>
<th>DRAM bus MB/s</th>
<th>DSP MIPS</th>
<th>eMMC MB</th>
<th>SDIO MB/s</th>
<th>HW crypto MB/s</th>
<th>video dec. Mb/s</th>
<th>video enc. Mb/s</th>
<th>Ethernet Mb/s</th>
<th>USB #1 Mb/s</th>
<th>USB #2 Mb/s</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>3D Navigation</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>positioning</td>
<td>500</td>
<td>0</td>
<td>20</td>
<td>100</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>...</td>
</tr>
<tr>
<td>dynamic route calc.</td>
<td>2800</td>
<td>0</td>
<td>30</td>
<td>100</td>
<td>0</td>
<td>2G</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>...</td>
</tr>
<tr>
<td>3D map rendering</td>
<td>1500</td>
<td>100</td>
<td>650</td>
<td>3200</td>
<td>0</td>
<td>26G</td>
<td>12</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td><strong>HMI</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3D HMI rendering</td>
<td>1200</td>
<td>70</td>
<td>400</td>
<td>800</td>
<td>0</td>
<td>2G</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>...</td>
</tr>
<tr>
<td>speech recognition</td>
<td>300</td>
<td>0</td>
<td>50</td>
<td>100</td>
<td>5000</td>
<td>8G</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>...</td>
</tr>
<tr>
<td>display output</td>
<td>0</td>
<td>0</td>
<td>20</td>
<td>300</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

- **cache hit rate to be considered**
- **don’t forget system load**

- **Examples**

- **to be extended**
## Benchmark your System by System Use Cases

### Use Case #1
- driver seat: 3D navigation
- passenger seat: browser
- internet connectivity
- hands-free telephony
- audio processing
- ...

### Use Case #2
- driver seat: 3D HMI
- passenger seat: Blu-Ray playback
- speech recognition
- eye tracking
- ...

### Use Case #3
- driver seat: 3D navigation
- passenger seat: 3D HMI rendering
- traffic sign recognition
- text to speech
- ...

|                  | CPU MIPS | GPU GFLOPS | DRAM MB | DRAM bus MB/s | DSP MIPS | eMMC MB | SDIO MB/s | HW crypto MB/s | video dec. Mb/s | video enc. Mb/s | Ethernet Mb/s | USB #1 Mb/s | USB #2 Mb/s | ...
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Use Case #1</strong></td>
<td>16000</td>
<td>110</td>
<td>2.8G</td>
<td>28000</td>
<td>8500</td>
<td>56000</td>
<td>35</td>
<td>64</td>
<td>12</td>
<td>0</td>
<td>24</td>
<td>0</td>
<td>0</td>
<td>...</td>
</tr>
<tr>
<td><strong>Use Case #2</strong></td>
<td>11000</td>
<td>130</td>
<td>1.9G</td>
<td>11000</td>
<td>5200</td>
<td>35000</td>
<td>28</td>
<td>33</td>
<td>28</td>
<td>0</td>
<td>0</td>
<td>28</td>
<td>0</td>
<td>...</td>
</tr>
<tr>
<td><strong>Use Case #3</strong></td>
<td>9500</td>
<td>210</td>
<td>2.1G</td>
<td>14000</td>
<td>3400</td>
<td>32000</td>
<td>16</td>
<td>12</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>17</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

All trademarks herein before mentioned are the property of their respective owners.
Pitfalls in estimating integer Performance

- Maximum raw integer of a quad Core A57, L1 Cache hit @ 2 GHz
  - 4 x 2000 MHz x ~4.5 MIPS/MHz = 36,000 MIPS

- But single Thread Cache miss performance is different
  - Actual performance depends on external memory speed
  - May simply drop down to a few k MIPS
    - E.g. 1 x 2000 MHz / 4 (ext. memory) x 4.5 MIPS/MHz = 2,250 MIPS

- To be considered
  - Cache hit rates
  - Bus latencies, penalties, resource blocking
  - SMP mapping, ...
Practical Approach for critical Use Case Benchmarking
Practical Approach for critical Use Case Benchmarking

- Execution of individual benchmarks results in performance values which may not be reached if different applications share a resource in real life applications
  - Results may lead to too optimistic system architecture decisions

- Five Step Approach
  1. Isolate functions
  2. Find applicable benchmarks
  3. Estimate workload per task and scale benchmarks
  4. Execute benchmarks at the same time
  5. Find bottlenecks, Check system load, Test priority changes
**Practical Approach: Isolate Functions**

- **Isolate Functions from Critical Use Case Matrix which stress dedicated Resources from the System Resource Model**

  ✓ Already done in theoretical approach

<table>
<thead>
<tr>
<th>Ressource</th>
<th>HMI</th>
<th>Media-Database</th>
<th>Route-Calculation</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU</td>
<td>+++</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPU</td>
<td>+</td>
<td>+</td>
<td>+++</td>
</tr>
<tr>
<td>Memory</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>File-Access</td>
<td>+++</td>
<td></td>
<td>++</td>
</tr>
</tbody>
</table>
Practical Approach: 2 Find Appropriate Benchmarks

- Each Use Case can be represented by one or more benchmarks

<table>
<thead>
<tr>
<th>Resource</th>
<th>HMI</th>
<th>...</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU</td>
<td>+++</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPU</td>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Memory</td>
<td>++</td>
<td></td>
<td></td>
</tr>
<tr>
<td>File-Access</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

HMI-Simulation or -MockUp using elements and technologies as required: e.g. animations, transitions, OpenGL...

Standard-Benchmark like pmbw (parallel bandwidth memory benchmark)
Practical Approach: Find Appropriate Benchmarks

- Isolate Functions from Critical Use Case Matrix which stress dedicated Resources from the System Resource Model

<table>
<thead>
<tr>
<th>Resource</th>
<th>...</th>
<th>Media-Database</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPU</td>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Memory</td>
<td>++</td>
<td></td>
<td></td>
</tr>
<tr>
<td>File-Access</td>
<td>+++</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

e.g. filebench-benchmark with database-access profile
Practical Approach: 2 Find Appropriate Benchmarks

- Isolate Functions from Critical Use Case Matrix which stress dedicated Resources from the System Resource Model

<table>
<thead>
<tr>
<th>Resource</th>
<th>...</th>
<th>Route Calculation</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU</td>
<td></td>
<td>+++</td>
</tr>
<tr>
<td>CPU</td>
<td></td>
<td>+++</td>
</tr>
<tr>
<td>Memory</td>
<td></td>
<td>++</td>
</tr>
<tr>
<td>File-Access</td>
<td></td>
<td>++</td>
</tr>
</tbody>
</table>

Route Calculation Benchmark performing typical algorithms on map-data
e.g. filebench with a defined multiple read access profile on a file profile fitting to typical map data
Practical Approach: Estimate Workload per Task

- Create Configurations for:

<table>
<thead>
<tr>
<th>Resource</th>
<th>Route Calculation</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU</td>
<td>+++</td>
</tr>
<tr>
<td>CPU</td>
<td>++</td>
</tr>
<tr>
<td>Memory</td>
<td>+</td>
</tr>
<tr>
<td>File-Access</td>
<td>++</td>
</tr>
</tbody>
</table>

Next to the behavior (e.g. load, files, number of accesses) you can define scheduler priorities within the configuration.

Route Calculation Benchmark performing typical algorithms on map-data

E.g. filebench with a defined multiple read access profile on a file profile fitting to typical map data

<table>
<thead>
<tr>
<th>Average Load</th>
<th>Peak Load</th>
</tr>
</thead>
<tbody>
<tr>
<td>Route Calc.</td>
<td>Route Calc.</td>
</tr>
<tr>
<td>Average Config</td>
<td>Peak Config</td>
</tr>
<tr>
<td>Filebench Average Config</td>
<td>Filebench Peak Config</td>
</tr>
</tbody>
</table>

All trademarks herein before mentioned are the property of their respective owners.
Practical Approach: Execute Benchmarks

For fast reproduction and modification of results it is recommended to use an execution framework.

The configurations for all load benchmarks and for the target benchmark (the benchmark under test) is loaded by the framework and executed in always the same way.

The framework:
- Checks the correct execution and detects hanging/dying processes
- Collects settings and results in a defined output format
- Informs with onscreen and webview about live values (e.g. fps)
- Is written in an easy portable script language (e.g. python)
Practical Approach: Execute Benchmarks

- The framework ensures reproducible results.
- The target benchmark runs on a specified system load.
- Each system resource can be stressed with a combination of real life usages.
- Negative side effects can be disclosed.
Results and Opportunities

- The results can be **compared for different systems**
  - e.g. compare SoC generations
- Bottlenecks can be identified
- Changes in current systems can be simulated in advance without generating/porting a whole system image/set of applications
- Simulations of load profiles can be **repeated** on different architectures/SoCs with less porting effort (only framework and benchmarks)
- Other benchmarks like EEMBC or SpecInt/SpecFP can be used as target benchmarks for specific performance analysis under specific load conditions
Application specific Performance Optimization with Tegra
Application specific Performance Optimization with Tegra

- **Certain Tasks consume complete available System Resources**
  - **Route Calculation**
    - Uses all available integer CPU performance
  - **Speech Recognition**
    - Uses all available DSP performance
  - **Camera inputs, Display Outputs**
    - High permanent usage of memory bandwidth

- **But available GPU performance on Tegra Processors often exceeds application specific demand**
  - Even high resolution 3D navigation and 3D HMI rendering does often not require full GPU performance
Application specific Performance Optimization with Tegra

- Moving Tasks to GPU enables higher system specific performance
  - Data types must fit
  - CUDA eases algorithm porting
  - K1: 192 CUDA cores

- Examples for moving Tasks to GPU
  - Speech recognition
  - Driver assistance
  - Image and video processing
  - Basic algorithms
    - FFT
    - Linear algebra, etc.
Future System Integration Challenges
Future System Integration Challenges

- SoC Performance drives Automotive Component Integration
  - Partial or full integration of instrument cluster, infotainment and driver assistance
Future System Integration Challenges

- **K1 ⇒ X1** Performance Scalability paves the way to further System Integration
  - CPU 32bit ⇒ 64bit
  - GPU Kepler ⇒ Maxwell
  - Memory Bandwidth ⇒ x 2 (LPDDR4)
  - Video ⇒ 4k

- Further Properties
  - CPU virtualization
  - GPU virtualization
Thank you for your Attention