Automated Parallelization for Embedded Multicore Platforms

Dr.-Ing.
Timo Stripf
Managing Director Technology
Outline

- Company Introduction
- Parallelization
- Vectorization (e.g. RISC-V)
- Platform Specification (SHIM and beyond)
- Summary
emmtrix Technologies GmbH - Basics

- Founded 2016
- Located in Karlsruhe
- Currently 14 people
- Software products:
  - emmtrix Parallel Studio
  - emmtrix Performance Estimation
  - emmtrix C++2C Compiler
- Services:
  - Tool Customization
  - Integration & Support
  - Trainings
  - Technical Consulting
emmtrix Focus Areas

Software Parallelization

- Parallel C code for
  - Multi-/Manycore CPUs
  - Vector processing units
  - GPUs andDSPs
- Interactive workflow
- Parallelization with functional safety

Performance Estimation

- Static performance analysis
  - C
  - LLVM IR
  - Assembly
- Simulation
- Profiling on hardware
- Intuitive visualization

Code Conversion

- Conversions:
  - Simulink to MATLAB®
  - MATLAB®/Octave/Scilab to C
  - C++ to C
- User-controlled optimizations
- Aimed at embedded systems and automatic analysis

Static Source Code Analysis

- Data dependencies analysis
- Event chain analysis
emmrrix Tool Workflow

Input
- C++
- Simulink®
- MATLAB®
- GNU Octave
- Scilab

Code Generation & Conversion
- C++ to C Compiler
- MATLAB®

Parallelization
- Parallel Studio
- ePS Qualification Kit

Target Platforms
- Multicore
- GPU
- DSP

Tools
- ISO 26262
- DO-178/330
- IEC 61508
- EN 50128

Platforms
- C sequential
- C++
- Parallel

Languages
- C++
- Simulink®
- MATLAB®
- GNU Octave
- Scilab
The emmtrix Solution - visualize and stay in control

- MATLAB®/Simulink Model
- emmtrix Workflow
- Hierarchical Program View
- Sequential Reference
- Parallel Schedule
Parallelization
Interactive Parallelization

MATLAB®

Scilab

C

Control

Feedback

CPUs

FPGAs

GPUs
emmtrix Workflow

**Input / C Code Generation**
- Convert input model to C code

**Performance Estimation**
- Static code analysis to estimate the performance on the target platform

**Schedule & Mapping**
- Interactive parallelization: visualization, transformations, constraints

**Parallel Code Generation**
- C code generation for the target platform

*Iterative optimization*
Automatic Parallelization Levels

- Algorithmic Level
  - Different algorithm versions in MATLAB
- Code Transformation Level
  - Transformation selection in GUI
- Task Level
  - Automatic, Constraints
- Communication Level
  - Automatic
Vectorization
Vector Processing

Key challenges:

- Efficient utilization of all slots (e.g. 16 * 32 bit operations)
- Provide required data at right time for the local memory
Vectorization in ePS

- Perform auto-vectorization (optimized for generated code)
- Optimize your code by selecting code transformations in the ePS GUI
- The generated “Vector C” code is readable
for (i2 = 0u; i2 < 15u; i2++) {
    for (i1 = 0u; i1 < 15u; i1++) {
        sum1 = 0.0f;
        for (i3 = 0u; i3 < 15u; i3++) {
            sum1 += In1[i3][i1] * In2[i3][i2];
        }
        Out1[i2][i1] = 0.00625f * sum1;
    }
}

for (i2 = 0u; i2 < 15u; i2++) {
    vstoreN(0.0f, &sum1[0]);
    for (i3 = 0u; i3 < 15u; i3 = i3 + 1) {
        vstoreN(vloadN(&sum1[0]) + vloadN(&In1[i3][0]) * In2[i3][i2], &sum1[0]);
    }
    vstoreN(0.00625f * vloadN(&sum1[0]), &Out1[i2][0]);
}
emmtrix Model Code Generator

- Generate C code from Simulink models and MATLAB scripts
  - Optimized for data intensive code (e.g. with matrix operations)
  - Generates code prepared for auto-vectorization
- Features fusion of Simulink blocks
  - Generate a single for-loop for multiple blocks
  - Reduces unnecessary operations and instruction memory usage
  - Improves data locality as well as memory consumption
- Control memory layout
  - e.g. row-major vs. column-major
  - Important for matrix multiplication optimization
Code fusion example

Unoptimized C code

```c
for (i2 = 0u; i2 < 15u; i2++)
    for (i1 = 0u; i1 < 15u; i1++)
        Transpose[i2][i1] = In2[i1][i2];
for (i4 = 0u; i4 < 15u; i4++) {
    for (i3 = 0u; i3 < 15u; i3++){
        sum1 = 0.0f;
        for (i5 = 0u; i5 < 15u; i5++)
            sum1 += In1[i5][i3] * Transpose[i4][i5];
        Product[i4][i3] = sum1;
    }
}
for (i7 = 0u; i7 < 15u; i7++)
    for (i6 = 0u; i6 < 15u; i6++)
        Out1[i7][i6] = 0.00625f * Product[i7][i6];
```

Fused C code

```c
for (i2 = 0u; i2 < 15u; i2++) {
    for (i1 = 0u; i1 < 15u; i1++) {
        sum1 = 0.0f;
        for (i5 = 0u; i5 < 15u; i5++)
            sum1 += In1[i5][i3] * Transpose[i4][i5];
        Product[i4][i3] = sum1;
    }
}
for (i7 = 0u; i7 < 15u; i7++)
    for (i6 = 0u; i6 < 15u; i6++)
        Out1[i7][i6] = 0.00625f * Product[i7][i6];
```

- Gain inside loop
- Transpose by index swapping
- Saved temporary variables (Product, Transpose)
SHIM and Platform Specification
Platform Specification

- Multicore platform
  - Architecture (SHIM v1)
    - Cores
    - Core types (microarchitecture)
    - Frequency
  - Properties (e.g. SIMD)
  - Code generation template
  - Communication performance
    (microarchitecture x microarchitecture x type)
    - Send blocking time
    - Receive blocking time
    - Transfer time
Platform Specification (Microarchitecture)

- Microarchitecture
  - Compiler information (types & macros)
  - Performance information

<table>
<thead>
<tr>
<th>Type</th>
<th>Requirement</th>
<th>Accuracy</th>
<th>Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>Readable C code</td>
<td>Low (compiler optimization)</td>
<td>Cost of C operators</td>
</tr>
<tr>
<td>LLVM IR</td>
<td>Generic compilable code</td>
<td>Medium (different compiler optimizations)</td>
<td>Cost of LLVM IR instruction</td>
</tr>
<tr>
<td>Assembly</td>
<td>Supported target compiler Installed target compiler</td>
<td>High (pipeline model)</td>
<td>Cost of assembly instr Assembly syntax Pipeline model</td>
</tr>
<tr>
<td>Functions</td>
<td>Readable C code</td>
<td></td>
<td>Cost of C functions, e.g. sin/cos (math function), memcpy</td>
</tr>
</tbody>
</table>
ADL Classification

- EMX Communication
  - Performance
- SHIM Instr.
  - Performance (LLVM IR)

SHIM
- SHIM Communication

Figure 1: Gajski-Kuhn Y-chart
How can we increase the adoption of SHIM?

- **Tool providers**
  - Processor ADLs are typically very tool driven (content defined by tool's needs)
    - Extensibility
  - Benefit from reusability
    - Open database / catalogue of SHIM descriptions of different processors

- **Hardware providers**
  - ?
QUESTIONS?