Technology and Design Tools for Multicore Embedded Systems Software Development

Yuriy Sheynin, Alexey Syschikov, Boris Sedov

Saint Petersburg State University of Aerospace Instrumentation
Why do we need such technology?

1. “Two-in-one” developer is required: skilled domain experts + skilled programmer

2. Contradictive requirements to hardware platforms

3. Development of an algorithm and program should be started before the selection of a specific platform

4. Hardware platforms become more and more complex, includes many cores, are heterogeneous in all aspects (cores, memory, interconnect)

5. In order to achieve the necessary requirements an adaptation of algorithms for the platform and the platform for algorithms is needed
The life cycle of the program in the technology

Frontend

Algorithms design and programming

Middlend

Performance evaluation

Backend

Deployment to target platforms
The life cycle of the program in the technology

- Algorithms design and programming
- Performance evaluation
- Deployment to target platforms
The life cycle of the program in the technology

- Algorithms design and programming
- Performance evaluation
- Deployment to target platforms

Platform 1
The life cycle of the program in the technology

1. Algorithms design and programming
2. Performance evaluation
3. Deployment to target platforms

Platforms:
- Platform 1
- Platform N
- Platform 2
The high-level technology structure

Expert in the domain area

Visual development environment → Validation → Verification → Debugging (interactive)

Static analyzer

Platform description

Programmer

Library functions

VPL simulator

Platform simulator

Optimizer

Code generators

Tables
Graphics
Statistics

Characteristics
Graphics
Statistics

Sequential
C/C++
Assembler

Parallel
OpenMP
Threads
RT run-time
The high-level technology structure

Algorithms design and programming

Static analyzer

Platform description

Programmer

Library functions

VPL simulator

Platform simulator

Optimizer

Code generators

Tables Graphics Statistics

Characteristics Graphics Statistics

Sequential C/C++ Assembler

Parallel OpenMP Threads RT run-time

Expert in the domain area

Visual development environment

Validation

Verification

Debugging (interactive)
The high-level technology structure

Simulation and early estimation

1. Expert in the domain area
   - Visual development environment
   - Validation
   - Verification
   - Debugging (interactive)

2. Static analyzer
   - XIR executable specification
   - Platform description
   - Programmer
   - Library functions
   - VPL simulator
   - Platform simulator
   - Optimizer
   - Code generators
   - Tables Graphics Statistics
   - Characteristics Graphics Statistics
   - Sequential C/C++ Assembler
   - Parallel OpenMP Threads RT run-time
The high-level technology structure

Deployment to target platforms

1. Expert in the domain area
   - Visual development environment
   - Validation
   - Verification
   - Debugging (interactive)

2. Static analyzer
   - VPL simulator
     - Tables Graphics Statistics
     - Characteristics Graphics Statistics

3. Platform description
   - Programmer
     - Library functions
     - Optimizer
     - Code generators
     - Sequential C/C++ Assembler
     - Parallel OpenMP Threads RT run-time
The Visual Programming Language (VPL)

- Separation of design and programming (coding)
The Visual Programming Language (VPL)

- Separation of design and programming (coding)

C/C++
OpenCL
Asm
...
The Visual Programming Language (VPL)

- Separation of design and programming (coding)
- Flexibility and ease-of-change at any design stage
- Explicit parallel program scheme control and management
- No direct coder influence on a parallel program scheme
- Decreasing errors possibility without sacrificing parallel program visibility
- Efficient program maintenance during the whole lifecycle
The Visual Programming Language (VPL)

- Cognitive advantages:
  - clear view of the development process
  - traceability of the dependency graph
  - calculations control structures
  - shared data
  - natural parallelism
  - potential pipelining

Program on VPL – scheme that consist from calculation operator, control operators and share data
VPL and Domain Specific Programming

- DSL libraries provide convenience of design within an application domain
- Easy re-use of development results
- Easy to involve domain experts in embedded software development
- Simplicity of language and libraries extensions by users

- CImg
- OpenCV
- OpenVX
- Lowlevel math
- Arduino
- ...
1. Analysis of the domain area

2. Creation of the functional elements (FE) library

3. Development of FE functionality on C++ & OpenCV

An example: DSL creation for image processing (OpenCV)
An example: DSL usage for image processing

Image recognition

4. The scheme is designed of DSL and basic VPL language elements
An example: DSL usage for image processing
Face/eyes recognition

4. The scheme is designed of DSL and basic language elements
About OpenVX

OpenVX
• C-based programming approach with mixed C/non-C computing model
• Includes functions and data types of video processing domain area
• Functions library can be expanded, but it is inconvenient (non-portable)

OpenVX support in VIPE
• Full implementation for spec. v.1.0.1 (working on v.1.1)
• VPL:
  • DSL for OpenVX functions
  • OpenVX-specific data objects
• Code generation:
  • Plain C mode with OpenVX functions (vxu)
  • OpenVX graph mode
  • **Mixed mode with OpenVX graphs and other DSLs**

An OpenVX graph – a limited subset of VPL program schemes. VPL scheme + OpenVX functions combines all benefits
Asynchronous Growing Processes (AGP) formal computational model

**AGP defines:**
- VPL language syntax
- semantics of VPL language objects
- control units

**AGP provides:**
- formal verification
- identical results in different run-time environments
- dynamics of parallel computations
- combination of working in shared and distributed memory models

AGP – the single model for all types of parallel computations and kernel - data interaction (shared memory / message passing)
Visual programming environment: VIPE
Development process support tools:

- parallel program scheme validation

  - syntax checking
  - types checking
  - links correctness
  - scheme completeness
  - etc.
Interactive tools

Development process support tools:
• parallel program scheme validation
• verification (in progress)
• unambiguity
• deadlocks
• livelocks
• finiteness
• etc.
Interactive tools

Development process support tools:
- parallel program scheme validation
- verification (in progress)
- interactive debugging, etc.
- step-by-step debug
- breakpoints
- watches
- data transfers
- computation traces
- operators executions
- functional debugging by serial execution
- etc.
Early program estimation and evaluation

1. Expert in the domain area
2. Visual development environment
3. Validation
4. Verification
5. Debugging (interactive)
6. Static analyzer
7. Platform description
8. Programmer
9. Library functions
10. VPL simulator
11. Platform simulator
12. Optimizer
13. Code generators
15. Characteristics Graphics Statistics
16. Sequential C/C++ Assembler
17. Parallel OpenMP Threads RT run-time
Performance evaluation. Visual Profiler

Hot-spots detection

Modes:
• Absolute execution time of each node
• Relative execution time of each node
• Hot-spots
Performance evaluation. Static Analysis
Fast, early estimation of the program performance on the many core platform

Time reduce

\[ R = \frac{T_n}{T_1} \times 100\% \]

Speedup

\[ S = \frac{T_1}{T_n} \]

Efficiency

\[ E = \frac{T_1}{N \times T_n} \]

\( T_1 \) – program execution time on the 1 processor

\( T_n \) – program execution time on \( N \) processors

\( N \) – number of processors
Performance evaluation. VPL Simulator

Allows estimating:

1. Performance requirements for cores of the embedded system
2. Memory requirements
3. Load balance of various allocations
4. Volume and intensity of data exchange
5. Efficiency of hardware occupation
6. Bottlenecks of hardware platform, program and task distribution
Support of heterogeneous platforms programming

- Mapping operators to one or several core types (CPU, GPU, DSP, DMA)
  - Operators on various core types
  - Data on various data types
  - Data exchanges on various connection types
- Selecting the implementation for data processing operators
- Preparation of initial data and the results of operator of the program, taking into account the specifics of the different communication mechanisms
Heterogeneous allocation

- Memory pre-allocation
- Start of computations
- Dynamic unrolling on CPU
- Offload the parallel loop to DSP cluster
- Dynamic unrolling on DSP (another loop)
- Results back to CPU, end of computation
Deployment to target platforms

Working prototypes
- ANSI C
- C++
- RT-run-time on a multicore platform (in progress)
- Parallel OpenMP

Proof of concept
- Parallel threads
- MPI
- Assembler MIPS, DSP
Use cases and demonstrators
Use case: face identification
Task description

• Program is developed with VIPE
Use case: face identification

- **Software part:** low quality of face recognition
- **Hardware part:** Ci20 (perspective – ELISE by ELVEES)
- **Computations only on the CPU**
- **Works slowly**
Terminator Vision System. Student project

Autonomous Cyber-Physical System combining multicore computations, control and mechanical parts. The Vision System identifies people from the database and tracks them with rotating camera. Project presented on hackster.io + Imagination challenge: https://www.hackster.io/contests/Ci20

- Project is developed with VIPE
- Face recognition is performed by using training neural network
- Database of faces was created for face classification
- Tracking is performed by using servo, which is controlled by Arduino that receives commands from the Ci-20 board
Use case: number plate recognition

- Program developed with VIPE
- 2 threads are used
- OpenGL
Use case: number plate recognition

- Software part: low quality of number plate recognition
- Hardware part: Ci20 (perspective – ELISE by ELVEES)
- Computations only on the CPU
- Works slowly
Use case: number plate recognition

Camera → Regions Of Interests → Symbol detection of each number plate → Result output

© 2006 OMS & MCIT GUAP

Symbol detection of each number plate

Result output
Use case: number plate recognition
Scheme of working with Imagination Creator Ci20 board

PC with VIPE → C code → Compile → Run → Result

Webcam

Wi-Fi
VIPE one button deployment
Feature tracking (OpenVX)

DSL and design

DSL, based on OpenVX

Feature tracking program in VIPE
Feature tracking (OpenVX) Results

Feature tracking program run on the x86 platform with using the sample implementation by Khronos
Feature tracking
Static analysis

Performance estimation of the feature tracking program with **sequential** frame processing

Performance estimation of the feature tracking program with **parallel** frame processing
Large amount of time is taken by image format conversion function (from OpenVX format and back)

Profiling of the feature tracking program
Traffic radar object detection is developed in the PaPP project, part of the European technical platform ARTEMIS/ECSEL.
Traffic radar object detection
Static analysis

However, the results were worse than expected. Static analysis of subprogram "Data processing units" shows close to a linear reduction of time for 8 cores.
Traffic radar object detection
Visual Profiler

Visual Profiler shows, that a large amount of time is taken by function for reading the input file (prototype uses data from the input file).

File reading function is in sequential part, hence the parallelism is limited by Amdahl's law. Actual process of getting the input data should be optimized to take less time. Evaluation of program with reduced operating time of reading function shows satisfactory results.
## Traffic radar object detection

Comparison of the results of analysis, simulation and execution

<table>
<thead>
<tr>
<th>Cores (simulator) or threads (OpenMP)</th>
<th>Static analysis</th>
<th>Modeling VPL</th>
<th>Execution</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>sec. / %</td>
<td>sec. / %</td>
<td>sec. / %</td>
</tr>
<tr>
<td>Without OpenMP</td>
<td></td>
<td></td>
<td>1.60</td>
</tr>
<tr>
<td>1</td>
<td>1.26 / 100</td>
<td>1.29 / 100</td>
<td>1.65 / 100</td>
</tr>
<tr>
<td>2</td>
<td>0.64 / 50.8</td>
<td>0.88 / 68.2</td>
<td>1.00 / 60.6</td>
</tr>
<tr>
<td>3</td>
<td>0.60 / 47.6</td>
<td>0.72 / 55.8</td>
<td>0.81 / 49.1</td>
</tr>
<tr>
<td>4</td>
<td>0.34 / 30.0</td>
<td>0.55 / 42.6</td>
<td>1.25 / 76.7</td>
</tr>
</tbody>
</table>

### Hardware platform
- Core i7 8 cores -> VirtualBox VM 4 cores
- Ubuntu 14.04, GCC 4.9.2.

### Input data
- 12 MB signal samples
Summary

• Technology covers various requirements of embedded SW development
  • Design, programming, evaluation, porting etc.
• DSLs for involving domain specialists into development process
• Rapid SW prototyping for early customer presentations
• Formal model basis for proofed and predictable results, including debugging
• Fast tools adaptation for new cores and platforms

• Supporting development tool for heterogeneous cores, platforms, system software infrastructure