Mapping of Applications to Multi-Processor Systems

Peter Marwedel
TU Dortmund, Informatik 12
Germany

2014年 01 月 17 日

These slides use Microsoft clip arts. Microsoft copyright restrictions apply.
Structure of this course

Numbers denote sequence of chapters
The need to support heterogeneous architectures

Energy efficiency a key constraint, e.g. for mobile systems

Unconventional architectures close to IPE

© Hugo De Man/Philips, 2007

© Renesas, MPSoC'07

How to map to these architectures?
Practical problem in automotive design

Which processor should run the software?

- Early
- Quickly
- Cost-efficient

Evaluate alternatives („what if ?“)
- Mapping
- Scheduling
- Communication
A Simple Classification

<table>
<thead>
<tr>
<th>Architecture fixed/Auto-parallelizing</th>
<th>Fixed Architecture to be designed</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Starting from given task graph</strong></td>
<td><strong>Map to CELL, Hopes, Qiang XU (HK) Simunic (UCSD)</strong></td>
</tr>
<tr>
<td><strong>Auto-parallelizing</strong></td>
<td><strong>Mnemee (Dortmund) Franke (Edinburgh) MAPS</strong></td>
</tr>
</tbody>
</table>
Example: System Synthesis

Given:

- algorithm
- mappings
- architectures

Goal:

- schedule
- mapping
- architecture

Objectives: cost, latency, power consumption
Basic Model – Problem Graph

Problem graph $G_P(V_P,E_P)$:

Interpretation:

- $V_P$ consists of functional nodes $V_P^f$ (task, procedure) and communication nodes $V_P^c$.
- $E_P$ represent data dependencies.
Definition: A specification graph is a graph $G_S=(V_S,E_S)$ consisting of a problem graph $G_P$, an architecture graph $G_A$, and edges $E_M$. In particular, $V_S=V_P \cup V_A$, $E_S=E_P \cup E_A \cup E_M$. 
Design Space

Communication Templates

Computation Templates

Scheduling/Arbitration

Which architecture is better suited for our application?

Architecture # 1

Architecture # 2
Evolutionary Algorithms for Design Space Exploration (DSE)

- Selection
- Recombination
- Mutation

"Chromosome" = encoded allocation + binding

Individual

Allocation

Binding

Decode allocation

Decode binding

Scheduling

Fitness evaluation

Fitness

User constraints

Design point (implementation)
Challenges

• Encoding of (allocation+binding)
  • simple encoding
    • eg. one bit per resource, one variable per binding
    • easy to implement
    • many infeasible partitionings
  • encoding + repair
    • eg. simple encoding and modify such that for each $v_p \in V_P$ there exists at least one $v_a \in V_A$ with $\beta(v_p) = v_a$
    • reduces number of infeasible partitionings
• Generation of the initial population, mutation
• Recombination
EXPO – Tool architecture (1)

MOSES

task graph, scenario graph, flows & resources

EXPO

system architecture performance values

SPEA 2

selection of “good” architectures

Exploration Cycle

© L. Thiele, ETHZ
EXPO – Tool architecture (2)

Tool available online: http://www.tik.ee.ethz.ch/expo/expo.html
EXPO – Tool (3)

Implementation Nr. 60611 (EXPO, Institute TIK, ETH Zurich)

Scenario: Scen2
Optimal Scaling Factor: 0.530
Total Memory: 8.295

- DSP
  Utilization: 79%

- CheckSum
  Utilization: 4%

- LookUp
  Utilization: 7%

Flow: RTSend
- RTPx
- VoiceEnc
- LinkTx
- Schedule
- Encrypt
- AHVerify
- Classify
- LinkRx
Priority: 5
Acc. Waiting Time in Queue: 0.000

Flow: NRTDecrypt
- ESPDecaps
- ProcessIP
- IPModify
- LinkTx
- Schedule
- Decrypt
- AHVerify
- Classify
- LinkRx
Priority: 4
Acc. Waiting Time in Queue: 0.000

Flow: RTRcv
- Dejitter
- VoiceDec
- ProcessIP
- RTPx
- Classify
- LinkRx
Priority: 1
Acc. Waiting Time in Queue: 0.000

Flow: NRTForward
- ProcessIP
- VerifyIP
- UDPRx
Priority: 3
Acc. Waiting Time in Queue: 23.088

Initialisation sequence started.
Static parameters read.
Problem specification read.
Initial population constructed.
Initial population written to file.
Generation counter set to 1.
****** Generation 1******
All active gene IDs read.
Population before cleaning: 101 elements.
Population after cleaning: 101 elements.
Clean of population finished.
Population written to file.
Genes for variation read.
Variation finished.
****** Generation 2******
All active gene IDs read.
Application Model

Example of a simple stream processing task structure:
behavioral specification of a video codec for video compression
Exploration – Case Study (2)

problem graph of the video coder

Diagram showing the problem graph of the video coder with nodes and arrows indicating the flow of operations.
Exploration – Case Study (3)
More Results

Performance for encryption/decryption

Performance for RT voice processing

- DSP
  NRT: 64%
  RT: 39%

- Cipher
  NRT: 71%
  RT: 0%

- LookUp
  NRT: 15%
  RT: 6%

- Classifier
  NRT: 27%
  RT: 11%

- DSP
  NRT: 35%
  RT: 39%

- LookUp
  NRT: 1%
  RT: 6%

- Classifier
  NRT: 1%
  RT: 11%
Extension: considering memory characteristics with MAMOT

Example
Gains resulting from the use of memory information

Design Space Exploration with SystemCoDesigner (Teich et al., Erlangen)

- System Synthesis comprises:
  - Resource allocation
  - Actor binding
  - Channel mapping
  - Transaction modeling

- Idea:
  - Formulate synthesis problem as 0-1 ILP
  - Use Pseudo-Boolean (PB) solver to find feasible solution
  - Use multi-objective Evolutionary algorithm (MOEA) to optimize Decision Strategy of the PB solver

System Synthesis (Actor Binding)

- $A$ denotes the set of actors
- Actor binding activation $\alpha: A \times R \rightarrow \{0, 1\}$
- $\alpha(a, r) = 1$ binds actor $a$ onto resource $r$
- $\forall a \in A: \sum \alpha(a, r) = 1$ (Each actor is bound exactly once)
A 3rd approach based on evolutionary algorithms: SYMTA/S:

# A Simple Classification

<table>
<thead>
<tr>
<th>Architecture fixed/Auto-parallelizing</th>
<th>Fixed Architecture</th>
<th>Architecture to be designed</th>
</tr>
</thead>
</table>
| **Starting from given task graph**    | **Map to CELL, Hopes**  
  Qiang XU (HK)  
  Simunic (UCSD) | **COOL codesign tool; EXPO/SPEA2**  
  SystemCodesigner |
| **Auto-parallelizing**                | **Mnemee (Dortmund)**  
  **Franke (Edinburgh)**  
  MAPS | **Daedalus** |
A fixed architecture approach: Map → CELL

The problem of allocating and scheduling task graphs on processors in a distributed real-time system is \textbf{NP-hard.}
Partitioning into Allocation and Scheduling

Resource constraints

Allocation:
INTEGER PROGRAMMING

Master problem (MP)

Obj. Function

All+Sch
LB for cost

Timing constraint

UB for cost

No good: unfeasible SP
Optimality cut: SP solution
is optimal UNLESS a better
one exists with a different
allocation

Scheduling:
CONSTRAINT PROGRAMMING

Subproblem (SP)

Iterations stop when MP becomes unfeasible!
HOPES Proposal

Model-based Programming (PeaCE model)
KPN

UML

Common Intermediate Code (CIC)

Generic API

CIC (w/ API) translator

Per-processor code

Static C code analysis

API lib.

Perf. & power estimation

Virtual prototyping

Processor ISS

SW Platform 1

Target HW platform

Jan. 24, 2007

Soonhoi Ha, SNU

© p. marwedel, informatik 12, 2014

technische universität dortmund
## A Simple Classification

<table>
<thead>
<tr>
<th>Architecture fixed/ Auto-parallelizing</th>
<th>Fixed Architecture</th>
<th>Architecture to be designed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Starting from given task graph</td>
<td>Map to CELL, Hopes, Qiang XU (HK) Simunic (UCSD)</td>
<td>COOL codesign tool; EXPO/SPEA2 SystemCodesigner</td>
</tr>
<tr>
<td>Auto-parallelizing</td>
<td>Mnemee (Dortmund) Franke (Edinburgh) MAPS</td>
<td>Daedalus</td>
</tr>
</tbody>
</table>
Daedalus Design-flow

Library of IP cores

Explore, modify, select instances

Sesame

Platform specification

Mapping specification

Kahn Process Network

Sequential application

KPNgen

System-Level Specification

RTL-Level Specification

Synthesizable VHDL

C/C++ code for processors

Xilinx Platform Studio (XPS)

Explore, modify, select instances

Ed Deprettere et al.: Toward Composable Multimedia MP-SoC Design, 1st Workshop on Mapping of Applications to MPSoCs, Rheinfels Castle, 2008
Example architecture instances for a single-tile JPEG encoder:

1. **2 MicroBlaze processors (50KB)**
   - 16KB Vin, DCT
   - 2KB
   - 32KB Q, VLE, Vout

2. **6 MicroBlaze processors (120KB)**
   - 8KB Vin
   - 4x2KB DCT, Q
   - 32KB VLE, Vout
   - 4x16KB DCT, Q

3. **1 MicroBlaze, 1HW DCT (36KB)**
   - 32KB Vin, Q, VLE, Vout
   - 4KB DCT

4. **4 MicroBlaze, 2HW DCT (68KB)**
   - 8KB Vin
   - 2KB DCT
   - 2KB Q
   - 8KB VLE, Vout
   - 32KB 2KB
   - 2KB 8KB
   - 2KB 8KB
   - 2KB 8KB
Sesame DSE results:
Single JPEG encoder DSE

Performance-memory trade-off DSE

Millions of clock cycles / tile

Memory utilization (KB)

© E. Deprettere, U. Leiden
A Simple Classification

<table>
<thead>
<tr>
<th>Architecture fixed/ Auto-parallelizing</th>
<th>Fixed Architecture</th>
<th>Architecture to be designed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Starting from given task graph</td>
<td>Map to CELL, Hopes, Qiang XU (HK) Simunic (UCSD)</td>
<td>COOL codesign tool; EXPO/SPEA2 SystemCodesigner</td>
</tr>
<tr>
<td>Auto-parallelizing</td>
<td>Mnemee (Dortmund) Franke (Edinburgh) MAPS</td>
<td>Daedalus</td>
</tr>
</tbody>
</table>

Qiang XU (HK) Simunic (UCSD)

COOL codesign tool; EXPO/SPEA2 SystemCodesigner
Auto-Parallelizing Compilers

Discipline “High Performance Computing”:
- Research on vectorizing compilers for more than 25 years.
- Traditionally: Fortran compilers.
- Such vectorizing compilers usually inappropriate for Multi-DSPs, since assumptions on memory model unrealistic:
  - Communication between processors via *shared memory*
  - Memory has only *one single* common *address space*

*De Facto* no auto-parallelizing compiler for Multi-DSPs!
Work of Franke, O’Boyle (Edinburgh)
Work of Daniel Cordes, Olaf Neugebauer (TU Dortmund)
Example: Edge detection benchmark

Task Parallelism

```c
1. int main() {
   // Initialize temporary image buffers
2.     for (i = 0; i < N; i++) {
3.         for (j = 0; j < N; ++j) {
4.             image_buffer2[i][j] = 0;
5.             image_buffer3[i][j] = 0;
6.         }
7.     }
8.   // Initialize filter
9.     convolve2d(image_buffer1, filter, image_buffer3);
10.    // Initialize filter2
11.     convolve2d(image_buffer3, filter2, image_buffer1);
12.    // Initialize filter3
13.     convolve2d(image_buffer3, filter3, image_buffer2);
14.    // Combine gradients and apply threshold
15.     for (i = 0; i < N; i++) {
16.         for (j = 0; j < N; ++j) {
17.             temp1 = abs(image_buffer1[i][j]);
18.             temp2 = abs(image_buffer2[i][j]);
19.             temp3 = (temp1 > temp2 ? temp1 : temp2);
20.             image_buffer3[i][j] = (temp3 > T) ? 255 : 0;
21.         }
22.     }
23. }
24. }
```
MaCC Application: Multi-Objective Task-Level Parallelization

Restrictions of embedded architectures
- Low computational power, small memories, battery-driven, …

➤ Good trade-off between different objectives must be found
➤ As fast as necessary instead of as fast as possible

<table>
<thead>
<tr>
<th></th>
<th>Time [Cycles]</th>
<th>Energy [mJ]</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Core</td>
<td>125,256,949</td>
<td>36.433</td>
<td>1.00</td>
</tr>
<tr>
<td>2 Cores</td>
<td>74,172,007</td>
<td>58.095</td>
<td>1.69</td>
</tr>
<tr>
<td>3 Cores</td>
<td>53,804,696</td>
<td>79.607</td>
<td>2.33</td>
</tr>
<tr>
<td>4 Cores</td>
<td>44,389,652</td>
<td>103.655</td>
<td>2.82</td>
</tr>
</tbody>
</table>

Speedup vs Energy Consumption for Matrix Multiplication on MPARM with MEMSIM
Multi-Objective Task-Level Approach (2)

**Graph Description:**
A 3D bar graph showing trade-offs between communication overhead, energy consumption, and speedup of execution time. The graph highlights Pareto-Optimal solutions and dominated points. The x-axis represents speedup of execution time, the y-axis represents communication overhead, and the z-axis represents energy consumption.

**Details:**
- **Communication overhead:** 1x, 4x, 8x, 12x, 16x, 20x
- **Energy:** 100%, 160%, 220%, 280%, 340%
- **Speedup:** 1x, 1.2x, 1.4x, 1.6x, 1.8x, 2x, 2.2x

**Benchmark:**
- *Edge detect* benchmark from UTDSP benchmark suite
- Target architecture: MPARM with MEMSIM energy model (1-4 cores)
Pipeline Parallelization

1. Extract pipeline stages (horizontal splits)

\[ x_n^t = \begin{cases} 
1, & \text{if node } n \text{ is mapped to pipeline stage } t \\
0, & \text{otherwise} 
\end{cases} \]

\[ \forall n \in \text{Nodes} : \sum_{t \in \text{Stages}} x_n^t = 1 \]

ILP formulation:
Heterogeneous Pipeline Parallelization

Heterogeneous MPSoCs can be more efficient than homogeneous

- Cores behave differently on same parts of application
- Efficient balancing of tasks very difficult

\[ \text{\begin{align*}
\text{Performance} & \quad \text{Power} \\
\text{Highest Cortex-A15} & \quad \text{Operating Point} \\
\text{Lowest Cortex-A15} & \quad \text{Operating Point} \\
\text{Highest Cortex-A7} & \quad \text{Operating Point} \\
\text{Lowest Cortex-A7} & \quad \text{Operating Point}
\end{align*}} \]

\[ \text{Use ILP as clear mathematical model integrating cost models} \]
\[ \text{Combine mapping with task extraction} \]

\[ \begin{align*}
(1) & \quad \min \sum x_i \cdot c_i \\
(2) & \quad \sum b_i \cdot x_i \leq C \\
(3) & \quad \forall x_i \leq 1 \\
(4) & \quad \forall x_i \geq 0
\end{align*} \]
Heterogeneous pipeline parallelization results

- Cycle accurate simulator: CoMET (Vast)
- 100, 250, 500, 500 MHz ARM1176 cores
- Baseline: Sequential on 100 MHz core

- Average speedup: 8.9x
- Max. Speedup: 11.9x
MAPS-TCT Framework
Summary

- Clear trend toward multi-processor systems for embedded systems, there exists a large design space
- Using architecture **crucially** depends on **mapping tools**
- Mapping applications onto heterogeneous MP systems needs allocation (if hardware is not fixed), binding of tasks to resources, scheduling
- Two criteria for classification
  - Fixed / flexible architecture
  - Auto parallelizing / non-parallelizing
- Introduction to proposed Mnemee tool chain

Evolutionary algorithms currently the best choice