Specification verification and optimization of distributed embedded real-time systems

Algorithm Architecture Adequation methodology

Yves SOREL
Yves.Sorel@inria.fr

http://www.syndex.org
INRIA
French National Institute for Research in Computer Science and Control
http://www.inria.fr

• Since 1968
• 3500 persons
• 6 centers
• Budget: 135 MEuro
• 20 Startup companies

• Research
• Knowledge Transfer
• Training
• W3C
• International collab.
Five research themes

1. Communication systems
2. Cognitive systems
3. Symbolic systems
4. Numerical systems
5. Biological systems
INRIA Context

• AOSTE project-team
• Theme Com C: Embedded systems and mobility
• Models and methods of Analysis and Optimization for Systems with real-Time and Embedding constraints
• Implementation onto embedded platforms: Algorithm-Architecture Adequation Methodology for the optimized implementation of distributed embedded real-time applications (SynDEx software)
Applications

- Automobile (AEE, EAST, ECLIPSE)
- Mobile robotic (CyCab)
- Signal processing: Software radio (Mitsubishi ITE)
- Telecommunication: SoC UMTS (PROMPT)
- Image processing: Automatic guidance (MBDA), MPEG4 (INSA)
- ...

INRIA
Goals

- Safe design
- Rapid prototyping
- Optimization
- Automatic code generation
- Reduced development cycle
Characteristics

- **Algorithms**: Automatic control, signal & image processing
- **Reactive**: *Stimulus event* - *Operations* – *Reaction event*
- **Real-Time**: Constraints: Latency = bounded Reaction Time
  Cadence = bounded Input Rate
- **Distributed**: Power, modularity, wires minimization

**Heterogeneous Multicomponent Architecture**
- Network of processors and specific integrated circuits
- Specific integrated circuits = ASIC, ASIP, FPGA, IP
- **Embedded**: Resources minimization
Algorithm Architecture Adequation Method.

- **Global approach** based on the Synchronous Languages Semantics and the hardware RTL models
- **Unified Model**: Directed graphs
  - **Algorithm**: Operation / Data-Conditioning Dependence
  - **Architecture**: FSM / Connection
  - **Implementation**: Graphs Transformations
- **Adequation**: Optimized Implementation (best matching)
- **Macro-Generation**:
  - Real-Time Executives for Multicomponent
  - Structural VHDL for Integrated Circuit Synthesis
Off-line versus on-line approaches

**AAA Methodology**

- Off-line → Off-line (as much as possible)
- On-line → Off-line (when unavoidable)

**Classic design**

- On-line → Off-line

---

**Static**: distrib./sched. off-line without preemption

**Dynamic**: distrib./sched. in-line or off-line with preemption

---

**Advantages**

- Off-line: deterministic, low over-head
- On-line: data dependent durations

**Drawbacks**

- Off-line: not always possible, knowledge of environment necessary
- On-line: high over-head, soft real-time
Algorithm specification: Directed hyper-graph

- **Vertex**: Conditioned operation: inputs-computations-outputs
- **Directed Hyper-Edge**: Dependence (Diffusion): Data with or without precedence, precedence only, or conditioning (Exec or not)
  => *Partial order = Potential parallelism*
- **Infinite repetition of sub-graph**: Reactive Infinite Loop
- **Finite repetition of sub-graph**: Finite Loop
- Factorized conditioned data-dependence graph
- May be obtained from compilers of: Synchronous Languages, AIL, Scicos, AVS, CamlFlow,...
Algorithm example: Adaptative equalizer

Data dependence
with or without precedence
Precedence only

Inter-repetition Dependence
3 coefficients digital filter: Finite repetition of 3

\[ Y_n = \Sigma (h_i * X_{n-i}) \text{ for } i=0 \text{ to } 2 \]

Fork: \( H(h_0, h_1, h_2) \text{ et } X(x_n, x_{n-1}, x_{n-2}) \)

Join: \( Y(y_n, y_{n-1}, y_{n-2}) \)
Conditioned Algorithm: Modulo 3 Counter

\[
\begin{array}{c|c|c|c|c|c|c|c}
\hline
\text{t} & 0 & 1 & 2 & 0 & 1 & 2 & 1 \\
\hline
\text{ZR} & 1 & 2 & 3 & 1 & 2 & 1 & 0 \\
\text{S} & 0 & 0 & 1 & 0 & 1 & 2 & 0 \\
\text{Cond} & 1 & 2 & 0 & 1 & 0 & 2 & 2 \\
\hline
\end{array}
\]

If Cond = 1

If Cond = 0

conditioned RAZ

conditioned RAZ
Finite state machine with AAA/SynDEx (1/2)
Finite state machine with AAA/SynDEx (2/2)
Multi-period

Algorithm Operation multi_rhythm (main)

Window  Edit

clock0_10  Clock10_20
algo10ms_1  algo10ms_2

algo_20ms

Algorithm Operation algo_10

Window  Edit

in  f10ms  out

0  1  0

Algorithm Operation algo_2

Window  Edit

in  f20ms  out

0  0  1

Architecture archi (main)

Window  Edit

op1 (PC)  y  s

Lcp (w/TCP)  y  s

syncro (SYNC)
time (timer)

Algorithm Operation multi_rhythm onto archi

Window  Edit

op1  opr2  time  syncro  tcp

in  in1  f10ms  f20ms  clock0_10

D(in)=D(out)=1
D(f10ms)=9
D(f20ms)=12
Algorithm verification

• Synchronous languages
  • Esterel, Lustre, Signal, SyncCharts ...
  • Modular specification
  • No hardware constraint, independent from Physical Time => Logical Time, events ordering
  • Reaction simultaneous with Stimulus
  • Verifications:
    • Dependence cycle only with delay
    • Reactions order consistent with stimuli order
    • Logical temporal properties: ex. event always occurs
• **Vertex (FSM sequential machine)**
  • Operator: executes sequentially operations
  • Communicator: executes sequentially communications
  • Memory with or without arbiter
    • Random Access (RAM): non synchronized comm.
      • Stores Program and Data, Shared memory Comm.
    • Sequential Access (SAM): synchronized comm., R/W order
      • Stores Data
      • Message passing Comm., Point-to-point, multi-point with or without hardware diffusion
  • **Bus/mux/demux with ou without arbiter**
    • Bus/mux/demux: selects a memory among several
    • Bus/mux/demux/arbiter: arbitrates shared memory
Architecture specification: Directed graph (2)

• Directed edge
  • Connection between two vertices: models data transfers
  • Connections must follow a set of rules:
    • Operators must not be connected together
    • Communicators must not be connected together
    • Memories must not be connected together
    • Bus/mux/demux may be connected together
    • ...

• Macro-RTL model: operation, macro-register
Architecture example

Processor1
- Opr1
- Com1a
- Com1b
- RAM
- D/P

Message Passing Com Pt to Pt

Processor2
- Opr2
- Com2a
- Com2b
- RAM
- D/P

Processor3
- Opr3
- Com3a
- RAM
- D/P

Message Passing Com Multipoint

RAM
D/P

SAM

SAM

RAM
D

RAM
D

RAM
D

Bus/mux/demux/arbiter

Bus/mux/demux

Shared Memory Com

Specific IC

Opr4
Multicomponent implementation (1)

The set of all possible implementations is described as the composition of three binary relations:

\[(\text{Gal}, \text{Gar}) \xrightarrow{\text{rout o distrib o sched}} (\text{Gal'}, \text{Gar'})\]

- **Routing**: creation of all the inter-operator routes
- **Distribution**: spatial allocation
  - Partitioning and allocation: operations onto operator
  - Partitioning of inter-partition edges according to routes
  - Creation and allocation:
    - Communication vertices onto communicators of the route
    - Allocation vertices onto memories
    - Identity vertices onto bus/mux/demux/ with or without arbiter
Multicomponent implementation (2)

- **Scheduling**: *temporal allocation*
  - Partial Order $\rightarrow$ Total Order for:
    - Each partition of operations allocated onto an operator
    - Each partition of communication operations allocated onto a communicator

Routing, distribution and scheduling lead to a partial order consistent with the initial partial order of the algorithm graph

Graph transformation: External Composition Law
Implementation graph $Gal' = Gal \ast Gar$
Implementation example: Point to Point SAM
Implementation example: Multipoint SAM

Without | With Broadcast Support
Optimization principles

• **Operations/Operators characterization**
  • Measures: duration, memory, comp./comm. interferences

• **Choice of one implementation among all**
  • Satisfying real-Time (latency=cadence) and distribution constraints
  • Minimizing resources

• **Distribution/Scheduling:** Off-Line, with or without preemption

• **NP-hard problems:** Heuristics
  • Fast: Greedy: list-scheduling, etc for rapid prototyping
  • Slow: Neighboring: Tabou, Simulated Anealing, etc
Optimization example
Latency=Cadence, off-Line without preemption

• **Earliest start from start** date:
  \[ S(oi) = \max \{ E(x_j) \mid \forall x_j \in \text{pred}(oi) = \emptyset \} \]

• **Earliest end from start** date:
  \[ E(oi) = S(oi) + \Delta(oi) \]

• **Latest end from end** date:
  \[ \bar{E}(oi) = \max \{ \bar{S}(x_j) \mid \forall x_j \in \text{succ}(oi) = \emptyset \} \]

• **Latest start from end** date:
  \[ \bar{S}(oi) = \bar{E}(oi) + \Delta(oi) \]

• **Scheduling Flexibility**:
  \[ F(oi) = R - S(oi) - \bar{S}(oi) \]

• **Scheduling Penalty**:
  \[ P(oi) = R - R' \]

• **Scheduling Pressure**:
  \[ \sigma(oi) = P(oi) - F(oi) \]
List-Scheduling greedy heuristic

1. Initialize the list of candidates with operations without predecessor:
   \[
   \text{Cand} = \{ o_i \mid \text{pred}(o_i) = \emptyset \}
   \]

2. While the list is not empty:
   1. For each operation \( o_i \) of the list search the best operator (taking into account communications costs),
   \[
   \text{o}_\text{pr}_i = \min_{p_i \in \text{Proc}} \sigma(o_i, p_i)
   \]
   2. Select from the list the most urgent operation to schedule,
   \[
   \text{o}_\text{urgent} = \max_{o_i \in \text{Cand}} \sigma(o_i, \text{o}_\text{pr}_i)
   \]
   3. Remove the operation from the list and add all its successors, which are now schedulable,
   \[
   \text{Cand} = \text{Cand} - \text{o}_\text{urgent} + \text{Succ}_\text{ordo}(\text{o}_\text{urgent})
   \]
Real-time executives generation

- Without deadlock, very low overhead
- Processor independent Macro-Code (.m4)
  - Extensible, easily portable (C, asm, …)
- Processor dependent executive kernel (.m4x)
  Macros definitions for:
  - Microcontrollers: MPC555, MC68332, 80C196
  - DSP: Sharc, TMS320C40
  - Microprocessors: i80x86, PPC G4, C/Unix
  - Communication Media: links, CAN, RS232, TCP/IP
Graphs transformations

• Optimized implementation graph in execution graph by adding system vertices:
  • Extraction of the infinitely repeated sub-graph
  • Explicit repetition by adding loop/endloop vertices
  • Adding init./finalisation vertices for inputs and outputs
  • calc/com synchronization by adding Pre/Suc vertices using semaphores

• Execution graph in macro-code
• Macro-code (file.m4) in source program: macro-processor (gm4) + executive libraries (macros definitions file.m4x)
Execution graph: explicit repetition

processor1

Opr1

in

loop

send

endloop

in

processor2

Opr2

out

loop

receive

endloop

out

RAM

D/P

Comr1a

SAM

Comr2a

RAM

D/P

all.Din/calc

all.Pin

all.Dlin

all.Din/calc

all.Dcalc/out

all.Dlout

all.Pout

all.Pcalc

all.Dcalc
Execution graph: synchronizations

processor1
Opr1
RAM
Comr1a
D/P

loop
Suc-E
in
Pre-F
Suc-F
Pre-E
endloop

s_empty
s_full

Pre-E
Suc-F
send

Pre-E
Suc-F
send

Pre-E
Suc-F
send

Pre-E
Suc-F
send

Pre-E
Suc-F
send

Pre-E
Suc-F
send
Macro-code generation

processor1

Opr1

RAM

D/P

Comr1a

in_init

loop

Suc-E

s_empty

in

in/calc

Pre-F

Suc-F

send

Pre-E

endloop

in_end

endloop

in_end

endloop

endprocessor_
Macro-code expansion and compilation

processor_( opr1, )

semaphores_(s_empty...)
alloc_(type_in/calc...)
thread_(comr1a)
pre0(s_empty)
loop_
sucF(s_full)
send(in/calc)
preE(s_empty)
endloop_
endthread_
main_
in_init()

spawn_thread(comr1a)
loop_
sucE(s_empty)
in(in/calc)
preF(s_full)
endloop_
in_end()endmain_
endprocessor_

MACRO PROCESSOR M4

MACROS EXECUTIVE KERNEL
Generic
Architecture Specific

Files .m4

Files .m4x

Sources

Exe1

Exe1

MACROS APPLICATION LIBRARIES

Files .m4x

COMPILERS
Distributed real-time executive generation: simple application
Executive generation: Synchro calc/com

- Buffer
- Intra-repetition Synchro
- Inter-repetition Synchro

Inter-repetition Boundary
include(syndex.m4x)dnl
processor_(555,root, ABCD, SynDEx v5.1c (c) INRIA 2000,
Thu Mar 16 14:07:12 2000)
semaphores_(
  B_d_empty_can, B_d_full_can, 
  A_c_empty_can, A_c_full_can, 
) alloc_(int,A_b)
alloc_(int,A_c)
alloc_(int,B_d)

main_
  spawn_thread_(can)
  sensor() loop_
    Suc0_(A_c_empty_can)
    sensor(A_b, A_c) Pre1_(A_c_empty_can)
    Suc0_(B_d_empty_can) Pre0_(B_d_empty_can) loop_
      Suc1_(A_c_full_can)
      send_(A_c, 555,root,p) Pre0_(A_c_empty_can)
      Suc1_(B_d_full_can) Pre1_(B_d_empty_can) loop_
        Suc1_(A_c_full_can)
        send_(B_d, 555,root,p) Pre0_(B_d_empty_can)
      endloop_
    endloop_
  sensor() wait_endthread_(can) endmain_
endprocessor_
Executive generation: Synchro. comm.

S : Send
R : Receive
N : Sync
Executive generation: Shared mem. comm.
Executive generation: Shared mem. comm.

```c
#include(syndex.m4x) dnl

processor(2160, root, Comm-RAM, ../..)
shared(ram)

semaphores(
    B_d_empty_ram, B_d_full_ram,
    A_c_empty_ram, A_c_full_ram
)
alloc(int, A_c_ram)
alloc(int, B_d_ram)

endshared

semaphores(
    B_d_empty_can, B_d_full_ram,
    A_c_empty_can, A_c_full_ram
)
alloc(int, A_b)
alloc(int, A_c)
alloc(int, B_d)

main_
spawn_thread_(ram)
sensor()
loop_
    Suc0_(A_c_empty_ram)
    Sen0_(A_b, A_c)
    Prel_(A_c_full_ram)
    Succ0_(B_d_empty_ram)
    compute(A_b, B_d)
    Prel_(B_d_full_ram)
endloop_
sensor()
wait_endthread_(ram)
endmain_

thread_(RAM, ram, root, p)
    loadDnto_(), p)
    Prel_(A_c_empty_ram)
    Prel_(B_d_empty_ram)
    loop_
        Suc0_(A_c_empty_ram)
        Suc1_(A_c_empty_ram)
        Succ1_(A_c_empty_ram)
        move(A_c, A_c_ram)
        Prel_(A_c_full_ram)
        Prel_(B_d_empty_ram)
        Succ1_(B_d_full_ram)
        Succ1_(B_d_empty_ram)
        move(B_d, 2160, root, p)
        Prel_(B_d_empty_ram)
        Prel_(B_d_full_ram)
    endloop_
    endthread_

main_
spawn_thread_(ram)
    Prel_(A_c_empty_ram)
    actuator() 
    Prel_(B_d_empty_ram)
    loop_
        Suc0_(A_c_full_ram)
        Suc1_(A_c_full_ram)
        Succ1_(A_c_full_ram)
        move(A_c, A_c_ram)
        Prel_(A_c_empty_ram)
        Prel_(B_d_empty_ram)
        Succ1_(B_d_full_ram)
        Succ1_(B_d_empty_ram)
        move(B_d_ram, B_d)
        Prel_(B_d_empty_ram)
        Prel_(B_d_full_ram)
    endloop_
    actuator(B_d_full)
    Prel_(B_d_empty_ram)
    endmain_
endprocessor
```

AAA/SynDEx
Configuration of standard executives

• Without Deadlock
• Translation of macro-code in standard executive configuration file
• Standard executives
  • RT-Linux, RT-AI
  • Osek
  • etc
System level CAD software: SynDEx

- AAA methodology support
- Interface with high-level languages
- Graphic interactif Unix X11 or Windows
- Algorithm and architecture graphs edition
- Off-line distribution and scheduling
- Real-time predicted diagrams visualization
- Real-time performances measure
- Real-time distributed executives generation
CyCab example

- Speed 30km/h
- Electric motors
- 4 wheel drive
- 2 steerings FWD RWD
- 1 to 4 MPC555, 1PC
- CAN Bus

Industrialized by Robosoft
http://www.robosoft.fr
Modeling/Simulation
Scilab/Scicos  Free at: www.scilab.org

Scilab/Scicos

CyCab Model

Masses → Suspension → Weels → Ground

Driver

Screen

Joystick, Radio...

Discret Controller

Motors

Coders, Camera...

SynDEEx

Control Laws  Delays

INRIA
Implementation
AAA/SynDEx Free: www.syndex.org

Scicos/Scilab

Control Laws

Delays

AAA/SynDEx

Algorithm

Architecture

Adequation
Distribution/Scheduling
Heuristics
Macro-Executive Generator

Constraints

Performances
Timing Diagram

m4 / gcc
Macro-Executives i80386 and MPC555
+ executive kernel + appli libs

Executives

CyCab
Manual driving application for the CyCab
Adaptive equalizer application
Integrated circuit implementation

- Algorithm graph transformed in implementation graph
  - Factorized vertex = VHDL component repeatedly executed on different data
  - Factorized hyperedge = VHDL signal
  - Component/signals controled by counters/mux/registers
  - Automatic synthesis of data and control paths
- Optimization
  - Surface/(latency=cadence) compromise
  - Defactorization: data parallelism, pipe-line, retiming
  - Until real-time constraints are satisfied
- SynDEx-IC A2SI ESIEE collaboration
Conclusion

• **Development cycle is dramatically reduced**
  • Safe design
  • Rapid prototyping and optimized implementation
  • Deadlock free executives, debug only executive Kernel
• **Work in progress**
  • Real-time multi-constraints: latencies and cadences (period)
    • Off-Line with or without preemption, strict period or jitter
    • In-Line for aperiodic events: slot shifting, adaptive scheduling
  • GALS: globally asynchronous locally synchronous
  • SynDEx/SynDEx-IC coupling: Codesign environment
  • Automatic Software/Hardware partitioning
  • Fault tolerance: collaboration with POP-ART team